Verfahren zur Objekterkennung
OpenCV-Funktionen

Ronny Kober
07. Juni 2005

1. Testumgebung

CPU : AMD Athlon XP 2600 2,08 GHz
Hauptspeicher : 512 MB DDR RAM 400 MHz
Betriebssystem : Microsoft Windows XP (Service Pack 2)

2. Template-Matching

wichtige verwendete OpenCV-Funktionen:
cvMatchTemplate(…) → Funktion Template-Matching
cvMinMaxLoc(…) → Positionsfindung des Minimum und Maximum
cvThreshold(…) → Binarisierung über Schwellwert

Funktionsweise:
Das Grundprinzip des Schablonenvergleichs, auch Template Matching genannt, besteht darin, ein Bild des gesuchten Objektes über das aktuelle Bild zu schieben, um dann an jeder Position ein Ähnlichkeitsmaß berechnen zu können. Wenn sich für eine Position ein hinreichender Ähnlichkeitswert ergibt, so wird davon ausgegangen, dass sich das gesuchte Objekt dort befindet.

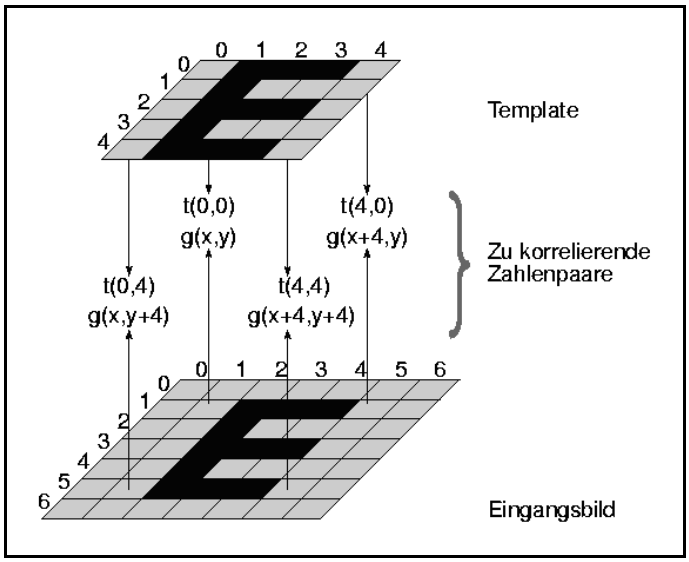

Abbildung 1: Übersicht des Template-Matching

Eingabedaten:
- Ausgangsbild
- Template, was kleiner als das Ausgangsbild sein muss

Ausgabedaten und Auswertung:
- Ergebnisbild hat die Größe des Ausgangsbildes minus der Template-Höhe und –Breite, gefüllt mit Werten als Ähnlichkeitsmaß
- je größer der Wert desto größer die Übereinstimmung
- Binarisierung über z.B. 95% des Maximums im Ergebnisbild
- weitere Auswertung des Binärbildes

Abspeicherung der Objektmerkmale:
- Template wird als Bild abgespeichert
- Komprimierung durch bestimmte Bildformate JPEG, GIF

Geschwindigkeit:
Bildgröße: 640 x 480 Pixel

Farbkanal: 1 (8Bit)
Template: 64 x 64
Dauer für die Erzeugung eines Ähnlichkeitsmaß-Bildes: über 2000 ms

Vorteile:
- Ähnlichkeitsmaß

Nachteile:
- Rotations- und Skalierungsinvariant
- sehr langsam

Beispiel:

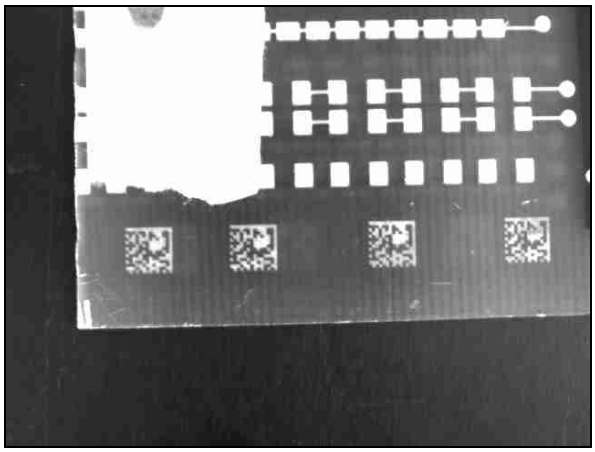

Abbildung 2: Eingangsbild

Abbildung 3: Template

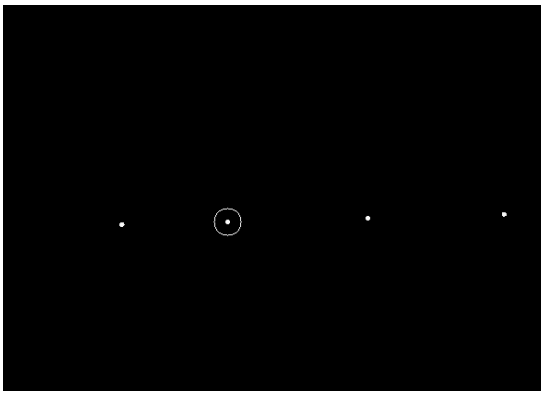

Abbildung 4: binarisiertes Ergebnisbild

Pyramidensuche:
Die Korrelation steigt bereits in der Umgebung eines Objektes deutlich an, je flächiger das Objekt ist. Es wäre also nicht unbedingt notwendig, das gesamte Bild Punkt für Punkt abzusuchen. So könnte das Template in größeren Sprüngen bewegt und nur in der Umgebung von Gebieten starker Korrelation müsste dann eine Punktweise Suche durchgeführt werden.

Unterabtastung:
Durch eine größere Schrittweite wird die Anzahl der Korrelationsrechnungen reduziert. Jedoch ist jede einzelne Korrelationsrechnung immer noch genauso aufwendig wie vorher. Dieser Rechenaufwand kann aber auf ähnliche Weise, wie bei der Pyramidensuche – und auf der Basis derselben Begründung des relativ stetigen Verlaufs der Korrelation – reduziert werden, indem nämlich nicht jeder Punkt des Templates verwendet wird, sondern z.B. nur jeder zweite, dritte oder vierte. Der Effekt auf die Rechenzeit ist ähnlich, ebenso die Auswirkungen auf das Ergebnis und die Einschränkungen der Anwendbarkeit.

Optimierung der Punktpositionen:
Bei der einfachen, äquidistanten Unterabtastung des Templates kann das Problem auftreten, dass die Punkte beliebig schlecht liegen können, beispielsweise unmittelbar auf einer Kante, was den Korrelationsvorgang bis auf ein Pixel von der Position abhängig machen kann. Mit einem auf diese Weise unterabgetasteten Template ist dann praktisch keine Pyramidensuche mehr möglich. Dieses Problem lässt sich beheben, indem die Positionen der zu verwendenden Punkte nach entsprechenden Kriterien optimiert werden (siehe Abbildung 5).

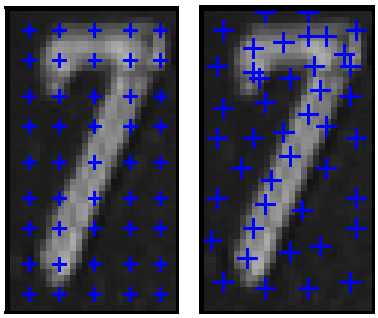

Abbildung 5: Regelmäßig verteilte - und optimierte Korrelationspunkte

Abkürzung der Rechnung:
Eine einfache Möglichkeit der Beschleunigung besteht darin, die Rechnung abzubrechen, sobald ein Misserfolg des Matchings offensichtlich wird. Allgemein gibt es sehr viel mehr Stellen im Bild, an denen das Matching erfolglos bleibt, als solche, an denen tatsächlich etwas gefunden wird. Je früher ein Misserfolg erkannt wird, desto früher kann die Berechnung für den Bildpunkt abgebrochen werden.

3. Hough-Transformation

wichtige verwendete OpenCV-Funktionen:
cvCanny(…) → Kantendetektor
cvHoughLines2(…) → Houghtransformation

Funktionsweise:
Die Hough-Transformation erkennt geometrische Strukturen, wie Linien oder Kreise, in einem binarisierten Bild. Dabei ist sie sehr robust gegenüber Bildstörungen und teilweisen Verdeckungen der Objekte.

Beispiel Liniendetektion:
Zur Detektion von Linien wird für jedes Vordergrundpixel eine Geradenschar (mit vorgegebener Schrittweite) berechnet und danach vom Koordinatenraum (x, y) in den Hough-Raum (r, Φ) übertragen. Zur Umrechnung dient dabei die Hesse'sche Normalform:

$$r = x \cdot \cos(\phi) + y \cdot \sin(\phi)$$

r ..... *Abstand*
x, y ..... *Koordinaten der Punkte*
Φ..... *Winkel*

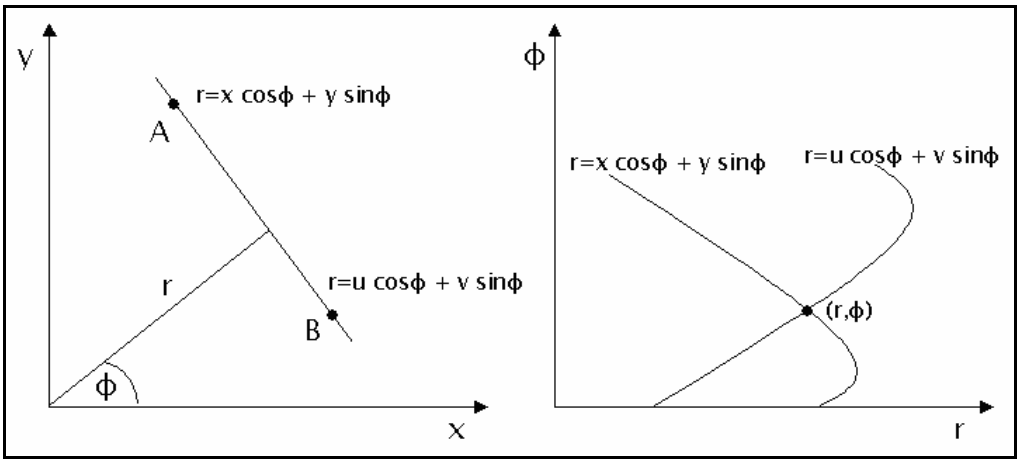

Abbildung 6: Gerade im Bild (links) entspricht Punkt im Hough-Raum (rechts)

Jede Gerade im (x, y)-Raum entspricht genau einem Punkt im (r, Φ)-Raum - jede Geradenschar entspricht somit einer Kurve im Transformationsraum, dem Akkumulator. Liegen nun viele Vordergrundpixel des Originalbildes auf einer Geraden, schneiden sich entsprechend viele Kurven im Akkumulator. Wird bei der Transformation der anfänglich leer (schwarz) initialisierte Akkumulator für jede Gerade inkrementiert, entstehen an den Stellen im Akkumulator besonders helle Pixel, auf dessen Gerade viele kollineare Punkte des binarisierten Ausgangsbildes liegen.

Eingabedaten:

- Ausgangsbild
- Schrittweite Winkel
- Schrittweite Radius

Ausgabedaten und Auswertung:

- Kantendetektion mittels des Canny-Algorithmus
- bestimmen der Geradengleichung mittels Hough-Transformation
- oder bestimmen von Strecken einer bestimmten Länge

Abspeicherung der Objektmerkmale:

- 2 Endpunkte
- Geradengleichungen
- Vektoren

Geschwindigkeit:

Bildgröße:	640 x 480
Farbkanal:	1 (8Bit)
Akkumulator-Schwellwert:	80
Schrittweite Winkel:	1°
Schrittweite Radius:	1 Pixel

Zeit für die reine Hough-Transformation: ca. 150 bis 220 ms

Vorteile:

- Detektion von verrauschten und gestörten Kanten

Nachteile:

- langsam

Beispiel:

Abbildung 7: Eingangsbild

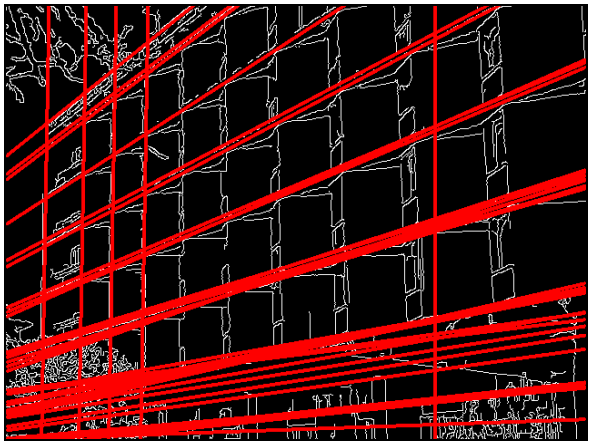

Abbildung 8: gefundene Geradengleichungen

Optimierungsansätze:

- je nach Bild und Objektgröße, kann eventuell das Ausgangsbild verkleinert werden
- größere Schrittweite des Winkels oder/und des Radius
- Intervallangabe des Winkels

4. Fitting-Verfahren

wichtige verwendete OpenCV-Funktionen:

cvPyrDown(…) → Verkleinerung durch Mittelung
cvPyrUp(…) → Vergrößerung
cvCanny(…) → Kantendetektor
cvFindContours(…) → Konturendetektor
cvFitEllipse(…) → bestmögliche Ellipse für Kontur
cvMinAreaRect2(…) → bestmögliche Rechteck für Kontur

Funktionsweise:

Gauß-Pyramide

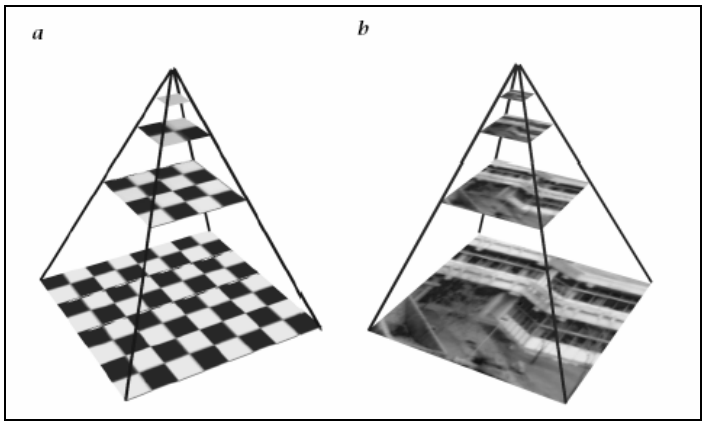

Abbildung 9: a) schematische Darstellung (die Quadrate des Schachbrettes entsprechen Bildpunkten) b) ein Beispiel

Stufe	Zeilen/Spalten
G_0	256
G_1	128
G_2	64
G_3	32
G_4	16
G_5	8
G_6	4
G_7	2

Abbildung 10: Bildgrößen der Stufen eines 256x256 großen Bildes

Algorithmus für Gaußpyramide:

1. Faltung des Ausgangsbildes mit den Gaußfilter
2. Bildverkleinerung auf die halbe Höhe und halbe Breite
3. das gefilterte verkleinerte Bild ist jetzt neues Ausgangsbild für 1.

$$\frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

Abbildung 11: Gaußfilter-Maske

Canny-Algorithmus:

Im ersten Schritt wird das Bild mit einem Gauß Filter geglättet, um auftretendes Rauschen zu unterdrücken. Der Filter wird üblicherweise mit einer $N \times N$ Matrix angenähert und auf jeden Bildpunkt angewendet. Nachdem die Störungen im Bild entfernt wurden werden nun mit dem Sobel Operator die Gradienten der einzelnen Punkte berechnet. Dabei entstehen zwei Bilder, wobei das eine die horizontalen und das andere die vertikalen Kanten betont.

Nachdem ∂ges ermittelt wurde, wird mit $\theta = arctan(\partial Y / \partial X)$ die Richtung θ des Gradienten bestimmt und diskretisiert. Da ein Punkt nur acht Nachbarn hat, muss nur entschieden werden, in welche von

den 4 möglichen Richtungen der Gradient verläuft (4 Richtungen, weil eine Kante, die nach rechts oben verläuft auch gleichzeitig nach links unten zeigt).
Im nächsten Schritt werden zunächst ein oberer und ein unterer Schwellenwert bestimmt. Nachdem die Richtung des Gradienten bekannt ist, werden nur die Punkte oberhalb des oberen Schwellenwertes betrachtet. Wird in Richtung des Gradienten, oder in Gegenrichtung ein Pixel gefunden, welches einen größeren Gradienten hat, wird das Ausgangspixel verworfen und von diesem Pixel aus weitergesucht. Falls kein Pixel mit einem größeren Gradienten in der Umgebung existiert, hat man das Pixel, das die Kante am besten verfolgt, gefunden. Nachdem man so die
Kanten auf Breite eins reduziert hat, schließt man etwaige unterbrochene Kanten. Dazu braucht man den unteren Schwellenwert. Alle Punkte, die unter der unteren Grenze liegen, werden verworfen. Alle Punkte, die zwischen den beiden Grenzen liegen, werden bei der Linienverfolgung beachtet. Abb. zeigt ein mit dem Canny Verfahren bearbeitetes Bild.
Es bleibt noch anzumerken, dass das Canny-Verfahren nicht in der Lage ist, aus geschlossenen Kantenzügen Regionen zu bilden. Hierzu bedarf es komplexerer Verfahren wie z.B. aktive Konturmodelle, auf die hier aber nicht genauer eingegangen wird.
Trotz aller Bemühungen des Canny Algorithmus mit Störungen im Bild und unterbrochenen Kanten zurechtzukommen, sind eben diese Faktoren ein großer Nachteil für kanten-basierte Verfahren. Bilder mit einem hohen Rauschanteil oder einem geringen Kontrast stellen noch immer ein großes Problem für diese Verfahren dar.

Abbildung 12: Original (links) und nach Canny bearbeitetes Bild (rechts)

Eingabe:
- Ausgangsbild

Ausgabedaten und Auswertung:
- Glättung des Bildes mittels des Gaußpyramiden-Verfahren
- Kantendetektion mittels des Canny-Algorithmus
- Detektion von Konturen im Binärbild
- Berechnen der am besten passenden Ellipse mithilfe der Methode der kleinsten Quadrate
- Detektion des umschreibende Rechteck über Bildung der konvexen Hülle

Abspeicherung der Objektmerkmale:
- Rechteck: 4 Eckpunkte
- Kreis/Ellipse: Höhe, Breite, Winkel und Mittelpunkt

Geschwindigkeit:
Bildgröße: 640 x 480
Farbkanal: 1 (8Bit)
komplette Auswertung mit Vorverarbeitung, Kantenbilderzeugung und Konturfindung: unter 50 ms

Vorteile:
- sehr schnell
- wenige Parameter

Nachteile:
- kann keine komplexen Objekte erfassen
- Probleme bei Überschneidung von Objekten

Beispiel:

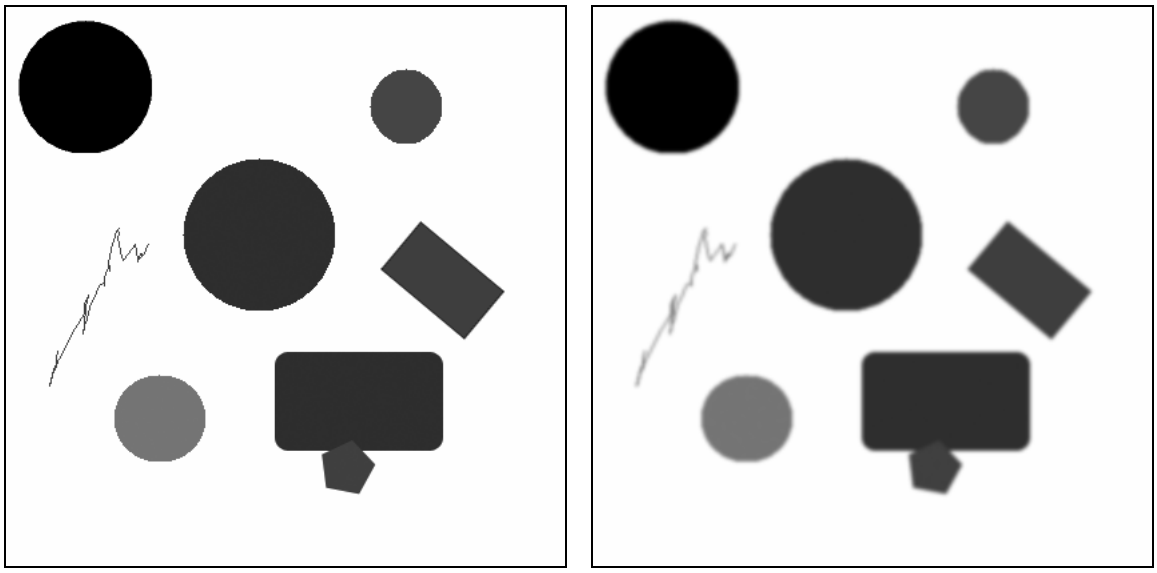

Abbildung 13: Eingangsbild (links) und geglättet (rechts)

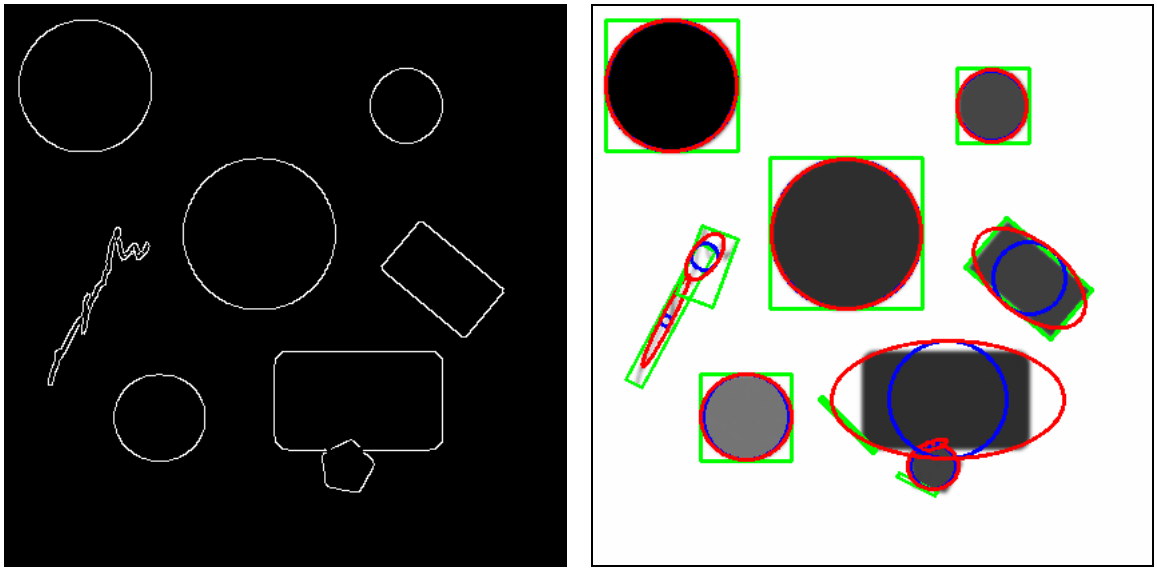

Abbildung 14: Canny-Kantenbild (links) und umschreibende Ellipse und -Rechteck

Optimierungsansätze:

- je nach Bild und Objektgröße, kann eventuell das Ausgangsbild verkleinert werden

5. Color-Tracking

wichtige verwendete OpenCV-Funktionen:

cvCvtColor(…) → Farbformat von RGB in HSV umwandeln
cvCvtPixToPlane(…) → Bild in die einzelnen Kanäle zerlegen
cvCreateHist(…) → Histogramm erzeugen
cvCalcBackProject(…) → Berechnung einer 2D Farb-Wahrscheinlichkeits-Verteilung
cvCamShift(…) → Color-Tracking

Funktionsweise:

HSI-System (Hue-Saturation-Intensity)
Einfacher in der Handhabung, als RGB ist das HSI-System, welches näher an die menschliche Wahrnehmung der Farbe angelehnt und somit auch verständlicher ist. In diesem System entspricht HUE (=Farbton) der Farbe, welche im symmetrischen Farbkreis (Abbildung 15) definiert wird, SATURATION (=Sättigung) dem Maß für den Anteil des reinen Farbtons in der Farbe und INTENSITY (=Intensität) dem Maß für die Menge des von Objekten reflektierten bzw. emittierten Lichtes.

H = 0..360 Grad (0° = Rot, 90° = Gelb-Grün, 180° = Hell-Blau, 270° = Violett)
S = 0..100 Prozent (0% nur Graustufen,100% entspricht reinem Farbton)
I = 0..100 Prozent (0% → Schwarz, 100% → max. Helligkeit)

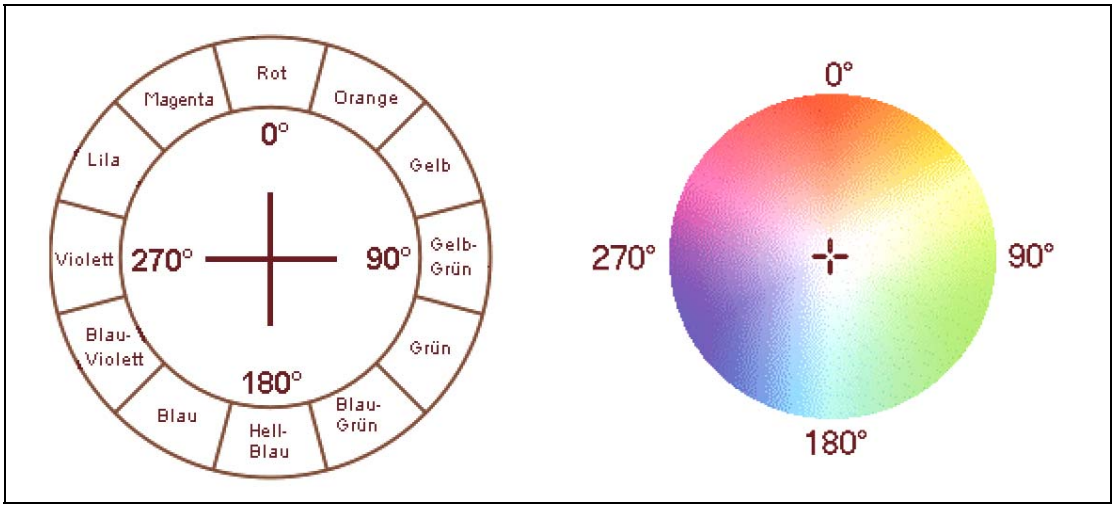

Abbildung 15: Symmetrische Farbkreise

Histogramme
Unter Histogrammen versteht man im allgemeinen die graphische Darstellung der Häufigkeitsverteilung von Messwerten. Bei den Messwerten handelt es sich in der Bildverarbeitung üblicherweise um Helligkeits- oder Farbstufen, die auf der x-Achse des Histogramms aufgetragen werden. Die y-Achse repräsentiert die Häufigkeit der jeweiligen Quantisierungsstufe, dass heißt die Anzahl der Pixel, die dem Farbton entsprechen.
Aus Histogrammen lassen sich eine Vielzahl von Bild-Informationen auslesen, wie zum Beispiel die Gesamtzahl der Pixel, der Mittelwert des Messwertes oder die Variationsbreite der Quantisierungsstufen.

CamShift Algorithmus
CamShift steht für “Continuously Adaptive Mean-SHIFT” Algorithmus, dessen Ablauf wird in Abb. veranschaulicht. Für jedes Video-Frame wird aus dem Bild eine Farb-Wahrscheinlichkeits-Verteilung berechnet, dies geschieht unter Verwendung des Farb-Histogramms des zu trackenden Objektes. Das Zentrum und die Größe des farbigen Objektes werden durch den CamShift-Algorithmus ermittelt, welcher auf das Farb-Wahrscheinlichkeits-Bild angewandt wird. Die aktuelle Größe und Orientierung des Tracking-Objektes werden ausgegeben und benutzt, um die Größe und Lage des Suchfensters im nächsten Frame festzulegen. Der Algorithmus wird für kontinuierliches Tracking wiederholt. Continuously Adaptive bedeutet, dass die Wahrscheinlichkeitsverteilung sich über der Zeit verändern kann, d.h., dass der CamShift-Algorithmus für Videoströme geeignet ist. Dieser Algorithmus ist eine Verallgemeinerung des MeanShift Algorithmus, der in Abb. im grauen Block dargestellt ist.
Der MeanShift Part des Algorithmus findet den Schwerpunkt (mean) von Objekten, indem er den Schwerpunkt eines zu untersuchendes Fensters in einem Bild so lange mit dem Schwerpunkt eines Objekts vergleicht, bis diese annähernd übereinstimmen. Die Objekte werden anhand ihrer Farbwahrscheinlichkeitsverteilung bestimmt, dass heißt je deutlicher sich das Objekt farblich vom Hintergrund unterscheidet, desto besser ist es zu tracken.

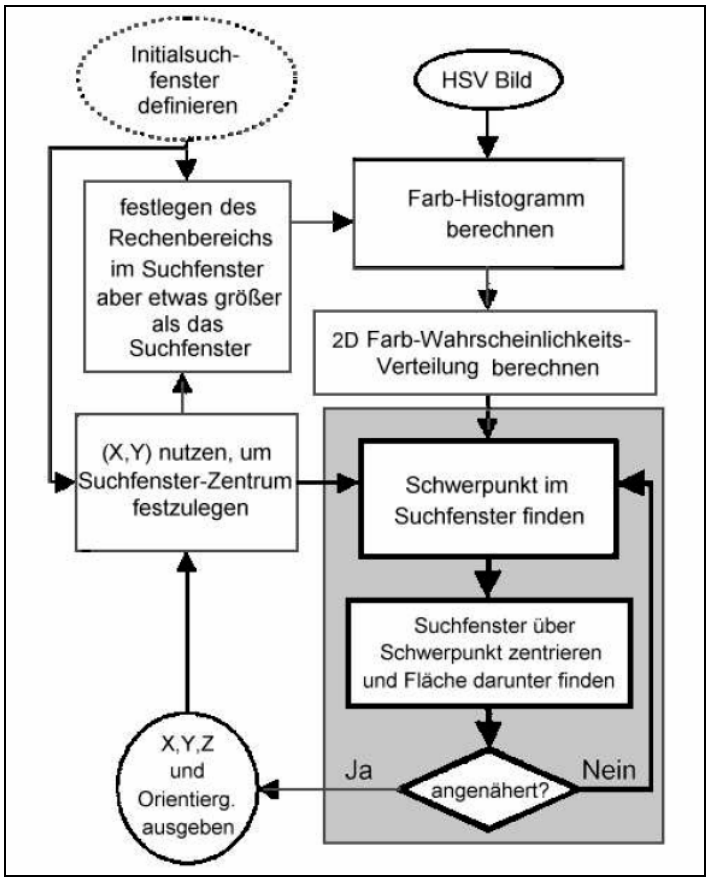

Abbildung 16: CamShift-Algorithmus

Ablauf des MeanShift-Algorithmus:

1) Größe des Suchfenster festlegen.
2) Initialposition des Suchfensters im Bild festlegen.
3) Berechnen der Position des Schwerpunkts im Suchfenster.
4) Suchfenster auf die in 3) berechneten Position des Schwerpunktes zentrieren.
5) Schritte 3 und 4 wiederholen bis Abbruchbedingungen erfüllt sind.

Ablauf des CamShift-Algorithmus:

1) Setzen des Rechenbereichs der Wahrscheinlichkeitsverteilung auf den gesamten Bildbereich.
2) Wählen der Startposition des 2D MeanShift-Suchfenster
3) Berechnung der Farb-Wahrscheinlichkeits-Verteilung im 2D Bereich zentriert an der Suchfensterposition in einem ROI (region of interest), welches etwas größer ist als die MeanShift-Fenstergröße.
4) Starten des MeanShift-Algorithmus um die Mitte des Suchfensters zu finden. Speichern des nullten Moments (Bereich oder Größe) sowie die Position der Mitte.
5) Für das nächste Frame zentriert man das Suchfenster an die Position die im 4) Schritt gespeichert wurde und setzt die Fenstergröße auf den Wert, den das nullte Moment bestimmt hat.
6) Zurück zu 3)

Eingabedaten:

- Ausgangsbild
- Template mit der Objektfarbe

Ausgabedaten und Auswertung:

- konvertieren des Ausgangsbildes und des Objektbildes in den HSV-Farbformat (= HSI)
- extrahieren des Hue-Kanals bei beiden Bildern
- Berechnung des Histogrammes das Hue-Kanals bei beiden Bildern

- 2D Farb-Wahrscheinlichkeits-Verteilung berechnen
- CamShift-Algorithmus anwenden

Abspeicherung der Objektmerkmale:
- Objekt wird als Bild gespeichert
- Komprimierung durch bestimmte Bildformate JPEG, GIF

Geschwindigkeit:
Bildgröße: 640 x 480
Farbkanal: 3 je (8Bit)
komplette Auswertung mit Vorverarbeitung (Histogramm erzeugen, Projektion): zw. 20 und 40 ms ist Parameterabhängig

Vorteile:
- wenige Parameter

Nachteile:
- sehr anfällig auf Hintergrundstörungen
- funktioniert nur auf Farbbilder

Beispiel:

Abbildung 17: Objektbild (Template)

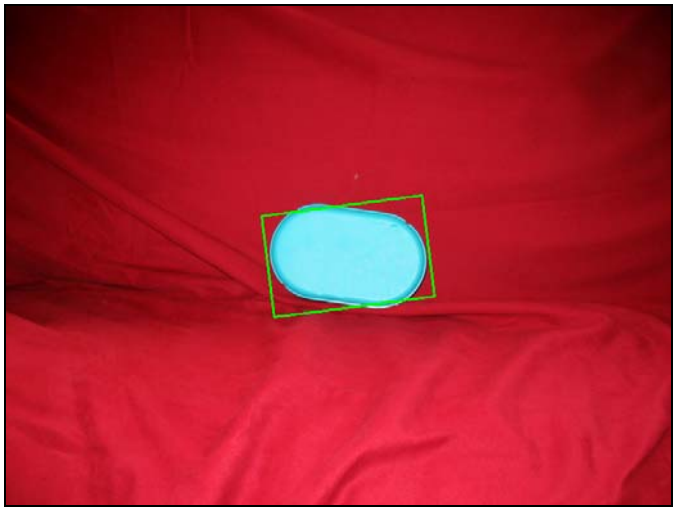

Abbildung 18: Suchbild mit erkannten Objekt