Aufgabe 2: Bildverarbeitung in einer Pipes&Filters Architektur (7 Pkte)

Ausgangspunkt dieser und einer noch folgenden Aufgabe ist folgender

Ausschnitt aus einer BVProduktbeschreibung:

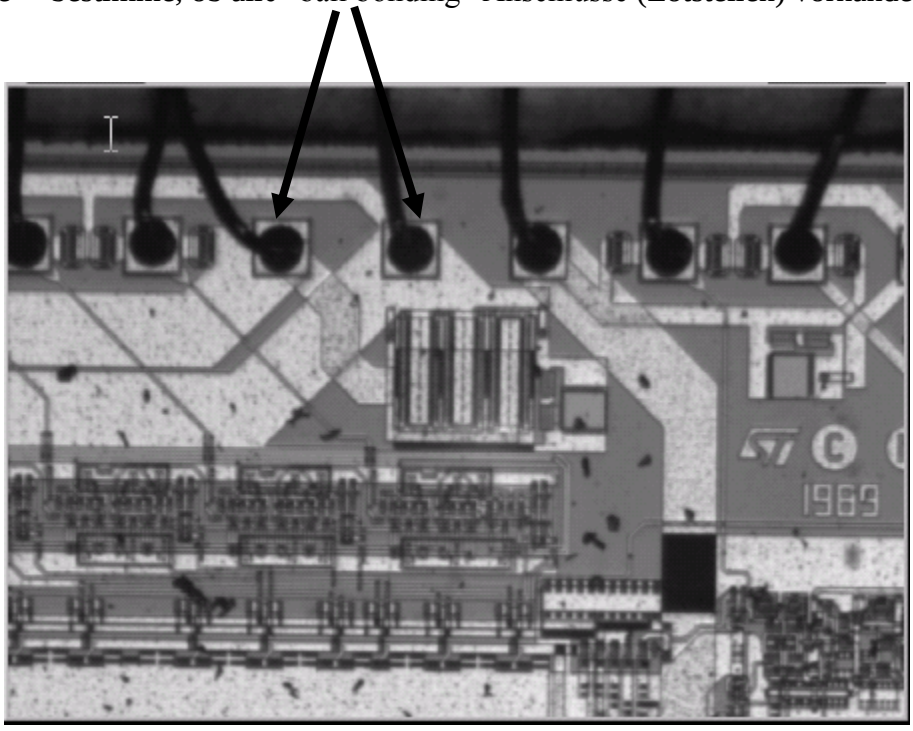
The typical solution of an image analysis problem consists of a combination of several operators. This concept allows much greater flexibility than others based on fewer, but more complex operators that are specialized to perform a certain task. Specialized operators are only suitable for special tasks and become worthless when the task changes.

In contrast to this, the operators shall be used in any combination. Among the great number of operators there might be some that implement the same task by different algorithms. This allows varying how fast and precisely a task is performed. Consider pattern matching (matching a search pattern in an image), for example: for a task where only the rough positions of matching points are needed, but where the results must be returned as fast as possible, the operator fast match may be used. In contrast to this, best match is the better choice when needing the exact positions (even with subpixel accuracy) by using a more time-consuming algorithm.

Benutzungsszenario eines BV-Produktes:

Aufgabe: Qualitätskontrolle – bestimme, ob alle "ball bonding" Anschlüsse (Lötstellen) vorhanden und richtig positioniert sind:

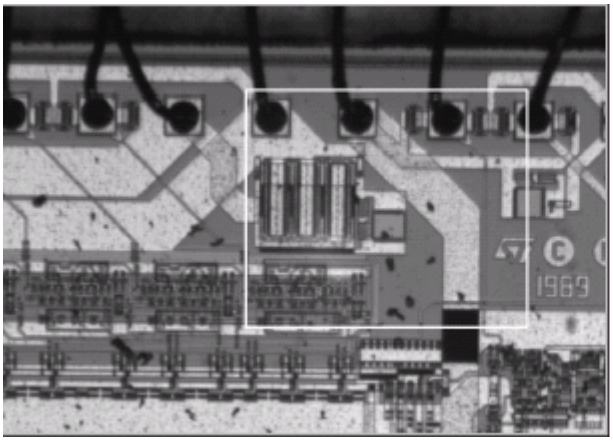
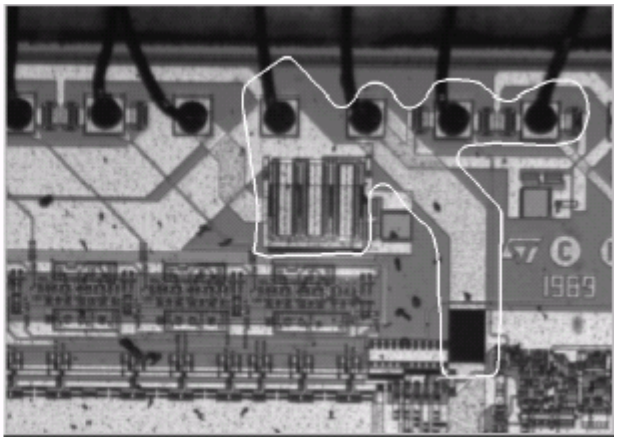
d.h., ob deren Positionen in einem zulässigen Toleranzbereich liegen



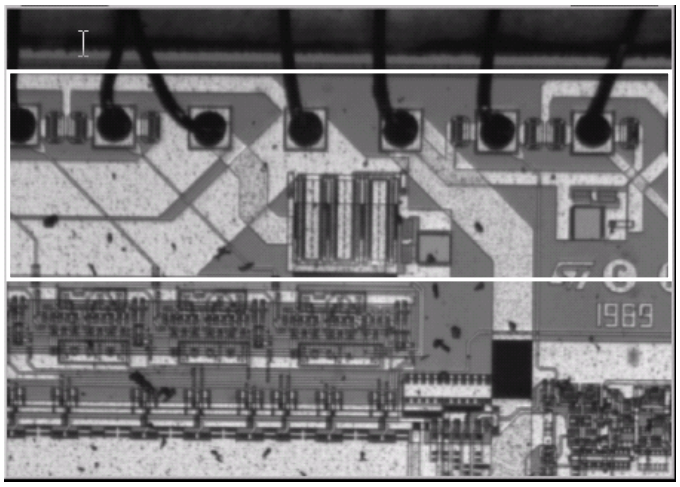
Die dargestellte Platine wird auf einem Fließband einer Produktionsstätte nach der Lötoperation unter einer Kamera vorbeilaufen, mit in etwa immer der gleichen Position.
Um ein Qualitätskontrollenprogramm zu erstellen, definiert der Benutzer nun Schritt für Schritt eine Bildanalyse, wobei jeder Schritt in einem Filter einer Pipes&Filter Architektur zu erfolgen hat.

Folgende Schritte sind durchzuführen:

1. *das Bild laden und visualisieren*
2. *eine ROI (region of interest[1]) definieren:*

wir können für unsere Aufgabe vereinfacht ein Rechteck annehmen:



Achtung: man muß nicht den komplizierten ROI Operator in JAI benutzen, sondern kann das Ganze ganz einfach so realisieren:

image = PlanarImage.wrapRenderedImage((RenderedImage)image.getAsBufferedImage(rectangle, image.getColorModel()));
wobei image ein PlanarImage ist, und rectangle ein java.awt.Rectangle ist, das die ROI angibt relativ zum Bild (rectangle = new Rectangle(int x, int y, int width, int height)). Achtung: Pixel oben links des

[1] Region of interest: each image object has a domain — its region of interest (ROI) — that can be changed by the user. When performing an image operator, it is processed only within the ROI. This concept allows to focus image processing and therefore to speed it up.

ausgeschnittenen Bildes hat wieder die Koordinaten 0,0. Sie müssen noch etwas tun, damit Sie die Position dieses Pixels im Originalbild mitspeichern im Bild (der letzte Filter braucht das!)
Option für den Benutzer: zeige das Rechteck in weiss mit dem Ausgangsbild

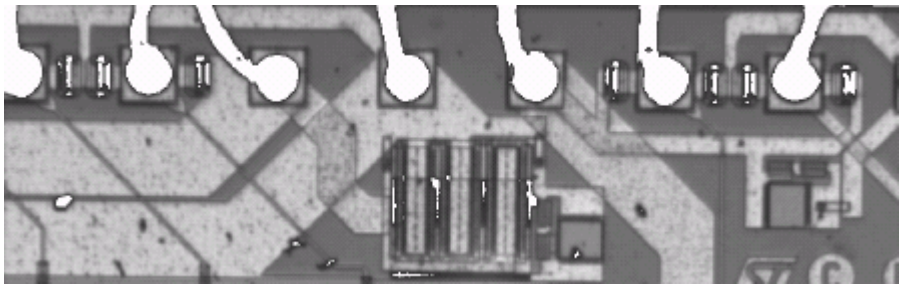
3. einen Operator zur Bildsegmentierung auswählen: Threshold Operator

aus der Operatorbibliothek (Filter) in JAI, mit einer Kurzbeschreibung des Operators/Filters (insbesondere die angebotenen Parameter, deren Werte eingestellt werden müssen zur Lösung der Qualitätskontrollenaufgabe)

hier: Threshold (Schwellwert)Operator zur Bildsegmentierung

3a. Parameterwerte des Operators wählen

z.B. 50 als Helligkeits-Schwellwert: wähle alle stark dunklen Teile des Bildes
Option für den Benutzer: überlagere in der Anzeige das Resultat mit dem Ausgangsbild (man könnte auch das Original wählen), und zwar in Weiß.



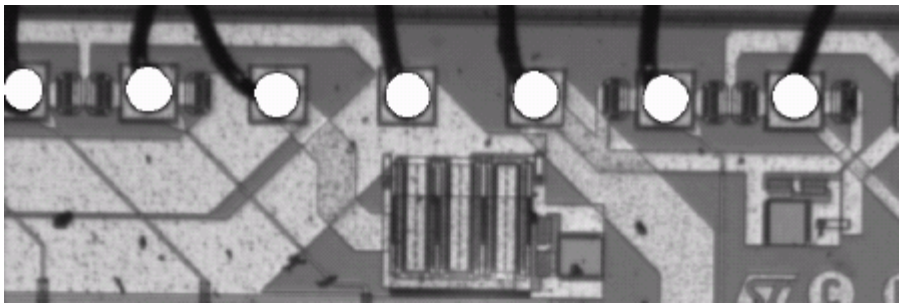
4. beseitige lokale Störungen (z.B. schwarzer Fleck im 2. Anschluss von rechts)
wähle z.B. einen Median Filter

4a. wähle Parameter des Filters: Größe der Maske zur Medianberechnung

(hier ohne Bild)

5. *nun bleiben noch die Kabelanschlüsse der „balls“; man nutzt die Kreisform der Balls aus und benutzt einen Opening-Operator mit kreisförmiger Maske* (in JAI: "erode" und „dilate“):

5a wähle Parameter des Operators: Größe der Maske (Alternative: laufe mehrmals mit dem Operator über das Bild)



6. Resultatbild (ein Bild, in dem nur die „balls“ als Scheiben zu sehen sind.)in einer Datei abspeichern, aber nicht als Sink realisieren, sondern nach der Abspeicherung das unveränderte Bild weiterleiten.

7. die Scheiben zählen, ihre Zentren (Centroid, siehe unten) bestimmen, und prüfen, ob sie im Toleranzbereich der Qualitätskontrolle liegen. Letztere Information wird bei Erzeugung des Filters im "main" als Initialisierungsdaten an das Filterobjekt übergeben. Resultat in eine Datei schreiben. **Auf dem ILIAS liegt bereits ein fertig implementierter Filter!**

In der Realität hätte nun der Benutzer die Möglichkeit, diese Anwendung anhand mehrerer Bilder der Platine (Position unter der Kamera variiert, Lötstellen in Ausmaß, exakter Position, Helligkeit variieren) manuell zu testen, um Parameterwerte und ausgewählte Operatoren zu validieren. Dieser Schritt wird der Einfachheit halber in unserem Übungsbeispiel nicht vollzogen.

Konkrete Aufgabenstellung:

a) (5 Punkte)

Erstellen Sie eine Anwendung in einer Pipes&Filters Architektur in Java mit den Bildoperatoren als Filter mit Hilfe von JAI (Java Advanced Imaging, Dateien dazu auf dem Ilias) oder Catalano, und zwar im "Push" und im "Pull" Verfahren (beides soll laufen). Im letzten Filter sollen die Mittelpunkte der "balls" bestimmt, mit Sollwerten verglichen, und das Resultat in einer einfachen Textdatei abgespeichert werden. Für die Bestimmung der Zentren der Lötstellen in Pixelkoordinaten liegt bereits ein Filter in Source-Code auf dem ILIAS.
Benutzen Sie für das Ganze das Framework aus Aufgabe 1. Die konkreten Bildverarbeitungsfilter sollen nur speziellen Code für ihren speziellen Bildverarbeitungsoperator enthalten, und die allgemeine Filterfunktionalität (der Push- und Pull-Mechanismus z.B.) soll auf einer Ebene abstrakter Klassen implementiert sein.
Es reicht, wenn die Pipeline von einer Klasse im main gestartet werden kann (keine GUI nötig: wir werden zu einer komponentenbasierten GUI in der Folgeaufgabe kommen).

b) (2 Punkte)

Erweitern Sie die Anwendung so, dass sie in mehreren Threads laufen kann. Fügen Sie dazu an den Schnittstellen der Threads synchronisierende Pipes ein, z.B. mit Java.IO.PipedInputStream und Java.IO.PipedOutputStream. Kann man eine Performanceverbesserung feststellen? Lassen Sie dazu das Löststellenbild 50-mal durchlaufen und messen Sie die Zeit.

Deliverables: Source-Code und Executables und eine einseitige Textbeschreibung, wie Sie es gelöst haben..