Übung 2, Bildverarbeitung in einer „Pipes & Filters“-Architektur

Doku – Benutzungsszenario eines Bildverarbeitung-Produktes

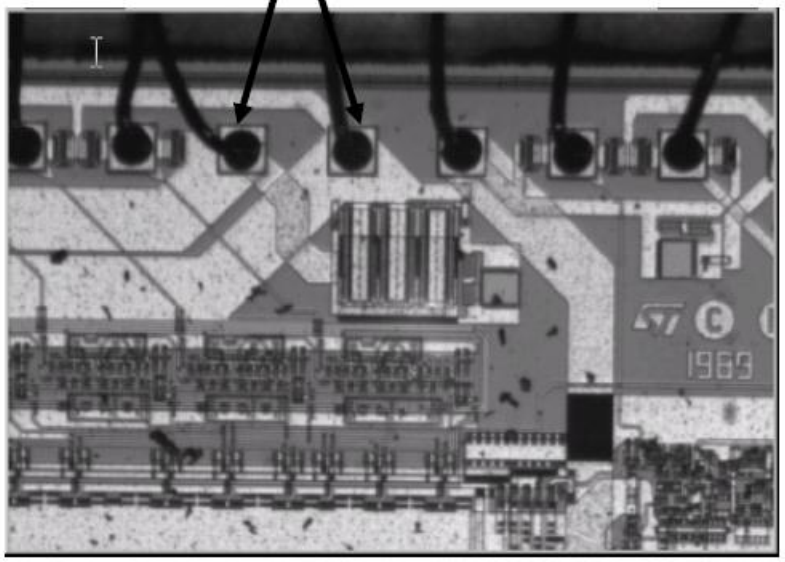
By Neumann, Rafael und Feurstein, Raphaela

1 Inhalt

2 Aufgabenstellung

Qualitätskontrolle – Bestimme, ob alle „ball bonding“-Anschlüsse (Lötstellen) vorhanden und richtigpositioniert sind. → Das heißt, ob deren Positionen in einem zulässigen Toleranzbereich liegen.

„ball bonding“



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 Qualitätskontroll-Programm zu erstellen, definiert der Benutzer nun Schritt für Schritt eine Bildanalyse, wobei jeder Schritt in einem Filter einer “Pipes & Filter“-Architektur zu erfolgen hat.

3 Schritte der Durchführung

3.1 Pull-Pipeline Image Processing

In der Pull-Pipeline Image Processing wird eine Float-Kernelmatrix erstellt, mit deren Hilfe die Aufgabenstellung abgearbeitet werden kann. Bevor einer dieser Schritte sinnvoll erarbeitet werden kann, werden mithilfe des Utils „Logical Operations“ die logischen Operatoren AND und OR definiert. Dafür werden die beiden Methoden PlanarImage andImage() und PlanarImage orImage kreiert. Dies soll später helfen die Bilder zu filtern, um einen Nachher-Vorher-Vergleich der Veränderungen im eingelesenen Bild „loetstellen.jpg“ zu erstellen.

- ***andImage() ...***
  - ... erstellt einen Parameter-Block, in diesen die beiden vergleichenden Bilder hinzugefügt werden.
  - ... verbindet die beiden Bilder zu einem einzigen Bild.

- ***orImage() ...***
  - ... erstellt ebenfalls einen Parameter-Block, in das die beiden vergleichenden Bilder gespeichert werden.
  - ... verwendet entweder das erste oder das zweite Bild.

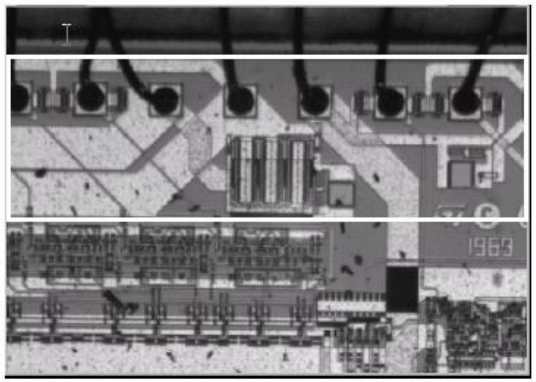
3.1.1 Das Bild laden und visualisieren.

3.1.2 Eine ROI (Region of interest) definieren

Das Originalbild wird in der Pull-Pipeline mithilfe des Pull-Filters Image Source eingelesen.

```
ImageSourcePullFilter imgSrc = new ImageSourcePullFilter(null, "loetstellen.jpg",0, 60, 450, 50);
```

- ***Der Pull-Filter Image Source ...***
  - ... erbt von der abstrakten Klasse Pull-Filter die abstrakte Methode Planar Image, um Daten aus der Pull-Pipeline zu lesen und retour zu schicken.
  - ... beinhaltet zwei Konstruktoren, die helfen die Daten des Bildes einzulesen.
  - ... definiert durch die Methode PlanarImage process die ROI, die einen Teil des Originalbildes einliest und zurück schickt. Dabei hilft das Util „Region of Interest“ dem Pull-Filter Image Source mit einem Rechteck das eingelesene Bild mit der Methode getregionOfInterst() so zu stutzen wie es durch den User definiert wurde.



Damit das Bild visualisiert werden kann, wird es der Pull-Pipe Planar Image übergeben.

```
PlanarImagePullPipe sourceImgPipe = new PlanarImagePullPipe(imgSrc);
```

- ***Die Pull-Pipe Planar Image …***
  - … erbt von der abstrakten Klasse Pull-Pipe, um die gefilterten Daten des Pull-Filters Image Source aus der Pull-Pipeline zu erhalten.

Anschließend wird diese Pull-Pipe mithilfe der statischen Methode displayImage(), die vom Util Display Image erzeugt wird, gelesen und visualisiert.

```
displayImage(sourceImgPipe.read());
```

- ***Das Util Display Image …***
  - … erstellt ein Frame fürs Display, das durch die ContentPane einen Inhalt erhält.
  - … erstellt eine Instanz fürs DisplayJAI, die durch eine JScrollPane im ContentPane-Container gespeichert wird.
  - … erhält die Informationen über das Bild, die einem Textlabel JLabel im ContentPane-Container hinzugefügt werden.
  - … schließt das Programm mithilfe eines Beendigungsoperators, indem das Frame geschlossen wird.
  - … legt die Frame-Größe und die Sichtbarkeit des Frames fest.

3.1.3 Einen Operator zur Bildsegmentierung auswählen: Threshold Operator

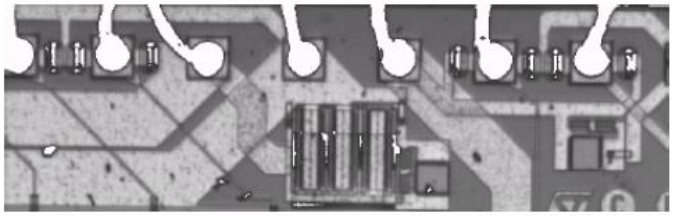
3.1.3.1 Parameterwerte des Operators wählen

Beschreibung:
Thresholding ist auch bekannt als Binärer Bildverstärkungskontrast, es liefert ein einfaches Mittel um die Grenzen eines Objektes zu definieren, die auf einem Hintergrundkontrast auftauchen. Der Threshold-Operator nimmt ein gereinigtes Bild und bildet alle Pixel in diesem Bild ab, die sich innerhalb des spezifizierten Bereichs befinden. Dieser Bereich wird mit einem minimalen und maximalen Wert spezifiziert.

Die Begrenzungen des sourceImgPipe-Objekts werden mithilfe des Pull-Filters Image Threshold herausgefiltert.

```
ImageThresholdPullFilter tresholdFilter = new ImageThresholdPullFilter(sourceImgPipe, 0, 47, 255, 3);
```

- ***Der Pull-Filter Image Threshold …***
  - … erbt ebenfalls wie der Pull-Filter Image Source die abstrakte Methode process() von der abstrakten Klasse Pull-Filter.
  - … definiert durch die Methode PlanarImage process() den Threshold-Operator, um die definierten Grenzen zu sichern. Dabei hilft das Util „Threshold“ dem Pull-Filter Image Threshold mit den gesetzten Operator-Parametern, die in einem Parameter-Block erstellt wurden, um später das veränderte Bild retour zu schicken.



Nun müssen die gefilterten Daten des Pull-Filters Image Threshold an die Pull-Pipe Planar Image übergeben werden, damit die Pull-Pipeline Image Processing mithilfe der statischen Methode displayImage() – Util „Display Image“ – die Daten visualisieren kann.

```
PlanarImagePullPipe imgThresholdPipe = new PlanarImagePullPipe(tresholdFilter);
displayImage(imgThresholdPipe.read());
```

3.1.4 Beseitige lokale Störungen (z.B. schwarzer Fleck im 2. Anschluss von rechts): z.B. Median Filter

3.1.4.1 Wähle Parameter des Filters: Größe der Maske zur Medianberechnung

Beschreibung:

Ein Median-Filter wird dafür genutzt um lokale Störungen eines Bildes zu beseitigen. Lokale Störungen treten oft als zufällige helle oder dunkle Pixel auf, die sich über ein Bild verteilen. Diese Störungen sind normalerweise heller oder dunkler als die anderen Pixel des Bildes und können mithilfe eines Median-Filters einfach gefunden werden, indem sie mit den anderen Pixeln verglichen werden.

Die auftretenden Störungen im imgThresholdPipe-Objekt der Pull-Pipe Planar Image werden mithilfe des Pull-Filters Image Median herausgefiltert.

```
ImageMedianPullFilter medianPullFilter  = new ImageMedianPullFilter(imgThresholdPipe);
```

- ***Der Pull-Filter Image Median …***
  - .. erbt wie die anderen Pull-Filter davor die abstrakte Methode process() von der abstrakten Klasse Pull-Filter.
  - … definiert durch die Methode Planar Image process() den Median, der mithilfe der Methode medianFilter() sichert. Dabei hilft das Util „Image Filters“ mit der Methode Planar Image medianFilter(), um kleine störende Pixelgruppen zu entfernen bzw. beseitigen. Außerdem fügt es das Bild mithilfe der Methode Planar Image toGrayScale() einem Parameter-Block hinzu.

Damit nicht mit den beseitigen lokalen Störungen weitergearbeitet wird, werden die Daten an die Pull-Pipe Filtered Image übergeben.

```
FilteredImagePullPipe filterimagePullPipe  = new FilteredImagePullPipe(medianPullFilter);
```

- ***Die Pull-Pipe Filtered Image …***
  - … erbt wie die vorherige Pull-Pipe von der abstrakten Klasse Pull-Pipe, um die gefilterten Daten des Pull-Filters Image Median aus der Pull-Pipeline zu erhalten.

Anschließend wird das Objekt dieser Pull-Pipe durch die statische Methode displayImage() mit dem Util „Display Image“ eingelesen, um die Daten in der Pull-Pipeline Image Processing zu visualisieren.

```
displayImage(filterimagePullPipe.read());
```

3.1.5 Opening-Operator mit kreisförmiger Maske

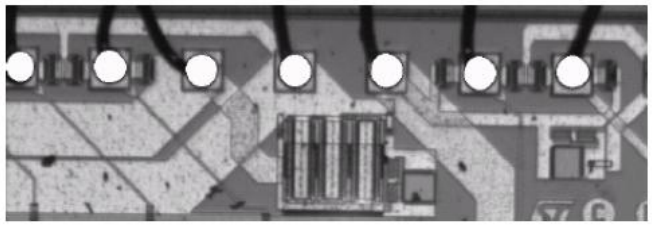
Es bleiben nur noch die Kabelanschlüsse der „balls“ übrig, bei der die Kreisform der Balls aus genutzt wird, was über die Benützung eines Opening-Operator mit kreisförmiger Maske bewirkt werden kann.

3.1.5.1 Wähle Parameter des Operators: Größe der Maske (Alternative: laufe mehrmals mit dem Operator über das Bild)

Beschreibung:

Der Opening-Operator ist einer der morphologischen Operatoren, die eingesetzt werden, um die Gestalt eines Bildobjektes zu verändern, damit die Störungen nach einer Segmentierung beseitigt werden können.

Beim Opening-Operator wird zuerst die Erosion durchgeführt, die sich zum Ziel gesetzt hat alle (Teil-)strukturen, die kleiner als das Strukturelement sind zu entfernen. Anschließend in der Dilatation wird die ursprüngliche Größe des Bildobjektes mit Ausnahme der vollständig entfernten Teilstrukturen wiederhergestellt.



Dem Pull-Filter Image Open wird nicht nur das filterimagePullPipe-Objekt der Pull-Pipe Filtered Image übergeben, sondern auch die Kernelmatrix und deren festgelegten Größe.

```
ImageOpenPullFilter imageOpenFilter = new ImageOpenPullFilter(filterimagePullPipe, kernelMatrix, 11);
```

- ***Der Pull-Filter Image Open …***
  - … erbt wie die vorherigen Pull-Filter die abstrakte Methode process() von der abstrakten Klasse Pull-Filter.
  - … legt die benötigten Parameter für die Pull-Pipeline Image Processing fest.
  - … definiert durch die Methode PlanarImage process() die Schritte Erosion und Dilatation des Opening-Operators, um sie zu sichern. Dabei übergibt das Util „Dilatation And Erosion“ dem Pull-Filter Image Open, was die Open-Operation bei der

Bildausgabe dieser Operation aus der genutzten Kernelmatrix benötigt, um die Erosion mit der Methode PlanarImage doErode() und Dilatation mit der Methode PlanarImage doDeliate() durchdurchzuführen.

Nun wird das imageOpenFilter-Objekt des Pull-Filters Image Open mithilfe der statischen Methode displayImage() mit dem Util „Display Image“ eingelesen, um es in der Pull-Pipeline Image Processing zu visualisieren.
displayImage(imageOpenFilter.read());

Erst jetzt wird der Pull-Pipe Balled Image das imageOpenFilter-Objekt übergeben.
BalledImagePullPipe balledImagePullPipe = new BalledImagePullPipe(imageOpenFilter);

- ***Die Pull-Pipe Balled Image …***
  - … erbt wie die vorherigen zwei Pull-Pipes von der abstrakten Klasse Pull-Pipe, um die gefilterten Daten des Pull-Filters Image Open aus der Pull-Pipeline Image Processing zu erhalten.

3.1.6 Resultatbild abspeichern bzw. an ein Programmteil weiterzureichen

Beschreibung:
Das Resultatbild ist ein Bild, in dem nur die „balls“ als Scheiben zu sehen sind. Es soll abgespeichert bzw. an ein Programmteil weitegereicht werden, das die Scheiben zählt, ihre Zentren bestimmt, und prüft, ob sie im Toleranzbereich der Qualitätskontrolle liegen. Optioinal: „balls“ über das Originalbild anzeigen (display).

Das balledImagePullPipe-Objekt der Pull-Pipe Balled Image wird dem Pull-Filter Mark Centers übergeben.
MarkCentersPullFilter markCentersFilter = new MarkCentersPullFilter(balledImagePullPipe);

- ***Der Pull-Filter Mark Centers …***
  - … erbt wie gehabt von der abstrakten Klasse Pull-Filter die abstrakte Methode process().
  - … definiert durch die Methode PlanarImage process() die Mittelpunkte und Zentren der Scheiben („balls“). Dabei hilft das Util „Find Middle Points In White Balls“ mit der Methode PlanarImage getMiddlePointsAndMarkMiddle() die Pixel zu finden, die in den Farbbereich von Weiß und Hellgrau (240 – 255) fallen, um sie zu sammeln und ihre Zentrum zu finden.

Dafür geht die Methode horizontal und vertikal durch das Raster, um die Farbe jedes Pixels zu überprüfen und falls sie in den definierten Bereich fallen, sollen sie der Pixelsammlung hinzufügt werden. Weiteres überprüft sie, ob eine Pixelsammlung vollständig ist, um das Zentrum der weißen Pixel mit einem blauen Punkt hervorzuheben und die Koordinaten auszugeben, die in eine Textdatei geschrieben werden.

Das Util „Find Middle Points in White Balls“ berechnet auch das Zentrum in einem Set aus weißen Pixeln mit der Methode Point getCenter(). In dieser wird das Minimum und Maximum der Koordinaten in X- und Y-Richtung gefunden und die Hälfte der Distanz verwendet, um die Koordinaten in einem Point-Objekt zu übergeben.

3.1.7 Schrittfolge mit Parametereinstellungen abspeichern

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.

3.2 Push-Pipeline Image Processing

Alle Schritte, die in der Pull-Pipeline Image Processing eingelesen wurden, werden nun in die Push-Pipeline Image Processing geschrieben. Dafür wird erneut eine statische Float-Kernelmatrix erstellt.

```
static float[] kernelMatrix  = new float[]{
                0,0,0,0,0,1,0,0,0,0,0,
                0,0,0,1,1,1,1,1,0,0,0,
                0,0,1,1,1,1,1,1,1,0,0,
                0,1,1,1,1,1,1,1,1,1,0,
                1,1,1,1,1,1,1,1,1,1,1,
                1,1,1,1,1,1,1,1,1,1,1,
                1,1,1,1,1,1,1,1,1,1,1,
                0,1,1,1,1,1,1,1,1,1,0,
                0,1,1,1,1,1,1,1,1,1,0,
                0,0,1,1,1,1,1,1,1,0,0,
                0,0,0,0,1,1,1,0,0,0,0
        };
```

Als erster Schritt wird eine Push-Pipe Output erzeugt.

```
OutputPushPipe outpushPipe = new OutputPushPipe(null);
```

- ***Die Push-Pipe Output …***
  - … erbt von der abstrakten Klasse Push-Pipe die abstrakte Methode write, die das Bild in unserer Frame Display Image schreibt.

Nun wird das outpushPipe-Objekt dieser Push-Pipe dem Push-Filter Mark Centers übergeben, der die Mittelpunkte und Zentren aller Pixel im Farbbereich Weiß bis Hellgrau (240 – 255) mithilfe des Utils „Find Middle Points In White Balls“ berechnet.

```
MarkCentersPushFilter markCenterPush = new MarkCentersPushFilter(outpushPipe);
```

Die geschriebenen gefilterten Daten des markCenterPush-Objektes des Push-Filters Mark Centers werden der Push-Pipe Planar Image übergeben, um sie für den nächsten Filter zu schreiben.

```
PlanarImagePushPipe pp1 = new PlanarImagePushPipe(markCenterPush);
```

Weiteres übernimmt nun der Push-Filter Image Open die Daten aus dem pp1-Objekt der Push-Pipe Planar Image, verwendet die Float-Kernelmatrix und legt deren Größe fest.

```
ImageOpenPushFilter imageOpenPushFilter = new ImageOpenPushFilter(pp1, kernelMatrix, 11);
```

Das Objekt imageOpenPushFilter des Push-Filters Image Open beinhaltet die geschriebenen Daten, die der Push-Pipe Planar Image übergeben werden.

```
PlanarImagePushPipe pp2 = new PlanarImagePushPipe(imageOpenPushFilter);
```

Das pp2-Objekt wird dem Push-Filter Image Median übergeben, damit dieser die lokalen Störungen in die Push-Pipeline Image Processing schreibt.

```
ImageMedianPushFilter medianPushFilter = new ImageMedianPushFilter(pp2);
```

Wie vorher wird auch das medianPushFilter-Objekt vom Push-Filter Image Median der Push-Pipe Planar Image übergeben.

```
PlanarImagePushPipe pp3 = new PlanarImagePushPipe(medianPushFilter);
```

Die gespeicherten Daten in der Push-Pipe Planar Image werden nun dem Push-Filter Image Threshold übergeben, damit er mithilfe den definierten ROI des Benutzers einen gewählten Pixelbereich des Bildes ausschneidet.

```
ImageThresholdPushFilter thresholdPushFilter = new ImageThresholdPushFilter(pp3, 0, 47, 255, 3);
```

Auch der ausgeschnittene Bildbereich, der das thresholdPushFilter-Objekt des Push-Filters Image Threshold beinhaltet, wird an die Push-Pipe Planar Image übergeben.

```
PlanarImagePushPipe pp4 = new PlanarImagePushPipe(thresholdPushFilter);
```

Zu guter Letzt wird das pp4-Objekt der Push-Pipe Planar Image an den Push-Filter Image Source übergeben, damit dieser das Bild einliest und es durch die Methode write() in die Push-Pipeline schreibt.

```
ImageSourcePushFilter imgSourcePush = new ImageSourcePushFilter(pp4, "loetstellen.jpg",0, 60, 450, 50);
imgSourcePush.write();
```

4 Diagramme

4.1 UML-Package-Diagramm