반도체 칩 이미지검출 및 인식 기술을 이용한 계수 정확도 향상에 관한 연구 (openCV, YOLOv4)

박진우, 이선우*, 유하얀**,박호민***,김양중

한국산업기술대 산업기술 경영대학원 SW융합공학과,
스마트팩토리융합학과*,(주)유씨티**,정보통신기술공학과***

jamespark@kpu.ac.kr

A Study on enhancing counting accuracy of semiconductor chip to detect and recognize with x-ray machine based on openCV, YOLOv4

PARK JIN WOO, LEE SUN WOO*,YU HA YAN**, PARK HO MIN***,

KOREA POLYTECHNIC Univ, UCT CO.,LTD,*

요 약

본 논문은 x-ray머신으로 이미지를 확보하여 반도체 칩 인식 정확도 향상을 위해 openCV, YOLO v4 알고리즘을 사용하였다. 칩 릴 테이프를 선 검출 후 칩 영역을 검출하기 위해 3가지 방법으로 이미지인식을 진행한 이유는 openCV Match, labelling,, YOLOv4의 각각 활용하여 결과를 알아보고 장점과 단점을 알고자 하였다. 가장 계수능력이 좋은 방법을 찾아 설득력이 있는 방법을 찾고자 한 것이다. 생산현장에서 릴 형태의 반도체 칩을 검출하는 것은 창고관리와 재고관리를 하는데 자재관리 시간을 절약하고 계수 정확도 향상하는 데 있어 매우 중요한 개선방법이다. 따라서, 반도체 릴에 있는 수량파악을 정확하게 해낼 수 있다면 인력과 시간을 줄일 수 있는 중요한 포인트이다.

스마트 공장의 일환으로 x-ray머신을 이용한 계수기를 도입하였으나 계수 정확도 면에서 만족할 만한 성과를 보이지 못했으며 오차가 발생하였다. 현재 update를 하고 있으나 각 릴마다의 오차는 재고관리에서 수량의 오차를 발생하게 되어 실물과 일치하지 않는다. 이미지를 읽고 계수의 능력을 향상하기 위해서 최적의 방법을 찾는 것은 중요한 연구과제이다. coding을 실시하여 결과를 검토하였다. openCV매칭 ,openCV 라벨링으로서는 인식능력에 한계를 보였으며, 상용화단계에 이르기까지는 많은 단계의 프로그램 구성이 필요하였다. YOLO-V4를 이용하여 반도체 칩 릴 테이프 인식 과정을 모의실험한 결과, openCV매칭,openCV라벨링에서 수작업했던 부분의 단점을 보완하여 계수를 정확하게 할 수 있었다. 본 논문의 연구 결과를 통해 알 수 있듯, YOLO 기반 X-ray 머신 이미지 반도체 칩 검출 및 인식 기술은 반도체 릴 테이프 칩 계수 속도 향상에 이바지할 수 있을 것으로 기대한다.

Ⅰ. 서 론

본 논문에서는 YOLOv4알고리즘을 사용하여 릴 테이프의 반도체 칩 계수 능력을 향상했다.

반도체 칩 계수를 위해 YOLOv4를 이용하기전에 openCV를 이용하여 이미지 검출의 방법을 코딩으로 분석해 보고 그 결과를 확인해 보고자 하였다.단순히 YOLOv4를 운영하였을 때 생기는 알고리즘에만 의존하지 않고 이미지검출 부분에서 기초자료를 확보하고 이미지확보를 위해 어떤 코드가 사용되었는지를 직접 실습해보고, 이를 이용해 결과값을 도출하여 실제 결과가 이미지 계수에 영향을 주는지 유의미성을 알고자 하였다.이것은 기초 이미지 데이터를 순차적으로 확보해 나가는 과정을 포함하여 이미지 인식을 통한 과정과 결과를 비교하고자 한 것이다..

본 논문은 객체 검출 관련 연구에 관해서 서술하였으며. 또한, 모의실험 상황과 실험 결과를 설명하였다. 마지막에서 결론을 제시했다.

Ⅱ. 본론

본 논문은 다음과 같이 연구 분야에 대한 이론 및 실습으로 접근을 하였다.

A. openCV : matching

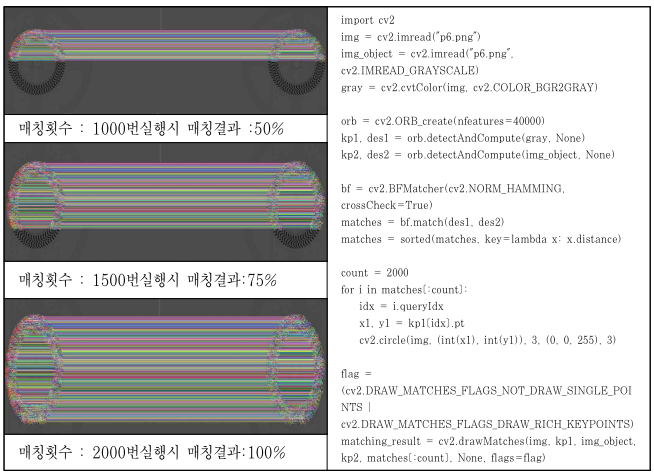
학습이 아닌 객체인식을 위해서는 특징점(Key Point)과 기술자(Descriptor)를 활용한다. 특징점은 영상에서 배경과 구분되면서 고유한 실별지점을 의미하며, 서로 다른 이미지에서도 하나 이상의 지점이 특별하게 구분할 수 있는 작은 부분을 가리킨다.기술자는 서로 다른 이미지에서 특징점이 어떤 연관성을 가졌는지 구분하게 한다.기술자는 각 특징점이 가진 지역적 특정 정보를 갖고 있다.서로 다른 특징점에서 차이를 구분해 특징점끼리 서로 매칭되게 한다.

전수 조사 매칭클래스는 두가지 매개변수만 활용해 객체를 인식할 수 있다.거리측정법(normType)에는 질의기술자(Query Descriptors)와 훈련기술자(Train Descriptors)를 비교할 때 사용되는 거리계산 측정법을 지정한다. 거리측정법의 수식은 다음과 같다.

플래그	수식
cv2.Norm_L1	$dist(\vec{a},\vec{b}) = \sum_i abs(a_i - b_i)$
cv2.Norm_L2	$dist(\vec{a},\vec{b}) = [\sum_i (a_i - b_i)]^{\frac{1}{2}}$
cv2.Norm_L2SQR	$dist(\vec{a},\vec{b}) = \sum_i (a_i - b_i)]^2$
cv2.Norm_Hamming	$dist(\vec{a},\vec{b}) = \sum_i (a_i = = b_i)^2 ? 1 : 0$
cv2.Norm_Hamming2	$dist(\vec{a},\vec{b}) = \sum_i (a_i = = b_i)^2$ and $(a_{i+1} = = b_{i+1})] ? 1 : 0$

img변수는 검출하려는 객체가 포함된 이미지이며,img_object는 객체이미지이다. 이미지를 불러와 최대 피처수 (nfeatures)를 50,000으로 설정하였고,매칭을 더 많이 실행하면 상대적으로 정확도가 올라간다.매칭은 유효거리가 짧을수록 품질이 우수하다.특징점,색인,거리를 알았다면 이미지 위에 시각적으로 표시할 수 있다.특징점 매칭 그리기함수를 활용하여 매칭그리기 진행하였다.

매칭횟수를 각각 1000,1500,2000번으로 진행하였을 때 매칭이 더 성공적으로 이뤄지고 있다.하지만 매칭하고자 하는chip뿐 아니라 Reel의 형태까지도 특징으로 잡아 매칭하였으며 원하는 chip개수 결과를 나타내주지 못하였다.

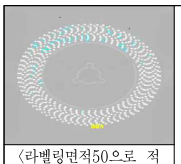



```
import cv2
img = cv2.imread("p6.png")
img_object = cv2.imread("p6.png",
cv2.IMREAD_GRAYSCALE)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

orb = cv2.ORB_create(nfeatures=40000)
kp1, des1 = orb.detectAndCompute(gray, None)
kp2, des2 = orb.detectAndCompute(img_object, None)

bf = cv2.BFMatcher(cv2.NORM_HAMMING,
crossCheck=True)
matches = bf.match(des1, des2)
matches = sorted(matches, key=lambda x: x.distance)

count = 2000
for i in matches[:count]:
    idx = i.queryIdx
    x1, y1 = kp1[idx].pt
    cv2.circle(img, (int(x1), int(y1)), 3, (0, 0, 255), 3)

flag =
(cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POI
NTS |
cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
matching_result = cv2.drawMatches(img, kp1, img_object,
kp2, matches[:count], None, flags=flag)
```

B. openCV : labelling

라벨링(Labeling)이란? 이진화 한 이미지에서 객체를 각각 분별하기 위해 인접한 픽셀 값들끼리 그룹화하여 번호를 매긴 것이다.즉, 인접한 화소들을 묶어 하나의 객체로 판단하는 방식이며,라벨링을 하면 한 장의 이미지에 있는 흰색 영역들을 개별적으로 다룰 수 있다.

area의 영역을 50,10으로 적용하였을 때 작게 할수록 라벨링은 전체부품에 좋은 결과를 나타나게 되었다. 하지만 x-ray계수기에 적용하기 위해서는 좀 더 안정적인 구조를 가져야 한다. 즉 이미지가 투입되었을 때 이미지영역을 인식하고 이미지를 처리해야 한다는 점에서 YOLO -V4를 이용하여 적용하고자 하였다.




```
contours = cv.findContours(img_edge.copy(), cv.RETR_LIST,
            cv.CHAIN_APPROX_SIMPLE)
cv.drawContours(img_edge, contours[0], -1, (0, 0, 0), 1)
cv.namedWindow("result", cv.WINDOW_NORMAL)
cv.imshow("result", img_edge )
cv.waitKey(0)
nlabels, labels, stats, centroids = cv.connectedComponentsWithStats(img_edge)
for i in range(nlabels):
    if i < 2:
        continue
    area = stats[i, cv.CC_STAT_AREA]
    center_x = int(centroids[i, 0])
    center_y = int(centroids[i, 1])
    left = stats[i, cv.CC_STAT_LEFT]
    top = stats[i, cv.CC_STAT_TOP]
    width = stats[i, cv.CC_STAT_WIDTH]
    height = stats[i, cv.CC_STAT_HEIGHT]
    if area >50,20:
        cv.rectangle(img_color, (left, top), (left + width, top + height),
                     (0, 0, 255), 1)
    #cv.circle(img_color, (center_x, center_y), 5, (255, 0, 0), 1)
```

C. YOLO-V4

YOLO는 Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi가 실시간 객체 검출을 위하여 개발한 합성곱 신경망 네트워크이다. YOLO는 You Only Look Once의 약자이다.

YOLO는 객체 인식 문제를 회귀분석 문제로 전환한다. YOLOv4 의 작업을 요약하면 1)효율적이고 강력한 object detector를 1080Ti, 2080Ti같은 환경에서도 빠르고 정확하게 훈련시킬 수 있다.2) detector의 훈련과정에서 Bag-ofFreebies 와 Bag-of-Specials methods 의 영향을 검증한다.

3) SOTA의 method를 수정하고 만듦으로써 single GPU 훈련에서도 적합하고 효율적이게 한다. 이 과정에는 CBN,PAN,SAM 같은 기술들이 포함된다.YOLOv4의 주된 목표는 제품 시스템에서 빠른 작동속도를 가져가면서도 연산량(BFLOP)의 숫자를 줄인다기보다 병렬적으로 최적화된 계산이 가능할 수 있게 하는 것이다.

D. 본 논문에서의 모의 실험 상황과 실험 결과

D-1.실험장비 구성은 아래 표<표 2-1> 과 같다.

<표 2-1> 실험장비

그림 1 x-ray이미지와 x-ray계수기 릴 장착화면

장비	모델
㈜유씨티 AI서버	서버이름:AI Service OS : Microsoft 서버 with Tensorflow (64-bit) 서버사양 : 인텔 제온 스케일러블 실버 4210 Dual (64bit , L3캐시 : 13.75MHz, 20스레드 ,3.2GHz) RAM: 삼성전자 DDR4-2666 128G VGA: 지포스 RTX 3090 24GB SSD: 삼성전자 860 EVO (2TB)
X-RAY머신	Haweye-1000
YOLO 버전	Darknet for YOLOv4
기타	OpenCV 3.4.0, labelImg-master

본 논문에서 YOLO 적용 딥러닝 기반 예측 모델 개발 순서는 딥러닝 모델을 학습하고, 이를 통해 사용자 데이터에서 객체 예측 모델을 개발하는 1. 응용 목적 정의 2. 알고리즘 선정 및 환경 구성 3. 학습 데이터 준비 4. 데이터 라벨링 5. 딥러닝 학습 6. 학습 결과 검증 및 재 학습 순서로 진행했다.

첫째, 본 논문에서는 YOLO 단일 이미지 확보를 위해 360도 회전하여 python코딩으로 이미지를 360개 생성한다.

```
for import time
import os
from PIL import Image
import sys
image_filename = sys.argv[1:]
el in FILELIST:
    for i in range(360):
        if COUNT > 1000:
            break
        image = Image.open(1.png)
        new_temp_name = "%05d.png" % COUNT
        COUNT += 1
```

```
        image = image.rotate(i+1)
        image = image.resize((Xdim, Ydim))
        image.save(out_dir + "/" + new_temp_name)
        image.close()
    print("Process Done.")
```

반도체 Reel이미지가 너무 작아서 확대하여 360개의 그림화일을 생성하였으며 라벨링을 진행하였다.

둘째, 학습 데이터 전처리 과정으로서의 이미지 라벨 작업을 위해 이미지 라벨 툴 labelImg-master, Yolo_Label-master, Yolo_mark-master 등을 사용했으며, 최종적으로 다음 그림[2-3]처럼 labelImg-master을 이용해 이미지 라벨 작업을 했다.

Darket을 통한 학습과정과 결과는 다음 순서에 따랐다.

1. C:\DEV\darknet-master\build\darknet\x64\data\img 폴더에 라벨링 데이터 & 이미지 파일 복사 01-1.jpg , 01-1.txt
2. C:\DEV\darknet-master\build\darknet\x64\data 에서 obj.data 파일 확인 classes= 1
   train = data/train.txt
   valid = data/validation.txt
   names = data/obj.names
   backup = backup/
3. C:\DEV\darknet-master\build\darknet\x64\data 에서 obj.names chip (classes = 1)
4. C:\DEV\darknet-master\build\darknet\x64\data 에서 train.txt C:/DEV/darknet-master/build/darknet/x64/data/img/01JPG
5. Anaconda Prompt(바탕화면) 에서 관리자 권한으로 실행
6. Ankeker data 계산
   ./darknet.exe detector calc_anchors data/obj.data data/yolov4-uct.cfg -num_of_clusters 9 -width 416 -height 416
7. C:\Users\User\YoloTr>jupyter notebook Day_20200727_YoloGPUTest.jpynb 실행
   1) cd C:\DEV\darknet-master\build\darknet\x64 (shift Enter 실행)
   2) !dir
   3) !darknet detect cfg/yolov4.cfg yolov4.weights data/dog.jpg (Yolo 확인)
   4)!darknet detector train data/obj.data yolov4-uct.cfg darknet53.conv.74 (Yolo 학습)
   5)!darknet detector test data/obj.data yolov4-uct.cfg backup/yolov4-uct_last.weights data/img/0001.png (추론)

학습이 이루어지는 동안 매100번 마다 학습 된 weights 파일이 생성되며 한 파일당 약 260MB의 용량을 가졌다. 본 연구는 총 2,000번 학습하고자 계획, 종료된 학습 시점의 weights 파일을 이용하여 예측할 수 있었다.

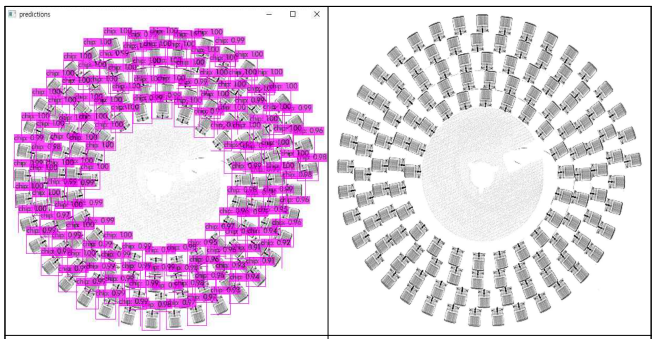
객체 검출 정확도는 다음 식(1)을 통해 계산했다.

$$Precision = \frac{TP}{TP+FP}$$

D-2. 실험 결과

성능 평가는 YOLO v3를 이용하여 학습하여 형성된 weights 파일을 이용하여 립 테이프에 있는 칩 검출 성능을 평가하였다.

지표	OpenCV Matching	OpenCV labelling	YoloV4
검출능력 Precision	100% (2000회 이상)	90% (이미지 크기20이하)	100%



chip이미지 검출결과:100% | 원본이미지 Chip Reel

실험 결과 Reel 부품 적용시에는 이미지에 대한 구체적인 라벨링 선행 작업이 필요함을 알았으며, 이를 위해 색상과 물체구성에 대한 차별적인 특징이 포함된 이미지를 구현해야 한다.

Ⅲ. 결론

본 논문에서는 다양한 형태의 반도체 칩 인식 계수 시간 향상을 위해 OpenCV함수를 이용하여 검출을 시도하였으며, YOLOv3 알고리즘을 사용하여 비교하고자 하였다. 기존 릴 테이프의 반도체 칩 검출은 계수 시간이 최소 수분이 소요되었으나, 본 논문 결과에서의 OpenCV TEST결과 계수 시간은 비교가 불가할 정도로 속도가 빨라졌음을 확인하였다. 그러나 이미지를 얻어내는 방법과 이미지 구성이 중요하여 예측 정확성 문제는 실험 환경의 개선과 이미지를 자동으로 라벨링하는 프로그램을 연계하여 학습을 수행해야 할 것이다.

향후 본 논문에서 추가로 연구를 할 경우, 이에 대한 고려도 필요하며, 반도체 칩 릴 테이프 이미지만 대상으로 하지 않고, 더 다른 분야로 연구를 넓히는 것도 필요하다며 고정형에서 이동형으로 좀더 작은 DEVICE에서 구현하도록 IMBEDED SW연구를 계속하고자 한다.

참 고 문 헌

[1] C#과 파이썬을 활용한 OpenCV4프로그래밍 , 윤대희 지음 , 위키북스,2021.4.15

[2] 알짜배기 예제로 배우는 OpenCV, 이정주,BJPUBLIC,2020.4.29

[3] YOLOV4: https://webnautes.tistory.com/1417