National Research University Higher School of Economics

Graduate School of Business

Programme «Big Data Systems»

**Project Report**

Defect detection with computer vision

Performed by student:

Victor E. Kozlovskiy .

(Name)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(Signature)*

**Project supervisor:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(position, full name)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(grade)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(Date) (signature)*

**Moscow 2021**

### Introduction

Historically, quality assurance has progressed through three development stages, moving from downstream to upstream of the entire manufacturing processes. These three stages are: inspection oriented quality assurance, statistical process control (SPC), and design oriented quality assurance. These methods successfully address the problem of the high cost of data collection and the requirement for human intervention to identify and solve problems. Today, these methods are state-of-the-art techniques and are still being used in industries.

Quality assurance started with inspection. Inspection was required for assembly when people started producing interchangeable parts. However, it would be too expensive to inspect every part in mass production. Moreover, if the inspection was destructive, there would be nothing left after inspecting every part. Inspection could also be very tedious for the inspectors during

In manufacturing, quality control is a process that ensures customers receive products free from defects and meet their needs. When done the wrong way, it can put consumers at risk.

Major recalls like these can be prevented through effective quality control in manufacturing. Some common tools used to support quality control include:

* Statistical process control (SPC) monitors and controls quality by tracking production metrics. It helps quality managers identify and solve problems before products leave the facility.
* Six Sigma uses five key principles to ensure products meet customers’ needs and have zero defects.

When supported by lean tools like Total Productive Maintenance (TPM), 5S, and Kaizen, most if not all defects can be eliminated.

### Defect detection with computer vision

In the manufacture of mechanical products in complex industrial processes, defects such as internal holes, pits, abrasions, and scratches arise, due to failure in design and machine production equipment as well as unfavorable working conditions. Products may also easily corrode and be prone to fatigue because of daily application. These defects increase the costs incurred by enterprises, shorten the service life of manufactured products, and result in an extensive waste of resources, thereby causing substantial harm to people and their safety. Hence, detecting defects is a core competency that enterprises should possess in order to improve the quality of the manufactured products without affecting production. Automatic defect-detection technology has obvious advantages over manual detection. It not only adapts to an unsuitable environment but also works in the long run with high precision and efficiency. Research on defect-detection technology can reduce the production cost, improve production efficiency and product quality, as well as lay a solid foundation for the intelligent transformation of the manufacturing industry. Therefore, many scholars have reviewed defect-detection-related technologies and applications to provide references for the application and research of defect-detection technology. Defect-detection technology is a hot topic in the industry and academia.

### Practice

Initial requirements:

* Web-camera
* Python
* OpenCV library

Two basic libraries are used in this work to create computer-vision script: NumPy and OpenCV.

**NumPy** is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

**OpenCV** (Open Source Computer Vision Library) is an open-source library that includes several hundreds of computer vision algorithms. Originally developed by Intel. The library is cross-platform and free for use under the open-source Apache 2 License. Starting with 2011, OpenCV features GPU acceleration for real-time operations.

**IMutils** is also required to make basic image processing functions such as translation, rotation, resizing, skeletonization, displaying Matplotlib images, sorting contours, detecting edges etc.

All needed packages are installed with pip manager:

pip install numpy

pip install opencv-python

pip install imutils

### Task 1

Dynamically recognize arrangement of simple objects in relation to their initial position. Indicate when a book is rotated at angle greater than some specified value.

To solve the problem the following approach is proposed:

1. Recognize objects using basic library operators
2. Recognize angle of object’s tilt
3. Capture video stream from web-camera
4. Create graphical output indicating expected changes

First we read image (figure 1) from static file, convert it to grayscale and blur to get rid of high-frequenced noises (figure 2):

gray = cv.cvtColor(image, cv.COLOR\_BGR2GRAY)

gray = cv.GaussianBlur(gray, (15, 15), 0)

cv.imwrite("output/gray.jpg", gray)



Figure 1 – Original static image



Figure 2 – Blurred image

Sobel operator (figure 3) showed more recognizable edges than Laplace operator (figure 4).



Figure 3 – Result of Sobel operator



Figure 4 – Result of Laplace operator

Get edged image using Canny operator (figure 5):

edged = cv.Canny(gray, 10, 250)

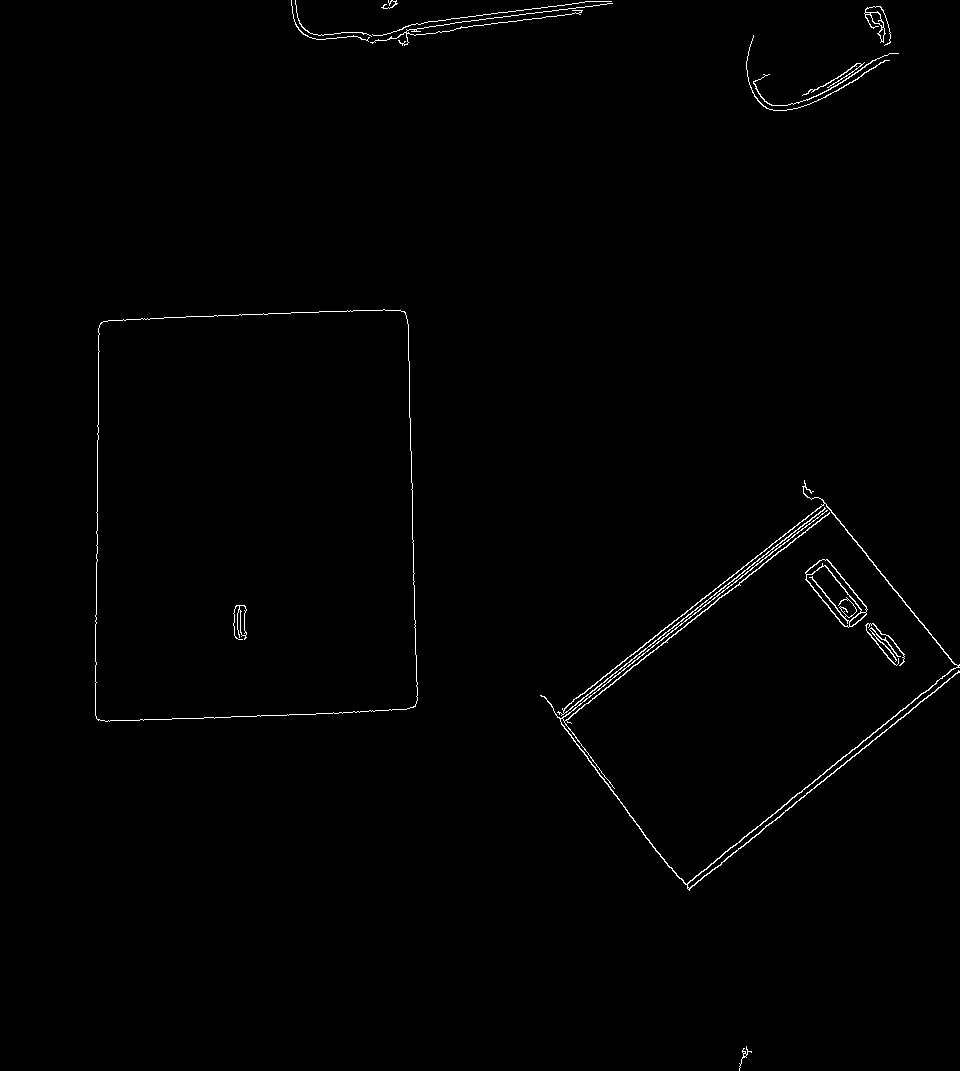


Figure 5 – Result of Canny operator

Make obtained contours closed (figure 6):

kernel = cv.getStructuringElement(cv.MORPH\_RECT, (7, 7))

closed = cv.morphologyEx(edged, cv.MORPH\_CLOSE, kernel)

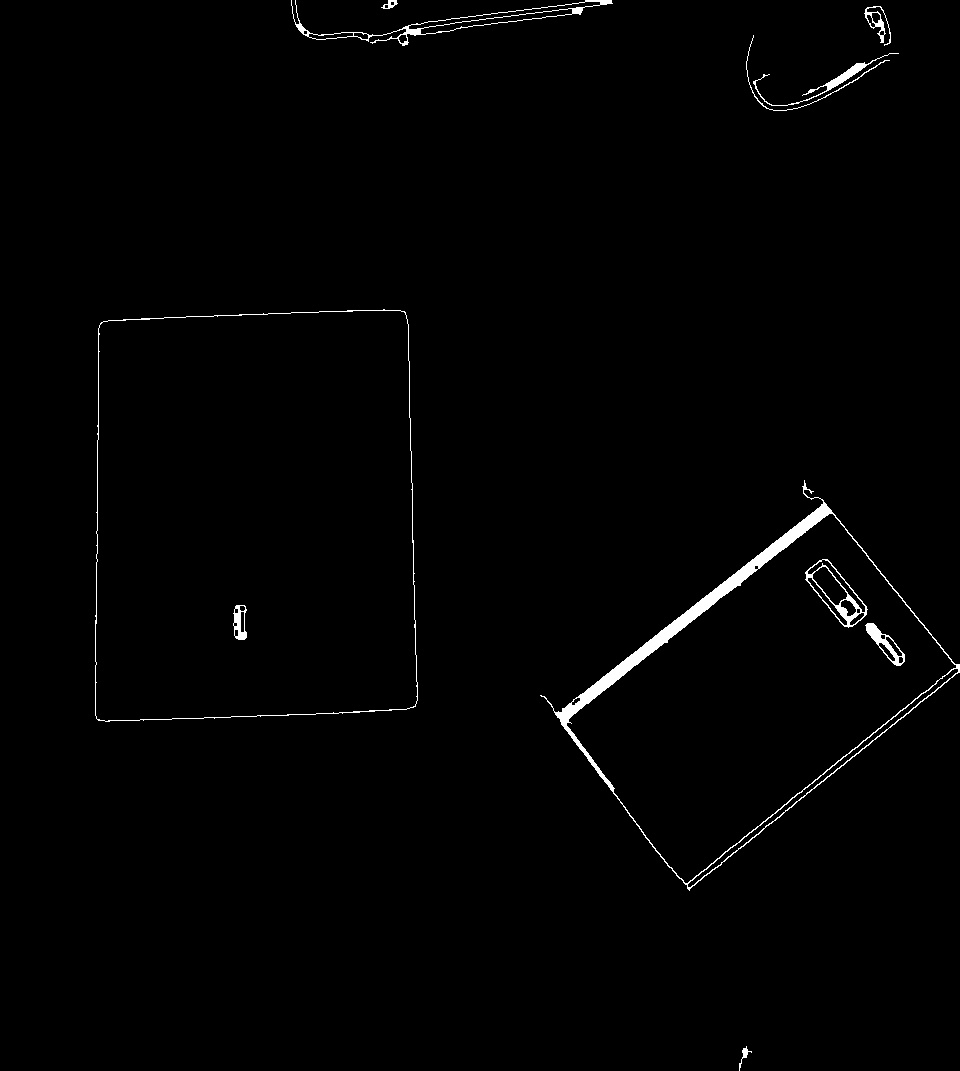


Figure 6 – Closed contours

Find separate contours with IMutils:

cnts = cv.findContours(closed.copy(), cv.RETR\_EXTERNAL, cv.CHAIN\_APPROX\_SIMPLE)

cnts = imutils.grab\_contours(cnts)

Then approximate each of found contours to closed polygon, approximate central line for ones that have 4 vertices, and calculate its tilt angle.

for c in cnts:

    peri = cv.arcLength(c, True)

    approx = cv.approxPolyDP(c, 0.1 \* peri, True)

    if (len(approx) in [4]):

        cv.drawContours(image, [approx], -1, (0, 255, 0), 4)

        rows,cols = image.shape[:2]

        [vx,vy,x,y] = cv.fitLine(c, cv.DIST\_L2, 0, 0.0001, 0.0001)

        lefty = int((-x\*vy/vx) + y)

        righty = int(((cols-x)\*vy/vx)+y)

        line\_start = (cols-1,righty)

        line\_end = (0,lefty)

        cv.line(image, line\_start, line\_end, (0,255,0), 2)

        angle\_pi = (righty - lefty)/(cols-1)

        angle = int(np.arctan(angle\_pi)\*180/np.pi)

        text\_start = (int(x[0]),int(y[0]))

        rect = np.array( [[[text\_start[0]-10,text\_start[1]],[text\_start[0]+50,text\_start[1]],[text\_start[0]+50,text\_start[1]-30],[text\_start[0]-10,text\_start[1]-30]]], dtype=np.int32 )

        cv.fillPoly(image, rect, (255, 255, 255), cv.LINE\_4)

        cv.putText(image, str(angle), text\_start, cv.FONT\_HERSHEY\_COMPLEX\_SMALL, 1, (0,0,255))

Central line and value of its angle are put to output image using OpenCV (figure 7).

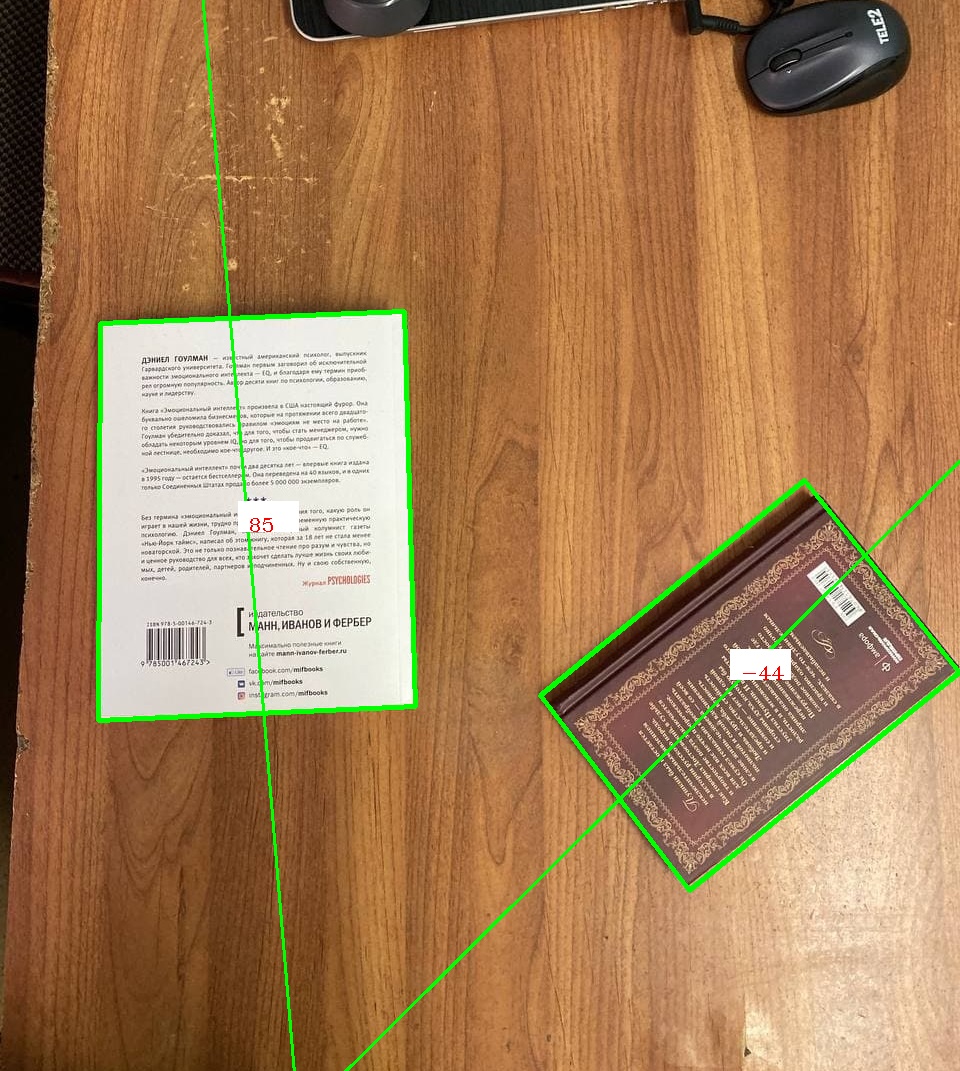


Figure 7 – Output image

Video stream from web-camera is handled with OpenCV’s VideoCapture function:

cap = cv.VideoCapture(0)

while True:

    ret, image = cap.read()

Quality of detection is worse for dynamic video-stream ­– redundant objects are being recognized and indicated (figure 8).

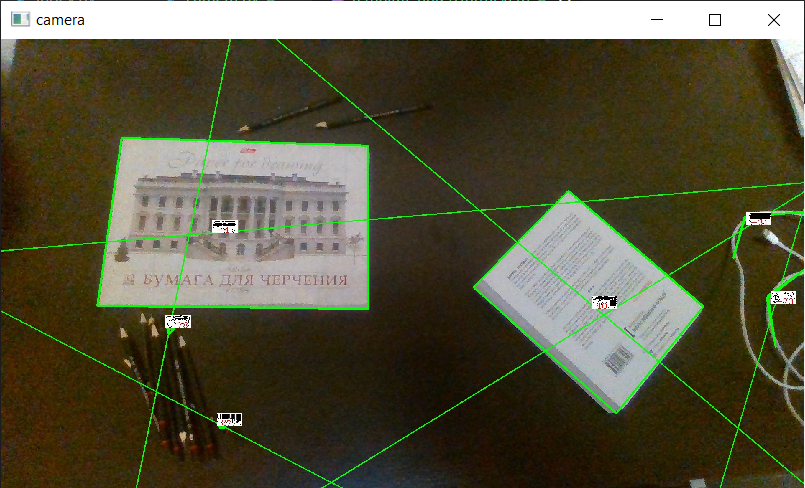


Figure 8 – Web-camera output image processed

### Task 2

Dynamically detect the appearance of some object based on image of its outline. Find location of a detail using its flat vector image.

To solve the problem the following approach is proposed:

1. Recognize contours on input image using basic library operators
2. Convert vector image of detail to a bitmap
3. Detect contours with the strongest matching with the detail’s bitmap
4. Create graphical output indicating found matchings

Initial idea how to match two images (original model and captured detail) was to use FASTnBRIEF method.

Oriented FAST and rotated BRIEF (ORB) is a fast robust local feature detector, that can be used in computer vision tasks like object recognition or 3D reconstruction. It is based on the FAST keypoint detector and a modified version of the visual descriptor BRIEF (Binary Robust Independent Elementary Features). Its aim is to provide a fast and efficient alternative to SIFT.FASTnBRIEF. The algorithm has been applied to a template image and a detail photo (figure 9):

import cv2

def FAST\_n\_BRIEF(img1, img2, n=1000, threshold=40):

    test1=cv2.imread(img1,0)

    test2=cv2.imread(img2,0)

    fast = cv2.FastFeatureDetector\_create(threshold=threshold)

    kp1=fast.detect(test1,None)

    kp2=fast.detect(test2,None)

    mark1=cv2.drawKeypoints(test1,kp1,None,color=(0,0,255),flags=0)

    orient1=cv2.drawKeypoints(test1,kp1,None,color=(0,0,255),flags=cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS)

    mark2=cv2.drawKeypoints(test2,kp2,None,color=(255,0,0),flags=0)

    orient2=cv2.drawKeypoints(test2,kp2,None,color=(255,0,0),flags=cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS)

    brisk = cv2.BRISK\_create(thresh=75)

    kp1, des1 = brisk.compute(test1, kp1)

    kp2, des2 = brisk.compute(test2, kp2)

    bf = cv2.BFMatcher(cv2.NORM\_HAMMING)

    matches = bf.match(des1,des2)

    matches = sorted(matches, key = lambda x:x.distance)

    print(len(matches))

    result = cv2.drawMatches(mark1,kp1,mark2,kp2,matches[:min(n, len(matches))],None,matchColor=[0,255,0], flags=2)

    cv2.imwrite("output.png",result)

FAST\_n\_BRIEF('../output/template.jpg', 'input.JPG')

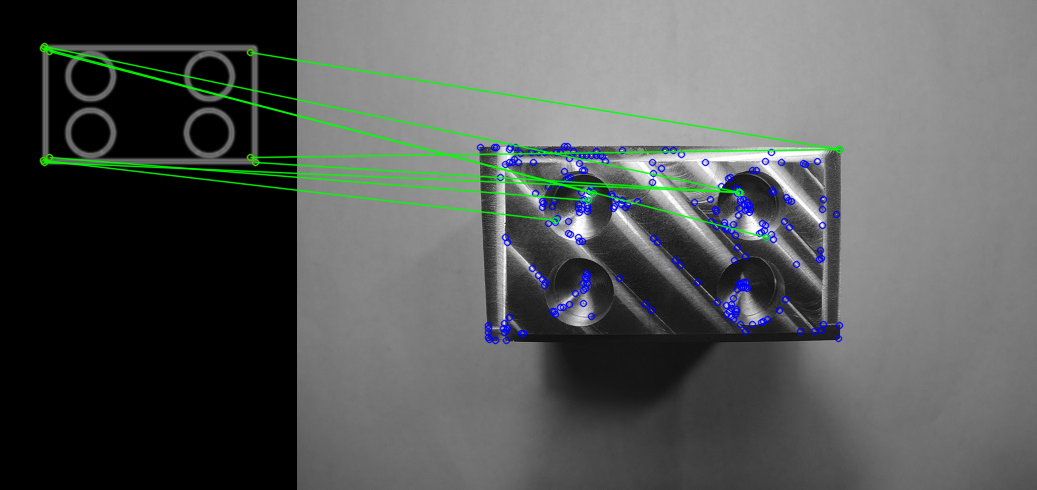


Figure 9 – FASTnBRIEF algorithm applied to pattern and detail photo

As it is seen in the figure corners of reference model are recognized as the most representative points. Also geneva waves create redundant features matching.

Function of input image edges finding:

def edge\_image(image):

    img = cv.medianBlur(image, 15)

    img = cv.Canny(img, 20, 40)

    return img

Function of circles finding:

def find\_circles(image):

    circles = cv.HoughCircles(image, cv.HOUGH\_GRADIENT, 1, minDist=20, param1=10, param2=20, maxRadius=50)

    circles = circles[0]

    print(circles)

    max\_radius = np.max(list(map(lambda c: c[2], circles)))

    circles = list(filter(lambda c: abs(c[2] - max\_radius) < 10, circles))

    return circles

Function of comparison of pattern embeddings:

def embedding\_diff(e1, e2):

    return abs(abs(e1) - abs(e2))

result = {

    0: "OK",

    1: "Item is not polished or too much input noise",

    2: "Two holes missed or incomplete",

    3: "One hole missed or incomplete",

    4: "Wrong holes geometry",

}

Function of result formulating:

def check\_defect(circles):

    if len(circles) < 2 or len(circles) > 4:

        return result[1]

    if len(circles) < 3:

        return result[2]

    if len(circles) < 4:

        return result[3]

    input\_embedding = dist\_matrix\_determinant(circles)

    difference = embedding\_diff(original\_embedding, input\_embedding)

    if difference > threshold:

        return result[4]

    return result[0]

Function of drawing found circles into input image:

def draw\_circles(file, image, c):

    for circle in c:

        cv.circle(image,(int(circle[0]), int(circle[1])), int(circle[2]), (255,0,17), -1)

    cv.imwrite('output/'+file, image)

Main program algorithm:

for filename in input\_filenames:

    img = cv.imread('input/'+filename, 0)

    edged = edge\_image(img)

    circles = find\_circles(edged)

    draw\_circles(filename, edged, circles)

    result\_text = check\_defect(circles)

    print(result\_text)

    # print to image

    text\_start = (5, 30)

    rect = np.array(

        [

            [

                [text\_start[0] - 10, text\_start[1]],

                [text\_start[0] + 500, text\_start[1]],

                [text\_start[0] + 500, text\_start[1] - 30],

                [text\_start[0] - 10, text\_start[1] - 30],

            ]

        ],

        dtype=np.int32,

    )

    cv.fillPoly(img, rect, (255, 255, 255), cv.LINE\_4)

    cv.putText(img, result\_text, text\_start, cv.FONT\_HERSHEY\_COMPLEX\_SMALL, 1, (0, 0, 255))

    cv.imwrite("output/result/"+filename, img)

After finding circles at edged input images (figure 10) verdict is formulated using one of 5 output options (figure 11).

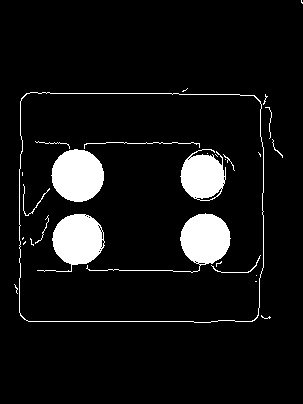
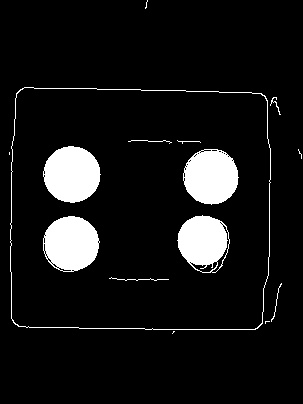
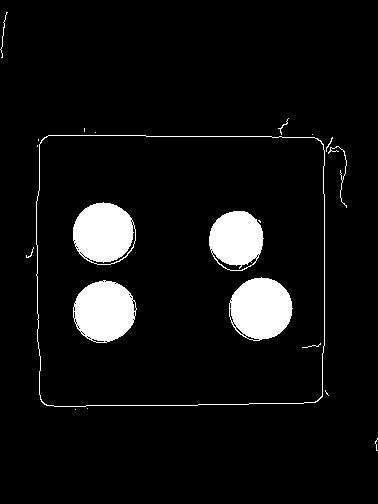
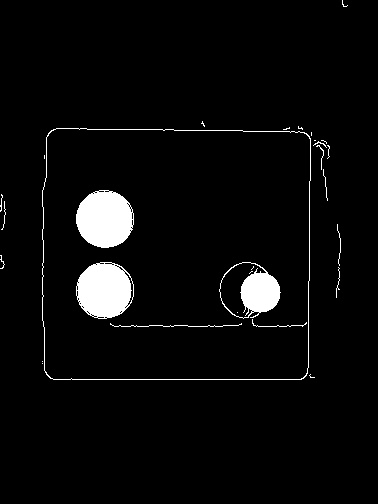


Figure 10 – Found circles visualization

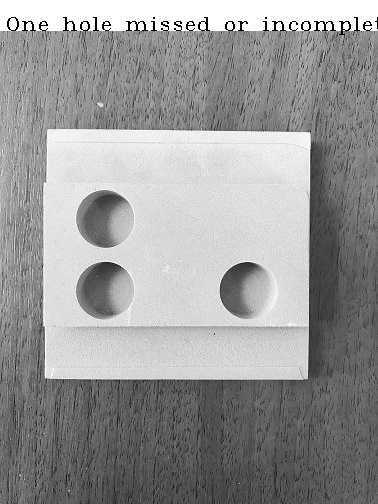


Figure 11 (a) – Output: One hole missed or incomplete

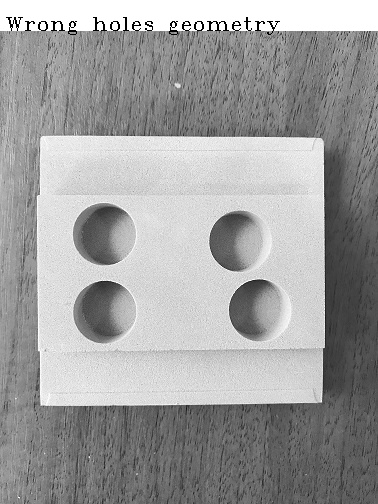


Figure 11 (b) – Output: Wrong holes geometry

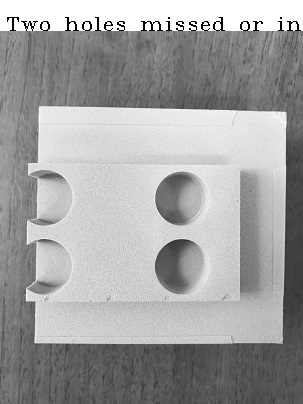


Figure 11 (c) – Output: Two holes missed or imcomplete

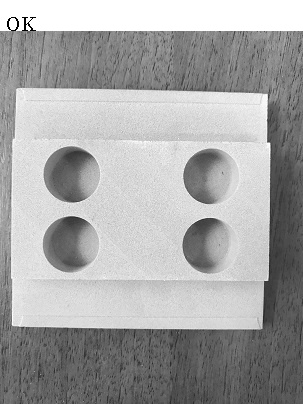


Figure 11 (d) – Output: OK

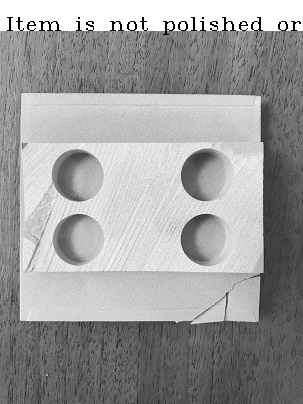


Figure 11 (e) – Output: Item is not polished or too much noise

## Conclusion

Defect detection using computer vision is a small part of a big solution for preventive analytics and nondestructive testing leading to lean manufacturing. It can remove the gap between high level management and primary production level.

Important advantage of proposed approach is that our module written on Python programming language can be easily implemented with minimal hardware requirements. Implementation of different part models support and some operators which can handle Geneva waves can significantly improve developed system

**Assessment sheet**

Defect detection with computer vision

Consultation project

01.03.2021 – 06.06.2021

|  |  |  |  |
| --- | --- | --- | --- |
| **Project supervisor**  Full name, position | |  | |
| **Student[[1]](#footnote-1)**: | | **Victor E. Kozlovskiy** | |
| Master’s Programme | | **Big Data Systems** | |
| Group № | | **MBD202** | |
| Components of the final grade[[2]](#footnote-2) | | Grade on a 10-point scale | | Notes (if necessary) | |
| **О пр** – Grade for the project result (product) | |  | |  | |
| **О сп** Grade for the methods and technologies used | |  | |  | |
| **О р** Grade for the implementation of the project’s work | |  | |  | |
| **О к** Grade for the developed competencies | |  | |  | |
| **О гр** Grade for the student's individual contribution to group work | |  | |  | |
| **О ком** Grade for team work | |  | |  | |
| **О з** Grade for the presentation, project defense | |  | |  | |
| **О вз** Grade given by other project participants (peers evaluation) | |  | |  | |
| **О с** Student self-assessment | | 10 | |  | |
| Grade calculation (formula with weight coefficient) | |  | | | |
| **Final grade for the project** | |  | | Project supervisor’s signature | |
| **Number of credits** | | 3 | |

Date\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. For group projects, an assessment sheet is filled out for each group member [↑](#footnote-ref-1)
2. Only necessary components are used, if some component is not used in the assessment, then a dash is put in the corresponding line [↑](#footnote-ref-2)