JSS Mahavidyapeetha

JSS Science And Technology University

Mysuru

570 006

"EVENT 4"

"Text Document Scanner"

Report submitted in partial fulfillment of curriculum prescribed for Image Processing(CS552) for the award of the degree of

Bachelor Of Engineering

In

Computer Science And Engineering

by

USN	Name
01JST18CS135	Sinchana R
01JST18CS181	Chaitanya Kodlekere
01JST18CS046	K S Shriniket
01JST18CS073	Nivedita M Hanamasagar

Submitted To,

Prof.Shruthi N M

Assistant Professor

Dept. of CS&E, SJCE Mysore

Contents

1 INTRODUCTION

1.1 OBJECTIVE OF THE PROJECT

The objective of this project is to develop an Document Scanner that turns your paper documents to PDF documents. This Document Scanner Application efficiently scan paper documents such as receipts, notes, whiteboard discussions, business cards, certificates, etc using your phone camera. Pictures stored in mobile phones and pictures taken by mobiles are the mainly focus of this application. The purpose of this application is to recognize text in text images, and convert it into scanned image in order to reuse it later.

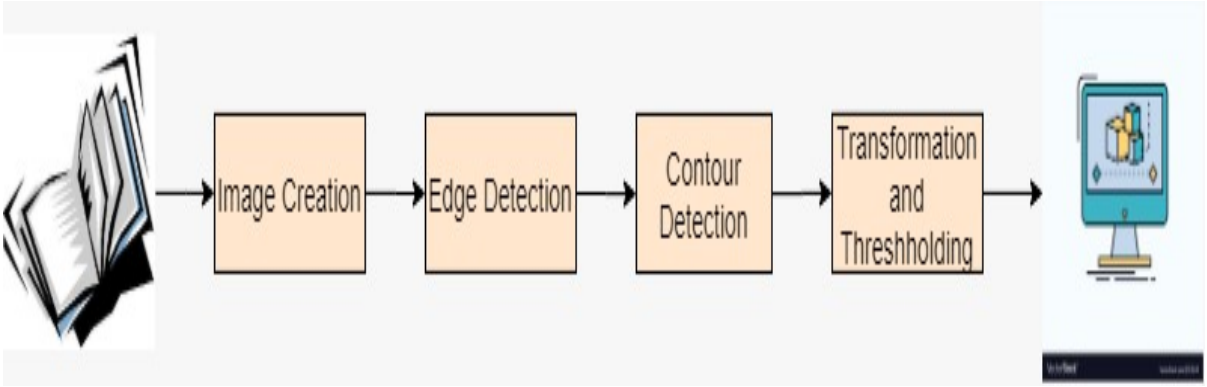
1.2 SCOPE OF THE PROJECT

This project is a prototype for document scanner and it is restricted within the college premises. This project in its current state is restricted to scan images taken from mobiles or any other source. This project can be extended to add new features. The project can also be modified to take live feed and convert it into scanned image.

2 OVERVIEW

2.1 COMPONENTS OF DOCUMENT SCANNER SYSTEM

A typical Document Scanner system consists of the following components shown :-



2.1.1 IMAGE CREATION

This section of the process in to convert the paper documents into images using the camera of the mobile or any other source. This image is used as an input for the document scanner.

2.1.2 EDGE DETECTION

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge de-

tection is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision.

In our case we find the edges of the pages to be scanned.We convert the image from RGB to grayscale then, perform Gaussian blurring to remove high frequency noise, and then perform Canny edge detection.

2.1.3 CONTOUR DETECTION

Contours detection is a process that can be explained simply as a curve joining all the continuous points (along with the boundary), having same colour or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

In our case, a document scanner scans in a piece of paper. A piece of paper is assumed to be a rectangle. And a rectangle has four edges. Therefore, we assume that the largest contour in the image with exactly four points is our piece of paper to be scanned.

2.1.4 PERSPECTIVE TRANSFORM AND THRESHOLD

Perspective Transformation Perspective Transformation is the process of changing the perspective of a given image for getting better insights about the required information.Perspective Transformation requires the points on the image from which want to gather information by changing the perspective. It also requires the points inside which we want to display our image. Then, we get the perspective transform from the two given set of points and wrap it with the original image.

Threshold

Image thresholding is a simple form of image segmentation. It is a way to create a binary image from a grayscale or full-color image. This is typically done in order to separate "object" or foreground pixels from background pixels to aid in image processing.

In our case,the last step in building document scanner is to take the four points representing the outline of the document and apply a perspective transform to obtain a top-down, "birds eye view" of the image.To obtain the black and white feel to the image, we then take the warped image, convert it to grayscale and apply adaptive thresholding.

3 ALGORITHM OF APPLICATION

3.1 STEP 1 : EDGE DETECTION

The first step to building our document scanner is to perform edge detection. We convert the image from RGB to grayscale. Then we perform Gaussian blurring to remove high frequency noise.Canny edge detection is then applied to noiseless image to find the edges of the page in the image. The following code of the program achieves this task:-

```
# load the image and compute the ratio of the old height
# to the new height, clone it, and resize it
image = cv2.imread(args["image"])
ratio = image.shape[0] / 500.0
orig = image.copy()
image = imutils.resize(image, height = 500)

# convert the image to grayscale, blur it, and find edges in the image
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (5, 5), 0)
edged = cv2.Canny(gray, 75, 200)
# show the original image and the edge detected image
print("STEP 1: Edge Detection")
cv2.imshow("Image", image)
cv2.imshow("Edged", edged)
```

3.2 STEP 2: FINDING CONTOURS

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same colour or intensity. The contours are a useful tool for shape analysis and object detection and recognition. For better accuracy, use of binary image is advised. Hence, we apply canny edge detection before finding contours. We are assuming that largest contour in the image with exactly four points is containing the text part which we are going to scan. This is also a reasonably safe assumption — the scanner program simply assumes that the document you want to scan is the main focus of our image. And it's also safe to assume that the piece of paper has four edges. The code below helps us to find contours:

```
# find the contours in the edged image, keeping only the
# largest ones, and initialize the screen contour
cnts = cv2.findContours(edged.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
cnts = sorted(cnts, key = cv2.contourArea, reverse = True)[:5]

# loop over the contours
for c in cnts:
# approximate the contour
peri = cv2.arcLength(c, True)
```

approx = cv2.approxPolyDP(c, 0.02 * peri, True)

 # if our approximated contour has four points, then we can assume that we have found our screen
if len(approx) == 4:
screenCnt = approx
break

 # show the contour (outline) of the piece of paper
print("STEP 2: Find contours of paper")
cv2.drawContours(image, [screenCnt], -1, (0, 255, 0), 2)
cv2.imshow("Outline", image)

3.3 STEP 3: FOUR POINT TRANSFORM

Before we move to final step in building a mobile document scanner, we have to take the four points representing the outline of the document and apply a perspective transform to obtain a top-down, "birds eye view" of the image. Hence, we use a customised function named "four_point_transform". The customized function is as follows:-

Four Point Transform Function

 # import the necessary packages
import numpy as np
import cv2

def order_points(pts):
 # initialzie a list of coordinates that will be ordered such that the first entry in the list is the top-left, the second entry is the top-right, the third is the bottom-right, and the fourth is the bottom-left
rect = np.zeros((4, 2), dtype = "float32")

 # the top-left point will have the smallest sum, whereas the bottom-right point will have the largest sum
s = pts.sum(axis = 1)
rect[0] = pts[np.argmin(s)]
rect[2] = pts[np.argmax(s)]

 # now, compute the difference between the points, the top-right point will have the smallest difference, whereas the bottom-left will have the largest difference
diff = np.diff(pts, axis = 1)
rect[1] = pts[np.argmin(diff)]
rect[3] = pts[np.argmax(diff)]

 # return the ordered coordinates
return rect

```
    def four_point_transform(image, pts):
# obtain a consistent order of the points and unpack them individually
rect = order_points(pts)
(tl, tr, br, bl) = rect

# compute the width of the new image, which will be the maximum distance between bottom-right and bottom-leftx-coordiates or the top-right and top-left x-coordinates
widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2)
widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
maxWidth = max(int(widthA), int(widthB))

# compute the height of the new image, which will be the maximum distance between the top-right and bottom-right y-coordinates or the top-left and bottom-left y-coordinates
heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
maxHeight = max(int(heightA), int(heightB))

# now that we have the dimensions of the new image, construct the set of destination points to obtain a "birds eye view", of the image, again specifying points in the top-left, top-right, bottom-right, and bottom-left order
dst = np.array([
    [0, 0],

    [maxWidth - 1, 0],

    [maxWidth - 1, maxHeight - 1],

[0, maxHeight - 1]], dtype = "float32")

# compute the perspective transform matrix and then apply it
M = cv2.getPerspectiveTransform(rect, dst)
warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))

# return the warped image
return warped
```

3.4 STEP 4: APPLY PERSPECTIVE TRANSFORM AND THRESHOLD

The last step in building a document scanner is to take the four points representing the outline of the document and apply a perspective transform to obtain a top-down, "birds eye view" of the image. Here, we use the "four_point_transform" function to obtain the bird-eye view which has been discussed in the above section.

 # apply the four point transform to obtain a top-down view of the original image
warped = four_point_transform(orig, screenCnt.reshape(4, 2) * ratio)
 # convert the warped image to grayscale, then threshold it to give it that 'black and white' paper effect
warped = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)
T = threshold_local(warped, 11, offset = 10, method = "gaussian")
warped = (warped ¿ T).astype("uint8") * 255

 # show the original and scanned images
print("STEP 3: Apply perspective transform")
cv2.imshow("Original", imutils.resize(orig, height = 650))
cv2.imshow("Scanned", imutils.resize(warped, height = 650))
cv2.waitKey(0)
cv2.destroyAllwindows()

4 PROGRAM OF THE APPLICATION

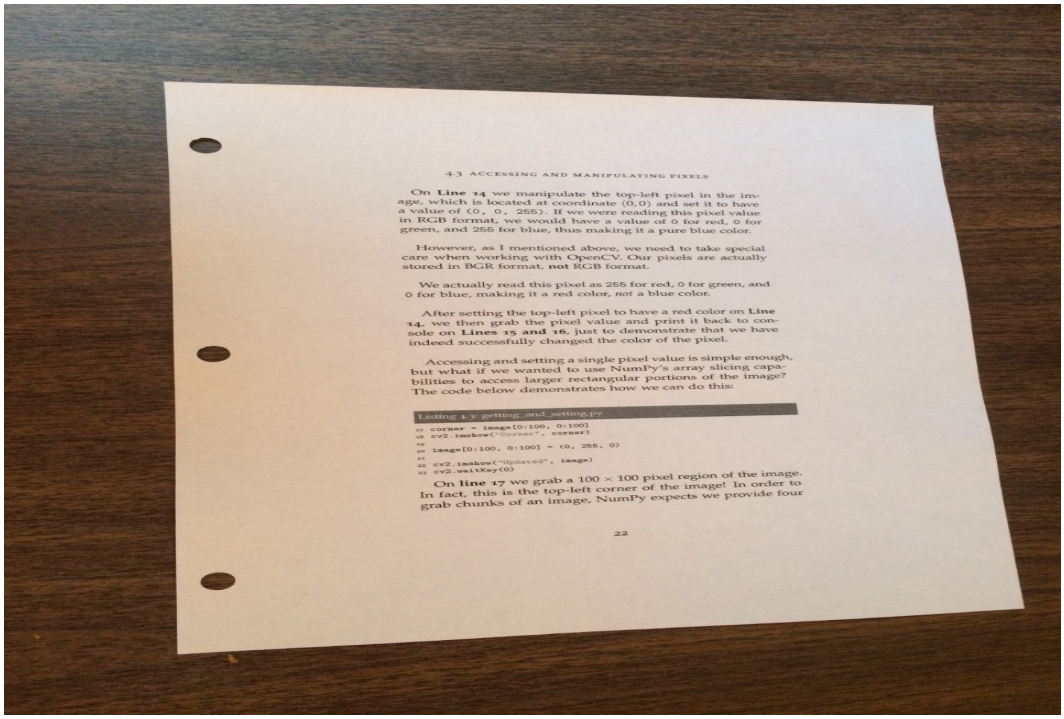
Program

```
# import the necessary packages
from pyimagesearch.transform import four_point_transform
from skimage.filters import threshold_local
import numpy as np
import argparse
import cv2
import imutils

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "–image", required = True,
help = "Path to the image to be scanned")
args = vars(ap.parse_args())

# load the image and compute the ratio of the old height
# to the new height, clone it, and resize it
image = cv2.imread(args["image"])
ratio = image.shape[0] / 500.0
orig = image.copy()
image = imutils.resize(image, height = 500)

# convert the image to grayscale, blur it, and find edges in the image
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (5, 5), 0)
edged = cv2.Canny(gray, 75, 200)
# show the original image and the edge detected image
print("STEP 1: Edge Detection")
cv2.imshow("Image", image)
cv2.imshow("Edged", edged)

# find the contours in the edged image, keeping only the
# largest ones, and initialize the screen contour
cnts = cv2.findContours(edged.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
cnts = sorted(cnts, key = cv2.contourArea, reverse = True)[:5]

# loop over the contours
for c in cnts:
# approximate the contour
peri = cv2.arcLength(c, True)
approx = cv2.approxPolyDP(c, 0.02 * peri, True)

# if our approximated contour has four points, then we can assume that we have
found our screen
if len(approx) == 4:
```

```
screenCnt = approx
break

# show the contour (outline) of the piece of paper
print("STEP 2: Find contours of paper")
cv2.drawContours(image, [screenCnt], -1, (0, 255, 0), 2)
cv2.imshow("Outline", image)


# apply the four point transform to obtain a top-down view of the original image
warped = four_point_transform(orig, screenCnt.reshape(4, 2) * ratio)
# convert the warped image to grayscale, then threshold it to give it that 'black and white' paper effect
warped = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)
T = threshold_local(warped, 11, offset = 10, method = "gaussian")
warped = (warped ¿ T).astype("uint8") * 255

# show the original and scanned images
print("STEP 3: Apply perspective transform")
cv2.imshow("Original", imutils.resize(orig, height = 650))
cv2.imshow("Scanned", imutils.resize(warped, height = 650))
cv2.waitKey(0)
cv2.destroyAllwindows()
```

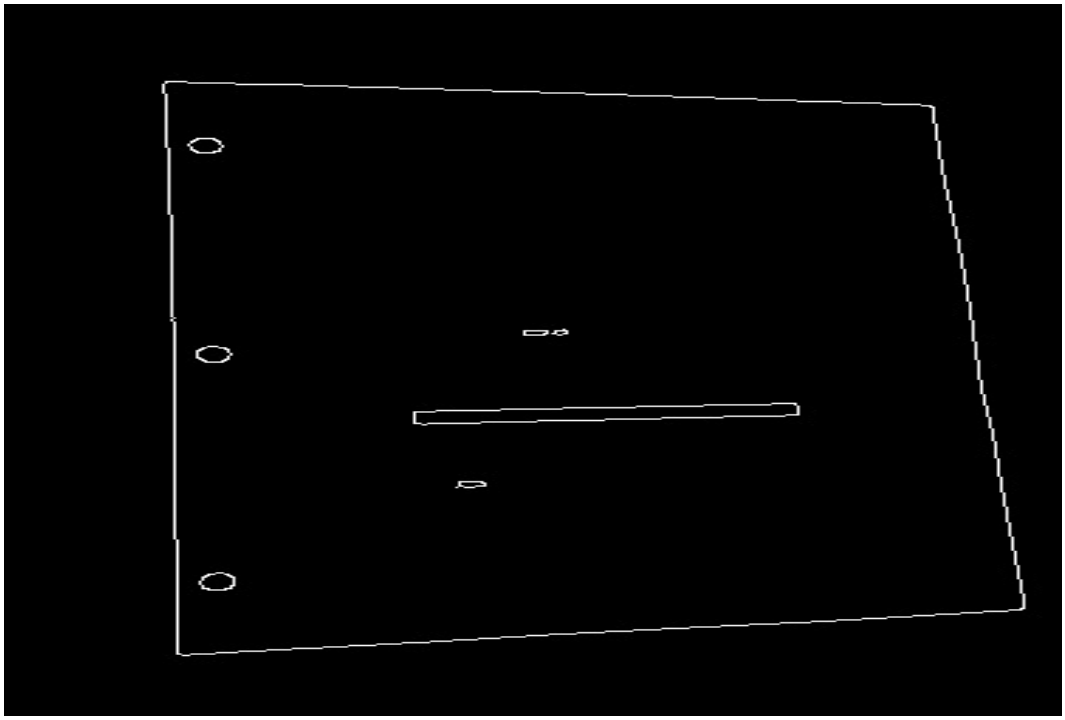
5 TESTING AND RESULTS

1. **Input-Image 1**

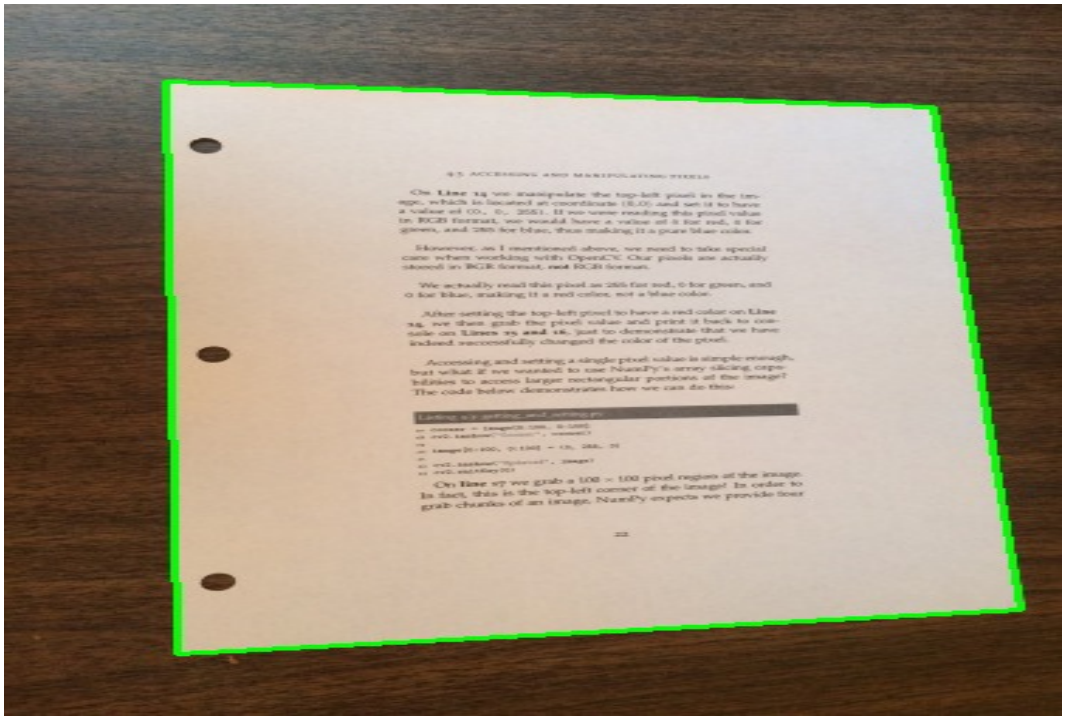


2. **Output**

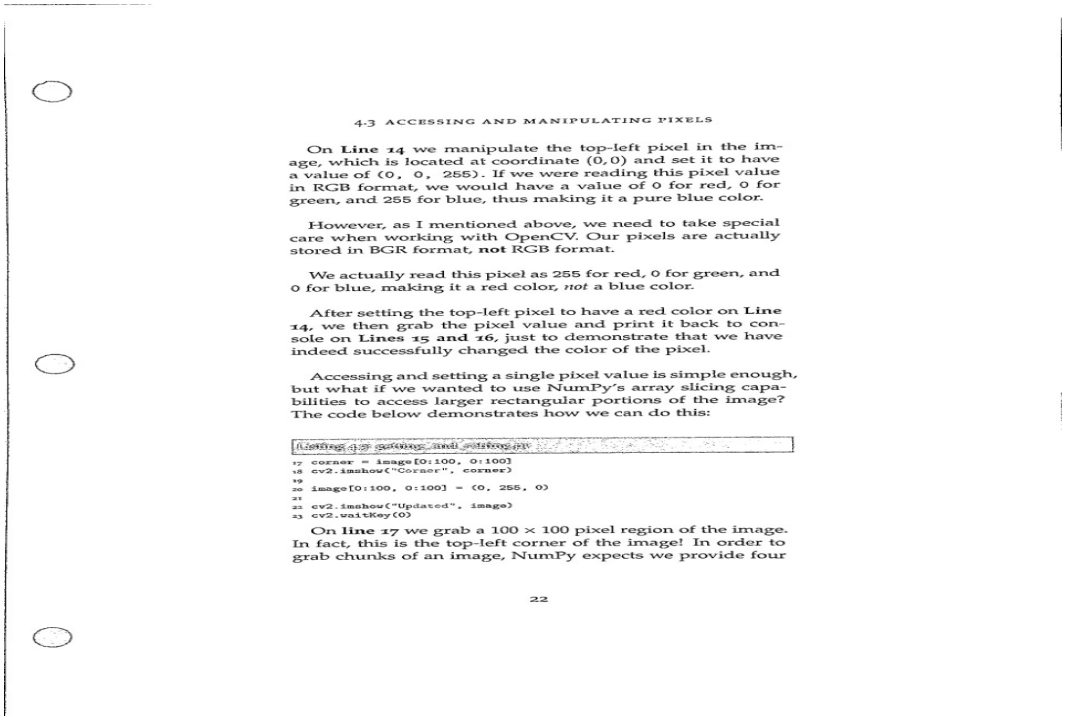
- Edge Detection



- Contour Detection



- Scanned Image



4-3 ACCESSING AND MANIPULATING PIXELS

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0, 0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, *not* a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

```
17 corner = image[0:100, 0:100]
18 cv2.imshow("Corner", corner)
19
20 image[0:100, 0:100] = (0, 255, 0)
21
22 cv2.imshow("Updated", image)
23 cv2.waitKey(0)
```

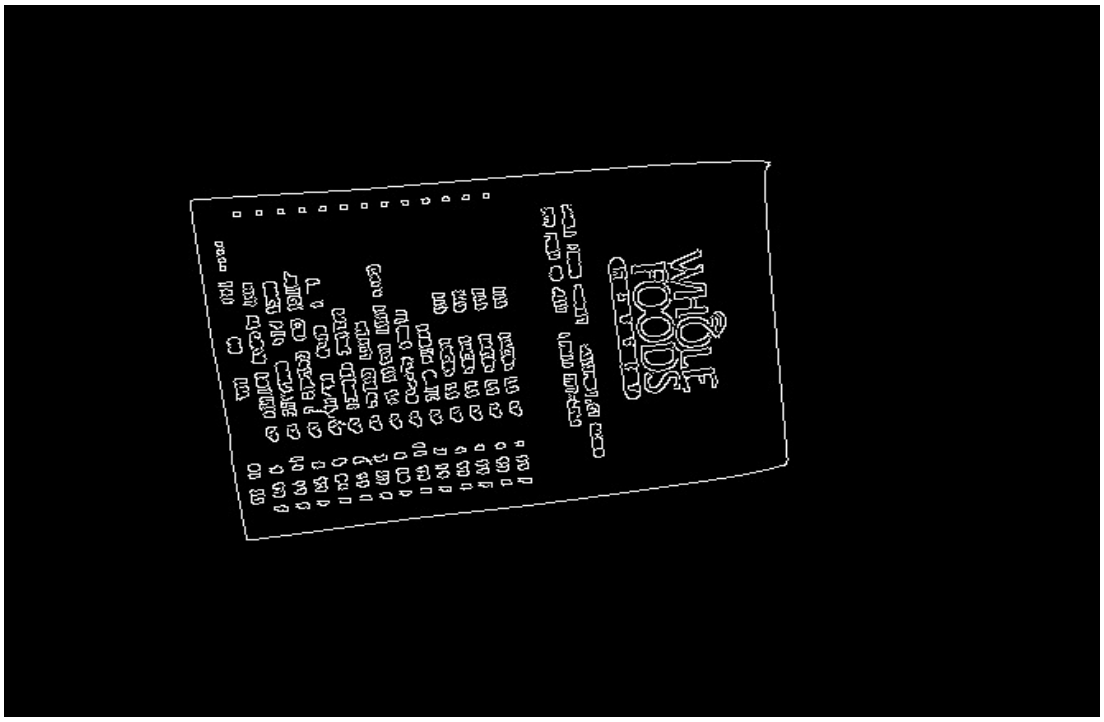
On **line 17** we grab a 100 × 100 pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

22

3. **Input-Image-2**

4. **Output 2**

- Edge Detection



- Contour Detection

- Scanned Image