# Detecting packages and orientation with OpenCV

Arturo Lopez, arlopf95@gmail.com 12/07/2022

## Detecting packages with Python and OpenCV

For this project, as seen in Figure 1, the goal is to detect the packages that are going to be sorted and their respective orientation, this could be helpful to determine if the orientation affect if the package failed to divert, with image processing more data that could be obtain for optimize the process, like size, type of package, gap between packages and more.

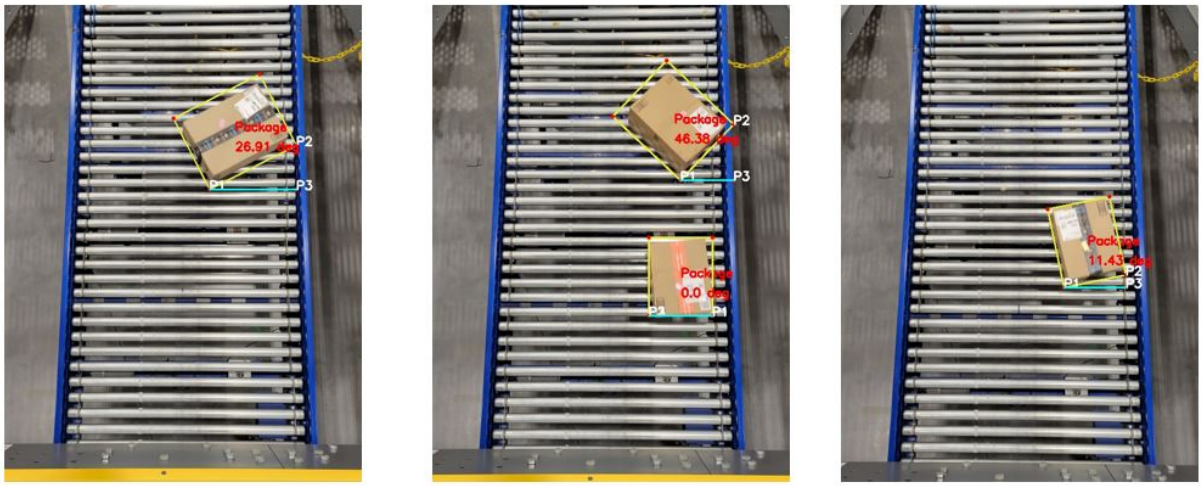


Figure 1: Example of packages and orientation recognition.

### Input the image

In Figure 2, a package can be observed moving along the conveyor.

Figure 2: Image of a package moving along the conveyor.

First, for localizing the package we need to apply an adaptive threshold, in OpenCV, the adaptive threshold function is used to segment images based on local thresholding. The adaptive thresholding algorithm calculates the threshold value for a pixel based on the average intensity of its neighboring pixels. This is useful for images where the lighting conditions vary across the image, for this example, we are using a Gaussian adaptive threshold, the Gaussian threshold function is used to segment images based on global thresholding using a Gaussian-weighted sum of neighborhood pixels. This is useful for images where the intensity values are normally distributed, see Figure 4, for this we need to convert the input image into a gray scale image, see Figure 3. See Listing 1.

```
# Read input image
img = cv2.imread('20221112_014332.jpg')
# Create a gray scale image from input image
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
# adaptive threshold
thresh = cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.
    THRESH_BINARY_INV,51,45)
```

Listing 1: Applying adaptive Gaussian threshold to input image.

Figure 3: Gray scale image of input image.

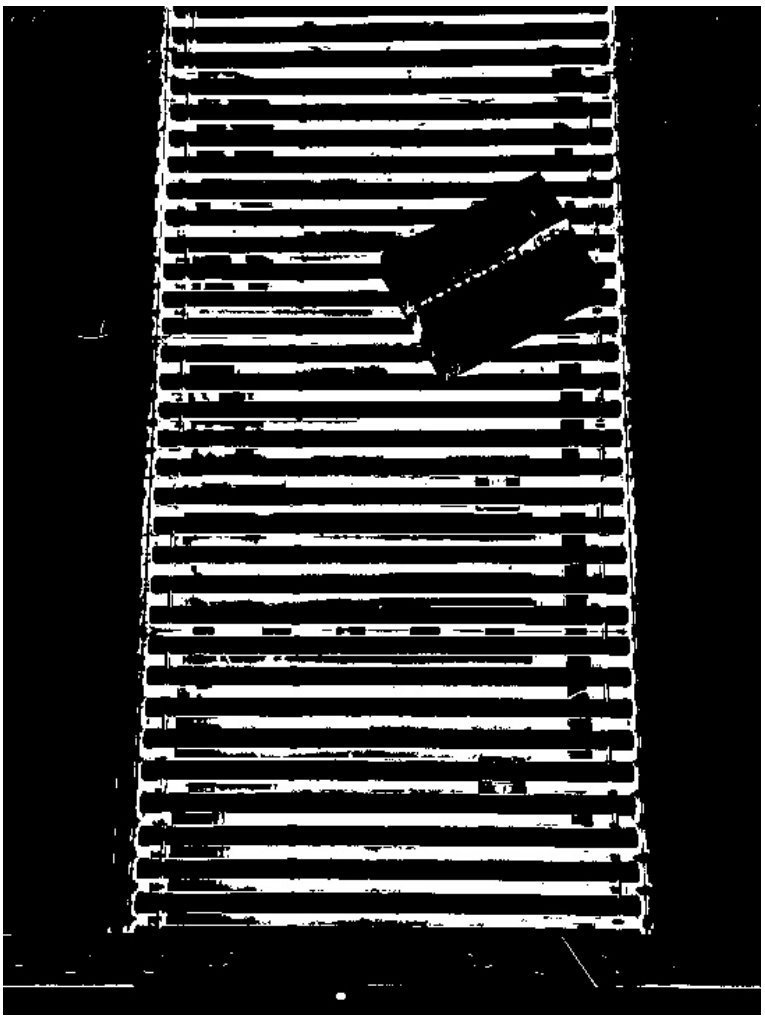

Figure 4: Adaptive Gaussian threshold applied to gray scale image.

### Detecting Contours

For detecting contours, first we applied "thresh = cv2.bitwise_not(thresh)", which takes the input threshold image and applies the bitwise_not operation, which inverts the image. This is because the input image is a black background with white packages, and the further processing is easier when the background is white. See Figure 5.

Then, we need to create a rectangular kernel and use it to apply to morphological opening operation to the threshold image. This helps to remove small white regions (noise) in the image, making it easier to identify the packages. So we applied "kernel = cv2.getStructuringElement (cv2.MORPH_RECT, (11,11))" and "opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=2)".

The cv2.findContours function is used to identify contours (boundaries) in the resulting image. cnts variable stores the contours found in the image. The contours are filtered using a minimum and maximum area threshold to avoid detecting small noise as packages.

For each valid contour, the cv2.boundingRect function is used to get the (x,y) coordinates of the upper-left corner of a rectangle that can enclose the contour, as well as its width and height. These parameters are used to draw a rectangle around the detected package using the

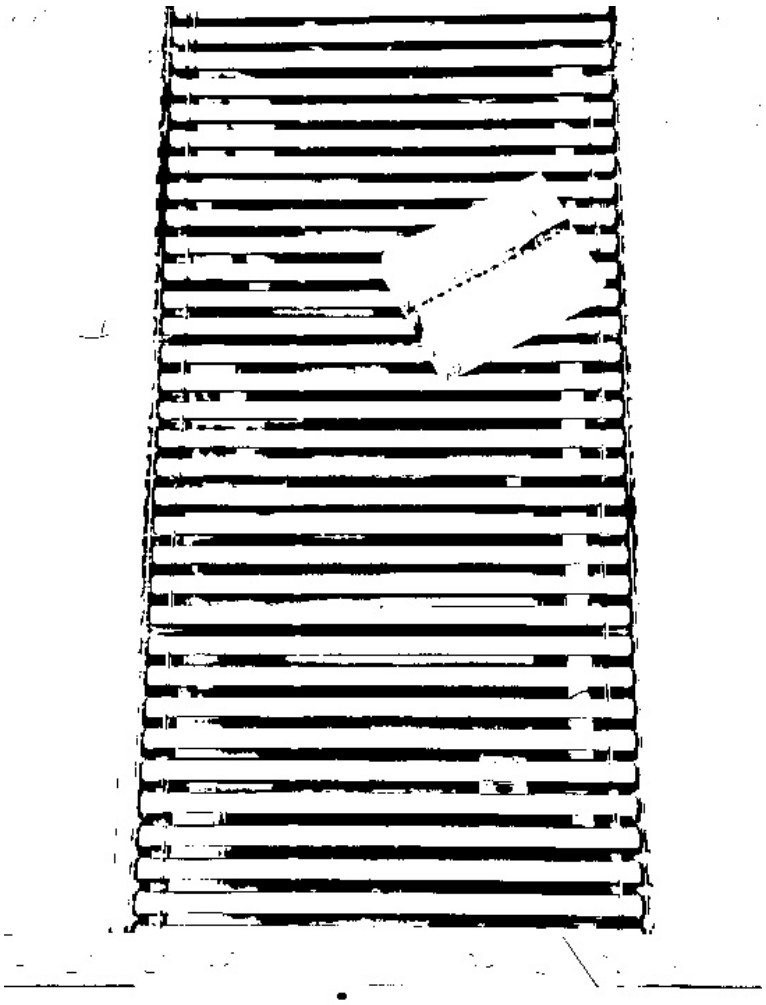

Figure 5: bitwise_not operation applied to thresh image.

cv2.rectangle function. A label "Package" is also added to the center of the rectangle using the cv2.putText function.

Finally, the code draws a rotated rectangle around the detected package using cv2.minAreaRect(c) function which finds the minimum area rectangle that encloses the contour. cv2.boxPoints(rect) function returns the four corner points of the rotated rectangle which are then drawn using cv2.drawContours. See Figure 6, see Listing 2.

```
# Morph open
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (11,11))
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=2)

# Draw rectangles, the 'area_treshold' value was determined empirically
cnts = cv2.findContours(opening, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]
area_treshold_min = 2000 # Package Area bigger than 2000, to prevent small noise-
    looking packages
area_treshold = 60000
for c in cnts:
    if cv2.contourArea(c) > area_treshold_min and cv2.contourArea(c) < area_treshold
```

```
    :
      x,y,w,h = cv2.boundingRect(c)
      #cv2.rectangle(img, (x, y), (x + w, y + h), (36,255,12), 3)
      cv2.putText(img, 'Package', (int((x+x+w)/2), int((y+y+h)/2)), cv2.
FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

      ## BEGIN - draw rotated rectangle
      rect = cv2.minAreaRect(c)
      box = cv2.boxPoints(rect)
      box = np.int0(box)
      cv2.drawContours(img,[box],0,(52,235,235),2)
      ## END - draw rotated rectangle
```

Listing 2: Detecting Contours of Package.


Figure 6: Contours and package detected.

### Detecting vertices of the package

Once we have a bounding box ("box") around the object, we can applied the "cv2.circle()" function four times to draw a circle at each of the four corners, see Listing 3, see Image 7.

```
    cv2.circle(img, (box[0][0],box[0][1]), radius=3, color=(0, 0, 255), thickness=-1)
    cv2.circle(img, (box[1][0],box[1][1]), radius=3, color=(0, 0, 255), thickness=-1)
    cv2.circle(img, (box[2][0],box[2][1]), radius=3, color=(0, 0, 255), thickness=-1)
    cv2.circle(img, (box[3][0],box[3][1]), radius=3, color=(0, 0, 255), thickness=-1)
```

Listing 3: Applying adaptive threshold to input image.


Figure 7: Circles draw around the four corners of the bounding box around the package.

### Orientation of the package

Now that we know the vertices of the package, we can get the orientation, see Listing 4:

1. Get the lowest vertex of the package, which we are calling P1.
2. Get the second lowest vertex of the package, which we are calling P2.
3. To get orientation, we can get a third point which we are calling P3, which is going to be located at the same X coordinate of P2, and Y coordinate of P1. We need to check the relative position of P1 to P2 (whether it is on the right or the left).

If $X_P1 > X_P2$ then, P1 is on the left of P2, see Figure 8.

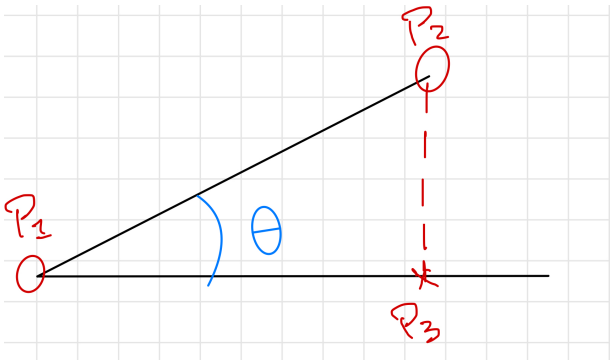


Figure 8: P1 at the left of P2.

Solving for $\theta$,

$$\theta = \tan^{-1} \frac{Y_{P2} - Y_{P3}}{X_{P3} - X_{P1}} \tag{1}$$

Since we are working with pixels, we need to multiply the numerator (Y coordinates) by -1.

$$\theta = \tan^{-1} \frac{(-1)(Y_{P2} - Y_{P3})}{X_{P3} - X_{P1}} \tag{2}$$

Else, if $X_P1 < X_P2$ then, P1 is on the right of P2, see Figure 9.

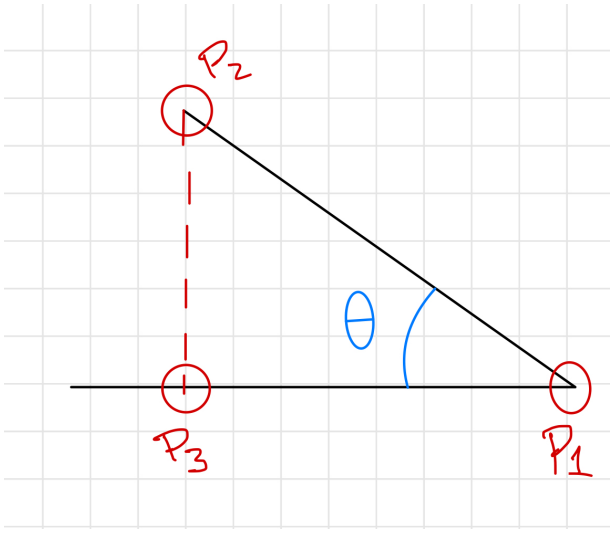


Figure 9: P1 at the right of P2.

Solving for $\theta$,

$$\theta = \tan^{-1} \frac{Y_{P2} - Y_{P3}}{X_{P1} - X_{P3}} \tag{3}$$

Since we are working with pixels, we need to multiply the numerator (Y coordinates) by -1.

$$\theta = \tan^{-1} \frac{(-1)(Y_{P2} - Y_{P3})}{X_{P1} - X_{P3}} \tag{4}$$

```
1
      # Get orientation angle
3     # First, lets get the lower point, which is the one with highest y coordinate
      lowestX, lowestY = 0, 0
5     secondLowestX, secondLowestY = 0, 0
      for b in box:
7       if b[1:] > lowestY:
          secondLowestX, secondLowestY = lowestX, lowestY
9         lowestX, lowestY = b
        else:
11        if b[1:] > secondLowestY:
            secondLowestX, secondLowestY = b
13
      #print(lowestX, lowestY)
15    #print(secondLowestX, secondLowestY)
      #print(box)
17    cv2.putText(img, 'P1', (lowestX, lowestY), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
    (255, 255, 255), 2)
      cv2.putText(img, 'P2', (secondLowestX, secondLowestY), cv2.
    FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)
19    cv2.circle(img, (secondLowestX, lowestY), radius=3, color=(0, 0, 255),
    thickness=-1)
      cv2.putText(img, 'P3', (secondLowestX, lowestY), cv2.FONT_HERSHEY_SIMPLEX,
    0.6, (255, 255, 255), 2)
21    # Draw reference line for orientation
      cv2.line(img, (lowestX, lowestY), (secondLowestX, lowestY), (252, 252, 3), 2)
23
      # Check if P3 is on the right or left of P1
25    if (secondLowestX >= lowestX): # P2 on the right of P1
        # theta = tg^-1 (((-1)*(YP2 - YP1))/(XP2 - XP1))
27      # theta = tg^-1 ((-1)*(secondLowestY - lowestY)/(secondLowestX - lowestX))
        theta = math.degrees(math.atan((-1)*(secondLowestY - lowestY)/(secondLowestX
     - lowestX)))
29    else: # P2 on the left of P1
        # theta = tg^-1 ((-1)*(YP2 - YP1)/(XP1 - XP2))
31      # theta = tg^-1 ((-1)*(secondLowestY - lowestY)/(lowestX - secondLowestX))
        theta = math.degrees(math.atan((-1)*(secondLowestY - lowestY)/(lowestX -
    secondLowestX)))
33    # Display orientation on degrees
      cv2.putText(img, str(round(theta, 2))+ " deg", (int((x+x+w)/2), int((y+y+h)/2)
    +30), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
```

Listing 4: Getting orientation of the package on degrees.


Figure 10: Gray scale image of input image.

### Applying to video

This can be also applied to video, for this we input the video and analyze frame by frame.

```
cap = cv2.VideoCapture('20221110_182744.mp4')


# Check if camera opened successfully
if (cap.isOpened()== False):
  print("Error opening video stream or file")

# Read until video is completed
while(cap.isOpened()):
  # Capture frame-by-frame
  ret, frame = cap.read()
  if ret == True:

    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    thresh = cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.
    THRESH_BINARY_INV,51,45)
```

```
16      .
        .
18      .
```

Listing 5: Applying adaptive threshold to input image.