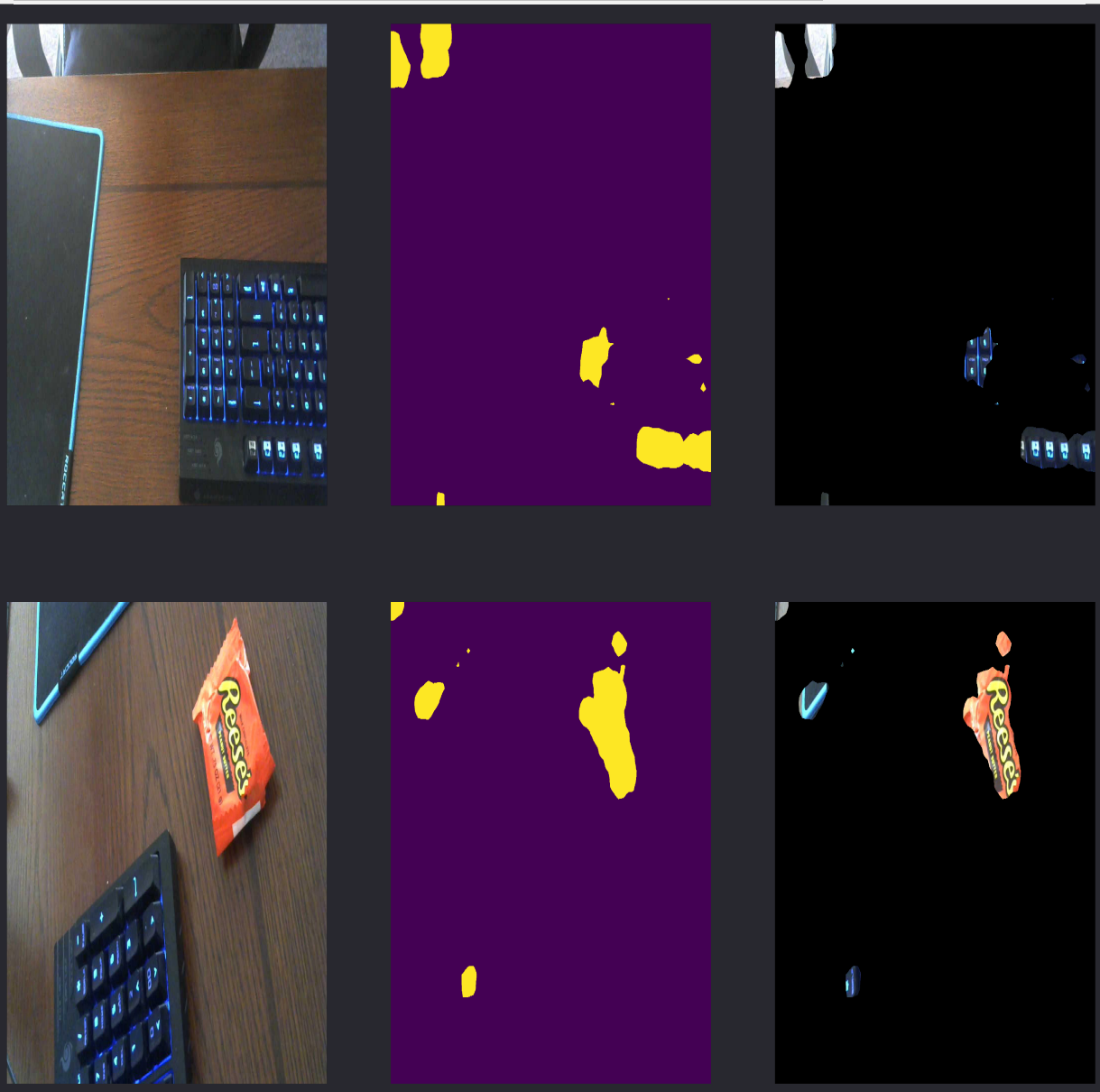
[21]
```
'''
Detecting and Displaying Image Salience with a Binary Threshold
'''

%matplotlib inline
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Load the images
img1 = cv2.imread('images/scene.jpg', -1)
img2 = cv2.imread('images/scene_with_candy_2.jpg', -1)

### Detect Salience
saliencyDetector = cv2.saliency.StaticSaliencySpectralResidual_create()

success, img1SalienceMap = saliencyDetector.computeSaliency(img1, None)
success, img2SalienceMap = saliencyDetector.computeSaliency(img2, None)

threshold = 0.3

# Apply a binary threshold
img1SalienceMap[img1SalienceMap < threshold] = 0
img1SalienceMap[img1SalienceMap > threshold] = 1
img2SalienceMap[img2SalienceMap < threshold] = 0
img2SalienceMap[img2SalienceMap > threshold] = 1

# Convert from float32 to uint8
img1SalienceMapMask = img1SalienceMap.astype(np.uint8)
img2SalienceMapMask = img2SalienceMap.astype(np.uint8)

# Apply the mask and display the result
img1Masked = cv2.bitwise_and(img1, img1, mask = img1SalienceMapMask)
img2Masked = cv2.bitwise_and(img2, img2, mask = img2SalienceMapMask)
###

# Create subplots to show images
f, axarr = plt.subplots(2,3,figsize=(100,100))

# Convert colors from
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img1Masked = cv2.cvtColor(img1Masked, cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
img2Masked = cv2.cvtColor(img2Masked, cv2.COLOR_BGR2RGB)

# Hide axes
axarr[0,0].axis('off')
axarr[0,1].axis('off')
axarr[0,2].axis('off')
axarr[1,0].axis('off')
axarr[1,1].axis('off')
axarr[1,2].axis('off')

# Display images in plot
axarr[0,0].imshow(img1, interpolation='nearest', extent=[0,400,0,1], as
axarr[0,1].imshow(img1SalienceMap, interpolation='nearest', extent=[0,4
axarr[0,2].imshow(img1Masked, interpolation='nearest', extent=[0,400,0,
axarr[1,0].imshow(img2, interpolation='nearest', extent=[0,400,0,1], as
```

```
axarr[1,1].imshow(img2SalienceMap, interpolation='nearest', extent=[0,4
axarr[1,2].imshow(img2Masked, interpolation='nearest', extent=[0,400,0,

plt.show()
```

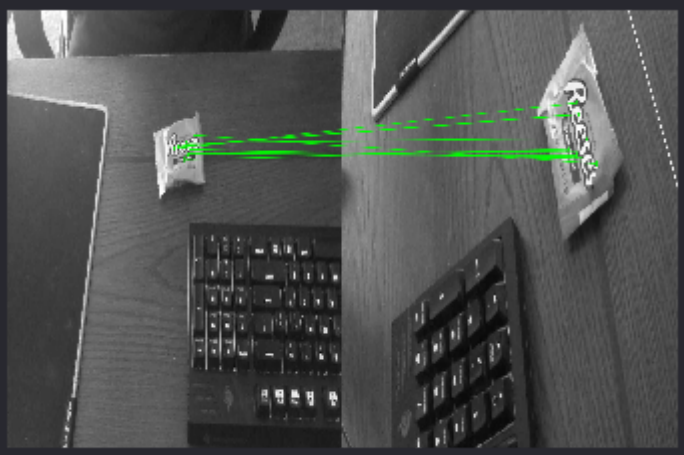


[2]

```
'''
Feature Matching using ORB Feature Detector and Binary Descriptor

ORB feature detection using an oriented FAST detection method and rota
'''

%matplotlib inline
import numpy as np
import cv2
import matplotlib.pyplot as plt
from PIL import Image

# Load the images in gray scale
img1 = cv2.imread('images/scene_with_candy.jpg', 0)
img2 = cv2.imread('images/scene_with_candy_2.jpg', 0)

### Detect Salience
```

```
saliencyDetector = cv2.saliency.StaticSaliencySpectralResidual_create()

success, img1SalienceMap = saliencyDetector.computeSaliency(img1, None)
success, img2SalienceMap = saliencyDetector.computeSaliency(img2, None)

threshold = 0.3

# Apply a binary threshold
img1SalienceMap[img1SalienceMap < threshold] = 0
img1SalienceMap[img1SalienceMap > threshold] = 1
img2SalienceMap[img2SalienceMap < threshold] = 0
img2SalienceMap[img2SalienceMap > threshold] = 1

# Convert from float32 to uint8
img1SalienceMapMask = img1SalienceMap.astype(np.uint8)
img2SalienceMapMask = img2SalienceMap.astype(np.uint8)

# Apply the mask and display the result
img1Masked = cv2.bitwise_and(img1, img1, mask = img1SalienceMapMask)
img2Masked = cv2.bitwise_and(img2, img2, mask = img2SalienceMapMask)
###

# Detect the SIFT key points and compute the descriptors for the two
sift = cv2.xfeatures2d.SIFT_create()
keyPoints1, descriptors1 = sift.detectAndCompute(img1Masked, None)
keyPoints2, descriptors2 = sift.detectAndCompute(img2Masked, None)

# Create brute-force matcher object
bf = cv2.BFMatcher()

# Match the descriptors
matches = bf.knnMatch(descriptors1, descriptors2, k=2)

# Select the good matches using the ratio test
goodMatches = []

for m, n in matches:
    if m.distance < 0.7 * n.distance:
        goodMatches.append(m)

# Apply the homography transformation if we have enough good matches
MIN_MATCH_COUNT = 10

if len(goodMatches) > MIN_MATCH_COUNT:
    # Get the good key points positions
    sourcePoints = np.float32([ keyPoints1[m.queryIdx].pt for m in goo
    destinationPoints = np.float32([ keyPoints2[m.trainIdx].pt for m i

    # Obtain the homography matrix
    M, mask = cv2.findHomography(sourcePoints, destinationPoints, meth
    matchesMask = mask.ravel().tolist()

    # Apply the perspective transformation to the source image corners
    h, w = img1.shape
    corners = np.float32([ [0, 0], [0, h - 1], [w - 1, h - 1], [w - 1
    transformedCorners = cv2.perspectiveTransform(corners, M)

    # Draw a polygon on the second image joining the transformed corne
    img2 = cv2.polylines(img2, [np.int32(transformedCorners)], True, (
else:
    print("Not enough matches are found - %d/%d" % (len(goodMatches),
```

```
    matchesMask = None

# Draw the matches
drawParameters = dict(matchColor=(0, 255, 0), singlePointColor=None, ma
result = cv2.drawMatches(img1, keyPoints1, img2, keyPoints2, goodMatche

# Display the results
plt.axis("off")
plt.imshow(result, extent=[0,400,0,1], aspect='auto')
plt.show()
```



[3]
```
'''
Change Detection based on
Feature Matching using ORB Feature Detector and Binary Descriptor
ORB feature detection using an oriented FAST detection method and rota
'''

%matplotlib inline
import numpy as np
import cv2
import matplotlib.pyplot as plt
from PIL import Image

# Load the images in gray scale
img1 = cv2.imread('images/scene.jpg', 0)
img2 = cv2.imread('images/scene_with_candy_2.jpg', 0)

### Detect Salience
saliencyDetector = cv2.saliency.StaticSaliencySpectralResidual_create()

success, img1SalienceMap = saliencyDetector.computeSaliency(img1, None)
success, img2SalienceMap = saliencyDetector.computeSaliency(img2, None)

threshold = 0.3

# Apply a binary threshold
img1SalienceMap[img1SalienceMap < threshold] = 0
img1SalienceMap[img1SalienceMap > threshold] = 1
img2SalienceMap[img2SalienceMap < threshold] = 0
img2SalienceMap[img2SalienceMap > threshold] = 1

# Convert from float32 to uint8
```

```
    img1SalienceMapMask = img1SalienceMap.astype(np.uint8)
    img2SalienceMapMask = img2SalienceMap.astype(np.uint8)

    # Apply the mask and display the result
    img1Masked = cv2.bitwise_and(img1, img1, mask = img1SalienceMapMask)
    img2Masked = cv2.bitwise_and(img2, img2, mask = img2SalienceMapMask)
    ###

    # Detect the SIFT key points and compute the descriptors for the two
    sift = cv2.xfeatures2d.SIFT_create()
    keyPoints1, descriptors1 = sift.detectAndCompute(img1Masked, None)
    keyPoints2, descriptors2 = sift.detectAndCompute(img2Masked, None)

    # Create brute-force matcher object
    bf = cv2.BFMatcher()

    # Match the descriptors
    matches = bf.knnMatch(descriptors1, descriptors2, k=2)

    # Select the bad matches using the ratio test
    badMatches = []

    for m, n in matches:
        if m.distance > 0.7 * n.distance:
            badMatches.append(m)

    # Get the bad key points positions
    changePoints = np.float32([ keyPoints1[m.queryIdx].pt for m in badMatc

    # This contains the coordinates of points in the region of a salient
    # found in the second image and not found (or very poorly matched) in
    print(changePoints)

    # TODO: Find a way to represent this visually
```