BILKENT UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

CS484
HOMEWORK 1

BERAT BİÇER
21503050

OCT. 21, 2018
ANKARA, TURKEY

Question 1

Dilation script in Python 3:

```
def dilation(image, kernel):
	kernel_x, kernel_y = kernel.shape
	kernel_center_x, kernel_center_y = int(kernel_x / 2), int(kernel_y / 2) # index, not
width
	image_x, image_y = image.shape
	output = np.zeros((image_x, image_y))

	kernel_points = [(x - kernel_center_x, y - kernel_center_y) for x in range(kernel_x) for
y in range(kernel_y) if kernel[x][y] == 255]

	for i in range(image_x):
		for j in range(image_y):
			if image[i][j] > 0:
				output[i][j] = 255
				for k in range(len(kernel_points)):
					kpkx, kpky = kernel_points[k][0], kernel_points[k][1]
					if (0 <= i + kpkx and i + kpkx < image_x) and (0 <= j + kpky
and j + kpky < image_y):
						output[i + kpkx][j + kpky] = 255
	return output
```

Erosion script in Python 3:

```
def erosion(image, kernel):
	kernel_x, kernel_y = kernel.shape
	kernel_center_x, kernel_center_y = int(kernel_x / 2), int(kernel_y / 2) # index, not
width
	image_x, image_y = image.shape
	output = np.zeros((image_x, image_y))

	kernel_points = [(x - kernel_center_x, y - kernel_center_y) for x in range(kernel_x)
for y in range(kernel_y) if kernel[x][y] == 255]

	for i in range(image_x):
		for j in range(image_y):
				flag = True
				for k in range(len(kernel_points)):
					kpkx, kpky = kernel_points[k][0], kernel_points[k][1]
					if (0 <= i + kpkx and i + kpkx < image_x) and (0 <= j +
kpky and j + kpky < image_y):
						if image[i + kpkx][j + kpky] != 255:
							flag = False
							break
				if flag == True:
					output[i][j] = 255
	return output
```

Question 2

Python script for this question:

```
# Question 2
sonnet_orig = cv2.imread("../../Data/sonnet.png", 0)
sonnet = cv2.adaptiveThreshold(sonnet_orig, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY, 11, 4)

tmp = erosion(sonnet, np.full((1, 25), 255))
tmp = erosion(tmp, np.full((1, 11), 255))
tmp = dilation(tmp, np.full((5, 1), 255))
tmp = erosion(tmp, np.full((13, 1), 255))

i, j = tmp.shape
for x in range(i):
       for y in range(j):
              tmp[x][y] = 0 if sonnet[x][y] == 0 and tmp[x][y] == 0 else 255

cv2.imshow("image", tmp)
cv2.waitKey(0)
```

The script first applies adaptive threshold on input image, since we need to take into account spatial variations in lighting pattern. Using *adaptiveThreshold* function from OpenCV with block size 11 and a constant $C = 4$, we apply adaptive threshold on input image. To eliminate noise caused by thresholding, we apply the following algorithm:
- we first widen horizontal boundaries of each line with successive erosion operations on the background with horizontal-line kernels.
- Then, we apply dilation on the background to widen vertical boundaries of the lines with vertical-line kernels.
- Then, we apply erosion on vertical boundaries once more to fit vertical boundaries with the text.
- Lastly, we apply logical-or operation on the thresholded image and the boundary we obtained.

This way, we eliminate noise that lies outside this boundary. However, to eliminate all salt-and-pepper noise caused by thresholding we need to apply a median filter, which we are not allowed to.

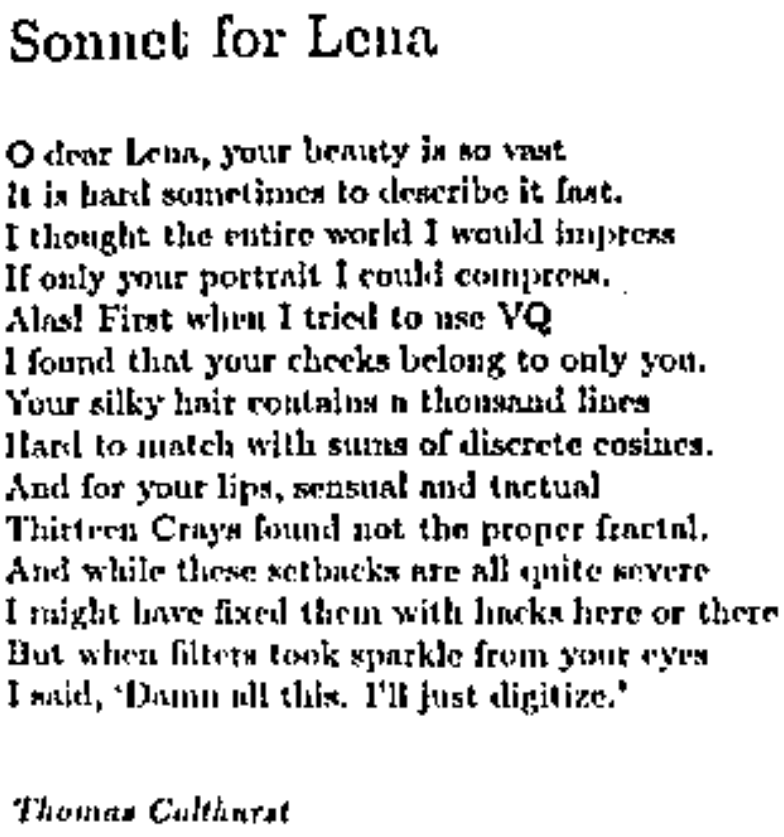
For discussion, one major obstacle in the solution was input dimensions. Since given image is relatively small, pixels between characters were even smaller. Hence, we were unable to apply dilation or erosion directly, considering readability has significance.

Selecting parameters of the threshold function was an issue. If constant C is too low(for example 1), resulting image has lots of salt-and-pepper noise. If C is too high(as $C = 15$), we loose pixel values in the output. Empirically, we decided that $C = 4$ is satisfactory: It removes enough noise in the output while preserving much of character integrity.

Using different kernels returned different results. Our initial purpose was to apply erosion on background directly, in order to enhance readability. Using kernels in shapes cross, circle, and rectangular proved unsuccessful since space between consecutive characters, in some cases, were too small to even increase the dimension of the kernel by 1. Thus, we decided that input image is too small to directly apply erosion. One solution might be to scale up the image, apply erosion on background with a rectangular kernel, then scale it down to the original size. However since we are forbidden to use scaling, this option is infeasible as well.

Lastly, OpenCV proved useful in adaptive thresholding. Resulting image was, in fact, readable enough with acceptable amount of noise. Therefore, it is safe to say that steps after thresholding are unnecessary and can be removed completely.

Output is as follows:



Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Colthurst

Question 3

Python script for this question:

```
# Question 3
graveyard_orig = cv2.imread("../../Data/airplane_graveyard.jpg")
b,g,r = cv2.split(graveyard_orig)
ret, graveyard = cv2.threshold(b, 225, 255, cv2.THRESH_BINARY)
tmp = graveyard
tmp = erosion(tmp, np.full((1,2), 255))
tmp = dilation(tmp, np.full((1, 2), 255))
tmp = erosion(tmp, np.full((2,1), 255))
tmp = dilation(tmp, np.full((2, 1), 255))

tmp = dilation(tmp, np.full((10, 10), 255))

x, y = tmp.shape
for i in range(x):
       for j in range(y):
              tmp[i][j] = 255 if graveyard[i][j] == 255 and  tmp[i][j] == 255 else 0

# source for connected components:
# https://stackoverflow.com/questions/46441893/connected-component-labeling-in-python
tmp = np.array(tmp, dtype=np.uint8)
lret, labels = cv2.connectedComponents(tmp)
label_hue = np.uint8(179*labels/np.max(labels))
blank_ch = 255*np.ones_like(label_hue)
labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])
labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)
labeled_img[label_hue==0] = 0

cv2.imshow('image', labeled_img)
cv2.waitKey()
```

The script starts by opening the image with OpenCV and acquiring RGB pixel values. Next, we threshold the blue histogram with a threshold value that is high enough, to be able to differentiate aircrafts in the image. Next, we apply opening with small line kernels in succession to eliminate as much noise as possible. Then, we dilate the noise-free image for larger aircraft boundaries, and apply logical-and operation with the thresholded image so that only pixels which appear with density 255 in both images are visible in the output. Lastly, we apply connected component analysis and false coloring for visualization.

For discussion, first issue was to figure out how to split color histograms using OpenCV. There were alternative pixel-based approaches that required traversal, however for performance purposes we decided to use *cv2.split* function from OpenCV. This way, we easily acquired RGB histograms even though we only use blue histogram.

Next, we needed to decide how to threshold the grayscale image obtained by splitting into color histograms. We tried 3 alternatives: first was to use Otsu's method on the grayscale image before splitting color histograms. This proved insufficient since frequency of noise in the image was too high to handle. Then, we tried adaptive thresholding with block size 11 and constant 50. Values less than 50 proved less efficient in eliminating noise, however values greater than or equal to 50 eliminates aircraft boundaries as well as the noise in the image. If we were to choose this approach, the output of

connected component analysis would fail to differentiate individual planes since boundaries are not complete, resulting in a single connected region. Lastly, we tried thresholding with limit 225 on blue histogram. This option successfully distinguished aircrafts from their surroundings, therefore we used this option.

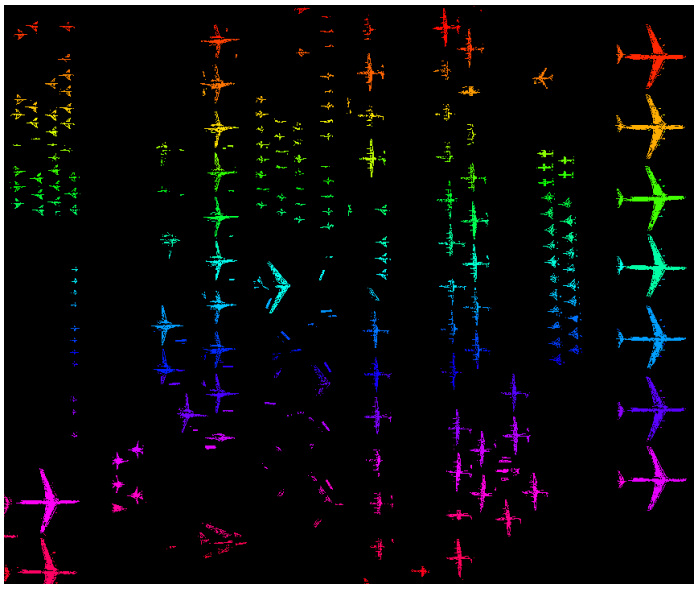
Next, we applied opening with small line segments to eliminate the noise in the image, at the same time avoiding damage to aircraft boundaries. We used different line segments in terms of size. Any option longer than *np.full((1, 2), 255)* or *np.full((2, 1), 255)* seemed inefficient in preserving aircraft boundaries, so we decided to use the values just above.

Next, we applied logical and operation on thresholded image with noise-free, dilated version of it, in order to clearly acquire aircraft boundaries without noise. This approach is similar to the one in Question 2, and proved just as efficient.

Lastly, we applied connected components analysis using *cv2.connectedComponents*, and used a pre-existing labeling algorithm found here: *https://stackoverflow.com/questions/46441893/connected-component-labeling-in-python*. Here, we observed the superiority of Matlab over OpenCV, when compared their easiness of use and performance in terms of this homework. Finding a functional connected components analysis and labeling algorithm for OpenCV required significant amount of research, while in Matlab same can be done with a single function call. Also, please note that since there is no ready-to-use coloring algorithm as in Matlab, our output is different from the sample output located in the assignment.

Hardest part of this problem, for us, was to figure out an algorithm for preprocessing. In previous attempts, we applied thresholding on the original image without color histograms. It proved difficult to eliminate noise and preserve aircraft boundaries at the same time, while working in extremely high time complexity caused by a large number of sequential morphological operations. To give a perspective, one such attempt completed in ~5 minutes. However, problem became easier when we used the blue color histogram. Even then, it was difficult to obtain all aircrafts(specifically ones that are tiny) in the original image without using filters, since we could easily eliminate salt-and-pepper noise with median filter without losing boundaries.

Output image:



Question 4

Part 1: Train Station

Python script for Frame 2:

```
station_background = cv2.imread("../../Data/station/0.png")
station_1 = cv2.imread("../../Data/station/1.png")

station_background_gray = cv2.cvtColor(station_background, cv2.COLOR_BGR2GRAY)
station_1_gray = cv2.cvtColor(station_1, cv2.COLOR_BGR2GRAY)
diff_1 = station_1_gray - station_background_gray
ret, tmp = cv2.threshold(diff_1, 80, 255, cv2.THRESH_BINARY)

tmp = erosion(tmp, np.array([[0, 255, 0], [255, 255, 255], [0, 255, 0]]))
tmp = erosion(tmp, np.full((11, 11), 255))
tmp = dilation(tmp, np.full((11, 11), 255))
save = tmp
tmp = erosion(tmp, np.full((50, 50), 255))
tmp = dilation(tmp, np.full((250, 200), 255))
tmp = save - tmp
tmp = dilation(tmp, np.full((250, 50), 255))

x, y = tmp.shape
for i in range(x):
       for j in range(y):
              tmp[i][j] = 255 if tmp[i][j] == 255 and save[i][j] == 255 else 0

tmp = np.array(tmp, dtype=np.uint8)
lret, labels = cv2.connectedComponents(tmp)
label_hue = np.uint8(179*labels/np.max(labels))
blank_ch = 255*np.ones_like(label_hue)
labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])
labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)
labeled_img[label_hue==0] = 0

cv2.imshow('image', labeled_img)
cv2.waitKey()
```

For frame 2, we first convert the background and frame 2 into gray-scale and substract background from frame 2. This way, we acquire pixels that have different values in two images. Then, we apply thresholding to convert resulting gray-scale image into a binary one. Next, we apply an erosion and a successive opening to reduce background image into a specific area just underneath the person. This image is assigned to a variable named *save.* Then, we apply successive erosion and dilation operations to acquire the person's head. This way, we later dilate the head region and apply logical-and operation with variable *save* to acquire the person's boundaries. Then, we apply connected components analysis and color the connected regions.

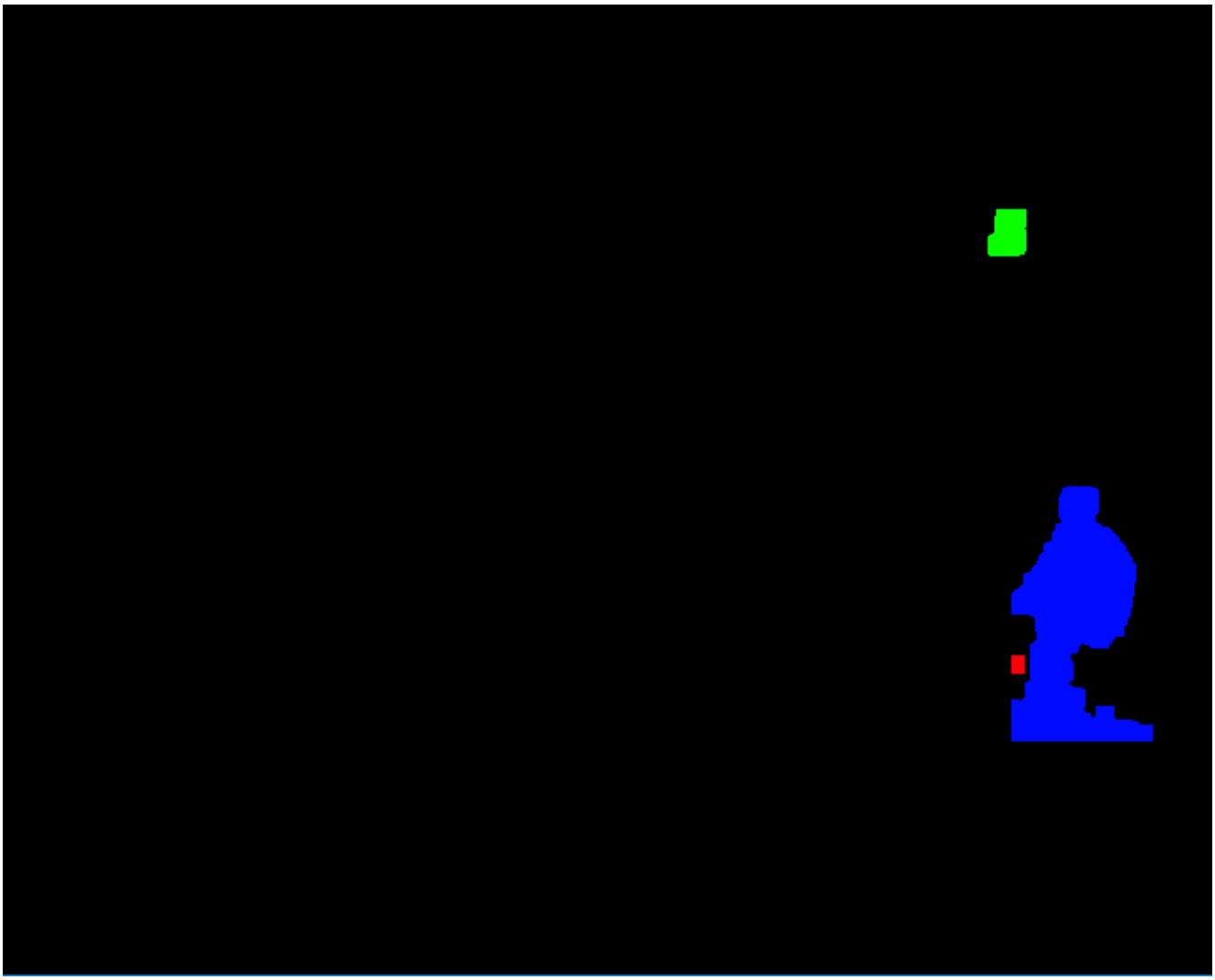
First issue with this solution is that overall time complexity is high, since we use multiple morphological operations in succession. Next, because frequency of noise in the thresholded image is quite high beneath the person, it was hard to acquire an accurate boundary of his legs and feet. Lastly, we were unable to fully remove the noise in the image: a rectangular region above the person's head is visible in the output, since to eliminate this region we'd lost the person's head as well.

In our experience, it was difficult to find a good threshold value that would eliminate as much noise as possible while preserving the person's boundary. As we change the threshold value, we observed that either the noise below the person's legs grow or the boundary of the person becomes indistinguishable from the background. What we tried was basically to form a balance between these two outcomes.

Next, it was difficult to determine the kernel shape for each operation. Since determining the kernel is a crucial step in morphological operations, we empirically choose specific values among many alternatives. However, it is possible to find better ones than those we use.

Also, the kernels we use are sometimes extremely large rectangular regions. This fact points out that our algorithm may be simplified into a core set of morphological, logical, and arithmetic operations we failed to determine. However, more effort is required to determine whether this is the case.

Output is as follows:



Python script for Frame 3:

```
station_background = cv2.imread("../../Data/station/0.png")
station_2 = cv2.imread("../../Data/station/2.png")
station_background_gray = cv2.cvtColor(station_background, cv2.COLOR_BGR2GRAY)
station_2_gray = cv2.cvtColor(station_2, cv2.COLOR_BGR2GRAY)
diff_2 = station_2_gray - station_background_gray
ret, tmp = cv2.threshold(diff_2, 50, 255, cv2.THRESH_BINARY)

tmp = erosion(tmp, np.array([[0, 255, 0], [255, 255, 255], [0, 255, 0]]))
tmp = dilation(tmp, np.array([[0, 255, 0], [255, 255, 255], [0, 255, 0]]))
tmp = erosion(tmp, np.full((9, 9), 255))
tmp = dilation(tmp, np.full((25, 9), 255))

save = tmp

tmp = erosion(tmp, np.full((1, 55), 255))
tmp = dilation(tmp, np.full((1, 75), 255))

x, y = tmp.shape
for i in range(x):
	for j in range(y):
		tmp[i][j] = 255 if tmp[i][j] == 0 and save[i][j] == 255 else 0

tmp = erosion(tmp, np.full((13, 23), 255))
tmp = dilation(tmp, np.full((71, 41), 255))

x, y = tmp.shape
for i in range(x):
	for j in range(y):
		tmp[i][j] = 255 if tmp[i][j] == 255 and save[i][j] == 255 else 0

tmp = dilation(tmp, np.full((25, 1), 255))
tmp = erosion(tmp, np.full((1, 5), 255))

tmp = np.array(tmp, dtype=np.uint8)
lret, labels = cv2.connectedComponents(tmp)
label_hue = np.uint8(179*labels/np.max(labels))
blank_ch = 255*np.ones_like(label_hue)
labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])
labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)
labeled_img[label_hue==0] = 0

cv2.imshow('image', labeled_img)
cv2.waitKey()
```

In this section we start just as in previous frame, by subtracting background from the frame and thresholding. Next, we apply opening, dilation, erosion, and logical-and in arbitrary order to minimize the noise in the image. Lastly, before applying connected components analysis and coloring, we enlarge the boundaries of the people so that disconnected body parts are merged, to get fully connected bodies.

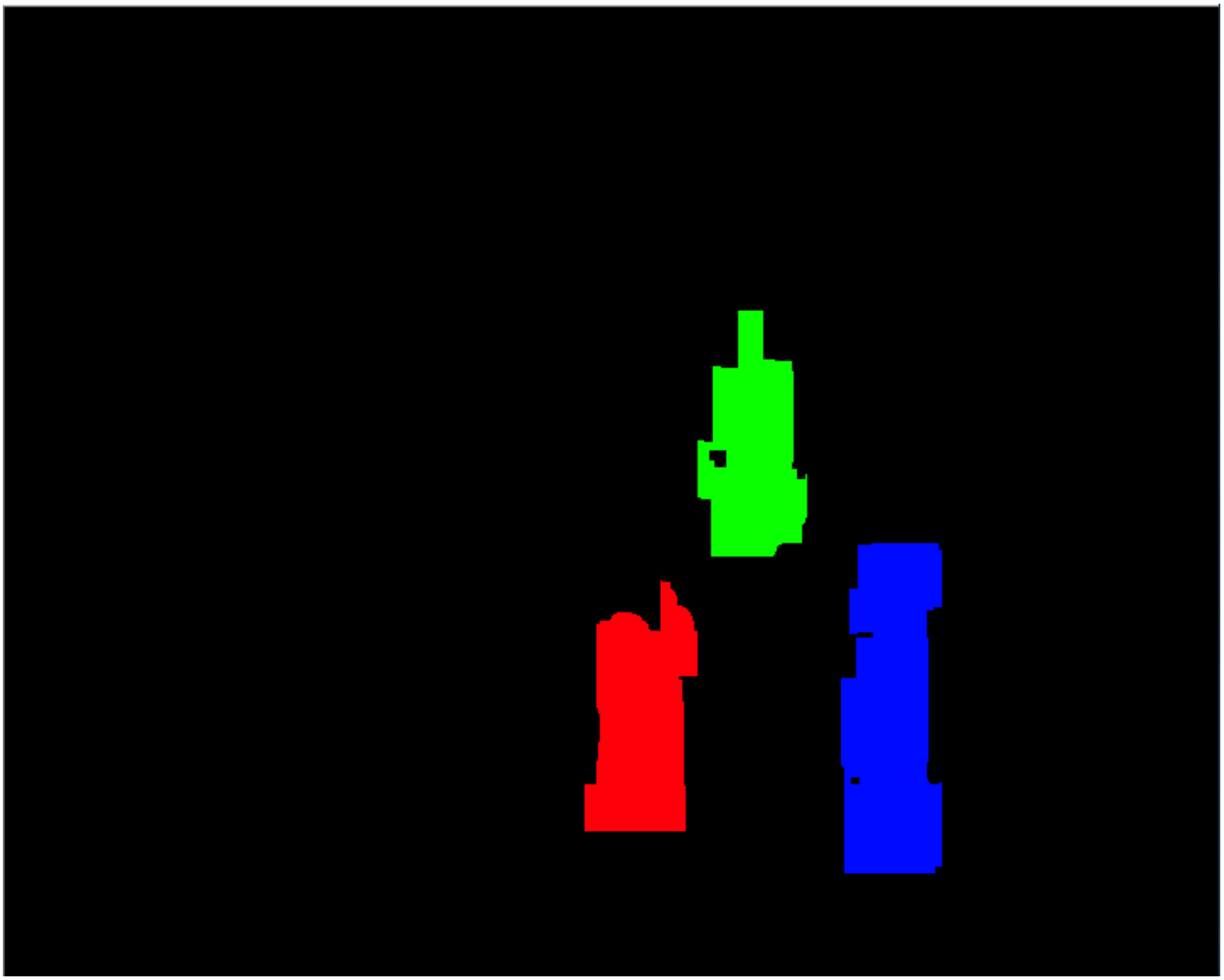
First issue with this solution is high time complexity caused by many sequential morphological operations. Second issue is the loss of boundaries: in the original image, we can see that people have different poses whereas in the output they all seem to be standing straight without moving. We can also see that some areas that do not belong to any person in the image is colored as “person” (for example

briefcase belonging to the person in bottom left side). This is the result of morphological operations we use to reduce the background noise caused.

Overall, our solution seems to successfully mark the boundaries of the people in the image. However, it is not an accurate one. One alternate solution would be to split the image into three rectangular parts, each contains one person and some background, then applying similar operations as above over a smaller region. By doing so we could reduce the noise further without using filters. However, this approach seems too case-specific, thus we decided not to follow it.

In our experience, it was difficult to determine which kernel to use in each operation. In many cases, we loaded the image into Microsoft Paint to count pixels, trying to decide kernels that would eliminate/ enlarge specific regions through morphological operations. However, because from previous frame, we roughly had a working algorithm, this solution took significantly less effort than the previous one.

The output is as follows:



Python script for Frame 4:

```
station_background = cv2.imread("../../Data/station/0.png")
station_3 = cv2.imread("../../Data/station/3.png")
station_background_gray = cv2.cvtColor(station_background, cv2.COLOR_BGR2GRAY)
station_3_gray = cv2.cvtColor(station_3, cv2.COLOR_BGR2GRAY)
diff_3 = station_3_gray - station_background_gray

ret, tmp = cv2.threshold(diff_3, 220, 255, cv2.THRESH_BINARY_INV)
save = tmp
tmp = erosion(tmp, np.full((9,1),255))
tmp = dilation(tmp, np.full((9,1), 255))
tmp = erosion(tmp, np.full((5,5),255))
tmp = dilation(tmp, np.full((11,1), 255))
tmp = dilation(tmp, np.full((1,11), 255))

x, y = tmp.shape
for i in range(x):
       for j in range(y):
              tmp[i][j] = 255 if tmp[i][j] == 255 and save[i][j] == 255 else 0

save2 = tmp
tmp = erosion(tmp, np.full((93,43), 255))
tmp = dilation(tmp, np.full((701,151), 255))
tmp = dilation(tmp, np.full((25,25), 255))

x, y = tmp.shape
for i in range(x):
       for j in range(y):
              tmp[i][j] = 255 if tmp[i][j] == 255 and save2[i][j] == 255 else 0

tmp = erosion(tmp, np.full((5,5), 255))
tmp = dilation(tmp, np.full((5,5), 255))

tmp = np.array(tmp, dtype=np.uint8)
lret, labels = cv2.connectedComponents(tmp)
label_hue = np.uint8(179*labels/np.max(labels))
blank_ch = 255*np.ones_like(label_hue)
labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])
labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)
labeled_img[label_hue==0] = 0

cv2.imshow('image', labeled_img)
cv2.waitKey()
```

In this section we start by subtracting background from the frame and then threshold the result. Next, we apply opening, dilation, erosion, and logical-and in arbitrary order to minimize the noise in the image. Lastly, before applying connected components analysis and coloring, we apply an opening to remove the noise around the boundaries that previous operations fail to eliminate.

First issue with this solution is high time complexity caused by many sequential morphological operations as well. Second problem is the amount of noise thresholding results in. In majority of this algorithm we try to remove the noise in the right side of the image while preserving the people's boundaries. It is possible to reduce the noise elimination into a smaller algorithm, however this requires further work to prove.

This problem was harder to solve in general, compared to the previous ones. One reason was the noise present in the image, which complicated region extraction process as we discuss below. Another reason was determining kernel size for each morphological operation we apply. Because kernel size affects the time complexity of the operation, to get a reasonable execution time we empirically choose the values above among many alternatives carefully.

In our experience, noise elimination was the hardest part of this solution. Because the noise is present with high frequency and surface area, preserving the people's boundaries while eliminating noise was problematic. This problem is visible in the output as well: in order to merge the 3 separate regions belonging to the same person we need to apply dilation with a rectangular unit. Any kernel used for dilation would decrease the accuracy of the boundary around the people below. We observed that to merge the regions of interest we would require a large kernel, about size 50×50. However, this would lead to inaccurate representation of the people in the bottom part of the image since some areas that do not belong to either person would be marked as "person". This phenomenon is also visible in the outer boundary of the same region as well. Therefore, we decided to leave the image as it is.

Output is as follows:

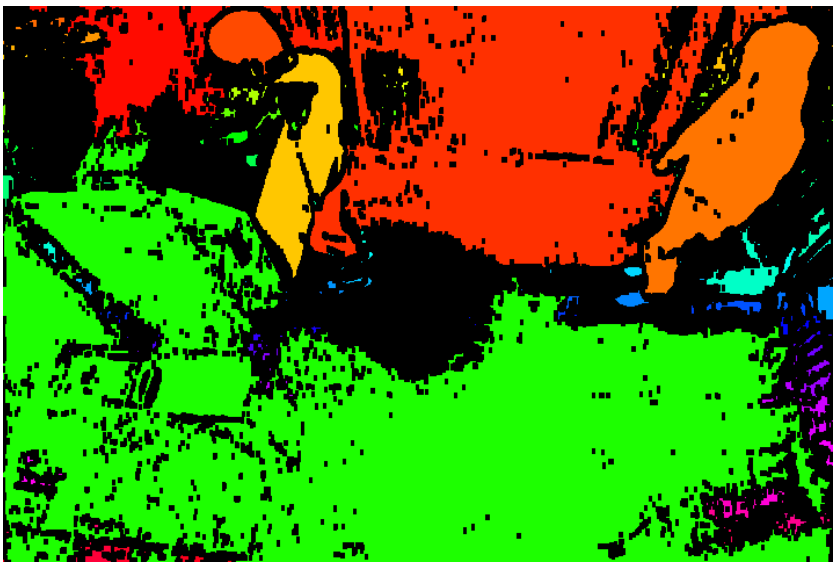
Part 2: Photocopy Machine

Python script for Frame 2:

```
photocopy_background = cv2.imread("../../Data/copyMachine/0.png")
copy_1 = cv2.imread("../../Data/copyMachine/1.png")

photocopy_background_gray = cv2.cvtColor(photocopy_background, cv2.COLOR_BGR2GRAY)
copy_1_gray = cv2.cvtColor(copy_1, cv2.COLOR_BGR2GRAY)
diff_1 = copy_1_gray - photocopy_background_gray
ret, tmp = cv2.threshold(diff_1, 200, 255, cv2.THRESH_BINARY_INV)

tmp = erosion(tmp, np.full((5, 1), 255))
tmp = erosion(tmp, np.full((1, 3), 255))

tmp = np.array(tmp, dtype=np.uint8)
lret, labels = cv2.connectedComponents(tmp)
label_hue = np.uint8(179*labels/np.max(labels))
blank_ch = 255*np.ones_like(label_hue)
labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])
labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)
labeled_img[label_hue==0] = 0

cv2.imshow('image', labeled_img)
cv2.waitKey()
```

In this section we start by subtracting background from the frame and then threshold the result. Next, we apply two erosions to minimize the noise in the image. Lastly, we apply connected components analysis and coloring. Here, due to assignment deadline, we failed to eliminate all noise in the image.

In our experience, noise elimination was the hardest part of this solution. Because the noise is present with high frequency and surface area, preserving the people's boundaries while eliminating noise was problematic. Thresholded image contains large amount of noise, and this makes eliminating the noise extremely hard. In the output, we managed to distinguish one person clearly, while failing to merge the boundaries of the second one (on left). We believe boundaries of the person on the left can be enchanted further with vertical line segments, however assignment deadline permits us to experiment further. The output is as follows:



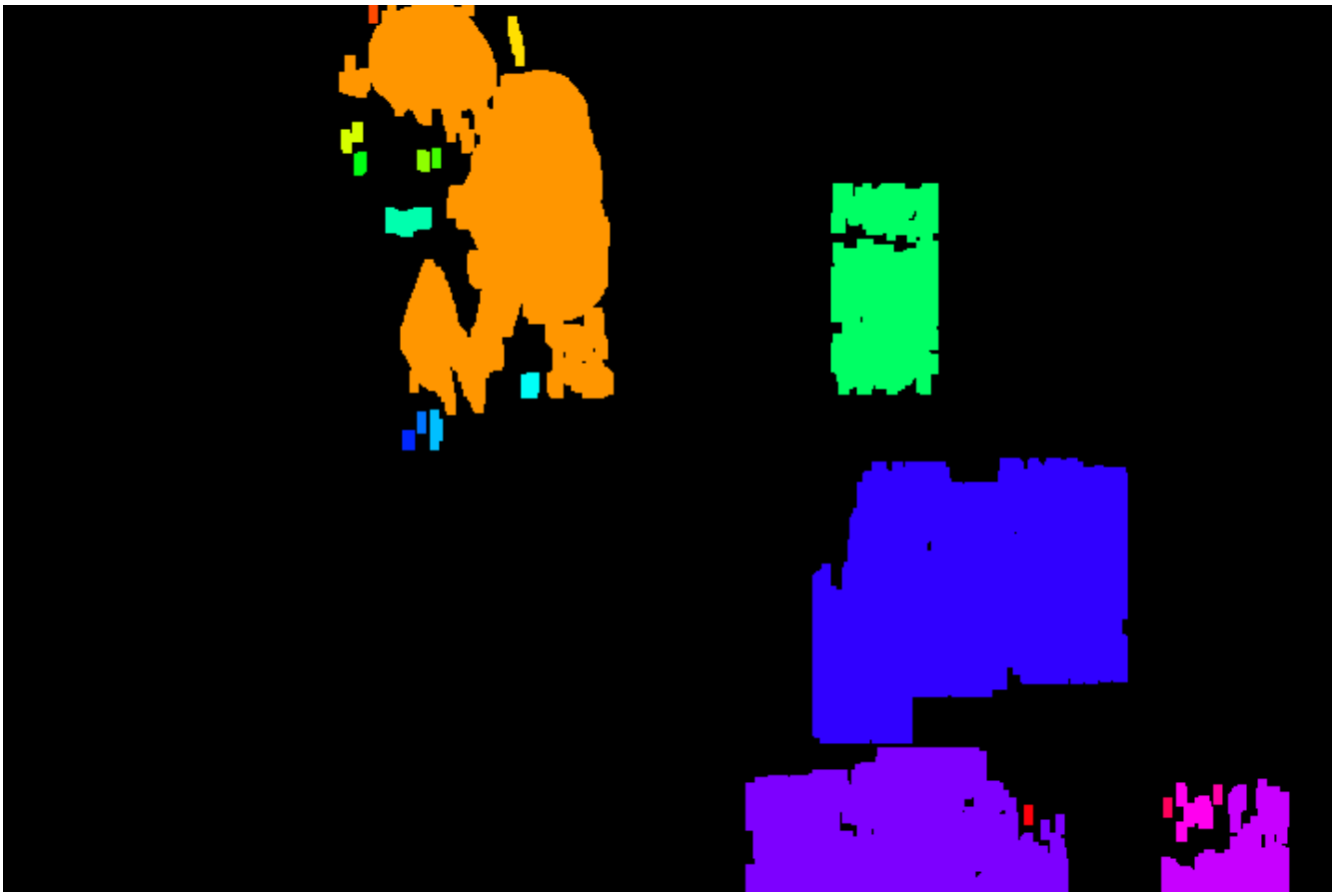
Python script for Frame 3:

```
photocopy_background = cv2.imread("../../Data/copyMachine/0.png")
copy_2 = cv2.imread("../../Data/copyMachine/2.png")

photocopy_background_gray = cv2.cvtColor(photocopy_background, cv2.COLOR_BGR2GRAY)
copy_2_gray = cv2.cvtColor(copy_2, cv2.COLOR_BGR2GRAY)
diff_2 = copy_2_gray - photocopy_background_gray
ret, tmp = cv2.threshold(diff_2, 200, 255, cv2.THRESH_BINARY_INV)

tmp = erosion(tmp, np.full((3,3), 255))
save = tmp

tmp = erosion(tmp, np.full((15,15), 255))
tmp = dilation(tmp, np.full((100, 50), 255))
tmp = erosion(tmp, np.full((15,15), 255))
tmp = dilation(tmp, np.full((15, 15), 255))

x, y = tmp.shape
for i in range(x):
       for j in range(y):
              tmp[i][j] = 255 if tmp[i][j] == 255 and save[i][j] == 255 else 0

tmp = dilation(tmp, np.full((11, 5), 255))

tmp = np.array(tmp, dtype=np.uint8)
lret, labels = cv2.connectedComponents(tmp)
label_hue = np.uint8(179*labels/np.max(labels))
blank_ch = 255*np.ones_like(label_hue)
labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])
labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)
labeled_img[label_hue==0] = 0

cv2.imshow('image', labeled_img)
cv2.waitKey()
```

In this section we start by subtracting background from the frame and then threshold the result. Next, we apply multiple erosion and dilation operations to minimize the noise in the image. Lastly, we apply connected components analysis and coloring. Here, due to assignment deadline, we failed to eliminate all noise in the image, however the person is visible enough with accurate boundaries.

For discussion, it was easier to detect the person on the left compared to detecting people in other frames of this part, however the color of the person's pants makes it harder to accurately draw boundaries around his legs. One solution might be to draw color histogram of the image to determine which colorband allows clearer differentiation, however assignment deadline prohibits further experiments.

The output is as follows:



Python script for Frame 4:

```
photocopy_background = cv2.imread("../../Data/copyMachine/0.png")
copy_3 = cv2.imread("../../Data/copyMachine/3.png")

photocopy_background_gray = cv2.cvtColor(photocopy_background, cv2.COLOR_BGR2GRAY)
copy_3_gray = cv2.cvtColor(copy_3, cv2.COLOR_BGR2GRAY)
diff_3 = copy_3_gray - photocopy_background_gray
ret, tmp = cv2.threshold(diff_3, 220, 255, cv2.THRESH_BINARY_INV)

save = tmp

tmp = dilation(tmp, np.full((3,3), 255))
tmp = erosion(tmp, np.full((3,3), 255))

x, y = tmp.shape
for i in range(x):
       for j in range(y):
              tmp[i][j] = 255 if tmp[i][j] == 255 and save[i][j] == 255 else 0

tmp = erosion(tmp, np.full((5,5), 255))

tmp = np.array(tmp, dtype=np.uint8)
lret, labels = cv2.connectedComponents(tmp)
label_hue = np.uint8(179*labels/np.max(labels))
blank_ch = 255*np.ones_like(label_hue)
labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])
labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)
labeled_img[label_hue==0] = 0

cv2.imshow('image', labeled_img)
cv2.waitKey()
```

In this section we start by subtracting background from the frame and then threshold the result. Next, we apply dilation, erosion, and logical-and in order to minimize the noise in the image. Lastly, we apply connected components analysis and coloring. Here, due to assignment deadline, we failed to eliminate all noise in the image; however, the person is clearly visible with accurate boundaries.

In our experience, noise elimination was the hardest part of this solution. Because the noise is present with high frequency and surface area, preserving the people's boundaries while eliminating noise was problematic. Thresholded image contains large amount of noise, and this makes eliminating the noise extremely hard. One solution would be to use square kernels of large size (11×11) for dilation, however because the person's boundary is quite thin, such a kernel is not preferable.

The output image is as follows: