Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Отчет по лабораторной работе №1
«Гистограммы, профили и проекции»
по дисциплине «Техническое зрение»

Выполнил: студент гр. R3338,
Кирбаба Д.Д.
Преподаватель: Шаветов С.В.,
канд. техн. наук, доцент ФСУ и Р

Санкт-Петербург, 2022

Цель работы

Освоение основных яркостных и геометрических характеристик изображений и их использование для анализа изображений.

Теоретическое обоснование применяемых методов

Основной единицей изображения является пиксель, который описывается следующими составляющими: положение (*x, y)*, интенсивность *I*. Для представления информации об интенсивности изображения строится гистограмма $Hist(i)$ – график, показывающий распределение пикселей по интенсивности *I*. Анализируя гистограмму, можно узнать сведения о том, какие тона преобладают в изображении, определить диапазон интенсивностей (контраст), детектировать объекты по определенным диапазонам яркости.

Арифметические операции с изображениями – простейшие методы для увеличения или уменьшения интенсивности объекта, однако следует помнить, что при данном преобразовании может теряться информация об изображении.

Также необходимо уметь строить кумулятивную гистограмму – это такая гистограмма, элементом i которой является сумма элементов гистограммы с индексами $\leq i$:

$$Hist_c(i) = \sum_{m=0}^{i} Hist(m).$$

Растяжение динамического диапазона, равномерное преобразование, экспоненциальное преобразование, преобразование по закону Рэлэя, преобразование по закону степени 2/3, гиперболическое преобразование – все эти способы анализа изображения по сути являются функциями, имеющими следующий общий вид:

$$I_{new} = f(I, \dots),$$

где I и I_{new} – массивы значений интенсивностей исходного и нового изображений соответственно. В зависимости от вида преобразования, в функциях используются также параметры (минимальное или максимальное значение интенсивности изображения, коэффициенты нелинейности, функция распределения вероятностей исходного изображения).

Различные преобразования используют для изменения свойств изображения (в данном случае яркостных свойств) как на отдельных частях, так и на целом изображении.

Для более эффективного преобразования можно не применять функции преобразования к каждому пикселю, а заранее вычислить все значения для массива $I = [0,1,2, \dots ,255]$. Таким образом останется только заменить исходные значения интенсивности на новые значения. Полученная в итоге таблица соответствия называется *LUT (Lookup table).* Данный метод имеет широкое практическое применение.

Также применяются следующие способы анализа изображения: профиль изображения (функция интенсивности по некоторому направлению) и проекция изображения на направление (сумма интенсивностей на некоторую прямую, перпендикулярную заданному направлению). С помощью данных методов можно выделять особые точки профиля или проекции, и таким

образом, получив информацию о контурах объекта, локализовать объект на монотонном фоне (проекция) или исследовать штрихкод (профиль).

Ход выполнения работы

1. Гистограммы

Picture 1: исходное слабоконтрастное изображение (слева – RGB, справа – Grayscale).

Контрастность изображения равна:

$$I_{max} - I_{min} = 255 - 83 = 17$$

1.1. Смещение гистограммы

Выполним смещение гистограммы на 100 градаций влево, так как исходное изображение слишком светлое.

Listing 1. Сдвиг гистограммы с помощью NumPy в Python.

```
# reading an image
img = cv.imread(img_data_folder + "snowforest.jpg",
cv.IMREAD_GRAYSCALE)

offset = -100

# make a shift
new_img = np.clip(img.astype(np.int16) + offset, 0,
255).astype(np.uint8)
```

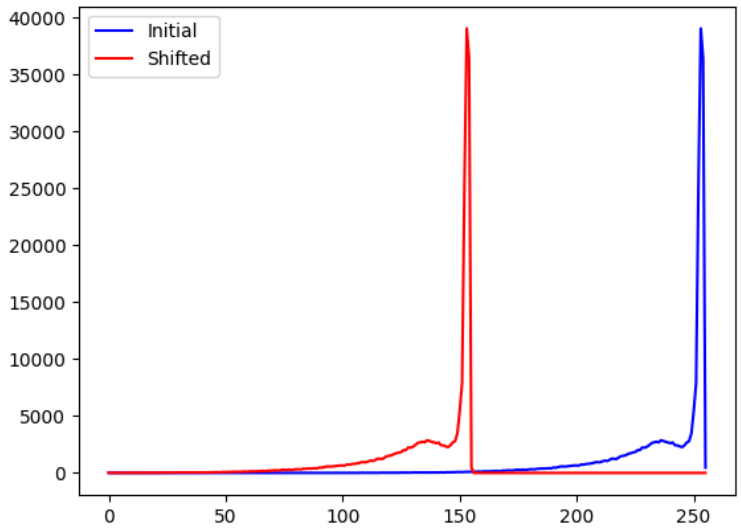

Figure 2: Гистограммы исходного и результирующего изображений.

Figure 3: Исходное (слева) и смещенное (справа) изображения.

Так как в исходном изображении преобладают светлые тона, то и на построенной гистограмме мы видим, что почти все пиксели имеют интенсивность в диапазоне $(210, 255)$.

Сдвигом на 100 тонов влево мы добились уменьшения яркости (картинка стала темнее) и, соответственно, смещения гистограммы.

1.2. Линейное выравнивание

Применим метод линейного выравнивания изображения для того, чтобы увеличить контраст и распределить более равномерно количество пикселей по интенсивностям.

Listing 2. Линейное выравнивание на Python.

```
# reading an image
img = cv.imread(data_folder + "snowforest.jpg", cv.IMREAD_GRAYSCALE)

l = 255 # max intensity
n_pix = img.size

# calculate cumulative histogram for the initial image
hist_size = 256
hist_range = (0, 256)
hist = cv.calcHist([img], [0], None, [hist_size], hist_range)
norm_hist = (l / n_pix) * hist
cumul_hist = np.clip(np.cumsum(norm_hist), 0, 255)

# get the linear aligned image
new_img = cumul_hist[img].astype(np.uint8)
```

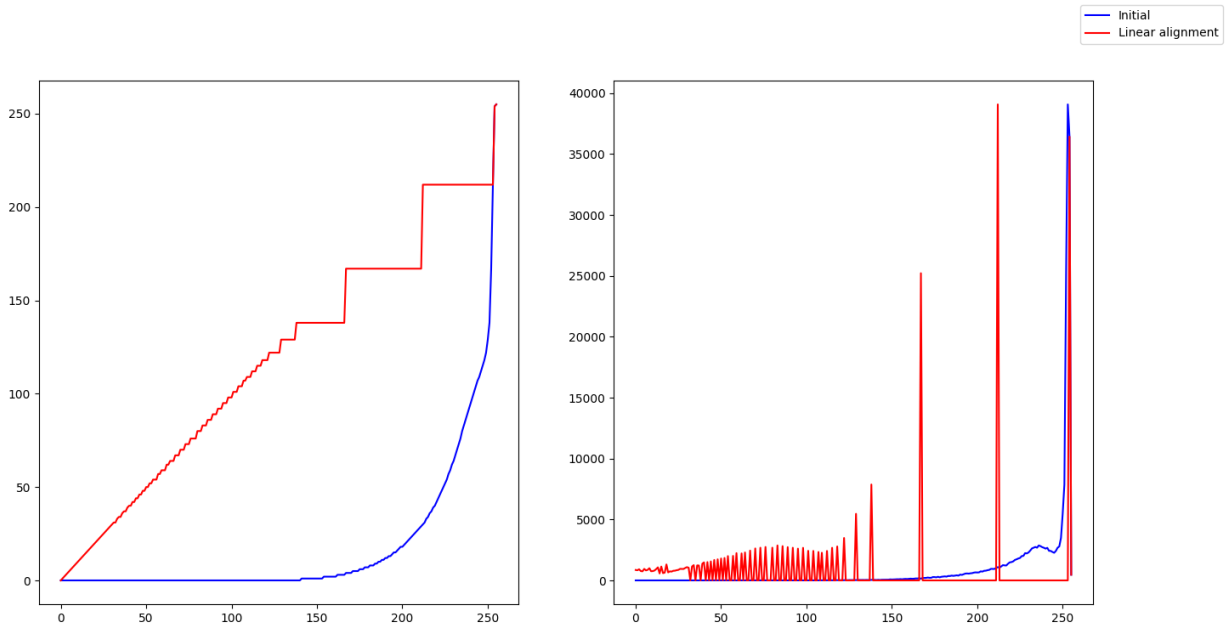

Figure 4: Кумулятивные гистограммы (слева) и обычные гистограммы (справа) двух изображений (до и после применения выравнивания).

Как мы видим, кумулятивная гистограмма после применения метода выравнивания стала похожа на прямую линию, что говорит о том, что пиксели с примерно одинаковой вероятностью имеют каждое значение интенсивности (так как кумулятивная гистограмма аппроксимирует функцию распределения пикселей по интенсивности). Однако из-за большого количества пикселей с интенсивностью $(240, 255)$ лежащих в узком диапазоне линия переходит в ломаную, из-за этого на новом изображении будут резкие переходы во светлых тонах.

Figure 5: Исходное (слева) и новое (справа) изображения.

Применим данный метод к другой картинке с более равномерным распределением пикселей.

Figure 6: Исходное (слева) и новое (справа) изображения.

Детали изображения стали более различимы. Если в исходном изображении темные интенсивности доминировали, то после применения метода, картинка стала более контрастной.

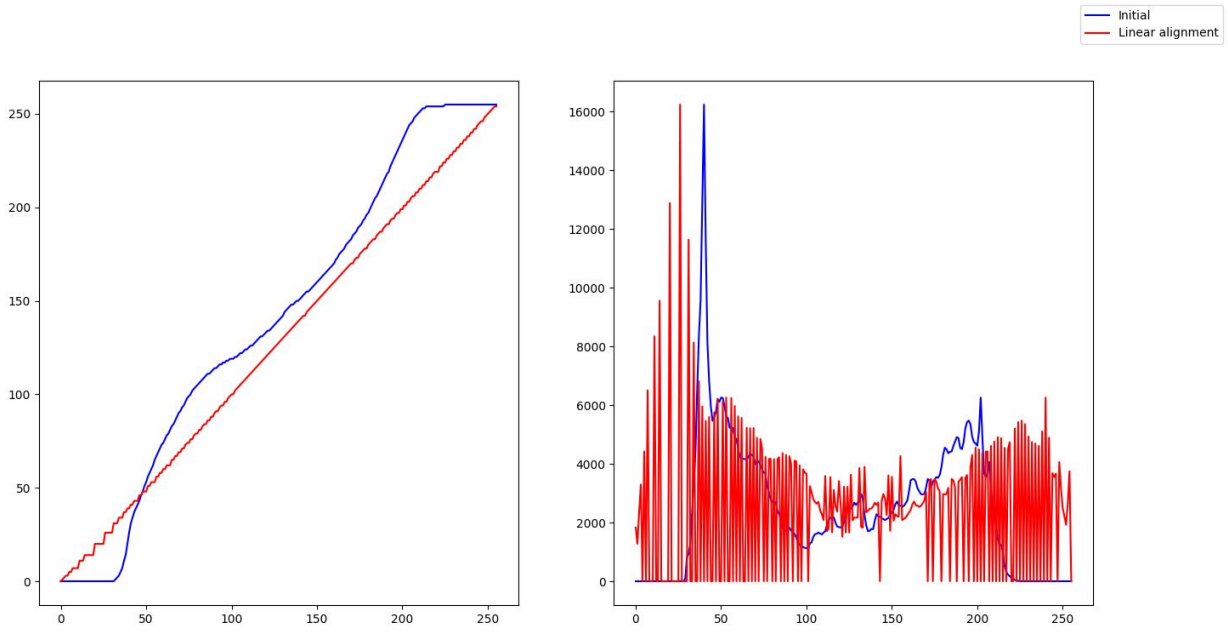

Figure 7: Кумулятивные гистограммы (слева) и обычные гистограммы (справа) двух изображений (до и после применения выравнивания).

Убедились, что проблема разрыва в светлых тонах была из-за наличия большого количества пикселей в узком диапазоне, так как при втором изображении метод работает хорошо.

1.3. Растяжение динамического диапазона

Как видно из формулы:

$$I_{new} = \left(\frac{I - I_{min}}{I_{max} - I_{min}}\right)^{\alpha} \quad , \alpha - \text{коэффициент нелинейности}$$

Данный метод позволяет расширить динамический диапазон за счет сдвига $(I - I_{min})$, а также влияет на новые интенсивности путем возведения в степень α.

Листинг 3. Растяжение динамического диапазона на Python.

```
# reading an image
img = cv.imread(img_folder + "snowforest.jpg", cv.IMREAD_GRAYSCALE)
i_max = np.max(img)
i_min = np.min(img)

# get the stretched image
non_linearity_factor = 0.7
new_img = np.power((img.astype(np.float32) - i_min) / (i_max - i_min),
non_linearity_factor)
new_img = (255 * new_img).astype(np.uint8)
```

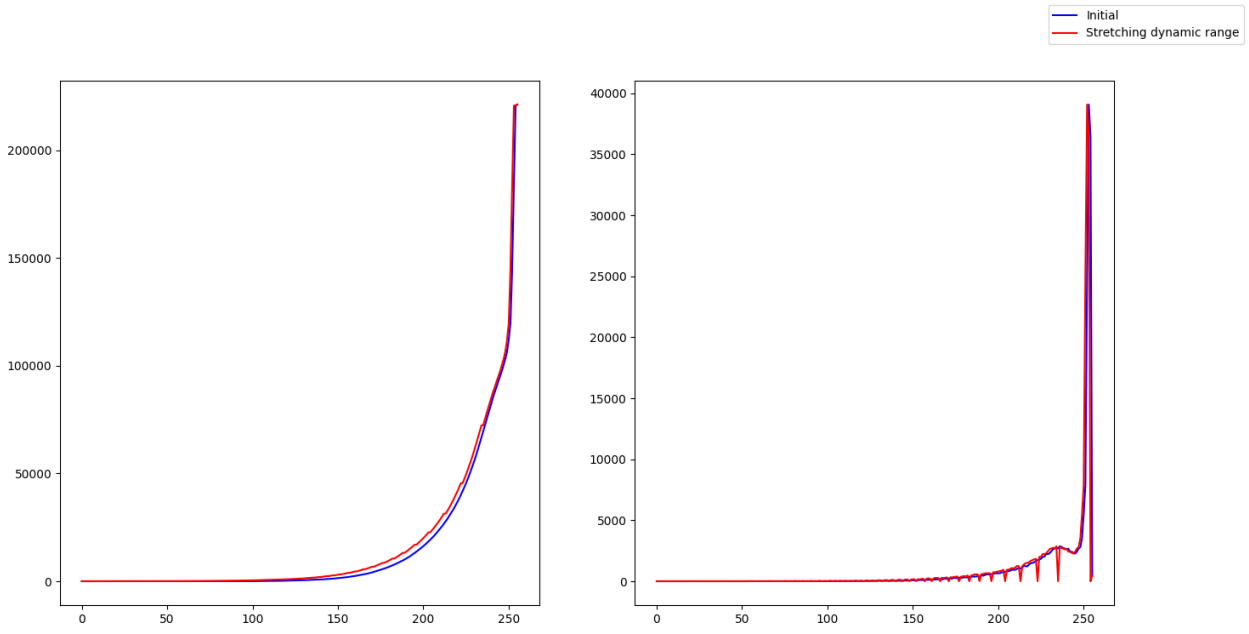

Figure 8: Кумулятивные и обычные гистограммы изображений до и после преобразований при $\alpha = 0.7$.

Figure 9: Исходное (слева) и новое (справа) изображения при $\alpha = 0.7$.

Применение метода не дало существенных результатов, так как коэффициент нелинейности мал ($\alpha = 0.7$) для нужного нам увеличения исходных интенсивностей.

Попробуем $\alpha = 7$.

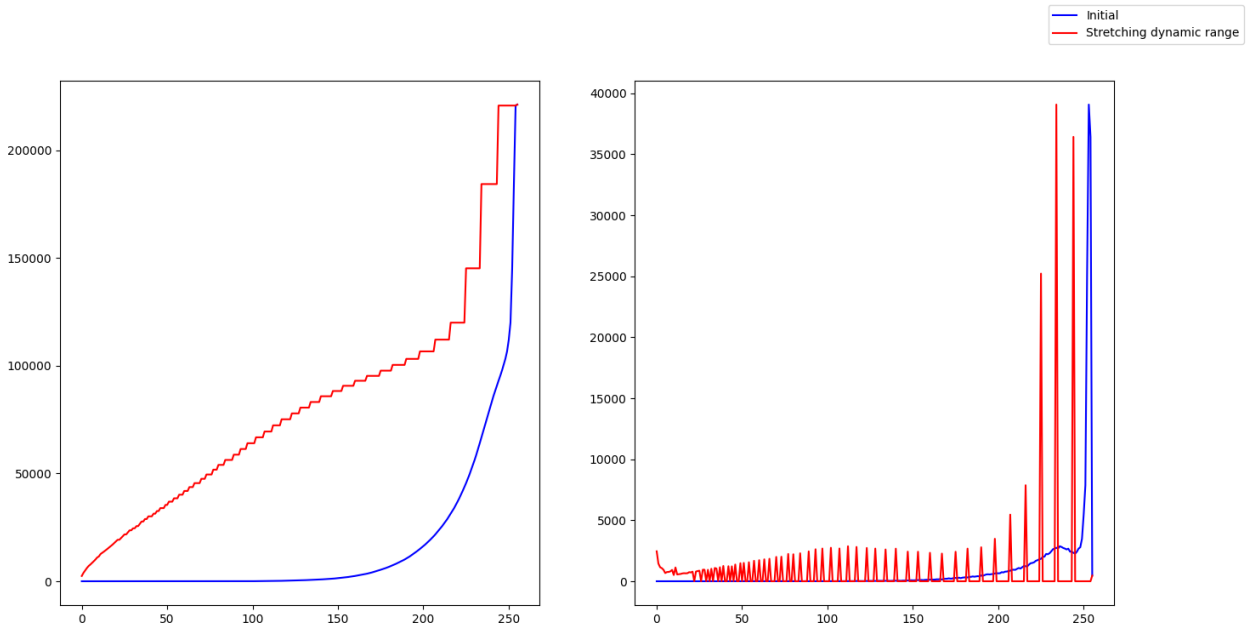

Figure 10: Кумулятивные и обычные гистограммы изображений до и после преобразования при $\alpha = 7$.

Figure 11: Исходное (слева) и новое (справа) изображения $\alpha = 7$.

В итоге, данный метод позволяет усилить или наоборот уменьшить исходную конфигурацию интенсивностей изображения в зависимости от выбора степени. Тогда же как рассмотренный ранее метод линейного выравнивания старается сравнять вероятности появления каждого значения из всего диапазона интенсивностей.

1.4. Равномерное преобразование

Данное преобразование практически аналогичен линейному выравниванию, за исключением того, что равномерное преобразование делает отступ на значение $+ I_{min}$, а значит, что диапазон интенсивностей будет меньше.

Listing 4. Равномерное преобразование на Python.

```
# reading an image
img = cv.imread(src_path + "snowforest.jpg", cv.IMREAD_GRAYSCALE)
i_max = np.max(img)
i_min = np.min(img)

# make a cumulative hist of the initial image
hist_range = [0, 256]
hist_size = 256
hist = cv.calcHist([img], [0], None, [hist_size], hist_range)
cumul_hist = np.cumsum(hist) / img.size

# get the transformed image
new_img = np.clip((i_max - i_min) * cumul_hist[img] + i_min, 0,
255).astype(np.uint8)
```

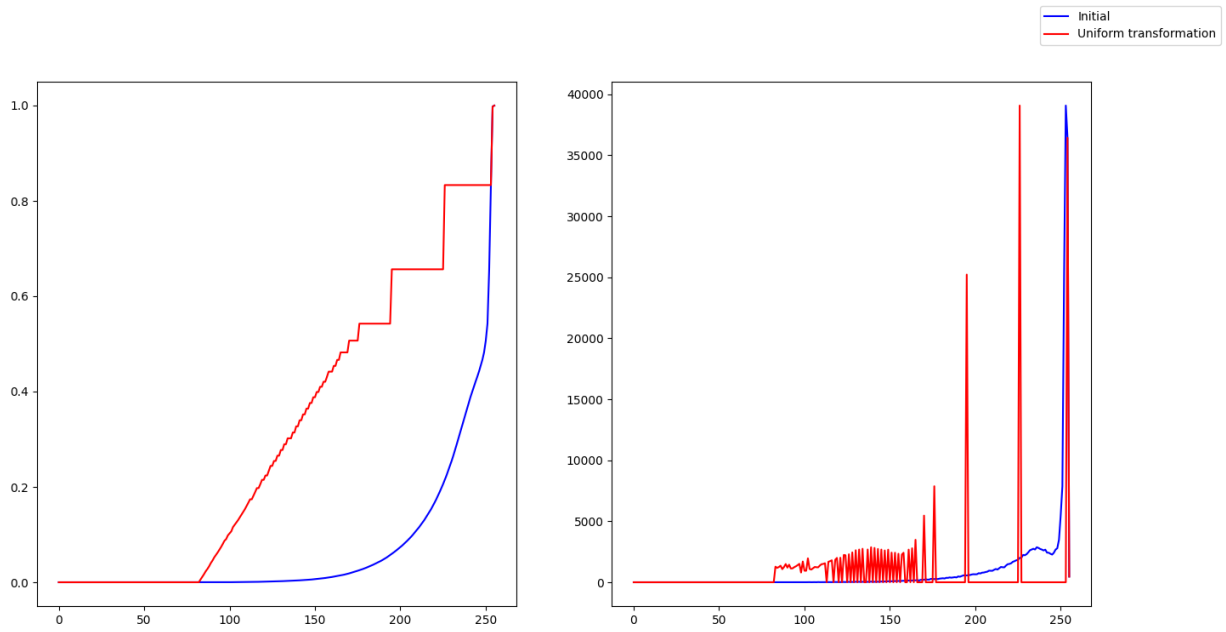

Figure 12: Кумулятивные и обычные гистограммы изображений до и после преобразования.

Figure 13: Исходное (слева) и новое (справа) изображения.

Сходство с методом равномерного выравнивания большое, однако из-за отступа тут не присутствуют темные тона.

1.5. Экспоненциальное преобразование

$$I_{new} = I_{min} - \frac{1}{\alpha} ln(1 - P(I))$$

Listing 5. Экспоненциальное преобразование на Python.

```
# reading an image
img = cv.imread(img_path + "snowforest.jpg", cv.IMREAD_GRAYSCALE)
i_min = np.min(img)

# make a cumulative hist of the initial image
hist_size = 256
hist_range = (0, 256)
hist = cv.calcHist([img], [0], None, [hist_size], hist_range)
cumul_hist = np.cumsum(hist) / img.size
cumul_hist = np.where(cumul_hist < 0.99999, cumul_hist, 0.99999) #
exclude unit values to avoid infinity in the logarithm afterwards


# get the transformed image
slope_factor = 2
new_img = np.clip(i_min - (1 / slope_factor) * 255 * np.log(1 -
cumul_hist[img]), 0, 255).astype(np.uint8)
```

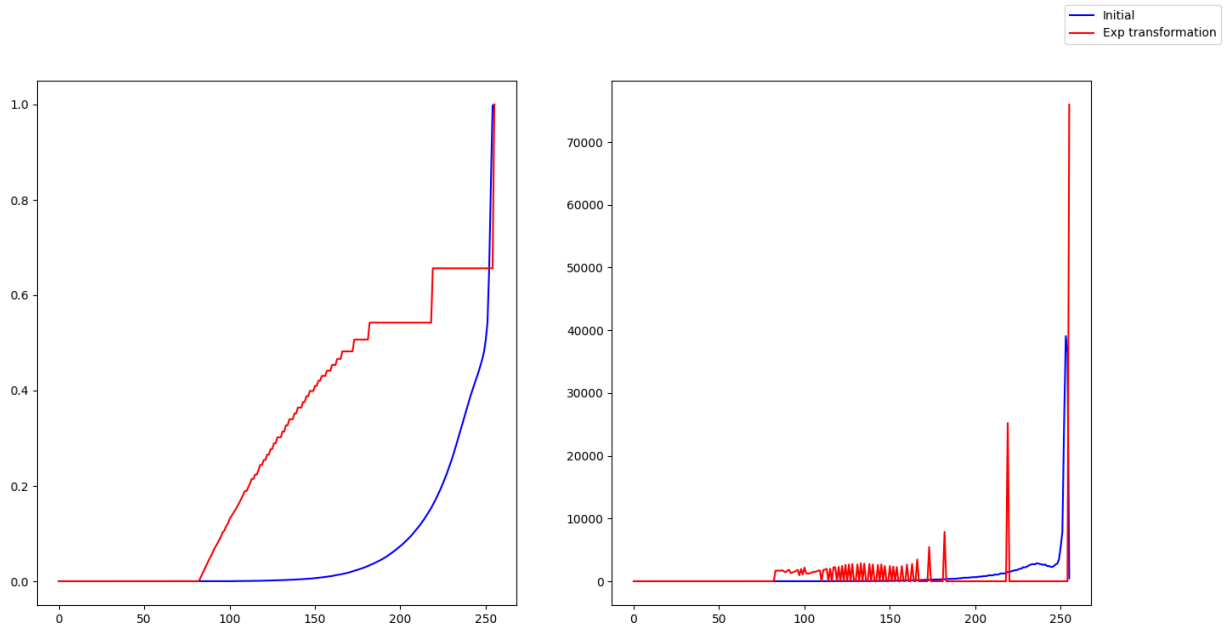

Figure 14: Кумулятивные и обычные гистограммы изображений до и после преобразования.

Figure 15: Исходное (слева) и новое (справа) изображения.

Данное преобразование не увеличивает диапазон интенсивностей, так как в формуле присутствует слагаемое $I_{min} + \cdots$. Также кумулятивная гистограмма будет выглядеть как сдвинутый логарифм с определенным коэффициентом, который является по сути гипер-параметром α.

1.6. Преобразование по закону Рэлея

$$I_{new} = I_{min} + \left(2\alpha^2 \ln\left(\frac{1}{1 - P(I)}\right)\right)^{\frac{1}{2}}$$

Listing 6. Преобразование по закону Рэлея на Python.

```
# reading an image
img = cv.imread(src_path + "snowforest.jpg", cv.IMREAD_GRAYSCALE)
i_min = np.min(img)

# make a cumulative hist of the initial image
hist_range = [0, 256]
hist_size = 256
hist = cv.calcHist([img], [0], None, [hist_size], hist_range)
cumul_hist = np.cumsum(hist) / img.size
cumul_hist = np.where(cumul_hist < 0.99999, cumul_hist, 0.99999) #
exclude unit values to avoid infinity in the logarithm afterwards

# get the transformed image
alpha = 0.3
new_img = i_min + 255 * (2 * (alpha ** 2) * np.log(1/(1 -
cumul_hist[img]))) ** (1 / 2)
new_img = np.clip(new_img, 0, 255).astype(np.uint8)
```

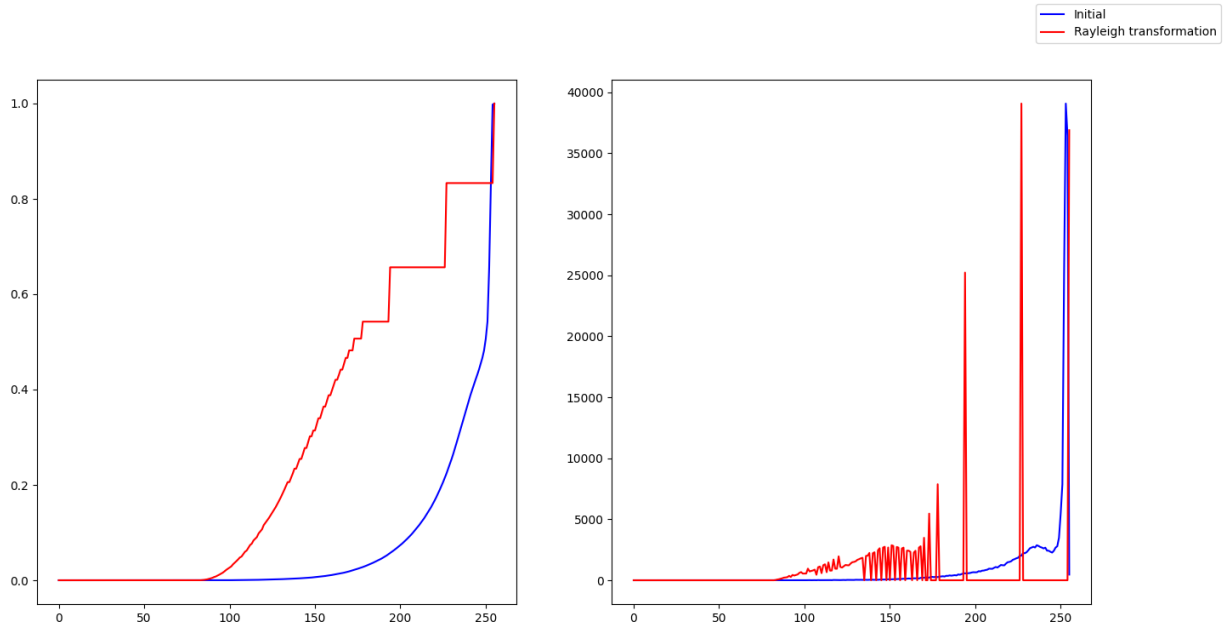

Figure 16: Кумулятивные и обычные гистограммы изображений до и после преобразования.

Figure 17: Исходное (слева) и новое (справа) изображения.

Данное преобразование схоже с экспоненциальным, однако кумулятивная гистограмма напоминает логистическую функцию, таким образом, наибольшее количество пикселей будет иметь интенсивность – среднюю из всего диапазона (так как в середине логистической функции – производная максимальна (это не так на моем примере – из-за обилия светлых тонов в узком диапазоне)). В то время как при экспоненциальном преобразовании наибольшее количество писклей будет стремиться иметь интенсивность в начале диапазона (рост логарифма наибольший в начале).

1.7. Преобразование по закону степени 2/3

$$I_{new} = P(I)^{\frac{2}{3}}$$

Данное преобразование будет очень схоже с экспоненциальным, так как также больше всего увеличивает интенсивность пикселей, находящихся в начале диапазона.

Также стоит отметить, что данное преобразование не имеет смещения и из-за этого будет увеличен диапазон интенсивностей.

Listing 7. Преобразование по закону степени 2/3 на Python.

```
# reading an image
img = cv.imread(src_path + "snowforest_small_gray.jpg",
cv.IMREAD_GRAYSCALE)

# make a cumulative hist of the initial image
hist_range = [0, 256]
hist_size = 256
hist = cv.calcHist([img], [0], None, [hist_size], hist_range)
cumul_hist = np.cumsum(hist) / img.size
```

```
# get the transformed image
new_img = cumul_hist[img] ** (2 / 3)
new_img = np.clip(new_img * 255, 0, 255).astype(np.uint8)
```

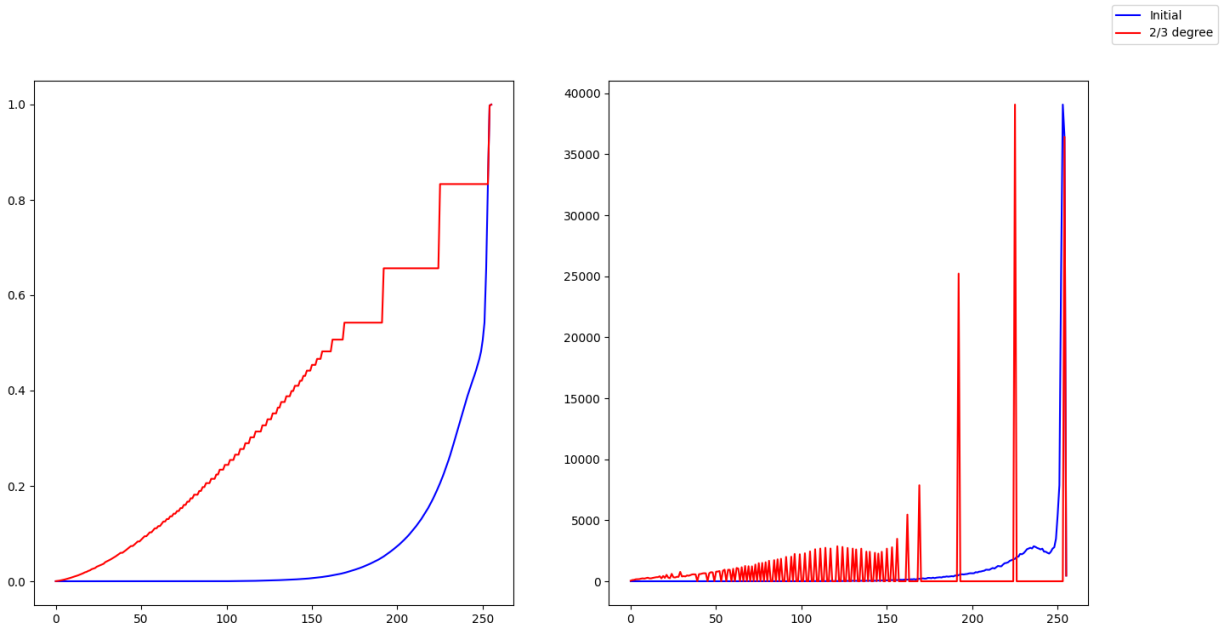

Figure 18: Кумулятивные и обычные гистограммы изображений до и после преобразования.

Figure 19: Исходное (слева) и новое (справа) изображения.

Преобразование по закону 2/3 степени будет стремится увести наибольшее количество пикселей в области с большой интенсивностью, так как наибольший рост кумулятивная гистограмма будет иметь ближе к концу диапазона интенсивности.

1.8. Гиперболическое преобразование

$$I_{new} = \alpha^{P(I)}, \qquad \alpha = I_{min}$$

Listing 8. Гиперболическое преобразование на Python.

```
# reading an image
img = cv.imread(src_path + "snowforest.jpg", cv.IMREAD_GRAYSCALE)

# make a cumulative hist of the initial image
hist_range = [0, 256]
hist_size = 256
hist = cv.calcHist([img], [0], None, [hist_size], hist_range)
cumul_hist = np.cumsum(hist) / img.size

# get the transformed image
alpha = np.min(img) / 255
new_img = alpha ** cumul_hist[img]
new_img = np.clip(new_img * 255, 0, 255).astype(np.uint8)
```

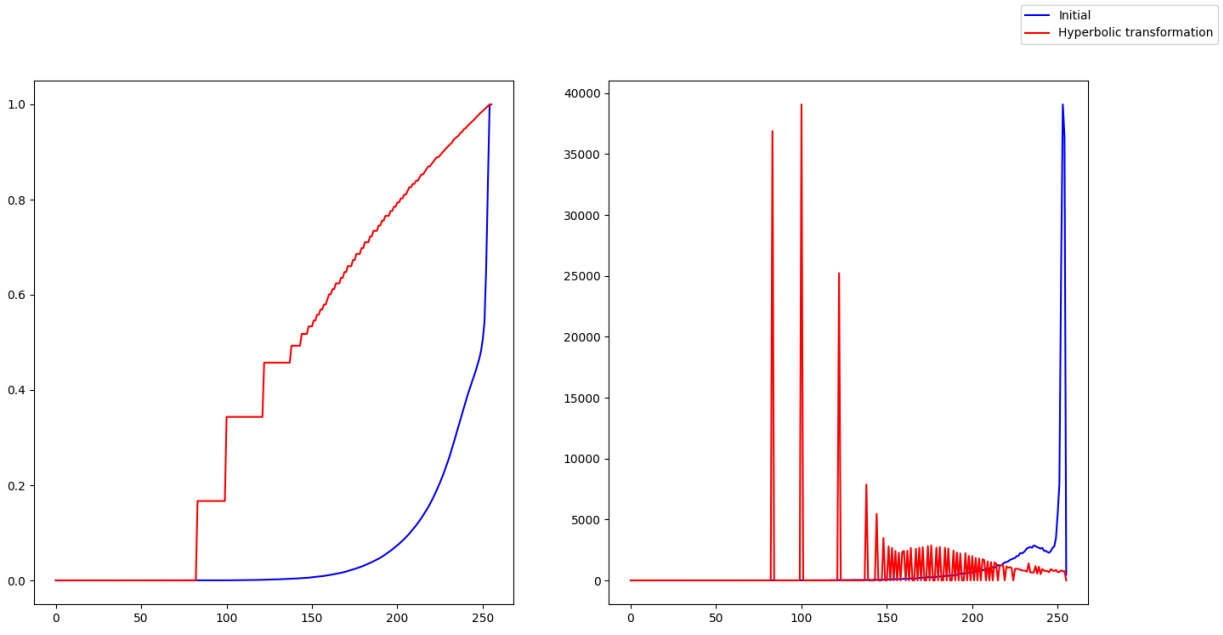

Figure 20: Кумулятивные и обычные гистограммы изображений до и после преобразования.

Figure 21: Исходное (слева) и результирующее (справа) изображения.

1.9. Встроенные функции преобразования

Рассмотрим две функции для автоматического выравнивания гистограммы из библиотеки OpenCV: *equalizeHist()* и *CLAHE*.

Listing 9. Встроенные функции преобразования библиотеки OpenCV на Python.

```
# reading an image
img = cv.imread(src_path + "snowforest_small_gray.jpg",
cv.IMREAD_GRAYSCALE)

# use equalizeHist()
eq_img = cv.equalizeHist(img)

# ust CLAHE
clahe = cv.createCLAHE()
clahe_img = cv.CLAHE.apply(clahe, img)
```

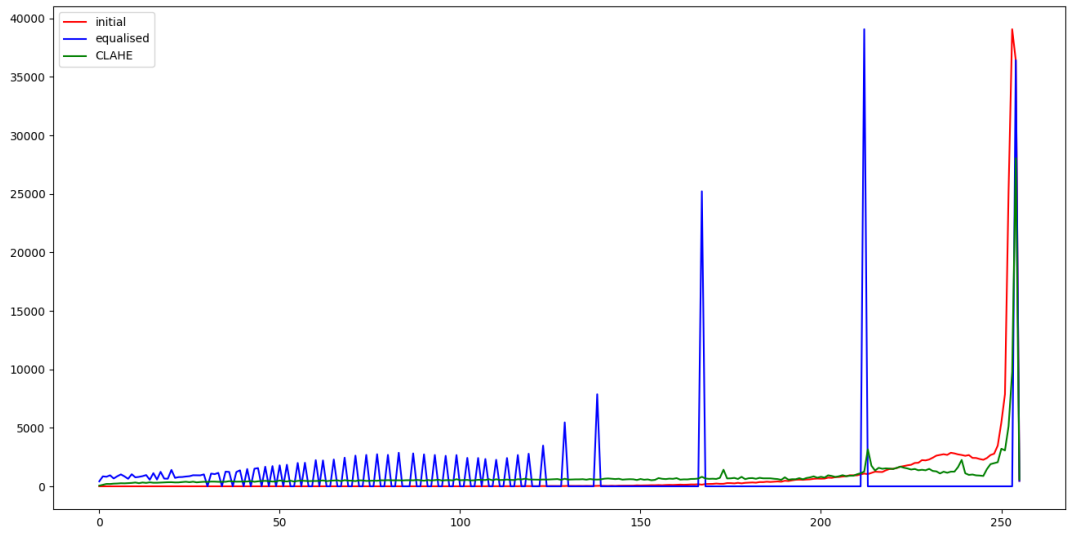

Figure 22: Гистограммы исходного изображения и двух изображений после преобразований.

Figure 23: Исходное изображение (слева), изображение после equalizeHist() (в центре) и изображение после CLAHE (справа)

Функция выравнивания гистограммы *equalizeHist()* дает очень схожий результат работы с преобразованием по закону степени 2/3 и обычным линейным выравниванием.

Метод *CLAHE* сработал лучше всего из всех предоставленных выше методов.

1.10. Таблица поиска

Применим метод *LUT* с реализацией динамического растяжения диапазона к изображению и сравним с таким же преобразованием, но без использования *LUT*.

Listing 10. LUT на Python.

```
# reading an image
img = cv.imread(src_path + "snowforest.jpg", cv.IMREAD_GRAYSCALE)
i_max = np.max(img)
i_min = np.min(img)

# create and fill lut
alpha = 0.7
lut = np.arange(256, dtype=np.uint8)
lut = (lut - i_min) / (i_max - i_min)
lut = np.where(lut > 0, lut, 0)
lut = np.clip(255 * np.power(lut, alpha), 0, 255).astype(np.uint8)

# get transformed image
img_new = cv.LUT(img, lut)
```

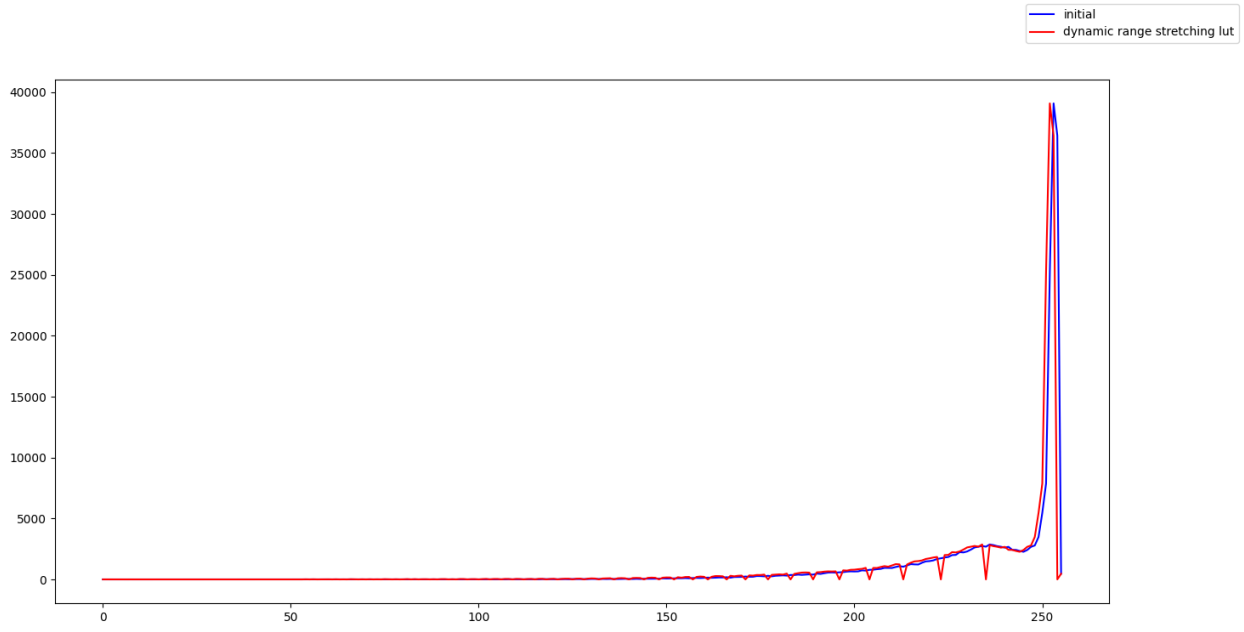

Figure 24: Гистограммы исходного и результирующего изображений.

Figure 25: Исходное (слева) и результирующее (справа) изображения.

Результаты полностью совпадают при применении как LUT, так и при прямом расчете интенсивностей при растяжении динамического диапазона, что и предполагалось.

Однако LUT работает намного быстрее (если речь идет о применении заранее вычисленной таблицы), так как нам необходимо запомнить всего лишь массив из 256 значений, а далее просто заменить исходные интенсивности на соответствующие в LUT. Таким образом, можно составить таблицы для любого преобразования яркостных характеристик изображения.

2. Профили

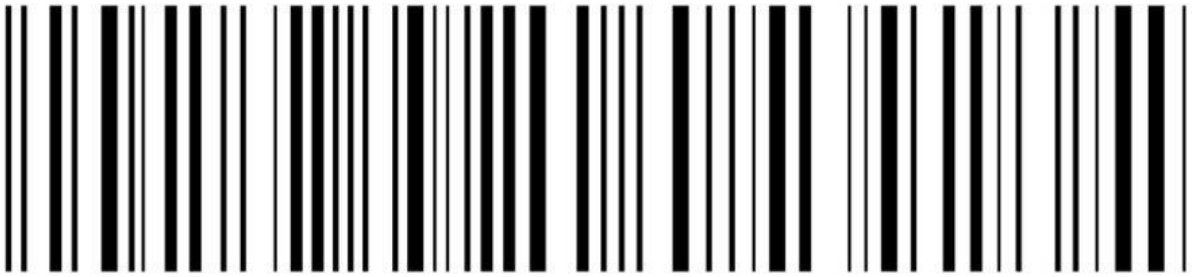

Figure 26: Исходное изображение со штрихкодом.

Построим профиль изображения вдоль горизонтальной линии, проходящей по середине изображения.

Listing 11. Построение профиля на Python.

```
# reading a barcode
img = cv.imread(src_path + "barcode.jpg", cv.IMREAD_GRAYSCALE)

# make a profile array, using slice of the initial img array
profile = img[int(img.shape[0] / 2), :]
```

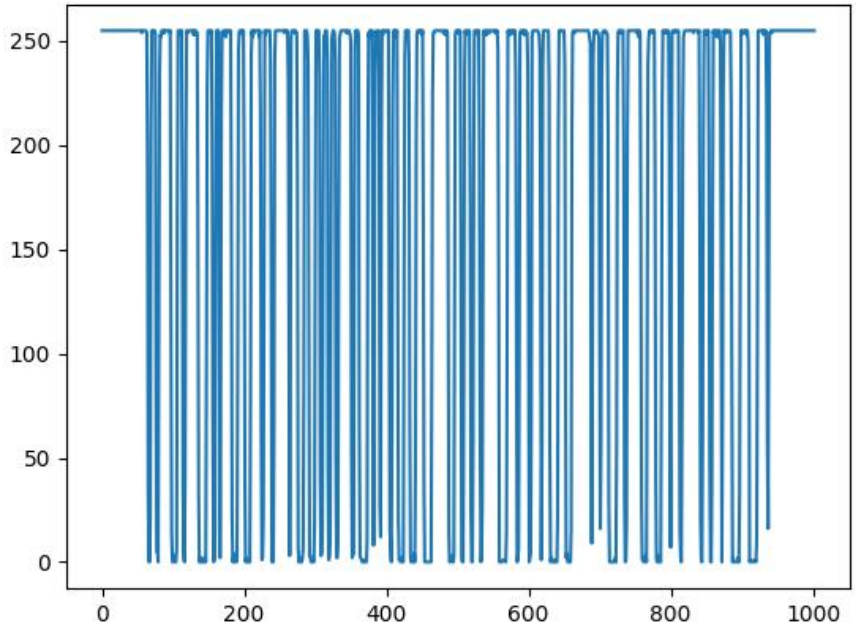

Figure 27: Профиль штрихкода.

Таким образом мы можем без потерь хранить данные штрихкода в виде одномерного массива, созданного с помощью профиля по горизонтали.

3. Проекции

Figure 28: Исходное изображение.

Попробуем определить прямоугольную границу круизного лайнера с помощью построения проекций и их анализа.

Listing 12. Построение проекций на Python.

```
# reading an image
img = cv.imread(src_path + "cruise.jpg", cv.IMREAD_GRAYSCALE)

# calculate projection for oX
x_proj = np.sum(img, axis=0)
# calculate projection for oY
y_proj = np.sum(img, axis=1)


# plot the projections graphs
plt.rcParams["figure.figsize"] = (16,8)
figure, axis = plt.subplots(1, 2)
axis[0].plot(np.arange(x_proj.shape[0]), x_proj, 'r', label="oX
projection")
```

```
axis[1].plot(y_proj, np.arange(y_proj.shape[0] - 1, -1, -1), 'b',
label="oY projection")
figure.legend(loc="upper right")
plt.show()
```

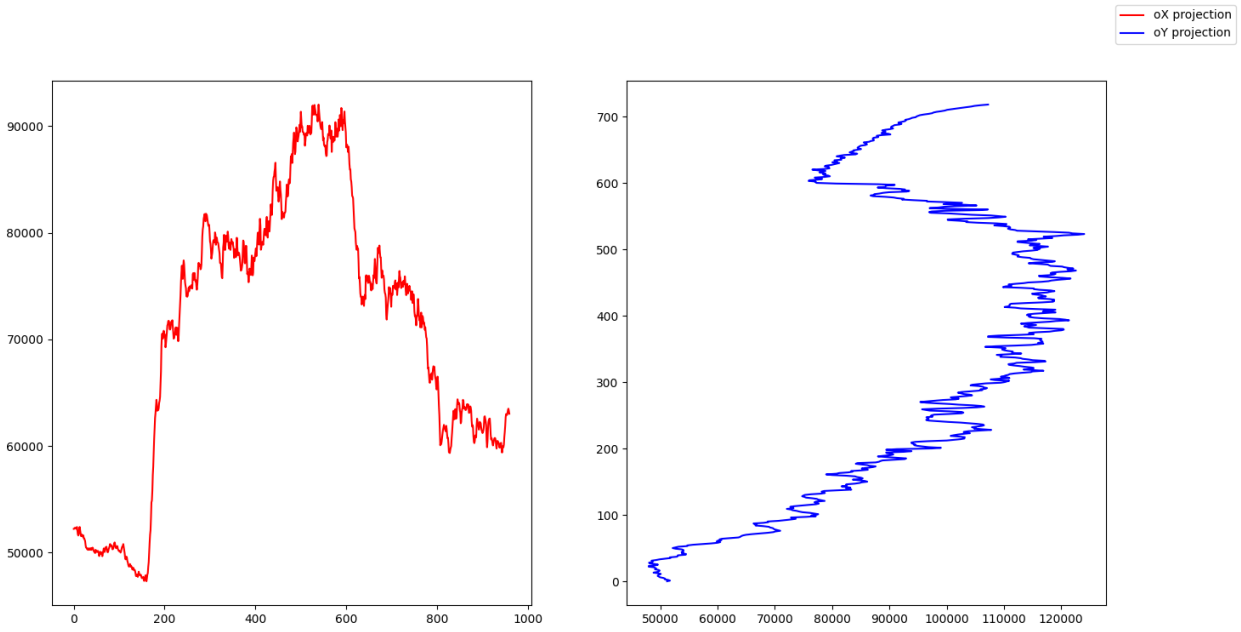

Figure 29: Графики проекций на оси oX (слева) и oY (справа).

По данным графикам можно примерно определить границы объекта.

oX: (160, 800); oY: (150, 620).

Изобразим границы на графиках:

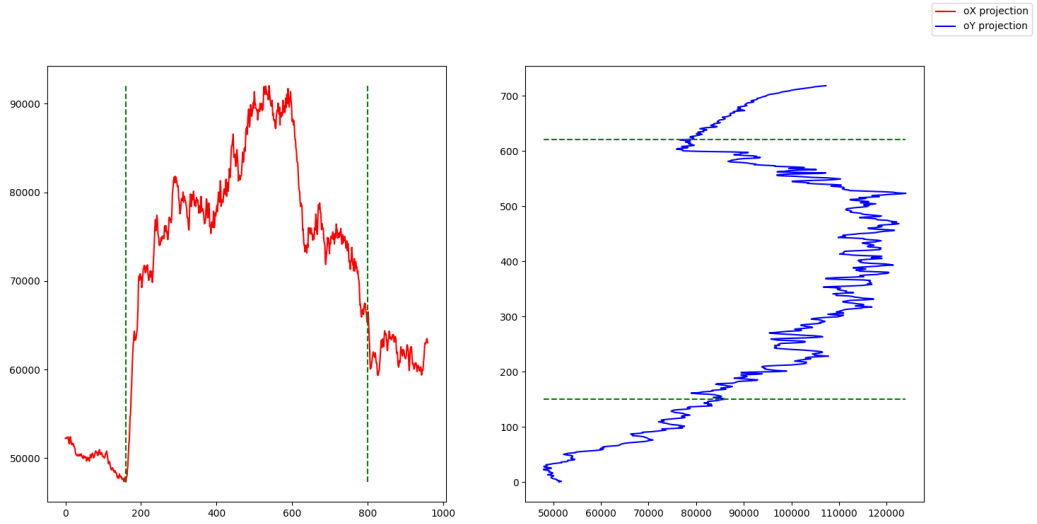

Figure 30: Графики проекций с границами объектов.

Теперь выделим объект на картинке с вычисленными выше границами:

Listing 13. Построение прямоугольника на картинке с помощью OpenCV, Python.

```
start_point = (160, img.shape[0] - 620)  # top left corner
end_point = (800, img.shape[0] - 150)  # bottom right corner
color = (0, 0, 255)  # color in BGR
thickness = 2  # line thickness

# draw a calculated borders of the object on the image
img = cv.rectangle(img, start_point, end_point, color, thickness)
```

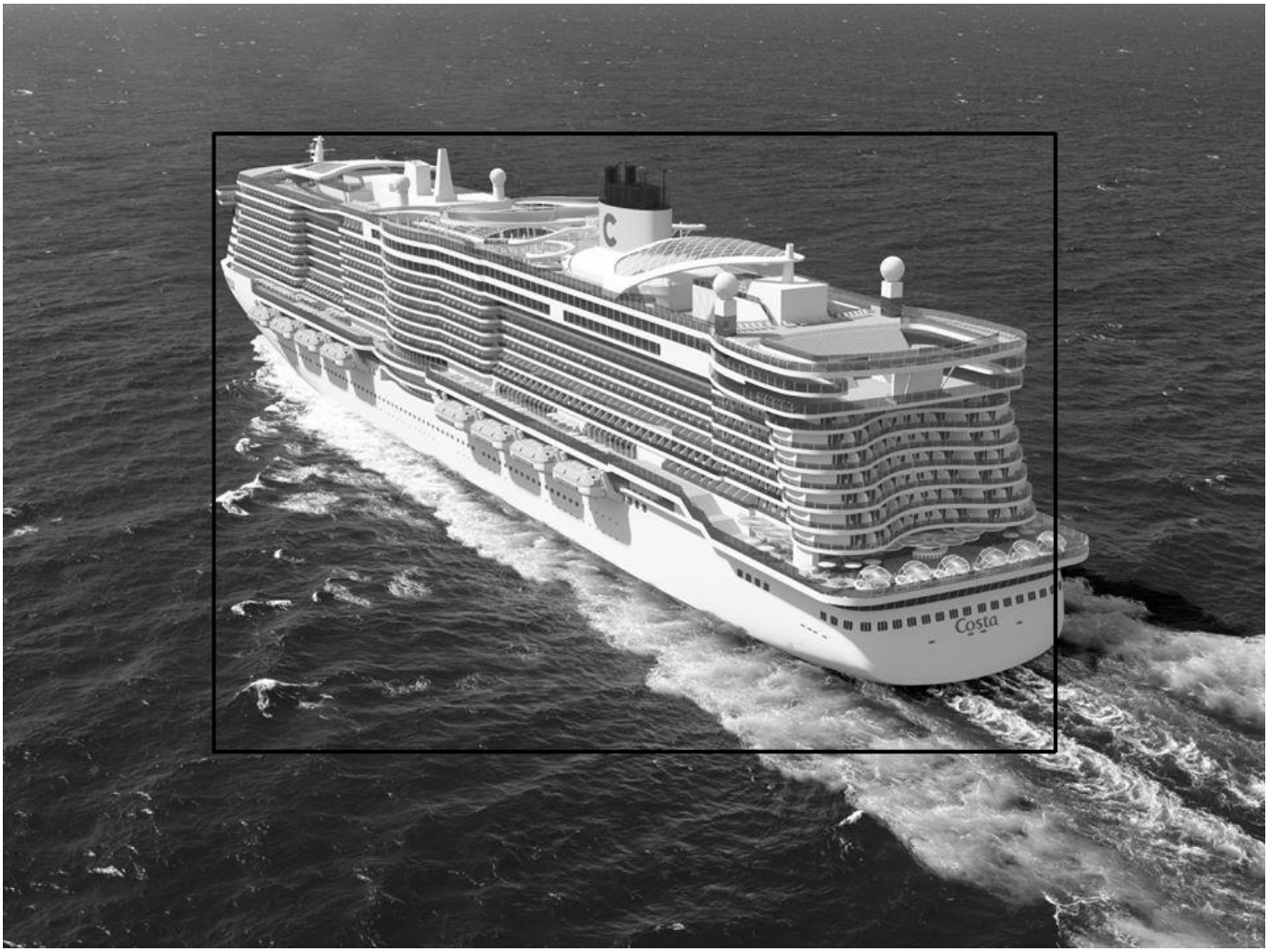

Figure 31: Изображение с отображением прямоугольника, содержащего объект.

Как видно, сегментировать объект получилось не очень хорошо, так как в правом нижнем углу это мешает сделать образовавшийся след от движения судна.

Но в целом, при работе с монотонным фоном и выделяющимся объектом использованием проекций можно определить границы объекта (качество сегментирования будет зависеть от разницы между интенсивностью фона и объекта).

Теперь попробуем определить положение текста на картинке с помощью проекций.

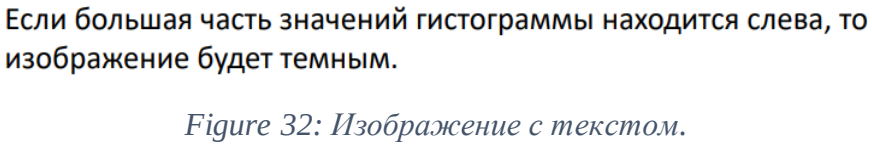

Figure 32: Изображение с текстом.

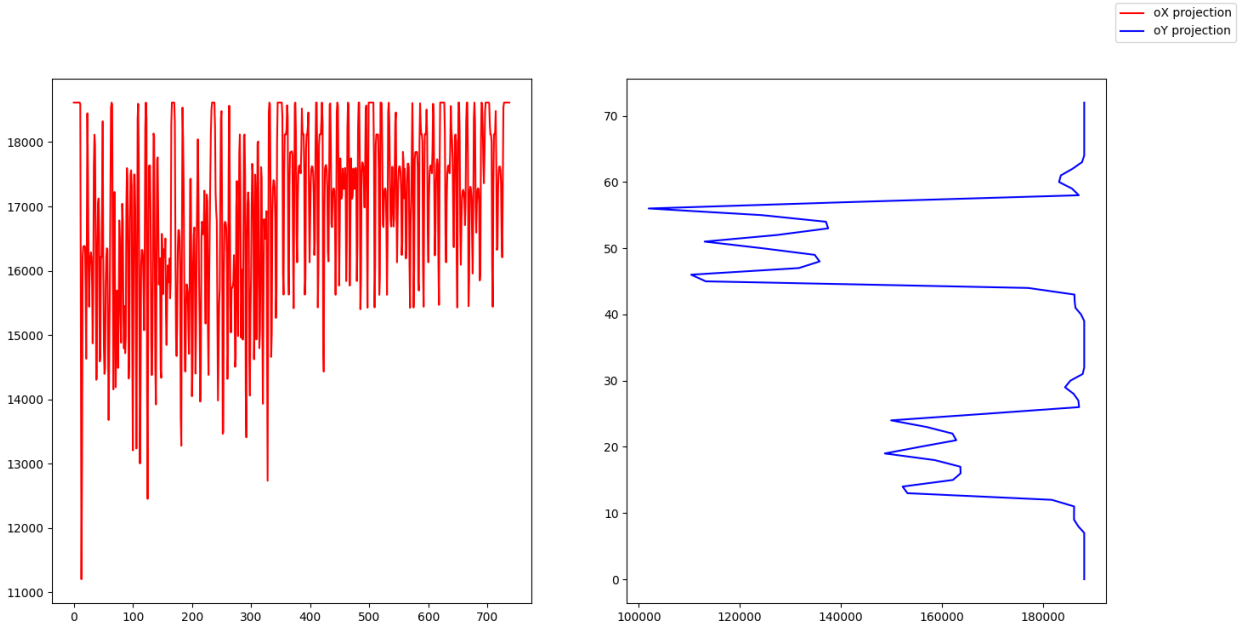

Figure 33: Графики проекций на оси oX и oY.

Анализируя графики выше, мы может определить, что на картинке 2 строки текста и что, первая строка примерно в 2 раза длиннее второй.

Попробуем выделить границы текста, однако теперь из-а того, что мы имеет чистый белый фон, можем выделять границы не примерно по графику, а более точно.

Listing 14. Выделение границ текста на Python.

```
# calculate the oX, oY boundaries
margin = 255  # the value in the array must change by at least
"margin"
# so that we can assume that the void has been replaced by text or
vice versa
```

```
x_boundaries = []
for i in range(x_proj.size - 1):
    if np.abs(x_proj[i].astype(np.int32) - x_proj[i + 1]) > margin:
        x_boundaries.append(i)
y_boundaries = []
for i in range(y_proj.size - 1):
    if np.abs(y_proj[i].astype(np.int32) - y_proj[i + 1]) > margin:
        y_boundaries.append(i)

start_point = (x_boundaries[0], img.shape[0] - y_boundaries[-1])  # top left corner
end_point = (x_boundaries[-1], img.shape[0] - y_boundaries[0])  # bottom right corner
```

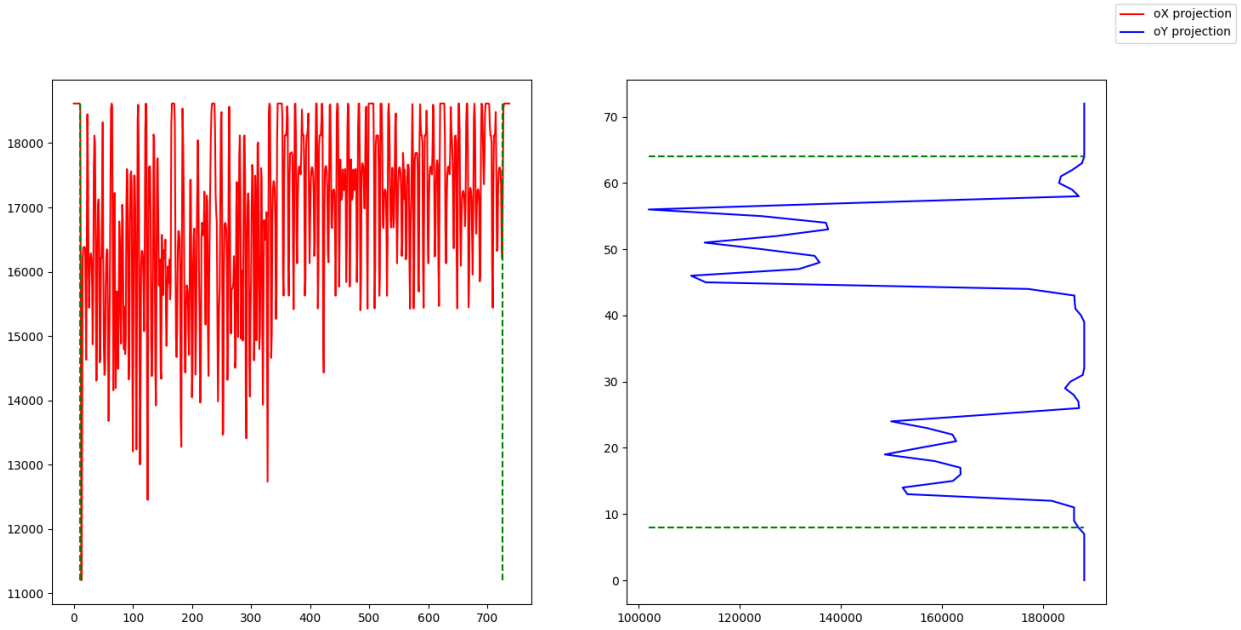

Figure 34: Графики проекций с вычисленными границами текста.

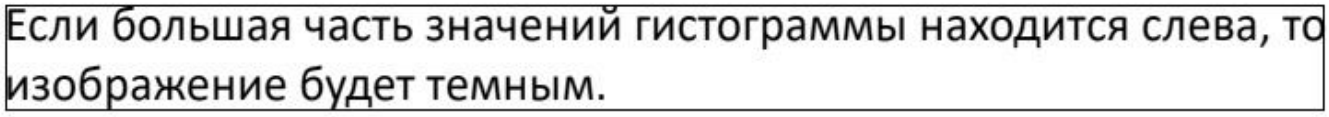

Figure 35: Текст с отображением рамки.

Как видно, удалось точно определить границы текстового блока прямоугольником.

Кстати, можно выделить каждую букву в тексте, используя только проекции на оси.

Выводы

Первая часть работы была посвящена применению различных преобразований для изменения яркостных свойств изображения. Построение и анализ гистограммы изображения – позволяет наглядно увидеть каким образом можно улучшить картинку, выбрав то или иное преобразование. Проводя самые простые арифметические операции с интенсивностями пикселей (сдвиг на константу), можно изменять яркость изображения, однако будут потери. Методы линейного выравнивания и равномерного преобразования полезно использовать, когда необходимо одинаково распределить пиксели по всем интенсивностям (только в методе линейного выравнивая будет задействован весь диапазон интенсивностей, а при равномерном преобразовании – только $[I_{min}, 255]$). Метод растяжения динамического диапазона также увеличивает контрастность, но, кроме этого, за счет присутствия коэффициента нелинейности α интенсивности нового изображения не будут меняться линейно. Методы экспоненциального, по закону Рэлея, по закону степени 2/3 преобразований позволяют нелинейно изменять интенсивности изображения, делая упор на усилие отдельных областей интенсивностей (начало диапазона, середина диапазона и конец диапазона, соответственно). Также для применения вышеописанных фильтров на практике непосредственно в устройствах было бы затруднительно проводить все вычисления на устройстве, вместо этого можно использовать LUT. Это увеличит быстродействие, ведь необходимо просто запомнить массив из 255 значений, LUT можно заменить любое преобразование яркостных свойств.

Во второй части были применялись на практике профили изображения на прямую. Это хороший инструмент, который может давать информацию об интенсивности в интересующем нас направлении (например – более компактное хранение штрихкодов).

В заключительной части были проведены попытки определить геометрическое положение объектов на монотонном фоне с помощью исследования графиков проекций на координатные оси. Это хороший инструмент, однако как я убедился, при большой немонотонности фона определение границ объекта либо становится невозможным, либо велика погрешность.