

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ»

В.И. Костылев, Ю.С. Левицкая

**ОБРАБОТКА И АНАЛИЗ ИЗОБРАЖЕНИЙ
В СРЕДЕ MATLAB**

Учебное пособие

Воронеж
Издательский дом ВГУ
2019

Утверждено научно-методическим советом физического факультета ВГУ
23 января 2019 г., протокол № 1

Рецензент – доктор технических наук, доцент, профессор кафедры
информационных технологий управления В.А. Дурденко

Подготовлено на кафедре электроники физического факультета Воронежского государственного университета.

Рекомендовано магистрам физического факультета Воронежского государственного университета.

Для направления 03.04.03 – Радиофизика

Содержание

Введение.....	4
1. Ввод, обработка и вывод изображений.....	6
2. Блок обработки больших изображений.....	11
3. Создание галереи преобразованных изображений.....	15
4. Разворот изображения для увеличения формы.....	28
5. Нахождение угла поворота изображения и масштаб.....	31
6. Удаление размытия изображения с помощью регуляризованного фильтра.....	35
7. Размытие изображения для удаления тонких линий.....	44
8. Удаление нечётких изображений с помощью фильтра Винера.....	46
9. Улучшение мультиспектральных цветовых композитных изображений.....	52
10. Исправление неравномерное освещение и анализ объектов переднего плана..	62
11. Вычисление статистики для больших изображений.....	69
12. Улучшение изображения с низким уровнем освещенности.....	80
13. Гранулометрия снежинок.....	85
14. Измерение областей в оттенках серого.....	90
15. Использование фазовой корреляции в качестве шага предварительной обработки при регистрации изображения.....	94
16. Исследование срезов/проекций из 3-мерного набора данных МРТ.....	99
17. Обрезание изображений с использованием алгоритма Люси-Ричардсона....	108
18. Обрезание изображений с использованием алгоритма слепой деконволюции.	121
19. Измерение угла пересечения линий.....	130
20. Измерение радиуса рулона ленты.....	135
21. Идентификация круглых объектов.....	137
22. Нахождение длины движущегося маятника.....	143
23. Обнаружение автомобилей в видеоролике.....	150
Заключение.....	157
Литература.....	158

Введение

Цифровая обработка изображений является одним из приоритетных направлений прикладной радиофизики. Это объясняется тем, что изображения используются в качестве средства получения визуальной информации в системах наблюдения, технического зрения, видеотелефонии, телевидения, автономных интеллектуальных системах, телемедицине и др. Поэтому методы обработки визуальной информации, обеспечивающие повышение визуального качества восприятия изображений, сжатие данных для хранения и передачи по каналам связи, а также анализ, распознавание и интерпретацию зрительных образов для принятия решения и управления поведением автономных информационных систем играют все более важную роль.

Любая из процедур обработки и анализа изображений содержит в своей структуре этап предварительной обработки, включающий сглаживание, фильтрацию шумов, повышение четкости и контрастности. Кроме того, предварительная обработка изображений включает в себя коррекцию нелинейности датчика, яркости и контраста изображения, устранение геометрических искажений, подчеркивание интересующих объектов относительно фона и т.п. Часто, на данном этапе осуществляется коррекция возмущений в изображении, обусловленных расфокусировкой оптики, размытостью изображения в результате движения объекта, погрешностями датчика, либо шумами, добавленными при передаче изображений по каналам связи.

В данном учебном пособии приведены универсальные методы первичной обработки изображений, предназначенные для решения различных практических задач. Показано, как считывать изображение в рабочую область компьютера, настраивать контрастность изображения, записывать отредактированное изображение в файл, как выполнить

обнаружение края изображения путем разделения изображения на блоки, как расширить изображение, как использовать регуляризованную деконволюцию для обработки изображений, как размыть двоичное изображение, как использовать деконволюцию Винера для размытых изображений. Показана базовая структура изображений и методы улучшения данных для изображений с мультиспектральными данными, как измерять параметры объектов в полутоновом изображении, как использовать фазовую корреляцию в качестве предварительного шага автоматической регистрации изображения, как использовать алгоритм Люси-Ричардсона для обработки изображений, как использовать скрытую деконволюцию. Имеются примеры геометрических преобразований, а также показано, как использовать Image Processing ToolboxTM для визуализации и анализа видео или последовательностей изображений.

Экспериментальная часть работы выполнена с применением пакета программирования Matlab.

1. Основной ввод изображений, обработка и вывод

В этом пункте показано, как считывать изображение в рабочую область, настраивать контрастность изображения, а затем записывать отредактированное изображение в файл.

Шаг 1: чтение и отображение изображения.

Загрузите изображение в рабочую область, используя команду `imread`.

В этом примере считывается один из образцов изображений, включенных в набор инструментов, изображение молодой девушки в файле с именем `pout.tif` и хранится в массиве с именем `I`. Команда `imread` извлекает информацию о том, что форматом графического файла является формат Tagged Image File Format (TIFF).

```
I = imread('pout.tif');
```

Отобразите изображение, используя функцию `imshow`. Вы также можете просмотреть изображение в приложении Image Viewer. Функция `imtool` открывает приложение Image Viewer, которое представляет собой интегрированную среду для отображения изображений и выполнения некоторых общих задач обработки изображений. Приложение Image Viewer предоставляет все возможности отображения изображений `imshow`, но также предоставляет доступ к нескольким другим инструментам для навигации и изучения изображений, таких как полосы прокрутки, инструмент пиксельной области, инструмент «Информация об изображении» и инструмент «Коррекция контрастности».

```
imshow(I)
```



Рисунок 1. – Вывод изображения функцией imshow

Шаг 2: проверьте, как изображение появляется в рабочей области. Проверьте, как функция imread сохраняет данные изображения в рабочей области, используя команду whos. Вы также можете проверить переменную в браузере Workspace. Функция imread возвращает данные изображения в переменной I, которая представляет собой массив элементов размером 291 на 240 из данных uint8.

```
whos I
  Name      Size            Bytes  Class    Attributes
  I         291x240        69840  uint8
```

Шаг 3: улучшение контрастности изображения.

Просмотрите распределение интенсивности пикселей изображения. Изображение pout.tif представляет собой несколько низкоконтрастное изображение. Чтобы увидеть распределение интенсивностей изображения, создайте гистограмму, вызвав функцию imhist. (вызовите imhist с помощью команды figure, чтобы гистограмма не перезаписывала отображение изображения I в текущем окне рисунка.) Обратите внимание, как гистограмма показывает, что диапазон интенсивности изображения довольно узкий. Диапазон не охватывает потенциальный диапазон [0, 255],

и отсутствует высокое и низкое значения, которые могут привести к хорошему контрасту.

```
figure  
imhist(I)
```

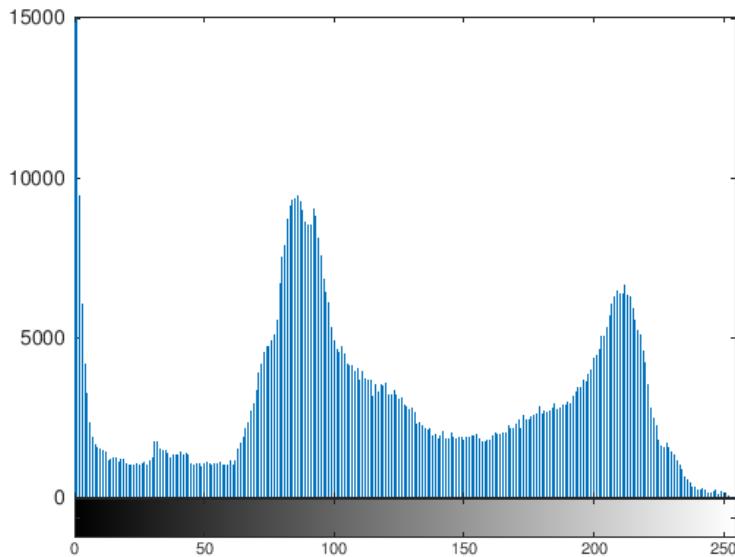


Рисунок 2. – Гистограмма изображения

Улучшите контрастность изображения, используя функцию histeq. Гистограмма выравнивает значения интенсивности во всем диапазоне изображения. Отображение изображения. (Набор инструментов включает в себя несколько других функций, которые выполняют настройку контрастности, включая imadjust и adapthisteq, и интерактивные инструменты, такие как инструмент «Контрастность контраста», доступный в Image Viewer.)

```
I2 = histeq(I);  
figure  
imshow(I2)
```



Рисунок 3. – Изображение с улучшенной контрастностью,
используя функцию histeq

Вызовите функцию imhist еще раз, чтобы создать гистограмму выравниваемого изображения I2. Если вы сравните две гистограммы, вы увидите, что гистограмма I2 более распространена по всему диапазону, чем гистограмма I.

```
figure  
imhist(I2)
```

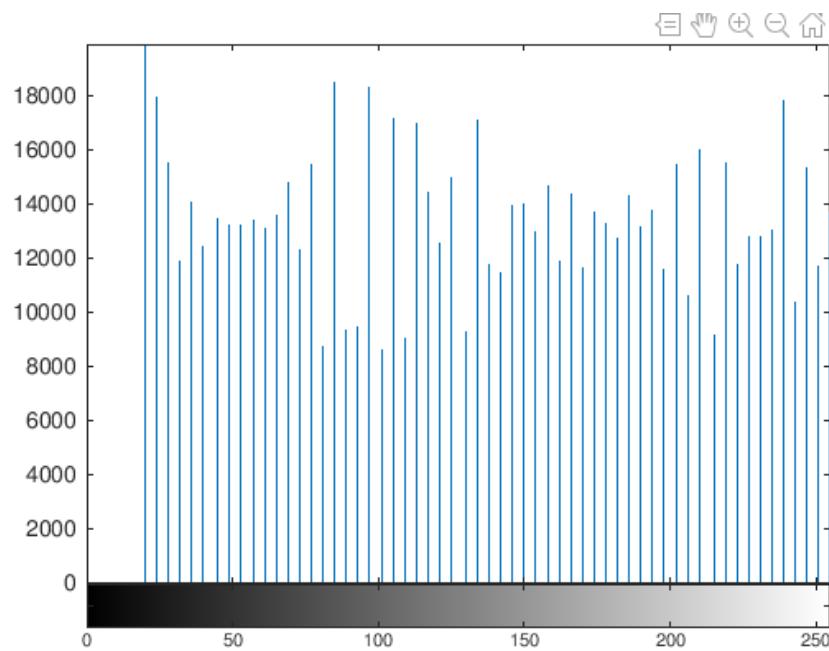


Рисунок 4. – Гистограмма контрасного изображения

Шаг 4: запишите скорректированное изображение в файл диска.

Запишите отредактированное изображение I2 в файл диска, используя функцию `imwrite`. Этот пример включает расширение имени файла `.png` в имени файла, поэтому функция `imwrite` записывает изображение в файл в формате Portable Network Graphics (PNG), но вы можете указать другие форматы.

```
imwrite (I2, 'pout2.png');
```

Шаг 5: проверьте содержимое нового файла. Посмотрите, что `imwrite` написал в файл диска, используя функцию `imfinfo`. Функция `imfinfo` возвращает информацию об изображении в файле, такую как его формат, размер, ширина и высота.

```
imfinfo('pout2.png')
```

2. Блок обработки больших изображений

В этом примере показано, как выполнить обнаружение края изображения формата TIFF путем разделения изображения на блоки. При работе с большими изображениями, обычные методы обработки изображений не всегда применимы. Изображения могут быть либо слишком большим, чтобы загрузить в память, или же они могут быть загружены в память, но будут слишком большими для обработки.

Чтобы избежать этих проблем, можно обрабатывать большие изображения постепенно: чтение, обработка и наконец запись результаты обратно на диск. Функция `blockproc` поможет вам с этим процессом. С помощью `blockproc`, укажите изображение, размер блока и дескриптор функции. `blockproc` делит входное изображения на блоки заданного размера, обрабатывает их с помощью дескриптора функции по одному блоку за раз, а затем собирает результаты вывода изображения. `blockproc` возвращает выходные данные в память или в новый файл на диске.

Рассмотрим результаты выполнения обнаружения края без блока обработки. В этом примере используется небольшое изображение, `cameraman.tif`, для иллюстрации алгоритма, но блок обработки зачастую является более полезным для больших изображений.

```
file_name = 'cameraman.tif';
I = imread(file_name);
normal_edges = edge(I,'canny');
imshow(I)
title('Original Image')
```



Рисунок 5. – Оригинальное изображение

```
figure  
imshow(normal_edges)  
title('Conventional Edge Detection')
```

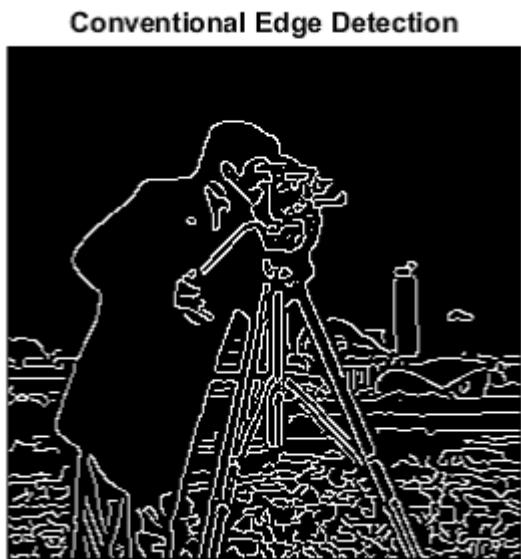


Рисунок 6. – Обычное обнаружение края

Теперь рассмотрим ту же задачу с помощью блока обработки. Функция `blockproc` имеет встроенную поддержку для TIFF изображений, поэтому не нужно полностью читать файл в память с помощью `imread`. Вместо этого вызывается функция, используя имя файла в качестве

входных данных. `blockproc` читает один блок за раз, делая этот процесс идеальным для больших изображений.

При работе с большими изображениями часто используется параметр «`Destination`», чтобы указать файл, в который `blockproc` будет записывать выходное изображение. Однако в этом примере, результаты возвращаются в память.

В этом примере используется размер блока `[50 50]`. В общем, выбор больших размеров блока дает лучшую производительность для `blockproc`. Соответствующий размер блоков варьируются исходя из имеющихся ресурсов компьютера.

```
% You can use an anonymous function to define the function handle.  
The  
% function is passed a structure as input, a "block struct", with  
several  
% fields containing the block data as well as other relevant  
information.  
% The function should return the processed block data.  
edgeFun = @(block_struct) edge(block_struct.data,'canny');  
block_size = [50 50];  
block_edges = blockproc(file_name,block_size,edgeFun);  
figure  
imshow(block_edges)  
title('Block Processing - Simplest Syntax')
```

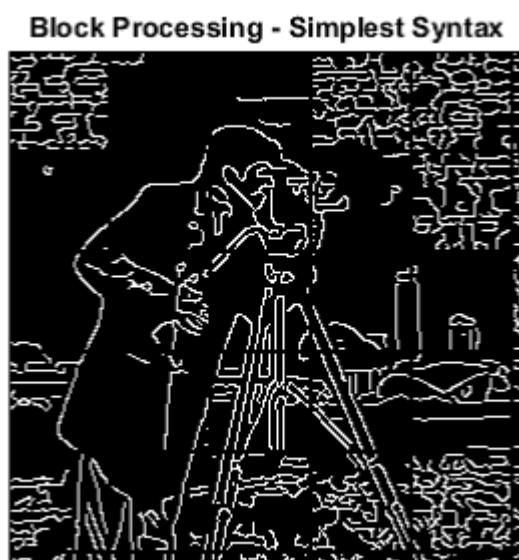


Рисунок 7. – Блок обработки – самый простой синтаксис

Обратите внимание на особенности блока обработки. Определение того, является ли пиксель пограничным пикселием не требует информации от соседних пикселей. Это означает, что каждый блок не может обрабатываться полностью отдельно от окружающих пикселей. Чтобы исправить это, используется параметр `blockproc('BorderSize')` для указания вертикальной и горизонтальной границы вокруг каждого блока. Необходимые размеры границ выбираются в зависимости от выполняемой задачи.

```
border_size = [10 10];
block_edges =
blockproc(file_name,block_size,edgeFun,'BorderSize',border_size);
figure
imshow(block_edges)
title('Block Processing - Block Borders')
```

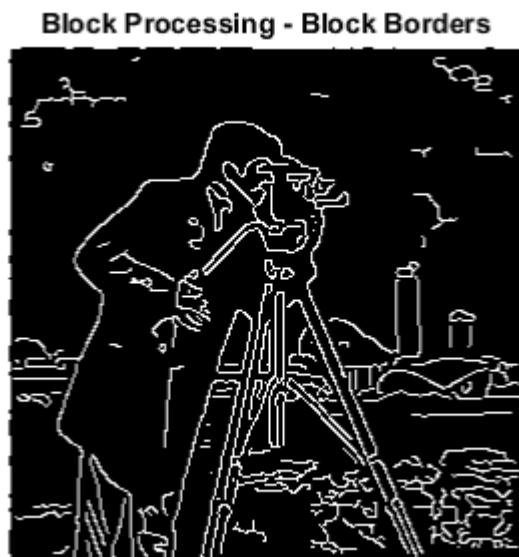


Рисунок 8. – Блок обработки – границы блока

3. Создание галереи преобразованных изображений

Двумерное геометрическое преобразование является отображением, которое сопоставляет каждую точку в евклидовой плоскости с другой точкой в евклидовой плоскости. В этих примерах геометрическое преобразование определяется правилом, которое указывает, как сопоставить точку с декартовыми координатами (x, y) в другую точку с декартовыми координатами (u, v) . Шаблон шахматной доски полезен при визуализации сетки координат в плоскости входного изображения и типа искажений, возникающих при каждом преобразовании.

Изображение 1. Создать шахматную доску.

Шахматная доска создает изображение с прямоугольными плитами и четырьмя уникальными углами, что позволяет легко увидеть, как изображение шахматной доски искажается геометрическими преобразованиями. После запуска этого примера один раз попробуйте изменить изображение I на ваш любимый образ.

```
sqsize = 60;
I = checkerboard(sqsize,4,4);
nrows = size(I,1);
ncols = size(I,2);
fill = 0.3;
imshow(I)
title('Original')
```

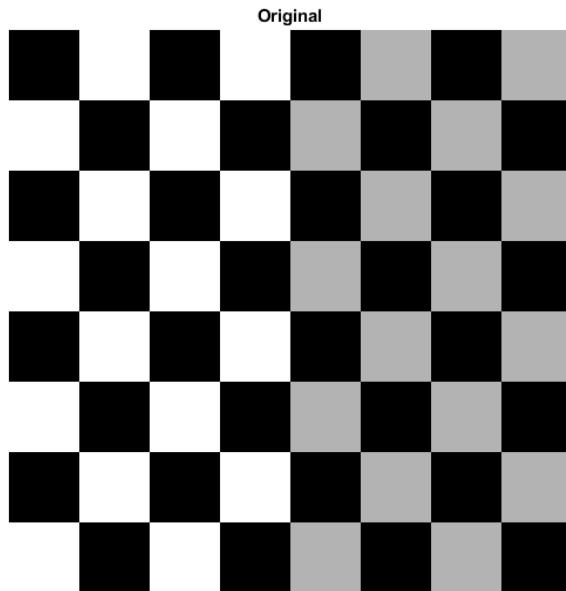


Рисунок 9. – Оригинальное изображение

Изображение 2. Применить невосприимчивое сходство к шахматной доске.

Неотражающие преобразования подобия могут включать в себя поворот, масштабирование и перевод. Формы и углы сохраняются. Параллельные линии остаются параллельными. Прямые линии остаются прямыми.

Для нерефлексивного подобия,

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} T$$

где Т - матрица 3 на 3, которая зависит от 4 параметров.

```
% Try varying these 4 parameters.
scale = 1.2;          % scale factor
angle = 40*pi/180;    % rotation angle
tx = 0;                % x translation
ty = 0;                % y translation
sc = scale*cos(angle);
ss = scale*sin(angle);
T = [ sc -ss 0;
      ss sc 0;
      tx ty 1];
```

Поскольку нерефлексивные сходства являются подмножеством аффинных преобразований, создайте аффинный объект 2d, используя:

```
t_nonsim = affine2d(T);
I_nonreflective_similarity = imwarp(I,t_nonsim,'FillValues',fill);
imshow(I_nonreflective_similarity);
title('Nonreflective Similarity')
```

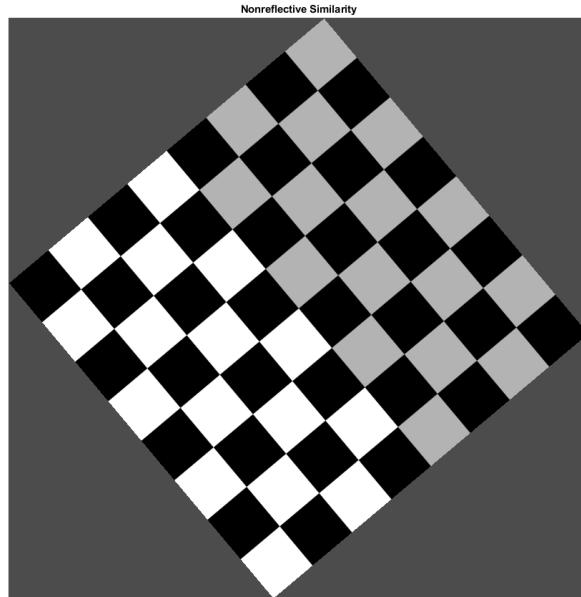


Рисунок 10. – Неотражающее сходство изображения

Если вы измените либо tx , либо ty на ненулевое значение, вы заметите, что оно не влияет на выходное изображение. Если вы хотите увидеть координаты, которые соответствуют вашему преобразованию, включая перевод, включая в себя пространственную справочную информацию:

```
[I_nonreflective_similarity,RI] =
imwarp(I,t_nonsim,'FillValues',fill);
imshow(I_nonreflective_similarity,RI)
axis on
title('Nonreflective Similarity (Spatially Referenced)')
```

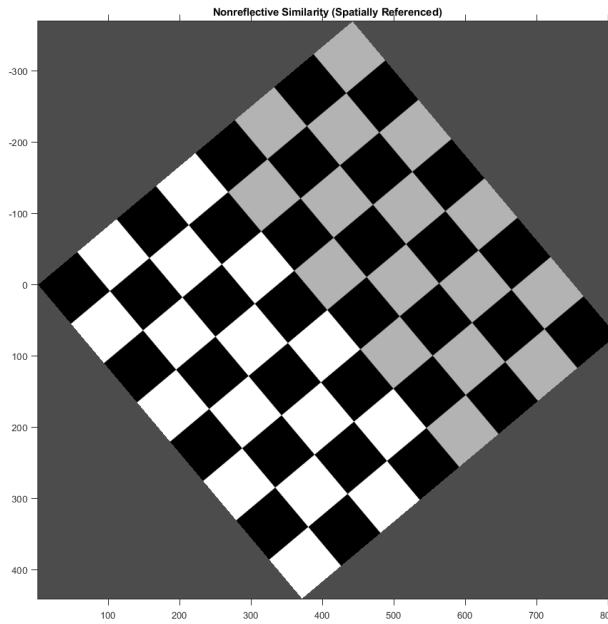


Рисунок 11. – Неотражающее сходство изображения
(пространственная привязка)

Обратите внимание, что передача выходного пространственного ссылочного объекта RI из imwarp показывает перевод. Чтобы указать, какую часть выходного изображения вы хотите увидеть, используйте пару «имя-значение» OutputView в функции imwarp.

Изображение 3. Применить сходство с шахматной доской.

В преобразовании подобия аналогичные треугольники сопоставляются с аналогичными треугольниками. Неотражающие преобразования подобия являются подмножеством преобразований подобия.

Для подобия уравнение такое же, как для нерефлексивного подобия:

$$[uv] = [xy]T$$

где Т - матрица 3 на 3, которая зависит от 4 параметров плюс необязательное отражение.

```
% Try varying these parameters.  
scale = 1.5; % scale factor  
angle = 10*pi/180; % rotation angle  
tx = 0; % x translation  
ty = 0; % y translation
```

```

a = -1; % -1 -> reflection, 1 -> no reflection
sc = scale*cos(angle);
ss = scale*sin(angle);
T = [ sc -ss 0;
      a*ss a*sc 0;
      tx ty 1];

```

Поскольку сходства являются подмножеством аффинных преобразований, создайте объект `affin2d`, используя:

```
t_sim = affine2d(T);
```

Как и в приведенном выше примере перевода, извлеките выходной объект пространственного реферирования `RI` из функции `imwarp` и передайте `RI`, чтобы отобразить отражение.

```

[I_similarity,RI] = imwarp(I,t_sim,'FillValues',fill);
imshow(I_similarity,RI)
axis on
title('Similarity')

```

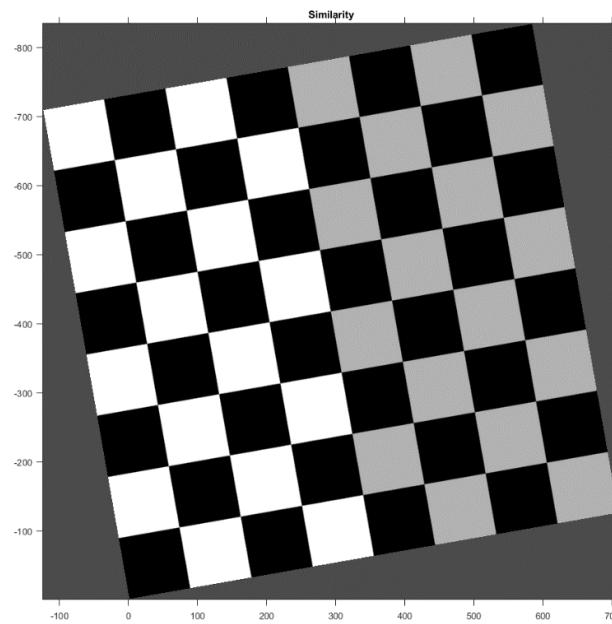


Рисунок 12. – Сходство с шахматной доской

Изображение 4. Применить аффинную трансформацию к шахматной доске.

При аффинном преобразовании размеры x и y могут быть масштабированы или сдвинуты независимо и может быть сдвиг, отражение

и / или поворот. Параллельные линии остаются параллельными. Прямые линии остаются прямыми. Сходства являются подмножеством аффинных преобразований. Для аффинного преобразования уравнение такое же, как для подобия и нерефлексивного подобия:

$$[uv] = [xy]T$$

где Т - матрица 3 на 3, где все шесть элементов первого и второго столбцов могут быть разными. Третий столбец должен быть $[0; 0; 1]$.

% Try varying the definition of T.

```
T = [1 0.3 0;
      1 1 0;
      0 0 1];
t_aff = affine2d(T);
I_affine = imwarp(I,t_aff,'FillValues',fill);
imshow(I_affine)
title('Affine')
```

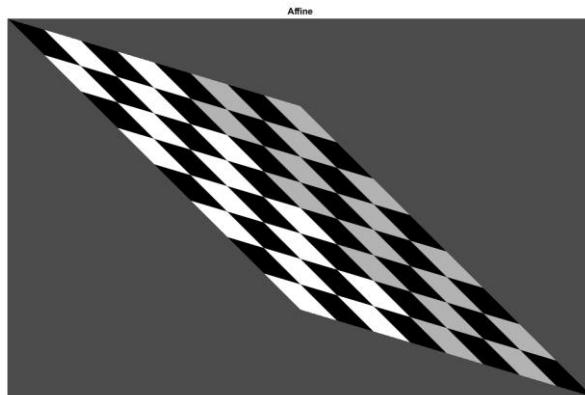


Рисунок 13. - Аффинное преобразование

Изображение 5. Применить проективное преобразование к шахматной доске.

В проективном преобразовании четырехугольники отображаются в четырехугольник. Прямые линии остаются прямыми, но параллельные линии необязательно остаются параллельными. Аффинные преобразования являются подмножеством проективных преобразований.

Для проективного преобразования

$$\begin{bmatrix} up & vp & wp \end{bmatrix} = \begin{bmatrix} x & y & w \end{bmatrix} T,$$

$$u = \frac{up}{wp},$$

$$v = \frac{vp}{wp},$$

где Т - матрица 3 на 3, где все девять элементов могут быть разными.

$$T = \begin{bmatrix} A & D & G \\ B & E & H \\ C & F & I \end{bmatrix}.$$

Вышеупомянутое матричное уравнение эквивалентно этим двум выражениям:

$$u = \frac{Ax + By + C}{Gx + Hy + I},$$

$$v = \frac{Dx + Ey + F}{Gx + Hy + I}.$$

Попробуйте изменить любой из девяти элементов Т.

```
T = [1 0 0.002;
      1 1 0.0002;
      0 0 1];
t_proj = projective2d(T);
I_projective = imwarp(I,t_proj, 'FillValues', fill);
imshow(I_projective)
title('Projective')
```

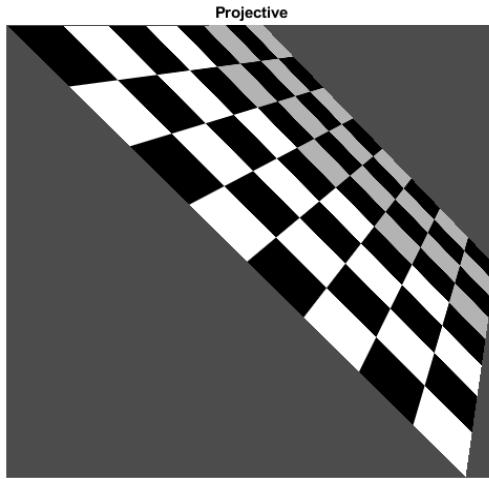


Рисунок 14. – Проективное преобразование

Изображение 6. Применить кусочно-линейную трансформацию на шахматную доску.

В кусочно-линейном преобразовании аффинные преобразования применяются отдельно к областям изображения. В этом примере верхние левые, верхние правые и нижние левые точки шахматной доски остаются неизменными, но треугольная область в правом нижнем углу изображения растягивается так, что нижний правый угол преобразованного изображения равен 50% вправо и на 20% ниже исходной.

```
movingPoints = [0 0; 0 nrows; ncols 0; ncols nrows];  
fixedPoints = [0 0; 0 nrows; ncols 0; ncols*1.5 nrows*1.2];  
t_piecewise_linear = fitgeotrans(movingPoints,fixedPoints,'pw1');  
I_piecewise_linear = imwarp(I,t_piecewise_linear,'FillValues',fill);  
imshow(I_piecewise_linear)  
title('Piecewise Linear')
```

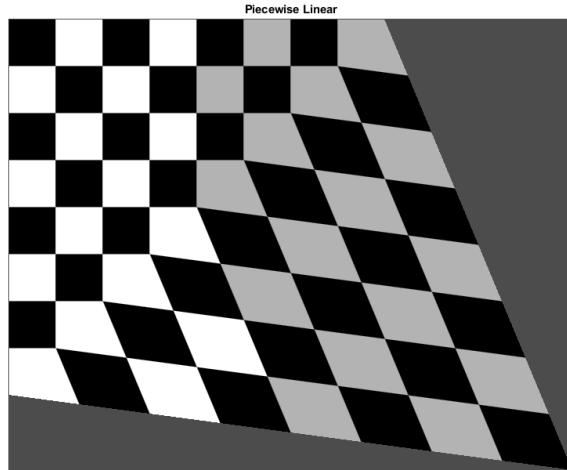


Рисунок 15. - Кусочно-линейная трансформация

Изображение 7. Применить синусоидальную трансформацию к шахматной доске.

В этом примере и в следующих двух примерах показано, как можно создать явное сопоставление для сопоставления каждой точки в регулярной сетке (x_i, y_i) с другой точкой (u_i, v_i) . Это сопоставление хранится в объекте `geometricTransform2d`, который используется `imwarp` для преобразования изображения. В этом синусоидальном преобразовании x-координата каждого пикселя не изменяется. Координата y каждой строки пикселей смещается вверх или вниз по синусоидальной схеме.

```
a = ncols/12; % Try varying the amplitude of the sinusoid
ifcn = @(xy) [xy(:,1), xy(:,2) + a*sin(2*pi*xy(:,1)/nrows)];
tform = geometricTransform2d(ifcn);
I_sinusoid = imwarp(I,tform,'FillValues',fill);
imshow(I_sinusoid);
title('Sinusoid')
```

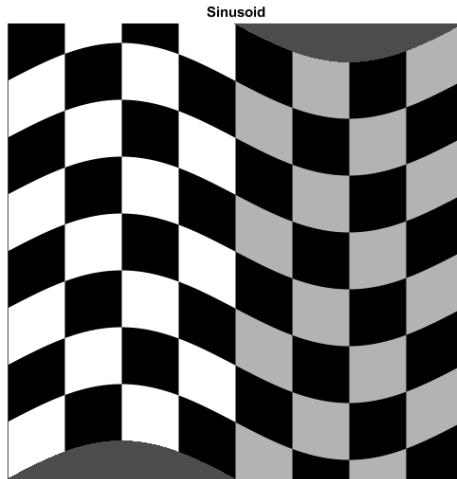


Рисунок 16. - Синусоидальная трансформация

Изображение 8. Применить трансформацию ствола к шахматной доске.

Искажение ствола возмущает изображение радиально наружу от его центра. Искажение больше от центра, что приводит к выпуклым сторонам.

Сначала определите функцию, которая отображает индексы пикселей на расстояние от центра. Используйте функцию `meshgrid` для создания массивов x-координат и у-координат каждого пикселя, причем начало координат находится в верхнем левом углу изображения.

```
[xi,yi] = meshgrid(1:ncols,1:nrows);
```

Переместите начало координат в центр изображения. Затем преобразуйте декартовы координаты x- и у-координат в координаты цилиндрического угла (`theta`) и радиуса (`r`), используя функцию `cart2pol`. `r` изменяется линейно по мере увеличения расстояния от центрального пикселя.

```
xt = xi - ncols/2;  
yt = yi - nrows/2;  
[theta,r] = cart2pol(xt,yt);
```

Определите амплитуду а кубического члена. Этот параметр настраивается. Затем добавьте кубический член в r так, чтобы r изменялся нелинейно с расстоянием от центрального пикселя.

```
a = 1; % Try varying the amplitude of the cubic term.  
rmax = max(r(:));  
s1 = r + r.^3*(a/rmax.^2);
```

Преобразуем обратно в декартову систему координат. Переместите начало координат в верхний правый угол изображения.

```
[ut,vt] = pol2cart(theta,s1);  
ui = ut + ncols/2;  
vi = vt + nrows/2;
```

Храните сопоставление между (x_i, y_i) и (u_i, v_i) в объекте geometricTransform2d. Используйте imwarp для преобразования изображения в соответствии с отображением пикселей.

```
ifcn = @(c) [ui(:) vi(:)];  
tform = geometricTransform2d(ifcn);  
I_barrel = imwarp(I,tform,'FillValues',fill);  
imshow(I_barrel)  
title('Barrel')
```

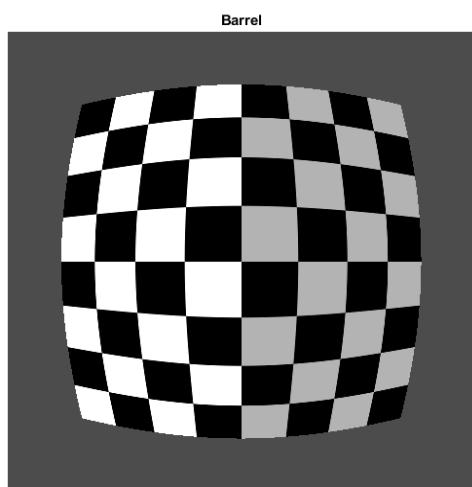


Рисунок 17. - Трансформация ствола

Изображение 9. Применить переходное смягчение к шахматной доске.

Пин-смягчающее искажение является обратным искажением ствола, поскольку кубический член имеет отрицательную амплитуду. Искажение все еще находится дальше от центра, но искажение появляется как вогнутые стороны.

Вы можете начать с тех же значений тета и r, что и для преобразования ствола. Определите другую амплитуду, b, кубического члена. Этот параметр настраивается. Затем вычтите кубический член в r так, чтобы r изменялся нелинейно с расстоянием от центрального пикселя.

```
b = 0.4; % Try varying the amplitude of the cubic term.  
s = r - r.^3*(b/rmax.^2);
```

Преобразуем обратно в декартову систему координат. Переместите начало координат в верхний правый угол изображения.

```
[ut,vt] = pol2cart(theta,s);  
ui = ut + ncols/2;  
vi = vt + nrows/2;
```

Храните сопоставление между (x_i , y_i) и (u_i , v_i) в объекте geometricTransform2d. Используйте imwarp для преобразования изображения в соответствии с отображением пикселей.

```
ifcn = @(c) [ui(:) vi(:)];  
tform = geometricTransform2d(ifcn);  
I_pin = imwarp(I,tform,'FillValues',fill);  
imshow(I_pin)  
title('Pin Cushion')
```

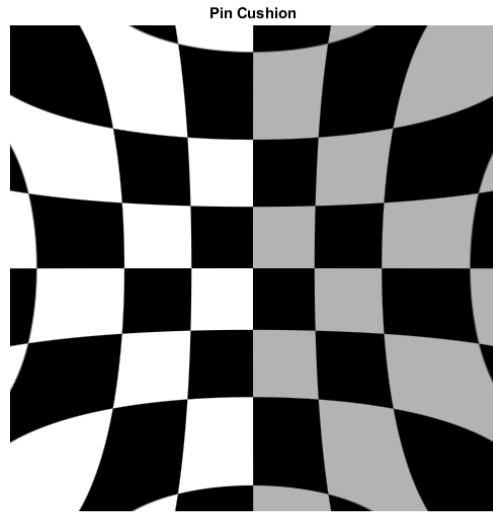


Рисунок 18. - Переходное смягчение

4. Разворот изображения для увеличения формы

В этом примере показано, как расширить изображение с помощью функции `imdilate`. Операция морфологической дилатации расширяет или утолщает объекты переднего плана в изображении.

Создадим простой образ бинарного образца, содержащий один объект переднего плана: одна квадратная область в середине изображения.

```
BW = zeros(9,10);  
BW(4:6,4:7) = 1
```

```
BW = 9 x 10
```

```
0 0 0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0 0 0  
0 0 0 1 1 1 1 0 0 0  
0 0 0 1 1 1 1 0 0 0  
0 0 0 1 1 1 1 0 0 0  
0 0 0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0 0 0
```

```
imshow(imresize(BW,40, 'nearest'))
```

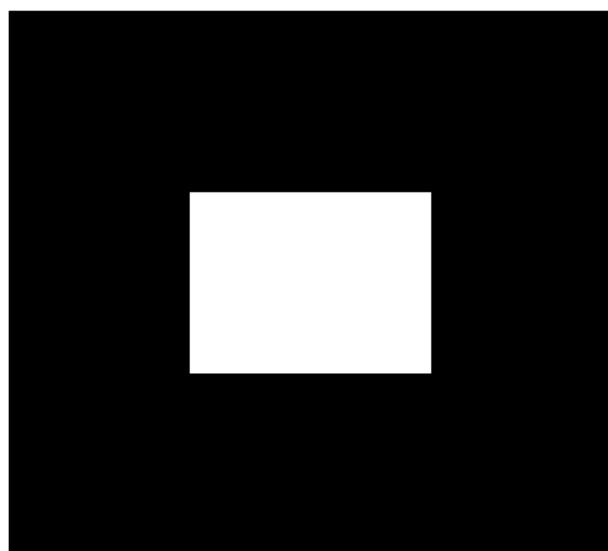


Рисунок 19. – Бинарный объект

Создадим элемент структурирования, используя imdilate. Чтобы расширить геометрический объект, создаем элемент структурирования, который имеет ту же форму, что и объект.

```
SE = strel('square',3)
```

```
SE =
```

```
strel is a square shaped structuring element with properties:
```

```
Neighborhood: [3x3 logical]
```

```
Dimensionality: 2
```

Развернем изображение, передав входное изображение и структурирующий элемент для imdilate. Обратите внимание, как расширение добавляет ранг 1 ко всем сторонам объекта переднего плана.

```
BW2 = imdilate(BW,SE)
```

```
BW2 = 9 × 10
```

```
0 0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0 0  
0 0 1 1 1 1 1 1 0 0  
0 0 1 1 1 1 1 1 0 0  
0 0 1 1 1 1 1 1 0 0  
0 0 1 1 1 1 1 1 0 0  
0 0 1 1 1 1 1 1 0 0  
0 0 1 1 1 1 1 1 0 0  
0 0 0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0 0 0
```

```
imshow(imresize(BW2,40,'nearest'))
```

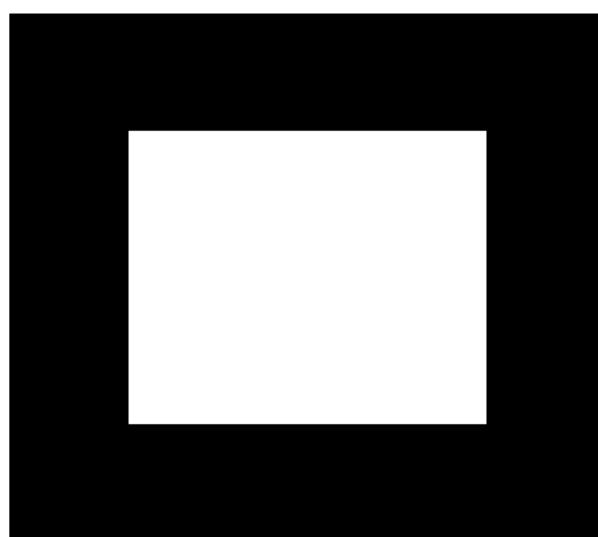


Рисунок 20. – Развернутый увеличенный объект

Для сравнения создадим структурирующий элемент другой формы.
Развернем исходное изображение, используя новый элемент структурирования.

```
SE2 = strel('diamond',1);
BW3 = imdilate(BW,SE2);
imshow(imresize(BW3,40,'nearest'))
```

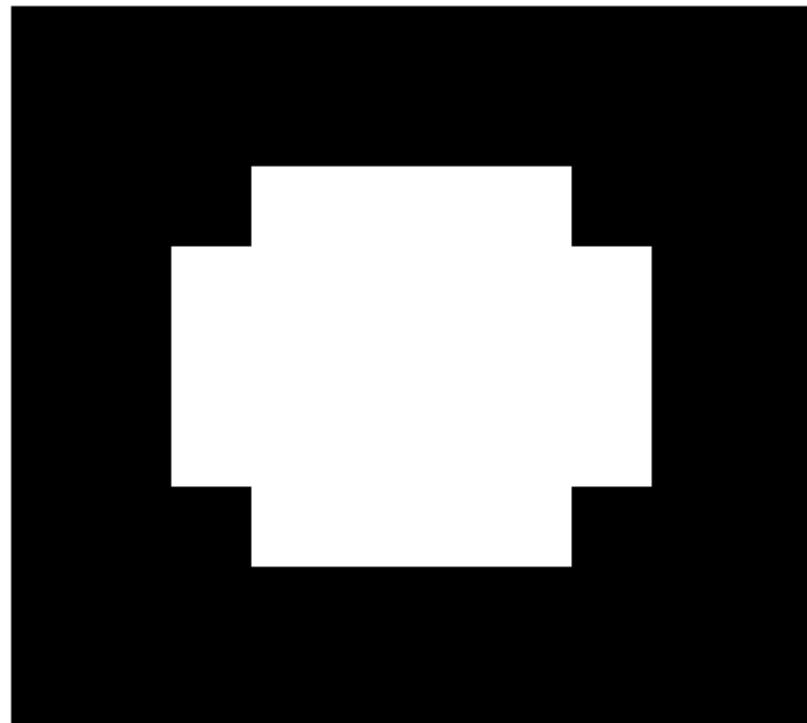


Рисунок 21. – Изображение с новым элементом структурирования

5. Нахождение поворота изображения и масштаб

В этом примере показано, как выровнять или показать два изображения, которые отличаются поворотом и изменением масштаба. Вы можете использовать fitgeotrans для поиска угла поворота и коэффициента масштабирования после ручного подбора соответствующих точек. Затем вы можете преобразовать искаженное изображение, чтобы восстановить исходное изображение.

Шаг 1: загрузка изображения.

Загрузим изображение в рабочее поле.

```
original = imread('C:\Users\vanov\OneDrive\Desktop\cat.jpg');
imshow(original);
text(size(original,2),size(original,1)+15, ...
    'Image courtesy of Massachusetts Institute of Technology', ...
    'FontSize',7,'HorizontalAlignment','right');
```



Рисунок 22. – Оригинальное изображение

Шаг 2: изменение размера и поворота изображения.

```
scale = 0.7;
distorted = imresize(original,scale); % Try varying the scale factor.
theta = 30;
distorted = imrotate(distorted,theta); % Try varying the angle,
theta.
figure, imshow(distorted);
```



Рисунок 23.- Изображение с измененным размером и поворотом

Шаг 3: выбор контрольных точек.

Используйте инструмент выбора контрольных точек, чтобы выбрать не менее двух пар контрольных точек.

```
movingPoints = [151.52 164.79; 131.40 79.04];  
fixedPoints = [135.26 200.15; 170.30 79.30];
```

Вы можете запустить остальную часть примера с этими выбранными точками, но попытайтесь выбрать свои собственные точки, чтобы увидеть, как результаты меняются.

```
cpselect(distorted,original,movingPoints,fixedPoints);
```

Сохраните контрольные точки, выбрав меню «Файл», затем «Сохранить точки в рабочей области». Сохраняйте точки, переписывая переменные movingPoints и fixedPoints.

Шаг 4: оценка преобразований.

Установите преобразование без отражения в ваших контрольных точках.

```
Tform = fitgeotrans(movingPoints,fixedPoints,'nonreflectivesimilarity');
```

После того, как вы выполнили шаги 5 и 6, повторите шаги с 4 по 6, то попробуйте использовать affine вместо NonreflectiveSimilarity. Что

происходит? Являются ли результаты такими же хорошими, как и NonreflectiveSimilarity?

Шаг 5: разрешение шкалы и угла.

Геометрическое преобразование tform содержит матрицу преобразования в tform.T. Поскольку вы знаете, что преобразование включает только поворот и масштабирование, математика относительно проста для восстановления масштаба и угла.

```
Let sc = s*cos(theta)
Let ss = s*sin(theta)
Then, Tinv = invert(tform), and Tinv.T = [sc -ss 0;
ss sc 0;
tx ty 1]
where tx and ty are x and y translations, respectively.
tformInv = invert(tform);
Tinv = tformInv.T;
ss = Tinv(2,1);
sc = Tinv(1,1);
scale_recovered = sqrt(ss*ss + sc*sc)
scale_recovered = 0.7000
theta_recovered = atan2(ss,sc)*180/pi
theta_recovered = 29.3741
```

Восстановленные значения scale_recovered и theta_recovered должны соответствовать значениям, заданным на шаге 2: изменение размера и поворот изображения.

Шаг 6: восстановление оригинального изображения.

Восстановите исходное изображение, преобразуя искаженное, повернутое и масштабированное изображение, используя геометрическую трансформацию tform и то, что вы знаете о пространственной привязке оригинала. Параметр Name / Value OutputView используется для указания разрешения и размера сетки повторно выбранного полученного изображения.

```
Roriginal = imref2d(size(original));
recovered = imwarp(distorted,tform,'OutputView',Roriginal);
```

Сравните восстановленные оригиналы, глядя на них бок о бок в монтаже.

```
figure, imshowpair(original,recovered,'montage')
```



Рисунок 24. – Оригинальное (слева)
и восстановленное (справа) изображение

Восстановленное (правое) изображение не соответствует исходному (левому) изображению из-за искажения и восстановления. В частности, уменьшение изображения приводит к потере информации. Артефакты вокруг краев связаны с ограниченной точностью преобразования. Если бы вы выбрали больше точек, в шаге 3: выберите контрольные точки, то преобразование будет более точным.

6. Удаление размытия изображения с помощью регуляризованного фильтра

В этом примере показано, как использовать регуляризованную деконволюцию для обработки изображений. Регуляризованная деконволюция может эффективно использоваться, когда на восстановленное изображение (например, гладкость) применяются ограничения, и известна ограниченная информация об аддитивных шумах. Размытое и шумное изображение восстанавливается с помощью ограниченного алгоритма восстановления наименьших квадратов, который использует регуляризованный фильтр.

Шаг 1: загрузка изображения.

Пример читает RGB-изображение и отображает его 256-на-256-на-3. Функция deconvreg может обрабатывать массивы любого измерения.

```
I = imread('C:\Users\vanov\OneDrive\Desktop\water.jpg');
I = I(125+(1:256),1:256,:);
f1 = figure;
imshow(I);
figure(f1);
title('Original Image');
text(size(I,2),size(I,1)+15, ...
'Image courtesy of Alan Partin, Johns Hopkins University', ...
'FontSize',7,'HorizontalAlignment','right');
```

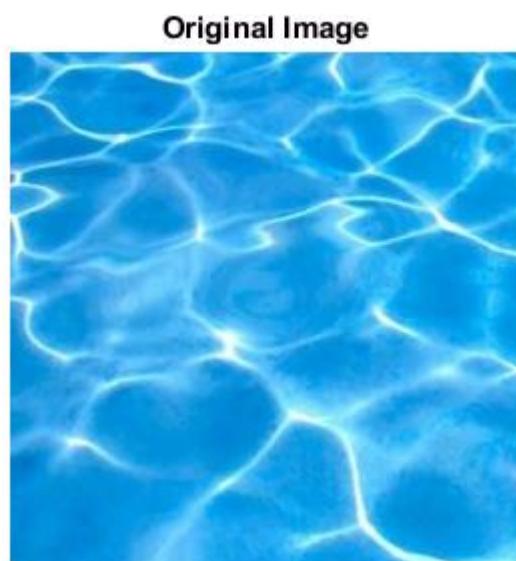


Рисунок 25. – Оригинальное изображение

Шаг 2: наложение размытия и шума.

Смоделируем изображение в реальной жизни, которое может быть размыто (например, из-за движения камеры или отсутствия фокуса) и зашумлено (например, из-за случайных помех). Пример моделирует размытие путем свертывания гауссовского фильтра с истинным изображением (с использованием imfilter). Гауссовский фильтр представляет собой функцию распределения по точкам, PSF.

```
PSF = fspecial('gaussian',11,5);
Blurred = imfilter(I,PSF,'conv');
f2 = figure;
imshow(Blurred);
figure(f2);
title('Blurred');
```

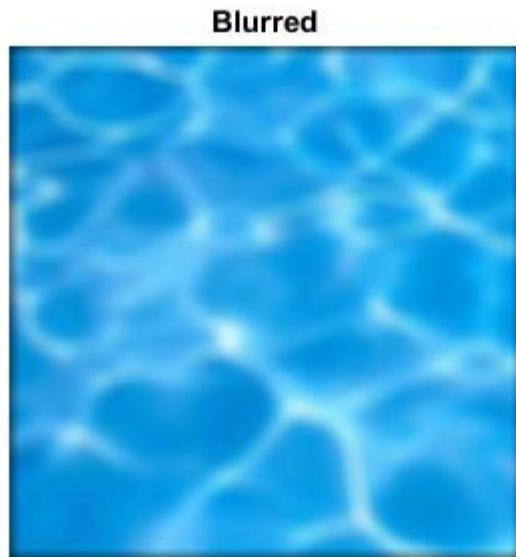


Рисунок 26. – Размытое изображение

Мы имитируем шум, добавляя гауссовский шум дисперсии V к размытому изображению (используя imnoise).

```
V = .02;
BlurredNoisy = imnoise(Blurred,'gaussian',0,V);
f3 = figure;
imshow(BlurredNoisy);
figure(f3);
title('Blurred & Noisy');
```

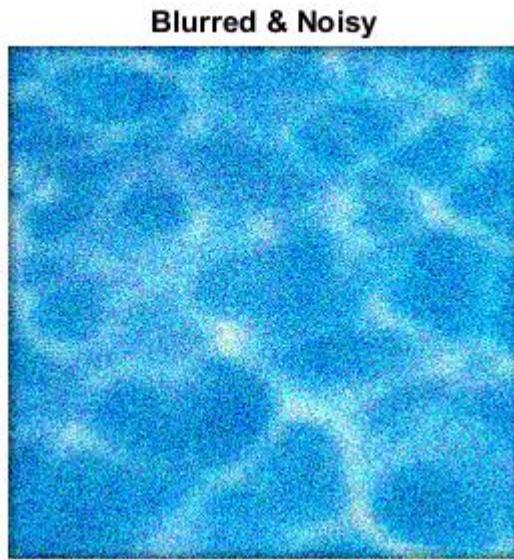


Рисунок 27. – Размытое и зашумленное изображение

Шаг 3: восстановление размытого и зашумлённого изображения.

Восстановите размытое и зашумлённое изображение, подавая мощность шума NP, в качестве третьего входного параметра. Чтобы проиллюстрировать, насколько чувствителен алгоритм к значению мощности шума NP, пример выполняет три реставрации.

Первое восстановление, reg1, использует истинный NP. Обратите внимание, что пример выводит здесь два параметра. Первое возвращаемое значение, reg1, является восстановленным изображением. Второе возвращаемое значение, LAGRA, является скалярным множителем Лагранжа, на котором сходится deconvreg. Это значение используется далее в этом примере.

```
NP = V*numel(I); % noise power
[reg1, LAGRA] = deconvreg(BlurredNoisy,PSF,NP);
f4 = figure;
imshow(reg1);
figure(f4);
title('Restored with NP');
```



Рисунок 28. – Восстановление изображение с шумом

Второе восстановление, reg2, использует слегка завышенную мощность шума, что приводит к плохому разрешению.

```
reg2 = deconvreg(BlurredNoisy,PSF,NP*1.3);
f5 = figure;
imshow(reg2);
figure(f5);
title('Restored with larger NP');
```

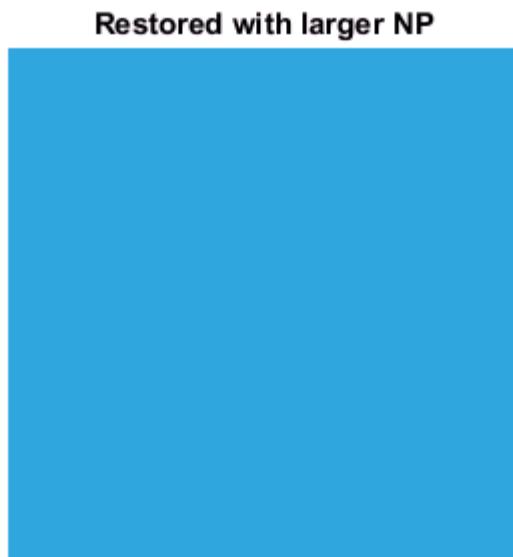


Рисунок 29. - Восстановление изображения с завышенной мощностью шума

Третье восстановление, reg3, получает оценочное значение NP. Это приводит к подавляющему усилинию шума и «звонам» из границ изображения.

```
reg3 = deconvreg(BlurredNoisy,PSF,NP/1.3);
f6 = figure;
imshow(reg3);
figure(f6);
title('Restored with smaller NP');
```

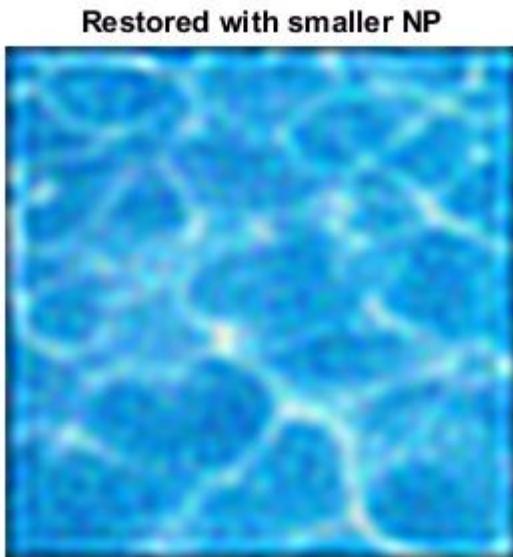


Рисунок 30. – Восстановление изображения с учетом мощности шума

Шаг 4: уменьшение усиления шума и звона.

Уменьшите усиление шума и звона вдоль границы изображения, вызвав функцию edgetaper до деконволюции. Обратите внимание, что восстановление изображения становится менее чувствительным к параметру мощности шума. Используйте значение мощности шума NP в предыдущем примере.

```
Edged = edgetaper(BlurredNoisy,PSF);
reg4 = deconvreg(Edged,PSF,NP/1.3);
f7 = figure;
imshow(reg4);
figure(f7);
title('Edgetaper effect');
```

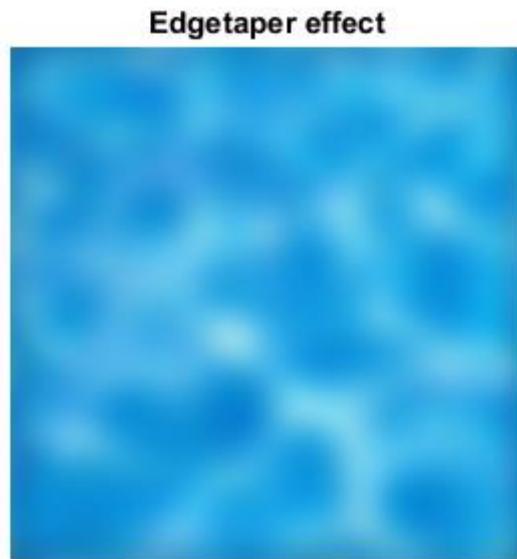


Рисунок 31. – Краевой эффект

Шаг 5: использование множителя Лагранжа.

Восстановите размытое и шумное изображение, считая, что оптимальное решение уже найдено и задан соответствующий множитель Лагранжа LAGRA. В этом случае любое значение, переданное для мощности шума, NP, игнорируется.

Чтобы проиллюстрировать, насколько чувствителен алгоритм к значению LAGRA, пример выполняет три реставрации. Первое восстановление (reg5) использует выход LAGRA из более раннего решения (выход LAGRA из первого решения на шаге 3).

```
reg5 = deconvreg(Edged,PSF,[],LAGRA);
f8 = figure;
imshow(reg5);
figure(f8);
title('Restored with LAGRA');
```



Рисунок 32. – Первое восстановление изображения с использованием множителя Лангранжа

Второе восстановление (reg6) использует 100 * LAGRA, что увеличивает значимость ограничения. По умолчанию это приводит к чрезмерному сглаживанию изображения.

```
reg6 = deconvreg(Edged,PSF,[],LAGRA*100);  
f9 = figure;  
imshow(reg6);  
figure(f9);  
title('Restored with large LAGRA');
```



Рисунок 33. – Второе восстановление изображения с использованием множителя Лангранжа

Третья реставрация использует LAGRA / 100, которая ослабляет ограничение (требование гладкости, установленное для изображения). Оно усиливает шум и в конечном итоге приводит к чистой обратной фильтрации для LAGRA = 0.

```
reg7 = deconvreg(Edged,PSF,[],LAGRA/100);  
f10 = figure;  
imshow(reg7);  
figure(f10);  
title('Restored with small LAGRA');
```



Рисунок 34. – Третье восстановление изображения с использованием множителя Лангранжа

Шаг 6: использование других ограничений.

Восстановите размытое и шумное изображение, используя другое ограничение (REGOP) в поиске оптимального решения. Вместо ограничения гладкости изображения (по умолчанию REGOP является лапласианским), ограничивайте гладкость изображения только в одном измерении (1-D лапласиан).

```
REGOP = [1 -2 1];  
reg8 = deconvreg(BlurredNoisy,PSF,[],LAGRA,REGOP);  
f11 = figure;  
imshow(reg8);  
figure(f11);
```

```
title('Constrained by 1D Laplacian');
```

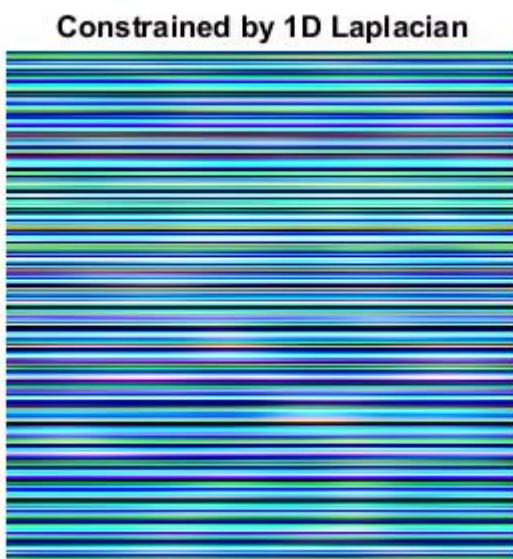


Рисунок 35. – Восстановление изображения при ограничении гладкости

7. Размытие изображения для удаления тонких линий

В этом примере показано, как размыть двоичное изображение с помощью функции imerode.

Считаем двоичное изображение в рабочей области.

```
BW1 = imread('circbw.tif');
figure
imshow(BW1)
```

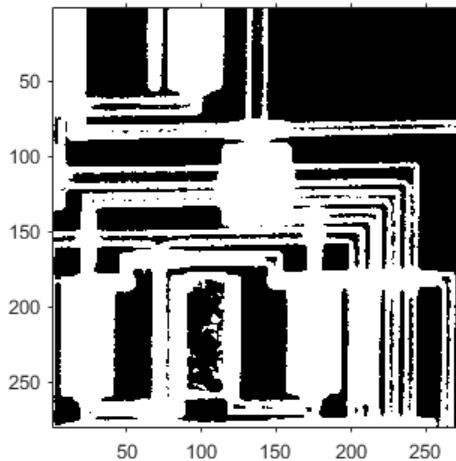


Рисунок 36. – Двоичное изображение

Создадим элемент структурирования. Следующий код создает объект элемента диагональной структуры.

```
SE = strel('arbitrary',eye(7))
SE =
strel is a arbitrary shaped structuring element with properties:

    Neighborhood: [7x7 logical]
    Dimensionality: 2
```

Изменим изображение, указав входное изображение и структурирующий элемент в качестве аргументов функции imerode.

```
BW2 = imerode(BW1,SE);
```

Отобразим исходное изображение и эродированное изображение. Особое внимание следует обратить на диагональные полосы в правой части

выходного изображения. Это связано с формой структурирующего элемента.

```
figure  
imshow(BW2)
```

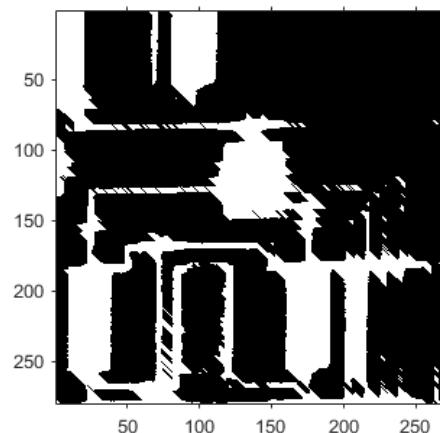


Рисунок 37. - Исходное и эродированное изображение

8. Удаление нечётких изображений с помощью фильтра Винера

В этом примере показано, как использовать деконволюцию Винера для размытых изображений. Деконволюция Винера полезна, когда известны или оценены функции точечного распространения и уровень шума.

Загрузка изображения

```
I = im2double(imread('C:\Users\vanov\OneDrive\Desktop\cat.jpg'));
imshow(I);
title('Original Image (courtesy of MIT)');
```



Рисунок 38. – Название рисунка

Имитация размытия движения

Имитируйте размытое изображение, которое можно получить от движения камеры. Создайте функцию распределения по точкам PSF, соответствующую линейному движению по 21 пикслю (LEN=21) под углом 11 градусов (THETA=11). Чтобы имитировать размытие, сверните фильтр с изображением с помощью imfilter

```
LEN = 21;
THETA = 11;
PSF = fspecial ('motion', LEN, THETA);
blurred = imfilter(I, PSF, 'conv', 'circular');
imshow(blurred);
title('Blurred Image');
```



Рисунок 39. – Имитация изображения в движении

Восстановление размытого изображения

Простой синтаксис для deconvwnr является deconvwnr(A, PSF, NSR), где A – размытое изображение, PSF – аппаратная функция, NSR – отношение мощности шума к мощности сигнала. Размытое изображение, полученное на шаге 2, не имеет шума, поэтому NSR имеет значение 0.

```
Wnr = deconvwnr(blurred, PSF, 0);
imshow(wnr1);
title('Restored Image');
```



Рисунок 40. – Восстановленное изображение

Имитация размытия и шума

Теперь добавим шум.

```

noise_mean = 0;
noise_var = 0.0001;
blurred_noisy = imnoise(blurred, 'gaussian', noise_mean, noise_var);
imshow(blurred_noisy)
title('Simulate Blur and Noise')

```



Рисунок 41. – Размытое и зашумленное изображение

Восстановление размытого и зашумлённого изображения: первая попытка

В первой попытке восстановления сообщим deconvwnr, что шума нет. При NS=0 фильтр Винера эквивалентен идеальному обратному фильтру. Идеальный фильтр может быть очень чувствительным к шуму, как показано далее:

```

Wnr = deconvwnr(blurred_noisy, PSF, 0);
Imshow(wnr2)
Title('Restoration of Blurred, Noisy Image - NSR = 0')

```

Restoration of Blurred, Noisy Image - NSR = 0



Рисунок 42. – Восстановление изображения без учета наличия шума

Шум получился настолько сильным, что виден только слабый силуэт человека.

Восстановление размытого и зашумлённого изображения: вторая попытка

Во второй попытке даём оценку отношения мощности шума к мощности сигнала:

```
signal_var = var(I(: ));  
wnr3 = deconvwnr(blurred_noisy, PSF, noise_var / signal_var);  
imshow(wnr3)  
title('Restoration of Blurred, Noisy Image - Estimated NSR');
```

Restoration of Blurred, Noisy Image - Estimated NSR



Рисунок 43. – Восстановление изображения с учетом мощности шума к мощности сигнала

Имитация размытия и 8-битного шума квантования

Даже визуально не заметное количество шума может повлиять на результат. Давайте попробуем сохранить входное изображение в представлении uint8 вместо преобразования его в double.

```
I = imread('C:\Users\vanov\OneDrive\Desktop\cat.jpg');  
class(I)
```

Если передать изображение uint8 в imfilter, то он будет его квантовать, чтобы получить другое изображение uint8.

```
blurred_quantized = imfilter(I, PSF, 'conv', 'circular');  
class(blurred_quantized)  
  
ans =  
'uint8'
```

Restoration of Blurred, Quantized Image - Estimated NSR



Рисунок 44. - Восьми битовое представление исходного изображения

Восстановление размытого квантового изображения: первая попытка

В нашей первой попытке восстановления мы скажем deconvwnr, что шума нет ($NSR = 0$). Когда $NSR = 0$, фильтр Винера эквивалентен идеальному обратному фильтру. Идеальный обратный фильтр может быть чрезвычайно чувствителен к шуму во входном изображении, как показано на следующем изображении:

```

wnr2 = deconvwnr(blurred_noisy, PSF, 0);
imshow(wnr2)
title('Restoration of Blurred, Noisy Image - NSR = 0')
Restoration of blurred, quantized image - NSR = 0

```



Рисунок 45. – Восстановление четкости изображения без учета наличия шума

Восстановление размытого квантового изображения: вторая попытка

Теперь подставляем оценку NSR к deconvwn

```

uniform_quantization_var = (1/256)^2 / 12;
signal_var = var(im2double(I(:)));
wnr5 = deconvwnr(blurred_quantized, PSF, ...
    uniform_quantization_var / signal_var);
imshow(wnr5)
title('Restoration of Blurred, Quantized Image - Estimated NSR');

```

Restoration of Blurred, Quantized Image - Estimated NSR

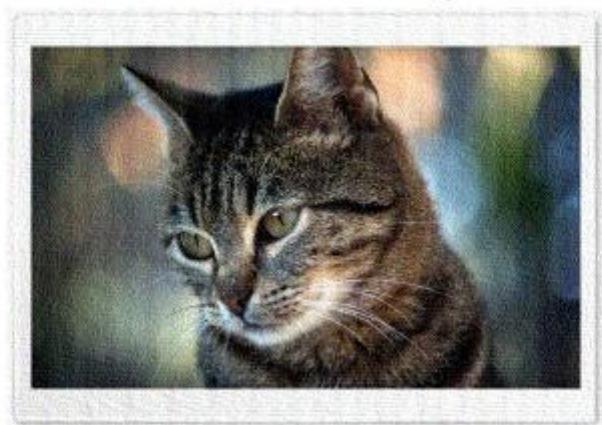


Рисунок 46. – Восстановление изображения с учетом мощности шума к мощности сигнала

9. Улучшение мультиспектральных цветовых композитных изображений

В этом примере показана базовая структура изображений и методы улучшения для изображений с мультиспектральными данными. Часто необходимо увеличить мультиспектральные блеск или отражения чтобы создать изображение подходящее для визуальной интерпритации . В этом примере используются снимки проекта Landsat покрывающие часть Парижа. Семь спектральных диапазонов(слоев) хранятся в одном файле в формате Erdas LAN. Рассматриваемые способы включают:

- чтение многоспектральных данных из файлов Erdas LAN;
- построение цветовых композиций из разных комбинаций слоёв;
- улучшение изображений с контрастным растяжением;
- усиление изображений с растяжением декорреляции;
- использование диаграмм рассеяния.

Шаг 1: получить истинное изображение из мультиспектрального изображения.

Файл paris.lan содержит 7-ми слойное, 512x512, Landsat изображение. За 128-байтовым заголовком следуют значения пикселей, которые чередуются по линии (BIL) в порядке увеличения номера строки. Они хранятся как беззнаковые 8-битные целые числа в порядке возрастания. Прочтите строки 3, 2, и 1 из файла LAN используя функцию multibandread MATLAB®. Эти строки охватывают видимую часть спектра. Они отображают красную, зеленую и синюю плоскости, результатом наложения которых будет правдоподобное изображение. Последний аргумент multibandread указывает строки и их порядок для чтения, чтобы вы могли получить RGB изображение за одно действие.

```
truecolor = multibandread('paris.lan', [512, 512, 7], 'uint8=>uint8',  
128, 'bil', 'ieee-le', {'Band','Direct',[3 2 1]});
```

Истинное изображение отличается слабой контрастностью и несбалансированностью цветов.

```
f1 = figure;
imshow(truecolor);
figure(f1);
title('Truecolor Composite (Un-enhanced)')
text(size(truecolor,2), size(truecolor,1) + 15, ...
    'Image courtesy of Space Imaging, LLC',...
    'FontSize', 7, 'HorizontalAlignment', 'right')
```

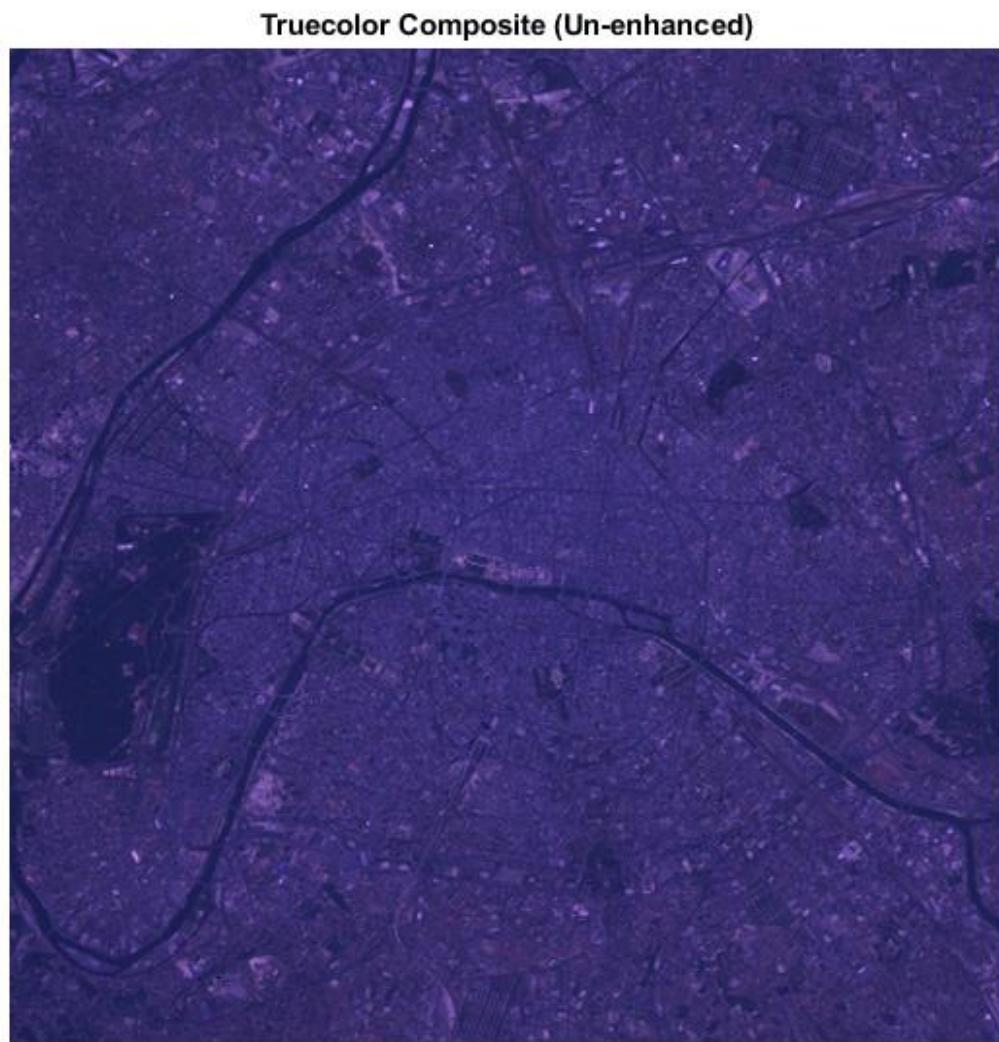


Рисунок 47. – Цветная композиция (без улучшений)

Шаг 2: использование гистограмм для исследования неулучшенного истинного изображения.

Рассматривая гистограмму красного слоя, заметно, что данные сосредоточены в небольшой части доступного динамического диапазона. Это одна из причин, почему истинное изображение выглядит тускло.

```
f2 = figure;
imhist(truecolor(:,:,1))
figure(f2);
title('Histogram of the Red Band (Band 3)')
```

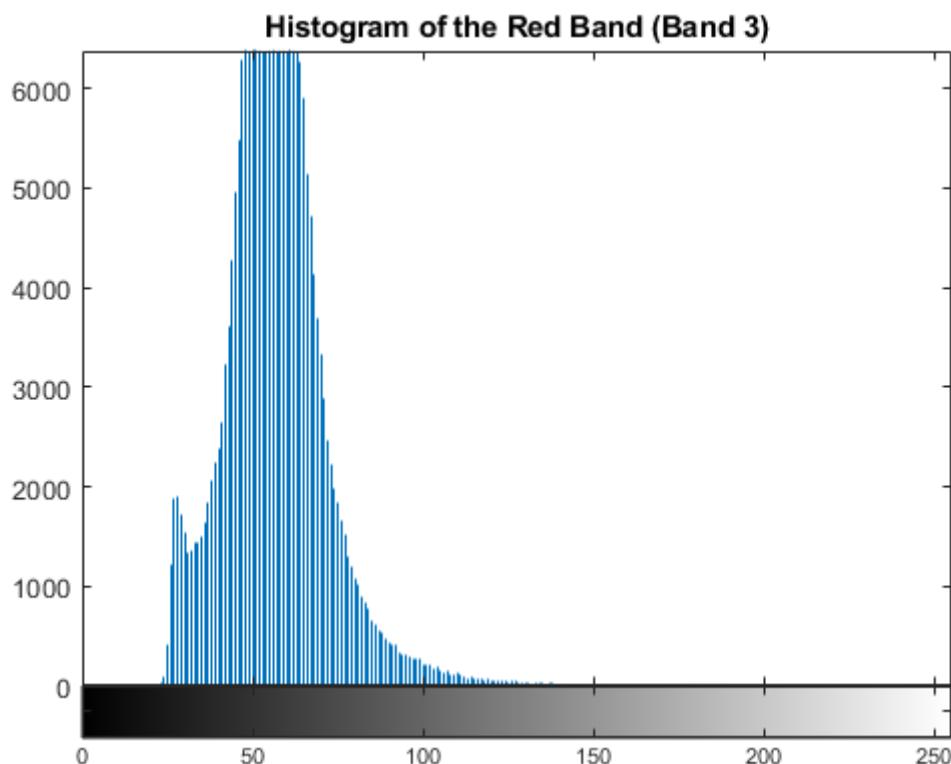


Рисунок 48. – Гистограмма красного слоя

Шаг 3: использование корреляции для исследования неулучшенного истинного изображения.

Другая причина тусклого изображения заключается в том, что видимые слои сильно коррелируют друг с другом. Двух и трех слойные диаграммы рассеяния это хороший способ для оценки степени корреляции между спектральными слоями. Их можно легко построить используя `plot`.

```

r = truecolor(:,:,1);
g = truecolor(:,:,2);
b = truecolor(:,:,3);
f3 = figure;
plot3(r(:),g(:),b(:),'.')
grid('on')
xlabel('Red (Band 3)')
ylabel('Green (Band 2)')
zlabel('Blue (Band 1)')
figure(f3);
title('Scatterplot of the Visible Bands')

```

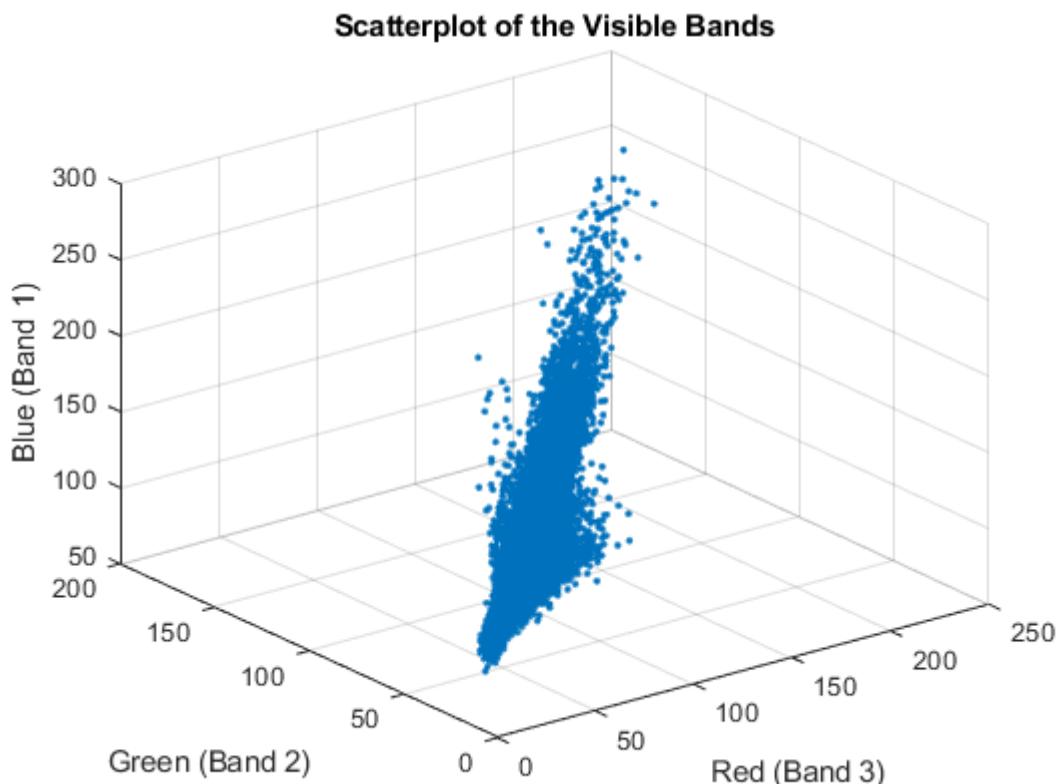


Рисунок 49. – График рассеяния видимых слоев

Явное линейное возрастание диаграммы рассеяния показывает что видимые слои сильно коррелируют. Это помогает объяснить монохроматический вид неулучшенного истинного изображения.

Шаг 4: улучшение истинного изображения контрастным расширением.

При использовании imadjust применительно к линейно контрастному растяжению истинного изображения становится легче распознать особенности поверхности.

```
stretched_truecolor = imadjust(truecolor,stretchlim(truecolor));  
f4 = figure;  
imshow(stretched_truecolor)  
figure(f4);  
title('Truecolor Composite after Contrast Stretch')
```

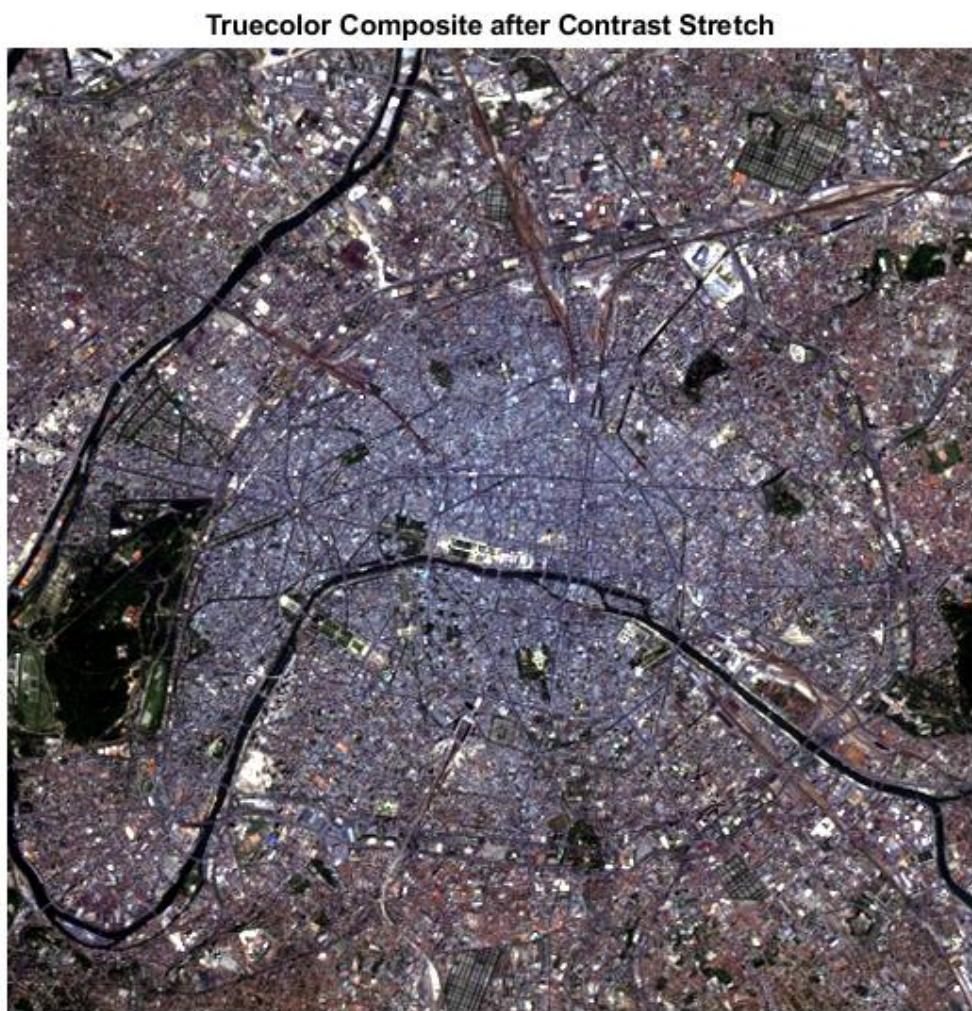


Рисунок 50. – Цветное изображение после усиления контраста

Шаг 5: проверка гистограммы после расширения контраста.

Гистограмма красного слоя после расширения контраста показывает что данные стали распределены по значительно большему динамическому диапазону.

```
f5 = figure;
imhist(stretched_truecolor(:,:,1))
figure(f5);
title('Histogram of Red Band (Band 3) after Contrast Stretch')
```

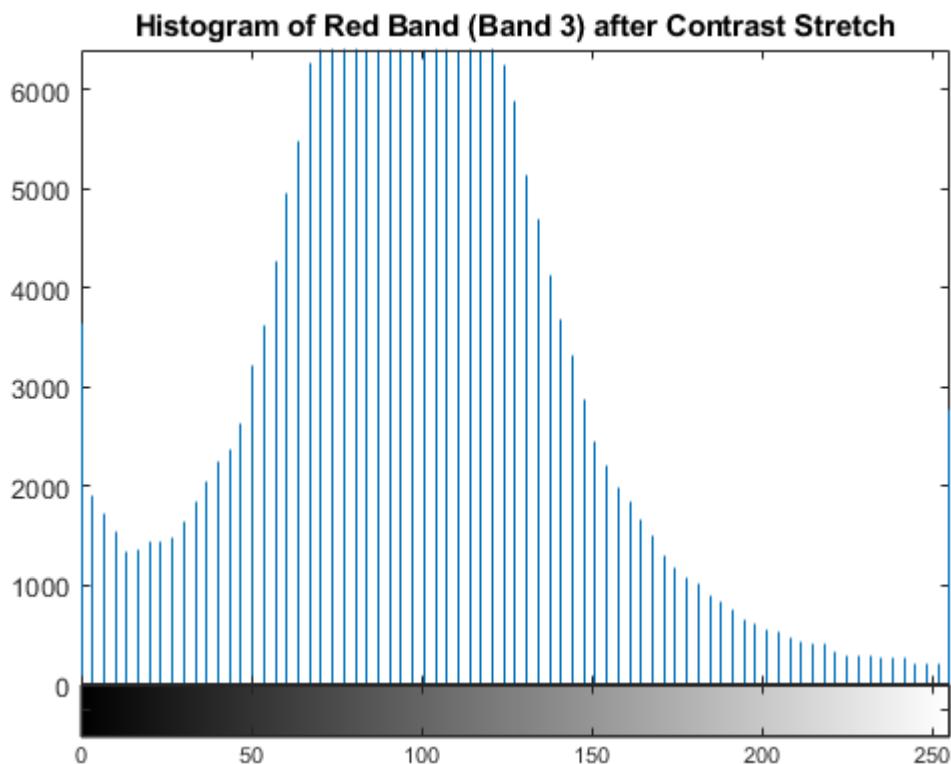


Рисунок 51. – Гистограмма красного слоя после усиления контраста

Шаг 6: улучшение истинного изображения с помощью декорреляции.

Ещё один способ улучшить изображение – улучшение с помощью декорреляции. Для этого используется decorrstretch. Дополнительные параметры "Tol" и 0,01.

```
decorrstretched_truecolor = decorrstretch(truecolor, 'Tol', 0.01);
f6 = figure;
imshow(decorrstretched_truecolor)
figure(f6);
title('Truecolor Composite after Decorrelation Stretch')
```

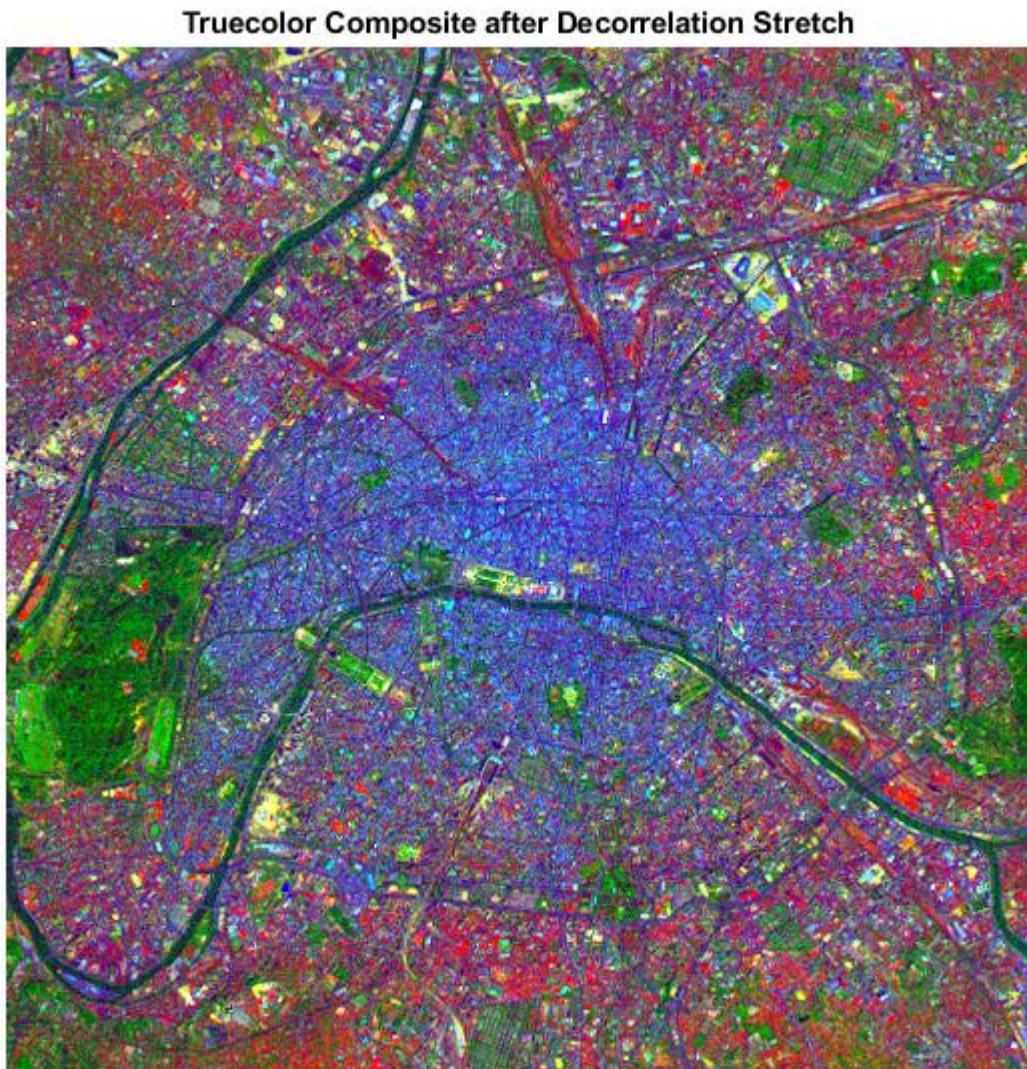


Рисунок 52. – Цветное изображение после декорреляции слоев

Свойства поверхности стали гораздо более заметными, но иначе. Спектральные различия изображения были преувеличены. Заметным примером является область зеленого цвета на левом краю. Эта зеленая зона - Булонский лес, большой парк на западном краю Парижа.

Шаг 7: проверка корреляции после декорреляции.

Как и ожидалось, диаграмма рассеяния после растяжения декорреляции показывает сильное уменьшение корреляции.

```
r = decorrstretched_truecolor(:,:,1);
g = decorrstretched_truecolor(:,:,2);
b = decorrstretched_truecolor(:,:,3);
f7 = figure;
```

```

plot3(r(:),g(:),b(:),'.')
grid('on')
xlabel('Red (Band 3)')
ylabel('Green (Band 2)')
zlabel('Blue (Band 1)')
figure(f7);
title('Scatterplot of the Visible Bands after Decorrelation Stretch')

```

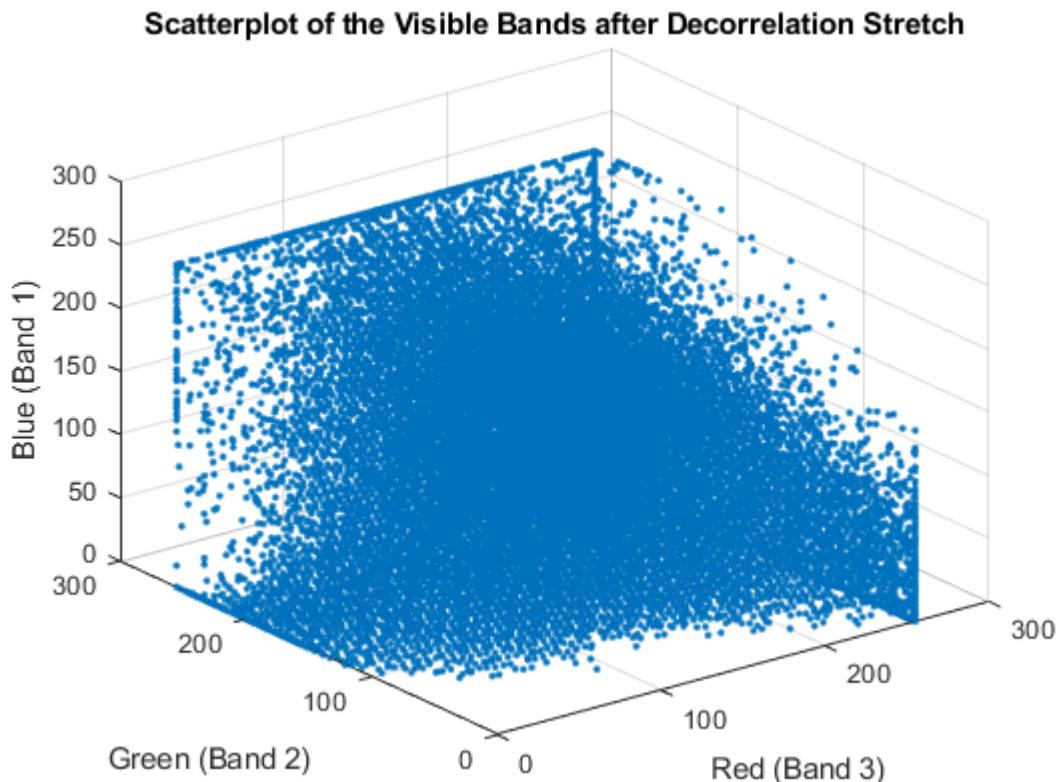


Рисунок 53. – Точечный график видимых слоев после декорреляции

Шаг 8: создание и улучшение CIR-изображения.

Как и в случае с видимыми слоями, информация из слоёв Landsat, содержащих невидимые части спектра, может быть просмотрена путем создания и улучшения композитных RGB изображений. Диапазон ближнего инфракрасного (NIR) (диапазон 4) важен из-за высокой отражательной способности хлорофилла в этой части спектра. Это еще более полезно в сочетании с видимыми красными и зелеными слоями (группы 3 и 2 соответственно) для формирования составного изображения цвета инфракрасного (CIR) цвета. Цветные инфракрасные (CIR) композиты

обычно используются для нахождения растительности или оценки ее состояния.

Построим изображение CIR, прочитав исходный LAN файл и составим изображение RGB, которое отображает 4, 3 и 2 слои, красный, зеленый и синий соответственно.

```
CIR = multibandread('paris.lan', [512, 512, 7], 'uint8=>uint8', ...
    128, 'bil', 'ieee-le', {'Band','Direct',[4 3 2]});
```

Несмотря на то, что ближний инфракрасный (NIR) слой (Band 4) меньше коррелирует с видимыми слоями, чем видимые слои друг с другом, декорреляция помогает лучше различать особенности и различия.

```
stretched_CIR = decorrstretch(CIR, 'Tol', 0.01);
f8 = figure;
imshow(stretched_CIR)
figure(f8);
title('CIR after Decorrelation Stretch')
```

CIR after Decorrelation Stretch

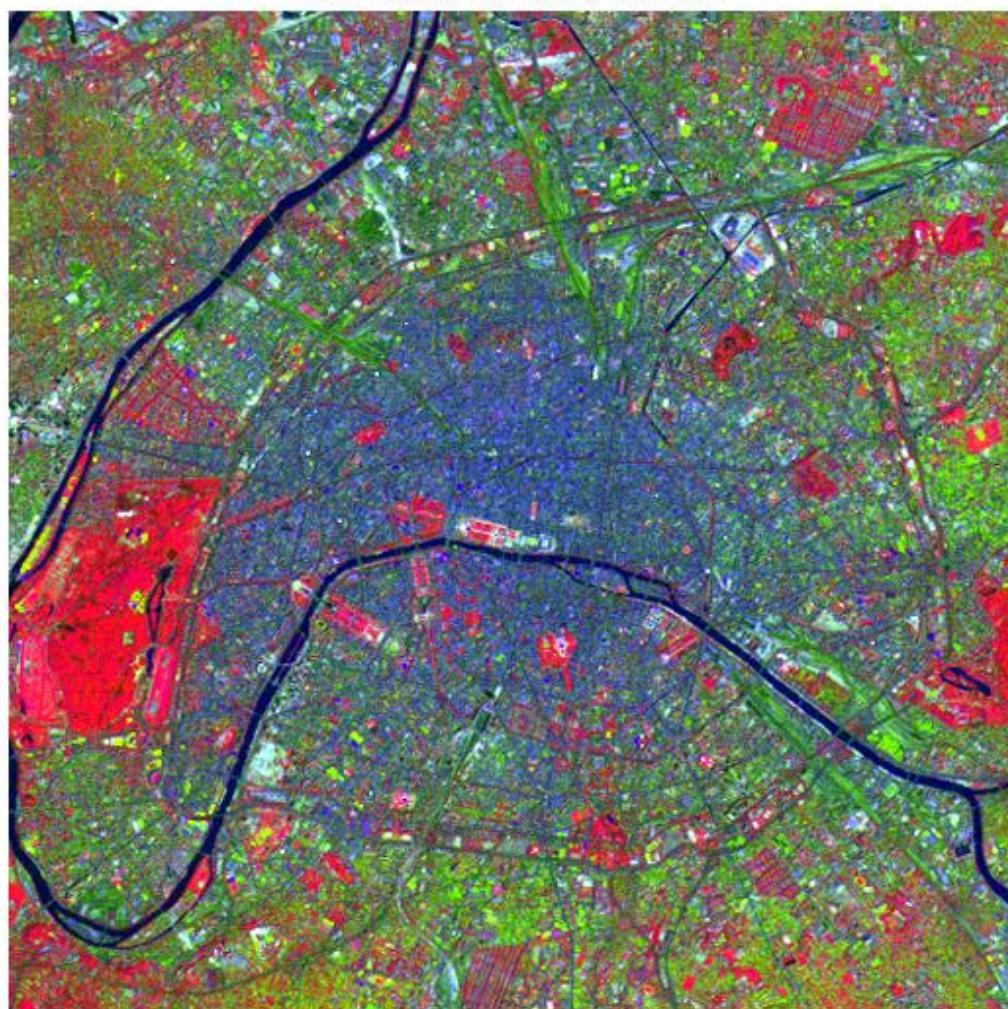


Рисунок 54. – Инфракрасное изображение после декорреляции слоев

Свойством цветных инфракрасных композитов является то, что они выглядят красными в областях с высокой плотностью растительности (из-за наличия хлорофилла). Обратите внимание, что парк Булонский лес окрашен красным в CIR-композите, что согласуется с его зеленым внешним видом в декоррелированном truecolor изображении.

10. Исправление неравномерного освещение и анализ объектов переднего плана

В этом примере показано, как улучшить изображение в виде шага предварительной обработки перед анализом. В этом примере вы исправляете неравномерное фоновое освещение и преобразуете, изображение в двоичное изображение, чтобы было легко идентифицировать объекты переднего плана. Затем вы можете проанализировать объекты, например, найти область каждой пиксели, и вы можете вычислить статистику для всех объектов на изображении.

Предварительная обработка изображения

Загрузите изображение

```
I = imread('C:\Users\vanov\OneDrive\Desktop\ger1.jpg');  
imshow(I);
```

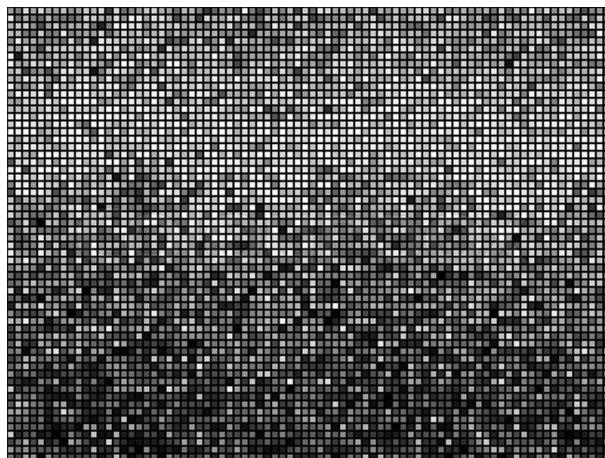


Рисунок 55. – Оригинальное изображение

Предварительно обработайте изображение, чтобы сделать фоновое освещение более однородным.

В качестве первого шага удалите весь передний план с помощью морфологического открытия. Операция открытия удаляет небольшие объекты, которые не могут полностью содержать элемент

структурирования. Определите дискообразный структурирующий элемент с радиусом 15, который полностью помещается внутри одного пикселя.

```
se = strel('disk',15);
```

```
se =
```

```
strel is a disk shaped structuring element with properties:
```

```
Neighborhood: [29x29 logical]
```

```
Dimensionality: 2
```

Чтобы выполнить морфологическое открытие, используйте `imopen` со структурирующим элементом.

```
background = imopen(I,se);  
imshow(background)
```

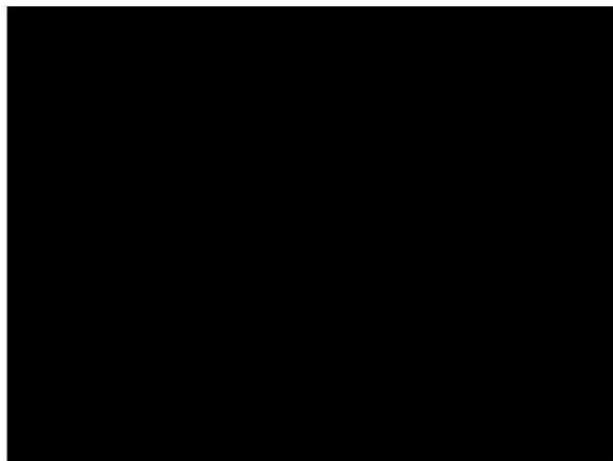


Рисунок 56. – Морфологическое открытие изображения

Вычтите фоновое приближенное изображение, фон, исходное изображение, `I`, и просмотрите результирующее изображение. После вычитания скорректированного фонового изображения из исходного изображения результирующее изображение имеет однородный фон, но теперь для анализа он немного темный.

```
I2 = I - background;  
imshow(I2)
```

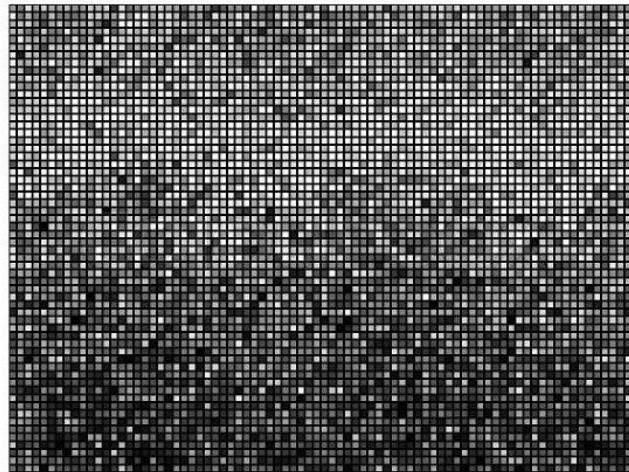


Рисунок 57. – Результирующее изображение

Используйте `imadjust` для увеличения контрастности обработанного изображения `I2`, насыщая 1% данных при низких и высоких интенсивностях и растягивая значения интенсивности, чтобы заполнить динамический диапазон `uint8`.

```
I3 = imadjust(I2);  
imshow(I3)
```

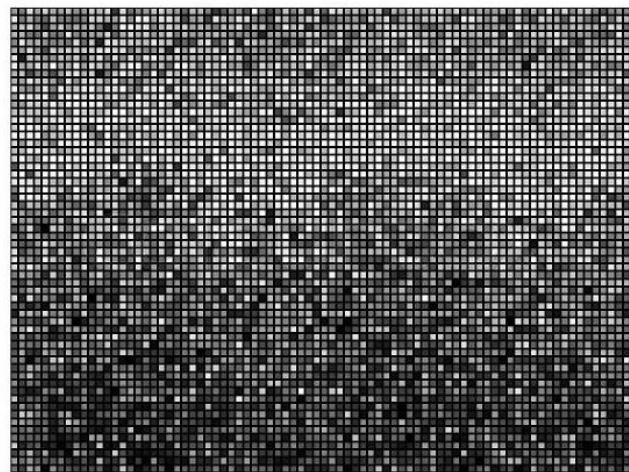


Рисунок 58. – Результирующее контрастное изображение

Обратите внимание, что предыдущие два этапа могут быть заменены одним шагом с использованием `imtophat`, который сначала вычисляет морфологическое открытие, а затем вычитает его из исходного изображения.

```
I2 = imtophat(I,strel('disk',15));
```

Создайте двоичную версию обработанного изображения, чтобы вы могли использовать функции панели инструментов для анализа. Используйте функцию imbinarize для преобразования изображения в оттенках серого в двоичное изображение. Удалите фоновый шум с изображения с помощью функции bwareaopen.

```
bw = imbinarize(I3);
bw = bwareaopen(bw,50);
imshow(bw)
```



Рисунок 59. – Двоичная версия обработанного изображения

Определение объектов в изображении

Теперь, когда вы создали двоичную версию исходного изображения, вы можете выполнить анализ объектов на изображении.

Найдите все подключенные компоненты (объекты) в двоичном изображении. Точность ваших результатов зависит от размера объектов, от параметра подключения (4, 8 или произвольное количество) и от того, касаются ли какие-либо объекты (в этом случае они могут быть помечены как один объект). Некоторые из пикселей в бинарном изображении *bw* касаются друг друга.

```
cc = bwconncomp(bw,4);
cc =
```

```

struct with fields:
  Connectivity: 4
    ImageSize: [600 800]
    NumObjects: 2665
    PixelIdxList: {1x2665 cell}
cc.NumObjects
ans =2665

```

Посмотрите на пиксель, обозначенный 50 на фотографии.

```

grain = false(size(bw));
grain(cc.PixelIdxList{50}) = true;
imshow(grain);

```



Рисунок 60. – Анализ объектов на изображении

Визуализируйте все подключенные компоненты изображения, создав матрицу меток, а затем отобразите ее как псевдо слоевое индексированное изображение.

Используйте `labelmatrix` для создания матрицы меток из вывода `bwconncomp`. Обратите внимание, что `labelmatrix` сохраняет матрицу меток в наименьшем числовом классе, необходимом для количества объектов.

```

labeled = labelmatrix(cc);
whos labeled

```

Name	Size	Bytes	Class	Attributes
labeled	600x800	960000	uint16	

Используйте `label2rgb` для выбора цвета фона и того, как объекты в матрице меток сопоставляются с цветами в цветовой палитре. В псевдо цветном изображении метка, идентифицирующая каждый объект в

матрице меток, сопоставляется с другим цветом в связанной матрице цветов.

```
RGB_label = label2rgb(labeled,'spring','c','shuffle');
imshow(RGB_label)
```

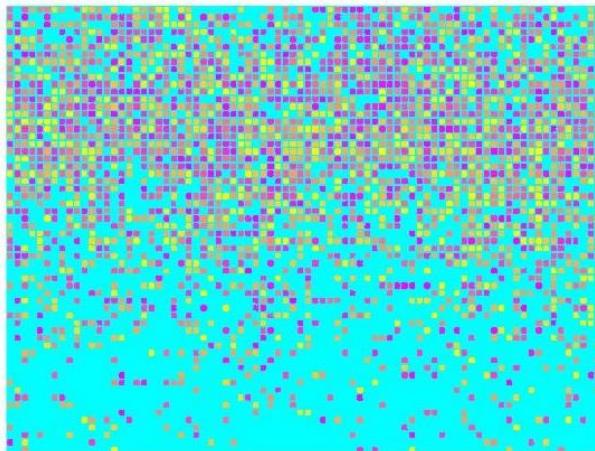


Рисунок 61. – Псевдо слоевое индексированное изображение

Статистика области вычислений

Вычислите область каждого объекта в изображении с помощью `regionprops`. Каждый пиксель является одним соединенным компонентом в `cc`-структуре.

```
graindata = regionprops(cc,'basic');
graindata =
2665x1 struct array with fields:
    Area
    Centroid
    BoundingBox
```

Создайте новый вектор `grain_areas`, который измеряет площадь для пикселя.

```
grain_areas = [graindata.Area];
```

Найдите область 50-го компонента.

```
grain_areas(50)
ans = 59
```

Найдите и покажите пиксель с наименьшей площадью.

```
[min_area, idx] = min(grain_areas)
min_area = 50
```

```
idx = 158
grain = false(size(bw));
grain(cc.PixelIdxList{idx}) = true;
imshow(grain)
```

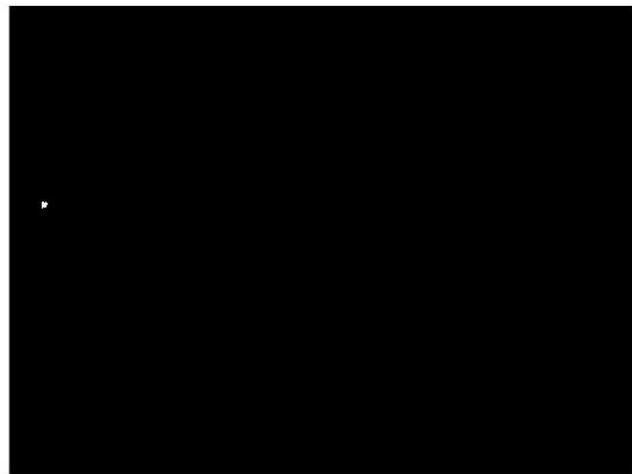


Рисунок 62. – Пиксель с наименьшей площадью

Используйте команду гистограммы для создания гистограммы областей пикселей.

```
histogram(grain_areas)
title('Histogram of Rice Grain Area')
```

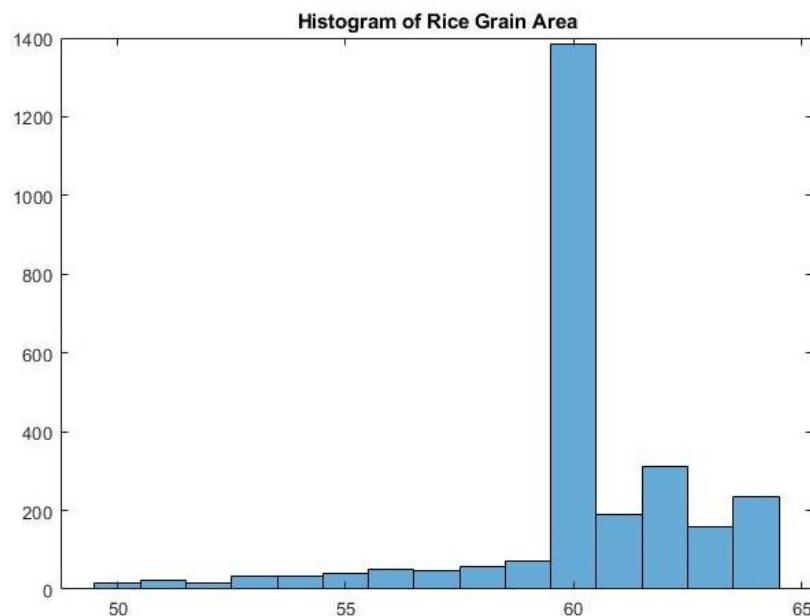


Рисунок 63. – Гистограмма областей пикселей

11. Вычисление статистики для больших изображений

В этом примере показано, как использовать blockproc для получения статистики из больших изображений, а затем использовать эту информацию для более точной обработки изображений по блоку. Blockproc функция хорошо подходит для применения операций к изображению поблочно, сборки результатов и возврата их в качестве нового изображения. Однако многие алгоритмы обработки изображений требуют «глобальной» информации об изображении, которая недоступна если вы рассматриваете только один блок данных изображения за раз. Эти ограничения могут оказаться проблематичными при работе с слишком большими изображениями из-за загрузки памяти.

В этом примере выполняется задача, аналогичная той, что описана в примере Enhanced Multispectral Color Composite Images, но адаптирована для больших изображений с использованием blockproc. Вы будете улучшать видимые слои файла Erdas LAN rio.lan. Такие типы методов обработки блоков обычно более полезны для больших изображений, но для этого примера будет использоваться небольшое изображение.

Шаг 1: постройте композит Truecolor.

Используя blockproc, прочтите данные из файла rio.lan, содержащие файлы изображений Landsat в формате Erdas LAN. Blockproc имеет встроенную поддержку чтения только для файлов TIFF и JPEG2000. Чтобы читать другие типы файлов, вы должны написать адаптер изображения для поддержки ввода-вывода для вашего конкретного формата файла. В этом примере используется предварительно построенный класс Image Adapter, LanAdapter, который поддерживает чтение файлов LAN. Для получения дополнительной информации о написании классов адаптера изображения см. руководство пользователя, описывающее, как был создан класс LanAdapter.

Формат LAN Erdas содержит видимый красный, зеленый и синий спектры в слоях 3, 2 и 1 соответственно. Используйте blockproc для сложения видимых слоев в RGB-изображение.

```
% Создайте объект LanAdapter, взаимодействующий с rio.lan.  
input_adapter = LanAdapter('rio.lan');
```

```
% Выберите видимые слои R, G и B.
```

```
input_adapter.SelectedBands = [3 2 1];
```

```
% Создайте функцию блока, чтобы просто вернуть данные блока без изменений.
```

```
identityFcn = @(block_struct) block_struct.data;
```

```
% Создайте исходное изображение truecolor.
```

```
truecolor = blockproc(input_adapter,[100 100],identityFcn);
```

```
% Отображение неулучшенных результатов.
```

```
figure;  
imshow(truecolor);  
title('Truecolor Composite (Un-enhanced)');
```

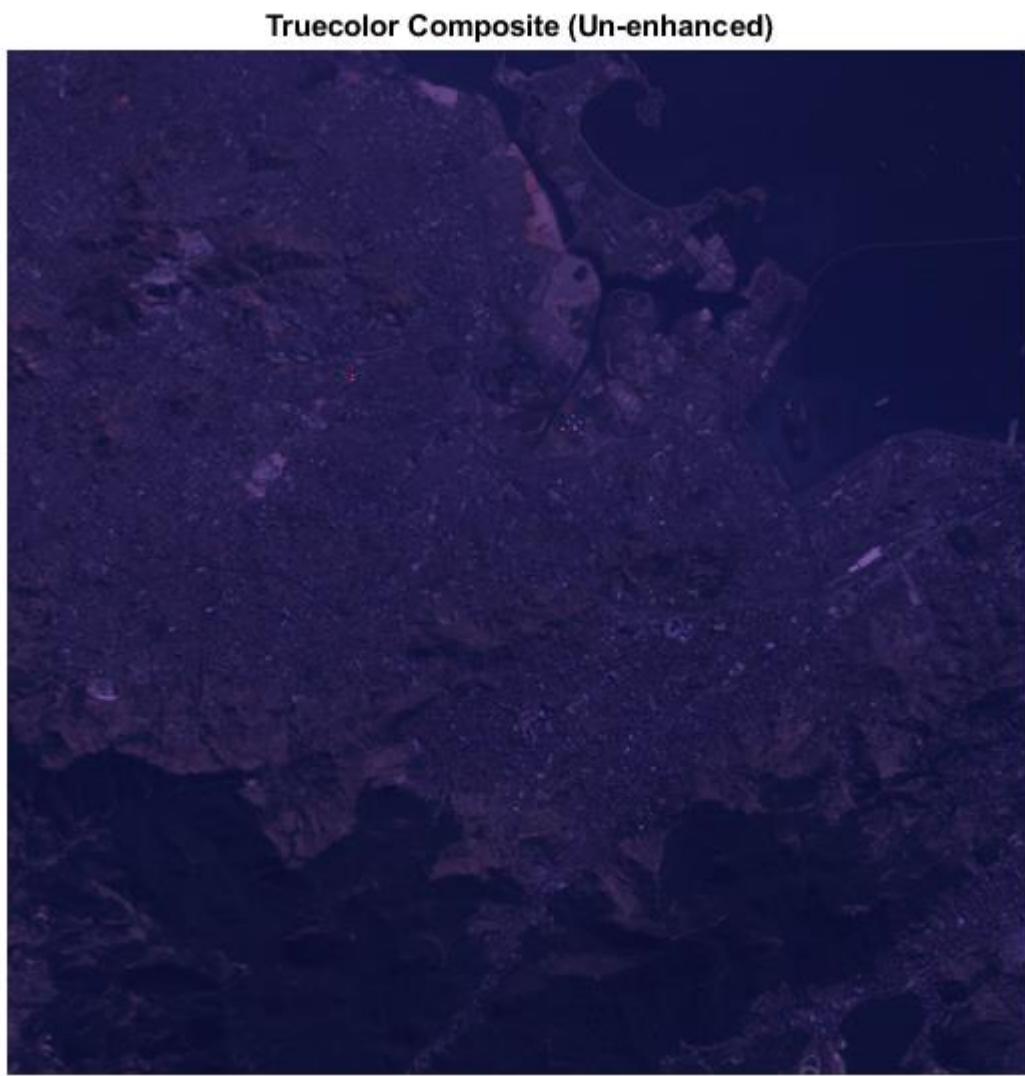


Рисунок 64. – Композитный цвет (не улучшенный)

Полученное truecolor изображение аналогично изображению paris.lan в примере Enhanced Multispectral Color Composite Images. Изображение RGB выглядит тусклым, с небольшим контрастом.

Шаг 2. улучшение изображения - первая попытка.

Во-первых, попробуйте растянуть данные в динамическом диапазоне с помощью `blockproc`. Эта команда просто определяет новый дескриптор функции, который вызывает `strchlim` и `imadjustin` блок данных по отдельности.

```
adjustFcn = @(block_struct) imadjust(block_struct.data,...
```

```

stretchlim(block_struct.data));
truecolor_enhanced = blockproc(input_adapter,[100 100],adjustFcn);
figure
imshow(truecolor_enhanced)
title('Truecolor Composite with Blockwise Contrast Stretch')

```

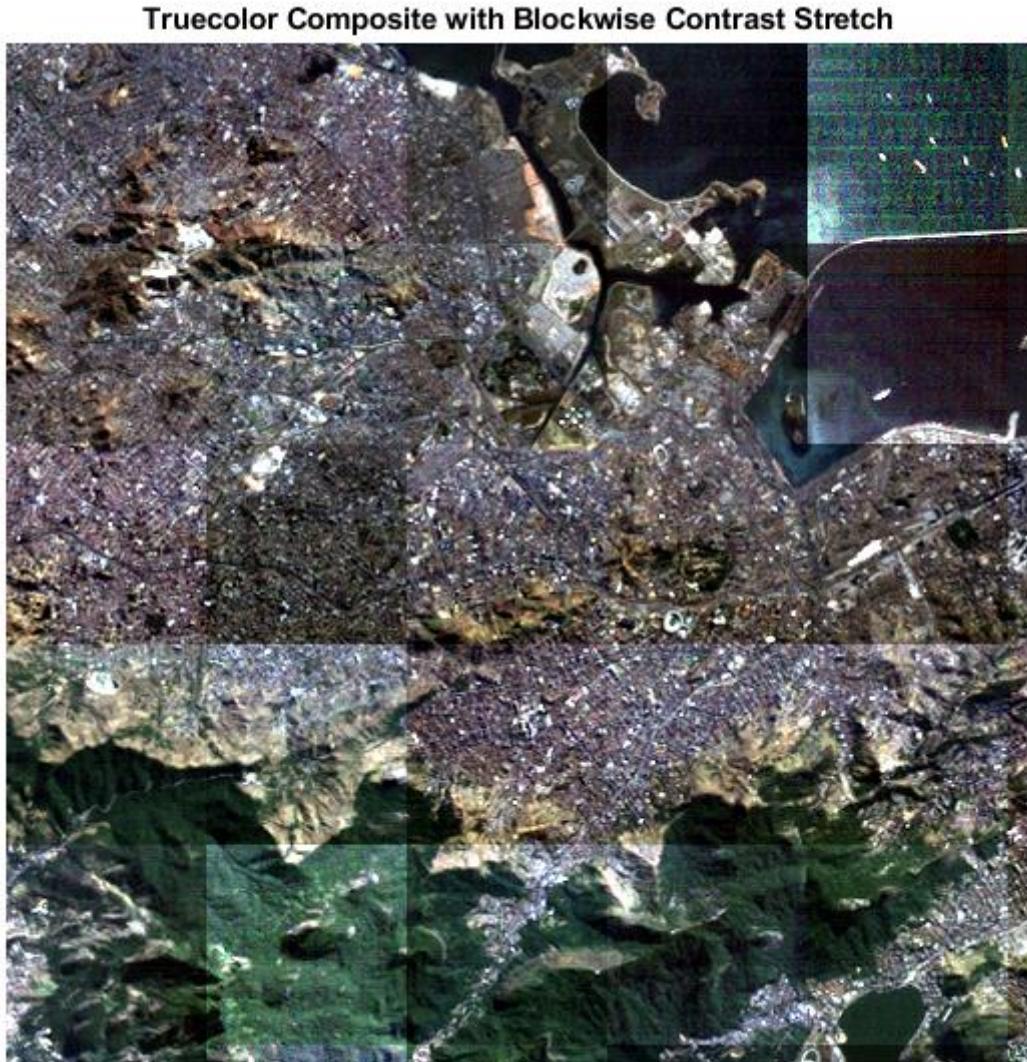


Рисунок 65. – Композитный цвет с контрастной растяжкой

Вы можете сразу увидеть, что результаты неверны. Проблема заключается в том, что функция растяжения вычисляет гистограмму на исходном изображении и использует эту информацию для вычисления пределов растяжения. Поскольку каждый блок настраивается отдельно от

соседних, каждый блок вычисляет различные пределы от своей локальной гистограммы.

Шаг 3: изучение класса сумматоров гистограммы.

Чтобы исследовать распределение данных по динамическому диапазону изображения, вы можете вычислить гистограмму для каждого из трех видимых диапазонов.

При работе с достаточно большими изображениями вы не можете просто вызвать imhist для создания гистограммы изображения. Один из способов постепенного построения гистограммы - использовать blockproc с классом, который суммирует гистограммы каждого блока изображения.

Изучение класса HistogramAccumulator .

```
type HistogramAccumulator
classdef HistogramAccumulator < handle

properties
    Histogram
    Range
end

methods

    function obj = HistogramAccumulator()
        obj.Range = [];
        obj.Histogram = [];
    end

    function addToHistogram(obj,new_data)
        if isempty(obj.Histogram)
            obj.Range = double(0:intmax(class(new_data)));
            obj.Histogram = hist(double(new_data(:)),obj.Range);
        else
            new_hist = hist(double(new_data(:)),obj.Range);
            obj.Histogram = obj.Histogram + new_hist;
        end
    end
end
```

Класс - простая оболочка вокруг функции hist, позволяющая добавлять данные к гистограмме постепенно. Это не особенность blockproc.
Рассмотрим простое использование класса HistogramAccumulator.

```
% Создание HistogramAccumulator объекта.
hist_obj = HistogramAccumulator();

% Разделить изображение на 2 половины.
full_image = imread('liftingbody.png');
top_half = full_image(1:256,:);
bottom_half = full_image(257:end,:);

% Вычисление гистограмму пошагово.
addToHistogram(hist_obj, top_half);
addToHistogram(hist_obj, bottom_half);
computed_histogram = hist_obj.Histogram;

% Сравнение с результатами IMHIST.
normal_histogram = imhist(full_image);

% Оценка результата. Гистограммы численно идентичны.
figure
subplot(1,2,1);
stem(computed_histogram,'Marker','none');
title('Incrementally Computed Histogram');
subplot(1,2,2);
stem(normal_histogram,'Marker','none');
title('IMHIST Histogram');
```

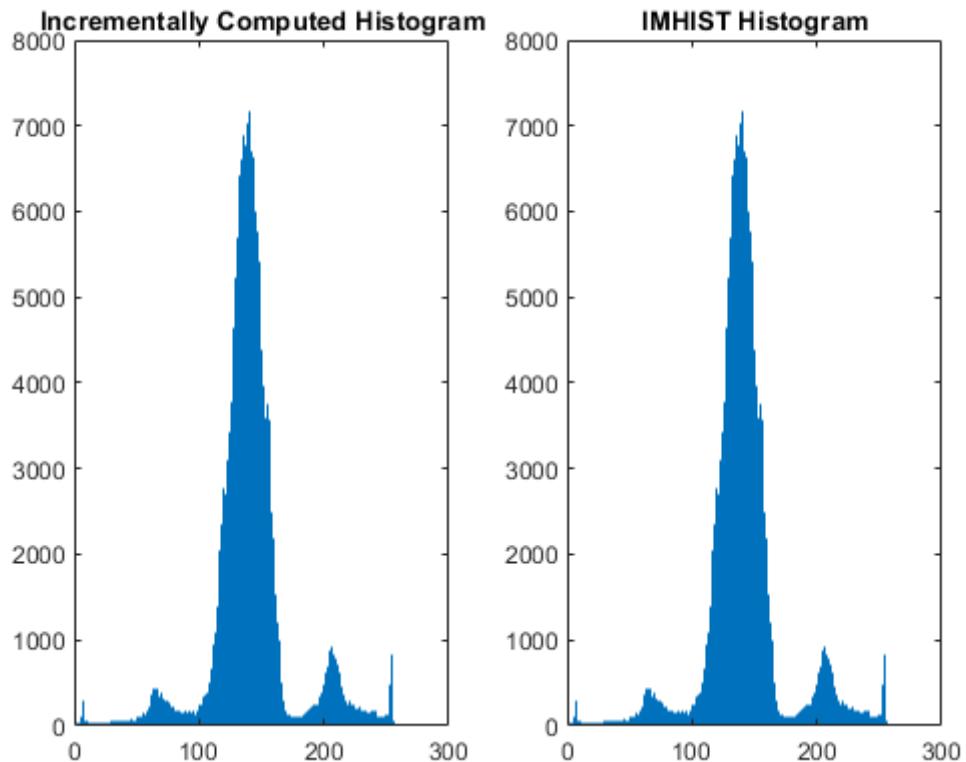


Рисунок 66. – Постепенно вычисляемая гистограмма(слева), IMHIST гистограмма(справа)

Шаг 4: использование HistogramAccumulator Class с BLOCKPROC.

Теперь используйте класс HistogramAccumulator с blockproc, чтобы построить гистограмму красного слоя rio.lan. Вы можете определить функцию для blockprocthat, которая вызовет метод addToHistogram для каждого блока данных. Просмотрев эту гистограмму, вы увидите, что данные сосредоточены в небольшой части доступного динамического диапазона. Другие видимые полосы имеют одинаковые распределения. Это одна из причин, почему оригинальный композит truecolor выглядит тусклым.

```
% Создание объекта HistogramAccumulator.
hist_obj = HistogramAccumulator();

% Настройка функции blockproc
addToHistFcn = @(block_struct) addToHistogram(hist_obj,
block_struct.data);
```

```

% Вычислить гистограмму красного канала. Обратите внимание, что
addToHistFcn
% function utythbhetn вывод. Поскольку функция handle которую
% передают в blockproc, ничего не возвращает, blockproc тоже ничего
% не вернет.
input_adapter.SelectedBands = 3;
blockproc(input_adapter,[100 100],addToHistFcn);
red_hist = hist_obj.Histogram;

% Показать результат.
figure
stem(red_hist,'Marker','none');
title('Histogram of Red Band (Band 3)');

```

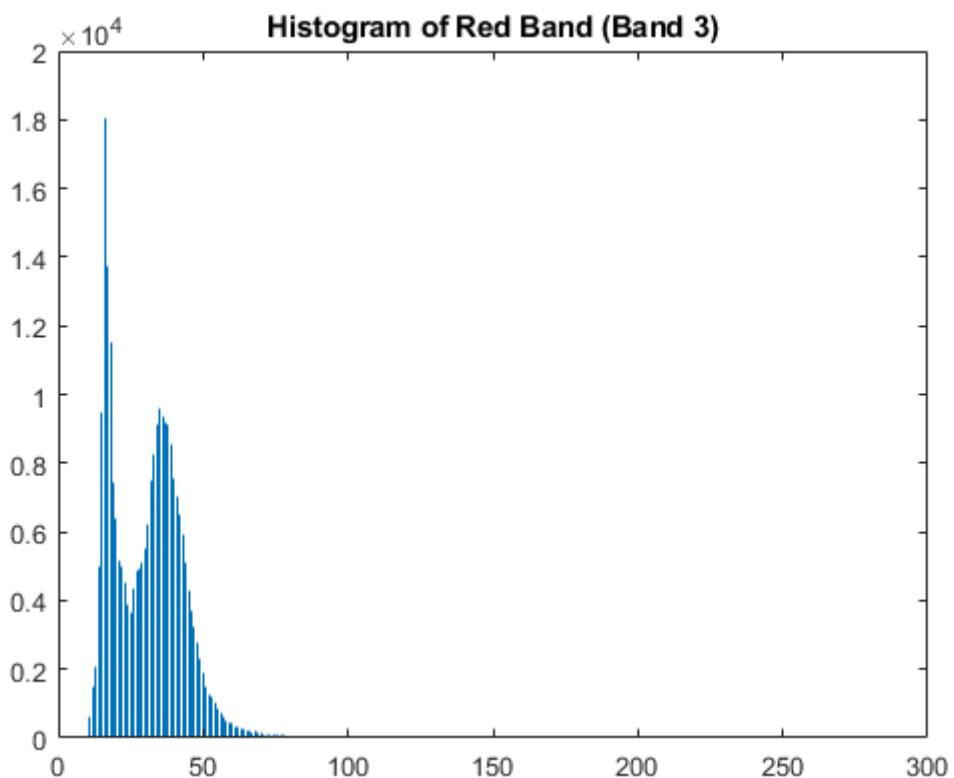


Рисунок 67. – Гистограмма красного слоя

Шаг 5: улучшение Truecolor комозита с помощью контрастного растяжения.

Теперь можно выполнить правильное контрастное растяжение изображения. Для обычных рабочих процессов в памяти вы можете просто использовать функцию stretchlim для вычисления аргументов для imadjust

(например, для параметра MultispectralImageEnhancementExampleExample). При работе с большими изображениями, как мы видели, растяжение трудно адаптируется для использования с blockproc, поскольку она опирается на полную гистограмму изображения.

После того, как вы вычислили гистограммы изображений для каждого из видимых слоев, вычислите правильные аргументы для манипуляции вручную.

Сначала получите гистограммы для зеленых и синих полос.

```
% Вычислить гистограмму для зеленого канала.  
hist_obj = HistogramAccumulator();  
addToHistFcn = @(block_struct) addToHistogram(hist_obj,  
block_struct.data);  
input_adapter.SelectedBands = 2;  
blockproc(input_adapter,[100 100],addToHistFcn);  
green_hist = hist_obj.Histogram;  
  
% Вычислить гистограмму для синего канала.  
hist_obj = HistogramAccumulator();  
addToHistFcn = @(block_struct) addToHistogram(hist_obj,  
block_struct.data);  
input_adapter.SelectedBands = 1;  
blockproc(input_adapter,[100 100],addToHistFcn);  
blue_hist = hist_obj.Histogram;  
  
% Теперь вычислите CDF каждой гистограммы и подготовьтесь к вызову  
imadjust.  
computeCDF = @(histogram) cumsum(histogram) / sum(histogram);  
findLowerLimit = @(cdf) find(cdf > 0.01, 1, 'first');  
findUpperLimit = @(cdf) find(cdf >= 0.99, 1, 'first');  
red_cdf = computeCDF(red_hist);  
red_limits(1) = findLowerLimit(red_cdf);  
red_limits(2) = findUpperLimit(red_cdf);  
green_cdf = computeCDF(green_hist);  
green_limits(1) = findLowerLimit(green_cdf);  
green_limits(2) = findUpperLimit(green_cdf);  
blue_cdf = computeCDF(blue_hist);  
blue_limits(1) = findLowerLimit(blue_cdf);  
blue_limits(2) = findUpperLimit(blue_cdf);  
  
% Вычисление аргумента IMADJUST.  
rgb_limits = [red_limits' green_limits' blue_limits'];
```

```
% Масштабирование до [0,1].
rgb_limits = (rgb_limits - 1) / (255);

Создайте новый adjustFcn, который применяет глобальные пределы
растяжки и используйте blockproc для настройки изображения truecolor.
adjustFcn = @(block_struct) imadjust(block_struct.data,rgb_limits);

% Выберите весь RGB.
input_adapter.SelectedBands = [3 2 1];
truecolor_enhanced = blockproc(input_adapter,[100 100],adjustFcn);
figure;
imshow(truecolor_enhanced)
title('Truecolor Composite with Corrected Contrast Stretch')
```



Рисунок 68. – Композитный материал с корректным контрастным
растяжением

Полученное изображение значительно улучшилось, а данные покрывают больше динамического диапазона, и с помощью blockproc вы избегаете загрузки всего изображения в память.

12. Улучшение изображения с низким уровнем освещенности

Изображения, снятые на открытых пространствах, могут быть сильно ухудшены из-за плохих условий освещения. Эти изображения могут иметь низкие динамические диапазоны с высоким уровнем шума, которые влияют на общую производительность алгоритмов компьютерного зрения. Чтобы алгоритмы компьютерного зрения были надежными в условиях низкой освещенности, используйте улучшение изображения с низким освещением. Гистограмма пиксельной инверсии изображений с низким освещением или изображений HDR очень похожа на гистограмму туманных изображений. Таким образом, можно использовать методы удаления дымки для улучшения изображений с низким освещением.

Использование методов удаления дымки для улучшения изображений с низким освещением состоит из трех этапов:

Шаг 1: инверсия изображения с низким освещением.

Шаг 2: применение алгоритма удаления мутности к инвертированному изображению с низким освещением.

Шаг 3: инверсия расширенного изображения.

Улучшение изображения с низким уровнем освещенности с использованием алгоритма Dehazing

Импорт изображения RGB, снятого при слабом освещении.

```
A = imread('lowlight_11.jpg');  
figure, imshow(A);
```



Рисунок 69. – Оригинал изображения

Инверсия изображения. Области с низким освещением в исходном изображении кажутся туманными.

```
AInv = imcomplement(A);  
figure, imshow(AInv);
```



Рисунок 70. – Инверсия изображения

Уменьшение тумана, при помощи функции *imreducehaze*.

```
BInv = imreducehaze(AInv);  
figure, imshow(BInv);
```

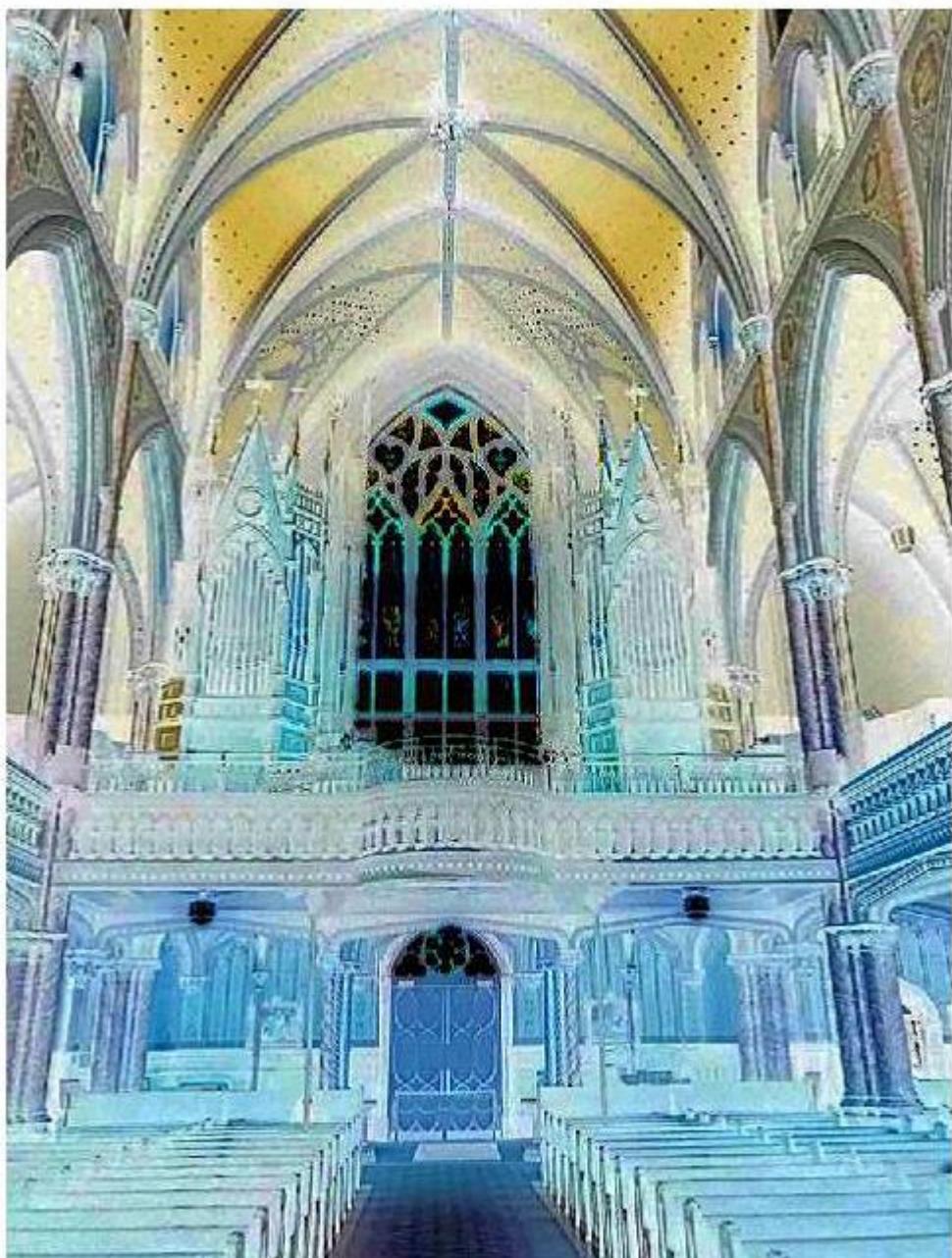


Рисунок 71. – Инверсия изображения с уменьшением тумана

Инверсия результата, для получения улучшенного изображения.

```
B = imcomplement(BInv);
```

Отображение исходного и улучшенного изображения.

```
figure, montage({A, B});
```

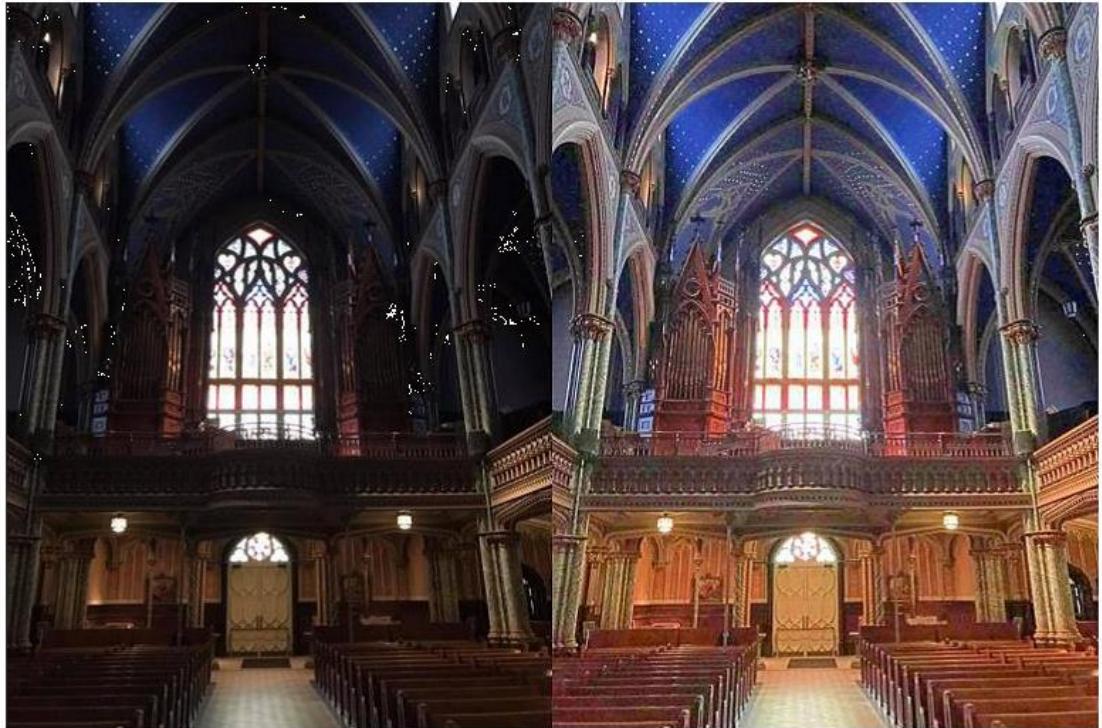


Рисунок 20. – Исходное и улучшенное изображение

13. Гранулометрия снежинок

В этом примере показано, как рассчитать распределение по размерам снежинок в изображении с помощью гранулометрии. Гранулометрия определяет распределение по размеру объектов в изображении без явной сегментации (обнаружения) каждого объекта.

Чтение изображения

Считайте изображение «snowflakes.png», которое представляет собой фотографию снежинок.

```
I = imread('snowflakes.png');  
imshow(I)
```

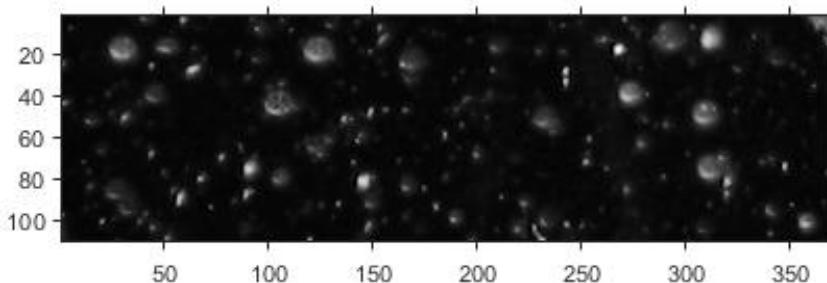


Рисунок 73. – Оригинальное изображение

Улучшение контрастности

Для начала нужно максимизировать контраст интенсивности изображения. Это можно сделать с помощью функции adapthisteq, которая выполняет адаптивное выравнивание гистограммы с ограничением по контрасту. Измените масштаб изображения с помощью функции imadjust, чтобы оно заполнило весь динамический диапазон типа данных.

```
claheI = adaptthisteq(I, 'NumTiles' , [10 10]);  
claheI = imadjust(claheI);  
imshow(claheI)
```

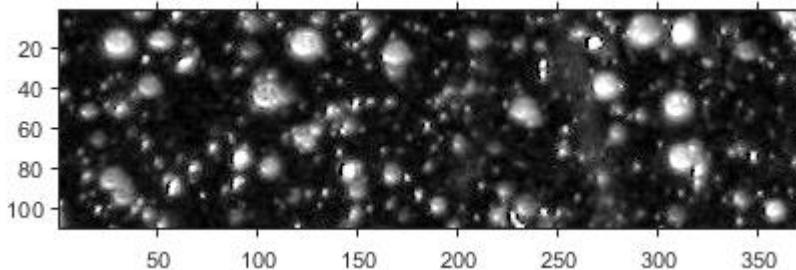


Рисунок 74. – Контрастное изображение

Определение распределения площади поверхности интенсивности в расширенном изображении

Гранулометрия оценивает распределение площади поверхности снежинок в зависимости от размера; сравнивает объекты изображения с камнями, размеры которых могут быть определены путем просеивания их через экраны увеличивающегося размера и сбора остатков после каждого прохода. Объекты изображения просеиваются, открывая изображение с помощью структурирующего элемента увеличения размера и подсчета оставшейся площади поверхности интенсивности (суммирование значений пикселей на изображении) после каждого открытия.

Выбирайте предельный счетчик, чтобы площадь поверхности интенсивности уменьшалась, когда вы увеличиваете размер вашего структурирующего элемента. Для отображения оставьте первую запись в массиве поверхности пустой.

```
radius_range = 0:22;
intensity_area = zeros(size(radius_range));
for counter = radius_range
    remain = imopen(claheI, strel('disk', counter));
    intensity_area(counter + 1) = sum(remain(:));
end
figure
plot(intensity_area, 'm - *')
grid on
title('Sum of pixel values in opened image versus radius')
xlabel('radius of opening (pixels)')
ylabel('pixel value sum of opened objects (intensity)')
```

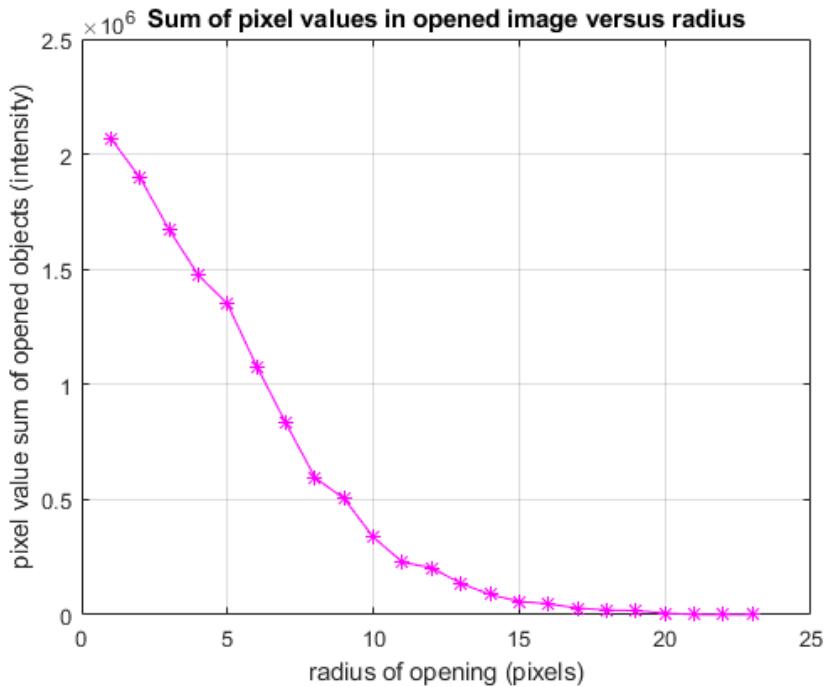


Рисунок 75. – Сумма значений пикселей в изображении в зависимости от радиуса

Вычисление первой производной распределения

Значительное падение площади поверхности интенсивности между двумя последовательными отверстиями указывает на то, что изображение содержит объекты размера, сопоставимого с меньшим отверстием. Это эквивалентно первой производной массива площади поверхности интенсивности, которая содержит распределение по размерам снежинок на изображении. Вычислим первую производную с помощью функции diff.

```
intensity_area_prime = diff(intensity_area);
plot(intensity_area_prime, 'm - *')
grid on
title('Granulometry (Size Distribution) of Snowflakes')
ax = gca;
ax.XTick = [0 2 4 6 8 10 12 14 16 18 20 22];
xlabel('radius of snowflakes (pixels)')
ylabel('Sum of pixel values in snowflakes as a function of radius')
```

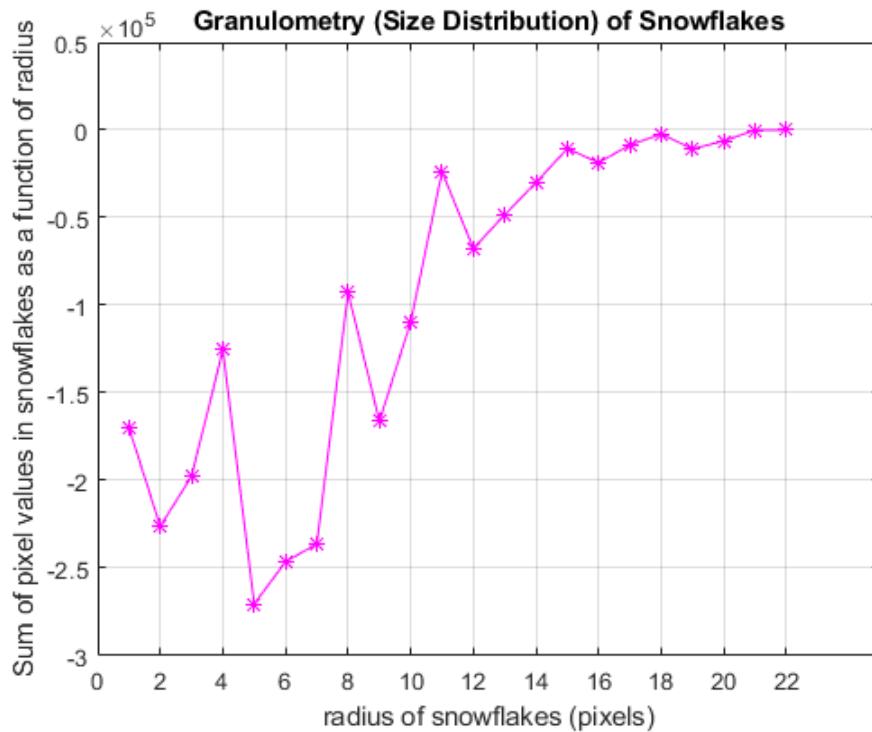


Рисунок 76. – Гранулометрия (распределение по размерам) снежинок

Извлечение снежинок с определенным радиусом

Обратите внимание на минимумы и радиусы, где они встречаются на графике. Минимумы говорят о том, что снежинки на изображении имеют эти радиусы. Чем меньше точка минимума, тем выше суммарная интенсивность снежинок на этом радиусе. Например, самая отрицательная точка минимума возникает на отметке 5 пикселей. Вы можете извлечь снежинки с радиусом 5 пикселей со следующими шагами.

```
open5 = imopen(claheI,strel('disk',5));
open6 = imopen(claheI,strel('disk',6));
rad5 = imsubtract(open5,open6);
imshow(rad5,[])
```

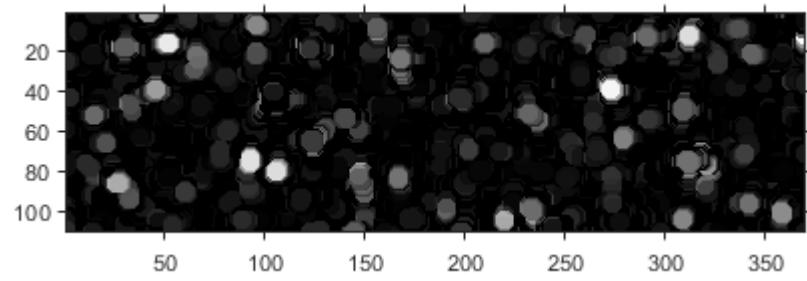


Рисунок 77. – Изображение снежинок с радиусом 5 пикселей

14. Измерение областей в оттенках серого

В этом примере показано, как измерять свойства объектов в полутоновом изображении. Чтобы выполнить это, сначала отрисуйте изображение в градациях серого, чтобы получить двоичное изображение объектов. Затем используйте `regionprops` для анализа исходных значений пикселей в оттенках серого, соответствующих каждому объекту в двоичном изображении.

Создание изображений в градациях серого.

Используйте вспомогательную функцию `propsSynthesizeImage`, чтобы создать изображение в градациях серого, содержащее пять отдельных областей.

```
I = propsSynthesizeImage;
imshow(I)
title('Synthetic Image')
```

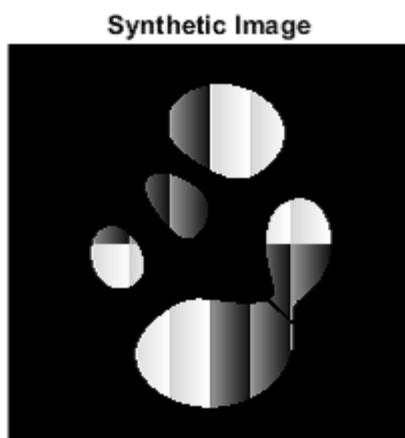


Рисунок 78. – Синтетическое изображение

Создание двоичного образа

Выделите изображение в градациях серого, создав двоичное изображение, содержащее объекты изображения.

```
BW = I > 0;
imshow(BW)
title('Binary Image')
```



Рисунок 79. – Двоичное изображение

Вычисление свойств объекта с использованием пиксельных значений изображения в градациях серого

Функция `regionprops` поддерживает несколько свойств, которые могут использоваться с изображениями в оттенках серого, включая «`WeightedCentroid`», «`MeanIntensity`», «`MinIntensity`» и «`MaxIntensity`». Эти свойства используют исходные значения пикселей объектов для их вычислений.

Например, вы можете использовать `regionprops` для вычисления центроида и взвешенного центра тяжести объектов на изображении. Обратите внимание, как вы передаете двоичное изображение (`BW`), содержащее ваши объекты, и исходное изображение в оттенках серого (`I`) в качестве аргументов в `regionprops`.

```
s = regionprops(BW,I,['Centroid','WeightedCentroid']);
```

Чтобы сравнить взвешенные местоположения центроидов с невзвешенными центроидными местоположениями, отобразите исходное изображение, а затем наложите центроиды на изображение.

```
imshow(I)
title('Weighted (red) and Unweighted (blue) Centroids');
hold on
numObj = numel(s);
for k = 1 : numObj
plot(s(k).WeightedCentroid(1), s(k).WeightedCentroid(2), 'r*')
```

```

plot(s(k).Centroid(1), s(k).Centroid(2), 'bo')
end
hold off

```

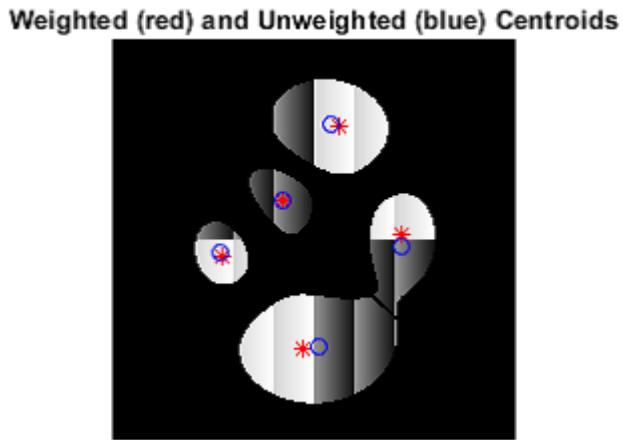


Рисунок 80. – Взвешенные (звездочки) и не взвешенные (кружочки) центроиды

Вычислить свойства на основе значения пользовательских пикселей

Можно использовать свойство «PixelValues» делать пользовательские вычисления, основанные на значения пикселей исходного изображения в оттенках серого. Свойство «PixelValues» возвращает вектор, содержащий значения серого пикселей в объекте.

Например, вычислим стандартное отклонение каждого объекта.

```

s = regionprops(BW,I,['Centroid','PixelValues','BoundingBox']);
imshow(I)
title('Standard Deviation of Regions')
hold on
for k = 1 : numObj
s(k).StandardDeviation = std(double(s(k).PixelValues));
text(s(k).Centroid(1),s(k).Centroid(2), ...
sprintf('%2.1f', s(k).StandardDeviation), ...
'EdgeColor','b','Color','r');
end
hold off

```

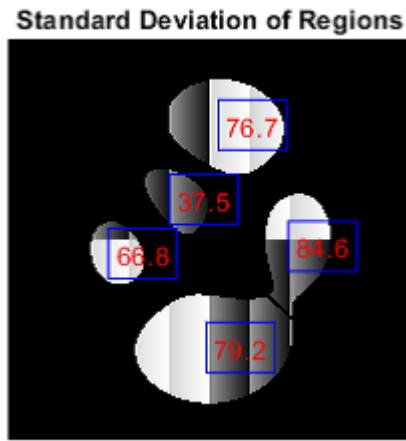


Рисунок 81. – Стандартное отклонение каждого объекта

Эта цифра показывает стандартное отклонение измерения. Вы можете также просмотреть результаты другими способами:

```
figure
bar(1:numObj,[s.StandardDeviation])
xlabel('Region Label Number')
ylabel('Standard Deviation')
```

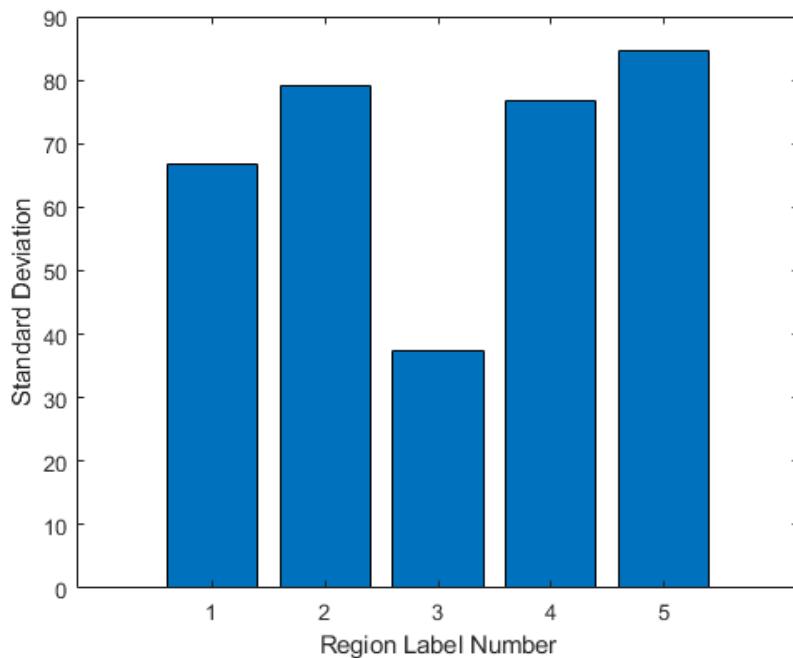


Рисунок 82. – Отношение стандарта отклонения от номера метки объекта

15. Использование фазовой корреляции в качестве шага предварительной обработки при регистрации изображения

В этом примере показано, как использовать фазовую корреляцию в качестве предварительного шага автоматической регистрации изображения. В этом процессе вы выполняете фазовую корреляцию, используя `imregcorr`, а затем передаете результат этой регистрации в качестве начального условия регистрации на основе оптимизации, используя `imregister`. Фазовая корреляция и регистрация на основе оптимизации - это дополнительные алгоритмы. Фазовая корреляция хороша для нахождения грубого выравнивания, даже для сильно несогласованных изображений. Оптимационная регистрация хороша для нахождения точного выравнивания, учитывая хорошее начальное состояние.

Загрузите изображение, которое будет ссылочным изображением при регистрации.

```
fixed = imread('C:\Users\vanov\OneDrive\Desktop\bird.png');  
imshow(fixed);
```



Рисунок 83. – Оригинальное изображение

Создайте незарегистрированное изображение, преднамеренно искажая это изображение, используя поворот, изотропное масштабирование и сдвиг в направлении Y.

```
theta = 170;
rot = [cosd(theta) sind(theta) 0;...
        sind(theta) cosd(theta) 0;...
        0 0 1];
sc = 2.3;
scale = [sc 0 0; 0 sc 0; 0 0 1];
sh = 0.5;
shear = [1 sh 0; 0 1 0; 0 0 1];
tform = affine2d(shear*scale*rot);
moving = imwarp(fixed,tform);
```

Добавьте шум к изображению и покажите результат.

```
moving = moving + uint8(10*rand(size(moving)));
imshow(moving)
```



Рисунок 84. – Изображение с шумами

Оцените регистрацию, необходимую для согласования этих двух изображений. imregcorr возвращает объект affine2d, который определяет преобразование.

```
tformEstimate = imregcorr(moving,fixed);
```

Примените расчетное геометрическое преобразование к несогласованному изображению. Укажите «OutputView», чтобы убедиться,

что зарегистрированное изображение имеет тот же размер, как эталонное изображение. Отображение исходного изображения и зарегистрированного изображения бок о бок. Вы можете видеть, что imregcorr отлично справился с поворотом и масштабированием различий между изображениями. Зарегистрированное изображение, movingReg, очень близко к выравниванию с исходным изображением, исправлено. Но некоторая несоосность остается. imregcorr отлично справляется с искажениями вращения и масштабирования, но не искажает сдвиг.

```
Rfixed = imref2d(size(fixed));  
movingReg = imwarp(moving,tformEstimate,'OutputView',Rfixed);  
imshowpair(fixed,movingReg,'montage');
```

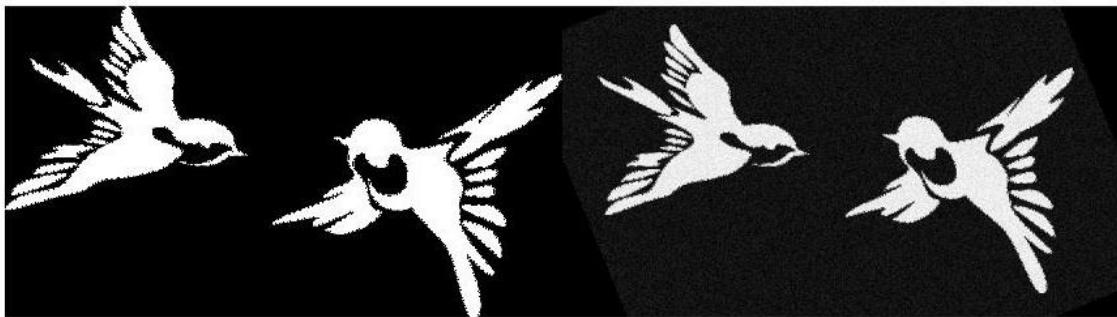


Рисунок 85. – Изображение с шумами наложенное на оригинал

Просмотрите выровненное изображение, наложенное на исходное изображение, используя imshowpair. В этом представлении imshowpair использует цвет, чтобы выделить области несоосности.

```
imshowpair(fixed,movingReg,'falsecolor');
```



Рисунок 86. – Несоосность изображений в цвете

Чтобы завершить регистрацию, используйте `imregister`, передав оценочное преобразование, возвращенное `imregcorr`, в качестве начального условия. `imregister` более эффективен, если два изображения грубо выравниваются в начале операции. Преобразование, оцененное `imregcorr`, предоставляет эту информацию для регистрации. В примере используются значения оптимизатора по умолчанию и метрики для регистрации двух изображений, сделанных с использованием одного и того же датчика («мономодальный»).

```
[optimizer, metric] = imregconfig('monomodal');
movingRegistered = imregister(moving, fixed, ...
    'affine', optimizer,
    metric, 'InitialTransformation', tformEstimate);
```

Отобразите результат этой регистрации. Обратите внимание, что `imregister` добился очень точной регистрации, учитывая хорошее начальное условие, предоставленное `imregcorr`.

```
imshowpair(fixed, movingRegistered, 'Scaling', 'joint');
```

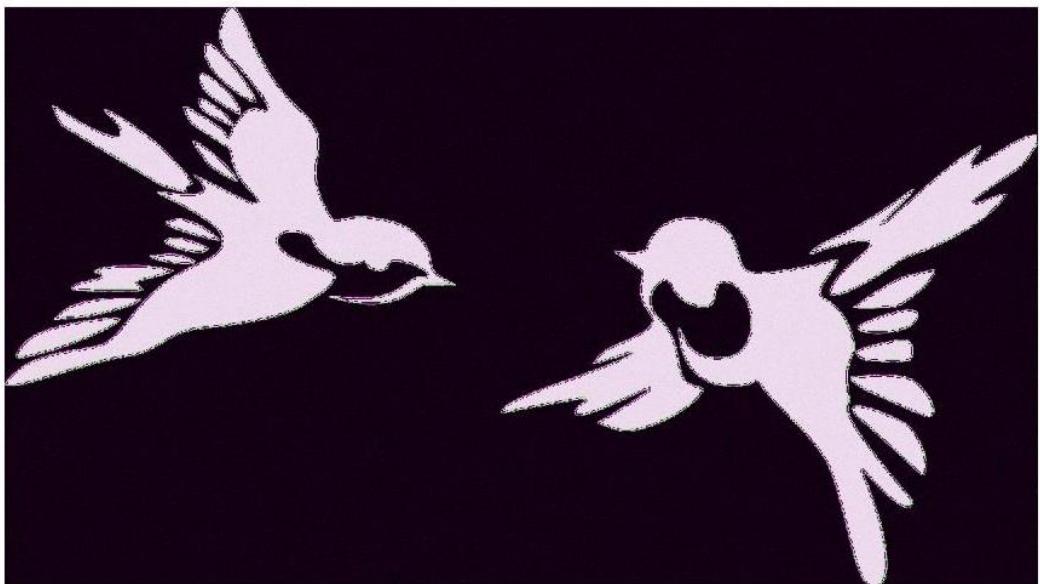


Рисунок 87. – Результат регистрации

16. Исследование срезов/проекций из 3-мерного набора данных МРТ

Данный пример показывает, как анализировать данные из 3-мерного набора МРТ используя функции imtransform и tformarray.

Шаг 1: загрузка и просмотр горизонтального МРТ.

Этот пример использует данные МРТ, которые идут вместе с MATLAB и которые используются как обучающие примеры для montage и immovie. Загрузка mri.mat добавляет две переменные в рабочую область: D (128-by-128-by-1-by-27, class uint8) и grayscale colormap, map (89-by-3, class double).

D включает в себя 27 128-by-128 изображений горизонтальных срезов человеческого черепа, просканированного с помощью МРТ. Величины в D имеют значения от 0 до 88, поэтому цветовая карта (colormap) необходима для создания фигуры с удобным визуальным диапазоном. D является совместимой с montage из-за своей размерности. Первые две размерности являются пространственными, третья - цветовое пространство. Size(D, 3) будет равно 3 для RGB последовательности изображений. Четвертое измерение - временное, но в нашем случае оно тоже является пространственным. Таким образом, мы имеем три пространственных измерения в D и мы можем использовать imtransform или tformarray чтобы конвертировать горизонтальные срезы в сагиттальные (со стороны) или в коронарные (фронтальные) срезы (вид спереди или сзади головы).

Пространственные измерения упорядочены следующим образом:

- измерение 1: от передней к задней части головы;
- измерение 2: с левой части головы к правой;
- измерение 4: с нижней части к верхней.

Важный момент заключается в том, что интервалы дискретизации разные вдоль этих трех измерений: вдоль вертикального измерения расстояние в 2.5 раза больше, чем вдоль горизонтального измерения.

Загрузим данные МРТ, чтобы увидеть горизонтальные срезы (27 штук).

```
load mri;
montage(D,map)
title('Horizontal Slices');
```

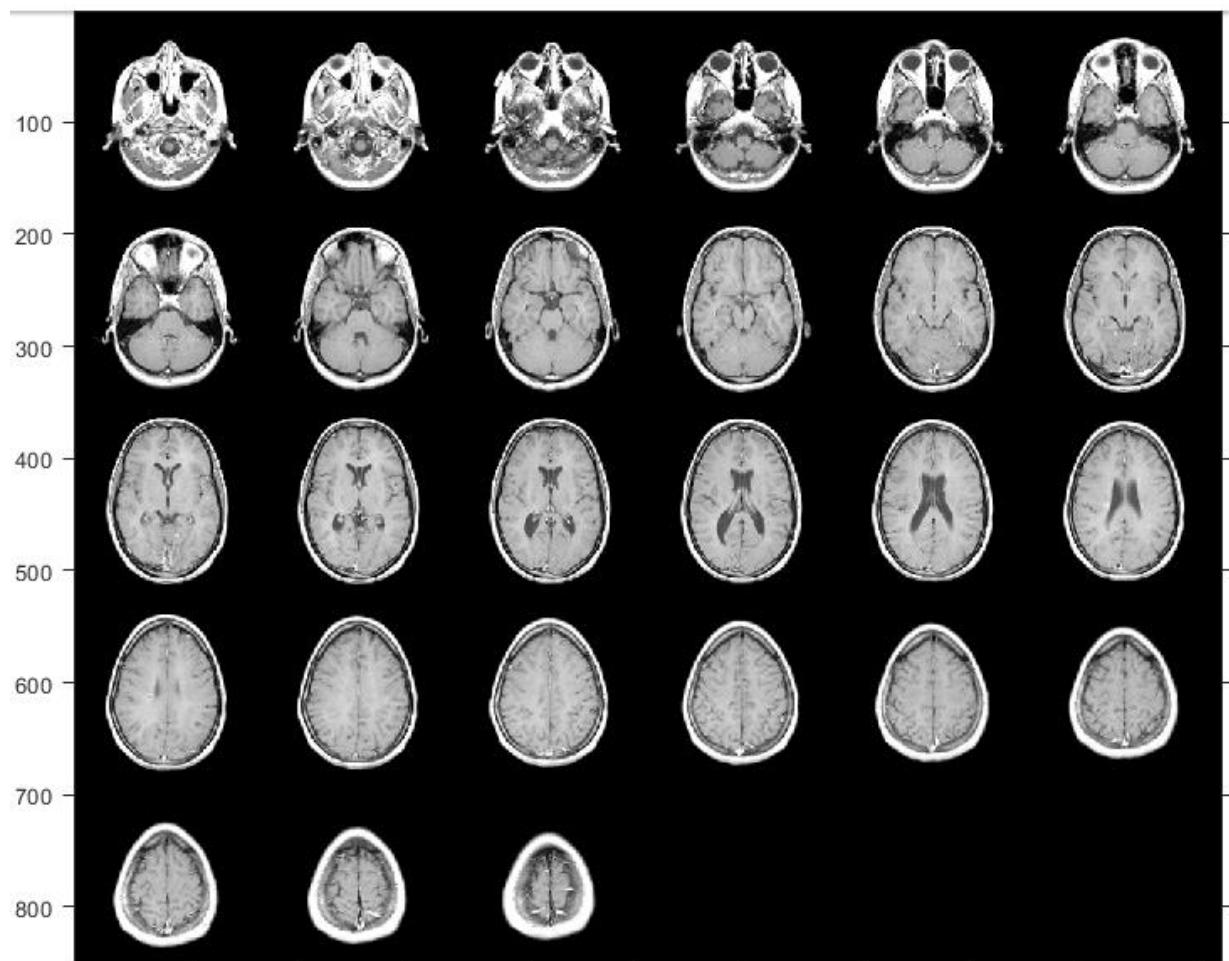


Рисунок 88. –Изображение МРТ человеческого мозга

Шаг 2: извлечение сагиттальных срезов из горизонтальных с помощью IMTRANSFORM.

Мы можем сформировать сагиттальный срез из данных МРТ извлекая подмножество из D и трансформируя его в набор разных интервалов и пространственной ориентации измерений D.

Следующее выражение извлекает всю нужную информацию для среднесагиттального среза.

```
M1 = D(:,64,:,:); size(M1)
ans = 1x4
128      1      1      27
```

Однако мы не можем просмотреть M1 как изображение, потому что она 128-на-1-на-1-на-27. reshape (или squeeze) может конвертировать M1 в 128-на-27 изображение, которое можно просмотреть с помощью imshow.

```
M2 = reshape(M1,[128 27]); size(M2)
ans = 1x2
128      27
figure, imshow(M2,map);
title('Sagittal - Raw Data');
```

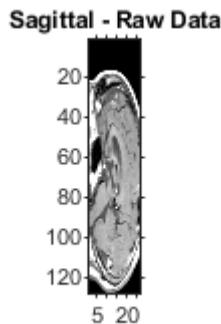


Рисунок 89. - Среднесагиттальный срез

Измерения в M2 упорядочены следующим образом:

- измерение 1: от передней к задней части головы;
- измерение 2: с нижней части к верхней.

Мы можем добиться более приемлемого вида, трансформируя M2, чтобы изменить его ориентацию и увеличить дискретизацию вдоль вертикального измерения в 2.5 раза. Мы можем сделать это поэтапно,

начиная с транспонирования, но аффинное преобразование более экономично в использовании памяти и к тому же позволяет сделать это в один шаг.

```
T0 = maketform('affine',[0 -2.5; 1 0; 0 0]);
```

Матрица 2 на 2, [0 -2.5; 1 0], сочетает в себе вращение и масштабирование.

После преобразования мы имеем:

- измерение 1: с верхней части к нижней;
- измерение 2: от передней к задней части головы.

Вызыва imtransform(M2,T0,'cubic') было бы достаточно, чтобы применить т. к. M2 и обеспечить хорошее разрешение во время интерполяции вдоль верхнего и нижнего направления. Однако, кубическая интерполяция не нужна в направлении от передней к задней части головы, т. к. нет изменения количества отсчетов вдоль 2-го измерения. Поэтому мы определяем ближайшую повторную выборку в этом измерении с более высокой эффективностью и идентичными результатами.

```
R2 = makeresampler({'cubic','nearest'},'fill');  
M3 = imtransform(M2,T0,R2);  
Figure, imshow(M3,map);  
Title('Sagittal - IMTRANSFORM')
```

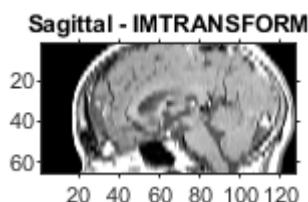


Рисунок 90. - Ближайшая повторная выборка в измерении с более высокой эффективностью и идентичными результатами

Шаг 3: извлечение сагиттальных срезов из горизонтальных с помощью TFORMARRAY.

На этом шаге мы получим такие же результаты, но используя tformarray, дабы перейти от 3-х измерений к 2-м. Шаг два начинался с массива с тремя пространственными измерениями и заканчивался с массивом с двумя пространственными измерениями. Но промежуточные изображения (M_1 , M_2) вынуждают вызывать imtransform, которая создает M_3 . Эти промежуточные изображения не нужны, если мы используем tformarray. imtransform удобно использовать для 2-D в 2-D трансформации, но tformarray поддерживает N-N и M-D преобразования, где M может быть не равно N .

С помощью аргумента TDIMS_A tformarray позволяет нам определить перестановки для входного массива. Так как мы хотим создать изображение с:

- измерением 1: с верхней части к нижней;
- измерением 2: от передней к задней части головы,

и извлечь одну сагиттальную плоскость посредством исходного измерения 2, мы определяем `tdims_a = [4 1 2]`. Мы создаем `tform` с помощью структуры, начинающейся с аффинного преобразования T_1 , которое масштабирует измерение 1 множителем -2.5 и добавляет смещение величиной 68.5 с целью сохранить координаты положительными. Вторая часть композиции - это преобразование T_2 , которое извлекает 64-ую сагиттальную плоскость, используя простую функцию INVERSE_FCN.

```
T1 = maketform('affine',[ -2.5 0; 0 1; 68.5 0]);
inverseFcn = @(X,t) [X repmat(t.tdata,[size(X,1)1])];
T2 = maketform('custom',3,2,[],inverseFcn,64);
Tc = maketform('composite',T1,T2);
```

Заметим, что T_2 и T_c преобразует 3-D входные данные в 2-D.

Мы используем такой же подход, но включая третье измерение.

```
R3 = makeresampler({'cubic','nearest','nearest'},'fill');
```

tformarray преобразует три пространственных измерений D в два измерения в один шаг. Мы имеем 66-на-128 выходное изображение с 27-ю исходными плоскостями расширенных до 66 в вертикальном направлении.

```
M4 = tformarray(D,Tc,R3,[4 1 2],[1 2],[66 128],[],0);
```

Результат идентичен предыдущему.

```
figure, imshow(M4,map);
title('Sagittal - TFORMARRAY');
```

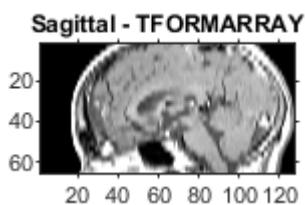


Рисунок 91. - Сагиттальных срезов из горизонтальных с помощью TFORMARRAY

Шаг 4: создание и отображение сагиттальных срезов.

Мы создаем четырехмерный массив, который может быть использован при генерации последовательности изображений, которые идут слева направо, начинается с 30 плоскостей и имеет 35 кадров всего.

Преобразованный массив имеет:

- измерение 1: с верхней части к нижней;
- измерение 2: от передней к задней части головы;
- измерение 4: слева направо.

Как и на прошлом шаге, мы меняем местами элементы массива, используя TDIMS_A = [4 1 2], снова преобразуем вертикальное измерение. Наше аффинное преобразование похоже на T1, за исключением того, что мы добавляем третье измерение с новыми элементами с целью отобразить 30, 32, ... 98 в 1, 2, ..., 35. Все это позволяет установить 35 кадр на середину сагиттального среза.

```
T3 = maketform('affine',[-2.5 0 0; 0 1 0; 0 0 0.5; 68.5 0 -14]);
```

Теперь TSIZE_B = [66 128 35] включает в себя 35 кадров в четвертом измерении (слева направо).

```
S = tformarray(D,T3,R3,[4 1 2],[1 2 4],[66 128 35],[],0);
```

Выведем на экран все сагиттальные срезы на одной картинке.

```
S2 = padarray(S,[6 0 0 0],0,'both');  
figure, montage(S2,map)  
title('Sagittal Slices');
```

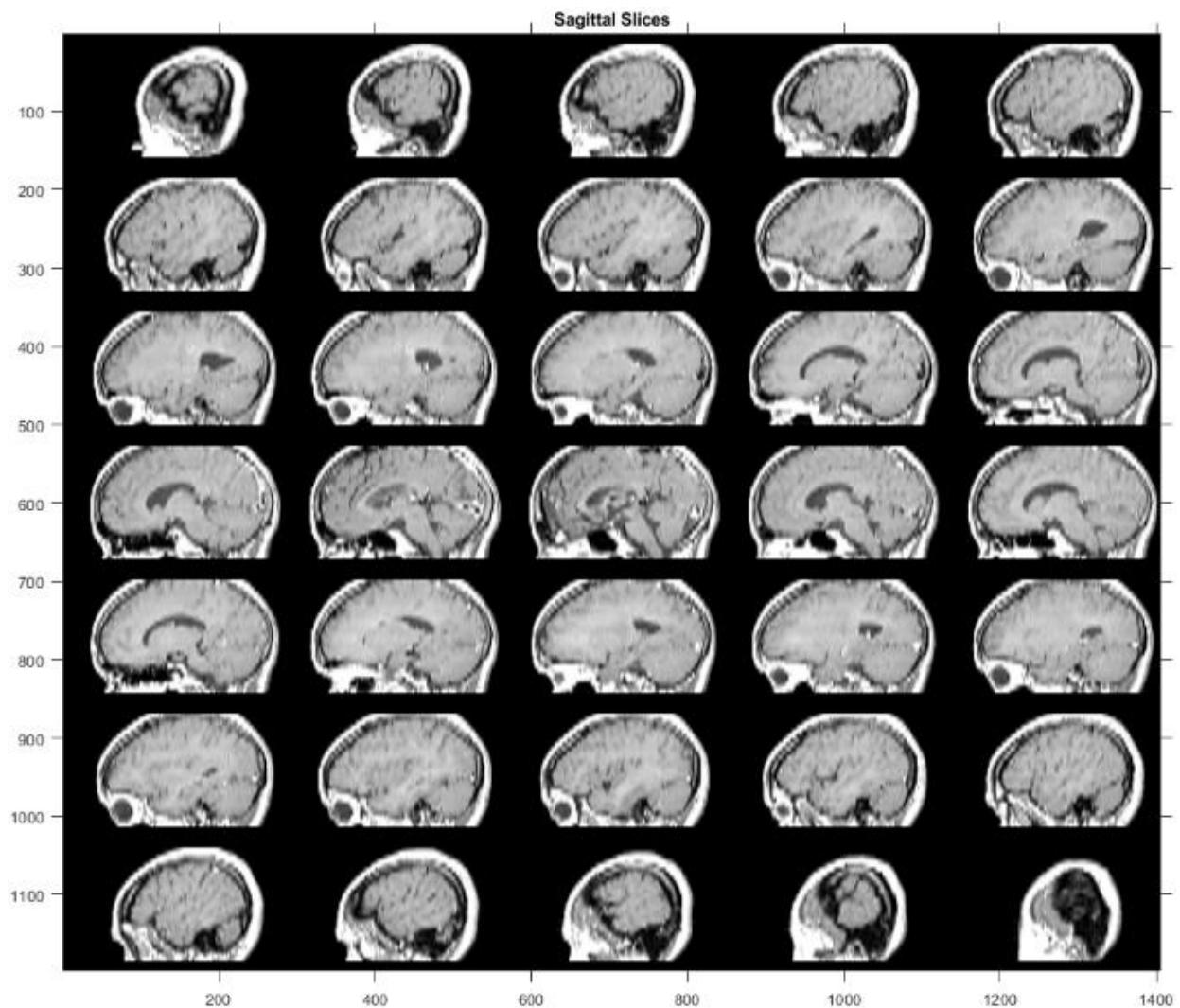


Рисунок 92. – Изображение всех сагиттальных срезов

Шаг 5: создание и отображение коронарных срезов.

Конструирование коронарных срезов схоже с построением сагиттальных срезов. Мы меняем TDIMS_A [4 1 2] на [4 2 1]. Создаем

серию из 45 кадров, начиная с 8 плоскостей, и двигаясь с задней части к передней, пропуская все другие кадры. Измерения выходного массива следующие:

- измерение 1: с верхней части к нижней;
- измерение 2: слева направо;
- измерение 4: от задней к передней части головы.

```
T4 = maketform('affine',[ -2.5 0 0; 0 1 0; 0 0 -0.5; 68.5 0 61]);
```

Вызывая tformarray, TSIZE_B = [66 128 48] определяет вертикальное, боковое и фронтальное измерения соответственно.

```
C = tformarray(D,T4,R3,[4 2 1],[1 2 4],[66 128 45],[],0);
```

Заметим, что все перестановки в массиве, произведенные на шагах 3, 4 и 5, были частью операций функции tformarray

Отобразим коронарные срезы на рисунке.

```
C2 = padarray(C,[6 0 0],0, 'both');  
figure, montage(C2,map)  
title('Coronal Slices');
```

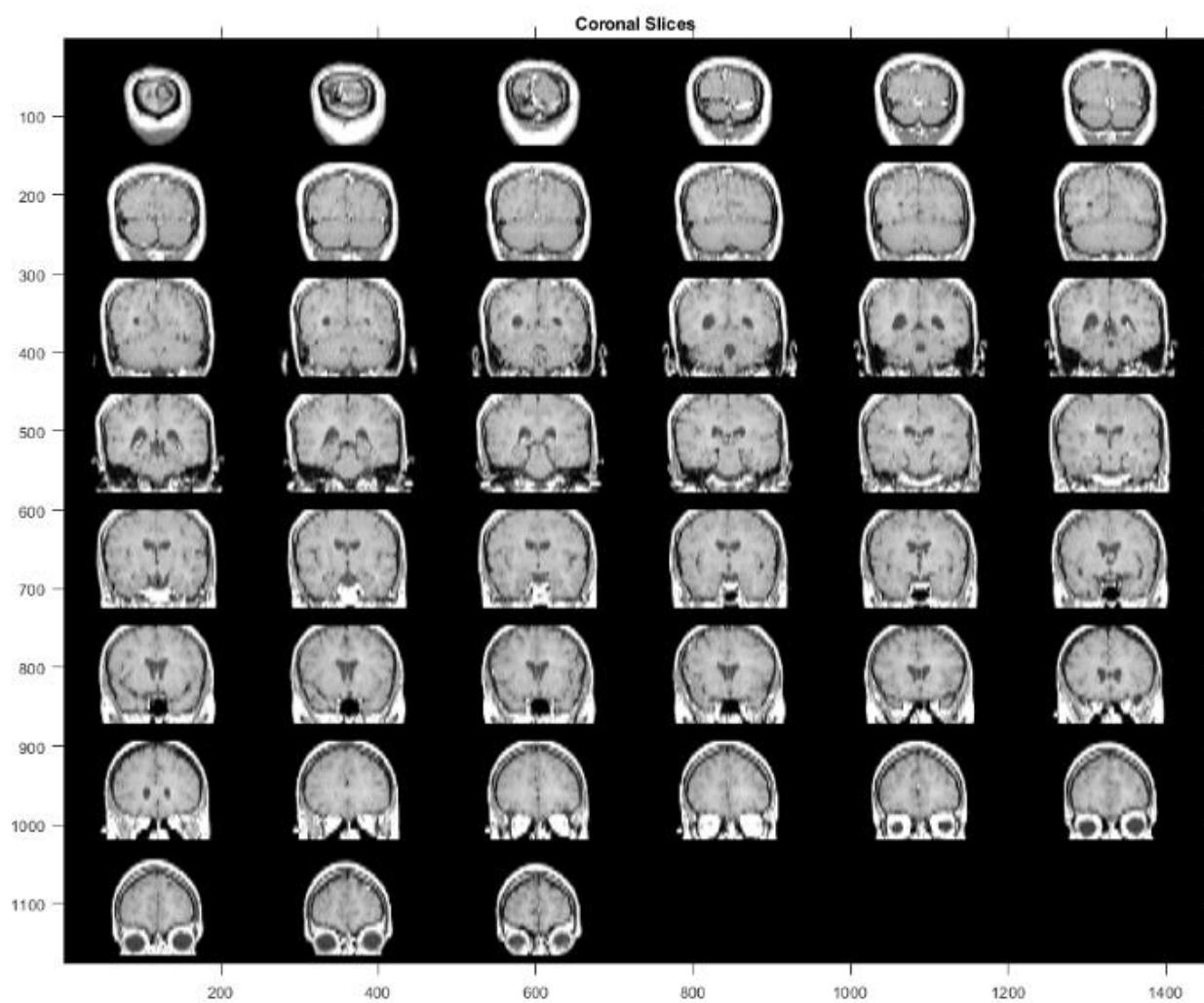


Рисунок 93. – Изображение всех коронарных срезов

17. Обрезание изображений с использованием алгоритма Люси-Ричардсона

В этом примере показано, как использовать алгоритм Люси-Ричардсона для дефлтерации изображений. Его можно эффективно использовать, когда известна функция разнесения по точкам PSF (оператор размытия), но мало информации или нет информации о шуме. Размытое и шумное изображение восстанавливается итеративным, ускоренным, затухающим алгоритмом Люси-Ричардсона. Вы можете использовать характеристики оптической системы в качестве входных параметров для улучшения качества восстановления изображения.

Шаг 1: чтение изображения.

Пример читается в RGB-изображении и отображает его 256-на-256-на-3. Deconvlucy Функция может обрабатывать массивы любой размерности.

```
I = imread('board.tif');
I = I(50+(1:256),2+(1:256),:);
figure;
imshow(I);
title('Original Image');
text(size(I,2),size(I,1)+15, ...
    'Image courtesy of courtesy of Alexander V. Panasyuk, Ph.D.', ...
    'FontSize',7,'HorizontalAlignment','right');
text(size(I,2),size(I,1)+25, ...
    'Harvard-Smithsonian Center for Astrophysics', ...
    'FontSize',7,'HorizontalAlignment','right');
```



Рисунок 94. – Оригинальное изображение

Шаг 2: имитировать размытие и шум.

Имитировать изображение в реальной жизни, которое может быть размыто из-за движения камеры или отсутствия фокуса. Изображение также может быть шумным из-за случайных помех. Пример моделирует размытие, свертывая фильтр Гаусса с истинным изображением (используя `imfilter`). Затем гауссовский фильтр представляет собой функцию с расширением точки PSF.

```
PSF = fspecial('gaussian',5,5);
Blurred = imfilter(I,PSF,'symmetric','conv');
figure;
imshow(Blurred);
title('Blurred');
```

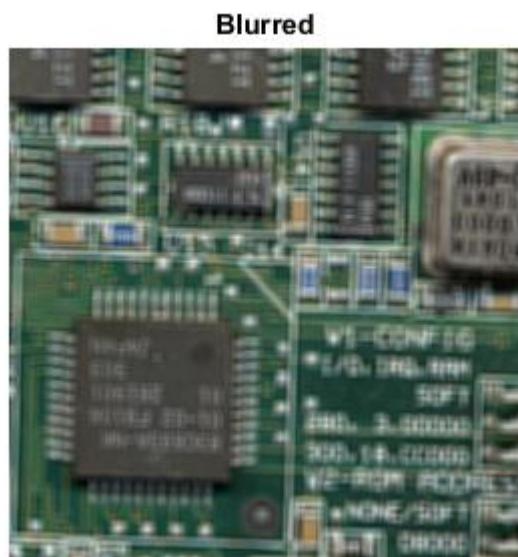


Рисунок 95. – Размытое изображение

В примере имитируется шум, добавляя гауссовский шум дисперсии к размытому изображению (используя `imnoise`). Дисперсия шума используется позже для определения параметра демпфирования алгоритма.

```
V = .002;  
BlurredNoisy = imnoise(Blurred, 'gaussian', 0, V);  
figure;  
imshow(BlurredNoisy);  
title('Blurred & Noisy');
```

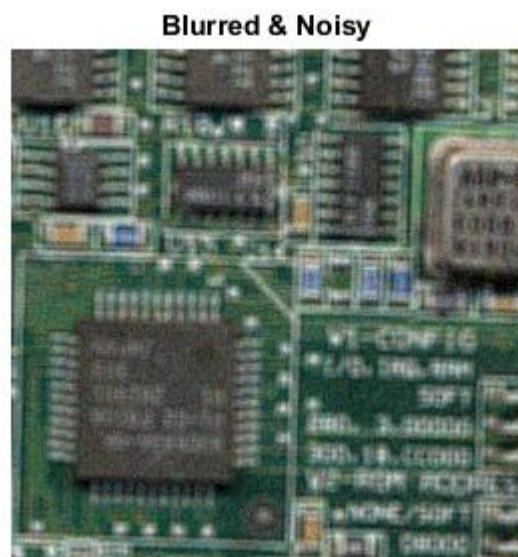


Рисунок 96. – Размытое изображение с добавлением гауссовского шума

Шаг 3: восстановление размытого и шумного изображения.

Восстановите размытое и шумное изображение, обеспечивающее PSF и используя только 5 итераций (по умолчанию 10). Вывод представляет собой массив того же типа, что и входное изображение.

```
luc1 = deconvlucy(BlurredNoisy,PSF,5);  
figure;  
imshow(luc1);  
title('Restored Image, NUMIT = 5');
```

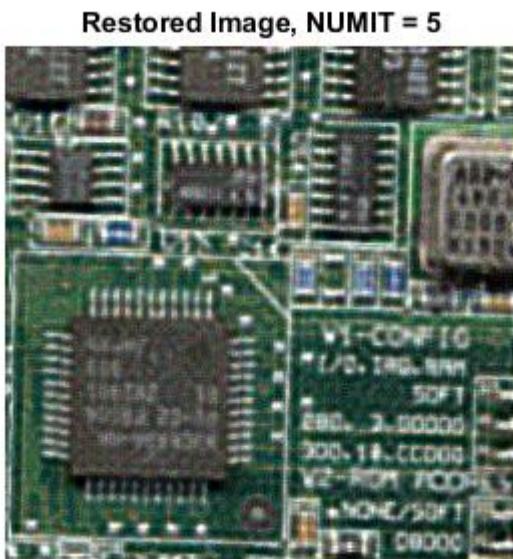


Рисунок 97. – Востановление рисунка путем пяти итераций

Шаг 4: итерация, чтобы исследовать восстановление.

Полученное изображение изменяется с каждой итерацией. Чтобы исследовать эволюцию восстановления изображения, вы можете выполнить деконволюцию поэтапно: выполните набор итераций, посмотрите результат, а затем возобновите итерации с того места, где они были остановлены. Для этого входное изображение должно быть передано как часть массива ячеек. Например, запустите первый набор итераций, передав {BlurredNoisy} вместо BlurredNoisy параметра входного изображения.

```
luc1_cell = deconvlucy({BlurredNoisy},PSF,5);
```

В этом случае вывод luc1_cell, становится массивом ячеек. Выход ячейки состоит из четырех числовых массивов, где первый - это

BlurredNoisy изображение, второе - восстановленное изображение класса double, третий массив - результат одноименной итерации, а четвертый - внутренний параметр итерированный набор. Второй числовой массив выходного массива ячеек, изображение luc1_cell{2}, идентичен выходному массиву Шаг 3, изображения luc1, за исключением исключения их класса (вывод ячейки всегда дает восстановленное изображение класса double).

Чтобы возобновить итерации, сделайте вывод из предыдущего вызова функции, массива ячеек luc1_cell и передайте его в deconvlucy функцию. Используйте число итераций по умолчанию (NUMIT = 10). Восстановленное изображение является результатом 15 итераций.

```
luc2_cell = deconvlucy(luc1_cell,PSF);
luc2 = im2uint8(luc2_cell{2});
figure;
imshow(luc2);
title('Restored Image, NUMIT = 15');
```

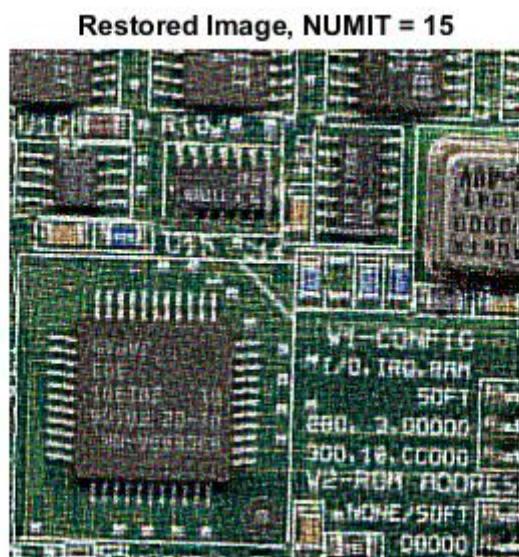


Рисунок 98. – Восстановленное изображение
в результате 15 итераций

Шаг 5: усиление контроля шума путем демпфирования.

Последнее изображение, luc2 является результатом 15 итераций. Хотя он более резкий, чем предыдущий результат из 5 итераций, изображение развивается «крапчатым» внешним видом. Пятна не соответствуют каким-либо реальным структурам (сравнивают их с истинным изображением), но вместо этого являются результатом слишком тонкого подбора шума в данных.

Чтобы контролировать усиление шума, используйте параметр демпфирования, указав DAMPAR параметр. DAMPAR должен быть того же класса, что и входное изображение. Алгоритм гасит изменения в модели в регионах, где различия малы по сравнению с шумом. Обратите внимание, что изображение более плавное.

```
DAMPAR = im2uint8(3*sqrt(V));
luc3 = deconvlucy(BlurredNoisy,PSF,15,DAMPAR);
figure;
imshow(luc3);
title('Restored Image with Damping, NUMIT = 15');
```

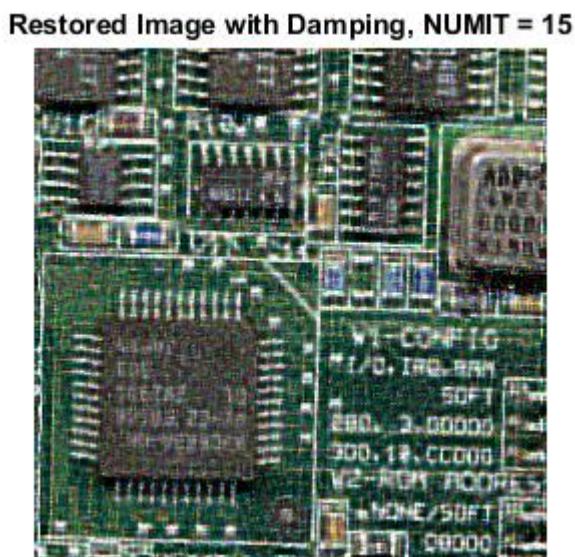


Рисунок 99. – Изображение с меньшим шумом, используя параметр демпфирования

Следующая часть этого примера исследует WEIGHT и SUBSMPL входные параметры функции deconvlucy, используя имитацию звезды изображения (для простоты и скорости).

Шаг 6: создание образца изображения.

В примере создается черно-белое изображение четырех звезд.

```
I = zeros(32);
I(5,5) = 1;
I(10,3) = 1;
I(27,26) = 1;
I(29,25) = 1;
figure;
imshow(1-I,[],'InitialMagnification','fit');
ax = gca;
ax.Visible = 'on';
ax.XTickLabel = [];
ax.YTickLabel = [];
ax.XTick = [7 24];
ax.XGrid = 'on';
ax.YTick = [5 28];
ax.YGrid = 'on';
title('Data');
```

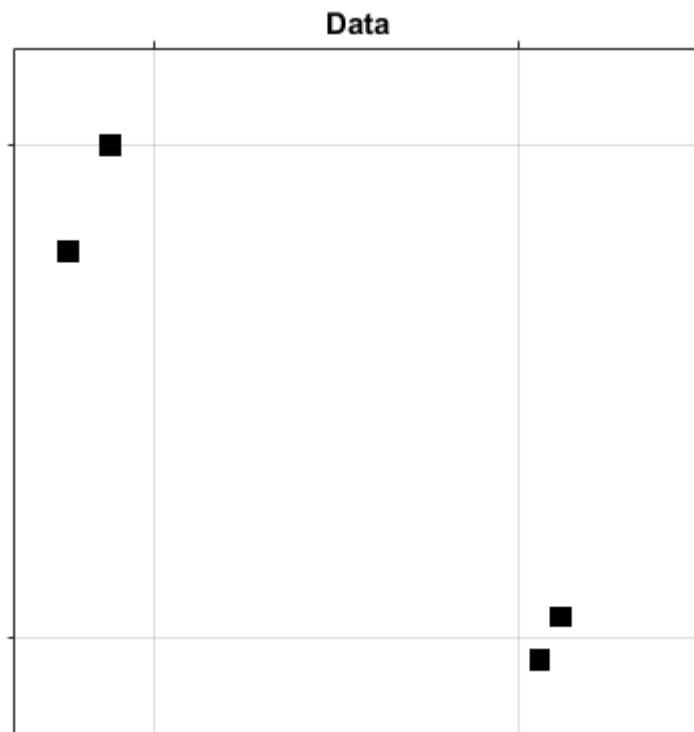


Рисунок 100. – Образ изображения

Шаг 7: моделирование размытия.

Пример имитирует размытие изображения звезд, создавая гауссовский фильтр PSF и свертывая его с истинным изображением.

```
PSF = fspecial('gaussian',15,3);
Blurred = imfilter(I,PSF,'conv','sym');
```

Теперь смоделируйте камеру, которая может наблюдать только часть изображений звезд (видно только размытие). Создайте массив весовых функций, WEIGHT, который состоит из единиц в центральной части Размытого изображения («хорошие» пиксели, расположенные в пунктирных линиях) и нулей по краям («плохие» пиксели - те, которые не принимают сигнал).

```
WT = zeros(32);
WT(6:27,8:23) = 1;
CutImage = Blurred.*WT;
```

Чтобы уменьшить количество звонков, связанных с границами, примените функцию edgetaper с данным PSF.

```
CutEdged = edgetaper(CutImage,PSF);
figure;
imshow(1-CutEdged,[],'InitialMagnification','fit');
ax = gca;
ax.Visible = 'on';
ax.XTickLabel = [];
ax.YTickLabel = [];
ax.XTick = [7 24];
ax.XGrid = 'on';
ax.YTick = [5 28];
ax.YGrid = 'on';
title('Observed');
```

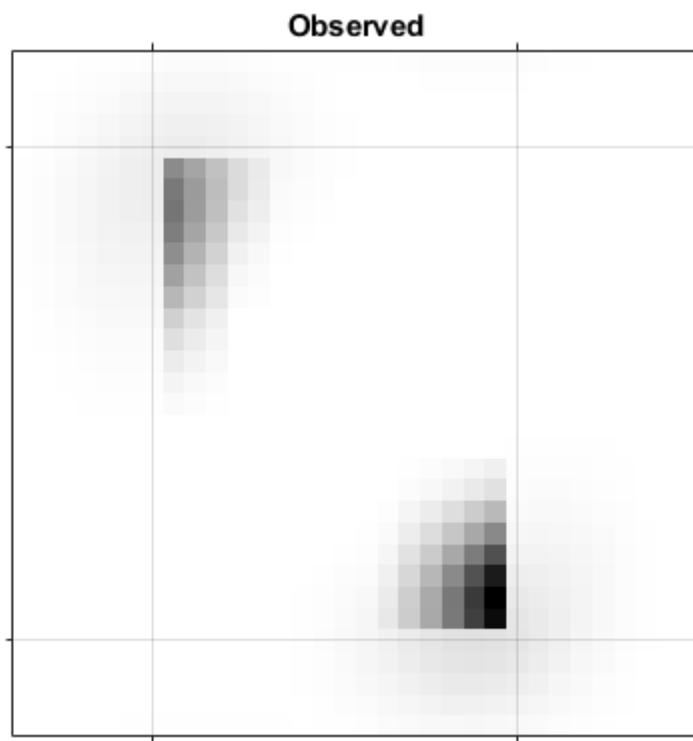


Рисунок 101. – Моделирование размытия

Шаг 8: предоставьте массив BECA.

Алгоритм взвешивает каждое значение пикселя в соответствии с массивом WEIGHT при восстановлении изображения. В нашем примере используются только значения центральных пикселей (где WEIGHT = 1), а «плохие» значения пикселей исключены из оптимизации. Однако алгоритм может помещать мощность сигнала в положение этих «плохих» пикселей, выходящих за пределы камеры. Обратите внимание на точность разрешенных позиций звезды.

```

luc4 = deconvlucy(CutEdged,PSF,300,0,WT);
figure;
imshow(1-luc4,[],'InitialMagnification','fit');
ax = gca;
ax.Visible = 'on';
ax.XTickLabel = [];
ax.YTickLabel = [];
ax.XTick = [7 24];
ax.XGrid = 'on';
ax.YTick = [5 28];

```

```
ax.YGrid = 'on';
title('Restored');
```

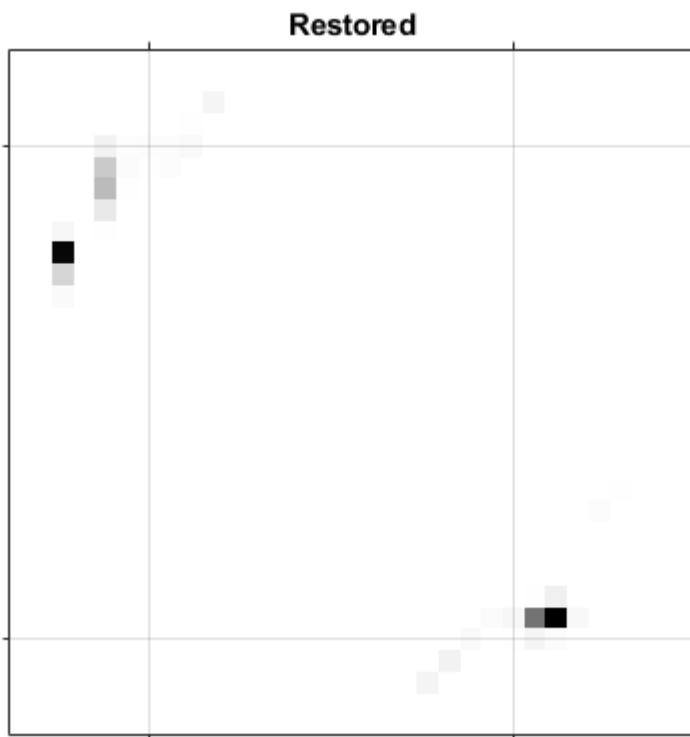


Рисунок 102. – Востановленное изображение
в соответствии с массивом WEIGHT

Шаг 9: предоставьте более тонкий PSF.

deconvlucy может восстанавливать недокадровое изображение с учетом более тонкой выборки PSF (более тонкой по времени SUBSMPL). Чтобы имитировать плохо разрешенное изображение и PSF, пример объединяет Blurred изображение и исходный PSF, по два пикселя в одном, в каждом измерении.

```
Binned = squeeze(sum(reshape(Blurred,[2 16 2 16])));
BinnedImage = squeeze(sum(Binned,2));
Binned = squeeze(sum(reshape(PSF(1:14,1:14),[2 7 2 7])));
BinnedPSF = squeeze(sum(Binned,2));
figure;
imshow(1-BinnedImage,[],'InitialMagnification','fit');
ax = gca;
ax.Visible = 'on';
ax.XTick = [];
```

```
ax.YTick = [];
title('Binned Observed');
```

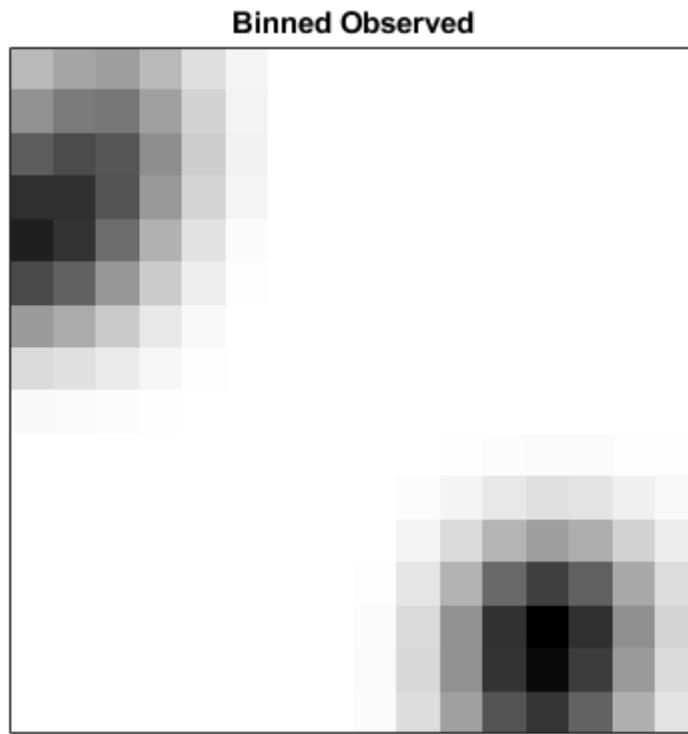


Рисунок 103. – Восстановленное недокадровое изображение
с учетом выборки PSF

Восстановите недокадровое изображение, BinnedImage используя PSS без дискретизации BinnedPSF. Обратите внимание, что luc5 изображение отличает только 3 звезды.

```
luc5 = deconvlucy(BinnedImage,BinnedPSF,100);
figure;
imshow(1-luc5,[],'InitialMagnification','fit');
ax = gca;
ax.Visible = 'on';
ax.XTick = [];
ax.YTick = [];
title('Poor PSF');
```

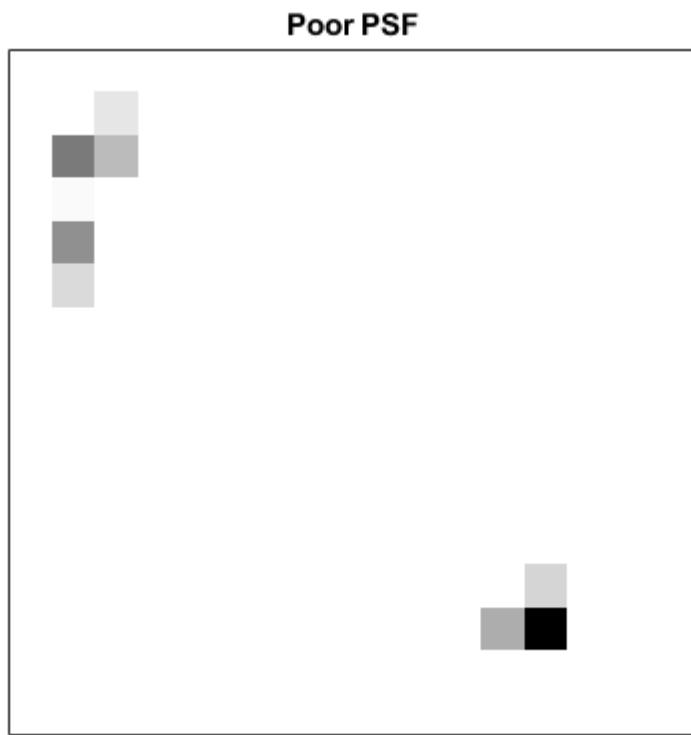


Рисунок 104. – Восстановленное недокадровое изображение, BinnedImage используя PSS без дискретизации BinnedPSF

Следующий пример восстанавливает недокадровое изображение (BinnedImage), на этот раз используя более тонкий PSF (определенный в сетке SUBSMPL-times finer). Реконструированное изображение (luc6) решает положение звезд более точно. Обратите внимание, как он распределяет энергию между двумя звездами в правом нижнем углу изображения. Это намекает на существование двух ярких объектов, а не одного, как в предыдущем восстановлении.

```
luc6 = deconvlucy(BinnedImage,PSF,100,[],[],[],2);
figure;
imshow(1-luc6,[],'InitialMagnification','fit');
ax = gca;
ax.Visible = 'on';
ax.XTick = [];
ax.YTick = [];
title('Fine PSF');
```

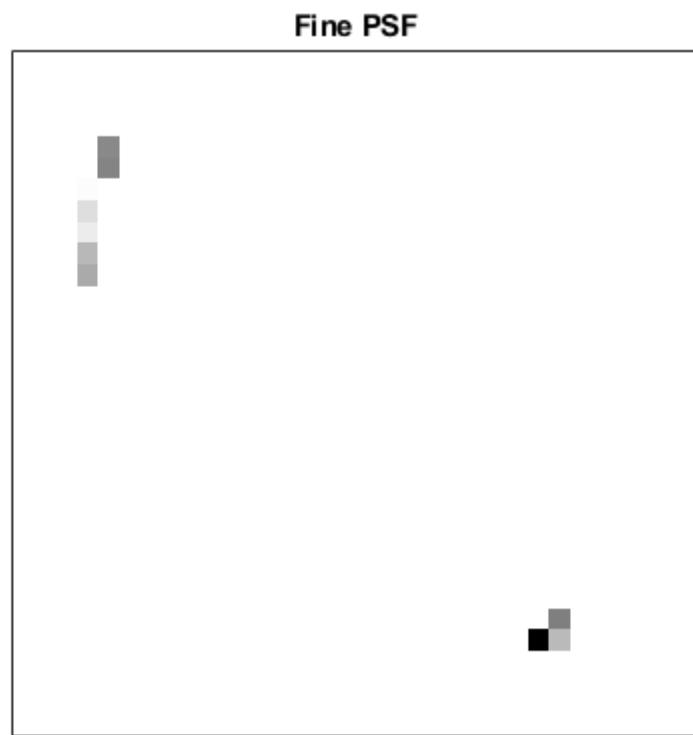


Рисунок 105. – Восстановленное недокадровое изображение, используя
более тонкий PSF (определенный в сетке SUBSMPL-times finer)

18. Обрезание изображений с использованием алгоритма слепой деконволюции

В этом примере показано, как использовать скрытую деконволюцию для деблок изображений. Алгоритм слепой деконволюции можно эффективно использовать, когда информация об искажении (размытость и шум) не известна. Алгоритм восстанавливает изображение и функцию распределения точек (PSF) одновременно. Ускоренный, затухающий алгоритм Ричардсона-Люси используется на каждой итерации. Дополнительные параметры оптической системы (например, камеры) могут использоваться в качестве входных параметров, которые могут помочь улучшить качество восстановления изображения. Ограничения PSF могут быть переданы через пользовательскую функцию.

Шаг 1: чтение изображения.

Пример читается на изображении интенсивности. Deconvblind Функция может обрабатывать массивы любой размерности.

```
I = imread('cameraman.tif');
figure;imshow(I);title('Original Image');
text(size(I,2),size(I,1)+15, ...
'Image courtesy of Massachusetts Institute of Technology', ...
'FontSize',7,'HorizontalAlignment','right');
```



Рисунок 106. – Оригинальное изображение

Шаг 2: имитация размытия.

Имитировать изображение в реальной жизни, которое может быть размыто (например, из-за движения камеры или отсутствия фокуса). Пример моделирует размытие, свертывая фильтр Гаусса с истинным изображением (используя `imfilter`). Затем гауссовский фильтр представляет собой функцию с расширением точки PSF.

```
PSF = fspecial('gaussian',7,10);
Blurred = imfilter(I,PSF,'symmetric','conv');
figure;imshow(Blurred);title('Blurred Image');
```



Рисунок 107. – Изображение, размытое по Гауссу

Шаг 3: восстановление размытого изображения с использованием PSF различных размеров.

Чтобы проиллюстрировать важность знания размера истинного PSF, этот пример выполняет три реставрации. Каждый раз, когда восстановление PSF начинается с однородного массива (массив из нулей).

Первое восстановление J1 и P1 использует недоразмерный массив UNDERPSF для первоначальной догадки PSF. Размер массива UNDERPSF на 4 пикселя меньше в каждом измерении, чем истинный PSF.

```
UNDERPSF = ones(size(PSF)-4);  
[J1, P1] = deconvblind(Blurred,UNDERPSF);  
figure;imshow(J1);title('Deblurring with Undersized PSF');
```

Deblurring with Undersized PSF



Рисунок 108. – Размытое изображение с недооценкой PSF

Второе восстановление J2и P2использует массив из них OVERPSF для начального PSF, который на 4 пикселя длиннее в каждом измерении, чем истинный PSF.

```
OVERPSF = padarray(UNDERPSF,[4 4],'replicate','both');  
[J2, P2] = deconvblind(Blurred,OVERPSF);  
figure;imshow(J2);title('Deblurring with Oversized PSF');
```

Deblurring with Oversized PSF



Рисунок 109. – Размытое изображение с небольшим PSF

Третье восстановление J3 и P3 использует массив из них INITPSF для начального PSF, который имеет точно такой же размер, что и истинный PSF.

```
INITPSF = padarray(UNDERPSF,[2 2],'replicate','both');  
[J3, P3] = deconvblind(Blurred,INITPSF);  
figure;imshow(J3);title('Deblurring with INITPSF');
```



Рисунок 110. – Размытое изображение с INITPSF

Шаг 4: анализ восстановленного PSF.

Все три реставрации также создают PSF. На следующих рисунках показано, как анализ восстановленного PSF может помочь угадать правильный размер для исходного PSF. В истинном PSF, гауссовском фильтре, максимальные значения находятся в центре (белый) и уменьшаются на границах (черный).

```
figure;  
subplot(221);imshow(PSF,[],'InitialMagnification','fit');  
title('True PSF');  
subplot(222);imshow(P1,[],'InitialMagnification','fit');  
title('Reconstructed Undersized PSF');  
subplot(223);imshow(P2,[],'InitialMagnification','fit');  
title('Reconstructed Oversized PSF');  
subplot(224);imshow(P3,[],'InitialMagnification','fit');  
title('Reconstructed true PSF');
```

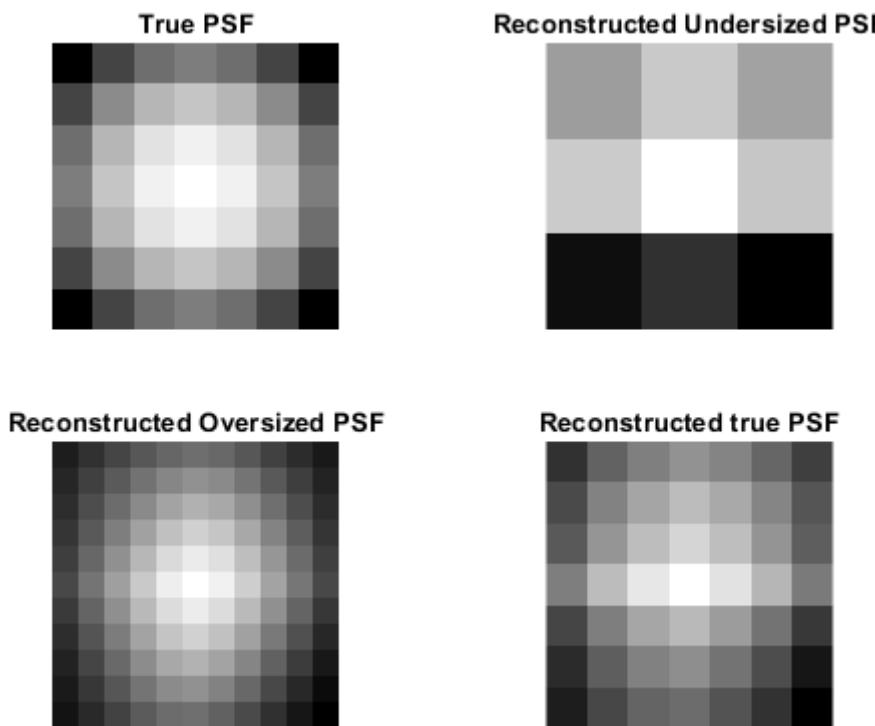


Рисунок 111. – Анализ восстановленного PSF

PSF, восстановленный при первом восстановлении, Р1 явно не вписывается в ограниченный размер. Он имеет сильное изменение сигнала на границах. Соответствующее изображение, J1 не показывает улучшенной ясности по сравнению с размытым изображением Blurred.

PSF, восстановленный во втором восстановлении Р2, становится очень гладким по краям. Это означает, что восстановление может обрабатывать PSF меньшего размера. Соответствующее изображение, J2 показывает некоторое затухание, но оно сильно повреждено звонком.

Наконец, PSF, восстановленный в третьем восстановлении Р3, является несколько промежуточным между Р1 и Р2. Массив, очень Р3 похож на истинный PSF. Соответствующий образ J3 показывает значительное улучшение; однако он все еще поврежден звоном.

Шаг 5: улучшение восстановления.

Шум в восстановленном изображении, J3 происходит вдоль областей резкого контраста интенсивности изображения и вдоль границ изображения. В этом примере показано, как уменьшить эффект шума, указав функцию взвешивания. Алгоритм взвешивает каждый пиксель в соответствии с WEIGHT массивом при восстановлении изображения и PSF. В нашем примере мы начинаем с поиска «острых» пикселей с помощью функции edge. По результатам проб и ошибок мы определяем, что желаемый пороговый уровень равен 0,3.

```
WEIGHT = edge(Blurred,'sobel',.08);
```

Чтобы расширить область, мы используем imdilate и передаем элемент структурирования se.

```
se = strel('disk',2);
WEIGHT = 1-double(imdilate(WIGHT,se));
WEIGHT([1:3 end-(0:2)],:) = 0;
WEIGHT(:,[1:3 end-(0:2)]) = 0;
figure; imshow(WIGHT); title('Weight array');
```

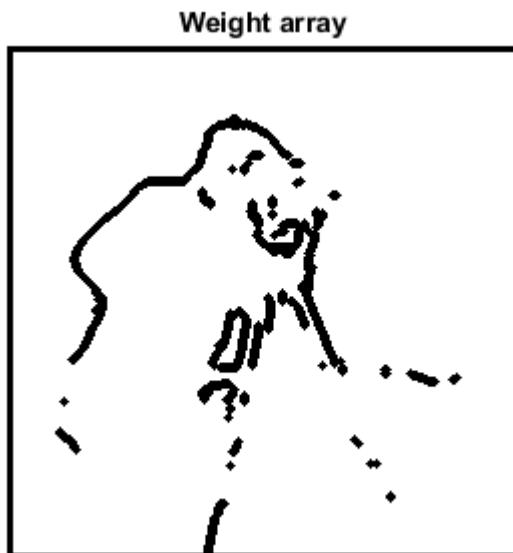


Рисунок 112. – Восстановление изображения алгоритмом взвешивания каждого пикселя в соответствии с WEIGHT массивом и PSF

Изображение восстанавливается вызовом deconvblind с WEIGHT массивом и увеличением количества итераций (30). Почти все шумы подавляются.

```
[J, P] = deconvblind(Blurred, INITPSF, 30, [], WEIGHT);  
figure; imshow(J); title('Deblurred Image');
```



Рисунок 113. – Восстановленное изображение
вызовом deconvblind с WEIGHT массивом

Шаг 6: использование дополнительных ограничений при восстановлении PSF.

В этом примере показано, как вы можете указать дополнительные ограничения для PSF. Функция, FUN приведенная ниже, возвращает модифицированный массив PSF, который deconvblind использует для следующей итерации.

В этом примере FUN изменяется PSF, обрезая его P1 и P2 количество пикселей в каждом измерении, а затем отбрасывая массив обратно до его исходного размера нулями. Эта операция не изменяет значения в центре PSF, но эффективно уменьшает размер PSF $2*P1$ и $2*P2$ пиксели.

```
P1 = 2;  
P2 = 2;
```

```
FUN = @(PSF) padarray(PSF(P1+1:end-P1,P2+1:end-P2),[P1 P2]);
```

Анонимная функция FUN, передается deconvblind последним. Информацию о предоставлении дополнительных параметров функции см. В разделе Параметрирование функций в документации по математике MATLAB FUN.

В этом примере размер исходного PSF, на OVERPSF4 пикселя больше, чем истинный PSF. Установка P1 = 2 и P2 = 2 в качестве FUN эффективных параметров делает ценное пространство OVERPSF того же размера, что и истинный PSF. Следовательно, результат, JF и PF, аналогичен результату деконволюции с правильным размером PSF и без FUN вызова, J и Pc шага 4.

```
[JF, PF] = deconvblind(Blurred,OVERPSF,30,[],WEIGHT,FUN);  
figure;imshow(JF);title('Deblurred Image');
```



Рисунок 114. – Деконволюция изображения с правильным размером PSF

Если бы мы использовали негабаритный исходный PSF, то OVERPSF без ограничивающей функции FUN результирующее изображение было бы похоже на неудовлетворительный результат J2, достигнутый на шаге 3.

19. Измерение угла пересечения

Чтение изображения и рисование стрелок, указывающих на 2 луча представляющих для нас интерес.

```
RGB = imread('balka.png');
imshow(RGB);
text(size(RGB,2),size(RGB,1)+15,'Image courtesy of Jeff Mather',...
'FontSize',7,'HorizontalAlignment','right');
line([300 328],[85 103],'color',[1 1 0]);
line([268 255],[85 140],'color',[1 1 0]);
text(150,72,'Measure the angle between these beams','Color','y',...
'FontWeight', 'bold');
```



Рисунок 115. – Угол между лучами

Извлечение области, в которой мы хотим определить угол.

Обрезаем изображение, чтобы получить только лучи, выбранные нами ранее. Этот шаг облегчит извлечение краев двух металлических балок.

```
start_row = 34;
start_col = 208;
cropRGB = RGB(start_row:163, start_col:400, :);
imshow(cropRGB)
```



Рисунок 116. – Фрагмент изображения для определения угла

Преобразование изображения.

Преобразовываем изображение в черно-белое для последующего извлечения координат ребер с помощью процедуры bwtraceboundary.

```
I = rgb2gray(cropRGB);
BW = imbinarize(I);
BW = ~BW;
imshow(BW)
```



Рисунок 117. – Черно-белый фрагмент изображения

Нахождение начальной точки на каждой границе.

Процедура bwtraceboundary требует указания одной точки на границе. Эта точка используется в качестве начальной для процесса трассировки границы. Чтобы извлечь край нижнего луча, выбираем столбец на изображении и проверяем его, пока не произойдет переход от пикселя фона к пикселию объекта. Сохранить это место для дальнейшего использования в bwtraceboundary. Повторяем эту процедуру для другого луча, но на этот раз трассируем по горизонтали.

```
dim = size(BW);
col1 = 4;
row1 = find(BW(:,col1), 1);
```

```
row2 = 12;
col2 = find(BW(row2,:), 1);
```

Трассировка границ.

Процедура bwtraceboundary используется для извлечения (X, Y) местоположений граничных точек. Для обеспечения максимальной точности расчетов угла и точки пересечения важно извлечь как можно больше точек, принадлежащих кромкам балки. Количество точек следует определять экспериментально. Поскольку начальная точка для горизонтальной полосы была получена путем сканирования с севера на юг, безопаснее всего задать начальный шаг поиска, чтобы указать на внешнюю сторону объекта, т. е. "Север".

```
boundary1 = bwtraceboundary(BW, [row1, col1], 'N', 8, 70);
boundary2 = bwtraceboundary(BW, [row2, col2], 'E', 8, 90,'counter');
imshow(RGB); hold on;
plot(offsetX+boundary1(:,2),offsetY+boundary1(:,1),'g','LineWidth',2)
;
plot(offsetX+boundary2(:,2),offsetY+boundary2(:,1),'g','LineWidth',2)
;
```



Рисунок 118. – Трассировка границ

Приближаем линии к границам.

Хотя пары координат (X, Y) были получены на предыдущем шаге, не все точки лежат точно на прямой. Какие из них следует использовать для вычисления угла и точки пересечения? Предполагая, что все полученные точки одинаково важны, подгоняем линии к местоположениям граничных пикселей. Уравнение для линии $y = [x \ 1]*[a; b]$. Мы можем решить для параметров "a" и "b" в смысле наименьших квадратов, используя оператор polyfit.

```
ab1 = polyfit(boundary1(:,2), boundary1(:,1), 1);
ab2 = polyfit(boundary2(:,2), boundary2(:,1), 1);
```

Находим угол пересечения. Используем линии из точек , чтобы найти угол.

```
vect1 = [1 ab1(1)];
vect2 = [1 ab2(1)];
dp = dot(vect1, vect2);
length1 = sqrt(sum(vect1.^2));
length2 = sqrt(sum(vect2.^2));
angle = 180-acos(dp/(length1*length2))*180/pi
```

Находим точку пересечения.

Решаем систему двух уравнений, чтобы получить (X,Y) координаты точки пересечения.

```
intersection = [1 ,-ab1(1); 1, -ab2(1)] \ [ab1(2); ab2(2)];
intersection = intersection + [offsetY; offsetX]
```

Наносим результаты на изображение.

```
inter_x = intersection(2);
inter_y = intersection(1);
plot(inter_x,inter_y,'yx','LineWidth',2);
text(inter_x-60, inter_y-30, [sprintf('%1.3f',angle),'{\circ}',... 
'Color','y','FontSize',14,'FontWeight','bold']);
interString = sprintf('(%2.1f,%2.1f)', inter_x, inter_y);
text(inter_x-10, inter_y+20, interString,... 
'Color','y','FontSize',14,'FontWeight','bold');
```



Рисунок 119. – Изображение с вычисленными результатами

20. Измерение радиуса рулона ленты

В этом примере показано, как измерить радиус рулона ленты, который частично затенен диспенсером ленты. Используется оператор imfindcircles для выполнения данной задачи.

Чтение изображения.

Чтение изображения и написание на нем названия выполняемой работы.

```
RGB = imread('lenta.png');
imshow(RGB);
hTxt = text(15,15, 'Estimate radius of the roll of
tape','FontWeight','bold','Color','y');
```



Рисунок 120. – Изображение рулона ленты

Нахождение окружности.

Находим центр и радиус окружности на изображении с помощью оператора imfindcircles.

```
Rmin = 60;
Rmax = 100;
```

```
[center, radius] = imfindcircles(RGB,[Rmin Rmax], 'Sensitivity',0.9)
center = 1x2
236.9291 172.4747
radius = 79.5305
```

Выделение круга и его центра.

```
viscircles(center,radius);
hold on;
plot(center(:,1),center(:,2), 'yx', 'LineWidth',2);
hold off;
delete(hTxt);
message = sprintf('The estimated radius is %2.1f pixels', radius);
text(15,15,message, 'Color','y', 'FontWeight','bold');
```



Рисунок 121. – Расчетный радиус ленты 79,5 пикселей

21. Идентификация круглых объектов

Данный пример показывает, как классифицировать объекты на основе их “округлости”, используя bwboundaries - метода отслеживания границ.

Шаг 1: чтение изображения.

```
RGB = imread('pillsetc.png');  
imshow(RGB);
```

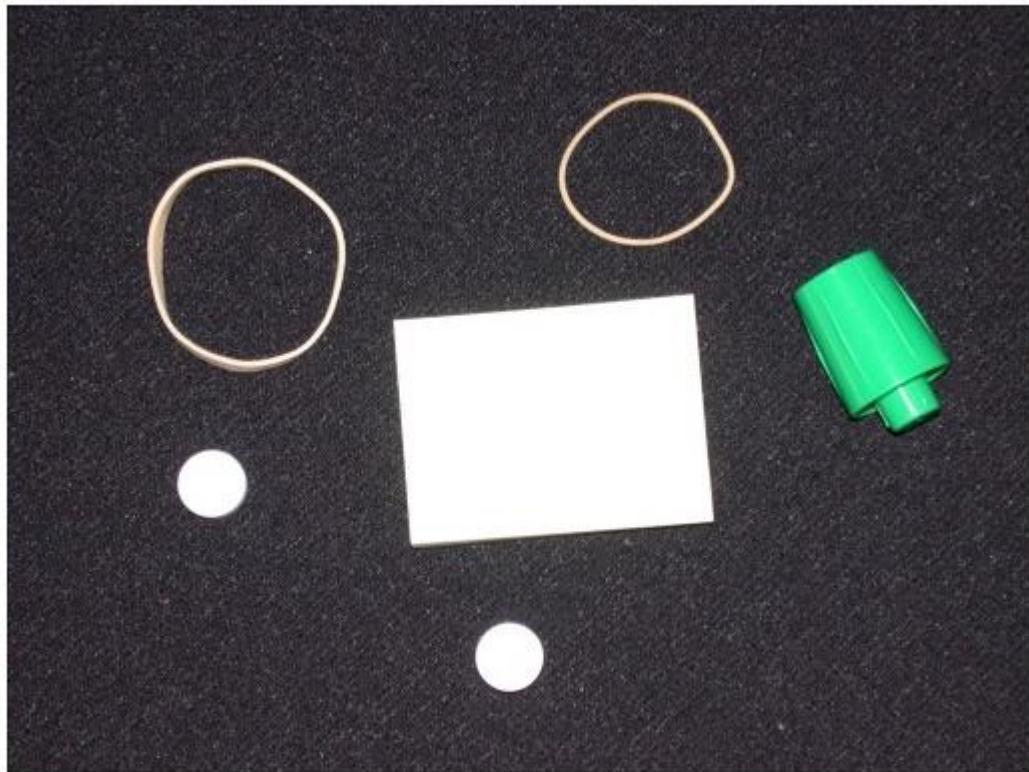


Рисунок 122. – Оригинальное изображение

Шаг 2: подготовка изображения.

Конвертируйте изображение в черно-белый формат, тем самым подготовив его для использования bwboundaries

```
I = rgb2gray(RGB);  
Bw = imbinarize(I);  
imshow(bw)
```

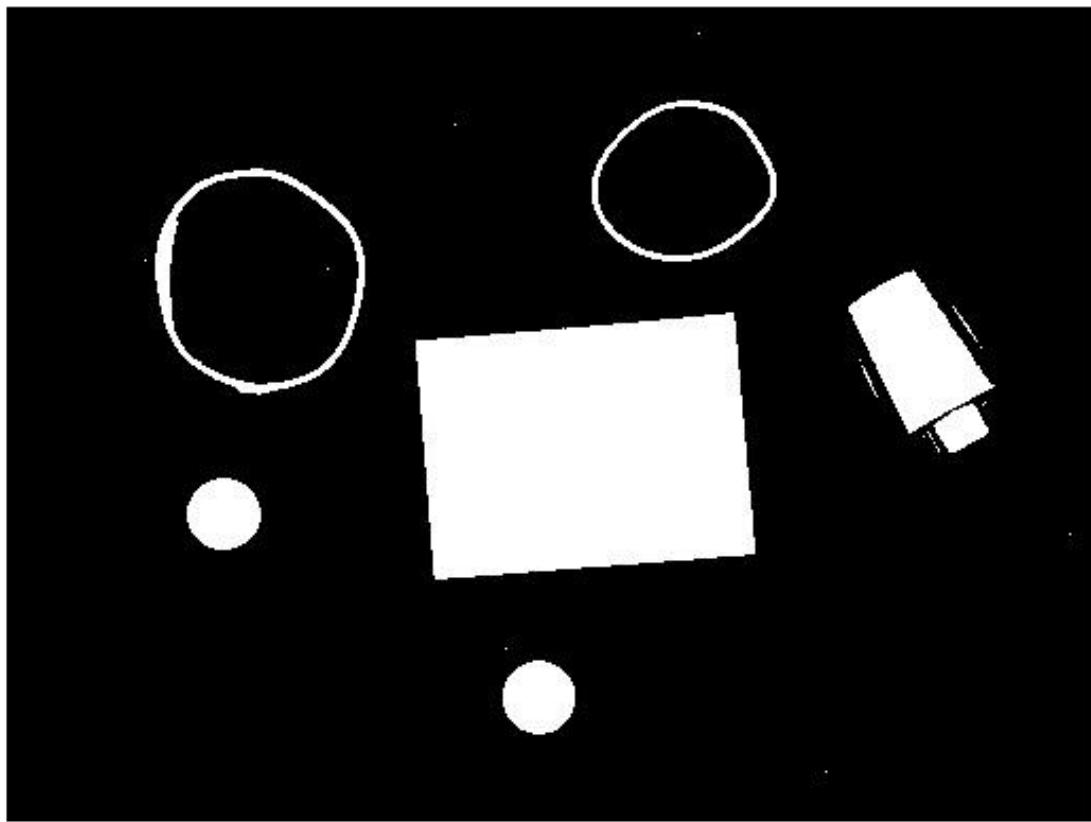


Рисунок 123. – Бинарное изображение

Шаг 3: удаление шума.

Используя морфологические функции, избавимся от пикселей, которые не принадлежат к интересующим нас объектам.

```
%remove all object containing fewer than 30 pixels  
bw = bwareaopen(bw,30);  
  
%fill a gap in the pen's cap  
se = strel('disk',2);  
bw = imclose(bw,se);  
  
%fill any holes, so that regionprops can be used to estimate  
%the area enclosed by each of the boundaries  
bw = imfill(bw, 'holes');  
  
imshow(bw)
```

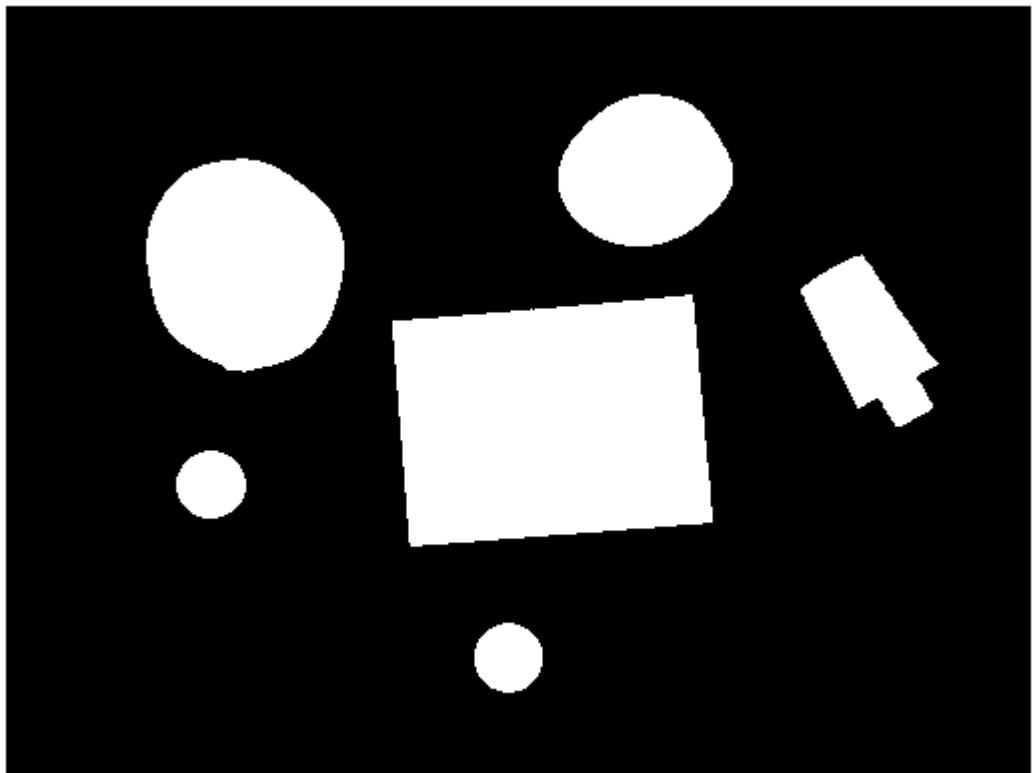


Рисунок 124. – Изображение без шумов

Шаг 4: нахождение границ.

Сфокусируемся только на внешних границах. Опция 'noholes' ускорит процесс обработки, т. к. в этом случае bwboundaries не будет рассматривать внутренние контуры.

```
[B,L] = bwboundaries(bw, 'noholes');

%Display the label matrix and draw each boundary
imshow(label2rgb(L, @jet, [.5 .5 .5]))
hold on
for k = 1:length(B)
    boundary = B{k};
    plot(boundary(:,2), boundary(:,1), 'w', 'LineWidth', 2)
end
```

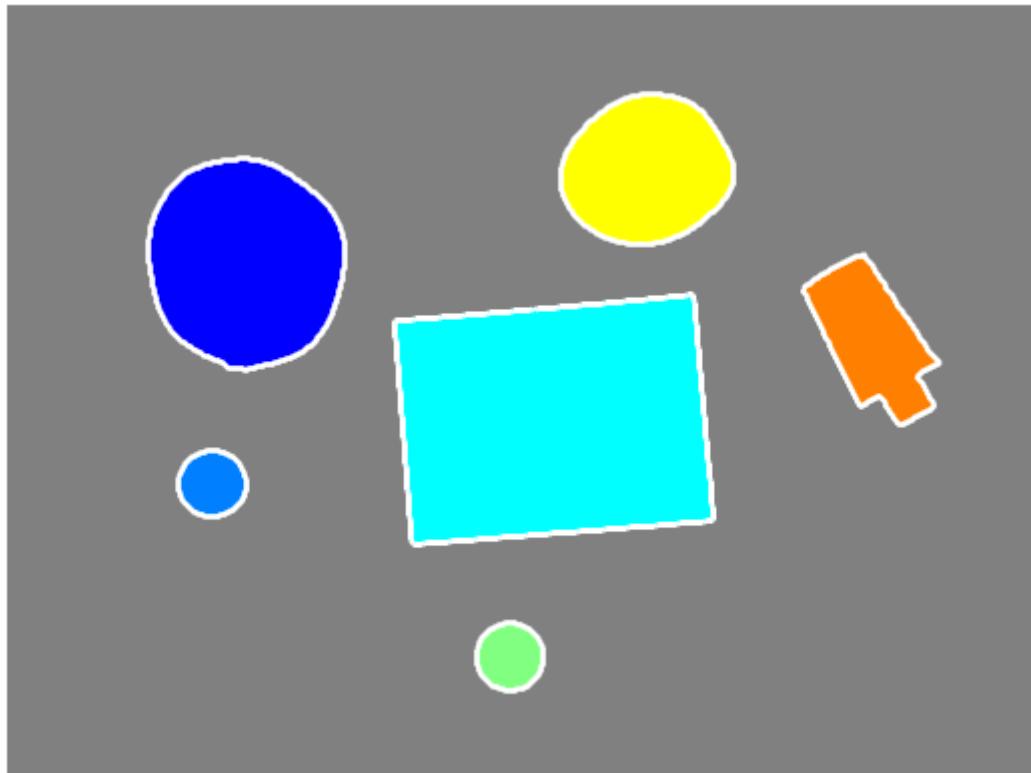


Рисунок 125. – Нахождение границ

Шаг 5: нахождение круглых объектов.

Оценим площадь и периметр каждого объекта. Используем эти данные для формирования показателя, характеризующего округлость объекта:

$$\text{metric} = 4 * \pi * \text{area} / \text{perimeter}^2.$$

Этот показатель равен 1 для круга, для остальных форм он принимает значения меньше единицы. Процесс распознавания может контролироваться выбором подходящего порога. В нашем примере он равен 0.94, поэтому лишь два объекта будут распознаны как окружности.

Используем `regionprops` чтобы получить оценочные значения площадей всех фигур. Заметим, что матрица, возвращаемая функцией `bwboundaries` может быть использована еще раз функцией `regionprops`.

```
stats = regionprops(L, 'Area', 'Centroid');  
threshod = 0.94;
```

```

% loop over the boundaries
for k = 1:length(B)

    % obtain (X,Y) boundary coordinates corresponding to label 'k'
    boundary = B{k};

    % compute a simple estimate of the object's perimeter
    delta_sq = diff(boundary).^2;
    perimeter = sum(sqrt(sum(delta_sq,2)));
    % obtain the area calculation corresponding to label 'k'
    area = stats(k).Area;

    % compute the roundness metric
    metric = 4*pi*area/perimeter^2;

    % display the results
    metric_string = sprintf('%2.2f', metric);

    % mark objects above the threshold with a black circle
    if metric > threshold
        centroid = stats(k).Centroid;
        plot(centroid(1),centroid(2), 'k');
    end

    text(boundary(1,2)-35,boundary(1,1)+13,metric_string,'Color','y',...
        'FontSize',14,'FontWeight','bold');

end

title(['Metrics closer to 1 indicate that ', ...
    'the object is approximately round']);

```

Metrics closer to 1 indicate that the object is approximately round

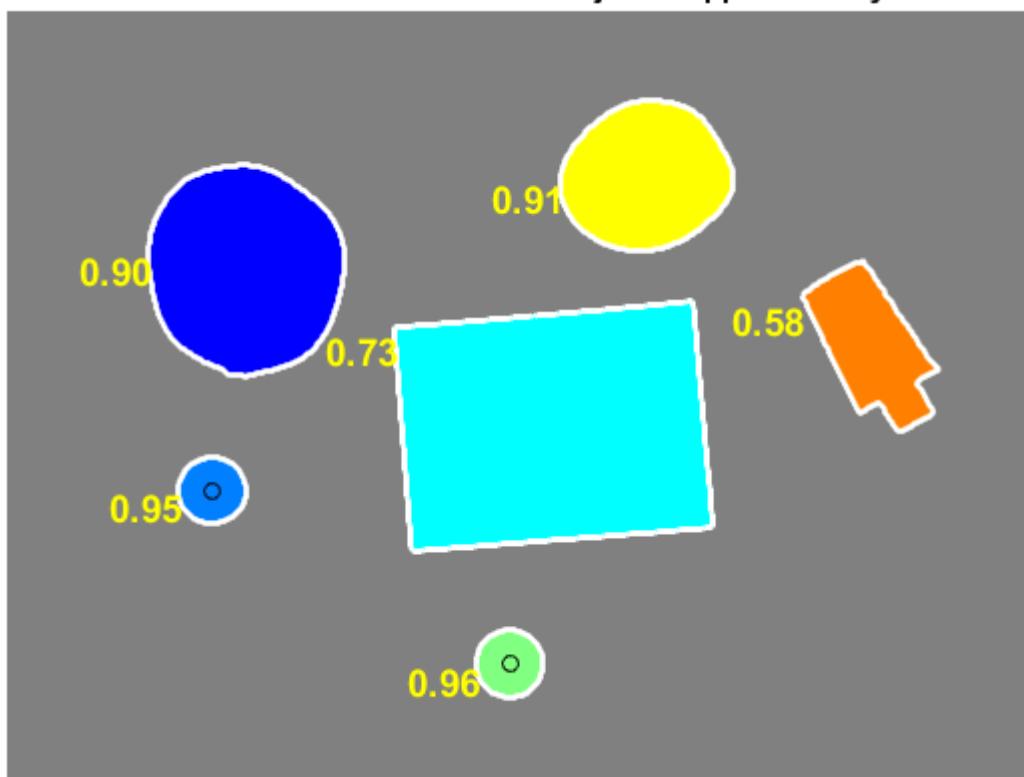


Рисунок 126. – Метрики ближе к 1 указывают, что объект приблизительно круглый

22. Нахождение длины маятника в движении

Данный пример демонстрирует, как рассчитать длину маятника в движении. Мы можем “захватить” изображение с помощью Image Acquisition Toolbox™ и анализировать его посредством Image Processing Toolbox™.

Шаг 1: получение изображения.

Сначала производится загрузка кадра изображения маятника в движении. Кадры, находящиеся в файле pendulum.mat были получены с помощью функции в Image Acquisition Toolbox.

```
% Access an image acquisition device (video object).
% vidimage=videoinput('winvideo',1,'RGB24_352x288');

% Configure object to capture every fifth frame.
% vidimage.FrameGrabInterval = 5;

% Configure the number of frames to be logged.
% nFrames=50;
% vidimage.FramesPerTrigger = nFrames;

% Access the device's video source.
% src=getselectedsource(vidimage);

% Configure device to provide thirty frames per second.
% src.FrameRate = 30;

% Open a live preview window. Focus camera onto a moving pendulum.
% preview(vidimage);

% Initiate the acquisition.
% start(vidimage);

% Wait for data logging to finish before retrieving the data.
% wait(vidimage, 10);

% Extract frames from memory.
% frames = getdata(vidimage);

% Clean up. Delete and clear associated variables.
% delete(vidimage)
% clear vidimage

% load MAT-file
```

```
load pendulum;
```

Шаг 2: исследование последовательности с помощью IMPLAY.

Выполним следующую команду, чтобы исследовать последовательность изображений в implay.

```
implay(frames);
```

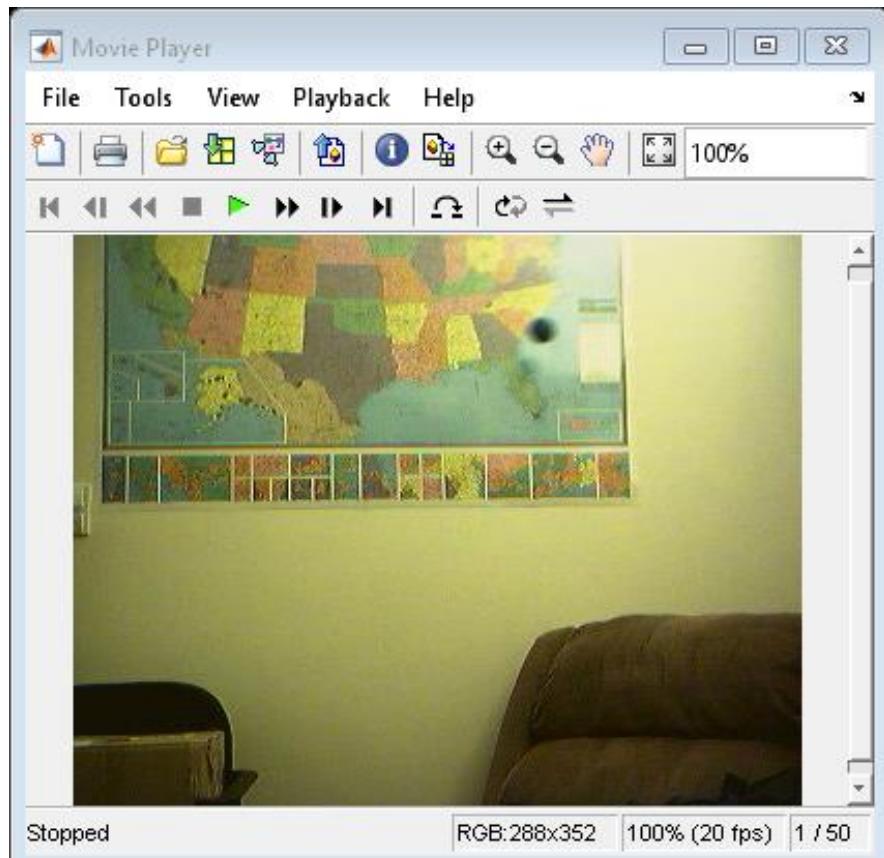


Рисунок 127. – Фрагмент видеозаписи

Шаг 3: выбор области, где колеблется маятник.

Как видно из рисунка, маятник качается в верхней части каждого кадра. Создадим новую серию кадров, которая будет содержать только нужную область.

Чтобы обрезать несколько кадров, используя imcrop, для начала применим imcrop на одном кадре и сохраните изображение. Затем, используя размер прошлого кадра, создадим серию изображений того же

размера. Для удобства используем функцию rect, которая была загружена pendulum.mat в imcrop.

```
nFrames = size(frames,4);
first_frame = frames(:,:, :,1);
first_region = imcrop(first_frame,rect);
frame_regions = repmat(uint8(0), [size(first_region) nFrames]);
for count = 1:nFrames
    frame_regions(:,:,:,count) = imcrop(frames(:,:,:,count),rect);
end
imshow(frames(:,:,:,1))
```



Рисунок 128. – Кадр с качающимся маятником

Шаг 4: выделение маятника в каждом кадре.

Как можно заметить, маятник гораздо темнее, чем задний фон. Поэтому мы можем выделить маятник в каждом кадре, конвертируя их в шкалу серого цвета (grayscale). Затем установить некий порог яркости использую imbinarize и удалить структуры на заднем плане с помощью imopen и imclearborder.

Затем создадим массив с сегментированными кадрами.

```
seg_pend = false([size(first_region,1) size(first_region,2)
nFrames]);
centroids = zeros(nFrames,2);
se_disk = strel('disk',3);
for count = 1:nFrames
```

```

fr = frame_regions(:, :, :, count);

gfr = rgb2gray(fr);
gfr = imcomplement(gfr);

bw = imbinarize(gfr,.7); % threshold is determined
experimentally
bw = imopen(bw,se_disk);
bw = imclearborder(bw);
seg_pend(:, :, count) = bw;

montage({fr,labeloverlay(gfr,bw)}); pause(0.2)

end

```

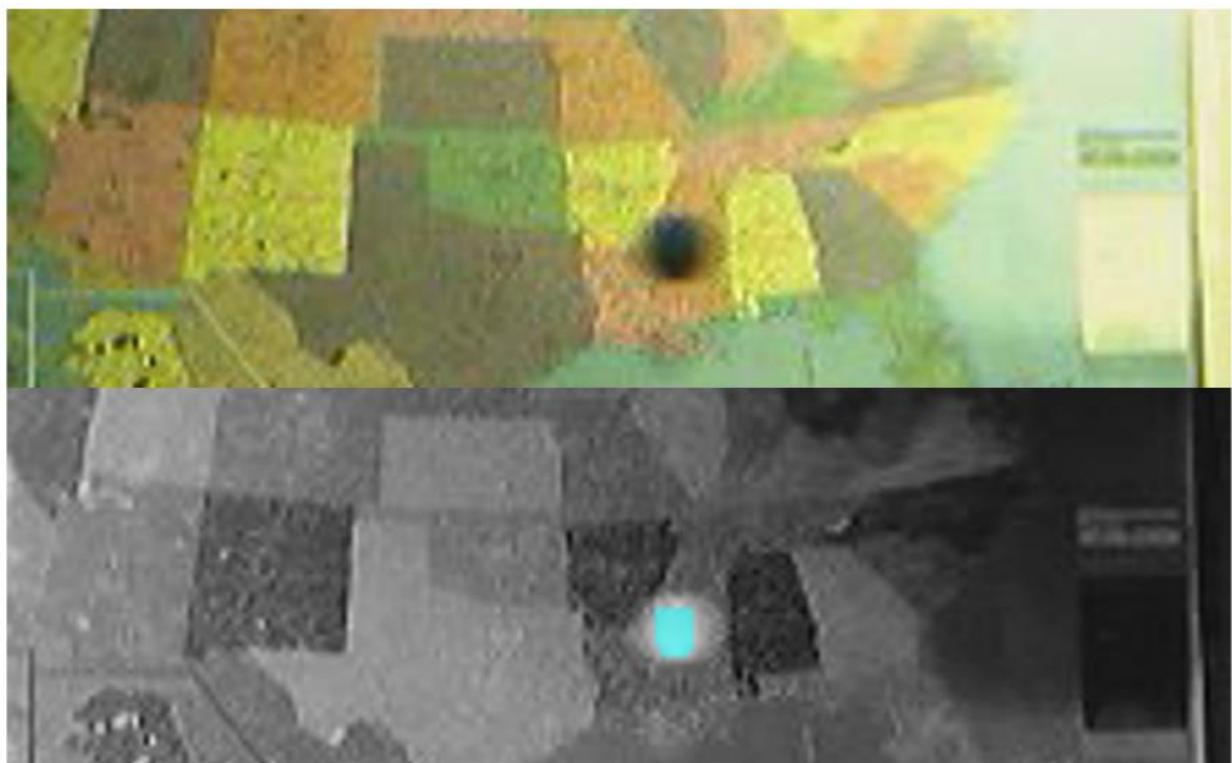


Рисунок 129. – Выделение маятника

Шаг 5: нахождение центра маятника на каждом кадре.

Мы видим, что контур маятника немного меняется от кадра к кадру.

Это не очень большая проблема т. к. нам нужно найти только центр. Зная его, мы можем найти длину маятника.

Используем regionprops для расчета центра маятника.

```
pend_centers = zeros(nFrames,2);
```

```

for count = 1:nFrames
    property = regionprops(seg_pend(:,:,count), 'Centroid');
    pend_centers(count,:) = property.Centroid;
end

```

Изобразим центр маятника посредством plot.

```

x = pend_centers(:,1);
y = pend_centers(:,2);
figure
plot(x,y, 'm.')
axis ij
axis equal
hold on;
xlabel('x');
ylabel('y');
title('Pendulum Centers');

```

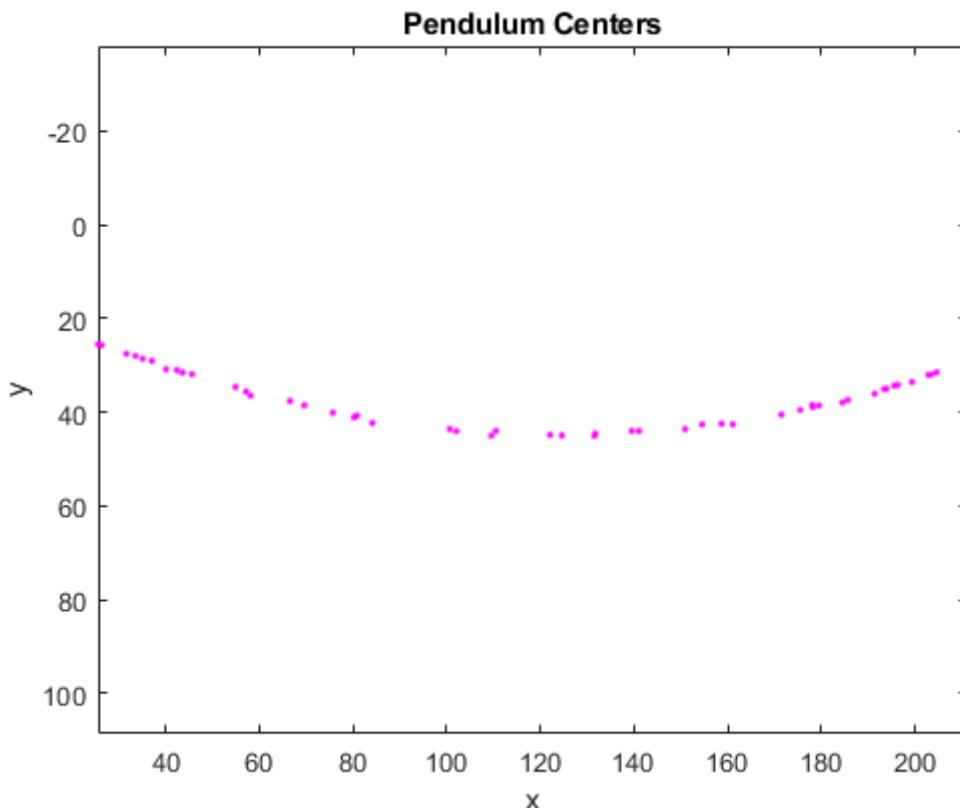


Рисунок 130. – Выделение центра маятника на каждом кадре видео

Шаг 6: расчет радиуса с помощью окружности, проходящей через центры.

Перепишем уравнение окружности:

$$(x-x_c)^2 + (y-y_c)^2 = \text{radius}^2$$

где (x_c, y_c) --- центр, с помощью параметров a, b, c :

$$x^2 + y^2 + a*x + b*y + c = 0$$

где $a = -2*x_c$, $b = -2*y_c$, и $c = x_c^2 + y_c^2 - radius^2$.

Решить это уравнение для параметров a, b, c можно с помощью МНК.

Перепишем уравнение выше как:

$$a*x + b*y + c = -(x^2 + y^2),$$

а затем

$$[x \ y \ 1] * [a; b; c] = -x^2 - y^2.$$

Решим это уравнение используя оператор $()$.

Радиус окружности есть длина нашего маятника в пикселях.

```
abc = [x y ones(length(x),1)] \ -(x.^2 + y.^2);
a = abc(1);
b = abc(2);
c = abc(3);
xc = -a/2;
yc = -b/2;
circle_radius = sqrt((xc^2 + yc^2) - c);
pendulum_length = round(circle_radius)

pendulum_length =
```

253

Визуализируем окружность и ее центр на графике.

```
circle_theta = pi/3:0.01:pi*2/3;
x_fit = circle_radius*cos(circle_theta)+xc;
y_fit = circle_radius*sin(circle_theta)+yc;

plot(x_fit,y_fit,'b-');
plot(xc,yc,'bx','LineWidth',2);
plot([xc x(1)],[yc y(1)],'b-');
text(xc-110,yc+100,sprintf('Pendulum length = %d pixels',
pendulum_length));
```

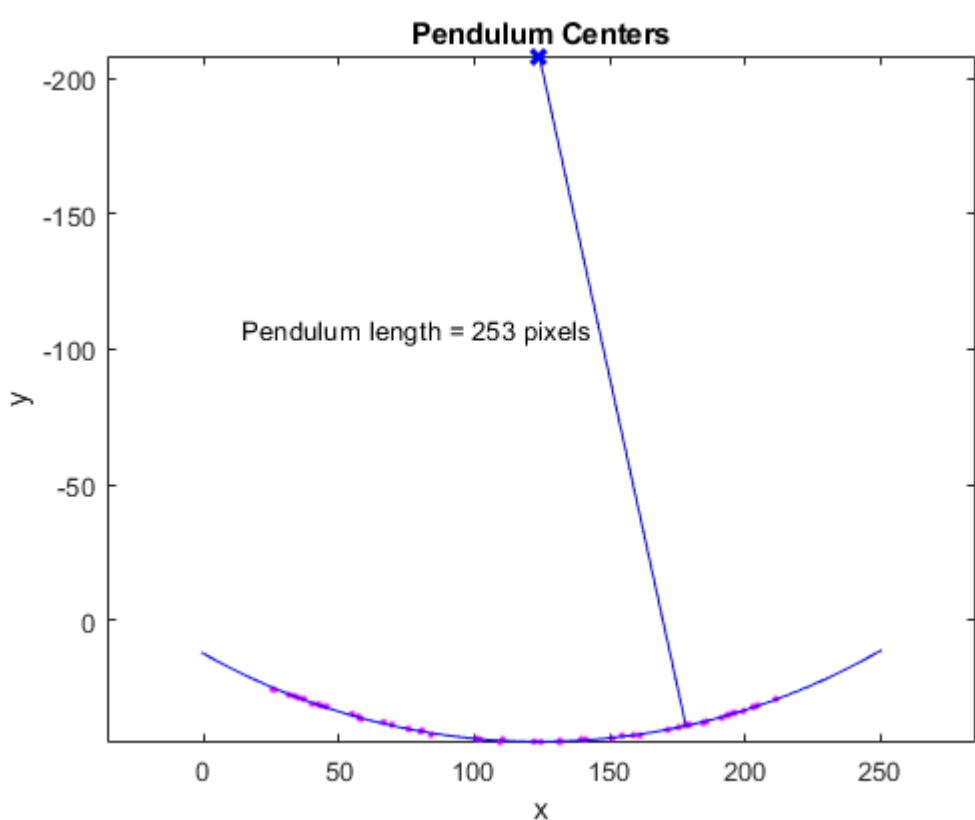


Рисунок 131. – Расчет длины окружности маятника

23. Обнаружение автомобилей в видеоролике

В этом примере показано, как использовать Image Processing Toolbox™ для визуализации и анализа видео или последовательностей изображений. В этом примере используются функции VideoReader (MATLAB®), implay и другие функции Image Processing Toolbox для обнаружения светлых автомобилей в видеоролике. Обратите внимание, что VideoReader имеет специфичные для платформы возможности и, возможно, не сможет прочитать Motion JPEG2000 на некоторых платформах.

Шаг 1: доступ к видео с помощью VideoReader.

Функция VideoReader создает мультимедийный объект, который может считывать видеоданные из мультимедийного файла. Информация о том, какие форматы поддерживаются на вашей платформе, указана в документации для VideoReader.

Используйте VideoReader для доступа к видео и получения основной информации о нем.

```
trafficVid = VideoReader('traffic.mj2')
trafficVid =
```

VideoReader with properties:

General Properties:

```
Name: 'traffic.mj2'
Path:
```

```
'/mathworks/devel/bat/Bdoc18b/build/matlab/toolbox/images/imdata'
```

```
Duration: 8
```

```
CurrentTime: 0
```

```
Tag: ''
```

```
UserData: []
```

Video Properties:

```
Width: 160
```

```
Height: 120
```

```
FrameRate: 15
```

```
BitsPerPixel: 24
```

```
VideoFormat: 'RGB24'
```

Функция get предоставляет дополнительную информацию о видео, такую как его продолжительность в секундах.

```
get(trafficVid)
obj =
    VideoReader with properties:

    General Properties:
        Name: 'traffic.mj2'
        Path: '/mathworks-devel/bat/Bdoc18b/build/matlab/toolbox/images/imdata'
        Duration: 8
        CurrentTime: 0
        Tag: ''
        UserData: []

    Video Properties:
        Width: 160
        Height: 120
        FrameRate: 15
        BitsPerPixel: 24
        VideoFormat: 'RGB24'
```

Шаг 2: изучите видео с помощью IMPLAY.

Исследуйте видео с помощью implay.

```
implay('traffic.mj2');
```

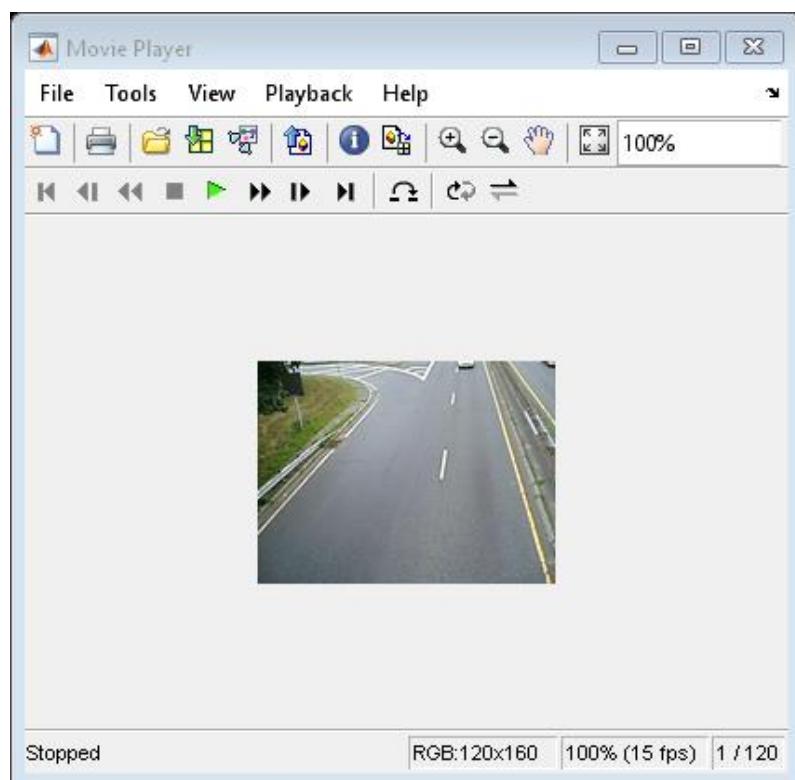


Рисунок 132. – Фрагмент видеозаписи

Шаг 3: разработайте свой алгоритм.

При работе с видеоданными может быть полезно выбрать показательный кадр из видео и разработать алгоритм на этом кадре. Затем этот алгоритм может быть применен к обработке всех кадров в видео.

Для маркировки автомобиля изучите кадр, включающий в себя как светлые, так и темные автомобили. Если изображение имеет множество структур, таких как видео кадры, полезно максимально упростить изображение, прежде чем пытаться обнаружить объект, представляющий интерес. Один из способов сделать это для поиска автомобилей - это подавление всех объектов на изображении, которые не являются светлыми автомобилями (темные автомобили, дорожки, трава и т. д.). Как правило, для удаления этих посторонних объектов требуется комбинация методов.

Один из способов удаления темных автомобилей из видеокадров - использовать функцию `imextendedmax`. Эта функция возвращает двоичное изображение, которое идентифицирует регионы с значениями интенсивности выше определенного порога, называемым региональными максимумами. Все остальные объекты в изображении с значениями пикселей ниже этого порога становятся фоном. Чтобы устраниТЬ темные автомобили, определите среднее значение пикселя для этих объектов на изображении. (Используйте `rgb2gray` для преобразования исходного видео с RGB в черно-белое). Вы можете использовать инструмент области пикселя в `implay` для просмотра значений пикселей. Укажите среднее значение пикселя (или значение немного выше) в качестве порогового значения при вызове `imextendedmax`. В этом примере установите значение 50.

```
darkCarValue = 50;
darkCar = rgb2gray(read(trafficVid,71));
noDarkCar = imextendedmax(darkCar, darkCarValue);
imshow(darkCar)
figure, imshow(noDarkCar)
```



Рисунок 133. – Исходное изображение



Рисунок 134. – Устранение темного авто,
среднее значение пикселя = 50

Обратите внимание на то, что на обработанном изображении, большинство объектов темного цвета удаляются, но остаются многие другие посторонние объекты, например дорожная разметка. Региональная обработка максимумов не будет удалять полосы, потому что их значения пикселей выше порога. Чтобы удалить эти объекты, вы можете использовать морфологическую функцию `imopen`. Эта функция использует морфологическую обработку для удаления небольших объектов из двоичного изображения при сохранении больших объектов. При использовании морфологической обработки вы должны определить размер и форму структурирующего элемента, используемого в операции. Поскольку маркировка дорожек - это длинные и тонкие объекты, используйте структурный элемент в форме диска с радиусом, соответствующим ширине дорожных знаков. Вы можете использовать инструмент области пикселя в `implay` для оценки ширины этих объектов. В этом примере установите значение 2.

```
sedisk = strel('disk',2);
noSmallStructures = imopen(noDarkCar, sedisk);
imshow(noSmallStructures)
```

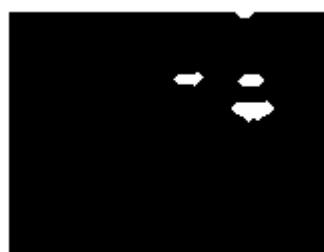


Рисунок 135. – Удаление фона с помощью функции `imopen`

Чтобы завершить алгоритм, используйте `regionprops`, чтобы найти центр тяжести объектов в `noSmallStructures` (должны быть только светлые автомобили). Используйте эту информацию для размещения метки на светлых автомобилях в оригинальном видео.

Шаг 4: применить алгоритм к видео.

Приложение для маркировки автомобилей обрабатывает видео покадрово. (Поскольку типичное видео содержит большое количество кадров, потребуется много памяти для чтения и обработки всех кадров одновременно).

Небольшое видео (как в этом примере) можно было обработать сразу, существует много функций, которые предоставляют эту возможность. Для получения дополнительной информации смотрите документацию.

Для более быстрой обработки предопределите память, используемую для хранения обработанного видео.

```
nframes = trafficVid.NumberOfFrames;
I = read(trafficVid, 1);
taggedCars = zeros([size(I,1) size(I,2) 3 nframes], class(I));

for k = 1 : nframes
    singleFrame = read(trafficVid, k);

    % Convert to grayscale to do morphological processing.
    I = rgb2gray(singleFrame);

    % Remove dark cars.
    noDarkCars = imextendedmax(I, darkCarValue);

    % Remove lane markings and other non-disk shaped structures.
    noSmallStructures = imopen(noDarkCars, sedisk);

    % Remove small structures.
    noSmallStructures = bwareaopen(noSmallStructures, 150);

    % Get the area and centroid of each remaining object in the
    frame. The
    % object with the largest area is the light-colored car. Create
    a copy
    % of the original frame and tag the car by changing the centroid
    pixel
    % value to red.
    taggedCars(:,:,:,k) = singleFrame;

    stats = regionprops(noSmallStructures, {'Centroid','Area'});
    if ~isempty([stats.Area])
        areaArray = [stats.Area];
```

```

[junk,idx] = max(areaArray);
c = stats(idx).Centroid;
c = floor(fliplr(c));
width = 2;
row = c(1)-width:c(1)+width;
col = c(2)-width:c(2)+width;
taggedCars(row,col,1,k) = 255;
taggedCars(row,col,2,k) = 0;
taggedCars(row,col,3,k) = 0;
end
end

```

Шаг 5: визуализация результатов.

Получите частоту кадров исходного видео и используйте его, чтобы увидеть taggedCars в imshow.

```

frameRate = trafficVid.FrameRate;
imshow(taggedCars,frameRate);

```

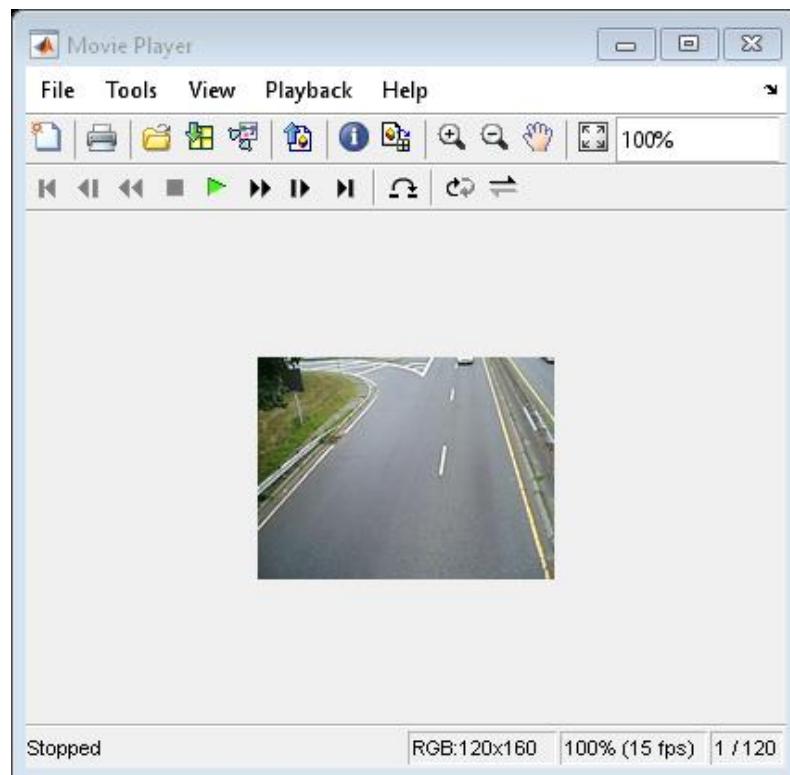


Рисунок 136. – Визуализация результатов

Заключение

Цифровая обработка изображений становится обязательным инструментом при анализе изображений во всех областях радиофизики. В данном методическом пособии даны основные методы обработки изображений, приведен ряд примеров решения прикладных задач посредством обработки изображений разных объектов, характеристики оптической системы в качестве входных параметров для улучшения качества восстановления изображения, а также представлены реализации алгоритмов с базовыми функциями обработки изображений.

Литература

1. Гонсалес Р. Цифровая обработка изображений в среде MATLAB / Р. Гонсалес, Р. Вудс, С. Эддинс, пер. с англ. В.В. Чепыжова. - М.: Техносфера, 2006. – 615с.
2. Новейшие методы обработки изображений / А.А. Потапов [и др.] под общ. ред. А.А. Потапова. - М.:Физматлит, 2008. - 496с.
3. Старовойтов В.В. Цифровые изображения: от получения до обработки / В.В. Старовойтов, Ю.И. Голуб. - Минск: ОИПИ НАН Беларусии, 2014. – 202с.
4. Яне Б. Цифровая обработка изображений. - М: Техносфера, 2007. - 584с.

Учебное издание

Костылев Владимир Иванович,
Левицкая Юлия Сергеевна

ОБРАБОТКА И АНАЛИЗ ИЗОБРАЖЕНИЙ
В СРЕДЕ MATLAB

Учебное пособие

Издано в авторской редакции

Подписано в печать 07.02.2019. Формат 60×84/16
Уч.-изд. л. 7,5. Усл. печ. л. 9,2. Тираж 25. Заказ 40

Издательский дом ВГУ
394018 Воронеж, пл. им. Ленина, 10

Отпечатано с готового оригинал-макета
в типографии Издательского дома ВГУ
394018 Воронеж, ул. Пушкинская, 3