

МИНОБРНАУКИ РОССИИ

---

Санкт-Петербургский государственный электротехнический  
университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

---

Н. А. ОБУХОВА    А. А. МОТЫКО

# **ВИДЕОАНАЛИТИКА И ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗ ДАННЫХ**

Учебное пособие

Санкт-Петербург  
Издательство СПбГЭТУ «ЛЭТИ»  
2018

УДК 004.92  
ББК 32.972.13  
И00

**Обухова Н. А., Мотыко А. А.**

И00

Видеоаналитика и интеллектуальный анализ данных: учеб. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2018. 79 с.

ISBN 000-0-0000-0000-0

Даны теоретические материалы по основам видеоаналитики и интеллектуальному анализу данных. В первой главе рассмотрены вопросы сегментация движущихся объектов на основе оценки энергии движения и поля векторов движения, метод синтезирования панорамного изображения. Вторая глава посвящена методам классификации объектов на основе расстояния Махаланобиса, кластеризации данных  $k$ -средних и регрессионного анализа. Рассмотрены наиболее важные и популярные алгоритмы, используемые при решении практических задач разработки интеллектуальных телевизионных систем. Теоретический материал дополнен практикумом по программированию на языке C++, предполагающим самостоятельную работу студентов в пакете Microsoft Visual Studio.

Предназначено для студентов радиотехнического факультета, обучающихся по направлению «Радиотехника», по программам «Аудиовизуальная техника» и «Инфокоммуникационные технологии анализа и обработки пространственной информации».

УДК 00000000

ББК 00000000

Рецензенты:

Утверждено

редакционно-издательским советом университета

в качестве учебного пособия

ISBN 000-0-0000-0000-0

© СПбГЭТУ «ЛЭТИ», 2016

## Оглавление

1. МЕТОДЫ ВИДЕОАНАЛИТИКИ И ВИДЕОНАБЛЮДЕНИЯ .....	4
1.1. Сегментация движущихся объектов на основе оценки энергии движения .....	4
1.2. Сегментация движущихся объектов на основе оценки поля векторов движения .....	11
1.3. Синтез панорамных изображений .....	23
2. МЕТОДЫ ИНТЕЛЛЕКТУАЛЬНОГО АНАЛИЗА ДАННЫХ.....	32
2.1. Классификация объектов с помощью расстояния Махаланобиса .....	32
2.2. Кластеризация методом $k$ -средних.....	37
2.3. Регрессионный анализ .....	41
3. ПРАКТИКУМ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C++ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ КОМПЬЮТЕРНОГО ЗРЕНИЯ OPENCV .....	47
3.1. Установка рабочей среды для программирования .....	47
3.2. Начало работы с OpenCV. Материалы для выполнения задания из раздела 1.1 .....	53
3.3. Совмещение с шаблоном. Материалы для выполнения задания из раздела 1.2 .....	61
3.4. Автоматический синтез панорамных изображений. Материалы для выполнения задания из раздела 1.3 .....	64
3.5. Операции с матрицами и векторами. Материалы для выполнения задания из раздела 2.1 .....	69
3.6. Кластеризация с помощью алгоритма К-средних. Материалы для выполнения задания из раздела 2.2 .....	73
3.7. Разделение и слияние каналов матриц. Материалы для выполнения задания из раздела 2.3 .....	76
СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ .....	78

# ГЛАВА 1. МЕТОДЫ ВИДЕОАНАЛИТИКИ И ВИДЕОНАБЛЮДЕНИЯ

## 1.1. СЕГМЕНТАЦИЯ ДВИЖУЩИХСЯ ОБЪЕКТОВ НА ОСНОВЕ ОЦЕНКИ ЭНЕРГИИ ДВИЖЕНИЯ

При разработке практически любой системы компьютерного зрения возникает задача автоматической сегментации целевых объектов. Под сегментацией в обработке изображений понимают разделение изображения на области, для которых выполняется определенный критерий однородности. Например, однородность может пониматься, как близость элементов разложения изображения по яркости или цветовому тону. Более конкретизированный термин «сегментация объекта» подразумевает разделение объекта и фона, иначе говоря – выделение объекта на изображении, или в видеоряде.

Если исходными данными являются изображения, то сегментацию объектов проводят по признакам яркости, цветности, локальной детальности (сгусткам высокочастотной энергии) и формы. Для оценки формы применяют аппарат контурного анализа, а оценку детальности традиционно осуществляют с помощью меры Розенфельда-Троя.

Дополнительно, для решения задачи сегментации объектов используют текстурные и цветовые признаки, а также более сложные техники, основанные на применении метода локальных бинарных шаблонов, Виолы-Джонса, ориентированных градиентов и других алгоритмов из области машинного обучения.

В видеоаналитике часто встречается задача, в которой исходные данные представляют собой видеопоследовательность, а целевые объекты перемещаются в поле зрения камер системы, то есть обладают признаком движения. Учет этого признака обеспечивает устойчивую сегментацию областей изображения, принадлежащих движущимся объектам на неподвижном, в том числе сложном, фоне.

При разработке алгоритмов сегментации объектов с учетом признака движения необходимо учитывать характер движения. Область интереса может обладать жестким (ригидным) и нежестким движением. Жесткое предполагает перемещение всех элементов объекта, в соответствии с основным направлением движения. Примерами являются движущийся автомобиль, летательный аппарат, корабль. Нежестким движением обладают, например, идущий человек, летящая птица.

Достаточно простой и эффективный метод сегментации объектов на основе признака движения основан на оценке «энергии движения». Под этим термином

понимают межкадровую разность телевизионных сигналов. В случае двух цифровых изображений, представляющих соседние (или отстоящие друг от друга на некоторое время  $t$  кадры, оценка движения может быть найдена как поэлементная абсолютная разность двух кадров:

$$MAD(x, y) = |L(x_j, y_i, t) - L(x_j, y_i, t-1)|,$$

где,  $L(x, y, t)$  – значение яркости пикселя с координатами  $(x, y)$  в кадре  $t$ ,  $L(x, y, t-1)$  – значение яркости пикселя с координатами  $(x, y)$  в кадре  $t-1$ .

Препарат межкадровой разности двух кадров представлен на рисунке 1.1.

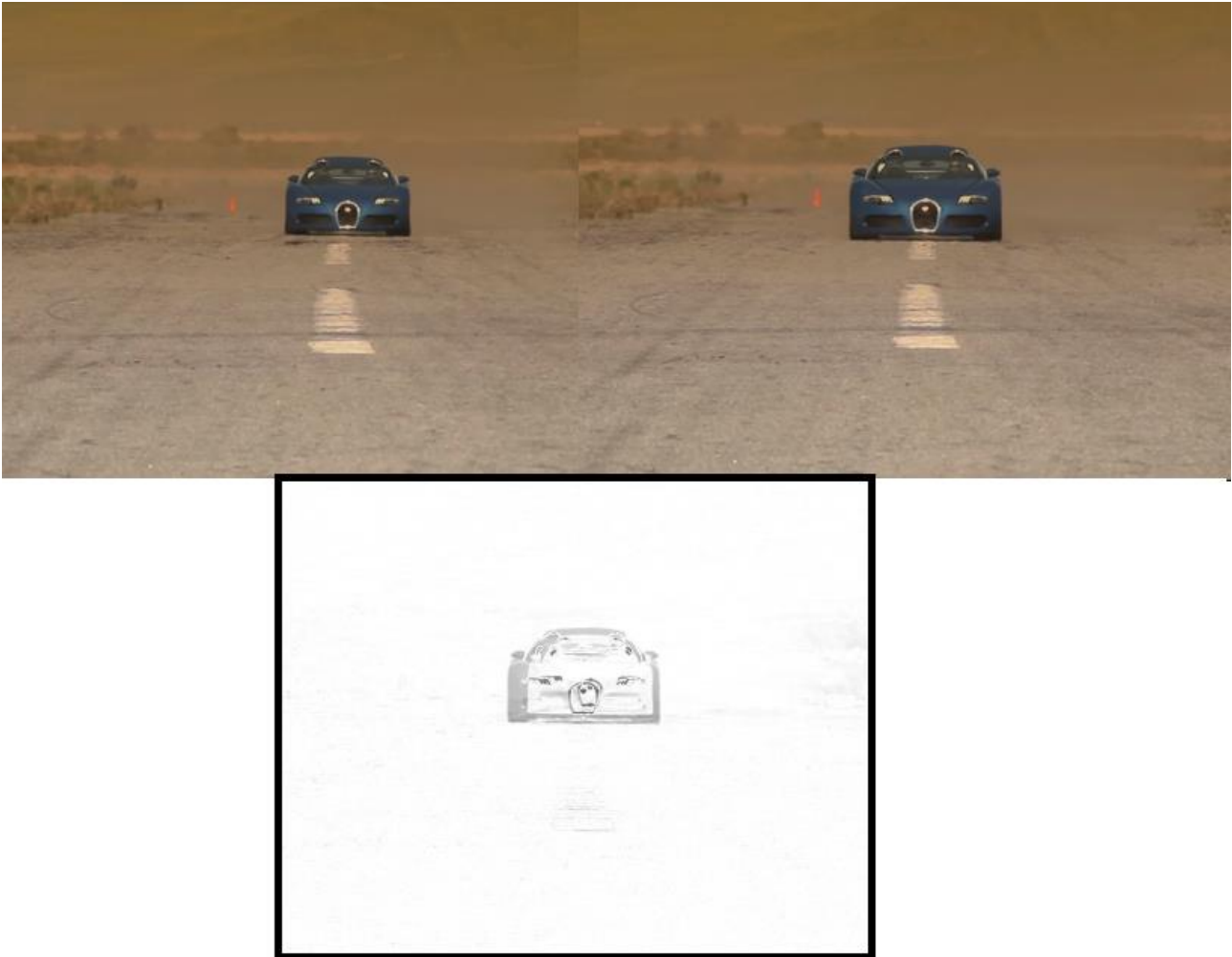


Рис. 1.1. Пара кадров (верхний ряд) и их разность (внизу).

Очевидно, что в областях, где присутствует движение, абсолютная разность яркостей пикселей будет отлична от нуля, а там, где движение отсутствует будет близка к нулю.

Следует учитывать, что с помощью видеокамеры регистрируется только видимое двумерное движение объекта, которое является проекцией реального движения объекта в трехмерном пространстве на плоскость изображения. Видимое движение порождено изменениями яркости пикселей  $L(x_j, y_i, t)$  с координатами  $x$  и  $y$  в кадре  $t$  по отношению к кадру  $t-1$ . Однако, ненулевой

результат межкадровой разности не эквивалентен только видимому движению, так как, например, в статической сцене с изменяющейся освещенностью, где на самом деле движения нет, или может быть вызван шумами. К ненулевой межкадровой разности приведет и наличие глобального движения в сцене, которое возникает в случае движущейся камеры. При анализе видеоряда и сегментации объектов глобальное движение следует предварительно компенсировать.

При отсутствии глобального движения алгоритм сегментации объектов на основе энергии движения состоит из следующих шагов:

1. получение препарата межкадровой разности;
2. формирование контурного препарата для текущего кадра  $t$ ;
3. бинарное квантование контурного препарата и межкадровой разности;
4. объединение двух полученных препаратов с помощью операции логического «И» – получение изображения, содержащего оценку энергии движения, относящейся только к текущему кадру;
5. морфологическая фильтрация полученного изображения;
6. формирование строба объектов.

Второй и третий шаги необходимы потому, что изображение межкадровой разности отражает энергию движения, как предыдущего, так и текущего кадров. В результате центр области движения не совпадает с центром движения реального объекта. Определение контуров объектов в текущем кадре  $t$  и объединение препаратов по «И» устраняет этот фактор.

Для выделения контуров в текущем кадре  $t$  видеопоследовательности могут применяться различные алгоритмы (оператор Собеля, разницы гауссиан и так далее). Традиционным в данном случае является использование оператора Собеля. Алгоритм использует ядра (маски) размером 3 на 3 элемента для выделения горизонтальных и вертикальных контуров соответственно:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}.$$

Для изображения (кадр  $t$ ) осуществляют свертку с каждым из ядер для вычисления приближённых значений производных по горизонтали  $G_x$  и по вертикали  $G_y$ . Приближённое значение величины градиента  $G$  можно поэлементно вычислить путём использования полученных приближенных значений производных по направлениям:

$$G(x, y) = \sqrt{G(x, y)_x^2 + G(x, y)_y^2}.$$

Пример контурного препарата, полученный для кадра  $t$  на рисунке 1 приведен на рисунке 1.2.

Контурное изображение  $G(x,y)$  затем бинарно квантуют так же, как и изображение межкадровой разности  $MAD(x,y)$ , что дает возможность получить объединенное изображение  $K(x,y)$ , выполнив поэлементно логическую операцию «И»:

$$K(x, y) = MAD(x, y) \& G(x, y).$$

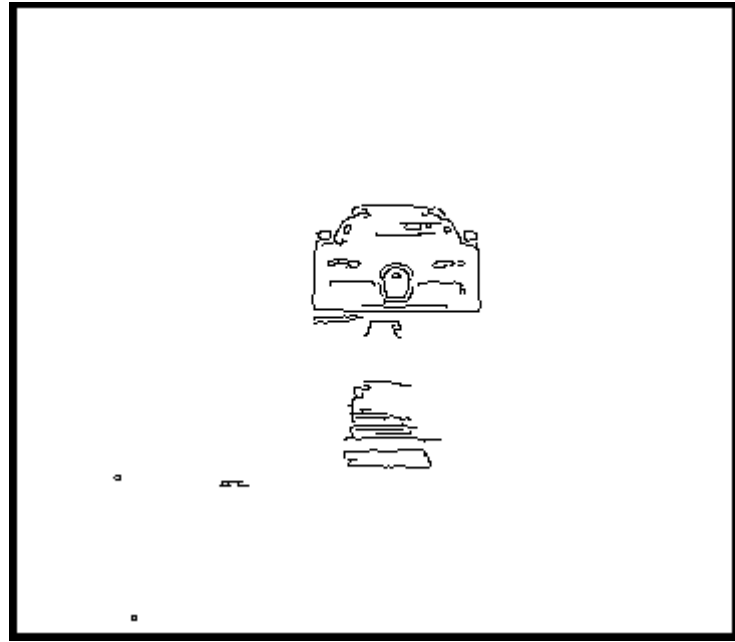


Рис. 1.2. Контурный препарат

В результате получают изображение (рис. 1.3), в котором выделенные движущиеся объекты занимают положение, соответствующее текущему кадру.

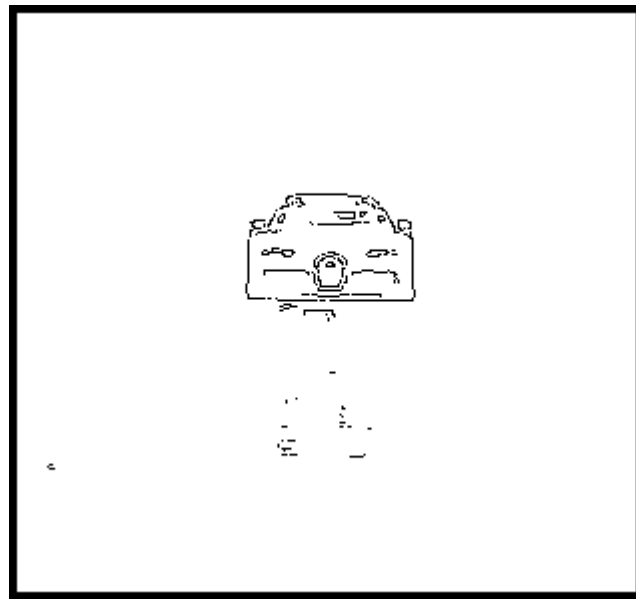


Рис. 3. Выделенный слой движения.

Полученный таким образом препарат  $K(x,y)$  может содержать множество нежелательных фрагментов, вызванных, например, шумами. Для снижения

помех целесообразно провести предварительную фильтрацию исходного изображения (например, медианным фильтром), а затем изображения межкадровой разности.

Изображения движущихся объектов на результирующем изображении  $K(x,y)$  могут иметь разрывы, лакуны и прочие дефекты, которые затруднят точную сегментацию. Для удаления этих дефектов используют так называемую морфологическую фильтрацию, основанную на логических операциях. Морфологическую фильтрацию применяют к цифровым изображениям как бинарно квантованным, так и полутоновым.

Логические операции совершают между маской  $S$  и областью цифрового изображения, выделенной этой маской. Результат операции устанавливают в новую битовую матрицу на место, где находится фокус маски. Используют маски различной формы и размерности.

Морфологическая операция «открытие» (последовательное применение операций эрозии и наращивания) с маской малого размера (относительно предполагаемого размера объекта) позволяет подавить импульсный шум на изображении. Применение после этого операции «закрытие» (последовательное применение операций наращивания и эрозии) с увеличенным размером маски способствует удалению разрывов и заполнению лакун в изображениях объектов.

На заключительном шаге алгоритма организуют наложение строка на объект (рис. 4). Для формирования строка в методе сегментации объектов на основе «энергии движения» используют алгоритм проекций. Для результирующего изображения  $K(x,y)$ , прошедшего морфологическую фильтрацию, в каждом столбце и в каждой строке подсчитывают количество ненулевых («белых» в рассматриваемом примере) пикселей. По полученным данным строят соответствующие гистограммы, сопоставляющие номеру строки (столбца) – число ненулевых пикселей.

Затем выделяют проекции – совокупность строк (столбцов), у которых число ненулевых пикселей превышает некий порог и которые образуют на гистограмме непрерывный отрезок. Проекции, длина которых меньше заданной величины (определяемой априорным знанием о предполагаемых размерах объектов), отбрасывают, чтобы исключить изолированные помехи. На базе оставшихся проекций формируют строки.

Если в кадре движутся несколько объектов, то для установления соответствия между проекциями по горизонтали и вертикали используют правило: число пикселей в горизонтальной и вертикальной проекции одного объекта должны быть равны.



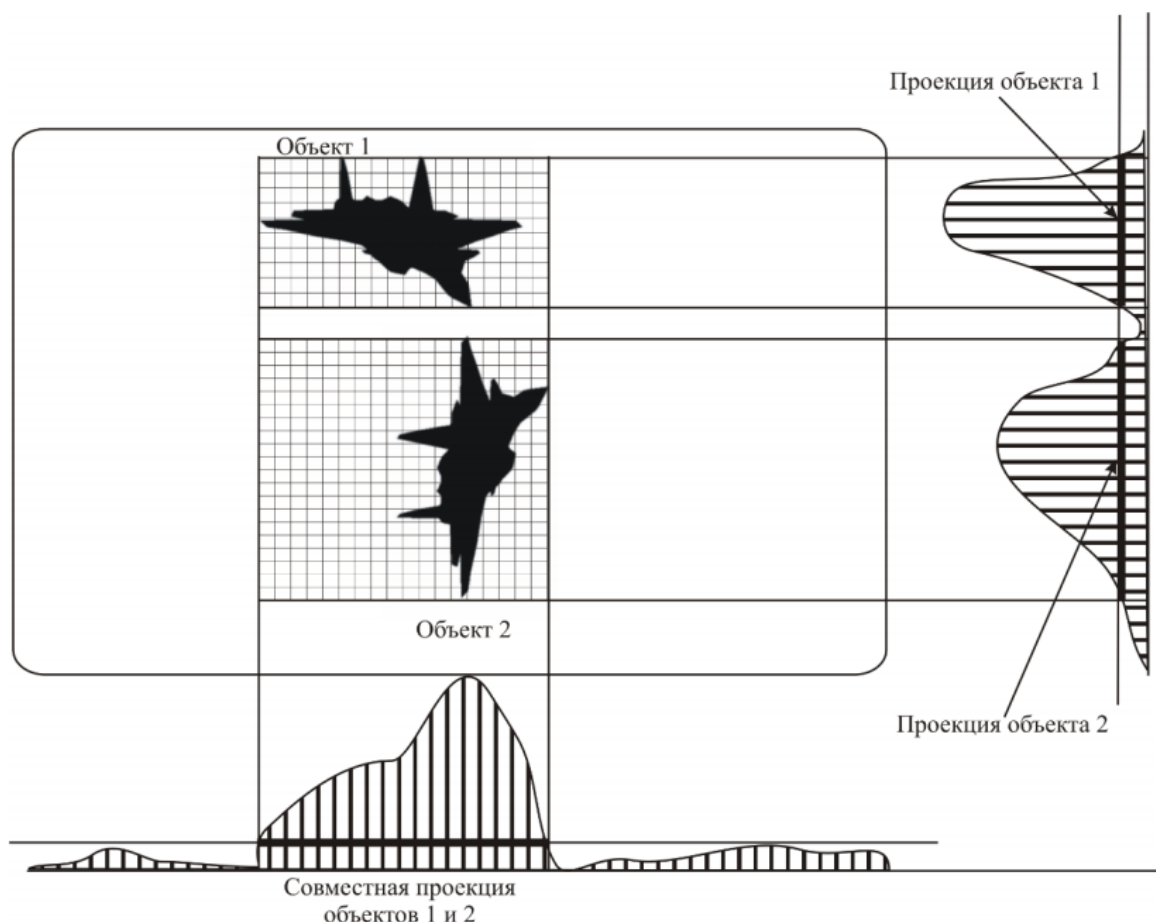


Рис. 4. Стробирование объектов

Если проекции объектов перекрывают друг друга, то либо не удастся установить центр тяжести объекта (при перекрытии по одному направлению, либо будут выделены сразу два объекта (при перекрытии по двум направлениям), что будет являться ошибкой сегментации.

Для того чтобы разрешить эту ситуацию возможно использование дополнительной координатной оси (например, оси под углом 45 градусов к координатным).

К ограничениям алгоритма нужно отнести тот факт, что с его помощью локализуется место в кадре, где зафиксировано движение. Однако о направлении и скорости движения посредством данного алгоритма судить нельзя. В результате даже если пара объектов движется в противоположные стороны с различными скоростями, но их проекции перекрываются, разделить данные объекты будет невозможно. Также без данных о направлении и скорости в общем случае невозможно реализовать трекинг объектов.

Следует отметить, что в данном случае рассмотрен алгоритм сегментации движущихся объектов при отсутствии в сцене глобального движения. То есть предполагается, что камера неподвижна и фон статичен. В случае, если камера

перемещается, то применение описанного алгоритма возможно только после предварительной компенсации глобального движения.

### **Задание на моделирование**

Разработать программу, реализующую алгоритм сегментации движущегося объекта с помощью оценки энергии движения. Исходными данными являются два кадра видеопоследовательности со статичным фоном, на которых присутствует перемещающийся объект.

В программе должны быть реализованы следующие процедуры, с выводом результата на экран:

- загрузка пары кадров,
- получение препарата межкадровой разности,
- получение контурного препарата с помощью одного из алгоритмов, указанных преподавателем: Собеля, разницы гауссиан, Кэнни, Робертса.
- объединение препаратов по логическому «И»
- обработка сформированного изображения с помощью морфологических фильтров (фильтрация и заливка),
- наложение строба на объект.

### **Контрольные вопросы**

1. Каким образом формируют препарат межкадровой разности изображений?
2. Перечислите основные способы получения контурного препарата.
3. Раскройте принцип объединения изображений в соответствии с операцией логического «И»?
4. Поясните основные достоинства алгоритма сегментации объектов на основе энергии движения.
5. Поясните основные ограничения алгоритма сегментации на основе энергии движения.

## 1.2. СЕГМЕНТАЦИЯ ДВИЖУЩИХСЯ ОБЪЕКТОВ НА ОСНОВЕ ОЦЕНКИ ПОЛЯ ВЕКТОРОВ ДВИЖЕНИЯ

Для сегментации и сопровождения объектов по признаку движения может быть использована оценка поля векторов движения. Это поле описывает видимое перемещение объектов в кадре. Векторы движения позволяют не только детектировать факт перемещения объекта, но и оценить его скорость и направление.

Наличие информации о направлении и скорости позволяет решить большое число прикладных задач:

- сегментировать объекты на сложном фоне, находящиеся в непосредственной близости друг к другу;
- разрешить ситуации окклюзии при сопровождении путем выявления объекта, находящегося на переднем плане;
- построить модель движения по совокупности векторов (на основе информации, полученной в одном кадре) и осуществить сопровождение объекта.

Аппарат векторов движения (векторов оптического потока) был разработан для применения в стандартах видеокompрессии *MPEG*. Однако, в видеоаналитике векторы движения также стали очень популярны и востребованы. В задачах компьютерного зрения вектор движения показывает смещение фрагмента изображения в кадре  $t+1$  относительно кадра  $t$ .

Для определения векторов движения основным является уравнение оптического потока, полученное на основе допущения о постоянстве яркости  $L(x,y,t)$  точки (пикселя) при движении

$$\frac{d}{dt}L(x,y,t)=0.$$

Так как  $x=f(t)$  и  $y=f(t)$ , вычисления нужно проводить по формуле сложной производной:

$$\frac{d}{dt}L(x,y,t)=\frac{\partial L}{\partial x}\cdot\frac{\partial x}{\partial t}+\frac{\partial L}{\partial y}\cdot\frac{\partial y}{\partial t}+\frac{\partial L}{\partial t}=\langle\nabla\mathbf{L},\mathbf{v}\rangle+\frac{\partial L}{\partial t}=0,$$

где  $L(x,y,t)$  – яркость пикселя с координатами  $x$  и  $y$  в момент времени  $t$ ;  $\langle...\rangle$

обозначает скалярное произведение векторов, а  $\nabla\mathbf{L}=\left(\frac{dL}{dx},\frac{dL}{dy}\right)^T$  – вектор-градиент;  $\mathbf{v}=[v_x, v_y]^T$  – вектор скорости (оптического потока).

Вместо производных по времени и пространству используют их целочисленные приближения

$$dL/dx \approx \Delta L/\Delta x; \quad dL/dy \approx \Delta L/\Delta y; \quad dL/dt \approx \Delta L/\Delta t$$

и вычисляют приращения яркости в соседних пикселях по вертикали и горизонтали, а также в соседнем кадре, т. е. приравнивают  $\Delta x$  и  $\Delta y$  одному пикселю, а  $\Delta t$  - одному кадру (рис. 2.1).

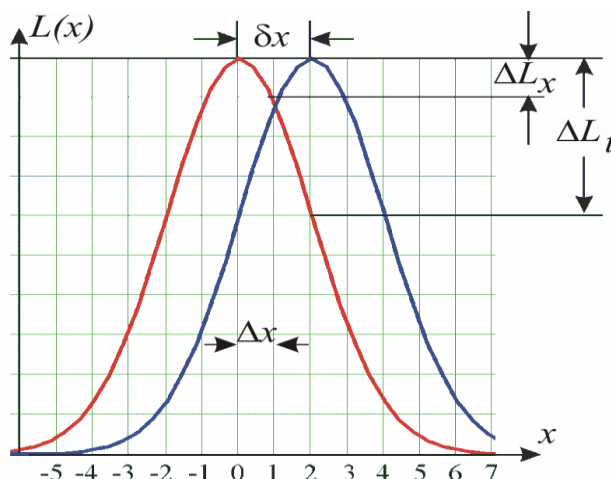


Рис. 2.1. Иллюстрация к расчету вектора движения

Из анализа уравнения оптического потока можно сделать следующие выводы (которые объясняют ограничения метода):

1. Уравнение оптического потока является недоопределенным и позволяет найти только сонаправленную с яркостным вектором-градиентом  $\nabla \mathbf{L} = (\frac{\partial L}{\partial x}, \frac{\partial L}{\partial y})$

компоненту векторов оптического потока. Ортогональная с яркостным вектором-градиентом компонента может принимать любые значения, не изменяя скалярного произведения, и поэтому не может быть определена однозначно.

Для полной оценки векторов оптического потока необходимо ввести требование гладкости — близости скоростей у группы соседних пикселей. Принятие решения о размере такой группы называют проблемой апертуры.

2. Однозначное определение векторов оптического потока возможно только в случае, если компоненты яркостного вектора градиента отличны от нуля, т.е. имеют место изменения яркости по горизонтали и вертикали. В случае гладкой поверхности достоверную оценку векторов найти нельзя.

3. Уравнение оптического потока предполагает постоянство яркости при движении точки вдоль траектории. Подсветки, тени, блики (которые часто возникают при видеонаблюдении на открытом воздухе), низкая детальность, прозрачные и зеркальные поверхности реальных объектов нарушают это

утверждение, что затрудняет вычисления и приводит к ошибкам при определении векторов оптического потока.

Перечисленные факторы обуславливают высокую вероятность появления ошибочных, так называемых «аномальных векторов», которые не соответствуют реальному перемещению, существующему в видеопоследовательности, и могут быть не сонаправленными с движением объекта, к которому они принадлежат. Аномальные векторы в задачах видеокомпрессии приводят только к уменьшению степени сжатия, в то время, как в системах видеоаналитики их следствием является разделение объектов интереса на части (ошибка сегментации), снижение точности моделей движения, потеря объекта при сопровождении.

Для поиска векторов движения существует три группы методов: дифференциальные, фазовые и корреляционные.

**Дифференциальные методы** определяют векторы оптического потока исходя из предположения, что изображение является непрерывным (дифференцируемым) в пространстве и во времени. Методы этой группы делятся на глобальные и локальные, а также на методы первого и второго порядка на основании используемых производных.

*Глобальные методы* используют основное уравнение оптического потока и добавляют к нему некую функцию ошибок. В известном методе Хорна-Шунка, оптический поток определяют путем минимизации функционала

$$\int_S \left( (\langle \nabla \mathbf{L}, \mathbf{V} \rangle + L_t)^2 + \lambda^2 \text{tr}((\nabla \mathbf{V})^T, (\nabla \mathbf{V})) \right) d\mathbf{r} = \min,$$

$$\nabla \mathbf{V} = \left( \frac{d^2 x}{dt^2}, \frac{d^2 y}{dt^2} \right),$$

где первое слагаемое – уравнение оптического потока, а второе – функция ошибок, учитывающая близость скоростей;  $\lambda$  – заранее заданная константа;  $\text{tr}(\dots)$  – след матрицы, равный сумме ее диагональных элементов;  $S$  – область изображения, для которой ищут минимум функции;  $\mathbf{r} = (x, y)^T$  – вектор пространственных координат;  $L_t$  – обозначена производная яркости по времени.

Преимуществом метода Хорна-Шунка является высокая плотность получаемых векторов движения. Для оценки векторов, принадлежащих внутренним гладким частям объекта, используются значения, полученные для хорошо выделяющихся движущихся внешних границ областей объекта. К

недостаткам относят относительно низкую устойчивость к шуму (по сравнению с локальными методами).

*Локальные методы* основаны на предположении, что в окрестности каждого пикселя значение оптического потока одинаково, таким образом можно записать основное уравнение оптического потока для группы пикселей (маска **S**) окрестности и решить полученную систему уравнений методом наименьших квадратов.

Минимизируют квадратичную функцию ошибки

$$\sum_{x,y \in S} (\langle \nabla L(x, y, t), \mathbf{v} \rangle + L_t(x, y, t))^2 = \min.$$

Для взвешенного метода наименьших квадратов

$$\sum_{x,y \in S} [\mathbf{W}(x, y)] (\langle \nabla L(x, y, t), \mathbf{v} \rangle + L_t(x, y, t))^2 = \min,$$

где  $\mathbf{W}(x, y)$  – диагональная весовая матрица, усиливающая влияние центральных пикселей маски **S** при оценке векторов оптического потока.

Элементы на диагонали матрицы  $w(x, y) = w(x) \otimes w(y)$ , где знак  $\otimes$  означает кронекерово произведение (каждый с каждым). Например, при маске **S** (5x5) пикселей, принимают  $w(-2) = w(2) = 0,0625$ ;  $w(-1) = w(1) = 0,25$ ;  $w(0) = 0,375$ , т. е. каждому из 25 пикселей изображения в окне присвоен весовой коэффициент.

Оценка векторов движения определяется соотношением:

$$\mathbf{v} = [\mathbf{A}^T \mathbf{W} \mathbf{A}]^{-1} \mathbf{A}^T \mathbf{W} \mathbf{b},$$

где

$$\mathbf{A} = \begin{bmatrix} L_x(x-2, y-2, t) & L_y(x-2, y-2, t) \\ L_x(x-2, y-1, t) & L_y(x-2, y-1, t) \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ L_x(x+2, y+1, t) & L_y(x+2, y+1, t) \\ L_x(x+2, y+2, t) & L_y(x+2, y+2, t) \end{bmatrix};$$

$$\mathbf{W} = \begin{bmatrix} w(-2,-2) & & & & & \\ & w(-2,-1) & & & & \\ & & \ddots & & & \\ & & & \ddots & & \\ & & & & \ddots & \\ & & & & & w(2,1) \\ & & & & & & w(2,2) \end{bmatrix};$$

$$\mathbf{b} = \begin{bmatrix} L_t(x-2, y-2, t) \\ L_t(x-2, y-1, t) \\ \\ \\ L_t(x+2, y+1, t) \\ L_t(x+2, y+2, t) \end{bmatrix}.$$

Локальный дифференциальный метод Лукаса-Канаде [2.5], базирующейся на вышеописанных принципах, практически является самым популярным алгоритмом для вычисления оптического потока в системах компьютерного зрения. Алгоритм Лукаса-Канаде менее чувствителен к шуму на изображениях, чем глобальные методы, однако он не может определить направление движения пикселей внутри однородных областей.

**Фазовые алгоритмы** основаны на методе фазовой корреляции [2.6] для сопоставления изображений (в данном случае – фрагментов изображений, блоков). Метод использует переход в частотную область для вычисления взаимной спектральной плотности и обратное преобразование для получения функции кросс-корреляции, максимум которой позволяет определить смещение, то есть вектор движения.

**Корреляционные методы** определяют векторы оптического потока на основе смещений, при которых достигается максимальное соответствие фрагментов изображения текущего и предыдущего кадров. Определение наилучшего соответствия выполняется путем поиска максимума корреляционной функции.

Наиболее часто используют метод сопоставления блоков, принятый во многих стандартах видеокодирования.

Метод состоит из следующих основных шагов:

- текущий кадр делится на неперекрывающиеся квадратные блоки размером  $M \times N$  пикселей;
- для каждого блока формируется область поиска в предыдущем кадре, которая имеет размер  $(2d+M+1) \times (2d+N+1)$  пикселей, где  $d$  – это максимально возможное смещение в горизонтальном и вертикальном направлениях (рис. 2.2);

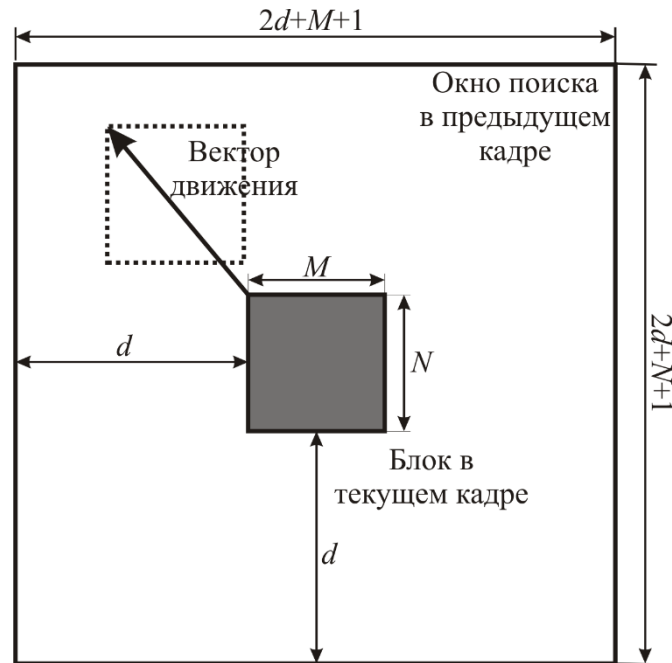


Рис. 2.2. Совмещение блоков

- выполняется совмещение блоков текущего кадра с блоками предыдущего кадра видеопоследовательности, определяется, какому блоку в области поиска текущий блок лучше всего соответствует, и оценивается величина смещения положения блока в текущем кадре относительно предыдущего – вектор движения.

Считается, что все пиксели блока претерпевают одинаковое перемещение и им приписывается один и тот же вектор движения.

Задача определения векторов движения в этом случае решается путем минимизации целевой функции, характеризующей степень соответствия (совпадения) двух блоков, на множестве различных положений обрабатываемого блока в области поиска.

**Формирование целевой функции**, оценивающей степень соответствия между блоком текущего кадра и блоком предыдущего кадра, может быть выполнено в нескольких вариантах:

- 1) средняя абсолютная разность (*MAD*):



$$MAD(v_x, v_y) = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |L_c(x_k + i, y_l + j) - L_p(x_k + v_x + i, y_l + v_y + j)|,$$

где  $L_c(\dots)$  и  $L_p(\dots)$  – яркости пикселя в текущем и предыдущем кадре соответственно;  $(x_k, y_l)$  – координаты пикселя левого верхнего угла текущего блока;  $N \cdot N$  – размер блока;  $(v_x, v_y)$  – один из возможных векторов движения;

2) среднеквадратическая ошибка ( $MSE$ ):

$$MSE(v_x, v_y) = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (L_c(x_k + i, y_l + j) - L_p(x_k + v_x + i, y_l + v_y + j))^2;$$

3) нормированная функция взаимной корреляции ( $NCCF$ ):

$$NCCF(v_x, v_y) = \frac{\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} L_c(x_k + i, y_l + j) * L_p(x_k + v_x + i, y_l + v_y + j)}{\sqrt{\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} L_c^2(x_k + i, y_l + j) * \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} L_p^2(x_k + v_x + i, y_l + v_y + j)}};$$

4) максимальное число соответствующих пикселей ( $MPC$ ):

$$MPC(v_x, v_y) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} T(x_k + i, y_l + j, v_x, v_y),$$

$$T(s, t, v_x, v_y) = 1, \text{ если } |L_c(s, t) - L_p(s + v_x, t + v_y)| < Th, \text{ иначе } T(s, t, v_x, v_y) = 0,$$

где  $Th$  – предопределенный порог.

В отличие от предыдущих критериев блок наилучшего соответствия, найденный с использованием  $MPC$ , тот, который дает самое большое значение целевой функции. Обычно в качестве целевой функции используется  $MAD$ , так как она дает характеристику, близкую к характеристике  $MSE$ , но не требует операций умножения.

**Совмещение блоков.** Самым простым и надежным алгоритмом, позволяющим выполнить совмещение блоков, является полный перебор  $FS$  (*full search*), но из-за большого объема вычислений он обладает низкой скоростью. Тем не менее, в большинстве случаев в системах видеонаблюдения и видеоаналитики, где критически важна точность сопровождения объектов, используют именно его. В области сжатия видео, или при видеонаблюдении в условиях высокого отношения сигнал/шум исходных данных, применяют другие подходы, которые направлены на ускорение процесса поиска минимума. Их можно разделить на два класса:

- алгоритмы, уменьшающие число вычислений при определении целевой функции;

- алгоритмы, уменьшающие число контрольных точек в области поиска.

Примерами, относящимися к первому классу, являются алгоритм неполного определения целевой функции, алгоритм остановки на полпути (PDS) и остановки на полпути с нормировкой (NPDS).

В рамках второго класса можно выделить три группы алгоритмов:

1. основанные на свойстве унимодальности целевой функции; примерами являются алгоритм поиска тремя итерациями (3SS), алгоритмы логарифмического, ортогонального и поперечного поиска, алгоритм поиска по квадрантам;
2. учитывающие одновременно унимодальность минимизируемой функции и скорость оцениваемого движения; к этой группе относятся алгоритмы двух видов: ориентированные на работу с видеопоследовательностями, в которых преобладает медленное движение, такие как блочный градиентный поиск, четырехшаговый алгоритм (4SS), а также производящие предсказание характера оцениваемого движения (быстрое/медленное) и далее использующие наиболее эффективный для данного вида движения подход, – гибридные алгоритмы.
3. предсказывающие начальное приближение: алгоритмы с предсказанием и иерархический поиск.

В качестве примера рассмотрим работу алгоритма 3SS. Многие из вышеперечисленных подходов будут отличаться от него только расположением точек для вычисления значений целевой функции, или числом шагов.

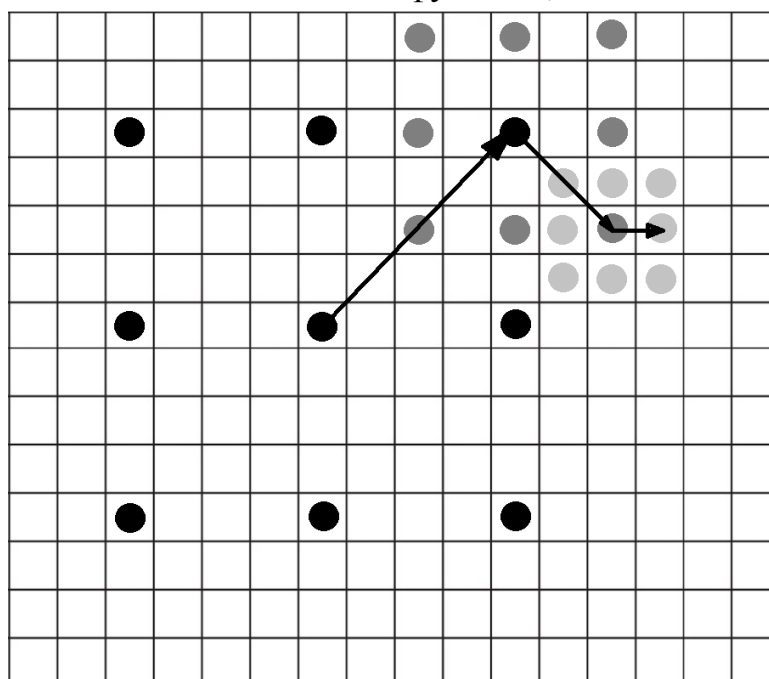


Рис. 2.3. Иллюстрация алгоритма 3SS.

Для каждого блока в кадре  $t$  на первом этапе вычисляется целевая функция ( $MAD$ ,  $MSE$ , или  $NCCF$ ) в девяти точках, то есть мера корреляции с девятью блоками в кадре  $t+1$  (черные точки на рис. 2.3), отстоящих друг от друга на некоторый шаг  $d$  и формирующих квадрат. Затем, уменьшив шаг  $d$  вдвое, вычисляют еще 8 значений целевой функции относительно точки, принадлежащей блоку, который имеет наибольшую корреляцию с блоком кадра  $t$  – минимум целевой функции (темно-серые точки на рис. 2.3). На последнем, третьем, шаге операцию повторяют, снова уменьшив шаг  $d$  в два раза. В результате, для каждого блока в кадре  $t$  находят наиболее похожий на него блок в кадре  $t+1$  и соответственно определяют вектор движения.

Разумеется, ни один алгоритм не гарантирует безошибочного нахождения векторов движения. На практике негативное влияние на корректность определения векторов могут оказывать шумы, блок может сместиться на расстояние, выходящее за границы области поиска. Все эти факторы приводят к появлению ошибочно найденных векторов, которые затрудняют сегментацию.

Для снижения влияния аномальных векторов движения на результат обработки, полученное поле векторов целесообразно обработать – выполнить пространственную или временную фильтрацию.

**Рекурсивная векторная медианная фильтрация** обеспечивает высокую эффективность обработки поля векторов в пределах одного кадра (рис 2.4).

Под медианой множества векторов понимается такой вектор из рассматриваемого множества, у которого сумма расстояний до всех других минимальна. Расстояние между двумя векторами,  $\mathbf{U}(x_u, y_u)$  и  $\mathbf{V}(x_v, y_v)$  вычисляется на основе нормы  $L_2$ :

$$\|\mathbf{U} - \mathbf{V}\|_{l_2} = \sqrt{(x_u - x_v)^2 + (y_u - y_v)^2}$$

В применении к задаче удаления аномальных векторов под медианной фильтрацией понимается замена каждого вектора движения векторной медианой множества, составленного из самого вектора и восьми его ближайших соседей. При вычислении расстояний используются только ненулевые векторы, т.е. каждый ненулевой вектор превращается в векторную медиану, вычисленную с помощью его восьми ненулевых соседей (рис. 2.4). Это делается для того, чтобы избежать замены ненулевых векторов на нулевые, когда в этом соседстве доминируют нулевые векторы.





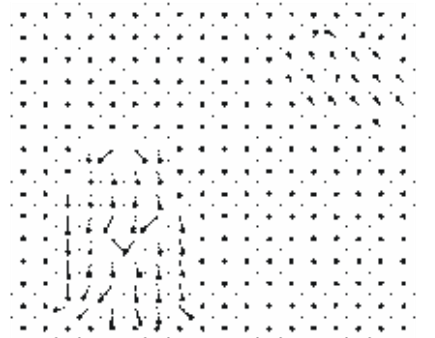
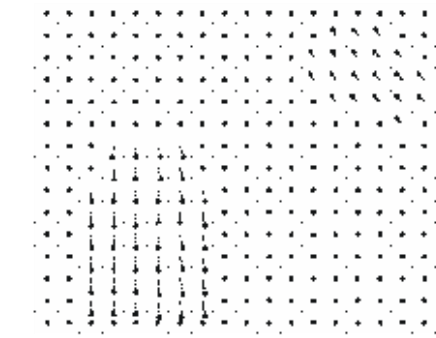

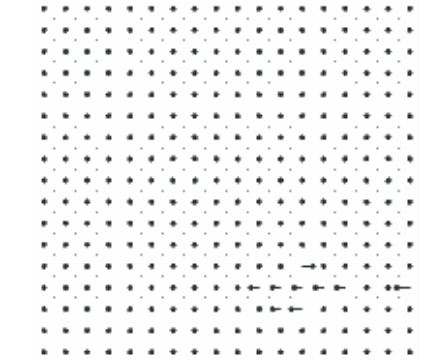
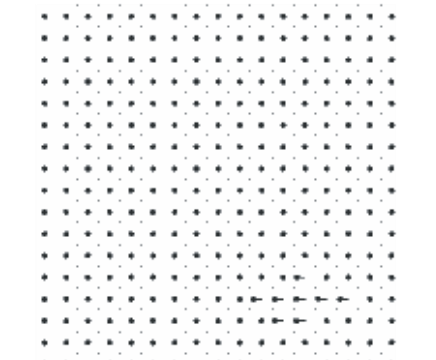
Вид кадра	Исходное поле векторов	Поле векторов после фильтрации
		
		
		

Рис. 2.4. Медианная фильтрация полей векторов движения

Пространственная медианная фильтрация поля векторов движения позволяет повысить точность сегментации объектов интереса за счет устранения аномальных векторов движения и вызванных ими разрывов внутри сегментированного объекта. Однако объекты, которые по размерам меньше маски медианного фильтра, могут быть потеряны.

На заключительном этапе алгоритма сегментации объектов на основе векторов движения найденные векторы, прошедшие фильтрацию, объединяют в

кластеры по признаку сонаправленности и пространственной связанности.

**Алгоритм кластеризации** состоит из следующих шагов.

1. Последовательно перебирают все блоки изображения, для блоков с ненулевым вектором движения проверяют наличие метки, если метка отсутствует, то для данного блока формируют новую метку.
2. Вокруг блока с созданной меткой рассматривают 8 его соседей. Если встречается ненулевой вектор и принимается решение, что векторы сонаправленные, то текущая метка присваивается соседнему блоку и шаг 2 повторяется. В противном случае повторяют шаг 1.

Процедуру выполняют до тех пор, пока существуют неразмеченные блоки. Результатом обработки являются сегментированные движущиеся объекты.

Таким образом, алгоритм сегментации объектов на основе векторов движения состоит из следующих шагов:

- разбиение изображения на блоки;
- определение векторов движения;
- пространственная фильтрация поля векторов движения;
- кластеризация блоков, для которых определены ненулевые векторы движения, определение их принадлежности к соответствующим объектам.

### **Задание на моделирование**

Разработать программу, моделирующую алгоритм сегментации объектов на основе векторов движения. Исходными данными являются два кадра видеопоследовательности со статичным фоном, на которых присутствует пара разнонаправленно перемещающихся объектов. В программе должны быть реализованы следующие процедуры с выводом результата на экран:

- загрузка пары кадров,
- разбиение кадров на блоки,
- реализация корреляционного алгоритма поиска векторов движения (с выводом векторов поверх изображения), с использованием в качестве целевой функции по указанию преподавателя MAD, или MSE (среднюю квадратичную ошибку) и одну из процедур для организации поиска (по указанию преподавателя): полный перебор, 3SS, 4SS, логарифмический, или ортогональный поиск.
- фильтрация поля векторов движения (демонстрация результата на изображении),

- кластеризация векторов и сегментация объектов (с отображением результата на кадре).

### **Контрольные вопросы**

1. Дайте определение векторам движения.
2. На основании какого предположения сформировано основное уравнение оптического потока?
3. Перечислите основные ограничения уравнения оптического потока.
4. Какие группы методов нахождения векторов движения вы знаете?
5. Приведите пример корреляционного метода.

### 1.3. СИНТЕЗ ПАНОРАМНЫХ ИЗОБРАЖЕНИЙ

Формированием панорамных изображений стали заниматься практически сразу после появления технологии фотографии. Первые панорамные фотоаппараты были созданы уже в середине XIX века. С наступлением эры цифровой видеотехники, решение задачи получения панорамных снимков стало еще более востребованным.

Применительно к системам видеонаблюдения говорят о задаче формирования не фото, а видеопанорамы: единого видеоизображения, получаемого от нескольких источников (камер), или от одной поворотной камеры. В случае синтеза видеопанорамы с помощью нескольких камер, при достаточной частоте обновления изображения (25 кадров в секунду и выше) говорят о видеопанораме в реальном времени. При применении поворотной камеры панорама не будет отображать наблюдаемую сцену в реальном времени.

Видеопанорамное изображение в системах наблюдения целесообразно осуществлять, когда размер объекта, или площадь зоны наблюдения превышает поле зрения камеры. Представление видеоданных в виде отдельных фрагментов, полученных от несвязанных камер, затрудняет восприятие оператором происходящего, снижает быстроту принятия решений, требует от него большего внимания. Использование в данном случае широкоугольного объектива приведет к геометрическим искажениям в кадре при компенсации которых произойдет потеря разрешения.

Задача синтеза панорамы (предположим, из двух фрагментов) может быть решена путем корреляционного совмещения одного изображения с другим. Причем, совмещаемое изображение должно иметь степени свободы по масштабу и углам поворота. Последовательно трансформируя один из фрагментов (изменяя масштаб и углы поворота с некоторым шагом) и совмещая его (изменяя смещение одного снимка относительно второго) с другим находят максимум отклика корреляционной функции в некоторой точке  $r$ , тем самым определяя оптимальные параметры для сшивки изображений.

$$r = \frac{\sum_{x,y} (L_1(x,y) \cdot L_2(x,y))}{\sqrt{\sum_{x,y} (L_1(x,y))^2 \cdot \sum_{x,y} (L_2(x,y))^2}}$$

где  $L_1$  – опорное изображение,  $L_2$  – совмещаемое изображение.

Данный метод, однако, мало применим на практике. Во-первых, он очень чувствителен к шумам и неодинаковой яркости фрагментов. Во-вторых, даже при фрагментах небольшого размера он требует огромного количества вычислительных ресурсов.

Общепринятый алгоритм для синтеза панорам состоит из следующих шагов.

1. Нахождение характерных точек (в англоязычной литературе используют термин «ключевая» точка «key point») на исходных изображениях в зоне перекрытия. Под характерной точкой понимают некоторый малый фрагмент изображения, в котором как значение яркостного градиента, так и производная (скорость изменения) градиента по направлению высоки. Как правило, характерных точками являются различные углы на изображениях (рис. 3.1).

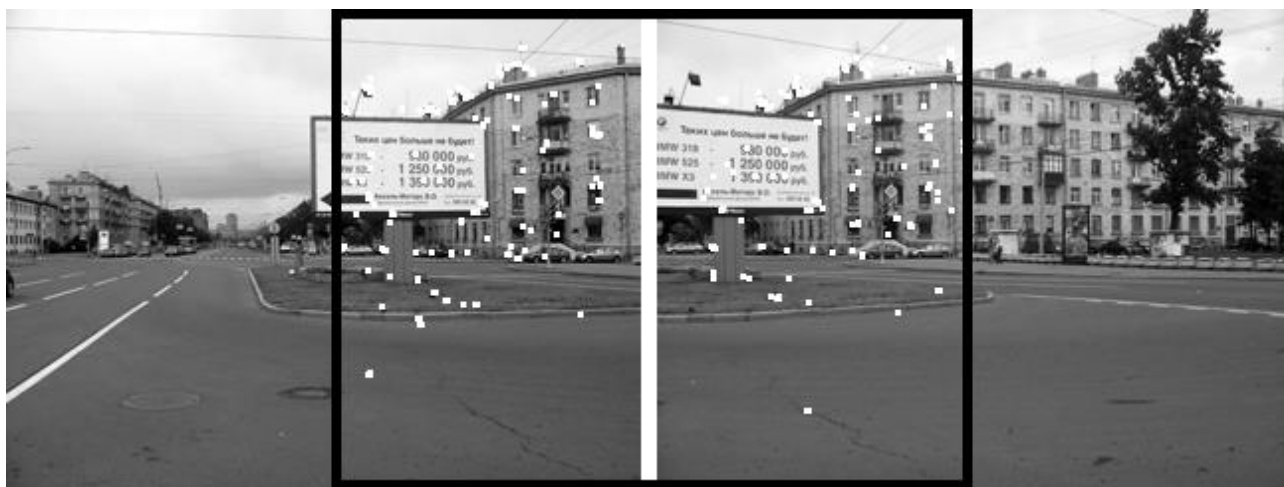


Рис. 3.1. Характерные точки сшиваемых изображений  
(зона перекрытия показана рамкой)

2. Нахождение в зоне перекрытия одних и тех же особенностей на различных снимках (рисунок 3.2). То есть определение соответствующих друг другу пар характерных точек

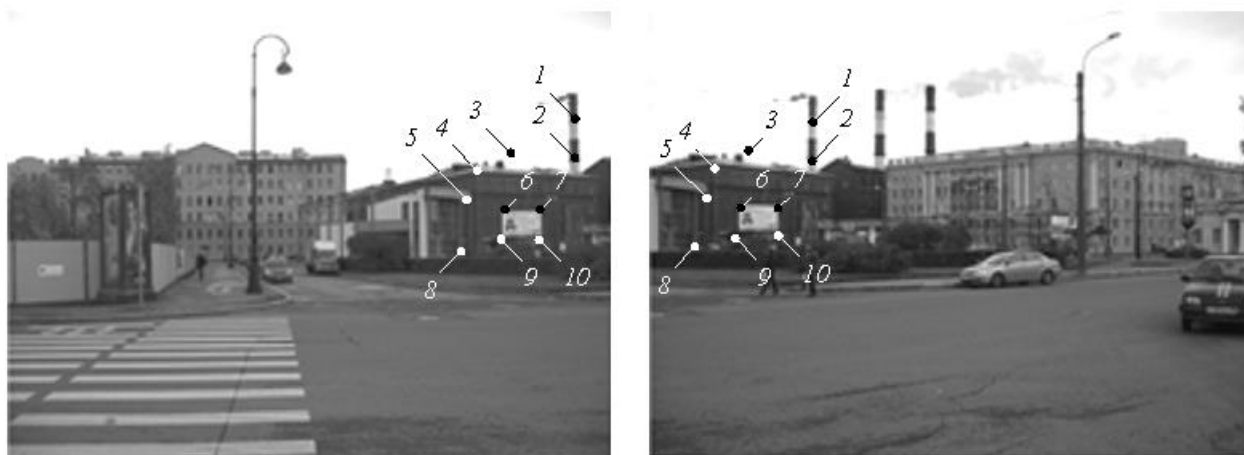


Рис. 3.2. Пары согласованных точек на сшиваемых изображениях  
Согласованные пары точек — это основа для синтеза панорамы. С помощью установленных пар на последующих этапах решают задачи



калибровки, трансформации и объединения снимков. В случае синтеза панорамы из нескольких кадров (частей) при неизвестном заранее расположении каждого снимка, на данном этапе проводят процедуру регистрации – идентификации местоположения отдельных изображений на общей панораме и установление взаимных соответствий характерных точек.

3. В современных программных пакетах следующим шагом алгоритма сшивки является калибровка изображений. Эта процедура направлена на минимизацию искажений объектива, оптических дефектов, различий экспозиции. С помощью информации о согласованных парах характерных точек минимизируют влияние дисторсии (геометрических искажений) объектива на точность сшивки панорамы.
4. Ключевым этапом является процедура идентификации параметров уравнений трансформации изображений и последующее преобразование фрагментов с объединением в единую панораму. Данный этап требует задания вида трансформации, которое определяется типом создаваемой панорамы. Например, при отсутствии, или несущественности перспективных искажений у фрагментов имеет смысл использовать аффинное преобразование. Примером такого изображения может быть «сшивка» панорамы из отсканированных частей единого документа (географической карты, картины, и т.п.). В общем случае применяют перспективное преобразование, учитывающее все возможные искажения снимков.
5. Заключительным этапом является блэндинг. Это комплексная процедура, направленная на повышение визуального качества панорамы, включающая выравнивание яркости и цветовой палитры фрагментов, маскирование «швов», удаление «призраков» (движущихся объектов). Кроме того, в блэндинг включают и процедуру проецирования панорамы на заданную поверхность – сферическую, цилиндрическую, эквидистантную и пр.

Простейшим примером, демонстрирующим процесс синтеза единого изображения, является построение панорамы из двух фрагментов. Для нахождения характерных точек существует множество алгоритмов – «детекторов». Самыми распространенным являются SIFT и SURF. Базовым детектором, лежащем в основе многих других, более совершенных является угловой детектор Харриса. Угол на изображении интересен тем, что по нему легко локализовать соответствующую особенность на различных снимках. В

угловой точке градиент яркости меняет свое направление в предельном случае на 90°.

Чтобы отличить угловую точку от наклонной линии, которая тоже имеет производные по обоим направлениям, в автоматическом экстракторе углов в каждой точке вычисляют собственные значения  $\lambda_1$  и  $\lambda_2$  матрицы гессиан

$$\mathbf{H} = \begin{bmatrix} \sum_{x,y} \left( \frac{\partial^2 L_{(x,y)}}{\partial x^2} \right) & \sum_{x,y} \left( \frac{\partial^2 L_{(x,y)}}{\partial x \partial y} \right) \\ \sum_{x,y} \left( \frac{\partial^2 L_{(x,y)}}{\partial x \partial y} \right) & \sum_{x,y} \left( \frac{\partial^2 L_{(x,y)}}{\partial y^2} \right) \end{bmatrix},$$

где суммы берут по заранее заданному размеру окна (например, блоку 5x5 пикселей).

Затем вычисляют коэффициент обусловленности  $cond(\mathbf{H}) = \lambda_{\max}/\lambda_{\min}$ . Когда коэффициент обусловленности матрицы  $\mathbf{H}$  близок к единице, причем оба собственных значения большие, фрагмент считают угловым. Если фон однородный - оба собственных значения близки к нулю. Для изображения наклонной линии одно из собственных значений будет нулевым.

Реализация этого алгоритма сопряжена с существенными вычислительными затратами. Чтобы сократить объем вычислений используют двухшаговый алгоритм извлечения угловых точек. На первом шаге применяют «функцию углового отклика»  $CR = \min|dL/dx, dL/dy|$ , представляющую собой абсолютное значение меньшей составляющей вектора градиента. Матрицу  $\mathbf{H}$  вычисляют только для потенциальных характерных точек, где  $CR$  превышает заранее заданный порог. Все предполагаемые угловые фрагменты, найденные на первом шаге, далее оценивают через вычисление собственных значений. Первый шаг играет главную роль в сокращении объема вычислений, требуемых на втором шаге. Степень сокращения зависит от содержания изображений и значения порога, заданного на первом шаге. Как правило, второй шаг применяют менее чем к пяти процентам площади изображения. В результате реализации этих процедур автоматически находят некоторое число характерных точек в области перекрытия сшиваемых изображений, достаточное для вычисления параметров преобразования.

После выделения характерных точек на двух соседних изображениях формируют пары согласованных характерных точек, представляющих на изображениях одни и те же области. Для этого производят корреляционное сопоставление блоков с центрами в найденных характерных точках. Каждый

блок первого изображения сравнивают со всеми блоками второго изображения в зоне перекрытия. Мера подобия двух блоков с центрами в характерных точках  $p_n(x_p, y_p)$  и  $q_m(x_q, y_q)$

$$SM(p_n, q_m) = \frac{D_x(p_n, q_m) + D_y(p_n, q_m)}{2},$$

где  $m, n$  – число пикселей в сравниваемых блоках по вертикали и горизонтали, соответственно,

$$D_x(p_n, q_m) = 1 - \frac{\sum_{i=1}^m \sum_{j=1}^n \left| \frac{dL_p(x_{p,j}, y_{p,i})}{dx} - \frac{dL_q(x_{q,j}, y_{q,i})}{dx} \right|}{\sum_{i=1}^m \sum_{j=1}^n \frac{dL_p(x_{p,j}, y_{p,i})}{dx} + \sum_{i=1}^m \sum_{j=1}^n \frac{dL_q(x_{q,j}, y_{q,i})}{dx}},$$

$$D_y(p_n, q_m) = 1 - \frac{\sum_{i=1}^m \sum_{j=1}^n \left| \frac{dL_p(x_{p,j}, y_{p,i})}{dy} - \frac{dL_q(x_{q,j}, y_{q,i})}{dy} \right|}{\sum_{i=1}^m \sum_{j=1}^n \frac{dL_p(x_{p,j}, y_{p,i})}{dy} + \sum_{i=1}^m \sum_{j=1}^n \frac{dL_q(x_{q,j}, y_{q,i})}{dy}}.$$

Значение  $SM(p_n, q_m)$  изменяется в диапазоне от нуля до единицы, причем  $SM(p_n, q_m)=1$  для идентичных блоков. В результате каждой точке из множества **P** сопоставляют точку из множества **Q** с мерой подобия  $SM(p_n, q_m)$  и формируют множество согласованных пар **PQ**.

Шумы в сигнале и различие конфигураций блоков из-за недостаточного перекрытия изображений приводит к образованию ложно согласованных пар во множестве **PQ**.

Существуют различные подходы для фильтрации ошибочно согласованных пар. Одним из самых простых и эффективных является проверка на соотношение значений меры подобия. Для каждой ключевой точки  $p_k$  на одном изображении определяют меры подобия со всеми точками на втором сшиваемом фрагменте. Далее берут два максимальных значения  $SM(p_k, q_{m1})$  и  $SM(p_k, q_{m2})$  и получают их отношение. Предполагается, что если для некоторой ключевой точки  $p_k$  одного фрагмента найдено истинное соответствие на втором, то мера подобия для данной точки будет значительно большей, чем для второй по близости характерной точки. Напротив, если характерная точка не имеет аналога на сшиваемом фрагменте, то ее значения меры подобия с двумя наиболее похожими точками будут приблизительно равны. Таким образом, пара точек считается согласованной, если выполняется условие:

$$\frac{SM(p_k, q_{m1})}{SM(p_k, q_{m2})} > t$$

где значение  $t$  обычно выбирают равным 2-3.

При синтезе панорамы из двух изображений, полученных от соседних камер с общим оптическим центром, используют модель перспективного преобразования, учитывающая все возможные искажения растров (масштаб, поворот, смещение, перспективные искажения). Уравнение трансформации:

$$\begin{pmatrix} x_{ip} \\ y_{ip} \\ 1 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{pmatrix} x_{iq} \\ y_{iq} \\ 1 \end{pmatrix},$$

где  $x_{ip}, y_{ip}$  и  $x_{iq}, y_{iq}$  – координаты соответствующих пикселей в трансформируемом и в опорном изображениях, соответственно;  $h_{11}, \dots, h_{33}$  – искомые параметры.

Для оценки параметров уравнения трансформации используют аппарат регрессионного анализа. На основании координат пар согласованных характерных точек ( $\{(x_{ip}, y_{ip}), (x_{iq}, y_{iq})\}, i=1..N, N>5$ ) и выбранной модели уравнения составляют матрицу плана эксперимента

$$\mathbf{A} = \begin{bmatrix} x_{1,p} & y_{1,p} & 1 & 0 & 0 & 0 & -x_{1,p} \cdot x_{1,q} & -y_{1,p} \cdot x_{1,q} & -x_{1,q} \\ 0 & 0 & 0 & x_{1,p} & y_{1,p} & 1 & -x_{1,p} \cdot y_{1,q} & -y_{1,p} \cdot y_{1,q} & -y_{1,q} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ x_{N,p} & y_{N,p} & 1 & 0 & 0 & 0 & -x_{N,p} \cdot x_{N,q} & -y_{N,p} \cdot x_{N,q} & -x_{N,q} \\ 0 & 0 & 0 & x_{N,p} & y_{N,p} & 1 & -x_{N,p} \cdot y_{N,q} & -y_{N,p} \cdot y_{N,q} & -y_{N,q} \end{bmatrix}.$$

Вектор неизвестных параметров  $\mathbf{h}=(h_{11}, \dots, h_{33})^T$  определяют методом наименьших квадратов путем решения системы уравнений:

$$\mathbf{A} \cdot \mathbf{h} = \mathbf{b},$$

где  $\mathbf{b}$  - вектор откликов (значений целевой функции).

Так как матрица  $\mathbf{A}$  не квадратная, она не имеет обратной матрицы. Поэтому используют псевдообратную матрицу  $\mathbf{A}^+=(\mathbf{A}^T \cdot \mathbf{A})^{-1} \cdot \mathbf{A}^T$ . Тогда

$$\mathbf{h} = (\mathbf{A}^T \cdot \mathbf{A})^{-1} \cdot \mathbf{A}^T \cdot \mathbf{b} = \mathbf{A}^+ \cdot \mathbf{b}. \quad (3.1)$$

Для задачи наименьших квадратов данное выражение называют нормальной системой относительно  $\mathbf{h}$ .

Расчет псевдообратной матрицы - трудоемкая процедура, требующая значительного объема вычислений. Существует большое количество численных методов решения данной задачи. Популярны подходы, основанные на ортогональных разложениях матрицы  $\mathbf{A}$ . В задаче построения панорамных изображений часто используют сингулярное разложение.

С помощью какого-либо численного метода (например, *QR*-алгоритма, который реализован в большинстве математических пакетов) матрицу  $\mathbf{A}$  представляют в виде произведения:

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T,$$

где  $\mathbf{\Sigma}$  - диагональная матрица.

Элементы на диагонали матрицы  $\mathbf{\Sigma}$  являются сингулярными числами матрицы  $\mathbf{A}$ , причем расположены по убыванию.  $\mathbf{U}$ ,  $\mathbf{V}^T$  - матрицы, столбцы которых содержат левые и правые сингулярные векторы для соответствующих им значений собственных чисел на диагонали матрицы  $\mathbf{\Sigma}$ . Причем

$$\mathbf{V} \cdot \mathbf{V}^T = \mathbf{I}, \quad \mathbf{U} \cdot \mathbf{U}^T = \mathbf{I},$$

где  $\mathbf{I}$  - единичная матрица, так как  $\mathbf{U}$ ,  $\mathbf{V}$  – ортогональные матрицы.

Столбцы матрицы  $\mathbf{U}$  - это собственные векторы  $\mathbf{A}^T \cdot \mathbf{A}$ , а столбцы матрицы  $\mathbf{V}^T$  – собственные векторы  $\mathbf{A} \cdot \mathbf{A}^T$ .

Имея сингулярное разложение, можно представить решение задачи наименьших квадратов в явном виде, не прибегая к трудоемкому обращению матриц:

$$\begin{aligned} \mathbf{A}^+ &= (\mathbf{V} \cdot \mathbf{\Sigma} \cdot \mathbf{U}^T \cdot \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T)^{-1} \cdot \mathbf{V} \cdot \mathbf{\Sigma} \cdot \mathbf{U}^T = \\ &= (\mathbf{\Sigma} \cdot \mathbf{V}^T)^{-1} \cdot (\mathbf{V} \cdot \mathbf{\Sigma})^{-1} \cdot (\mathbf{V} \cdot \mathbf{\Sigma}) \cdot \mathbf{U}^T = \mathbf{V} \cdot \mathbf{\Sigma}^{-1} \cdot \mathbf{U}^T. \end{aligned}$$

Результат обращения диагональной матрицы

$$\mathbf{\Sigma}^{-1} = \mathbf{diag}\left(\frac{1}{\sqrt{\lambda_1}}, \dots, \frac{1}{\sqrt{\lambda_n}}\right),$$

где  $\lambda_1, \dots, \lambda_n$  - собственные числа. Таким образом, вектор решения МНК

$$\mathbf{x} = \mathbf{V} \cdot \mathbf{\Sigma}^{-1} \cdot \mathbf{U}^T \cdot \mathbf{y}.$$

Определив матрицу правых собственных векторов  $\mathbf{V}^T$  и выбрав ее последний столбец, которому соответствует минимальное собственное значение  $\mathbf{A}$ , расположенное на последней строке диагональной матрицы  $\mathbf{\Sigma}$ , получают вектор параметров, дающий минимальную невязку для выражения (3.1).

Таким образом, параметры уравнения трансформации определяют из сингулярного разложения матрицы  $\mathbf{A}$ , как значения последнего (девятого) столбца матрицы  $\mathbf{V}^T$ . Для оценки параметров уравнения трансформации используют алгоритм устойчивой оценки *RANSAC*:

1. Случайным образом из множества согласованных точек выбираются пять пар. По ним определяют параметры уравнения трансформации (создают текущую гипотезу уравнения). Для каждой пары множества согласованных точек вычисляют ошибку трансформации

$$E_i = \begin{pmatrix} x_{ip} \\ y_{ip} \\ 1 \end{pmatrix} - \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{pmatrix} x_{iq} \\ y_{iq} \\ 1 \end{pmatrix}.$$

2. Подсчитывают число пар характерных точек (так называемых «не выбросов», в англоязычной литературе «*inliers*»), для которых ошибка трансформации  $E$  при текущей гипотезе не превышает некоторого порога (обычно 2-3 пикселя), остальные пары отбрасывают.

3. Шаги 1, 2 повторяют заранее определенное число итераций. На каждой итерации получают и проверяют новую гипотезу с помощью очередных случайно выбранных пяти пар согласованных точек. В результате запоминают параметры гипотезы, для которой получилось больше всего «не выбросов».

4. По пяти парам согласованных точек, с помощью которых была получена гипотеза, запомненная на шаге 3, вновь определяют параметры уравнения трансформации.

5. Исключают выбросы (то есть пары точек, для которых ошибка превышает порог) и пересчитывают параметры уравнения трансформации, но уже с учетом всех пар «не-выбросов».

По найденным параметрам уравнения выполняют трансформации изображений и «сшивку» панорамы. Пример снимков-фрагментов и созданного единого изображения приведен на рисунке 3.3



Рис. 3.3. Пример созданной панорамы (нижний ряд) из двух фрагментов (верхний ряд)

## **Задание на моделирование**

Разработать программу, реализующую синтез панорамы из двух фрагментов с помощью библиотеки OpenCV.

Необходимо подобрать два фрагмента (зона перекрытия изображений должна занимать не менее 25% их площади) и следуя инструкции из раздела 7 осуществить синтез панорамы.

При работе программы должны быть визуализированы основные этапы:

- изображения с найденными ключевыми точками;
- результат процедуры поиска согласованных пар;
- результат трансформации изображения (один из снимков является опорным и остается без изменений, трансформируют только второй фрагмент);
- результат построения панорамы.

## **Контрольные вопросы**

6. Перечислите основные этапы создания панорамного изображения.
7. Какие точки на изображении являются характерными, или ключевыми?
8. Объясните принцип работы уголкового детектора Харриса?
9. Дайте определение перспективному преобразованию изображения?
10. Для чего служит и в чем заключается алгоритм RANSAC?

## **ГЛАВА 2. МЕТОДЫ ИНТЕЛЛЕКТУАЛЬНОГО АНАЛИЗА ДАННЫХ**

### **2.1. КЛАССИФИКАЦИЯ ОБЪЕКТОВ С ПОМОЩЬЮ РАССТОЯНИЯ МАХАЛАНОБИСА**

Интеллектуальный анализ данных («data mining») – это область знания, включающая в себя различные методы и алгоритмы для поиска и выделения из данных какой-либо целевой информации.

Найденная информация может представлять ценность для пользователя, как материал, на основании которого он самостоятельно может делать какие-либо заключения, или может служить исходными данными для построения и работы интеллектуальной компьютерной системы.

В интеллектуальный анализ данных входят:

- статистические методы;
- методы машинного обучения.

Методы первой группы базируются на теории вероятности и математической статистике, они включают в себя корреляционный, дисперсионный и факторный анализ, байесовские методы, дискриминантный анализ и другие.

Методы машинного обучения основаны теории оптимизации, теории графов, численных методах.

Методы разделяют на

- обучение с учителем;
- обучение без учителя.

Обучение с учителем предполагает, что для задачи, требующей решения, существуют так называемые обучающие примеры – случаи, где входным данным сопоставлен корректный ответ. Существуют алгоритмы «обучения», то есть настройки параметров математических моделей (нейронных сетей, лесов решающих деревьев, опорных векторов и других) с помощью анализа имеющихся примеров. В результате «обученные» математические модели способны генерировать ответ при подаче на вход нового образца, не встречавшегося при обучении.

Методы обучения с учителем решают, например, задачи классификации и регрессии (частный случай прогнозирования). На этом основаны системы распознавания образов и детектирования объектов.



К обучению без учителя (случай, когда имеется выборка данных, но нет обучающих примеров) относят методы кластеризации и понижения размерности признакового пространства.

Простой и эффективный алгоритм классификации может быть построен на основе меры Махаланобиса, которая определяет расстояние между случайной величиной, описываемой вектором значений, и некоторым распределением случайных величин. Расстояние Махаланобиса обобщает евклидово и фактически позволяет установить сходство между некоторым объектом и выборкой объектов, соответствующих определенному классу.

Расстояние между объектом и выборкой определяются следующим образом.

$$d(\mathbf{x}) = \sqrt{(\mathbf{x} - \boldsymbol{\mu})^T \mathbf{S}^{-1} (\mathbf{x} - \boldsymbol{\mu})}$$

где  $\mathbf{x} = (x_1, x_2, \dots, x_n)$  – многомерный вектор значений признаков объекта,  $\boldsymbol{\mu} = (\mu_1, \mu_2, \dots, \mu_n)$  – вектор средних значений признаков объектов, составляющих выборку,  $\mathbf{S}$  – матрица ковариации признаков.

Квадратная симметричная матрица ковариации  $\mathbf{S}$  в данном случае состоит из попарных ковариаций средних значений признаков (элементов  $\boldsymbol{\mu}$ ).

$$\mathbf{S} = \begin{bmatrix} cov(\mu_1, \mu_1) & cov(\mu_1, \mu_2) & \dots & cov(\mu_1, \mu_m) \\ cov(\mu_2, \mu_1) & cov(\mu_2, \mu_2) & \dots & cov(\mu_2, \mu_m) \\ \dots & \dots & \dots & \dots \\ cov(\mu_m, \mu_1) & \dots & \dots & cov(\mu_m, \mu_m) \end{bmatrix}$$

Каждый элемент матрицы вычисляется как:

$$cov(\mu_n, \mu_k) = M((\mu_n - M(\mu_n))(\mu_k - M(\mu_k))),$$

где  $M$  – означает математическое ожидание.

В результате в матрице ковариации некоторой выборки (совокупности объектов одного класса, применительно к задаче классификации) на главной диагонали располагаются дисперсии признаков, а вне главной диагонали стоят значения, показывающие меру линейной зависимости признаков.

Нетрудно заметить, что если  $\mathbf{S}$  представляет собой единичную матрицу, то расстояние Махаланобиса вырождается в евклидово расстояние. В случае, если матрица ковариации является диагональной, расстояние становится стандартизованным евклидовым.

Отсюда следует вывод, что применение нормы Евклида в случае классификации возможно только тогда, когда все признаки имеют одну размерность и масштаб (что в результате и делает  $\mathbf{S}$  единичной матрицей). Иначе,

вклад признаков, измеряющихся в маленьком масштабе (например, от 0 до 100) будет полностью нивелирован вкладом признаков с большим масштабом (например, от 1000 до 100000).

Чтобы нивелировать негативный эффект от разного масштаба признаков проводят их нормализацию и в результате используют стандартизованное евклидово расстояние. Это расстояние позволяет работать с признаками в разном масштабе, но не учитывает линейную зависимость между ними. Разницу при использовании расстояния Махаланобиса и стандартизованного евклидова поясняет рисунок 4.1.

Белыми кружками обозначены объекты выборки, составляющие класс. Черными – объекты, для которых нужно определить расстояния до класса (центр обозначен черной звездочкой). Объект **В** будет ближе к классу, чем объект **А** по стандартизованной евклидовой метрике (пунктирная окружность), но дальше по метрике Махаланобиса (овал со сплошной линией).

Это связано с тем, что использование расстояния Евклида предполагает, что данные имеют изотропное нормальное распределение, а расстояние Махаланобиса допускает анизотропное распределение.

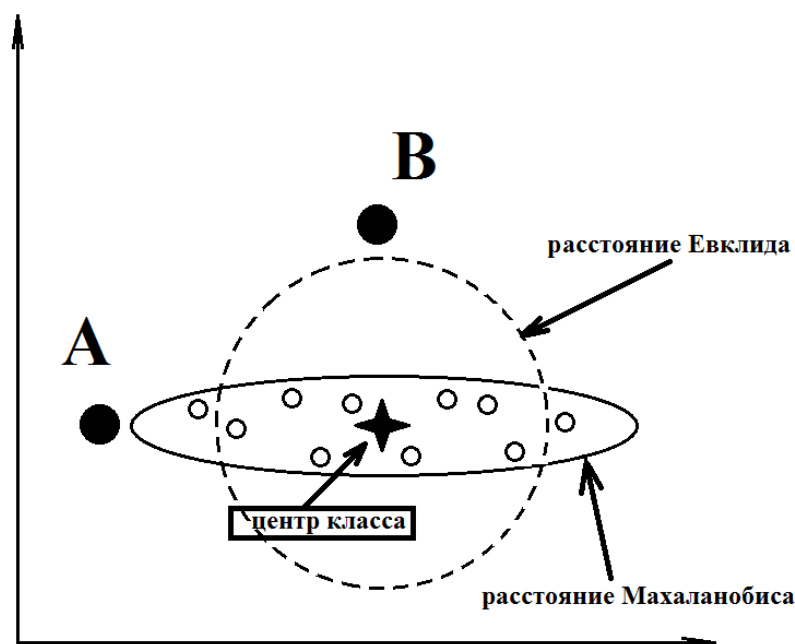


Рис. 4.1. Иллюстрация к использованию различных метрик при классификации объектов.

На практике в задачах классификации чаще всего ковариация между признаками объектов не нулевая, то предпочтительнее использовать именно расстояние Махаланобиса.

Таким образом задача классификации объекта  $\mathbf{x}$ , то есть определение его принадлежности к одному из имеющихся классов, сводится к оценке расстояний  $d(\mathbf{x})$  от объекта до каждого из классов и идентификации минимального.

Для проведения расчетов необходимо вычислить матрицы ковариации  $\mathbf{S}$  и векторы средних значений признаков  $\boldsymbol{\mu}$  для каждого класса. Это возможно сделать с помощью имеющихся *обучающих примеров* (выборки объектов, значения признаков которых, как и их принадлежность к классам известна). Неизвестные математические ожидания для классов объектов  $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_k, \mu_n)$  заменяются на их оценки:

$$\widehat{\mu}_k = \frac{1}{N_k} \sum_{i=1}^{N_k} x_i,$$

где  $N$  – число элементов в обучающей выборке для класса,  $x_i$  – значение признака, входящего в вектор обучающего примера  $\mathbf{x} = (x_1, x_2, x_i, x_n)$ . Далее вычисляются оценки для ковариационных матриц.

В машинном обучении с учителем популярны следующие методы, использующие метрику Махаланобиса.

- Линейный дискриминантный анализ
- Квадратичный дискриминантный анализ.

Линейный дискриминантный анализ служит для разделения объектов на два класса. В нем принимается допущение о гомоскедастичности классов (то есть, о равенстве ковариационных матриц двух классов).

$$\mathbf{S}_1 = \mathbf{S}_2 = \mathbf{S}_\Sigma.$$

Классификационное правило элементарно: объект относят к классу, до которого меньше расстояние Махаланобиса.

Квадратичный дискриминантный анализ позволяет проводить многоклассовую классификацию и не требует равенства ковариационных матриц классов.

Объект считается принадлежащим классу  $k$ , до которого меньше значение

$$f_k = (\mathbf{x} - \boldsymbol{\mu}_k)^T \mathbf{S}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) + \ln(\det(\mathbf{S}_k^{-1})).$$

где  $\det$  – обозначение определителя матрицы. Данная формула выводится из вероятностной модели, которая моделирует условное распределение данных.

### Задание на моделирование

Разработать программу, моделирующую алгоритм классификации объектов с помощью расстояния Махаланобиса. В качестве исходных данных

нужно использовать базу данных ирисов Фишера (<http://archive.ics.uci.edu/ml/datasets/Iris>). База содержит образцы для трех различных классов ирисов. Необходимо выполнить следующие действия.

1. Загрузить из текстовых файлов данные в программу, при этом корректно обработать принадлежность к классам и значения признаков.
2. Данные для каждого класса случайным образом разделить на две подгруппы, содержащие 90% и 10% образцов.
3. По группам, содержащим 90% выборки оценить векторы средних значений и матрицы ковариаций признаков.
4. Реализовать процедуру классификации по расстоянию Махаланобиса, обучить классификатор.
5. С помощью тестовой выборки (10% образцов) реализовать процедуру тестирования точности классификатора.
6. Вывести результат.

### **Контрольные вопросы**

6. Сформулируйте основное отличие методов обучения с учителем от методов обучения без учителя.
7. Дайте определение ковариации. Что означает ковариация признаков объекта?
8. Объясните, чему соответствуют элементы матрицы ковариации в методе классификации с помощью расстояния Махаланобиса?
9. Прокомментируйте, как определяется расстояние Махаланобиса.
10. Объясните, как связаны расстояние Махаланобиса и расстояние Евклида.

## 2.2. КЛАСТЕРИЗАЦИЯ МЕТОДОМ К-СРЕДНИХ

Одной из основных задач, которую решают методы машинного обучения без учителя является кластеризация.

Кластеризация (кластерный анализ) – многомерная статистическая процедура, выполняющая сбор данных, содержащих информацию о выборке объектов, и затем упорядочивающая объекты в однородные группы. Результат кластеризации представлен на рисунке 4.1. Слева изображена исходная выборка объектов, обладающих двумя признаками (оси  $x1$ ,  $x2$ ), а справа – результат кластеризации: объекты разделены на три группы (обозначены разной заливкой), согласно близости значений признаков.

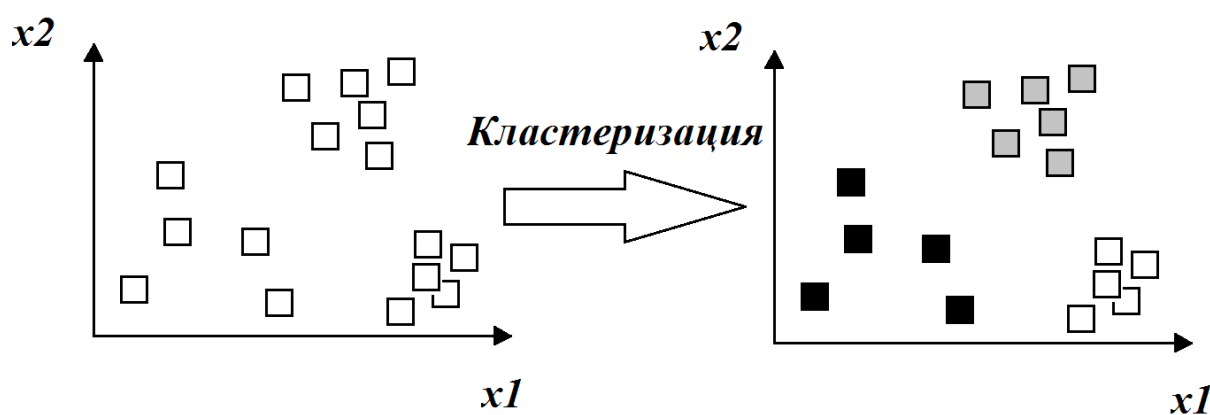


Рис. 5.1. Результат кластеризации объектов

Задача кластеризации часто встречается при разработке интеллектуальных видеосистем. Например, путем кластеризации можно осуществить сегментацию изображения по цветовым и яркостным признакам. Выполнение кластерного анализа связано с решением следующих задач:

- определение выборки (совокупности объектов) для кластеризации;
- определение признаков, по которым будет осуществляться разбиение образцов на группы;
- установление числа групп (кластеров);
- выбор метрики для определения сходства между образцами (например, евклидово расстояние);
- выбор метода кластеризации.

Выбор признакового пространства очень важный шаг в анализе. Необходимо выбрать те признаки, которые существенно различны у формируемых групп, и по которым целесообразно проводить разделение. В некоторых задачах выбор очевиден. Допустим, сегментацию нефтяных пятен на

поверхности воды логично осуществлять по цвету. В других ситуациях выбор нетривиален и для его осуществления нужно выполнить статистический анализ признаков с выявлением наиболее эффективного набора для кластеризации.

Решение задачи упрощается, если заранее известно на сколько групп нужно разделить объекты. В противном случае, выполняют дополнительное исследование для определения числа кластеров (например, используют «метод локтя»).

Алгоритмы кластеризации принято разделять на группы по применяемым подходам: вероятностные, генетические, графовые и другие.

В данном разделе рассматривается один из самых популярных алгоритмов – К-средних. В результате работы он разбивает множество элементов векторного пространства на заранее известное число кластеров  $k$ . Алгоритм состоит из следующих шагов.

1. Случайным образом устанавливают точки – центры кластеров в векторном пространстве признаков.
2. Вычисляют евклидовы расстояния  $d_i$ , ( $i = 1..k$ ), где  $k$  – число кластеров, от каждого объекта (с вектором значений координат  $\mathbf{x}=(x_1, x_2,...,x_n)$ ) до каждого центра кластера (с вектором значений координат  $\mathbf{y}=(y_1, y_2,...,y_n)$ ).

$$d_i(x, y) = \sqrt{\sum_n (\mathbf{x}_n - \mathbf{y}_{i,n})^2}$$

По минимальному расстоянию до центра устанавливают принадлежность каждого объекта некоторому кластеру.

3. На основании определенной принадлежности объектов кластерам заново вычисляют (уточняют) координаты каждого центра кластера.
4. Повторяют шаги 2-3 до тех пор, пока алгоритм не сойдется (координаты центров кластеров перестанут изменяться).

Фактически, при работе алгоритм минимизирует суммарное квадратичное отклонение объектов от центров кластеров  $E$ .

$$E = \sum_{i=1}^k \sum_{\mathbf{x}_j \in K_i} (\mathbf{x}_j - \mathbf{y}_i)^2.$$

где  $k$  – число кластеров,  $K_i$  – кластеры, сформированные на момент расчета  $E$ .

Алгоритм сходится, так как количество возможных разбиений множества объектов конечно, а на каждом шаге суммарное квадратичное отклонение  $E$  уменьшается.

В работе К. Бишоп (*Bishop*, см. в списке литературы) приведена иллюстрация работы алгоритма (см. рисунок 5.2).

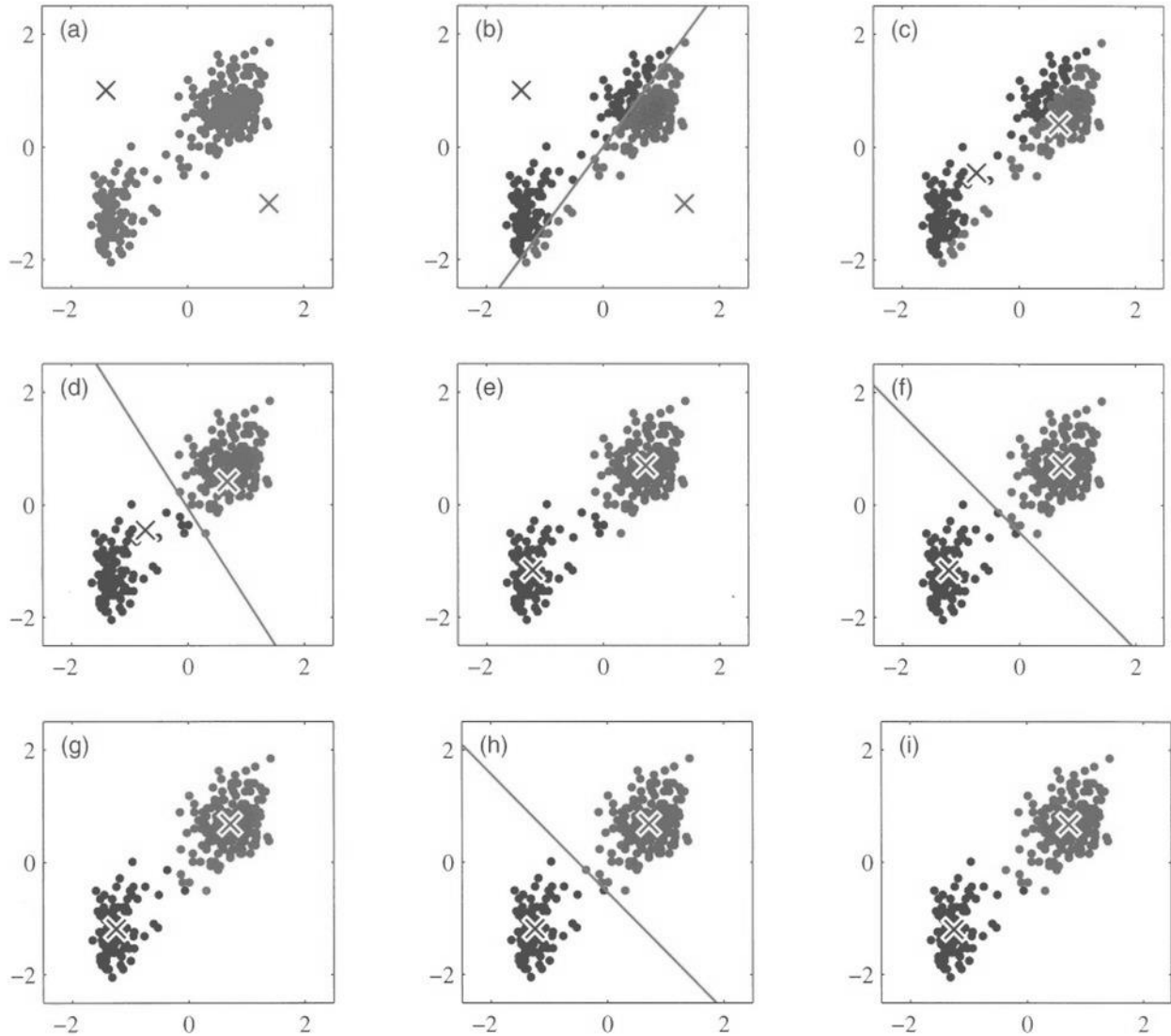


Рис. 5.2. Итерации работы алгоритма кластеризации  $k$ -средних. От начальных бросков центров (a), до финального разбиения объектов на два кластера (i).

Алгоритм  $k$ -средних не лишен недостатков. Основные проблемы, помимо необходимости знать заранее число кластеров  $k$ , при его использовании следующие:

- алгоритм достигает минимума  $E$ , но не гарантирует, что этот минимум глобальный, следовательно, нельзя быть уверенным в том, что найденное разбиение оптимально;

- результат поиска минимума  $E$  зависит от начального броска центров кластеров, поэтому большинство модификаций  $k$ -средних связаны с оптимизацией начального выбора координат центров.

Тем не менее, алгоритм  $k$ -средних прекрасно справляется с большим количеством встречающихся на практике задач. За счет простоты и эффективности он является одним из самых популярным алгоритмом кластеризации.

### **Задание на моделирование**

Разработать программу, моделирующую алгоритм кластеризации  $k$ -средних. Исходными данными является изображение, содержащее объекты разного цвета на однотонном фоне. В программе должны быть реализованы следующие процедуры с выводом результата на экран:

- загрузка изображения,
- задание вручную числа кластеров  $k$  (число кластеров определяется преподавателем),
- реализация алгоритма  $k$ -средних, где в качестве признаков используются значения  $R$ ,  $G$ ,  $B$  пикселей, или значения яркости (определяется преподавателем); результатом является изображение-маска, содержащая  $k$  цветов, соответствующее сформированным кластерам.

### **Контрольные вопросы**

11. Поясните, чем отличаются методы обучения с учителем и без учителя?
12. Укажите, что является результатом решения задачи кластеризации?
13. Объясните, почему важно определить набор признаков для кластеризации?
14. Раскройте принцип работы алгоритма  $k$ -средних.
15. Укажите и прокомментируйте недостатки алгоритма  $k$ -средних.



## 2.3. РЕГРЕССИОННЫЙ АНАЛИЗ

В практике создания интеллектуальных телевизионных систем часто возникает необходимость осуществления регрессионного анализа. Регрессионный анализ – это статистический метод исследования влияния одной или нескольких независимых переменных на зависимую переменную. Независимые переменные иначе называют предикторами, а зависимые переменные — откликами.

Часто встречающейся задачей регрессионного анализа в практике интеллектуального анализа данных является оценка неизвестных параметров математических моделей по выборочным данным. Базовый метод для решения данной задачи называется методом наименьших квадратов (МНК).

МНК это математический метод, основанный на минимизации суммы квадратов отклонений некоторой функций от искомым переменных. В частности, метод наименьших квадратов может использоваться для «решения» (оптимального по критерию минимизации суммы квадратов отклонений) системы линейных уравнений.

Система линейных уравнений в матричной форме представляет собой уравнение вида:

$$\mathbf{Ax} = \mathbf{b}$$

где  $\mathbf{A}$  – матрица параметров размером  $m \times n$ , причем  $m > n$  (то есть, количество уравнений больше количества переменных),  $\mathbf{b}$  – известный вектор значений некоторой переменной, зависящий от вектора  $\mathbf{x}$ , значения которого требуется установить. Решением данной переопределенной системы в контексте МНК будет такой вектор  $\mathbf{Ax}$ , который минимально отличался бы от  $\mathbf{b}$  по критерию суммы квадратов отклонений. Так как  $\mathbf{A}$  – прямоугольная матрица, то для решения задачи необходимо получить псевдообратную матрицу  $\mathbf{A}^+$ :

$$\mathbf{A}^+ = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$$

Вектор  $\mathbf{x}$ , можно идентифицировать путем:

$$\mathbf{x} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \cdot \mathbf{b}$$

Для оценки неизвестных параметров при аппроксимации данных полиномиальной моделью в регрессионном анализе метод наименьших квадратов применяют в следующем виде. Пусть выбрана некоторая модель и переменная  $y$  зависит от  $x$  следующим образом:

$$y = a_0 x^0 + a_1 x^1 + \dots + a_n x^n,$$

причем имеются наблюдения – известные значения  $y$  при определенном  $x$ .

Тогда для оценки параметров  $a_0, a_1, \dots, a_n$  необходимо решить следующую систему:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & x_2^3 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & \dots & \dots & \dots & x_N^n \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_N \end{bmatrix}.$$

Первая (левая) матрица в данном уравнении называется матрицей плана эксперимента (часто используют обозначение  $\mathbf{F}$ ) и содержит соответствующие полиномиальной модели известные значения  $x$ , матрица-столбец  $\mathbf{y}$  – содержит известные значения «откликов» (определенные  $y_i$  при  $x_i$ ). Параметры  $\mathbf{a} = (a_0, a_1, \dots, a_n)$  вычисляют как:

$$\mathbf{a} = (\mathbf{F}^T \mathbf{F})^{-1} \mathbf{F}^T \cdot \mathbf{y}$$

Применение аппарата регрессионного анализа может быть проиллюстрировано на практическом примере. В практике построения телевизионных систем возникает необходимость согласования растров отдельных каналов  $R$ ,  $G$  и  $B$ . Рассогласованность каналов может происходить из-за латеральных хроматических aberrаций, вносимых объективом.

Смещения, вызванные боковыми хроматическими aberrациями, являются точечными симметричными, их величина пропорциональна расстоянию от принципиальной точки (обычно центр кадра). Примеры приведены на рис. 6.1.

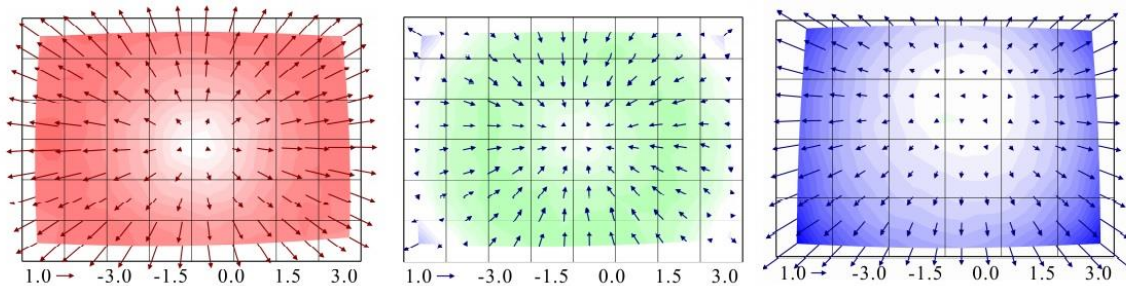


Рис. 6.1 – Векторы смещения в каналах  $R$ ,  $G$ ,  $B$ .

В результате проекция одной и той же точки для разных каналов не совпадает в пространственных координатах матрицы (и, соответственно, кадра). Это приводит к ухудшению визуального качества изображения из-за искажений цветов, заметных на резких границах (рис. 6.2).

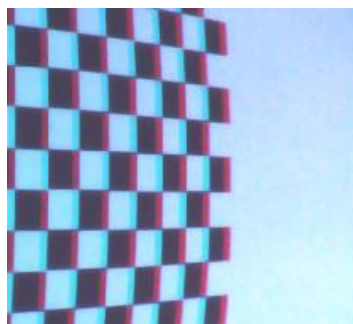


Рис. 6.2 – Цветовые aberrации на изображении.

Для компенсации латеральных aberrаций применяют следующий алгоритм.

1. Формируют калибровочный тест, примерный вид которого изображен на рисунке 6.3, и осуществляют его снимок исследуемой камерой. Полученное изображение разделяют на три слоя, представляющие каналы R, G, B. Зеленый канал используют в качестве опорного, к которому будут сводить красный и синий.

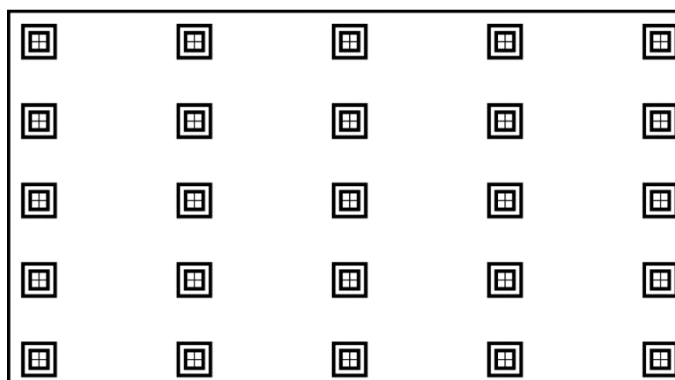


Рис. 6.3. Пример калибровочного теста для коррекции цветовых aberrаций.

2. Реализуют наложение на изображение опорного зеленого канала изображение красного канала. Далее выбирают последовательно набор точек (центров небольших блоков размером, например, 8 на 8 пикселей) на изображении красного канала и путем совмещения с зеленым каналом определяют смещения по вертикали и горизонтали для каждой точки. Процесс проиллюстрирован на рисунке 6.4.

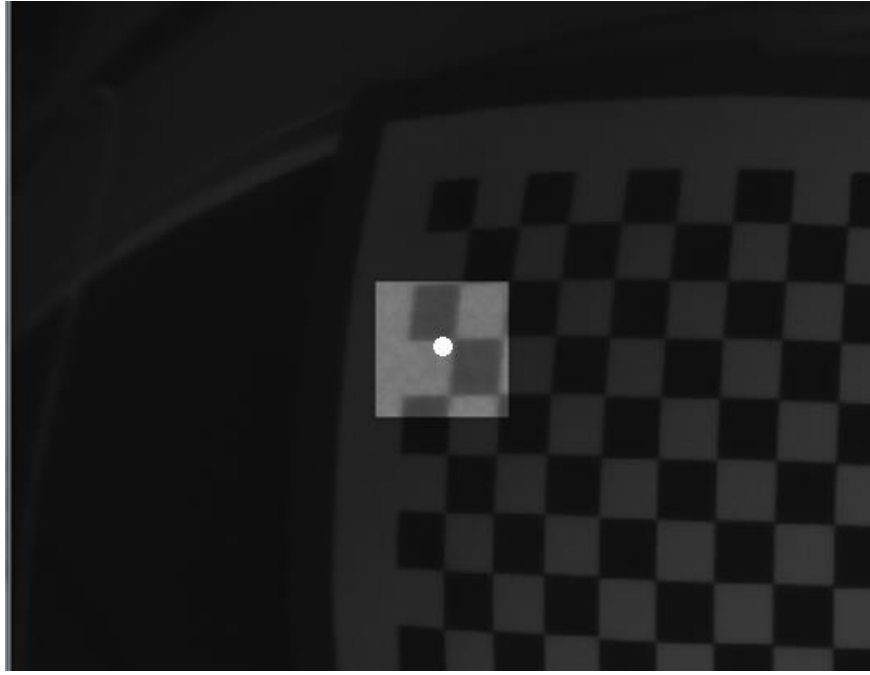


Рис. 6.4. совмещение блока красного канала с подложкой – зеленым каналом.

3. Оценивают коэффициенты корректирующего полинома, который позволяет трансформировать красный канал по образцу зеленого. Для этого, опираясь на найденные смещения  $(\Delta x_i, \Delta y_i)$  канала  $R$  относительно  $G$  в известных точках  $(x_i, y_i)$ , идентифицируют параметры моделей:

$$\Delta x = a_{0x} + a_{1x}x + a_{2x}y + a_{3x}xy + a_{4x}x^2 + a_{5x}y^2 + a_{6x}x^2y + a_{7x}y^2x + a_{8x}x^3 + a_{9x}y^3.$$

$$\Delta y = a_{0y} + a_{1y}x + a_{2y}y + a_{3y}xy + a_{4y}x^2 + a_{5y}y^2 + a_{6y}x^2y + a_{7y}y^2x + a_{8y}x^3 + a_{9y}y^3.$$

Коэффициенты вычисляются согласно МНК:

$$\mathbf{A}_x = (\mathbf{F}^T \mathbf{F})^{-1} \mathbf{F}^T \Delta \mathbf{x}$$

$$\mathbf{A}_y = (\mathbf{F}^T \mathbf{F})^{-1} \mathbf{F}^T \Delta \mathbf{y}$$

$$\mathbf{F} = \begin{bmatrix} \mathbf{1} & x_1 & y_1 & x_1^2 & \dots & y_1^3 \\ \mathbf{1} & x_2 & y_2 & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{1} & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{1} & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{1} & x_N & y_N & \dots & \dots & y_N^3 \end{bmatrix}$$

4. Оценив параметры модели производят трансформацию изображения  $R$  канала. Подставляя в модель значения «новых» координат  $(x_i, y_i)$ , с

помощью вычисляемых  $(\Delta x_i, \Delta y_i)$  определяют пиксель из исходного изображения, который следует поместить в данные координаты. Для увеличения визуального качества используют билинейную, или бикубическую интерполяцию.

5. Для синего канала проводят аналогичные процедуры.

В результате компенсации aberrаций по вышеприведенному алгоритму изображение на рисунке 6.2 будет иметь вид, представленный на рис. 6.5.

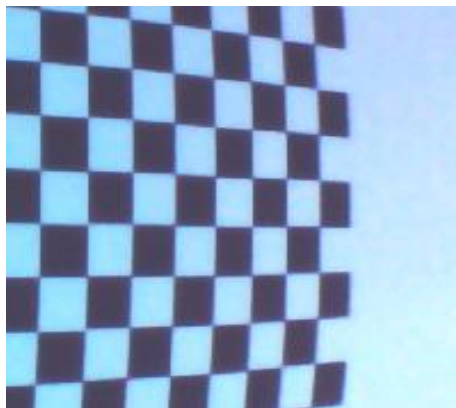


Рис. 6.5 – Полученное изображение без цветовых aberrаций.

### **Задание на моделирование**

Разработать программу, реализующую оценку параметров корректирующей полиномиальной модели и коррекцию растров с целью минимизации хроматических aberrаций.

В качестве исходных данных может выступать любой снимок, содержащий хроматические aberrации. Снимок можно получить самостоятельно на базе фотографии некоторого теста, подобного рис. 6.4. Изображение должно быть разложено на отдельные каналы  $R$ ,  $G$ ,  $B$ , после чего растры  $R$  и  $B$  должны быть незначительно искажены в каком-либо редакторе (например, в Photoshop). В качестве вносимых искажений могут выступать дисторсии типа бочка/подушка.

После объединения каналов на резких гранях изображения должны появиться заметные aberrации.

Для полученного изображения должны быть выполнены описанные процедуры коррекции. Программное обеспечение должно иметь возможность демонстрировать отдельные каналы, найденные коэффициенты корректирующих полиномов, изображения растров до и после трансформации, а также результирующее изображение с минимизированными aberrациями.

### **Контрольные вопросы**

11. Объясните, в чем суть регрессионного анализа?
12. Поясните области применения и принцип работы метода наименьших квадратов.
13. Каким образом формируется матрица плана эксперимента?
14. Укажите, сколько должно быть точек для корректной работы МНК?
15. Как должны быть расположены точки на плоскости изображения для достижения компенсации аберраций?

## ГЛАВА 3. ПРАКТИКУМ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C++ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ КОМПЬЮТЕРНОГО ЗРЕНИЯ OPENCV

### 3.1. Установка рабочей среды для программирования

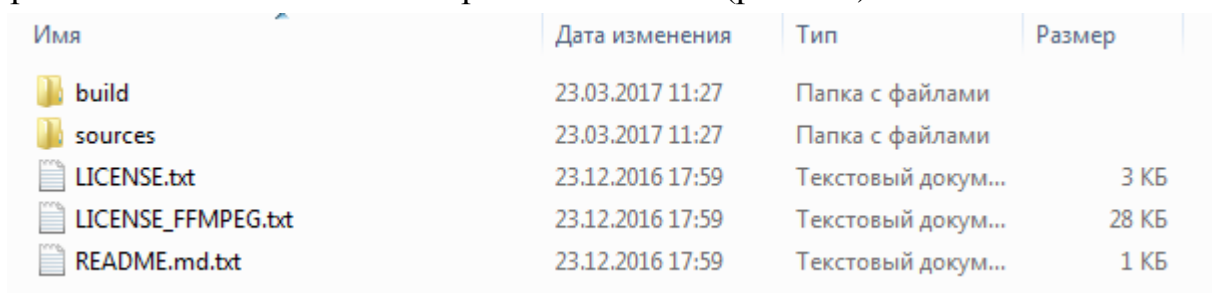
Данный раздел содержит вспомогательный материал для выполнения практических заданий. В качестве среды разработки выбран пакет Microsoft Visual Studio (MSVC) 2015 Express, распространяемый бесплатно. Информация за исключением отличий в дизайне также актуальна и при использовании более новых релизов MSVC, например, Visual Studio 2017 Community, пришедшему на смену Express.

В работе будет использована библиотека компьютерного зрения OpenCV с открытым исходным кодом, которая фактически является на сегодняшний день стандартом для разработчиков систем компьютерного зрения.

В качестве операционной системы подразумевается использование ОС Windows, однако, за исключением IDE (среды разработки, в данном случае MSVC) и настройки проекта, вся информация актуальна и при работе в Unix системах.

Для начала работы необходимо скачать с официального сайта (<https://www.visualstudio.com/vs/express/>) актуальную бесплатную версию Visual Studio. Установка IDE не составляет труда, достаточно следовать встроенным инструкциям.

С сайта OpenCV ([www.opencv.org](http://www.opencv.org)) нужно скачать самораспаковывающийся архив с библиотекой и затем распаковать его (рис. 3.1).



Имя	Дата изменения	Тип	Размер
build	23.03.2017 11:27	Папка с файлами	
sources	23.03.2017 11:27	Папка с файлами	
LICENSE.txt	23.12.2016 17:59	Текстовый докум...	3 КБ
LICENSE_FFMPEG.txt	23.12.2016 17:59	Текстовый докум...	28 КБ
README.md.txt	23.12.2016 17:59	Текстовый докум...	1 КБ

Рис. 3.1. Вид распакованного архива OpenCV.

При выборе версии OpenCV следует обращать внимание на следующий факт. Библиотека OpenCV поставляется в виде исходных кодов для самостоятельной компиляции пользователем на своей машине (папка «source» в распакованном архиве), а также в виде заранее собранных файлов библиотек (prebuilt binaries, папка «build»), компиляция которых выполнена для разных версий системы (x86 или x64) и различных версий компилятора MSVC.

Имеющийся набор собранных библиотек в архивах различных версий OpenCV отличается. Например, для версии OpenCV 3.2 в папке «...\\opencv\\build\\x64» содержатся библиотеки, собранные для использования в Visual Studio 2015 (vc14). В свою очередь, архив версии OpenCV 2.4.11 содержит собранные файлы для версий MSVC 2010 - 2013, причем, как для x86, так и для x64 систем.

Рассмотрим настройку проекта для взаимодействия с OpenCV на примере создания консольного приложения. Загрузив Visual Studio, в контекстном верхнем меню нажимаем *File – New – Project*. В появившемся окне следует выбрать пункт «Win32 Console Application» (рис. 3.2)

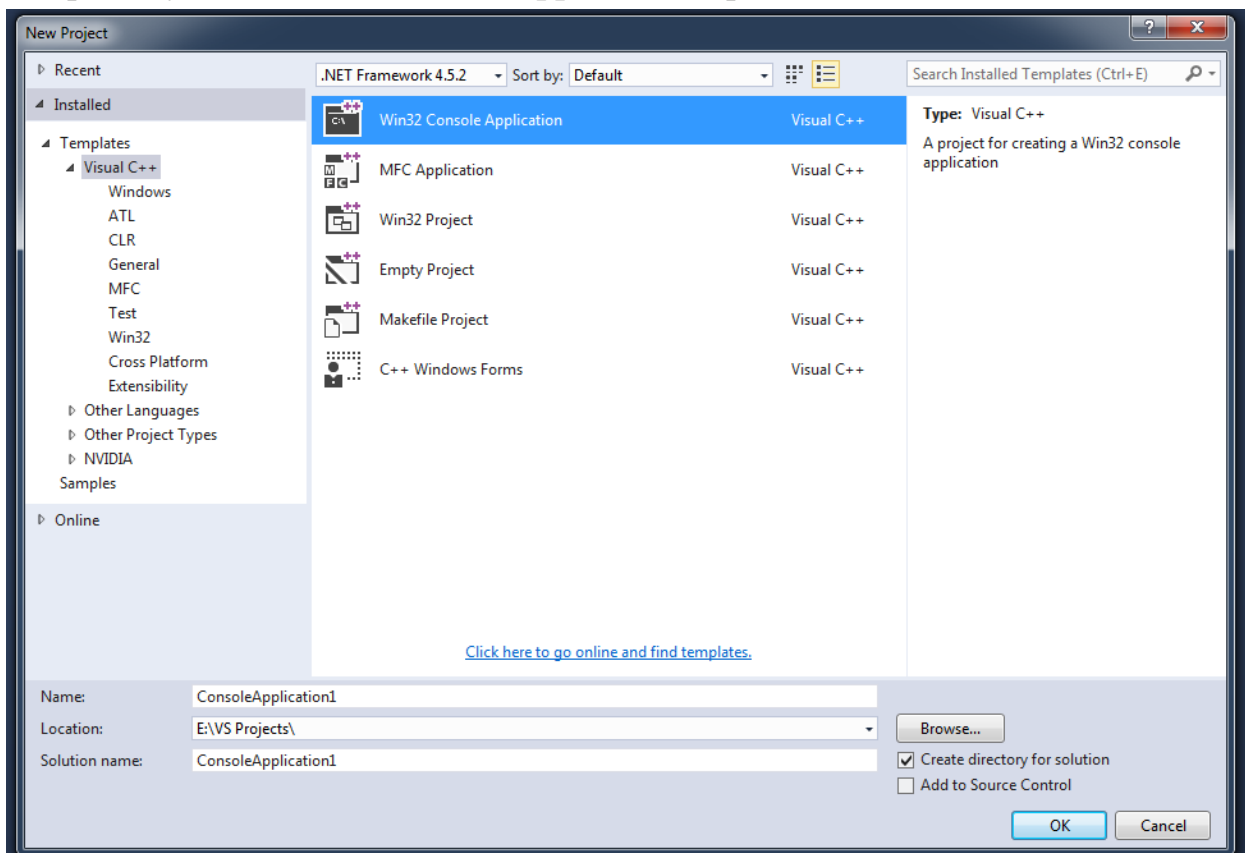


Рис. 3.2. Создание консольного приложения.

В соответствующих полях «Name» и «Location» можно указать имя и место хранения будущего проекта. Далее следует нажать «Ок».

В следующих двух диалоговых окнах нужно нажать «Next» и «Finish» (рис. 3.3) соответственно.

В результате на экране появится окно редактора с файлами исходных кодов нового проекта. Нас интересует основной файл, содержащий функцию

```
int main()
{
    return 0;
}
```



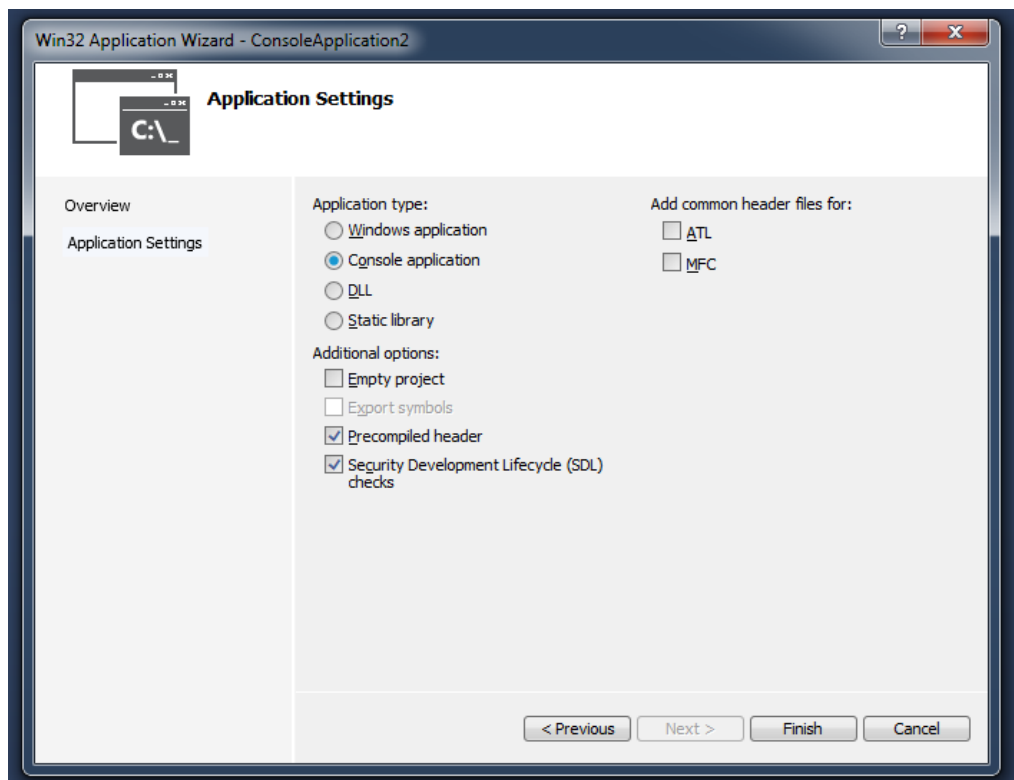


Рис. 3.3. Завершение создания проекта.

Для использования библиотеки OpenCV необходимо подключить к проекту соответствующие заголовочные файлы, например, для использования базовых функций обработки изображений достаточно подключить *highgui.hpp* и *imgproc.hpp*

```
#include "stdafx.h"
#include <imgproc.hpp>
#include <highgui.hpp>

int main()
{
    return 0;
}
```

В редакторе кода строки с новыми подключенными заголовочными файлами будут подчеркнуты, как ошибки, потому, что Visual Studio не обладает информацией о месте нахождения OpenCV.

Для настройки среды разработки необходимо выполнить следующие действия.

1. Открыть панель свойств проекта, выбрав в контекстном меню пункт *Project* (рис. 3.4)

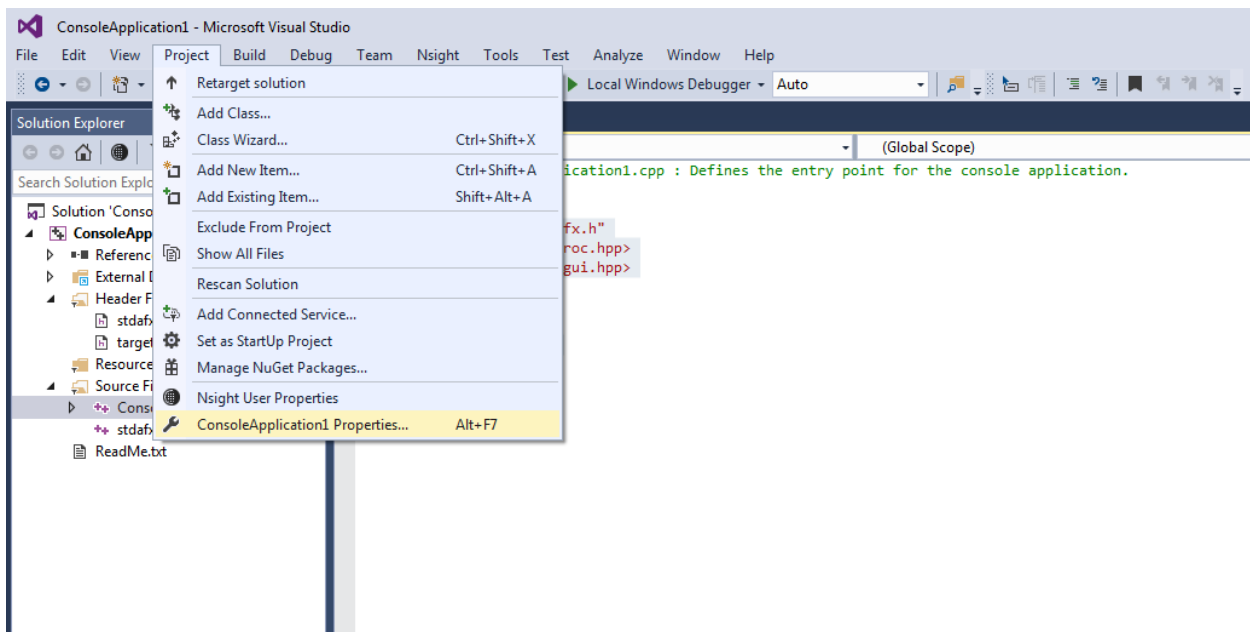


Рис. 3.4. Открытие меню свойств проекта

2. Нужно выбрать вкладку *VC++ Directories* и открыв пункт *Include Directories* добавить в него путь до заголовочных файлов OpenCV. Эти файлы хранятся в папке распакованного архива библиотеки, в директории `...\opencv\build\include\opencv2`. Также следует отдельно добавить вышестоящий каталог `...\opencv\build\include`. После добавления директории нужно нажать кнопку «Применить», или «Apply» для сохранения изменений. Результат добавления должен примерно соответствовать рисунку 3.5.

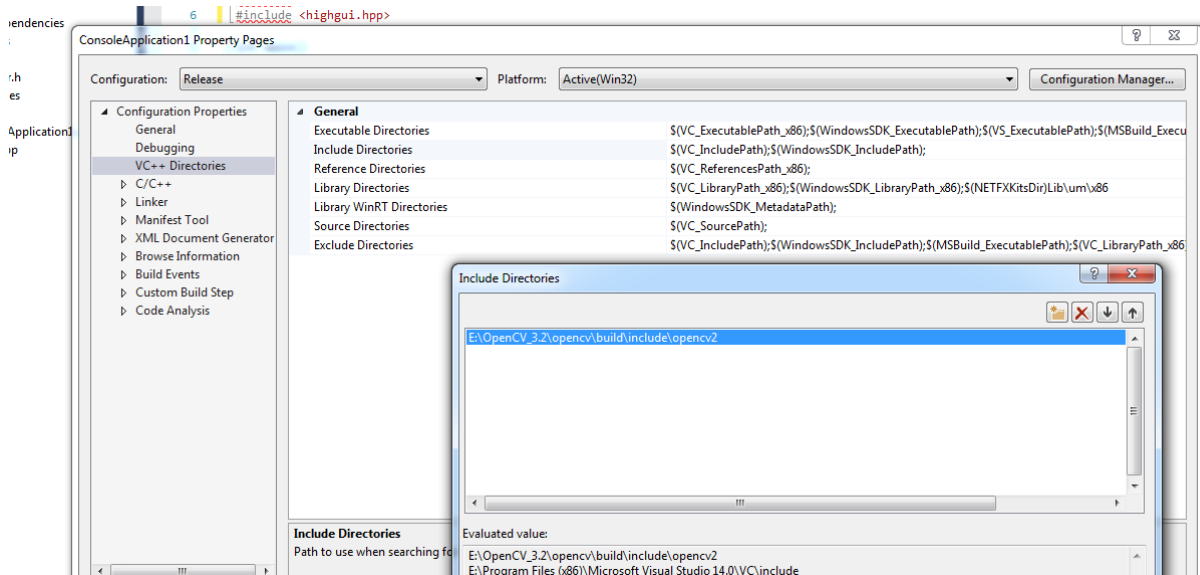


Рис. 3.5 Добавление заголовочных файлов OpenCV

3. Добавить в систему информацию о местонахождении библиотек OpenCV. Для этого нужно отредактировать строку *Library Directories* в меню *VC++ Directories*. В данном случае необходимо включить каталог,

где хранятся собранные библиотеки (с расширением файлов *lib*) для используемой версии Visual Studio. В рассматриваемом случае это `...\opencv\build\x64\vc14\lib`. Следует иметь в виду, что в OpenCV версий 3.XX все библиотеки собраны в одну, под названием *opencv\_world3xx.lib*, или в случае *Debug* версии *opencv\_world3xxd.lib*. В версиях OpenCV.2XX названия и число библиотек совпадало с заголовочными файлами, например, *opencv\_highgui2410.lib*. Результат подключения каталога представлен на рис. 3.6.

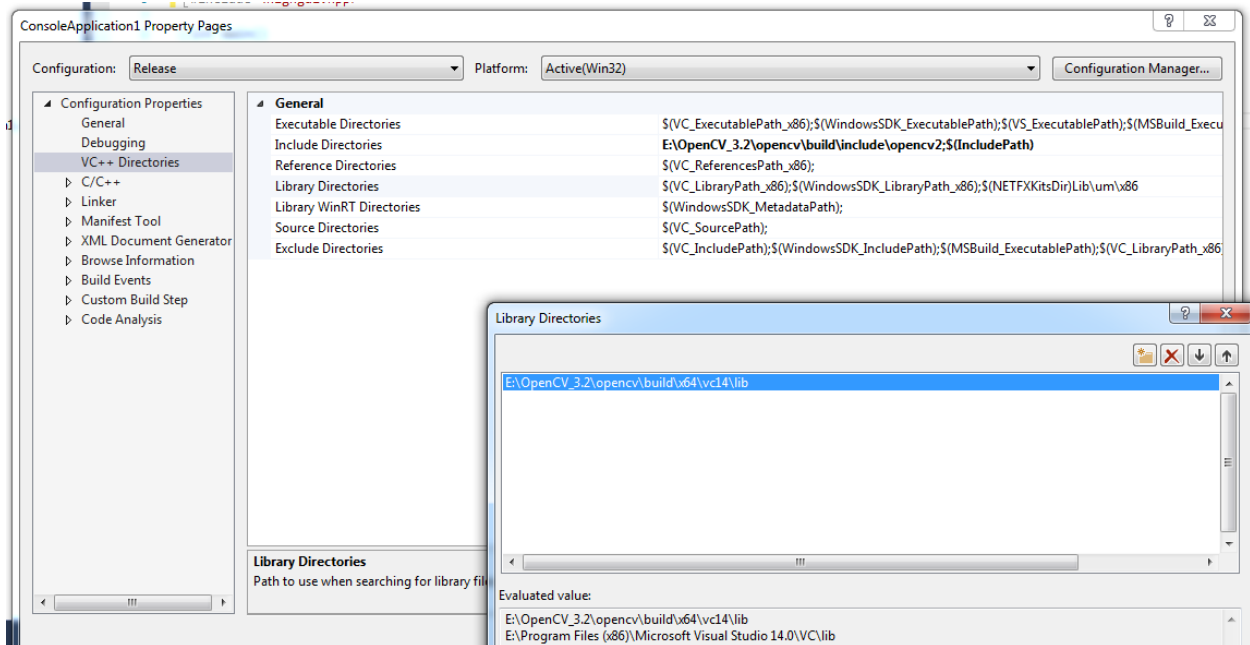


Рис. 3.5. Подключение библиотек OpenCV.

4. На вкладке *Linker* нужно отредактировать пункт *Input*, прописав в него названия используемых библиотек. Например, на рис. 3.6. показан результат добавления *opencv\_world320.lib*

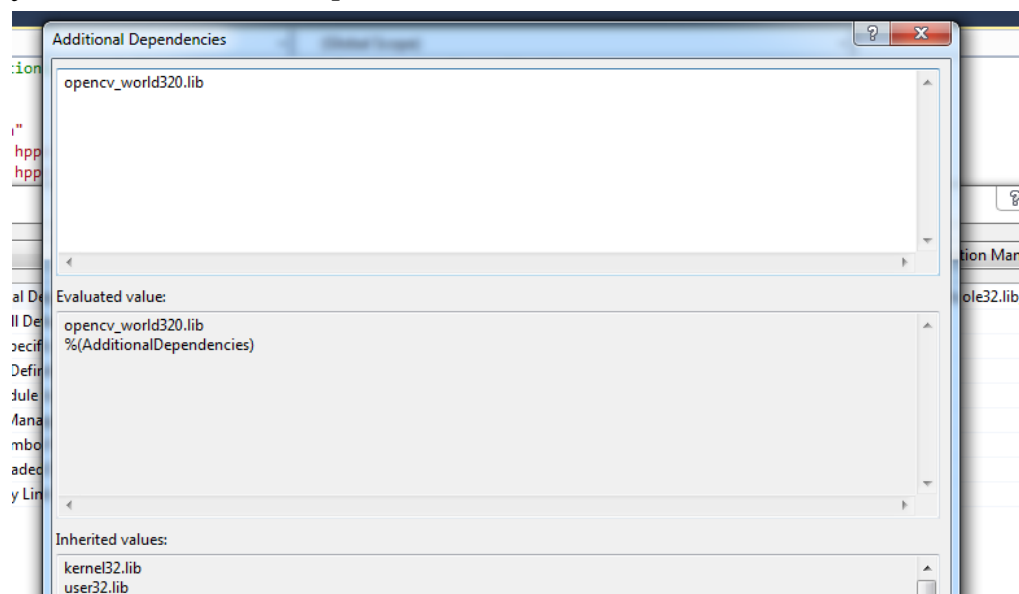


Рис. 3.6. Добавление зависимостей

Если все было сделано правильно, то подключения заголовочных файлов OpenCV в редакторе кода перестанут подчеркиваться, как ошибки и проект можно будет компилировать.

Следует обратить внимание на несколько моментов.

1. Любая комбинация конфигурации проекта (*Release*, или *Debug*) и платформы (x86, или x64) имеет свои настройки (которые можно посмотреть в заголовке окна свойств проекта), которые должны быть прописаны. Иллюстрация приведена на рис. 3.7. Частая ошибка состоит в том, что пользователь пытается компилировать проект с одними настройками, а OpenCV подключена для других.

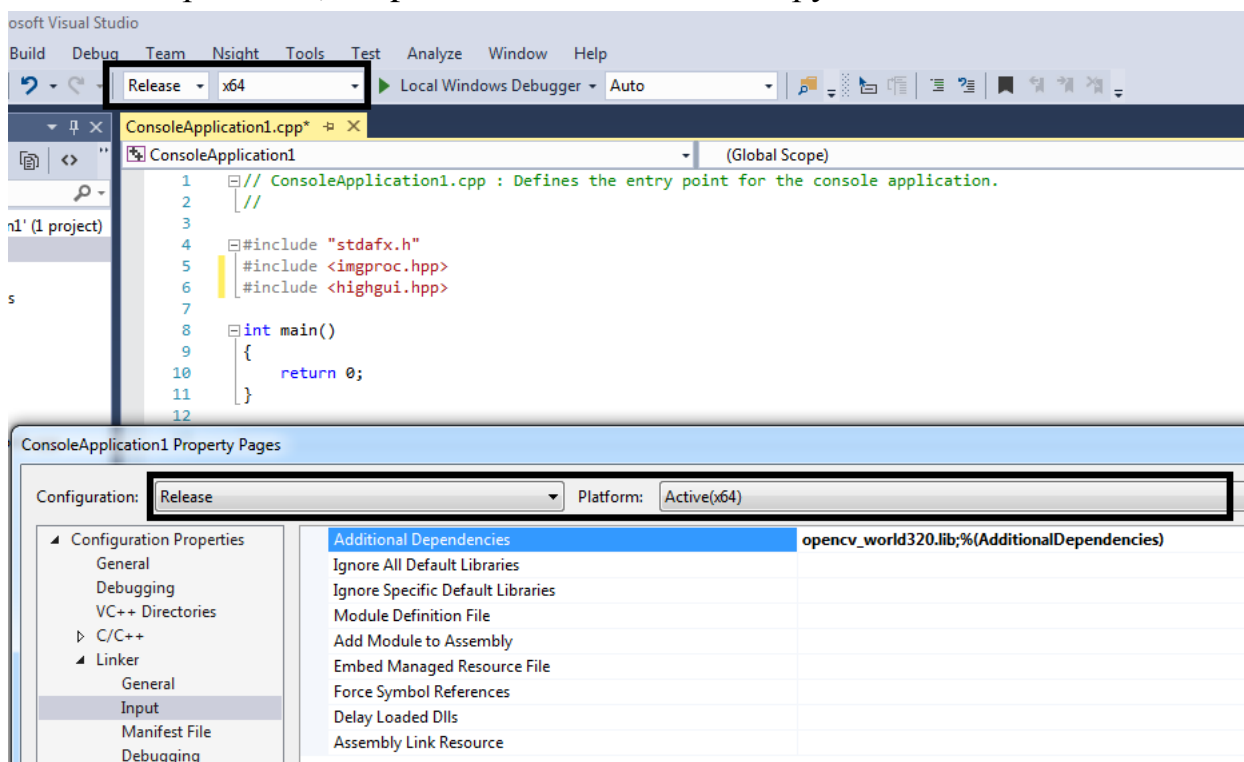


Рис. 3.7. Настройка свойств проекта соответствующей конфигурации

2. Версия библиотек для *Debug* имеет суффикс «d» в имени. Использовать в *Debug* режиме библиотеки для *Release* нельзя.
3. Для запуска программы в папку с exe файлом приложения должны быть добавлены динамические библиотеки с расширением dll, которые хранятся в папке `..\opencv\build\x64\vc14\bin`. Более удобным путем является добавления этого пути в системную переменную PATH в Windows. Делается это с помощью панели управления (*Панель управления-> Система-> Изменить параметры-> Дополнительно-> Переменные среды-> поле PATH*). Затем следует перезагрузить компьютер.

Результатом проделанных операций является проект, полностью настроенный для использования библиотеки OpenCV. В следующих разделах последовательно приводятся полезные сведения о работе с библиотекой, в соответствии с практическими заданиями.

### 3.2. Начало работы с OpenCV. Материалы для выполнения задания из раздела 1.1

Библиотека OpenCV имеет отличную подробную официальную документацию, доступную по адресу <https://docs.opencv.org/>, а также многочисленные уроки и примеры [https://docs.opencv.org/master/d9/df8/tutorial\\_root.html](https://docs.opencv.org/master/d9/df8/tutorial_root.html), которые доступны для каждой версии пакета (веб форма позволяет выбрать необходимую версию).

Кроме того, на тематических форумах также рассматривается широкий круг вопросов, связанный с особенностями применения OpenCV.

Данный материал не ставит перед собой цель повторить, или каким-то образом переписать существующую документацию. Здесь лишь приводится краткое описание отдельных вопросов, функций и алгоритмов, которые являются полезными при выполнении того, или иного практического задания. Для всестороннего и полного понимания рассматриваемых вопросов следует обращаться к официальной документации.

Базовым классом OpenCV является класс `cv::Mat`, который представляет собой многомерный массив чисел (одноканальный, или многоканальный). Класс может использоваться для хранения матриц, изображений, векторов, векторных полей, тензоров и так далее. В случае использования класса для хранения некоторой матрицы, элементы массива будут хранить значения ее элементов, при записи в *Mat* изображения, элементы будут отображать, например, яркости пикселей.

У данного класса существует множество конструкторов, то есть создать объект класса *Mat* можно разными способами.

Для начала работы создадим объект класса *Mat* и сразу же инициализируем его, прочитав изображение с диска с помощью функции `cv::imread`. Затем выведем картинку на экран (функция `cv::imshow()`). Здесь и далее при упоминании названий функций, аргументы опускаются для экономии места.

```
#include "stdafx.h"
#include <imgproc.hpp>
#include <highgui.hpp>
```

```
using namespace cv;

int main()
{
    Mat img = imread("C:/Users/Public/Pictures/Sample
Pictures/tulips.jpg");

    imshow("img", img);

    waitKey();

    return 0;
}
```

В данном примере мы загружаем и выводим на экран одну из стандартных картинок Windows – tulips.jpg.

Функция `cv::waitKey()` вводит интервал в микросекундах (указывается в скобках), в течение которого система ожидает нажатия клавиши. В случае пустого аргумента – ожидание будет неограниченным. В данном случае эта функция используется для того, чтобы вывести результат на экран и не завершать программу немедленно.

Класс *Mat* содержит множество методов и свойств. Одними из важных и часто используемых являются свойства *Mat::cols*, и *Mat::rows*, представляющие собой число столбцов и строк соответственно. В случае работы с изображениями, *cols* означает размер в пикселях по горизонтали, *rows* по вертикали. Для примера использования этих свойств используем функцию изменения размера картинки. Она принимает на вход минимум три параметра – исходное изображение (матрицу), изображение для записи результата и переменную типа `cv::Size` – структуру, задающую размер. Результат выполнения программы – уменьшение изображения вдвое (рис. 3.8) и ее листинг приведены ниже. В данном примере структура `cv::Size` инициализируется прямо при передаче в функцию в качестве аргумента.

```
#include "stdafx.h"
#include <imgproc.hpp>
#include <highgui.hpp>
using namespace cv;

int main()
{
    Mat img1 = imread("C:/Users/Public/Pictures/Sample
Pictures/tulips.jpg");
    Mat img2;
    cv::resize(img1, img2, cv::Size(img1.cols / 2, img1.rows / 2));
```



```

imshow("img1", img1);
imshow("img2", img2);

waitKey();
return 0;
}

```

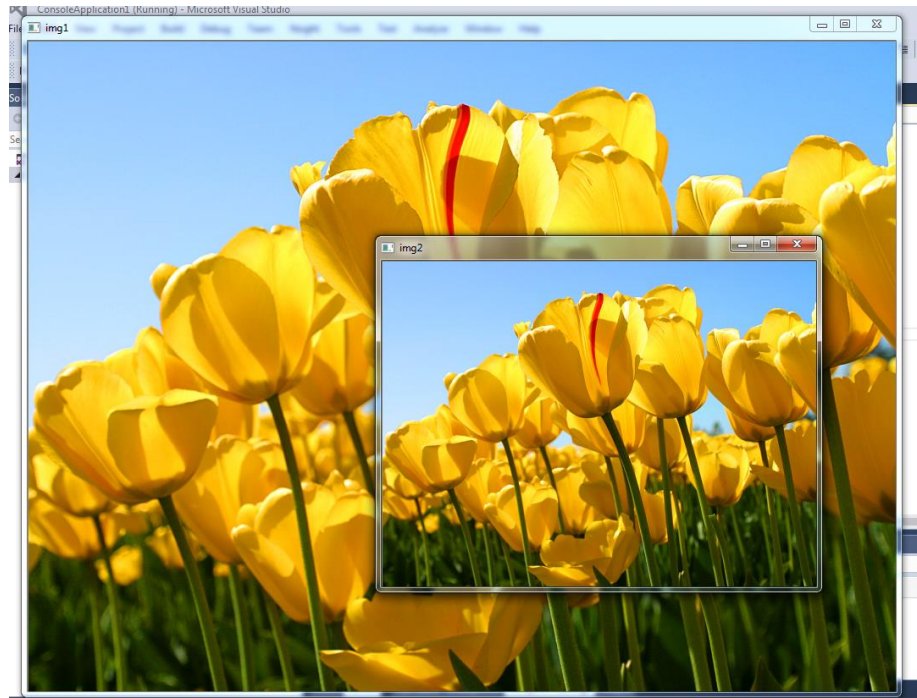


Рис. 3.8. Результат уменьшения в два раза размера изображения

Часто необходимо подготовить «пустое» (заполненное нулями, или черным цветом) изображение определенного размера. Это удобно реализовать с помощью соответствующего конструктора, например:

```
Mat BlackImage = Mat::zeros(cv::Size(100,100), CV_8UC3);
```

В данном случае будет создано изображение, содержащие пиксели черного цвета, размером 100 на 100. Следует обратить внимание на параметр со значением *CV\_8UC3*. Это определенный в OpenCV формат, означающий буквально «8 unsigned char 3», то есть три канала типа unsigned char по 8 бит. Например, в данном формате хранится стандартное 24-х битовое RGB изображение. Для работы с данным форматом в OpenCV определен тип данных *cv::Vec3b* (название можно расшифровать, как вектор, в данном случае это следует понимать, как контейнер, из трех байт – то есть трех unsigned char)

Изображение в оттенках серого, как известно, имеет один канал и глубину 8 бит. Для такого изображения в OpenCV используется формат *CV\_8UC1*, или просто *CV\_8U*. Тип данных для пикселей в данном формате – обычный unsigned char.

OpenCV предоставляет большие возможности по линейной и нелинейной фильтрации изображений. Например, в первой работе требуется получить

межкадровую разность изображений. С помощью OpenCV это выполняется всего одной функцией `cv::absdiff()`:

```
int main()
{
    Mat img1 = imread("C:/../k1.png",0);
    Mat img2 = imread("C:/../k2.png", 0);
    Mat ifd;

    cv::absdiff(img2, img1, ifd);
    imshow("img1", img1);
    imshow("img2", img2);
    imshow("ifd", ifd);
    waitKey();
    return 0;
}
```

На рис. 3.9. представлены исходные кадры и результат вычитания. Следует обратить внимание на то, что функция чтения изображений *imread* в данном примере вызывается со вторым аргументом, равным нулю (по умолчанию этот параметр равен единице, что соответствует чтению цветного изображения 24 бит на пиксель). Это означает, что чтение произойдет в формате grayscale (8 бит на пиксель).



Рис. 3.9. Исходные кадры (верхний ряд) и результат межкадровой разницы (внизу)



Получение контурного препарата также легко реализуется с помощью OpenCV. Существуют реализации популярных фильтров Собеля (рис. 3.10) и Кэнни (3.11).

```
int main()
{
    Mat img1 = imread("C:/../k1.png",0);
    Mat img2 = imread("C:/../k2.png", 0);
    Mat sobel, canny;

    cv::Sobel(img2, sobel, -1, 1, 1, 3);
    cv::Canny(img2, canny, 50, 200);

    cv::threshold(sobel, sobel, 5, 255, CV_THRESH_BINARY_INV);

    imshow("img2", img2);
    imshow("sobel", sobel);
    imshow("canny", canny);

    waitKey();

    return 0;
}
```

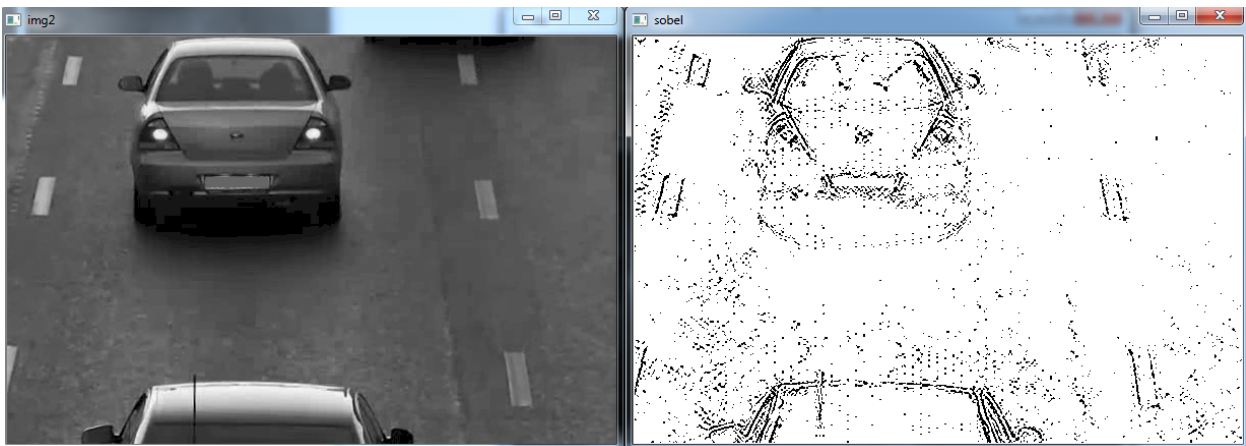


Рис. 3.10. Результат фильтра Собеля



Рис. 3.11. Результат фильтра Кэнни

Каждый фильтр, как и любая функция, имеет свой характерный набор параметров. Как правило, в качестве первого параметра выступает изображение (объект *Mat*) к которому будет применяться фильтр, а в качестве второго – изображение, куда будет сохранен результат.

Часто встречается параметр *depth* – глубина. Он отвечает за глубину выходного изображения (например, можно установить CV\_8U, или CV\_32F). Если установить *depth* = -1, то глубина будет соответствовать входному изображению. Параметры фильтра Собеля *dx*, *dy*, устанавливаются для определения по какому направлению (по вертикали и/или по горизонтали) будут определяться производные. Если параметр равен нулю – производная не вычисляется, если единице – то будет вычислена первая производная, если двум – вторая. Последний параметр – размер маски фильтра (например 3, 5, 7 и т.д.).

Фильтр Кэнни определяется двумя параметрами – верхним и нижним порогом. Все пиксели со значениями яркости выше второго порога – относятся к граням, со значениями ниже первого – к фону, для пикселей с яркостями, заключенными между порогами принимается решение, на основании соответствующего анализа.

В приведенном примере также встречается функция пороговой бинаризации *cv::threshold()*, которая часто встречается в практических задачах. Третий параметр отвечает за инициализацию порога, четвертый за устанавливаемое максимальное значение яркости, пятый – предопределенная константа, отвечающая за тип бинаризации (обычная, инвертированная и т.д.).

Для выполнения первого задания будет полезна функция из набора побитовых матричных операций OpenCV – *cv::bitwise\_and()* реализует побитовое логическое «И» для изображений (также существуют *cv::bitwise\_or()*, *cv::bitwise\_xor()*, *cv::bitwise\_not()*).

На рисунке 3.12 приведен результат операции логического «И» для контурного препарата фильтра Кэнни (рис. 7.11) и межкадровой разности (рис. 7.9).

```
int main()
{
    Mat img1 = imread("C:/../k1.png",0);
    Mat img2 = imread("C:/../k2.png",0);
    Mat ifd, sobel, canny, combined;

    cv::absdiff(img2, img1, ifd);
    cv::Canny(img2, canny, 50, 200);
    bitwise_and(ifd, canny, combined);

    cv::threshold(combined, combined, 5, 255, CV_THRESH_BINARY_INV);
}
```

```

    imshow("combined", combined);
    waitKey();
    return 0;
}

```

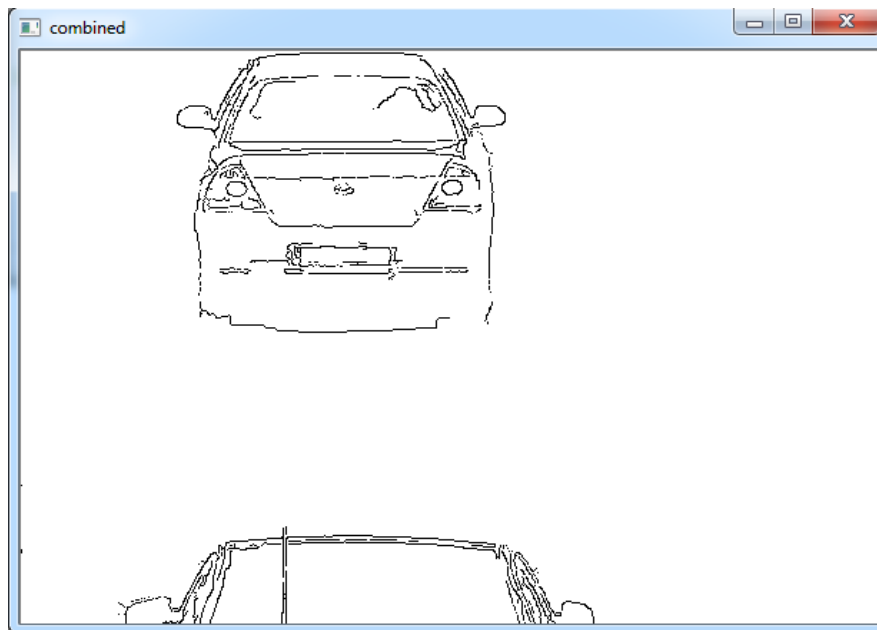


Рис. 3.12. Результат логического «И».

На рисунке видно, что в результате обработки в кадре остались только движущиеся объекты.

OpenCV обладает основными функциями из аппарата математической морфологической фильтрации: эрозией, дилатацией, открытием, закрытием, top hat, black hat. Для выполнения фильтрации требуется определить морфологический элемент (его форму, размер, точку фокуса), а затем включить его в аргументы функции-фильтра.

```

Mat element = cv::getStructuringElement(CV_SHAPE_RECT, Size(31, 31),
Point(16, 16));
morphologyEx(combined, combined, CV_MOP_ERODE, element);

```

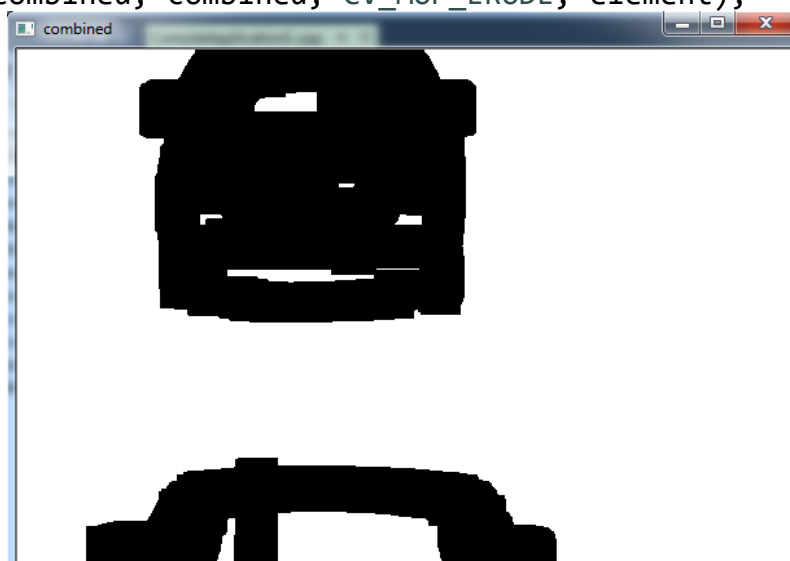


Рис. 3.13. Выполнение морфологической эрозии

В данном примере с помощью функции `cv::getStructuringElement` создается структурный элемент прямоугольной формы (форма может быть также в виде эллипса, или креста) определенного размера (тип `cv::Size()`) и с центром в определенной точке (тип `cv::Point()`). Затем выполняется фильтрация – эрозия (рис. 7.13). Стоит обратить внимание на то, что результат фильтрации записывается в ту же переменную, которая подается на вход. Многие функции OpenCV обладают такой возможностью перезаписи (работа функции *in place*).

Для начала работы с OpenCV и выполнения первого практического задания необходимо освоить механизм доступа к отдельным пикселям изображений (элементам матрицы). Для этого рассмотрим пример, который инвертирует цвет изображения. Все пиксели перебираются в цикле и их яркость меняется на противоположенную по условию. Предположим, что в соответствии с предыдущим примером мы получили результат морфологической фильтрации и записали его в объект *Mat* с именем *combined*. Выполним следующее:

```
for (int y = 0; y < combined.rows; y++)
    for (int x = 0; x < combined.cols; x++) {
        if (combined.at<unsigned char>(y, x) == 255) {
            combined.at<unsigned char>(y, x) = 0;
        }
        else {
            combined.at<unsigned char>(y, x) = 255;
        }
    }
```

Результат приведен на рисунке 3.14

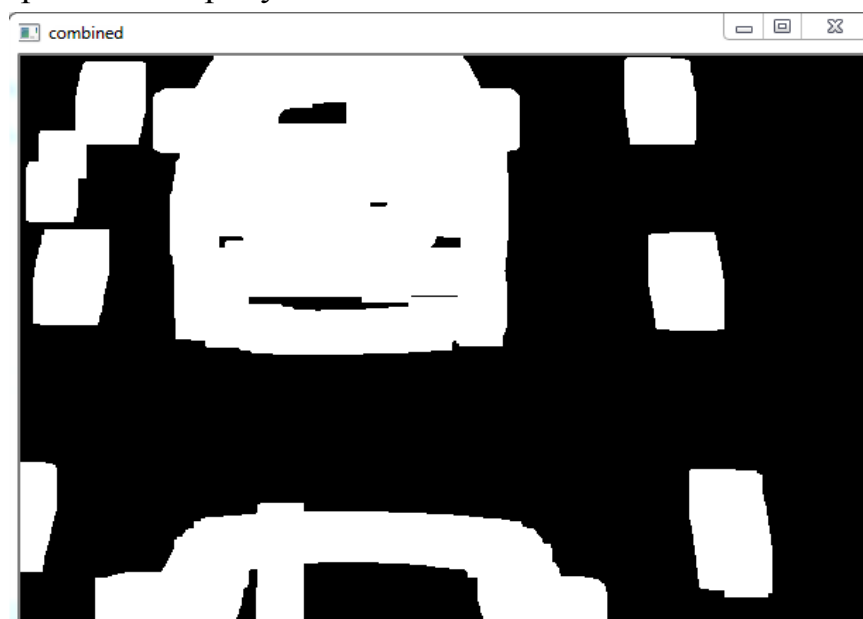


Рис. 3.14. Результат инвертирования цвета

Пределы цикла установлены с помощью рассмотренных ранее свойств класса *Mat* – *cols* и *rows*. Для доступа к конкретному пикселю использована

шаблонная функция *at*, у которой в угловых скобках указывается тип данных, который должен соответствовать формату, в котором записан пиксель. Например, для grayscale изображений это CV\_8U, то есть *unsigned char*, для цветного изображения CV\_8UC3, то есть *cv::Vec3b*, для матрицы, хранящей числа с плавающей точкой CV\_32F – *float* и так далее. В круглых скобках указываются координаты пикселя (элемента матрицы). Важно понимать, что при работе с цветным изображением по адресу с некоторыми координатами хранится структура *cv::Vec3b*, представляющая собой массив из трех *unsigned char*. То есть, допустим, для получения значения зеленого канала у некоторого пикселя с координатами ( *y*, *x*), функция выглядела бы следующим образом:

```
unsigned char Green = combined.at<Vec3b>(y, x)[1];
```

Красный и синий каналы могли бы быть получены:

```
unsigned char Red = combined.at<Vec3b>(y, x)[2];
```

```
unsigned char Blue = combined.at<Vec3b>(y, x)[0];
```

При работе с пикселями, да и в целом с матрицами в OpenCV следует иметь в виду, что часто (при инициализации, при доступе с помощью *at*) координаты подаются в формате (*y*, *x*), что непривычно для начинающего пользователя. При этом запись координат в структурах *cv::Size()* и *cv::Point()* – традиционная (*x*, *y*). Эта неоднозначность возникает отчасти из-за особенностей реализации *Mat*, отчасти из-за того, что в матрице в отличие от изображения, наоборот, принято первыми перечислять строки, а затем столбцы. При программировании нужно относиться к этому внимательно, часто путаница в координатах вызывает ошибки.

### 3.3. Совмещение с шаблоном. Материалы для выполнения задания из раздела 1.2

Совмещение с шаблоном – общее название техник обработки изображений, связанных с поиском определенного фрагмента на снимке, или видео. Во втором практическом задании это может быть полезно при поиске векторов движения полным перебором. Шаблоном в данном случае будет являться блок из первого изображения, для которого мы ищем соответствующий блок на втором.

Для выделения (вырезания) блоков удобно воспользоваться структурой *cv::Rect (x, y, width, height)*. Например, можно вырезать небольшой фрагмент из изображения и сохранить его для дальнейшей работы. Для этого надо при инициализации нового объекта класса *Mat* в качестве аргумента указать ограничивающий прямоугольник.

```
int main()
{
```

```

Mat img1 = imread("C:/../tulips.jpg");
cv::Rect rect(100, 100, 200, 200);

Mat img2, img3;
img2 = img1(rect);
img3 = img1(rect).clone();

imshow("img1", img1);
imshow("img2", img2);
imshow("img3", img3);
waitKey();

return 0;
}

```

Приведенный код иллюстрирует (рис. 3.15) пример вырезания фрагментов из исходного изображения. Для этого создается прямоугольник *rect* с координатами верхнего левого угла (100,100) и длиной и шириной, равным 200. Затем с его помощью инициализируются два новых изображения.

Важным отличием является то, что *img3* за счет использования функции *clone()*, является независимой «глубокой копией» соответствующего фрагмента *img1*. Изменяя его, программист не повредит исходный снимок.

*Img2*, напротив, ссылается на ту же область памяти, что и фрагмент *img1*. Внесенные изменения в *img2* отразятся и на оригинале.

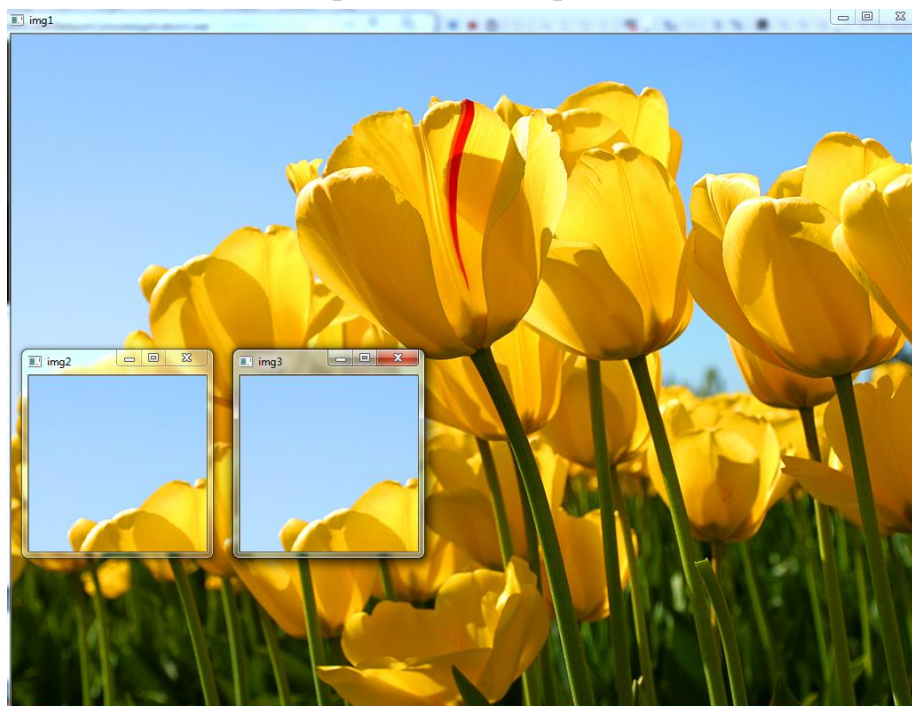


Рис. 3.15. Вырезание части изображения

Поиск шаблона на изображении рассмотрим на примере – найдем координаты вырезанного прямоугольника на исходной картинке. Для этого необходимо предварительно создать объект класса *Mat* для сохранения



результата, выполнить функцию OpenCV *cv::matchTemplate* и определить координаты совпадения с помощью функции *cv::minMaxLoc*.

```
#include "stdafx.h"
#include <imgproc.hpp>
#include <highgui.hpp>
#include <iostream>

using namespace cv;

int main()
{
    Mat img1 = imread("C:/../tulips.jpg");
    cv::Rect rect(100, 100, 200, 200);

    Mat img2, result;

    img2 = img1(rect).clone();

    //Определяем размеры матрицы - результата
    int result_cols = img1.cols - img2.cols + 1;
    int result_rows = img1.rows - img2.rows + 1;

    result.create(result_rows, result_cols, CV_32FC1);

    /// Сопоставление с шаблоном и нормировка результата
    matchTemplate(img1, img2, result, CV_TM_SQDIFF);
    normalize(result, result, 0, 1, NORM_MINMAX, -1, Mat());

    /// Определение точки наилучшего соответствия
    double minVal; double maxVal; Point minLoc; Point maxLoc;
    minMaxLoc(result, &minVal, &maxVal, &minLoc, &maxLoc, Mat());

    std::cout << minLoc << std::endl;

    imshow("img1", img1);
    imshow("img2", img2);
    imshow("result", result);

    waitKey();

    return 0;
}
```

Результат приведен на рисунке 3.16. В данном случае матрица *result* инициализируется с помощью метода *create* – это еще один возможный вариант создания объекта. Тип *CV\_32FC1* в явном виде указывает на то, что нам требуется одноканальная матрица с элементами типа *float*. Четвертый аргумент функции *cv::matchTemplate* – метод сопоставления, вернее, применяемая метрика. В данном случае выбрана среднеквадратическая ошибка, но может

быть использован и коэффициент корреляции (CV\_TM\_CCORR), и другие метрики. Функция нормировки приводит все значения к диапазону от нуля до единицы – это в том числе позволяет вывести изображение формата CV\_32FC1 на экран.

Как видно на скриншоте консольного вывода, координаты были определены правильно – это точка (100, 100). На изображении-результате этим координатам соответствует самый черный пиксель, что логично, так как в этой точке значение среднеквадратической ошибки минимально.

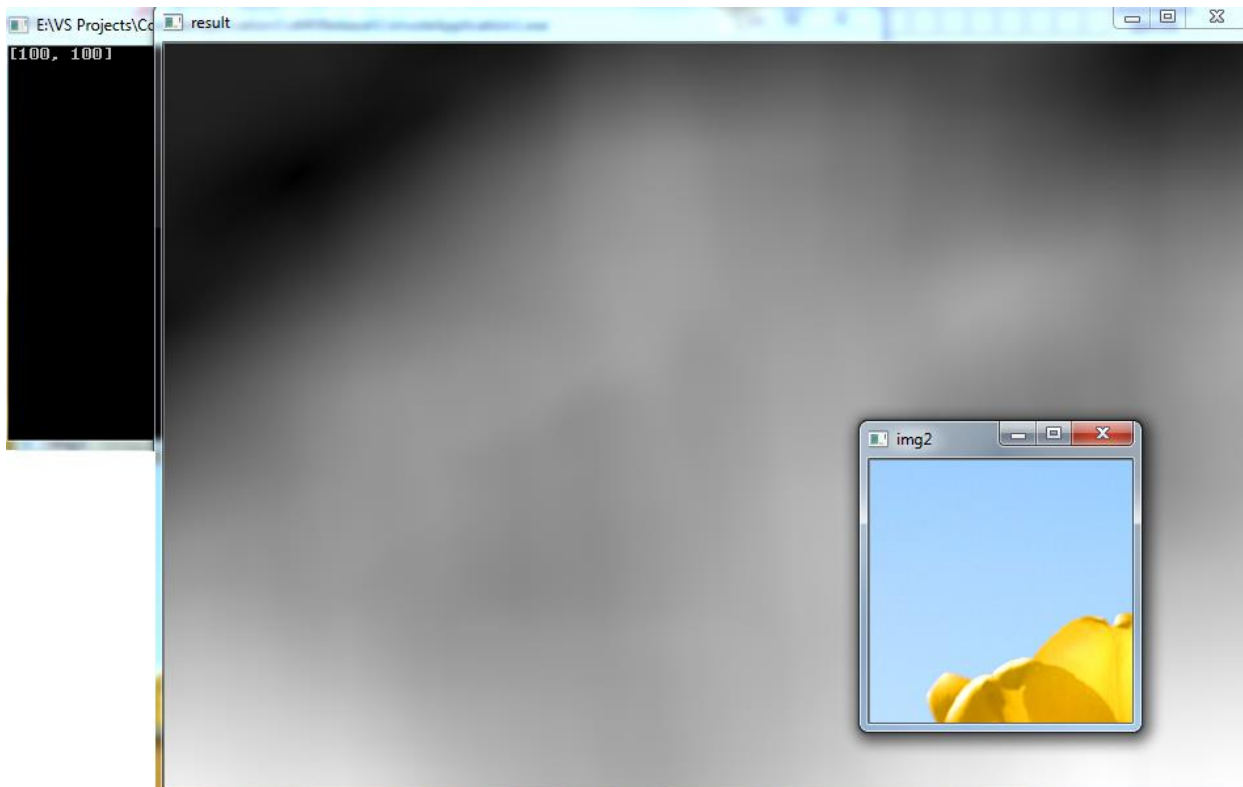


Рис. 3.16. Результат сопоставления с шаблоном.

### 3.4. Автоматический синтез панорамных изображений. Материалы для выполнения задания из раздела 1.3

Для сшивки панорам в OpenCV существует специальный модуль. В последних версиях этот процесс полностью автоматизирован и вынесен на высокий уровень – в класс *cv::Stitcher*. Тем не менее, для понимания процесса создания панорамного изображения полезно будет обратиться к примеру из ранних версий OpenCV. Нижеприведенный пример не содержит таких процедур, как калибровка снимков и блэндинг, что сказывается на качестве итогового изображения, но зато позволяет наглядно продемонстрировать основные этапы алгоритма.



В начале с помощью детектора характерных точек (в примере использован ORB, но в OpenCV существует множество других вариантов) найдем *keypoints*, вычислим их дескрипторы и покажем результат (рис. 3.17)

```
#include "stdafx.h"
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>

using namespace cv;

int main()
{
    //Загрузка изображений с диска
    Mat img1 = imread("C:/../mnt1.jpg");
    Mat img2 = imread("C:/../mnt2.jpg");

    //Инициализация детектора, подготовка контейнеров для характерных точек и
    их дескрипторов
    Ptr<ORB> detector = ORB::create();
    std::vector<KeyPoint> keypoints1, keypoints2;
    Mat descriptor1, descriptor2;

    //Поиск характерных точек на сшиваемых изображениях и вычисление их
    дескрипторов
    detector->detect(img1, keypoints1);
    detector->detect(img2, keypoints2);
    detector->compute(img1, keypoints1, descriptor1);
    detector->compute(img2, keypoints2, descriptor2);

    //Отрисовка найденных характерных точек
    Mat keypoints1draw, keypoints2draw;
    drawKeypoints(img1, keypoints1, keypoints1draw);
    drawKeypoints(img2, keypoints2, keypoints2draw);

    imshow("keypoints1draw", keypoints1draw); imshow("keypoints2draw",
    keypoints2draw);

    waitKey();
    return 0;
}
```

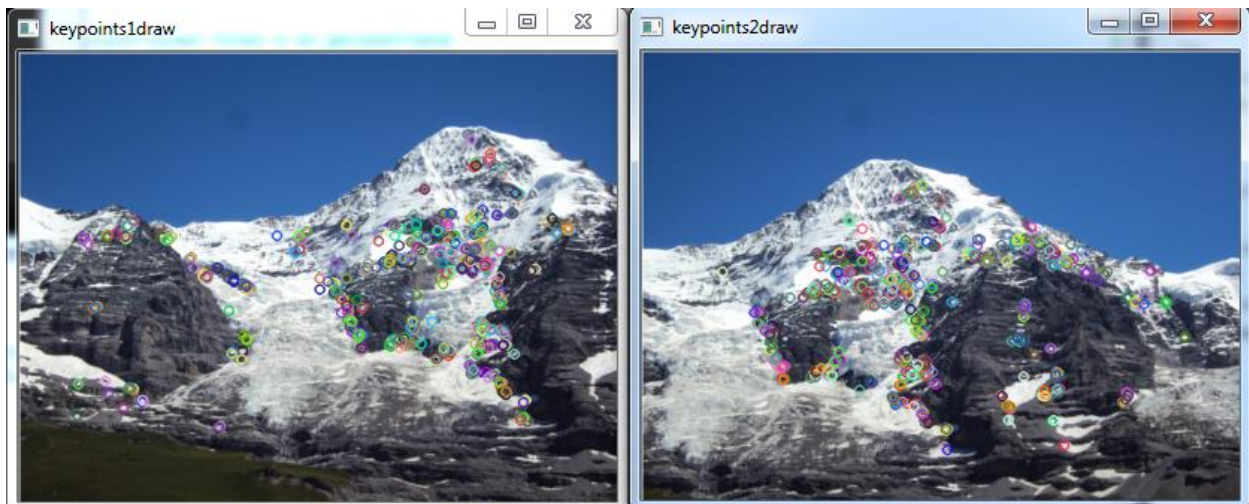


Рис. 3.17. Найденные характерные точки на изображениях

Дополним программу предыдущего примера сопоставлением дескрипторов. Для полного понимания приведенного кода, желательно самостоятельно разобраться с помощью документации, что представляют собой классы *DMatch* и *KeyPoint*

```
//Сопоставление дескрипторов точек с помощью BFMatcher
BFMatcher matcher(NORM_HAMMING);
std::vector<std::vector<DMatch>> matches;
matcher.knnMatch(descriptor1, descriptor2, matches, 2);

std::vector<KeyPoint> matched1, matched2;
std::vector<DMatch> good_matches;

float match_ratio = 0.5f;

for (size_t i = 0; i < matches.size(); i++) {
    DMatch first = matches[i][0];
    float dist1 = matches[i][0].distance;
    float dist2 = matches[i][1].distance;

    if (dist1 < match_ratio * dist2) {
        int new_i = static_cast<int>(matched1.size());
        matched1.push_back(keypoints1[first.queryIdx]);
        matched2.push_back(keypoints2[first.trainIdx]);
        good_matches.push_back(DMatch(new_i, new_i, 0));
    }
}

Mat dMatches;
drawMatches(img1, matched1, img2, matched2, good_matches, dMatches);

imshow("Matches", dMatches);
```

Данный код реализует сопоставление с помощью алгоритма Brute Force Matching, суть которого состоит в выяснении дистанций (евклидово расстояние между векторами-дескрипторами) от каждой характерной точки одного изображения до каждой характерной точки другого.

Затем проводится фильтрация, в результате которой остаются только те пары точек, для которых расстояние до второго ближайшего дескриптора вдвое больше, чем до первого. Фильтрация основана на предположении, что расстояние между реально соответствующими друг другу дескрипторами много меньше, чем до ошибочно определенных. На рисунке 3.18 представлен результат.

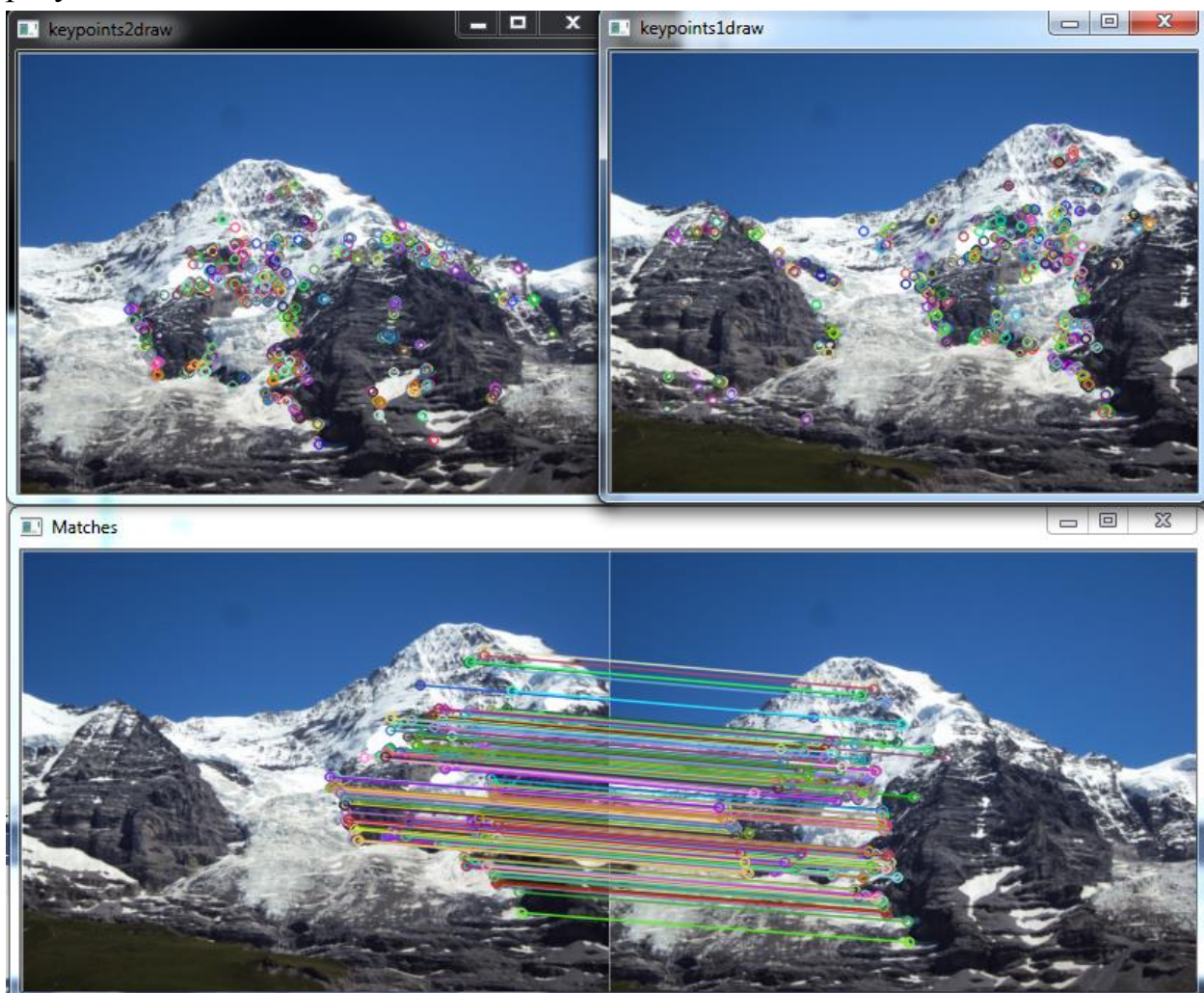


Рис. 3.18. Результат сопоставления точек.

В заключительной части примера на основании сопоставленных точек оценивается матрица гомографии, необходимая для трансформации второго фрагмента соответственно первому. На рисунках 3.19 и 3.20 приведены последовательно трансформированный второй фрагмент и сшитая панорама.

```
//Формируем векторы из сопоставленных точек  
std::vector<Point2f> img1_pts;  
std::vector<Point2f> img2_pts;
```



```

for (int i = 0; i < good_matches.size(); i++){
    img1_pts.push_back(matched1[i].pt);
    img2_pts.push_back(matched2[i].pt);
}
//Находим матрицу гомографии H
Mat H = findHomography(img2_pts, img1_pts, CV_RANSAC);
//Используем найденную H для трансформации второго фрагмента
cv::Mat result;
warpPerspective(img2, result, H, cv::Size(img1.cols+img2.cols/2,
img1.rows));
imshow("Result", result);
waitKey();
//Копируем первый фрагмент в соответствующую часть панорамы
cv::Mat half(result, cv::Rect(0, 0, img1.cols, img1.rows));
img1.copyTo(half);
imshow("Result", result)

```

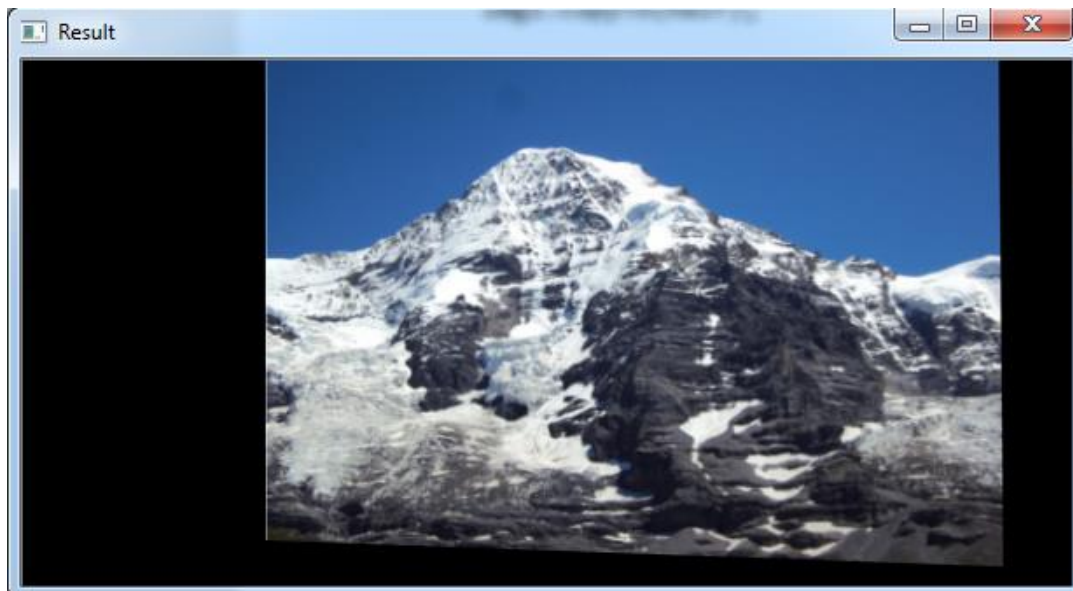


Рис. 3.19. Трансформированный второй фрагмент

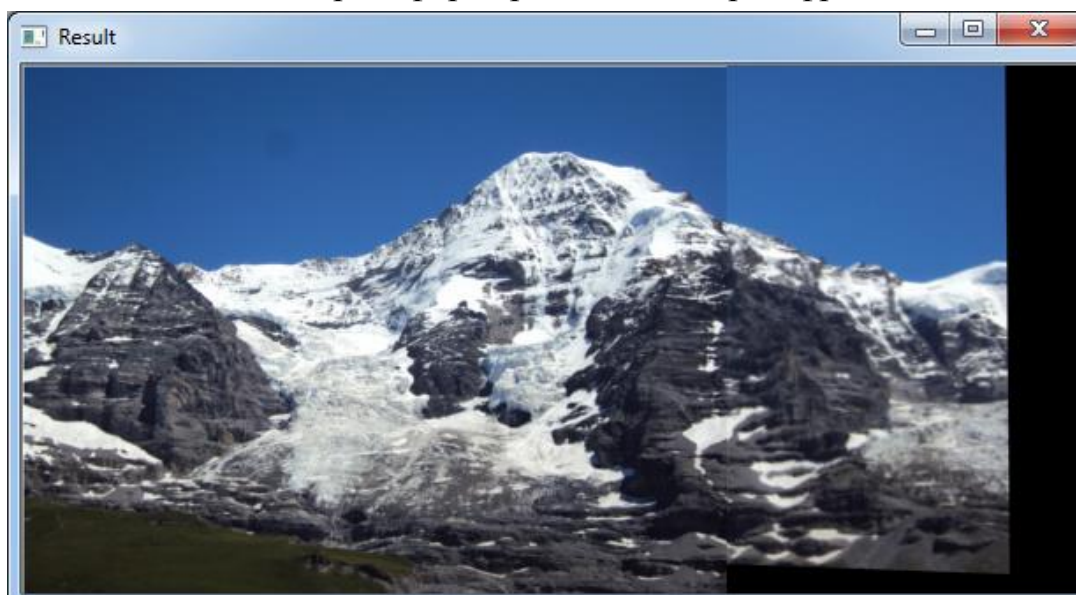


Рис. 3.20. Итоговая панорама.

### 3.5. Операции с матрицами и векторами. Материалы для выполнения задания из раздела 2.1

Библиотека OpenCV содержит большой инструментарий из области машинного обучения (модуль ml). Это и функции для подготовки данных, и реализации популярных алгоритмов: метода опорных векторов (SVM), лесов решающих деревьев (RDF), байесовского классификатора, нейронных сетей (NN) и других. Существуют также классы для работы с моделями – результатами глубокого обучения.

Полностью готовой реализации дискриминантного анализа в библиотеке нет, данный алгоритм нужно синтезировать самостоятельно. Тем не менее, в OpenCV реализовано большое количество инструментов, которые будут полезны при имплементации алгоритма, в частности есть функция, вычисляющая расстояние Махаланобиса между векторами и функция для вычисления ковариационной матрицы.

Полезно в целом ознакомиться с возможностями OpenCV при работе с матрицами (не только содержащими изображения). Эти знания помогут при выполнении практических заданий.

В практикуме уже встречались функции, относящиеся к операциям над матрицами: `cv::absdiff()`, `cv::bitwise_and()` и некоторые другие. Одними из самых частых в использовании являются функции для элементарных матричных операций, для которых также реализованы перегруженные операторы, или методы класса. В примере для простоты операции проводятся над единичной матрицей, и матрицей, состоящей из одних единиц. Полезно при реализации примера попробовать и другие операнды.

```
#include "stdafx.h"
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>

int main()
{
    //Создадим две матрицы, единичную и состоящую из единиц.
    Mat eye = Mat::eye(3, 3, CV_32F);
    Mat box = Mat::ones(3, 3, CV_32F);

    //Выведем их в консоль
    std::cout << "eye = " << std::endl;
    std::cout << eye << std::endl;
    std::cout << "box = " << std::endl;
    std::cout << box << std::endl;
}
```

```

//Сложение матриц
Mat mat_sum;
add(eye, box, mat_sum);
std::cout << "mat_sum = " << std::endl;
std::cout << mat_sum << std::endl;

//или вариант с перегруженным оператором
mat_sum = eye + box;
std::cout << "mat_sum = " << std::endl;
std::cout << mat_sum << std::endl;

//Вычитание матриц
Mat mat_sbtrct;
subtract(eye, box, mat_sbtrct);
std::cout << "mat_sbtrct = " << std::endl;
std::cout << mat_sbtrct << std::endl;

//или вариант с перегруженным оператором
mat_sbtrct = eye - box;
std::cout << "mat_sbtrct = " << std::endl;
std::cout << mat_sbtrct << std::endl;

//Умножение матрицы на скаляр
Mat mat_sc_mat;
mat_sc_mat = eye*5.0;
std::cout << "mat_sc_mat = " << std::endl;
std::cout << mat_sc_mat << std::endl;

//Поэлементное умножение, или деление
Mat ew_mult, ew_div;
eye.mul(box);
//или
multiply(eye, box, ew_mult);
std::cout << "ew_mult = " << std::endl;
std::cout << ew_mult << std::endl;

ew_div = eye / box;
//или
divide(eye, box, ew_div);
std::cout << "ew_div = " << std::endl;
std::cout << ew_div << std::endl;

//Матричное перемножение
Mat mtx_mult;
mtx_mult = eye*box;
std::cout << "mtx_mult = " << std::endl;
std::cout << mtx_mult << std::endl;

//Транспонирование матриц
Mat mtx_transposed;

```

```

transpose(eye, mtx_transposed);
//или
mtx_transposed.t();
std::cout << "mtx_transposed = " << std::endl;
std::cout << mtx_transposed << std::endl;

//Инвертирование матриц
Mat mtx_inv;
invert(eye, mtx_inv);
//или
eye.inv();
std::cout << "mtx_inv = " << std::endl;
std::cout << mtx_inv << std::endl;

waitKey();
return 0;
}

```

Кроме этого очень полезны функции, вычисляющие различные характеристики матриц. На рис. 3.21 показан примерный вывод программы.

```

#include "stdafx.h"
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>

int main()
{
    //Создадим две матрицы, единичную и состоящую из единиц.
    Mat eye = Mat::eye(3, 3, CV_32F);
    Mat box = Mat::ones(3, 3, CV_32F);

    //Выведем их в консоль
    std::cout << "eye = " << std::endl;
    std::cout << eye << std::endl;
    std::cout << "box = " << std::endl;
    std::cout << box << std::endl;

    //Сумма всех элементов
    cv::Scalar sum_ = sum(box);
    std::cout << "sum_ = " << std::endl;
    std::cout << sum_[0] << std::endl;

    //След матрицы (сумма элементов главной диагонали)
    cv::Scalar trace_ = trace(box);
    std::cout << "trace_ = " << std::endl;
    std::cout << trace_[0] << std::endl;

    //Норма
    cv::Scalar norm_ = norm(box);

```

```

std::cout << "norm_ = " << std::endl;
std::cout << norm_[0] << std::endl;

//Определитель
cv::Scalar det_ = determinant(box);
std::cout << "det_ = " << std::endl;
std::cout << det_[0] << std::endl;

//Собственные числа и векторы
Mat eigen_values, eigen_vectors;
eigen(box, eigen_values, eigen_vectors);
std::cout << "eigen_values = " << std::endl;
std::cout << eigen_values << std::endl;
std::cout << "eigen_vectors = " << std::endl;
std::cout << eigen_vectors << std::endl;

waitKey();
return 0;
}

```

```

E:\VS Projects\ConsoleApplication1\x64\Release\ConsoleApplication1.exe
mtx_transposed =
[1, 0, 0;
 0, 1, 0;
 0, 0, 1]
mtx_inv =
[1, 0, 0;
 0, 1, 0;
 0, 0, 1]
sum_ =
9
trace_ =
3
norm_ =
3
det_ =
0
eigen_values =
[3;
 0;
 0]
eigen_vectors =
[0.57735026, 0.57735032, 0.57735032;
 0.70710677, -0.70710677, 0;
 -0.40824831, -0.40824831, 0.81649661]

```

Рис. 3.21. Консольный вывод программы-примера.

Помимо перечисленных, OpenCV имеет еще целый ряд функций для работы с матрицами. Стоит упомянуть сравнение матриц, спектральные преобразования, проецирования, разложения. Для выполнения практического задания будут полезны:

```

//Вычисление ковариационной матрицы
Mat covar_, mean_;
calcCovarMatrix(box, covar_, mean_, COVAR_ROWS);
std::cout << "covar_ = " << std::endl;
std::cout << covar_ << std::endl;

//Вычисление расстояния Махаланобиса
//Создаем единичную матрицу, две строки которой якобы содержат
векторы признаков двух объектов
Mat mtx_samples = Mat::ones(2, 2, CV_64F);
Mat covar2_, mean2_;

```



```

//Вычисляем и инвертируем ковариационную матрицу
calcCovarMatrix(mtx_samples, covar2_, mean_2, COVAR_ROWS);
invert(covar2_, covar2_);
//Вычисляем расстояние Махаланобиса
Mat r1 = mtx_samples.row(0);
Mat r2 = mtx_samples.row(1);
double dist = Mahalanobis(r1, r2, covar2_);
std::cout << "dist = " << std::endl;
std::cout << dist << std::endl;

```

При работе с функциями OpenCV, содержащими флаги (как, например, `cv::calcCovarMatrix`) следует сверяться с руководством для того, чтобы в зависимости от ситуации выставить соответствующие значения. Это обеспечит корректную работу и желаемый результат.

### 3.6. Кластеризация с помощью алгоритма К-средних. Материалы для выполнения задания из раздела 2.2

В практическом задании требуется самостоятельно реализовать алгоритм кластеризации К-средних. OpenCV имеет готовую реализацию данного алгоритма, пример использования которой приведен в данном разделе. Выполнение данного примера поможет контролировать корректность собственной имплементации К-средних, а также поможет в дальнейшем изучении OpenCV.

За основу взято некоторое цветное изображение (рис. 3.22). У него достаточно ограниченная палитра и с помощью визуального анализа можно предположить, что оптимальное количество кластеров, на которые целесообразно разделить пиксели изображения, сгруппировав их по цвету, равно четырем. Четыре кластера, таким образом, должны соответствовать участкам изображения, содержащим небо, скалу, лед и землю, покрытую зеленью.

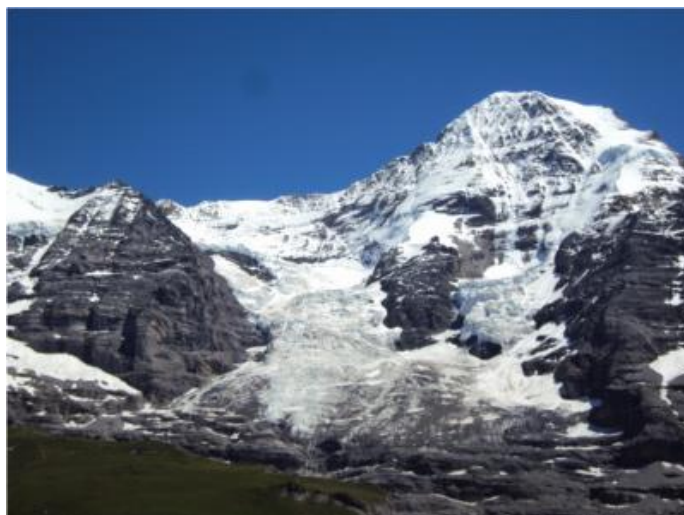


Рис. 3.22 Исходное изображение для кластеризации.

Для начала в качестве признаков для кластеризации используем значения R, G, B пикселей.

```
#include "stdafx.h"
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>

using namespace cv;

int main()
{
    //Загрузка изображения с диска
    Mat img1 = imread("C:/../mnt1.jpg");
    imshow("Input", img1);

    //Осуществим небольшое размытие изображения, чтобы уменьшить
    влияние шума на результат
    blur(img1, img1, Size(3, 3));

    //Подготовим необходимые для работы матрицы. p – должна содержать
    данные для кластеризации, в bestLabels – будут записаны метки для
    каждого пикселя, в centers – координаты центров кластеров.
    Mat p = Mat::zeros(img1.cols*img1.rows, 3, CV_32F);
    Mat bestLabels, centers, clustered;

    //Заполним матрицу p значениями R,G,B пикселей исходного
    изображения
    for (int i = 0; i<img1.cols*img1.rows; i++) {
        p.at<float>(i, 0) = (float)img1.at<Vec3b>(i)[0] / 255.0;
        p.at<float>(i, 1) = (float)img1.at<Vec3b>(i)[1] / 255.0;
        p.at<float>(i, 2) = (float)img1.at<Vec3b>(i)[2] / 255.0;
    }
    //Объявим число кластеров равным четырем и вызовем функцию
    кластеризации
    const int K = 4;
    cv::kmeans(p, K, bestLabels,
    TermCriteria(CV_TERMCRIT_EPS + CV_TERMCRIT_ITER, 10, 1.0), 5,
    KMEANS_PP_CENTERS, centers);
    //Сгенерируем цвета для отрисовки кластеров
    int colors[K];
    for (int i = 0; i<K; i++) {
        colors[i] = 255 / (i + 1);
    }
    //Создадим и заполним матрицу-результат для отображения на экране
    clustered = Mat(img1.rows, img1.cols, CV_32F);
    for (int i = 0; i<img1.cols*img1.rows; i++) {
        clustered.at<float>(i / img1.cols, i%img1.cols) =
        (float)(colors[bestLabels.at<int>(0, i)]);
    }
}
```

```

}
//Конвертируем в тип unsigned char для того, чтобы воспользоваться
//сгенерированными цветами и проведем медианную фильтрацию для
//визуального улучшения результата.
clustered.convertTo(clustered, CV_8U);
medianBlur(clustered, clustered, 3);

imshow("Result", clustered);

waitKey();
return 0;
}

```

Результат приведен на рисунке 3.23.

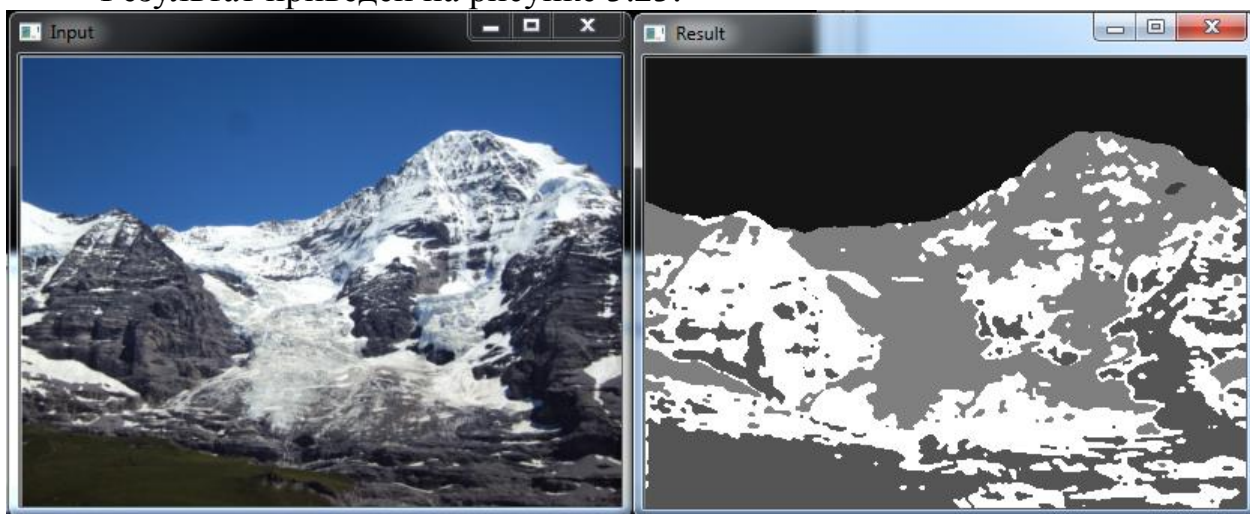


Рис. 3.23. Результат кластеризации при использовании R,G,B признаков.

Как видно, результат получился достаточно неточный. Часть пикселей скалы было отнесено к траве. Заметно, что кластеризация скорее была выполнена по яркости пикселей, чем по цвету.

Для сравнения, попробуем перед заполнением матрицы *p* добавить в код всего одну функцию, которая конвертирует исходное изображение из цветового пространства RGB в пространство HSV (цветовой тон, насыщенность, яркость):

```
cvtColor(img1, img1, CV_BGR2HSV);
```

Функция *cvtColor()* очень часто используется на практике и позволяет осуществлять конвертацию между всеми основными цветовыми пространствами. Самым популярным использованием является перевод изображения из цветного в оттенки серого. Например, для исходного изображения это выглядело бы так:

```
cvtColor(img1, img1, CV_BGR2GRAY);
```

После внесенного изменения результат выглядит уже вполне удовлетворительно (рис. 3.24):

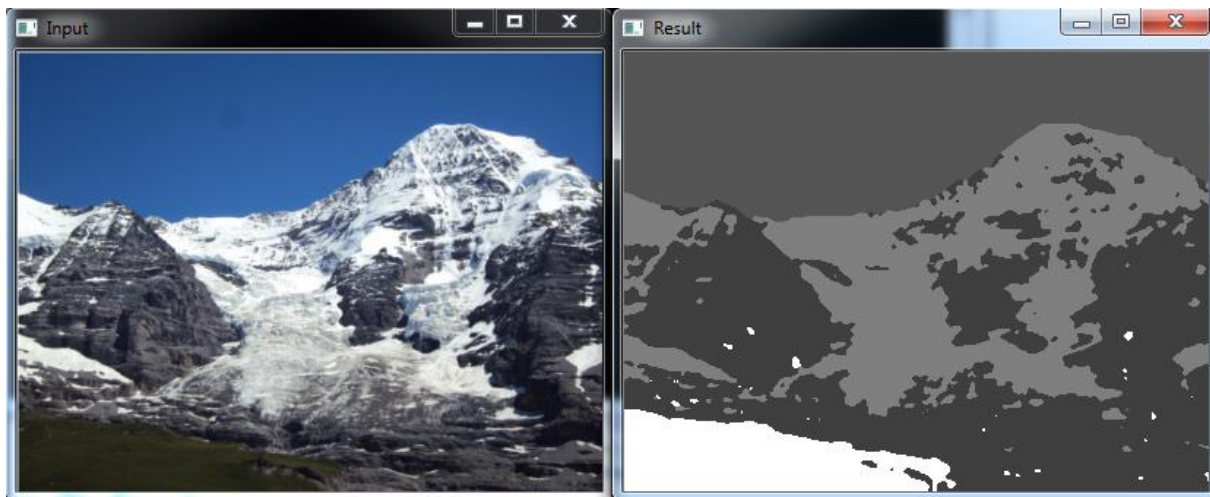


Рис. 3.24. Результат кластеризации при использовании H,S,V признаков.

Очевидно, что несмотря на наличие незначительных ошибок, в целом пиксели изображения были корректно разделены на четыре кластера: небо, скалу, снег и траву.

### 3.7. Разделение и слияние каналов матриц. Материалы для выполнения задания из раздела 2.3

Заключительное задание является достаточно узкоспециализированной задачей цифровой обработки изображений. Разумеется, готового решения в OpenCV для нее не существует. Тем не менее, весь вышеизложенный материал позволяет решить данную задачу средствами библиотеки.

Дополнительно, для выполнения шестого задания необходимо разделять изображения на отдельные каналы и, наоборот, получать цветное изображение, путем совмещения трех каналов. В OpenCV есть удобные функции для этого `cv::split()` и `cv::merge()`.

Ниже приведен пример разделения каналов.

```
//Загружаем цветное трехканальное изображение
Mat RGB_image = imread("C:/../img.png");
//Создаем вектор из объектов класса Mat
std::vector<cv::Mat>sp;
//Разделяем на каналы
cv::split(RGB_image, sp);
//Копируем каждый канал в отдельный объект Mat для дальнейшей работы.
R_image = sp[2].clone();
G_image = sp[1].clone();
B_image = sp[0].clone();
```

Загруженное цветное изображение имело формат *CV\_8UC3*, а отдельные изображения-каналы имеют формат *CV\_8UC1* или просто *CV\_8U*.

Для объединения каналов используем обратную процедуру и функцию `cv::merge()`:

```
//Создаем вектор из объектов класса Mat
std::vector<cv::Mat>sp;
//Загружаем в него отдельные каналы
sp.push_back(B_image.clone());
sp.push_back(G_image.clone());
sp.push_back(R_image.clone());
//Объединяем в единое изображение RGB_image
cv::merge(sp, RGB_image);
```

При выполнении задания имеет смысл воспользоваться функцией взвешенного суммирования изображений (для реализации наложения каналов).

```
//Взвешенное сложение матриц (изображений), после имени изображения
следует константа, определяющая вес. Также есть возможность указать
значение, прибавляемое к сумме (здесь равное нулю)
addWeighted(image1, 0.25, image2, 0.75, 0, output_image);
```

Реализация интерфейса программы может быть различной. Однако, если потребуется осуществлять захват и обработку нажатий кнопки мышки на окне с изображением, выводимым OpenCV (с помощью `cv::imshow()`), то нужно обратить внимание в документации на специальную функцию `cv::SetMouseCallback()` и пример ее использования.

## СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Обухова Н.А., Тимофеев Б.С. Сегментация и сопровождение объектов в сложных условиях видеонаблюдения, Информационно-управляющие системы, № 6, 2008, с. 9-15.
2. Тимофеев Б.С., Обухова Н.А. и др., Обработка изображений в прикладных телевизионных системах (под ред. Б.С. Тимофеева), СПб: Изд-во СПбГУАП, 2012, с. 272
3. Гонсалес Р., Вудс Р. Цифровая обработка изображений. М.: Техносфера, 2005, 1072 с
4. B. Horn and B. Schunck, Determining optical flow, Artificial Intelligence, vol 17, pp 185–203, 1981
5. B. Lucas and T. Kanade, An iterative image registration technique with an application to stereo vision. Proceedings of Imaging Understanding Workshop, pages 121—130
6. Bishop, Pattern Recognition and Machine Learning, Chapter in Journal of Electronic Imaging 16(4), 140-155, 2006
7. Флах П. Машинное обучение, М.: ДМК Пресс, 2015, 400 с.
8. Дрейпер Н., Смит Г. Прикладной регрессионный анализ. Множественная регрессия = Applied Regression Analysis. — 3-е изд. — М.: «Диалектика», 2007. — С. 912.

Обухова Наталия Александровна,  
Мотыко Александр Александрович

**Видеоаналитика и интеллектуальный анализ данных**

Учебное пособие

Редактор \_\_\_\_\_

---

Подписано в печать \_\_\_\_\_. Формат 60×84 1/16.  
Бумага офсетная. Печать цифровая. Печ. л. 8,25.  
Гарнитура «Times New Roman». Тираж \_\_\_\_ экз. Заказ 000.

---

Издательство СПбГЭТУ «ЛЭТИ»  
197376, С.-Петербург, ул. Проф. Попова, 5