Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

А.В. Бобков

Системы распознавания образов

Учебное пособие

BAUMANPRESS
Москва
ИЗДАТЕЛЬСТВО
МГТУ им. Н. Э. Баумана
2018

УДК 004.93'1
ББК 32.97
Б72

Издание доступно в электронном виде на портале *ebooks.bmstu.ru* по адресу: http://ebooks.bmstu.press/catalog/200/book1831.html

Факультет «Информатика и системы управления»
Кафедра «Системы автоматического управления»

Рекомендовано Редакционно-издательским советом МГТУ им. Н.Э. Баумана в качестве учебного пособия

Бобков, А. В.

Б72 Системы распознавания образов : учебное пособие / А. В. Бобков. — Москва : Издательство МГТУ им. Н. Э. Баумана, 2018. — 187, [3] с. : ил.

ISBN 978-5-7038-4867-8

Приведены основные подходы к решению задач обработки изображений, определения параметров объектов, поиска объектов на изображении. Рассмотрены теоретические основы методов распознавания, особенности их программно-алгоритмической реализации, а также практическое применение в задачах автоматического управления

Для студентов МГТУ им. Н.Э. Баумана, обучающихся по специальности «Системы автоматического управления»; может быть полезно инженерам и исследователям, которые занимаются проектированием систем распознавания образов, а также специалистам других направлений деятельности, чьи области интересов включают в себя системы распознавания как важный составной элемент.

УДК 004.93'1
ББК 32.97

ISBN 978-5-7038-4867-8

Предисловие

Учебное пособие «Системы распознавания образов» знакомит с основными подходами к решению задач обработки изображений, определения параметров объектов, поиска объектов на изображении. В пособии рассмотрены теоретические основы методов распознавания, особенности их программно-алгоритмической реализации, а также практическое применение в задачах автоматического управления.

Задача пособия — ознакомить студентов с тремя уровнями анализа изображений.

1. *Нижний уровень.* Изучение особенностей и специфики получения изображений, обработка изображений на уровне отдельных точек: коррекция ракурса и условий видимости, фильтрация шума и помех (глава 1).

2. *Средний уровень.* Рассмотрение задачи представления объектов в виде совокупности структурных элементов: контуров, остовов, многоугольников, отрезков и дуг, пятен, особых точек (главы 2 и 3).

3. *Высокий уровень.* Решение задач поиска и идентификации объектов, определения их взаимосвязей (главы 4 и 5).

Учебное пособие может представлять интерес как для инженеров и исследователей, непосредственно занимающихся проектированием систем распознавания образов, так и для специалистов других направлений деятельности, чьи области интересов включают в себя системы распознавания как важный составной элемент.

Список сокращений

БПЛА — беспилотный летательный аппарат
БПФ — быстрое преобразование Фурье
ДПФ — дискретное преобразование Фурье
ДТЧП — дискретное теоретико-числовое преобразование
ИК — инфракрасный (диапазон)
ИПФ — импульсная переходная функция
КТ — ключевая точка
МНК — метод наименьших квадратов
МО — математическое отклонение
ПП — пространство параметров
СКО — среднеквадратическое отклонение
ЦМ — центр масс
GHT — Generalized Hough Transform, обобщенное преобразование Хафа
HT — Hough Transform, алгоритм (преобразование) Хафа
SHT — Standart Hough Transform, стандартное преобразование Хафа

Введение

Понятие изображения

Объекты, с которыми работают системы автоматического управления, существуют объективно, независимо от человека, его сознания и знаний о них. Информация о свойствах объекта доступна только по результатам измерений (а также чувственного восприятия). Такой набор измерений называется *изображением объекта* в широком смысле (рис. В.1). В узком смысле под изображением будем понимать видеоизображение, т. е. цифровую фотографию объекта — прямоугольный массив отсчетов интенсивности видимого света, отраженного от объекта.

Именно такое изображение, более привычное и понятное, нашло наиболее широкое применение, поэтому будем ориентироваться именно на него.

Однако проблема состоит в том, что полученное изображение не несет в себе в явном виде информацию о свойствах объекта; оно несет информацию об интенсивности электромагнитного излучения, пришедшего с данного направления, а не о классе объекта, его скорости и направлении движения, траектории, намерениях, взаимодействии с другими объектами. Все эти свойства, или *вектор состояния объекта* (в терминах теории управления), должны быть получены из исходного изображения в процессе *распознавания* (см. рис. В.1). Таким образом, в широком смысле под распознаванием изображения будем понимать процесс получения свойств объекта из результатов доступных измерений (в терминах теории управления — это задача синтеза

Рис. В.1. Постановка задачи распознавания

наблюдателя). В узком смысле под распознаванием будем понимать задачи классификации объекта.

Распознавание изображения необходимо как важный компонент системы управления в целом. На основе свойств объекта, полученных в результате распознавания изображения, системой управления будут приняты решения и выработаны команды управления, которые будут поданы на объект. Таким образом контур управления замыкается, и система управления становится ее важной частью. К ней как компоненту системы предъявляются технические требования по производительности, точности и надежности работы, а также — по стоимости технической реализации.

Производительность системы распознавания оценивается скоростью работы автоматической системы в режиме реального времени. Система распознавания должна успевать обрабатывать кадры изображения по мере их поступления с учетом скоростных характеристик объекта: если время распознавания кадра будет слишком велико, объект может, например, уйти из поля зрения камеры и будет потерян.

Другое требование по производительности связано со спецификой контура управления. Система распознавания как компонент системы управления вносит задержку между изменением состояния объекта и обнаружением этого объекта (т. е. может быть представлена *звеном чистого запаздывания* сигнала). Наличие запаздывания может приводить к потере устойчивости системы управления.

Надежность системы распознавания связана с необходимостью сохранения ее работоспособности под воздействием факторов окружающей среды. Строго говоря, это свойство правильнее называть *робастностью*. Среди факторов, влияющих на робастность, можно выделить три группы.

1. Шумы и помехи. На изображении всегда будет присутствовать шум, связанный, например, с тепловыми флуктуациями сенсора, и избавиться от него полностью невозможно. Объект может частично или полностью перекрываться другими объектами, попадать в тень, создавать блики. Кроме того, возможно наличие специфических помех, связанных, например, с электромагнитным воздействием соседней аппаратуры. Доля шума и помех особенно сильно возрастает при плохих условиях видимости, на фоне слабого полезного сигнала.

2. Изменения условий видимости. Для систем, работающих на улице, вне контролируемых условий освещенности, изменения условий видимости становятся важным фактором. Эти изменения могут происходить вследствие сезонных и суточных изменений условий освещенности, погодных условий, появления посторонних источников света и т. д.

3. Изменения ракурса. В процессе съемки может изменяться ракурс съемки объекта: объект перемещается по изображению, приближается или удаляется, изменяясь по размерам, поворачиваясь по всем трем осям.

Система распознавания должна строиться таким образом, чтобы воздействие указанных факторов не снижало существенно качество ее работы.

Особенности задач распознавания. В общем случае задача распознавания математически относится к некорректно поставленным задачам, и общих

методов ее решения не существует. Всегда приходится искать частные решения, учитывающие специфику и контекст практической задачи.

Технически анализ изображения достаточно прост — его можно получать с помощью существующих технических средств, процесс получения изображения обычно носит пассивный характер и не требует воздействия на объект. Изображение в качестве источника сигнала, как правило, достаточно информативно и содержит большое количество информации об объекте. Вместе с тем, сам объем информации в изображении огромен, и, как следствие, возрастает вычислительная сложность алгоритмов распознавания и возникает необходимость обрабатывать изображения в режиме реального времени.

Способы получения изображений

Изображения могут быть получены различными физическими способами.

Наиболее простой и наиболее часто используемый на практике способ — это получение *фотографического изображения* в видимом диапазоне. Это фотографии и видеоизображения. В дальнейшем будем говорить в основном об этом типе изображений как об изображении в узком смысле, предполагая, что методы обработки видеоизображений и изображений иной физической природы во многом сходны.

Видеоизображения можно подразделить на цветные и черно-белые (полутоновые). Точки черно-белого изображения содержат значения суммарной интенсивности падающего в данном направлении отраженного света. Точки цветного изображения содержат по меньшей мере три значения, соответствующие интенсивностям падающего света в трех спектральных диапазонах. Как правило, спектральные диапазоны выбирают таким образом, чтобы их середина соответствовала максимальной чувствительности одного из трех видов колбочек человеческого глаза, отвечающих за восприятие красного, зеленого и синего цветов.

Видеоизображения могут быть статическими, представляющими один момент времени (фотография), или в виде набора снимков, полученных с некоторым интервалом (телевизионное изображение).

Инфракрасное изображение представляет собой изображение в области спектра ниже порога чувствительности человеческого глаза. Инфракрасный (ИК) участок спектра разделяют на ближний и дальний.

Ближний ИК-участок во многом сходен с видимым участком спектра, однако невидим человеческим глазом. Тем не менее ближнее ИК-излучение, как правило, воспринимается обычными фото- и видеокамерами, если не принять специальных мер. В ближнем ИК-диапазоне работают ИК-прожекторы подсветки, подсвечивая изображение невидимым для человека, но вполне видимым для камеры излучением. В этом же диапазоне работают, например, ИК-пульты дистанционного управления.

Дальний ИК-участок представляет собой спектр теплового излучения тел с температурой −50... + 200 °С. Этот участок спектра также называют

тепловым. Особенность этого диапазона в том, что в нем хорошо видны нагретые (или, наоборот, охлажденные) по отношению к окружающей среде предметы. На тепловом изображении хорошо заметны, например, люди и животные, двигатели наземной и авиационной техники. Тепловое излучение очень сложно скрыть или замаскировать, поскольку это приведет к уменьшению теплоотвода и перегреву объекта, поэтому оно может использоваться там, где обычное изображение малоинформативно. Примеры тепловизионных изображений показаны на рис. В.2.

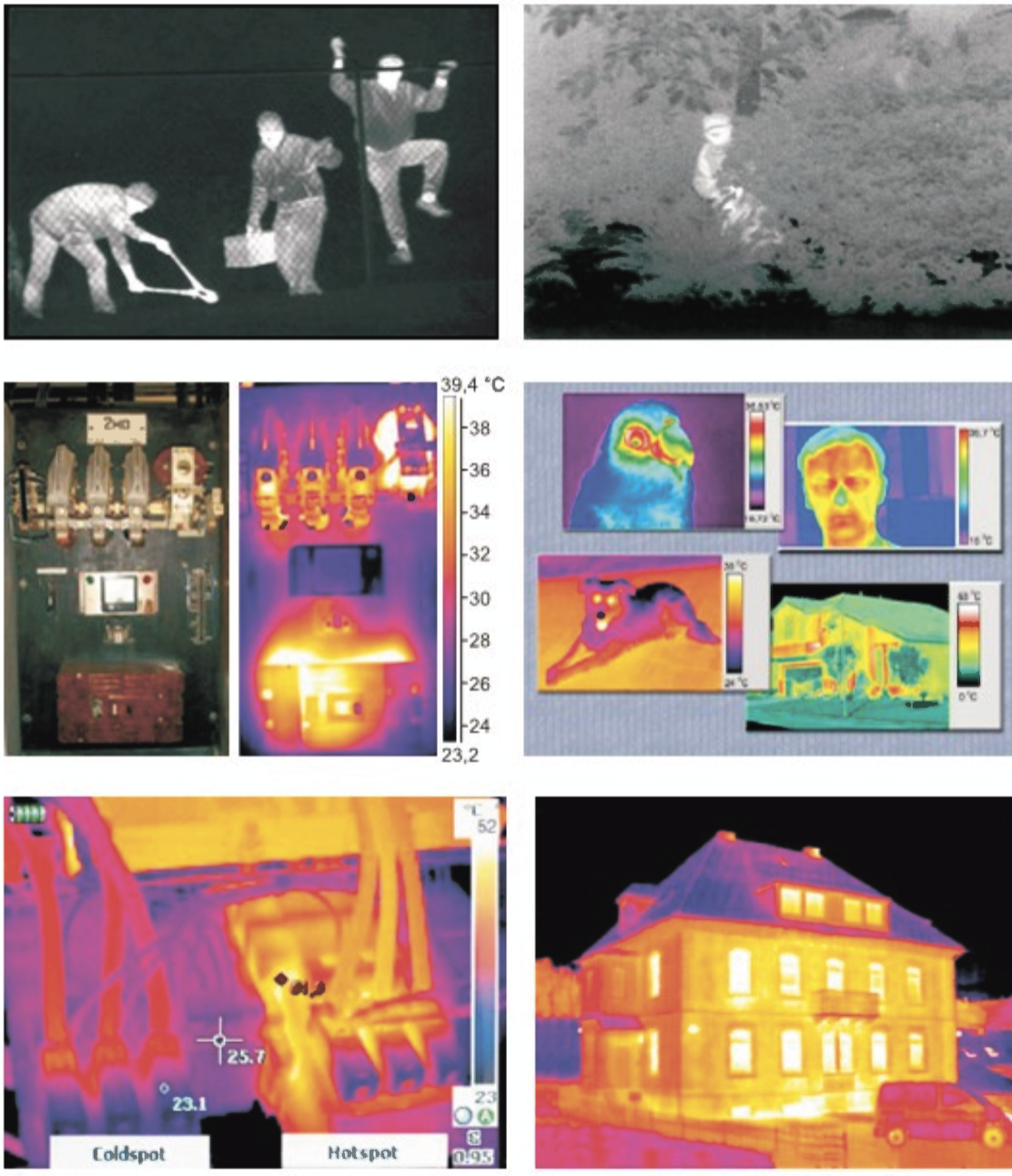


Рис. В.2. Примеры тепловых изображений

Тепловое изображение невозможно получить с помощью обычной камеры, для этого применяются специальные устройства — *тепловизоры*, использующие чувствительную к тепловому излучению матрицу. Как правило, в структуре матрицы тепловизора существует пассивное (резервуар с жидким азотом) или активное (холодильник) охлаждение, что делает тепловизор достаточно дорогостоящим устройством.

Рентгеновские изображения (рис. В.3) позволяют увидеть внутреннюю структуру объекта и выявить ее дефекты, что невозможно сделать с помощью отраженных или испускаемых объектом электромагнитных волн. Рентгеновское излучение лежит в более высокочастотной области спектра. Его фотоны обладают высокой энергией, что позволяет им проходить сквозь объекты, непрозрачные в других областях спектра.

Рентгеновское излучение подразделяют на поддиапазоны: можно выделить «мягкое» и «жесткое» рентгеновское излучение. «Мягкий» рентген

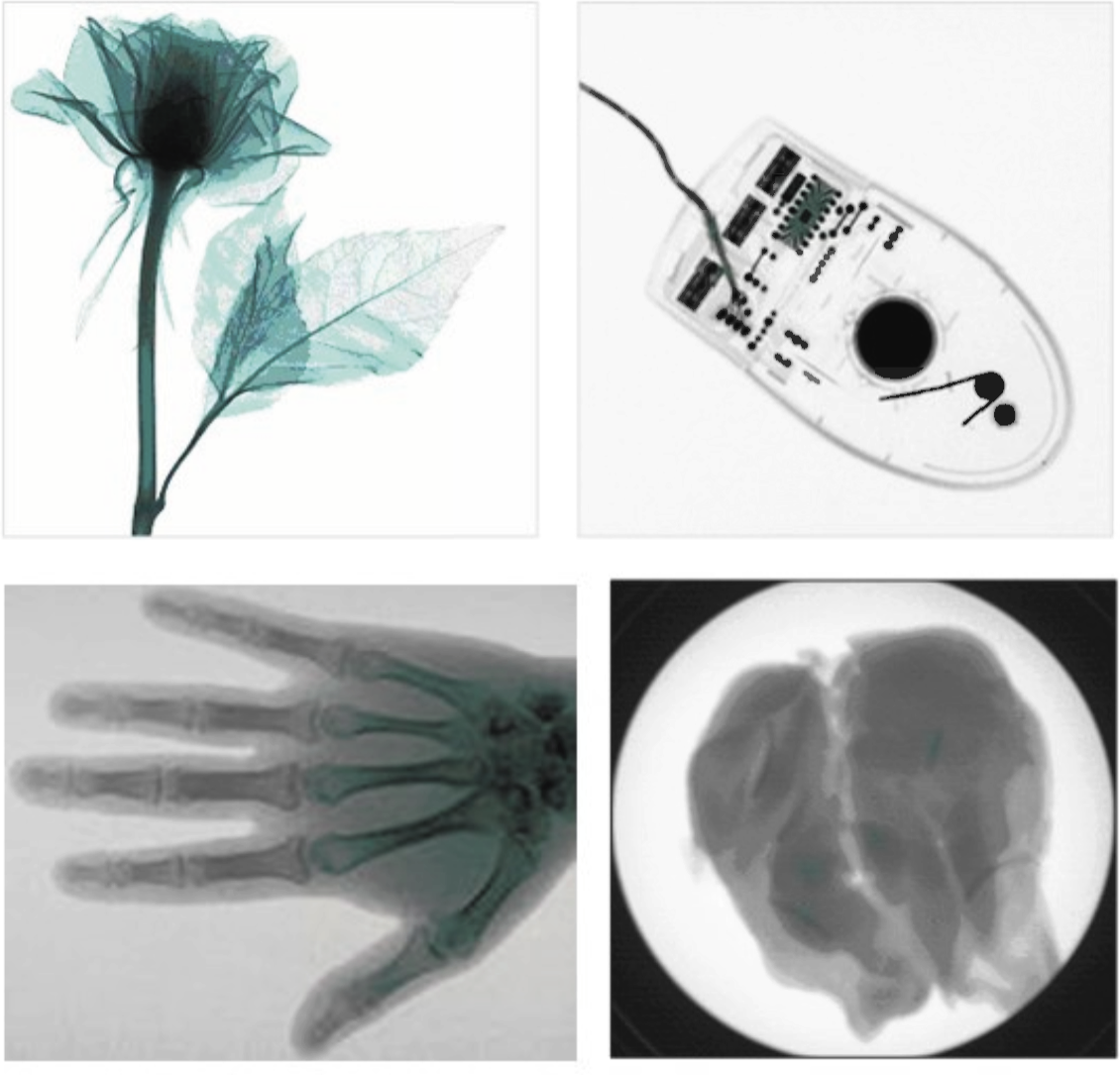


Рис. В.3. Примеры рентгеновских изображений

нашел широкое применение в биомедицинской технике, в установках по контролю качества изготовления пластиковых изделий, например многослойных печатных плат.

«Жесткий» рентген может использоваться для неразрушающего контроля качества металлических изделий, однако в силу его чрезмерной опасности для человека его применение требует комплекса соответствующих мер, что сдерживает его широкое применение. Поэтому «жесткий» рентген используется лишь для контроля критически важных изделий, контроль которых другими методами невозможен.

Радиолокационное изображение позволяет наблюдать объект посредством регистрации отраженных от него радиоволн. В простейшем случае регистрируются направление на объект, задержка распространения радиосигнала, позволяющая вычислить расстояние до объекта, и интенсивность сигнала, дающие представление о материале и размерах объекта.

В более сложном случае можно последовательно сканировать изучаемый объект и измерять характеристики каждой его точки, получая тем самым радиолокационный аналог растрового изображения. Более того, современная техника позволяет получать моментальный радиолокационный снимок объекта (рис. В.4), во многом идентичный растровому снимку.

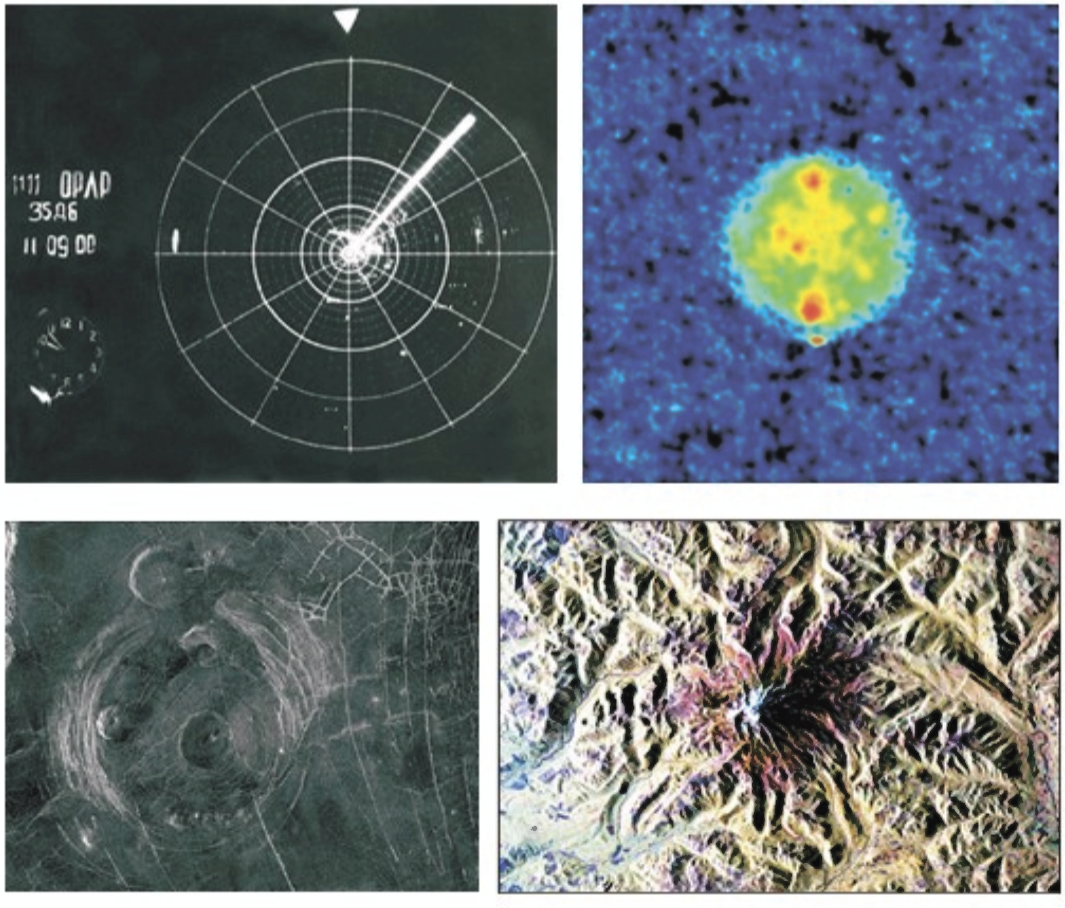


Рис. В.4. Примеры радиолокационных изображений

В *дальнометрическом изображении* каждая точка несет информацию о расстоянии до соответствующей точки объекта. Для измерения используется определение времени прихода отраженного сигнала с известной скоростью распространения в данной среде. В качестве сигнала могут быть выбраны ультразвук (эхолокация, сонар), лазерный луч или радиосигнал (рис. В.5).

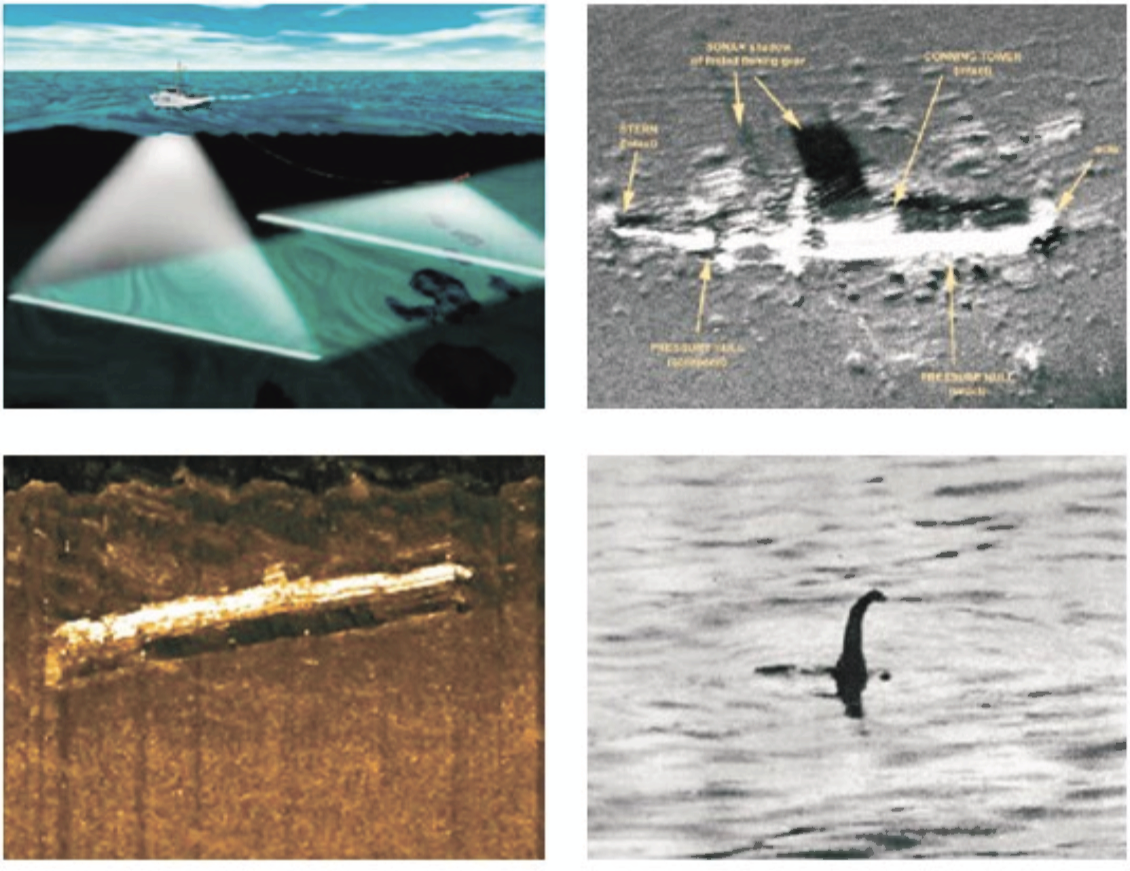


Рис. В.5. Примеры дальнометрических изображений

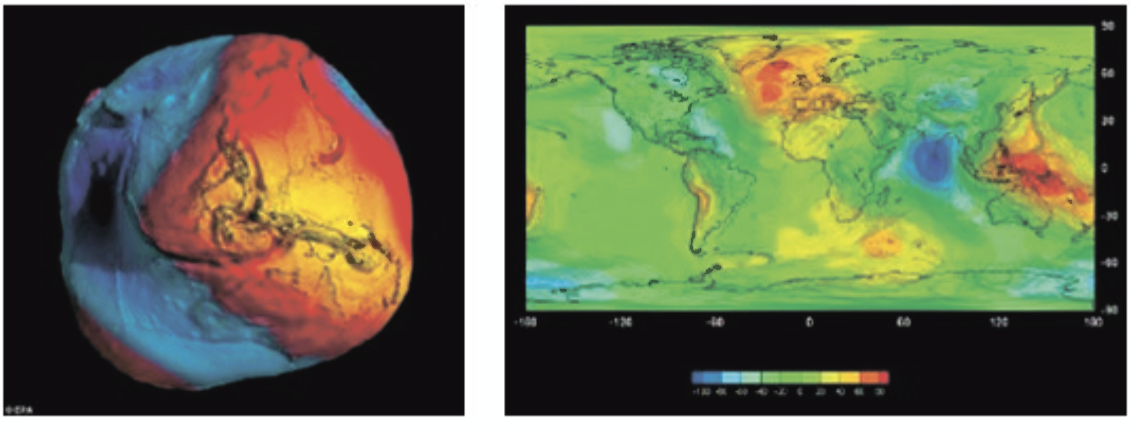


Рис. В.6. Примеры гравитационных изображений

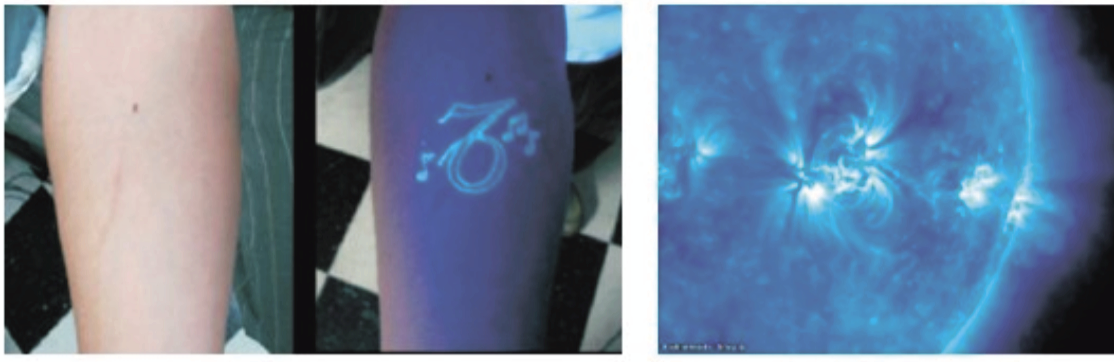


Рис. В.7. Примеры ультрафиолетовых изображений

Помимо рассмотренных принципов для получения изображений могут использоваться поля и иной (магнитной, гравитационной и т. д.) физической природы (рис. В.6 и В.7). Они находят применение, например, в задачах спутниковой навигации.

Задачи распознавания изображений

Задачами распознавания изображений являются:

автоматизация конвейера;

контроль качества изготовления (подсчет числа отверстий в детали), отбраковка, сортировка, определение ориентации детали, захват деталей из навала;

автоматический захват и сопровождение цели;

уклонение от столкновений;

автоматическая стыковка;

определение собственного положения по видеоизображению (для коррекции инерциальной навигационной системы на критических участках траектории либо в качестве самостоятельной дублирующей системы навигации);

биометрическая идентификация и контроль доступа;

автоматическое распознавание текстов;

распознавание дорожных сцен;

распознавание в биомедицинских приложениях;

системы распознавания в быту («умный дом»).

Особенности решения задачи распознавания изображений. Необходимо отметить, что задачи распознавания изображений решаются относительно простыми методами. Однако по мере их решения могут возникнуть различные нюансы, которые связаны с природой решаемых задач и учет которых требует значительного числа совершенно разнородных и весьма нетривиальных методов.

Рассмотрим в качестве примера задачу поиска объекта на изображении. Пусть имеется изображение, содержащее искомый объект. Будем считать, что изображение имеет вид двумерного массива, где нули соответствуют темным точкам объекта, а единицы — светлым. Пусть оператор также задал образец

для поиска — также в виде двумерного массива, но меньшего размера. Требуется разработать алгоритм обнаружения объекта-образца на изображении.

Простейший способ решения задачи заключается в том, чтобы, прикладывая образец к изображению во всех возможных положениях, считать число совпавших точек. Если в каком-либо положении совпали все точки — значит, именно там и находится искомый объект.

Как видно, ничего сложного в таком алгоритме нет, и читатель, несомненно, сможет легко реализовать подобный алгоритм с помощью любого из известных ему языков программирования. Но, несмотря на кажущуюся простоту такого алгоритма поиска, реальные задачи могут оказаться сложнее и могут потребовать внести в алгоритм существенные коррективы. Предположим, например, что объект на изображении и образец не совпадают точка в точку (например, из-за воздействия шума). Тогда данный критерий обнаружения объекта по факту совпадения всех точек уже не подходит, поскольку совпадет лишь часть точек. Насколько велика эта часть, зависит от многих факторов: уровня шума, наличия на изображении объектов, похожих на образец; допустимой вероятности пропуска объекта или ошибочного обнаружения. Это уже отдельная, достаточно сложная задача.

Более того, объект на изображении также может быть снятым немного в другом (заранее не известном) масштабе, ракурсе, при иной ориентации, нежели заданный оператором объект поиска. Это сильно расширяет понятие «положение», включая в него, помимо координат x и y, масштаб, угол поворота и параметры ракурса. Добавление каждого нового параметра намного увеличивает число возможных положений: их число умножается на число градаций этого параметра. Так, если объект на изображении может быть дополнительно повернут на любой угол с шагом в 1°, число возможных положений возрастет в 360 раз, и просмотреть все возможные положения за сколько-нибудь разумное время невозможно. В этом случае необходимо использовать стратегию поиска, позволяющую избежать просмотра всех возможных положений. Это можно сделать, например, путем выполнения поиска в дуальных пространствах — пространстве собственных параметров гипотез поиска или в спектральном пространстве.

Далее, цвет и форма объекта в ряде задач также могут меняться: например, объект или его окружение могут давать блики и тени, в результате чего темные области становятся светлыми и наоборот. Яркость отдельных точек в этом случае больше не является надежным признаком, позволяющим найти объект. Следует использовать другие, более надежные признаки, например контур объекта, особенности формы, окраски или текстуры, характер его движения и др. Какие именно признаки следует выделять и использовать в той или иной задаче — отдельный и достаточно сложный вопрос.

Таким образом, задача автоматического распознавания изображений — это практически всегда комплексная задача, вовлекающая применение широкого спектра различных методов, состав и мера применимости которых всегда анализируются в контексте решаемой задачи. Вместе с тем, можно отобразить наиболее общий подход к задачам распознавания изображений.

Этапы решения задачи распознавания

В самом общем случае решение задачи распознавания состоит из следующих этапов:

1) получение изображения;
2) предобработка;
3) фильтрация;
4) выделение признаков;
5) анализ изображения (распознавание и сопоставление);
6) принятие решения.

Некоторые этапы могут быть пропущены, если в них нет необходимости. В ряде случаев этапы могут идти нелинейно, например, если на последующих этапах была получена дополнительная информация, необходимая для работы предыдущих этапов.

Рассмотрим этап получения изображения, а этапам предобработки, фильтрации, выделения признаков и анализа изображения будут посвящены последующие главы.

Этап получения изображения. Получение изображения, несмотря на кажущуюся простоту, — сложная инженерная задача, и от ее удачного решения зачастую может зависеть результат решения всей задачи распознавания. Этап получения изображения предполагает определение физических принципов получения изображения, выбор параметров камеры и формата изображения, выбор ракурса и учет условий видимости.

Выбор физических принципов получения изображения. Как правило, физический принцип получения изображения зависит от условий видимости объекта в той или иной области спектра (контрастности по сравнению с фоном, прозрачности объекта либо фона и т. д.), а также от стоимости устройства получения изображения и его характеристик — разрешающей способности, времени экспонирования и др. Например, если необходимо получить изображение объекта в условиях низкой освещенности, следует использовать либо камеру с ИК-подсветкой, либо (при невозможности ее применения) тепловое изображение.

На этом этапе следует также предусмотреть дополнительные средства измерения, способные существенно упростить задачу распознавания. Например, можно оценить скорость объекта по последовательности видеоизображений, но гораздо проще это сделать, дополнительно оборудовав систему доплеровским измерителем скорости. Аналогично для определения расстояния до объекта систему желательно оборудовать дальномером, а для определения собственного положения и перемещения камеры можно применять датчики угловой скорости и акселерометры. Эти достаточно простые и дешевые устройства могут существенно упростить дальнейшее решение задач распознавания.

При прочих равных условиях будем считать, что для получения изображения будет применяться обычная фото- или видеокамера.

Выбор параметров камеры и формата изображения. Основными параметрами видеокамеры являются:

- разрешение;
- размер изображения;
- время экспозиции;
- чувствительность;
- формат изображения:

  аналоговое (телевизионный формат);

  цифровое — кодированный видеопоток (со сжатием, без сжатия) или в виде файлов (несжатые — raw, bmp; сжатые — jpg).

Выбор ракурса. На этом этапе осуществляется выбор оптимального размещения камеры, для того чтобы передавать изображение без искажений (в идеале — плоское изображение) и при этом не мешать функционированию остальных систем.

Учет условий видимости. В ряде задач (например, в задачах технического зрения) имеется возможность устанавливать свои условия видимости, позволяющие более легко и надежно отделять объект от фона. Зачастую это существенно упрощает распознавание, и пренебрегать такой возможностью не стоит.

Условия видимости объекта можно менять, либо используя те или иные источники освещения и варьируя их тип, количество и положение, либо изменяя фон, его яркость, цвет, отражательную способность, текстуру и другие свойства.

Источник освещения следует выбирать так, чтобы он обеспечивал достаточную контрастность изображения и в то же время не давал сложных теней и бликов, которые могут помешать дальнейшему распознаванию. При возможности выбора фона он должен существенно отличаться от точек объекта по яркости или цвету.

Глава 1
ПРЕДОБРАБОТКА ИЗОБРАЖЕНИЙ

Этап предобработки необходим, поскольку на этапе получения изображения не всегда удается получить его в желаемом ракурсе, условиях видимости и в надлежащем качестве. Тем не менее, если параметры ракурса, освещенности и помех заранее известны, то появляется возможность их скорректировать алгоритмически. Для этого после получения изображения вводят этап предобработки. Отметим, что этот этап можно пропустить, если подобная предобработка не требуется либо, наоборот, для ее осуществления не хватает информации.

Предобработка изображения включает следующие этапы:

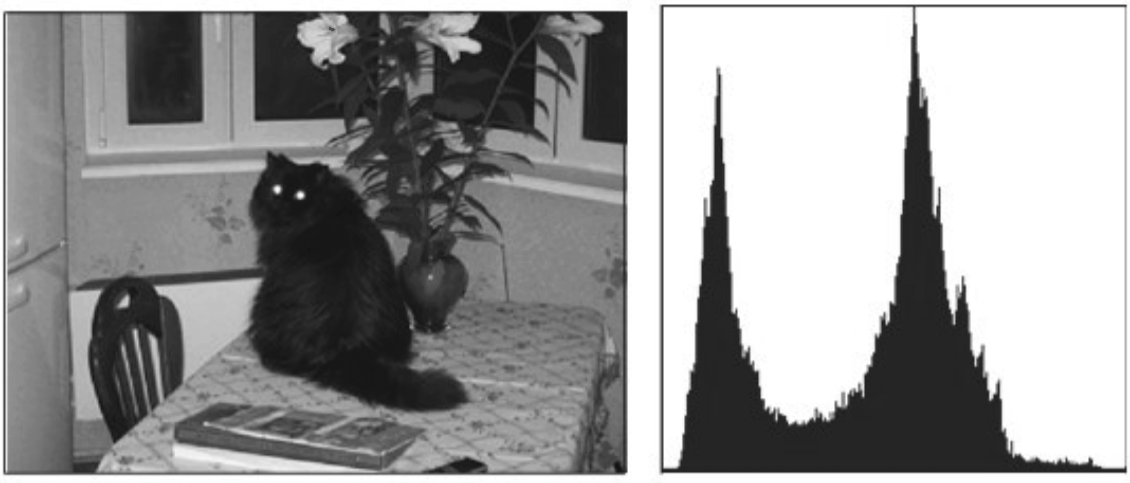
1) коррекция условий видимости;
2) коррекция ракурса;
3) подавление помех и шума.

Коррекция условий видимости

Коррекция условий видимости необходима в том случае, если условия получения изображений существенно отличаются от желаемых. Это может быть обусловлено разной интенсивностью источника освещения (день, утренние или вечерние сумерки, облачность), положением источника света (например, из-за суточного движения солнца), типом источника света (солнечный свет, лампа накаливания, люминесцентная лампа), влиянием помех (засветка от фар проезжающих автомашин) и другими факторами. Однако, зная параметры освещенности, зачастую алгоритмически можно скорректировать изображение так, чтобы приблизить его к желаемому.

Понятие о гистограмме яркости. Алгоритмы предобработки станут существенно более понятными, если ввести в рассмотрение *гистограмму яркости*. Она строится следующим образом. Для каждого из возможных значений яркости $I = 0, \ldots, N - 1$ подсчитаем, сколько раз $H(I)$ данная яркость встречается на изображении. Получим гистограмму $H(I)$ (рис. 1.1).

Алгоритм построения гистограммы яркости следующий. Представим гистограмму яркости массивом счетчиков. Будем просматривать все точки изображения и для каждой из них будем увеличивать на единицу счетчик, соответствующий ее яркости. В результате получим искомую гистограмму. Такой алгоритм выполняется достаточно быстро, за один проход по изображению.



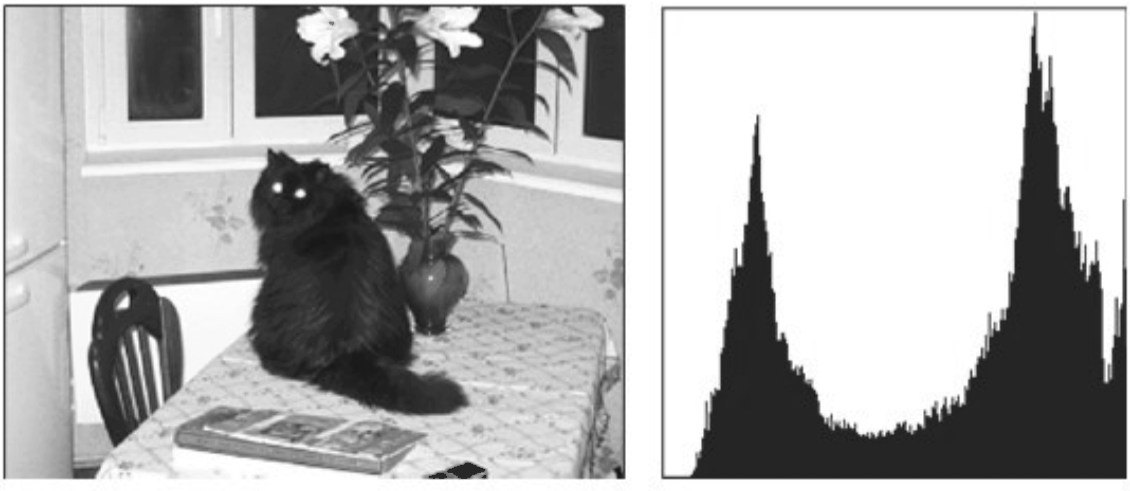
а *б*

Рис. 1.1. Изображение (*а*) и его гистограмма яркости (*б*)

В дальнейшем будем считать, что изображение представлено 256 градациями яркости, пронумерованными от 0 до 255, поскольку в большинстве практических задач используется именно такое представление.

Яркость изображения в терминах гистограммы яркости будет представлять собой положение центра масс гистограммы. Линейная коррекция яркости будет сводиться к добавлению фиксированного числа к значениям яркости каждой точки, что будет приводить к сдвигу гистограммы вправо или влево на это число. Если при этом новые значения яркости выходят за пределы гистограммы, т. е. за пределы максимального или минимального значения яркости, их необходимо заменять на соответствующее ближайшее (максимальное либо минимальное) значение (рис. 1.2).

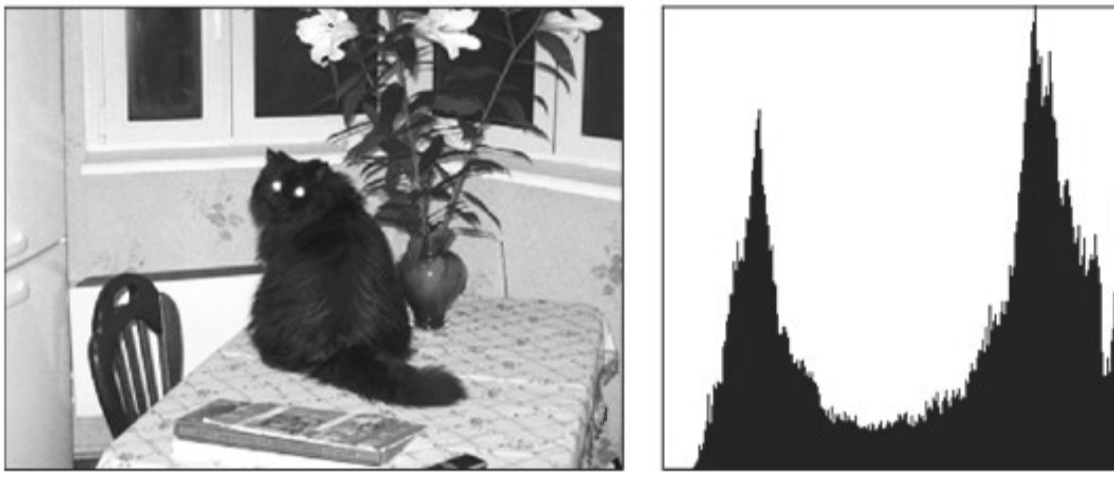
Контрастность изображения в терминах гистограммы яркости будет представлять собой диапазон, занимаемый ненулевыми значениями гистограммы. Коррекция контрастности приведет к увеличению или уменьшению



а *б*

Рис. 1.2. Изображение с повышенной яркостью (*а*) и его гистограмма яркости (*б*)

ширины гистограммы. При этом положение центра масс гистограммы (т. е. яркость изображения) не изменится (рис. 1.3). Яркость и контрастность изображения — две взаимно независимые величины. Тем не менее после коррекции контрастности необходимо заменить значения яркости точек, вышедших за пределы допустимых значений, на ближайшие допустимые значения, что может привести к небольшому изменению яркости изображения.



а *б*

Рис. 1.3. Изображение с повышенной контрастностью (*а*) и его гистограмма яркости (*б*)

Автоматическая коррекция яркости и контрастности. Можно выполнять автоматическую коррекцию яркости и контрастности таким образом, чтобы гистограмма яркости заняла весь диапазон значений. Это будет соответствовать наилучшей контрастности, при которой все еще не происходит потери информации об исходных яркостях точек.

Пусть гистограмма яркости занимает диапазон от a до b; при этом полный диапазон, который может занимать гистограмма, составляет от 0 до $N - 1$. Тогда формула коррекции выглядит следующим образом:

$$I' = (I - a)\,(N - 1)/(b - a). \qquad (1.1)$$

Отметим, что число точек изображения, яркость которых необходимо скорректировать согласно формуле (1.1), как правило, намного больше возможных значений яркости N. Поэтому для каждого значения яркости можно занести новое, скорректированное значение в таблицу $T[I]$, а потом воспользоваться этой таблицей для пересчета яркостей отдельных точек.

Алгоритм коррекции может быть записан следующим образом[1].

1. Найти минимальную a и максимальную b яркости точек изображения.
2. Заполнить таблицу пересчета яркости:
   $T[i] = (i - 1)(N - 1)/(b - a)$, $i = 0, \ldots, N - 1$.

[1] Для записи алгоритма здесь и далее будем использовать метаязык, который без особого труда можно перевести на любой язык программирования высокого уровня.

3. Для каждой точки изображения с яркостью $I[x, y]$ взять новое значение яркости из таблицы T:
$I[x, y] = T[I[x, y]]$, $x = 0, ..., X_{max}$, $y = 0, ..., Y_{max}$.

Таким образом, получили быстрый однопроходный алгоритм, не требующий сложных операций (умножения и деления) для коррекции яркости точек. Результат работы алгоритма показан на рис. 1.4.

Автоматическая коррекция яркости и контрастности с потерями. «Хвосты» гистограммы яркости содержат, как правило, относительно мало точек, тогда как занимаемый ими диапазон значений достаточно широк. Зачастую тут могут оказаться шумовые точки и помехи, например блики. Идея *коррекции с потерями* заключается в том, чтобы пренебречь этими «хвостами». Пусть общее число точек в «хвосте» составляет не более x % от общего числа точек (в зависимости от задачи x может принимать значения от 2 до 10 %). Тогда границу «хвоста» можно найти, суммируя столбцы гистограммы до тех пор, пока не наберем x % общего числа точек изображения.

Модифицированный алгоритм будет выглядеть следующим образом.

1. Построить гистограмму яркостей:
   1.1. Обнулить ячейки $H[i]$, $i = 0, ..., N - 1$.
   1.2. Для каждой точки с яркостью $I[x, y]$ увеличить на единицу ячейку $H[I[x, y]]$, $x = 0, ..., X_{max}$, $y = 0, ..., Y_{max}$.
2. Найти границы «хвостов»:
   2.1. Пока масса левого «хвоста» меньше x %, увеличить длину «хвоста» a на единицу.
   2.2. Пока масса правого «хвоста» меньше x %, уменьшить положение «хвоста» b на единицу.
3. Заполнить таблицу пересчета яркости:
   $T[i] = (i - 1)(N - 1)/(b - a)$, $i = 0, ..., N - 1$.
4. Для каждой точки изображения с яркостью $I[x, y]$ взять новое значение яркости из таблицы T:
   $I[x, y] = T[I[x, y]]$, $x = 0, ..., X_{max}$, $y = 0, ..., Y_{max}$.

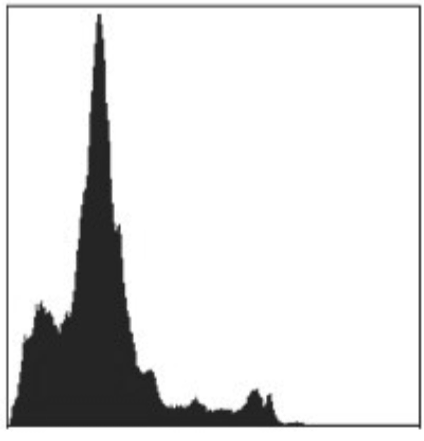
Необходимо отметить, что коррекция яркости и контрастности и дает хороший визуальный эффект, тем не менее она не привносит в изображение никакой новой информации. Такая коррекция имеет смысл, если с изображением дальше будет работать оператор. Если же изображение будет обрабатываться автоматически, вопрос о целесообразности рассматриваемой коррекции остается нерешенным.

Нелинейная коррекция. Таблицу пересчета яркости T, построенную в предыдущем алгоритме, можно рассматривать как переходную характеристику, задающую пересчет яркостей точек старого изображения в яркости нового.

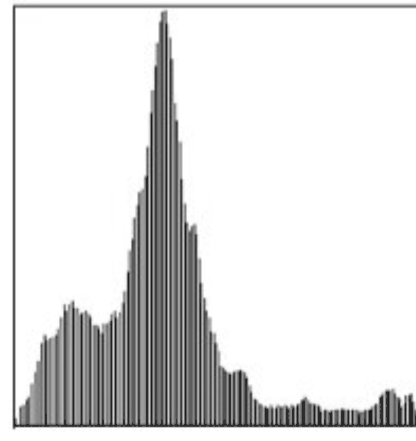
Очевидно, что в предыдущих алгоритмах переходная характеристика выглядела как отрезок, соединяющий точки a и b. Однако прямая линия — не единственная возможная форма коррекции. Точки a и b можно соединить и другими кривыми, проходящими как выше, так и ниже отрезка. Для аппроксимации таких кривых можно использовать, например, полиномы вида $I' = I^{\gamma}$, где γ — некоторая константа. При $\gamma < 1$ переходная характеристика

а

б



в



г

Рис. 1.4. Автоматическая коррекция яркости и контрастности: *а* — исходное изображение; *б* — скорректированное изображение; *в* — гистограмма яркости исходного изображения; *г* — гистограмма яркости скорректированного изображения

пойдет над отрезком, при $\gamma > 1$ — под отрезком. Такой вид коррекции называется гамма-коррекцией (рис. 1.5).

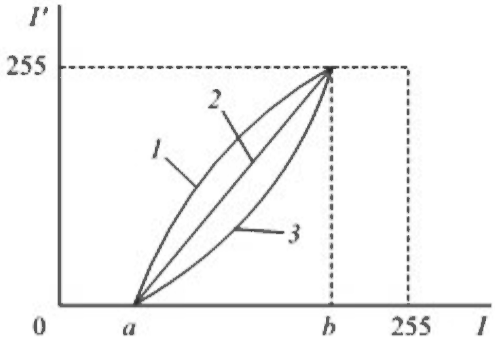


Рис. 1.5. Схема гамма-коррекции: *1* — $\gamma < 1$; *2* — $\gamma = 1$; *3* — $\gamma > 1$

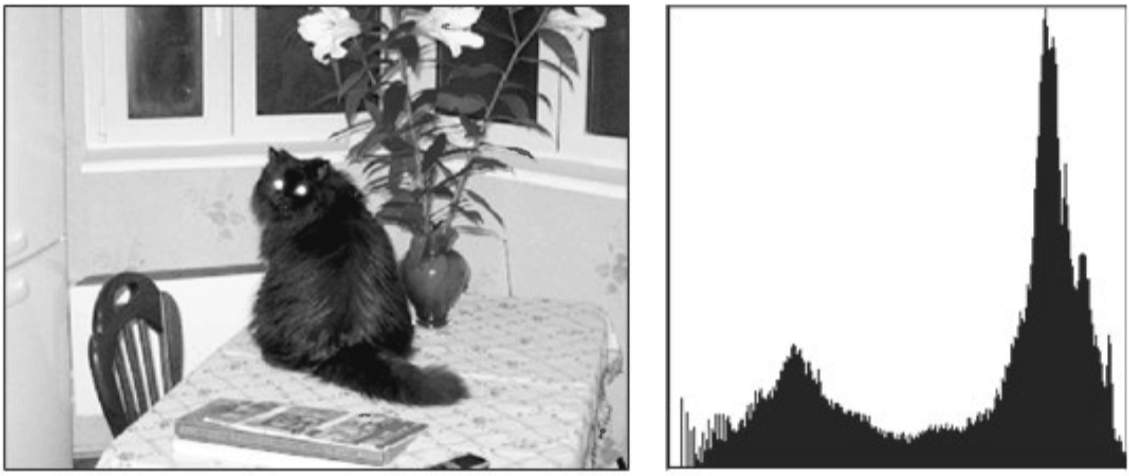
Гамма-коррекция выполняется по формуле

$$I'[x,y] = \left(\frac{I[x,y]-a}{b-a}\right)^{\gamma} \cdot 255,$$

где γ — показатель гамма-коррекции; a — минимальная яркость; b — максимальная яркость.

Как и в предыдущем случае, следует сначала заполнить таблицу пересчета яркости $I' = T[I]$, а затем в соответствии с ней следует скорректировать яркость каждой точки. Таким образом, алгоритм гамма-коррекции работает достаточно быстро, несмотря на наличие операции деления и возведения в степень.

Гамма-коррекция также дает очень хороший визуальный эффект (рис. 1.6) и тоже не привносит в изображение никакой новой информации.



а *б*

Рис. 1.6. Изображение с гамма-коррекцией (*а*) и его гистограмма яркости (*б*)

Цветовая коррекция. На цветном изображении, кроме коррекции яркости и контрастности, появляется задача коррекции соотношения между составляющими его цветами. Примерами такой коррекции служат коррекция насыщенности и балансировка белого.

Коррекция насыщенности. Мы привыкли видеть цветное изображение, состоящее из красной, зеленой и синей компонент в качестве базовых. Такое представление является биологически обусловленным: цветочувствительные клетки человеческого глаза (колбочки) имеют максимумы чувствительности именно в красной, зеленой и синей областях спектра. Однако такое представление не является единственно возможным. Например, в издательском

деле и полиграфии используются иные базовые цвета — бирюзовый, пурпурный, желтый и черный, смесь которых также позволяет получить любой из воспринимаемых человеческим глазом оттенков.

Рассмотрим пространство цветов *RGB*. Направим ось зеленого влево вверх и на нас, ось красного — вправо, на нас, ось синего — влево вниз и на нас (рис. 1.7). Тогда луч, исходящий из черного цвета (0, 0, 0) через белый (255, 255, 255), будет смотреть прямо на нас.

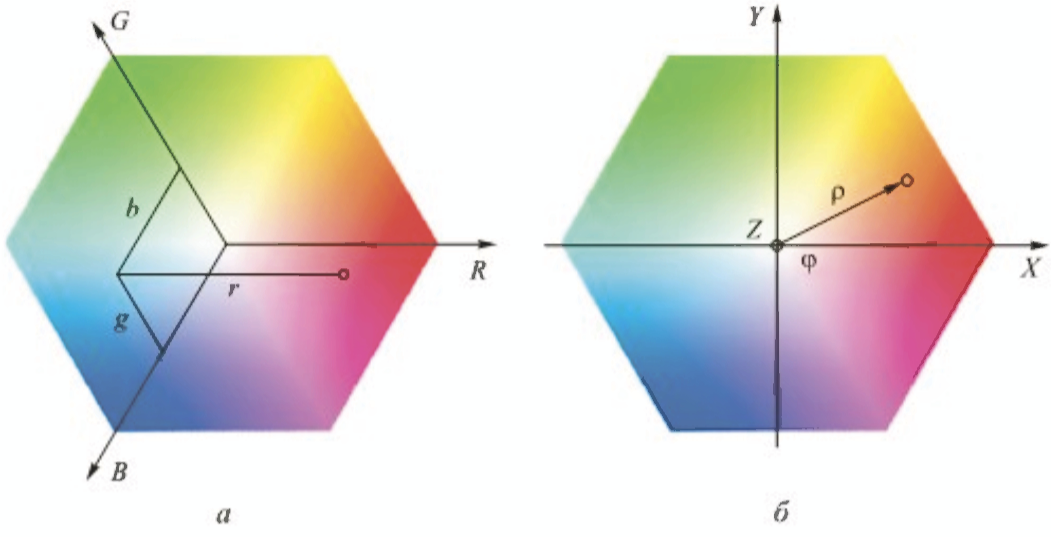


Рис. 1.7. Пространства цветов *RGB* (*а*) и *XYZ* (*б*)

Направим теперь ось *X* слева направо через черную точку, ось *Y* — снизу вверх через черную точку, ось *Z* — на нас через черную точку.

Координаты *X*, *Y*, *Z* связаны с координатами *R*, *G*, *B* следующим образом:

$$X = (2R - G - B)/2;$$

$$Y = (G - B) \cdot \sqrt{3}/2;$$

$$Z = (R + G + B)/3.$$

В обратную сторону:

$$R = Z + 2X/3;$$

$$G = Z - X/3 + Y/\sqrt{3};$$

$$B = Z - X/3 - Y/\sqrt{3}.$$

Ось *Z* направлена от черной точки к белой, проходя через все оттенки серого, это — ось яркости. Для того чтобы определить оси *X*, *Y*, переведем координаты (*x*, *y*) в полярную форму (ρ, φ):

$$\rho = \sqrt{X^2 + Y^2};$$

$$\varphi = \operatorname{arctg}(Y/X).$$

В обратную сторону:

$$X = \rho \cos \varphi;$$

$$Y = \rho \sin \varphi.$$

Точки, имеющие одинаковые значения φ — точки одного оттенка; например, все точки с $\varphi = 0$ — красные. Кроме того, они могут иметь одинаковую яркость z. Чем же будут различаться эти точки? Они будут различаться степенью выраженности цвета по сравнению с серым. Эта характеристика называется *насыщенностью*. Более насыщенные точки будут располагаться дальше от серого и ближе к краям цветового куба (см. рис. 1.7). Коррекция насыщенности заключается в изменении длины вектора ρ при неизменных φ и z.

Коррекция может быть линейной: $\rho' = k\rho + \rho_0$ или нелинейной: $\rho' = \rho^{\gamma} + \rho_0$.

На практике нелинейная коррекция с $\gamma = 1{,}5...2$ предпочтительнее, поскольку увеличивает насыщенность областей, уже имеющих некоторый явно выраженный цвет, и практически не затрагивает серые области.

Повышение насыщенности может применяться не только для улучшения визуального качества изображения для последующего просмотра оператором, но и для упрощения дальнейшей автоматической обработки, например, если дальше будут использоваться алгоритмы, работающие с цветовыми пятнами.

Пример работы алгоритма нелинейной коррекции насыщенности показан на рис. 1.8.

а *б*

Рис. 1.8. Пример нелинейной коррекции насыщенности:
а — исходное изображение; *б* — скорректированное изображение

Баланс белого. Цвета точек на изображении сильно зависят от типа источника освещения. Так, один и тот же лист бумаги будет выглядеть слегка голубоватым при люминесцентном освещении и желтоватым — при ламповом. Человеческий глаз этого не замечает, но мозг автоматически корректирует видимые цвета в соответствии с его представлениями. Поэтому белый лист будет казаться белым при любом освещении — нам известен его реальный цвет.

В задачах автоматической обработки изображений подобных априорных знаний нет и коррекция цвета проводится алгоритмически. Примером является коррекция цветных фотографий с других планет: поскольку никто не знает, какие на самом деле объекты на поверхности других планет, а чувствительность камер в разных диапазонах может со временем меняться, восстановление действительного цвета объектов представляется весьма сложной задачей.

Каким же образом можно корректировать цвета? Точно таким же, как это делает человек. Вспомним, что известные художники прошлого клали перед холстом черный цилиндр и белый платок — образцы чисто черного и белого цветов, т. е. образцы, цвет которых заведомо известен. Аналогично в кадре автоматической камеры можно поместить настроечный образец, цвет или набор цветов которого заранее известен.

Если задача не подразумевает возможности внесения такого образца, можно попытаться обнаружить объекты с заведомо известным цветом, например потолок в жилом помещении — как правило, он имеет белый цвет. Если же найти такой объект не представляется возможным, то его можно попробовать синтезировать из имеющихся. Простейший способ — предположить, что все цвета изображения должны давать серый цвет. В этом случае эталоном служит средний цвет изображения.

Если выбрать эталонный цвет не представляется возможным, то цветовая коррекция также невозможна.

Поскольку задача цветовой коррекции исторически сводилась к приведению белых образцов на изображении к белому цвету, этот вид цветовой коррекции получил название «балансировка белого». Пример балансировки белого показан на рис. 1.9.

Пусть $C_0 = (R_0, G_0, B_0)$ — наблюдаемый цвет образца, который в реальности имеет белый цвет, тогда соответствующий образцу серый цвет равной яркости равен:

$$I_0 = (R_0 + G_0 + B_0)/3.$$

а *б*

Рис. 1.9. Пример балансировки белого:
а — исходное изображение (получено при освещении лампой накаливания); *б* — скорректированное изображение

Отсюда реальный цвет C_1, соответствующий наблюдаемому цвету C_0 и имеющий равную ему яркость

$$C_1 = (I_0, I_0, I_0).$$

Возьмем линейную модель яркости следующего вида:

$$R' = k_R R; \quad G' = k_G G; \quad B' = k_B B,$$

где R, G, B — цвета исходного изображения; R', G', B' — цвета скорректированного изображения; $k_R = I_0/R_0$, $k_G = I_0/G_0$, $k_B = I_0/B_0$ — постоянные множители.

Нетрудно убедиться, что такое преобразование переводит цвет образца C_0 в равнояркий ему серый цвет C_1.

В пространстве цветов *RGB* указанные преобразования будут приводить к сжатию и растяжению осей координат (рис. 1.10).

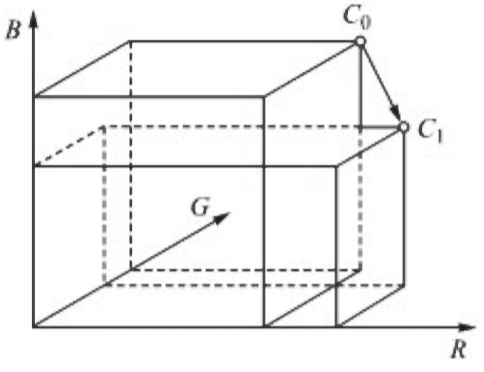


Рис. 1.10. Схема балансировки белого в пространстве *RGB*

Коррекция ракурса

Необходимость в геометрической коррекции изображения возникает в случаях, когда оно было получено не в тех ракурсе, масштабе, ориентации, которые требуются для дальнейшей обработки, а в тех, в которых позволяют это делать особенности решаемой задачи. При этом параметры съемки имеющегося и желаемого изображений зачастую известны. Имея эти данные, можно преобразовать имеющееся изображение так, как оно выглядело бы в желаемых условиях.

Примером подобной ситуации может служить задача распознавания государственных регистрационных номеров автомобиля для контроля соблюдения скоростного режима. В этом случае камеру желательно размещать непосредственно перед номером, поскольку это даст наиболее четкое и наименее искаженное изображение. Однако это технически невозможно, поэтому камеру размещают так, чтобы она не создавала помех движению — на столбах, пролетах мостов и путепроводов, а получаемое изображение корректируют математическими методами (рис. 1.11).

Другой пример — изображение, получаемое с борта беспилотного летательного аппарата (БПЛА). Для создания условий наилучшей видимости и наименьших искажений камера БПЛА должна смотреть вертикально вниз (рис. 1.12). Однако в этом случае оператору не видно, куда он летит, и управление им становится затруднительно. Поэтому камера направлена вниз и вперед, давая оператору лучший обзор, но при этом внося в изображение проективные искажения. При автоматической обработке полученных изображений эти искажения также подлежат коррекции математическими методами.

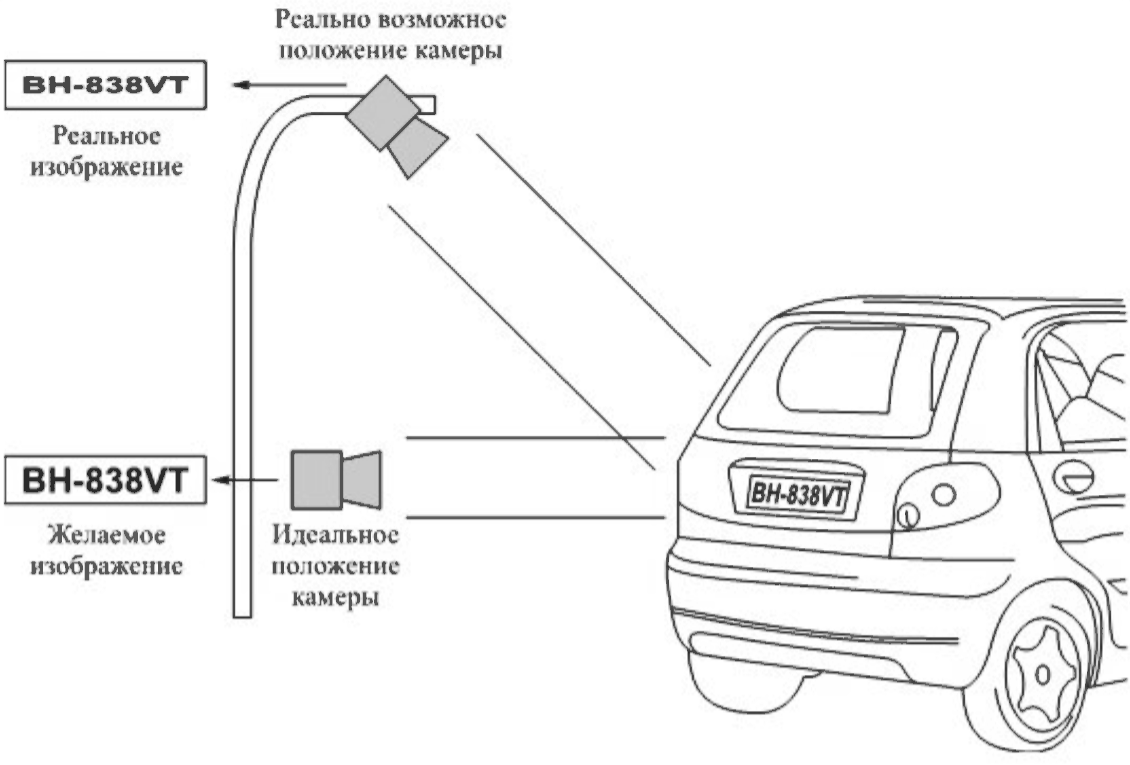


Рис. 1.11. Размещение камеры в задаче распознавания регистрационных номеров автомобиля

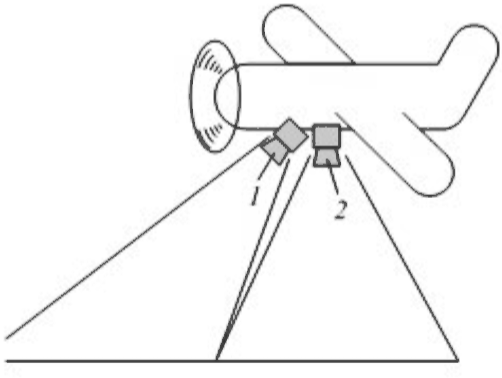


Рис. 1.12. Размещение камеры в задаче обработки изображений с борта БПЛА: *1* — положение камеры, дающее наилучший обзор; *2* — положение камеры, дающее минимальные искажения

Типы и модели геометрических искажений. Рассмотрим типы геометрических искажений.

Параллельный перенос. Искажения в виде параллельного переноса могут возникать, например, при несовпадении центра получаемого изображения с желаемым центром объекта. Преобразование параллельного переноса — это простейший вид преобразований, сводящийся, как правило, к копированию прямоугольного фрагмента исходного изображения. Формулы параллельного переноса имеют вид

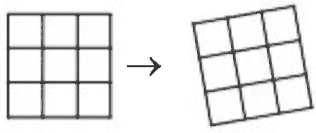
$$x' = x + \Delta x; \quad y' = y + \Delta y,$$

где x, y — исходные координаты; x', y' — координаты точек желаемого изображения на исходном, Δx, Δy — параметры преобразования.

Если Δx, Δy — целые числа, то параллельный перенос сводится к копированию прямоугольного фрагмента исходного изображения. Перенос по координатам x и y может выполняться независимо друг от друга и в любом порядке и образовывать группу преобразований параллельного переноса.

Поворот. Поворот относительно центра координат на угол φ описывается формулами

$$x' = x\cos\varphi + y\sin\varphi,$$
$$y' = x\sin\varphi - y\cos\varphi.$$



При повороте относительно точки, несовпадающей с центром координат, изображение будет дополнительно смещено:

$$x' = x\cos\varphi + y\sin\varphi + x_0 - x_0';$$
$$y' = x\sin\varphi - y\cos\varphi + y_0 - y_0',$$

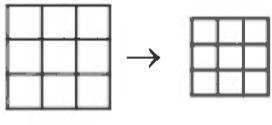
где x_0, y_0 — координаты центра поворота; x_0', y_0' — координаты центра поворота в новой системе координат,

$$x_0' = x_0\cos\varphi + y_0\sin\varphi;$$
$$y_0' = x_0\sin\varphi - y_0\cos\varphi.$$

Следует отметить, что при повороте на угол, не кратный 90°, узлы сетки нового изображения не совпадут с узлами сетки исходного, и поворот не может быть сведен к копированию соответствующих точек. Кроме того, часть узлов сетки попадет в области, отсутствующие на исходном изображении.

Масштабирование. Необходимость масштабирования изображения возникает, если оно снято не в том разрешении или не с того расстояния, которые требуются для дальнейшей обработки. Преобразование масштабирования описывается формулами

$$x' = Mx;$$
$$y' = My,$$



где M — масштаб изображения.

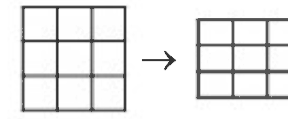
Если M не является целым числом, то узлы сетки исходного и желаемого изображения также не будут совпадать.

Преобразования масштабирования и поворота образуют группу преобразований сдвига. Они, как и параллельный перенос по двум независимым координатам, могут выполняться в любом порядке и независимо друг от друга. Если преобразовать изображение в полярную систему координат, то поворот и масштабирование в исходной системе будут приводить к параллельному переносу по углу и длине радиуса-вектора соответственно, и наоборот. Таким образом, поворот и масштабирование можно рассматривать как аналог параллельного переноса в полярной системе координат.

Преобразования параллельного переноса, поворота и масштабирования образуют группу преобразований подобия, или гомотетии. Эти преобразования также называют жесткими преобразованиями. Для них значения углов геометрических фигур сохраняются, а прямые линии остаются прямыми.

Масштабирование по одной из координат. Масштабирование по одной из координат описывается формулами

$$x' = M_x x;$$
$$y' = M_y y,$$

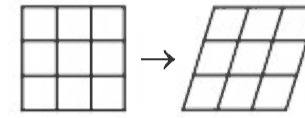


где M_x, M_y — масштаб изображения по оси x и y соответственно.

Если масштабы по осям не совпадают, то преобразование не будет являться жестким и не будет сохранять углы фигур. Однако преобразование останется линейным.

Косой сдвиг. Косой сдвиг описывается формулами

$$x' = x + py;$$
$$y' = y + qx,$$



где p, q — параметры косого сдвига.

Косой сдвиг может возникать в случае, если скорость движения объектов изображения сопоставима со скоростью получения строк изображения (скорость выражена в числе точек изображения в секунду). Необходимо отметить, что на момент написания данного пособия подавляющее большинство камер получают изображение именно построчно, а полнокадровые камеры, хотя и известны, существуют в основном в виде экспериментальных установок и слишком дорогостоящи для широкого практического применения.

При построчном получении изображения разные строки относятся к разным моментам времени. Если за это время объект смещается на расстояние, сопоставимое с размером точки, то его изображение окажется косо сдвинутым. Если направление движения объекта известно заранее (например, в задачах автоматизации конвейеров и прокатных станов), то решением задачи может быть разворот камеры таким образом, чтобы объект двигался не вдоль строк, а поперек. Другой вариант — использование более быстродействующей камеры с меньшим временем экспозиции (однако уменьшение времени экспозиции одновременно снижает и чувствительность). Если эти меры оказываются неэффективными, то изображение необходимо корректировать математическими методами.

Параллельный перенос, поворот, масштабирование и косой сдвиг — это линейные преобразования, и вместе они образуют группу ***аффинных преобразований***. Аффинное преобразование в общем виде можно описать формулами

$$x' = k_1 x + k_2 y + k_3;$$
$$y' = k_4 x + k_5 y + k_6,$$

где k_1, ..., k_6 — коэффициенты аффинного преобразования.

Более строго, преобразование T называется линейным, если для любых координат x_1, x_2 и для любого числа c выполняются следующие условия:

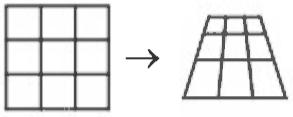
$$T(x_1 + x_2) = T(x_1) + T(x_2); \quad T(cx) = cT(x).$$

Преобразование T является аффинным, если преобразование $T(x) \to T(0)$ также линейно. Аффинные преобразования линейны в том смысле, что они преобразуют прямые линии также в прямые.

Нелинейные искажения. Большое количество искажений, с которыми приходится иметь дело на практике, несводимы к линейным преобразованиям. Такие искажения могут возникать из-за перспективы, свойств объектива, непараллельности фокальной плоскости и плоскости сцены, а также из-за того, что сцена неплоская. Например, существенно нелинейными будут искажения изображения в приведенной задаче получения изображения с борта низковысотного БПЛА (см. рис. 1.12).

Проективные искажения. Проективное искажение возникает, когда фокальная плоскость камеры не параллельна плоскости сцены (сцену будем считать плоской). В случае когда фокальная плоскость параллельна оси сцены x и составляет с осью y угол φ, а сама камера находится на расстоянии H от плоскости сцены, проективное преобразование можно описать формулами

$$x' = xH\cos\varphi/(H\cos\varphi + y\sin\varphi);$$
$$y' = yH/(H\cos\varphi + y\sin\varphi).$$



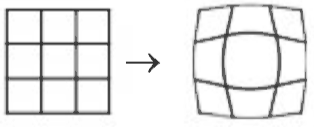
Проективные искажения имеют место, например, при съемке дорожных сцен с камеры в автомобиле или при съемке сцены камерой мобильного робота в задачах автономной навигации.

Перспективные искажения. Перспективные искажения являются более общим случаем проективных искажений для ситуаций, в которых сцену нельзя считать плоской. Для моделирования и коррекции перспективных искажений можно использовать полиномиальные преобразования:

$$x' = (k_1 x + k_2 y + k_3)/(k_4 x + k_5 y + k_6);$$
$$y' = (k_7 x + k_8 y + k_9)/(k_{10} x + k_{11} y + k_{12}),$$

где k_1, ..., k_{12} — коэффициенты преобразования.

Дисторсия. Дисторсия может возникать из-за того, что расстояние от камеры до различных точек сцены существенно различается, так что сцену уже нельзя считать плоской. Типичный пример дисторсии — съемка панорамы широкоугольной камерой. При этом изображение принимает характерный подушкообразный вид.



Локальные искажения. Искажения называют локальными, если не существует общей (глобальной) формулы, подходящей для коррекции всех областей изображения. Примерами могут служить искажения медицинских изображений тканей и органов вследствие их движения или изгиба; изображений, полученных в результате маловысотной аэрофотосъемки: если высота съемки сопоставима с перепадами рельефа, то эти перепады будут вызывать локальные искажения, присущие области с данным профилем.

Один из возможных подходов к коррекции локальных искажений связан с разбиением изображения на подобласти, в пределах которых искажения можно считать глобальными. Далее строят модели глобальных искажений для каждой подобласти, а затем сшивают полученные модели по границам областей.

Методы коррекции геометрических искажений

Для определенности будем считать, что имеется изображение, полученное в одном масштабе, и требуется получить из него изображение в другом масштабе (тип преобразования может быть любым). Будем считать также, что масштабы в общем случае не являются кратными. Тогда узлы координатной сетки нового изображения не будут совпадать с узлами исходного (рис. 1.13).

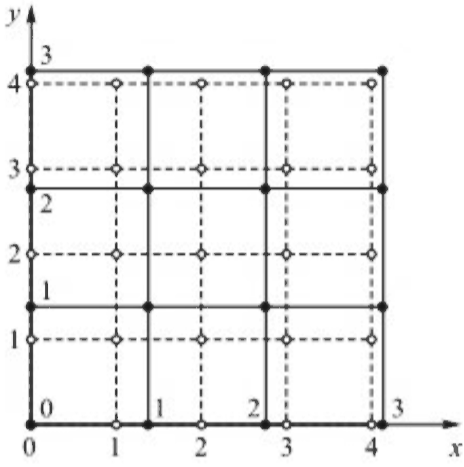


Рис. 1.13. Схема расположения узлов исходной и новой сетки на изображении:
∘ — узлы старой сетки; • — узлы новой сетки

Поскольку изображение является цифровым сигналом и яркости точек известны лишь в узлах координатной сетки, определять яркость точек в узлах новой сетки можно лишь путем их аппроксимации по яркости известных точек. Аппроксимация может быть, например, линейной или полиномиальной. При аппроксимации будем руководствоваться принципом локальности: поскольку яркость точки изображения в большей мере зависит от соседних точек и в меньшей — от дальних (которые могут принадлежать совсем другим объектам), то для аппроксимации следует использовать лишь некоторое количество ближайших точек.

Метод ближайшего соседа. Простейший вид аппроксимации — копирование ближайшей соседней точки. Этим методом можно пользоваться, когда необходима максимальная скорость обработки (в ущерб качеству изображения). У такого метода есть очевидные недостатки. Так, при увеличении новое изображение будет выглядеть состоящим из «квадратиков» (прямоугольных областей точек одинакового цвета). Это сделает невозможным такие операции, как определение направления точки контура, поскольку контур в таком случае пойдет по границе квадрата строго вертикально или горизонтально. При уменьшении, новое изображение будет копировать шумовые точки, повышая общий уровень шума на изображении.

Линейная аппроксимация. Лучший результат позволяет получить двумерная линейная аппроксимация по нескольким соседним точкам. Линейную аппроксимационную поверхность можно задать формулой

$$I(x, y) = ax + by + c,$$

где $I(x, y)$ — искомая яркость точки в положении (x, y); a, b, c — коэффициенты аппроксимации, подлежащие определению.

Поскольку коэффициентов три, для их определения потребуется три линейно независимых уравнения относительно трех ближайших точек исходного изображения. В простейшем случае целесообразно взять фиксированные точки, например слева снизу, слева сверху и справа снизу от точки (x', y'). Координаты первой точки получают округлением x' и y' до целого,

координаты остальных — добавлением единицы к координате x' и y' соответственно.

Рассмотрим методику расчета коэффициентов аппроксимации. Сам расчет несложен, однако методика потребуется далее для более сложных аппроксимаций.

Пусть (x', y') — координаты точки, яркость в которой надо определить (это некоторое дробное рациональное число). Точки A, B, C выбраны в качестве ближайших, как показано на рис. 1.14, с координатами

$$x_A = \text{int}\,(x'), \quad y_A = \text{int}\,(y');$$
$$x_B = x_A + 1, \quad y_B = y_A;$$
$$x_C = x_A, \quad y_C = y_A + 1,$$

где int ( ) — операция выделения целой части.

Для упрощения вычислений перейдем к смещенной системе координат $(\Delta X, \Delta Y)$, центром которой выберем точку A. Очевидно, что

$$\Delta x = x - x_A;$$
$$\Delta y = y - y_B.$$

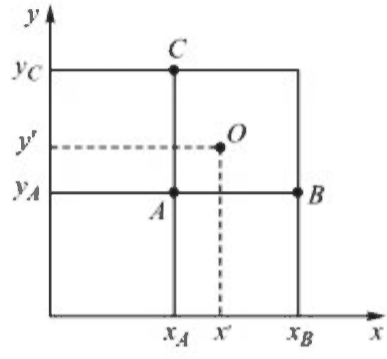


Рис. 1.14. Выбор опорных точек билинейной аппроксимации

Запишем уравнения аппроксимации для точек A, B, C в этой системе координат. Потребуем, чтобы аппроксимирующая поверхность проходила в точках A, B, C через значение их яркости, и получим

$$A: \; I_A = a \cdot 0 + b \cdot 0 + c;$$
$$B: \; I_B = a \cdot 0 + b \cdot 1 + c;$$
$$C: \; I_C = a \cdot 1 + b \cdot 0 + c,$$

где I_A, I_B, I_C — яркости в точках A, B, C.

Из первого уравнения находим $c = I_A$. Подставив его во второе и третье равенства, соответственно получим $b = I_B - I_A$, $a = I_C - I_A$. Таким образом,

$$a = I_C - I_A;$$
$$b = I_B - I_A;$$
$$c = I_A.$$

Выражение для аппроксимации яркости в точке (x', y') будет иметь вид

$$I(x', y') = a(x' - x_A) + b(y' - y_A) + c.$$

Очевидно, что для вычисления яркости в каждой точке изображения потребуются две операции умножения.

Качество линейной аппроксимации относительно невысокое, и при увеличении изображения она также приводит к образованию «квадратиков» из точек близкого цвета. Это связано с тем, что при переходе через линию координатной сетки значения коэффициентов аппроксимации изменяются скачком, поскольку меняются две из трех опорных точек. Для устранения этого недостатка необходимо строить аппроксимации по большему числу точек.

Квадратическая аппроксимация. Такая аппроксимация описывается двумерным полиномом второй степени:

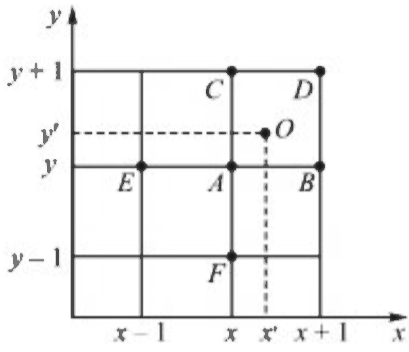


Рис. 1.15. Выбор опорных точек квадратической аппроксимации

$$I(x, y) = ax^2 + by^2 + 2cxy + dx + ey + f,$$

где a, b, c, d, e, f — коэффициенты аппроксимации.

Для определения шести коэффициентов аппроксимации потребуется шесть уравнений. Точки можно выбирать различным образом, но так, чтобы уравнения были линейно независимы. Выберем точки, как показано на рис. 1.15.

Аналогично предыдущему случаю, для упрощения вычислений воспользуемся смещенной системой координат с центром в точке A. Тогда можно записать следующие уравнения:

$$A\ (\Delta x = 0, \Delta y = 0)\colon I_A = a\cdot 0 + b\cdot 0 + 2c\cdot 0 + d\cdot 0 + e\cdot 0 + f;$$
$$B\ (\Delta x = 0, \Delta y = 0)\colon I_B = a\cdot 1 + b\cdot 0 + 2c\cdot 0 + d\cdot 1 + e\cdot 0 + f;$$
$$E\ (\Delta x = 0, \Delta y = 0)\colon I_E = a\cdot 1 + b\cdot 0 + 2c\cdot 0 - d\cdot 1 + e\cdot 0 + f;$$
$$C\ (\Delta x = 0, \Delta y = 0)\colon I_C = a\cdot 0 + b\cdot 1 + 2c\cdot 0 + d\cdot 0 + e\cdot 1 + f;$$
$$F\ (\Delta x = 0, \Delta y = 0)\colon I_F = a\cdot 0 + b\cdot 1 + 2c\cdot 0 + d\cdot 0 - e\cdot 1 + f;$$
$$D\ (\Delta x = 0, \Delta y = 0)\colon I_D = a\cdot 1 + b\cdot 1 + 2c\cdot 1 + d\cdot 1 + e\cdot 1 + f.$$

Из первого уравнения находим

$$f = I_A,$$

из второго и третьего уравнений —

$$2a + 2f = I_B + I_E \Rightarrow a = (I_B + I_E - 2I_A)/2;$$
$$2d = I_B - I_E \Rightarrow b = (I_B - I_E)/2;$$

из четвертого и пятого

$$2b + 2f = I_C + I_F \Rightarrow a = (I_B + I_E - 2I_A)/2;$$
$$2e = I_C - I_F \Rightarrow b = (I_C - I_F)/2.$$

Подставив результаты в последнее уравнение, получим

$$c = (I_A + I_D - I_C - I_F)/2.$$

Из данных уравнений видно, что для вычисления коэффициентов аппроксимации a, ..., f не требуется операций умножения, а деление на два легко заменяется операцией побитового сдвига. Итоговое уравнение будет иметь вид

$$I(x', y') = a(x' - x_A)^2 + b(y' - y_A)^2 + 2c(x' - x_A)(y' - y_A) +$$
$$+ d(x' - x_A) + e(y' - y_A) + f,$$

где $(x' - x_A)$ и $(y' - y_A)$ — дробные части x' и y' соответственно.

Для вычисления одного значения яркости потребуется восемь операций умножения, что в 4 раза больше, чем в алгоритме линейной аппроксимации. Как следствие, этот алгоритм в 4 раза медленнее, однако качество аппроксимации выше: при переходе через линию сетки в любом направлении меняется только половина опорных точек.

Аналогично строят алгоритмы и более высокого порядка, позволяющие получить более высокое качество аппроксимации, но требующие существенно больше вычислений. Так, бикубическая аппроксимация строится по 10 точкам и требует 24 операций умножения для расчета яркости в каждой точке.

Билинейная аппроксимация. Можно строить аппроксимацию по произвольному числу точек, например по четырем или пяти. Рассмотрим аппроксимацию по четырем точкам, окружающим анализируемую точку (рис. 1.16): $A(x, y)$, $B(x + 1, y)$, $C(x, y + 1)$ и $D(x + 1, y + 1)$.

Построим вспомогательные точки $E(x', y)$ и $F(x', y + 1)$. Точка E лежит на отрезке AB, а точка F — на отрезке CD. Это позволяет найти яркость в точках E, F с помощью одномерной линейной аппроксимации по известным яркостям в точках A, B и C, D соответственно (рис. 1.17).

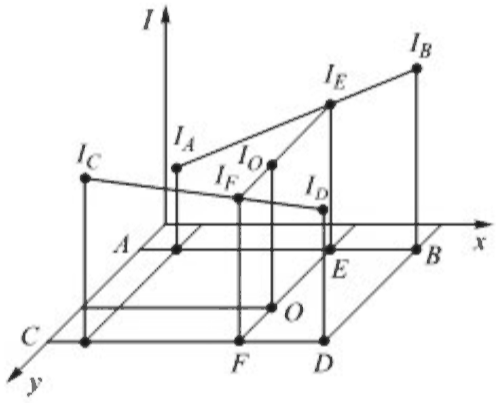


Рис. 1.16. Схема билинейной аппроксимации

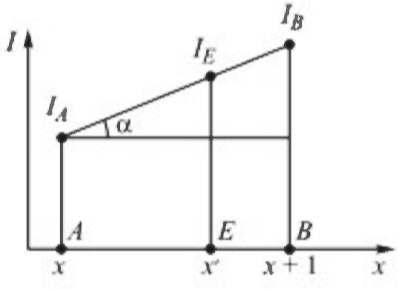


Рис. 1.17. Схема одномерной линейной аппроксимации яркости в точке E

Из рис. 1.17 видно, что

$$\operatorname{tg}\alpha = (I_E - I_A)/(x' - x) = (I_B - I_A)/1,$$

откуда

$$I_E = (I_B - I_A)(x' - x) + I_A.$$

Аналогично для точки F

$$I_F = (I_D - I_C)(x' - x) + I_C.$$

Точки O, E и F лежат на одной прямой, и для получения яркости в точке O можно также воспользоваться одномерной линейной аппроксимацией по яркостям точек E и F:

$$I_O = (I_F - I_E)(y' - y) + I_E.$$

Таким образом, получена модифицированная линейная аппроксимация. Она требует трех операций умножения вместо двух, как в алгоритме билинейной аппроксимации, но строится уже по четырем опорным точкам вместо трех и обеспечивает лучшее качество аппроксимации. Теперь при переходе через линию сетки поменяется лишь половина опорных точек, как в алгоритме биквадратической аппроксимации.

Сравнение линейной и квадратической аппроксимации. Пример реализации линейной и квадратической аппроксимации показан на рис. 1.18, из которого видно, что полученные изображения мало различаются визуально, но при их дальнейшей обработке различие в качестве алгоритма становится весьма существенным.

Особенности программной реализации алгоритмов геометрической коррекции. Координаты узлов новой сетки x' и y', очевидно, не являются целыми числами, и для использования потребуется их представление и соответствующие вычисления в формате с плавающей точкой. Это неудобно, поскольку вычисления с плавающей точкой выполняются медленнее целочисленных, поэтому формулы аппроксимации следует привести к виду, пригодному для целочисленных вычислений (вычислений с фиксированной точкой).

Представим x' и y' в виде приближенной дроби:

$$x' \approx x''/2^N;$$
$$y' \approx y''/2^N.$$

Здесь x'' и y'' — целые числа, а N выбирается таким, чтобы выдержать заданную точность, не выходя при этом за разрядную сетку. Обычно $N = 10...16$.

Тогда

$$x'' = \text{int}\,(2^N x'),$$
$$y'' = \text{int}\,(2^N y'),$$

где int ( ) — операция выделения целой части.

Формулу двумерной линейной аппроксимации по нескольким точкам можно записать в следующем виде:

$$I(x', y') = ax' + by' + c,$$

отсюда

$$I(x'', y'') = (ax'' + by'')/2^N + c.$$

Операцию целочисленного деления на 2^N можно заменить операцией побитового арифметического сдвига вправо на показатель степени. Тогда

$$I(x'', y'') = (ax'' + by'')\ \text{SAR}\ N + c,$$

где SAR N — операция арифметического побитового сдвига вправо на N разрядов (Shift Ariphmetic Right, SAR).

Сдвиг — это короткая целочисленная операция. *Арифметическим* называется сдвиг, при котором старший разряд интерпретируется как знаковый и при сдвиге не изменяется (в отличие от логических сдвигов). Использование

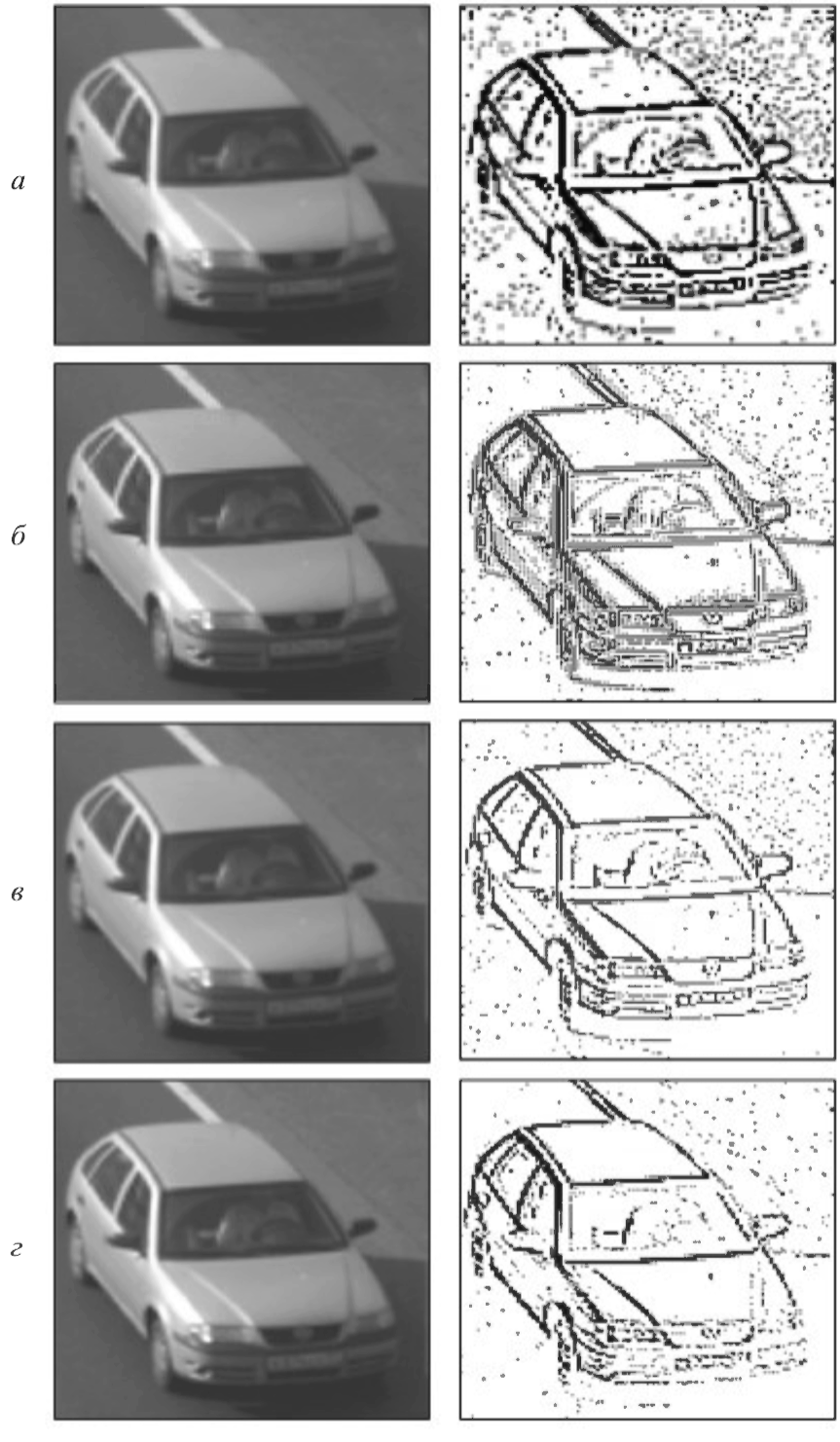


Рис. 1.18. Пример реализации линейной и квадратической аппроксимации при увеличении изображения в 2 раза (слева представлено полученное изображение, справа — его контур):
а — исходное изображение; *б* — увеличение методом соседней точки; *в* — линейная аппроксимация; *г* — квадратическая аппроксимация

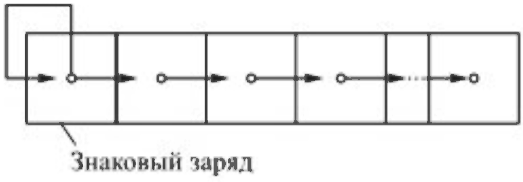


Рис. 1.19. Схема арифметического сдвига вправо

арифметического сдвига (рис. 1.19) необходимо для корректной обработки отрицательных чисел, поскольку коэффициенты аппроксимации a и b могут быть как положительными, так и отрицательными. Следует учитывать, что операторы >> и << языка C++, а также соответствующие им операторы shr и shl языка Паскаль являются именно логическими сдвигами, а не арифметическими и для подобной цели не подходят.

Аналогично преобразуют и формулы для более сложных видов аппроксимации, например для квадратической:

$$I(x', y') = ax'^2 + by'^2 + 2cx'y' + dx' + ey' + f \Rightarrow$$

$$\Rightarrow I(x'', y'') = (ax'^2 + by'^2 + 2cx'y' + 2^N dx' + 2^N ey')/2^{2N} + f \Rightarrow$$

$$\Rightarrow I(x'', y'') = (ax'^2 + by'^2 + 2cx'y' + dx' \text{ SAL } N + ey' \text{ SAL } N) \text{ SAR } 2N + f,$$

где SAL N — операция побитового арифметического сдвига влево.

Следует обратить внимание, что операция деления на 2^N должна выполняться последней (не считая окончательного добавления f), поскольку она наиболее сильно снижает точность вычислений с фиксированной точкой. Также важно убедиться, что максимальное значение первых трех слагаемых $ax'^2 + by'^2 + 2cx'y'$ не выходят за разрядную сетку используемого представления, поскольку там уже содержатся по две операции умножения вместо одной, как в предыдущем случае. При необходимости следует либо уменьшить N, снизив точность вычислений, либо воспользоваться «длинной» арифметикой, представляя x'' и y'' двойными словами (достаточно сделать это только для указанных слагаемых), либо вернуться к арифметике с плавающей точкой, если имеется такая возможность.

Особенности реализации циклов перебора узлов разрядной сетки. Для упрощения будем считать, что над изображением выполняется линейное геометрическое преобразование. Пусть x и y — целочисленные счетчики циклов по узлам новой координатной сетки, изменяющиеся от нуля до новых значений ширины минус один и высоты минус один соответственно. Линейное геометрическое преобразование координат можно представить в виде

$$x' = k_1 x + k_2 y + k_3;$$
$$y' = k_4 x + k_5 y + k_6,$$

где $k_1, \ldots, k_6$ — некоторые константы, определяющие тип и параметры преобразования. Например, при повороте на угол φ

$$k_1 = \sin\varphi; \quad k_2 = \cos\varphi; \quad k_3 = 0; \quad k_4 = \cos\varphi; \quad k_5 = -\sin\varphi; \quad k_6 = 0.$$

Очевидно, что в общем случае значения $k_1, \ldots, k_6$ не являются целыми числами. Но их также можно приближенно представить в виде числа с фиксированной точкой:

$$k_i \approx k_i''/2^K, \quad i = 1, \ldots, 6,$$

где k_i'' — целое число, а K выбирается исходя из требуемой точности представления k_i и из имеющейся разрядной сетки.

Тогда координаты новых узлов сетки x' и y' в старой системе координат будут

$$x' = (k_1''x + k_2''y + k_3)/2^K;$$
$$y' = (k_4''x + k_5''y + k_6)/2^K.$$

Учитывая, что $x'' = \text{int}\,(x' \cdot 2^N)$ и $y'' = \text{int}\,(y' \cdot 2^N)$, получаем

$$x'' = \text{int}\,(k_1''x + k_2''y + k_3)/2^{K-N},$$
$$y'' = \text{int}\,(k_4''x + k_5''y + k_6)/2^{K-N}.$$

В частном случае при выборе $K = N$ будем иметь достаточно компактную запись:

$$x'' = k_1''x + k_2''y + k_3;$$
$$y'' = k_4''x + k_5''y + k_6.$$

Из этих выражений видно, что для вычисления x'' и y'' используется по две операции умножения, однако соседние значения будут отличаться на фиксированную величину:

$$x''(x + 1, y) = x''(x, y) + k_1;$$
$$x''(x, y + 1) = x''(x, y) + k_4;$$
$$y''(x + 1, y) = y''(x, y) + k_2;$$
$$y''(x, y + 1) = y''(x, y) + k_5,$$

поэтому операцию умножения можно исключить. Для этого необходимо хранить предыдущие значения x'' и y'', а также их значения в начале строки и в цикле вычислять последующие по предыдущим на основе приведенных формул.

В случае $K \neq N$ вычисления аналогичны, но появляется дополнительная операция сдвига.

По такой же методике выполняется разработка алгоритмов и для других геометрических преобразований изображения, не сводящихся к линейным.

Особенности преобразования изображения при масштабировании. Если преобразования изображения сводятся только к масштабированию, то двумерное преобразование можно заменить двумя последовательными одномерными преобразованиями, выполняемыми сначала над строками изображения, а затем над получившимися столбцами (или наоборот). При этом объем вычислений может существенно уменьшиться. Так, при замене модифицированной билинейной аппроксимации на две линейные потребуется две операции умножения вместо трех при том же результате (при масштабировании точка F верхней строки, как это показано на рис. 1.16, совпадает с точкой E нижней строки, поэтому строить аппроксимацию достаточно один раз, а не два). При замене двумерной квадратической аппроксимации двумя одномер-

ными квадратическими число операций умножения снизится с восьми до шести, а число опорных точек возрастет от шести до девяти.

Еще одна особенность масштабирования заключается в том, что при существенном уменьшении размеров изображения с соотношением масштабов более двух возникнет ситуация, когда часть точек исходного изображения не будет участвовать в расчетах. Это означает, что получаемое изображение будет иметь более высокий уровень шума. Чтобы этого избежать и задействовать в расчетах все точки, масштабирование следует проводить в два этапа.

1. Изображение масштабируется с ближайшим целым масштабом M_1:

$$M_1 = \text{int}\,(M),$$

где M — требуемый масштаб.

Когда масштаб — целое число, операция масштабирования представляет собой вычисление средней яркости в квадрате со стороной, равной значению масштаба. При вычислении средней яркости уровень шума будет существенно снижен.

2. Полученное промежуточное изображение используется для нахождения окончательного результата с требуемым масштабом. Масштаб на этом этапе

$$M_2 = M/M_1.$$

Пример. Пусть требуется получить изображение в масштабе 1:4,5 (M = 4,5). На 1-м этапе получаем изображение в масштабе M_2 = int (4,5) = 4. Для этого разбиваем площадь исходного изображения на квадраты 4×4 и находим в них средние значения, из которых формируем промежуточное изображение, затем получаем результат. Для этого потребуется масштаб $M_2 = M/M_1 = 4{,}5/4 = 1{,}125$.

Если бы мы здесь воспользовались прямым алгоритмом, без построения промежуточного, в вычислениях участвовала бы только каждая 20-я точка и уровень шума повысился бы в 4,5 раза по сравнению с результатом двухступенчатого алгоритма.

Указанные особенности относятся только к масштабированию изображений, и относить их к другим преобразованиям неправомерно.

Особенности других преобразований. При использовании преобразований, не сводящихся к масштабированию и параллельному переносу, часть узлов новой сетки может попадать в области, информация о которых отсутствует. Эта проблема характерна прежде всего для поворота (рис. 1.20).

Для учета этого обстоятельства применяют один из следующих подходов.

Обрезка изображения. Со всех сторон нового изображения удаляются полосы, полностью вмещающие неинформативные области. Этим методом часто пользуются в предпечатной подготовке любительских фотографий. Обрезка может применяться при сравнительно небольших поворотах — до 10°. При поворотах на больший угол площадь отрезаемых полос быстро растет, и при повороте на 45° превышает 80 %.

Маркировка. Неинформативные области помечаются специальным маркером (например, отрицательным числом), показывая тем самым отсутствие информации о яркости области. Это наиболее универсальный метод,

и именно его следует применять в автоматических системах обработки изображений.

Восстановление изображения. Восстановление данных в неинформативных областях иногда возможно с учетом контекста решаемой задачи, например, если заранее известен цвет фона или если изображение является кадром видеопоследовательности и известно взаимное расположение кадров.

Особенности преобразования цветных изображений. Цветное изображение следует разложить на составляющие цвета, получив набор полутоновых черно-белых изображений, каждое из которых отвечает за яркость своего компонента. Затем проводится преобразование каждой полутоновой компоненты. При этом узлы сетки рассчитываются один раз для всех компонент, а аппроксимация строится для каждой компоненты отдельно. После получения аппроксимации яркости каждой цветовой компоненты из них может быть сформирован результирующий цвет.

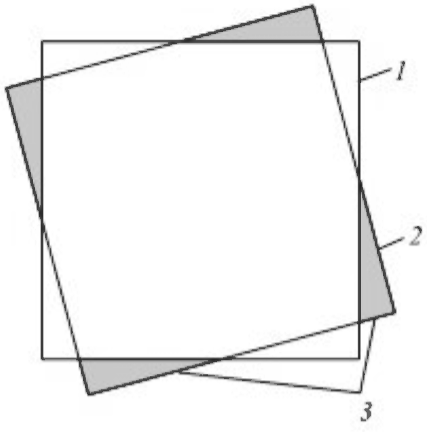


Рис. 1.20. Выход узлов новой сетки за границы изображения:
1 — исходное изображение; *2* — результирующее изображение; *3* — области, в которых отсутствует информация о яркости

Фильтрация изображений

Задача фильтрации — устранить либо ослабить влияние мешающих факторов (шума и помех), либо подчеркнуть полезные признаки. Очевидно, что для этого характер полезных или вредных признаков должен быть известен заранее из контекста решаемой задачи, иначе корректная фильтрация будет невозможна и может приносить больше вреда, чем пользы.

Рассмотрим коротко лишь наиболее распространенные подходы к решению задачи фильтрации[1].

Наиболее общие подходы к фильтрации изображений — это использование ордерных и сверточных фильтров. Ордерный фильтр предполагает использование информации о соседних точках для фильтрации изображения в текущей точке. Самый известный фильтр из этой группы — это медианный фильтр. Сверточные фильтры предполагают выполнение фильтрации данных как результат вычисления свертки изображения с импульсной переходной функцией некоторого двумерного фильтра. Типичный представитель этой группы — фильтр гауссовского сглаживания.

Рассмотрим эти группы фильтров более подробно.

Медианный фильтр. Это, пожалуй, наиболее популярный и широко используемый фильтр с достаточно неплохим шумоподавлением и отсутствием существенных недостатков.

[1] Более полное изложение теории фильтрации можно найти, например, в [1].

Медианный фильтр последовательно применяется к каждой точке изображения, а результат фильтрации заносится в новое изображение, не меняя точек исходного. Каждая текущая точка сравнивается с соседними точками (обычно используется окрестность из восьми точек — 8-окрестность).

Если яркость текущей точки существенно отличается от яркости точек окрестности, то фильтр считает ее точкой шума. Ее яркость должна быть скорректирована: она заменяется на медиану яркостей соседних точек.

Чтобы найти медиану яркостей, они должны быть отсортированы, и тогда медианная яркость будет иметь средний индекс (рис. 1.21). Поскольку размер типовых окрестностей — четный (обычно 8 или 4) и не имеет целого среднего индекса, в последовательность необходимо добавить также и яркость текущей точки.

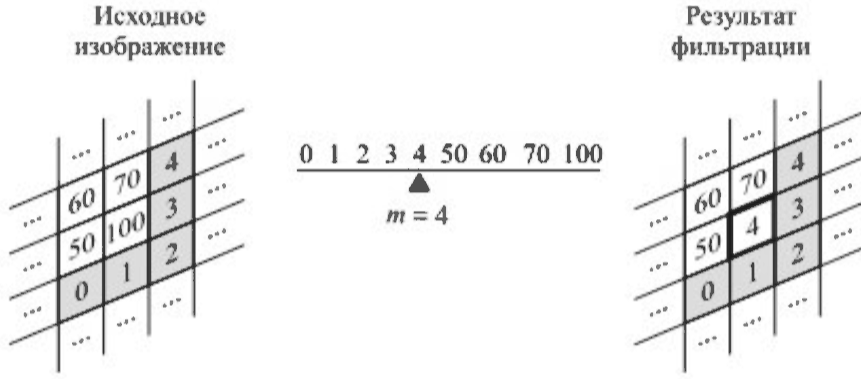


Рис. 1.21. К пояснению принципа медианной фильтрации (m — медиана)

Почему следует брать именно медиану, а не, например, среднюю яркость? Потому что распределение яркостей на изображении, как правило, не является нормальным; характерно именно многомодальное распределение, поскольку цвет фона и объектов значительно различается. Элементов со средней яркостью на изображении может и не быть. Например, для изображения с красным объектом на зеленом фоне средним цветом будет серый. Точки же с медианной яркостью на изображении обязательно присутствуют, и как правило, в значительном количестве.

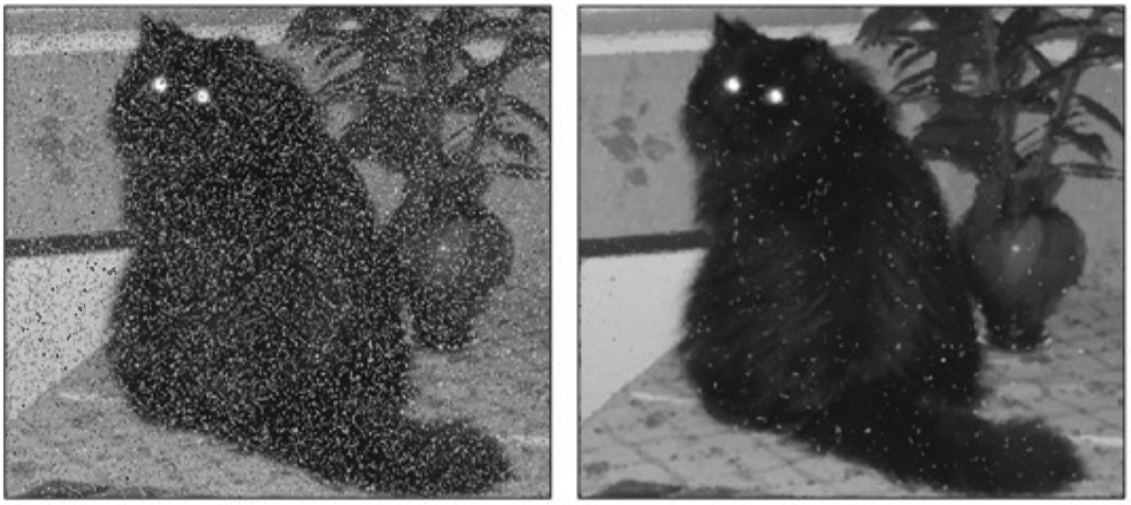
Пример работы медианного фильтра показан на рис. 1.22. Видно, что медианный фильтр весьма эффективен против стохастического шума и позволяет восстанавливать изображение в практически безнадежных случаях. При этом фильтр не затрагивает сами объекты на изображениях.

Вместе с тем, медианный фильтр имеет и существенные недостатки. Он эффективен только против стохастического шума (тепловой шум в сенсорных датчиках); если имеется шум другого характера (структурированная помеха), то медианный фильтр его не уберет (рис. 1.23).

Медианный фильтр позволяет легко находить и убирать отдельные искаженные точки, но не пары, тройки таких точек и т. д. (рис. 1.24).

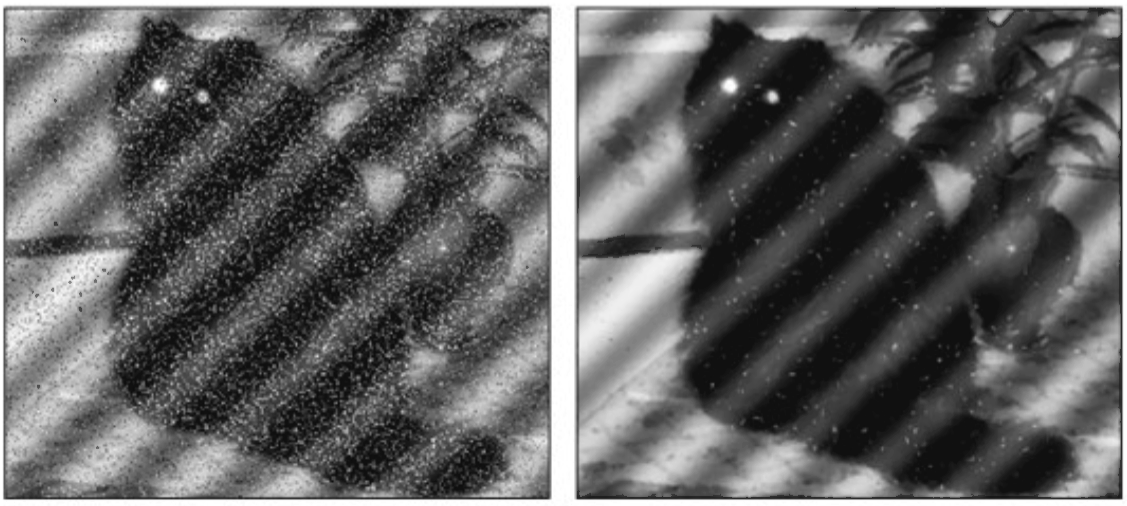
Также характерной особенностью фильтра является подавление острых углов объектов. Дело в том, что угловая точка объекта имеет больше соседей-точек фона, чем точек объекта, и медианой будет именно точка фона. Поэтому если в дальнейшем предполагается обработка угловых точек, то от применения медианного фильтра следует отказаться.

При фильтрации цветных изображений точка признается шумовой, если хотя бы одна из ее цветовых компонент максимальна либо минимальна.



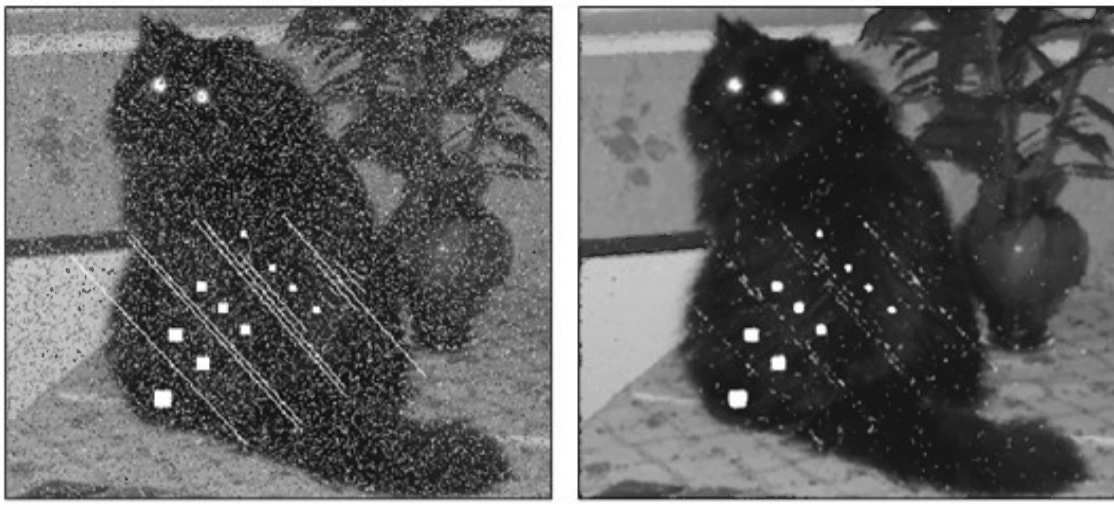
а *б*

Рис. 1.22. Медианная фильтрация:
а — исходное изображение со стохастическим шумом; *б* — результат медианной фильтрации



а *б*

Рис. 1.23. Медианная фильтрация структурированной помехи:
а — исходное изображение со структурированной помехой; *б* — результат медианной фильтрации (фильтр устранил мелкий шум, но не смог устранить помеху)



а *б*

Рис. 1.24. Медианная фильтрация крупных шумовых пятен:
а — иссходное изображение с пятнами и царапинами; *б* — результат медианной фильтрации (фильтр устранил мелкий шум, но не смог устранить крупные пятна; тонкие шумовые линии ослаблены, но полностью не исчезли)

Сверточные фильтры. Сверточный фильтр (также называемый масочным) в общем виде предполагает вычисление интеграла свертки фильтруемого сигнала $S(x)$ и импульсной переходной функции (ИПФ) фильтра $K(x)$:

$$S'(x) = \int_0^T K(\tau) S(x+\tau)\, d\tau.$$

В отличие от фильтрации сигналов, яркость точек изображения является дискретной функцией двух координат (x, y), а не от времени. Для такого двумерного дискретного сигнала интеграл свертки может быть приближенно заменен двойной суммой:

$$I'[x,y] = \sum_{\Delta x=0}^{n_X-1} \sum_{\Delta y=0}^{n_Y-1} K[\Delta x, \Delta y] I[x+\Delta x, y+\Delta y], \quad x = 0, \ldots, N_x - 1,\ y = 0, \ldots, N_y - 1,$$

где n_x, n_y — область значений ИПФ (размер фильтра); N_x, N_y — размеры изображения.

Для точек изображения может быть сформулирован *принцип локальности*: яркость текущей точки изображения связана с яркостями точек своей окрестности малого размера и не зависит от удаленных точек.

Смысл данного принципа достаточно очевиден: соседние точки, как правило, принадлежат одному объекту и имеют близкие характеристики, тогда как удаленные обычно принадлежат разным объектам, и характеристики таких точек не связаны.

Принцип локальности позволяет существенно ограничить размер n ИПФ фильтра небольшой окрестностью текущей точки, что резко снижает объем вычислений и делает возможным прямой расчет свертки.

Для большинства практически значимых фильтров размер их ИПФ составляет 3×3, реже — 5×5. Фильтры большего размера используются крайне редко (например, размер фильтра Гаусса может доходить до 21×21). Фильтр, как правило, имеет нечетный размер, поскольку направления вверх и вниз, вправо и влево на изображении равноценны и фильтр должен быть симметричным. Это свойство также называют *изотропностью*. Для фильтрации временных сигналов изотропность фильтра не требовалась, поскольку само время неизотропно и направлено от прошлого к будущему, тогда как для изображения изотропнность становится важной характеристикой.

Каким же образом работает сверточный фильтр? Подобно медианному сверточный фильтр просматривает (последовательно или параллельно) все точки изображения, и для каждой точки вычисляется отклик фильтра на окрестность этой точки. Для этого яркости точек окрестности умножаются на соответствующий элемент фильтра. Все вычисленные произведения суммируются, умножаются на общий нормирующий множитель, и полученный результат заносится в соответствующую ячейку нового, отфильтрованного изображения (рис. 1.25).

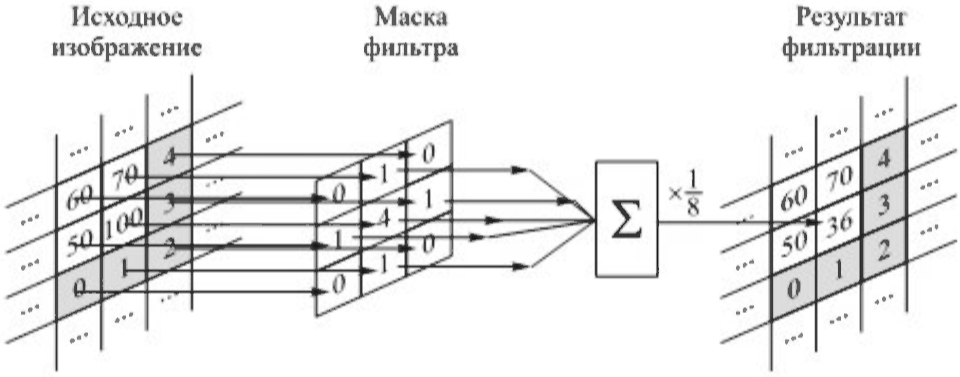


Рис. 1.25. К пояснению принципа работы сверточного фильтра

Следует отметить, что, как и при медианной фильтрации, необходим отдельный массив-приемник результатов фильтрации: если результат фильтрации записать обратно в массив исходного изображения, то данные, необходимые для фильтрации соседних точек, будут потеряны и фильтр будет работать некорректно.

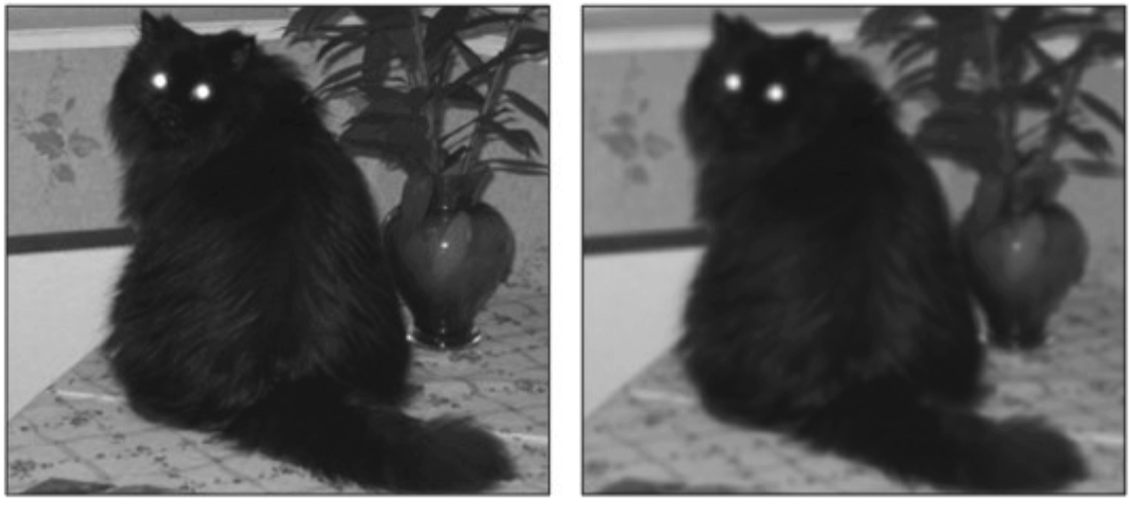
Примеры сверточных фильтров. Простейшим сверточным фильтром, который широко используется в обработке сигналов, является фильтр усреднения по окну, или фильтр размытия (Blur Filter). Его ИПФ представляет собой квадратную матрицу, составленную из единиц. Фильтр размера 3×3 будет иметь следующий вид:

$$Blur = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

Аналогично строятся фильтры большего размера.

Нормирующий множитель 1/9 вводится для того, чтобы сумма элементов матрицы равнялась единице.

Фильтр размытия заменяет яркость каждой точки на среднюю по ее окрестности. В результате изображение размывается и его резкость снижается. Особенно сильно страдают границы объектов (рис. 1.26) (при медианной фильтрации такого не происходит).



а *б*

Рис. 1.26. Применение фильтра размытия:
а — исходное изображение; *б* — результат фильтрации

Видно, что при обработке изображений фильтр размытия дает, скорее, отрицательный результат, поэтому желательно иметь способ построения более полезных фильтров. Для этого вспомним свойства корреляционной функции и сформулируем *принцип оптимальной фильтрации*: нормированная взаимно корреляционная функция *Rxy* сигналов *X* и *Y* является максимальной, если *X* = *Y*. Но сигнал *X* можно рассматривать как фильтруемое изображение, а сигнал *Y* — как ИПФ фильтра. Тогда оптимальным фильтром для выделения какого-либо объекта будет сам объект.

Фильтр Гаусса. Основное назначение фильтра Гаусса — подавление на изображении стохастического (белого) шума. Это может быть, например, тепловой шум в сенсорах фотоматрицы, который становится особенно заметным при слабой освещенности объекта. В основе гауссовской фильтрации лежит предположение, что объекты на изображении носят ярко выраженный низко- и среднечастотный характер, т. е. имеют большие и средние размеры, тогда как шум имеет всечастотный характер.

Импульсная переходная функция фильтра Гаусса представляет собой гауссоиду, связанную с дискретной сеткой изображения (рис. 1.27).

Простейший фильтра Гаусса размером 3×3 имеет следующую форму:

$$Gauss(\sigma) = \frac{1}{4+\sigma}\begin{pmatrix} 0 & 1 & 0 \\ 1 & \sigma & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

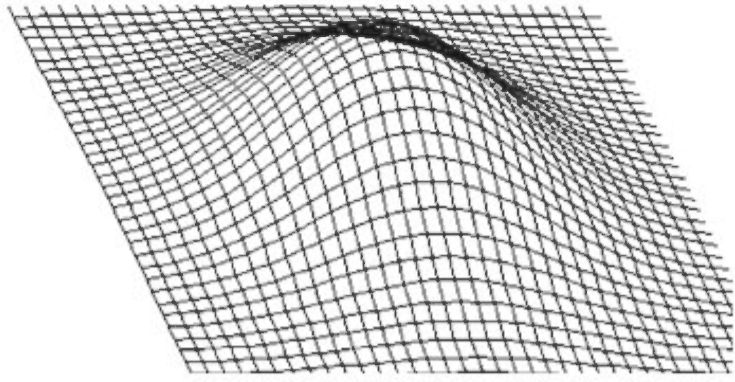


Рис. 1.27. Двумерная гауссоида

Значение σ связано со среднеквадратическим отклонением (СКО) гауссоиды и определяет эффективность фильтра: при низких значениях σ (соответствующих широкой гауссоиде) фильтр становится фильтром размытия и убирает не только шум, но и границы объектов. При высоких σ фильтр превращается в дискретную δ-функцию и изображение практически не меняется.

Аналогично из двумерной гауссоиды формируется фильтр Гаусса большего размера.

Результаты применения фильтра Гаусса с различными значениями σ показаны на рис. 1.28. Видно, что, меняя значение σ, можно достигать боль-

Рис. 1.28. Применение фильтра Гаусса:
а — исходное изображение, содержащее 30 % шумовых точек;
б, *в*, *г* — результаты фильтрации при σ = 8, 4 и 1 соответственно

шего или меньшего шумоподавления с меньшим либо большим размытием объектов на изображении. В целом этот фильтр дает более слабое подавление шума по сравнению с медианным, однако он воздействует на шумовые пятна неединичного размера.

Фильтр границ. Простейший фильтр границ может быть описан парой матриц:

для вертикальных границ

$$E_X = \begin{pmatrix} +1 & -1 \end{pmatrix};$$

для горизонтальных границ

$$E_Y = \begin{pmatrix} +1 \\ -1 \end{pmatrix}.$$

Эти матрицы можно объединить в одну:

$$E_2 = \begin{pmatrix} +2 & -1 \\ -1 & 0 \end{pmatrix}.$$

Легко убедиться, что такой фильтр действительно позволяет выделять границы (рис. 1.29, *б*), однако не является изотропным: он будет смещать реальные точки границы влево и вниз. Тем не менее легко построить и симметричный изотропный фильтр, просуммировав не две, а четыре матрицы:

$$E_3 = \begin{pmatrix} 0 & -1 & 0 \\ -1 & +4 & -1 \\ 0 & -1 & 0 \end{pmatrix}.$$

Результаты выделения границ этими фильтрами показаны на рис. 1.29, *в*. Матрица E_3 очень похожа на матрицу фильтра Гаусса, и далее будет показано, что такое совпадение неслучайно.

Фильтр углов. В соответствии с принципом оптимальной фильтрации оптимальным детектором угла должен быть фильтр с ИПФ в форме угла:

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & +1 & +1 \\ -1 & +1 & +1 \end{pmatrix}.$$

Однако поскольку в таком фильтре присутствуют пять отрицательных элементов и четыре положительных, на однородном изображении фильтр даст ненулевой отклик. Сбалансировать фильтр можно, домножив отрицательные элементы на четыре, а положительные — на пять. Тогда сумма элементов фильтра станет нулевой, и отклик фильтра на однородное изображение также будет нулевым. Получим фильтр следующего вида:

$$C_0 = \frac{1}{20}\begin{pmatrix} -4 & -4 & -4 \\ -4 & +5 & +5 \\ -4 & +5 & +5 \end{pmatrix}.$$

а

б

в

Рис. 1.29. Применение фильтра границ:
а — исходное изображение; *б*, *в* — результаты фильтрации с матрицей E_2 и матрицей E_3 соответственно

Нормирующий множитель 1/20 необходим для того, чтобы максимальный отклик фильтра был равен единице, если максимальная яркость точек изображения также равна единице (если максимальная яркость равна k, то и максимальный отклик фильтра будет равен k). Легко убедиться, что максимальный отклик фильтр C_0 будет давать в левом верхнем углу объекта. Для детектирования других углов, матрицу фильтра следует развернуть на соответствующий угол. Поворот матрицы на угол, кратный 90°, тривиален, а поворот на угол 45° даст следующую матрицу:

$$C_{45} = \frac{1}{20}\begin{pmatrix} -4 & -4 & -4 \\ -4 & +5 & -4 \\ +5 & +5 & +5 \end{pmatrix}.$$

Таким образом, для детектирования прямых углов с шагом ориентации 45° потребуется восемь различных масок. Пример работы детектора углов с маской C_{45} приведен на рис. 1.30.

а *б*

Рис. 1.30. Применение детектора углов с маской C_{45}:
а — исходное изображение; *б* — результат фильтрации

Обратные фильтры. Если известен фильтр, выполняющий какое-либо действие, то на его основе можно построить и обратный фильтр, выполняющий противоположное действие.

Пусть I — исходное изображение, а $F(I)$ — результат его фильтрации. Тогда разность $D = I - F(I)$ позволит получить отфильтрованные элементы, устраненные фильтром F. Теперь можно выполнить противоположное действие, добавляя отфильтрованные элементы к исходному изображению:

$$F^{-1}(I) = I + D = 2I - F(I).$$

Фильтр резкости. В качестве фильтра F рассмотрим фильтр Гаусса, описанный выше. Как было показано, он устраняет мелкие (высокочастотные) детали, в том числе и границы объектов. Именно эти элементы и даст разность $D = I - F(I)$. Теперь можно добавить эти элементы к исходному изображению. В результате границы исходного изображения станут гораздо более четкими (рис. 1.31). Такой фильтр называют фильтром резкости (Sharp-фильтром).

а *б*

Рис. 1.31. Применение фильтра резкости:
а — исходное изображение; *б* — результат фильтрации

а *б*

Рис. 1.32. Негативный эффект повышения резкости:
а — исходное изображение; *б* — результат фильтрации

Следует отметить, что повышение резкости может давать как положительный, так и отрицательный эффект (рис. 1.32), поэтому чрезмерно повышать резкость изображения не следует.

Запишем фильтр резкости в форме масок. Маска разности

$$M_D = \delta - G = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} - \frac{1}{4+\sigma}\begin{pmatrix} 0 & 1 & 0 \\ 1 & \sigma & 1 \\ 0 & 1 & 0 \end{pmatrix} = \frac{1}{4+\sigma}\begin{pmatrix} 0 & -1 & 0 \\ -1 & +4 & -1 \\ 0 & -1 & 0 \end{pmatrix},$$

где $\delta = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ — маска исходного изображения, представляющая собой дискретную δ-функцию.

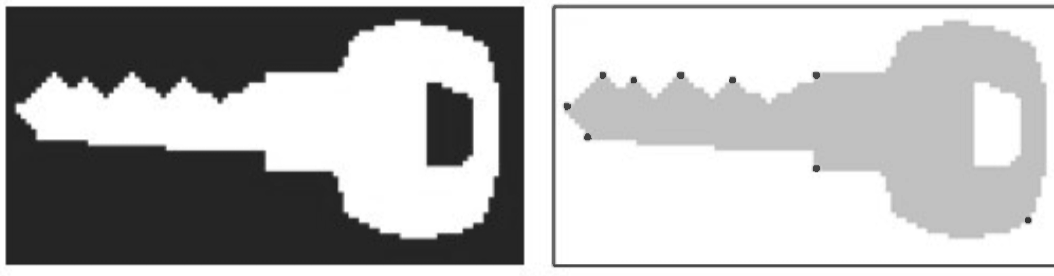
Очевидно, что маска M_D есть не что иное, как ранее рассмотренный фильтр границ E_3 (с точностью до постоянного множителя перед маской). Маска фильтра резкости S будет иметь вид

$$S = \delta + M_D = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \frac{1}{4+\sigma}\begin{pmatrix} 0 & -1 & 0 \\ -1 & +4 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \frac{1}{4+\sigma}\begin{pmatrix} 0 & -1 & 0 \\ -1 & 8+\sigma & -1 \\ 0 & -1 & 0 \end{pmatrix}.$$

Детектор углов на основе медианного фильтра. Разность D и обратный фильтр F^{-1} можно построить даже в тех случаях, когда прямой фильтр F не является сверточным или вообще не может быть описан формулой (в частности, ордерные фильтры, примером которых может служить медианный фильтр). Основным недостатком медианного фильтра является то, что он удаляет с изображения углы объектов. Однако это свойство позволяет ис-

пользовать медианный фильтр в качестве детектора углов. Действительно, разность D исходного и отфильтрованного изображения будет содержать значительные отклики в местах расположения углов.

Результат работы медианного детектора углов показан на рис. 1.33.



а *б*

Рис. 1.33. Применение медианного детектора углов:
а — исходное изображение; *б* — результат фильтрации

В отличие от масочного детектора углов, разностный медианный фильтр позволяет находить углы независимо от их ориентации. Однако кроме углов фильтр будет также выделять мелкие детали и шумовые точки (которые также удаляет медианный фильтр). Поэтому высокочастотный шум должен быть предварительно удален с изображения другими методами, не затрагивающими угловые точки.

Глава 2
БИНАРИЗАЦИЯ И МОРФОЛОГИЧЕСКИЕ АЛГОРИТМЫ

Бинаризация

Под *бинаризацией* понимают разметку изображения, при которой точки объекта получают маркировку 1, а точки фона — маркировку 0.

Очевидно, что для выполнения бинаризации необходимо знать, чем точки объекта отличаются от точек фона. В простейшем случае это можно сделать, сравнивая их яркости. Например, если все точки объекта ярче некоторого заданного порога, а точки фона — наоборот, темнее, то все точки с яркостью выше пороговой помечаются как точки объекта, а ниже пороговой — как точки фона (рис. 2.1). В более общем случае точки объекта могут занимать определенный диапазон яркостей (как светлее, так и темнее точек фона), но в него не должны попадать яркости точек фона.

Помимо яркости можно использовать и другие характеристики точек объекта, например их цвет. Типичным является решение, при котором цвет фона выбирается непересекающимся с цветом объекта (например, неестественно синий или зеленый). Такой фон называется *ключевым*, он часто используется как в задачах технического зрения, так и в иных областях (в частности, в кинематографии для комбинированных съемок). В этом случае точки ключевого цвета идентифицируются как принадлежащие фону, а остальные — объекту.

Другой возможностью размечать точки фона и объекта является использование градиента яркости. Модуль вектора градиента яркости определяет степень гладкости объекта, а его направление — преобладающее направление текстуры. В более общем случае можно использовать и другие текстурные признаки, такие как шаг текстуры, ее энергия, энтропия, направленность, преобладающие гармоники и т. д.

Возможно комбинированное использование нескольких признаков, например, путем задания диапазонов допустимых значений каждого из параметров. В этом случае можно применить бинаризацию, даже если каждый признак по отдельности не позволяет это сделать.

Если же точки объекта не имеют признаков, по которым их можно отличить от точек фона (например, если характеристики фона заранее неизвестны), то бинаризация изображения невозможна и следует использовать иные методы.

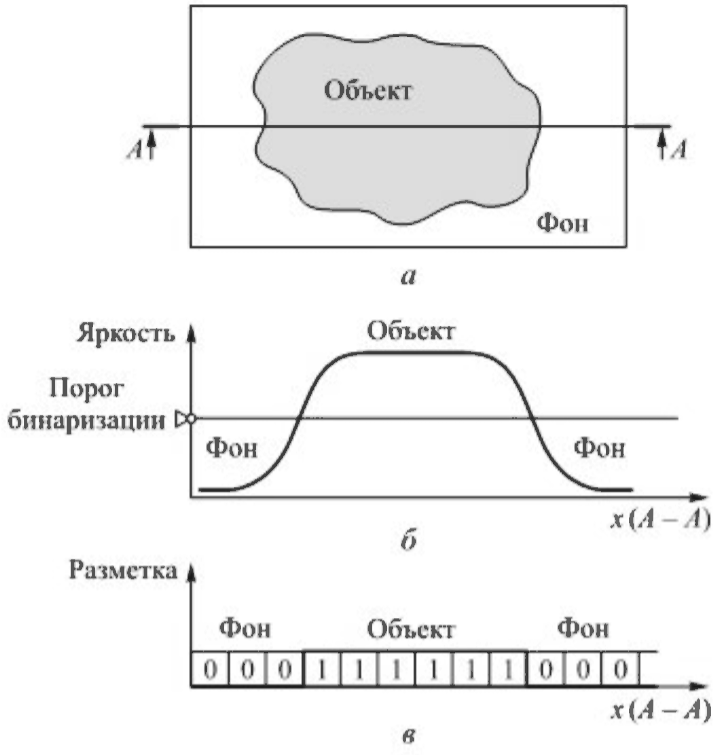


Рис. 2.1. Принцип бинаризации изображения по яркости:
а — исходный объект; *б* — яркость объекта в сечении *А–А*; *в* — бинарная разметка объекта

Далее для упрощения изложения будем считать, что точки объекта и фона различаются по яркости и объект ярче фона; при этом в случае необходимости следует задействовать и иные диапазоны значений и характеристики.

Бинаризация со статическим порогом. В задачах с контролируемыми условиями освещенности (например, в большинстве задач технического зрения) настройка порога бинаризации может быть выполнена по тестовому изображению, и дальше выбранный порог не меняется (см. рис. 2.1)

Определить желаемый порог бинаризации можно вручную, подбирая значение, обеспечивающее наилучшее качество обработки тестовой выборки изображений, или по гистограмме яркостей путем выбора порога, который отделит моды объекта от мод фона.

Следует отметить, что небольшое изменение порога бинаризации может приводить к весьма существенному изменению результата (рис. 2.2).

Динамическая бинаризация. Если условия освещенности в процессе работы системы распознавания меняются (либо заранее неизвестны), порог бинаризации также необходимо подстраивать под новые условия. Для этого порог бинаризации привязывается к какой-либо характеристике, нечувствительной к условиям освещенности.

Рис. 2.2. Бинаризованные изображения с разными порогами

Привязка порога к средней яркости. Простейший способ динамической бинаризации — привязать порог к средней яркости изображения через постоянный множитель. Множитель вычисляется для тестовой выборки и среднестатистических условий освещенности. Тогда с увеличением средней яркости вследствие изменения освещенности порог бинаризации также будет возрастать, а с уменьшением — снижаться.

Этот метод эффективен, если размеры объекта не изменяются во времени. Если размеры объектов и их цвет изменяются, то средняя яркость (а вместе с ней и порог бинаризации) могут меняться и при неизменном освещении. Также метод не будет работать, если яркость точек объекта и фона зависит от освещенности нелинейно, например если объект дает блики или отбрасывает тени.

Привязка порога к числу точек объекта. Если размер объекта примерно известен и, соответственно, можно подсчитать число точек, принадлежащих объекту, а яркость точек объекта выше яркости точек фона, то можно определить порог таким образом, чтобы яркости всех точек объекта всегда оказывались выше этого порога.

Проще всего найти порог по гистограмме яркости. Для этого необходимо суммировать моды гистограммы справа налево, пока сумма не достигнет заданного числа точек объекта, а номер моды, на которой это произошло, выбрать в качестве искомого значения порога (рис. 2.3).

Рис. 2.3. Выбор порога бинаризации по известным размерам объекта

Выбор порога по бимодальной гистограмме яркости. Если известно, что гистограмма яркости носит выраженный бимодальний характер, где одна группа мод соответствует объекту, а другая — фону, то можно привязать порог бинаризации непосредственно к значению между этими модами. Теперь при любом, в том числе и нелинейном, изменении условий освещенности порог будет всегда оставаться между модами. Если при этом площади объекта и фона примерно равны, то порог окажется равным средней яркости.

Наиболее очевидный подход заключается в том, чтобы аппроксимировать две группы мод гладкими функциями, например гауссоидами, и найти точку их пересечения. Однако такой подход нецелесообразен, если моды имеют негауссовский характер. Поэтому для определения положения мод желательно использовать иные, например статистические, методы (рис. 2.4).

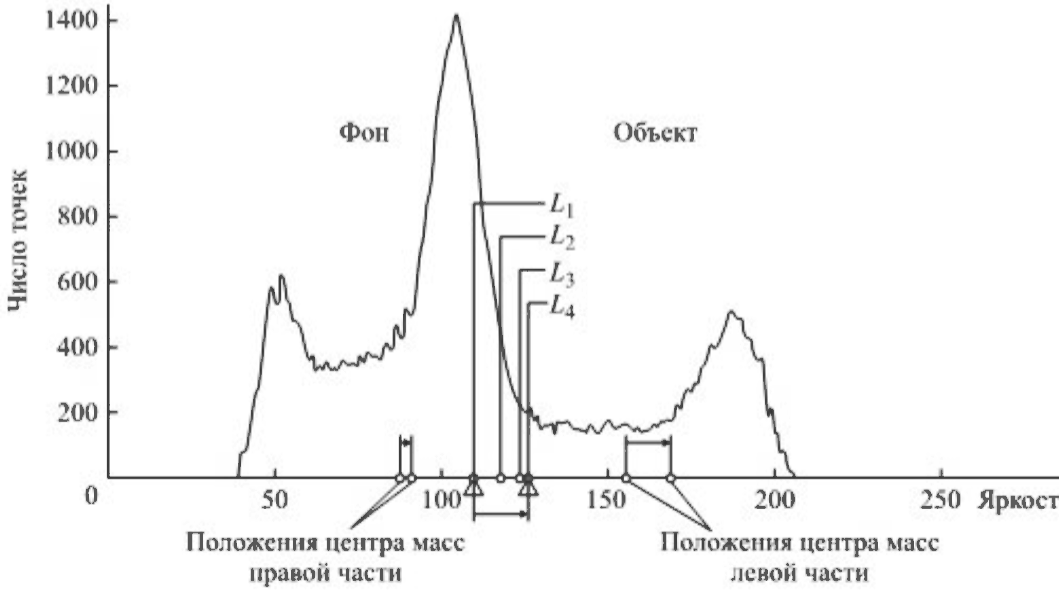


Рис. 2.4. Схема статистического алгоритма бинаризации:
L_1–L_4 — положения порога

Статистический алгоритм можно записать в следующем вмде

1. Принять за первое приближение порога бинаризации среднюю яркость изображения, которую можно рассчитать как центр масс X_c гистограммы яркости:

$$X_c = \frac{\sum_i iH[i]}{\sum_i H[i]},$$

где $H[i]$ — моды гистограммы яркости.
2. Разделить гистограмму на левую и правую части по найденному порогу. Определить центры масс левой и правой частей.
3. Выбрать новый порог посередине между центрами масс левой и правой частей (как их среднее арифметическое).
4. Повторять шаги 2, 3, пока порог не перестанет изменяться.

На практике данный алгоритм сходится достаточно быстро, за три-четыре итерации.

Если гистограмма яркости не является бимодальной, то этот алгоритм будет работать некорректно: порог может «убегать» вправо или влево до последней моды без разделения гистограммы на желаемые части. Если площади объекта и фона примерно равны, то порог будет найден за одну итерацию, и он будет равен средней яркости изображения. Алгоритм можно расширить и на большее число мод. В этом случае он преобразуется в алгоритм кластеризации.

Привязка к средней яркости точек границ. В некоторых задачах выделить точки границ гораздо проще, чем найти уровень бинаризации (методы выделения точек границ будут рассмотрены в гл. 3). В этом случае можно привязать уровень бинаризации непосредственно к средней яркости точек границ. Однако может возникнуть ситуация, когда существенная часть точек границ окажется принадлежащей объекту (либо, наоборот, фону). Чтобы избежать этого, средняя яркость рассчитывается не только по точкам границ, но и по их окрестностям.

Морфологические алгоритмы

Введем следующие обозначения. Пусть A — двумерный массив, содержащий исходное бинаризованное изображение. Будем считать, что точки, принадлежащие объекту, помечены единицами, а принадлежащие фону — нулями, независимо от яркости и иных характеристик объекта и фона. Пусть B — также двумерный массив, где отличные от нуля ячейки (разметка) принадлежат интересующим нас элементам объекта.

Следует отметить, что большинство морфологических операций требует отдельного массива-источника A и отдельного массива-приемника результата B. В одном массиве морфологические операции выполняться не могут. Если существуют серьезные ограничения на размер используемой памяти (например, для бортовых приложений), то в этом случае массивы A и B попеременно становятся источником и приемником информации.

Обозначим $A0$ текущую точку, над которой выполняется операция. Соответственно результат обработки точки $A0$ и ее окрестности будет записан

A6	*A7*	*A8*
A5	*A0*	*A1*
A4	*A3*	*A2*

Рис. 2.5. Обозначение точек окрестности

в элемент $B0$, координаты которого совпадают с координатами $A0$. При обработке изображения каждая точка поочередно становится текущей точкой $A0$. Если не указано иное, будем считать, что точки обрабатываются слева направо сверху вниз (как они располагаются на экране компьютера).

Для упрощения дальнейшей записи обозначим точки в окрестности $A0$ как $A1$–$A8$, например, по спирали по ходу часовой стрелки, как показано на рис. 2.5.

Фильтрация шума. Рассмотрим простейший морфологический алгоритм для фильтрации шума. Что собой представляет шум на бинаризованном изображении? Это либо нули, случайно появившиеся в точках объекта вместо единиц, либо единицы в точках фона вместо нулей. Если представить, что единицами обозначен более светлый объект, а нулями — более темный фон, то в первом случае получим хаотически расположенные темные точки на светлом объекте («перчинки»), а во втором — светлые точки на темном фоне («соль»). Такой шум традиционно называют «соль и перец» (*salt and pepper*).

Фильтрация шума типа «соль». Шум типа «соль» — это единицы на фоне нулей фона. Обнаружить такую шумовую точку очень просто: все ее соседние точки — нули. Найдя такую точку, ее следует заменить на нуль, а остальные точки оставить без изменений.

Запишем данный алгоритм на псевдоязыке, который несложно перевести на любой язык программирования (например, Паскаль).

Алгоритм на псевдоязыке

Для каждой точки $A0$:
найти $\sigma = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8$;
если $\sigma = 0$ и $A0 = 1$, то $B0 = 0$,
иначе $B0 = A0$.

Алгоритм на языке Паскаль

```
For i:=1 to Width-2 do
For j:=1 to Height-2 do
Begin
    Sigma:=A[i-1,j-1]+A[i-1,j]+A[i-1,j+1]+
    A[i,j-1]+A[i,j+1]+A[i+1,j-1]+A[i+1,j]+
    A[i+1,j+1];
    If (Sigma=0)and(A[I,j]=1)
    Then B[I,j]:=0
    Else B[I,j]:=A[I,j];
End;
```

Алгоритм на псевдоязыке позволяет записывать морфологические алгоритмы в достаточно компактной и наглядной форме, поэтому далее будем пользоваться именно такой формой записи.

Фильтрация шума типа «перец». Аналогично можно выполнить фильтрацию шума типа «перец». Шум типа «перец» — это нуль на объекте, помеченном единицами, и все соседи шумовой точки — единицы. Такую точку следует заменить на единицу, а остальные оставить без изменений.

Алгоритм на псевдоязыке и на языке Паскаль будет выглядеть следующим образом.

Алгоритм на псевдоязыке

Для каждой точки $A0$:
найти $\sigma = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8$;
если $\sigma = 8$ и $A0 = 0$, то $B0 = 1$,
иначе $B0 = A0$.

Алгоритм на языке Паскаль

```
For i:=1 to Width-2 do
For j:=1 to Height-2 do
Begin
    Sigma:=A[i-1,j-1]+A[i-1,j]+A[i-1,j+1]+
    A[i,j-1]+A[i,j+1]+A[i+1,j-1]+A[i+1,j]+
    A[i+1,j+1];
    If (Sigma=8)and(A[i,j]=0)
    Then B[i,j]:=1
    Else B[i,j]:=A[i,j];
End;
```

Совместная фильтрация шума «соль и перец». Можно объединить оба приведенных алгоритма в один, чтобы он фильтровал оба типа шума. Тогда на псевдоязыке и языке Паскаль он будет выглядеть следующим образом.

Алгоритм на псевдоязыке

Для каждой точки $A0$:
найти $\sigma = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8$;
если $\sigma = 0$ и $A0 = 1$, то $B0 = 0$,
иначе
  если $\sigma = 8$ и $A0 = 0$, то $B0 = 1$,
  иначе $B0 = A0$.

Алгоритм на языке Паскаль

```
For i:=1 to Width-2 do
For j:=1 to Height-2 do
Begin
    Sigma:=A[i-1,j-1]+A[i-1,j]+A[i-1,j+1]+
    A[i,j-1]+A[i,j+1]+A[i+1,j-1]+A[i+1,j]+
    A[i+1,j+1];
    If (Sigma=0)and(A[i,j]=1)
    Then B[i,j]:=0
    Else
        If (Sigma=8)and(A[i,j]=0)
        Then B[i,j]:=1
        Else B[i,j]:=A[i,j];
End;
```

Пример работы алгоритма показан на рис. 2.6, *б*. Алгоритм фильтрует только те точки, которые полностью окружены точками другой разметки. Это условие можно несколько смягчить, позволив фильтру убирать спаренные точки шума, у которых может быть одна соседняя точка своей разметки. Тогда алгоритм будет выглядеть следующим образом.

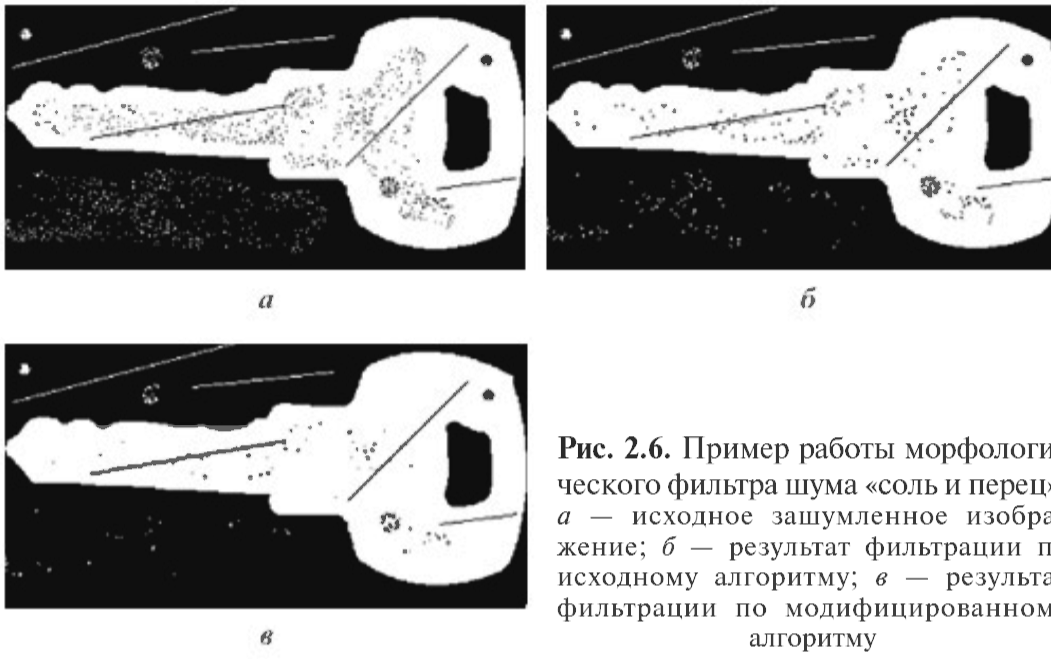
Алгоритм на псевдоязыке

Для каждой точки $A0$:
найти $\sigma = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8$;
если $\sigma < 2$ и $A0 = 1$, то $B0 = 0$,
иначе
  если $\sigma > 6$ и $A0 = 0$, то $B0 = 1$,
  иначе $B0 = A0$.

Алгоритм на языке Паскаль

```
For i:=1 to Width-2 do
For j:=1 to Height-2 do
Begin
    Sigma:=A[i-1,j-1]+A[i-1,j]+A[i-1,j+1]+
    A[i,j-1]+A[i,j+1]+A[i+1,j-1]+A[i+1,j]+
    A[i+1,j+1];
    If (Sigma<2)and(A[i,j]=1)
    Then B[i,j]:=0
    Else
        If (Sigma>6)and(A[i,j]=0)
        Then B[i,j]:=1
        Else B[i,j]:=A[i,j];
End;
```

Пример работы модифицированного алгоритма приведен на рис. 2.6, *в*. Видно, что фильтр убирает существенно больше шумовых точек, в том числе и сдвоенные. Однако более крупные объекты, например линии, по-прежнему остаются без изменений.



а *б* *в*

Рис. 2.6. Пример работы морфологического фильтра шума «соль и перец»: *а* — исходное зашумленное изображение; *б* — результат фильтрации по исходному алгоритму; *в* — результат фильтрации по модифицированному алгоритму

Фильтр выделения границ. Далее будем использовать два близких термина: «граница» и «контур». Под *границами* будем понимать отдельные точки объекта, лежащие рядом с точками фона. *Контур* — некоторая кривая (или набор кривых), соединяющая точки границы между собой. Таким образом, контур — это более высокоуровневое представление границ.

С точки зрения морфологии будем иметь два типа границ: границы объекта и границы фона. Точки границ объекта принадлежат объекту (имеют разметку объекта), а среди соседних точек имеются точки фона. Аналогично, точки границ фона принадлежат фону и имеют соседние точки с объектом.

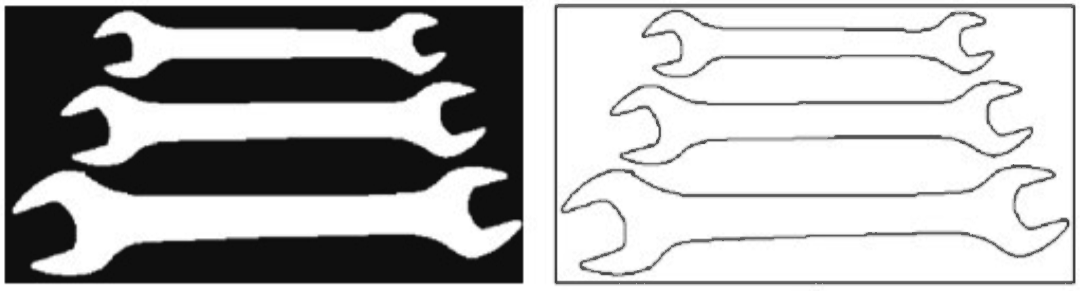
В соответствии с изложенным алгоритм выделения границ объекта может быть записан следующим образом.

Для каждой точки $A0$:
вычислить $\sigma = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8$;
если $\sigma < 8$ и $A0 = 1$, то $B0 = 1$, иначе $B0 = 0$.

Пример работы алгоритма показан на рис. 2.7.

Алгоритм выделения границ фона строится аналогично.

Отметим, что границы, выделяемые морфологическим фильтром, всегда замкнуты и не имеют разрывов. Если объект выходит за рамки изображения, то продолжением границ объекта следует считать границы самого изображения так, как если бы все точки вне изображения были помечены как фон.



а *б*

Рис. 2.7. Пример работы морфологического фильтра границ:
а — исходное изображение; *б* — найденные точки границы

В этой версии точке границы требуется как минимум две соседние точки противоположной разметки, что обеспечит меньшую чувствительность к шуму и более гладкий контур.

Для каждой точки $A0$:
вычислить $\sigma = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8$;
если $\sigma < 7$ и $A0 = 1$, то $B0 = 1$, иначе $B0 = 0$.

Фильтры эрозии и дилатации. Фильтр эрозии позволяет удалить точки объекта, лежащие на его границах. Таким образом, размер всех объектов на изображении будет уменьшен на две точки, по одной с каждой стороны. Объекты размером одна-две точки при этом будут удалены. При N-кратном применении фильтра размеры объектов уменьшатся на $2N$ точек, а объекты с размерами $2N$ и менее будут удалены.

Алгоритм фильтра эрозии может быть записан следующим образом.

Для каждой точки $A0$:
найти $\sigma = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8$;
если $\sigma < 8$ и $A0 = 1$, то $B0 = 0$, иначе $B0 = A0$.

Фильтр эрозии применяется для удаления крупных шумовых элементов, ошибочно помеченных как объект, если размер самого объекта известен заранее. В этом случае фильтр позволяет оставить объект, подавив шумовые участки, по размеру меньшие объекта.

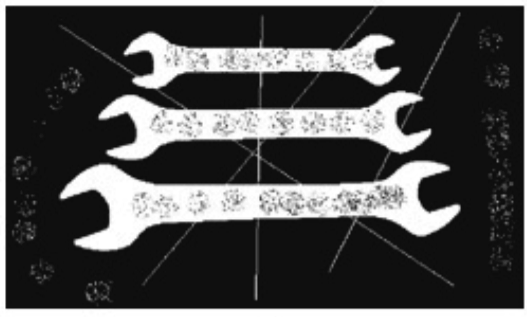
Пример работы фильтра эрозии представлен на рис. 2.8, *б*.

Аналогичным фильтром для обработки фона является фильтр дилатации. Фильтр помечает единицами точки фона, лежащие по соседству с точками объекта. Таким образом, если объект имеет отверстия, размеченные нулями, то их диаметр будет уменьшен на две точки. Отверстия диаметром две точки и менее при этом исчезнут. Соответственно, N-кратное применение фильтра приведет к удалению отверстий диаметром не более $2N$.

Алгоритм фильтра дилатации может быть записан следующим образом.

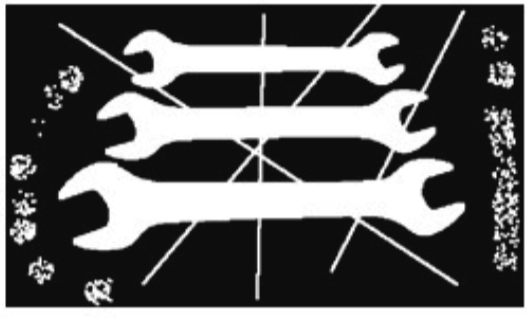
Для каждой точки $A0$:
найти $\sigma = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8$;
если $\sigma > 0$ и $A0 = 0$, то $B0 = 1$, иначе $B0 = A0$.

Фильтр дилатации используется для удаления крупных шумовых элементов, ошибочно помеченных как отверстия в объекте. Если размеры реаль-



а

б



в

Рис. 2.8. Пример работы фильтров эрозии и дилатации:
а — исходное изображение; *б* — результат эрозии; *в* — результат дилатации

ных отверстий известны, то последовательное применение фильтра удалит весь шум меньшего размера. Пример работы фильтра дилатации показан на рис. 2.8, *в*.

Операции размыкания и замыкания. Операции эрозии и дилатации не являются полностью взаимно обратными. Например, если выполнить сначала эрозию, а затем дилатацию, то дилатация не сможет восстановить элементы объектов толщиной менее трех, которые были устранены эрозией, тогда как другие элементы восстановятся без изменений. Такая операция называется *размыканием*. Она позволяет удалять тонкие перемычки между объектами и мелкие шумовые элементы, не затрагивая важные крупные объекты.

Последовательность дилатации и эрозии называется *замыканием*. Она устраняет мелкие шумовые элементы внутри объекта, не затрагивая крупные отверстия.

Используя дилатацию и эрозию на глубину более одной точки, получаем удобный инструмент для удаления произвольных шумовых элементов размером не более заданного (рис. 2.9).

Алгоритм расчета карты расстояний. Многократное применение фильтров эрозии и дилатации требует значительного объема вычислений. Его можно сократить, если предварительно разметить точки объекта (фона) расстоянием до ближайшей точки фона (объекта). Имея такую разметку, можно сразу определить результат N-кратной фильтрации, убирая точки с разметкой, меньшей либо равной N.

Воспользуемся двухпроходным алгоритмом. За первый проход, слева направо сверху вниз, найдем расстояния от точек объекта до левых и верх-

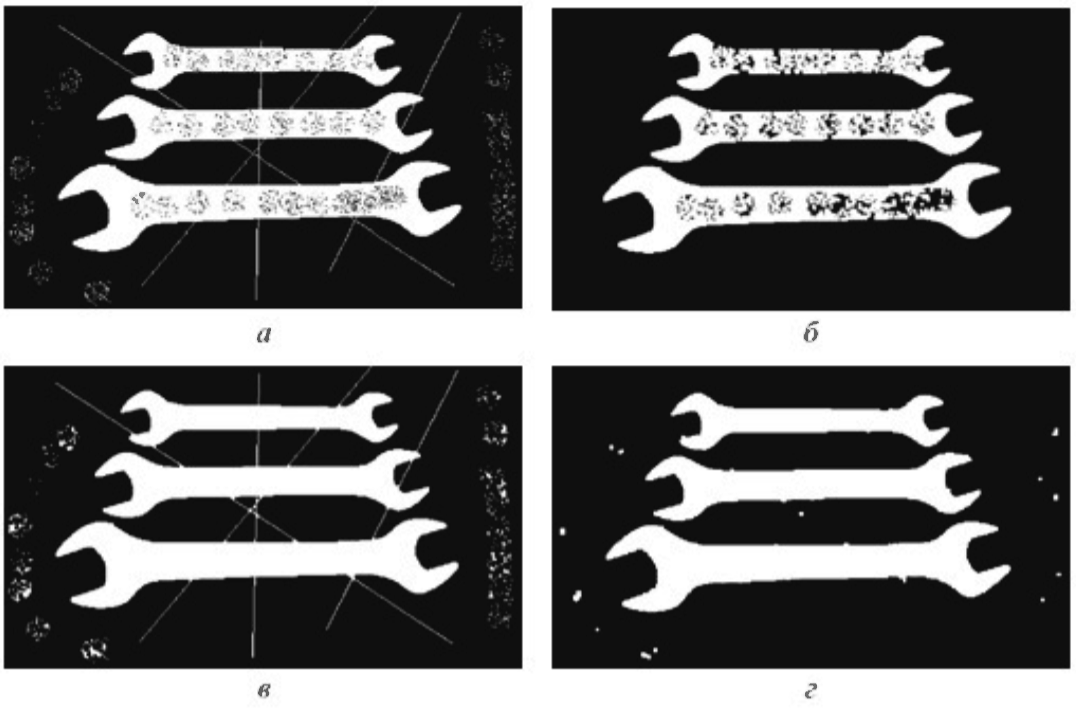


Рис. 2.9. Применение операций размыкания и замыкания:
а — исходное изображение; *б* — размыкание; *в* — замыкание; *г* — размыкание + замыкание

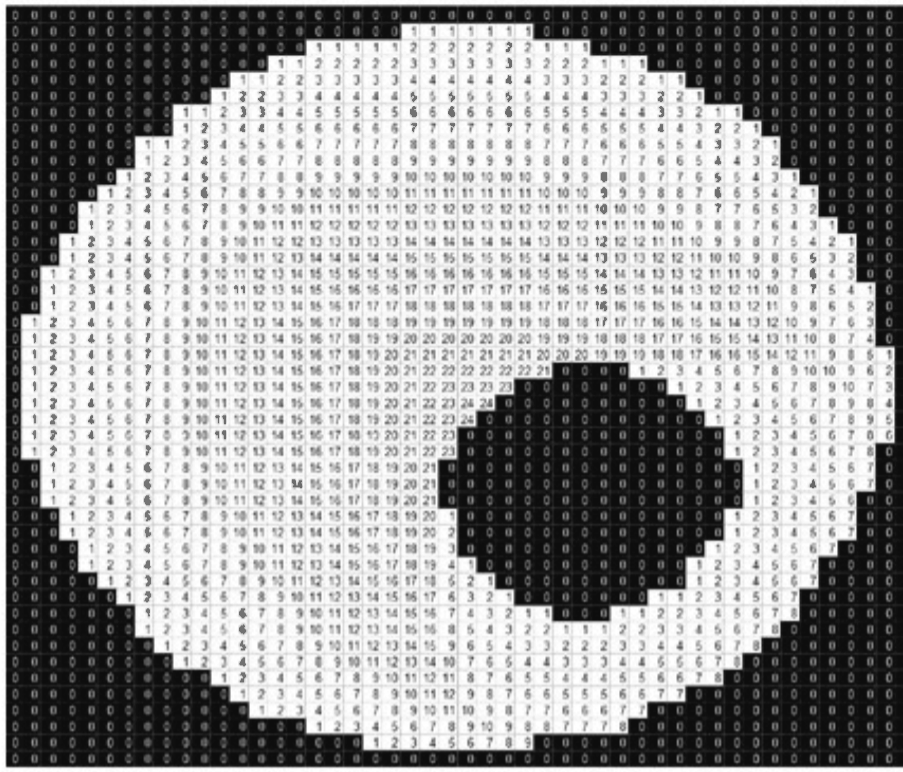
них границ, а за второй проход, в обратную сторону, справа налево снизу вверх, — расстояния до правых и нижних границ, и выберем наименьшее.

Воспользуемся следующим свойством: если задан порядок просмотра точек (например, слева направо сверху вниз), то для текущей точки всегда можно указать соседние (в данном случае верхнюю и левую), которые уже были просмотрены ранее и для которых вычислено расстояние. Тогда расстояние до текущей точки можно найти как минимальное расстояние из просмотренных соседних точек плюс единица. Получим следующий алгоритм.

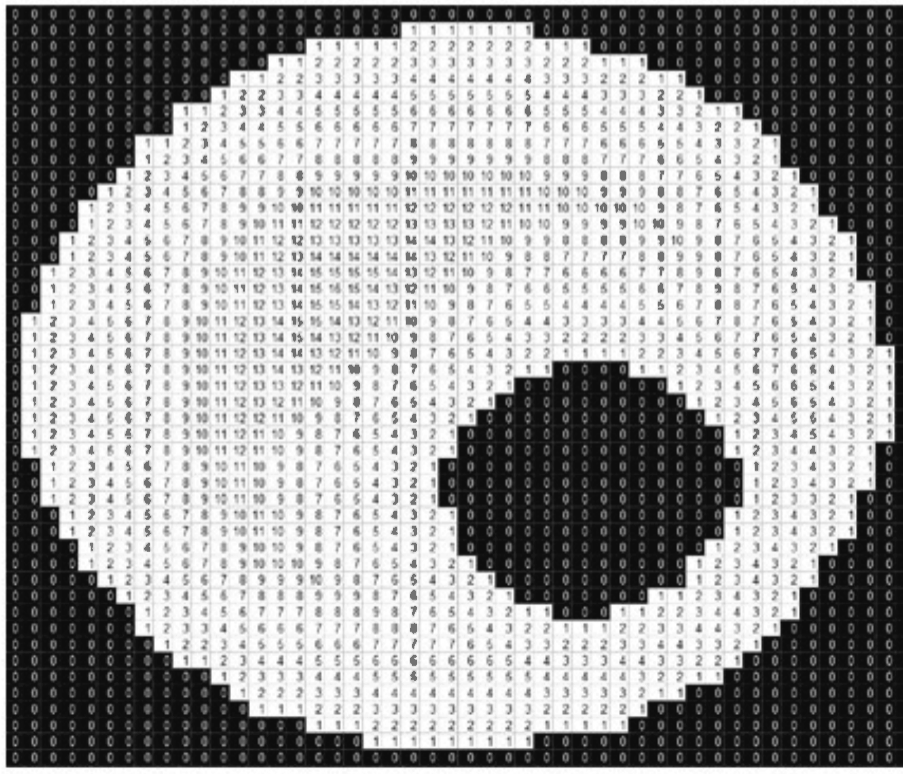
1. Проход — слева направо сверху вниз.
Для каждой точки $A0$:
если $A0 = 0$, то $B0 = 0$,
иначе $B0 = \min(B5, B7) + 1$;
2. Проход — справа налево снизу вверх.
Для каждой точки $B0$:
если $B0 \neq 0$, то $B0 = \min(B1 + 1, B3 + 1, B0)$.

Результаты работы алгоритма даны на рис. 2.10.

Алгоритм с учетом диагональных границ. Рассмотренный алгоритм корректно учитывает только расстояния до горизонтальных и вертикальных границ, тогда как расстояния до диагональных границ рассчитываются грубо. Эту проблему легко решить, если ввести в рассмотрение также и диагональные элементы — $A6$ и $A8$ для первого прохода, и $A2$, $A4$ — для второго. Расстояния до них можно принять равным $\sqrt{2}$ точек. Получим следующий алгоритм.



а



б

Рис. 2.10. Пример работы алгоритма расчета карты расстояний:
а — результат первого прохода; *б* — результат второго прохода

1. Проход — слева направо сверху вниз.
Для каждой точки $A0$:
если $A0 = 0$, то $B0 = 0$,
иначе $B0 = \min(B5 + 1, B6 + \sqrt{2}, B7 + 1, B8 + \sqrt{2})$.
2. Проход — справа налево снизу вверх.
Для каждой точки $B0$:
если $B0 \neq 0$, то $B0 = \min(B1 + 1, B2 + \sqrt{2}, B3 + 1, B4 + \sqrt{2}, B0)$.

Этот алгоритм можно реализовать и в целых числах, например, добавляя значения 10 и 14 вместо 1 и $\sqrt{2}$.

Аналогичный алгоритм может использоваться и для расчета карты расстояний для точек фона. В этом случае достаточно проинвертировать разметку исходного изображения.

Рассчитанная карта расстояний применяется для выполнения операций эрозии и дилатации на произвольную глубину всего за один проход. Карта расстояний может потребоваться и для других операций, таких как расчет лент и поиск скелетных линий.

Разметка объектов и определение их числа. Прежде чем переходить к определению параметров отдельных объектов, их идентификации и сравнению с шаблоном, необходимо отделить точки разных объектов друг от друга, например, пронумеровав их разными числами. Этот процесс называется *разметкой* объектов. Разметка может потребоваться не только для объектов, но и для отверстий в них. В этом случае сначала следует разметить сами объекты, отнести их к отдельным массивам, инвертировать, и затем разметить отверстия аналогично разметке объектов.

Алгоритм разметки может выполняться по аналогии с заливкой объектов краской. Будем последовательно просматривать все точки объекта (точки фона будем сразу маркировать нулями). Если точка объекта не имеет соседних размеченных точек, то она принадлежит новому объекту. Увеличим счетчик маркировок на единицу и пометим точку новой разметкой.

Если точка имеет хотя бы одну размеченную соседнюю точку, то в нее копируется разметка этой точки (которая, очевидно, принадлежит тому же объекту). Особым является случай, когда две и более соседние точки получили разную разметку. Это означает, что были найдены две или более частей одного и того же объекта, и их надо объединить в единое целое. Простейший вариант — перемаркировать все точки с большей разметкой в меньшую. Однако такой вариант потребует многократного повторного просмотра размеченных точек и, как следствие, большого объема вычислений.

Возможным решением будет введение таблицы перемаркировок, содержащей соответствие между маркировкой точек и номером объекта, которому принадлежит маркировка. Тогда одному объекту может соответствовать несколько маркировок, а все операции объединения маркировок будут происходить не на изображении, а в достаточно небольшой по размеру таблице.

Таблицу перемаркировок можно представить одномерным массивом $T[n]$, где индекс n — маркировка, а значение $T[n]$ — номер объекта, соответствующий маркировке n.

Тогда алгоритм разметки может выглядеть следующим образом.

Задать начальные значения:
$K = 0$; (текущее число разметок)
$N = 0$; (текущее число объектов)
$T[0] = 0$; (разметка фона).
1. Проход – слева направо сверху вниз:
Для каждой точки $A0$:
если $A0 = 0$, то $B0 = 0$,
иначе:
если $A5 = 0$ и $A7 = 0$, то:
$K = K + 1$; $N = N + 1$; $T[K] = N$; $B0 = K$; (новый объект)
иначе:
если $A5 = 0$ и $A7 = 1$, то: $B0 = B7$; (продолжение $A7$)
иначе:
если $A5 = 1$ и $A7 = 0$, то: $B0 = B5$; (продолжение $A5$)
иначе: (оставшийся вариант — $A0 = A5 = A7 = 1$)
если $B5 = B7$ то: $B0 = B5$;
иначе: (требуется слияние)
пусть n_1 — меньший номер объекта,
пусть n_2 – больший номер объекта:
$n_1 = \min(T[B5], T[B7])$;
$n_2 = \max(T[B5], T[B7])$;
Для каждой ячейки таблицы $T[i]$, $i = 1, ..., K$:
если $T[i] = n_2$, то $T[i] = n_1$;
если $T[i] > n_2$, то $T[i] = T[i] - 1$;
$N = N - 1$;
$B0 = k_1$.
2. Проход — замена маркировки номером объекта:
Для каждой точки $B0$:
$B0 = T[B0]$.

Алгоритм требует всего два прохода по изображению и работает достаточно быстро. К сожалению, его нельзя выполнять параллельно, поскольку порядок просмотра точек имеет принципиальное значение.

Характеристики бинарного изображения

Для эффективного решения задач поиска, идентификации и сравнения объектов, их обычно требуется представить в виде небольшого количества общих характеристик, а не координат отдельных безликих точек. Такие характеристики могут, например, отражать положение особых точек объекта, характер распределения точек вдоль осей, форму объекта и многие другие его особенности. Рассмотрим более подробно некоторые из этих характеристик.

Коды Фримана. Коды Фримана, или цепные коды (Freeman Code, Chain Code) являются одним из первых способов описания бинарного объекта в виде более общих характеристик, чем положения отдельных точек. Коды Фримана задают набор элементарных единичных векторов, по которым необходимо пройти, чтобы совершить полный обход контура и вернуться в исходную точку.

Пусть известны точки границ объекта (рис. 2.11). Выберем на границе отправную (стартовую) точку, например левую нижнюю. Обойдем контур по «правилу правой руки» (как бы все время держась правой рукой за объект и не отпуская его).

Если есть возможность повернуть направо относительно предыдущего направления движения — поворачиваем и движемся направо; в противном случае аналогично проверяем направления вперед, влево и назад.

Направление каждого единичного шага будем кодировать одним из чисел 0, 1, 2 или 3 (см. рис. 2.11). Список таких направлений при обходе контура и называется кодом Фримана.

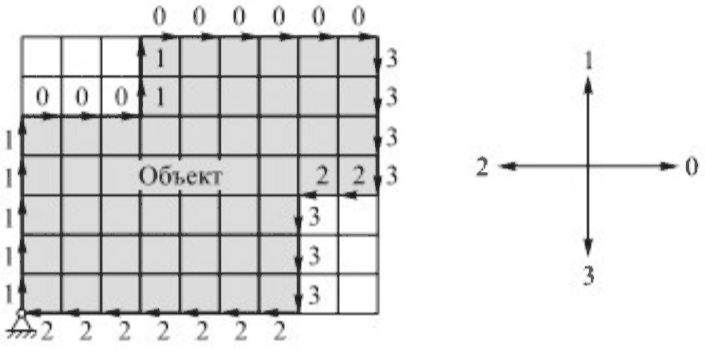


Рис. 2.11. К пояснению понятия кодов Фримана

Поскольку границы бинарных объектов всегда замкнуты и неразрывны, то, обойдя весь контур, окажемся в стартовой точке, из которой начинали движение.

Код Фримана компактен: он требует всего 2 бит для кодирования каждого шага, что было весьма удобно для ранних компьютеров с их небольшим объемом памяти. Кроме того, он весьма «дружественный» по отношению к оконечному оборудованию. Так, перемещая фрезу по пути, заданному кодом Фримана, можно изготовить твердую копию объекта, вырезав его из фанеры, жести, листовой стали и т. п.

Возможен также вариант кода Фримана с восемью направлениями вместо четырех. Он будет содержать также и диагональные шаги. Для формирования такого кода при выделении границ необходимо использовать 4-соседство вместо 8 для удаления лишних точек границ и лишних элементов в коде Фримана.

Свойства кода Фримана:

1) инвариантность к изменению положения объекта. При перемещении объекта по изображению будет перемещаться и стартовая точка. Таким образом, код Фримана не изменится;

2) инвариантность к выбору стартовой точки. Если выбрать другую стартовую точку, то код Фримана циклически сместится; при этом его структура останется неизменной. Такой тип инвариантности будем называть *структурной инвариантностью*;

3) структурная инвариантность к поворотам на угол, кратный 90°. При повороте объекта на угол 90° произойдет следующее:

- направления вправо (0) превратятся в направления вверх (1);
- направления вверх (1) превратятся в направления влево (2);
- направления влево (2) превратятся в направления вниз (3);
- направления вниз (3) превратятся в направления вправо (0).

Если при повороте происходит изменение стартовой точки, то код Фримана дополнительно циклически сместится. Однако структура кода Фримана при этом не изменится.

Недостатки кода Фримана:

1) чувствительность к поворотам на произвольный угол и к масштабированию. При этих преобразованиях структура кода претерпевает существенные изменения, в том числе из-за эффектов привязки точек объекта к дискретной сетке;

2) чувствительность к шуму. Отдельные шумовые точки могут приводить к весьма значительным изменениям кода (рис. 2.12)

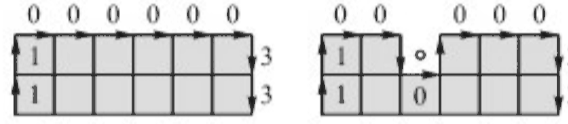


Рис. 2.12. Влияние шума на код Фримана:
а — исходный код; *б* — точка шума

Наличие подобных недостатков заставляет искать более надежные и устойчивые к шуму и изменениям ракурса признаки.

Центроидальный профиль. Будем последовательно перебирать точки границ объекта, начиная с крайней правой и далее против хода часовой стрелки. Будем строить радиус-вектор из центра масс объекта до каждой текущей точки границы, как показано на рис. 2.13. При движении конца радиус-вектора по точкам границ он будет поворачиваться относительно центра объекта, а его длина будет изменяться в зависимости от формы объекта.

Зависимость длины такого радиус-вектора от его направления называют *центроидальным профилем*.

Пусть (x_C, y_C) — координаты центра масс объекта, а $\rho(\varphi)$ — радиус-вектор из центра масс до точек границы объекта (см. рис. 2.13). Двигаясь после-

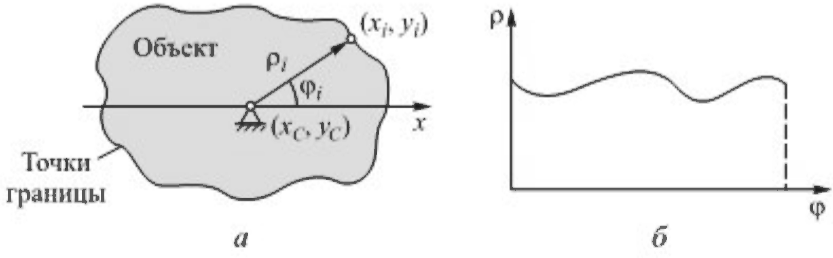


Рис. 2.13. К пояснению понятия центроидального профиля:
а — объект; *б* — центроидальный профиль объекта

довательно по всем точкам границ, будем получать различные значения ρ и φ. Зависимость ρ(φ) и является центроидальным профилем. Ее можно также рассматривать и как массив ρ[φ], где φ — индекс массива ρ.

Центроидальный профиль удобно использовать и для идентификации объектов, и для поиска отклонений объекта от образца.

Свойства центроидального профиля:

1) инвариантность к положению. Поскольку построение центроидального профиля выполняется относительно центра масс объекта (который будет перемещаться неотрывно от самого объекта), то форма центроидального профиля не зависит от положения объекта на изображении;

2) инвариантность к поворотам. При поворотах объекта на произвольный угол Δφ изменится только положение оси ρ: она также сместится на Δφ (центроидальный профиль циклически сдвинется на величину Δφ);

3) инвариантность к масштабированию. При изменении размера объекта в *K* раз центроидальный профиль также увеличится в *K* раз. Однако можно обеспечить и полную инвариантность к масштабированию, если выполнить нормирование центроидального профиля, например, разделив его на максимальное значение. Более надежный способ нормирования заключается в делении на среднее значение ρ, поскольку максимальное значение может оказаться шумовой точкой, тогда как среднее значение устойчиво к шуму;

4) устойчивость к шуму. Небольшие шумовые элементы будут приводить лишь к незначительным изменениям центроидального профиля, причем только в местах расположения этих шумовых элементов.

Недостатки центроидального профиля:

1) чувствительность к изменению положения центра масс объекта. Даже при небольших изменениях положения центра масс, вызванных, например, попаданием части объекта в тень или физическим отсутствием части объекта, форма центроидального профиля может изменяться весьма значительно;

2) не все объекты могут быть описаны при помощи центроидального профиля. Он хорошо подходит для описания выпуклых объектов без отверстий, но для объектов с отверстиями или для объектов сложной невыпуклой формы одному значению φ может соответствовать сразу несколько значений ρ (рис. 2.14). В этом случае центроидальный профиль нельзя представить ни функцией, ни массивом.

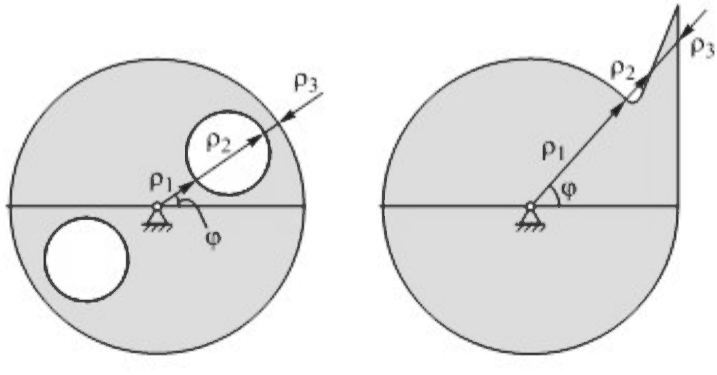


Рис. 2.14. Фигуры, для которых невозможно построить центроидальный профиль

Если выбрать одно максимальное значение ρ из нескольких возможных, то это, безусловно, будет искажать форму объекта, однако сделает центроидальный профиль более универсальным.

Построение центроидального профиля не требует последовательного просмотра точек границ, их можно рассматривать в произвольном порядке. Для каждой точки границы будем рассчитывать радиус-вектор ρ(φ) и отмечать его в массиве-приемнике ρ[φ]. Тогда получим следующий алгоритм.

1. Задать исходный пустой массив ρ[0...360] = (0, 0, ..., 0).
2. Рассчитать центр масс объекта (x_C, y_C):

$$M_x = \sum_{x=1}^{N}\sum_{y=1}^{N} xA[x,y]; \quad M_y = \sum_{x=1}^{N}\sum_{y=1}^{N} yA[x,y]; \quad M_0 = \sum_{x=1}^{N}\sum_{y=1}^{N} A[x,y];$$

$x_c = M_x/M_0$; $y_c = M_y/M_0$.

3. Для каждой точки границы объекта (x, y):

вычислить $\Delta x = x - x_C$;

$\Delta y = y - y_C$;

$\rho_1 = \sqrt{\Delta x^2 + \Delta y^2}$;

$\varphi_1 = \text{arctg}\,(\Delta y/\Delta x)$;

занести значения в массив:

если $\rho[\varphi_1] < \rho_1$, то $\rho[\varphi_1] = \rho_1$.

Недостаток этого алгоритма в том, что массив ρ[φ] может содержать «провалы» — такие значения φ, для которых не было найдено ни одной точки границы. Это особенно характерно для объектов небольшого размера, где шаг вдоль границы на один пиксель может превышать угол в одну градацию φ. Однако его легко исправить, скопировав в пустые ячейки ρ соседние значения, т. е. построить интерполяцию по ближайшим ненулевым соседним точкам.

Алгоритм заполнения пропусков может иметь следующий вид.

1. Найти первое непустое значение ρ[φ*] > 0.
2. Для всех значений φ = φ* + 1...360°:
   если ρ[φ] = 0, то ρ[φ] = ρ[φ − 1].
3. Для всех значений φ = φ* − 1...0° (в обратном порядке):
   если ρ[φ] = 0, то ρ[φ] = ρ[φ + 1].

Центроидальный профиль является удобным и эффективным инструментом для идентификации и сравнения объектов и широко применяется прежде всего в области технического зрения.

Ленты и скелетные линии. *Лентой* (*ribbon*) называется совокупность направляющей линии и образующей фигуры. Образующая, двигаясь вдоль направляющей, изменяет свои размеры и заполняет все внутреннее пространство описываемого объекта (рис. 2.15).

В качестве образующей чаще всего используется окружность радиусом $R(x, y)$, квадрат или восьмиугольник со стороной $a(x, y)$.

Направляющая представляет собой геометрическое место точек, равноудаленных от ближайших точек границ объекта. Расстояние от точки направляющей до ближайшей точки границы равно радиусу образующей окружности (или половине стороны квадрата, если образующая — квадрат, и т. д.).

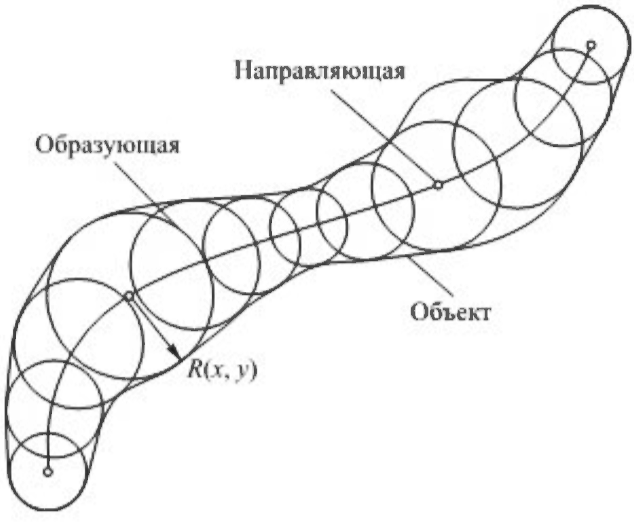


Рис. 2.15. Описание объекта в виде ленты

Направляющую линию также часто называют *скелетной линией* (*skeleton*), поскольку она формирует остов объекта, его каркас.

Наиболее распространенный метод построения скелетной линии — метод «встречного пала», он же метод «пожар в лесу» (в английской версии *prairie fire* — пожар в прериях). Метод апеллирует к способу тушения лесных пожаров методом встречного пала, при котором пожарные, вырубив просеку, создают встречную волну огня. Две волны огня, схлестываясь, взаимно гасят друг друга, оставляя полосу нетронутого леса на месте встречи.

Данный алгоритм эмулирует волну, последовательно устраняя из состава объекта точки его границ, за исключением точек связности. Точка объекта называется *точкой связности*, если она соединяет две и более части объекта в единое целое. При удалении таких точек объект будет распадаться на части. Очевидно, что алгоритм не должен затрагивать такие точки.

Проверить, является ли точка точкой связности, достаточно легко. Для этого надо пройти по 8-окрестности текущей точки, например, по часовой стрелке, и подсчитать число переходов из единицы в нуль. Если число таких переходов больше единицы, то данная точка является точкой связности (рис. 2.16). Вообще говоря, число переходов равно числу частей объекта, которые связывает данная точка.

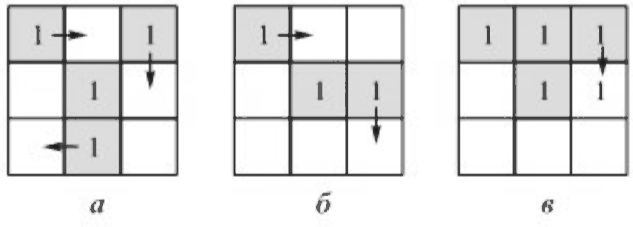


Рис. 2.16. Выделение точек связности:
а — три перехода (точка связывает три части объекта); *б* — два перехода (точка связывает две части объекта); *в* — один переход (точка не является точкой связности)

Алгоритм работает следующим образом. Необходимо последовательно убирать из объекта точки, лежащие на его границах, за исключением точек связности. Этот процесс повторяется до тех пор, пока можно найти хотя бы одну точку, подлежащую удалению. По окончании работы алгоритма останутся только точки связности, формирующие скелетную линию. Номер итерации, на которой найдена точка связности, равна расстоянию от нее до ближайшей границы объекта, и равна, соответственно, радиусу образующей.

Характерной особенностью такого алгоритма будет наличие «рожек», или «шпор» — длинных отростков вблизи краев объекта (рис. 2.17, *а*). Данный алгоритм может выполняться параллельно. При последовательной реализации он будет работать довольно медленно и заранее не известное количество времени, что требует альтернативных подходов с небольшим фиксированным числом проходов.

а *б*

Рис. 2.17. Сравнение работы алгоритмов выделения лент:
а — метод «пожар в лесу»; *б* — метод на основе карты расстояний

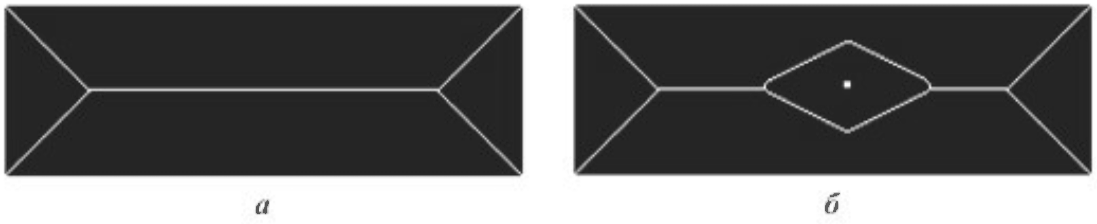
Альтернативой методу «пожар в лесу» может быть метод с использованием карты расстояний. Сначала рассчитываются расстояния от точек объектов до точек границ. Точки, равноудаленные от границ, будут давать локальные максимумы разметки (рис. 2.17, *б*). Такой алгоритм потребует всего три прохода — два для построения карты расстояний и один для поиска локальных максимумов.

В этом алгоритме скелетная линия может быть разорванной и состоять из отдельных изолированных точек, поэтому при необходимости их следует соединить в одну линию. Другим недостатком является получение линии неединичной толщины (для объектов четного размера). Здесь дополнительно вводится правило предпочтения: например, правые точки важнее левых, а верхние важнее нижних; более важные направления добавляют +0,5 единиц расстояния при поиске локальных максимумов. Это позволит получить линию одной толщины.

Свойства лент:

1) подходят для описания протяженных объектов (и не подходят для непротяженных);

2) крайне чувствительны к наличию шумовых точек на самом объекте: наличие даже единичной шумовой точки способно ее изменить до неузнаваемости (рис. 2.18). Поэтому перед построением ленты необходимо тща-



а *б*

Рис. 2.18. Влияние шума на форму ленты:
а — лента объекта без искажений; *б* — лента объекта с одной шумовой точкой вблизи его центра

тельным образом устранить подобный шум, применяя, например, операцию замыкания областей с большой глубиной.

Выпуклые оболочки. Выпуклые оболочки (Convex Hull) представляют собой описание объекта в виде иерархии выпуклых многоугольников (как правило, восьмиугольников), каждый из которых полностью включает в себя объект, отверстие в объекте либо соответствующую часть объекта в зависимости от уровня иерархии.

Пусть имеется объект произвольной формы (рис. 2.19). Оболочкой 1-го уровня будет многоугольник, полностью содержащий в себе объект. При этом каждая сторона многоугольника должна проходить хотя бы через одну точку объекта.

Видно, что оболочка 1-го уровня является весьма грубым приближением объекта. Она содержит множество областей, не принадлежащих объекту, и вообще не похожа на исходный объект. Поэтому такое представление нуждается в уточнении: из оболочки 1-го уровня необходимо удалить пустые области. Представим эти области также в виде выпуклых оболочек. Назовем их оболочками 2-го уровня (рис. 2.20).

Результат стал более похожим на исходный объект, однако теперь часть областей, принадлежащих объекту, оказались из него исключены. Чтобы устранить этот недостаток, будем вводить в рассмотрение оболочки 3-го, 4-го и т. д. уровней, пока не будет достигнута желаемая точность представ-

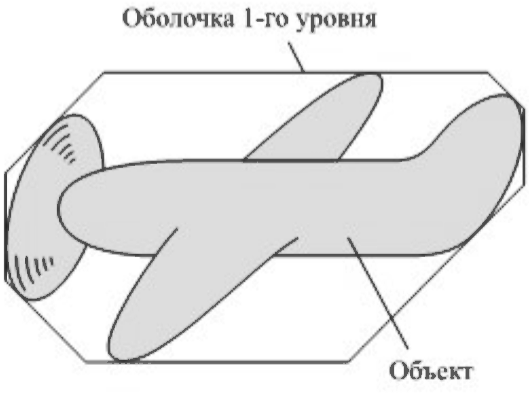


Рис. 2.19. Описание объекта в виде выпуклых оболочек

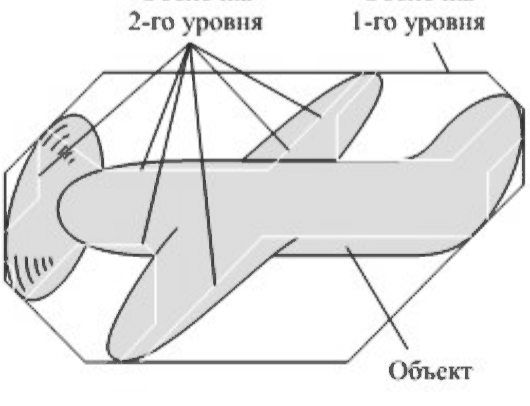


Рис. 2.20. Выпуклые оболочки 1-го и 2-го уровней

ления. При этом оболочки нечетного порядка будут добавляться к объекту, а четного — вычитаться. В результате получим иерархическое представление объекта в виде набора выпуклых многоугольников, от более общих деталей к менее значимым.

Для построения оболочки можно использовать морфологический и статистический подходы.

Морфологический подход. Этот подход заключается в поиске точек фона, создающих вогнутость на объекте, и в добавлении их к объекту до тех пор, пока вогнутостей не останется и объект не станет выпуклым. Точку вогнутости найти легко: это точка фона, у которой четыре и более соседние точки — точки объекта.

Алгоритм формирования оболочек можно записать таким образом.

1. Для каждой точки изображения A0:
   вычислить $\sigma = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8$;
   если $A0 = 0$ и $\sigma > 3$, то $B0 = 1$, иначе $B0 = A0$.
2. Если были найдены новые точки, то $A = B$ и повторить шаги 1, 2.

Алгоритм может потребовать достаточно большого числа проходов, и выполнять его желательно с использованием параллельных вычислительных средств.

Статистический подход. Из рис. 2.21 видно, что правые и левые вертикальные стороны оболочки всегда проходят через крайние левую и правую точки объекта. Аналогично верхняя и нижняя горизонтальные стороны всегда проходят через крайние верхнюю и нижнюю точку объекта.

Для того чтобы найти оставшиеся диагональные стороны восьмиугольника, введем еще две оси — d_1 и d_2, повернутые на угол 45°:

$$d_1 = \frac{1}{\sqrt{2}}(y+x); \quad d_2 = \frac{1}{\sqrt{2}}(y-x).$$

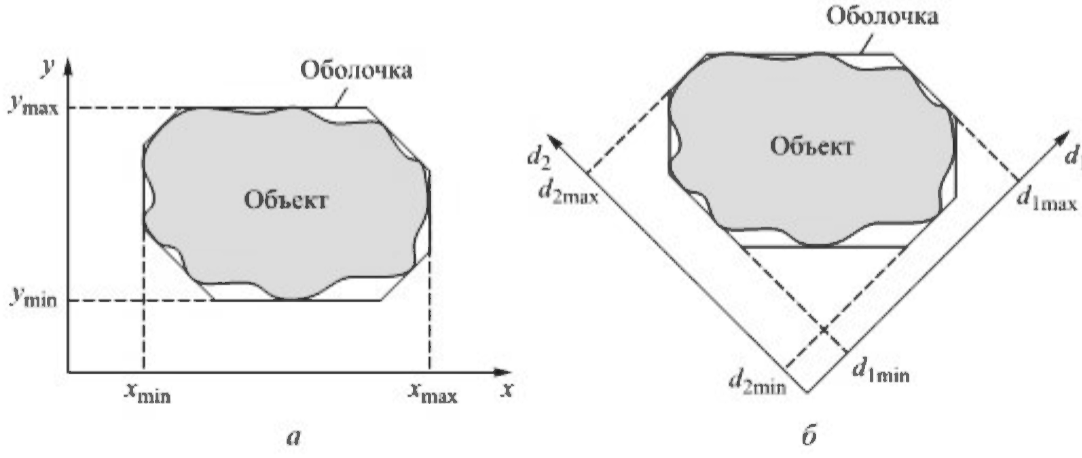


Рис. 2.21. Параметры выпуклых оболочек:
а — параметры вертикальных и горизонтальных сторон; *б* — параметры диагональных сторон

Как и в предыдущем случае, найдем максимумы и минимумы величин $(x + y)$ и $(y - x)$, которые и будут определять положение диагональных сторон восьмиугольника.

Это позволяет найти параметры оболочки всего за один проход, причем возможна как последовательная, так и параллельная реализация.

Алгоритм можно обобщить на случай произвольных n-угольных оболочек. В этом случае находят максимальные и минимальные значения проекций точек на набор осей $d(\varphi)$:

$$d(\varphi) = x \cos \varphi + y \sin \varphi,$$

где φ — угол наклона оси.

Для этого случая значения $\cos \varphi$ и $\sin \varphi$ рассчитывают заранее и представляют в виде приближенной дроби $m/2^n$, где m, n — целые числа.

Выпуклые оболочки, в отличие от предыдущих способов, позволяют описывать объект иерархически — от более важных деталей к менее важным. Для решения задач идентификации и сравнения с шаблоном можно взять необходимое число оболочек верхнего уровня, используя оставшиеся уровни для уточнения результата.

Глава 3
АНАЛИЗ ИЗОБРАЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ ГРАНИЦ ОБЪЕКТОВ

Выделение границ

Существует большое количество практических задач, в которых разделить точки объекта и фона по их яркости или по какой-то другой характеристике не представляется возможным. Это может быть связано с отсутствием данных о соотношении яркостей фона и объекта интереса, с существенными изменениями освещенности и условий видимости, присутствием на изображении мешающих объектов и многими другими факторами. Однако и в таких случаях точки объекта могут существенно отличаться от точек фона (некоторым заранее не известным образом), что позволяет найти точки, лежащие на границе объекта и фона и использовать их для дальнейшего анализа. Это возможно потому, что точки границы, хотя и не несут информации о цвете и яркости объекта, несут информацию о его форме.

Что же такое точки границ и как их находить на исходном неразмеченном изображении? Пусть имеется некоторое изображение, содержащее объект (рис. 3.1). Рассмотрим функцию яркости в некотором сечении A–A, проходящем через объект. Для упрощения будем считать, что сечение параллельно

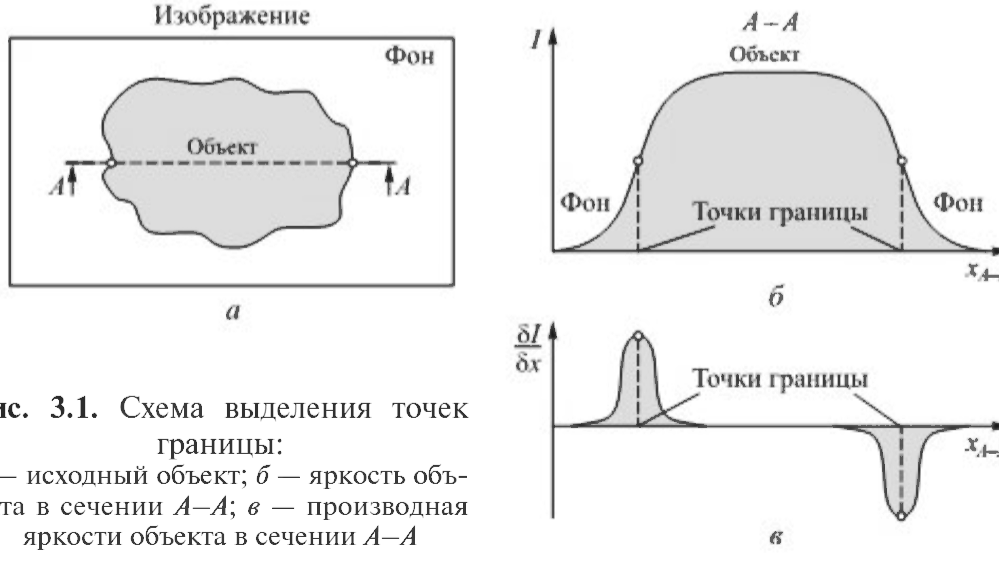


Рис. 3.1. Схема выделения точек границы:
а — исходный объект; *б* — яркость объекта в сечении A–A; *в* — производная яркости объекта в сечении A–A

оси x. Точкой границы объекта и фона можно считать точки, в которых яркость изображения при движении вдоль сечения (при переходе от объекта к фону или наоборот) изменяется наиболее быстро. Если бы функция яркости $I(x)$ была бы непрерывной и дифференцируемой, то точки границ соответствовали бы локальным экстремумам первой производной $\delta I/\delta x$.

Однако изображение — это двумерная функция, и сечение $A–A$ может проходить под любым углом к границе (в том числе и вдоль нее). В этом случае производная уже может и не дать локальный максимум в точке границы, и следует рассматривать обе частные производные по координатам x и y одновременно. Для этого введем в рассмотрение вектор $\vec{G}(x,y)=\left(\frac{\partial I(x,y)}{\partial x},\frac{\partial I(x,y)}{\partial y}\right)$, составленный из частных производных функции яркости изображения в данной точке. Этот вектор представляет собой вектор градиента яркости (рис. 3.2).

Модуль вектора градиента будет показывать наибольшую скорость изменения яркости, т. е. степень выраженности границы в данной точке. Модуль градиента может быть вычислен по формуле

$$\left|\vec{G}(x,y)\right|=\sqrt{\frac{\partial I(x,y)}{\partial x}^2+\frac{\partial I(x,y)}{\partial y}^2}.$$

Градиент яркости указывает направление, в котором яркость изменяется наиболее быстро. Поэтому он всегда перпендикулярен направлению границ. Направление градиента можно вычислить по формуле

$$\arg\vec{G}=\operatorname{arctg}\frac{\delta I/\delta y}{\delta I/\delta x}.$$

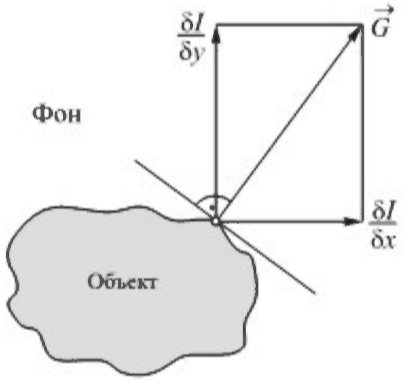


Рис. 3.2. Градиент яркости изображения

В приложениях, чувствительных к объему вычислений, модуль градиента яркости можно рассчитать по приближенным формулам без использования квадратного корня:

$$\left|\vec{G}(x,y)\right|\approx\max\left(\frac{\partial I(x,y)}{\partial x},\frac{\partial I(x,y)}{\partial y}\right)$$ — низкая точность,

$$\left|\vec{G}(x,y)\right|\approx\frac{1}{2}\left[\frac{\partial I(x,y)}{\partial x}+\frac{\partial I(x,y)}{\partial y}+\max\left(\frac{\partial I(x,y)}{\partial x},\frac{\partial I(x,y)}{\partial y}\right)\right]$$ — средняя точность.

Как правило, точное значение модуля градиента в дальнейших вычислениях не используется (его только сравнивают с порогом), и такая аппроксимация вполне допустима.

Для вычисления направления градиента удобно применять функцию atan2 (y,x), реализованную в большинстве современных языков программирования. Она рассматривает (x, y) как точку и вычисляет угол направления

на нее в диапазоне 0...360°. Для ускорения вычислений также можно использовать и значения арктангенса, заданные таблично с требуемой точностью.

Поскольку изображение — это дискретная функция, то для него следует построить дискретный аналог производной, например в виде конечной разности. Простейший способ — воспользоваться определением производной

$$\frac{\partial I(x,y)}{\partial x} = \lim_{\Delta x \to 0} \frac{I(x+\Delta x, y) - I(x,y)}{\Delta x}.$$

Минимальное значение Δx — это шаг сетки, равный единице. Поэтому можно записать приближенное равенство

$$\frac{\partial I(x,y)}{\partial x} \approx \{\Delta x = 1\} \approx I(x+\Delta x, y) - I(x,y),$$

аналогично для оси Y

$$\frac{\partial I(x,y)}{\partial y} \approx \{\Delta y = 1\} \approx I(x, y+\Delta y) - I(x,y).$$

Производную можно также записать в виде маски фильтра:

для производной по x: $M_x = [1 \quad -1]$;

для производной по y: $M_y = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$.

Очевидно, что такой простой подход вполне работоспособен (рис. 3.3), однако он имеет существенные недостатки:

• крайне низкая точность определения направления градиента. Для направлений границ, не кратных 45°, производные вычисляются слишком грубо и не позволяют найти точное направление границы;

• высокая чувствительность к шуму. Поскольку для вычислений каждой производной используются лишь две точки, то вероятность зашумления хотя бы одной из них очень высока; в результате случайные шумовые точки будут помечены как границы объекта, тогда как реальные границы могут быть пропущены;

• неизотропность. Результат выделения границ будет различным при разных направлениях прохода по изображению; более того, само положение точки границы будет смещаться относительно реального положения, причем смещение зависит от направления прохода.

Для устранения этих недостатков следует учитывать большее число точек окрестности текущей точки, более точную формулу для замены производной, которая будет изотропна и симметрична относительно текущей точки.

Этим требованиям отвечает фильтр Собела с размерами масок 3×3 и 5×5. Маски фильтра Собела 3×3 имеют вид

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}; \qquad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

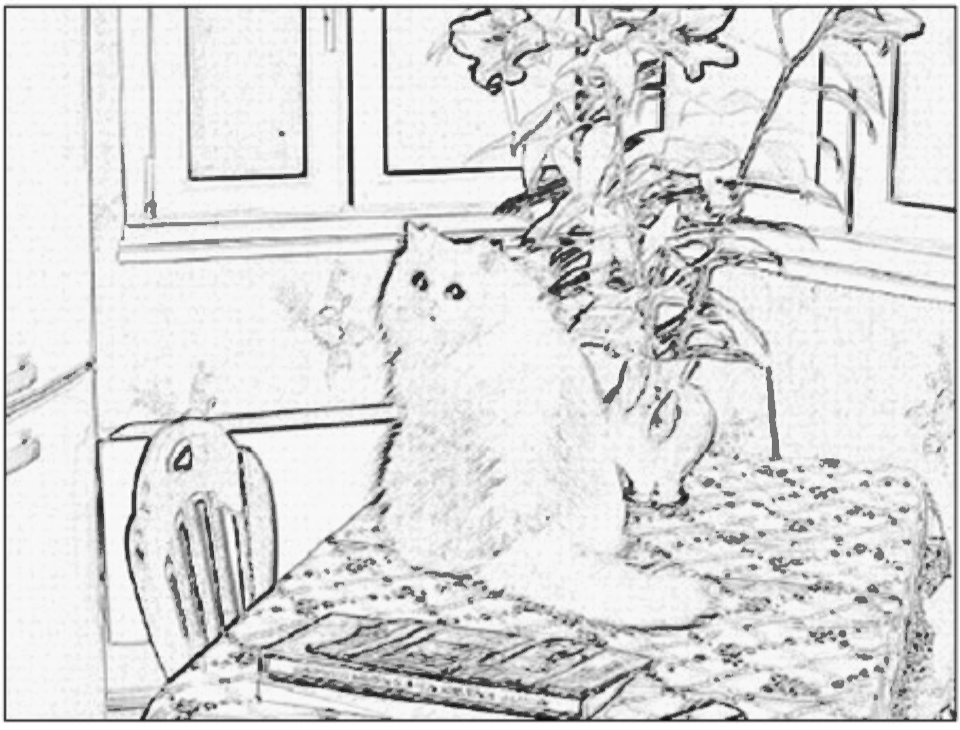


Рис. 3.3. Выделение границ с помощью простейшего фильтра

Рис. 3.4. Выделение границ с помощью фильтра Собела 3×3

Тогда формулы для вычисления производной будут таковы:

$$\frac{\partial I(x,y)}{\partial x} \approx \sum_{\Delta x=-1}^{1} \sum_{\Delta y=-1}^{1} I[x+\Delta x, y+\Delta y] S_x[\Delta x, \Delta y];$$

$$\frac{\partial I(x,y)}{\partial y} \approx \sum_{\Delta x=-1}^{1} \sum_{\Delta y=-1}^{1} I[x+\Delta x, y+\Delta y] S_y[\Delta x, \Delta y].$$

Пример выделения границ с помощью фильтра Собела 3×3 приведен на рис. 3.4.

Возможны фильтры Собела и большего размера — 5×5, 7×7, но они используются реже из-за большего объема вычислений. Маски фильтра Собела 5×5 имеют вид

$$S_x = \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \\ -4 & -8 & 0 & 8 & 4 \\ -6 & -12 & 0 & 12 & 6 \\ -4 & -8 & 0 & 8 & 4 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix}; \quad S_y = \begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}.$$

Обработка границ

Если рассматривать амплитуды градиента яркости в различных точках, то можно заметить, что он будет больше нуля не только в точках границы, но и в точках фона или объекта, которые границами не являются (см. рис. 3.4). Это обусловлено небольшой разностью освещенности соседних точек, наличием у объекта текстуры и мелких деталей, влиянием шума и помех. Кроме того, ширина граничной области может составлять несколько пикселей, особенно для недостаточно резких границ. Тогда возникает задача отделить настоящие точки границ от остальных.

Цель этапа обработки границ — пометить точки изображения как принадлежащие либо не принадлежащие границам объекта (бинаризация границ) и уменьшить толщину контура до единицы (рис. 3.5).

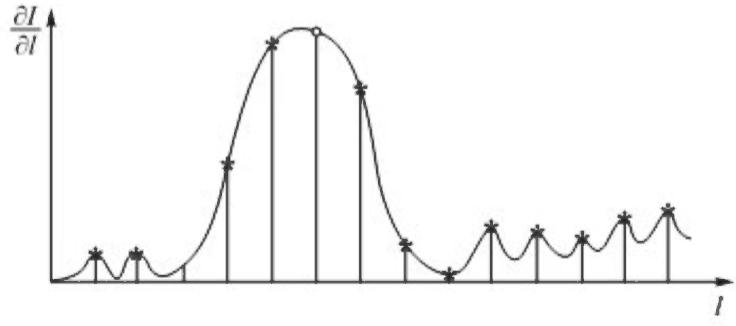


Рис. 3.5. Задача обработки границ:
* — точки, подлежащие удалению; ∘ — точка, которая должна остаться

Будем считать, что для каждой точки изображения известен вектор градиента яркости, найденный, например с помощью фильтра Собела.

Уменьшение толщины границы. Простейший способ уменьшить толщину границы — применить морфологический алгоритм типа «пожар в лесу», рассмотренный ранее, однако данный алгоритм многопроходный и работает достаточно медленно. Более быстрый способ заключается в использовании принципа немаксимального подавления.

Рассмотрим сечение изображения, проходящее через анализируемую точку в направлении градиента яркости (рис. 3.6). Из рисунка видно, что если анализируемая точка является центральной точкой границы, то амплитуда градиента яркости образует локальный максимум — точки вокруг нее имеют меньшую амплитуда градиента. Если точка не является центральной, то в ее окрестности должны быть точки с большей амплитудой градиента, которые необходимо удалять.

Однако этот принцип будет справедлив только в сечении градиента яркости. Если рассмотреть двумерный профиль градиента яркости (рис. 3.7), то очевидно, что в окрестностях центральной точки границы будут присутство-

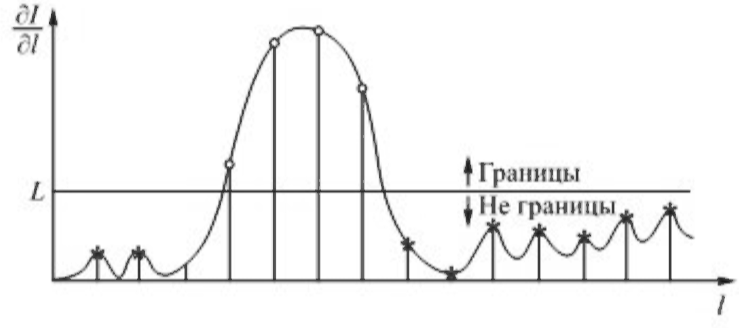


Рис. 3.6. Принцип бинаризации границ:
* — точки, подлежащие удалению; ∘ — точки, которые должны остаться

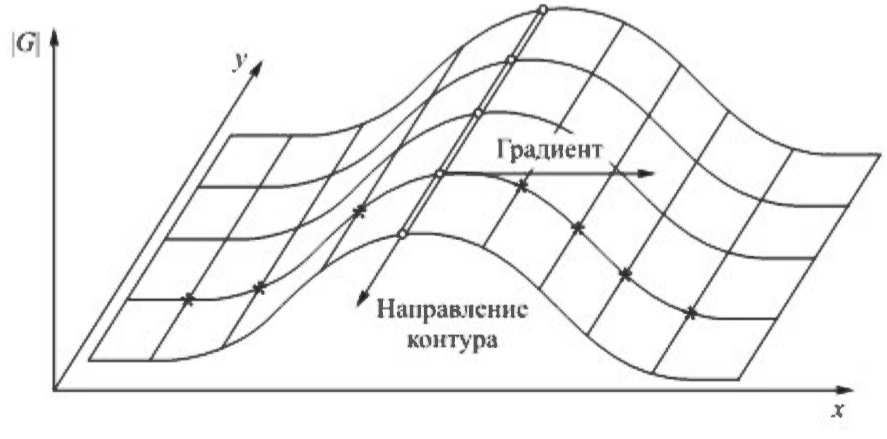


Рис. 3.7. К пояснению принципа немаксимального подавления (точки вдоль градиента подлежат подавлению, точки вдоль контура — нет):
* — точки, подлежащие удалению; ∘ — точки, которые должны остаться

вать как точки с меньшей амплитудой градиента (в направлении градиента), так и с примерно одинаковой или даже большей (в направлении контура). Таким образом, для поиска центральных точек границы необходимо анализировать только точки, лежащие в направлении градиента яркости.

Также необходимо отметить, что возможна ситуация, когда две соседние точки границы имеют равный градиент. Такая ситуация часто возникает на искусственно созданных изображениях. Чтобы получить контур единичной толщины, необходимо добавить правило, удаляющее одну из этих равноценных точек. Например, можно удалять точку с меньшей координатой x, а при их равенстве — с меньшей координатой y.

Алгоритм немаксимального подавления можно записать следующим образом.

> Для каждой точки изображения (x, y), $x = 2, \ldots,$ ширина $- 2$, $y = 2, \ldots,$ высота $- 2$ с известным градиентом яркости $\vec{G}$:
>
> по углу направления градиента яркости выбрать соседние точки в сечении градиентом (например, взять их из таблицы);
>
> если хоть в одной из этих соседних точек амплитуда градиента яркости больше, чем в текущей, то текущую точку пометить как не содержащую границ (разметку выполнить в отдельном массиве);
>
> если хотя бы в одной из этих соседних точек амплитуда градиента яркости равна амплитуде в текущей точке и соседняя точка лежит выше либо правее текущей, то текущую точку пометить как не содержащую границ (разметку выполнить в отдельном массиве);
>
> если ни одно из этих условий не выполнено, текущую точку пометить как точку границы.

Алгоритм может быть применен только к точкам, отстоящим на две точки от края изображения, поскольку для крайних точек нельзя подсчитать градиент, а для их соседних точек нельзя применить алгоритм немаксимального подавления, поскольку для этого нужно знать градиент в крайних точках. Разметка точек выполняется в отдельном массиве.

Для упрощения алгоритма список соседних точек для каждого направления градиента рассчитывается заранее и заносится в таблицу. Алгоритм выполняется за один проход и не требует сложных вычислительных операций.

Результаты работы алгоритма немаксимального подавления показаны на рис. 3.8.

Бинаризация границ. Задача бинаризации границ — удалить точки с незначительной амплитудой градиента яркости, не являющиеся точками границ. Поскольку порог бинаризации границ не зависит от яркости изображения, а зависит только от контрастности, которая, как правило, меняется существенно меньше, для большинства задач порог бинаризации границ может быть выбран постоянным (2...4 %, т. е. 5...10 единиц, если максимальное значение амплитуды градиента нормировано в область 0...255).

Если выбрать статический порог бинаризации невозможно, то его значение можно привязать к средней амплитуде градиента яркости либо к желаемому числу точек границ. При привязке порога к средней амплитуде его устанавливают в области 110...120 % средней амплитуды. Средняя ам-

Рис. 3.8. Результаты работы алгоритма немаксимального подавления

плитуда рассчитывается по всем точкам изображения. Гистограмма амплитуды градиента носит, как правило, существенно одномодальный характер (рис. 3.9).

При отборе заданного числа точек сначала выполняется алгоритм немаксимального подавления и по найденным центральным точкам границ строится гистограмма амплитуды градиента. Затем моды гистограммы суммируются от бóльших значений к меньшим до тех пор, пока их сумма не достигнет заданного значения. Номер моды, на которой достигнуто заданное число точек, равен искомому значению порога.

Рис. 3.9. Гистограмма амплитуды градиента яркости

Желаемое число точек границы, как правило, может быть найдено из соображений быстродействия. Если это невозможно, то следует использовать эмпирическое «правило 13 %»: *изображение средней детализации содержит примерно* (13±1) *% точек границ*. Это правило также полезно помнить при оценке числа точек границы для определения времени работы алгоритмов обработки границ.

Алгоритмы бинаризации и немаксимального подавления применяются к точкам границ одновременно: чтобы стать точкой границы, точка должна удовлетворять обоим условиям.

Рис. 3.10. Результат обработки границ (бинаризация и уменьшение толщины)

Результаты работы бинаризации границ и немаксимального подавления показаны на рис. 3.10.

Выделение линий

Прежде всего необходимо отметить, что прямые линии на изображении могут проходить только через границы изображения. Внутри самого объекта, а также на фоне линий, как правило, нет. Единственное исключение — случай, когда объект является протяженным (например, провода, треки частиц и т. д.). В этом случае прямая будет проходить не по границам, а параллельно им и между ними.

Таким образом, детектирование прямых всегда начинается с этапа получения и обработки точек границ, который был рассмотрен ранее.

Пусть имеется набор точек границы. Для каждой точки границы известны ее координаты и градиент яркости. Необходимо найти прямые, проходящие через заданное число точек границ.

Наиболее простой и очевидный алгоритм заключается в просмотре комбинаций пар точек. Каждая пара точек однозначно задает прямую, и после расчетов ее параметров легко определить, через какое число точек пройдет прямая. Однако число возможных пар точек на изображении очень велико (если точки контура составляют 13 % точек изображения, то число пар равно $0{,}017N^4$, где N — размер стороны изображения), и такой алгоритм будет в вычислительном отношении неэффективным. Следует построить алгоритм, который обладал бы линейной сложностью относительно числа точек границ.

Выделение прямых с помощью алгоритма Хафа. Рассмотрим уравнение прямой, проходящей через точку границы $A(x_i, y_i)$:

$$y_i = kx_i + b,$$

где k, b — параметры искомой прямой.

В этом уравнении x_i и y_i являются константами, а k и b — неизвестными. Запишем это уравнение в другом виде:

$$b = -x_i k + y_i.$$

Легко убедиться, что это уравнение некоторой прямой, но в других координатах — k и b, а параметрами прямой являются $-x_i$ и y_i.

Введем в рассмотрение пространство параметров (ПП) прямой $<k, b>$. В соответствии с приведенными выше формулами каждой точке пространства $<k, b>$ соответствует прямая в пространстве координат изображения $<x, y>$, а каждой точке границы $A(x_i, y_i)$ в пространстве координат изображения — прямая в пространстве параметров прямой $<k, b>$. Эти два пространства являются *дуальными* (рис. 3.11).

Предположим, что несколько точек границы лежат на одной прямой. Каждая из этих точек в ПП прямой будет порождать свою прямую. Однако все эти прямые должны пересечься в точке $<k^*, b^*>$, где k^*, b^* — параметры прямой, проходящей через все рассматриваемые точки (см. рис. 3.11). Таким образом, для обнаружения прямых на изображении для каждой точки границы необходимо построить соответствующую прямую в ПП прямых, а затем найти точки пересечения существенного числа таких прямых.

Для представления ПП прямых введем двумерный массив счетчиков (рис. 3.11, *в*) с индексами $[k, b]$. Каждая ячейка будет хранить число прямых, проходящих через заданную точку (k, b). Тогда алгоритм поиска прямых будет выглядеть следующим образом.

1. Для каждой точки границы с координатами $A(x_i, y_i)$:
   построить прямую $b = -x_i k + y_i$ в пространстве $<k, b>$;
   для каждого возможного значения k вычислить соответствующее значение b и увеличить счетчик $[k, b]$ на единицу.
2. Для каждой ячейки $[k, b]$ массива счетчиков:
   если значение счетчика $[k, b]$ больше порогового, то занести прямую с параметрами k, b в список найденных прямых.

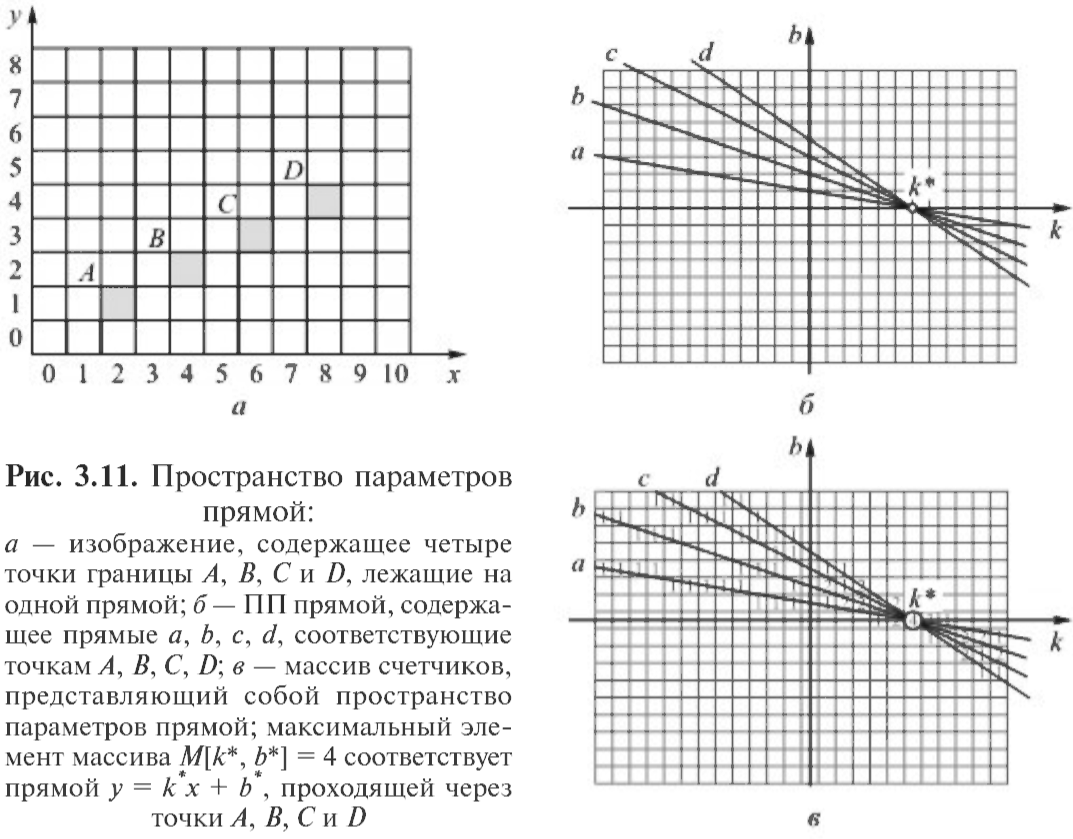
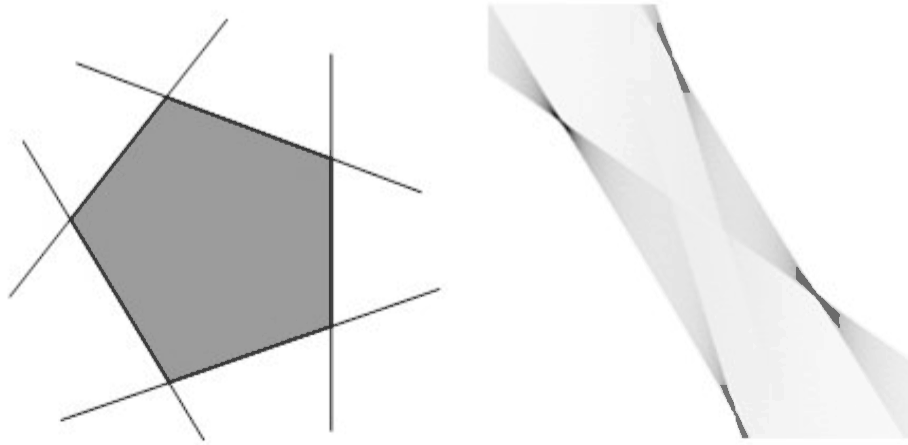


Рис. 3.11. Пространство параметров прямой:
а — изображение, содержащее четыре точки границы *A*, *B*, *C* и *D*, лежащие на одной прямой; *б* — ПП прямой, содержащее прямые *a*, *b*, *c*, *d*, соответствующие точкам *A*, *B*, *C*, *D*; *в* — массив счетчиков, представляющий собой пространство параметров прямой; максимальный элемент массива $M[k^*, b^*] = 4$ соответствует прямой $y = k^* x + b^*$, проходящей через точки *A*, *B*, *C* и *D*

Под пороговым значением здесь понимается минимально допустимое число точек границы, через которые проходит прямая. Оно может быть задано заранее или рассчитано, например, таким образом, чтобы находить заданное число прямых.

Такой алгоритм называется *алгоритмом* (*преобразованием*) *Хафа* (Hough Transform, HT). Пример работы алгоритма Хафа представлен на рис. 3.12



а *б*

Рис. 3.12. Пример работы алгоритма Хафа:
а — изображение с найденными прямыми; *б* — пространство параметров, где хорошо видны четыре локальных максимума, соответствующие четырем прямым на изображении

Недостатком алгоритма Хафа является невозможность выделения вертикальных линий: для них значение k уходит в бесконечность, а b не определено. Чтобы этого не происходило, необходимо отдельно детектировать горизонтальные прямые (с углом наклона до 45°) в пространстве $<x, y>$ и отдельно — вертикальные (с углом наклона более 45°) в пространстве $<y, x>$. Такой алгоритм позволяет детектировать линии любой ориентации, однако при этом выделяемые линии оказываются неравноценными: точность описания линий с меньшим углом наклона выше, чем с большим. Пространство поиска $<k, b>$ оказывается *анизотропным*, т. е. различные направления в нем неравноценны.

Использование нормальной формы прямой. Для нахождения изотропного ПП прямой воспользуемся *нормальной формой* параметризации прямой, которая предусматривает определение прямой через ее *вектор нормали* (ρ, φ):

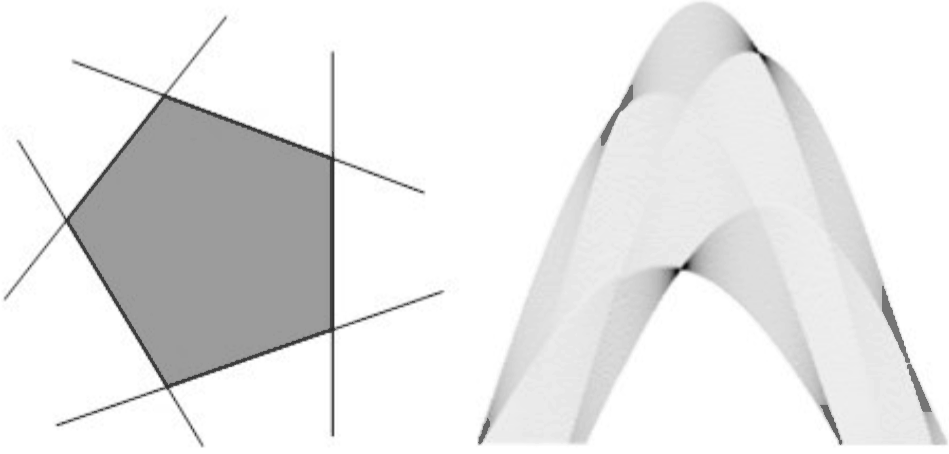
$$\rho = x \cos \varphi + y \sin \varphi,$$

где ρ — длина нормали; φ — угол наклона нормали; x, y — координаты точки, принадлежащей прямой.

Вместо пространства $<k, b>$ будем рассматривать пространство $<\rho, \varphi>$. Алгоритм детектирования прямых изменится и примет следующий вид.

1. Для каждой точки границы с координатами $A\ (x_i, y_i)$:
   построить синусоиду $\rho = x_i \cos \varphi + y_i \sin \varphi$ в пространстве $<\rho, \varphi>$;
   для φ от 0 до 360° вычислить соответствующее значение $\rho(\varphi)$ и увеличить счетчик $H[\rho, \varphi]$ на единицу.
2. Для каждой ячейки $H[\rho, \varphi]$ массива счетчиков:
   если значение счетчика $H[\rho, \varphi]$ больше порогового, то занести прямую с параметрами ρ, φ в список найденных прямых.

Результаты работы алгоритма показаны на рис. 3.13, из которого видно, что алгоритм сходен с предыдущим, но вместо прямых в ПП строятся синусоиды.



а *б*

Рис. 3.13. Пример работы алгоритма в нормальной форме:
а — изображение с найденными прямыми; *б* — пространство параметров, где хорошо видны пять локальных максимумов, соответствующие пяти прямым на изображении

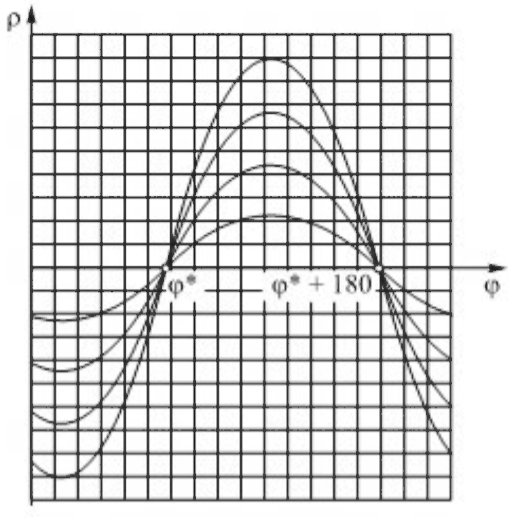


Рис. 3.14. Пространство параметров нормалей к прямым

Если шаг сетки $\Delta\varphi$ (в радианах) больше шага сетки $\Delta\rho$, то генерируемые алгоритмом синусоиды не будет сплошными и могут не пересекаться. В этом случае следует аппроксимировать соседние точки синусоиды отрезками прямых.

Из рис. 3.14 видно, что одной прямой на изображении в пространстве $<\rho, \varphi>$ будут соответствовать два максимума: (ρ, φ) и $(-\rho, \varphi+180°)$, один из которых следует исключить. Для этого отбросим отрицательные значения ρ или будем изменять φ от 0 до 180°.

Максимальное значение ρ_{max} для верхней границы массива счетчиков найти достаточно легко: это максимально возможное расстояние до прямой на изображении. Расстояние будет максимальным, если прямая проходит через самый дальний угол изображения. Таким образом, ρ_{max} равно расстоянию до дальнего угла изображения.

Вычисление синусов и косинусов в данном алгоритме проводится заранее, значения с требуемым шагом переводятся в формат с фиксированной точкой и заносятся в таблицу.

Объем вычислений в этом алгоритме равен $2NK$ целочисленных умножений, где N — число точек границы; K — число шагов по углу φ. Таким образом, данный алгоритм требует минимум в 2 раза больше операций умножения, чем предыдущий. Однако требуемое количество градаций по φ существенно меньше количества градаций по k в предыдущем алгоритме.

Уменьшение объема вычислений путем учета направления градиента яркости. Вектор градиента яркости обладает весьма полезным свойством — он всегда перпендикулярен направлению контура в данной точке. Отсюда вектор градиента и вектор нормали прямой должны быть параллельны (примерно параллельны, поскольку существует погрешность в вычислении направления градиента). Следовательно, необходимо учитывать не все значения угла нормали φ, а лишь те, которые отвечают линиям с заданным градиентом.

Пусть имеется точка $A(x, y)$ с градиентом яркости $\vec{G} = (g_x, g_y)$, через которую можно провести одну и только одну прямую l, перпендикулярную $\vec{G}$ (рис. 3.15). Обозначим координаты вектора нормали к прямой l как (x_0, y_0). Тогда можно записать:

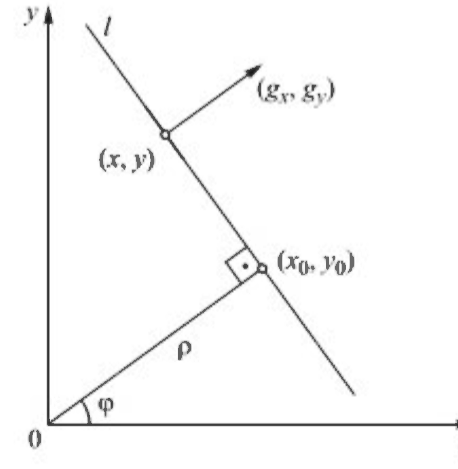


Рис. 3.15. Схема вычисления параметров нормали к линии

1) вектор нормали параллелен вектору градиента: $(x_0, y_0) \parallel (g_x, g_y)$;

2) вектор градиента параллелен любому ненулевому вектору, лежащему на прямой l, поэтому

$$(x - x_0, y - y_0) \perp (g_x, g_y).$$

Отсюда

$$x_0/g_x = y_0 g_y \Rightarrow g_y x_0 + g_x y_0 = 0;$$

$$(x - x_0)\, g_x + (y - y_0) g_y = 0 \Rightarrow g_x x_0 + g_y y_0 = g_x x + g_y y.$$

Получим систему линейных уравнений относительно (x_0, y_0):

$$\begin{cases} g_y x_0 - g_x y_0 = 0; \\ g_x x_0 + g_y y_0 = g_x x + g_y y, \end{cases}$$

определитель которой

$$\Delta = g_x^2 + g_y^2.$$

Система не имеет решения только тогда, когда точка (x, y) имеет нулевой градиент яркости и не является точкой границы, но такие точки в данном случае заведомо исключены.

Решение системы:

$$x_0 = \frac{g_x (g_x x + g_y y)}{g_x^2 + g_y^2};$$

$$y_0 = \frac{g_y (g_x x + g_y y)}{g_x^2 + g_y^2}.$$

Может показаться, что удобнее использовать пространство $<x_0, y_0>$ вместо $<\rho, \varphi>$. Однако это пространство не является изотропным и имеет сингулярность вблизи начала координат. Поэтому использовать его не следует.

Если x_0 и y_0 одновременно не равны нулю, то их можно использовать для расчета ρ и φ:

$$\rho = \sqrt{x_0^2 + y_0^2} = \frac{|g_x x + g_y y|}{\sqrt{g_x^2 + g_y^2}};$$

$$\varphi = \operatorname{arctg} \frac{y_0}{x_0} = \operatorname{arctg} \frac{g_x}{g_y}.$$

Если $g_x x + g_y y < 0$, то к φ следует добавить 180°. Если $g_x x + g_y y = 0$, то x_0 и y_0 равны нулю одновременно. Такой вектор нормали задает бесконечно

много прямых, проходящих через центр координат. Однако в этом случае можно рассчитать ρ и φ, зная, что нормаль параллельна градиенту:

$$\rho = 0;$$

$$\varphi = \operatorname{arctg}\frac{g_x}{g_y}.$$

Значения $\frac{g_x}{\sqrt{g_x^2 + g_y^2}}$, $\frac{g_y}{\sqrt{g_x^2 + g_y^2}}$ и $\operatorname{arctg}\frac{g_x}{g_y}$ могут быть рассчитаны заранее и затабулированы для всех возможных значений (g_x, g_y). В этом случае для вычисления ρ и φ потребуется две операции целочисленного умножения.

Алгоритм детектирования прямых с использованием градиента выглядит следующим образом.

1. Для каждой точки границы A с координатами (x_i, y_i) и ненулевым градиентом $\vec{G} = (g_x, g_y)$:
   вычислить

$$\rho = \frac{g_x x + g_y y}{\sqrt{g_x^2 + g_y^2}};$$

$$\varphi = \operatorname{arctg}\frac{g_x}{g_y};$$

   если $\rho < 0$, то $\rho = -\rho$, $\varphi = \varphi + 180°$;
   увеличить счетчик $H[\rho, \varphi]$ на единицу.
2. Для каждой ячейки $H[\rho, \varphi]$ массива счетчиков:
   если значение счетчика $H[\rho, \varphi]$ больше порогового, то занести прямую с параметрами ρ, φ в список найденных прямых.

Если обратный корень и арктангенс заданы таблично, то такой алгоритм потребует всего $2N$ операции целочисленного умножения, где N — число точек границ, что намного быстрее, чем предыдущие алгоритмы. Однако он имеет и недостаток: из-за неточного определения градиента локальные максимумы в пространстве поиска будут размываться, и надежность обнаружения прямых снизится. Потребуется дополнительная обработка для надежного обнаружения и восстановления максимумов.

Использование соседних точек для уточнения градиента. Точность определения параметров нормали можно значительно повысить, если отказаться от использования градиента и применять реально существующие точки границ в окрестности анализируемой точки.

Пусть имеется точка $A(x, y)$ с градиентом яркости $\vec{G} = (g_x, g_y)$, направление которого известно с точностью ε. Тогда точки прямой l, проходящей через точку $A(x, y)$, следует искать в направлении $\alpha = \arg\vec{G} - 90° \pm \varepsilon$ (рис. 3.16).

Точки непосредственно вблизи $A(x, y)$ для уточнения градиента использовать невозможно: они дадут слишком большую погрешность, превышающую ε. Точки на расстоянии одной клетки дадут погрешность ±22,5°, на рас-

Рис. 3.16. Схема поиска соседних точек прямой

стоянии двух — 11,25°, на расстоянии трех — 5,625°, на расстоянии четырех — 2,8°. Типичная погрешность вычисления направления градиента с помощью фильтра Собела 3×3 составляет примерно 5°. Поэтому использовать точки на расстоянии менее четырех клеток не имеет смысла, так же как и точки, значительно удаленные от $A(x, y)$ — они, скорее всего, принадлежат другим объектам. Дальнюю границу области поиска можно оценить, исходя из предполагаемого расстояния между объектами на изображении, а также из соображений производительности. Обычно дальнюю границу можно задать в пределах 7–10 точек. Точки в области поиска также должны иметь градиент, близкий к $\vec{G} \pm \varepsilon$.

Смещения координат точек в области поиска вычисляются заранее и заносятся в строки таблицы, отвечающие соответствующему направлению градиента. Теперь вместо значений (g_x, g_y) можно использовать их более точный аналог — восстановленный градиент (g'_x, g'_y):

$$g'_x = y - y_n;$$
$$g'_y = x_n - x,$$

где x_n, y_n — координаты найденной соседней точки.

Получим следующий алгоритм.

1. Для каждой точки границы A с координатами (x_i, y_i) и ненулевым градиентом $\vec{G} = (g_x, g_y)$:

   перебрать соседние точки из области поиска:

   $x_n = x + \Delta x[i, \arg \vec{G}\,]$;

   $y_n = y + \Delta y[i, \arg \vec{G}\,]$;

   если (x_n, y_n) — точка границы с направлением градиента, равным $\alpha = \arg \vec{G} \pm \varepsilon$, то:

   вычислить

   $g'_x = y - y_n$;

   $g'_y = x_n - x$;

   $$\rho = \frac{g'_x x + g'_y y}{\sqrt{g_x^2 + g_y^2}};$$

$$\varphi = \operatorname{arctg} \frac{g'_x}{g'_y};$$

если $\rho < 0$, то $\rho = -\rho$, $\varphi = \varphi + 180°$;

увеличить счетчик $H[\rho, \varphi]$ на единицу.

2. Для каждой ячейки $H[\rho, \varphi]$ массива счетчиков:

если значение счетчика $H[\rho, \varphi]$ больше порогового, то занести прямую с параметрами (ρ, φ) в список найденных прямых.

Данный алгоритм существенно точнее и надежнее предыдущего, однако объем вычислений также возрастает пропорционально длине области поиска.

Недостаток этого алгоритма: значение счетчика $H[\rho, \varphi]$ не равно числу точек, принадлежащих прямой, а равно сумме числа пар точек, найденных в области поиска. Это может затруднять поиск нужных линий, если линии сильно различаются по качеству.

Тем не менее пользоваться следует именно этим алгоритмом, поскольку он дает самый надежный результат при приемлемом объеме вычислений.

Детектирование отрезков

На практике полезнее иметь представление объекта не в виде прямых, а в виде отрезков: отрезки более локализованы в пространстве, и можно легко отделить отрезки, принадлежащие объекту и не принадлежащие ему. В случае с прямыми это сделать затруднительно.

Отрезок, в отличие от прямой, будет иметь уже четыре параметра вместо двух. Например, это могут быть координаты концов отрезка. Однако такая параметризация, хотя и является наиболее удобной, не позволяет строить алгоритмы поиска, подобные рассмотренным выше.

Классическая схема поиска отрезков. Классическая схема поиска отрезков с помощью преобразования Хафа предусматривает наличие следующих параметров (рис. 3.17):

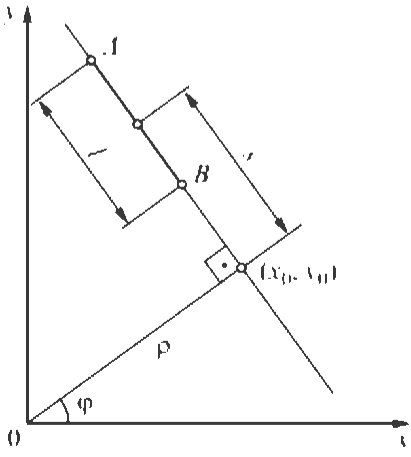


Рис. 3.17. Параметры отрезка в классической схеме поиска отрезков

ρ, φ — длина и угол наклона нормали к прямой, проходящей через отрезок;

l — длина отрезка;

s — расстояние от центра отрезка до конца вектора нормали.

Недостатки подобной схемы очевидны.

Во-первых, она весьма чувствительна к точности определения угла нормали φ: небольшие неточности его определения ведут к весьма существенному смещению отрезка относительно его реального положения (рис. 3.18) для больших значений s.

Во-вторых, необходимо иметь четырехмерное пространство параметров, которое, будучи представлено четырехмерным массивом счетчиков, потребует огромных объемов

расчетов и соответствующих затрат времени на его обслуживание. Даже небольшое количество в 256 градаций по каждому из четырех измерений исчерпывают лимит в 4 Гбайт для 32-разрядных систем.

Поиск отрезков с использованием расширенного пространства поиска. Чтобы избавиться от описанных выше недостатков, разобьем параметры отрезка на идентификационные и дескриптивные (описательные). Идентификационные параметры служат для того, чтобы отличать одни отрезки от других, и в данном случае схема <ρ, φ, *l*, *s*> вполне пригодна. Дескриптивные параметры применяют для наиболее точного описания параметров отрезка. Здесь удобно использовать координаты концов отрезка и аналогичные признаки, дающие высокое качество.

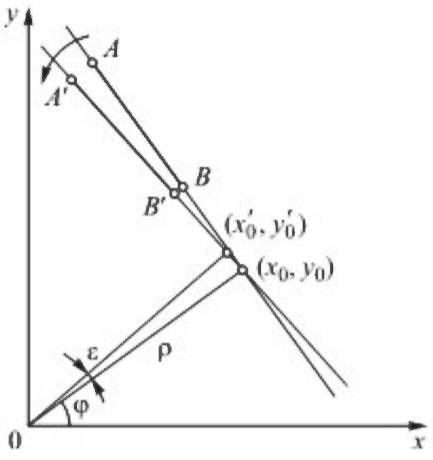


Рис. 3.18. Отклонение отрезка от реального положения при наличии погрешности определения параметров нормали

Можно ли уменьшить число идентификационных параметров? Факт, что не все комбинации отрезков могут реально существовать на изображении, позволяет так считать. Отрезки *AD* и *CB* на рис. 3.19 в действительности образуют один отрезок.

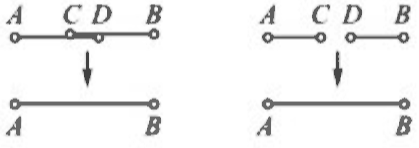


Рис. 3.19. Пояснение к алгоритму детектирования отрезков

Предположим для начала, что на одной прямой может лежать только один отрезок. В этом упрощенном случае отрезок может быть однозначно идентифицирован прямой, проходящей через него. Тогда идентификационные параметры отрезка можно сократить с четырех до двух, например до параметров нормали <ρ, φ>.

Остальные параметры, необходимые для полного описания отрезка, отнесем к дескриптивным. Для их учета и накопления расширим ячейки счетчика: пусть каждый счетчик теперь содержит не только число точек в отрезке, но и дополнительную дескриптивную информацию, например максимальную и минимальную координаты *x* и *y* отрезка (рис. 3.20). Получим *расширенное* пространство поиска: оно по-прежнему двумерно, но имеет более сложную структуру из-за учета дескриптивных параметров. Помимо максимальных и минимальных координат расширенное пространство поиска может хранить и другую информацию, например направление градиента вдоль отрезка.

Необходимо отметить, что максимальная и минимальная координаты — это не концы отрезка (рис 3.21), а координаты минимальной по размеру прямоугольной области, содержащей отрезок. Из рис. 3.21 видно, что координаты углов нижнего левого и верхнего правого углов этой области не обязательно совпадают с концами отрезков.

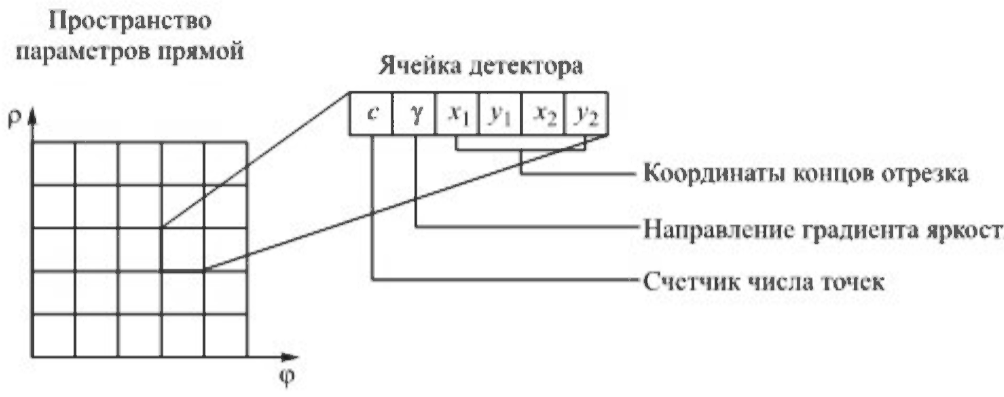


Рис. 3.20. Расширенное пространство поиска

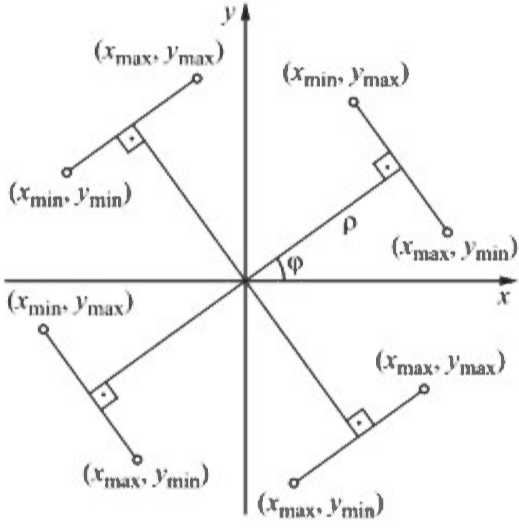


Рис. 3.21. К пояснению принципа коррекции концов отрезка

Однако получить координаты концов отрезков не составляет труда. Если угол φ принадлежит первой или третьей четверти, то для получения координат концов отрезка нужно поменять местами y_{min} и y_{max}.

Возможна ситуация, когда из-за эффекта квантования $\varphi = \pm 90°$, а $x_{min} \neq x_{max}$. При этом нормаль может быть ошибочно отнесена как к четным, так и нечетным четвертям. В этом случае следует брать значения x_{min} и x_{max} одинаковыми и равными среднему значению:

$$x_{ср} = (x_{min} + x_{max})/2.$$

Аналогично необходимо поступить и с граничными углами $\varphi = 0$ и $\varphi = 180°$ для y_{min} и y_{max}.

Алгоритм поиска в расширенном пространстве поиска можно записать в следующем виде.

1. Для каждой точки границы A с координатами (x_a, y_a) и градиентом $\bar{G} = (g_x, g_y)$:
   рассчитать <ρ, φ>;
   найти ячейку $H[\rho, \varphi]$;
   увеличить счетчик: $H[\rho, \varphi] = H[\rho, \varphi] + 1$;
   скорректировать концы отрезка:
   $x_{min}[\rho, \varphi] = \min(x_{min}[\rho, \varphi], x_a)$;
   $y_{min}[\rho, \varphi] = \min(y_{min}[\rho, \varphi], y_a)$;
   $x_{max}[\rho, \varphi] = \max(x_{max}[\rho, \varphi], x_a)$;
   $y_{max}[\rho, \varphi] = \max(y_{max}[\rho, \varphi], y_a)$.
2. Найти локальные максимумы $h^* = H[\rho^*, \varphi^*]$, соответствующие наиболее длинным и четким отрезкам;
   если φ^* принадлежит первой или третьей четверти, то $x_{min}[\rho, \varphi]$ и $x_{max}[\rho, \varphi]$ поменять местами.

Очевидно, что алгоритм является модификацией алгоритма поиска прямых. Добавляются только операции сравнения концов, удельный вес которых в общей доле вычислений достаточно мал.

Сепарация коллинеарных отрезков при прогрессивном порядке просмотра точек. Рассмотрим случай, когда на одной прямой могут лежать несколько отрезков и каждый из них должен быть найден отдельно. Если задан определенный порядок просмотра точек границ, например слева направо и сверху вниз, то будут просмотрены все точки одного отрезка и лишь затем — другого, независимо от взаимного расположения отрезков в пространстве (рис. 3.22). Это позволяет выделять отрезки последовательно, но такой прием не будет работать при параллельном выполнении алгоритма.

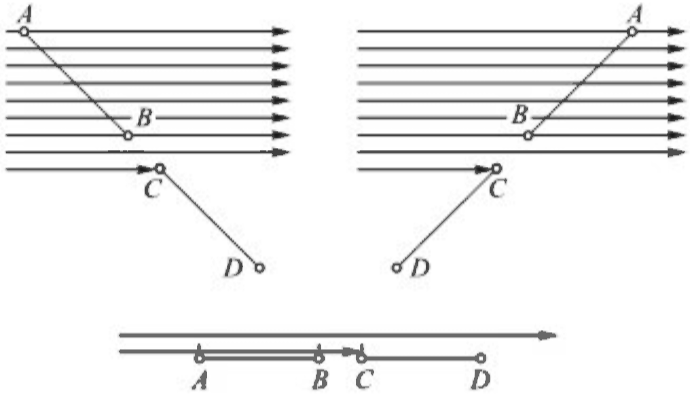


Рис. 3.22. Сепарация отрезков при прогрессивном порядке просмотра точек границ

Алгоритм выделения отрезков в этом случае модифицируется и будет выглядеть следующим образом.

1. Для каждой точки границы A_i с координатами (x_A, y_A) и градиентом $\vec{G}_A = (g_x, g_y)$:

найти параметры нормали < ρ, φ> к отрезку;

найти ячейку $H[\rho, \varphi]$, отвечающую данному отрезку;

если расстояние от A_i до ближайшей точки отрезка не превышает значения допустимого зазора L_0, считать данную точку продолжением текущего отрезка (корректировать концы отрезка и число точек в нем);

иначе считать точку A началом нового отрезка;

если число точек в отрезке $H[\rho, \varphi].C$ больше порога N, то перенести отрезок из ячейки в список найденных отрезков;

создать новый отрезок $H[\rho, \varphi]$, состоящий из одной точки A:

$H[\ \rho, \varphi\ ] = 1;$

$x_{\min}[\ \rho, \varphi\ ] = x_{\max}[\ \rho, \varphi\ ] = x_a;$

$y_{\min}[\ \rho, \varphi\ ] = y_{\max}[\ \rho, \varphi\ ] = y_a.$

2. Просмотреть пространство поиска и выписать оставшиеся существенные отрезки из пространства поиска в список отрезков.

Сравнивая этот алгоритм с предыдущим, можно заметить, что добавились операции вычисления расстояния от текущей точки до ближайшей точки отрезка, а также операции переноса отрезка в список, суммарная доля

которых в общем объеме вычислений пренебрежимо мала. Для упрощения вычислений расстояние можно рассчитать по приближенной формуле

$$L \approx (|x_0 - x_a|, |y_0 - y_a|),$$

где (x_0, y_0) — точка отрезка, ближайшая к текущей точке A.

Поскольку точка A лежит на прямой, проходящей через отрезок, то (x_0, y_0) — один из концов отрезка. Тогда

$$L = \max\left(\min\left(|x_{\min} - x_a|, |x_{\max} - x_a|\right), \min\left(|y_{\min} - y_a|, |y_{\max} - y_a|\right)\right).$$

Операцию вычисления расстояния L по этой формуле легко совместить с операцией коррекции концов отрезка. Порог L_0 выбирается исходя из минимального расстояния между объектами на изображении.

Характеристики алгоритма выделения отрезков с использованием преобразования Хафа. Рассмотренный ранее алгоритм позволяет выделять отрезки с очень высокой производительностью. Это обусловлено тем, что каждая точка границы изображения рассматривается один раз, и число выделяемых операций для нее весьма невелико — расчет [ρ, φ], коррекция концов отрезка и проверка на выход за пределы отрезка. Дополнительные операции, например разворот отрезков и их склейка, также требуют относительно небольшого объема вычислений.

Таким образом, алгоритм с использованием преобразования Хафа хорошо подходит для приложений реального времени. Его точность также довольно высока: поскольку концами отрезка являются реально существующие точки границ, то концы отрезка оказываются вычисленными с точностью позиционирования точек границы.

К недостаткам алгоритма следует отнести чрезмерную фрагментацию отрезков, которую нельзя до конца убрать даже с помощью алгоритмов склейки. В результате изображение оказывается представленным излишне большим числом существенно перекрывающихся отрезков.

Выделение прямых с помощью метода RANSAC

Метод RANSAC является популярной альтернативой выделения линий с помощью преобразования Хафа. Он не использует пространство поиска и в силу этого не имеет издержек и особенностей, связанных с его обслуживанием и обработкой. Данный метод создает временный список гипотез, которые проверяются и удаляются по мере необходимости.

Рассмотрим, как работает этот метод.

Аналогично преобразованию Хафа, последовательно просматриваются все точки границ A_i с градиентом $\vec{G}_A$. Для каждой найденной точки, исходя из направления градиента, формируется область поиска соседних точек (рис. 3.23), так же как и при преобразовании Хафа. Для каждой точки границы B_i, попавшей в эту область поиска и имеющей градиент, совпадающий по направлению с градиентом $\vec{G}_A$, строится одна либо несколько гипотез $<A_i\ B_i>$ о наличии на изображении прямой, проходящей через эти точки.

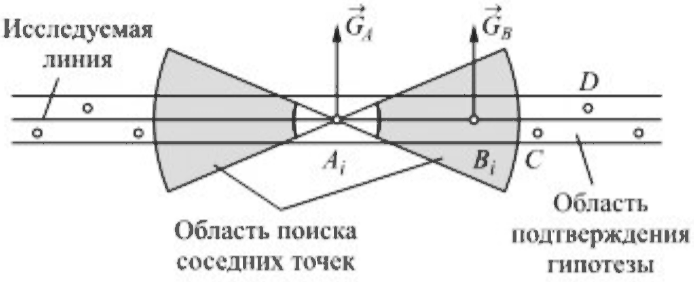


Рис. 3.23. Схема работы метода RANSAC

Чтобы подтвердить или опровергнуть гипотезу $<A_i\ B_i>$, строится еще одна область — область подтверждения гипотез (рис. 3.24). Если гипотеза верна, то в эту область должно попасть существенное число точек границ. В зависимости от реализации метода область может быть прямоугольной (для упрощения вычислений) или треугольной, так чтобы учитывать возможную погрешность вычисления направления прямой для дальних точек.

Далее находят все точки, попавшие в эту область. Если таких точек оказалось слишком мало, то гипотеза отвергается. Можно ужесточить требования к точкам области, принимая в расчет только точки с направлением градиента, совпадающим по направлению с $\vec{G}_A$.

Далее уточняют параметры прямой по координатам найденных точек. Проще и быстрее это сделать с помощью метода наименьших квадратов, однако он дает достаточно низкую точность для прямых, близких к вертикальным. Для повышения точности следует либо отдельно применять его к вертикальным и горизонтальным прямым, либо предварительно развернуть точки так, чтобы исследуемая прямая располагалась вертикально.

Уточненные параметры найденной прямой добавляются в список найденных прямых, а точки, принадлежащие ей, подлежат удалению, чтобы избежать их повторного просмотра. Это достаточно важный момент, поскольку метод требует большого числа операций для каждой пары точек $<A_i\ B_i>$. В преобразовании Хафа эти операции отсутствовали — каждая найденная пара просто увеличивала счетчик в пространстве поиска.

Построение областей поиска и построения гипотез. Область поиска соседних точек может быть построена, как и в методе Хафа, с помощью таблицы смещений: для каждого направления градиента $\vec{G}_A$ заранее рассчитывается список смещений точек области поиска относительно текущей точки A.

Для области подтверждения такой подход не работает: число точек во всех возможных областях поиска слишком велико и не может быть задано таблицей. Проверять условие попадания в область поиска для каждой точки границы нецелесообразно, поскольку их слишком много (сотни тысяч).

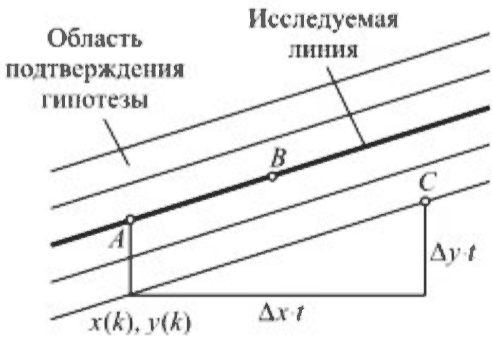


Рис. 3.24. Схема задания области подтверждения гипотез

Наиболее простой способ заключается в задании области подтверждения в виде совокупности параллельных прямых, проходящих через все точки области (см. рис. 3.24). Уравнения этих прямых можно задать в параметрическом виде:

$$x(t) = x_A + x(k) + \Delta x \cdot t;$$
$$y(t) = y_A + y(k) + \Delta y \cdot t,$$

где $x(t)$, $y(t)$ — искомые координаты точек области поиска; x_A, y_A — координаты текущей точки A; $x(k)$, $y(k)$, Δx, Δy — параметры k-й прямой; $t = 0, \pm 1, \pm 2, \pm 3, \ldots$ — свободный параметр.

Число параллельных прямых, задающих прямоугольную область поиска, очевидно, равно ширине этой области.

Поскольку одна из величин Δx, Δy должна быть меньше единицы, то ее следует представлять величиной с фиксированной или плавающей точкой. А чтобы избежать большого количества операций умножения $\Delta x \cdot t$, $\Delta y \cdot t$, эти величины можно накапливать отдельно, добавляя к ним Δx и Δy на каждой итерации цикла по t. Легко заметить, что $\Delta x \cdot t$, $\Delta y \cdot t$, являются общими для всех k прямых, которые различаются только величинами $x(k)$ и $y(k)$.

Подобно преобразованию Хафа, метод RANSAC также может быть использован для выделения отрезков. Для этого достаточно запоминать максимальные и минимальные значения координат точек границ, как это осуществлялось в преобразовании Хафа. Точно так же метод можно использовать для выделения окружностей, эллипсов и других фигур: вместо разметки откликов в пространстве поиска появится операция проверки гипотез и удаления учтенных точек.

В целом, метод RANSAC работает существенно медленнее методов с преобразованием Хафа, однако позволяет преодолеть негативные эффекты, связанные, например, с расщеплением откликов в пространстве поиска и с необходимостью соответствующей обработки.

Склейка отрезков. Отрезки, выделенные с использованием преобразования Хафа, будут сильно фрагментированы вследствие того, что небольшие смещения точек границы из-за дискретизации и шума приводят к изменениям ρ и φ, в результате чего такие точки попадают в разные ячейки счетчиков H. Как следствие, один отрезок «рассыпается» на несколько (рис. 3.25). Чтобы получить исходный отрезок из фрагментов, необходима операция склейки. Она может быть осуществлена исходя либо из параметров отрезка, либо из параметров точек границ, принадлежащих отрезку. Склейка отрезков в любом случае проходит иерархически, начиная с более длинных и важных и заканчивая более короткими и маловажными. Для этого список отрезков необходимо предварительно отсортиро-

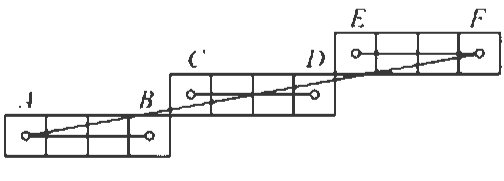


Рис. 3.25. Фрагментация отрезков: отрезок AF из-за дискретизации будет разбит на отрезки AB, CD и EF, попадающие в разные ячейки пространства поиска

вать по числу точек в отрезке. Поскольку число отрезков и их фрагментов может измеряться тысячами, для сортировки следует использовать алгоритм типа быстрой сортировки.

Геометрическая склейка отрезков. Для геометрической склейки отрезков будем просматривать пары отрезков от более длинных к более коротким. Для каждой пары отрезков (рис. 3.26) *AC* и *DB* будем вычислять новый отрезок *AB*, а также расстояние *d* между ближайшими концами отрезков и расстояние ε_i от концов отрезков до отрезка *AB*. Если эти параметры лежат в пределах допустимого, то отрезки *AC* и *DB* замещаем новым отрезком *AB*.

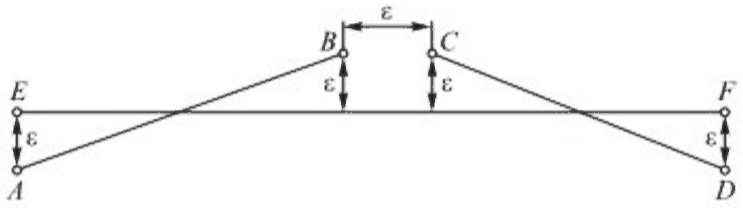


Рис. 3.26. Схема склейки отрезков

Геометрическая склейка оперирует отдельными отрезками, а не точками изображения, и требует относительно небольшого числа операций.

Верификация отрезков. Для верификации отрезков строится область верификации, отстоящая от текущего отрезка на расстояние ε (рис. 3.27).

Далее находят точки границ, попавшие в область верификации и имеющие градиент яркости, примерно перпендикулярный текущему отрезку. Эти точки ранее могли быть посчитаны как принадлежащие текущему отрезку, так и принадлежащие его фрагментам. Теперь можно уточнить общее число точек отрезка и его параметры. Если текущий отрезок набрал достаточное число точек и признан верифицированным, то его точки удаляются с изображения и больше не будут участвовать в верификации других, более коротких фрагментов. Таким образом, меньшие фрагменты не пройдут верификацию и будут удалены из списка найденных отрезков.

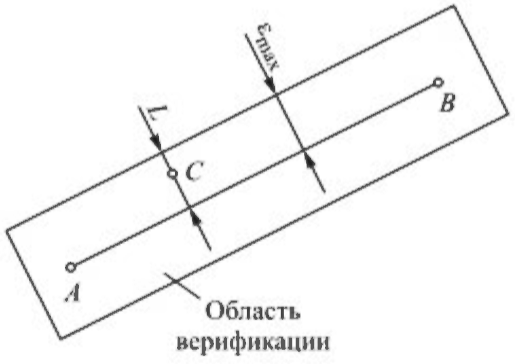


Рис. 3.27. Верификация отрезков

Процесс верификации отрезков медленнее, чем геометрическая склейка, поскольку предусматривает многократный просмотр большого числа точек границ. Однако она дает более качественный результат, так как напрямую использует положение реальных точек отрезка на изображении. Для достижения наилучшего качества геометрическую склейку и верификацию применяют совместно (сначала склейку, затем верификацию).

Выделение углов

Имея изображение в виде набора отрезков, можно достаточно легко найти угловые точки объекта как точки пересечения соответствующих пар отрезков (рис. 3.28).

Зная координаты концов отрезков AB и CD, можно найти точку их пересечения. Для этого запишем

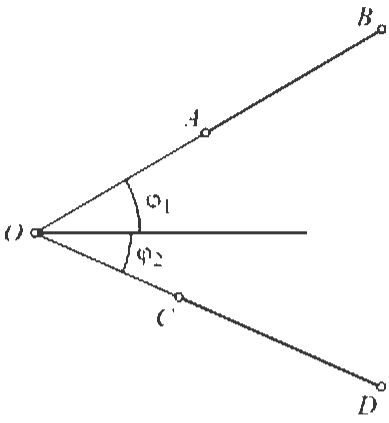


Рис. 3.28. Схема вычисления углов

$$\operatorname{tg}\varphi_1 = \frac{y_A - y_0}{x_A - x_0} = \frac{y_B - y_A}{x_B - x_A};$$

$$\operatorname{tg}\varphi_2 = \frac{y_C - y_0}{x_C - x_0} = \frac{y_D - y_C}{x_D - x_C}.$$

Обозначим $\Delta x_{BA} = x_B - x_A$, $\Delta y_{BA} = y_B - y_A$, $\Delta x_{DC} = x_D - x_C$, $\Delta y_{DC} = y_D - y_C$, тогда

$$\begin{cases} \Delta x_{BA} y_A - \Delta x_{BA} y_0 = \Delta y_{BA} x_A - \Delta y_{BA} x_0; \\ \Delta x_{DC} y_C - \Delta x_{DC} y_0 = \Delta y_{DC} x_C - \Delta y_{DC} x_0. \end{cases}$$

Перенеся x_0 и y_0 в левую часть, получим

$$\begin{cases} \Delta y_{BA} x_0 - \Delta x_{BA} y_0 = \Delta y_{BA} x_A - \Delta x_{BA} y_A; \\ \Delta y_{DC} x_0 - \Delta x_{DC} y_0 = \Delta y_{DC} x_C - \Delta x_{DC} y_C. \end{cases}$$

Обозначим

$$d_{AB} = \Delta y_{BA} x_A - \Delta x_{BA} y_A;$$

$$d_{CD} = \Delta y_{DC} x_C - \Delta x_{DC} y_C.$$

Тогда определитель системы

$$\Delta = \Delta y_{DC}\, \Delta x_{BA} - \Delta y_{BA}\, \Delta x_{DC}.$$

Определитель будет равен нулю, если один из отрезков нулевой длины или если отрезки параллельны. Такие отрезки исключим из дальнейшего расчета.

Решение системы будет следующим:

$$x_0 = \frac{\Delta x_{BA} d_{CD} - \Delta x_{DC} d_{AB}}{\Delta};$$

$$y_0 = \frac{\Delta y_{BA} d_{CD} - \Delta y_{DC} d_{AB}}{\Delta}. \qquad (3.1)$$

Поскольку число возможных пар отрезков может оказаться довольно большим, их следует предварительно отфильтровать и брать в расчет только близкорасположенные отрезки, расстояния между ближайшими концами

которых не превышают желаемого. Также, рассчитав заранее углы наклона отрезков, можно отобрать пары отрезков, дающих угол требуемых размера и ориентации.

Описанный подход имеет сразу несколько неоспоримых преимуществ:

• можно обнаруживать углы крайне низкого качества — скругленные, размытые, затененные или скрытые, поскольку метод работает с большим

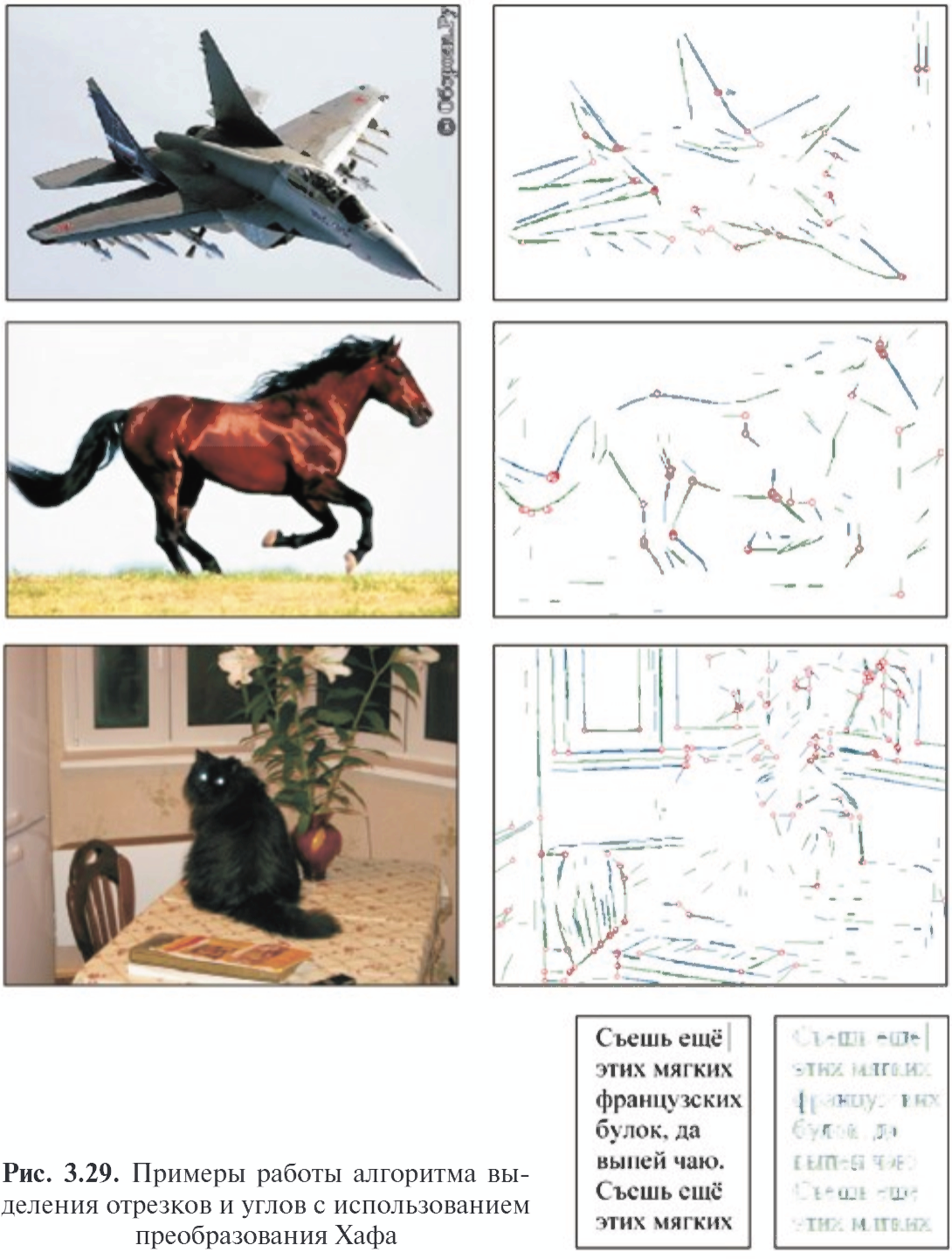


Рис. 3.29. Примеры работы алгоритма выделения отрезков и углов с использованием преобразования Хафа

числом точек, образующих стороны угла, а не с их небольшим числом вблизи острия угла, как в подходах, основанных на фильтрации;

• можно сразу отобрать углы желаемых размера, ориентации и положения, причем эти характеристики вычисляются с достаточной точностью, а также благодаря учету статистических данных от большого числа точек.

Очевидно, что метод не может применяться, когда выделение отрезков невозможно, например на диффузных или зубчатых изображениях.

Другой особенностью данного метода является большое число дублирующих друг друга углов. Они могут более или менее совпадать по размеру и положению, но каждый из них будет генерироваться своей парой отрезков. Поскольку число углов, как правило, невелико, можно просмотреть все пары углов и объединить совпадающие.

В целом, метод достаточно надежен, точен и производителен для использования в практических задачах (рис. 3.29).

Выделение окружностей

Задача выделения окружностей возникает прежде всего в приложениях технического зрения. Окружностями зачастую являются края деталей и отверстия круглой формы. Эти элементы, с одной стороны, достаточно четкие для детектирования окружностей, а с другой — их удобно использовать для решения широкого круга задач, от определения ориентации детали до измерения ее параметров и контроля качества.

За пределами области технического зрения окружности встречаются значительно реже. Окружностями могут быть, например, контуры зрачков в задачах биометрической идентификации, границы метеоритных кратеров в задачах ориентации над поверхностью планет или эритроциты в задачах медицинской диагностики.

Прежде чем приступать к разработке алгоритмов поиска окружностей, необходимо иметь в виду следующее. Большинство объектов, которые привычно считают круглыми, на изображении не дают контуров в виде окружности — они превращаются в эллипсы и овалы. Более того, даже идеальная окружность на изображении не является таковой в геометрическом смысле: ее точки оказываются привязанными к дискретной сетке изображения, и она фактически представляет собой многоугольник.

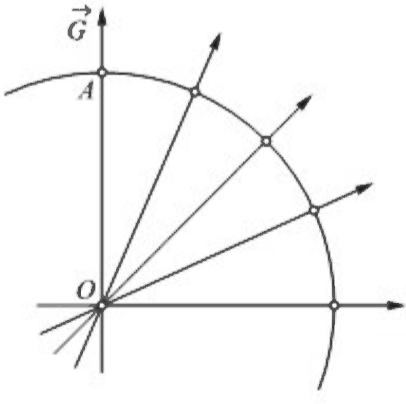


Рис. 3.30. К пояснению принципа работы простейшего алгоритма поиска окружностей

В этом смысле окружности и дуги, в отличие от отрезков, не являются универсальным аппроксимирующим контуром. Их следует использовать только в тех задачах, где они действительно есть.

Простейший алгоритм поиска окружностей. Приведенные ранее подходы с использованием преобразования Хафа можно обобщить и на более широкий класс фигур, например на окружности.

Рассмотрим наиболее простой подход, напрямую использующий преобразование Хафа. Пусть, как и прежде, имеем изображение, представленное набором точек границ $A_i\,(x_i, y_i)$ с градиентом яркости $\vec{G}_i = (g_x, g_y)$ (рис. 3.30).

Если провести прямые через точки границ в направлении градиента, то они должны пересечься в центре окружности O.

Алгоритм поиска окружностей в этом случае будет выглядеть следующим образом.

1. Для каждой точки границы $A_i\,(x_i, y_i)$ с градиентом $\vec{G}_i = (g_x, g_y)$ построить прямую в пространстве поиска, проходящую через A_i в направлении $\vec{G}_i$:
   $x_t = x_A + t\, g_x/|G|$,
   $y_t = y_A + t\, g_y/|G|$,
   $H[x_t, y_t] = H[x_t, y_t] + 1$,
   $t = 0, \pm 1, \pm 2, \ldots$.
2. В пространстве поиска найти положение локальных максимумов. Их положение соответствует положению центров искомых окружностей.

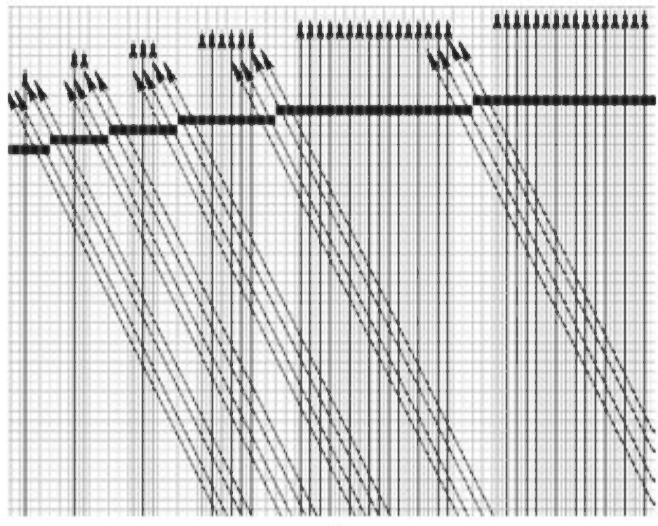
Важно отметить, что в рассматриваемом случае пространство поиска совпадает с пространством изображения, поскольку искомые параметры — это координаты центра окружности. В данном алгоритме отсутствует возможность вычисления радиуса окружности и его необходимо находить другими методами.

Однако при попытке реализовать этот алгоритм на практике обнаруживается, что он не работает! Причина в том, что направление градиента известно лишь с некоторой точностью, и небольшой ошибки достаточно, чтобы построенные прямые прошли мимо центра окружности (рис 3.31). Центральный максимум, который должен быть в точке O, размылся по сравнительно большой площади достаточно сложным образом, и его поиск затруднен.

Чтобы преодолеть возникшую сложность, необходимо учитывать точность определения градиента. Как и в случае с прямыми, ее можно повысить, используя информацию о расположении соседних точек.

а *б*

Рис. 3.31 (начало). Возникновение ошибок при поиске окружностей:
а — исходное изображение; *б* — отклики в пространстве параметров



в

Рис. 3.31 (окончание). Возникновение ошибок при поиске окружностей:
в — увеличенный фрагмент пространства параметров

Вследствие ошибок определения градиента построенные прямые более не проходят через центр *O*. Стрелками на рис. 3.31, *в* показано расчетное направление градиента в точках окружности.

Алгоритм с использованием пары точек. Для повышения точности расчета центра окружности следует находить в окрестности текущей точки *A* соседние точки *B* и использовать их для расчета положения точки *O*. Сформируем для текущей точки *A* область поиска соседних точек (рис. 3.32). Координаты смещения точек области поиска относительно точки *A* в зависимости от ее градиента рассчитываются заранее и заносятся в таблицы $\Delta x[\varphi_{Ai}]$, $\Delta y[\varphi_{Ai}]$, где φ_{Ai} — направление градиента в точке *A*; *i* — ее порядковый номер смещения, $i = 0, 1, 2, \ldots$.

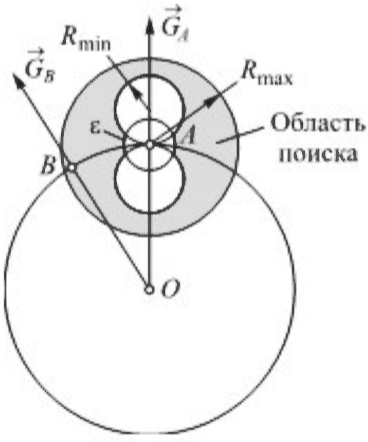


Рис. 3.32. Схема поиска окружностей по парам точек

Параметр R_{min} определяется минимальным радиусом детектируемой окружности (при малом радиусе R_{min} окружность на изображении преобразуется в многогранник и ее поиск описываемой группой методов будет затруднен).

Параметр ε определяется точностью расчета направления градиента: соседние точки, расположенные слишком близко к точке *A*, дадут большую погрешность определения центра окружности, как в случае, представленном на рис. 3.31.

Параметр R_{max} определяется максимальным радиусом детектируемой окружности, а также требованием быстродействия: с увеличением R_{max} число соседних точек и соответствующий объем перебора быстро растут, тогда как вероятность найти точку именно той же окружности, что и точка *A*, снижается.

Зная координаты точек A и B, а также их градиенты $\vec{G}_A$ и $\vec{G}_B$, легко найти положение точки O:

$$\begin{cases} OA \parallel \vec{G}_A; \\ OB \parallel \vec{G}_B \end{cases} \Rightarrow \begin{cases} \dfrac{x_O - x_A}{y_O - y_A} = \dfrac{g_{xA}}{g_{yA}}; \\ \dfrac{x_O - x_B}{y_O - y_B} = \dfrac{g_{xB}}{g_{yB}} \end{cases} \Rightarrow \begin{cases} g_{yA}x_O - g_{xA}y_O = g_{yA}x_A - g_{xA}y_A = d_A; \\ g_{yB}x_O - g_{xB}y_O = g_{yB}x_B - g_{xB}y_B = d_B, \end{cases}$$

где $d_A = g_{yA}x_A - g_{xA}y_A$; $d_B = g_{yB}x_B - g_{xB}y_B$.

Получим систему линейных уравнений относительно x_0 и y_0 с определителем

$$\Delta = g_{yB}\,g_{xA} - g_{yA}\,g_{xB}.$$

Определитель обращается в нуль, когда один из градиентов равен нулю (что невозможно для точки границы) либо когда градиенты параллельны:

$$g_{yB}\,g_{xA} - g_{yA}\,g_{xB} = 0 \Rightarrow$$

$$\Rightarrow g_{xA}/g_{yA} = g_{xB}/g_{yB} \Rightarrow$$

$$\Rightarrow (g_{xA}, g_{yA}) \| (g_{xB}, g_{yB}).$$

Соседние точки с совпадающими (или примерно совпадающими) направлениями градиентов следует исключать из расчета.

Решение системы будет иметь вид

$$x_O = \frac{d_A g_{xB} - d_B g_{xA}}{\Delta};$$
$$y_O = \frac{d_B g_{yA} - d_A g_{yB}}{\Delta}. \qquad (3.2)$$

Алгоритм выделения окружностей будет выглядеть следующим образом.

1. Для каждой точки границы A_i с координатами (x_A, y_A) и градиентом яркости $\vec{G}_A = (g_{xA}, g_{yA})$:
   построить область поиска соседних точек;
   для каждой точки границы B_j с координатами (x_B, y_B) из области поиска и градиентом яркости $\vec{G}_B = (g_{xB}, g_{yB})$:
   если угол между $\vec{G}_A$ и $\vec{G}_B$ больше допустимого, то вычислить центр окружности $O(x_O, y_O)$ по формулам (3.2) и увеличить счетчик $H[x_O, y_O]$ на единицу.
2. Среди счетчиков $H[x, y]$ найти локальные максимумы, положения которых соответствуют положениям центров искомых окружностей.

Результаты работы алгоритма показаны на рис. 3.33.

Следует отметить, что теперь значение счетчика H не будет равно числу точек окружности (оно равно числу пар точек в области поиска).

а *б*

Рис. 3.33. Результаты работы алгоритма поиска окружностей по парам точек: *а* — исходное изображение; *б* — отклики в пространстве параметров

Данный алгоритм работает гораздо надежнее предыдущего и вполне может использоваться на практике. Однако из рис. 3.33 видно, что окружность по-прежнему дает довольно размазанный отклик по той же причине — из-за недостаточно высокой точности определения градиента.

Повысить точность расчета центра окружности можно, исключив из расчета градиент одной из точек. Это возможно, поскольку известно, что обе точки окружности должны лежать на равном расстоянии от искомого центра (рис. 3.34).

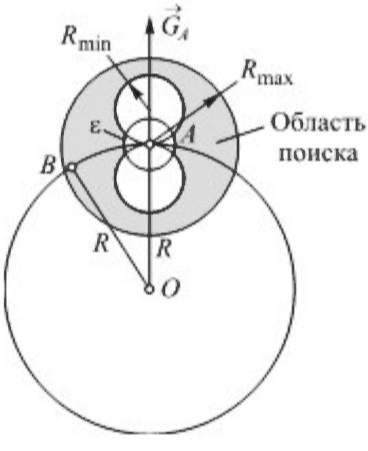


Рис. 3.34. Модифицированная схема вычисления центра окружности

Получить уравнения можно следующим образом:

$$\begin{cases} OA \parallel \vec{G}_A; \\ OA = OB \end{cases} \Rightarrow$$

$$\Rightarrow \begin{cases} \dfrac{x_O - x_A}{y_O - y_A} = \dfrac{g_{xA}}{g_{yA}}; \\ (x_A - x_O)^2 + (y_A - y_O)^2 = (x_B - x_O)^2 + (y_B - y_O)^2 \end{cases} \Rightarrow$$

$$\Rightarrow \begin{cases} g_{yA} x_O - g_{xA} y_O = g_{yA} x_A - g_{xA} y_A; \\ (2x_A - 2x_B) x_O + (2y_A - 2y_B) = x_A^2 + y_A^2 - x_B^2 + y_B^2. \end{cases}$$

В этом случае определитель системы

$$\Delta = 2g_{yA}(y_A - y_B) + 2g_{xA}(x_A - x_B).$$

Определитель равен нулю тогда и только тогда, когда отрезок AB параллелен направлению вектора градиента $\vec{G}_A$.

Решение системы будет иметь вид

$$x_O = \frac{2d_1(y_A - y_B) + d_2 g_{xA}}{\Delta};$$
$$y_O = \frac{g_{yA} d_2 - 2(x_A - x_B) d_1}{\Delta}, \qquad (3.3)$$

где $d_1 = g_{yA} x_A - g_{xA} y_A$, $d_2 = x_A^2 + y_A^2 - x_B^2 + y_B^2$ — правые части системы уравнений.

Алгоритм поиска окружностей останется без изменений, изменится лишь формула для расчета центра окружности — вместо (3.2) следует использовать (3.3). Результаты работы алгоритма показаны на рис. 3.35, из которого видно, что отклики окружности в пространстве поиска стали небольшими и дают острые максимумы. Этот алгоритм вполне можно применять на практике.

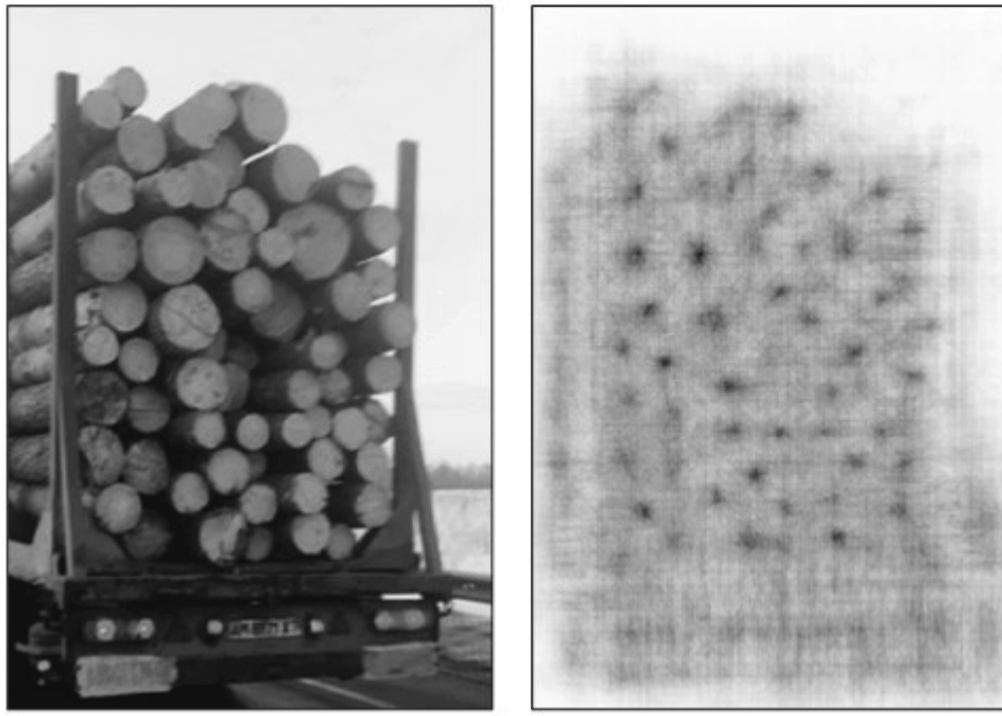
а *б*

Рис. 3.35. Результаты работы модифицированного алгоритма поиска окружностей по парам точек:
а — исходное изображение; *б* — отклики в пространстве параметров

В целом, данный алгоритм обладает достаточно высокой надежностью и приемлемым быстродействием, поэтому именно его следует выбирать для большинства практических приложений. Пример детектирования окружностей в задаче определения радиусов бревен погруженного леса показан на рис. 3.36.

Поиск окружностей по парам противолежащих точек. Наибольшая точность расчета положения центра окружности может быть достигнута в случае наиболее удаленных точек. Однако рассмотренный выше алгоритм не позволяет использовать такие точки, так как это приводит к слишком большим областям поиска, огромным количествам пар точек и неприемлемо большому времени вычислений.

Этот алгоритм можно модифицировать, сформировав область поиска так, чтобы туда попадали противолежащие точки. Для этого область поиска следует вытянуть вдоль градиента яркости в текущей точке (рис. 3.37).



а *б*

Рис. 3.36. Результаты работы модифицированного алгоритма поиска окружностей по парам точек:
а — исходное изображение; *б* — отклики в пространстве параметров

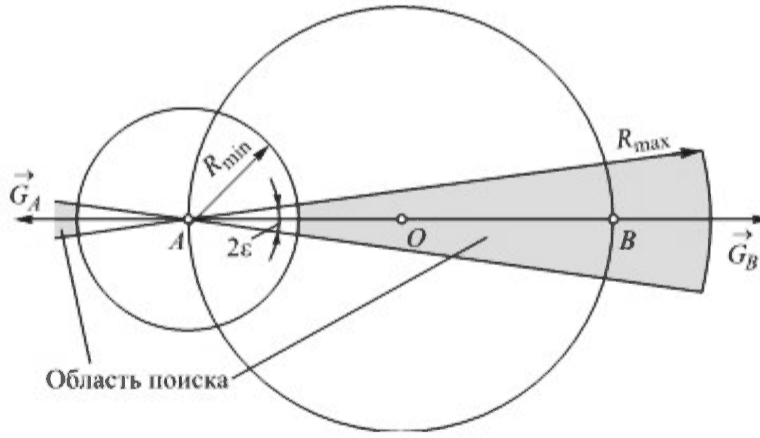


Рис. 3.37. Схема поиска окружностей по парам противолежащих точек

Параметры области поиска определяются соответственно минимальным R_{min} и максимальным R_{max} радиусами детектируемых окружностей, угол ε — точностью определения градиента яркости.

Необходимо найти в области поиска противолежащие точки B_i, их градиент должен быть противонаправлен точке A. Тогда центр искомой окружности будет находиться на середине отрезка AB.

Алгоритм поиска в этом случае будет выглядеть следующим образом.

1. Для каждой точки границы A_i с координатами (x_A, y_A) и градиентом яркости $\vec{G}_A = (g_{xA}, g_{yA})$:
   сформировать область поиска противолежащих точек;
   для каждой точки границы B_j из области поиска:
      если ее градиент противонаправлен градиенту точки A_i,
   то вычислить центр окружности:
      $x_O = (x_A + x_B)/2$;
      $y_O = (y_A + y_B)/2$
   и увеличить счетчик $H[x_0, y_0]$ на единицу.
2. Среди счетчиков $H[x_O, y_O]$ найти локальные максимумы, положение которых соответствует центрам искомых окружностей.

Данный алгоритм дает достаточно острые и неразмытые максимумы в положениях центров окружностей (рис. 3.38), требует приемлемого объема вычислений, в силу чего довольно широко используется на практике. Однако он имеет существенный недостаток: если часть окружности невидима или искажена, алгоритм будет неработоспособен, поскольку невозможно будет найти противолежащие точки. По этой же причине алгоритм непригоден для поиска дуг. В этих случаях следует использовать предыдущий алгоритм.

а *б*

Рис. 3.38. Результаты работы алгоритма поиска окружностей по парам противолежащих точек:
а — исходное изображение; *б* — отклики в пространстве параметров (отклик правой верхней окружности отсутствует, поскольку не удается найти противолежащие точки)

Можно построить аналогичные алгоритмы для любых других пар точек окружности, необязательно противолежащих. Однако это будет приводить к бóльшим по размеру областям поиска и, соответственно, к большему объему вычислений.

Поиск окружностей по тройкам точек. Радикальное решение проблемы низкой точности определения градиента заключается в полном отказе от градиента в расчетах. Вместо градиента можно использовать координаты точек

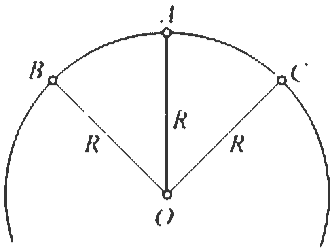


Рис. 3.39. Схема вычисления центра окружности по трем точкам

границы, значения которых известны с очень высокой точностью. Однако для определения центра окружности потребуются уже три точки границы (рис. 3.39).

Рассчитать положение центра окружности по трем точкам можно следующим образом:

$$OA^2 = OB^2 = OC^2 = R^2.$$

Получим систему трех уравнений с тремя неизвестными — x_O, y_O, R:

$$\begin{cases}(x_O - x_A)^2 + (y_O - y_A)^2 = R^2;\\ (x_O - x_B)^2 + (y_O - y_B)^2 = R^2;\\ (x_O - x_C)^2 + (y_O - y_C)^2 = R^2.\end{cases}$$

Эти уравнения не линейные, а квадратные, однако по рис. 3.39 видно, что система имеет единственное решение и лишний корень должен сократиться.

Исключим из уравнений R^2:

$$\begin{cases}(x_O - x_A)^2 + (y_O - y_A)^2 = (x_O - x_B)^2 + (y_O - y_B)^2;\\ (x_O - x_A)^2 + (y_O - y_A)^2 = (x_O - x_C)^2 + (y_O - y_C)^2.\end{cases}$$

Раскрыв скобки, получим

$$\begin{cases}x_O^2 - 2x_O x_A + x_A^2 + y_O^2 - 2y_O y_A + y_A^2 = x_O^2 - 2x_O x_B + x_b^2 + y_O^2 - 2y_O y_B + y_B^2;\\ x_O^2 - 2x_O x_A + x_A^2 + y_O^2 - 2y_O y_A + y_A^2 = x_O^2 - 2x_O x_C + x_C^2 + y_O^2 - 2y_O y_C + y_C^2.\end{cases}$$

Перенесем компоненты с x_0, y_0 влево, без них — вправо:

$$\begin{cases}2(x_B - x_A)x_O + 2(y_B - y_A)y_O = x_B^2 - x_A^2 + y_B^2 - y_A^2 = d_1;\\ 2(x_C - x_A)x_O + 2(y_C - y_A)y_O = x_C^2 - x_A^2 + y_C^2 - y_A^2 = d_2.\end{cases}$$

Как видим, компоненты, содержащие квадрат, сократились, и получили линейную систему с определителем

$$\Delta = 2(\Delta x_{BA}\Delta y_{CA} - \Delta x_{CA}\Delta y_{BA}).$$

Он равен нулю, когда отрезки AB и BC коллинеарны; в этом случае центр окружности O уходит в бесконечность. Другая возможность — совпадение точек (A, B), (A, C) либо (B, C) — устраняется выбором области поиска.

Решение системы следующее:

$$\begin{aligned}x_O &= \frac{d_1(y_C - y_A) - d_2(y_B - y_A)}{\Delta};\\ y_O &= \frac{d_2(x_B - x_A) - d_1(x_C - x_A)}{\Delta},\end{aligned} \tag{3.4}$$

где $d_1 = x_B^2 + y_B^2 - x_A^2 + y_A^2$; $d_2 = x_C^2 + y_C^2 - x_A^2 - y_A^2$.

Алгоритм поиска может быть построен следующим образом.

1. Для каждой точки границы A_i с координатами (x_A, y_A) с градиентом яркости $\vec{G}_A = (g_{xA}, g_{yA})$:
   сформировать область поиска соседних точек в соответствии с рис 3.39.
   Для каждой несовпадающей пары точек B_p и C_q вычислить Δx.
   Если $\Delta x \neq 0$, то вычислить x_O, y_O по формулам (3.4) и увеличить счетчик $H[x_O, y_O]$ на единицу.
2. Среди значений счетчиков найти локальные максимумы, положение которых соответствует положению искомых центров окружностей.

Результаты работы алгоритма приведены на рис. 3.40. Алгоритм позволяет формировать острые локальные максимумы в пространстве поиска, однако он перебирает уже тройки точек, которых довольно много. Поэтому производительность данного алгоритма невысока.

а *б*

Рис. 3.40. Результаты работы алгоритма поиска окружностей по тройкам точек: *а* — исходное изображение; *б* — отклики в пространстве параметров

Увеличить производительность алгоритма можно, если рассматривать не все тройки точек, а лишь фиксированное число случайных точек, но при этом снизится надежность обнаружения окружности.

Другой способ повышения производительности связан с дополнительной проверкой инкрементируемого счетчика $H[x_O, y_O]$: если он уже содержит достаточное число точек, то соответствующая окружность переносится в список найденных окружностей, а принадлежащие ей точки изымаются из изображения для сокращения времени поиска других окружностей.

Здесь выигрыш по времени сильно зависит от момента обнаружения окружности: чем раньше найдена окружность и изъяты ее точки, тем больший выигрыш во времени будет получен. Поэтому стратегию перебора точек A_i следует строить так, чтобы в первую очередь просматривались точки, с наибольшей вероятностью принадлежащие искомой окружности.

Вычисление радиуса окружности. Рассмотренные выше алгоритмы позволяли находить лишь центр окружности, тогда как часто необходимо знать и ее радиус. Значение радиуса найти легко: это расстояние от текущей точки до вычисленного центра O окружности. Но точка A необязательно принадлежит

окружности с центром в точке O — это может быть случайная точка другого объекта либо просто шум. Таким образом, возникает задача накопления наиболее вероятного значения радиуса окружности.

Ситуация может осложниться и в том случае, когда несколько окружностей имеют общий центр, но разные радиусы (концентрические окружности).

Наиболее простой способ накопления значений радиуса связан с введением в пространство поиска дополнительной координаты, отвечающей за радиус. Пространство поиска станет трехмерным: $<x_O, y_O, R>$. Теперь, вычислив для текущей точки A_i (или для пары/тройки точек) центр окружности, также вычисляем радиус $R = \sqrt{(x_A - x_O)^2 + (y_A - y_O)^2}$ и увеличиваем счетчик $H[x_O, y_O, R]$.

Следует обратить внимание, что не надо заменять ось радиуса осью квадрата радиуса, чтобы избежать вычисления квадратного корня, поскольку возрастет общее число ячеек в счетчике H (это приведет к существенным затратам времени на операции поиска в пространстве поиска и его инициализации). Также нарушится изотропность пространства поиска: у пространства поиска появятся «любимые» и «нелюбимые» радиусы, сильно различающиеся по качеству детектирования.

Правильным подходом здесь будет табличное вычисление корня. Диапазон радиусов искомых окружностей, как правило, можно знать заранее и сформировать для них таблицу корней.

Введение новой оси в пространстве поиска обычно приводит к росту объема вычислений: замедляются поиск максимумов в ПП и их обработка. Поэтому в ряде случаев можно использовать двумерное пространство ПП, но расширенное для поиска радиуса. Тогда каждая ячейка ПП будет хранить не только счетчик числа точек окружности, но и сумму радиусов (их отношение даст средний радиус).

Такой подход не требует больших объемов памяти и не снижает производительность. Однако он не позволяет обрабатывать коллинеарные окружности, а значение радиуса будет подвержено воздействию шума (хотя центр окружности будет вычисляться с прежней надежностью).

Чтобы получить желаемую комбинацию точности определения радиуса и производительности, можно сочетать эти подходы. Например, иметь не одну, а несколько расширенных ячеек для накопления средних радиусов в нескольких диапазонах. При этом числитель и знаменатель среднего радиуса хранятся раздельно для упрощения вычислений.

Выделение дуг. Для выделения дуг можно использовать подход с расширенным ПП, как это было сделано для поиска отрезков. В простейшем случае в расширенном ПП необходимо хранить минимальный и максимальный углы для точек, принадлежащих текущей дуге. К сожалению, такой подход не работает, поскольку угол — это периодическая функция и указать минимальный и максимальный углы невозможно. Из рис. 3.41, *а* видно, что минимальный угол $\varphi = 0$ будет у точки C, а не у точки A; максимальный угол $\varphi = 360°$ — у точки D, лежащей на пиксель ниже точки C, а не у точки B.

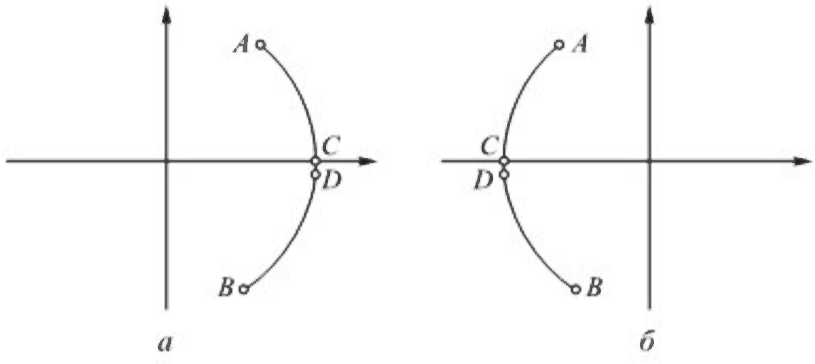


Рис. 3.41. Проблема выделения дуг:
а — положение максимального и минимального углов при изменении угла от 0 до 360° (минимальный угол у точки *C*, максимальный — у точки *D*); *б* — положение максимального и минимального углов при изменении угла от −180° до +180° (минимальный угол у точки *C*, максимальный — у точки *D*)

Если угол изменять в диапазоне −180°... +180°, то это не решит проблему (рис. 3.41, *б*): минимальный и максимальный углы оказываются у точек *C* и *D*, находящихся внутри дуг.

Одно из возможных решений данной проблемы заключается в том, что отсчет угла точек следует вести от угла центральной точки дуги (рис. 3.42). В этом случае дуга однозначно описывается углом центральной точки $\varphi_{ср}$ и своей угловой длиной $2\Delta\varphi$.

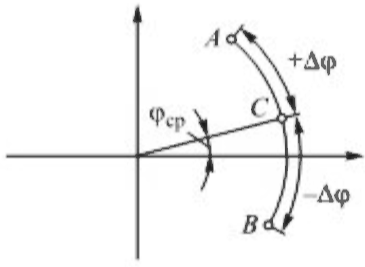


Рис. 3.42. Схема параметризации дуги от среднего угла

В этом случае в расширенное ПП необходимо добавить средний угол дуги (отдельно числитель и знаменатель) и угловую длину дуги.

Алгоритм поиска дуг будет выглядеть следующим образом.

1. Для каждой точки границы A_i:
   вычислить центр окружности $O(x_O, y_O)$ и радиус R (любым из вышеприведенных методов);
   найти счетчик дуги $H[x_O, y_O, R]$;
   вычислить угол наклона отрезка OA_i (с помощью таблицы арктангенсов);
   вычислить угол $\angle A_iOC = (\varphi_A - \varphi_C) \bmod 360$.
   Если он больше половины текущей дуги, то точка A_i находится за пределами дуги и необходима коррекция концов дуги:
   если $\angle A_iOC > +\Delta\varphi$, то $\Delta\varphi' = \varphi_C - \Delta\varphi + \varphi_A$; $\varphi_C = (\Delta\varphi + \Delta\varphi')/2$;
   если $\angle A_iOC < -\Delta\varphi$, то $\Delta\varphi' = \varphi_C + \Delta\varphi - \varphi_A$; $\varphi_C = (\Delta\varphi + \Delta\varphi')/2$;
   увеличить счетчик числа точек дуги $H[x_O, y_O, R].C$.
2. Среди значений H найти локальные максимумы, параметры соответствующих записей H перенести в список дуг.

Очевидно, что алгоритм поиска дуг незначительно отличается от алгоритма поиска окружностей, и является надстройкой над ним.

К сожалению, данный алгоритм не позволяет различать дуги, лежащие на одной окружности: порядок просмотра точек, как в случае с отрезками, не позволит отделить такие дуги одну от другой. Однако их можно разделить позже, на этапе верификации найденных окружностей.

Верификация окружностей. Как и в случае с отрезками, верификация окружностей предусматривает повторный просмотр точек границ, потенциально принадлежащих окружности, в целях уточнения параметров этой окружности (как правило, ее радиуса, точного числа точек, а также концов дуг в случае поиска дуг). В отличие от верификации отрезков, для верификации окружностей может требоваться на порядки меньше времени, чем для детектирования, и ее целесообразно использовать во время детектирования в целях исключения из рассмотрения точек, принадлежащих уже найденным окружностям.

Необходимость верификации заключается в том, что число точек границы, «проголосовавших» за данную окружность, гораздо меньше реального числа точек этой окружности из-за дискретности изображения и ПП, воздействия шума и т. д.

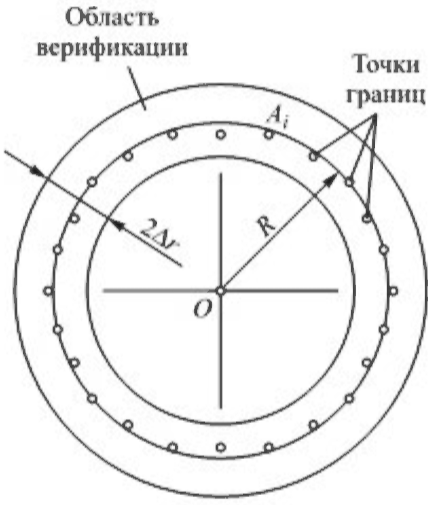


Рис. 3.43. Схема верификации найденных окружностей

Наиболее простой метод задания области верификации — табличный: для каждого радиуса заранее рассчитывают координаты точек области верификации относительно центра окружности. Достаточно рассчитать точки 1/8 области с углом 0...45°, остальные координаты получают благодаря симметрии.

Можно генерировать области верификации и аналитически, перебирая точки внутри квадрата со стороной $2R$ и с центром в точке O и отсеивая точки с расстоянием до точки O в пределах $R \pm \Delta r$ (рис. 3.43). Это необходимо, когда радиус искомой окружности изменяется в широких пределах и задать область верификации таблицей не удается.

Верификация начинается с подсчета числа точек в области верификации; окружности со слишком малым числом точек отбрасываются. Затем уточняют центр окружности, ее радиус и остальные параметры, например концы дуг. После этого найденные точки подтвержденной окружности изымаются из изображения, чтобы они не мешали поиску и верификации оставшихся окружностей.

Выделение эллипсов

Задача выделения эллипсов часто встречается на практике, поскольку эллипс более универсальный объект, чем окружность. Эллипсами являются контуры отверстий, видимых под углом, спилы деревьев, клетки на гистологических препаратах и т. д. Однако в отличие от окружностей для описания эллипса требуется уже шесть параметров:

$$a_1x^2 + a_2y^2 + a_3xy + a_4x + a_5y + a_6 = 0.$$

Поэтому для реализации преобразования Хафа в чистом виде потребуется шестимерное пространство поиска со всеми вытекающими последствиями.

В практических приложениях не все параметры эллипса одинаково важны (как правило, наиболее важным является центр эллипса), что позволяет определять параметры эллипса по частям: сначала находят его примерное положение, а затем уточняют параметры.

Одним из наиболее значимых свойств эллипса, позволяющим находить его на изображении, является то, что прямая, проходящая через центры параллельных хорд эллипса, также проходит и через его центр (рис. 3.44).

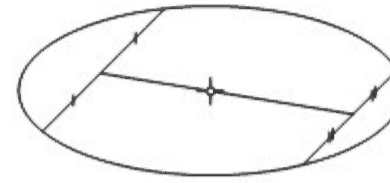


Рис. 3.44. К пояснению свойства хорд эллипса

Искать четверки точек, дающие параллельные хорды, нецелесообразно: таких четверок слишком много, поэтому прямой метод не годится. Вместо хорд можно использовать касательные, считая их предельным приближением с нулевой длиной.

Тогда для каждой точки контура можно задать область поиска параллельных касательных. Очевидно, что для этого вторая точка должна иметь противонаправленный градиент яркости (рис. 3.45).

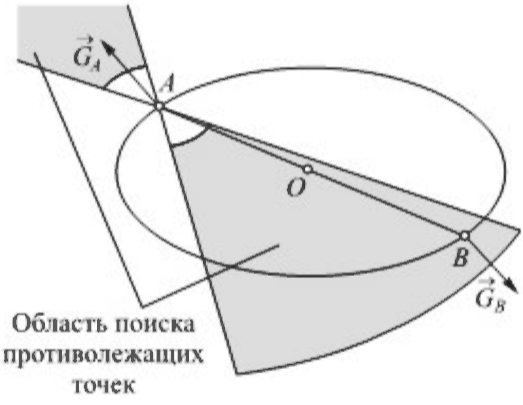


Рис. 3.45. Схема поиска эллипса по противоположным точкам

Область поиска, как и в случае с окружностью, будет строиться исходя из направления градиента в текущей точке A, значение φ зависит от максимального эксцентриситета эллипса, а $R_{\min}$ и $R_{\max}$ — от минимального и максимального размера полуосей. Область поиска желательно задать табличным способом.

Алгоритм поиска эллипсов в этом случае можно записать следующим образом.

1. Для каждой точки границы A_i с координатами (x_A, y_A) и градиентом яркости $\vec{G}_A$:
   построить область поиска противолежащих точек;
   для каждой точки границы B_j из области поиска с градиентом $\vec{G}_B \approx -\vec{G}_A$ ($\arg \vec{G}_B \approx \arg \vec{G}_A \pm 180°$):
   вычислить центр эллипса:
   $x_O = (x_A + x_B)/2$; $y_O = (y_A + y_B)/2$;
   увеличить счетчик $H[x_O, y_O]$.
2. Среди счетчиков $H[x_O, y_O]$ найти локальные максимумы, положения которых соответствуют положениям центров искомых эллипсов.

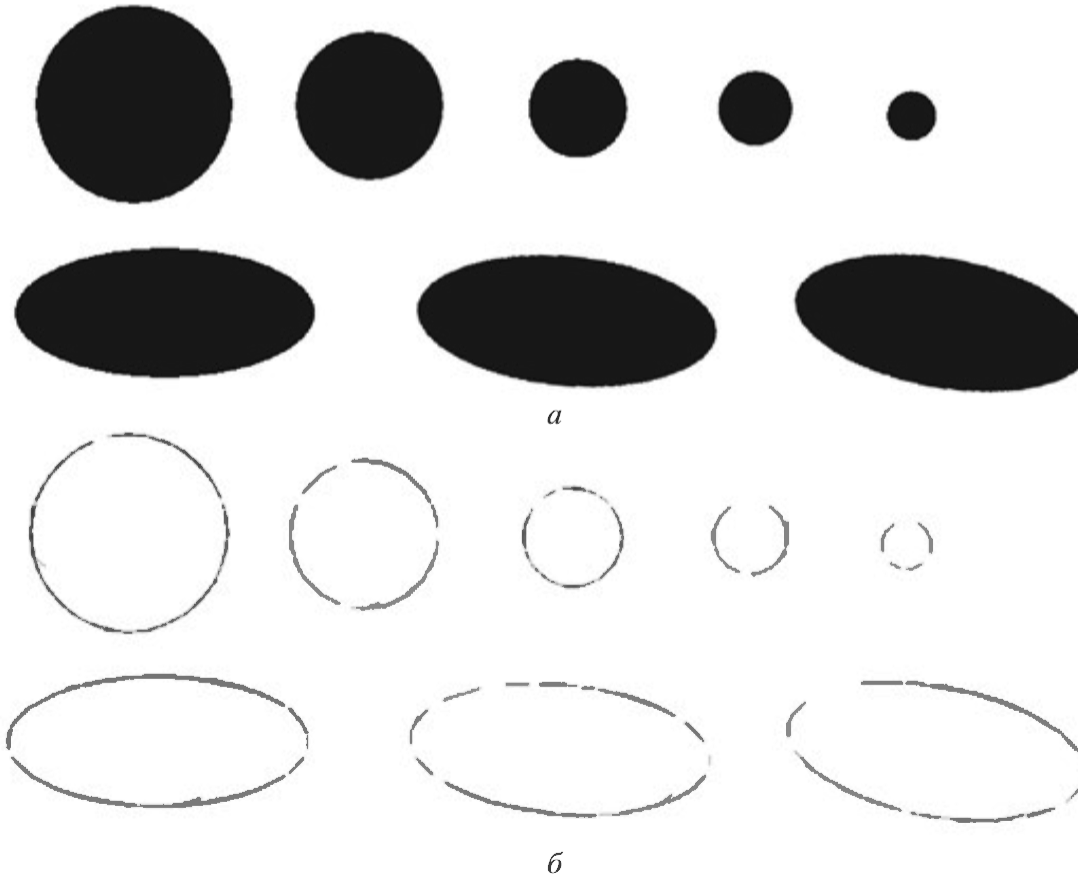
Данный алгоритм требует гораздо большего объема вычислений, чем аналогичный алгоритм поиска окружностей по противоположным точкам,

поскольку размер области поиска (и соответствующий объем перебора) существенно возрастает. Алгоритм также требует наличия полного эллипса, иначе нахождение противоположных точек будет невозможным.

Поиск эллипсов с использованием аппроксимации отрезками. Эллипсы, которые видим на изображении, в действительности — сложные выпуклые многоугольники. Поэтому было бы логично описать их набором отрезков, а затем комбинировать найденные отрезки для обнаружения эллипса.

На рис. 3.46 приведены результаты аппроксимации эллипса отрезками с помощью алгоритма выделения отрезков, рассмотренного ранее. Видно, что даже эллипсы небольшого размера представлены достаточно детализированно для дальнейшего поиска.

Будем рассматривать каждый отрезок в качестве хорды эллипса. Отсортируем отрезки по их углу наклона и разобьем на группы так, чтобы в каждой группе оказались параллельные друг другу отрезки. В пространстве поиска построим отрезки, соединяющие средние точки каждой пары. Эти отрезки должны пройти через центр эллипса. Точки пересечения большого числа отрезков в ПП дадут центр эллипса.



а

б

Рис. 3.46. Пример аппроксимации эллипса отрезками:
а — исходное изображения; *б* — результат его аппроксимации отрезками

Такой подход достаточно производителен, поскольку здесь перебирается относительно небольшое число параллельных отрезков в пределах каждой группы.

Аналогично алгоритмам поиска окружностей, алгоритм поиска эллипсов может иметь надстройки для поиска других параметров (полуосей, эксцентриситета, концов дуг) как при прямом проходе, так и на этапе верификации.

Для построения отрезков в пространстве поиска можно использовать алгоритм Брезенхема или алгоритм с приращениями. Для уточнения положения точки пересечения отрезков можно использовать антиалайзинг.

В этом случае алгоритм можно сформулировать следующим образом.

1. Для каждого отрезка AB на изображении со средним градиентом яркости $\vec{G}_{AB}$:
   найти отрезок CD, параллельный AB, со средним градиентом яркости $\vec{G}_{CD}$, примерно противонаправленным $\vec{G}_{AB}$;
   найти средние точки отрезков E для AB и F для CD;
   найти среднюю точку $O(x_O, y_O)$ отрезка EF, соединяющего центры найденных отрезков;
   увеличить счетчик $H[x_O, y_O]$.
2. Среди значений счетчика H найти локальные минимумы, соответствующие положениям центров искомых эллипсов.

Выделение парабол

Рассмотренный подход с использованием преобразования Хафа можно распространить на широкий круг кривых, заданных в аналитической форме.

Рассмотрим задачу обнаружения парабол на изображении. Она возникает, когда необходимо, например, найти на изображении или видео объекты, свободно падающие под действием гравитационного поля.

Уравнение параболы можно записать в виде

$$y_i = ax_i^2 + bx_i + c,$$

где a, b, c — искомые параметры параболы; x_i , y_i — координаты точек, через которые проходит парабола.

Запишем дуальное уравнение:

$$c = y_i - ax_i^2 - bx_i.$$

Очевидно, что это уравнение плоскости в пространстве $<a, b, c>$.

Алгоритм поиска будет сводиться к поиску точек пересечения таких плоскостей.

1. Для каждой точки $A(x, y)$, потенциально принадлежащей искомой параболе:
   для всех возможных значений коэффициентов a и b параболы:
   вычислить $c = y_i - ax_i^2 - bx_i$;
   увеличить счетчик $H[a, b, c]$ на единицу.
2. Среди значений $H[a, b, c]$ найти локальные минимумы, положение которых соответствует искомым параметрам $<a^*, b^*, c^*>$ параболы, а значение — числу точек, принадлежащих этой параболе.

Этот алгоритм пригоден для наиболее общего случая, однако, если об искомой параболе имеются дополнительные данные, работу алгоритма можно существенно ускорить.

Пусть, например, известен коэффициент a параболы (в задачах, связанных со свободным падением, a равно ускорению свободного падения g). Тогда величина $y_i - ax_i^2$ становится константой, и вместо точек пересечения плоскостей в трехмерном пространстве $<a, b, c>$ достаточно искать точку пересечения прямых в пространстве $<b, c>$.

Уравнения этих прямых будут иметь вид

$$c = -bx_i + [y_i - ax_i^2].$$

Алгоритм детектирования парабол в этом случае может быть записан в следующем виде.

1. Для каждой точки $A(x, y)$, потенциально принадлежащей искомой параболе:
   для всех возможных значений b:
   вычислить $c = -bx_A + [y_A - ax_A^2]$;
   увеличить счетчик $H[b, c]$ на единицу.
2. Среди значений $H[b, c]$ найти локальные минимумы, положение которых соответствует искомым параметрам $<b^*, c^*>$ параболы, а значение — числу точек, принадлежащих этой параболе.

Сравнивая этот алгоритм с оригинальным алгоритмом Хафа поиска прямых, легко заметить, что они практически тождественны. Поиск прямых и парабол имеет почти одинаковые алгоритмы с точностью до формулы вычисления c.

В случае, когда парабола является элементом контура, для ее точек можно рассчитать градиент яркости G. Знание градиента яркости также позволяет существенно упростить алгоритм.

Градиент яркости в каждой точке параболы должен быть перпендикулярен направлению касательной к параболе в этой точке. Тангенс угла наклона этой касательной будет равен значению производной в этой точке:

$$\operatorname{tg}\alpha = \frac{\delta y}{\delta x} = 2ax_A + b.$$

В то же время условие перпендикулярности позволяет записать

$$\operatorname{tg}\alpha = \frac{g_x}{g_y},$$

откуда, приравнивая правые части, получим

$$2ax_A + b = \frac{g_x}{g_y} \quad \Rightarrow \quad a(b) = \frac{1}{2x_A}\left(\frac{g_x}{g_y} - b\right).$$

Таким образом, каждому параметру параболы b будет соответствовать только одно значение параметра a, задаваемое координатой точки и ее градиентом яркости.

Алгоритм поиска в этом случае может быть записан в следующем виде.

1. Для каждой точки контура $A(x_A, y_A)$ с градиентом яркости $\vec{G} = (g_x, g_y)$:
   для всех возможных значений b:
   вычислить $a(b) = [g_x/g_y - b]/(2x_A)$;
   вычислить $c(a, b) = -x_A b + [y_A - a{x_A}^2]$;
   увеличить счетчик $H[a, b, c]$.
2. Среди значений $H[a, b, c]$ найти локальные минимумы, положение которых соответствует искомым параметрам параболы.

Такой алгоритм хотя и оперирует трехмерным пространством параметров, но отклики ищутся как точки пересечения прямых, а не плоскостей. Это существенно повышает и быстродействие, и надежность алгоритма благодаря уменьшению числа ложных откликов.

Глава 4
МЕТОДЫ СОПОСТАВЛЕНИЯ ИЗОБРАЖЕНИЙ

Задачи сопоставления и распознавания

К числу основных задач обработки изображений относятся задачи поиска и распознавания объекта. Задача поиска встречается там, где необходимо найти заданный образец в кадре, или положение объекта с одного кадра на другом кадре, или положение кадра, полученного в одном спектре относительно того же кадра, но в другом спектре, например в задаче совмещения фотоснимков в инфракрасном и видимом диапазонах. Такие задачи традиционно называют *задачами сопоставления* или *регистрации изображений*.

Другой важной задачей является задача распознавания, или *идентификации*, объекта по его изображению, которое предполагает отнесение объекта на изображении к одному из заранее заданных классов объекта путем сравнения с заданными образцами или наборами параметров.

Задачи сопоставления и распознавания, как правило, решаются совместно. Действительно, чтобы распознать объект, его необходимо сначала найти на изображении. И наоборот, прежде чем найти объект, надо понимать, что собираемся искать. В нетривиальных случаях задачи распознавания и сопоставления решаются итеративно: сначала находят предполагаемое положение объекта на изображении, потом проводят распознавание найденной области, определение типа объекта, его модели и других параметров, после чего снова уточняют положение объекта, определяют второстепенные параметры ракурса и ориентации, а затем вновь может следовать уточняющее распознавание.

Другой вариант — иерархический анализ, когда распознавание и сопоставление используются сначала для анализа крупных деталей, а затем все более мелких. Примером может служить задача распознавания текста на странице: сначала определяются крупные блоки, предположительно содержащие текст, затем в пределах блоков выявляются строки и выполняется первичная идентификация шрифта, потом строки разбиваются на отдельные буквы, после чего распознаются уже отдельные буквы. Далее осуществляется обратная операция — сборка: из найденных букв собираются слова с их проверкой и распознаванием по орфографическому словарю, затем из слов собираются предложения с проверкой синтаксиса и т. д. Здесь видим иерархическое чередование методов сопоставления и распознавания.

Чтобы рассмотреть большое число существующих на данный момент методов сопоставления изображений, целесообразно применять некоторую классификационную схему. Классическая схема предполагает использование следующих признаков классификации:

- признаки, по которым будет осуществляться сопоставление;
- параметры положения искомого объекта, подлежащие определению в процессе сопоставления;
- мера близости объектов;
- стратегия оптимизации меры близости.

Математическая постановка задачи сопоставления изображений. Пусть имеются изображение $I[x, y]$, заданное в виде прямоугольного массива отсчетов интенсивности излучения, и образец $S[x, y]$ в виде такого же массива (как правило, меньшего размера).

Пусть $f(x, y)$ — вектор-функция преобразования координат, преобразующая координаты фрагмента изображения в координаты образца. Пусть $g(I)$ — функция преобразования яркости точек изображения в яркость точек образца. Тогда задачу сопоставления можно сформулировать следующим образом: требуется найти такое преобразование координат f и яркости g, что все точки объекта и изображения совпадут:

$$S[x, y] = g\left(I\left[f(x, y)\right]\right) \text{ для всех } x = 0, \ldots, n_x - 1,\ y = 0, \ldots, n_y - 1,$$

где n_x, n_y — ширина и высота образца.

Однако на практике добиться такого совпадения невозможно, хотя бы потому, что на изображении всегда могут присутствовать шум и помехи, которых нет на образце. Поэтому требование равенства яркостей точек следует ослабить.

Введем функцию отклонения яркости объекта от яркости образца. Чем больше яркость образца отличается от яркости объекта, тем больше будет значение этой функции. Можно использовать различные виды функций, например модуль разности яркостей, квадрат разности и др.

Обозначим $\|S, I\|$ меру близости (норму) точки образца и точки изображения. Чтобы получить функцию отклонения в общем виде, запишем ее сначала в виде нормы $\|S, I\|$. Далее введем в рассмотрение функционал качества сопоставления $J(f, g)$:

$$J(f, g) = \sum_{x=0}^{n_x-1} \sum_{y=0}^{n_y-1} \left\| S[x, y], g\left(I\left[f(x, y)\right]\right) \right\|.$$

Здесь $J(f, g)$ — функционал, поскольку его аргументами являются не числовые значения, а функция преобразования координат и функция преобразования яркости.

Очевидно, что наилучшее совпадение образца и изображения будет при таких функциях f, g, при которых значение $J(f, g)$ минимально. Таким образом, задача сопоставления изображений сводится к минимизации функционала качества совпадения $J(f, g)$:

$$J(f,g)=\sum_{x=0}^{n_x-1}\sum_{y=0}^{n_y-1}\left\|S[x,y],g\left(I\left[f(x,y)\right]\right)\right\|\to\min_{f,g}.$$

Известно, что в общем случае такая задача решения не имеет, хотя бы потому, что число всевозможных функций f и g бесконечно велико. Чтобы решить данную задачу на практике, необходимо задать конкретный вид функций f и g, исходя из контекста решаемой задачи. Это позволит перейти от функционала качества к функции качества, которая будет зависеть уже от параметров функций f и g, а не от их класса. Параметры функций f и g будем называть *параметрами поиска*. Очевидно, что, задав этих параметры, будем точно знать искомое положение объекта на изображении.

Выбор параметров поиска

Рассмотрим отдельно параметры функции преобразования координат и функции преобразования яркости.

Выбор параметров функции преобразования координат. В простейшем случае, когда изображение можно считать плоским, а размеры и ориентация объекта заранее известны, положение объекта задается параметрами *параллельного переноса*:

$$(x',y')=f(x,y)=\begin{cases}x'=x+\Delta x;\\ y'=y+\Delta y,\end{cases}$$

где Δx, Δy — параметры параллельного переноса.

Значения Δx и Δy — это фактически параметры смещения некоторой референтной точки образца на изображении относительно начала координат. Референтной может быть выбрана любая точка, поскольку параметры Δx и Δy от ее выбора не зависят. Как правило, это либо центр образца, либо начало координат в системе координат образца.

Большинство практических задач стараются свести именно к определению параметров параллельного переноса, поскольку определение других параметров связано с большими вычислительными затратами. Другие параметры стремятся установить путем дополнительных измерений (например, найти размер объекта по расстоянию до него) или проигнорировать их, считая разновидностью помех, и определить позже, когда параметры параллельного переноса уже будут найдены.

Ориентация объекта. В простейшем случае, когда объект можно считать плоским, а единственная ось поворота перпендикулярна его плоскости, ориентация объекта задается углом φ его поворота относительно некоторой заранее заданной плоскости (чаще всего горизонтальной):

$$(x',y')=f(x,y)=\begin{cases}x'=x\cos\varphi+y\sin\varphi;\\ y'=y\cos\varphi-x\sin\varphi.\end{cases}$$

Положение оси вращения может быть выбрано различным, например проходящим через центр объекта или начало его координат. Угол поворота φ не зависит от положения оси вращения. Однако от положения оси вращения зависят параметры Δx и Δy параллельного переноса. При повороте и парал-

лельном переносе расстояния между соответствующими точками объекта не изменяются.

Размеры объекта. Если реальные размеры объекта не меняются, то размер объекта на изображении будет определяться расстоянием от объекта до камеры, а также характеристиками объектива. Изменения размеров объекта относительно некоторого заранее заданного значения можно описать коэффициентом масштабирования M (масштабом) объекта:

$$(x', y') = f(x, y) = \begin{cases} x' = Mx; \\ y' = My. \end{cases}$$

Совокупность преобразований параллельного переноса, поворота и масштабирования образует группу преобразований подобия (гомотетии); они линейны, и при этих преобразованиях значения углов объекта не изменяются. Геометрический смысл линейности преобразования заключается в том, что прямые линии преобразуются также в прямые линии.

Еще одним линейным преобразованием, как отмечалось ранее, является косой сдвиг (в практических задачах определить параметры косого сдвига требуется достаточно редко). Вместе эти линейные преобразования образуют группу *аффинных преобразований*. В общем виде аффинные преобразования можно записать как

$$(x', y') = f(x, y) = \begin{cases} x' = a_1 x + a_2 y + a_3; \\ y' = a_4 x + a_5 y + a_6, \end{cases}$$

где a_1, ..., a_6 — параметры аффинного преобразования.

Такая группа аффинных преобразований будет содержать уже шесть параметров поиска (a_1, ..., a_6), подлежащих определению.

Нелинейные преобразования. Нелинейные преобразования встречаются, когда объект на изображении нельзя считать плоским или плоскость объекта непараллельна фокальной плоскости камеры. В более сложных случаях объект может оказаться гибким (например, изображения органов в биомедицинских приложениях), а оптика камеры может создавать дополнительные искажения.

Будем выделять следующие типы преобразований:

• проективные преобразования. Возникают в случае, если объект можно считать плоским, но его плоскость непараллельна фокальной плоскости камеры, их взаимное расположение неизвестно и не допускает выполнения коррекции проекции алгоритмически. В этом случае параметры проективного преобразования могут быть заданы в виде

$$(x', y') = f(x, y) = \begin{cases} x' = \dfrac{a_1 x + a_2 y + a_3}{a_4 x + a_5 y + a_6}; \\ y' = \dfrac{a_7 x + a_8 y + a_9}{a_{10} x + a_{11} y + a_{12}}, \end{cases}$$

где a_1, ..., a_{12} — параметры проективного преобразования;

• локальные преобразования. Встречаются, например, в биомедицинских приложениях, работающих с гибкими органами, или используются для обработки низковысотных аэрофотоснимков, где локальные искажения могут возникать из-за разности высот поверхности, сопоставимой с высотой полета летательного аппарата. Для этого случая общей формулы преобразования координат $f(x, y)$, справедливой для всех областей, не существует. При наличии локальных преобразований все изображение разбивают на подобласти, в рамках которых можно использовать единое преобразование координат. Каждая подобласть получает свой набор параметров поиска. Затем они объединяются в одну область путем сглаживания параметров поиска на стыках подобластей.

Следует обратить внимание, что нелинейные преобразования имеют большое число параметров поиска, и это делает фактически невозможным поиск объекта в режиме реального времени или приближенных к нему условиях. Поэтому полный набор параметров нелинейных преобразований вычисляют, как правило, во второй проход, когда основные параметры уже определены и найдено примерное местоположение искомого объекта. Число параметров поиска для первичной локализации объекта не должно превышать двух-трех в зависимости от условий задачи, прежде всего от размера изображения.

Преобразование яркости. В простейшем случае можно считать, что изменение яркости объекта на изображении относительно образца носит аддитивный характер, т. е.

$$I' = g(I) = I + \Delta I,$$

где ΔI — изменение яркости.

Значение ΔI можно найти статистически, через среднюю яркость изображения $\overline{I}$ и образца $\overline{S}$:

$$\Delta I = \overline{S} - \overline{I},$$

однако эта формула дает корректную оценку лишь в том случае, если на изображении отсутствуют другие (мешающие) объекты со средней яркостью, существенно отличающейся от яркости искомого объекта.

Эта формула также будет давать некорректную оценку, если яркость точек объекта существенно отличается от яркости точек фона. Если диапазон яркостей и число точек объекта и фона поддается оценке, то получить более точную взвешенную оценку можно получить расчетом по формуле

$$\Delta I = \overline{S} - a_{\text{о}}\overline{I}_{\text{о}} - a_{\text{ф}}\overline{I}_{\text{ф}},$$

где $a_{\text{о}}$, $a_{\text{ф}}$ — весовые коэффициенты для выравнивания числа точек образца и фона; $\overline{I}_{\text{о}}$, $\overline{I}_{\text{ф}}$ — средняя оцениваемая яркость точек объекта и фона.

В более сложном случае изменение яркости может носить как аддитивный, так и мультипликативный характер (например, из-за разных условий освещенности):

$$I' = g(I) = kI + \Delta I,$$

где k — коэффициент, учитывающий изменение контрастности.

Как и в случае с яркостью, контрастность можно оценить статистически:

$$k = \frac{\sigma_S}{\sigma_I},$$

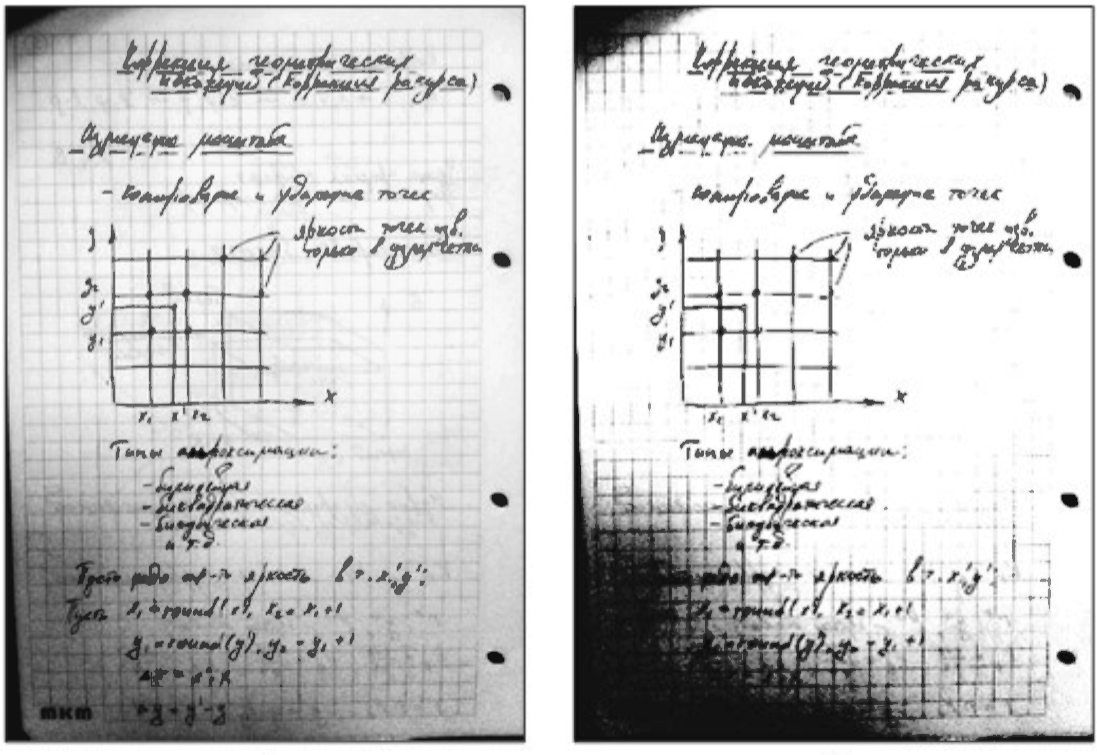
где σ_S, σ_I — СКО яркостей точек образца и изображения соответственно.

Однако такая оценка оказывается довольно грубой вследствие еще более сильного влияния шума и помех на значение СКО.

Преобразование яркости, как и преобразование координат, может быть локальным и глобальным. Глобальное преобразование яркости не зависит от координат и справедливо для всех областей изображения, тогда как локальное требуется в тех случаях, когда условия освещенности областей различны. Примером может служить изображение, полученное с использованием фотовспышки (рис. 4.1) или локального точечного источника света.

Вследствие того, что точечный источник света расположен слишком близко к объекту, расстояние и освещенность отдельных его частей существенно различаются.

В случае локального преобразования яркости его параметры также можно оценить. Для этого изображение разбивают на небольшие области, в пределах которых преобразование можно считать не зависящим от координат (эти области могут перекрываться). Затем проводится «сшивка» параметров областей: в зависимости от типа задачи строится общая модель яркости, как



а *б*

Рис. 4.1. Получение изображения с локальной моделью яркости:
а — исходное изображение; *б* — бинаризованное изображение

правило, в виде линейной или квадратичной функции, а ее параметры уточняются по параметрам яркости локальных областей (например, с использованием метода наименьших квадратов).

Другим примером локальных искажений общего преобразования яркости служат тени и блики. Непосредственное выявление этих искажений достаточно затруднительно и обычно требует наличия модели сцены и модели освещения. Например, в задачах распознавания лиц сначала строят грубую модель сцены, уточняют положение и тип источника света, а затем уже определяют возможные области теней и бликов и проводят их коррекцию.

В случаях когда в качестве признаков сопоставления используются геометрические признаки (углы, границы, центры отверстий и т. п.), не имеющие характеристик яркости, преобразование яркости $g(I)$ можно игнорировать. Это особенно удобно в тех задачах, где модель яркости достаточно сложна и имеется большое число мешающих факторов. Исключение поиска параметров преобразования яркости значительно ускоряет вычисления, но вместе с тем снижает надежность, поскольку довольно существенная часть информации (данные о яркости) оказывается отброшенной и в вычислениях не используется.

Выбор признаков сопоставления

Классификация признаков. В общем случае все признаки, по которым может проводиться и сопоставление, и распознавание изображений, подразделяют на следующие группы:

- алгебраические и геометрические. Алгебраические характеристики представляют собой числовые значения параметров объекта, такие как размер, площадь, периметр, цвет и т. д., а также статистические (статистические моменты) и векторные (спектр) характеристики. Геометрические характеристики — геометрические места точек — это особые точки объекта (центр масс объекта, углы, координаты отверстий), местоположение элементов контура объекта и др.;
- площадные и точечные. Площадные характеристики опираются непосредственно на совокупность точек изображения в некоторой области, тогда как точечные характеризуют список особых точек, элементов контура и других объектов, из которых состоит описание изображения;
- восстановимые и невосстановимые. Восстановимые признаки позволяют полностью или частично восстановить исходное изображение, а невосстановимые не позволяют это сделать. Примером восстановимого признака может служить спектр Фурье, а невосстановимого — статистические моменты.

Типовые признаки. Рассмотрим признаки, которые наиболее часто используются для решения практических задач.

Площадные признаки. Простейшим признаком сопоставления являются непосредственно сами точки изображения. В этом случае задача сопоставления достаточно проста: требуется найти такие положение образца на изображении и преобразование яркости, чтобы яркость точек образца и изображе-

ния были наиболее близкими. Использование яркостей точек изображения в качестве признаков сопоставления иногда называют методом площадей. Он удобен тем, что не требует дополнительного выделения признаков помимо уже известных яркостей точек изображения, но вместе с тем имеет и недостатки. Точки изображения помимо координат различаются яркостью, которая может быть легко искажена шумом, помехами и изменением условий освещенности. Как следствие, надежность метода площадей сильно зависит от условий освещенности изображения: если освещение носит сложный или заранее не известный характер, то надежность данного метода будет низкая. Кроме того, точек на изображении довольно много, и соответствующий вектор признаков будет иметь большую размерность. Это приводит к большому объему вычислений, поэтому данные методы, как правило, работают медленно.

Для устранения этих недостатков следует перейти к использованию более крупных признаков. Их на изображении меньше, они имеют больше характеристик для идентификации, и они более устойчивы к изменению условий освещенности.

В качестве площадных признаков может использоваться не только массив интенсивностей точек изображения, но и другие характеристики: цветные и мультиспектральные изображения (их можно рассматривать как набор массивов интенсивностей), массив значений тона и насыщенности, массив амплитуд направлений градиента яркости, массив характеристик текстур и т. д.

Признаки бинаризованных изображений. Если соотношения яркостей искомого объекта и фона заранее известны, то точки объекта можно отделить от фона по их яркости, а затем описать в виде набора признаков. Такими признаками могут быть, например, цепные ходы, главные и второстепенные оси, центроиды, выпуклые оболочки, скелетные линии и другие геометрические признаки. Кроме того, применяют и алгебраические признаки — статистические моменты, площадь и периметр объекта, средняя яркость и др. Все эти признаки выделяются достаточно быстро и надежно, однако сильно зависят от качества проведенной бинаризации. Если выполнить бинаризацию с достаточной надежностью невозможно, то и использование описанных выше признаков также невозможно.

Контурные признаки. В случае если соотношение яркости искомого объекта и фона заранее неизвестно, то все еще можно выделить точки границы. Эти точки отличаются от остальных высокой скоростью изменения яркости, и для их выявления не требуется дополнительной информации о яркости объекта и фона. Сами точки границы могут использоваться в качестве признаков или могут быть собраны в более крупные объекты, как правило, отрезки, реже — дуги, эллипсы, сплайны. Точка границы сама по себе не имеет яркости (яркость в этой точке быстро изменяется и не может быть точно определена), но при этом может иметь дополнительные признаки, такие как направление границы или радиус ее кривизны. Точек границ меньше, чем точек изображения (≈ 13 % общего числа точек), их характеристики не зависят от изменения яркости, поэтому совпадающие точки могут быть найдены с достаточной надежностью.

Однако отдельные точки границ весьма чувствительны к шуму и геометрическим искажениям (поскольку точки границы выделяются фактически дифференцирующим фильтром, который подчеркивает шум).

Кроме того, число точек все еще слишком велико, поэтому их следует объединять в более крупные и более надежные признаки, такие, как отрезки и дуги.

Представление контура в виде отрезков, дуг и других аналогичных признаков позволяет оперировать относительно небольшим числом (около 10^3) достаточно надежных признаков, которые можно легко идентифицировать на образце и на изображении. Это позволяет создавать достаточно надежные и производительные алгоритмы сопоставления.

Тем не менее и здесь имеются свои недостатки. Во-первых, не все изображения можно описать набором подобных признаков (в меньшей степени это касается отрезков, в большей — дуг и других кривых). Если границы объекта сложные, диффузные или сильно изрезанные, то их представление в виде крупных объектов затруднено. Подобные границы достигаются, например, в задачах обработки аэрофотоснимков: диффузные участки имеются на границах лесов, сильно изрезанных побережий и т. д. Во-вторых, описание контуров изображения в виде прямых или кривых линий неоднозначно, и одно и то же изображение можно описать различными способами. Простейший пример — аппроксимация отрезками окружности: для заданной погрешности возможно бесконечно много аппроксимаций, различающихся только углом поворота. Особенно критичным этот фактор будет для сплайнов, в связи с чем их использование в подобных задачах крайне ограниченно. В-третьих, для надежного выделения искомых элементов необходимы вспомогательные вычисления. Важно выбрать алгоритм выделения элементов, который обладал бы необходимым балансом надежности выделения признаков и производительности вычислений. И в-четвертых, сопоставление по признакам типа отрезков и дуг требует достаточно сложного и нетривиального алгоритма вычисления меры близости.

Ключевые точки. Ключевыми точками называются точечные признаки, положение которых однозначно определено как на изображении, так и на образце. В качестве ключевых точек могут использоваться локальные экстремумы яркости, углы, центры окружностей и других фигур, точки границ с большой кривизной, а также специально привнесенные в изображение точечные маркеры и метки.

Помимо координат ключевым точкам, как правило, приписывается дополнительная информация, существенно упрощающая их идентификацию: значение и ориентация угла для угловых точек, радиус кривизны для точек большой кривизны контура, радиус окружности для центров окружностей. Еще одной общей характеристикой ключевых точек может быть гистограмма направления границ (дескриптор), характеризующая преобладающее направление границ в окрестности ключевой точки.

Сопоставление по ключевым точкам осуществить проще, чем по протяженным признакам, поскольку качество совпадения имеет ясный физический смысл — число совпавших ключевых точек в данном положении. Также

можно использовать нечеткие меры, требующие не точного совпадения ключевых точек, а попадания расстояния между ключевыми точками в определенный диапазон с соответствующим весовым коэффициентом.

В целом методы с использованием ключевых точек достаточно производительны и надежны при условии, что на изображении объекта удается выделить необходимое для идентификации число ключевых точек. Ключевые точки практически нечувствительны к изменению освещенности и позволяют учитывать небольшие изменения ракурса. Тем не менее, они существенно менее надежны, чем, например, контурные и площадные признаки. Применяя ключевые точки, необходимо удостовериться, что используемый тип изображения будет содержать выбранные ключевые точки в числе, достаточном для решения задачи, а алгоритм выделения ключевых точек находит их с требуемой надежностью.

Признаки в виде пятен. Пятно представляет собой компактную в геометрическом смысле область точек со сходными статистическими характеристиками. Пятна используются в тех случаях, когда граница объекта диффузна, а соотношение яркости фона и объекта неизвестно, или объект имеет произвольный цвет. В этом случае изображение разбивается на области однородной раскраски, а затем выделяются такие области, которые располагаются в определенном соотношении и образуют искомый объект.

Характеристики, которые используются для описания и идентификации пятен, во многом совпадают с характеристиками бинаризованных объектов. В первую очередь это характеристики положения (положение центра масс, размер, ориентация, главной оси), формы (среднее отклонение точек пятна от центра масс в проекции на ось с наклоном φ как аналог центроидального профиля), цвета (средняя, медианная и преобладающая яркости, тон, насыщенность), характеристики текстуры и т. д.

Большое разнообразие идентификационных параметров позволяет легко находить похожие пятна объекта и образца, однако существенной проблемой является неоднозначность разбиения изображения на пятна. Как и в случае с контуром, изображения могут быть разбиты на пятна большим числом способов, и найти критерий оптимального разбиения достаточно сложно.

Другой проблемой является выбор меры близости: из-за неоднозначности разбиения пятна изображения и образца не будут в точности совпадать, поскольку некоторые пятна будут разбиты на более мелкие, некоторые могут слиться с элементами фона.

Мера близости

Простейшей мерой близости может служить сумма модулей или сумма квадратов отклонений параметров объекта от образца.

Рассмотрим одномерный случай. Пусть имеются одномерная функция $I(x)$ (рис. 4.2, *а*) и одномерный образец $S(x)$ (рис. 4.2, *б*), который надо найти на изображении. Будем накладывать образец на изображение и вычислять сумму квадратов отклонений (заштрихованные области на рис. 4.2, *а*). Эта сумма отклонений имеет смысл ошибки или отклонения фрагмента изо-

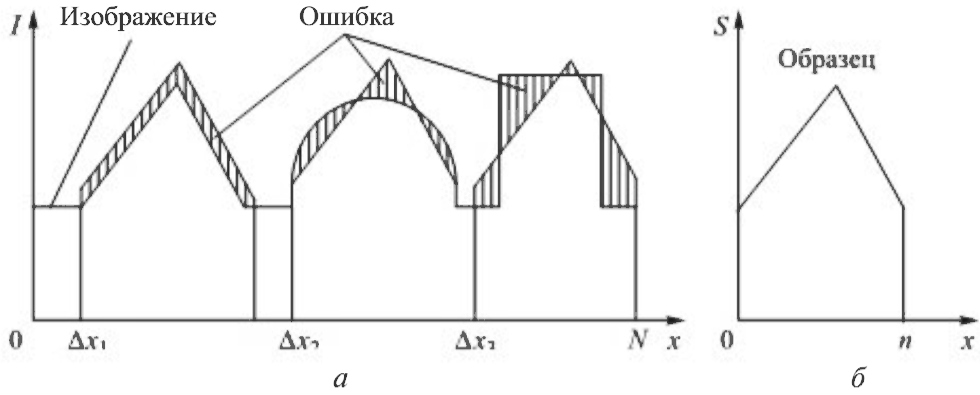


Рис. 4.2. Вычисление меры близости как суммы квадратов ошибки

бражения от образца $E(\Delta x)$: чем меньше $E(\Delta x)$ и чем меньше заштрихованная площадь на рис. 4.2, *а*, тем больше фрагмент похож на образец.

Отклонение $E(\Delta x)$ можно рассчитать как сумму модулей отклонений или как сумму квадратов отклонений:

$$E_1(\Delta x) = \sum_{x=0}^{n_x-1} \left| S[x] - I[x+\Delta x] \right|, \ \Delta x = 0, \ldots, N_x - n_{x-1};$$

$$E_2(\Delta x) = \sum_{x=0}^{n_x-1} \left(S[x] - I[x+\Delta x] \right)^2, \ \Delta x = 0, \ldots, N_x - n_{x-1}.$$

Первый вариант проще для непосредственного расчета и нечувствителен к большим шумовым выбросам, а второй вариант более удобен для аналитических преобразований.

Для двумерного случая (т. е. для обычного изображения) эти отклонения примут вид

$$E_1(\Delta x, \Delta y) = \sum_{x=0}^{n_x-1} \sum_{y=0}^{n_y-1} \left| S[x,y] - I[x+\Delta x, \ y+\Delta y] \right|;$$

$$E_2(\Delta x, \Delta y) = \sum_{x=0}^{n_x-1} \sum_{y=0}^{n_y-1} \left(S[x,y] - I[x+\Delta x, \ y+\Delta y] \right)^2; \qquad (4.1)$$

$$\Delta x = 0, \ldots, N_x - n_{x-1}, \quad \Delta y = 0, \ldots, N_y - n_{y-1},$$

где N_x, N_y — размеры изображения; n_x, n_y — размеры образца.

В общем случае, если необходимо найти и преобразование яркости g, и преобразование координат f, отклонения E_1 и E_2 могут быть записаны в виде функционала:

$$E_1(f,g) = \sum_{x=0}^{n_x-1} \sum_{y=0}^{n_y-1} \left| S[x,y] - g\left(I[f(x,y)] \right) \right|;$$

$$E_2(f,g) = \sum_{x=0}^{n_x-1} \sum_{y=0}^{n_y-1} \left(S[x,y] - g\left(I[f(x,y)] \right) \right)^2.$$

Корреляционная мера. Отклонение E_2 может быть записано также и в виде корреляционной функции. Для этого раскроем в выражении (4.1) квадрат:

$$E_2(\Delta x, \Delta y) = \sum_{x=0}^{n_x-1}\sum_{y=0}^{n_y-1}\big(S[x, y]\big)^2 + \sum_{x=0}^{n_x-1}\sum_{y=0}^{n_y-1}\big(I[x+\Delta x,\ y+\Delta y]\big)^2 - \\ - 2\sum_{x=0}^{n_x-1}\sum_{y=0}^{n_y-1} S[x, y]\, I[x+\Delta x,\ y+\Delta y].$$

Слагаемое $S[x, y]^2$ представляет собой суммарную энергию образца, она не зависит от положения (Δx, Δy) и может быть вычислена заранее. Слагаемое $I[\Delta x, \Delta y]^2$ — локальная энергия изображения (энергия фрагмента изображения), она зависит от положения (Δx, Δy), однако может быть вычислена за фиксированное число проходов (за три прохода) по изображению. Третий компонент — взаимнокорреляционная функция изображения и образца $R_{I,S}(\Delta x, \Delta y)$. Она имеет знак «—», и ее рост ведет к уменьшению отклонения E_2.

Таким образом, вычисление отклонения E_2 может быть сведено к вычислению корреляции $R_{I,S}$, а для корреляционной функции применима теорема о свертке:

$$F(R) = F(I)\, F^*(S),$$

где $F(\ )$ — операция вычисления преобразования Фурье.

Таким образом, корреляционную функцию можно вычислять в спектральной области, что приводит к существенному сокращению объема вычислений.

Точечные меры близости. Простой и логически прозрачной мерой близости объектов, заданных набором ключевых точек, является число совпадающих точек объекта и образца. Однако даже при небольших геометрических искажениях изображения (например, из-за небольшого изменения ракурса съемки) ключевые точки объекта и образца уже не будут точно совпадать и подсчет числа совпавших точек не даст результата.

Исправить ситуацию можно, если не требовать точного совпадения точек, а ввести некоторую допустимую погрешность ε: точки объекта и образца будут считаться совпадающими, если расстояния между ними не превышает ε. Такая модель достаточно хороша при большом числе ключевых точек.

Дальнейшее уточнение меры близости связано с введением понятия *взвешивания*: неточно совпавшие точки считаются не одной точкой, а некоторой долей, которая быстро убывает с ростом расстояния между точками. Соответствующий весовой коэффициент может быть, например, линейной, экспоненциальной и т. д. функцией расстояния (рис. 4.3).

Вид $W(\varepsilon)$ важен при сравнении по относительно небольшому (до 10) числу точек. Если же число точек превышает несколько десятков, то достаточно суммировать точки, попадающие в ε-окрестность: вид коэффициен-

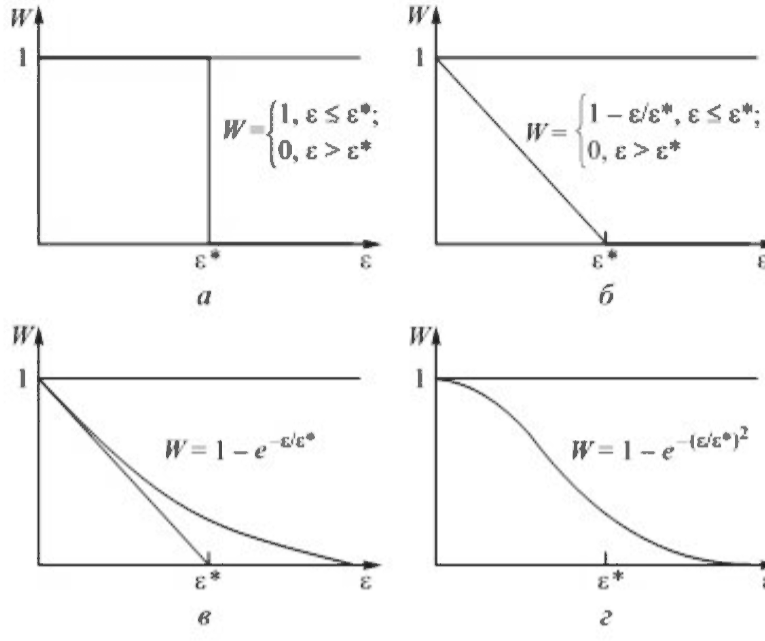


Рис. 4.3. Возможные варианты весовой функции $W(\varepsilon)$:
а — пороговая; *б* — линейная; *в* — экспонента; *г* — гауссоида

та $W(\varepsilon)$ практически не будет влиять на результаты. Примером такой ситуации является задача структурного изображения лиц на фотографиях. В этой задаче требуется группировать найденные ключевые точки — возможные положения структурных элементов лица (глаз, носа, рта, ушей и др.) — в соответствующие им лица с корректным взаимным расположением элементов. Однако взаимное положение элементов лица может изменяться в достаточно широких пределах — из-за крупных или мелких черт лица, округлой или вытянутой формы лица, поворота лица относительно любой из трех осей, изменения ракурса съемки и других факторов. Поэтому требуется не точное совпадение элементов, а их попадание в определенный диапазон. При этом важен сам факт попадания отклонения ε в данный диапазон, а не его значение. И наоборот, в задаче склейки аэрофотографий по маркерам важно именно суммарное отклонение ключевых точек двух кадров.

Помимо взаимного отклонения точек ε, взвешиваться может и сама ключевая точка. В этом случае весовой коэффициент может зависеть от типа точки (если имеются точки разных типов), от вероятности правильного обнаружения точки на изображении. Если точкам приписаны какие-либо дополнительные характеристики (величина угла для угловой точки, радиус кривизны и направление контура для точек высокой кривизны контура, радиус окружности для центра окружности и т. п.), то весовой коэффициент также может зависеть от меры близости этих характеристик.

Мера близости протяженных признаков. При сопоставлении отрезков по протяженным признакам (отрезкам, дугам, сплайнам и т. д.) мера близости принимает более сложный вид, чем при сопоставлении по точкам. В простейшем случае, когда изображение представлено набором отрезков, она

выражается в виде суммарной длины взаимно перекрывающихся отрезков изображения и образца (в пикселях). Однако проблема заключается в том, что перекрываться более чем в одной точке могут лишь параллельные отрезки, но из-за шума, эффектов дискретизации и других факторов отрезки могут поворачиваться на небольшой угол и становиться непараллельными соответствующим отрезкам образца.

Фактически «отрезки», из которых состоит контур объекта, — это не отрезки в геометрическом смысле, а прямоугольники шириной в одну точку и длиной, равной длине отрезка. В этом случае мерой близости может служить суммарная площадь перекрытия таких отрезков-прямоугольников S, равная площади ромба со стороной $a = 1/\sin\varphi$. Здесь φ — угол между отрезками (рис. 4.4); $S = \min(1/\sin\varphi, l_1, l_2)$, где l_1, l_2 — длина первого и второго отрезка соответственно.

Однако это лишь верхняя оценка площади пересечения. Реально площадь может быть меньше, если она выходит за пределы одного из отрезков (рис. 4.5).

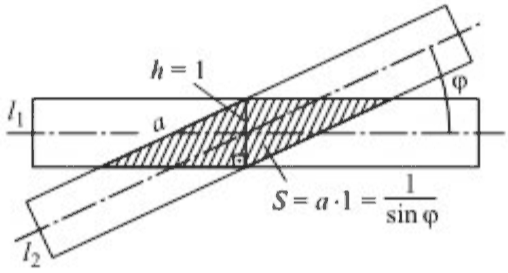


Рис. 4.4. Схема вычисления меры пересечения отрезков

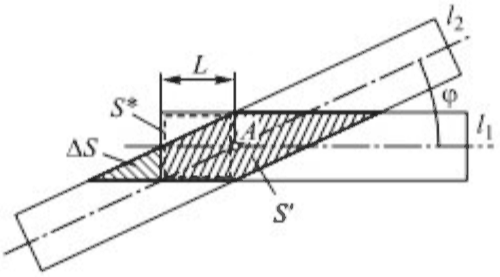


Рис. 4.5. Схема вычисления меры пересечения отрезков в случае, когда точка пересечения отрезков находится вблизи одного из их концов

Чтобы учесть излишнюю площадь пересечения, найдем точку пересечения отрезков A и расстояние L до ближайшего конца отрезка (см. формулу (3.1)). Таким образом, если $S/2 > L$, то площадь пересечения выходит за границы отрезка и половина площади пересечения должна быть заменена на площадь $S^* = L \cdot 1$:

$$S' = S/2 + L.$$

Аналогично следует проанализировать и остальные координаты концов отрезков. Имея точку пересечения отрезков, можно исключить из процесса поиска пары отрезков, которые вообще не пересекаются.

Мера близости отрезков также может учитывать с помощью весового коэффициента некоторые характеристики отрезков, облегчающие их идентификацию. Помимо ориентации такими характеристиками могут быть длина отрезка, число точек границы на отрезке (отрезки могут иметь разрывы либо быть штриховыми), средняя амплитуда градиента вдоль отрезка и т. д. Весовой коэффициент может, с одной стороны, ослаблять вклад ненадежно

выделяемых отрезков по сравнению с более надежными, а с другой — уменьшать вклад менее похожих пар отрезков по сравнению с более похожими.

Подобным образом может строиться сопоставление гетерогенных признаков, например точек границы изображения с отрезками образца. Такое представление позволяет избежать этапа выделения отрезков на изображении с возможной потерей части информации. Мерой близости в данном случае будет служить число точек границ, совпадающих с отрезками образца с допустимой точностью ε при выбранном пространственном преобразовании. Точку границы можно рассматривать как элементарный отрезок единичной длины, направление которого перпендикулярно градиенту яркости. Зная это направление, можно отобрать отрезки образца, параллельные контуру, и учитывать только их.

При сопоставлении точек границ с окружностями или эллипсами направление контура в этой точке позволяет находить пару точек окружности или эллипса на образце, с которыми она будет совпадать.

Сопоставление дуг. Сопоставление по другим признакам — дугам, эллипсам, сплайнам — строится во многом сходным образом. Как и в случае с отрезками, дугу на изображении следует заменить участком кольца единичной ширины. Мерой близости дуг объекта и образца будет сумма площадей перекрытия фрагментов колец изображения и образца.

Для сопоставления следует использовать дуги с совпадающими центрами и радиусами, поскольку в противном случае площадь перекрытия будет пренебрежимо мала.

Чтобы учесть возможное влияние шума на искажение радиуса и положение центра дуги, введем допустимое отклонение ε, в рамках которого центры и радиусы можно считать совпадающими.

Площадь перекрытия (в пикселях)

$$S = R|\varphi_2 - \varphi_1|,$$

где R — радиус дуг; φ_1, φ_2 — начальный и конечный углы первой и второй дуг (в радианах).

Нормирование мер близости. Во многих задачах требуется не только найти наилучшее значение меры близости, но и определить, является ли оно достаточным. Это может быть в тех случаях, когда на изображении присутствует несколько искомых объектов или искомый объект отсутствует. В этих случаях простое вычисление наилучшей меры близости задачу решить не позволит. Для решения подобных задач мера близости должна быть приведена в некоторый диапазон (желательно от 0 до 1), чтобы затем можно было выбрать пороговое значение, имеющее простой физический смысл, например желаемую вероятность обнаружения объекта.

Простейший вариант нормирования — деление меры на ее максимальное значение — в большинстве случаев дает неудовлетворительные результаты: максимальное значение на практике не может быть достигнуто, и значения скапливаются в небольшой окрестности нуля, что не решает проблему.

Одним из наиболее общих приемов нормирования является использование косинуса угла между сравниваемыми векторами. Косинус угла изменяется в диапазоне $[-1, 1]$, и его модуль можно трактовать как вероятность

совпадения векторов. Отрицательные значения косинуса угла трактуются как инверсия признаков объекта (например, инверсия цветов).

Рассмотрим нормирование корреляционной меры близости. Пусть S — образец размером $n \times n$, а $I(\Delta x, \Delta y)$ — фрагмент изображения того же размера $n \times n$, смещенный относительно начала координат на $(\Delta x, \Delta y)$. Будем рассматривать и образец, и фрагмент изображения как векторы размером $n \times n$.

Тогда косинус угла φ между векторами S и I

$$\cos\varphi[\Delta x, \Delta y] = \frac{\sum_{x=0}^{n-1}\sum_{y=0}^{n-1} S[x,y]I[x+\Delta x, y+\Delta y]}{\sqrt{\sum_{x=0}^{n-1}\sum_{y=0}^{n-1} S[x,y]^2}\sqrt{\sum_{x=0}^{n-1}\sum_{y=0}^{n-1} I[x+\Delta x, y+\Delta y]^2}} =$$

$$= \frac{R_{I,S}}{\sqrt{E_S}\sqrt{E_I[\Delta x, \Delta y]}} = R'_{I,S}[\Delta x, \Delta y],$$

где $R'_{I,S}$ — нормированная корреляционная функция; $R_{I,S}$ — исходная (ненормированная) корреляционная функция; E_S — энергия образца; E_I — энергия фрагмента изображения.

В числителе стоит ненормированная корреляционная функция $R_{S,I}$, в знаменателе — корень квадратный из произведения энергии образца и энергии фрагмента изображения. Получили хорошо известную формулу для нормирования широкого класса мер близости, имеющих корреляционную природу, если признаки объекта и образца представить в виде векторов.

Взвешенные меры близости. В практических задачах далеко не все точки образца равноценны: одни несут важную информацию, другие могут относиться к фону, шуму, бликам и т. д. Поэтому в ряде случаев целесообразно задать веса точек образца (а если образец представлен набором признаков — веса этих признаков).

В простейшем случае в качестве веса могут выступать постоянные множители, например, для случая взвешенной среднеквадратической ошибки E_2^W:

$$E_2^W[\Delta x, \Delta y] = \sum_{x=0}^{n_x-1}\sum_{y=0}^{n_y-1} W[x,y]\big(S[x,y] - I[x+\Delta x, y+\Delta y]\big)^2,$$

где $W[x, y]$ — весовой коэффициент точки $S[x, y]$ образца.

Раскрыв квадрат, получим

$$E_2^W[\Delta x, \Delta y] = \sum_{x-0}^{n_x-1}\sum_{y=0}^{n_y-1} W[x,y]S^2[x,y] + \sum_{x-0}^{n_x-1}\sum_{y=0}^{n_y-1} W[x,y]I^2[x+\Delta x, y+\Delta y] -$$

$$- \sum_{x-0}^{n_x-1}\sum_{y=0}^{n_y-1}\big[W[x,y]S[x,y]\big]I[x+\Delta x, y+\Delta y].$$

Первое слагаемое представляет собой взвешенную энергию образца, второе — взвешенную энергию фрагмента, третье — взвешенную корреляционную функцию.

Величины $W[x, y]S^2[x, y]$ и $W[x, y]S[x, y]$ рассчитывают заранее, взвешенная энергия фрагмента также представляет собой корреляционную функцию и для ускорения вычислений может быть подсчитана в спектральной области (через теорему о свертке).

Настройку коэффициентов $W[x, y]$ осуществляют с использованием обучающего набора образцов таким образом, чтобы взвешенная мера близости для примеров образцов была как можно выше, а для всех остальных — как можно ниже. Аналитическое нахождение коэффициентов $W[x, y]$ возможно, но, как правило, оно сводится к решению линейных матричных уравнений высокого порядка, поэтому здесь удобнее пользоваться оптимизационными методами. Поскольку однозначный выбор коэффициентов $W[x, y]$ возможен только при числе образцов не менее $n \times n$, при малом числе образцов следует вводить дополнительные ограничения на коэффициенты $W[x, y]$ во избежание их устремления в бесконечность. Можно, например, потребовать, чтобы их модуль или сумма их квадратов не превышали единицу.

Масочные меры близости. В ряде задач требуется, чтобы форма образца отличалась от прямоугольной, однако обрабатывать такие формы алгоритмически довольно сложно. Одним из возможных выходов из ситуации может быть использование маски.

Маска представляет собой массив размером $n \times n$, где соответствующие точки имеют разметку в виде единиц для точек, принадлежащих объекту, и нулей — для не принадлежащих. Тогда точки, не принадлежащие объекту, можно легко исключить из вычислений меры близости.

Например, для меры в виде среднеквадратической ошибки получим

$$E_2^M[\Delta x, \Delta y] = \sum_{x=0}^{n_x-1} \sum_{y=0}^{n_y-1} M[x, y]\big(S[x, y] - I[x + \Delta x, y + \Delta y]\big)^2,$$

где $M[x, y]$ — маска.

Для корреляционной меры

$$R_{I,S}^M[\Delta x, \Delta y] = \sum_{x=0}^{n_x-1} \sum_{y=0}^{n_y-1} \big(M[x, y]S[x, y]\big) I[x + \Delta x, y + \Delta y].$$

Значение $M[x, y]S[x, y]$ можно вычислить заранее. Для этого достаточно заполнить нулями точки $S[x, y]$, не принадлежащие искомому образцу и относящиеся к фону.

Следует обратить внимание, что заполнение нулями проводится после всех операций по нормированию образца (центрирования, коррекции яркости и т. п.), иначе вычисленные значения корреляционной функции будут сильно искажены: она весьма чувствительна к наличию постоянной аддитивной ошибки.

Стратегии оптимизации меры близости

Простейшей стратегией оптимизации (к сожалению, практически неработоспособной) является стратегия полного перебора возможных положений образца на изображении.

Для каждого возможного положения образца, задаваемого параметрами преобразования f, будем вырезать на изображении соответствующий фрагмент и преобразовывать его в систему координат образца (в соответствии с преобразованием f). Затем скорректируем яркости точек фрагмента в соответствии с преобразованием g, после чего вычислим меру близости между преобразованным таким образом фрагментом и образцом (рис. 4.6). Среди найденных таким образом мер близости вычислим наилучшее значение, параметры которого f и g и будут задавать искомое положение объекта на изображении.

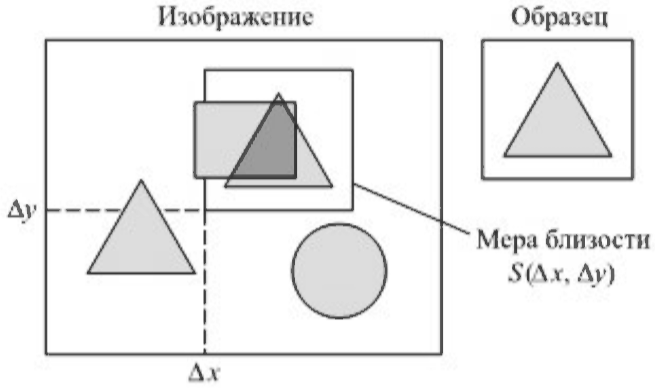


Рис. 4.6. Схема стратегии полного перебора

Для небольшого уменьшения объема вычислений преобразование g может быть задано табличным способом. В ряде случаев также можно избежать явного выполнения преобразования координат f над фрагментом изображения, если заранее подготовить несколько образцов объекта в разных ракурсах. Они получаются из первоначального объекта путем применения обратного преобразования координат f^{-1}. Однако в этом случае потребуется провести нормирование мер близости для каждого экземпляра образца, поскольку различные преобразования могут дать неодинаковое число точек в разных экземплярах образца. Корректное вычисление нормирующего коэффициента для выбранной меры близости может оказаться весьма сложной задачей.

Стратегия полного перебора на практике применяется в основном только для уточнения положения объекта и никогда — для полноценного поиска объекта. Это обусловлено тем, что она требует огромного объема вычислений.

Приведем простой пример. Пусть имеется изображение размером 2000×2000 точек (4 мП), на котором необходимо найти объект размером 1000×1000. Будем рассматривать самый простой в вычислительном плане случай: изображения объекта и образца получены в одинаковом ракурсе, с одинакового расстояния и при одинаковых условиях освещенности. Таким образом, неизвестны только параметры сдвига $(\Delta x, \Delta y)$, которые и предстоит определить. Число возможных положений образца на изображении равно $(2000 - 1000)\times$ $\times(2000 - 1000) = 10^6$. Для вычисления меры близости для каждого положения образца потребуется также $1000 \cdot 1000 = 10^6$ операций (например, операций возведения в квадрат для меры в виде суммы квадратов ошибок).

Итого суммарно потребуется 10^{12} операций. Много это или мало? Это очень много. Современный процессор i7 (2015) с пиковой производительностью $3 \cdot 10^{11}$ целых операций в секунду потратит на вычисления около получаса! Понятно, что в задачах реального времени такая стратегия просто неработоспособна. Если эту задачу еще усложнить, приблизив к на практике, например ввести необходимость определения размера объекта и его ориентации, то стратегия не сможет дать решения за сколько-нибудь разумное время.

Тем не менее для случая, когда число возможных значений параметров поиска относительно невелико, стратегию полного перебора все еще применяют на практике, например, в задачах уточнения положения объекта, когда основные параметры положения уже найдены другими, более производительными (но менее точными) методами и требуется лишь уточнение результата.

Радикально уменьшить объем вычислений можно двумя способами: сократить число признаков, по которым осуществляется поиск, или перейти к другому пространству, в котором мера близости будет вычисляться за небольшое число операций.

В первом случае вместо всех точек изображения используют лишь небольшое число особых точек (или других геометрических объектов). Особые точки позволяют сразу опускать положения, в которых не совпадает ни одна из точек. Кроме того, число точек, участвующих в вычислении меры близости, тоже существенно сокращается. Однако такой путь вовсе не без издержек. Во-первых, необходимо затратить определенное время на поиск ключевых точек. Во-вторых, снижается общая надежность поиска из-за возможного пропуска ключевых точек, ложного срабатывания. И наконец, выбранных ключевых точек может просто не оказаться на изображении или они будут в недостаточном количестве.

Во втором случае изображение предварительно преобразуется в другую область, в которой операции вычисления меры близости существенно упрощаются. Наиболее известным примером такого преобразования является преобразование Фурье. Оно позволяет заменить ресурсоемкую операцию вычисления кольцевой свертки на простую операцию попарного перемножения элементов спектра. Поэтому, если мера близости может быть представлена в виде кольцевой свертки (например, взаимнокорреляционная функция, сумма квадратов отклонений), то ее можно подсчитать в спектральной области, перемножив элементы спектра и уменьшив тем самым объем вычислений на порядки.

Аналогичным свойством обладает и менее известное дискретное теоретико-числовое преобразование (ДТЧП). В отличие от спектра Фурье, ДТЧП дает целочисленный спектр, более удобный для вычислений.

Другой пример преобразования области — обобщенное преобразование Хафа (Genevalized Hough Transform, GHT). В этом случае изображение преобразуется в пространство параметров поиска таким образом, что искомый объект стягивается в точку откликом, равным числу совпавших точек объекта и образца, т. е. со значением, равным мере близости. Теперь положение объекта можно найти в пространстве поиска как положение максимального отклика.

Разумеется, оба подхода — и уменьшение числа признаков, и пространственное преобразование — могут применяться одновременно.

Рассмотрим их более подробно.

Спектральные методы поиска. В основе спектральных методов лежит теорема о свертке, которая гласит, что преобразование Фурье от кольцевой свертки векторов $\vec{S}$ и $\vec{I}$ равно произведению спектра вектора $\vec{S}$ на комплексно-сопряженное спектра вектора $\vec{I}$:

$$F\left[\left(\vec{S}\otimes\vec{I}\right)\right]=F\left[\vec{S}\right]F^{*}\left[\vec{I}\right],$$

где $\left(\vec{S}\otimes\vec{I}\right)$ — кольцевая свертка $\vec{S}$ и $\vec{I}$; $F[\ ]$ — операция вычисления преобразования Фурье.

Для использования этого подхода меру близости R следует представить в виде кольцевой свертки и вычислить обратное преобразование Фурье:

$$R=F^{-1}\left[F(S)F^{*}(I)\right].$$

Пусть мера близости описывается в виде корреляционной функции $R_{S,I}$:

$$R_{S,I}[\Delta x,\Delta y]=\sum_{x=0}^{n_x-1}\sum_{y=0}^{n_y-1}S[x,y]I[x+\Delta x,y+\Delta y];$$

$$\Delta x=0,\ldots,N_x-n_{x-1},\quad \Delta y=0,\ldots,N_y-n_{y-1}.$$

Корреляционная функция представляет собой линейную свертку с обратным порядком следования аргументов. Преобразуем ее в кольцевую свертку. Для этого будем считать функцию $I[x, y]$ периодической с периодами N_x и N_y. Функцию $S[x, y]$ дополним нулями до размеров N_x и N_y и также будем считать периодической с периодами N_x и N_y.

Следует обратить внимание, что, если осуществляется нормирование или центрирование функции $S[x, y]$, то оно должно проводиться до дополнения нулями (рис. 4.7).

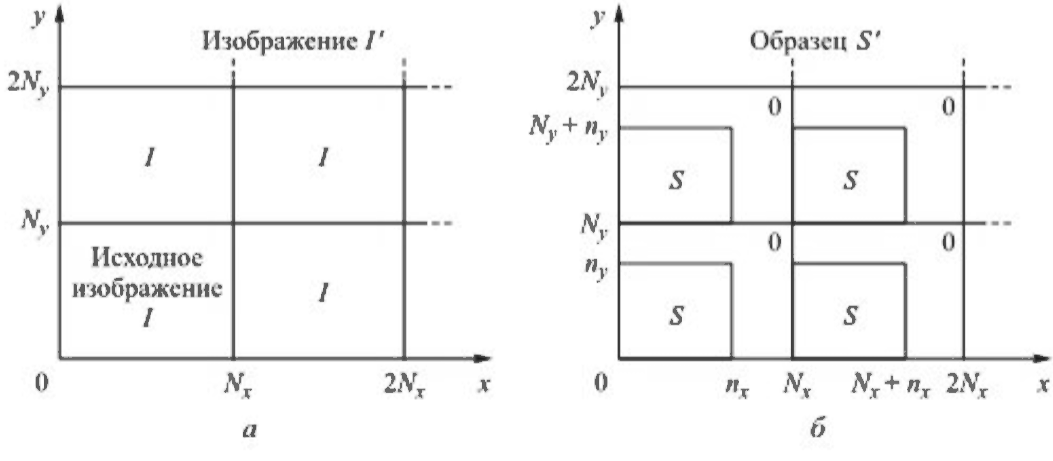


Рис. 4.7. Представление изображения (*а*) и образца (*б*) в виде двумерных периодических функций с периодами N_x и N_y

Для модифицированных таким образом функций S', I' можно записать

$$R_{S',I'}[\Delta x,\Delta y]=\sum_{x=0}^{n_x-1}\sum_{y=0}^{n_y-1} S[x,y]\,I[(x+\Delta x)\bmod N_x,(y+\Delta y)\bmod N_y]=$$

$$=(S\otimes I)=F^{-1}[F(S)F^{*}(I)].$$

Вычисление быстрого преобразования Фурье. Рассмотрим одномерный случай. Для него дискретное преобразование Фурье имеет вид

$$Z[k]=\sum_{n=0}^{N-1} x[n]W_N^{nk},\quad k=0,\ldots,N-1, \tag{4.2}$$

где $x[n]$ — исходная дискретная функция из N отсчетов; $z[k]$ — спектр функции $x[n]$, также содержащий N гармоник; W_N — комплексная константа,

$$W_N=e^{-2\pi j/N};\quad W_N^p=\cos\frac{2\pi p}{N}-j\sin\frac{2\pi p}{N}.$$

Вычисление спектра по формуле (4.2) потребует слишком большого объема вычислений, пропорционально N^2. Однако его можно существенно сократить, если воспользоваться симметрией функции W_N^p. Эта функция является периодической с периодом N, для нее будут выполняться следующие равенства:

$$W_N^0=1;$$

$$W_N^{N/2}=-1;$$

$$W_N^{N/4}=-j;$$

$$W_N^{3N/4}=j;$$

$$W_N^{Nk}=W_N^0=1 \text{ (для любого целого } k);$$

$$W_N^{N/2+Nk}=W_N^{N/2}=-1 \text{ (для любого целого } k).$$

Наиболее простой способ построить ускоренный алгоритм — разделить отсчеты функции $x[n]$ на четные и нечетные:

$$z[k]=\underbrace{\sum_{n=0}^{N/2-1} x[2n]W_N^{2nk}}_{\text{(четные: замена } n\to 2n\text{)}}+\underbrace{\sum_{n=0}^{N/2-1} x[2n+1]W_N^{nk}W_N^{k}}_{\text{(нечетные: замена } n\to 2n+1\text{)}}.$$

Здесь и далее будем считать N кратным 2. Если это не так, размер выборки может быть дополнен до длины N нулями.

Во втором слагаемом величина W_N^k не зависит от индекса суммирования k и ее можно вынести за скобки:

$$z[k]=\sum_{n=0}^{N/2-1} x[2n]W_N^{2nk}+W_N^k\sum_{n=0}^{N/2-1} x[2n+1]W_N^{2nk}. \tag{4.3}$$

Обозначив первую сумму как $A[k]=\sum_{n=0}^{N/2-1} x[2n]W_N^{2nk}$, а вторую — как $B[k]=\sum_{n=0}^{N/2-1} x[2n+1]W_N^{2nk}$, получим

$$z[k]=A[k]+W_N^k B[k].$$

Вычислим теперь $z[k + N/2]$. Подставим в (4.3) вместо k значение $k + N/2$:

$$z[k+N/2]=\sum_{n=0}^{N/2-1} x[2n]W_N^{2nk}W_N^{2nN/2}+W_N^kW_N^{N/2}\sum_{n=0}^{N/2-1} x[2n+1]W_N^{2nk}W_N^{2nN/2}.$$

Вследствие симметрии функции W_N^p, отмеченной выше, будут иметь место равенства

$$W_N^{2nN/2}=W_N^{nN}=1;$$
$$W_N^{N/2}=-1.$$

Последнее выражение можно записать в виде

$$z[k+N/2]=\sum_{n=0}^{N/2-1} x[2n]W_N^{2nk}-W_N^k\sum_{n=0}^{N/2-1} x[2n+1]W_N^{2nk}.$$

Очевидно, что первая сумма равна $A[k]$, а вторая — $B[k]$, поэтому

$$z[k+N/2]=A[k]-W_N^kB[k],\quad k=0,\ldots,N/2.$$

Таким образом, один раз вычислив значения $A[k]$ и $B[k]$, можно найти сразу две гармоники: $z[k]$ и $z[k + N/2]$, т. е. достаточно получить лишь половину значений $A[k]$ и $B[k]$.

Однако объем вычислений можно еще сократить. Рассмотрим выражения для $A[k]$ и $B[k]$:

$$A[k]=\sum_{n=0}^{N/2-1} x[2n]W_N^{2nk};$$
$$B[k]=\sum_{n=0}^{N/2-1} x[2n+1]W_N^{2nk},\quad k=0,\ldots,N/2-1.$$

Сравнивая их с выражением для дискретного преобразования Фурье, легко убедиться, что они также являются преобразованиями Фурье, но отдельно для четных и нечетных отсчетов функции $x[n]$. Следовательно, к ним можно применить тот же подход.

Ограничимся рассмотрением $A[k]$. Разобьем вновь члены суммы на четные и нечетные:

$$A[k]=\underbrace{\sum_{n=0}^{N/4-1} x[4n]W_N^{4nk}}_{\{\text{четные: замена } n\to 2n\}}+\underbrace{\sum_{n=0}^{N/4-1} x[4n+2]W_N^{2nk}W_N^{2k}}_{\{\text{нечетные: замена } n\to 2n+1\}}.$$

Снова обозначив первую и вторую суммы:

$$C[k]=\sum_{n=0}^{N/4-1}x[4n]W_N^{4nk};\qquad D[k]=\sum_{n=0}^{N/4-1}x[4n+2]W_N^{4nk}$$

получим

$$A[k]=C[k]+W_N^{2k}D[k],\quad k=0,\ldots,N/4-1.$$

Вычислим теперь $A[k+N/4]$, подставив вместо k значение $k+N/4$, тогда

$$A[k+N/4]=\sum_{n=0}^{N/2-1}x[2n]W_N^{2nk}W_N^{2nN/2}+W_N^{2k}W_N^{2N/2}\sum_{n=0}^{N/2-1}x[2n+1]W_N^{2nk}W_N^{2nN/2}=$$

$$=\sum_{n=0}^{N/2-1}x[2n]W_N^{2nk}-W_N^{2k}\sum_{n=0}^{N/2-1}x[2n+1]W_N^{2nk}.$$

Заметив, что первая сумма есть $C[k]$, а вторая — $D[k]$, запишем

$$A[k+N/4]=C[k]-W_N^{2k}D[k],\quad k=0,\ldots,N/4-1.$$

Найдя коэффициенты $C[k]$ и $D[k]$, можем использовать их для вычислений сразу двух значений $A[k]$ и $A[k+N/4]$, которые в свою очередь участвуют в вычислениях уже четырех значений $z[k]$.

Формула для $B[k]$ выводится аналогично. В итоге для $B[k]$ получим

$$B[k]=E[k]+W_N^{2k}F[k];$$

$$B[k+N/4]=E[k]+W_N^{2k}F[k],$$

где $E[k]=\sum_{n=0}^{N/4-1}x[4n+1]W_N^{4nk}$, $F[k]=\sum_{n=0}^{N/4-1}x[4n+3]W_N^{4nk}$, $k=0,\ldots,N/4-1$.

Данный процесс можно продолжать для выражений C, D, E, F до тех пор, пока под знаком суммы не останется только одно значение x. Очевидно, этот процесс можно выполнять $\log_2 N$ раз, и каждый раз объем вычислений уменьшается вдвое. Суммарное число операций в таком алгоритме будет пропорционально $N\log_2 N$ по сравнению с N^2 в расчетах по формуле (4.2). При $N=1024$ выигрыш составит 100 раз. Поэтому такую схему называют быстрым преобразованием Фурье (БПФ, или Fast Fourier Transform, FFT).

Как будет выглядеть итоговая схема алгоритма? Обратим внимание, что для каждого коэффициента A, B, C, D и т. д., выполняется одна и та же операция: на вход поступают коэффициенты P и Q, второй из них умножается на W_N^p и складывается (или вычитается) с первым (рис. 4.8). Такие элементы называют «бабочками», из которых и строится схема алгоритма.

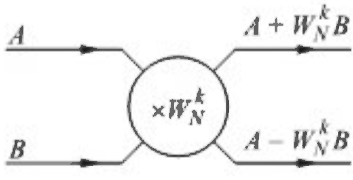


Рис. 4.8. Базовый оператор быстрого преобразования Фурье

Рассмотренная схема алгоритма называется алгоритмом Кули — Тьюки.

Пример. Схема быстрого преобразования Фурье для $N = 16$ показана на рис. 4.9.

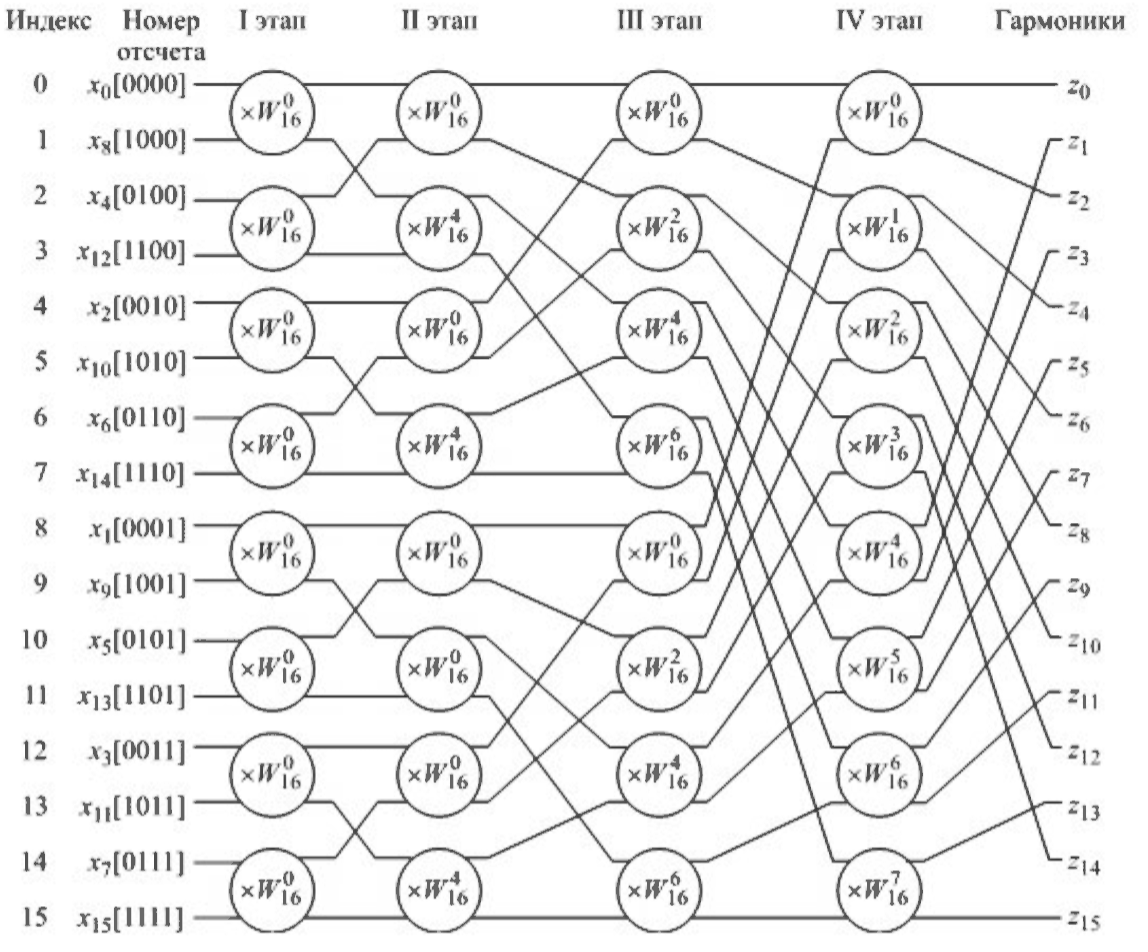


Рис. 4.9. Схема быстрого преобразования Фурье для $N = 16$

Следует отметить, что отсчеты функции $x[n]$ подаются на схему в обратном битовом порядке. Чтобы узнать, какой отсчет подается на i-й вход, нужно двоичный код i записать в обратном порядке, младшими битами направо. При реализации алгоритма стоит сразу рассчитать таблицу перестановок для отсчетов $x[n]$.

Элементы спектра $z[k]$ снимаются в обычном (прямом) битовом порядке. На примере видно, что число операций комплексного умножения при $N = 16$ будет $8 \cdot 4 = 32$, вместо $16 \cdot 16 = 256$ как в оригинальной схеме. В действительности число операций нетривиального умножения еще меньше, поскольку $W_{16}^0 = 1$ (15 операций умножения), далее $W_{16}^4 = -j$ (7 операций умножения) и общее число операций умножения можно снизить до $32 - 15 - 7 = 10$. Этого значения легко достичь, если делить $z[k]$ не на две, а на четыре части.

Отметим, что верхняя и нижняя части схемы идентичны и не связаны друг с другом вплоть до последнего этапа. Таким образом, можно легко

масштабировать эту схему путем дублирования и добавления нового этапа для любого размера, кратного 2^N.

Особенности программной реализации. Пусть имеется функция $x[n]$, отсчеты которой отсортированы в обратном битовом порядке, и она задана в виде массива $x[n]$. Обратный битовый номер отсчета будем называть *индексом*. Индексы, в отличие от номера отсчета, идут в прямом битовом порядке.

Из приведенного на рис. 4.9 примера видно, на I этапе первая «бабочка» обрабатывает значения $x[0000_2]$ и $x[0001_2]$; вторая — $x[0010_2]$ и $x[0011_2]$ и т. д. (табл. 4.1).

Индексы этих элементов отличаются только в младшем бите. Если его убрать, получится номер «бабочки», которая должна проводить операцию над данными отсчетами. И наоборот, чтобы по номеру «бабочки» (они нумеруются с нуля) получить индексы отсчетов, надо дописать справа соответственно нуль для первого отсчета и единицу для второго:

$$N_1 = 2k + 0 \cdot 2^p;$$
$$N_2 = 2k + 1 \cdot 2^p.$$

Операция умножения на 2 сдвигает номер «бабочки» на единицу влево, а в освободившийся младший бит записывается соответственно нуль или единица. Для сокращения объема вычислений операцию умножения на два можно заменить сложением с собой или сдвигом влево.

На II этапе будут проходить следующие загрузки:

первая «бабочка» — из $x[0000_2]$ и $x[0010_2]$;

вторая — из $x[0001_2]$ и $x[0011_2]$ и т. д. (см. табл. 4.1).

Индексы x для каждой «бабочки» отличаются только во втором слева бите. Если его вычеркнуть, получим номер «бабочки»

Формула для индексов k-й «бабочки» более сложная. Для ее вывода нужно разбить номер бабочки на правую (до разряда $p = 1$) и левую (разряды p и старше) части, например при помощи битовой маски. Маска *Mask* для выделения правой и левой частей номера бабочки имеет вид

$$Mask = 2^p - 1,$$

где p — номер этапа, начиная с нуля (первый этап имеет номер 0, второй — 1, i-й — $i - 1$). Напомним, что операцию возведения двойки в степень p можно заменить операцией бинарного сдвига влево:

$$2^p = 1 \; shl \; p.$$

Маска содержит $p - 1$ единиц в младших разрядах, соответствующих правой части номера бабочки, и остальные — нули, соответствующие левой части.

Правая часть номера «бабочки»:

$$R = k \; and \; Mask;$$

левая часть номера «бабочки»:

$$L = k - R.$$

Таблица 4.1

Порядок обмена в быстром преобразовании Фурье

Номер бабочки		Операции обмена			
(10)	(2)	I этап	II этап	III этап	IV этап
0	000	(0) 0000 ↔ 0001 (1)	(0) 0000 ↔ 0010 (2)	(0) 0000 ↔ 0100 (4)	(0) 0000 ↔ 1000 (8)
1	001	(2) 0010 ↔ 0011 (3)	(1) 0001 ↔ 0011 (3)	(1) 0001 ↔ 0101 (5)	(1) 0001 ↔ 1001 (9)
2	010	(4) 0100 ↔ 0101 (5)	(4) 0100 ↔ 0110 (6)	(2) 0010 ↔ 0110 (6)	(2) 0010 ↔ 1010 (10)
3	011	(6) 0110 ↔ 0111 (7)	(5) 0101 ↔ 0111 (7)	(3) 0011 ↔ 0111 (7)	(3) 0011 ↔ 1011 (11)
4	100	(8) 1000 ↔ 1001 (9)	(8) 1000 ↔ 1010 (10)	(8) 1000 ↔ 1100 (12)	(4) 0100 ↔ 1100 (12)
5	101	(10) 1010 ↔ 1011 (11)	(9) 1001 ↔ 1011 (11)	(9) 1001 ↔ 1101 (13)	(5) 0101 ↔ 1101 (13)
6	110	(12) 1100 ↔ 1101 (13)	(12) 1100 ↔ 1110 (14)	(10) 1010 ↔ 1110 (14)	(6) 0110 ↔ 1110 (14)
7	111	(14) 1110 ↔ 1111 (15)	(13) 1101 ↔ 1111 (15)	(11) 1011 ↔ 1111 (15)	(7) 0111 ↔ 1111 (15)

Индексы элементов для обмена:

$$N_1 = 2L + 0 \cdot 2^p + R;$$

$$N_2 = 2L + 1 \cdot 2^p + R.$$

Эта формула справедлива для любого этапа с номером p.

Двумерное преобразование Фурье. Поскольку изображение и образец являются двумерными функциями, для вычисления их корреляции следует использовать двумерное преобразование Фурье:

$$Z[v,w] = \sum_{k=0}^{N-1} \sum_{n=0}^{N-1} x[k,n] W_N^{kv+nw}, \quad v = 0, \ldots, N-1, \ w = 0, \ldots, N-1,$$

где $x[k, n]$ — исходная дискретная двумерная функция из $N \times N$ отсчетов; $z[v, w]$ — двумерный спектр функции $x[k, n]$, также содержащий $N \times N$ гармоник; W_N — комплексная константа,

$$W_N = e^{-2\pi j/N}; \quad W_N^p = \cos\frac{2\pi p}{N} - j\sin\frac{2\pi p}{N}.$$

Двумерное преобразование Фурье может быть вычислено с использованием нескольких одномерных преобразований Фурье. Для этого сначала вычисляют одномерное ДПФ от каждой строки y, а потом снова находят ДПФ уже от столбцов получившегося результата:

$$Y[v,w] = \sum_{n=0}^{N-1} x[n,w] W_N^{vn};$$

$$Z[v,w] = \sum_{k=0}^{N-1} Y[v,k] W_N^{wk}, \quad v = 0, \ldots, N-1, \ w = 0, \ldots, N-1.$$

Число операций в таком случае составит $2N^2 \log_2 N$.

При вычислениях двумерного БПФ на универсальных процессорах с защищенным режимом необходимо помнить, что операции над строками больших массивов из-за особенностей пересчета адресов выполняются быстрее, чем над столбцами. Поэтому матрицу Y перед вычислением Z желательно транспонировать, а после окончания вычислений — транспонировать матрицу Z.

Вычислять двумерное БПФ через одномерные достаточно просто и удобно, если имеется аппаратная поддержка вычисления одномерного БПФ, например в виде команд сигнального процессора для встраиваемых ЭВМ либо библиотечных функций графических ускорителей для универсальных и суперЭВМ. При отсутствии аппаратной поддержки одномерного БПФ замена ДПФ на набор одномерных зачастую неоправданна, так как требует большого объема вычислений.

Схема быстрого двумерного преобразования Фурье. Как и для одномерного преобразования, разобьем сумму на четные и нечетные компоненты. Теперь подсумм станет не две, а четыре, поскольку оба индекса суммирования могут быть как четными, так и нечетными:

$$z[v,w]=\underbrace{\sum_{k=0}^{N/2-1}\sum_{n=0}^{N/2-1}x[2k,2n]W_N^{2kv+2nw}}_{\{\text{четное } k\to 2k,\ \text{четное } n\to 2n\}}+\underbrace{\sum_{k=0}^{N/2-1}\sum_{n=0}^{N/2-1}x[2k,2n+1]W_N^{2kv+2nw+w}}_{\{\text{четное } k\to 2k,\ \text{нечетное } n\to 2n+1\}}+$$

$$+\underbrace{\sum_{k=0}^{N/2-1}\sum_{n=0}^{N/2-1}x[2k+1,2n]W_N^{2kv+2nw+v}}_{\{\text{нечетное } k\to 2k+1,\ \text{четное } n\to 2n\}}+\underbrace{\sum_{k=0}^{N/2-1}\sum_{n=0}^{N/2-1}x[2k+1,2n+1]W_N^{2kv+2nw+v+w}}_{\{\text{нечетное } k\to 2k+1,\ \text{нечетное } n\to 2n+1\}}=$$

$$=A[v,w]+W_N^w B[v,w]+W_N^v C[v,w]+W_N^{v+w}D[v,w],$$

где

$$A[v,w]=\sum_{k=0}^{N/2-1}\sum_{n=0}^{N/2-1}x[2k,2n]W_N^{2kv+2nw};$$

$$B[v,w]=\sum_{k=0}^{N/2-1}\sum_{n=0}^{N/2-1}x[2k+1,2n]W_N^{2kv+2nw};$$

$$C[v,w]=\sum_{k=0}^{N/2-1}\sum_{n=0}^{N/2-1}x[2k,2n+1]W_N^{2kv+2nw};$$

$$D[v,w]=\sum_{k=0}^{N/2-1}\sum_{n=0}^{N/2-1}x[2k+1,2n+1]W_N^{2kv+2nw}.$$

Вычислим $Z[v + N/2, w]$, $Z[v, w + N/2]$, $Z[v + N/2, w + N/2]$:

$$Z[v+N/2,w]=A[v,w]+W_N^w B[v,w]-W_N^v C[v,w]-W_N^{v+w}D[v,w];$$

$$Z[v,w+N/2]=A[v,w]-W_N^w B[v,w]+W_N^v C[v,w]-W_N^{v+w}D[v,w];$$

$$Z[v+N/2,w+N/2]=A[v,w]-W_N^w B[v,w]-W_N^v C[v,w]+W_N^{v+w}D[v,w].$$

Таким образом, один набор коэффициентов A, B, C, D используется для вычисления сразу четырех коэффициентов Z. Сама операция будет представлять собой «бабочку», но уже с четырьмя «крыльями» для четырех входных значений A, B, C, D и четырех выходных. Графическую схему двумерного БПФ здесь не приводим ввиду ее громоздкости.

Алгоритм двумерного БПФ строится аналогично алгоритму одномерного, за исключением того, что индексов входных значений будет не один, а два.

1. Отсортировать входные данные в обратном битовом порядке. Далее для каждого этапа $p = 0, \ldots, \log_2 N - 1$:
   для каждой «бабочки» с индексами $x = 0, \ldots, N/2 - 1$, $y = 0, \ldots, N/2 - 1$:
      вычислить индексы обмена ID:
      $Mask = 2^P - 1$,
      $R_x = x$ *and* $Mask$, $L_x = x - R_x$,
      $R_y = y$ *and* $Mask$, $L_y = y - R_y$,
      $\mathrm{ID}_{0X} = 2L_x + 0*2^P + R_x$,
      $\mathrm{ID}_{1X} = 2L_x + 1*2^P + R_x$,
      $\mathrm{ID}_{0Y} = 2L_y + 0*2^P + R_y$,

$$ID_{1Y} = 2L_y + 1*2^P + R_y.$$

2. Вычислить выходы «бабочки» x, y:

$$Z_{00} = X[ID_{0X}, ID_{0Y}] + W^w X[ID_{1X}, ID_{0Y}] + W^v X[ID_{0X}, ID_{1Y}] + W^{w+v} X[ID_{1X}, ID_{1Y}];$$
$$Z_{01} = X[ID_{0X}, ID_{0Y}] + W^w X[ID_{1X}, ID_{0Y}] - W^v X[ID_{0X}, ID_{1Y}] - W^{w+v} X[ID_{1X}, ID_{1Y}];$$
$$Z_{10} = X[ID_{0X}, ID_{0Y}] - W^w X[ID_{1X}, ID_{0Y}] + W^v X[ID_{0X}, ID_{1Y}] - W^{w+v} X[ID_{1X}, ID_{1Y}];$$
$$Z_{11} = X[ID_{0X}, ID_{0Y}] - W^w X[ID_{1X}, ID_{0Y}] - W^v X[ID_{0X}, ID_{1Y}] + W^{w+v} X[ID_{1X}, ID_{1Y}].$$

3. Записать вычисленные значения Z обратно в память:

$$X[ID_{0X}, ID_{0Y}] = Z_{00};$$
$$X[ID_{1X}, ID_{0Y}] = Z_{01};$$
$$X[ID_{0X}, ID_{1Y}] = Z_{10};$$
$$X[ID_{1X}, ID_{1Y}] = Z_{11}.$$

По окончании в массиве X получаем спектр в прямом битовом порядке. Число операций для изображения $N \times N$ составит

$$\underbrace{\frac{N}{2}\frac{N}{2}}_{\text{«бабочек»}} \underbrace{(\log_2 N)}_{\text{этапов}} \underbrace{\cdot 3}_{\text{умножения}} = \frac{3}{4} N^2 \log_2 N,$$

что существенно меньше, чем в случае использования последовательности одномерных преобразований.

Обратное преобразование Фурье. Обратное преобразование Фурье вычисляется по формуле

$$x[k,n] = \frac{1}{N^2} \sum_{v=0}^{N-1} \sum_{w=0}^{N-1} z[v,w] W_N^{-kv-nw}, \quad k = 0, \ldots, N-1, n = 0, \ldots, N-1.$$

Очевидно, что обратное преобразование Фурье отличается от прямого только знаком «—» в показателе степени при W_N (постоянный множитель $1/N^2$ перед суммой следует опустить, поскольку в реальных вычислениях он не участвует и учитывается алгоритмически). Таким образом, для получения алгоритма обратного преобразования Фурье достаточно поменять знак в показателе W_N. Поскольку W_N^p всегда задается табличным способом, то для получения обратного преобразования достаточно задать другую таблицу $W_N[p]$.

Свойства стратегии с использованием БПФ. Основным свойством стратегии на основе БПФ является ее весьма высокая вычислительная эффективность. Количество вычислительных операций даже в самом простом (рассмотренном выше) методе имеет порядок $O(N^2 \log_2 N)$.

Для приведенного выше примера $N = 2048$, тогда число операций составит примерно $3 \cdot 11 \cdot (2048)^2 = 1{,}32 \cdot 10^8$ операций. Процессор i7 затратит на эту задачу примерно 10 с.

Если этого недостаточно, можно применить аппаратное ускорение с помощью Фурье-процессоров. Алгоритм имеет высокий потенциал к распараллеливанию и количество вычислительных операций может быть доведено до $O(\log_2 N)$.

Метод также довольно устойчив к случайным шумам и небольшим искажениям формы объекта. Он может быть применен не только к исходному изображению, но и к его производным, к найденным границам и контурам. Это не повышает производительность, но обеспечивает снижение чувствительности к преобразованиям яркости.

Тем не менее у этой стратегии есть и недостатки:

1) позволяет определить только параметры параллельного переноса. Если необходимо также определить ориентацию объекта, его размеры, ракурс, придется создавать экземпляры образца для всех возможных ракурсов и ориентаций и последовательно искать их на изображении;

2) не позволяет подбирать модель преобразования яркости $g(I)$, что делает его чувствительным к различиям в освещенности объекта и образца. Это требует дополнительных методов оценки модели освещенности, а также удаления крупных шумов (теней, облаков и т. п.), наличие которых не позволяет построить корректную модель освещенности;

3) в явном виде работает только с квадратными изображениями, прямоугольные приходится обрезать или дополнять нулями до квадрата;

4) позволяет учитывать только один параметр точки — как правило, ее яркость. Одновременный учет цвета, оттенка радиуса кривизны, характеристик текстуры сильно затруднен, а учет циклических признаков (например, углов и направлений) — невозможен.

Рассмотренный алгоритм Кули — Тьюки является наиболее простым, логически прозрачным и весьма удобным для аппаратной реализации. Поэтому в большинстве практических приложений используется именно он. Однако этот алгоритм не является единственным.

Если разбивать суммы не на четные и нечетные, а на старшие и младшие, то получим несколько иную схему (с другой структурой и другим оператором-«бабочкой»), которая представляет собой зеркальное отображение рассмотренной ранее: на вход будут подаваться отсчеты N в прямом битовом порядке, а гармоники z будут сниматься в обратном битовом порядке. Использование такой схемы позволяет избежать этапа сортировки входных данных и итоговой корреляционной функции в обратный битовый порядок. В остальном алгоритм полностью идентичен предыдущему.

Схема RADIX-4. Если разбивать суммы не на две, а на четыре части, получим схему RADIX-4, которая будет требовать на четверть меньше операций умножения, однако размер входных данных N должен быть степенью четырех.

Схема с разбиением на простые произвольные множители. Разбивать сумму на две или четыре подсуммы — не единственная возможность. Можно разбить ее на три или пять подсумм, и получится аналогичная схема, но с несколько иными операторами. Более того, для каждого этапа алгоритма можно использовать свои разбиения. Это удобно, когда размеры изображения не являются степенью небольшого простого числа, но являются произведением степеней нескольких таких чисел.

Другие преобразования. Преобразование Фурье не единственное, для которого выполняется теорема о свертке. Наиболее известным аналогом преобразования Фурье на кольце целых чисел является дискретное теоретико-числовое преобразование в базисах простых чисел (например, чисел Ферма или Мерсена). Эти преобразования удобны тем, что спектр получается не комплексным, а целочисленным. Однако для его реализации требуется аппаратная поддержка арифметических операций по модулю, которых нет

в составе команд универсальных процессоров. Поэтому на данный момент использование ДТЧП ограничено в основном спецвычислителями.

Стратегия поиска объекта с помощью обобщенного преобразования Хафа. Обобщенное преобразование Хафа переносит точки пространства изображения в новое пространство (параметров поиска) так, что все точки объекта будут стянуты в одну. Аналог яркости в этой точке называется *откликом* объекта, он равен числу точек объекта на изображении, совпадающих с образцом, а координаты точки в ПП будут соответствовать координатам объекта на изображении. Тогда задачу поиска объекта можно свести к гораздо более простой задаче поиска максимального отклика в ПП.

Обобщенное преобразование Хафа можно описать следующим образом. Пусть имеется образец S, заданный в виде списка точек (A, B, C, ...). Это могут быть, например, точки границ объекта, положения углов, областей высокой кривизны и т. п. Пусть также имеется изображение I, представленное в виде списка точек (A', B', C', ...), которые могут принадлежать или не принадлежать объекту (рис. 4.10). Будем считать, что для каждой точки изображения можно указать точки объекта, которые ей соответствуют. Это соответствие точек определяется по приближенному равенству их параметров — яркости, направления контура, радиуса его кривизны и т. д.

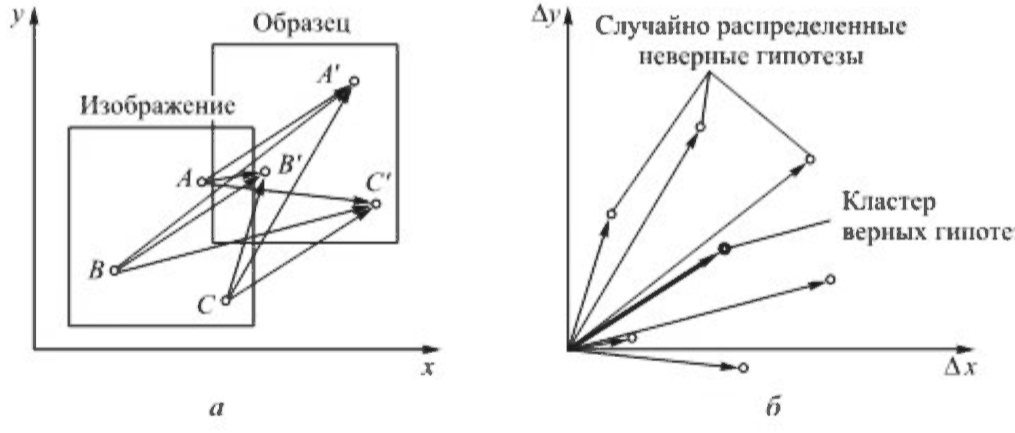


Рис. 4.10. Вычислительная схема обобщенного преобразования Хафа:
а — пространство координат изображения; *б* — пространство поиска

Каждая пара соответствующих точек объекта и изображения создает одну гипотезу H_{ij} о совпадении этих точек и, соответственно, о положении объекта на изображении:

$$H_{ij} = (\Delta x, \Delta y) = (x_i - x_j', y_i - y_j').$$

Если перенести все возможные гипотезы в пространство поиска $H(\Delta x, \Delta y)$, то все верные гипотезы совпадут, образуя суммарный отклик $H^*(\Delta x^*, \Delta y^*)$, а неверные гипотезы будут распределены хаотически, не создавая существенных максимумов. Значение отклика H^* равно числу совпавших точек, а координаты отклика $(\Delta x^*, \Delta y^*)$ — искомому положению объекта на изображении (см. рис. 4.10).

Для вычисления положения максимальных откликов представим **ПП** в виде двумерного массива счетчиков $H[x, y]$. Будем последовательно рассматривать все возможные гипотезы и для каждой из них увеличивать соответствующий счетчик.

По окончании перебора гипотез найдем локальные максимумы в массиве $H[x, y]$, положение которых будет соответствовать наиболее вероятным положениям объекта на изображении.

Свойства стратегии на основе обобщенного преобразования Хафа. Охарактеризуем данную стратегию в терминах соотношения производительности, точности и надежности. Количество вычислительных операций на этапе разметки гипотез будет равно произведению числа используемых точек изображения и среднего числа точек образца, соответствующих каждой точке изображения. Количество вычислительных операций на этапе поиска максимума в **ПП** равно числу ячеек массива H, представляющего **ПП**.

Рассмотрим пример: пусть образец размером 1000×1000 точек надо найти на изображении размером 2000×2000 точек с использованием точек границ, число которых обычно составляет 10...15 % его площади. Для упрощения вычислений возьмем нижнюю границу и получим $0{,}1 \cdot 2000^2 = 0{,}4 \cdot 10^6$ точек, аналогично число точек образца будет $0{,}1 \cdot 1000^2 = 0{,}1 \cdot 10^6$. Пусть направления границ разбиты с шагом 10°, тогда каждой точке изображения будет соответствовать в среднем $0{,}1 \cdot 10^6/36 \approx 2800$ точек образца.

Количество вычислительных операций на этапе разметки гипотез составит $0{,}4 \cdot 10^6 \cdot 2800 \approx 1{,}1 \cdot 10^9$, на этапе поиска максимума — $1000 \cdot 1000 = 10^6$. Видно, что это пренебрежимо мало по сравнению со значениями аналогичных параметров на предыдущем этапе.

Полученный объем вычислений больше, чем для стратегии с **БПФ**, однако в данном случае не содержится «длинных» операций типа умножения и деления, поэтому итоговое время вычислений на последовательном процессоре примерно сопоставимо.

Наиболее сложной проблемой при аппаратной реализации обобщенного преобразования Хафа является существенное число обращений к памяти, реализующей пространство поиска. В связи с этим непонятно, возможна ли эффективная параллельная реализация данной стратегии, но и последовательная реализация имеет вполне достаточную производительность для современных универсальных процессоров.

В качестве опорных точек для генерации гипотез, как правило, используются геометрические признаки (например, точки границ). Использование таких точек дает стратегии на основе обобщенного преобразования Хафа низкую чувствительность к изменениям условий видимости объекта, поскольку они практически не влияют ни на координаты опорных точек, ни на вероятность их обнаружения.

Стратегия на основе обобщенного преобразования Хафа позволяет учитывать дополнительные параметры точек (направление градиента, радиус кривизны контура и т. п.), что делает ее гораздо надежнее методов, основанных на расчете корреляции (при использовании одинаковых наборов признаков, например точек границ).

Вместе с тем, стратегия весьма чувствительна к изменениям ракурса съемки (наличию неучтенных геометрических искажений). При изменении ракурса верные гипотезы не будут попадать в одну общую точку в ПП, а будут распределены по некоторой окрестности достаточно нерегулярным и существенно негауссовым образом, который зависит как от формы самого объекта, так и от характера изменения ракурса.

Размытие отклика приводит к значительному снижению его значения: например, если максимум размылся всего на четыре соседние точки, его значение может снизиться в 4 раза. Это может привести к тому, что отклик объекта будет потерян в откликах помех. В то же время существует возможность преодолеть или существенно скомпенсировать этот недостаток путем дополнительной обработки ПП.

Стратегия на основе обобщенного преобразования Хафа также нечувствительна к отсутствию довольно большого числа точек объекта: отклик в ПП все равно будет равен числу оставшихся точек, сколько бы мало их ни было.

Особенностью обобщенного преобразования Хафа является размытие отклика в ПП при наличии геометрических искажений объекта. Размытие имеет сложную форму, следствием чего является несовпадение нового локального максимума с положением центра пятна (Δx^*, Δy^*). Поэтому помимо уменьшения значения отклика, что само по себе снижает надежность обнаружения объекта, происходит также и смещение его положения относительно реального положения.

На рис. 4.11 видно, почему возникает смещение положения максимума: наибольший отклик будет соответствовать совпадению одного из углов прямоугольников, тогда как правильное положение (Δx^*, Δy^*) не даст ни одного отклика.

Таким образом, при наличии геометрических искажений, нескомпенсированных предобработкой и самим процессом поиска, требуется дополнительная обработка ПП для восстановления положения и значения отклика правильных гипотез. Это необходимо для повышения и надежности, и точности.

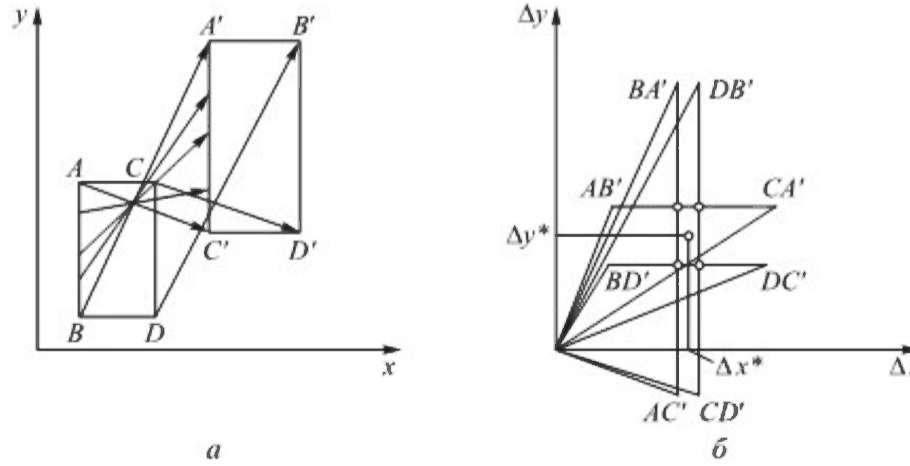


Рис. 4.11. Отклик объекта при наличии геометрических искажений:
а — пространство координат изображения; *б* — пространство поиска

Компенсация геометрических искажений. При наличии геометрических искажений найти положение объекта, при котором точки его границ совпадут с границами изображения, будет невозможно. Соответственно, в пространстве поиска верные гипотезы уже не будут совпадать, а будут занимать некоторую небольшую область — *кластер верных гипотез* (рис. 4.12). Максимум H^* при этом существенно уменьшается и может быть потерян

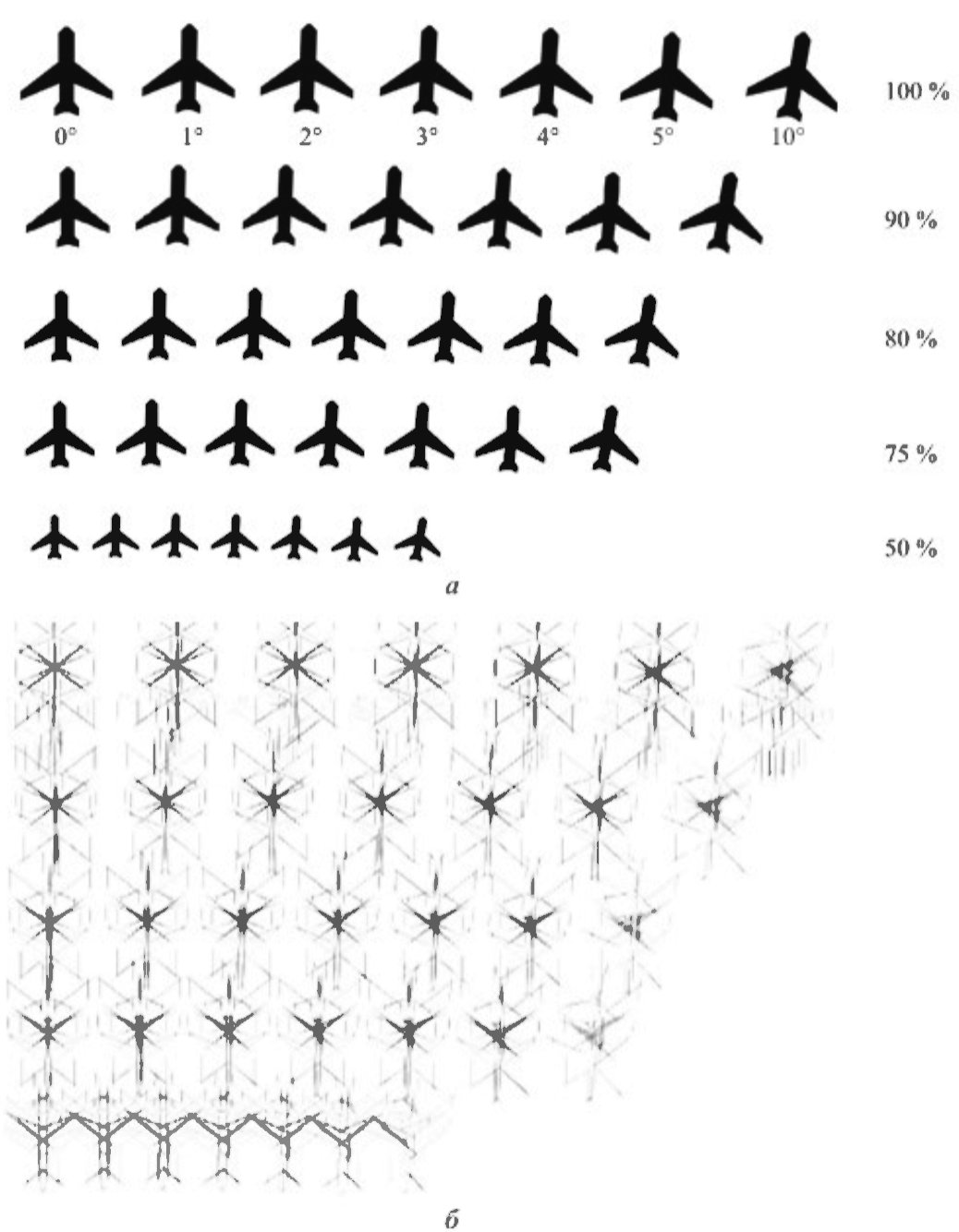


Рис. 4.12. Пространство поиска при наличии геометрических искажений: *а* — изображение объекта с различными масштабом и углом поворота; *б* — пространство поиска

в шуме. Более того, он может размываться кольцеобразно, так, что в центре кластера верных гипотез, соответствующего наиболее вероятному положению объекта на изображении, не окажется сколько-нибудь значимых H^*.

Чтобы устранить этот недостаток, необходимо восстанавливать значение и положение исходного, неискаженного максимума. Этот этап будем называть коррекцией геометрических искажений, поскольку устраняются последствия изменения ракурса съемки.

Для восстановления максимума можно использовать, например, суммирование откликов по некоторому окну, размеры которого оцениваются исходя из предполагаемого размытия максимума, вносимого геометрическими искажениями. В этом случае положение максимума будет восстановлено, а его значение снова поднимется над уровнем шума и сделает возможным надежное обнаружение объекта на изображении.

Алгоритм суммирования по окну требует достаточно небольшого числа операций (четыре сложения на точку), поскольку его можно реализовать за два прохода по изображению.

Однако восстановленный таким образом отклик H^* больше не будет равен числу совпавших точек образца и изображения, и рассчитать вероятность обнаружения объекта в заданном положении достаточно сложно: сильно зашумленные области изображения с обилием точек границ способны производить ложные восстановленные максимумы.

Чтобы исправить это, необходимо предварительно нормировать пространство поиска.

Нормирование пространства поиска. Целью нормирования является снижение стоимости отклика в областях изображения с большим числом точек и повышение ее в областях с меньшим. Однако первые эксперименты показали, что интуитивный подход — взять в качестве нормирующего множителя число точек контура в области или их квадрат — к положительным результатам не приводят.

В качестве меры близости образца и изображения удобно использовать косинус угла между векторами, представляющими точки границ образца и изображения. Эти векторы можно сформировать следующим образом. Пусть размерность обоих векторов равна размеру (числу всех точек) образца; при этом координата вектора равна 1, если данная точка есть точка границы, и 0 — если нет. Очевидно, что длина такого вектора равна корню квадратному из числа точек границ образца.

Аналогично для вектора изображения берем его фрагмент, равный по размеру образцу и начинающийся в точке $(\Delta x, \Delta y)$. Его длина также равна корню квадратному из числа точек границ во фрагменте. Тогда

$$\cos\varphi[\Delta x, \Delta y] = \frac{\sum_{k=1}^{n} I[\Delta x, \Delta y]_k S_k}{|I[\Delta x, \Delta y]||S|}.$$

Числитель этого выражения равен числу совпавших точек и близок по смыслу к $H(\Delta x, \Delta y)$, однако при вычислении $H(\Delta x, \Delta y)$ учитываются только точки с одинаковыми градиентами яркости. Чтобы учесть этот факт, введем расши-

ренные векторы образца $\vec{S}'$ и изображения $\vec{I}'$, состоящие из нескольких векторов $S[g_i]$ и $I[g_i]$ соответственно. В вектор $S[g_i]$ и $I[g_i]$ входят только те точки, которые имеют направление градиента g_i:

$$\vec{I}' = <I[g_1], I[g_2], ..., I[g_n]>;$$

$$\vec{S}' = <S[g_1], S[g_2], ..., S[g_n]>.$$

Очевидно, что общее число точек границ и длина векторов от этого не изменится. Тогда

$$\cos\varphi'[\Delta x, \Delta y] = \frac{\sum_{k=1}^{n} \vec{I}'[\Delta x, \Delta y]_k \vec{S}'_k}{|\vec{I}||\vec{S}|} = \frac{1}{\sqrt{\sum \vec{S}_k}} \frac{H[[\Delta x, \Delta y]}{\sqrt{\sum \vec{I}[\Delta x, \Delta y]_k}}.$$

Коэффициент $\frac{1}{\sqrt{\sum \vec{S}_k}} = \text{const}$ зависит только от образца и не зависит от координаты на изображении, и в большинстве случаев он может быть опущен и учтен позже. Нормирующим множителем в данном случае выступает отношение $\frac{1}{\sqrt{\sum \vec{I}[\Delta x, \Delta y]_k}}$. Подкоренное выражение знаменателя представляет собой сумму точек контура по окну. Как и в случае с восстановлением максимума, она может быть рассчитана за два прохода по изображению. На первом проходе вычисляется среднее в окне по строкам, на втором — среднее в окне по столбцам от предыдущего шага. Суммы в соседних окнах отличаются только на два компонента, и для получения последующей суммы достаточно скорректировать предыдущую, не вычисляя значение заново.

Вычисление единицы, деленной на корень, может быть выполнено табличным способом, поскольку подкоренное значение является целым числом, не превышающим общего числа точек образца. Значение может быть представлено в виде числа с фиксированной точкой для использования операции целочисленного умножения.

Для областей, не содержащих контуров и всегда дающих $H = 0$, нормирующий множитель выставлялся равным нулю во избежание появления неопределенности и деления на нуль.

Таким образом, алгоритм коррекции потребует дополнительно одну длинную операцию (целочисленное умножение) на точку и может выполняться в режиме реального времени.

Графовые методы. Пусть имеются изображение объекта и его образец, представленные набором ключевых точек. Будем считать, что каждая совпадающая ключевая точка объекта и образца является вершиной некоторого графа (в терминах GHT — такая пара образует гипотезу). Соединим дугами вершины, соответствующие совместимым гипотезам. Получим *граф соответствия* (рис. 4.13).

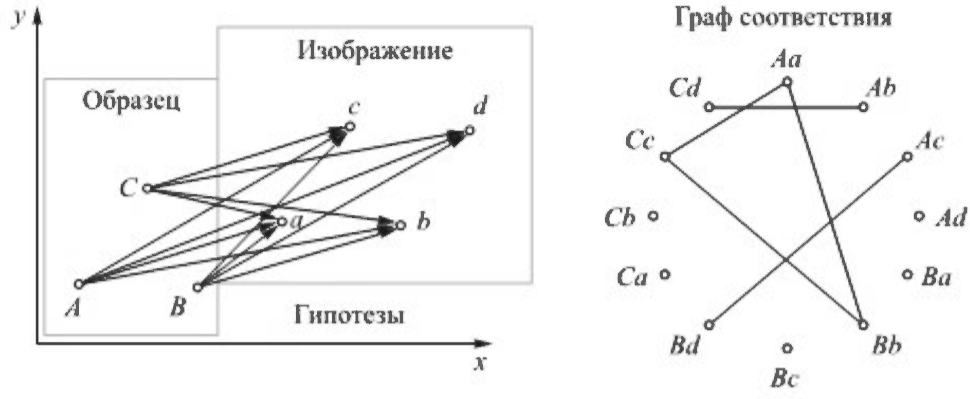


Рис. 4.13. Построение графа соответствия (точки образца обозначены большими буквами, точки изображения — маленькими)

Граф соответствия можно разбить на совокупность полносвязных подграфов. Число вершин в каждом подграфе равно числу совпадающих при данном преобразовании ключевых точек объекта и образца. Очевидно, что такой подграф отражает преобразование, при котором совпадет максимальное число ключевых точек объекта и образца и определяет искомое положение объекта на изображении.

Таким образом, задачу поиска объекта на изображении можно свести к задаче поиска максимальной клики графа сопоставления.

Будем считать, что ракурс и расстояние до объекта заранее известны, и требуется определить лишь положение $(\Delta x, \Delta y)$ объекта. В этом случае алгоритм можно сформулировать в следующем виде.

Для каждой ключевой точки α_i изображения:
  для каждой ключевой точки A_j, которая может совпадать с α_j:
    для каждой точки b_p изображения, $b_p \neq \alpha_i$:
      для каждой ключевой точки B_q, которая может совпадать с b_q:
        если гипотезы $A_j\alpha_j$ и $B_q b_q$ совместимы, т. е.
        $X(A_j) - X(\alpha_j) \approx X(B_j) - X(b_j)$ и $Y(A_j) - Y(\alpha_j) \approx Y(B_j) - Y(b_j)$,
        то добавить гипотезу в список сопоставимых гипотез $L(A_j\alpha_j)$.
        Если текущий список $L(A_j\alpha_j)$ оказался длиннее текущего наилучшего, то запомнить $L(A_j\alpha_j)$ как текущий наилучший.

Аналогично можно найти и несколько объектов, если запомнить N лучших списков или все списки длиннее заданного порога, в зависимости от контекста решаемой задачи. В отличие от предыдущих, данный метод позволяет легко учитывать неточное совпадение ключевых точек. Кроме того, он позволяет находить список совпавших ключевых точек.

Все это делает графовый метод незаменимым для дальнейшего уточнения ракурса объекта. Однако при этом резко возрастает объем вычислений до $O(N^4)$ для случая только неизвестного положения, по сравнению с $O(N^2)$ для обобщенного преобразования Хафа. Поэтому данный метод может применяться для относительно небольшого числа ключевых точек.

Следует обратить внимание, что в алгоритме внешний цикл выполняется по точкам изображения, а внутренний — по точкам образца. Это делается

для того, чтобы заранее отсортировать и проиндексировать точки образца по тем параметрам, которые позволяют идентифицировать ключевые точки объекта и образца. В результате во втором и четвертом циклах перебираются не все точки образца, а лишь те, которые гарантированно могут совпадать с текущей точкой изображения.

Таким образом, графовый подход позволяет радикально повысить точность обнаружения объекта, но при этом будет снижена либо производительность (при большом числе ключевых точек), либо надежность (при их малом числе).

Типовая область применения графовых подходов — прежде всего задачи склейки аэрофотографических космических снимков, в которых точность учета параметров играет ключевую роль.

Метод башни. Метод башни (или метод пирамиды) является не самостоятельной стратегией, а лишь надстройкой над другими стратегиями. Он работает следующим образом (рис. 4.14). Будем предварительно искать возможные положения объекта на огрубленном изображении. Поскольку при этом надежность обнаружения снизится, следует также кратно снизить и порог обнаружения. После того, как найдены вероятные положения объекта на грубом изображении, проводим уточняющий поиск его положения и уточнение меры близости. Области грубого изображения, в которых объект не найден, из поиска по точному изображению исключаются.

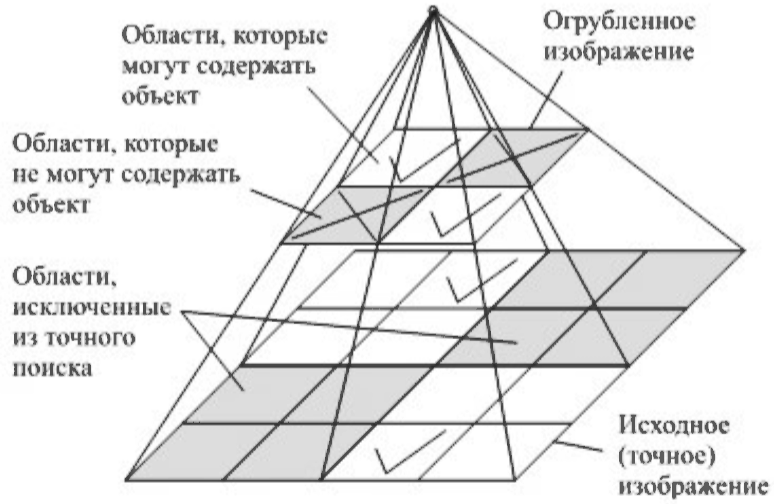


Рис. 4.14. Схема метода башни

Число уровней огрубления может быть более одного.

Рост производительности в методе башни связан с тем, что вычислительная сложность большинства стратегий пропорциональна 2–4-й степени размера стороны изображения. Огрубляя изображение в 2 раза, получаем прирост быстродействия в 4–16 раз при поиске по грубому изображению. Последующий поиск по исходному изображению займет значительно меньше времени, поскольку он проводится уже не по всему изображению, а по некоторой небольшой окрестности грубо найденного положения.

Следует иметь в виду, что рост производительности в этом методе не обходится без издержек: он приводит к снижению надежности обнаружения (объект, пропущенный на грубом уровне, теряется безвозвратно, и на более точном уровне найден не будет). Кроме того, существует критический уровень огрубления, при котором объект становится неразличим и нахождение его вероятных положений становится невозможным.

Необходимо отметить, что при совместном использовании метода башни и обобщенного преобразования Хафа будут «экономиться» не только и не столько вычисления, сколько оперативная память для реализации пространства поиска. Для многих приложений это может оказаться достаточно критическим.

Метод отсечений. Подобно методу башни, метод отсечений — не самостоятельная стратегия, а, скорее, надстройка над другими стратегиями. Если мера близости имеет смысл либо суммарного отклонения точек объекта от образца, либо суммарной ошибки, то она является положительно определенной формой. В этом случае нет необходимости рассчитывать ошибку до конца: достаточно рассчитывать, пока она не достигнет текущего наилучшего положения. При дальнейших расчетах ошибка может лишь расти, и эти вычисления излишни.

Легко заметить, что выигрыш в объеме вычислений сильно зависит от текущего наилучшего значения. Поэтому важным становится порядок просмотра положений: чем раньше будет найден объект, тем меньше объем вычислений. Просмотр следует начинать с наиболее вероятных положений объекта и заканчивать наименее вероятными.

Для расчета вероятности обнаружения можно использовать, например, огрубленные изображения, комплексируя тем самым метод отсечений с методом башни.

К сожалению, метод отсечений не может использоваться за пределами стратегии полного перебора, что сильно сужает область его применения.

Структурные методы поиска. Структурные методы поиска объекта предполагают, что неизменный характеристикой объекта является его структура — состав и взаимное положение структурных элементов. При этом внешний вид, яркость, ракурс и другие параметры объектов могут изменяться в широких диапазонах, что зачастую делает невозможным поиск объекта по его образцу. В структурных методах образцом является структура объекта в виде взаимосвязи положения структурных элементов.

Отправная точка любого структурного алгоритма — поиск отдельных элементов. Структурные элементы ищут при пониженном пороге — пропуск элемента более критичен, чем ложные срабатывания, которые будут отсеяны позже. Для поиска элементов может быть использован любой из рассмотренных методов поиска по образцу, удовлетворяющий требованиям скорости, точности и надежности, например стратегии с использованием преобразования Фурье или Хафа.

Далее среди найденных элементов выбираются их комбинации, которые могут соответствовать структуре искомого объекта. При этом необходимо учитывать, что часть структурных элементов могла быть пропущена на пре-

дыдущем этапе и, наоборот, некоторые элементы обнаружены ошибочно: ни один элемент не дает надежной информации об объекте, важна только их совокупность.

Для поиска групп элементов, соответствующих заданной структуре, можно применять подходы с использованием обобщенного преобразования Хафа либо графовые методы. Использование графовых методов обоснованно, поскольку число найденных на изображении структурных элементов (и их комбинаций), как правило, относительно невелико, и затраты на их перебор сопоставимы с обслуживанием пространства поиска в преобразовании Хафа (при более высокой точности и надежности).

В основе структурного поиска лежит понятие референтной точки (рис. 4.15). *Референтной точкой* элемента будем называть наиболее вероятное положение искомого объекта в предположении, что положение текущего элемента, относительно которого строится эта референтная точка, найдена верно. Референтные точки играют роль гипотез в преобразовании Хафа. Они обладают тем свойством, что верные референтные точки образуют достаточно тесные группы (кластер верных гипотез), тогда как референтные точки ошибочно найденных элементов располагаются хаотически (см. рис. 4.15). Таким образом, структурный поиск можно свести к поиску плотных групп референтных точек.

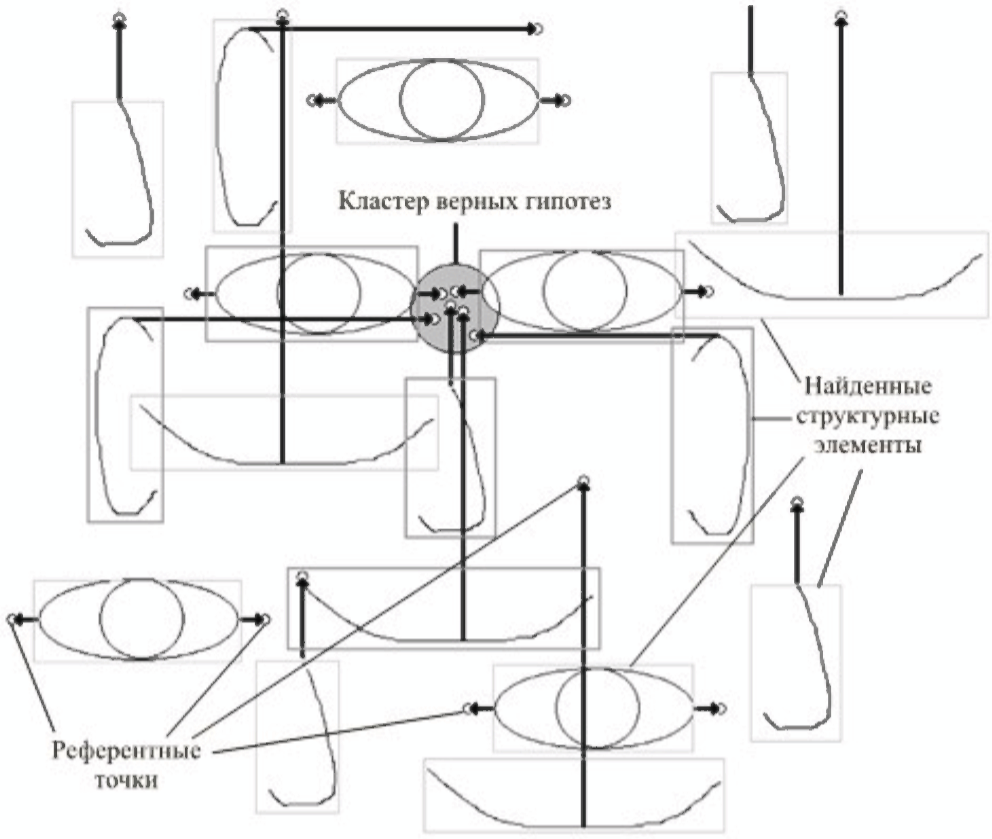
Поиск групп элементов можно проиллюстрировать на примере задачи поиска лица на фотографии (см. рис. 4.15). На первом этапе обнаруживаются возможные структурные элементы лица — глаза, нос, рот, уши. Поскольку ракурс лица может быть различным (а человек крутит головой по всем трем осям), то точного и надежного выделения структурных элементов по их образцу достичь достаточно сложно. Поэтому элементы детектируются «с запасом» — при пониженном пороге совпадения.

Далее происходит поиск групп элементов, образующих искомую структуру. Понятно, что структурные элементы могут находиться в весьма ограниченном числе областей: глаза — слева и справа от центра лица, рот и нос — ниже центра и т. д.

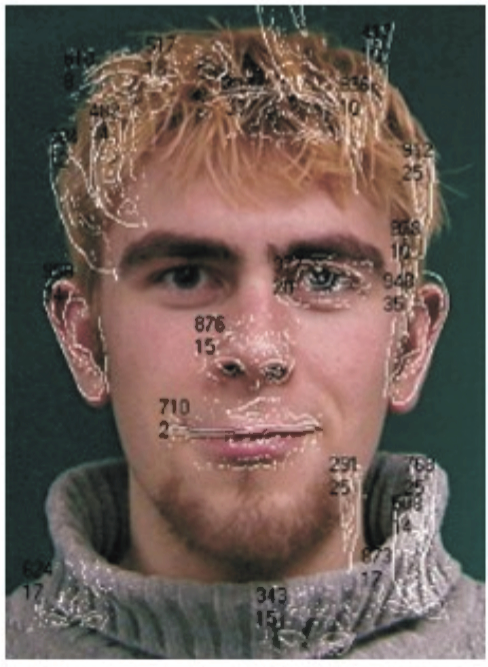
Возможный алгоритм поиска может выглядеть следующим образом.

1. Для каждого структурного элемента сформировать референтную точку — наиболее вероятное положение центра лица в предположении, что текущий элемент найден и идентифицирован верно.
2. Для каждой i-й референтной точки сформировать список совместимых с ней структурных элементов. С этой целью для каждого типа элементов найти референтную точку j-го экземпляра объекта этого класса, наиболее близкую к текущей. Если расстояние между этими референтными точками меньше порога, то j-й экземпляр объекта признать принадлежащим i-й референтной точке и его параметры добавить в список найденных элементов.
3. Списки объектов для референтных точек проанализировать на предмет удаления дублирующих записей (неизбежных в данном алгоритме), подсчитать число элементов, их состав и совокупный отклик.

После обнаружения объекта с заданной структурой возможны дополнительные этапы — восстановление пропущенных элементов, уточнение типа



а



б

в

Рис. 4.15. Схема структурного поиска лиц:
а — пояснение принципа работы; *б* — найденные элементы; *в* — найденные референтные точки

и положения элементов, определение ракурса и его коррекция, распознавание и т. д.

Поиск нескольких объектов. В целом ряде практических задач требуется найти не один, а сразу несколько объектов на одном и том же изображение (либо, наоборот, установить, что заданных объектов на изображении нет вообще). Число объектов, которые следует обнаружить, может быть задано по-разному, например напрямую или указанием вероятности обнаружения.

В простейшем случае задача решается последовательностью операций обнаружения наиболее вероятного положения объекта, изъятия его точек с изображения и повторения поиска до тех пор, пока не будет найдено требуемое число объектов (либо пока вероятность обнаружения очередного объекта не окажется ниже порога). Однако такой подход, хотя и работает, но требует огромного количества лишних повторяющихся вычислений.

Более быстрый и рациональный способ — изымать не точки объекта с изображения, а его отклик из пространства поиска. В этом варианте построение пространства поиска происходит только один раз.

Обратите внимание, что тут необходимо изымать из пространства поиска не только один, максимальный отклик, но и всю окрестность, порожденную найденным объектом (рис. 4.16). В противном случае следующим по значению откликом окажется максимум max-2, отвечающий уже найденному объекту 1.

Тем не менее такой алгоритм по-прежнему выполняет массу лишних повторных операций по поиску очередного максимума. Желательно получить еще более быстродействующий алгоритм. Для этого разобьем пространство поиска на подобласти размером примерно с искомый объект и будем искать максимумы отдельно в каждой подобласти. Это потребует однократного просмотра пространства поиска. Затем найденные максимумы занесем в список, отсортируем и подберем требуемое количество с нужной величиной отклика.

Однако, к сожалению, такой алгоритм будет работать некорректно (рис. 4.17). Если граница подобласти проходит вблизи существенного отклика max-1, то отклик его окрестности max-2 может оказаться больше отклика max-3 второго объекта, который будет потерян.

Выходом из ситуации является использование сетки с более мелким шагом — в половину размера объекта. Меньший шаг сетки гарантирует за-

Рис. 4.16. Пространство поиска при наличии нескольких объектов

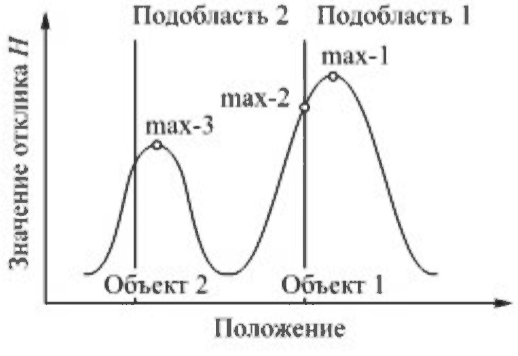


Рис. 4.17. Проблемы поиска максимума в подобласти

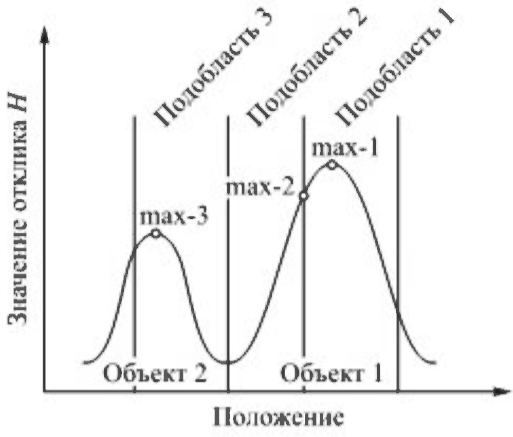


Рис. 4.18. Пространство поиска с малым размером подобласти

несение объекта 2 в список найденных объектов независимо от положения линии сетки относительно максимума в пространстве поиска. Однако, в свою очередь, это приведет к увеличению числа ложных срабатываний: окрестности одного отклика теперь будут восприниматься как отдельные объекты и также попадут в список (рис. 4.18). Поэтому потребуется дополнительный этап фильтрации списка.

Простейший способ фильтрации — от больших откликов к меньшим. Для этого список найденных объектов сортируется по возрастанию откликов (например, методом быстрой сортировки). Затем каждый очередной элемент сравнивается с предыдущими на предмет физической реализуемости: проверяется отсутствие перекрытия с ранее найденными объектами, имеющими больший отклик (рис. 4.19). Элементы, не прошедшие проверку, из списка удаляются.

Рис. 4.19. Фильтрация откликов с учетом физической реализуемости

Поскольку размер списка, как правило, невелик и редко превышает несколько тысяч элементов, то процесс фильтрации занимает незначительное время.

Фильтрация откликов от больших к меньшим — довольно простой, быстрый и надежный метод, который можно широко использовать на практике. Однако возможны и другие подходы, такие как фильтрация с обеспечением максимального плотного расположения объектов (например, в задачах распознавания букв в тексте).

Глава 5
ЗАДАЧА КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ

Теория классификации изображений на данный момент достаточно обширна, и ей посвящено большое количество книг. В настоящей главе кратко рассмотрим основные подходы к задаче классификации изображений.

Постановка задачи классификации объекта

Задача классификации может быть сформулирована следующим образом. Пусть имеется изображение объекта I, а также набор классов A, B, C и т. д., к которым может принадлежать объект. Требуется определить, к какому классу принадлежит объект (рис. 5.1).

В более общем случае может также существовать вариант, когда объект не принадлежит ни к одному из заданных классов. Это порождает второй вариант формулировки задачи распознавания.

Требуется определить, принадлежит объект к предъявленному классу A или не принадлежит (рис. 5.2). Такая задача называется задачей *идентификации*. Сложность ее в том, что на практике весьма сложно представить достаточное число образцов, не принадлежащих к классу A, поскольку их бесконечно много. В силу этого второй вариант задачи не удается свести к первому.

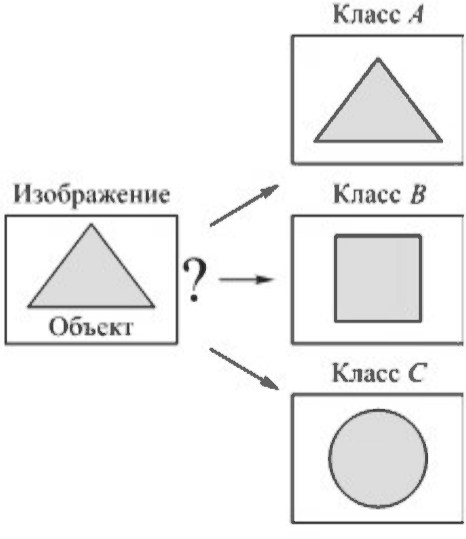


Рис. 5.1. Постановка задачи классификации

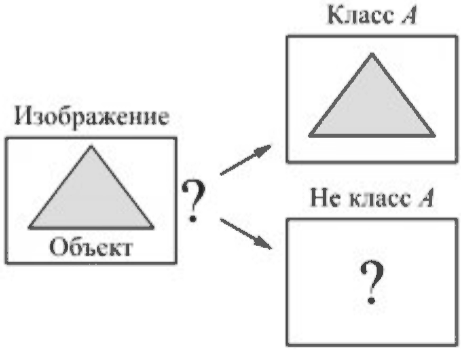


Рис. 5.2. Постановка задачи идентификации

Математическая постановка задачи. Будем считать, что изображение объекта представлено некоторым вектором параметров $X = <x_1, x_2, ..., x_n>$ (например, значениями x_i могут быть яркости отдельных точек изображения). Тогда заданные классы A, B, C можно представить в виде некоторых областей в пространстве параметров X, а сам классифицируемый объект I будет являться точкой в этом пространстве (рис. 5.3). Тогда задача классификации сведется к определению класса, расстояние L_i до которого будет минимальным.

В задаче идентификации требуется определить, попадает объект I в область, занимаемую классом A, или нет (рис. 5.4).

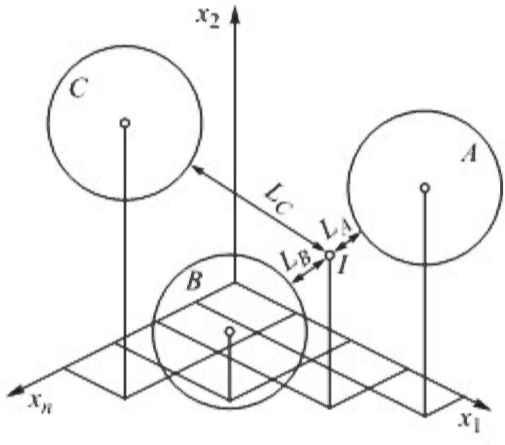


Рис. 5.3. Математическая постановка задачи классификации

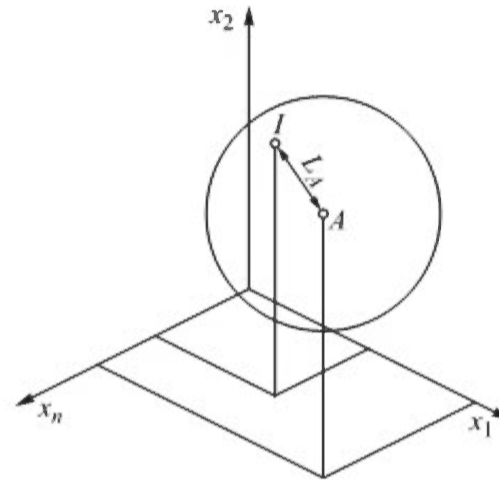


Рис. 5.4. Математическая постановка задачи идентификации

Для решения задач классификации и идентификации объекта, очевидно, необходимо задать три основных компонента:

1) параметры классификации X, по которым будет проходить классификация, или (что то же самое) задать форму представления объекта и классов;

2) области, занимаемые классами в пространстве параметров X;

3) способ расчета расстояния L_i от объекта I до класса A, или (что то же самое) задать меру близости объекта и образца.

Критерии качества классификации

Для того чтобы сравнивать различные алгоритмы и методы классификации между собой, необходимо выбрать некоторые критерии качества классификации.

Рассмотрим следующую задачу простейшей классификации по одному параметру.

Пусть имеется рыболовецкое судно (рис. 5.5), которое ловит рыбу. Пусть в сети попадается рыба двух видов, условно назовем их «килька» и «тюлька». Пусть килька является вполне законным и желанным объектом промыслового лова, а тюлька — нет: она невкусная, занесена в международную красную книгу и ловить ее нельзя.

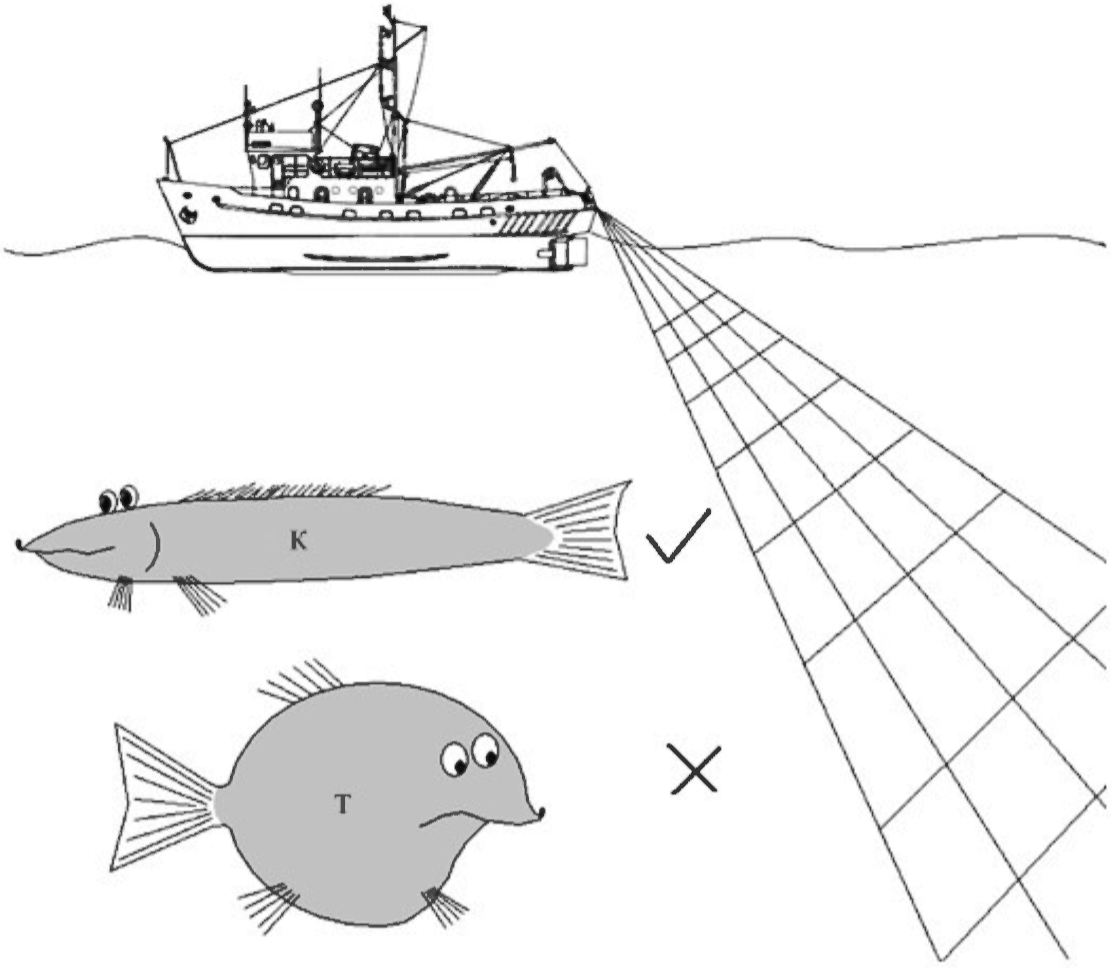


Рис. 5.5. Постановка задачи классификации

Простейший классификатор может быть построен следующим образом. Определим какой-либо параметр классификации x, по которому выбранные классы существенно различаются. Пусть это будет размер объекта (будем считать, что тюльки по размеру меньше, чем кильки). Проведем пробный вылов, выполним замер полученных образцов и по его результатам построим гистограмму распределения размеров (рис. 5.6). Если параметр классификации выбран корректно, то гистограмма будет иметь вид совокупности двух гауссоид, каждая из которых соответствует своему классу. Точка пересечения гауссоид даст пороговое значение L параметра классификации x: теперь все объекты с размером больше L можно классифицировать как кильки, а меньше — как тюльки.

Точка пересечения гауссоид называется *точкой равных ошибок* (Equal Error Ratio, EER). Чем меньше EER, тем точнее работает классификатор.

Очевидно, что такой классификатор не может работать безошибочно: так, по гистограмме видно, что имеется некоторое число тюлек, размеры которых превышают выбранный порог L. Они будут ошибочно классифицированы как кильки. Такая ошибка называется ошибкой типа «Ложное сра-

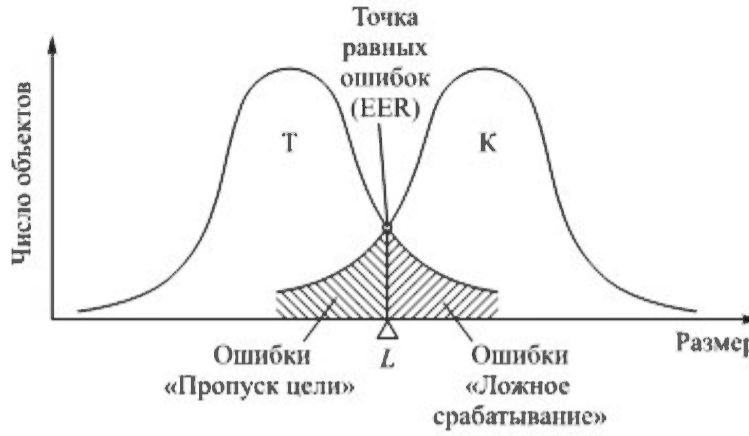


Рис. 5.6. К пояснению принципа работы одномерного классификатора

батывание» (False Positive, FP) или *ошибкой первого рода*. Кроме того, имеется некоторое число килек с размерами меньше порога L, которые будут ошибочно классифицированы как тюльки и будут отпущены обратно в море. Такая ошибка называется ошибкой типа «Пропуск цели» (False Negative, FN), или *ошибкой второго рода*.

Соотношение между этими типами ошибок можно изменить. Например, если тюльку ловить нельзя и за ее вылов предусмотрен крупный штраф, то значение порога L можно увеличить, сократив число ложных срабатываний. Однако при этом существенно возрастет и число пропусков цели.

График, показывающий баланс между ошибками пропуска цели и ложного срабатывания в зависимости от величины порога, называется *кривой принятия решения* (Decision Error Trade-off, DET). По осям откладываются число ошибок первого и второго рода, как правило, в логарифмическом масштабе, а точки графика соответствуют соотношению ошибок в зависимости от значения порога. ERR расположена в точке пересечения DET-кривой с диагональной линией $y = x$. Более качественный классификатор имеет более низкую DET-кривую (рис. 5.7).

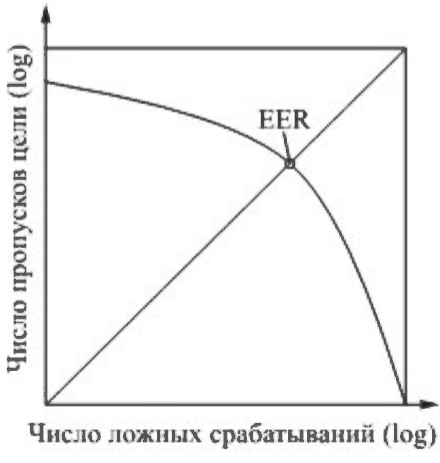


Рис. 5.7. DET-кривая

От абсолютного числа ошибок можно перейти к их вероятностям. Оценками вероятности будут служить соответственно доля ошибок «Пропуск цели» (False Accept Ratio, FAR) и доля ошибок «Ложное срабатывание» (False Reject Ratio, FRR). Зависимость долей «верно» и «неверно» классифицированных элементов искомого объекта (килек) называется *кривой чувствительности классификатора* (Receiver Operating Characteristic, ROC). Площадь под ROC-кривой (Area Under Curve, AUC) является еще одной важной ха-

рактеристикой классификатора: чем она больше, тем лучше классификатор. Значение AUC = 0,5 соответствует случайному угадыванию, а значение AUC = 1 соответствует классификатору, работающему без ошибок (рис. 5.8).

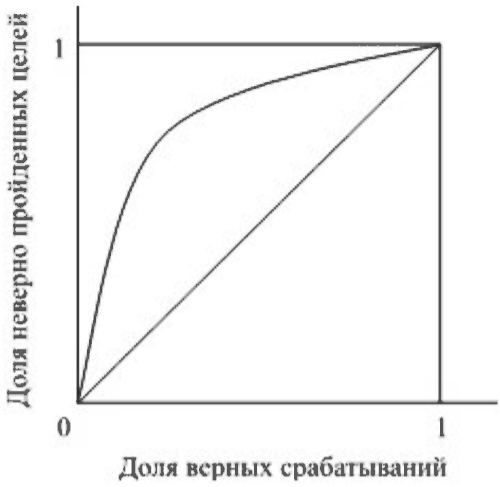


Рис. 5.8. ROC-кривая

Необходимо отметить, что для сравнения классификаторов с использованием критериев EER и AUC следует использовать одни и те же, идентичные тестовые выборки. Для этой цели имеются общеизвестные открытые тестовые выборки (например, база изображений PASCAL, базы лиц FERET и BioID, база рукописных цифр MNIST и многие другие), на которых тестируются вновь разрабатываемые алгоритмы классификации, а результаты их работы сравниваются с уже известными и описанными в литературе.

Другим вариантом сравнения классификаторов является запуск их алгоритмов на пользовательском наборе изображений. Однако для этого необходимо иметь программную реализацию соответствующего алгоритма, что не всегда возможно.

Многопараметрический классификатор. Классификатор, рассмотренный в предыдущей задаче, будет всегда работать с ошибками. Возникает закономерный вопрос: можно ли сделать аналогичный классификатор, но работающий без ошибок? Это возможно. Простейший способ — ввести в рассмотрение дополнительные параметры классификации, которые позволили бы отделить перекрывающиеся хвосты классов.

В предыдущую задачу наряду с первым параметром — «длиной» объекта — введем дополнительный параметр классификации — «ширину». Будем считать, что кильки преимущественно длинные и тонкие, а тюльки — скорее широкие и толстые. Тогда экземпляры класса килек в пространстве параметров будут лежать правее и ниже экземпляров тюлек (рис. 5.9).

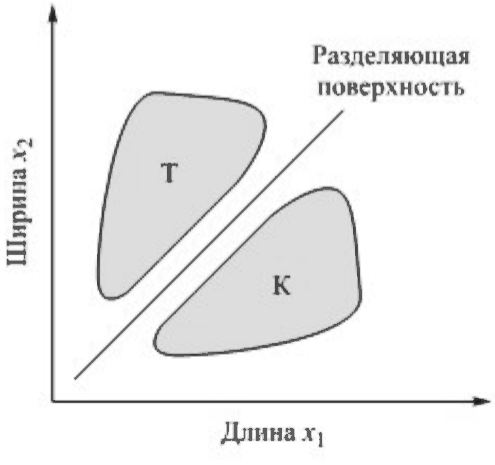


Рис. 5.9. Классификация по двум параметрам

Видно, что в пространстве параметров длина — ширина эти классы стали *разделимыми*, т. е. можно построить некоторую поверхность, отделяющую экземпляры одного класса от другого. Такая поверхность называется *разделяющей*. В случае, показанном на рис. 5.9, такой поверхностью является прямая вида

$$y = w_0 + w_1 x_1 + w_2 x_2,$$

где w_0, w_1, w_2 — параметры прямой (некоторые константы); x_1, x_2 — соответственно длина и ширина объекта.

Если $y(x_1, x_2) > 0$ и точка (x_1, x_2) лежит под прямой, то экземпляр можно классифицировать как кильку, иначе — как тюльку (рис. 5. 10).

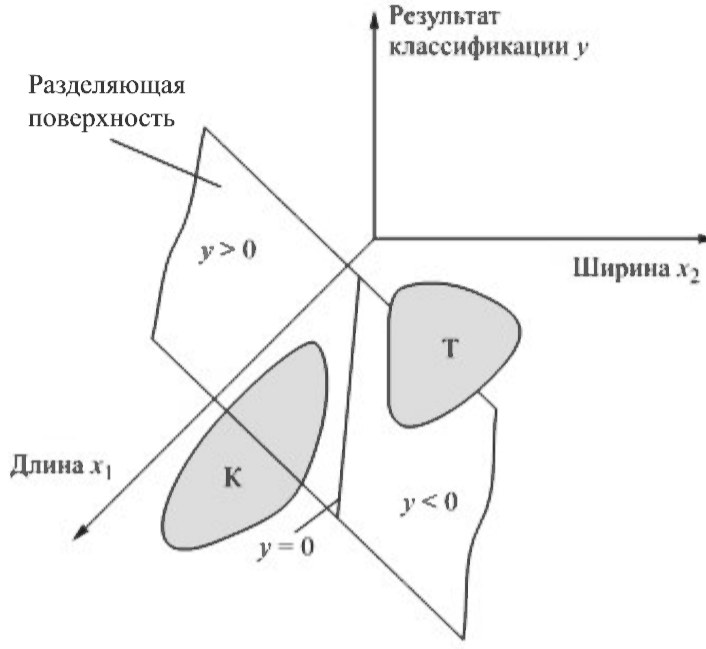


Рис. 5.10. Разделяющая поверхность

На основе анализа рис. 5.9 можно сделать вывод, что только параметров «длина» и «ширина» по отдельности для безошибочной классификации недостаточно: в проекциях как на ось x_1, так и на ось x_2 классы килек и тюлек будут перекрываться. Важна именно их совокупность.

Формы классов в пространстве признаков

Какие положения относительно друг друга могут занимать классы в пространстве выбранных параметров классификации? В первую очередь, они могут быть *разделимыми* и *неразделимыми*. Свойство разделимости предусматривает возможность построения разделяющей поверхности (сколь угодно сложной), которая отделила бы экземпляры одного класса от другого (или от других, если в классификации используется более двух). Если же построение подобной поверхности невозможно, то классы неразделимы и безошибочная работа классификатора будет невозможна. Ранее уже были приведены примеры разделимых и неразделимых классов: классы килек и тюлек неразделимы при классификации по одному параметру и становились разделимыми при добавлении второго параметра. Отсюда видно, что свойство разделимости зависит не только от природы классов, но и от выбора параметров классификации.

Разделимые классы в свою очередь могут быть *линейно разделимыми* и *линейно неразделимыми*. Линейная разделимость предполагает возможность построения линейной разделяющей поверхности вида

$$y = w_0 + w_1x_1 + w_2x_2 + ... + w_ny_n,$$

где w_i — некоторые константы; x_i — параметры классификации.

В случае линейной разделимости классов удается получить простые и надежные алгоритмы классификации.

Если линейную разделяющую поверхность построить невозможно, то классы будут линейно неразделимыми (рис. 5.11). В этом случае алгоритмы классификации становятся гораздо сложнее. И несмотря на то, что построить безошибочный классификатор все еще возможно, на практике не всегда удается найти удовлетворительное решение.

Еще более сложный для классификации случай — неполносвязные классы (рис. 5.12). Неполносвязный класс может состоять из нескольких изолированных частей, причем некоторые из них могут находиться внутри других классов. В этом случае требуются еще более сложная разделяющая поверхность, состоящая из нескольких отдельных частей, и, соответственно, еще более сложный классификатор.

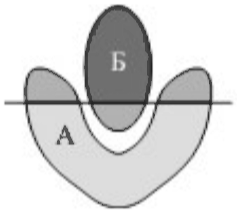


Рис. 5.11. Пример линейно неразделимых классов

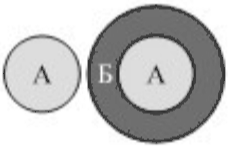


Рис. 5.12. Пример неполносвязных классов

Понятие о скрытых параметрах. Что же является фактором, который определяет форму класса в пространстве параметров классификации? Почему одни классы образуют выпуклые шарообразные множества, удобные для классификации, а другие имеют сложную, неочевидную и сложнопредсказуемую форму?

Одним из ключей к пониманию служат скрытые параметры. Это понятие пришло из квантовой механики. Наличие скрытых параметров подразумевает, что классификация основывается на параметрах, которые неизмеримы и недоступны для прямого наблюдения и должны быть оценены с той или иной точностью через параметры, непосредственно наблюдаемые. При этом сама возможность такой оценки может быть под большим вопросом.

Для примера рассмотрим следующую задачу. Пусть на плоскости заданы координаты некоторых точек. Точки могут принадлежать к двум классам: назовем их условно классом мух M и классом комаров K. Пусть требуется построить классификатор, который, имея на входе координаты точки на плоскости, мог бы отнести ее к одному из указанных классов. Или, в другой постановке, требуется сформировать разделяющую поверхность $z = f(x, y)$, которая отделила бы мух от комаров (рис. 5. 13).

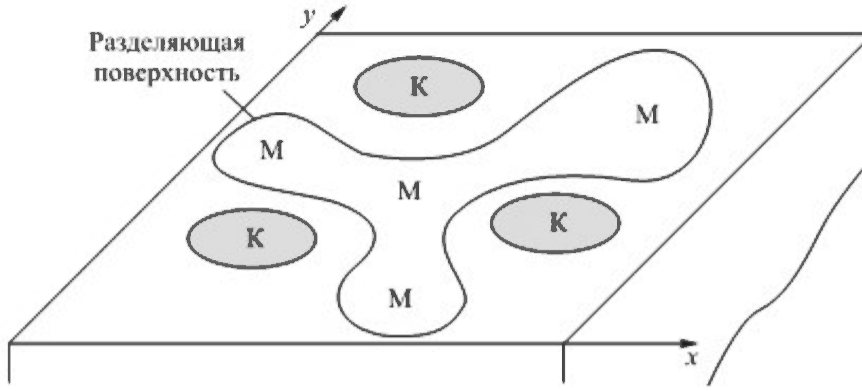


Рис. 5.13. Задача классификации со скрытыми параметрами

В случае разделимых классов эта задача будет иметь решение, но оно очень сложного вида — сложная нелинейная кривая, возможно, состоящая из нескольких изолированных элементов. Кроме того, она не будет робастной: небольшое изменение координат отдельных точек (из-за ошибок измерений или из-за их собственного движения) могут приводить к существенным, резким и непредсказуемым изменениям положения всей разделяющей поверхности.

Если в качестве параметров классификации выбрать не координаты точек, а какие-либо другие, отражающие сущностные характеристики классов (размер, масса, число лапок), подобного не происходит: можно построить крайне простой классификатор с линейной разделяющей поверхностью, устойчивый к шуму в исходных данных.

В этой задаче сущностные параметры скрыты, недоступны непосредственному измерению и их необходимо восстанавливать (неочевидным и неоднозначным образом) из измеримых и наблюдаемых параметров.

Скрытые параметры встречаются в задачах распознавания весьма часто, поскольку вместо сущностных параметров объекта нам доступны только яркости точек изображения. Если этот массив яркостей точек используется напрямую, без этапа предварительного выделения более крупных и обобщенных признаков среднего уровня в предположении, что классификатор найдет такие признаки сам, то существует опасность попадания в описанную ситуацию. При этом разделяющая поверхность окажется сложной в вычислительном плане, чувствительной к шуму и к параметрам отдельных образцов.

Требования к признакам классификации. Какими же свойствами должны обладать признаки классификации? Чтобы уйти от проблемы скрытых параметров, но сохранить производительность, точность и надежность вычислений, признаки должны обладать следующими свойствами.

1. *Представимость* объекта в виде простых элементов выбранного типа. Совокупность элементов должна описывать любые изображения из выбранной предметной области.

2. *Вычислительная реализуемость*. Описание объекта в виде набора признаков может быть легко получено и легко использовано.

3. *Инвариантность* — нечувствительность к изменениям условий наблюдения (ракурса и освещенности)

4. *Соответствие объекта и описания.* Один объект имеет только одно описание, и наоборот, одно описание соответствует только одному объекту.

5. *Сходство* — возможность описывать как сходства, так и различия объектов.

6. *Иерархичность* — возможность деления признаков на главные и второстепенные. Главные признаки несут существенную информацию об объекте, дополнительные — дают описание деталей и отличительных особенностей.

7. *Робастность* — устойчивость к воздействию шума, помех и отсутствию части информации.

Обработка параметров

Цель обработки параметров — упростить форму и взаимное расположение классов в пространстве признаков, сделав классы по возможности выпуклыми, шарообразными и исключив лишние параметры, тем самым существенно упростив дальнейшее построение и настройку классификатора.

Нормирование и центрирование признаков. Нормирование заключается в переходе от абсолютных, размерных входных параметров к параметрам относительным и безразмерным, желательно — изменяющимся в заданном диапазоне.

Параметры, выраженные в абсолютных единицах, сильно зависят от выбора единицы измерения. От того, что было взято за единицу (например, сантиметр или километр), будет зависеть и форма класса в пространстве параметров: при изменении единицы измерения он будет растягиваться и сплющиваться.

Еще хуже обстоит дело с физически разнородными параметрами. Здесь наличие абсолютных единиц приводит к тому, что в одной формуле классификации суммируются килограммы с километрами, что приводит к произвольному растяжению осей координат.

Центрирование предполагает вычитание из значений параметра математического ожидания (или его аналогов) для того, чтобы сделать разброс параметров симметричным относительно нуля. Центрирование не так принципиально, однако оно зачастую входит в формулу для нормирования. Кроме того, ряд классификаторов не имеет собственных средств центрирования и предполагает, что данные уже центрированы.

Каким же образом можно выбрать нормирующий множитель? Самый простой способ — воспользоваться максимальным и минимальным значением параметра — оказывается неэффективным: крайние значения параметра принадлежат к наименее типичным представителям выборки и могут являться просто результатом шума.

Более надежный способ — использовать для нормирования среднеквадратическое отклонение σ:

$$x' = \frac{x - \bar{x}}{\sigma}.$$

Такой подход опирается уже на всю выборку, а не на отдельные образцы (которые могут быть подвержены шуму), однако он тоже не лишен недостатков: СКО отражает разброс параметров только в случае нормального распределения параметров. Однако для задач классификации параметры не будут иметь нормального распределения. Распределение будет мультимодальным, при котором каждая мода соответствует своему классу.

Можно использовать аналоги математического ожидания и СКО в виде медианы M и расстояния между первым и третьим квартилем (*интерквартильный размах*) ΔQ соответственно:

$$x' = \frac{x - M}{\Delta Q}.$$

Расчет медианы и квартилей требует существенно больше времени, чем расчет СКО, в связи с необходимостью сортировки большого числа параметров, однако эта операция обычно выполняется лишь однажды на этапе подготовки и не представляет особой сложности.

Снижение размерности пространства признаков. Требование разделимости классов приводит к необходимости использовать большое число признаков. Однако при этом быстро растет и время настройки классификатора (как правило экспоненциально). Кроме того, в качестве признаков зачастую используются непосредственно яркости всех точек изображения, а их очень много. Как следствие, появляется необходимость в уменьшении числа используемых признаков, чтобы устранить те из них, которые не участвуют в классификации, но существенно замедляют процесс настройки.

Благодаря чему возможно сокращение числа признаков? Введем новые оси координат в рассмотренной ранее задаче о кильках и тюльках: пусть ось y_1 идет вправо вверх так, чтобы проекции образцов на нее имели максимальный разброс, а ось y_2 — перпендикулярно ей (рис. 5.14). Параметр y_1 для классификации оказывается бесполезным: проекции классов на его ось практически совпадают, поэтому его можно полностью исключить. А параметр y_2 позволяет в полной мере разделить классы. Таким образом, вместо двух параметров x_1 и x_2 можно использовать один — y_2, являющийся линейной комбинацией x_1 и x_2.

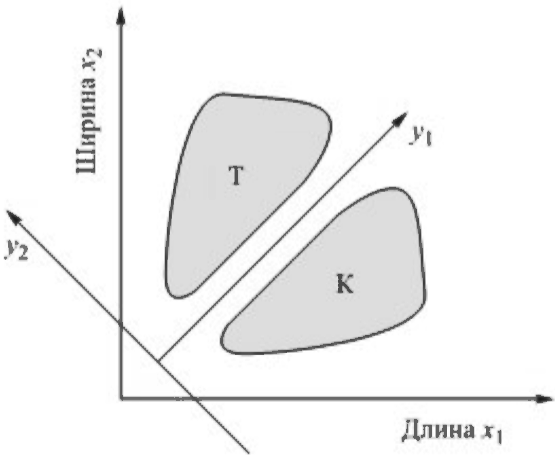


Рис. 5.14. Смена осей координат в задаче классификации

Если число образцов обучающей выборки n меньше числа используемых признаков N, то все эти образцы в пространстве признаков окажутся лежащими в одной гиперплоскости размерностью не выше $n - 1$. Например, две точки образца всегда лежат на одной прямой, три — в одной плоскости и т. д. Таким образом, размерность пространства признаков всегда можно умень-

шить как минимум до $n - 1$, перейдя из полноразмерного пространства в проекцию на гиперплоскость, содержащую все образцы.

Как же найти базис этой гиперплоскости? Одно из возможных решений заключается в использовании *метода главных компонент* (Principial Component Analysis, PCA). В качестве искомого базиса метод предполагает применение собственных векторов матриц ковариаций образцов.

Представим каждое изображение образца I_k в виде центрированного вектора признаков $\vec{x}_k$. Так, если в качестве признаков выбраны яркости всех точек, то $\vec{x}_k$ можно сформировать следующим образом:

$$x_k[i + N_x j] = I_k[i, j] - \bar{I}_k, \quad i = 1, \ldots, N_x, \quad j = 1, \ldots, N_y,$$

где N_x, N_y — размеры изображения; $\bar{I}_k$ — средняя яркость изображения.

Из полученных векторов-строк $\vec{x}_k$ можно собрать матрицу данных X:

$$X = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\}^{\mathrm{T}} = \begin{Bmatrix} x_1[1] & x_1[2] & \ldots & x_1[m] \\ x_2[1] & x_2[2] & \ldots & x_2[m] \\ \vdots & \vdots & \ddots & \vdots \\ x_n[1] & x_n[2] & \ldots & x_n[m] \end{Bmatrix},$$

которая имеет n строк по $m = N_x N_y$ элементов каждая.

Построим матрицу ковариаций изображений выборки

$$C = \frac{1}{n-1} X^{\mathrm{T}} X.$$

Элементы этой матрицы вычисляются следующим образом:

$$c_{ij} = \frac{1}{n-1} \sum_{k=1}^{n} x_{ki} x_{kj}.$$

Собственными числами λ_k и *собственными векторами* $\vec{V}_k$ матрицы ковариаций C называются такие числа и векторы, для которых выполняется равенство

$$C\vec{V}_k = \lambda_k \vec{V}_k, \quad k = 1, \ldots, n.$$

Здесь матрица C выступает в роли некоторого оператора, применение которого к собственному вектору эквивалентно умножению на скаляр — собственное число. Разложение по собственным векторам называют также *спектральным разложением*. Ортогональный базис собственных векторов выступает аналогом, например, гармонического базиса в преобразовании Фурье, а проекции на эти векторы — аналогом гармоник.

Число различных ненулевых собственных векторов равно рангу матрицы C, и оно будет не более $n - 1$. Ненулевые собственные векторы будут образовывать базис искомой гиперплоскости, содержащей все образцы. Собственные векторы имеют ту же размерность, что и вектор данных, и в ряде случаев их можно визуализировать, представив в виде изображения. Так, при

распознавании лиц по яркостям всех точек собственные векторы имеют вид лица человека (*собственные лица*). Подобную визуализацию можно использовать для отброса несущественных векторов вручную.

Полученные собственные векторы существенно неравноценны: часть из них имеют очень небольшой разброс параметров, зачастую — обусловленный воздействием шума, и не дают полезной информации для разделения классов. Такие векторы можно смело исключить из базиса.

Мерой «важности» вектора, отражающей разброс параметров вдоль него, служит его собственное число λ_k (для центрированной матрицы ковариаций собственное число имеет смысл дисперсии проекций параметров образцов на собственный вектор). Поэтому собственные векторы следует сортировать по убыванию собственных чисел, и брать те, доля собственных чисел которых является доминирующей (составляет 95...98 %). В ряде задач также отбрасывается и первый вектор, как в приведенной выше задаче, поскольку он также не несет признаков, полезных для классификации.

Однако при попытке применить данный метод на практике встречается неожиданное препятствие — огромный размер матрицы ковариаций C — порядка четвертой степени от размера стороны изображения. Так, сравнительно скромная картинка размером 256×256 точек потребует $4 \cdot 10^9$ элементов и около 16 Гбайт оперативной памяти, что является серьезным препятствием не только при работе на промышленных контроллерах и бортовых вычислителях, но и на вполне современных универсальных компьютерах.

Однако при расчете собственных векторов можно обойтись и без матрицы ковариаций. В этом поможет *сингулярное разложение* матрицы данных X.

Число σ называется *сингулярным числом* матрицы X, если существуют левый и правый *сингулярные векторы* — такие вектор-строка $\vec{b}$ из m элементов и вектор-столбец $\vec{a}$ из n элементов соответственно, что выполняются два условия:

$$X\vec{a} = \sigma\vec{b}^{\mathrm{T}} \text{ и } \vec{b}X = \sigma\vec{a}^{\mathrm{T}}.$$

Сингулярным разложением матрицы X называют ее представление в виде

$$X = B\,S\,A^{\mathrm{T}} \text{ или } x_{ij} = \sum_{k=1}^{n} \sigma_k b_k[i] a_k[j],$$

где B — левые сингулярные векторы; A — правые сингулярные векторы; S — диагональная матрица, состоящая из чисел σ_k по диагонали.

Правые сингулярные векторы $\vec{a}$ и являются собственными векторами $\vec{V}_k$ матрицы ковариаций, а соответствующие им собственные числа λ_k можно найти из сингулярного числа:

$$\lambda_k = \frac{1}{n-1}\sigma_k^2.$$

Найти сингулярное разложение матрицы данных можно с помощью приближения матрицы X матрицей $P = \{p_{ij}\}$ вида $p_{ij} = b_i\, a_j$ по методу наименьших квадратов:

$$J(\vec{a},\vec{b}) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}(x_{ij} - b_i a_j)^2 \to \min.$$

Алгоритм расчета сингулярных векторов. Будем искать минимум методом последовательных приближений. Сформируем произвольный начальный вектор $\vec{a}$ единичной длины. Зафиксируем значение $\vec{a}$ и рассчитаем для него наилучшее $\vec{b}$, дающее минимум $J(\vec{b})$. Затем зафиксируем найденное $\vec{b}$ и уже для него будем искать наилучшее $\vec{a}$. Будем продолжать этот процесс, пока он не сойдется, например пока полученное приращение ΔJ не окажется меньше заданной точности вычислений ε.

При фиксированном $\vec{a}$ минимум $J(\vec{b})$ будет достигаться при $\partial J/\partial b_i = 0$, откуда находим

$$b_i = \frac{\sum_{j=1}^{m} x_{ij} a_j}{\sum_{j=1}^{m} a_j^2}, \quad i = 1, \ldots, n.$$

Аналогично при фиксированном $\vec{b}$ минимум $J(\vec{a})$ будет достигаться при $\partial J/\partial a_j = 0$, откуда

$$a_j = \frac{\sum_{i=1}^{n} x_{ij} b_i}{\sum_{i=1}^{n} b_i^2}, \quad j = 1, \ldots, m.$$

После нахождения $\vec{a}$ и $\vec{b}$ удаляем их из матрицы X: $x'_{ij} = x_{ij} - b_i a_j$, повторяем процесс для полученной матрицы X' и т. д.

Таким образом, получаем представление матрицы X в виде суммы:

$$X = P_1 + P_2 + \ldots + P_n.$$

Далее рассчитываем сингулярные числа и нормируем векторы $\vec{a}$ и $\vec{b}$:

$$\sigma_k = \|\vec{a}_k\| \cdot \|\vec{b}_k\|, \qquad \lambda_k = \frac{1}{n-1}\sigma_k,$$

$$\vec{a}_{k\text{н}} = \vec{a}_k / \|\vec{a}_k\|, \qquad \vec{b}_{k\text{н}} = \vec{b}_k / \|\vec{b}_k\|,$$

$$\vec{V}_k = \vec{a}_{k\text{н}}, \quad k = 1, \ldots, q.$$

Данный алгоритм достаточно простой, и его можно использовать для широкого круга задач. Существуют модификации этого алгоритма, позволяющие повысить точность и устойчивость вычислений.

Переход в новый базис. После нахождения векторов базиса $\vec{V}_k$ исходные образцы обучающей выборки для настройки классификатора, а также новые изображения, подлежащие распознаванию, необходимо спроецировать в по-

строенный базис. Для этого каждое изображение I представляем в виде вектора признаков $\vec{x}$ и проецируем его на k-ю ось $\vec{V}_k$:

$$y_k = \sum_{i=1}^{m} x[i]V_k[i], \quad k = 1, \ldots, q.$$

Полученный вектор $\vec{y}$ является искомой линейной проекцией изображения в базис собственных векторов матрицы ковариаций. Он имеет на несколько порядков меньший размер, чем исходный вектор $\vec{x}$, но при этом сохраняет всю важную информацию, необходимую для распознавания.

Следует отметить, что при построении базиса информация о принадлежности объектов к тому или иному классу никак не используется: базис строится по всей выборке целиком, независимо от принадлежности объектов.

Кластеризация исходных данных. Цель кластеризации — разбить класс сложной формы на несколько более простых. В результате сложный классификатор можно заменить совокупностью существенно более легких в настройке и более быстрых. Так, неполносвязный класс можно заменить совокупностью полносвязных, а невыпуклый — совокупностью выпуклых (рис. 5.15).

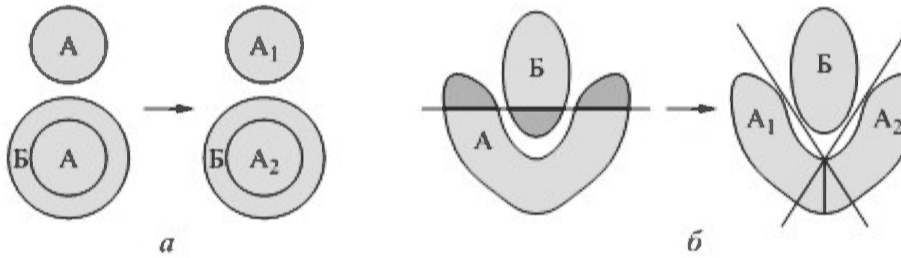


Рис. 5.15. Упрощение формы класса:
а — неполносвязный класс разделяется на два полносвязных; *б* — невыпуклый класс разбивается на два выпуклых линейно разделимых класса

Для кластеризации можно использовать достаточно простые алгоритмы, например метод k-средних. Ключевым элементом является соблюдение баланса между числом получаемых подклассов и степенью простоты классификатора: с некоторого момента дальнейшее разбиение уже не будет приводить к упрощению классификатора, а наоборот, будет увеличивать число необходимых разделяющих поверхностей. Более того, при наличии подклассов с небольшим числом элементов снижается способность классификатора к обобщению, и он становится чрезмерно чувствительным к шуму.

Одним из возможных подходов является оценка степени выпуклости получаемых подклассов. Если полученные подклассы являются более выпуклыми, чем исходный, то разбиение проведено удачно. Выпуклость можно оценить, например, сравнивая средний разброс параметров класса с разбросом вдоль главной оси.

Другой подход — оценка линейной разделимости. Если полученный подкласс линейно разделим со всеми негативными примерами, то дальнейшая кластеризация уже не нужна: получившийся классификатор — линейный,

наиболее простой. Построение линейного классификатора для проверки не требует существенных вычислительных ресурсов, и его можно использовать как критерий качества кластеризации.

Необходимо помнить, что кластеризация — процесс достаточно ресурсоемкий, поэтому ее применение ограничено задачами с относительно небольшим числом параметров классификации.

Насыщение выборки. Часто обучающая выборка содержит недостаточное число образцов (в отдельных особо важных областях пространства признаков или в целом) для построения качественного классификатора, а получение новых образцов сопряжено с определенными сложностями. Насыщение выборки, или *аугментация*, ставит задачей увеличить ее размер в случае недостаточности. Аугментация выполняется за счет добавления искусственно сформированных образцов. Новые образцы формируются из имеющихся за счет привнесения в них искажений, с которыми столкнется классификатор в реальной среде. Образцы могут подвергаться, например, воздействию шума и помех, искажениям ракурса, изменениям условий освещенности и др.

Недостаточное число образцов может привести к тому, что классификатор попытается «запомнить» особенности предъявленных ему образцов (например, цвет некоторых точек) вместо того, чтобы строить обобщенную модель. В этом случае при предъявлении нового изображения классификатор не сможет его корректно распознать. Такое поведение классификатора называют *переобучением*. Чтобы его избежать, требуется сформировать выборку достаточно большого размера (порядка числа настраиваемых параметров классификатора).

В образцы привносят именно те искажения, с которыми классификатор столкнется в дальнейшем. Например, если положение предъявляемого объекта не центрировано, в выборку добавляют сдвинутые вправо-влево и вверх-вниз копии имеющихся образцов. Если классификатор будет работать в меняющихся условиях освещенности, которые алгоритмически не корректируются, то добавляют образцы с изменением яркости и контрастности, и т. д. Необходимо помнить, что рост размера выборки увеличит и время настройки классификатора, поэтому добавлять образцы нужно только с теми искажениями, которые критичны для классификатора. То же касается и выбора образцов: добавляться должны наиболее важные из них, например лежащие вблизи предполагаемого местонахождения разделяющей поверхности. В этом случае может оказаться целесообразным приготовить заранее искаженные версии всех образцов, но в текущую выборку включать только те из них, которые кажутся важными в текущий момент.

Классификаторы

Рассмотрим основные типы классификаторов, являющиеся теоретической базой для построения более сложных и более точных алгоритмов.

Простейший линейный классификатор. Рассмотрим самый простой случай. Пусть имеется два класса с шарообразной формой в пространстве признаков, примерно одинаковые по размеру. Пусть точка A — центр масс первого класса (класса A), а B — центр масс второго класса. Геометрическое место точек, рав-

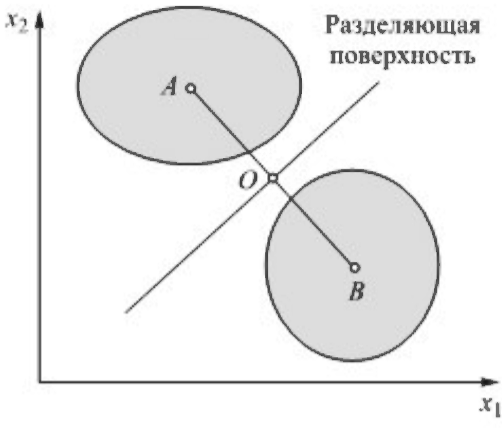


Рис. 5.16. К пояснению принципа работы простейшего классификатора

ноудаленных от A и B, будет находиться на гиперплоскости, перпендикулярной отрезку AB и проходящей через его середину, точку O. Будем считать эту гиперплоскость искомой разделяющей поверхностью (рис. 5.16).

Найдем уравнение разделяющей поверхности. Для любой точки разделяющей поверхности X должно выполняться условие перпендикулярности: $OX \perp AB$. Поэтому скалярное произведение векторов OX и AB должно быть равно нулю:

$$\sum (x_i - x_{iO})(x_{iA} - x_{iB}) = 0.$$

Раскроем первую скобку:

$$\sum x_i (x_{iA} - x_{iB}) - \sum x_{iO} (x_{iA} - x_{iB}) = 0.$$

Обозначим

$$w_i = x_{iA} - x_{iB};$$
$$w_0 = \sum x_{iO} (x_{iA} - x_{iB}),$$

где w_i, w_0 — константы.

Тогда уравнение разделяющей поверхности примет вид

$$w_0 + x_1 w_1 + x_2 w_2 + \ldots + x_N w_N = 0.$$

Классификатор будет работать следующим образом.

По обучающей выборке находим значения коэффициентов настройки w_i. Для предъявленного классификатору изображения вычисляем отклик y:

$$y = w_0 + x_1 w_1 + x_2 w_2 + \ldots + x_N w_N.$$

Если $y > 0$, то изображение принадлежит к классу A; если $y < 0$, то к классу B; если же $y = 0$, то изображение лежит непосредственно на разделяющей поверхности и равноудалено от обоих классов.

Преимущество такого классификатора — предельная простота его настройки: параметры классификатора w_i рассчитываются напрямую исходя из заданной обучающей выборки. Такие классификаторы называются *формируемыми*.

Вместе с тем такой классификатор имеет недостатки: он корректно работает только для эллиптических классов примерно одинаковых размеров. Если эти требования не выполняются, то классификатор будет работать с ошибками (рис. 5.17).

Данный классификатор может с успехом применяться в качестве первого приближения как отправная точка для более тонкой настройки классификатора.

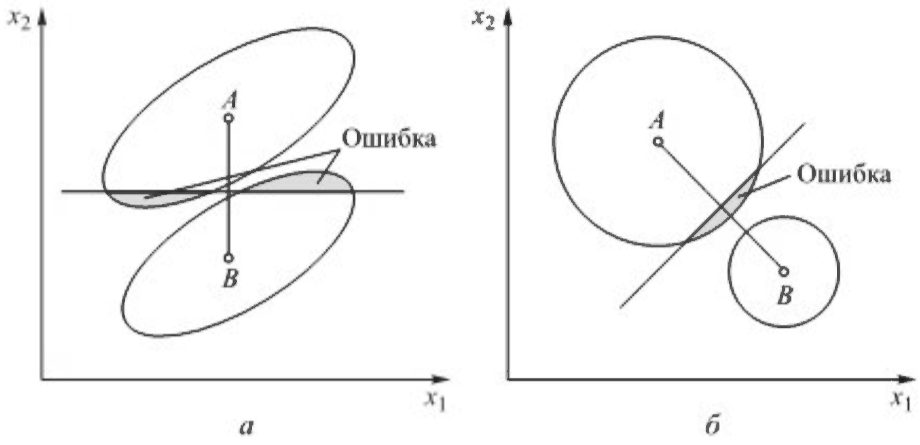


Рис. 5.17. Случаи, для которых простейший классификатор работает некорректно:
а — несферические классы; *б* — классы разных размеров

Персептрон. Установим на выходе линейного классификатора

$$u = w_0 + x_1 w_1 + x_2 w_2 + \ldots + x_N w_N$$

пороговый элемент, который будет выдавать единицу при $u \geq 0$ и нуль при $u < 0$. Чтобы одинаково учесть коэффициент w_0, добавим в вектор образца $\boldsymbol{x}$ нулевой элемент, всегда равный единице. Настроим параметры w_i так, чтобы классификатор выдавал единицу при предъявлении изображения X класса A и нуль — для всех остальных. Такой классификатор называется *персептроном Розенблатта*. Коэффициенты w_i персептрона называются *синапсическими весами* (рис. 5.18).

Рассчитать параметры w_i аналитически не представляется возможным. Однако они могут быть получены в итеративном процессе минимизации числа ошибок классификации. Такой процесс называется *обучением* классификатора. Один из наиболее известных и популярных методов обучения — ме-

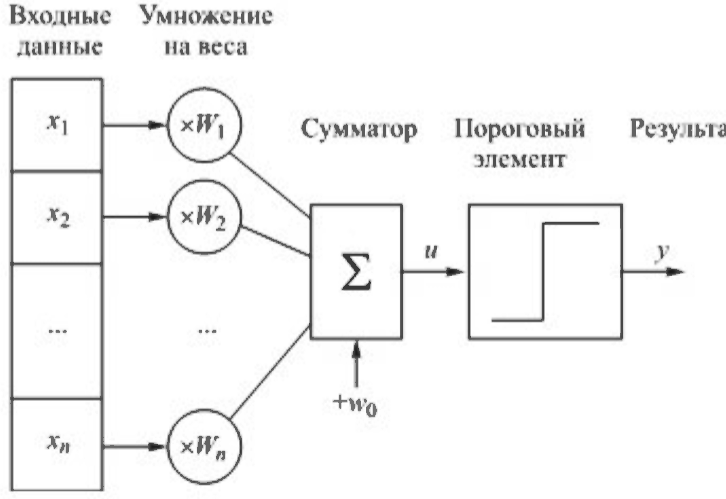


Рис. 5.18. Схема персептрона

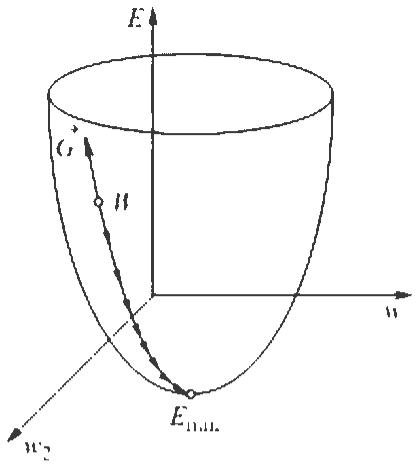


Рис. 5.19. К пояснению принципа наискорейшего спуска

тод градиентного спуска. Его смысл заключается в одновременной коррекции значений всех параметров w_i на значение, пропорциональное их вкладу в ошибку. Вклад параметра w_i в ошибку E — это не что иное, как частная производная $\Delta w_i = \partial E / \partial w_i$. Совокупность частных производных $<\Delta w_i>$ образует *вектор градиента ошибки* ΔW. Этот вектор направлен в сторону, противоположную наискорейшему убыванию ошибки. Поэтому, двигаясь в направлении, противоположном градиенту (по антиградиенту), можно достичь минимума ошибки наиболее быстро (рис. 5.19).

Расчет градиента ΔW требует дифференцируемости функции ошибки по параметрам классификатора. Однако пороговый элемент на выходе не позволяет найти производные. Поэтому его заменяют на гладкую дифференцируемую аппроксимацию, которая называется *функцией активации*. И сама функция активации, и ее производные должны иметь как можно более простой вид для ускорения вычислений. В качестве таких функций обычно выбирают одну из следующих:

• логистическая функция

$$y = \frac{1}{1+e^{-u}}.$$

Ее производная $y'(u) = y(u)(1 - y(u))$. Значения функции ограничены диапазоном (0, +1);

• гиперболический тангенс

$$y = \text{th}\,(u) = \frac{e^{x} - e^{-u}}{e^{x} + e^{-u}}.$$

Производная имеет вид $y'(u) = 1 - y(u)^2$. Эта функция активации — симметричная, она даст единицу для своих объектов и минус единицу — для чужих;

• линейные функции, например линейная с насыщением слева

$$y = \begin{cases} u, & u \geq 0; \\ 0{,}1u, & u < 0. \end{cases}$$

Ее производная $y'(u) = \begin{cases} 1, & u \geq 0; \\ 0{,}1, & u < 0. \end{cases}$

Считается, что гиперболический тангенс $y = 1{,}7159\,\text{th}\,\frac{2u}{3}$ обеспечивает наибольшую скорость сходимости, а линейная с насыщением слева — максимальную скорость вычислений. Логистическая функция имеет важное

теоретическое значение, но в современной практике встречается гораздо реже.

Расчет градиента ошибки. В качестве функции ошибки E выберем сумму квадратов отклонений ответа классификатора $y_k = y(x_k)$ на предъявленный образец x_k от требуемого ответа $\hat{y}_k$:

$$E = \frac{1}{2}\sum_{k=1}^{n}\left(y_k - \hat{y}_k\right)^2.$$

Найдем производную $\Delta w_i = \partial E / \partial w_i$ по правилу дифференцирования сложных функций:

$$\frac{\partial E}{\partial w_i} = \sum_{k=1}^{n}\left(\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial u_k}\frac{\partial u_k}{\partial w_i}\right).$$

Первый сомножитель под знаком суммы:

$$\frac{\partial E}{\partial y_k} = y_k - \hat{y}_k.$$

Второй компонент — производная активационной функции $\partial y_k / \partial u_k$ — зависит от ее выбора. Так, для логистической функции он равен $y_k(1 - y_k)$.

Наконец, третий компонент $\partial u_k / \partial w_i$ в силу линейности классификатора равен $x_k[i]$.

Собрав эти три сомножителя вместе, получим расчетную формулу для градиента ошибки. Например, для классификатора с логистической функцией получим известную формулу

$$\frac{\partial E}{\partial w_i} = \sum_{k=1}^{n}\left((y_k - \hat{y}_k)y_k(1 - y_k)x_k[i]\right).$$

Обучение классификатора. Обучение классификатора сводится к последовательности операций вычисления градиента ошибки и коррекции весов в направлении антиградиента. Найдя градиент ошибки, можно скорректировать веса w_i по формуле

$$w_i' = w_i - h\Delta w_i,$$

где h — некоторая постоянная, называемая *шагом обучения*.

Корректный выбор шага обучения h является важным фактором: большие значения шага не позволят точно попасть в минимум ошибки, а маленькие — существенно замедляют обучение. Поэтому шаг h желательно изменять динамически, подстраиваясь под текущую ситуацию. Вопрос выбора шага рассмотрим чуть позже.

Алгоритм обучения персептрона может выглядеть следующим образом.

1. Выбрать начальные значения весов W_0 (например случайные либо рассчитанные по формуле простейшего классификатора), начальное значение шага $h = h_0$ и вычислить начальную ошибку $E_0 = E(W_0)$.
2. Вычислить градиент ошибки ΔW.

3. Инициализировать счетчик неудач: $C = 0$.
4. Вычислить новый набор параметров: $W_1 = W_0 - h\Delta W$.
5. Вычислить ошибку $E_1 = E(W_1)$.
6. Если $E_1 < E_0$, т. е. новый набор параметров лучше,
   то запомнить новый набор параметров: $W_0 = W_1$, $E_0 = E_1$;
   увеличить шаг: $h = 2h$;
   вернуться к шагу 2.
7. Если $E_1 > E_0$, то:
   увеличить счетчик неудач: $C = C + 1$. Если он стал больше допустимого — то выход.
   уменьшить шаг: $h = h/4$, повторить вычисления, начиная с шага 4.

Классификатор может настраиваться либо по каждому образцу поочередно, либо по всем одновременно, либо по некоторой отобранной группе образцов (пакету). Обучение поочередно каждому образцу приводит к тому, что, когда классификатор дойдет до последнего образца, первые уже забыты. Обучение же на всех образцах сразу существенно замедляет процесс, вовлекая в него массу несущественных образцов. Поэтому наибольшее распространение получили именно пакетные методы.

Пакет может быть сформирован, например, из самых «неудобных» для классификатора образцов, дающих наибольшие ошибки, либо стохастически, либо комбинацией этих методов.

Выбор шага обучения. Простейший способ коррекции шага обучения, который был использован в приведенном выше алгоритме, заключается в том, чтобы увеличивать шаг при успешных итерациях, приводящих к снижению ошибки, и уменьшать — при неудачных. В приведенном выше алгоритме при успехе шаг увеличивался в 2 раза, при неудаче — уменьшался в 4 раза. В случае череды удач (или неудач), происходящих из-за слишком маленького (или большого) шага, он будет расти (или убывать) в геометрической прогрессии и быстро достигнет оптимального значения.

Более быстрый алгоритм, уменьшающий число проб и ошибок, заключается в аппроксимации функции ошибки в сечении градиентом некоторой параболой $e = ax^2 + bx + c$. Вершину параболы, находящуюся вблизи минимума ошибки и соответствующую оптимальному шагу, теперь можно найти аналитически: $h^* = -b/(2a)$.

Алгоритм выбора шага будет выглядеть следующим образом.

1. Пусть заданы начальные условия: вес W_0, ошибка $E_0 = E(W_0)$, градиент ΔW и предыдущий шаг $h = h_0$. Задать шаг первой известной точки: $h_0 = 0$.
2. Построить вторую точку для аппроксимации: $W_1 = W_0 - h\Delta W$, вычислить ошибку в ней: $E_1 = E(W_1)$ и запомнить ее шаг: $h_1 = h$.
3. Если полученная ошибка меньше ($E_1 < E_0$), то третью точку следует искать правее:
   3.1. Увеличить шаг: $h = 2h$ и запомнить его: $h_2 = h$.
   3.2. Найти третью точку: $W_2 = W_0 - h\Delta W$ и ошибку в ней: $E_2 = E(W_2)$.
   3.3. Если $E_0 + E_2 > 2E_1$, то функция ошибки быстро убывает, и следует продолжать идти в данном направлении. Перенести точки влево: $h_0 = h_1$; $h_1 = h_2$; $W_0 = W_1$; $W_1 = W_2$; $E_0 = E_1$; $E_1 = E_2$ и повторить шаг 3.1.

3.4. Если $E_0 + E_2 < 2E_1$, то парабола, построенная по этим точкам, будет обращена вершиной вниз, и можно найти ее минимум.

Если полученная ошибка больше начальной, то третью точку следует искать между двумя первыми: уменьшить шаг h_2, пока не выполнится условие $E_0 + E_1 > 2E_2$.

4. По тройке найденных точек построить параболу, найти положение ее вершины h_3:

$$h_3 = -\frac{1}{2}\frac{(E_1 - E_0)(h_2^2 - h_0^2) - (E_2 - E_0)(h_1^2 - h_0^2)}{(h_1 - h_0)(E_2 - E_0) - (h_2 - h_0)(E_1 - E_0)}.$$

5. Найти веса и ошибку, соответствующие шагу h_3: $W_3 = W_0 - h_3\Delta W$; $E_3 = E(W_3)$.

6. Среди ошибок $E_0, \ldots, E_3$ выбрать наименьшую и запомнить соответствующие веса, ошибку и шаг.

Построение параболы можно повторять многократно, используя наилучшую точку h и известные ближайшие к ней точки слева и справа. Данный алгоритм выбора шага является достаточно общим и может быть использован и в других, более сложных классификаторах.

Кусочно-линейный классификатор. Кусочно-линейный классификатор представляет собой совокупность простых линейных классификаторов, соединенных между собой операциями «И» и «ИЛИ» (рис. 5.20). Разделяющая поверхность такого классификатора будет кусочно-линейной, состоящей из соединенных фрагментов гиперплоскостей L_i.

Принцип работы классификатора, показанного на рис. 5.20, может быть следующим. Если предъявленный объект X не удовлетворяет первому правилу L_1, то он однозначно трактуется как не принадлежащий к классу А и классификация завершается. Если он удовлетворяет правилу L_1, то аналогично выполняется проверка правила L_2 (и других правил, если они имеются). Лишь те объекты, которые удовлетворили всем правилам, классифицируются как принадлежащие к классу А.

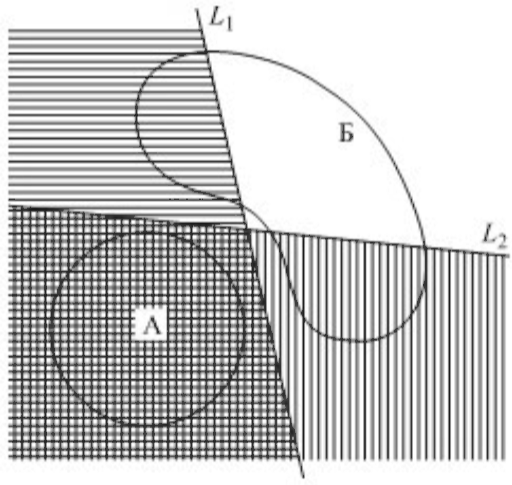


Рис. 5.20. К пояснению принципа каскадной фильтрации

Если отсортировать правила по убыванию числа отбрасываемых чужих образцов, то такой классификатор будет работать достаточно быстро: бóльшая часть негативных образцов будет отбрасываться на ранних этапах.

Принцип построения кусочно-линейного классификатора может быть следующим. Сначала строится первое приближение — формируется первый линейный классификатор (любым из способов), и настраивается его порог w_0 так, чтобы он делал минимально допустимое (например нулевое) число пропусков цели интересующего нас класса А (т. е. правильно классифицировал почти все элементы класса А), но при этом также правильно классифицировал и часть негативных образов класса Б (рис. 5.21, *а*).

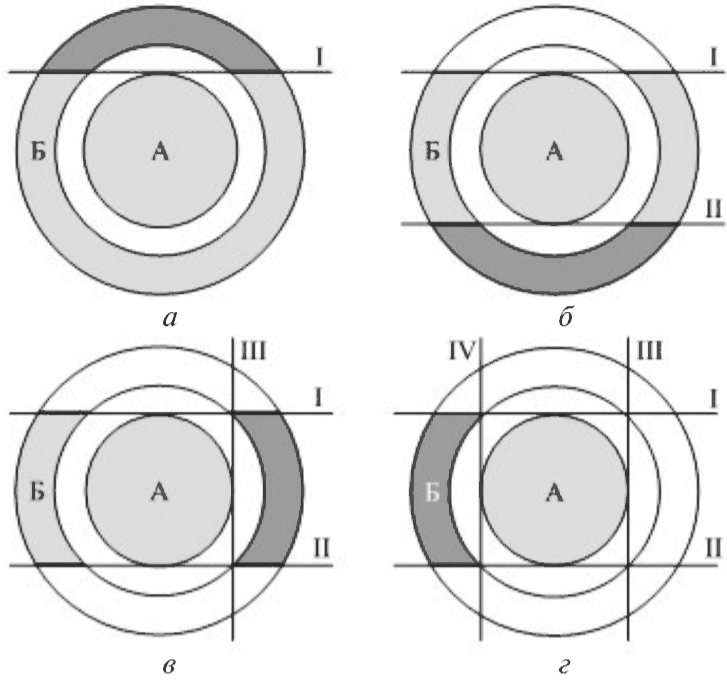


Рис. 5.21. Пример построения каскадного фильтра

Уберем из рассмотрения все образцы класса Б, которые распознаны верно, а по оставшимся построим аналогичным образом второй линейный классификатор (рис. 5.21, *б*). Продолжим добавлять линейные классификаторы до тех пор, пока не достигнем заданной точности, т. е. пока в классе Б не останется ни одного образца, на которых классификатор ошибся (рис. 5.21, *в*, *г*).

В случае если класс негативных примеров Б не имеет достаточного числа представителей для настройки очередного линейного классификатора, он может быть искусственно пополнен. Для этого на предыдущие каскады подают фрагменты произвольных изображений, заведомо не содержащих экземпляров класса А. Фрагменты, не отброшенные ни одним из уже настроенных классификаторов, используются как негативные примеры для настройки последующих.

Полученный таким образом классификатор называется *каскадным*, а процесс его построения — *усилением* слабых классификаторов или *бустингом*. Принцип усиления может быть расширен и на другие типы классификаторов. Каскадный фильтр обладает логической прозрачностью и высокой производительностью и при этом демонстрирует хорошую устойчивость к шуму и помехам.

Многослойный персептрон. Многослойный персептрон строится из нескольких слоев обычных персептронов. Выходы элементов предыдущего слоя соединены со всеми входами последующего, как показано на рис. 5.22. Таким образом, каждый элемент последующего слоя оказывается связанным с каждым элементом предыдущего.

Обычно многослойный персептрон имеет два или три слоя. Необходимо иметь в виду, что в разных источниках число слоев считается по-разному. Здесь под слоем будем понимать слой, имеющий настраиваемые коэффици-

енты. Двухслойный персептрон позволяет строить нелинейные разделяющие поверхности, трехслойный — неполносвязные (т. е. любые). С увеличением числа слоев резко возрастает сложность настройки классификатора и снижается качество его работы, поэтому желательно использовать минимально необходимое число слоев.

Число выходных элементов последнего слоя равно числу классов. Число элементов промежуточных слоев могут быть различными, и однозначной методики для их выбора сейчас не существует.

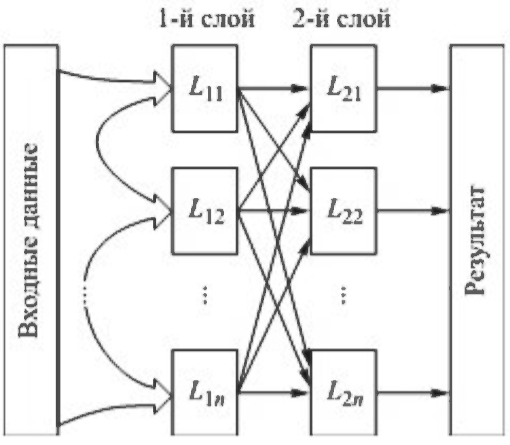


Рис. 5.22. Схема многослойного персептрона

Рассмотрим принцип работы такого классификатора. Все слои работают поочередно, начиная от входного и заканчивая выходным. На входной слой подается распознаваемое изображение. Каждый элемент входного слоя вычисляет свой отклик на предъявленное изображение и передает его на все элементы последующего слоя. После того как все элементы текущего слоя произвели свои вычисления, к работе приступают элементы следующего слоя и т. д., до тех пор, пока не отработают элементы последнего слоя. Объект считается принадлежащим к тому классу, чей выходной элемент даст наибольший ответ.

Разделяющая поверхность многослойного персептрона выглядит следующим образом. Элементы 1-го слоя формируют линейные разделяющие поверхности, отделяющие части классов А и Б (рис. 5.23). Элемент 2-го слоя соединяет эти части, используя аналоги логических функций «И» и «ИЛИ», и сглаживает полученную кривую.

Таким образом, многослойный персептрон можно считать аналогом каскадного фильтра. Если уже имеется настроенный каскадный фильтр, его можно использовать для настройки весов 1-го слоя.

Здесь можно увидеть роль нелинейной функции активации на выходе элемента: если бы ее не было, то многослойный классификатор можно было

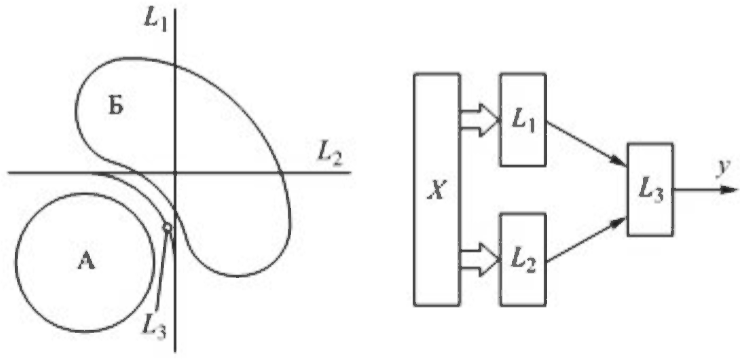


Рис. 5.23. Разделяющая поверхность L_3 многослойного персептрона

бы заменить простым линейным классификатором с соответствующей деградацией свойств.

Элементы многослойного персептрона работают параллельно и независимо друг от друга, поэтому он обладает высоким потенциалом к распараллеливанию вычислений. Кроме того, базовой операцией элементов является операция умножения с накоплением, которая может выполняться параллельно с привлечением аппаратной поддержки, например векторных вычислений.

Обучение многослойного персептрона. Подобно одиночному персептрону, многослойную сеть можно обучать методом градиентного спуска. Основные алгоритмы будут во многом сходны, но способ вычисления градиента ошибки будет другим, поскольку рассмотренный способ подходит только для последнего слоя. Для расчета используется метод *обратного распространения ошибки* (Back Propagation). Рассмотрим его более подробно.

Пусть, как и прежде, в качестве функции ошибки $E(X, W)$ взята сумма квадратов отклонений полученных ответов $Y(X, W)$ от требуемых ответов $\hat{Y}$.

Введем следующие обозначения:

L — число слоев;

l — номер слоя, $l = 1, ..., L$;

p — номер элемента в слое, $p = 1, ..., P_l$;

k — номер предъявленного сети образца, $k = 1, ..., n$;

i — номер веса в элементе, $i = 1, ..., N_l$;

$\hat{y}_{kp}$ — желаемый отклик на p-м выходе на k-й образец;

y_{klp} — отклик p-го элемента l-го слоя на k-й образец;

u_{klp} — отклик p-го элемента l-го слоя на k-й образец до порогового элемента;

w_{lpi} — i-й вес p-го элемента l-го слоя;

x_{klpi} — i-й вход p-го элемента l-го слоя для k-го образца.

Тогда ошибка сети составит

$$E(X,W) = \frac{1}{2}\sum_{k=1}^{n}\sum_{p=1}^{m_L}(y_{kLp} - \hat{y}_{kp})^2.$$

Найдем производную ошибки E по выходу ΔY:

$$\Delta Y_{kLp} = \frac{\partial E}{\partial y_{kLp}} = y_{kLp} - \hat{y}_{Lp}.$$

Производная ошибки по весу ΔW

$$\Delta W_{lpi} = \frac{\partial E}{\partial w_{lpi}} = \sum_{k=1}^{n}\frac{\partial E}{\partial y_{klp}}\frac{\partial y_{klp}}{\partial u_{klp}}\frac{\partial u_{klp}}{\partial w_{lpi}} = \sum_{k=1}^{n}\Delta Y_{klp}\Delta U_{klp}x_{klpi}, \qquad (5.1)$$

где $l = 1, ..., L$; ΔU — производная функции активации.

Помимо производной по весу найдем также и производную ошибки по входу ΔX:

$$\Delta X_{klpi} = \frac{\partial E}{\partial x_{kli}} = \sum_{p=1}^{m_l}\frac{\partial E}{\partial y_{klp}}\frac{\partial y_{klp}}{\partial u_{klp}}\frac{\partial u_{klp}}{\partial x_{kli}} = \sum_{p=1}^{m_l}\Delta Y_{klp}\Delta U_{klp}w_{lpi}. \qquad (5.2)$$

Входы элемента последнего слоя x_{klpi} — это выход элемента предыдущего слоя $y_{k\,(l-1)\,p}$ для всех i, поэтому

$$\Delta Y_{k(l-1)p} = \sum_{i=1}^{N_l} \Delta X_{klpi}. \tag{5.3}$$

Теперь можно подставить значения $\Delta Y_{k\,(l-1)\,p}$ в формулы (5.1)–(5.3), чтобы найти ΔW и ΔY и для предыдущего слоя. Будем продолжать аналогичным образом, пока не найдем весь градиент ошибки.

В данном алгоритме производные ошибки распространяются по сети послойно от выходных слоев сети ко входным в направлении, обратном направлению обработки сигнала. Поэтому данный алгоритм называется алгоритмом *обратного распространения*.

Алгоритм обратного распространения может быть записан и в более общем виде.

1. Для каждого элемента обнулить текущее значение градиента ΔW.
2. Предъявить сети следующий образец выборки и получить ответ Y.
3. Вычислить производную ошибки по выходу: $\Delta Y = Y - \hat{Y}$.
4. Для каждого элемента, получив на вход ΔY, вычислить ΔX и $\Delta W'$ по формулам (5.1) и (5.2).
5. Значение $\Delta W'$ добавить к текущему градиенту: $\Delta W = \Delta W + \Delta W'$
6. Значение ΔX передать как ΔY на все элементы, связанные с текущим.
7. Повторить шаги 4–6, пока не останется элементов, получивших ΔY.
8. Повторить шаги 2–7 для каждого образца из выборки (или пакета).

Данный алгоритм удобен тем, что его можно применять для любых сетей, независимо от топологии их связей: для расчета градиента каждому элементу достаточно знать свой вектор входа, свой отклик на него, вклад в ошибку ΔY, а также знать, какие элементы подключены к его входу. Все эти данные являются локальными для каждого элемента и не требуют общих знаний о структуре сети и принципах ее функционирования.

Сверточные сети. Сверточная сеть состоит из совокупности элементов — сверточных фильтров, собранных в слои, соединенные между собой через функции активации и слои огрубления (рис. 5.24).

Каждый сверточный фильтр, имеющий матрицу весов W, выполняет операцию двумерной свертки изображения I с этой матрицей и получает на выходе новые изображения $I' = I \otimes M$:

$$I'[x,y] = \sum_{\Delta x=1}^{m} \sum_{\Delta y=1}^{m} I[x+\Delta x, y+\Delta y] M[\Delta x, \Delta y],$$
$$x = 1, \ldots, N_x - m + 1, \quad y = 1, \ldots, N_y - m + 1,$$

где N_x, N_y — размеры входного изображения; m — размер маски фильтра.

Размеры выходного изображения при этом уменьшается на $m - 1$. Оно содержит результаты фильтрации: локальные максимумы яркости I' соответствуют положениям элементов, на поиск которых настроен фильтр. Каждый фильтр слоя порождает свое собственное изображение.

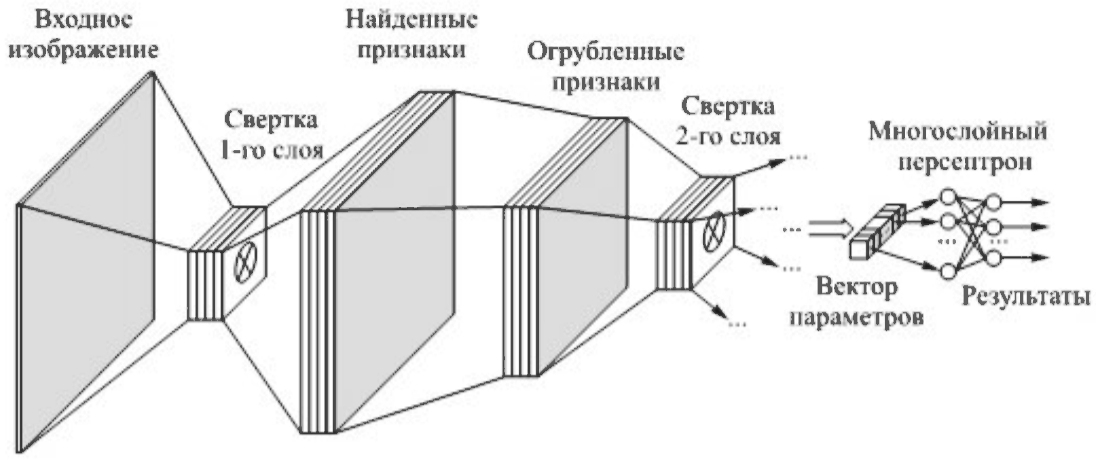


Рис. 5.24. Структура сверточной сети

Затем элементы каждого полученного изображения I' проходят через нелинейный элемент: как и в случае с многослойным персептроном, это необходимо, чтобы предотвратить превращение классификатора в линейный. В качестве нелинейного элемента обычно применяют функции активации, рассмотренные ранее. Чаще всего выбирают линейный элемент с насыщением слева из-за простоты вычисления функции и ее производной.

Следующий этап — огрубление. Он необходим как для уменьшения размера изображений, так и для придания нечувствительности к небольшим сдвигам и другим искажениям ракурса. Для огрубления применяют либо усреднение по блоку точек 2×2, либо (чаще) взятие максимума по блоку 2×2. При этом размеры изображений I' уменьшаются в 2 раза.

Огрубленное изображение подается на следующий слой фильтров. Маски этих фильтров уже трехмерные, поскольку свертка вычисляется для нескольких изображений I'. Результаты фильтрации снова проходят нелинейный элемент, и огрубление подается на следующий слой.

На выходе последнего огрубляющего слоя изображения окажутся огрубленными до 1×1, т. е. получим одномерный вектор. Далее этот вектор подается на любой другой классификатор, обычно на двухслойный персептрон, который дает окончательный ответ о принадлежности объекта к классу.

Структура сверточной сети предполагает, что объект будет описан в виде совокупности структурных элементов и связей между ними. Структурные элементы выделяются фильтрами первых слоев, а связи — последующими слоями. Это, с одной стороны, заставляет классификатор строить максимально общую (генерализованную) модель предметной области и позволяет избежать опасности переобучения, а с другой — радикально сокращает число настроечных параметров. Здесь вновь используется принцип локальности: точки изображения связаны лишь с небольшой окрестностью соседних точек и не связаны с удаленными точками. Поэтому размер фильтра можно выбрать равным размеру этой окрестности. Размеры фильтров начальных слоев выбирают в диапазоне от 7×7 до 11×11, последующих — от 3×3 до 5×5.

Настройка весов сверточной сети аналогична настройке многослойных сетей и может использовать, например, метод градиентного спуска с обратным распространением ошибки. Прохождение ошибки через сверточный слой полностью аналогично ее прохождению в многослойной сети. При прохождении ошибки через слой огрубления ошибка распространяется только по элементам, давшим максимум.

Важным свойством сверточных сетей является возможность перенесения настроек. Настройки нижних слоев, полученных для одной сети в данной предметной области, можно перенести на сеть другой архитектуры. При этом необходимо будет настроить только верхние слои (со сравнительно небольшим числом коэффициентов), заморозив веса нижних слоев. Это возможно потому, что нижние слои работают как детекторы структурных элементов, а в рамках одной задачи они обычно одинаковы.

Другой аналогичной операцией, допустимой для сверточной сети, является возможность замены классификатора верхнего слоя, например на более простой, при сохранении структуры и весов сверточных слоев. При этом настройке подвергаются только веса замененного классификатора. Здесь сверточные слои строят модель предметной области, а классифицироваться она уже может по-разному, в зависимости от условий поставленной задачи.

Недостатком сверточных сетей является то, что они применимы только для входных данных, имеющих вид изображения, т. е. таких, для которых справедлив принцип локальности. Это существенно сокращает область их применения. Кроме того, существуют сложности с выбором структуры сети: методики выбора параметров сети на данный момент не существует, и их определяют для каждой задачи опытным путем.

Литература

Основная

Гонсалес Р., Вудс Р., Эддинс С. Цифровая обработка изображений в среде MATLAB. М.: Техносфера, 2006. 615 с.

Форсайт Д., Понс Ж. Компьютерное зрение. Современный подход. М.: Вильямс, 2004. 466 с.

Шапиро Л., Стокман Дж. Компьютерное зрение. М.: БИНОМ, 2006. 752 с.

Дополнительная

Блейхут Р. Быстрые алгоритмы цифровой обработки сигналов. М.: Мир, 1989.

Хуанг Т.С. Быстрые алгоритмы в цифровой обработке изображений. М.: Радио и связь, 1984.

Brown L. A Survey of Image Registration Techniques // ACM Computing Surveys. 1992. Vol. 24. No. 4.

Canny J. A Computational approach to Edge Detection // IEEE trans. PAMI 8. 1986. P. 679–698.

Davies E.R. Computer & Machine Vision: Theory, Algorithms, Practicalities / London: Academic Press ltd, 2012.

Image Processing Toolbox. Обработка сигналов и изображений. URL: http://matlab.exponenta.ru/imageprocess/

Loncaric S. A survey of shape analysis techniques // Pattern recognition. 1998. Vol. 31. No. 8. P. 983–1001.

Ziou D., Tabbone S. Edge Detection Techniques — An Overview // International Journal of Pattern Recognition and Image Analysis. 1998. Vol. 8. No. 4. P. 537–559.

Оглавление

Учебное издание

Бобков Александр Валентинович

Системы распознавания образов

Редактор *К.А. Осипова*
Корректор *Н.А. Фетисова*
Художник *Э.Ш. Мурадова*
Компьютерная графика *О.В. Левашовой*
Компьютерная верстка *Н.Ф. Бердавцевой*

Оригинал-макет подготовлен
в Издательстве МГТУ им. Н.Э. Баумана.

В оформлении использованы шрифты
Студии Артемия Лебедева.

Подписано в печать 25.06.2018. Формат 70×100/16.
Усл. печ. л. 15,43. Тираж 100 экз. Изд. № 182-2016. Заказ

Издательство МГТУ им. Н.Э. Баумана.
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1.
press@bmstu.ru
www.baumanpress.ru

Отпечатано в типографии МГТУ им. Н.Э. Баумана.
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1.
baumanprint@gmail.com