А. А. Дунаев, И. В. Лобив, Д. Ю. Мехонцев, Ф. А. Мурзин, О. Н. Половинко, Д. Ф. Семич, А. В. Чепель, К. А. Ярков

АЛГОРИТМЫ БЫСТРОГО ПОИСКА ФРАГМЕНТОВ ФОТОГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ*

ВВЕДЕНИЕ

Поиск фрагментов в изображениях — важная задача, которая в том или ином виде возникает во многих отраслях человеческой деятельности. В качестве примера можно привести подсчет количества бактерий на фотоснимке, поиск заданного участка местности на аэрофотоснимке и т. д.

Нередко задача осложняется тем, что искомый фрагмент может отличаться от образца расположением, цветовыми характеристиками, такими как яркость, контрастность, насыщенность цвета и др. Его геометрия может быть искажена, к примеру, вытянута по какому-то направлению.

Наконец, при разработке систем машинного зрения в робототехнике, систем безопасности, военных систем наведения и целеуказания возникает еще одно требование, состоящее в том, что поиск должен осуществляться максимально быстро.

На данный момент в мире существует всего несколько программных продуктов, осуществляющих быстрый поиск образца на изображении для случая, когда образец внутри данного изображения может быть повернут. Еще меньше программ, позволяющих выполнить поиск образца, искаженного аффинным преобразованием.

Один из таких продуктов — MaxVision Toolkit фирмы Datacube. Этот пакет представляет собой набор удобных в использовании, быстрых и высокоточных инструментов для машинного зрения, включающий процедуры калибровки съемочной аппаратуры, ввода, подготовки и анализа изображений и поиска объектов. При поиске осуществляется коррекция перспективного искажения. Для работы пакета требуются специализированные аппаратные средства. Более подробную информацию о фирме и ее продуктах можно получить на веб-сайте фирмы [9].

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

Существуют и другие программные и программно-аппаратные комплексы, осуществляющие поиск объектов теми или иными методами [10–12].

Целью данной работы являлось создание алгоритмов и программ, обнаруживающих заданный фрагмент на изображении. Образец для поиска и изображение, в котором осуществляется поиск, являются обычными образами, т. е. могут быть загружены из файлов в одном из распространенных растровых форматов. Искомый фрагмент может быть повернут, растянут или иметь яркость, контраст и насыщенность цвета, отличные от аналогичных параметров образца.

Реализованная программа состоит из двух частей — вычислительного ядра и управляющей оболочки, осуществляющей пользовательский интерфейс. Вычислительное ядро можно с минимальными изменениями, связанными с особенностями конкретной аппаратной платформы, использовать при разработке других программ. Программа разработана для операционной системы Microsoft Windows NT/2000.

Пользовательский интерфейс обеспечивает удобное управление функциями программы, вывод результатов в понятной форме, возможность скрыть ту информацию, которая временно не требуется пользователю. Поддерживается большинство широко используемых форматов файлов изображений.

1. ОПИСАНИЕ АЛГОРИТМОВ

1.1. Определения и обозначения

Изображение, которое требуется отыскать, будем называть образцом, а то, на котором производится поиск — исходным изображением.

При обработке изображений в программе используется система цветовых координат RGB.

Цветное изображение размером $n \times m$ задается тремя матрицами $S_R = S_R(i, j)$, $S_G = S_G(i, j)$ и $S_B = S_B(i, j)$, где $0 \le i \le n-1$, $0 \le j \le m-1$. Значения элементов матриц $S_R(i, j)$, $S_G(i, j)$ и $S_B(i, j)$ изменяются в пределах от 0 до 255.

Рассмотрим две точки на изображении $p = (i, j)$ и $p' = (i', j')$, имеющие цвета (r, g, b) и (r', g', b') соответственно. Это означает, что

$S_R(i,j)=r$, $S_G(i,j)=g$, $S_B(i,j)=b$, $S_R(i',j')=r'$, $S_G(i',j')=g'$, $S_B(i',j')=b'$. Для краткости будем писать $S_R(p)$ вместо $S_R(i,j)$, $S_G(p)$ вместо $S_G(i,j)$ и т.д.

Определим цветовое расстояние между точками:

$$cd(p,p')=\max\{|S_R(p)-S_R(p')|,|S_G(p)-S_G(p')|,|S_B(p)-S_B(p')|\}. \quad (1.1)$$

Евклидова метрика определяется следующим образом:

$$\rho(p,p')=\sqrt{(i-i')^2+(j-j')^2}\ . \quad (1.2)$$

В процессе сканирования изображения, а также после некоторых преобразований, например, после фильтрации, некоторые цвета могут измениться. Цвета, которые прежде были идентичны, станут различными. В то же время обычно эти цветовые изменения невелики. Поэтому в дальнейшем мы будем использовать специальную константу, которая называется цветовой константой и обозначается C_V. Цветовая константа определяет порог цветового расстояния между двумя цветами, ниже которого эти цвета считаются идентичными, т.е. если

$$cd(p,p')\le C_V\ ,$$

то считается, что точки p и p' имеют один и тот же цвет.

Будем обозначать через $B_n(p)=B_n(i,j)$ квадрат размером $n\times n$ с центром в точке $p=(i,j)$, где n — нечетное.

Перед началом работы алгоритма поиска производится фильтрация некоторых шумов. Существует большое количество различных методов фильтрации, направленных на выделение или подавление тех или иных свойств сигнала. Это направление в обработке сигналов хорошо изучено [5–8]. В настоящей работе изображение подвергается сначала низкочастотной, а затем — медианной фильтрации.

1.2. Низкочастотная фильтрация

Целью применения двумерной низкочастотной фильтрации является получение изображения, имеющего более гладкие цветовые характеристики, чем исходное. Это нужно для того, чтобы сделать процесс поиска изображения более устойчивым. Для каждой точки $p=(i,j)$ вычисляется

среднее арифметическое яркостей в окрестности $B_n(p)$, где n — нечетное, т.е.

$$S'_{R,G,B}(i,j) = \frac{1}{n^2} \sum_{u=-\frac{n-1}{2}}^{\frac{n-1}{2}} \sum_{v=-\frac{n-1}{2}}^{\frac{n-1}{2}} S_{R,G,B}(i+u, j+v). \qquad (1.3)$$

1.3. Медианная фильтрация

Медианная фильтрация состоит в том, что в окрестности каждой точки изображения выбирается точка, значение цвета которой является средним в упорядоченной последовательности значений цветов всех точек окрестности, причем по каждой компоненте цвета фильтрация производится отдельно. Иными словами, мы вычисляем $p^* = (r^*, g^*, b^*)$, где r^* — это средний элемент в упорядоченной по возрастанию последовательности значений компоненты r всех точек окрестности, а g^* и b^* — соответственно g и b. Медианная фильтрация подавляет импульсные помехи.

Одновременно с подавлением импульсных помех удаляются точки с высокочастотными цветовыми характеристиками. Поставим в соответствие с точкой p некоторую ее окрестность $B_n(p)$, где n — нечетное. Тогда, если $cd(p, p') \le C_V$ для достаточно большого количества точек $p' \in B_n(p)$, что задается дополнительной константой, например, от 7 до 9 точек для окрестности 3×3 или от 20 до 25 точек для окрестности 5×5, то полагаем $S'(p) = (r^*, g^*, b^*)$ и $f(i,j) = 0$. В противном случае $S'(p) = (r^*, g^*, b^*)$ остается прежней и $f(i,j) = 1$. Характеристическая функция $f(i,j)$ показывает, подходит ли точка (i,j) для дальнейшего рассмотрения; она принимает значение 0 в точках, имеющих высокочастотные цветовые характеристики, и значение 1 в точках, не имеющих таких характеристик.

1.4. Выделение контуров

Пусть p', p'' — две точки исследуемого изображения,

$$S(p') = (r', g', b'),\ S(p'') = (r'', g'', b'').$$

Пусть теперь

$$cd[B_m(p)] = \max\{cd(p', p'') : p', p'' \in B_m(p)\} \quad (1.4)$$

— максимальное цветовое расстояние в окрестности $B_m(p)$ исследуемой точки. Теперь

$$f(p) = \begin{cases} 0, \ if \ cd[B_m(p)] \geq C_V, \\ 1, if \ cd[B_m(p)] < C_V. \end{cases}$$

— если цветовое расстояние превышает заданный порог, то через рассматриваемую точку проходит контур. Заключение о том, проходит контур через точку или нет, фиксируется соответствующими значениями функции f (0 или 1 соответственно).

1.5. Дополнительные условия преобразований

Описанные преобразования — как фильтрация, так и выделение контуров, — производятся как для изображения-образца, так и для исходного изображения.

1.6. Выбор опорных точек

Вопрос выбора точек, которые будут опорными, является интересным и достаточно глубоко изученным [1–4]. При выборе точек можно учитывать их яркостные характеристики. Также могут использоваться анализаторы топологических свойств изображения. Возможно применение градиентных или пространственно-частотных методов.

В настоящей работе используется эвристический метод, который при испытаниях показал хорошие результаты. Этот метод был предложен Д. Мехонцевым (Институт математики СО РАН), он является одним из вариантов эвристических методов, используемых другими авторами (В. Люцив, В. Малышев, А Можейко, Государственный оптический институт). Выбираются точки, максимально удаленные друг от друга в специальной метрике (правило выбора описано ниже), которая является линейной комбинацией цветовой и евклидовой метрик:

$$\mathrm{P}(p, p') = A \cdot \sqrt{(i - i')^2 + (j - j')^2} + B \cdot cd(p, p'), \quad (1.5)$$

где A и B — константы, в общем случае не зависящие от свойств конкретного изображения. Значения этих констант, при которых поиск наиболее устойчив, можно подобрать опытным путем.

Алгоритм выбора опорных точек работает следующим образом. Сначала изображение сканируется слева направо и сверху вниз. Первая точка контура, встретившаяся при сканировании, выбирается в качестве первой опорной точки S_1. После этого выбирается точка S_2, также лежащая на контуре и при этом максимально удаленная от S_1 в метрике (1.5), затем — точка S_3, лежащая на контуре и при этом максимально удаленная от S_1 и S_2 в метрике (1.6), и так далее до тех пор, пока не будет выбрано требуемое количество опорных точек.

1.7. Построение структуры вспомогательных данных

Для ускорения процесса поиска опорных точек на исходном изображении перед поиском генерируется специальная структура вспомогательных данных, ассоциированная с исходным изображением. На ее построение затрачивается время, пропорциональное размеру изображения $n \times m$, после чего появляется возможность находить опорные точки за логарифмическое время, т.е. за $C \cdot \log(n \times m)$ шагов, где C — константа, характеризующая время выполнения одного шага поиска и не зависящая от размеров исходного изображения.

Ниже описывается процесс построения деревообразной поисковой структуры. Исходное изображение последовательно делится на равные прямоугольники:

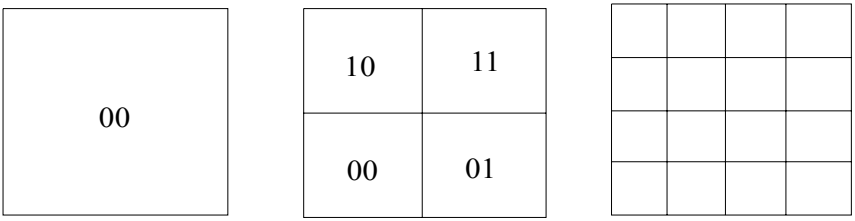


Рис. 1. Разбиение изображения на прямоугольники

Каждое последующее разбиение является более мелким, чем предыдущее. На первом шаге не имеется никакого разбиения. Последнее разбиение состоит из прямоугольников, размер которых равен одному пикселю. Предыдущее (более грубое) разбиение в этой последовательности конструируется из следующего (более тонкого). Прямоугольники, которые содержат данное разбиение, могут быть занумерованы парами целых чисел.

Заметим также, что совокупность всех этих разбиений можно представить в виде дерева, в корне которого располагается самое грубое разбиение, и следующим уровнем для каждого разбиения являются разбиения следующего порядка, на которые оно может быть разложено.

Пусть (i, j) — пара, используемая для нумерации прямоугольника. Обозначим через $R_{r,s}(i, j)$ соответствующий прямоугольник размера $r \times s$.

Теперь опишем метод построения грубых палитр. Значения r, g, b для любой точки представляются посредством 8 бит. Пусть $0 < C_r \le 8$, $0 < C_g \le 8$, $0 < C_b \le 8$, где C_i — компоненты цветовой константы.

Обозначим $\bar{r}$ байт, у которого старшие C_r бит $\bar{r}$ равны соответствующим значениям бит в r, младшие $8 - C_r$ бит $\bar{r}$ равны 0. Аналогично определим значения $\bar{g}$ и $\bar{b}$. В общем случае такая палитра содержит $2^{C_r} \times 2^{C_g} \times 2^{C_b} = 2^{C_r + C_g + C_b}$ цветов. Обозначим множество всех этих цветов

$Pal(C_r, C_g, C_b)$. Таким образом, мы построили грубую палитру, которая не содержит цветов, расстояние между которыми меньше цветовой константы.

Далее с $R_{r,s}(i,j)$ ассоциируется определенная на рассматриваемой палитре характеристическая функция $\chi_{k,ij}$. Эта функция зависит от узла в дереве. Узел характеризуется тройкой (k,i,j), где k — уровень в дереве и (i,j) — пара, индексирующая данный прямоугольник. Аргументы χ_{kij} суть тройка $(\widetilde{r},\widetilde{g},\widetilde{b})$ цветов грубой палитры. Функция χ_{kij} показывает, существует ли точка, имеющая цвет (r,g,b), в прямоугольнике $R_{r,s}(i,j)$ на уровне k, который является близким к цвету (r,g,b).

Пусть $(\widetilde{r},\widetilde{g},\widetilde{b}) \in Pal(C_r, C_g, C_b)$. Определим $\chi_{kij}(\widetilde{r},\widetilde{g},\widetilde{b})$ следующим образом: $\chi_{kij}(\widetilde{r},\widetilde{g},\widetilde{b}) = 1$, если существует такая точка $p \in R_{r,s}(i,j)$, что $S(p) = (r,g,b)$, и для некоторых r', g', b' таких, что $|r - r'| \le C_V$, $|g - g'| \le C_V$, $|b - b'| \le C_V$, выполнено $\overline{r'} = \widetilde{r}$, $\overline{g'} = \widetilde{g}$, $\overline{b'} = \widetilde{b}$.

В противном случае $\chi_{kij}(\widetilde{r},\widetilde{g},\widetilde{b}) = 0$.

Множество связанных с каждым узлом дерева характеристических функций $\chi = \{\chi_{kij} : 0 \le i \le \ldots, 0 \le j \le \ldots\}$ является той самой специальной структурой данных, упоминавшейся выше.

Функция, соответствующая разбиению A, может быть построена дизъюнкцией функций, соответствующих более точным разбиениям, на которые разбивается A.

1.8. Поиск опорных точек

Алгоритм поиска опорных точек работает следующим образом.

Шаг 1. Проверяем, существуют ли точки на исходном изображении, имеющие те же самые цвета, что и опорные точки, с точностью до грубой палитры.

Шаг 2. Рассматриваем более тонкое разбиение и пытаемся найти распределение опорных точек внутри соответствующих прямоугольников, принимая во внимание расстояния между опорными точками.

С каждым шагом мы рассматриваем все более мелкие разбиения. Это может быть сделано рекурсивно. Мы обходим описанное в предыдущем разделе дерево, описывающее вложения прямоугольников друг в друга.

Использование более и более мелких разбиений приведет к фиксации опорных точек.

Например, позиция точки p_1 может быть охарактеризована последовательностью чисел 00, 00, 11, 01, в которой числа индексируют прямоугольники на каждом следующем уровне дерева, в которых находится точка p_1. Мы должны обойти все такие последовательности, принимая во внимание дополнительную информацию (цвета и расстояния), и выбрать подходящую.

Переходя к более тонкому разбиению, мы находим более точные возможные позиции опорных точек. При этом возможны различные методы оптимизации этого процесса.

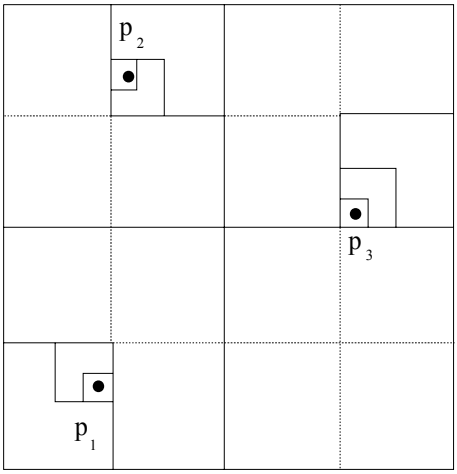


Рис. 2. Итерационный процесс фиксации опорных точек

Опишем алгоритм более детально. В любой момент мы работаем на некотором уровне разбиения, имеющем номер i. Равенство $i = 0$ обозначает, что мы имеем наиболее тонкое (пиксельное) разбиение. Число i возрастает при переходе от более тонкой к более грубой сетке.

Предположим, что выбраны некоторые прямоугольники, являющиеся подходящими для возможного расположения опорных точек $p_0, p_1, p_2, \ldots, p_k$.

Количество опорных точек, рассматриваемое на данном уровне, извлекается из массива max_point, который содержит количества опорных точек для каждого уровня.

Далее, обозначим k — номер последней найденной опорной точки. По ходу работы алгоритма, как только найдена очередная опорная точка, значение k увеличивается на 1. Как только k достигает max-point[i], т.е. последняя точка вписана в какой-то прямоугольник, мы переходим к более тонкой сетке, т.е. на уровень $i-1$.

На более тонкой сетке уточняется положение опорных точек с самого начала, т.е. с точки p_0. Отыскивается прямоугольник возможного расположения точки p_{k+1} на данном уровне i. Заметим, что поиск новых прямоугольников производится внутри прямоугольников, фиксированных на предыдущем уровне.

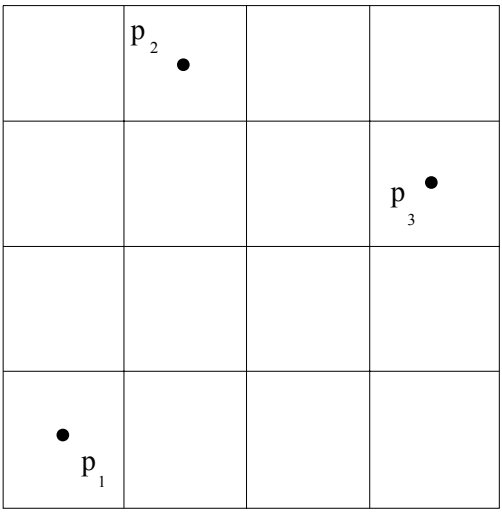


Рис. 3. Фиксация прямоугольников на данном уровне

Алгоритм поиска прямоугольника возможного расположения точки p_{k+1} выполняется следующим образом.

Предположим, что прямоугольники $Q_0,\cdots Q_k$ уже фиксированы. Рассмотрим новый прямоугольник Q_{k+1}. Введем параметры lp и $ps[i]$. Параметр ps будет зависеть от уровня.

Полагаем $lp = 0$, когда Q_{k+1} не содержит подходящего цвета. Этот случай не интересен, поскольку наличие цвета в прямоугольнике проверяется тривиальным образом.

Далее, $lp = 0$, если Q_{k+1} содержит подходящий цвет, но не соответствует по расстоянию между данным Q_{k+1} и Q_0. Аналогично $lp = 1$, если Q_{k+1} содержит подходящий цвет, Q_{k+1} имеет подходящее расстояние между данным Q_{k+1} и Q_0, но неподходящее расстояние между данным Q_{k+1} и Q_1.

В общем случае lp равно максимальному t такому, что Q_{k+1} имеет подходящее расстояние между Q_{k+1} и $Q_0,\cdots Q_{t-1}$, но неподходящее расстояние между Q_{k+1} и Q_t.

Таким образом, мы рассматриваем все возможные позиции для Q_{k+1}. Если подходящая позиция не найдена, то мы сдвигаем прямоугольник Q_{lp}. Это означает, что $Q_0,\cdots Q_{lp}$ выбраны корректно. Процесс продолжается, но мы ищем новую позицию для Q_{lp}. Записываем k в параметр ps в случае, когда $ps < k$. В самом начале полагаем $ps = 0$.

Мы можем получить ситуацию, когда процесс нахождения новых опорных точек не может быть продолжен на данном уровне. Тогда мы должны подняться на предыдущий уровень. В этом случае оказывается, что $lp = 0$, и нет никакой возможности найти новое расположение для p_0. Это возможно из-за того, что прямоугольники найдены внутри прямоугольников, фиксированных на предыдущем уровне, но нужные позиции не существуют на данном уровне. Тогда мы поднимаемся на предыдущий уровень и сдвигаем прямоугольник Q_{ps}, т.е. мы предполагаем, что позиции $p_0,\ldots,p_{ps-1}$ найдены корректно.

Из определения ps, данного выше, легко видеть, что этот параметр фиксирует максимальное число точек, которые были размещены успешно. Но параметр ps используется только в том случае, когда был осуществлен возврат на более грубую сетку.

Добавим, что мы должны учитывать не только расстояния между точками, но и их ориентацию, чтобы исключить отражения. Это может быть сделано проверкой знака соответствующего определителя

$$sign\begin{vmatrix} i_2 - i_1 & j_2 - j_1 \\ i_3 - i_2 & j_3 - j_2 \end{vmatrix},$$

где индексы у компонент соответствуют номерам точек, ориентация которых исследуется.

1.9. Поворот изображения

При финальном сравнении может оказаться, что образец, найденный на исходном изображении, повернут относительно его исходного положения. В этом случае применяется алгоритм поворота найденного участка исходного изображения, затем повернутый фрагмент сравнивается с образцом. Алгоритм работает следующим образом. На исходном изображении вычисляются координаты углов фрагмента, который необходимо повернуть.

Цвет пикселя результирующего изображения с координатами (i, j) устанавливается равным цвету пикселя исходного изображения, в который попал угол пикселя повернутого фрагмента с координатами (i, j). Искажения, вносимые таким алгоритмом поворота, слабо влияют на устойчивость поиска, поскольку на стадии финального сравнения используются фильтрованные изображения. Фильтры, как уже отмечалось ранее, придают изображениям необходимую гладкость и подавляют импульсные помехи, которые при искажениях, привнесенных при повороте, дали бы заметное изменение разности сравниваемых изображений.

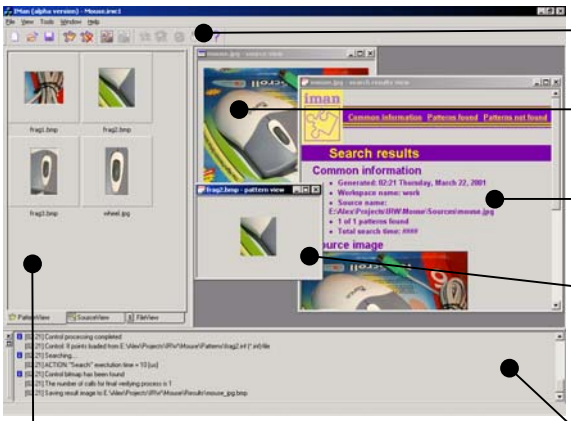
2. ОПИСАНИЕ РЕАЛИЗАЦИИ ПРОГРАММЫ

2.1. Общая схема реализации и интерфейс

Программа состоит из двух основных блоков, которые выполняют принципиально разные задачи. Первый блок реализует пользовательский интерфейс, будем в дальнейшем называть его интерфейсным блоком или просто интерфейсом. Второй блок выполняет подготовку изображений и их обработку.

Интерфейсная часть программы спроектирована по смешанной схеме; хотя одновременно может быть открыто не более одного документа, внешне программа выглядит типичным для многодокументных приложений образом (рис. 4). В нашем случае документом является так называемый проект, в котором хранится информация о файлах используемых исходных изображений и изображений-образцов.

В приложение встроен мастер создания нового проекта, позволяющий задать имя нового проекта и путь к папке, в которой этот проект будет создан. Мастер также позволяет задать один из двух режимов работы с добавляемыми в проект файлами изображений. В одном режиме файлы изображений копируются в папку проекта, а в проект добавляются только имена файлов, в другом режиме файлы не копируются, а в проект добавляются абсолютные пути к ним.



Панель инструментов.

Вид исходного изображения.

Вид результатов поиска.

Вид образца

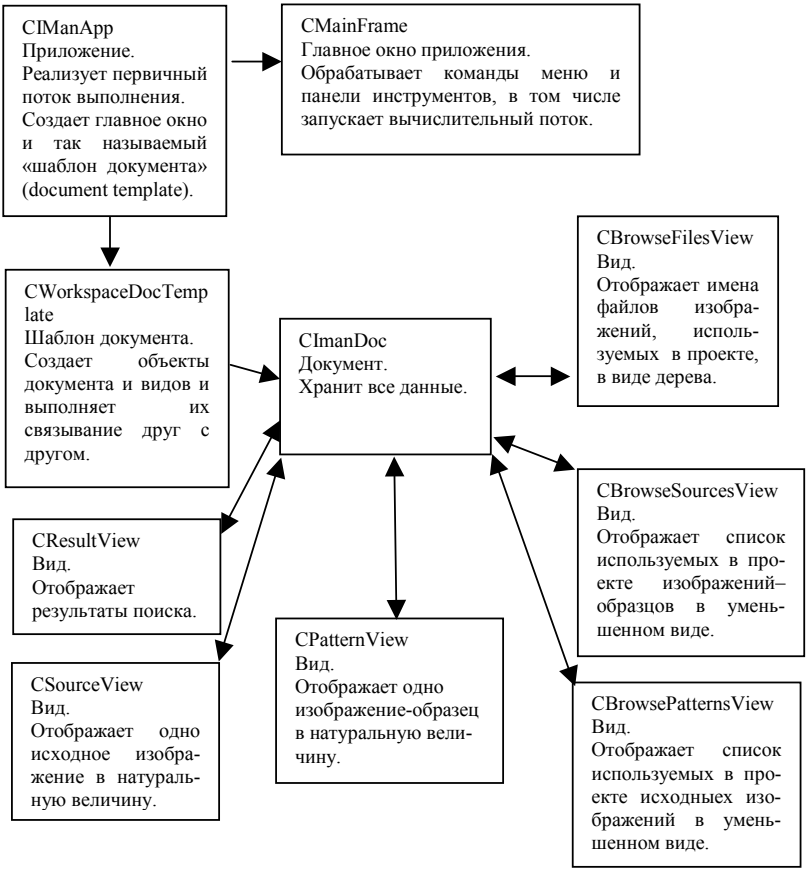
Виды предпросмотра исходных изображений, предпросмотра образцов и структуры проекта объединены в единую панель проекта, выбор нужного вида осуществляется выбором закладки внизу панели

Журнал событий

Рис. 4. Главное окно программы

Главное окно приложения позволяет выбрать исходное изображение и образцы для поиска, а затем выполнить поиск.

Результаты поиска сохраняются в формате HTML. Для масштабирования изображений используется библиотека NexgenIPL (свободно распространяемый продукт компании Binary Technologies, [13]). В процессе разработки классов панели проекта и журнала событий использованы некоторые решения, доступные на веб-сайте Codeguru, посвященном программированию [14].



В приложение встроены две утилиты, облегчающие поиск графических файлов на локальных дисках. Первая утилита отображает дерево папок выбранного локального диска, и пользователь имеет возможность открыть нужную ему папку и добавить в проект находящиеся в ней графические

файлы. Вторая утилита осуществляет поиск файлов по маске и выводит список файлов, найденных на заданном локальном диске, с тем чтобы пользователь мог добавить файлы в проект из списка найденного.

Интерфейс программы разработан с использованием библиотеки классов MFC в соответствии с идеологией «документ-вид».

Вычислительный блок реализован в виде набора классов, предназначенных исключительно для хранения данных, и нескольких процедур, выполняющих операции над ними. Вся работа с этими процедурами выполняется в специальной функции, запускаемой в отдельном потоке, приоритет которого устанавливается ниже, чем приоритет главного потока.

Теперь коротко остановимся на программной реализации фильтров. Везде ниже по тексту переменные m и n обозначают соответственно ширину и высоту изображения.

2.2. Реализация низкочастотного фильтра

Алгоритм двумерной низкочастотной фильтрации реализован стандартным для методов рекурсивной фильтрации образом, поэтому описание его реализации вынесено в Прил. 1.

2.3. Реализация медианного фильтра

Алгоритм медианной фильтрации так же, как и алгоритм двумерной низкочастотной фильтрации, реализован стандартным образом; описание реализации см. в прил. 1.

2.4. Выделение контуров

Выделение контуров выполняется точно так, как описано в п. 1.4, т.е. при каждой итерации алгоритма вычисляются цветовые расстояния для всех возможных пар точек исследуемой окрестности, затем выбирается максимальное значение расстояния.

2.5. Формирование вспомогательных данных

В программе функция χ_{kij} реализована с помощью подпрограммы-функции, обращение к которой имеет вид

```
this_color(level k, coordinate i, coordinate j, color RGB, int
                                    position, int mask)
```

Первые три параметра являются координатами, четвертый параметр задает цвет, который требуется найти. Объясним значения остальных двух параметров, position и mask.

Поскольку грубая палитра содержит $2^{C_r+C_g+C_b}$ цветов, для хранения значений функций χ_{kij} достаточно использовать $2^{C_r+C_g+C_b}/32$ слов длиной в 32 бита. Например, если $C_r = C_g = C_b = 4$, то достаточно использовать 128 32-битных слов.

Тройка $(\bar{r},\bar{g},\bar{b})$ может быть представлена в бинарной форме как конкатенация бинарных слов $\bar{r}$, $\bar{g}$, $\bar{b}$. Таким образом, мы получаем одно бинарное число вместо трех цветов. Обозначим это число bin. Тогда position = bin/32.

Значение этого числа следующее. Мы имеем бинарное слово длины $2^{C_r+C_g+C_b}$, разбитое на блоки длиной 32 бита. Значение position показывает, в каком блоке расположено значение bin.

При использовании структуры в поисковом процессе мы можем найти необходимый бит с помощью конъюнкции 32-битных слов с подходящей маской, имеющей форму $0\ldots010\ldots0$. Эта маска передается параметром mask.

2.6. Поиск опорных точек

Заметим, что в процессе поиска мы имеем дело с прямоугольниками, а не с точками.

Предположим, что мы имеем прямоугольники на некотором уровне дерева. Обозначим $d_{\min}$ и $d_{\max}$ — минимальное и максимальное расстояния между ними.

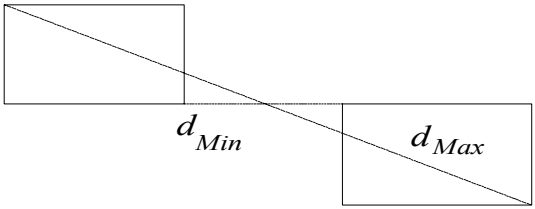


Рис. 5. Минимальное и максимальное расстояния между прямоугольниками

Пусть изображение S разделено на равные прямоугольники Q_{ij}. Всевозможные минимальные и максимальные расстояния между ними хранятся в виде двумерной таблицы расстояний. Более точно минимальные и максимальные расстояния от Q_{00} (прямоугольника с номерами 00) до Q_{ij} (прямоугольника с номерами *ij*) располагаются в позиции таблицы, имеющей номера *ij*. Расстояние между произвольными прямоугольниками может быть вычислено с помощью параллельного переноса одного из них в начало координат. Если левые нижние углы прямоугольников имеют координаты (i_1, j_1) и (i_2, j_2) соответственно, то после переноса координаты одного из них станут $(0,0)$, а другого

$$i_{new} = | i_1 - i_2 |, \; j_{new} = | j_1 - j_2 | .$$

Таким образом, расстояния могут быть взяты из той же самой таблицы расстояний.

Расстояния между опорными точками на образце известны. В процессе поиска опорных точек на исходном изображении приближенно проверяются расстояния между прямоугольниками на исходном изображении:

$$d_{\min} - pp \le d \le d_{\max} + pp ,$$

где d — известное расстояние на образце, а pp — допустимый дефект. В таблице расстояний хранятся $d_{\min} - pp$ и $d_{\max} + pp$, т.е. принимается во внимание дефект.

2.7. Организация обратной связи от вычислительного блока к интерфейсу

Передача информации от вычислительного блока к интерфейсу выполняется посредством сообщений Windows. Большинство функций ядра отправляют интерфейсу сообщения, которые заносятся в журнал событий.

ЗАКЛЮЧЕНИЕ

В результате проведенных исследований создан ряд алгоритмов, которые предназначены для быстрого поиска внутри данного изображения образцов, повернутых на некоторый угол и имеющих другой масштаб.

В алгоритме поиска используется метод опорных точек. На изображении-образце специальным образом выбираются опорные точки, которые затем отыскиваются на исходном изображении.

Результаты исследований могут быть применены в следующих областях:

- спутниковое объектное распознание, аэрофотосъемка и картографирование;
- для поиска в архиве фотографических изображений. Это могут быть фотографии кристаллической структуры металла, биологических материалов, полученных с помощью микроскопа и т.д.;
- на поточной линии; алгоритм может выбирать детали и действовать в системе машинного зрения для автоматизированных сортировочных машин, т.е. он может использоваться на сложных роботизированных производствах.

Реализована программа Iman, которая берет источник и список образцов, выясняет, включен ли каждый образец полностью в источник и устанавливает угол поворота для образцов, которые включены. Результаты поиска помещаются в отчет работы программы, который генерируется в виде HTML-файла, и может быть просмотрен с использованием внутреннего средства просмотра сообщений программы IMan или в броузере сети.

Разработанный алгоритм является одним из лучших в мире и вызывает большой интерес у японских специалистов в связи с применением в робототехнике. Главное преимущество нового подхода состоит в том, что все образцы разыскиваются одновременно. В типичных ситуациях, интересных с точки зрения приложений в робототехнике, время поиска образца измеряется миллисекундами.

СПИСОК ЛИТЕРАТУРЫ

1. **Братцев С. Г., Мурзин Ф. А., Нартов Б. К., Пунтус А. А.** Конфликт сложных систем. Модели и управление. — М.: Изд. МАИ, 1995.
2. **Братцев С. Г., Мурзин Ф. А., Нартов Б. К.** Исследования по обработке динамических изображений // Тез. междунар. конф. по обработке изображений и дистанционным исследованиям. — Новосибирск, 1990. — С. 41–43.
3. **Bratsev S. G., Murzin F. A., Nartov B. K.** The optimum search of targets and the processing of dynamical images // Proc. of the Intern. Symp. on Visual Analysis and Interface. — Novosibirsk, 1991. — P. 17.
4. **Грицык В. В.** Распараллеливание алгоритмов обработки информации в системах реального времени. — Киев: Наумова думка, 1981.

5. **Писаревский А.Н. и др.** Системы технического зрения. — М.: Машиностроение, 1988.
6. **Прэтт У.** Цифровая обработка изображений. — М: Мир, 1982. — Т. 1–2.
7. **Обработка** изображений / Под ред. Хуанга. — М.: Мир, 1979.
8. **Голд Б., Рэйдер Ч.** Цифровая обработка сигналов. — М.: Советское радио, 1973.
9. **Datacube**, Inc. `http://www.datacube.com`
10. **Media** Cybernetics, L. P. `http://www.optimas.com`
11. **Ronald A.** Massa Associates `http://www.way2c.com`
12. **Impuls** GmbH `http://www.impuls-imaging.com`
13. **Binary** Technologies `http://www.binary-technologies.de`
14. **Codeguru** `http://www.codeguru.com`

Приложение 1

РЕАЛИЗАЦИЯ ФИЛЬТРОВ

Реализация низкочастотного фильтра

Описание дано для фильтра с окрестностью размером 3×3.

Сначала для всех $0 \le j \le m-1$ вычисляются частичные суммы:

$$\sigma_R(j)=\sum_{i=0}^{2} S_R(i,j)\ ,\ \sigma_G(j)=\sum_{i=0}^{2} S_G(i,j)\ ,\ \sigma_B(j)=\sum_{i=0}^{2} S_B(i,j)\ .$$

Иными словами, суммируются яркости (отдельно для каждого цвета) по столбцам высотой 3 для всех столбцов внутри слоя, образованного тремя верхними рядами точек.

Вычислив $\sigma_R(0),\sigma_R(1),\sigma_R(2)$ и т.д., получим равенства:

$$S'_R(1,1)=\frac{1}{9}\sum_{k=0}^{2}\sigma_R(k)\ ,\ S'_G(1,1)=\frac{1}{9}\sum_{k=0}^{2}\sigma_G(k)\ ,\ S'_B(1,1)=\frac{1}{9}\sum_{k=0}^{2}\sigma_B(k)\ .$$

Затем, двигаясь слева направо, получим:

$$S'_R(1,j+1)=S'_R(1,j)-\sigma_R(j-1)+\sigma_R(j+2)\ ,$$
$$S'_G(1,j+1)=S'_G(1,j)-\sigma_G(j-1)+\sigma_G(j+2)\ ,$$
$$S'_B(1,j+1)=S'_B(1,j)-\sigma_B(j-1)+\sigma_B(j+2)\ .$$

Таким образом, в процессе продвижения от одной окрестности $B_3(1, j)$ к следующей за ней $B_3(1, j+1)$ мы вычитаем сумму, соответствующую самому левому столбцу старой окрестности, и прибавляем сумму, соответствующую самому правому столбцу новой окрестности.

После прохождения верхнего слоя сдвигаемся вниз и вычисляем

$$S'_R(2, j), S'_G(2, j), S'_B(2, j), 1 \le j \le n-2.$$

Необходимо вычислить аналогичные суммы для столбцов высотой 3. Это можно сделать таким способом. Удаляем из столбца его верхний элемент и добавляем снизу новый. В результате получим:

$$\sigma_R(k) := \sigma_R(k) - S_R(0,k) + S_R(3,k),$$
$$\sigma_G(k) := \sigma_G(k) - S_G(0,k) + S_G(3,k),$$
$$\sigma_B(k) := \sigma_B(k) - S_B(0,k) + S_B(3,k).$$

Таким образом, имеем равенства

$$\sigma_R(k) = \sum_{i=1}^{3} S_R(i,k),\ \sigma_G(k) = \sum_{i=1}^{3} S_G(i,k),\ \sigma_B(k) = \sum_{i=1}^{3} S_B(i,k).$$

С их помощью мы можем двигаться внутри следующего слоя, состоящего из первого, второго и третьего рядов точек. Легко видеть, что этот алгоритм можно продолжать от слоя к слою, причем при каждой следующей итерации алгоритма большая часть данных используется с предыдущей итерации.

Релизация медианного фильтра

В фильтре используется сортировка со слиянием и удалением элементов. В общем случае имеется квадрат точек (здесь, как и в предыдущем разделе, описана реализация фильтра с окрестностью размером 3×3). Данные, соответствующие этим точкам, уже отсортированы при предыдущей итерации алгоритма и представлены в виде упорядоченного списка. Данные, соответствующие правому столбцу, также сортируются и представляются в виде списка. После этого эти два списка накладываются один на другой, одновременно отбрасываются данные по самому левому столбцу.

Таким образом, используя сортировку со слиянием и удалением элементов, мы получаем данные, соответствующие следующему квадрату точек, причем эти данные уже отсортированы. Выбирая элемент, расположенный

в середине списка, мы будем иметь именно тот элемент, который нам нужен для медианной фильтрации.

Обработав верхний слой, сдвигаемся вниз. Предположим, получился столбец

$$\begin{pmatrix} (i-1,k) \\ (i,k) \\ (i+1,k \end{pmatrix}.$$

Затем мы сортируем этот столбец в порядке возрастания, принимая во внимания значения $S_R(l,k), l = i-1, i, i+1$. Другими словами, мы получим отсортированный список значений компоненты цвета точек, формирующих данный столбец.

Теперь нам нужен отсортированный список, соответствующий столбцу

$$\begin{pmatrix} (i,k) \\ (i+1,k) \\ (i+2,k \end{pmatrix},$$

который получается сдвигом вниз на один пункт. Его можно получить, используя сортировку с добавлением и удалением элементов. В данном случае мы удаляем верхний элемент и добавляем недостающий нижний элемент. Затем аналогичным образом сдвигаемся слева направо, чтобы построить следующий столбец.

Приложение 2

РЕЗУЛЬТАТЫ ИСПЫТАНИЙ

Испытания программы произведены на ЭВМ со следующей системной конфигурацией: процессор Intel Pentium III (тактовая частота 500–1200 МГц), объем ОЗУ 128–256 Мб, управляющая ОС — Microsoft Windows 2000.

Применялась следующая методика тестирования. Выбирается исходное изображение, из него (или из других изображений) выделяются образцы, которые необходимо найти. Затем несколько раз выполняется поиск выде-

ленных фрагментов. Статистика времени поиска вычисляется по результатам теста на нескольких различных исходных изображениях.

Тест 1. Среднее время поиска образца. Исходное изображение имеет размер 1024 на 768 пикселов, образцы выбирались размером 64 на 64 пиксела.

Тест 2. Среднее время поиска повернутого образца. Исходное изображение имеет размер 1024 на 768 пикселов, выбран образец размером 64 на 64 пиксела, который затем размножен поворотом на углы от 0° до 350° с шагом в 10°. Поворот образца выполнен при помощи пакета Adobe Photoshop. Аналогично производился поиск для каждого из полученных изображений.

Тест 3. Среднее время поиска фрагмента, не присутствующего на исходном изображении. Исходное изображение имеет размер 1024 на 768 пикселов, выбраны 5 образцов размером 64 на 64 пиксела, не присутствующие на исходном изображении и имеющие различные цветовые характеристики, от сходных характеристик исходного изображения до, в некотором смысле, противоположных. Соответственно, проводились попытки поиска для каждого из изображений.