Д. Ю. Мехонтцев, И. В. Лобив, К. С. Селезнев

СЛЕЖЕНИЕ И ОПРЕДЕЛЕНИЕ СКОРОСТИ ДВИЖУЩИХСЯ НА ПЛОСКОСТИ ОБЪЕКТОВ В РЕАЛЬНОМ ВРЕМЕНИ*

1. ВВЕДЕНИЕ

В последнее время в связи с ростом мощности компьютеров появилась возможность создания относительно дешевых программно–аппаратных систем, реализующих обработку большого количества информации, поступающей в реальном времени. Примером может служить система отслеживания движения объектов, на вход которой поступают изображения, полученные с видеокамеры, а на выходе имеются данные об объектах, попавших в поле зрения, а также об их динамических характеристиках, например, скорости и ускорении.

Нашей группой успешно реализована система такого рода, в которой в качестве аппаратной части выступает обычный персональный компьютер класса Pentium-2, а в качестве программной части используется логический модуль нашей собственной разработки, устройству которого и посвящена данная статья.

К особенностям системы можно отнести следующие пункты:

1) камера должна обозревать от кадра к кадру одну и ту же область пространства;
2) допускаются цветовые шумы и небольшое дрожание камеры;
3) отслеживаемые объекты могут плавно менять форму и размер (например, приближающийся автомобиль);
4) не допускается пересечение движущихся объектов одинакового цвета на продолжительное время.

2. ПОСТАНОВКА ЗАДАЧИ

Рассмотрим декартову систему координат (x, y, z). Возьмем некоторую точку $S(0,0,h_0)$. Под углом α через точку S проведена плоскость (плос-

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

кость экрана) параллельно оси y. На этой плоскости зададим экран — прямоугольник длиной L (параллельной оси y), высотой H и с пересечением диагоналей в точке S. Глаз наблюдателя E находится на расстоянии d от точки S плоскости экрана. По плоскости (x, y) вдоль вектора V движутся некоторые трехмерные объекты.

Наша задача — определить их мгновенную и среднюю скорость при прохождении в поле видимости. Расчеты должны происходить в реальном времени.

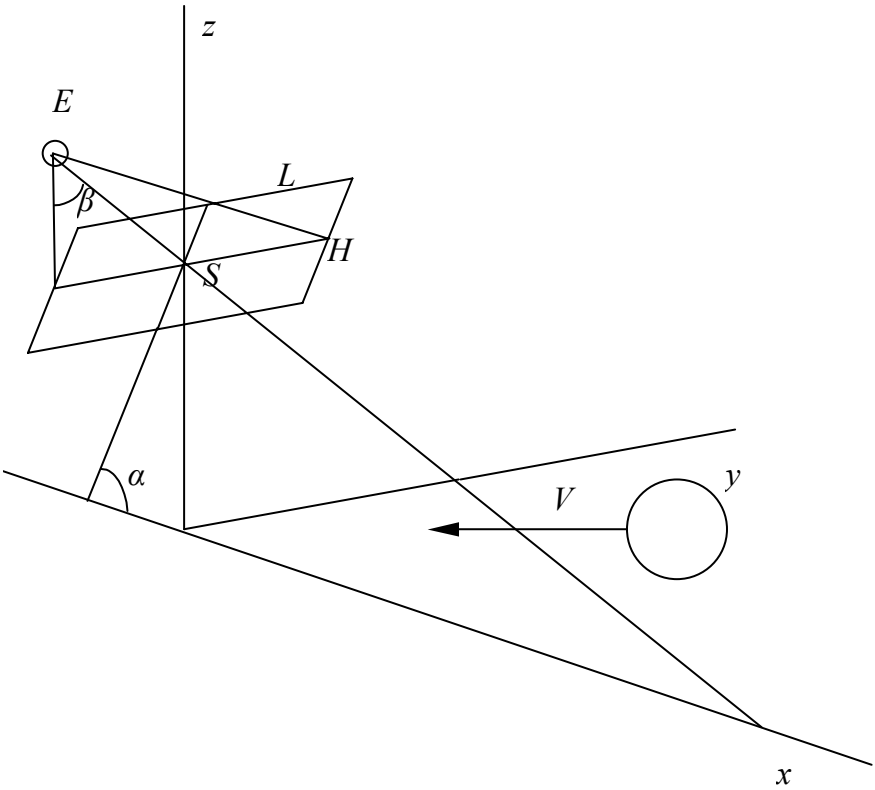

Рис. 1. Постановка задачи

3. БЛОК - СХЕМА

4. ОПИСАНИЕ РАБОТЫ АЛГОРИТМОВ

На вход логическому модулю подается последовательность изображений (bitmap sequence). Требуется некоторое время (10-20 кадров) для расчета дороги (автонастройки). Затем выделяются движущиеся объекты на изображении, и формируются цепочки. По мере поступления изображений (bitmaps) рассчитывается их скорость. В момент достижения объектом конца области видимости логический модуль добавляет его к списку найденных объектов.

Этап 0. Начало работы, инициализация алгоритмов.

Этап 1. Расчет «дороги».

Алгоритм расчета дороги состоит в следующем. Берем из потока несколько, желательно как можно более различных, кадров (для того, чтобы дорога определилась без помех). Далее для каждого пиксела проводим на всех взятых кадрах попарные сравнения и выявляем такие пикселы (pixels), которые не двигаются — это пикселы, принадлежащие «дороге», а остальные принадлежат машинам. Если на выходе получилась «дорога» без «дырок», то значит, она нашлась вполне удовлетворительно и вероятность ошибки снижается. Заметим, что эта процедура довольно трудоемкая, но она выполняется редко: один раз при настройке (например, один раз в час либо при резком изменении погодных условий).

Мы испробовали множество алгоритмов. В некоторых ситуациях подходит один (к примеру, когда требуется провести процедуру как можно быстрее), в других ситуациях — другой. Мы опишем только два: самый простой и тот, который используется в настоящее время.

Самый простой способ расчета «дороги».

Возьмем последовательность фреймов (frame) $F_0,...,F_{N-1}$. Определим фрейм дороги R, т.е. фрейм, на котором нет ни одной движущейся области — «машины». (Из этого, кстати, следует, что стоящие на обочине машины для нас тоже являются частью дороги, из-за их неподвижности в момент расчета).

Итак, введем вспомогательное множество D, в котором мы будем хранить пары вида (x, y), если мы уже приняли решение о цвете пикселя (x, y) фрейма «дорога». Соответственно, цвет пикселов, пары которых не включены в множество, находится под вопросом.

$D = 0,$ т.е. множество пикселов дороги изначально пусто;

$R_0 = F_0,$ нулевое приближение — это 0-й фрейм;

$R_1 = R_0 - F_1,$ первое приближение — из нулевого «вычесть» первый фрейм;

.....

$R_{N-1} = R_{N-2} - F_{N-1};$

где разность фреймов, $A - B$, определяется следующим образом:
$\langle i, j \rangle \in D \Rightarrow a_{ij} - b_{ij} = a_{ij},$

$$\langle i,j\rangle \notin D \Rightarrow a_{ij} - b_{ij} = \begin{cases} a_{ij}, \text{если } a_{ij} == b_{ij}. \text{В этом случае } D = D \cup (i,j); \\ \text{иначе неопределено.} \end{cases}$$

Сравнение двух цветов ($a_{ij} == b_{ij}$) мы ведем в некоторой цветовой норме, например,

$$c^1 == c^2 = \begin{cases} 1, \text{ если } m < C; \\ 0, \text{ иначе,} \end{cases}$$

где $m = \left|c_R^1 - c_R^2\right| + \left|c_G^1 - c_G^2\right| + \left|c_B^1 - c_B^2\right|$, C — константа.

В итоге получим

$$R = R_N = \begin{cases} r_{ij}^{N-1}, \text{если } r_{ij}^{N-1} \text{ определен;} \\ \mathrm{f}_{\mathrm{ij}}^0, \text{иначе.} \end{cases}$$

Легко увидеть основной недостаток данного алгоритма — фрейм «дороги» будет очень похож на 0-й фрейм, чуть меньше — на 1-й фрейм, ..., меньше всего — на $N-1$ фрейм, т.е. если 0-й фрейм последовательности мы взяли неудачно, то и фрейм «дороги» будет очень не похож на реальную дорогу, и мы очень часто будем ошибаться в выделении «машин».

Текущий способ расчета «дороги».

Рассмотрим последовательность цветов точки с координатами (x,y) на всех N фреймах, которые даны для вычисления «дороги», т.е. $c_0,\ldots,c_{N-1}$ — это соответствующие цвета. Рассмотрим подпоследовательность этой последовательности $c_j,\ldots,c_k$, где $0 \le j < N$, $j+1 < k < N$, такую что

$$c_j \neq c_{j+1},$$
$$c_j \neq c_{j+2},$$
$$\ldots,$$
$$c_j \neq c_{k-1},$$
$$c_j = c_k.$$

Результирующий цвет дороги в точке с координатами (x,y) будет цвет c_j, если такая подпоследовательность найдена, и c_0 — иначе.

Другими словами, первый цвет, который удовлетворил:

– первый раз нам встретился,
– исчез в одном или нескольких фреймах,
– далее он опять встретился.

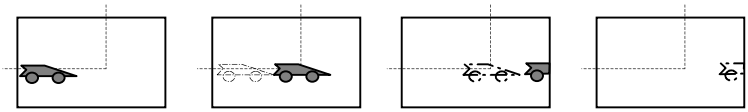

Рис. 2. Движение машины относительно точки (x, y).
Пунктиром показано положение машины на предыдущем фрейме

Этот способ расчета дороги очень хорошо согласуется с реальной жизнью: сначала на некоторое время дорога пуста, потом по ней быстро проезжает машина, и далее опять — начальное состояние. Конечно, можно возразить, а что будет, если друг за другом едут две, к примеру, синих машины. В этом случае можно попробовать посчитать, сколько из N цветов одинаковых, и принять этот цвет за дорогу. В наших экспериментах текущий алгоритм вполне хорошо себя зарекомендовал, если брать фреймы для дороги через достаточно большой промежуток времени друг от друга (мы брали через каждые 10 фреймов).

Этап 2. Начало цикла по кадрам. На вход с камеры начинают поступать изображения (bitmaps).

Этап 2.1 Предобработка кадра.

Этап локализации областей, где потенциально находятся движущиеся объекты

Берем текущий кадр и начинаем каждый пиксел сравнивать с «дорогой» и предыдущим кадром. Таким образом мы выделим области, где могут находиться машины и шумы. Кадр с помеченными областями назовем предобработанным. Опишем подробнее два алгоритма этого этапа локализации.

Простейший алгоритм локализации областей

Пусть F_i — текущий фрейм, а R — фрейм дороги. Рассмотрим цвет точки p с координатами (x, y) на этих фреймах, т.е. c_i, c_R соответственно.

Наша задача — понять, является ли точка p на фрейме F_i частью некоторого движущегося объекта или нет.

Пусть Ω_i — это объединение всех таких областей на фрейме F_i, тогда положим $p \in \Omega_i, \Leftrightarrow c_i \neq c_R$.

Улучшенный алгоритм локализации областей

Рассмотрим $\Omega'_{i-1} \subseteq \Omega_{i-1}$ — объединение всех областей, где точно находятся «машины» на предыдущем фрейме F_{i-1}.

Если $\Omega'_{i-1} = 0$, то пользуемся предыдущим алгоритмом.

В противном случае,

- если $p \in \Omega'_{i-1}$, то $p \in \Omega_i, \Leftrightarrow c_i \neq c_R$,
- если $p \notin \Omega'_{i-1}$, то $p \in \Omega_i, \Leftrightarrow c_i \neq c_{i-1}$.

Видно, что в данном алгоритме ответственность за решение лежит не только на фрейме дороги, но и на предыдущем фрейме. Это позволило всем расчетам стать более устойчивыми к таким явлениям, как дрожание камеры и др. (т.к. у нас все тесты были сняты вручную), и сгладило другие вредные эффекты (например, неудачное нахождение фрейма «дороги»).

Этап выделения контуров в областях, где потенциально находятся движущиеся объекты

Пусть K_j — множество пикселей j-го замкнутого контура. $Int(K_j)$ — внутренность K_j, т.е. все пиксели которые лежат внутри K_j, но не принадлежат ему. Пусть $K = \bigcup_j K_j$ — объединение контуров на текущем фрейме F_i, $Int(K) = \bigcup_j Int(K_j)$. Мы сканируем F_i в поисках пикселя $p \middle| p \notin K, p \notin Int(K)$, например, двигаем однопиксельный сканер слева направо, а сканируем сверху вниз. Пусть мы нашли такой пиксель, тогда он принадлежит к еще не обойденному контуру. Обойдем его, как показано на рис. 3.

Заметим, что этот алгоритм можно легко модифицировать таким образом, чтобы не обходить области толщиной в один или два пикселя (например, линию длиной в 5 пикселей на рис. 3). Такая модификация понадобилась нам для того, чтобы уменьшить влияние шумов, так как на первых тестах контуры были похожи на осьминогов с массивными телами и очень раз-

ветвленными щупальцами, по длине иногда даже в несколько раз превосходившие длину тела.

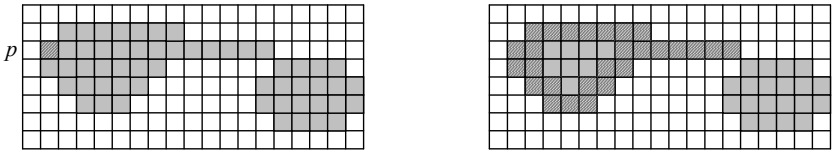

Рис. 3. Обхождение контура. p — «начало» контура.
Штриховкой показаны точки, которые принадлежат контуру

Таким образом мы выделим все контуры. Контуры у которых число точек мало или у них «неправильная» геометрия, являются шумами и отбрасываются. Оставшиеся контуры K_j являются некоторыми «машинами». Для них вычисляется ряд параметров (т.е. центр масс, угловые размеры) для того, чтобы подклеивать их в уже существующие цепочки «машин» D_m или создавать новую цепочку с этим головным элементом.

Итак, мы получили на этом этапе неупорядоченный массив K_j и полностью перешли от пространства пикселей к пространству контуров.

Этап 2.2. Начало цикла по контурам.

По одному начинаем рассматривать каждый контур K_j и соответствующий набор параметров (центр масс, угловые размеры, а также уже рассчитанные некоторые новые, такие как направление движения, среднюю скорость и др.).

Этап 2.2.1. Привязка контура к структуре цепочек.

Рассмотрим контур K_j и цепочку машин $D_z[1..n]$. Проверим, является ли K_j продолжением цепочки $D_z[1..n]$. Для этого сравним параметры и направление движения (машина, например, не может мгновенно изменить направление движения больше, чем на некоторый угол).

Этап 2.2.2. Анализ цепочки.

Пусть параметры K_j с наивысшей точностью совпали с параметрами цепочки $D_z[1..n]$. Тогда мы присоединяем K_j к этой цепочке и пересчитываем некоторые усредненные параметры, связанные с этой цепочкой.

Проанализируем, достигла ли машина конца области видимости. Если да, то замыкаем цепочку и проводим ее анализ на то, является ли она шумом. Если это не шум, значит вычисляем время движения, длину траектории, среднюю скорость и выводим эти данные.

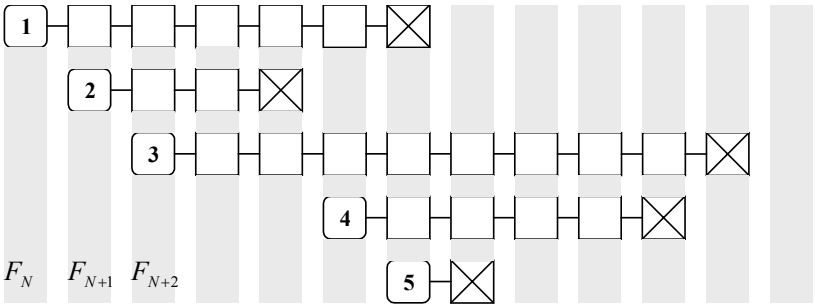

Рис. 4. Анализ цепочек (1..5), которые составлены из контуров (на рис. это прямоугольники), лежащих на фреймах $F_N, F_{N+1}, \ldots$

На рис. 4 показан пример такого анализа. Видно, что цепочки 2 и 4 возможно «склеить» (т.е. в конец цепочки 2 вставить цепочку 4), если у них совпадут средние параметры. Такого рода «склейки» применялись в нашем проекте, так как на некотором этапе мы пришли к тому, что цепочки из-за шумов и неточности вычислений рвались (т.е. начиналась новая цепочка вместо того, чтобы продолжить существующую). Из-за этого мы, бывало, «теряли» машину в центре области видимости. Цепочка 5 — это шум, и во время анализа мы ее вообще отбросим. По остальным рассчитаем среднюю скорость и интегральные параметры, если они еще не завершены.

5. РЕАЛИЗАЦИЯ

В качестве тестов были взяты несколько роликов движения машин по автостраде, отснятых цифровой камерой. Также были взяты несколько на-

боров входных параметров (для каждого ролика — свой набор, определяющий ранее введенные величины). Написан программный продукт "**HighWay**", реализованный на MS VC ++ 6.0 (MFC, C++, MS Direct Show). Он представлен в виде окна (на котором проигрывается avi-файл), списка изображений найденных объектов со скоростями, а также панели управления, с помощью которой можно задать входные параметры и выполнить команды: open video file, start, stop, pause, exit.

Сам продукт выполнен из двух независимых частей.

Логический модуль, в котором реализованы наши уникальные и общеизвестные алгоритмы, выполнен на C++.

Интерфейсная часть (оболочка) использует современные новейшие технологии обработки видеоинформации и написана на MFC и Windows SDK.

Запуск происходит в нескольких потоках: в первом работает распаковка видеофайла, во втором — центр управления, в третьем работает алгоритмическая часть.

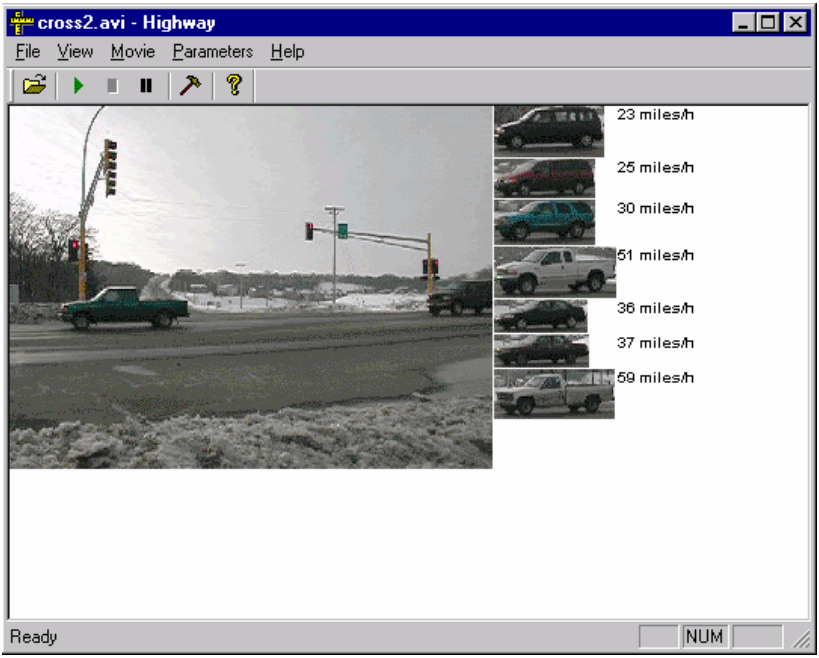

Рис. 5. Внешний вид продукта

Где можно взять для свободного просмотра продукт "HighWay"?
В разделе: http://gs.ieie.nsc.ru/projects/gso/html/gso/image/Light/

- Release 1.1.1 light.zip — сам продукт,
- cross2.zip — сжатый avi-файл теста,
- movie1.zip — сжатый avi-файл теста,
- movie2.zip — сжатый avi-файл теста,

а также
http://www.gadgetsoft.com/cgi-bin/default?page=home/divisions/image/highway

СПИСОК ЛИТЕРАТУРЫ

1. **Pratt, William K.** Digital image processing. 2nd ed.. — Wiley & Sons Ltd., 1991. — 438 p.
2. **Кнут Д.** Искусство программирования на ЭВМ. Т. 3: Сортировка и поиск. — М.: Мир, 1978.
3. The 4th All-Russian with invited foreign participants Conference «Pattern Recognition and Image Analysis: New Information Technologies» ROAI-98, Novosibirsk: IA&E SibRAS, 1998. — Vol. 1, 2.