AUTOMATYKA • 2006 • Tom 10 • Zeszyt 3

Marek Zachara*

Szybka metoda wektoryzacji krawędzi odcinkami w czasie rzeczywistym

1. Wprowadzenie

Przedstawiony w niniejszej pracy algorytm wektoryzacji obrazu powstał w wyniku prac nad aplikacją zdolną w czasie rzeczywistym analizować i interpretować obserwowaną scenę. Aplikacja taka byłaby bowiem użyteczna do konstrukcji samodzielnych robotów zdolnych do poruszania się w nieznanym dokładnie terenie (np. hala magazynowa). Jednym z podstawowych problemów związanym z tworzeniem takiej aplikacji jest wielkość strumienia danych, który musi być przetworzony. Osiągnięcie większej precyzji analizy wymaga na ogół zwiększenia rozdzielczości przetwarzanych ramek obrazu – a co za tym idzie, wielokrotnego zwiększenia ilości danych na każdą ramkę.

O ile w przypadku obrazów statycznych nie rodzi to poważnych problemów, to inaczej jest w przypadku przetwarzania rzeczywistego strumienia wideo. Na przykład, czterokrotne zwiększenie rozdzielczości wzdłuż każdej z osi (poziomej i pionowej) powoduje 16-krotne zwiększenie strumienia danych. Porównywanie dwóch klatek (np. pod kątem przesunięcia) powoduje dalszy wykładniczy wzrost liczby koniecznych obliczeń. W rezultacie okazuje się, że przy dużej rozdzielczości obrazu, wielu operacji nie da się wykonać w czasie rzeczywistym – nawet przy zastosowaniu maszyn wieloprocesorowych.

Ponieważ wzrost szybkości obliczeniowej komputerów nie jest wystarczający do skompensowania w przewidywalnym czasie kilku lat wymagań obliczeniowych takich procesów, niezbędne jest opracowanie algorytmów, które zredukują stopień złożoności problemu, bądź też znaczące zmniejszenie strumienia danych przy zachowaniu istotnej informacji [1].

Wektoryzacja jest ogólnie metodą przekształcenia informacji wizyjnej w struktury przestrzenne oparte na zbiorach odcinków (a czasem również powiązane z nimi tekstury), pozwalające dokonać zadowalającej rekonstrukcji treści obrazu przy jednoczesnej istotnej redukcji danych. Wekotryzacja niestety jest procesem złożonym – prezentowana jednak w niniejszym artykule metoda pozwala na szybką wektoryzację rzeczywistego strumienia danych, którego źródłem jest kamera wideo.

* Katedra Automatyki, Akademia Górniczo-Hutnicza w Krakowie; mzachara@agh.edu.pl

Istniejące implementacje algorytmów wektoryzacji są na ogół ściśle związane z wykonywanym zadaniem, co wynika z faktu, iż jest to proces nie tylko bardzo złożony obliczeniowo, ale i niejednoznaczny (istnieje zwykle wiele możliwości skonstruowania zbioru wektorów dla danego obrazu rastrowego). Istniejące implementacje obejmują najczęściej wektoryzację rysunków technicznych [7], pisma [6] oraz zdjęć i grafik komputerowych [9]. Istnieją też metody wektoryzacji w oparciu o bazę wiedzy (co wymaga znajomości wektoryzowanych obiektów) [10]. Dobry przegląd technik wektoryzacji można też znaleźć w [8] oraz [11].

Wszystkie te metody mają jednak podstawową wadę – nastawione są na wektoryzację statycznych obrazów. Podstawowym priorytetem jest wysoka jakość odwzorowania wektorowego, przy niewielkiej wadze przywiązywanej do czasu przetwarzania. Choć część z tych metod osiąga rezultaty kilku–kilkunastu sekund na jeden wektoryzowany obraz, nie jest to akceptowalne w przypadku zastosowań do przetwarzania w czasie rzeczywistym. Stąd też powstała potrzeba opracowania metody, która nawet kosztem mniejszej dokładności będzie w stanie realizować szybką wektoryzację obrazu z kamery.

2. Opis metody

Przedstawiona poniżej metoda wektoryzacji składa się z trzech podstawowych etapów:

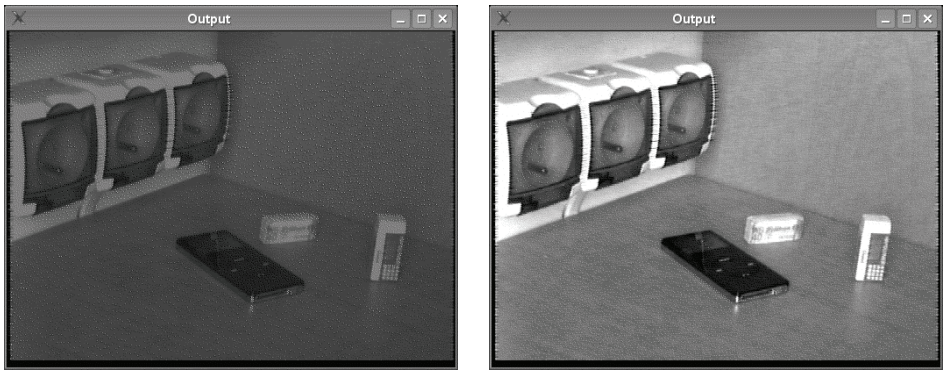
1. detekcji krawędzi,
2. aproksymacji krótkimi odcinkami,
3. łączenia krótkich odcinków leżących wzdłuż jednej prostej oraz posiadających wspólne punkty w jeden dłuższy odcinek.

Taki podział operacji jest wymuszony zastosowanym algorytmem, którego konstrukcja sprawia, że znacznie spada wydajność w przypadku ograniczenia restrykcji poszukiwania wektorów (co byłoby niezbędne do znajdywania dłuższych odcinków).

2.1. Przetwarzanie wstępne

Pierwszym etapem przetwarzania jest akwizycja obrazu. W celu zapewnienia płynności procesu i zredukowania liczby utraconych ramek, stosowane jest podwójne buforowanie kolejnych ramek obrazu. W zależności od warunków oświetleniowych korzystne może być zastosowanie metod poprawienia kontrastu. Wskazane jest w takim przypadku zastosowanie szybkich metod – np. równoważenia histogramu [2]. Ponieważ metoda ta wymaga tylko dwóch odczytów na jeden piksel ramki, nie powoduje zauważalnego zmniejszenia szybkości całego procesu. Przykład działania tej metody został zaprezentowany na rysunku 1.

Poprawa kontrastu z reguły zwiększa skuteczność wydzielania krawędzi z ramki obrazu, choć konkretny wpływ na przetwarzanie zależy od wybranego algorytmu detekcji krawędzi.



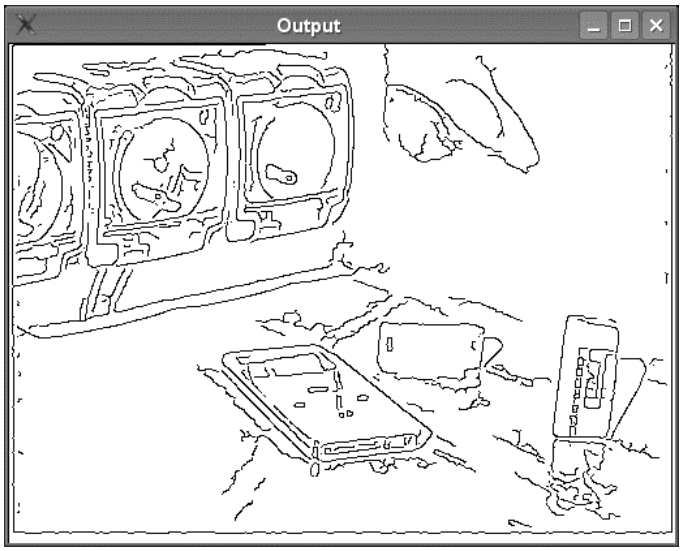
Rys. 1. Wstępna obróbka obrazu – wyrównanie histogramu

2.2. Detekcja krawędzi

Kolejnym krokiem przetwarzania, wciąż przy zachowaniu rastrowej struktury danych ramki, jest wykorzystanie algorytmu detekcji krawędzi. Istnieje możliwość wykorzystania różnych rozwiązań w zależności od potrzeb i przetwarzanej sceny. Dobre porównanie rezultatów działania popularnych algorytmów można znaleźć w [3].

Dla celów wektoryzacji przedstawionej w niniejszym artykule został wybrany algorytm Canny [4] ze względu na dobrą ciągłość wynikowych krawędzi oraz akceptowalny poziom artefaktów w wynikowym obrazie.

Rezultat działania tego algorytmu można zaobserwować na rysunku 2.



Rys. 2. Rezultat działania algorytmu detekcji krawędzi

Detekcja krawędzi ogranicza ilość przetwarzanych danych zgodnie z intencją przedstawioną na początku artykułu, w dalszym ciągu jednak rezultatem jej działania jest rastrowy komplet pikseli (choć zredukowany do binarnego zakresu). Aby efektywnie interpretować informację znajdującą się w obrazie, niezbędna jest wektoryzacja – czyli zamiana informacji rastrowej (to jest informacji o jasności każdego punktu) na wektory, czyli odcinki określone przez początek i koniec w kartezjańskiej przestrzeni ramki obrazu.

2.3. Wektoryzacja

Proces zamiany informacji rastrowej na wektorową nie jest trywialny ze względu na zmianę sposobu reprezentacji danych, a także na fakt, że przekształcenie to jest niejednoznaczne – tj. w szczególności może istnieć wiele reprezentacji wektorowych odpowiadających jednej reprezentacji rastrowej. Wielość możliwych reprezentacji wektorowych wynika m.in. z faktu, iż każdy wektor może być w niej reprezentowany jako suma mniejszych wektorów. Nie jest to pożądane z punktu widzenia dalszego wykorzystania informacji, niemniej może być rezultatem działania algorytmu wektoryzacji.

Prezentowany algorytm opiera się częściowo na idei kodu Freemana i metodzie śledzenia krawędzi [5] – tj. znajdywania przebiegu linii poprzez wyszukiwanie kolejnych zapalonych pikseli w sąsiedztwie aktualnie przetwarzanego. Śledzenie rozpoczyna się od znalezienia punktu, który nie należy do żadnego skonstruowanego już wektora. Tworzony jest wektor „zerowy" v o współrzędnych początku i końca znajdujących się w punkcie startowym

$$\vec{v} = \begin{bmatrix} x_0, y_o \\ x_0, y_0 \end{bmatrix} \tag{1}$$

gdzie:

x_0, y_0 – współrzędne x i y punktu początkowego,
v – budowany wektor.

Tworzona jest też dodatkowa tablica p, w której zapisywane są współrzędne punktów wchodzących w skład konstruowanego wektora. Tabela p dla n punktów wygląda następująco

$$p = \{(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)\} \tag{2}$$

gdzie:

p – tabela zawierająca współrzędne punktów budujących wektor v,
x_n, y_n – współrzędne n-tego punktu w tabeli.

Następnie algorytm przeszukuje najbliższe sąsiedztwo punktu końca wektora v, szukając zapalonych punktów. W przypadku znalezienia takiego punktu, konstruowany jest hipotetyczny wektor v_h

$$\vec{v}_h = \begin{bmatrix} x_0, y_o \\ x_h, y_h \end{bmatrix} \tag{3}$$

Przed konstruowaniem tego wektora sprawdzane jest, czy punkt x_h, y_h nie należy już do tablicy p. W takiej sytuacji wektor ten jest odrzucany i procedura nie jest kontynuowana. Dzięki temu uniknięto zagrożenia zapętlenia poprzez ciągłe dopisywanie tych samych punktów do konstruowanego wektora.

Wektor v_h jest następnie testowany na zgodność z istniejącą tablicą p. Innymi słowy, sprawdzane jest, czy każdy punkt znajdujący się w tablicy p nie jest oddalony od wektora v_h bardziej niż predefiniowane dopuszczalne odchylenie $d_{\max}$.

$$\forall p(n)\left\{\begin{array}{l}\text{dla}: \left|\dfrac{y_0-y_h}{x_0-x_h}\right|<1;\ \left|\dfrac{y_0-y_h}{x_0-x_h}*(p_x(n)-p_x(0))+p_y(0)-p_y(n)\right|<d_{\max}\\ \text{dla}: \left|\dfrac{y_0-y_h}{x_0-x_h}\right|\geq 1;\ \left|\dfrac{x_0-x_h}{y_0-y_h}*(p_y(n)-p_y(0))+p_x(0)-p_x(n)\right|<d_{\max}\end{array}\right\} \tag{4}$$

gdzie:

x_0, y_0 – współrzędne x i y punktu początkowego,
x_n, y_n – współrzędne n-tego punktu w tabeli,
$p(n)$ – n-ty punkt z tabeli p,
$p_x(n)$ – współrzędna x n-tego punktu z tabeli p,
$p_y(n)$ – współrzędna y n-tego punktu z tabeli p.

Jeśli dla danego wektora v_h wszystkie dotychczasowe punkty tablicy p spełniają określone wyżej kryterium, wektor v_h jest przyjmowany za prawidłowy i służy za podstawę do dalszej rozbudowy o kolejne punkty. Jednocześnie współrzędne punktu x_h, y_h są dodawane do tablicy p. Procedura wyszukiwania kolejnego sąsiedniego zapalonego punktu jest następnie powtarzana rekurencyjnie. Po osiągnięciu stanu, gdzie nie jest możliwe dodanie kolejnego punktu do konstruowanego wektora, jest on zwracany i następuje poszukiwanie kolejnego wektora w innym miejscu obrazu.

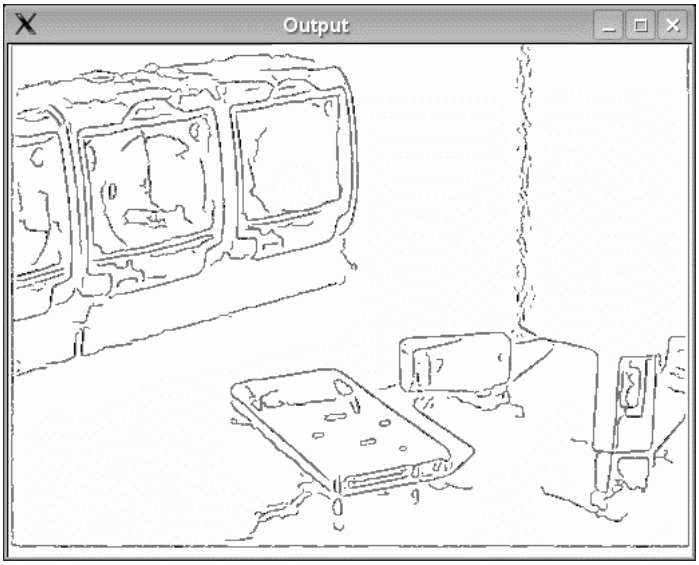
Dzięki temu rozwiązaniu, algorytm zapewnia elastyczność poszukiwania odcinków dopuszczając pewne odchylenia punktów od idealnej prostej – co jest niezbędne choćby ze względu na dyskretny charakter położenia punktów, ale również ogranicza poszukiwania do punktów leżących w określonym obszarze marginesu wzdłuż budowanego wektora.

Dobór współczynnika $d_{\max}$ ma istotne znaczenia dla efektywności działania algorytmu. Współczynnik mały ($d_{\max} < 0{,}5$) spowoduje szybkie działanie algorytmu, jednocześnie generując w rezultacie dużą liczbę krótkich wektorów, o długości kilku punktów. Zwiększenie współczynnika d_{max} do ok. 0,8–0,9 powoduje zwiększenie długości wyszukiwanych wektorów jednak przy zauważalnym wzroście czasu przetwarzania. Dla $d_{\max} > 1$ algorytm może przestać funkcjonować poprawnie, ze względu na fakt wyszukiwania wszystkich możliwych kombinacji „tras", czyli permutacji tabeli p tworzącej kolejne wektory v_h.

W implementacji doświadczalnej najlepsze wyniki dał model kombinowany, w którym początkowo przyjmowane jest $d_{\max} = 0{,}5$, a w trakcie budowy wektora, gdy jego długość osiągnie 4 punkty $d_{\max}$, jest zwiększany do wartości 0,85.

Jak było to opisane wcześniej, procedura budowy wektora kontynuuje poszukiwania nowych punktów możliwych do włączenie do wektora do czasu, aż nie uda się znaleźć

nowych. W takiej sytuacji wektor jest zwracany. Procedura szuka zaś w ramach obrazu kolejnego punktu, który mógłby służyć za początek wektora. Istotnym elementem tego procesu jest sposób wyboru punktów startowych oraz zapewnienie, aby nie były one wielokrotnie używane. W tym celu konstruowana jest kopia oryginalnego obrazu (ramki). Na kopii tej szukane są punkty startowe – wybierany jest pierwszy zapalony punkt. Jednak po znalezieniu wektora wszystkie punkty, które go tworzyły (zapisane w tablicy p), są z tej kopii oryginalnego obrazu usuwane. Dzięki temu punkty, które już stanowią część utworzonych wektorów, nie będą wykorzystywane jako startowe dla poszukiwania nowych. Warto zwrócić uwagę, że rozwiązanie to nie uniemożliwia wykorzystania tych samych punktów przez różne wektory – co może mieć miejsce np. w sytuacji, gdy linie przecinają się. Ponieważ szukanie punktów w ramach rozbudowy wektora v_h odbywa się na oryginalnej ramce obrazu, wszystkie punkty są tam obecne. Usuwając jednak z puli punktów startowych (czyli tych, od których zaczyna się tworzenie wektorów) punkty, które już zostały wykorzystane przy budowie poprzednich wektorów, eliminujemy redundancję w wynikowej grupie wektorów. Rezultat działania opisanego algorytmu został zaprezentowany na rysunku 3.

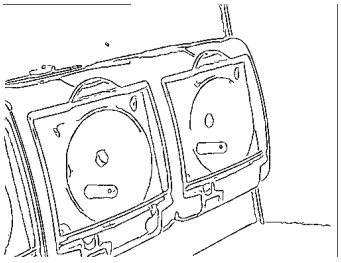
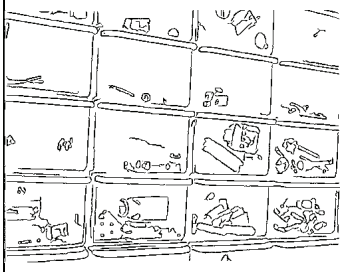


Rys. 3. Rezultat działania algorytmu wektoryzacji

3. Podsumowanie

Przedstawiony algorytm dobrze radził sobie z przetwarzaniem różnego rodzaju rzeczywistych scen. Wyniki działania dla przykładowych obrazów zostały zamieszczone w tablicy 1.

Tablica 1
Wyniki działania algorytmu wektoryzacji na rzeczywistych obrazach

Przykładowa ramka obrazu z kamery	Obraz po wektoryzacji	Rozkład liczby znalezionych wektorów dla wszystkich długości	
		1: 58 2: 948 3: 1179 4: 640 5: 385 6: 240 7: 77 8: 32 9: 36	10: 50 11: 44 12: 36 13: 52 14: 14 15: 0 16: 0 17: 34 18: 0
		1: 79 2: 1500 3: 1098 4: 624 5: 240 6: 120 7: 21 8: 8 9: 18	10: 0 11: 0 12: 0 13: 0 14: 0 15: 0 16: 0 17: 0 18: 0
		1: 126 2: 1510 3: 1209 4: 1076 5: 615 6: 210 7: 63 8: 8 9: 0	10: 10 11: 1 12: 11 13: 0 14: 14 15: 0 16: 0 17: 0 18: 0

Jak widać, rezultatem działania było na ogół 3000÷5000 wektorów na ramkę, niezależnie od analizowanej sceny. Znalezione wektory miały w przeważającej liczbie długość od 2 do 6 pikseli – co wynika z charakteru stosowanego algorytmu i założonego ograniczenia $d_{\max}$. Dłuższe wektory są znajdowane tylko w przypadku, gdy są zorienotwane dokładnie pionowo lub poziomo względem ramki obrazu. Warunkiem skutecznego działania algorytmu było dostarczenie dobrego jakościowo sygnału wejściowego z kamery (odpowiednie oświetlenie, niski poziom szumu). Biorąc pod uwagę wielkość ramki obrazu (512×384), uzyskane kilka tysięcy wektorów oznacza prawie 50-krotną redukcję informacji, co oczy-

wiście ułatwia dalszą analizę obrazu. Wydajność algorytmu również była zadowalająca – większość czasu przetwarzania zajmowała detekcja krawędzi przez algorytm Canny. O ile samo wykorzystanie tego algorytmu pozwalało osiągnąć wynik na poziomie 6 ramek na sekundę, to wektoryzacja z zastosowaniem przedstawionego algorytmu redukowała go do przeciętnie 4÷5 ramek na sekundę.

Wadą algorytmu jest niewątpliwie ograniczenie długości wyszukiwanych odcinków. W dalszym etapie prac planowane jest jednak poprawienie tych rezultatów przez odpowiednie łączenie znalezionych wektorów. Łączenie to będzie szczególnie efektywne jeśli uda się poprawnie zinterpretować niewielkie (np. jedno pikselowe) nieciągłości.

Literatura

[1] Zachara M.: *Real time object tracking and recognition*. MSc Thesis, University of Bristol, England, 1998
[2] Reinhardt J.: *Histogram equalization*. http://www.clarkson.edu/class/image_process/qa1/Histogram Equalization.htm
[3] Heath M. *et al*.: *Comparision of edge detectors*. http://marathon.csee.usf.edu/edge/edge_detection.html
[4] Parker J.R.: *Algorithms for Image Processing and Computer Vision*. Wiley Computer Publishing, USA, 1997, 23–28
[5] Klette R., Zamperoni P.: *Handbook of Image Processing Operators*. Wiley & Sons Ltd., Chichester, England, 1996, 332–340
[6] Gribov A., Bodansky E.: *Vectorization with the Vornoi L-diagram*. Seventh International Conference on Document Analysis and Recognition – ICDAR2003
[7] Song J., Su F., Tai C., Cai S.: *An object-oriented progressive-simplification based vectorization system for engineering drawings: model, algorithm and performance.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(8), August 2002, 1048–1060
[8] Wenyin L., Dov Dori D.: *Extracting Visual Information from Line Drawings.* http://www.cs.utah.edu/~tch/notes/wang/Raster.pdf
[9] Battiato S., Puglisi G., Impoco G.: *Vectorialisation of Raster Colour Images.* http://svg.dmi.unict.it/iplab/administrator/users/pubblicazioni/conference/GdC_06.pdf
[10] Delalandare M., Saidali Y., Ogier J., Trupin E.: *Vectorisation System Based on Strategic Knowledge and XML Representation.* Workshop on Graphics Recognition (GREC), 2003, 250–261
[11] Tombre K., Ah-Soon C., Dosch P., Masini G, Tabbone S.: *Stable and robust vectorization: How to make the right choices*. Lecture Notes in Computer Science, 1941, 2000, 3–17