Marek KRAFT, Michał FULARZ
POLITECHNIKA POZNAŃSKA, INSTYTUT AUTOMATYKI I INŻYNIERII INFORMATYCZNEJ

Porównanie sprzętowych implementacji dwóch popularnych detektorów cech punktowych

Mgr inż. Marek KRAFT

Ukończył studia na kierunku Automatyka i Robotyka na Wydziale Elektrycznym Politechniki Poznańskiej w roku 2005. W tym samym roku rozpoczął pracę na stanowisku asystenta w Instytucie Automatyki i Inżynierii Informatycznej tejże uczelni. Jego główne zainteresowania naukowe obejmują projektowanie i dedykowanych akceleratorów sprzętowych dla zastosowań w systemach wizyjnych, a także zagadnienia z zakresu przetwarzania obrazów, robotyki mobilnej, oraz projektowania systemów wbudowanych.

e-mail: marek.kraft@put.poznan.pl

Michał FULARZ

Jest studentem kierunku Automatyka i Robotyka oraz Informatyka na Wydziale Elektrycznym Politechniki Poznańskiej. Obecnie czynnie działający członek koła naukowego CybAiR. Zainteresowania naukowe autora obejmują cyfrowe przetwarzanie obrazów, systemy mikroprocesorowe, projektowanie i budowę robotów mobilnych, a także układy reprogramowalne.

e-mail: michal.fularz@gmail.com

Streszczenie

W artykule zaprezentowano sprzętowe implementacje dwóch detektorów narożników – detektora Harrisa i detektora FAST – w strukturach FPGA. Prędkość przetwarzania nie ustępuje prędkości uzyskiwanej na współczesnych komputerach osobistych, jednakże zastosowanie niedrogich układów FPGA umożliwia ograniczenie poboru mocy, a także kosztu oraz wymiarów kompletnego systemu. W artykule zawarto opis obu algorytmów, schematy blokowe ich sprzętowych implementacji, a także podsumowanie i porównanie ilości zasobów układu FPGA wykorzystywanych przez obie implementacje. Wykonano również wstępną analizę wyników uzyskanych przez zastosowanie zaimplementowanych detektorów na sekwencji obrazów.

Słowa kluczowe: FPGA, detektor narożników, Harris, FAST.

Comparison of hardware implementations of two popular corner detectors

Abstract

Many contemporary computer and machine vision applications require finding corresponding points in image sequences. For that purpose many point feature detectors have been developed. Most of them detect corners, i.e. points that mark object boundaries, or boundaries of significant object parts as features. In this paper there are presented the implementations of two popular corner detectors – the Harris [2] and FAST [3] corner detector – in FPGA structure. The proposed solutions enable processing of 512x512 pixel, 8-bit grayscale image data with the speed of over 400 frames per second (FAST), and over 350 frames per second (Harris). The processing speeds are the same or even better than those that can be achieved using modern high-performance PCs. FPGA implementations, however, are less power-hungry, relatively inexpensive and more compact, which is critical in many applications. Our implementations are targeted at applications in mobile robotics. The paper contains a short description of the implemented algorithms, block diagrams of the implemented architectures, as well as the summary of the FPGA resources required by both implementations. A preliminary analysis of performance of the implemented algorithms with regards to feature repeatability is also carried out. The results show that the implementation of the FAST algorithm has better performance in terms of speed. Also, the FAST algorithm performs better on image sequences with strong structure – urban, interiors etc. The Harris detector implementation, although in general slower and a little more resource-hungry than the FAST implementation (requires hardware multipliers), demonstrates better performance on poorly structured scene sequences – grass, dirt roads etc. These conclusions are consistent with the results of research carried out before [3, 4].

Keywords: FPGA, corner detector, Harris, FAST.

1. Wstęp

Systemy wizyjne są istotnym elementem systemów sensorycznych współcześnie konstruowanych robotów mobilnych. Informacja z tego typu czujników stanowi dane wejściowe dla algorytmów SLAM (*Simultaneous Localization And Mapping* – jednoczesnej lokalizacji i budowy mapy), rekonstrukcji trójwymiarowej i wielu innych. Istotnym etapem przetwarzania obrazu, będącym składową wielu z tych algorytmów, jest wykrywanie w obrazie cech punktowych (narożników). Wykrywane punkty charakterystyczne w założeniu powinny być dobrze zdefiniowane, co ma umożliwić ich niezawodne wykrycie, także w sekwencji obrazów przedstawiających ta samą scenę w różnych widokach.

Rosnąca wydajność obliczeniowa architektur wbudowanych umożliwia konstruowanie systemów wizji komputerowej o niskim poborze mocy, małych wymiarach i wadze. Cechy te mają szczególne znaczenie w przypadku aplikacji mobilnych, jednak jednoczesne ich spełnienie często jest trudne, lub wręcz niemożliwe. Jednym ze sposobów przezwyciężenia problemów może być zastosowanie do szczególnie wymagających pod względem obliczeniowym etapów przetwarzania (dla przykładu algorytmów przetwarzania obrazu) dedykowanych układów sprzętowych [1].

Dostępność reprogramowalnych układów cyfrowych FPGA i ilość udostępnianych przez nie zasobów umożliwiają implementację systemów składających się z dedykowanych bloków sprzętowych, procesorów ogólnego przeznaczenia, jak również układów pomocniczych (wspomaganie komunikacji, akwizycji itp.).

Niniejszy artykuł zawiera opis implementacji dwóch popularnych detektorów cech punktowych jako dedykowanych bloków sprzętowych w cyfrowym układzie rekonfigurowalnym.

2. Opis implementowanych algorytmów

Do implementacji w cyfrowym układzie rekonfigurowalnym wybrano po przeglądzie literatury dwa algorytmy detekcji narożników: dobrze znany i popularny detektor Harrisa [2], oraz niedawno zaproponowany algorytm FAST (*Features from Accelerated Segment Test* – cechy z przyspieszonego testu segmentu) [3].

Algorytm Harrisa. Procedurę wyznaczania położenia narożników rozpoczyna się od obliczenia wartości gradientów w kierunku poziomym $I_x(x,y)$ oraz pionowym $I_y(x,y)$ dla każdego punktu obrazu. Zalecanym, lecz nie obligatoryjnym punktem algorytmu Harrisa, jest wygładzenie obrazu za pomocą filtru Gaussa o odchyleniu standardowym σ zbliżonym do 1.

Kolejnym etapem jest wyznaczenie macierzy:

$$Q(x,y) = \begin{bmatrix} A & B \\ B & C \end{bmatrix} \tag{1}$$

gdzie:

$$A(x,y) = \sum_{(u,v)\in W(x,y)} w(u,v) I_x(x,y)^2 \tag{2}$$

$$B(x,y) = \sum_{(u,v)\in W(x,y)} w(u,v) I_x(x,y) I_y(x,y) \tag{3}$$

$$C(x,y) = \sum_{(u,v)\in W(x,y)} w(u,v) I_y(x,y)^2 \tag{4}$$

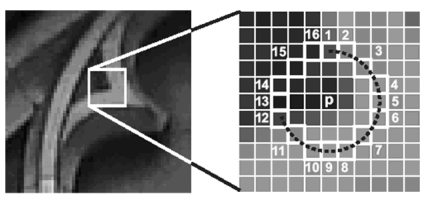
W(x,y) jest oknem umieszczonym w punkcie o współrzędnych *(x,y)*. Za wartości w oknie podstawia się najczęściej aproksymację funkcji Gaussa. Określenie obecności narożnika w danym punkcie obrazu dokonuje się na podstawie wartości własnych macierzy *Q(x,y)*. Aby uprościć obliczenia, wyznacza się aproksymację w postaci funkcji punktującej:

$$H(x,y) = \det\big(Q(x,y)\big) - k \cdot trace\big(Q(x,y)\big)^2 \tag{5}$$

Narożniki umiejscowione są w lokalnych maksimach funkcji punktującej.

Algorytm FAST. Algorytm FAST został opracowany do wykorzystania w zadaniu śledzenia obiektów. Dzięki dobrej powtarzalności cech wykrywanych w kolejnych obrazach tworzących sekwencję [2], algorytm jest przydatny również w zastosowaniach związanych z robotyką mobilną.

Detekcja wykonywana jest w oparciu o strukturę otoczenia badanego piksela. Piksel p o jasności I_p klasyfikowany jest jako narożnik, jeśli na okręgu utworzonym przez 16 pikseli leżących wokół rozpatrywanego punktu istnieje ciągły łuk n punktów o jasności wyższej niż I_p+t (piksele 'jasne'), lub niższej niż $I_p–t$ (piksele 'ciemne'), gdzie t oznacza pewną wartość progową (patrz rysunek 1). Test ten nazywany jest testem segmentu.



Rys. 1. Ilustracja testu segmentu
Fig. 1. Segment test illustration

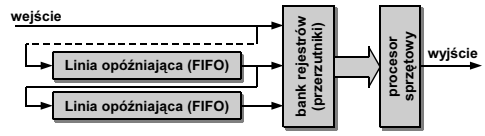
Test segmentu generuje wiele sąsiadujących ze sobą odpowiedzi pozytywnych. W celu uzyskania dobrze zlokalizowanych cech stosuje się tłumienie niemaksymalne. Funkcja punktująca, według wartości której wykonuje się tłumienie niemaksymalne, opisana jest wzorem:

$$V = \max\Big(\sum_{x \in S_{bright}} \big|I_{p\to x} - I_p\big| - t,\ \sum_{x \in S_{dark}} \big|I_p - I_{p\to x}\big| - t\Big) \tag{6}$$

Piksele jaśniejsze niż I_p+t oznaczone są przez S_{bright}, a piksele ciemniejsze niż $I_p–t$ przez S_{dark}. Przez $I_{p\to x}$ oznaczono piksele leżące na okręgu.

3. Implementacja algorytmów

Opisane w rozdziale 2 algorytmy zostały zaimplementowane w postaci dedykowanych procesorów w układzie FPGA XC3S500E-4 firmy Xilinx. Do opisu architektury użyto języka VHDL, a syntezę oraz symulację wykonano przy użyciu narzędzi wchodzących w skład pakietu ISE wymienionej wcześniej firmy. Opisywane algorytmy są algorytmami lokalnymi – przy rozpatrywaniu każdego piksela brane są pod uwagę piksele sąsiadujące, tworzące z wraz z badanymi pikselami kwadratowe okno. Implementowane układy współpracować będą z kamerami zgodnymi ze standardem CameraLink, lub bezpośrednio z sensorem obrazu. W związku z tym przystosowane są one do przyjmowania danych obrazowych podawanych w kolejności linia po linii, bez przeplotu. Aby umożliwić równoległe, jednoczesne przetwarzanie całego okna tak podawanych danych, wykorzystano klasyczną architekturę z buforami opóźniającymi FIFO (patrz rysunek 2) [1]. Bufory zaimplementowano przy pomocy dwuportowych pamięci BlockRAM zawartych w strukturze układu.



Rys. 2. Schemat blokowy buforów opóźniających umożliwiających jednoczesny dostęp do wszystkich pikseli w badanych oknie
Fig. 2. Block diagram of delay buffers enabling simultaneous access to all of the pixels in the window under investigation

Algorytm FAST. W oryginalnym opracowaniu algorytm FAST wykorzystuje algorytmy uczenia maszynowego. Odpowiednie wyuczenie umożliwia odrzucenie punktów-kandydatów nie będących narożnikami we wczesnym etapie przetwarzania [2]. Wynikiem zastosowania algorytmu uczącego jest program w języku C, składający się z dużej ilości zagnieżdżonych instrukcji warunkowych. Takie sformułowanie algorytmu nie jest korzystne do implementacji w dedykowanym układzie cyfrowym. Dla celów implementacji algorytm został nieco przeformułowany, co pozwala lepiej dopasować go do specyfiki platformy implementacji, ale zasada działania algorytmu pozostała bez zmian.

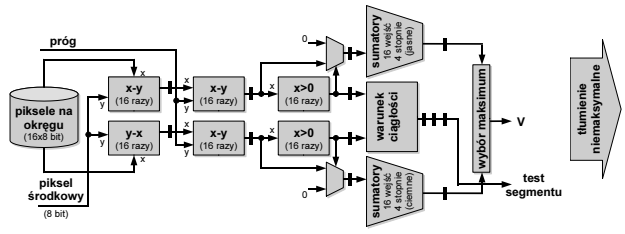
W pierwszym stopniu potoku w sposób równoległy obliczane są różnice – dla pikseli 'jasnych' 16 różnic między jasnością każdego z pikseli na okręgu, a jasnością piksela środkowego, zaś dla pikseli 'ciemnych' 16 różnic między jasnością piksela środkowego, a jasnością każdego z pikseli na okręgu. W drugim stopniu potoku od każdej z tych różnic odejmuje się wartość progu t. Otrzymuje się w ten sposób 2 grupy po 16 wyników – jedną dla pikseli 'jasnych', a jedną dla pikseli 'ciemnych'. Piksele 'jasne' i 'ciemne' spełniają następujące warunki:

$$I_p - I_{p\to x} - t > 0 \tag{7}$$

dla pikseli jasnych, oraz

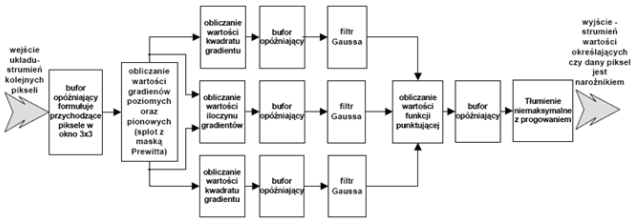
$$I_{p\to x} - I_p - t > 0 \tag{8}$$

dla pikseli ciemnych. W trzecim stopniu potoku za pomocą komparatorów sprawdzane są przez sprawdzenie znaku 32 obliczonych różnic warunki wyrażone wzorami (7) i (8). Jeśli obliczone wartości różnic są większe od zera, podawane są one dalej w celu obliczenia wartości funkcji punktującej oraz na odpowiednie wejście bloku sprawdzającego warunek ciągłości podawana jest logiczna '1'. W przeciwnym razie ujemny wynik zastępowany jest dla liczenia funkcji punktującej przez wartość 0, a na odpowiednie wejście bloku sprawdzającego warunek ciągłości podawane jest logiczne '0'. Dzięki wykonaniu wyżej opisanych kroków w kolejnych stopniach potoku możliwe jest obliczenie wartości funkcji punktującej, oraz wykonanie testu segmentu. W celu wyznaczenia wartości funkcji punktującej w dwóch oddzielnych, 4-stopniowych drzewach sumatorów (po jednym dla pikseli 'jasnych' i ciemnych') dodawane są wartości obliczonych wcześniej, nieujemnych różnic. Równolegle z obliczaniem funkcji punktującej wykonywany jest test segmentu. Do wykonania testu wykorzystywane są 9-wejściowe bramki AND (dla n=9). W celu wyrównania opóźnień konieczne było dodanie na wyjście bloku wykonującego test segmentu kilku przerzutników. Wynik testu segmentu oraz większa z dwóch z obliczonych wartości funkcji punktujących podawane są na blok wykonujący tłumienie niemaksymalne. Schemat blokowy układu przedstawiony jest na rysunku 3 (grubsze linie przecinające ścieżki danych oznaczają rejestry).



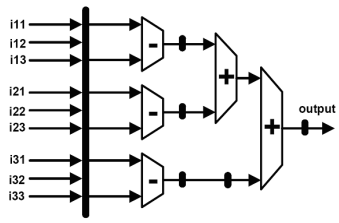
Rys. 3. Schemat blokowy implementacji detektora narożników FAST
Fig. 3. Block diagram of the implementation of FAST corner detector

Algorytm Harrisa. Algorytm został podzielony na etapy, którym odpowiadają poszczególne bloki przedstawione na schemacie blokowym na rysunku 4.



Rys. 4. Schemat blokowy implementacji detektora narożników Harrisa
Fig. 4. Block diagram of the implementation of Harris corner detector

Dane wejściowe zorganizowane zostały w odpowiednie okna rozmiaru 3x3 piksele za pomocą buforów FIFO (patrz rysunek 2). Kolejnym krokiem jest obliczenie pochodnych funkcji obrazowej (gradientów poziomych oraz pionowych) za pomocą odpowiednich masek Prewitta. Operacja ta jest wykonywana potokowo, co obrazuje rysunek 5.



Rys. 5. Potokowy sposób obliczania wartości gradientów
Fig. 5. Pipelined way of calculating gradients values

Następnie wykonano operację mnożenia w celu uzyskania wartości kwadratu poszczególnych pochodnych, a także ich iloczynu. Uzyskane wyniki ponownie zorganizowano do postaci okien rozmiaru 3x3 piksele. Te dane poddano filtracji za pomocą aproksymacji filtru Gaussa, dzięki czemu operacje mnożenia można było zastąpić operacjami przesunięcia bitowego. Ostatnim etapem było obliczenie wartości funkcji punktującej w danym punkcie obrazu. Przyjęto wartość $\kappa = 1/16$. Krok ten umożliwił ponowne uproszczenie obliczeń (zamiast dzielenia występuje operacja przesunięcia bitowego). Wynikiem działania tego algorytmu jest obraz o wymiarach identycznych jak wejściowy z zaznaczonymi punktami charakterystycznymi (krawędziami oraz narożnikami). Tak otrzymany wynik poddano tłumieniu niemaksymalnemu oraz progowaniu w celu uzyskania lokalizacji narożników.

4. Analiza wyników

Maksymalne częstotliwości taktowania podawane przez narzędzie do syntezy (XST firmy Xilinx) to 150 MHz dla implementacji detektora FAST oraz 120 MHz dla detektora Harrisa. Dla obrazu o rozmiarze 512x512 pikseli w 8-bitowej skali szarości daje to ponad 400 klatek na sekundę dla algorytmu FAST i ponad 350 klatek na sekundę dla algorytmu Harrisa. Ilość zasobów użytych do zaimplementowania obu algorytmów podsumowano w tabeli 1.

Tab. 1. Porównanie ilości zasobów układu FPGA użytych w celu zaimplementowania obu algorytmów
Tab. 1. Comparison of the FPGA resources used for implementation of both algorithms

zasób	FAST	Harris
BlockRAM	10	10
FF	1511	1464
LUT	1291	1040
Układy mnożące	0	6

Implementacja algorytmu FAST, choć operuje na oknie o stosunkowo dużym rozmiarze (7x7 pikseli), nie wymaga większej ilości pamięci BlockRAM niż implementacja algorytmu Harrisa, który z kolei potrzebuje większej ilości tych pamięci w pośrednich etapach przetwarzania. Większe jest zapotrzebowanie algorytmu FAST na przerzutniki. Pomimo faktu, że większość operacji jest wykonywana na stosunkowo krótkich wektorach logicznych (na danych 8-bitowych), duża liczba wykonywanych operacji implikuje konieczność przechowywania dużej ilości danych pośrednich w rejestrach. Duża ilość bloków arytmetycznych oraz komparatorów koniecznych do implementacji algorytmu FAST powoduje również użycie większej niż dla algorytmu Harrisa ilości tablic LUT. Warto jednakże podkreślić, że implementacja algorytmu FAST nie wymaga zastosowania układów mnożących. Badania wykazały również, że próby uproszczenia implementacji algorytmu Harrisa poprzez zaokrąglanie wyników pośrednich (odrzucanie z nich pewnej ilości młodszych bitów) szybko prowadzi do znaczącego pogorszenia jakości detekcji.

Zgodnie z podawanymi w pracy [3] wynikami, nasza implementacja algorytmu FAST działa z porównywalną prędkością, a implementacja algorytmu Harrisa do 13 razy szybciej w porównaniu z implementacją na komputerze PC z procesorem Opteron 2,6 GHz i 1GB pamięci operacyjnej.

5. Wnioski

W artykule zaprezentowano sprzętową implementację dwóch popularnych detektorów cech punktowych (narożników). Uzyskane wyniki utwierdzają nas w przekonaniu, że rekonfigurowalne układy cyfrowe są atrakcyjną platformą do implementacji algorytmów przetwarzania obrazu, szczególnie w aplikacjach mobilnych, a więc i w robotyce. Detektor FAST wydaje się więc mieć bardziej korzystne właściwości pod względem implementacyjnym i odznacza się dobrą powtarzalnością cech w środowiskach silnie ustrukturyzowanych (wnętrzach). Klasyczny algorytm Harrisa charakteryzuje się natomiast lepszą dokładnością i powtarzalnością w środowiskach zewnętrznych, na scenach o słabej strukturze. Wyniki przeprowadzonych przez autorów wstępnych badań w tym zakresie pokrywają się z wynikami literaturowymi [3, 4]. Wskazuje to na zasadność implementacji detektora Harrisa, pomimo jego stosunkowo niższej prędkości działania.

Przyszłe prace skoncentrowane będą na zintegrowaniu opisanych rozwiązań w połączeniu z układami prezentowanymi w artykułach [5] i [6] w systemie wizyjnym robota mobilnego na potrzeby wspomagania zadań związanych z nawigacją i samolokalizacją.

6. Literatura

[1] K. Wiatr: Akceleracja obliczeń w systemach wizyjnych. WNT, Warszawa 2003.
[2] C. G. Harris, M. J. Stephens: Combined Corner and Edge Detector, Proc. of the 4th Alvey Vision Conf., str. 147-151, Manchester, 1988.
[3] E. Rosten, T. Drummond: Machine Learning for High-speed Corner Detection, Proc. of the European Conf. on Computer Vision 2006, tom 1, str. 430-443.
[4] K. Konolige, M. Agrawal, J. Sola: Large Scale Visual Odometry for Rough Terrain, International Symposium on Research in Robotics, Hiroshima, Japonia, 2007.
[5] M. Kraft, A. Kasiński: Improved Median Filter Using Conditional Technique and Its Hardware Implementation, Proc. of 15th European Signal Processing Conference, Poznań, Polska, str. 1649-1652, 2007.
[6] M. Kraft, A. Kasiński: Morphological Edge Detection Algorithm and Its Hardware Implementation, Advances in Soft Computing, tom 45, str. 132-139, Springer, 2007.

Artykuł recenzowany