УДК [004.021:004.042]:004.9 DOI: 10.24151/1561-5405-2018-23-5-502-511

Поточная аппаратная реализация алгоритма SURF

Д.А. Гаврилов[1], А.В. Павлов[2]

[1]Московский физико-технический институт (технический университет), г. Москва, Россия
[2]ОАО «Институт точной механики и вычислительной техники имени С.А. Лебедева РАН», г. Москва, Россия

gavrilov.da@mipt.ru

Для выделения и сопровождения объектов на изображении применяется метод, основанный на алгоритмах выделения особых точек. Данный метод часто используется в модулях первичной обработки видеоданных для стабилизации фона. В работе предложена аппаратная реализация алгоритма Speeded Up Robust Features (SURF). Рассмотрен потоковый подход к реализации алгоритма SURF, позволяющий значительно ускорить обработку данных. Показано, что применение такого подхода решает основную проблему использования памяти несколькими модулями одновременно. Представлена общая схема алгоритма SURF, который позволяет выделить особые точки на изображении и создать их описание. Предложено использовать аппаратный ускоритель для поиска и описания особенных точек изображения, так как все указанные задачи требуется решать на высоких скоростях. Разработаны и описаны все модули аппаратной реализации процесса поиска ключевых точек, включая модули интегрального изображения, подсчета вторых производных, расчета гессиана, локального максимума, вычисления дескриптора. Проведено тестирование разработанного алгоритма. Результаты эксперимента показали, что реализация алгоритма SURF на основе потокового подхода позволяет повысить скорость обработки кадра до 100 кадров/с.

Ключевые слова: машинное зрение; ключевые точки; ПЛИС; поток данных

Для цитирования: Гаврилов Д.А., Павлов А.В. Поточная аппаратная реализация алгоритма SURF // Изв. вузов. Электроника. – 2018. – Т. 23. – № 5. – С. 502–511. DOI: 10.24151/1561-5405-2018-23-5-502-511

Streaming Hardware Based Implementation of SURF Algorithm

***D.A. Gavrilov*[1], *A.V. Pavlov*[2]**

[1]*Moscow Institute of Physics and Technology (Technical University), Moscow, Russia*
[2]*Lebedev Institute of Precision Mechanics and Computer Engineering RAS, Moscow, Russia*

gavrilov.da@mipt.ru

Abstract: For allocation and tracking of objects in the image the method based on the algorithms of singular points is applied. This approach is often used in the modules of video data initial processing for background stabilization. The hardware implementation of Speeded Up Robust Features (SURF) algorithm has been proposed. The streaming approach to the implementation of the SURF algorithm, permitting to speed up the data processing, has been realized. The application of such approach solves the main problem of using memory simultaneously by several modules. A general scheme of the SURF algorithm, enabling to allocate singular points on the image and to create their description, has been presented. It has been proposed to use a hardware accelerator for detection and description of singular points of the image, as all problems have to be solved at high speeds. All modules and hardware implementation of the interest point detection, including the integral image modules of calculating the second derivatives, of the Hessian calculation, the local maximum modules and the module for computing the descriptor have been developed and described. The developed algorithm has been tested. The experimental results have been shown that the implementation of the SURF algorithm based on a streaming approach enables to increase the processing speed up to 100 frames per second.

Keywords: machine vision; key points; FPGA; data flow

For citation: Gavrilov D.A., Pavlov A.V. Streaming hardware based implementation of SURF algorithm. *Proc. Univ. Electronics*, 2018, vol. 23, no. 5, pp. 502–511. DOI: 10.24151/1561-5405-2018-23-5-502-511

Введение. В системах наблюдения, обнаружения и навигации основным источником информации является видеопоток. В большинстве случаев все эти системы функционируют в режиме реального времени, что требует высокой скорости обработки исходных видеоданных. Тем не менее задача достижения высоких скоростей обработки данных остается актуальной, поэтому в большинстве случаев задействуется аппаратное ускорение.

Один из методов выделения и сопровождения объектов на изображении основан на выделении особых точек с помощью алгоритма Speeded Up Robust Features (SURF) [1].

В работах [2–4], где предлагается реализация алгоритмов выделения особых точек, основное внимание уделяется распараллеливанию внутренних подмодулей алгоритмов и разбиению кадра на несколько участков. Каждый модуль требует точек изображения, из-за чего в аппаратуре возникает сложность организации доступа к памяти. Другая

проблема – потребление ресурсов, размещенных на ПЛИС. Скорость обработки кадров при такой реализации составляет несколько десятков миллисекунд.

Цель настоящей работы – реализация потокового подхода к алгоритму SURF, позволяющего достичь скорости обработки кадра менее 10 мс. Выбор подхода связан с тем, что камера передает построчный поток пикселей и решается основная проблема использования памяти несколькими модулями одновременно.

Для достижения поставленной цели необходимо:

– выполнить теоретический анализ построения алгоритма SURF;

– разработать и описать основные модули аппаратной реализации процесса поиска ключевых точек с помощью алгоритма SURF;

– провести испытания потоковой реализации алгоритма.

Для аппаратной реализации потокового подхода к алгоритму SURF используется программное обеспечение Xilinx Vivado HLS [5], Vivado [6] и плата virtex7 vx709 development kit. Среда разработки Vivado HLS позволяет синтезировать код C++ в IP-ядро, что необходимо для синтезирования алгоритма SURF и загрузки его в аппаратную платформу.

Алгоритм SURF. Основным результатом алгоритма SURF являются обнаруженные ключевые точки, которые должны иметь следующие свойства: повторяемость (для обеспечения возможности поиска такой же точки на следующем кадре); инвариантность к повороту и масштабу; уникальность.

Обнаружение особых точек алгоритмом SURF основано на вычислении детерминанта матрицы Гессе (гессиана H) [7]. Значение гессиана используется для нахождения локального минимума или максимума яркости изображения. В этих точках значение гессиана достигает экстремума. Расчет производных происходит с помощью свертки пикселей изображения с фильтрами, представленными на рис.1, где белые области соответствуют значению +1, черные – значению –2 (на третьем фильтре – значению –1), серые – нулю.

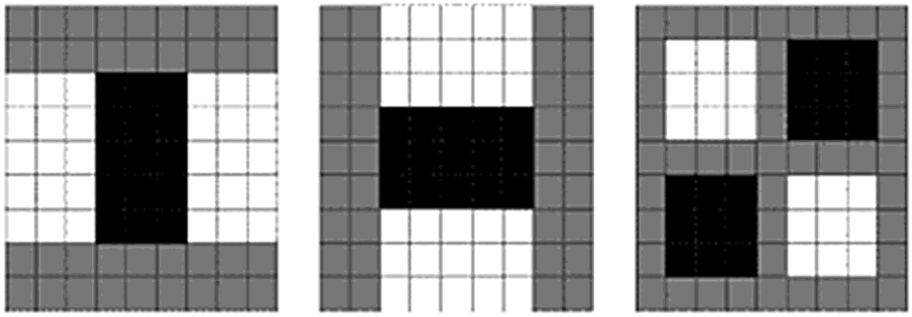

Рис.1. Фильтры, используемые в алгоритме
Fig.1. Filters used in the algorithm

Таким образом, в алгоритме SURF гессиан вычисляется следующим образом:

$$\det(h) = D_{xx}D_{yy} - (0{,}9D_{xy})^2,$$

где D_{xx}, D_{yy}, D_{xy} – свертки по фильтрам (см. рис.1); коэффициент 0,9 корректирует приближенный характер вычислений.

Для эффективного вычисления фильтров используется интегральное представление изображения. Интегральное представление – это матрица, размерность которой совпадает с размерностью исходного изображения, а элементы вычисляются по формуле

$$II(x,y)=\sum_{i=0,j=0}^{i\le x,j\le y} I(i,j),$$

где II (x, y) – яркость пикселя с координатами (x, y) интегрального изображения; I (i, j) – яркость пикселя с координатами (i, j) исходного изображения.

Гессиан не инвариантен относительно масштаба, поэтому требуется перебор нескольких масштабов. Все множество масштабов разбивается на октавы, каждая октава включает в себя четыре характерных масштаба. Первые два масштаба i-й октавы являются 2-м и 4-м масштабом $(i – 1)$-й октавы, где $i > 1$.

В первой октаве гессиан вычисляется для каждого второго пикселя изображения, во второй – для каждого четвертого, в третьей – для каждого восьмого и т.д. Такой подход позволяет сократить вычисления и плотнее покрыть требуемые масштабы. Минимальный размер фильтра 9, шаг между размерами фильтров в i-й октаве $6\cdot2^{(i-1)}$. Масштаб (пиксельный радиус) особой точки s вычисляется как s = размер фильтра·1/6. Точка считается локальным максимумом, если ее гессиан больше, чем у любого соседа в его масштабе, а также больше любого из соседей масштабом меньше и масштабом больше (всего 26 соседей).

Дескриптор особой точки представляет собой массив из 64 чисел. Для вычисления дескриптора вокруг особой точки формируется прямоугольная область, имеющая размер 20s. Далее область разбивается на 16 более мелких квадрантов. В каждом квадранте задается регулярная сетка 5×5, затем для каждой точки сетки ищется градиент с помощью фильтра Хаара [8]. Размер фильтра Хаара берется равным 2s. После нахождения 25 точечных градиентов квадранта вычисляются четыре величины $\sum dX, \sum|dX|, \sum dY, \sum|dY|$, которые являются компонентами дескриптора. Четыре компонента на каждый квадрант и 16 квадрантов дают 64 компонента дескриптора для всей области особой точки. При занесении в массив значения дескрипторов взвешиваются на гауссиан с центром в особой точке и с сигмой 3,3s. Данное преобразование необходимо для увеличения устойчивости дескриптора к шумам в удаленных от особой точки областях.

Модульная структура и описание реализации. Алгоритм SURF можно разбить на следующие модули: интегрального изображения, подсчета вторых производных, расчета гессиана, локального максимума, вычисления дескриптора. Для решения задачи разбиения на модули, организации структуры проекта и расчета внутренних констант все угловые точки фильтров параметризованы сдвигами относительно координаты ключевой точки. На основе указанных параметров вычисляются относительные задержки появления выходных структур модулей, количество таких структур, а также рассчитываются размеры внутренних буферов. Для уменьшения используемых ресурсов на ПЛИС и упрощения кода координаты точек вычисляются в каждом модуле лишь для контроля внутренних параметров, но не передаются в следующие модули. Координаты точек впервые передаются за пределы лишь в модуле локального максимума, так как там перераспределяется поток пикселей.

Связь между модулями организована в виде axi-stream интерфейса [9]. Данный интерфейс представляет собой шину данных, провод готовности данных и провод готов-

ности модуля к приему данных. Также в некоторых модулях имеются провода конца кадра и конца строки.

Модуль интегрального изображения. Модуль интегрального изображения в качестве входных данных оперирует потоком пикселей с камеры. Модуль является наиболее простым в реализации. Для потоковой реализации в этом случае необходимы счетчики строк (count_y) и столбцов (count_x), память для хранения предыдущей строки (buf_last_line) и накапливающий сумматор строки (buf_line). Схема модуля представлена на рис.2.

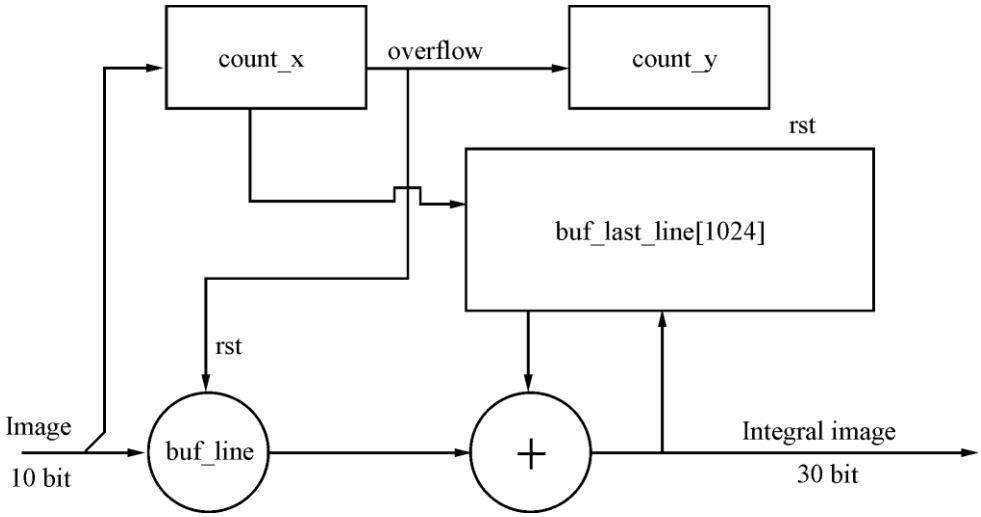

Рис.2. Схема модуля интегрального изображения
Fig.2. Diagram of the integral image module

Для каждого пришедшего пикселя с камеры модуль увеличивает сумматор buf_line на величину пикселя, проверяет count_y. В случае равенства счетчика нулю в буфер последней строки по индексу count_x записывается buf_line. Это же значение подается на выход. Если count_y отличен от нулевого значения, то вычисляется сумма buf_last_line по индексу count_x и buf_line, которая подается на выход и записывается в buf_last_line по индексу count_x. Сумматор текущей строки обнуляется перед подачей новой строки. Так как входной пиксель имеет ширину 10 бит, то размерность выходного пикселя может максимально составлять 30 бит.

Модуль подсчета второй производной. Модуль подсчета второй производной является параметризованным. Для каждой производной фильтра используются собственные параметры: ширина и высота изображения, номер октавы и размер фильтра. По указанным параметрам вычисляются координаты первых и последних требуемых пикселей. Для вычисления фильтров D_{xx} и D_{yy} требуются 8 точек интегрального изображения, для D_{xy} – 16. Стандартный метод фильтрации изображения – метод скользящего окна. В случае большого количества сильно различающихся по размеру фильтров данный метод нерационален, поэтому предлагается использование FIFO-памяти для каждой угловой точки фильтра. Блок-схема модуля расчета производной представлена на рис.3.

Тип точки определяется исходя из номера октавы (учитывается прореживание точек и размер фильтра, вычисляются координаты угловых точек). Размер каждого буфера определяется по размерам фильтра и изображения. Выбранный тип памяти удобен тем, что нет необходимости следить за потоком пикселей, что позволяет последовательно вычислять производные. Как только приходит последний требуемый пиксель, в модуле происходит расчет производной и формируется массив выходных данных.

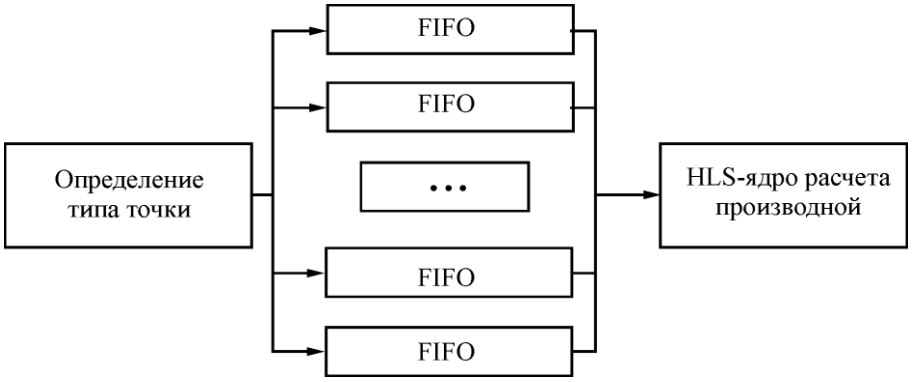

Рис.3. Блок-схема модуля расчета производной
Fig.3. Block diagram of calculation of the derivative module

В модуле подсчета второй производной остается открытым вопрос о ширине каждого фильтра (детального изучения данного вопроса в литературе авторами не обнаружено). Отметим, что эта тема требует дополнительного исследования с целью оптимального подбора рассматриваемого параметра для задачи, решаемой каждым конкретным фильтром. В предлагаемой реализации ширина фильтра составляет 1/3 от его длины.

Модуль расчета гессиана. Архитектура модуля расчета гессиана аналогична архитектуре модуля подсчета второй производной. С учетом того, что вторые производные по x приходят первыми, а по y – последними (это связано с размерами фильтров), требуется организовывать буфер для производных. В связи с тем что в модулях расчета производных поток сохраняется, в модуле расчета гессиана также допустимо использование FIFO-памяти для организации буферизации.

Модуль локального максимума. Для задачи поиска локального максимума рассмотрим фильтры группами по 3, не выходя за границы октавы. Основная сложность в данном модуле при потоковой реализации вычислений состоит в том, что фильтры малых размеров будут готовы значительно раньше, поэтому в данном случае здесь также требуется организация буферизации гессианов. Кроме того, некоторые масштабы вычисляются на i-й октаве, а могут использоваться в i+2. Данная особенность требует прореживания входных данных. На входе модуля локального максимума потоки данных сохраняются, но в процессе вычисления локального максимума поток будет нарушен, поэтому на выходе модуля добавляются координаты ключевого пикселя и его масштаб.

Модуль вычисления дескрипторов. Реализация модуля дескрипторов может быть представлена в виде следующих шагов:

– вычисление левой верхней координаты интересуемой области;
– вычитывание из памяти соответствующих пикселей интегрального изображения;
– вычисление производных;
– нормировка производных;
– компоновка нормированных производных в dX, dY, $|dX|$, $|dY|$;
– приписывание дескриптора к точке и организация потока.

Левая верхняя точка интересуемой области рассчитывается как координата описываемой точки минус 10 размеров этой точки по x и y. К моменту расчета дескриптора точки интересуемая область интегрального изображения уже сохранена в QDR-памяти платы. Для вычисления пары производных требуется вычитать из памяти восемь точек. Схема вычитывания точек из памяти представлена на рис.4.

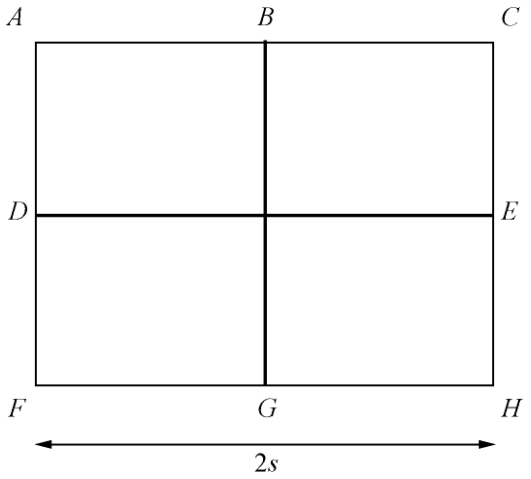

Рис.4. Схема вычитывания точек из памяти (*dX = BCHG – ABGH, dY = DEHF – ACED*)
Fig.4. Diagram of the calculate points from memory (*dX = BCHG – ABGH, dY = DEHF – ACED*)

Расстояние между точками *s* – размер точки, центральная точка (на пересечении отрезков BG и DE) не используется на данном шаге, но потребуется для вычисления следующей пары. Для простоты реализации контроллера памяти скользящее окно вычитывается за три шага с помощью вычитывания по трем точкам с шагом *s*. Таких окон для вычисления дескриптора потребуется 20·20=400, поэтому в сумме будет вычитана 21·21=441 точка. После вычитывания скользящего окна из памяти все точки подаются на вход ядра HLS, в котором вычисляются производные. Таким образом, получается поток производных в прогрессивной развертке по квадрантам дескриптора. Последним шагом является компоновка производных по 25 штук и вычисление *dX*, *dY*, |*dX*|, |*dY*|. На этом же шаге выполняется нормировка компонент дескриптора. Реализация данного шага представляет собой BRAM-память на 100 производных, дерево суммирования и ядро HLS, выполняющее нормировку. Последний подмодуль формирует поток точек, при этом каждая точка разбивается на несколько пакетов, которые подходят для передачи по PCIe.

Тестирование и результаты экспериментов. Многие библиотеки компьютерного зрения содержат реализацию алгоритма SURF. Однако с целью принятия во внимание возможных особенностей реализации и проведения независимых экспериментов необходимо осуществить реализацию данного алгоритма без использования библиотечных компонент. Кроме того, известно, что реализация алгоритмов на ПЛИС является гораздо более сложной и затратной задачей по сравнению с программной реализацией.

В результате исследования реализован класс на C++, позволяющий осуществлять процесс тестирования и оценки алгоритма на различных видеопотоках. Класс исполнен таким образом, что каждая axi-stream шина в ПЛИС представляет собой массив последовательных данных в классе. Кроме того, для каждого вычислительного модуля алгоритма на Verilog создан отдельный метод класса. Такой подход позволяет проводить тестирование Verilog-модулей на различных данных и предоставляет возможность ввести автоматизированное тестирование. Для организации процесса тестирования разработаны специальные модули на Verilog, которые, вычитывая последовательно данные из заранее подготовленного файла, способны генерировать axi поток в различных режимах:

– постоянная загрузка (данные подаются каждый такт). Такой тест проверяет способность реализации воспринимать сжатый поток данных и проверяет пропускную способность каждого модуля;

– случайная загрузка. Тестирование со случайной загрузкой данных позволяет найти ошибки в управлении шинами axi;

– прерывистая загрузка. Тест проверяет ошибки, которые могут возникнуть при случайной остановке потока данных.

В процессе тестирования к выходу исследуемого алгоритма подключается модуль, который может осуществить прием данных в аналогичных режимах, а также выполнить

проверку выходных данных на корректность путем сравнения их с эталонными данными. Такой подход позволяет идентифицировать ошибки не только в реализации модулей, но и в соединении модулей между собой. Заранее сгенерированные файлы эталонных данных получаются с помощью класса на C++.

Поиск точек запускается с приходом первого пикселя и заканчивается после прихода последнего пикселя с задержкой не более 8 мкс. Для вычисления дескрипторов точки должна появиться сама точка, а также определиться требуемая область интегрального изображения. При малой загруженности кадра особенностями вычисления дескрипторов должны закончиться одновременно с поиском точек. В том случае, если точек много, в процессе работы с кадром могут быть обработаны до 5000 точек.

Эксперимент заключается в подготовке видеопотока, содержащего разнообразные цели и фоновые обстановки, преобразовании видеопотока в бинарный вид и подготовке набора rtp-пакетов для передачи видео по Ethernet, а также подготовке модуля для сравнения всех данных, полученных с ПЛИС и вычисленных программно с помощью разработанного класса. Данный эксперимент проверяет точность реализации всех модулей, любая ошибка будет видна в конечном результате. Подготовленный видеопоток передается на ПЛИС по протоколу Ethernet. Обработанные данные через шину PCIe передаются на рабочую станцию под управлением операционной системы реального времени QNX. Все модули алгоритма тщательно протестированы, а эталонные данные сгенерированы с помощью идентичного класса. В результате проведенного эксперимента установлено 100%-ное совпадение полученных с ПЛИС и вычисленных эталонных данных.

Помимо точности реализации в ходе эксперимента проведена проверка скоростных возможностей представленной реализации алгоритма. Алгоритм аппаратно реализован на частоте 250 МГц. Проведено исследование работы алгоритма с использованием данных различной плотности при изменении скорости входных данных. Результаты экспериментальных исследований показали повышение максимальной скорости обработки до 100 кадров/с. Теоретически возможно получение обработки со скоростью не менее 200 кадров/с.

Результаты сравнения авторской реализации алгоритма SURF с реализациями других авторов представлены в таблице.

Результаты сравнения реализаций алгоритма SURF
Results of comparison of implementations of the algorithm SURF

Параметр	Реализация			
	[2]	[3]	[10]	Авторская
Скорость обработки, кадр/с	10	60	75	100
Максимальное изображение, пиксель	1024×1024	–	–	Зависит от «железа»
Целочисленные вычисления	Ограничение точности	Не используются	Не используются (за счет ЦПУ)	Не используются
Частичная реализация на ЦПУ	Да (управляющая логика)	Нет	Да (часть модулей)	Нет
Количество масштабов	8	6	8	10
Вычисление дескриптора	Нет	Да	Да (ЦПУ)	Да

Примечание. ЦПУ – центральное процессорное устройство.

Как видно из данных таблицы, скорость обработки кадров в реализациях [2, 3, 10] не превышает 75 кадров в секунду. Авторская реализация не требует использования целочисленных вычислений и в ней применяются 10-масштабные фильтры.

Заключение. Применение потокового подхода к алгоритму SURF решает основную проблему использования памяти несколькими модулями одновременно. Тестирование разработанного алгоритма и результаты эксперимента показали, что реализация алгоритма SURF на основе потокового подхода позволяет повысить скорость обработки кадра до 100 кадров/с.

Литература

1. ***Bay H., Essa A., Tuytelaars T., Van Gool L.*** Speeded-Up Robust Features (SURF) // Comput. Vis. Image Underst. – 2008. – Vol. 110. – No. 3. – P. 346–359.

2. ***Svab J., Faigl J., Krajník T.*** FPGA based speeded up robust features // IEEE Intern. Conf. on Technologies for Practical Robot Applications. – 2009. – P. 35–41.

3. ***Sledevic T., Serackis A.*** SURF algorithm implementation on FPGA // 13th Biennial Baltic Electronics Conf. – Tallinn, 2012. – P. 291–294.

4. ***Chang L., Sucar L.E.*** FPGA-based detection of SIFT interest keypoints // Mach. Vis. Appl. – 2013. – No. 24. – P. 371–392.

5. Vivado Design Suite User Guide. High-Level Synthesis. UG902 (v2016.2). – 2016. – P. 672. – URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_2/ug902-vivado-high-level-synthesis.pdf (дата обращения: 22.01.2018).

6. Vivado Design Suite User Guide. Getting Started. UG910(v2016.4). – 2016. – P. 24. – URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_4/ug910-vivado-getting-started.pdf (дата обращения: 22.01.2018).

7. ***Хайкин С.*** Нейронные сети: полный курс. – 2 изд. – М.: Изд. дом «Вильямс», 2006. – 1104 с.

8. ***Демьянкович Ю.К., Ходаковский В.А.*** Введение в теорию вейвлетов: курс лекций. – 2007. – С. 49. – URL: http://www.math.spbu.ru/parallel/pdf/dh_theory.pdf (дата обращения: 22.01.2018).

9. Vivado Design Suite. AXI Reference. UG1037 (v4.0). – 2017. – P. 175. – URL: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf (дата обращения: 22.01.2018).

10. An FPGA implementation of the SURF algorithm for the ExoMars programme / ***G. Lentaris, I. Stamoulias, D. Diamantopoulos et al.*** // Workshop on Reconfigurable Computing (WRC). – Germany, Berlin, 2013. – URL: https://www.researchgate.net/publication/255926863_An_FPGA_implementation_of_the_SURF_algorithm_for_the_ExoMars_programme (дата обращения: 04.10.2018).

Поступила 23.01.2018 г.; принята к публикации 19.06.2018 г.

Гаврилов Дмитрий Александрович – кандидат технических наук, доцент кафедры радиоэлектроники и прикладной информатики, руководитель лаборатории цифровых систем специального назначения Московского физико-технического института (технического университета) (Россия, 141701, г. Долгопрудный, Институтский пер., д.9), gavrilov.da@mipt.ru

Павлов Алексей Валерьевич – аспирант, младший научный сотрудник ОАО «Институт точной механики и вычислительной техники имени С.А. Лебедева РАН» (Россия, 119991, г. Москва, Ленинский проспект, 51), pavlov.av@mipt.ru

References

1. Bay H., Essa A., Tuytelaars T., Van Gool L. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst*., 2008, vol. 110, no. 3. pp. 346–359.

2. Svab J., Faigl J., Krajník T. FPGA based speeded up robust features. I*EEE International Conference on Technologies for Practical Robot Applications,* 2009, pp. 35–41.

3. Sledevic T., Serackis A. SURF Algorithm Implementation on FPGA. 1*3th Biennial Baltic Electronics Conference*. Tallinn, 2012, pp. 291–294.

4. Chang L., Sucar L.E. FPGA-based detection of SIFT interest keypoints. *Mach. Vis. Appl.*, 2013, no. 24, pp. 371–392.

5. *Vivado Design Suite User Guide. High-Level Synthesis. UG902 (v2016.2)*. 2016, p. 672. Available at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_2/ug902-vivado-high-level-synthesis.pdf (accessed: 22.01.2018).

6. *Vivado Design Suite User Guide. Getting Started. UG910(v2016.4).* 2016, p. 24. Available at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_4/ug910-vivado-getting-started.pdf (accessed: 22.01.2018).

7. Haykin S. *Neural networks: a comprehensive foundation.* 2nd ed. Moscow, Williams Publishing house Publ., 2006. 1104 p. (In Russian).

8. Demjankovich Y.K., Khodakovsky V.A. *Introduction to the theory of wavelets: a course of lectures.* 2007, P. 49. Available at: http://www.math.spbu.ru/parallel/pdf/dh_theory.pdf (accessed: 22.01.2018). (In Russian).

9. *Vivado Design Suite. AXI Reference. UG1037 (v4.0).* 2017, p. 175. Available at: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf (accessed: 22.01.2018).

10. Lentaris G., Stamoulias I., Diamantopoulos D., Siozios K., Soudris D. *An FPGA implementation of the SURF algorithm for the ExoMars programme*. Workshop on Reconfigurable Computing (WRC). Germany, Berlin, 2013. Available at: https://www.researchgate.net/publication/255926863_An_FPGA_implementation_of_the_SURF_algorithm_for_the_ExoMars_programme (accessed: 04.10.2018).

Submitted 23.01.2018; Accepted 19.06.2018.

Information about the authors:

Dmitry A. Gavrilov – Cand. Sci. (Eng.), Assoc. Prof. of the Radioelectronics and Applied Informatics Department, Head of the Digital Systems of Special Purpose Laboratory, Moscow Institute of Physics and Technology (Russia, 141701, Dolgoprudny, Institutsky pereulok, 9), gavrilov.da@mipt.ru

Alexey V. Pavlov – PhD student, Junior Researcher, Lebedev Institute of Precision Mechanics and Computer Engineering RAS (Russia, 119991, Moscow, Leninsky prospect, 51), pavlov.av@mipt.ru