УДК 004.932.2

МНОГОПОТОЧНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ВИЗУАЛЬНОЙ ЛОКАЛИЗАЦИИ БПЛА НА ОСНОВЕ ИЗВЕСТНОЙ 3D МОДЕЛИ ОКРУЖЕНИЯ И ТЕХНОЛОГИИ CUDA

А.К. Буйвал (*alexbuyval@gmail.com*)
М.А. Гавриленков (gavrilenkov@umlab.ru)
Брянский государственный технический университет, Брянск

Аннотация. В статье приводится описание реализации с использованием технологии CUDA алгоритма визуальной локализации БПЛА внутри помещения на основе сопоставления граней, полученных из изображения с видеокамеры и из смоделированного изображения, полученного на основе известной 3D модели окружающей среды (помещения). В конце статьи дается сравнение производительности последовательной и многопоточной реализации на основе моделирования в среде Gazebo.

Ключевые слова: визуальная локализация, фильтр частиц, ROS, Gazebo

Введение

В настоящий момент всё еще остается открытым вопрос о быстрой и точной локализации мобильного робота и в частности БПЛА в пространстве. Локализация мобильного робота в пространстве является нетривиальным процессом, требует детального рассмотрения во многих аспектах [Pink, 2009]. В связи с тем, что локализация в закрытом пространстве не может осуществляться по таким параметрам, как координаты GPS, на практике применяются либо дорогие лазерные дальномеры, либо более продвинутый подход – использование информации с бортовой камеры [Saurer, 2010] и фильтра частиц. Данный подход основывается на сравнении изображения, полученного с камеры, со смоделированным изображением, полученным на основе 3D модели помещения. Данный подход во многом превосходит альтернативные решения, но, к сожалению, не лишен недостатков. Весомым недостатком становится избыточность получаемой с камеры информации.

Избыточность информации ведет к большей затрате ресурсов на очищение изображения, нахождения ключевых точек и граней и т.д. К

сожалению, на сегодняшний момент не существует универсального решения этой проблемы. Дополнительным негативным фактором является ограниченность вычислительных ресурсов на борту БПЛА. Одним из выходов из сложившейся ситуации является использование альтернативных вычислительных мощностей на борту БПЛА.

В данной статье предлагается рассмотреть идею частичного использования GPU в системе локализации БПЛА для наиболее вычислительно сложных операций. В статье представлено описание реализации вычислительно сложных частей алгоритма, как на CPU, так и на GPU, а также приводится сравнение производительности при использовании этих двух подходов. Повышение производительности в работе алгоритма локализации также значительно повышает точность локализации, т.к. появляется возможность использования большего количества частиц.

1 Описание базового алгоритма локализации

Базовым элементов разработанной системы визуальной локализации является использование граней в изображении как характерных визуальных признаков. Ключевым моментом в таком подходе является сравнение и численная оценка схожести 2-х изображений: полученного с камеры и изображения, смоделированного на основе модели помещения [Nuske, 2009]. Предполагается, что алгоритм строит ряд гипотез о положении робота в пространстве, затем на основе этих гипотез он моделирует изображения, каждое из которых соответствует изображению, которое должен получить робот, если он находиться в этой предполагаемой точке. Таким образом, на каждой итерации алгоритма мы сравниваем одно изображение с камеры со множеством смоделированных изображений и определяем степень их схожести [Буйвал, 2015a].

1.1 Использование фильтра частиц для локализации робота

Для обработки данных о схожести изображений, а также данных с других доступных датчиков используется алгоритм локализации, основанный на множестве частиц (гипотез о местоположении робота) – фильтр частиц (particle filter). Данный метод широко известен и хорошо зарекомендовал себя в подобных задачах. Преимуществом этого метода является то, что он позволяет использовать множество гипотез, а также нелинейные модели как самой системы, так и нелинейные модели показаний датчиков [Буйвал, 2015a].

1.2. Измерительная модель фильтра частиц

Для того, чтобы оценить вероятность каждой гипотезы (частицы) необходимо оценить близость границ, полученных из изображения с камеры, к границам, полученным из смоделированного изображения, соответствующего гипотезе.

В разработанной системе используется метод ближайших граней. Данный метод не имеет такого недостатка, как учет только совпадающих на гранях пикселей. Т.о. если грани одного и того же конструктивного элемента на изображении с камеры и на смоделированном изображении не совпадают, но проходят близко друг к другу, то это все равно будет учитываться в конечной вероятности, т.к. в данном методе предлагается оценивать близость смоделированной грани к грани на изображении с камеры с помощью набора нормалей, построенных от смоделированной грани. Впервые данный метод был продемонстрирован [Nuske, 2009] и хорошо зарекомендовал себя для колесного робота.

В рассматриваемом методе выполняются последовательно следующие этапы:

1. Выделение границ на изображении, полученном с камеры и нахождение прямых среди них.
2. Рендеринг изображения согласно вектору состояния частицы, выделение границ на полученном изображении и нахождение прямых.
3. На каждой прямой формируется набор точек с постоянным шагом. Из каждой точки строится нормаль некой предельной длины. Если эта нормаль пересекается с какой-либо прямой из полученных на основе изображения с камеры, то длина такой нормали учитывается в общем весе рассматриваемой прямой по следующей формуле:

$$g(d) = exp\left(-\frac{d^2}{2\sigma^2}\right), \qquad (1)$$

где d – длина нормали, σ – параметр определяющий вес нормали в зависимости от длины нормали (позволяет либо усиливать влияние длинных нормалей, либо уменьшать).

Для каждой прямой вычисляется общий вес путем суммирования всех весов нормалей по следующей формуле:

$$l = \frac{\sum_{i=0}^{S} g(d_i)}{S}, \qquad (2)$$

где S – общее количество прямых в смоделированном изображении.

Вычисляется итоговая вероятность гипотезы на основе объединения весов каждой линии по следующей формуле:

$$W = \propto \cdot exp\left(k\frac{\sum_{n=0}^{m} l_n}{m}\right), \qquad (3)$$

где m – количество прямых, $\propto, k$ – параметры учета веса прямых в

итоговом результате [Буйвал, 2015b].

2 Реализация фильтра частиц на CPU

Основным и, пожалуй, самым значимым моментом в работе фильтра частиц является обновление его весов. Как описано выше на первых этапах алгоритма необходимо выделить грани на изображении и найти прямые. Для этого используются стандартные функции пакета OpenCV: Canny и HoughLinesP соответственно. Количество частиц в системе задается статически и никак не корректируется.

На рис. 1. представлена схема вычисления весов частиц. Из схемы видно, что основной проблемой при классическом подходе к вычислению весов, является итеративность этих вычислений, при этом, само вычисление веса для частицы («Вычисление веса частицы») является финальным шагом внешнего цикла, и зависит от параметров, полученных во внутренних циклах. Сложность алгоритма в данном случае сравнима с N*K*M, где N-число частиц, K-число прямых на смоделированном изображении, M-число нормалей на прямой. Причем на каждой итерации внутреннего цикла нам необходимо произвести вычисления согласно формуле 1. Существует возможность распараллелить данный алгоритм в рамках использования CPU, но выигрыш в данном случае будет прямо пропорционален количеству задействованных процессоров. Стоит отметить, что мобильные роботы, зачастую, не имеют на своем борту процессоров с большим числом ядер, поэтому параллелизм в рамках CPU малоэффективен.

3 Реализация фильтра частиц на GPU

Проанализировав всё вышеописанное, напрашивается вывод, что для быстроты вычисления весов модели, лучшим решением является использование параллельных вычислений с большим числом ядер. Наиболее перспективной и мощной технологией таких вычислений автор видит технологии CUDA, разработанную компанией NVIDIA.

Работая с CUDA, программист получает параллельную вычислительную систему, работающую по принципу SIMT (Single-Instruction, Multiple-Thread), когда одна команда параллельно выполняется множеством независимых потоков (threads). Совокупность всех этих потоков, запущенных в рамках одной задачи (рис.2), носит название grid.[Habrahabr, 2012]

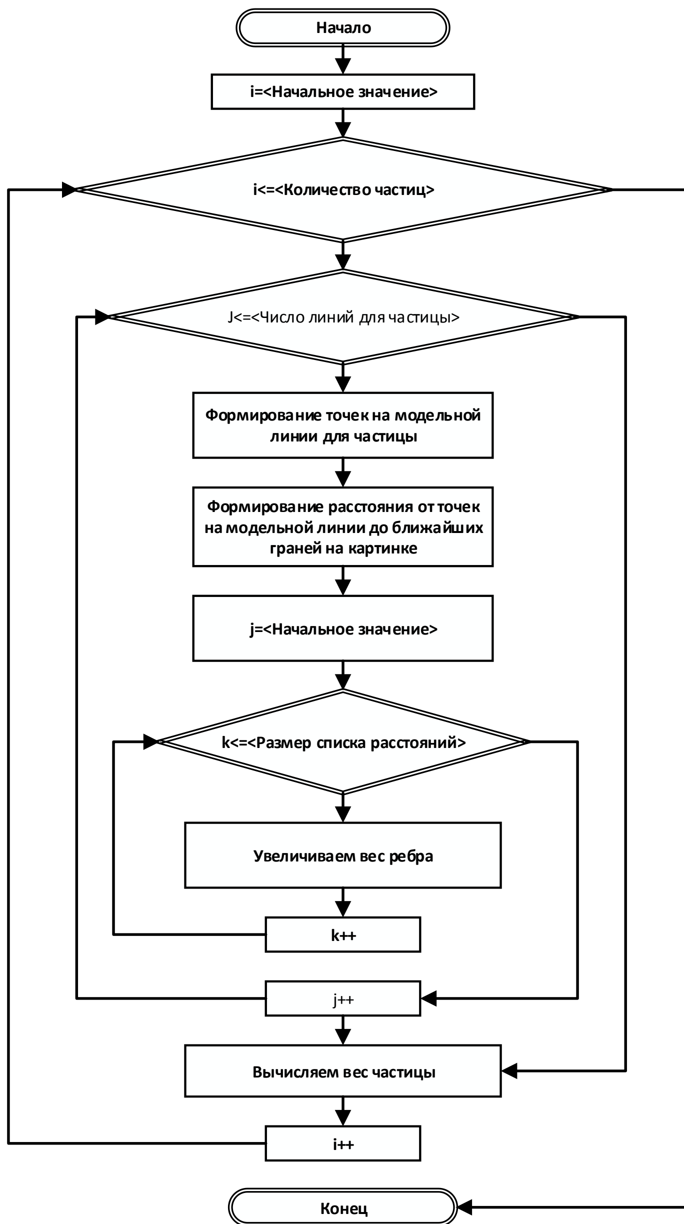

Рис. 1. Схема вычисления весов частиц с использованием CPU

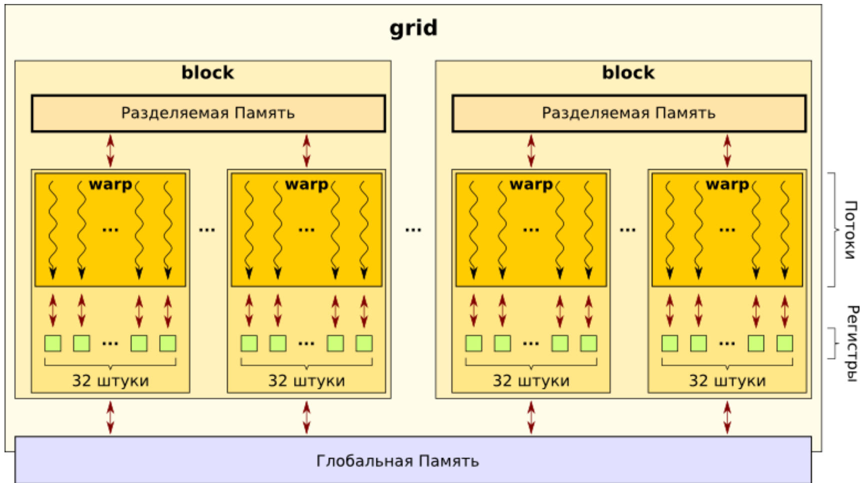

Рис 2. Иерархия потоков CUDA

В видеокарте, поддерживающий технологию CUDA, параллельное выполнение grid обеспечивается наличием в самой видеокарте большого количества скалярных процессоров (scalar processors). Данные процессоры и выполняют потоки (threads). Каждый скалярный процессор входит в более крупное образование (warp), тем самым являясь частью потокового мультипроцессора (streaming multiprocessor). Варпы также входят в другие крупные образования – блоки (blocks, рис. 2).

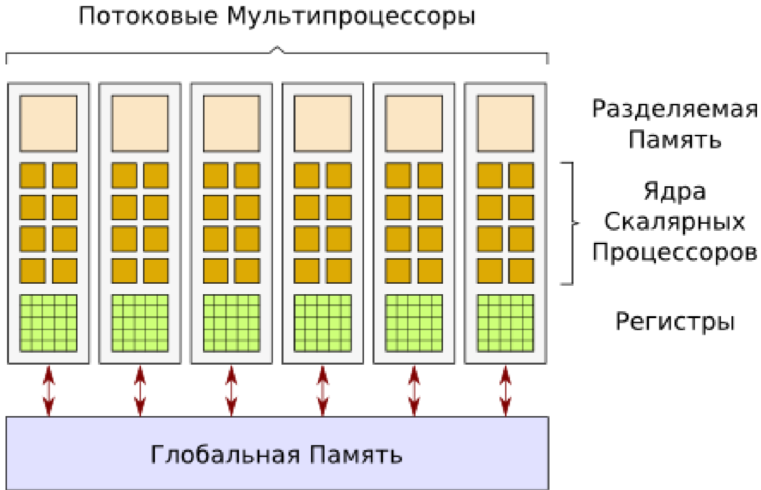

Рис. 3. Схема видеокарты с технологией CUDA

Другой, не менее важной особенностью, является организация памяти в CUDA и доступа потоков к её различным частям. Наивысшей степенью общедоступности для потоков обладает глобальная память (global memory), Расположение вне процессора делает этот тип памяти наиболее медленным, по сравнению с другими, предоставляемыми для вычислений на видеокарте.

Следующим типом в иерархии памяти идет разделяемая память (shared memory). Данный вид памяти расположен в каждом отдельном потоковом мультипроцессоре. Данная память доступна только тем потокам, которые выполняются на ядрах этого мультипроцессора. Так как к параллельному исполнению на одном мультипроцессоре может быть отведено более одного блока, то весь доступный в мультипроцессоре объем разделяемой памяти распределяется между этими блоками поровну. Необходимо упомянуть, что разделяемая память физически расположена где-то очень близко к ядрам мультипроцессора, поэтому обладает высокой скоростью доступа, сравнимой с быстродействием регистровой (registers) – основным видом памяти. Именно регистры могут служить операндами элементарных машинных команд, и являются более быстрой памятью [Habrahabr, 2012].

Несомненно, что при работе с несколькими потоками, следует использовать механизмы для синхронизации. CUDA предусматривает несколько команд для этих целей [Habrahabr, 2012]:

- **__syncthreads()** – самый верный способ. Эта функция заставит каждый поток ждать, пока все остальные потоки этого блока

достигнут этой точки или все операции по доступу к разделяемой и глобальной памяти, совершенные потоками этого блока, завершатся и станут видны потокам этого блока.

- **__threadfence_block()** будет заставлять ждать вызвавший её поток, пока все совершенные операции доступа к разделяемой и глобальной памяти завершатся и станут видны потокам этого блока.
- **__threadfence()** будет заставлять ждать вызвавший её поток, пока все совершенные операции доступа к разделяемой памяти станут видны потокам этого блока, а операции с глобальной памятью всем потокам на «устройстве». Под «устройством» понимается графический адаптер.
- **__threadfence_system()** подобна __threadfence(), но включает синхронизацию с потоками на CPU.

Исходя из всего вышеизложенного, была принята следующая структура алгоритма изменения весов фильтра частиц.

Как видно на схеме (рис. 4), вычисления с применением CUDA распараллеливают процесс работы вычисления весов частиц, что должно приводить к многократному ускорению этого процесса.

3.1. Дополнительные улучшения, основанные на использовании GPU

Как было описано выше, в предложенной подсистеме используются стандартные функции OpenCV [OpenCV, 2016a]. Основными используемыми функциями являются **Canny** [OpenCV, 2016b] и **HoughLinesP** [OpenCV, 2016c]. Данные методы выполняют довольно тяжеловесные операции по извлечению информации из изображения. Логичным ходом стал перенос вычислений из области вычисления процессора в область вычисления видеокарты [OpenCV, 2016e]. В статье [Luo, 2008] показана эффективность применения GPU для оператора Canny. В OpenCV существуют специальные альтернативные методы, которые способны выполняться на GPU: **cv::gpu::Canny** и **cv::gpu::HoughLinesP** [OpenCV, 2016d].

Процесс вызова GPU функций в таком случае выглядит следующим образом:

1. копирование исходные данных в память GPU;
2. вызов необходимых операторов OpenCV;
3. копирование полученных данных в память CPU.

4 Сравнительный анализ подходов

В ходе симуляционных экспериментов в среде ROS/Gazebo, был

проведен сравнительный анализ двух реализаций описанного алгоритма (только CPU и CPU+GPU), результаты которого приведены в таблице 1.

По результатам сравнения можно сделать вывод, что использование связки CPU+GPU дает огромный выигрыш в производительности. В среднем, в ходе эксперимента скорость выполнения алгоритма пересчета весов частиц увеличилась в среднем в 17,26 раз. Исходя из данных результатов можно сделать вывод, что данный подход является крайне эффективным.

Использование GPU для фильтра частиц является верным шагом к построению подсистемы локализации, главным достоинством которой является быстродействие.

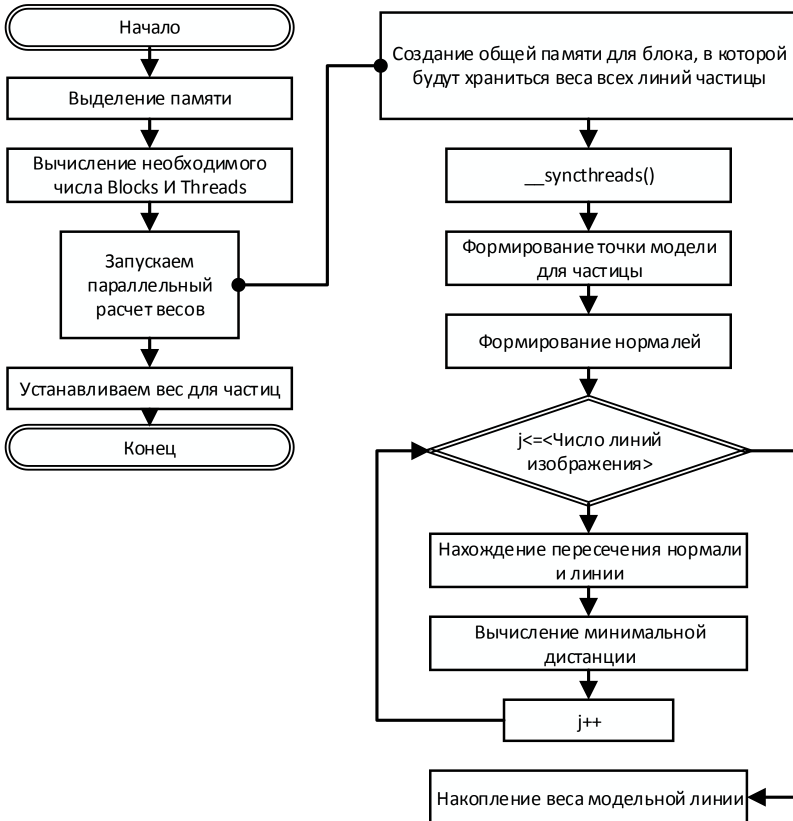

Рис. 4. Схема вычисления весов частиц с использованием GPU

Таблица 1.

Номер опыта	**1**	**2**	**3**	**4**	**5**	**6**	**7**
CPU time, мс	123,60	102,83	150,12	131,84	146,74	113,83	163,70
GPU time, мс	7,39	6,54	8,44	7,45	8,46	7,03	8,28
CPU/GPU, раз	16,7	15,7	17,7	17,6	17,3	16,1	19,7

5 Направления дальнейших исследований

В дальнейшей работе планируется испытать усовершенствованную в результате исследований систему на одноплатном микрокомпьютер **NDIVIA Jetson TX1**. Данный микрокомпьютер имеет встроенный GPU микропроцессор, что позволит получить существенный выигрыш во времени локализации при использовании разработанного алгоритма.

6 Заключение

В данной работе была модернизирована подсистема локализации БПЛА, а также было продемонстрировано преимущество использования GPU процессоров на борту БПЛА, были проанализированы стандартные функции библиотеки OpenCV для работы с CPU и GPU. В результате удалось добиться многократного ускорения работы алгоритмов локализации, что очень важно в условиях ограниченности ресурсов на борту БПЛА.

Список литературы

[Nuske, 2009] Nuske, S., Roberts, J., & Wyeth, G. Robust outdoor visual localization using a three-dimensional-edge map // Journal of Field Robotics. 2009 №26(9), 728-756.

[Буйвал, 2015a] Буйвал А.К. Локализация беспилотного летательного аппарата внутри помещений на основе визуальных геометрических маркеров и известной 3D модели окружающей среды. //Ж-л Робототехника и техническая кибернетика. 2015 №3(8), 71-75 стр.

[Буйвал, 2015b] Буйвал А.К., Гавриленков М.А. Визуальная локализация на основе геометрических признаков и 3D модели окружающей среды в системе ROS.//Ж-л Робототехника и техническая кибернетика. 2015 №4(9), 56-59 стр.

[Habrahabr, 2012] Habrahabr. CUDA: синхронизация блоков. –https:// habrahabr.ru/post/151897.

[OpenCV, 2016a] OpenCV. – http://opencv.org.

[OpenCV, 2016b] OpenCV. Описание функции Canny. – http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html.

[OpenCV, 2016c] OpenCV. Описание функции HoughLines. –http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghlines

[OpenCV, 2016d] OpenCV. Описание GPU функций Canny и HoughLines. – http://docs.opencv.org/2.4.13/modules/gpu/doc/image_processing.html.

[OpenCV, 2016e] OpenCV. Сравнительный анализ CPU и GPU. – http://opencv.org/platforms/cuda.html.

[Luo, 2008] Luo Y.,Duraiswami R. Canny Edge Detection on NVIDIA CUDA Perceptual Interfaces and Reality Laboratory Computer Science & UMIACS, University of Maryland, College Park. – http://www.umiacs.umd.edu/~ramani/pubs/luo_gpu_canny_fin_2008.pdf

[Pink, 2009] Pink, O., Moosmann, F., Bachmann, A. – Visual Features for Vehicle Localization and Ego-Motion Estimation. – http://www.mrt.kit.edu/z/publ/download/Pink_IV09.pdf`

[Saurer, 2010] Saurer, O., Fraundorfer, F., Pollefeys, M. – Visual localization using global visual features and vanishing points. – http://people.inf.ethz.ch/saurero/publications/saurer_2010clef_visual_localization.pdf