УДК 004.932, 004.042

ОБРАБОТКА СМЕЖНЫХ СТЕРЕОКАДРОВ С ПРИМЕНЕНИЕМ МНОГОЯДЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

А. К. Линьков, аспирант РТУ МИРЭА, Москва, Россия;
orcid.org/0009-0004-3725-508X, e-mail: alinkov20@yandex.ru

Рассматривается процесс обработки смежных стереокадров, а именно способ их предварительной обработки, а также алгоритм сопоставления фрагментов изображений (локальных особенностей) и формирования карты глубины. Показана структура алгоритма, которая представляет собой вложенные циклы. Из этого следует возможность исполнения таких алгоритмов на различных многоядерных вычислителях с использованием технологий параллельных вычислений. Эти технологии позволяют значительно повысить производительность обработки, приближая ее к реальному масштабу времени. Проводится обзор существующих классов многоядерных вычислительных систем. Анализируются их особенности и применимость для задачи обработки стереокадров.

Экспериментальные исследования проведены с использованием разработанной программы на языке Python, а также вспомогательных библиотек. Замерено время исполнения алгоритма в однопоточном режиме с использованием программной оптимизации и на массово-параллельном устройстве – графическом процессоре.

***Целью работы** является анализ алгоритма обработки смежных стереокадров, демонстрация различных подходов по оптимизации вложенных циклов, а также сопутствующий сравнительный анализ различных типов вычислительных систем и определение эффективности параллельного исполнения применительно к алгоритмам обработки стереоизображений.*

***Ключевые слова**: стереоизображение, карта глубины, фильтрация, алгоритмы поиска соответствий, параллельные вычисления, архитектура ЭВМ, графический процессор, ПЛИС.*

DOI*:* 10.21667/1995-4565-2025-91-209-219

Введение

Системы навигации на основе стереозрения имеют широкие перспективы применения в современной робототехнике, а также в других областях [1-3]. В качестве примеров можно привести автономных мобильных роботов [4], беспилотные летательные аппараты, беспилотные автомобили [5] и т. д. (рисунок 1).

Рисунок 1 – Примеры беспилотного роботизированного транспорта
Figure 1 – Examples of unmanned robotic vehicles

Системы технического зрения на основе камер весьма перспективны, на то есть несколько причин.

Многофункциональность. Системы на основе камер могут не только работать в стереорежиме для построения 3D-карты окружающей местности, но и использоваться для распознавания окружающих объектов. Примерами могут служить дорожные знаки для автомобилей [6], переносимые предметы для мобильных роботов и др. Помимо этого, камеры могут

использоваться и в качестве средств наблюдения и видеозаписи для обеспечения безопасности.

Пассивность. В отличие от дальномеров, действующих на основе излучения и приема сигналов различного рода (лазерное и инфракрасное излучение, радиоволны и т.д.), камеры лишь принимают световое излучение, но сами его не производят. Данная особенность имеет большое значение при широком распространении подобных устройств и увеличении их количества. Серьезной проблемой для активных (излучающих) датчиков радарного типа является «засветка» и создание помех от излучения других аналогичных датчиков, находящихся в непосредственной близости [7-8]. Это может привести к ложному обнаружению объектов и нестабильной работе датчиков. Также в случае лазерных дальномеров может иметь место проблема попадания лазерных лучей в глаза человека или животных.

Исходя из вышеописанных причин, а также относительной простоты алгоритмов обработки смежных стереокадров, было принято решение разработать и проанализировать описанные далее алгоритмы обработки. Целью исследования является анализ алгоритма обработки смежных стереокадров, демонстрация различных подходов по оптимизации вложенных циклов, а также сопутствующий сравнительный анализ различных типов вычислительных систем и определение эффективности параллельного исполнения применительно к алгоритмам обработки стереоизображений.

Теоретическая часть

Общий вид алгоритма обработки кадров

Рассматриваемый алгоритм состоит из следующих основных этапов.

1. Получение двух смежных кадров.
2. Обработка кадров – выделение информации о контурах, перепадах яркости и других локальных особенностей.
3. Для каждой точки (локальной особенности) на левом кадре – поиск наиболее близкого соответствия на правом кадре, согласно используемым метрикам.
4. Сохранение пар координат для точек с левого и правого кадров в качестве результата.
5. Визуализация результата в виде карты глубины (цветовой/тепловой карты и т.д.).

Так, для построения карты глубины на основе пары смежных стереокадров необходимо каждому фрагменту (локальной особенности) левого кадра поставить в соответствие фрагмент правого кадра. Сравнив координаты данного фрагмента на двух кадрах и зная геометрические параметры стереопары, можно вычислить расстояние от плоскости изображения до наблюдаемого объекта [9]. Повторив цикл поиска для каждой точки на паре смежных кадров, можно получить полную карту глубины. Пример исходных кадров и результат обработки представлен на рисунке 2. (Здесь и далее в качестве исходных использованы кадры из наборов данных [10]).

Рисунок 2 – Исходные кадры и результат обработки
Figure 2 – Original frames and processing result

Предварительная обработка

Для повышения качества дальнейшей обработки кадров необходима обработка изображения и выделение перепадов яркости с помощью фильтра [11]. Ключевым его отличием от

широко используемых «классических» фильтров является увеличение размера ядра до нескольких десятков пикселей. Это позволяет передать на результирующее изображение больше информации об окрестности каждого обрабатываемого пикселя.

Помимо этого, такая обработка позволит избавиться от необходимости калибровки стереопары по экспозиции (яркости), цветовому балансу и т.д. Пример обработки приводится на рисунок 3.

Рисунок 3 – Предварительная обработка кадра
Figure 3 – Frame pre-processing

Важно отметить, что данный алгоритм выполняется циклически для обоих смежных кадров. Это открывает широкие возможности по его оптимизации с помощью параллельных вычислений.

Сопоставление локальных особенностей

Для построения карты глубины (3D-карты) необходимо в цикле попарно сравнить между собой все фрагменты смежных кадров в пределах области поиска (рисунок 4). Для определения сходства фрагментов можно использовать различные численные метрики – «функции ошибки» [12]. Полученные в итоге пары координат (левого и правого фрагментов) выдаются в выходной массив в качестве результата. Далее их можно использовать для визуализации (рисунок 2), а также для дальнейшей обработки в системе управления роботом или транспортным средством, выдачи команд управления.

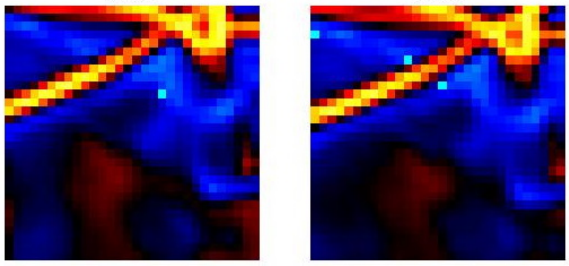

Рисунок 4 – Найденные сходные фрагменты
Figure 4 – Similar fragments found

Преимущества параллельных вычислений

Можно видеть, что большая часть алгоритмов обработки стереоизображений представляет собой циклы различной степени вложенности. Непосредственно вычисления внутри каждого цикла не несут большой вычислительной нагрузки. При последовательном (линейном) исполнении подобных циклов на обработку будет затрачиваться значительное время, а мощность вычислителя будет значительно недоиспользоваться. Поэтому видится целесообразным исполнение подобных алгоритмов в параллельном режиме с использованием многоядерных устройств.

При высоком разрешении исходных кадров имеет смысл для предварительной обработки разбить изображения на множество фрагментов (их количество будет зависеть от числа до-

ступных ядер вычислителя). Тогда описанный выше алгоритм обработки приобретает следующий вид.

1. Разбить два кадра на фрагменты (исходя из количества доступных вычислительных ядер).

2. Для каждого фрагмента отдельно в параллельном режиме:

– в цикле по фрагменту выполнить свертку изображения с ядром фильтра.

– на краях фрагмента применять данные, относящиеся к смежным фрагментам.

3. Объединить полученные результаты в выходной массив.

На рисунке 5 схематично показана разница во времени выполнения и загруженности ядер вычислителя на примере обработки 6 фрагментов изображения:

– обработка всего изображения без разбиения в однопоточном линейном режиме;

– последовательная обработка 6 фрагментов;

– параллельная обработка фрагментов, для обработки каждого используется отдельное ядро.

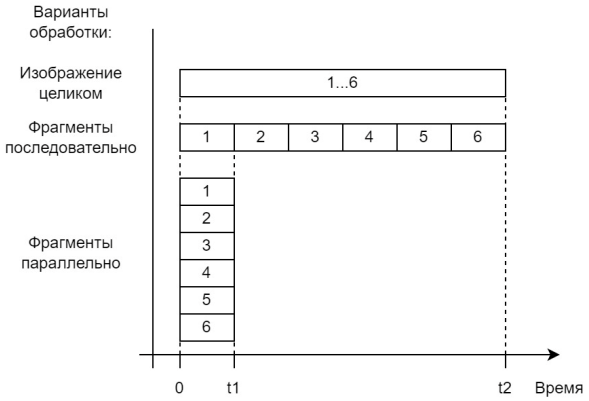

Рисунок 5 – Параллельная обработка фрагментов изображения
Figure 5 – Parallel processing of image fragments

Можно видеть, что при параллельном выполнении время обработки можно значительно сократить. В идеальных условиях (при наличии достаточного количества вычислительных ядер) общее время обработки (t1) будет близко ко времени обработки одного фрагмента изображения, т.е. быстрее приблизительно во столько раз, сколько имеется в наличии ядер.

Аналогичным образом можно представить параллельное выполнение поиска соответствий на правом и левом кадрах.

1. Распределить локальные особенности с левого кадра на группы в соответствии с количеством доступных вычислительных ядер.

2. Для каждой точки в пределах группы:

2.1. В соответствии с ее координатами и общими параметрами сцены определить возможный регион поиска на правом кадре.

2.2. Выполнить поиск наиболее близкого соответствия в пределах заданного региона.

2.3. Сохранить в массив результатов пару координат: искомой точки с левого кадра и найденного соответствия с правого кадра.

3. Объединить полученные на предыдущем шаге массивы со всех параллельных процессов.

4. Применить «Цветовую/тепловую карту», визуализировать полученную карту глубины либо передать ее для дальнейшей обработки.

Таким образом, при использовании параллельных вычислений, а также подхода SIMD (Single instruction Multiple data) можно значительно повысить производительность данного алгоритма. Это позволит приблизиться по скорости к реальному масштабу времени, что дает преимущества при использовании алгоритма в контуре управления различного беспилотного транспорта.

Классы многоядерных вычислителей и программная оптимизация

Наиболее распространенными вычислительными устройствами являются процессоры общего назначения (CPU), графические процессоры (GPU), а также программируемые логические интегральные схемы (ПЛИС/FPGA). Можно отметить, что при движении по этому перечню происходит увеличение количества вычислительных ядер, а также сужение их специализации.

Процессоры общего назначения (CPU)

Как правило, CPU используются в массовых потребительских устройствах, таких как ПК, смартфоны и др. Основными их особенностями является малое число ядер (обычно от 2 до 24) и сравнительно высокая тактовая частота (порядка единиц ГГц). Поэтому можно сказать, что возможности параллельных вычислений на CPU весьма ограничены, но для повседневных потребительских задач этого и не требуется. Основным преимуществом CPU при работе с повседневными задачами являются низкие задержки обработки. Структура CPU схематично показана на рисунке 6.

Рисунок 6 – Структура CPU
Figure 6 – CPU structure

В случае, если требующий параллелизации алгоритм необходимо выполнять на CPU с небольшим числом ядер, разработчик может использовать определенные процессорные инструкции для векторных и матричных вычислений (такие как SSE, AVX, AMX). Также можно использовать низкоуровневое программирование и специализированные компиляторы для языков высокого уровня.

Например, при разработке программ на языке Python можно использовать компилятор Numba [13]. Он переводит функции Python в оптимизированный машинный код во время выполнения программы. Особенно хорошо Numba работает с кодом, содержащим множество вложенных циклов, списков или операций с массивами. Оптимизация циклов и массивов позволяет полностью использовать возможности многоядерных процессоров и дает значительный прирост производительности по сравнению с однопоточным режимом.

Описанный выше алгоритм обработки стереоизображений также состоит из большого числа вложенных циклов, и поэтому использование такой оптимизации значительно ускорит его работу. Это будет показано далее в разделе «Экспериментальная часть».

Графические процессоры (GPU)

Графические процессоры имеют значительно большее число ядер по сравнению с процессорами общего назначения. Обычно их количество составляет тысячи или десятки тысяч. Однако при этом их тактовая частота существенно ниже. Очевидно, что изначально GPU были разработаны для расчета отображения трехмерной графики. Эту задачу можно разделить на множество мелких подзадач (расчет полигонов, источников света, теней и т.д.), каждая из которых в отдельности не представляет большой вычислительной сложности. Однако все эти задачи необходимо выполнять параллельно и с достаточно высокой производительностью. Именно этим и объясняется существующая структура графических процессоров (рисунок 7).

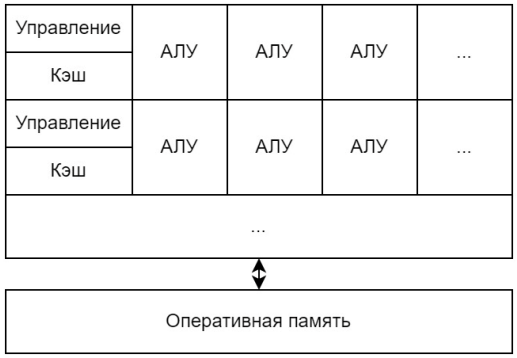

Рисунок 7 – Структура GPU
Figure 7 – GPU structure

В качестве примера оптимизации можно привести широко распространенную технологию параллельных вычислений CUDA от компании NVIDIA, позволяющую адаптировать программный код на популярных языках для параллельного исполнения на GPU [14].

CUDA – это программно-аппаратная архитектура параллельных вычислений, которая позволяет значительно увеличить производительность вычислений за счет использования графических процессоров от NVIDIA. Похожая на библиотеку компилятора Numba, CUDA позволяет разработчикам повысить производительность приложений, требующих интенсивных вычислений, за счет использования мощности графических процессоров для распараллеливания части вычислений.

Хотя существуют и другие технологии разработки для GPU, такие как OpenCL [15], технология CUDA для графических процессоров NVIDIA является наиболее популярной в настоящее время.

Технология CUDA использует одновременно CPU (называемый Хостом/Host) и GPU (называемый Устройством/Device). Код, изначально запущенный на Хосте, может управлять также памятью Устройства, а также вызывать функции, которые будут выполняться непосредственно на Устройстве. На GPU эти функции выполняются в многопоточном режиме, что позволяет значительно увеличить производительность. Стоит отметить, что сама по себе такая функция выглядит как обыкновенная последовательно исполняемая программа.

При помощи технологии CUDA программа выполняется следующим образом.

1. Хост выделяет память на Устройстве.
2. Хост копирует исходные данные на Устройство.
3. Хост запускает функцию на Устройстве.
4. Хост копирует результаты с Устройства обратно для дальнейшей обработки.

Устройство CUDA состоит из множества ядер, каждое из которых выполняет отдельную копию функции для обработки данных. Таким образом используется массово-параллельная обработка.

Основным преимуществом GPU является собственный конвейер вычислений, позволяющий значительно ускорить обработку векторизованных данных. Таким образом, предварительно распределив данные по потокам для каждого вычислительного ядра, можно добиться значительного повышения производительности. Это можно сделать как автоматически, так и вручную. Однако важно понимать, что также имеет значение время передачи данных на GPU и обратно. Помимо этого, необходимо учитывать и аспекты синхронизации данных между ядрами.

ПЛИС (FPGA)

Принцип работы ПЛИС с архитектурой FPGA значительно отличается как от CPU, так и от GPU. Основная идея FPGA заключается в синтезе специального вычислителя под конкретные задачи, т.е. программирование заменяется проектированием. Также важно отметить

так называемую «мелкозернистую архитектуру», количество вычислительных ядер в которой может быть порядка миллиона и более.

Микросхема FPGA состоит из логических ячеек и коммутаторов (рисунок 8). Каждый из них может быть перепрограммирован, т.е. можно изменять как работу логических ячеек (CLB – программируемый логический блок), так и соединений между ними (коммутационных матриц). В свою очередь, CLB состоит из блока LUT (таблицы истинности) и триггера. Это позволяет реализовывать любые логические функции, зависящие от входов блока.

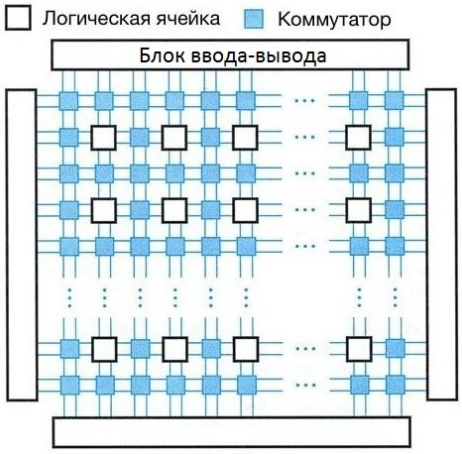

Рисунок 8 – Структура ПЛИС
Figure 8 – FPGA structure

Помимо логических ячеек и коммутационных матриц, большинство FPGA имеют дополнительные аппаратные блоки. Это может быть блок умножения с накоплением, блок DSP (цифровой обработки сигналов) и другие. Количество таких модулей также может быть весьма велико.

Таким образом, основное преимущество FPGA – гибкая структура и возможность реализовывать практически любые аппаратные схемы для скоростной обработки данных (за счет очень большого количества вычислительных ядер). При правильном подходе к проектированию структуры такого вычислителя можно получить производительность, значительно превосходящую GPU и близкую к реальному масштабу времени.

В качестве краткого вывода можно выделить следующие особенности ПЛИС по сравнению с другими вычислительными устройствами:

– низкая задержка обработки данных;
– высокая пропускная способность;
– высокое энергопотребление;
– высокая стоимость.

Экспериментальные исследования

Исходя из имеющегося в наличии оборудования, а также принимая во внимание результаты теоретического анализа, были проведены экспериментальные исследования. А именно на основе изображений из набора данных [10] было проведено построение карты глубины различными способами, с использованием функций ошибки SAD, SSD, NCC [12, 16, 17]. Оценивалось и сравнивалось время исполнения алгоритма с использованием различных аппаратных средств и программной оптимизации. Размер тестовых изображений составляет 640 на 439 пикселей.

Разработанная тестовая программа написана на языке Python 3 с использованием библиотек CV2, Numpy, Numba, а также стандартных системных инструментов. Время исполнения определялось при помощи библиотеки Timeit.

Для вычислений использовались следующие аппаратные средства:

CPU: Intel Core i5-1135G7 – 4 ядра, 8 потоков, кэш-память 8 МБ, тактовая частота от 2,4 до 4,2 ГГц;

GPU: NVIDIA GeForce MX330 – 384 ядра CUDA, выделенная память 2 ГБ, тактовая частота от 1,5 до 1,7 ГГц.

В качестве программной оптимизации для CPU использовался компилятор Numba. Проведено сравнение исполнения алгоритма в однопоточном режиме и в многопоточном с оптимизацией. Для обработки данных на GPU использовались средства разработки для технологии CUDA.

Результаты эксперимента представлены в таблице 1.

Таблица 1 – Время исполнения алгоритма
Table 1 – Algorithm execution time

Наименование используемой функции	Время исполнения алгоритма, с			Отношение времени исполнения на CPU и GPU
	CPU	CPU (Numba)	GPU (CUDA)	
SAD	1656,392	3,507	0,216	7660
SSD	1629,170	3,108	0,004	427279
NCC	4300,640	7,712	0,003	1505756

По результатам эксперимента можно видеть, что при исполнении алгоритма на CPU без использования каких-либо оптимизаций время составляет десятки минут. При использовании компилятора Numba и оптимизации вложенных циклов удается снизить время исполнения на несколько порядков – до единиц секунд (выигрыш составляет около 400-500 раз). Однако этого все еще недостаточно для работы в реальном времени. Наконец, использование GPU и технологии CUDA позволило увеличить скорость обработки еще на несколько порядков (зависит от алгоритма), что уже позволяет говорить о работе в реальном времени.

Так, из данных таблицы 1 видно, что время исполнения алгоритмов на GPU составляет в худшем случае около 0,2 с. Скорость работы составляет около 5 кадров в секунду, этого может быть уже достаточно для работы в «спокойной» обстановке в реальном времени. Можно также предположить, что при использовании более производительных GPU скорость обработки возрастет, что позволит использовать данные подходы при работе в более динамично меняющейся среде.

Также можно заметить, что для более сложных функций расчета выигрыш во времени исполнения становится значительно больше. Это объясняется тем, что данные функции вычисляются в каждом из множества вложенных циклов – их последовательное выполнение занимает больше времени (см. рисунок 5). Однако можно видеть, что время исполнения более простого алгоритма SAD на GPU больше по сравнению с другими алгоритмами. Это можно объяснить увеличением затрат на передачу данных с хост-устройства на GPU и обратно.

Исследование работы алгоритма на ПЛИС не проводилось ввиду отсутствия технической возможности. На основании проведенного обзора можно сделать предположение о дальнейшем повышении производительности, что потенциально даст возможность обрабатывать изображения и принимать решения по управлению в реальном времени. Помимо этого, ПЛИС имеют следующие преимущества:

– точное определение задержки при работе алгоритма – необходимо для работы в режиме реального времени;

– при той же производительности энергопотребление будет значительно ниже – хорошо для применения на автономных роботах и транспорте;

– меньшее тепловыделение и необходимость в отводе тепла – также положительно влияет на физические характеристики (размеры, вес) готового устройства (может использоваться даже для летательных аппаратов);

– также преимуществом ПЛИС является широчайший выбор интерфейсов связи с другими устройствами системы управления.

Единственным существенным недостатком может быть более высокая сложность и стоимость разработки программного обеспечения.

Заключение

В данной работе представлен разработанный алгоритм обработки стереоизображений и построения карты глубины. Показано, что производительность данного алгоритма может быть значительно увеличена путем использования различных средств параллельных вычислений.

Прикладное значение проведенного исследования заключается в определении производительности алгоритма при исполнении на вычислительных устройствах различных классов параллелизации. Проведено их сравнение и сделаны выводы о возможностях их использования. Также по полученным результатам можно оценить возможность использования таких систем для достижения обработки в реальном времени. Подобная обработка необходима для применения в различных транспортных средствах, автономных роботах и т.д.

Библиографический список

1. **Siciliano B., Khatib O., Kröger T.** Springer handbook of robotics. Berlin: springer, 2008. https://doi.org/10.1007/978-3-319-32552-1

2. **Евстигнеев Д.В.** Программно-алгоритмическое обеспечение интеллектуальных систем управления автономными мобильными роботами: дис. канд. тех. наук: 05.13.01 / Московский государственный институт радиотехники, электроники и автоматики. М., 2003. 218 с.

3. **Wiseman Y.** Autonomous vehicles. Research anthology on cross-disciplinary designs and applications of automation. Igi Global, 2022. С. 878-889. https://doi.org/10.4018/978-1-6684-3694-3.ch043

4. **Muhumed A.M.** Autonomous Delivery Robots for Urban Settings // Metropolia University of Applied Sciences, 2022. https://urn.fi/URN:NBN:fi:amk-2022060315318

5. **Duarte F., Ratti C.** The impact of autonomous vehicles on cities: A review // Journal of Urban Technology. 2018. Т. 25. №. 4. С. 3-18. https://doi.org/10.1080/10630732.2018.1493883

6. **Bousarhane B., Bensiali S., Bouzidi D.** Road signs recognition: state-of-the-art and perspectives // International Journal of Data Analysis Techniques and Strategies. 2021. Т. 13. № 1-2. С. 128-150. https://doi.org/10.1504/IJDATS.2021.114672

7. **Sun S., Petropulu A.P., Poor H.V.** MIMO radar for advanced driver-assistance systems and autonomous driving: Advantages and challenges //IEEE Signal Processing Magazine. 2020. Т. 37. № 4. С. 98-117. https://doi.org/10.1109/MSP.2020.2978507

8. **Xu W.** et al. Analyzing and enhancing the security of ultrasonic sensors for autonomous vehicles // IEEE Internet of Things Journal. 2018. Т. 5. № 6. С. 5015-5029. https://doi.org/10.1109/JIOT.2018.2867917

9. **Kang S.B.** et al. An active multibaseline stereo system with real-time image acquisition. Carnegie-Mellon University. Department of Computer Science, 1994.

10. **Scharstein D.** et al. High-resolution stereo datasets with subpixel-accurate ground truth // Pattern Recognition: 36th German Conference, GCPR 2014, Münster, Germany, September 2-5, 2014, Proceedings 36. Springer International Publishing, 2014. С. 31-42. https://doi.org/10.1007/978-3-319-11752-2_3

11. **Линьков А.К., Потехин Д.С.** Нормализация изображений и выделение локальных особенностей для формирования дальнометрической карты // Высокопроизводительные вычислительные системы и технологии. 2022. № 2. С. 29-35.

12. **Fsian H.** et al. Comparison of Stereo Matching Algorithms for the Development of Disparity Map// arXiv preprint arXiv:2210.15926. 2022.

13. **Lam S.K., Pitrou A., Seibert S.** Numba: A llvm-based python jit compiler // Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC. 2015. С. 1-6. https://doi.org/10.1145/2833157.2833162

14. **Luebke D.** CUDA: Scalable parallel programming for high-performance scientific computing // 2008 5th IEEE international symposium on biomedical imaging: from nano to macro. IEEE, 2008. С. 836-838. https://doi.org/10.1109/ISBI.2008.4541126

15. **Munshi A.** The opencl specification // 2009 IEEE Hot Chips 21 Symposium (HCS). IEEE, 2009. С. 1-314. https://doi.org/10.1109/HOTCHIPS.2009.7478342

16. **Szeliski R.** Computer vision: algorithms and applications. Springer Nature, 2022.

17. **Forsyth D.A., Ponce J.** Computer vision: a modern approach. Prentice hall professional technical reference, 2002. https://doi.org/10.5555/580035

UDC 004.932, 004.042

PROCESSING OF ADJACENT STEREO FRAMES USING MULTI-CORE COMPUTING SYSTEMS

A. K. Linkov, post-graduate student, RTU MIREA, Moscow, Russia;
orcid.org//0009-0004-3725-508X, e-mail: alinkov20@yandex.ru

This paper presents the process of adjacent stereo frames analysis, specifically the method of their preprocessing, as well as an algorithm for matching image fragments (local features) and generating a depth map. The structure of the algorithm, which consists of nested loops, is shown. This implies the possibility of executing such algorithms on various multi-core computing systems, using parallel computing techniques. These techniques would significantly improve processing performance, bringing it closer to real time scale. The existing classes of multi-core computing systems are analyzed. Their features and applicability to the task of stereo frame processing are reviewed.

Experimental studies were carried out using the developed program in Python language and auxiliary libraries. The execution time of the algorithm was measured in single-threaded mode, using software optimization, and on a massively parallel device – graphics processor.

***The aim of the work** is a comparative analysis of different types of computing systems, as well as demonstration of optimization possibilities for nested loops.*

***Keywords:** stereo image, depth map, filtering, matching algorithms, parallel computing, computer architecture, graphics processor, FPGA.*

DOI*:* 10.21667/1995-4565-2025-91-209-219

References

1. **Siciliano B., Khatib O., Kröger T.** Springer handbook of robotics. Berlin, springer, 2008. https://doi.org/10.1007/978-3-319-32552-1

2. **Evstigneev D.V.** Programmno-algoritmicheskoe obespechenie intellektual'nyh sistem upravlenija avtonomnymi mobil'nymi robotami (Software-algorithmic support of intelligent control systems for autonomous mobile robots). *PhD thesis*. Moscow, 2003. (in Russian)

3. **Wiseman Y.** Autonomous vehicles. *Research anthology on cross-disciplinary designs and applications of automation*. Igi Global, 2022, pp. 878-889. https://doi.org/10.4018/978-1-6684-3694-3.ch043

4. **Muhumed A.M.** Autonomous Delivery Robots for Urban Settings. *Metropolia University of Applied Sciences.* 2022. https://urn.fi/URN:NBN:fi:amk-2022060315318

5. **Duarte F., Ratti C.** The impact of autonomous vehicles on cities: A review. *Journal of Urban Technology*. 2018, vol. 25, no. 4, pp. 3-18. https://doi.org/10.1080/10630732.2018.1493883

6. **Bousarhane B., Bensiali S., Bouzidi D.** Road signs recognition: state-of-the-art and perspectives. *International Journal of Data Analysis Techniques and Strategies*. 2021, vol. 13, no. 1-2, pp. 128-150. https://doi.org/10.1504/IJDATS.2021.114672

7. **Sun S., Petropulu A.P., Poor H.V.** MIMO radar for advanced driver-assistance systems and autonomous driving: Advantages and challenges. *IEEE Signal Processing Magazine*. 2020, vol. 37, no. 4, pp. 98-117. https://doi.org/10.1109/MSP.2020.2978507

8. **Xu W.** et al. Analyzing and enhancing the security of ultrasonic sensors for autonomous vehicles. *IEEE Internet of Things Journal*. 2018, vol. 5, no. 6, pp. 5015-5029. https://doi.org/10.1109/JIOT.2018.2867917

9. **Kang S.B.** et al. An active multibaseline stereo system with real-time image acquisition. Carnegie-Mellon University, 1994.

10. **Scharstein D.** et al. High-resolution stereo datasets with subpixel-accurate ground truth. *Pattern Recognition: 36th German Conference*. Springer International Publishing. 2014, pp. 31-42. https://doi.org/10.1007/978-3-319-11752-2_3

11. **Linkov A.K., Potehin D.S.** Normalizacija izobrazhenij i vydelenie lokal'nyh osobennostej dlja formirovanija dal'nometricheskoj karty. *Vysokoproizvoditel'nye vychislitel'nye sistemy i tehnologii*. 2022, no. 2, pp. 29-35. (in Russian)

12. **Fsian H.** et al. Comparison of Stereo Matching Algorithms for the Development of Disparity Map. *arXiv preprint 2210.15926*. 2022.

13. **Lam S.K., Pitrou A., Seibert S.** Numba: A llvm-based python jit compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 2015, pp. 1-6. https://doi.org/10.1145/2833157.2833162

14. **Luebke D.** CUDA: Scalable parallel programming for high-performance scientific computing. *2008 5th IEEE international symposium on biomedical imaging: from nano to macro*. IEEE, 2008, pp. 836-838. https://doi.org/10.1109/ISBI.2008.4541126

15. **Munshi A.** The opencl specification. *2009 IEEE Hot Chips 21 Symposium (HCS)*. IEEE, 2009, pp. 1-314. https://doi.org/10.1109/HOTCHIPS.2009.7478342

16. **Szeliski R.** Computer vision: algorithms and applications. *Springer Nature*, 2022.

17. **Forsyth D. A., Ponce J.** *Computer vision: a modern approach. Prentice hall professional technical reference.* 2002. https://doi.org/10.5555/580035