САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

Курбанов Рауф Эльшад оглы

Распознавание лиц на базе контроллера ТРИК

Курсовая работа

Научный руководитель:
ст. преп. Кириленко Я. А.

Санкт-Петербург
2014

Оглавление

Введение

Задача распознавания лиц находит много практических приложений начиная программами для фотокамер и заканчивая многоступенчатыми системами аутентификации. Автоматический поиск лиц создает ощущение машинного интеллекта, что порождает соблазн воспользоваться преимуществом такого эффекта на поприще робототехники. Также задача поиска лиц уже успела войти в ряд канонических в компьютерном зрении, что предоставляет сильную базу проверенных подходов к её реализации. Исходя из этим факторов, мне была поставлена цель разработки встроенной реализации алгоритма на базе робототехнического контроллера ТРИК [5].

В контексте данной курсовой под названием ТРИК я буду подразумевать контроллер, входящий в состав одноименного конструктора. Далее стоит поподробнее описать особенности конфигурации оборудования, имеющие значения в моей работе.

1. Платформа

На плате контроллера находится процессор OMAP-L138 от Texas Instruments [3], объединяющий на одном кристалле процессор общего назначения ARM и процессор цифровой обработки сигналов DSP. Помимо всего прочего плата снабжена wi-fi модулем для передачи данных, сенсорным дисплеем и USB выходом, дающим возможность подключить внешнюю камеру. Плата конструктора находится под управлением операционной системы на базе ядра Linux.

2. Цель работы

- Разработка приложения для распознавания лиц в реальном времени на базе контроллера ТРИК
- Апробация алгоритма на реальной модели
- Знакомство с архитектурой целевой платформы
- Изучение и реализация алгоритма для распознавания лиц
- Получение опыта кросс-платформенной разработки

3. Алгоритм

Опираясь на описание из статьи Robust Real-Time Face Detection [4] и основываясь на приведенных там сравнениях с другими алгоритмами для распознавания лиц, мною был выбран алгоритм Виолы-Джонса. Это классический и проверенный алгоритм для поиска лиц на изображении, доказавший свою эффективность, и известный вычислительной легкостью.

Работу алгоритма можно разделить на три основных этапа:

1. Интегральное представление
2. Обучение
3. Поиск лица и принятие решения

На вход приложению поступает буфер с изображением в формате YUV422. Это запакованный формат для представления видео, но чтобы его описать, стоит рассказать, как устроенно цветовое пространство YUV.

В схеме YUV пиксель изображения представляется с помощью трех компонент: две цветовые компоненты Y и U. и компонента V, отвечающая за яркость.

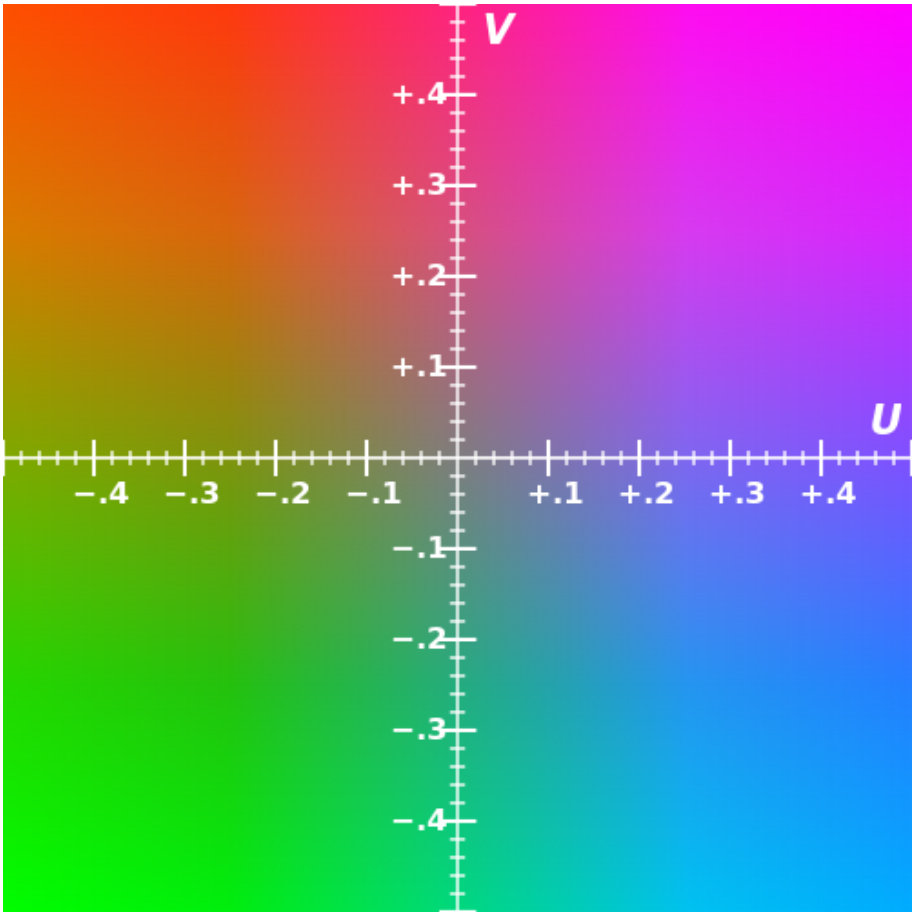

Рис. 1: Срез пространства YUV

Поскольку для работы алгоритма нужно интегральное представление изображения, которое не требует информации о цвете, то достаточно считать из входного буфера только компоненту яркости картинки, по которой и построим необходимое представление. Однако, как я упоминал ранее, изображение хранится в запакованном формате,а именно YUV422. Так выглядит макропиксель изображения.

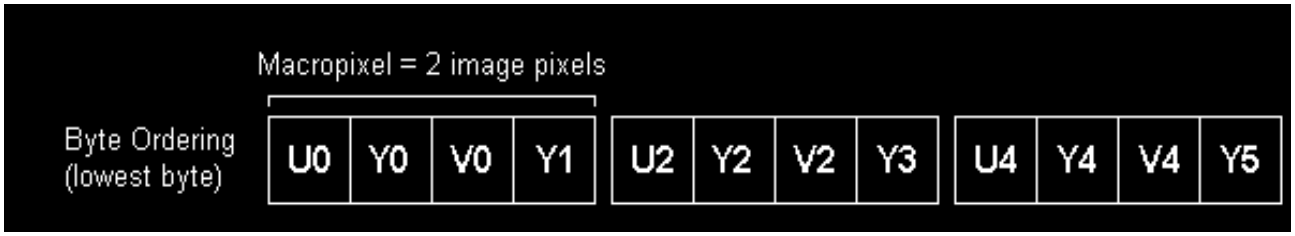

Рис. 2: Структура формата YUV422

По рисунку видно,что из каждого байта вы извлекаем компоненту яркости для двух пикселей картинки.

4. Интегральное представление

Интегральное изображение это такое представление картинки, в котором каждому пикселю сопоставляется сумма яркостей всех пикселей левее и выше данного.

Значения в интегральном представлении высчитываются по следующей формуле.

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

Такое представление позволяет нам быстро вычислить яркость произвольного прямоугольника на изображении следующий образом.

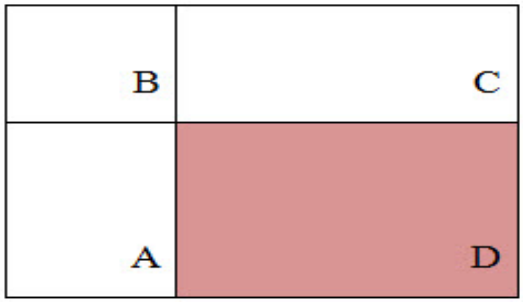

Рис. 3: Схема вычисления яркости в интегральном представлении

$$S(ABCD) = L(D) - L(A) - L(C) + L(B)$$

Полученные знания о яркости произвольной области на изображении понадобятся нам на следующем этапе.

5. Признаки Хаара

Признаком называется отображение $f : X \to Y$, где X- множество распознаваемых объектов, а Y — множество допустимых значений признака. В стандартном методе Виолы – Джонса используются прямоугольные признаки, изображенные на рисунке ниже, они называются примитивами Хаара:

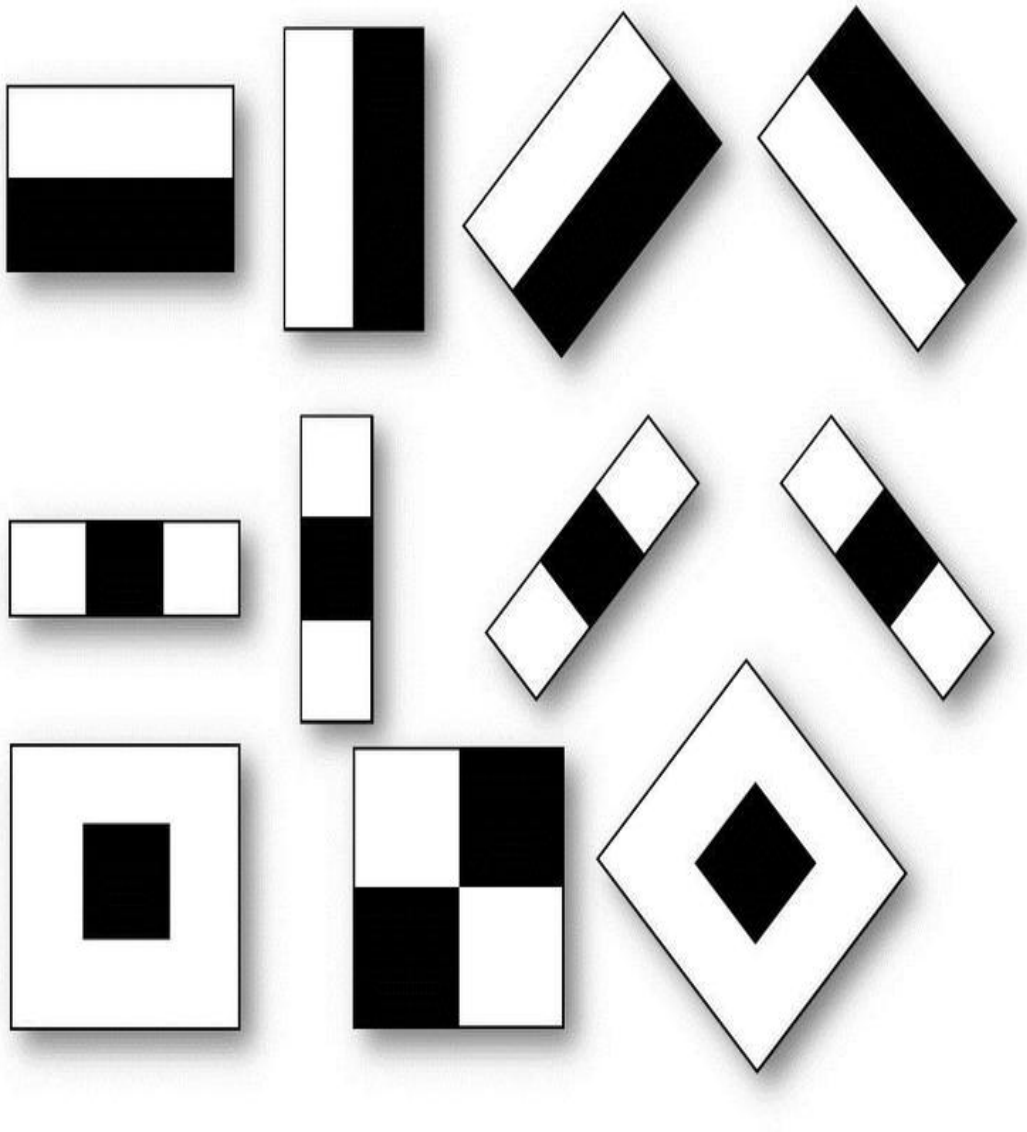

Рис. 4: Примитивы Хаара

Вычисляемым значением такого признака будет $F = X - Y$, где X – сумма значений яркостей точек закрываемых светлой частью признака, а Y – сумма значений яркостей точек закрываемых темной частью признака. Для их вычисления используется понятие интегрального изображения, рассмотренное выше. Признаки Хаара дают точечное значение перепада яркости по оси X и Y соответственно.

6. Обучение классификатора

В своей работе я опустил часть алгоритма, связанную с машинным обучением. Аргументированно это было тем, что основной задачей была реализация под конкретную платформу и задачи улучшить качество распознавания не ставилось. Поэтому я опишу лишь способ принятия решений, опустив описание метода обучения.

Итак, для каждого признака Хаара мы имеем некоторое значение, характеризующее, присутствует лицо на картинке или нет. Теперь назначим некоторое пороговое значение и скажем, что если классификатор выдает результат меньше этого , то лица нет, больше - есть.

Для каждого обрабатываемого фрейма “выносит вердикт” специально обученный каскад классификаторов. В нашем случае, каскад это множество классификаторов, которые выносят решение в соответствие со своим приоритетом. Порог классификациии выбран таким образом, что вероятность ложно определить отсутствие лица очень мало, в то время как часто классификатор может сказать, что лицо на картинке есть, даже если его там нет. Такой выбор порогового значения позволяет использовать систему голосования: все классификаторы в каскаде выносят своё решение в соответствие с назначенным им приоритетом. Если на каком-то шаге классификатор с большим приоритетом скажет, что на изображении лица нет, то дальнейшее принятие решения прекращается, что является некоторой эвристикой.

7. Сканирование окна

Краткое описание шагов процесса сканирования:

- есть исследуемое изображение, выбрано окно сканирования, выбраны используемые признаки;
- далее окно сканирования начинает последовательно двигаться по изображению с шагом в одну ячейку окна;
- при сканировании изображения в каждом фрейме вычисляются всевозможные варианты расположения признаков за счет изменения масштаба признаков и их положения сканирующего фрейма в окне;
- сканирование производится последовательно для различных масштабов;
- все найденные признаки попадают к классификатору, который “выносит вердикт”.

8. Запуск на ARM

Как только у меня появился рабочий прототип алгоритма, я попытался запустить его на процессоре ARM, не используя возможности предоставляемые DSP. Этот промежуточный этап не потребовал настолько больших усилий, чтобы его пропустить. Для его осуществления было достаточно основных навыков кросс-компиляции и сборки проекта под другую платформу. На данном шаге я получил не очень точное, но очень показательное наблюдение: скорость работы алгоритма на процессоре ARM была дольше чем на настольном компьютере примерно в 30 раз. Я не стал проводить точных замеров, так как даже такая поверхностная оценка ярко показывает, что оптимально реализовать алгоритм на плате, не задействовала второй процессор, невозможно. В тот момент такой результат был ожидаем, так как в текущей конфигурации контроллера ARM выполняет роль процессора общего назначения и не предназначен для сложных вычислений.

Исходный код доступен в моем репозитории:
https://github.com/Rauf-Kurbanov/vj_arm

9. Реализация под DSP

Digital signal processor - процессор цифровой обработки сигналов(далее DSP) это микропроцессор, со специализированной архитектурой. Из многочисленных особенностей такого процессора меня интересют поддержка векторно-конвейерной обработки и аппаратная реализация многократного повторения заданного набора команд.

Для использования первой возможности я планировал воспользоваться библиотекой DSPLib [1] на самом вычислительно затратном шаге алгоритма: в момент принятия решения классификатором. DSP Library (DSPLIB) это совокупность арифметических модулей, оптимизированных для исполнения на процессоре DSP. Для того, чтобы воспользоваться данной библиотекой требуется спроектировать приложение таким образом, чтобы числа над которыми производятся аналогичные операции были объединены в вектора, поскольку весь интерфейс библиотеки специализирован для выполнения векторных операций.

Для параллелизма же я планировал продублировать структуры данных, используемые при обработке изображений сканирующим окном, чтобы распараллелить данный процесс для разных окон. Однако, меня застигли непредвиденные сложности.

Исходный код мною также опубликован:
https://github.com/Rauf-Kurbanov/trik-vidtranscode-cv/tree/rover

10. Сложности архитектуры

Структура приложения исполняемого на двух совершено разных процессорах с разной архитектурой достаточно нетривиальна. Для её проектирования нужно обладать основательными знанием библиотеки Codec Engine [2], предоставляющей интерфейс для межпроцессорного взаимодействия. Такое приложение уже было разработано до меня, а мне предстояло реконструировать исходный код для своих нужд.

Опишу структуру такого приложения в общих чертах. У вас должно быть скомпилировано два проекта:

- приложение под ARM, отвечающее за ввод-вывод данных, а также получение данных с камеры через интерфейс video for linux 2; признаки;
- приложение под DSP, в котором должен быть инкапсулирован XDAIS алгоритм, принимающий входной буффер с данными и возвращающий буфер с результатом, который отправляется обратно на ARM.

При такой структуре, чтобы передать буфер с данными в XDAIS алгоритм нужно пройти множество промежуточных интерфейсов, за подробное описание которых я не возьмусь, а также существенно затронуть существующий код программы. В итоге, с целью избежать таких проблем, мне пришлось инстанцировать данные для классификации в статике, что впоследствии скажется на производительности, затруднив дублирование структур данных для параллелизма.

11. Сложности отладки

Вообще с кросс-платформенной разработка влечет за собой некоторые сложности. Во-первых, отсутствовал подобающий эмулятор, поэтому разрабатывать приходилось прямо на аппаратуре, что сильно увеличивает время разработки и создает зависимость от соответствующего оборудования. Во-вторых, не была решена проблема с отладочной печатью, да и отсутствие собственно отладчика сильно затрудняет процесс. Остались всего два метода отладки: запись в выходной буфер и возврат кода ошибки. Однако, они потеряли смысл, так как в моем случае приложение не завершалось свою работу.

Осуществить полноценную трассировку возможности также не было, однако, существующее приложение осуществляло на ARM оценку производительности DSP по времени отклика, а также выдавало приблизительный fps. Единственным средством отладки стало частично комментирование и кода и поиск узких мест исходя из данных о времени отклика.

Таким образом, я выяснил, что к моменту вычисления основной алгоритмически сложной части загрузка процессора достигает уже 80

12. Проблемы с реализацией

Изначально стояла задача разработать прототип приложения, скорость и качество работы были не критичны, главное интегрировать алгоритм в текущую архитектуру. Однако после анализа сложившейся критической ситуации выяснилось, что запустить “какую-то” реализацию алгоритма не получится, так как ввиду особенностей существовавшей до меня программной среды приложение падает при слишком большой загрузке процессора.

Мною был предприняты попытки получения алгоритмической эвристики за счёт подбора параметров в ущерб точности распознавания. Такими параметром стал шаг сканирующего, в результате я должен был получить прирост производительности, уменьшив диапазон масштабов лиц, которые могут быть распознаны.

Однако, после более глубокого анализа производительности выяснилось, что процессор не в состоянии выполнить даже одну итерацию алгоритма. Приняв ввиду все описанные ранее проблемы, лишившие меня возможности к полноценной оптимизации, я решил завершить свою работу, вынеся из неё выводы и рекомендации для возможных будущей реализаций задачи.

13. Рекомендации

- Подробно изучить библиотеку Codec Engine, грамотно инстанцировать память, не создавая препятствий для возможности будущего параллелизма.
- Возможно попытаться выбрать другой алгоритм для распознавания лиц, более оптимизированный для вычисления на DSP или же доработать алгоритмические эвристики для текущего.
- Более активно использовать DSPLib: не только в “узких местах” алгоритма, а вообще везде где только можно, для этого кардинально перестроить структуру программы относительно варианта для классической архитектуры.

Результаты

- Изучен и реализован под несколько целевых платформ алгоритм Виолы-Джонса для поиска лиц на изображении
- Получены знания об архитектуре процессора цифровой обработки сигналов
- Проведено обзорное знакомство с фреймворком Codec Engine, с разработкой приложения со сложным межпроцессорным взаимодействием
- В ходе исследования выявлены проблемы реализации данного алгоритма под целевую платформу, сформулированы пути решения возникший проблем
- Получен опыт и наработки для будущей реализации первоначальной задачи на основе данной работы, однако уже вне её рамок

Список литературы

[1] C674x DSPLIB wiki. — (Дата обращения 1.06.2014). URL: http://goo.gl/geVfDh.

[2] Codec Engine wiki. — (Дата обращения 1.06.2014). URL: http://processors.wiki.ti.com/index.php/Codec_Engine.

[3] Texas Instruments OMAP-L138. — (Дата обращения 1.06.2014). URL: http://www.ti.com/product/omap-l138/.

[4] Viola Paul, Jones Michael. Robust real-time face detection // International Journal of Computer Vision. — 2004. — Vol. 57. — P. 137–154.

[5] Кибернетический конструктор ТРИК. — (Дата обращения 1.06.2014). URL: http://www.trikset.com/.