САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра Системного программирования

Овчинников Сергей Андреевич

Эффективная детекция и локализация графического текста на видео

Курсовая работа

Научный руководитель:
доц. А. Т. Вахитов

Санкт-Петербург
2015

Оглавление

Введение

С ростом объемов цифрового мультимедиа в современном мире растет необходимость в своевременном извлечении и индексировании информации, содержащейся на видео. Особенный интерес представляют различного рода описания в ТВ-передачах и встроенные в видео субтитры. Помимо экстрагирования данных, детекция и локализация текста на видео может служить для препятствия выгорания пикселей [19] на экранах телевизоров и мониторах в связи с присутствием на видео статичного текста. Например, [1] использует детектор статичных регионов, встроенный в медиа процессор ТВ, для детекции субтитров.

Текст на видео может быть классифицирован на две логические категории: графический текст (рис. 1), который непосредственно наносится при монтаже или видеообработке и сценический текст (рис. 2), который содержится на предметах, захваченных на видео. Примером сценического текста могут служить дорожные знаки, билборды, текст на фургонах и надписи на майках. Появление сценического текста зачастую носит случайный характер и его распознавание служит лишь для целей навигации, наблюдения и отслеживания конкретного объекта, нежели индексации и извлечения информации. Примером графического текста являются заголовки, субтитры, ключевые слова, время и отметка расположения, имена людей и спортивные результаты. Расположение этого вида текста предугадывается, а сам текст имеет простой стиль и ориентирован на прочтение человеком .

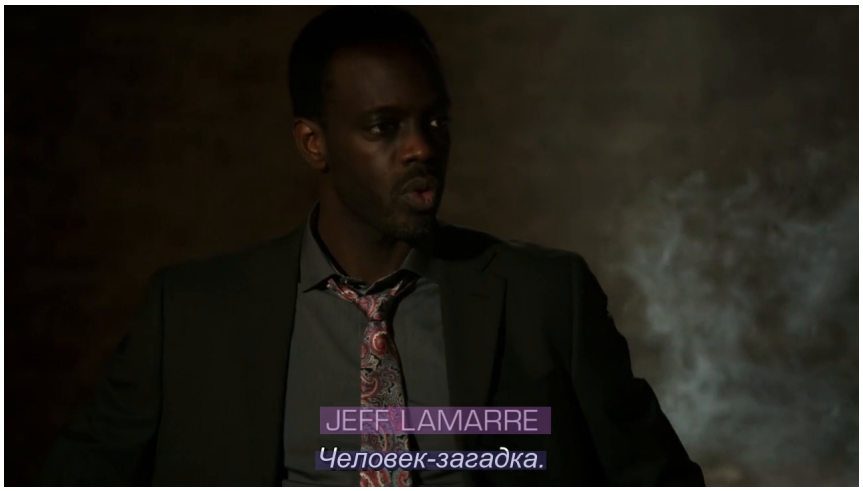

Рис. 1: Пример графического текста

В очень многих работах по детекции текста на видео пытаются [8] сдетектировать и локализовать оба вида текста на видео, что значительно сложнее, чем детектирование лишь графического текста, что выливается в низкую скорость работы имеющихся решений (около 1 fps) и низкий Recall rate $= \frac{\text{true positive}}{\text{true positive+false negative}}$ и высокий False alarm rate $= \frac{\text{false positive}}{\text{true positive+false positive}}$.

Рис. 2: Пример сценического текста

Обычно, извлечение текста из видео включает в себя детекцию, локализацию и сегментацию. Задачей детекции и локализации является определение наличия субтитров и отметка соответствующих регионов на видео; сегментация служит для определения лишь текстовых пикселей из обнаруженных регионов и составления бинарного изображения для дальнейшего оптического распознавания символов. Было показано, что сдетектированный и локализованный регион текста с изображения может быть распознан OCR с большей точностью ([4], [2]) . Причиной этого является тот факт, что локализованный регион текста менее сложен по сравнению со всем изображением и символы могут быть легко распознаны. Здесь детекция и локализация текста ускоряют затратный процесс OCR.

Большинство работ по данной теме базируются лишь на так называемом пространственном (”spatial”) анализе, когда детекция и локализация текста происходит для всех фреймов независимо. Это неэффективно в рамках поставленной задачи. Более того, авторы многих решений изначально не имеют перед собой цели достичь приемлемых результатов в части скорости работы. Также хочется отметить, что почти во всех работах датасетами являлись видео низких разрешений, что в нынешнее время бурного развития мультимедия не отражает реальной вычислительной эффективности предложенных алгоритмов. К тому же, авторы алгоритмов зачастую не предоставляют свои наработки для публичного использования, поскольку их работы носили не только исследовательский, но и закрытый, прикладной характер.

1. Постановка задачи

Целью работы является разработка алгоритма, результатом которого будет быстрая детекция и локализация графического текста на видео, и дальнейшее тестирование этого алгоритма. Т.е. в итоге мы должны получить границы строк с текстом (т.н. “bounding boxes”), на которых представлен графический текст, причем по скорости эта процедура должна быть сравнима с “real time” на видео с разрешением вплоть до $1280 * 720$. При этом мы допускаем следующие ограничения на детектируемый текст. Во-первых, он должен быть т.н. ”машинно-написанным”, т.е. обладать высокой насыщенностью шрифта (и выраженным соединительным штрихом [18] или горизонтальным элементом [17]) . Для большинства графического текста на видео это так. Во-вторых, он должен достаточное время быть статическим на фреймах (чтобы человек имел возможность его прочитать). Выбрано минимальное время, в течение которого текст должен оставаться статичным, равное 0.6 сек). Поэтому различные переходы текста (text transitions) сдетектированы не будут, а сдетектируется лишь уже стабилизированный текст. В-третьих, рассматривается только горизонтальный текст, так как он, как пишет Rainer Lienhart в [10], составляет более 99 процентов всего графического текста на видео, а рассмотрение и вертикального текста существенно ухудшает правильность детекции.

В ходе работы были поставлены следующие подзадачи:

1. Разработка и реализация алгоритма начальной (”грубой”) пространственной(”spatial”) детекции и локализации текста на фрейме

2. Разработка и реализация алгоритма временно́й (”temporal”) детекции и локализации текста на наборе фреймов

3. Разработка и реализация алгоритма уточненной локализации текста на наборе фреймов, где он оставался статичным

4. Тестирование полученного алгоритма и отдельных его составляющих

2. Известные результаты

На сегодняшний день было предложено много методов для решения поставленной задачи. Их можно классифицировать на 3 основные категории, основываясь на особенностях, которые они используют.

Среди методов так называемой группы градиентов, границ и уголков можно выделить подход Rainer Lienhart'a [10], который предложил интегрировать результаты, полученные с помощью многослойной нейронной сети с прямой связью, через которую пропускаются градиенты изображения разных масштабов, в единую карту признаков. Chen в [6] предложил метод машинного обучения (SVM и MLP), который использует контрастно-независимые признаки, и двухэтапную модель локализации и верификации регионов. Hua в [14] использовал детектор SUSAN, чтобы найти уголки и затем идентифицировать регионы с текстом. Методы границ и уголков просты в реализации, но, как правило, испытывают высокий False alarm в случаях со сложным фоном. Также эти методы требуют подбора пороговых значений, что может оказаться недостаточным для всего разнообразия видео.

Еще один подход базируется на классификации так называемых структур (texture-based). Эти методы используют предположение, что текст на изображении проявляет некие структурные особенности, что отличает их от фона. Эта категория методов пытается решить проблему ложных срабатываний (False alarm) и сложных фонов. Подход использует машинное обучение и менее зависим от эвристик. Обычно, чтобы извлечь структурные данные текста на изображении используются фильтр Габора, вейвлеты, быстрое преобразование Фурье и т.д.. Например, Doermann и Kia [9] предложили использовать как особенности ("the features") моменты вторых и третьих порядков в пространстве вейвлетов Хаара и нейронную сеть как классификатор. Jain в [7] предложил использовать фильтр Габора в качестве особенностей, поскольку преобразование Фурье дает лишь глобальную оценку частот на изображении, тогда как фильтр Габора "локализует" частоту на изображении. Как правило, методы, основывающиеся на структурах, используют дорогостоящие особенности изображения (число таких необходимых особенностей велико), а также они чувствительны к типу шрифта, его размеру и похожими на текст особенностям фона.

Среди методов связанных компонент можно выделить подход R. Jiang'a [12]. Сначала изображение разбивается на компоненты связанности, используя кластеризацию по цвету. Чтобы отделить текст от фона используется двухуровневая модель классификации, в которой все компоненты связанности по цвету проходят каскадный классификатор и оставшиеся после этого компоненты классифицируются SVM. Во всех методах, основанных на анализе цвета используется кластеризация. Следовательно, метод связанных компонент хорош, как правило, для детекции однородного текста.

Все вышеназванные методы выполняют пространственный ("spatial") анализ рас-

положения текста на видео, определяя координаты текстовых блоков в рамках системы (x, y). Для графического текста на видео можно рассматривать также временные рамки, в пределах которых субтитры на видео остаются статичными. Это разумно по двум причинам. Во-первых, если, например, рассматривать одночасовое видео с количеством кадров в секунду, равным 30, то придется выполнить обработку 108000 изображений, что является вычислительно дорогим процессом. Намного разумнее трудоемкие операции проводить уже не над каждым фреймом, а лишь раз для всех фреймов, где текст остается статичным. Во-вторых, корреляция текста между фреймами позволяет нам более точно проводить работу на этапе локализации и сегментации текста, ведь мы будем иметь дело с информацией сразу на нескольких фреймах. Xiaoqian Liu в [16] предложил свой способ детекции и сегментации, который учитывает приведенные выше соображения. Однако временная локализация там производится лишь основываясь на результатах уголкового детектора, что на практике не всегда дает точные результаты. Метод Xiaoqian Liu и Weiqiang Wang [11] основывается на предустановеленном для каждого конкретного датасета ожидаемом параметре шрифта, также их алгоритм неоптимально работает на видео больших разрешений и проводит слишком "грубую" локализацию.

3. Предложенный алгоритм

Далее будет описан используемый алгоритм. За основу взят подход, предложенный Xiaoqian Liu и Weiqiang Wang [11]. По сравнению с их работой, данный алгоритм не требует предварительного задания параметров для каждого отдельного датасета. Предложенный ими метод был оптимизирован в части скорости на этапе детекции границ текста (нестатичные контуры текста отфильтровываются еще до вычислительно затратного этапа работы с подконтурами), предложена оптимизация применения детектора уголков Харриса для нашего случая, добавлен способ динамического определения размеров окна для поиска однонаправленных подконтуров (оригинально, размеры окна устанавливались исходя из конкретного датасета вручную), скорректирован способ определения временных границ расположения текста на видео, а также выбран более точный метод для построения “bounding boxes”, базирующийся на основе проекционный данных. Хочется отметить, что ключевым моментом, определяющими хорошую скорость алгоритма, является то, что шаг по локализации текста (и, по необходимости, дальнейшая сегментация) осуществляется не на уровне конкретного фрейма, а на уровне набора фреймов, где текст остается статичен. Также мы рассматриваем всего 3 фрейма в секунду, что намного снижает трудоемкость процесса и позволяет приблизить алгоритм по скорости к “real time”. Кроме того, оба детектора, используемые в алгоритме (Харриса и Канни), основаны на градиентах изображения, которые вычисляются всего один раз. Также предложена оптимизация применения детектора уголков Харриса для нашего случая.

Для реализации алгоритма был выбран язык C++ и библиотека openCV. Выбор данной связки объясняется необходимостью в достаточной гибкости по работе с памятью, максимально эффективной работе отдельных частей алгоритма и наличием некоторых готовых оптимизированных решений.

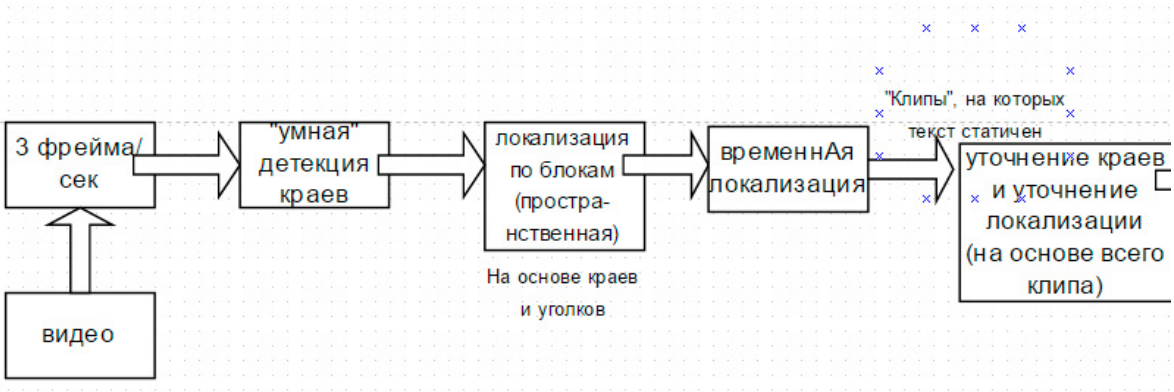

Рис. 3: Схема предложенного алгоритма

На схеме 3 проиллюстрирован подход, использованный в работе. Для каждого фрейма, взятого с частотой 3 фрейма в секунду, мы проводим ”умную” детекцию кра-

ев, оставляя по окончании данного этапа контуры, которые, вероятнее всего, являются текстовыми. Затем мы проводим так называемую "грубую" локализацию на основе краев и уголков, в итоге для каждого фрейма получая информацию, в каких регионах видео присутствует текст. Далее мы проводим временну́ю локализацию, определяя, в каких временны́х рамках текст оставался статичным на видео (будем называть этот промежуток видео клипом). После этого, мы интегрируем всю информацию, содержащуюся на каждом фрейме клипа, чтобы провести более точную пространственную локализацию на основе проекционных данных.

3.1. Начальная ("грубая") пространственная детекция и локализация текста на фрейме

Первым наблюдением является то, что можно рассматривать лишь 3 фрейма в секунду (ведь частота изменения графического текста не так уж и велика), что позволит не только многократно выиграть в скорости, но и использовать корреляцию фона на соседних фреймах (эти изменения будут достаточно существенны, в отличие, например, от того случая, когда бы рассматривались фреймы с большей частотой). Разбиение на кадры производится равномерно по времени, т.е. новый кадр поступает на вход каждые 0.33 секунды. Далее, применяем фильтр Гаусса, чтобы избавиться от мелких шумов. Ядро фильтра выбрано независимо от размера видео - $(3 * 3)$. Применение линейного фильтра относительно быстрая операция. Затем, переводим изображение в grayscale по формуле яркости, используемой в YUV: $y = 0.587 * green + 0.299 * red + 0.114 * blue$. Это позволит нам работать не с тремя каналами по отдельности. После, применяем детектор границ Канни [3], который базируется на градиентах (вычисляемых оператором Собеля всего раз в ходе алгоритма), принципе подавления немаксимумов для устранения удвоения краев и гистерезисе с двумя пороговыми значениями. Более подробно, мы применяем оператор Собеля с ядром размера 3 в вертикальном и горизонтальных направлениях ко всему изображению (формулы 1). Эти оценки производных в двух направлениях будут использоваться и в дальнейшем, так что мы заранее вычисляем их перед применением детектора Канни.

$$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A} \qquad \mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \tag{1}$$

$$\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2} \qquad \Phi = \operatorname{atan}(\mathbf{G}_y, \mathbf{G}_x) \tag{2}$$

Также заранее вычисляется модуль производной и направление градиента (формулы 2). Детектор Канни уточняет ранее вычисленные оператором Собеля границы.

Во-первых, применяется принцип подавления немаксимумов (рассматриваются соседи каждого пикселя и остаются лишь "сильнейшие" граничные пиксели), дабы получить один отклик на границу. Это пригодится на дальнейших этапах. Во-вторых, применяется двухпороговая фильтрация (все пиксели, не превосходящие нижний порог отфильтровываются, превосходящие верхний - называются "сильными", а остальные проверяются на связанность с сильными), в результате которой, несмотря на то, что мы отфильтровываем многие пиксели, мы сохраняем "неразрывность" контура (т.е. т.н. слабые пиксели, значения интенсивности градиента которых лежат между нижним и верхним порогом, но являющиеся 8-связными с "сильными", остаются в качестве границ). Оба вышеназванных свойства детектора играют большую роль в дальнейшем. Пороговые значения для детектора выбраны в "щадящем режиме", т.е. таким образом, чтобы не потерять возможные границы текста даже при низкой его контрастности (в дальнейшем все псевдотекстовые границы будут отфильтрованы). Время работы алгоритма напрямую зависит от количества граничных пикселей, например, 25000 граничных пикселей (с фрейма 1280*720) на машине Intel Core i5 при данных пороговых значениях алгоритм детектирует за 0.03 секунды.

Следующим нашим шагом является получение контуров на основе двоичной карты (единицами отмечены граничные пиксели : $E(x, y) = 1$), полученной на предыдущем этапе с помощью детектора Канни. Для поиска контуров границ выбран алгоритм, предложенный Satoshi Suzuki [13], который в итоге дает маршрут (в том смысле, что все его соседние точки являются 8-связными и в ходе обхода маршрута посещаются все точки границы). Формально,

$$C = \{c^{(i)} \mid i = 0..M\}, \text{ где } c^{(i)} = \langle c_j^{(i)} \;\; j = 0..N \mid E(c_j^{(i)}) = 1 \wedge c_j^{(i)} \text{ 8-связен с } c_{j-1}^{(i)} \;\; \forall j \in \overline{0..N} \rangle \quad \text{и } c^{(i)} \text{ максимально возможное} \tag{3}$$

Данный алгоритм работает вполне эффективно даже при большом числе искомых контуров. Например, на набор из 200 контуров в среднем уходит 0.005 секунды.

Далее, применяем фильтр по всем контурам, целью которого является отсечение нестатических контуров, которые на видео присутствуют менее, чем 0.6 секунды, что соответствует поставленной задаче. Этот фильтр учитывает не только положение контура, но и направление градиента (направления кластеризуются в 4 основных) на нем. Это значительно ускоряет метод, предложенный [11] ввиду того, что дальнейшая работа с подконтурами (которых очень большое число) контуров уменьшится в несколько раз. Более подробно, мы округляем направление градиента $\Phi(x, y)$ в одном из четырех направлений (отсчет углов начинается с направления $0Y$ по часовой

стрелке). Например, направление 1 (север-юг) задается следующим образом:

$$\Phi^1(x,y) = 1 \iff E(x,y) = 1 \wedge \Big((\Phi(x,y) \leq 22.5 \wedge \Phi(x,y) \geq 0) \vee (\Phi(x,y) > 157.5 \wedge \Phi(x,y) \leq 202.5) \vee (\Phi(x,y) > 337.5)\Big) \tag{4}$$

4 основных направления $\Phi^i \;\; i = 1..4$ проиллюстрированы на рис. 4.

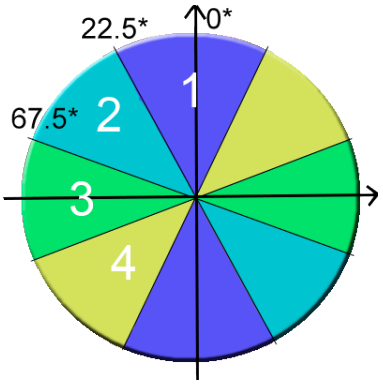

Рис. 4: Основные направления градиента

Далее, через $E_t(x,y)$ обозначим $E(x,y)$ для фрейма t, аналогично C_t и $\Phi^i_t(x,y)$. Удалим из набора C_t все $c^{(i)} \in C_t$, такие что

$$static = \Big|\Big\{c_j^{(i)} \mid c_j^{(i)} \in c^{(i)} \wedge \exists i \;\; : \Big(\Phi^i_t(c_j^{(i)}) = \Phi^i_{t-1}(c_j^{(i)}) \wedge \Phi^i_t(c_j^{(i)}) = \Phi^i_{t-2}(c_j^{(i)})\Big)$$
$$\vee \Big(\Phi^i_t(c_j^{(i)}) = \Phi^i_{t-1}(c_j^{(i)}) \wedge \Phi^i_t(c_j^{(i)}) = \Phi^i_{t+1}(c_j^{(i)})\Big) \vee \Big(\Phi^i_t(c_j^{(i)}) = \Phi^i_{t+1}(c_j^{(i)}) \wedge \Phi^i_t(c_j^{(i)}) = \Phi^i_{t+2}(c_j^{(i)})\Big)\Big\}\Big|$$
$$\frac{static}{\Big|\Big\{c_j^{(i)} \mid c_j^{(i)} \in c^{(i)}\Big\}\Big|} < StaticConst = 0.6$$

Затем применяется простой фильтр, целью которого является отсечение заведомо не соответствующих тексту по масштабу контуров (например, высота которых составляет более одной восьмой всего фрейма). Удалим из набора C_t все $c^{(i)} \in C_t$, такие что:

$$width(c^{(i)}) > maxWidth \qquad height(c^{(i)}) > maxHeight$$
$$width(c^{(i)}) < minWidth \qquad height(c^{(i)}) < minHeight$$

Константы задаются в зависимости от размера фрейма. Следующий этап представляет собой “умный” фильтр контуров, который основывается на свойствах, соответствующих границам субтитров. В частности, для контура текста, каждая линия, составляющая контур, содержит параллельную ей в некоторой окрестности. Более подробно, сначала разбиваем контур на 8-связные подконтуры, градиент каждой точки которых лежит в одном и том же диапазоне(как на рисунке 4) для всего подконтура

(всего 4 основных направления). Таким образом, разбили контур на однонаправленные подконтуры (рис. 5):

$$
\begin{aligned}
&c^{(i)} = \bigcup \{c^{(i),(k)}\} \\
&c^{(i),(k)} = \langle c_j^{(i),(k)} \in c^{(i)} \;\; j = 0..N \mid c_j^{(i),(k)} \;\text{8-связен с}\; c_{j-1}^{(i),(k)} \;\wedge \\
&\Phi^d(c_j^{(i),(k)}) = \Phi^d(c_{j-1}^{(i),(k)}) \;\; \forall d \in \overline{1..4} \;\; \forall j \in \overline{0..N} \rangle \;\text{и}\; c^{(i),(k)} \;\text{максимально возможное}
\end{aligned}
\tag{5}
$$

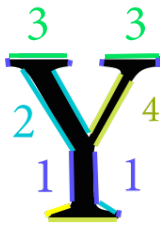

Рис. 5: Подконтуры контура

Далее, создаем окно для каждого из подконтура (причем пропорции окон для различных направлений подконтуров различны). Размер окна зависит от самого контура:

$$
\begin{aligned}
&basedOnStroke = 1.5 * horizontalStroke(c^{(i)}) \\
&basedOnHeight = 0.33 * height(c^{(i)}) \\
&window = min(basedOnStroke, basedOnHeight, maxStrokeSize)
\end{aligned}
\tag{6}
$$

$maxStrokeSize$ задается исходя из размеров фрейма. Величина basedOnStroke оценивает ширину горизонтального штриха контура, чтобы размеры окна была пропорциональны этому штриху. Величина basedOnHeight ограничивает размер окна исходя из высоты контура. В отличие от [11], где размер окна задается для каждого датасета вручную, мы задаем его на основе свойств самого изначального контура. Назовем $c^{(i)(d)}$ - множество подконтуров $c^{(i)(d)(k)}$ контура $c^{(i)}$, все пиксели которых лежат в направлении $d \in \overline{1..4}$. Далее, мы задаем окно поиска параллельных пикселей для подкон-

тура $c^{(i)(d)(k)}$ контура $c^{(i)(d)}$ следующим образом (обозначив $window2 = 2 * window/3$):

$$neighbor = \begin{cases} \{(x,y) \mid x(c_j^{(i)(d)(k)}) - window \le x \le x(c_j^{(i)(d)(k)}) + window \\ \qquad , \;\; y = y(c_j^{(i)(d)(k)}) \quad \text{для некого} \;\; c_j^{(i)(d)(k)} \in c^{(i)(d)(k)}\}, & \text{if } d = 1 \\ \{(x,y) \mid (x,y) \in \Big[(x(c_j^{(i)(d)(k)}) - window2, y(c_j^{(i)(d)(k)}) - window2); \\ \qquad (x(c_j^{(i)(d)(k)}) + window2, y(c_j^{(i)(d)(k)}) + window2)\Big] \\ \qquad \text{для некого} \;\; c_j^{(i)(d)(k)} \in c^{(i)(d)(k)}\}, & \text{if } d = 2 \\ \{(x,y) \mid y(c_j^{(i)(d)(k)}) - window \le y \le y(c_j^{(i)(d)(k)}) + window \\ \qquad , \;\; x = x(c_j^{(i)(d)(k)}) \quad \text{для некого} \;\; c_j^{(i)(d)(k)} \in c^{(i)(d)(k)}\}, & \text{if } d = 3 \\ \{(x,y) \mid (x,y) \in \Big[(x(c_j^{(i)(d)(k)}) - window2, y(c_j^{(i)(d)(k)}) + window2); \\ \qquad (x(c_j^{(i)(d)(k)}) + window2, y(c_j^{(i)(d)(k)}) - window2)\Big] \\ \qquad \text{для некого} \;\; c_j^{(i)(d)(k)} \in c^{(i)(d)(k)}\}, & \text{if } d = 4 \end{cases}$$

Далее, для всех подконтуров контура ищем лежащие в пределах окна и параллельные данному подконтуру линии. При недостаточном числе найденных подобных точек, весь контур отфильтровывается. Далее полученный контур будем называть “контур текста”. Более подробно, зададим число найденных в окрестности пикселей с нужным градиентом.

$$matches^{(i)(d)(k)} = \Big|(x,y) \in neighbor^{(i)(d)(k)} \mid \Phi^d(x,y) = 1\Big| \tag{7}$$

Следующее условие будет условием для удаления контура $c^{(i)}$ из C_t:

$$\frac{\sum\limits_{k,d} matches^{(i)(d)(k)}}{\big|\{c_j^{(i)} \in c^{(i)}\}\big|} < minParallelPixels = 0.4 \tag{8}$$

На предыдущем этапе мы получили уточненные контуры текста. Далее, будем использовать их как во временном анализе расположения текста на видео, так и в пространственном. Для начала, дробим фрейм на прямоугольные блоки, размеры которых зависят от размеров фрейма. Эти блоки вскоре послужат первым приближением к итоговым регионам. Обозначим за $R^{(i),(j)}$ регион с координатами (i,j). Размеры блоков зададим следующим образом:

$$ratio = frameW/frameH$$
$$zoneHeight = frameH/8 \qquad zoneWidth = (frameH/8 * ratio)$$

Затем, применяем детектор уголков Харриса [5]. Графический текст, какой бы стиль написания не применялся, имеет множество уголков. Детектор базируется на собственный числах матрицы Харриса, которая в свою очередь вычисляется с помо-

щью градиентов, уже вычисленных ранее. Поэтому применение данного детектора обойдется довольно дешево. Рассмотрим

$$A(x,y) = \sum_{u,v \in win(x,y)} w(u,v) \begin{bmatrix} G_x^2(u,v) & G_x G_y(u,v) \\ G_x G_y(u,v) & G_y^2(u,v) \end{bmatrix} \tag{9}$$
$$w(x,y) \equiv 1$$

Выбран размер окна $win(x,y)$, равный $(4 * 4)$, весовая функция - тождественная единица. Автокорреляционная матрица A отражает величину автокорреляции при малом сдвиге окна во всевозможных направлениях. Угол характеризуется большими изменениями функции $A(x,y)$ по всем возможным направлениям, что эквивалентно большим по модулю собственным значениям матрицы A. Находим максимум $V(x,y) = \det(A) - k * \mathrm{trace}^2(A)$ при $k = 0.08$ по всем (x,y), обзоначим его за $maxharris$. Назовем пиксель (x,y) уголком $(U(x,y) = 1)$, если $V(x,y) \geq 0.04 * maxharris$. Данная стратегия объясняется тем фактом, что если на фрейме присутствует хотя бы строка графического текста, то все остальные строки будут обладать сравнимой с ним контрастностью, таким образом может быть отфильтровывано множество нетекстовых блоков. Побочной стороной детектора является его невыдающаяся скорость - в среднем 0.07 секунды на фрейме размером $1280 * 720$. Такая скорость объясняется тем, что мы 3 раза применяем свертку с усредняющим фильтром. Однако, если сравнивать фильтр по критерию точность/быстродействие, это - один из лучших вариантов. К тому же, чуть ниже будет описан способ оптимизации применения детектора в нашем случае.

Далее, для каждого прямоугольного блока проверяем, достаточно ли он содержит "уголков" и пикселей контура текста, полученного на прошлом этапе . Все блоки, прошедшие этап, помечаются как текстовые (R^{text}):

$$R^{(i),(j)} \in R^{text} \iff \begin{cases} \dfrac{\left|(x,y) \mid (x,y) \in R^{(i),(j)} \wedge \exists m : (x,y) \in c^{(m)}\right|}{\left|(x,y) \mid (x,y) \in R^{(i),(j)}\right|} \geq 0.01 \\ \dfrac{\left|(x,y) \mid (x,y) \in R^{(i),(j)} \wedge U(x,y) = 1\right|}{\left|(x,y) \mid (x,y) \in R^{(i),(j)}\right|} \geq 0.002 \end{cases} \tag{10}$$

Данное приближение является начальным и служит большую роль для этапа временно́й локализации. На дальнейших этапах локализация текста будет уточнена. Как уже упоминалось, возможна оптимизация применения детектора Харриса. Так как многие блоки содержат недостаточно пикселей контура(исходя из 16), мы можем применять детектор Харриса лишь на регионах, прошедших эту проверку. Так как текстовых блоков, как правило, менее половины от общего числа, это ускорит процесс более чем вдвое: на тех фреймах, где текстовые блоки составляли 25 процентов,

получена скорость работы детектора в среднем 0.014 секунды на фрейм (на том же датасете, где изначально было 0.07 секунды).

3.2. Временна́я детекция и локализация текста на наборе фреймов

Временна́я локализация устанавливает временны́е границы видео, в пределах которых текст остается статичным. Точность на данном этапе напрямую зависит от точности на предыдущем, так здесь будет использоваться полученная ранее информация о распределении контуров. Основная проблема заключается в том, что алгоритм должен допускать небольшие искажения контуров на тех фреймах, где текст остается статичным (например из-за смены яркости фона), но при этом реагировать на появление новых текстовых блоков, ведь иначе текст будет неправильно локализован на всех фреймах "клипа" (набор фреймов, где текст остается статичным). Для достижения поставленной цели был выбран следующий подход. Мы сравниваем текущий фрейм с последним фреймом последнего имеющегося "клипа". Если результат сравнения удовлетворяет определенному критерию, мы добавляем фрейм в "клип", иначе создаем новый "клип" с этим фреймом в начале. Более подробно, обозначим $D(x,y) = i \in \overline{1..4}, \text{if } \Phi^i = 1 \wedge \exists m \mid (x,y) \in c^{(m)}, \text{else } 0$. Тогда отметим блок как изменившийся, если

$$R_t^{(i),(j)} \in R_{new}^{text} \iff \frac{\left|\{(x,y) \mid D_t(x,y) = D_{t-1}(x,y)\}\right|}{\left|\{(x,y) \mid D_t(x,y) \neq 0\}\right|} < 0.6 \tag{11}$$

Тогда условием того, что фрейм не входит в прошлый "клип" будет следующим:

$$\begin{aligned} newPixels &= \left|\{(x,y) \in R_{new}^{text} \mid D_t(x,y) \neq 0 \wedge D_t(x,y) \neq D_{t-1}(x,y)\}\right| \\ allPixels &= \left|\{(x,y) \in R_{new}^{text} \mid D_t(x,y) \neq 0\}\right| \\ F_t \in clip^i &\iff \frac{newPixels}{allPixels} > 0.1 \end{aligned} \tag{12}$$

Аналогично делаем симметричные проверки, меняя местами t и $t-1$ в формулах 11 и 12. Если условие в одну из сторон не выполняется - фрейм не входит в предыдущий "клип" и образует новый.

3.3. Уточненная локализация текста на наборе фреймов

На предыдущем этапе мы получили временны́е границы, в которых текст оставался статичным. Для всех накопленных в буфере фреймов (*clip*), где текст оставался статичным, обобщаем информацию о текстовых контурах путем применения порога относительно количества фреймов, на которых данный пиксель являлся граничным

текстовым . Если число таких фреймов больше порога, данный пиксель будем считать текстовым на результирующем изображении (далее - текстовый пиксель).

$$T(x,y) = 1 \iff \frac{\left|\{t \mid D_t(x,y) \neq 0 \ \ \wedge F_t \in clip\}\right|}{\left|\{F_t \in clip\}\right|} > 0.65 \tag{13}$$

Аналогичную процедуру проделываем с текстовыми блоками: преодолевшие порог блоки также будут присутствовать на результирующем изображении (далее - текстовые блоки).

$$P^{(i),(j)} \in P^{text} \iff \frac{\left|\{t \mid R_t^{(i),(j)} \in R_t^{text} \wedge F_t \in clip\}\right|}{\left|\{F_t \in clip\}\right|} > 0.65 \tag{14}$$

В отличие от [11] (где наше “грубое приближение” блоками считалось окончательным) здесь предлагается более точный способ локализации текста, основанный на проекционных данных ранее найденных текстовых пикселей. Далее объединяем текстовые блоки на результирующем изображении, которые являются 4-связными. Далее будем называть их текстовыми регионами:

$$Reg^{(k)} = \{P^{(i),(j)} \mid P^{(i),(j)} \ \text{4-связны}\} \tag{15}$$

Предложенный алгоритм может разделять строки с текстом, расположенные даже на одной высоте. Введем горизонтальные и вертикальные проекционные функции следующим образом:

$$F_{vert}(x,y) = \sum_{i=0}^{x} T(i,y) \qquad F_{horiz}(x,y) = \sum_{j=0}^{y} T(x,j) \tag{16}$$

Затем аппроксимируем текстовый региона минимальным объемлющим прямоугольником:

$$Rect^{(k)} = \min\{\text{Rectangle} \mid \text{Rectangle} \supseteq Reg^{(k)}\} \tag{17}$$

Этот прямоугольник послужит начальным приближением для проекционного процесса. В листинге 1 приведен пример псевдо-кода проекционного процесса. Чередуем процесс вертикальной и горизонтальной проекции до тех пор, пока число получающихся ограничивающих прямоугольников не перестанет расти.

Listing 1: проекционный процесс

```
out = verticalProjection(initialBoxes);
out = horizProjection(out);
int num = 0;
bool odd = true;
```

```
while (out.size() != num){
    num = out.size();
    out = odd ? verticalProjection(out) : horizProjection(out);
    odd =!odd;
}
```

Далее опишем процесс вертикальной проекции, необходимый для отделения строк друг от друга. Пусть на входе мы имеем $Rect_i^{(k)}$, где i - номер итерации вертикального процесса.Так как начальное "грубое" приближение регионов, возможно, включало не весь текст, мы определим понятие отступа, если $i = 1$. Пусть $y1$ и $y2$ - начальные координаты текстового региона $Rect_i^{(k)}$. Если $i = 1$, то расширим вертикальные границы региона $[y1; y2]$ на отступ $padding$: $[y1' = y1 - padding; y2' = y2 + padding]$, где $padding = (y2 - y1)/3$, иначе $y1' = y1$, $y2' = y2$. Далее высчитываем проекционные значения на основе проекционной функции $F_{vert}(x, y) : projVal[i] = F_{vert}(y2' + i, x2) - F_{vert}(y2' + i, x1)$, где $x1$ и $x2$ - начальные координаты текстового региона $Rect_i^{(k)}$. Затем мы применяем усредняющий фильтр к $projVal = meanFilter(projVal)$ с окном в 3 пикселя: это необходимо, так как порой происходят заметные скачки, объяснимые тем, что непрерывность контура текста иногда нарушается. Стоит отметить, что в случае горизонтальной проекции, размер окна установлен как $3 * (y2 - y1)$, так как мы допускаем пробелы между словами, локализуя всю строку целиком (размер пробела составляет не более трех высот символа). Затем мы находим локальные максимумы на $projVal$, без учета отступа (так как отступ служит лишь для того, чтобы мы "продолжили" границу региона на тот диапазон, а не для того, чтобы искать там самостоятельные текстовые регоины). Обозначим полученные максимумы за $maximus^{(i)}$. Затем отфильтруем те из них, значение которых меньше $0.15 * \max_i(maximus^{(i)})$.

Далее, будем искать "впадины" на гистограмме проекционных данных $projVal$, чтобы выявить потенциальные границы строк. Для этого применим метод расширяющихся границ интервала. Суть его состоит в следующем. Сначала сортируется набор $maximus$. Затем для каждого $maximus^{(i)} \in maximus$, еще не входящего ни в один выходной интервал в $outInterv$, выполняется расширение левой границы (листинг 2), пока проекционное значение в данной точке достаточно велико и точка не входит в уже имеющийся интервал в $outInterv$:

Listing 2: расширение левой границы

```
double check = max.val * minimumIndicator;
unsigned int cur = max.Idx - 1;
while (cur > 0 && projValues[cur] >= check && !contains(outInterv, cur)){
    cur--;
}
left = cur
```

Аналогично выполняется расширение правой границы $right$, и интервал $[left; right]$ заносится в $outInterv$. Все выходные значения из $outInterv$ декартово умножаются на $[x1; x2]$ и результат заносится в выходной out для всего вертикального проекционного процесса.

3.4. Тестирование алгоритма и отдельных его составляющих

По имеющимся у нас и [11] данным, нет хорошей современной публичной базы для тестирования детекции и локализации графического текста. Чтобы при тестировании соблюсти ограничения, накладываемые на текст исходя из задачи, вся выборка видео должна быть вручную размечена. Также стоит отметить, что тестирование проходит как по временным границам расположения текста на видео (так как этот этап влияет на все последующие), так и по пространственным. Кроме того, должна быть измерена вычислительная производительность детекции.

Начальное тестирование алгоритма проводилось на четырех видео, все в разрешении $1280 * 720$, чтобы лучше протестировать вычислительную эффективность метода, два из которых(1-й и 4-й) являются отрывками из фильмов с вшитыми субтитрами, пояснительными надписями и титрами, а два(2-й и 3-й, пример на рисунке 6) - новостные ролики с двух разных каналов. Вся разметка видео была сделано вручную, что, конечно, очень трудоемко. Результаты эксперимента приведены в таблице 1, где указаны следующие величины для пространственной и временно́й локализации:

$$\text{recall rate} = \frac{\text{верно локализованных строк}}{\text{реально верных}}$$
$$\text{precision rate} = \frac{\text{верно локализованных строк}}{\text{всего локализованных}} \tag{18}$$

Видео	Длит-ь, сек	$t_{\text{работы}}$, с/фрейм	precision простр.,%	recall простр.,%	precision врем.,%	recall врем.,%
1	729	0.275	2343/2427 = 0.965	2343/2477 = 0.946	335/357 = 0.941	335/349 = 0.966
2	593	0.314	7621/7746 = 0.983	7621/7836 = 0.972	66/70 = 0.942	66/72 = 0.917
3	536	0.297	4437/4521 = 0.980	4437/4599 = 0.964	76/89 = 0.853	76/81 = 0.938
4	756	0.281	2527/2629 = 0.961	2527/2648 = 0.954	369/381 = 0.969	369/378 = 0.976

Таблица 1: Результаты, полученные на 4-х роликах

Для временно́й локализации рассматриваются 3 случая : переход от фрейма, содержащему текст, к фрейму, содержающему другой текст; от фрейма без текста к фрейму с текстом; от фрейма с текстом к фрейму без текста. Для пространственной локализации, текст называется верно локализованным, если отношение площади пересечения прямоугольника, определенного алгоритмом, и прямоугольника, являющегося истинной локализацией, к площади их объединения составляет не менее 90% .

Рис. 6: Пример локализации на одном из новостных видео

Рис. 7: Пример локализации на видео с китайскими и английскими символами

Далее результат работы алгоритма был сравнен с результатами работ [11] и работами [16], [15]. Для тестирования временно́й локализации авторами [11] было выбрано видео из 12888 (каждый фрейм взят с интервалом в 0.33 секунды) фреймов с разрешением 1024*576, включающее пунктуационные символы, цифры, китайские и английские символы (рис. 7). Мы также протестировали алгоритм на этом видео. Из таблицы 2 видно, что предложенный алгоритм превосходит два из четырех метода, но уступает методу [11], так как там используются предустановленные для каждого конкретного датасета параметры ожидаемого шрифта. Также реализация алгоритма [11] была сравнена с нашей в части скорости на нескольких больших видео различных разрешений на процессоре Intel Core i5. Видно, что скорость нашего алгоритма значительно превосходит скорость работы [11].

Метод	precision врем.,%	recall врем.,%
Tang [15]	0.909	0.947
Wang 1 [16]	0.910	0.930
Wang 2 [11]	0.995	0.999
Предлож.	0.963	0.969

(a) Временна́я локализация на видео 1024*576

Метод	Видео	Число видео	$t_{\text{работы}}$, с/фр.
Wang 2 [11]	1280*720	3	0.382
Предл.			0.281
Wang 2 [11]	1024*576	3	0.257
Предл.			0.184
Wang 2 [11]	800*432	2	0.153
Предл.			0.112

(b) Производительность методов

Таблица 2: Сравнение методов

Также, как и в [11], мы провели пространственную локализацию на 100 "клипах" (каждый порожден нашим временны́м локализатором и каждый фрейм которого либо содержит один и тот же текст, либо не содержит ничего), 50 - отрывки из фильмов, 50 - из новостных роликов, все в разных разрешениях до $1280 * 720$ включительно. Результаты сведены в таблицу 3 .

Метод	precision простр.,%	recall простр.,%
Wang 2 [11]	0.952	0.968
Предл.	0.940	0.970

Таблица 3: Сравнение методов, пространственная локализация на "клипах"

Можно заметить, что наш алгоритм порождает больше ложных срабатываний, однако чуть реже "пропускает" истинно текстовые блоки. Это связано с динамическим определением размера окна поиска на этапе фильтрации контура. Размеры окна поиска для некоторых, похожих на текстовые, контуров слишком велики, что препятствует их фильтрации.

Заключение

В ходе работы были выполнены следующие задачи:

- Были разработаны и реализованы на языке C++ с использованием OpenCV алгоритмы временно́й и пространственной детекции локализации текста
- Алгоритм был протестирован на собственных датасетах с высоким разрешением видео
- Результаты работы алгоритма были сравнены с результатами других работ на основе их датасета
- Данная работа была представлена на конференции "Коллоквиум молодых исследователей по организации информации и системному программированию (CIMSP)". Тезисы будут опубликованы в сборнике материалов конференции

В целом, хочется отметить, что помимо детекции и локализации текста, данный алгоритм дает возможность для дальнейшей более точной сегментации текста на видео, ведь в результате работы помимо точных пространственных границ текста, мы получаем временные рамки появления текста, что в несколько раз может повысить точность сегментации путем использования корреляции текста на тех фреймах, где он остается статичным.

Мы добились неплохой скорости работы данного решения, что позволяет использовать его даже на видео высоких разрешений. Все это еще раз демонстрирует, что использование пространственно-временного анализа является основополагающим принципом достижения хорошей вычислительной эффективности для детекции и локализации графического текста на видео.

Список литературы

[1] Bahman Zafarifar Jingyue Cao, de Peter H. N. Instantaneously Responsive Subtit le Localization and Classification for TV Applications. — 2011. — Vol. 57 (Issue 1) of EEE International Conference on Consumer Electronics. — IEEE Xplore Digital Library : http://goo.gl/Iy7WxI.

[2] Boris Epshtein Eyal Ofek, Wexler Yonatan. Detecting text in natural scenes with stroke width transform. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2010. — 2010. — IEEE Xplore Digital Library : http://goo.gl/IK4EeK.

[3] Canny John. A Computational Approach to Edge Detection. — 1986. — Vol. PAMI-8 (Issue 6) of Pattern Analysis and Machine Intelligence, IEEE Transactions. — IEEE Xplore Digital Library : http://goo.gl/4KfQAc.

[4] Chen Xiangrong, Yuille Alan L. Detecting and reading text in natural scenes. — 2004. — Vol. 2 of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. — IEEE Xplore Digital Library : http://goo.gl/0QCkPv.

[5] Chris Harris Mike Stephens. A combined corner and edge detector. — 1988. — http://www.bmva.org/bmvc/1988/avc-88-023.pdf.

[6] Datong Chen Jean-Marc Odobez Jean-Philippe Thiran. A localization/verification scheme for finding text in images and video frames based on contrast independent features and machine learning methods. — 2004. — Vol. 19 (Issue 3) of Signal Processing: Image Communication, 2004. — Idiap Research Institute : http://goo.gl/K8wVEI.

[7] Jain A. K., Bhattacharjee S. Text segmentation using Gabor filters for automatic document processing. — 1992. — Vol. 5 (Issue 3) of Machine Vision and Applications - Special issue: document image analysis techniques. — http://goo.gl/tvOTaC.

[8] Jiamin Xu Palaiahnakote Shivakumara Tong Lu-Trung Quy Phan-Chew Lim Tan. Graphics and Scene Text Classification in Video. Pattern Recognition (ICPR), 2014 22nd International Conference. — 2014. — IEEE Xplore Digital Library : http://goo.gl/SS6uNs.

[9] Li H Doermann D, O Kia. Automatic text detection and tracking in digital video. — 2002. — Vol. 9 (Issue 1) of Image Processing, IEEE Transactions. — IEEE Xplore Digital Library : http://goo.gl/Zo4DvL.

[10] Lienhart Rainer, Wernicke Axel. Localizing and Segmenting Text in Images and Videos. — 2002. — Vol. 12 (Issue 4) of EEE transactions on circuits and systems for video technology, 2002. — IEEE Xplore Digital Library : http://goo.gl/bTxmnZ.

[11] Liu Xiaoqian, Wang Weiqiang. Robustly Extracting Captions in Videos Based on Stroke-Like Edges and Spatio-Temporal Analysis. — 2012. — Vol. 14 (Issue 2) of Multimedia, IEEE Transactions. — IEEE Xplore Digital Library : http://goo.gl/SM3psv.

[12] Renjie Jiang Feihu Qi Li Xu, Wu Guorong. Detecting and segmenting text from natural scenes with 2-stage classification. — 2006. — Vol. 2 of ISDA '06: Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications. — IEEE Xplore Digital Library : http://goo.gl/ZUhHKq.

[13] Suzuki Satoshi. Topological Structural Analysis of Digitized Binary Images by Border Following. — 1985. — Vol. 30 (Issue 1) of Computer Vision, Graphics, and Image Processing. — http://www.sciencedirect.com/science/article/pii/0734189X85900167.

[14] Xian-Sheng Hua Xiang-Rong Chert Liu Wenyin Hong-Jiang Zhang. Automatic Location of Text in Video Frames. — 1998. — Vol. 2 of Pattern Recognition, 1998. Proceedings. Fourteenth International Conference. — IEEE Xplore Digital Library : http://goo.gl/Pe9A1V.

[15] Xiaoou Tang Xinbo Gao Jianzhuang Liu HongJiang Zhang. A spatial-temporal approach for video caption detection and recognition. — 2002. — Vol. 13 (Issue 4) of Neural Networks, IEEE Transactions. — IEEE Xplore Digital Library : http://goo.gl/V7KAt9.

[16] Xiaoqian Liu Weiqiang Wang. Extracting Captions from Videos Using Temporal Feature. — 2010. — Vol. 14 (Issue 2) of Multimedia, IEEE Transactions on (Volume:14 , Issue: 2). — IEEE Xplore Digital Library : http://goo.gl/4O8LQb.

[17] paratype.ru. Горизонталь, Горизонтальный элемент (Arm). — http://www.paratype.ru/help/term/terms.asp?code=20.

[18] paratype.ru. Соединительный штрих (Hairline). — http://www.paratype.ru/help/term/terms.asp?code=155.

[19] tweakguides.com. Image Retention, Burn-in and Dead Pixels. — http://www.tweakguides.com/HDTV$_8$.*html*.