САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И СИСТЕМ

Чиликин Александр Сергеевич

Выпускная квалификационная работа бакалавра

Автоматическое построение панорамного изображения по видеоряду

Направление 010400

Прикладная математика и информатика

Научный руководитель,
кандидат физ.-мат. наук,
доцент
Погожев С. В.

Санкт-Петербург
2016

Содержание

Введение

Панорамная фотосъёмка – разновидность съёмки с большим углом охвата сцены. Применяется в случае, когда стандартные углы обзора объективов недостаточны для отображения всей требуемой сцены.

С помощью панорамных методик мы получаем возможность «поймать в кадр» больше пространства, чем при использовании стандартного комплекта фототехники. Вопросами панорамной съёмки фотографы озабочены давно. И если раньше усилия конструкторов фототехники, профессиональных фотографов и любителей были направлены на аппаратные решения, то теперь более перспективными видятся решения программные.

Рассмотрим несколько примеров. Рисунок 1 представляет собой фотографию объекта, выполненную широкоугольным объективом (28 мм) с максимально возможного удаления. По условиям съёмки дальше отойти невозможно. Ширины обзора не хватает, виден лишь фрагмент. Выход из ситуации – панорама, которую можно видеть на рис. 2. Похожие условия возникают при фотосъёмке интерьеров и архитектуры. Необходимо показать широкое помещение при условии явного недостатка пространства (рис. 3).

Рис. 1

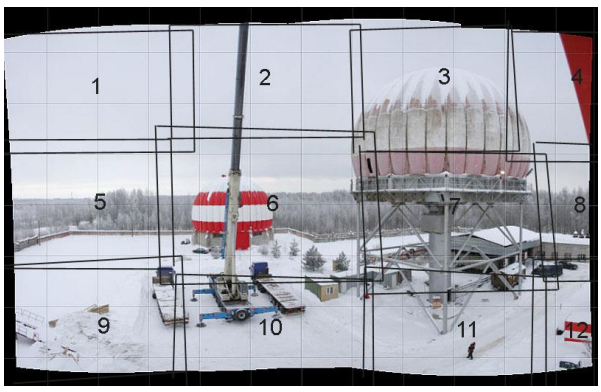

Рис. 2

Рис. 3. Комната

Восхитительной красоты фасады зданий зачастую просто неоткуда сфотографировать – дальше стены дома напротив не отойдёшь при всем желании (рис. 4).

Рис. 4. Зимний дворец

Создание панорам высокого и сверхвысокого разрешения для печати на многометровых баннерах – рисунок 5.

Рис. 5. Благовещенский мост

Рис. 6. Яндекс Панорамы

Другое применение панорамной съёмки – в онлайн сервисах Яндекс Панорамы и Google Street View, которые позволяют совершить виртуальное путешествие по сотням городов, осмотреть достопримечательности, сориентироваться в незнакомом районе, посмотреть на развязки и дорожные знаки (рис. 6).

Наконец, современная видеотехника позволяет снимать видео в высоком разрешении, которое также можно использовать для создания панорам. В этом отношении ныне весьма популярны экшн-камеры вроде GoPro.

Стоит отметить, что получение итогового изображения – достаточно долгий и трудоемкий процесс, как при ручном, так и автоматическом выполнении. В любом случае приходится иметь дело с оценкой оптимальных параметров съемки, перспективными искажениями, переменой освещения, перемещающимися объектами и многим другим. При программном решении задачи привлекаются методы обработки изображений и компьютерного зрения, которые, как правило, требуют большие вычислительные затраты.

Постановка задачи

Предположим, что у нас имеется некий летательный аппарат, например, квадрокоптер, перемещающийся на некоторой **постоянной** высоте. На нем установлена видеокамера, снимающая всё происходящее внизу. Съемка может производиться как в помещении, так и на открытой местности, поэтому нельзя исключать влияние на квадрокоптер внешних факторов – прежде всего ветра. Однако будем считать, что эти факторы не оказывают очень сильного воздействия на наш аппарат. Условимся также, что на видео обязательно присутствуют какие-либо различимые предметы, например, крыши домов. Полет над зеленым лугом или заснеженным белым полем, а также им подобные – исключим из рассмотрения.

На входе мы имеем видеофайл, на котором запечатлена карта местности, и нам хотелось бы построить панорамное изображение по этому видеофайлу.

Таким образом, появляется задача построения панорамного изображения по видеоряду. При этом желательно, чтобы программа:

- работала вне зависимости от разрешения видео, т. е. выдавала желаемый результат как на видео с малым разрешением, так и большим;
- вела себя устойчиво по отношению к небольшим внешним возмущениям, т. е. эти возмущения не оказывали бы сильного влияния на результат;
- имела в своей основе достаточно простой алгоритм, дабы избежать большого времени обработки, учитывая вычислительную емкость применяемых методов.

Обзор литературы

Поставленной задаче в целом посвящено достаточное количество литературы. Однако нельзя не отметить, что, как правило, она уже требует от читателя определенной подготовки.

Для начала отметим несколько фундаментальных трудов. Хорошим введением в область обработки изображений являются книги [1] и [2]. Первая носит исключительно теоретический характер, вторая – более практический. В обеих доступным образом представлены базовые понятия и наиболее популярные методы, касающиеся фильтрации в пространственной и частотной области, сжатия, морфологической обработки и сегментации. Обзорными, но в то же время очень хорошими книгами по компьютерному зрению являются [3], [4]. В них поверхностно затрагиваются все основные подобласти: нахождение и сопоставление особых точек, построение 3D структуры по набору изображений, оценка движения, распознавание. В частности, глава 9 книги [3] посвящена теме данной работы. В обеих книгах содержится огромное количество ссылок на научные статьи, посвященные соответствующим темам. Разобраться во всех геометрических аспектах работы с камерами, изображениями и видео поможет книга [4].

С отдельными вопросами поставленной задачи можно ознакомиться в упомянутых трудах. Однако основная литература по данной и смежным темам представлена многочисленными трудами конференций, среди которых можно отметить ICCV, ECCV, CVPR и российскую GraphiCon, и статьями в журналах, например, IJCV. Отметим несколько популярных статей. Статья [5] содержит общее видение проблемы и предлагает основные методы для ее решения. В [6] рассматривается вопрос об автоматическом выделении нескольких панорам из одного общего набора изображений. Работа [7] детализирует и дополняет [6] парой шагов. Статья [8] рассматривает задачу построения круговой панорамы по видео. Также много статей посвящается

отдельным частям проблемы – регистрации, устранению искажений камеры, устранению проблем, связанных с непостоянностью освещения и т. п.

Глава 1. Общая схема решения

В общем виде предлагаемое решение описанной задачи можно сформулировать следующим образом.

В самом начале требуется извлечь из видеофайла достаточное количество кадров. Это можно делать «на ходу», последовательно читая все фреймы и выбирая отдельные из них с нужной частотой, либо на предварительном этапе собрать набор последовательных кадров. В обоих случаях нужно обладать информацией о частоте кадров в секунду.

Затем, последовательно перебирая по два кадра из набора, необходимо произвести поиск и описание особых точек на этих кадрах. Именно эти особые точки позволят нам «приклеить» следующий кадр к предыдущему. О том, что такие особые точки изображения и как производится их описание, будет сказано в следующей главе на примере одного из таких методов.

Следующим шагом является сопоставление особых точек на основе их описания. Следует иметь в виду, что при этом не исключено получение ложных соответствий, даже при использовании некоторых эвристик.

Затем, имея два набора особых точек, следует найти преобразование, которое переводило бы точки следующего кадра в соответствующие точки предыдущего. При этом теоретически в зависимости от априорного знания о движении камеры можно заранее выбирать конкретный вид преобразования. Однако на практике такой подход, разумеется, не всегда позволяет получить приемлемый результат. Также стоит отметить, что поскольку наборы велики, и их размер явно превосходит минимально необходимый, возникает вопрос о поиске оптимального в некотором смысле преобразования.

После получения нужного преобразования имеет место техническая процедура склейки изображений и устранения видимых швов, если таковые имеются.

Глава 2. Используемые методы.

В этой главе будет приведено описание различных алгоритмов и методов, используемых в данной работе.

2.1. Особые точки

Особые точки (в разных источниках – features/characteristic points/local feature points/interest point/локальные особенности) – говоря неформально – “хорошо различимые” фрагменты изображения. Это точки (пиксели) с характерной (особой) окрестностью, т. е. отличающиеся своей окрестностью от всех соседних точек. Классический пример локальной особенности – вершина угла. Такие точки описываются вектором признаков, вычисляемых на основе интенсивности/градиентов и других характеристик точек окрестности. Используя особые точки, можно анализировать как изображения целиком, так и объекты на них. Хорошие характерные точки позволяют справиться с изменением масштаба, ракурса и перекрытиями сцены или объекта. Такие точки, очевидно, должны удовлетворять следующим требованиям:

- повторяемость – должны находиться в одном и том же месте сцены независимо от ракурса и освещения;
- локальность – должны быть расположены в малой области и не страдать от перекрытий;
- компактность – по сравнению с общим числом пикселей изображения их должно быть в разы меньше;
- значимость (уникальность) – каждая особенность должна иметь свое собственное описание (чтобы была возможность различать точки между собой).

Дескрипторы (алгоритмы, описывающие каждую особую точку в виде числового вектора признаков) особых точек должны быть:

- простыми – представление должно быть быстро вычислимым;

- уникальными – разные точки должны иметь разное представление;
- локальными – как и сама особая точка ее представление должно зависеть лишь от небольшой окрестности;
- инвариантными к максимально большему числу преобразований.

За пару последних десятилетий было разработано несколько хорошо зарекомендовавших себя на практике алгоритмов: SIFT, SURF, FAST, GLOH, HOG, KAZE. При решении поставленной задачи можно использовать любой из них. Нужно лишь подобрать его параметры таким образом, чтобы он искал относительно небольшое число особых точек и работал быстро, поскольку это трудоемкая операция. В ходе решения задачи использовались алгоритмы SURF, ORB и AKAZE. Чтобы стало понятнее, как именно с технической точки зрения производится поиск и описание особых точек, приведем краткое описание алгоритма SURF [9].

Метод SURF инвариантен к масштабу, поворотам в плоскости изображения, зашумленности, перекрыванию другими предметами, изменению яркости и контраста. Он ищет особые точки с помощью матрицы Гессе

$$H(x, y, \sigma) = \begin{bmatrix} L_{xx}(x, y, \sigma) & L_{xy}(x, y, \sigma) \\ L_{xy}(x, y, \sigma) & L_{yy}(x, y, \sigma) \end{bmatrix},$$

и ее определителя (также являющегося матрицей), который достигает экстремума в точках максимального изменения градиента яркости и хорошо детектирует пятна, углы и края линий. Здесь $L_{xx}(x, y, \sigma)$ – свертка изображения с лапласианом от гауссиана $\frac{\partial^2}{\partial x^2} g(\sigma)$ с параметром σ. Аналогично для $L_{xy}(x, y, \sigma)$ и $L_{yy}(x, y, \sigma)$. Однако гессиан не инвариантен относительно изменения масштаба, поэтому SURF использует разномасштабные фильтры для нахождения гессианов. При этом обычно производится пороговое отсечение с целью выделения «хороших» особенностей и контроля числа особых точек, возвращаемых алгоритмом. Вычисление матрицы должно производиться сверткой исходного

изображения с фильтрами, изображенными на рис. 7, однако алгоритм использует приближения фильтров на рис. 8, которые на практике имеют близкие к «настоящим» значения, но вычисляются гораздо быстрее при помощи интегрального представления. При этом вычисляется $\det(H_{approx}) = D_{xx}(x,y,\sigma)D_{yy}(x,y,\sigma) - 0.9^2 D_{xy}^2(x,y,\sigma)$, где буквой D обозначены фильтры на рисунке 8, а $\sigma = 1.2\frac{\text{размер фильтра}}{9}$ называется шкалой. Коэффициент 0.9 имеет теоретическое значение. Подробнее о нем можно узнать в оригинальной статье.

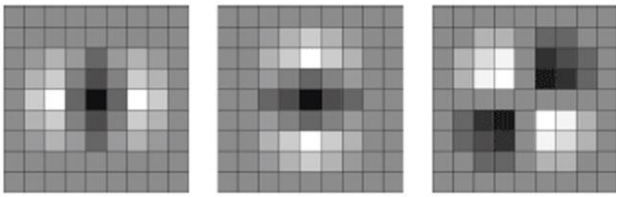

Рис. 7

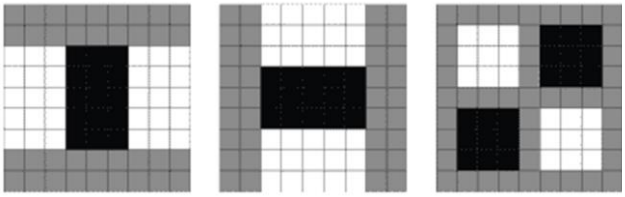

Рис. 8

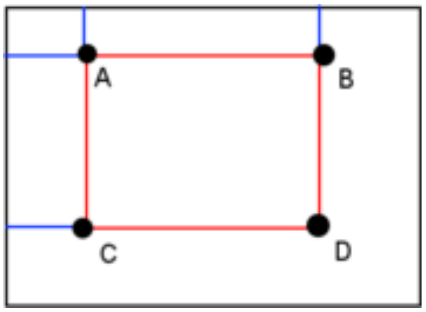

Рис. 9

Интегральным представлением изображения называется матрица

$$I(x,y) = \sum_{i \le x} \sum_{j \le y} f(i,j).$$

Она позволяет вычислять сумму интенсивностей в любом прямоугольной области (со сторонами, параллельными границам) на изображении за постоянное время. Например, на рисунке 9 обозначенная величина в прямоугольнике $ABDC$ равна $I(D) - I(C) - I(B) + I(A)$.

Как уже сказано выше, гессиан не инвариантен относительно масштаба, поэтому алгоритм перебирает различные масштабы фильтров и поочередно их применяет к данному пикселу. Из-за симметричности фильтров их масштаб не может быть произвольным – допустимы значения 9, 15, и так далее с шагом 6. Однако постепенно увеличивать размер фильтра на 6 не выгодно, потому что для крупных масштабов шаг 6 оказывается слишком мелким, а фильтры – избыточными. SURF разбивает все множество

масштабов на так называемые октавы, каждая из которых покрывает определенный интервал масштабов (рис. 10).

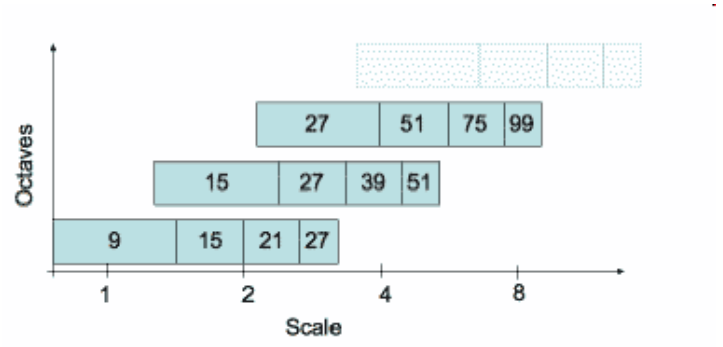

Рис. 10. Цифры – размер фильтров.

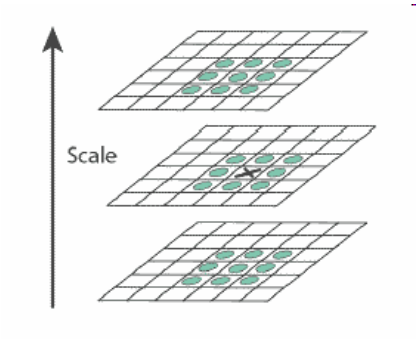

Рис. 11

Фильтры октавы считаются не для всех пикселей подряд. Первая октава считается для каждого второго пиксела изображения, вторая – для каждого четвертого, третья – для каждого восьмого и так далее. Две точки с расстоянием 2 масштаба 1 не могут содержать более одного максимума масштаба 2, 3 или более высоких. Удвоение шага пикселов для октав также позволяет экономить при расчёте фильтров.

Рассмотрим рисунок 11, демонстрирующий метод определения наличия экстремума в точке. Применяется подавление не-максимумов (non-maximal suppression). Пиксель, помеченный крестиком, является локальным максимумом, если его значение больше всех 26 соседей. Исходя из этого, можно заключить, что в октаве должно быть не менее трех фильтров.

Затем оценивается субпиксельное положение локального экстремума. Для этого куб на рисунке 11 интерполируется функцией $H(\mathbf{x}) = H + \frac{\partial H^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\partial^2 H}{\partial \mathbf{x}^2}\mathbf{x}$ и вычисляется максимум последней: $\hat{\mathbf{x}} = -\frac{\partial^2 H}{\partial \mathbf{x}^2}^{-1}\frac{\partial H}{\partial \mathbf{x}}$. Здесь $\mathbf{x} = (x, y, s)$.

Далее алгоритм определяет преобладающую ориентацию перепадов яркости в особой точке. Рассматриваются пикселы в окружности радиуса 6σ, где σ – масштаб особой точки (для первой октавы σ=2). Затем в каждой точке окружности вычисляются градиенты с использованием фильтров размера 4σ

на рис. 12. Полученные значения dx и dy умножаются на вес, определяемый как значение фильтра гаусса с центром в особой точке со стандартным отклонением 2σ, и наносятся на плоскость в виде точек (рис. 13). Вокруг начала координат вращается угловое окно размером $\pi/3$. Выбирается положение окна, при котором длина суммарного вектора для попавших в окно точек максимальна. Вычисленный таким образом вектор нормируется и принимается как приоритетное направление в области особой точки.

Затем для каждой особой точки вычисляется дескриптор – вектор из 64 (или 128) значений. Формируется квадрат, имеющий размер 20σ и ориентируемый вдоль приоритетного направления. Этот квадрат разбивается на 16 более мелких квадрантов (рис. 14), каждый из которых – еще на 25 более мелких. Для точки сетки ищется градиент, с помощью фильтров размера 2σ на рис. 12.

После нахождения 25 точечных градиента квадранта, вычисляются четыре величины $\sum dx, \sum dy, \sum|dx|, \sum|dy|$. Четыре компонента на каждый квадрант, и 16 квадрантов, дают 64 компонента дескриптора для всей области особой точки. При занесении в массив, значения дескрипторов взвешиваются на гауссиану с центром в особой точке и со стандартным отклонением 3.3σ. Еще для описания точки используется знак следа матрицы Гессе, который позволяет судить о том, максимум или минимум в данной точке.

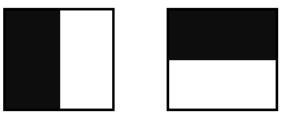

Рис. 12

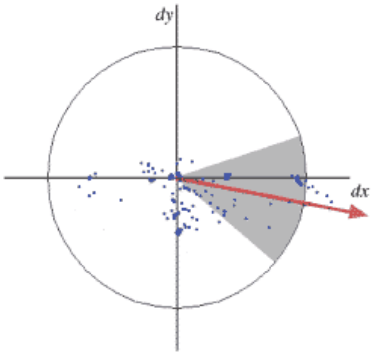

Рис. 13

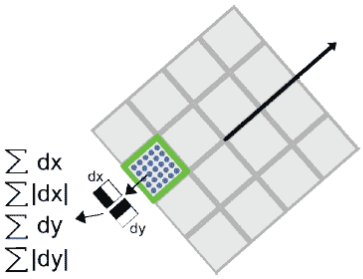

Рис. 14

Еще один метод называется ORB [11]. Он является комбинацией метода поиска FAST и метода описания BRIEF с рядом вычислительных оптимизаций. Сначала он находит точки методом FAST и применяет угловую оценку Харриса (Harris corner measure), выделяя N точек. Затем выделяет взвешенную окружность радиуса R с центром в особой точке и вычисляет ее центр масс. Вектор, являющийся разностью центра масс и особой точки, задает направление, обеспечивая инвариантность относительно поворота. Затем берётся квадрат размера RxR с центром в особой точке и ориентируется в полученном направлении. В этом квадрате выбирается n тестовых точек в соответствии с распределением $N(0, \frac{1}{25}R^2)$, где n задает размерность дескриптора. Далее с помощью сравнений интенсивностей в этих точках получается *бинарный* дескриптор особой точки.

Одним из новых методов является метод AKAZE [12]. В отличие от остальных, вместо гауссова размытия используются методы нелинейной диффузии (дифференциальные уравнения в частных производных), которые, удаляя шумы, гораздо лучше ведут себя по отношению к границам объектов.

2.2. Аффинное преобразование

Имея набор соответствующих друг другу особых точек на двух изображениях, нужно найти преобразование, которое переводило бы особые точки следующего кадра в точки предыдущего. При определении преобразования можно использовать два подхода. Что касается самого преобразования, то для решаемой задачи достаточно рассмотреть аффинный случай, а точнее преобразование подобия.

Преобразование подобия состоит из композиции масштаба, переноса и поворота. Ниже приведены соответствующие матрицы.

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, S = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}, t = \begin{bmatrix} tx \\ ty \end{bmatrix}.$$

В общем виде желаемое преобразование можно записывается следующим образом:

$$H = [RS|t] = \begin{bmatrix} \cos(\theta)s & -\sin(\theta)s & tx \\ \sin(\theta)s & \cos(\theta)s & ty \end{bmatrix}.$$

Матрица имеет размерность 2x3 в силу того, что в компьютерном зрении точки изображения принято рассматривать в однородных координатах с последующей нормализацией. Обозначив

$$a = \cos(\theta)\, s, b = \sin(\theta)\, s, c = tx, d = ty, H = \begin{bmatrix} \mathrm{a} & -b & c \\ \mathrm{b} & \mathrm{a} & d \end{bmatrix},$$

можно записать систему, которой должны удовлетворять одни и те же точки на двух кадрах:

$$\begin{bmatrix} \mathrm{a} & -b & c \\ \mathrm{b} & \mathrm{a} & d \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \text{ или } \begin{matrix} ax - by + c = x' \\ bx + ay + d = y' \end{matrix}$$

Для того чтобы найти параметры a, b, c, d, необходимо иметь всего лишь две пары точек:

$$\begin{bmatrix} x1 & -y1 & 1 & 0 \\ y1 & -x1 & 0 & 1 \\ x2 & -y2 & 1 & 0 \\ y2 & x2 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} x1' \\ y1' \\ x2' \\ y2' \end{bmatrix}.$$

Однако у нас в распоряжении находится довольно большое число соответствующих друг другу точек. При автоматическом установлении невозможно получить стопроцентно верный результат, часть соответствий наверняка будут являться ложными. Произвольно выбрав из них две пары, существует вероятность получить преобразование, не являющееся оптимальным для как можно большего числа точек, а худшем случае и вовсе неверное.

Первый подход состоит в применении алгоритма RANSAC [10] – итеративный метод оценки параметров модели на основе случайных выборок. Приведем его описание.

Пусть имеется набор D исходных данных из N элементов. Известно, что большинство его элементов должно удовлетворять некоторой

параметрической модели $M(P)$ с параметрами P, количество которых равно p. Имеется алгоритм, позволяющий вычислить параметры модели M по набору данных из k элементов. Необходимо вычислить оптимальные значения параметров $\hat{P}$, при которых данный набор исходных данных D удовлетворял бы модели M наилучшим образом. Элемент, удовлетворяющий данной модели $M(P)$ называется *inlier* для данной модели, не удовлетворяющий – *outlier*.

Возьмем случайным образом подмножество k элементов из D, обозначим его K. Вычислим на нем модель $M(P(K))$. Проверим все элементы D на соответствие модели $M(P(K))$. Определим количество inlier-ов в D для данной модели. Повторим весь этот процесс заданное количество раз. Модель, которой соответствует наибольшее число inlier-ов, будет результатом работы алгоритма.

Оценка, является или нет данный элемент исходных данных inlier-ом, производится следующим образом. Если ошибка соответствия элемента модели превышает некоторый порог, то этот элемент признается outlier-ом. Метод, по которому вычисляется ошибка соответствия элемента модели, выбирается отдельно для каждой конкретной задачи.

В общем виде схему работы RANSAC можно записать так:

1. Повторить заданное число раз m
   1.1. Выбрать подмножество из k элементов
   1.2. Вычислить параметры $\bar{P}$ модели для текущего подмножества
   1.3. Определить соответствие модели $M(\bar{P})$ исходным данным.
2. Выбрать наиболее соответствующую D модель.

В рамках поставленной задачи набором D являются пары особых точек, одна с предыдущего кадра, другая со следующего. Модель M – матрица аффинного преобразования, параметры P – две пары точек. Алгоритм, позволяющий вычислить $M(P)$ – решение линейной системы уравнений. Оценка количества inlier-ов производится следующим образом.

Преобразованием $M(\tilde{P})$ вычисляются образы точек следующего кадра, после чего считаются евклидовы расстояния между ними и точками предыдущего кадра. Если расстояние между образом и соответствующей ему точкой больше заданного порога, то такая точка считается outlier-ом.

Глава 3. Описание алгоритма и результаты

Итоговая программа реализована на языке программирования Python с использованием библиотек NumPy, SciPy и OpenCV. Обозначенный стек в последнее время приобретает популярность, поскольку предоставляет легкий в использовании высокоуровневый интерфейс к математическим функциям и процедурам самого различного характера.

На вход программе подаются:

- видеофайл,
- количество особых точек, которое нужно искать на кадрах = 1000 (не всегда используется),
- количество соответствий, по которым будет строиться матрица преобразования = 100,
- пороговое значение для алгоритма RANSAC = 5.0,
- количество итераций для алгоритма RANSAC = 1500,
- число кадров в секунду видеофайла = 60,
- задержка в секундах, с которой будет производиться извлечение кадров = 0.5.

Справа от знака равно стоят значения, с которыми работала программа.

В начале программы первым кадром видеофайла инициализируются переменная R с итоговой панорамой и переменная P с предыдущим кадром. Производится описание первого кадра (поиск особых точек и извлечение дескрипторов), которое запоминается как описание предыдущего кадра. Переменная H с общим преобразованием инициализируется единичной матрицей размерности 3. Алгоритм представляет собой один большой цикл, в котором мы с заданной задержкой извлекаем кадры из видео и последовательно их обрабатываем, пока не достигнем конца. Рассмотрим $n-$ный шаг цикла.

Итак, у нас имеется текущий кадр C. Вычисляются особые точки изображения и производится их описание. Имея два набора особых точек

(для P и C) и их количественные описания, необходимо найти соответствия между ними. Происходит это следующим образом. Для дескриптора каждой особой точки P находится ближайший по норме (зависящей от вида дескриптора: для числового дескриптора SURF используется норма L^2, для бинарных дескрипторов ORB и AKAZE используется норма Хэмминга), дескриптор точки на C. Аналогичные действия производятся для точек C по отношению к точкам P. Если каждая из двух точек, одна с P, другая с C – является «лучшей» для другой, то мы принимаем это соответствие, иначе – отклоняем как ложное. Как отмечалось выше, нельзя гарантировать точного совпадения между всеми точками P и C. Поэтому применяется алгоритм RANSAC, после чего мы получаем оптимальное преобразование A и соответствия, которые ему не удовлетворяют. В условиях данной задачи их число не должно быть большим.

Далее наборы точек сортируются по возрастанию по расстоянию между ними.

Далее выполняется склеивание изображений. К текущему кадру применяется преобразование с матрицей A. Оно должно приклеивать текущий кадр к предыдущему. Поскольку размеры изображений одинаковы, а текущий кадр сдвинут относительно предыдущего, то часть пикселей образа I текущего кадра должна иметь либо отрицательные, либо превышающие размеры P. Если это не так и I содержится в P, то мы идем на следующий шаг цикла. Пусть Δx – полученный сдвиг образа по ширине, Δy – по высоте. Предположим, например, что мы получили $\Delta x < 0, \Delta y < 0$. Это значит, что I выходит за пределы P на некоторое количество пикселей влево и вверх. Найдем направление, в котором образ сдвинут максимально. Пусть, например, это будет «вверх». Тогда на C оставим только верхнюю половину и еще $k = 0.1h$ пикселей (h – высота изображения) ниже середины и будем приклеивать к итоговой панораме только эту часть. Таким образом, рядом с серединой P всегда будет находиться середина C, и на результате будет оставаться только «вид сверху». Рассмотрим рисунки 15, 16 и 17. Без этого

действия мы бы увидели один из вариантов, изображенных на рисунках 15 и 16. С данным шагом на результате будет то, что изображено на рисунке 17. Аналогичные действия производятся, если максимальным сдвигом образа является любое другое направление. Если бы таким направлением оказалось «влево», то мы брали бы помимо левой половины изображения еще $k = 0.1w$ пикселей (w – ширина изображения).

Рис. 15

Рис. 16

Рис. 17

Вместо такого подхода можно было бы использовать фазовую корреляцию, выполняемую еще до вычисления особых точек. Таким образом можно было бы ожидать сокращение времени работы программы, но практика показала, что этот метод в различных вариациях склонен иногда выдавать ошибочные значения. Кроме этого итоговые результаты обладали худшими по сравнению с другими визуальными характеристиками. Поэтому было принято решение отказаться от фазовой корреляции.

Далее к C применяется преобразование с матрицей HA, являющейся произведением общего преобразования на полученную на текущем шаге. Это действие должно приклеить C к итоговой панораме R, но мы этого пока не делаем. Образ кадра выходит за пределы панорамы на некоторое количество пикселей. Эта ситуация регистрируется, и к итоговой панораме с нужных сторон добавляется требуемое для полного помещения образа кадра число пикселей. При этом матрица преобразования не поменялась, поэтому для корректного позиционирования нужно умножить ее слева на матрицу T

параллельного переноса текущего кадра в противоположные стороны на добавленное число пикселей. После этого можно делать финальное приклеивание с матрицей HTA.

В конце нужно не забыть обновить общее преобразование: $H = HTA$

При описанном подходе на итоговой панораме остаются швы, которые легко удаляются медианным фильтром. При этом четкость изображения визуально остается такой же.

При решении задачи на основе SURF в соответствии с эвристикой вычисления неполного набора особых точек был применен довольно прямолинейный адаптивный алгоритм: если число полученных точек для кадра было меньше желаемого, то пороговое значение гессиана уменьшалось на 100 и точки вычислялись заново; если же число точек хотя бы в 1.5 раза превосходило число желаемых, то значение гессиана для следующей итерации увеличивалось на 400. Алгоритму ORB можно явно задать число желаемых особых точек. Было замечено, что алгоритм AKAZE находит меньшее число точек по сравнению с другими, работая примерно так же быстро, поэтому для него не устанавливалось подобных ограничений.

Параметры видеофайлов:

- Продолжительность: 1 минута 1 секунда, 1 минута 7 секунд
- Ширина кадра: 1920 пикселей
- Высота кадра: 1090 пикселей
- Битрейт: 16000 кб/сек
- Частота кадров в секунду: 60 кадров
- Тип файла: H264 – MPEG-4 AVC (part 10)

Для первого видео программы на основе алгоритмов SURF, ORB, AKAZE показали следующее время работы соответственно: 231 сек, 225 сек, 255 сек. Для второго видео программы на основе алгоритмов SURF, ORB, AKAZE показали следующее время работы соответственно: 235 сек, 220 сек, 228 сек.

Результаты работы для алгоритмов SURF, ORB, AKAZE для первого видеофайла (рис 18-20):

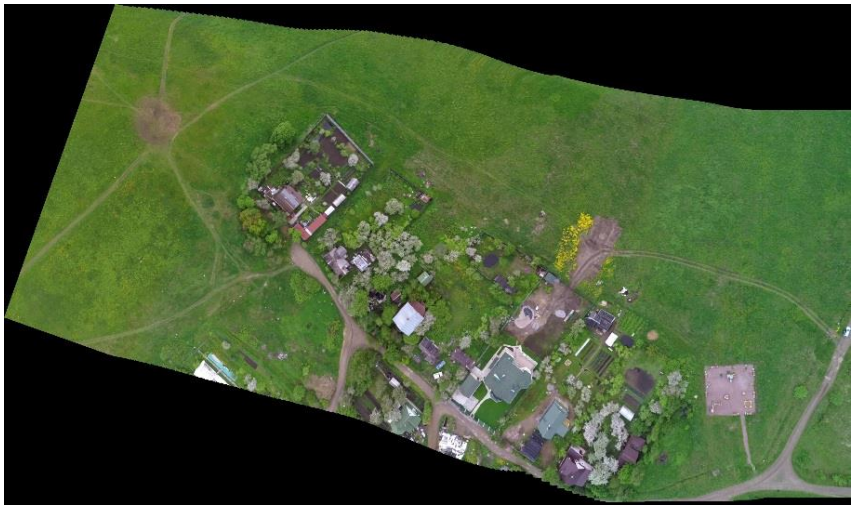

Рис. 18

Рис. 19

Рис. 20

Результаты работы для алгоритмов SURF, ORB, AKAZE для второго видеофайла (рис. 21-23):

Рис. 21

Рис. 22

Рис. 23

Как можно видеть, все методы работают примерное одинаковое время и выдают практически неотличимые результаты для тестовых видеофайлов.

Заключение

В ходе проведенной работы был сформирован алгоритм для склеивания панорамного изображения по видеоряду, в котором производится съемка в одной плоскости, а также проведено исследование и сравнение методов поиска и описания особых точек. Полученный алгоритм не является привязанным к содержанию видео и его разрешению из-за использования алгоритма на основе поиска особых точек.

В данной реализации не использовались алгоритмы блендинга, не производилось оценки параметров камеры и устранение дисторсий, поэтому полученные результаты могут «проигрывать» панорамам, создаваемым специальными программами. Для сравнения приведем такую панораму, полученную с помощью программы Image Composite Editor, разработанную в Microsoft Research (рис. 24)

Из достоинств приведенного алгоритма можно выделить скорость работы.

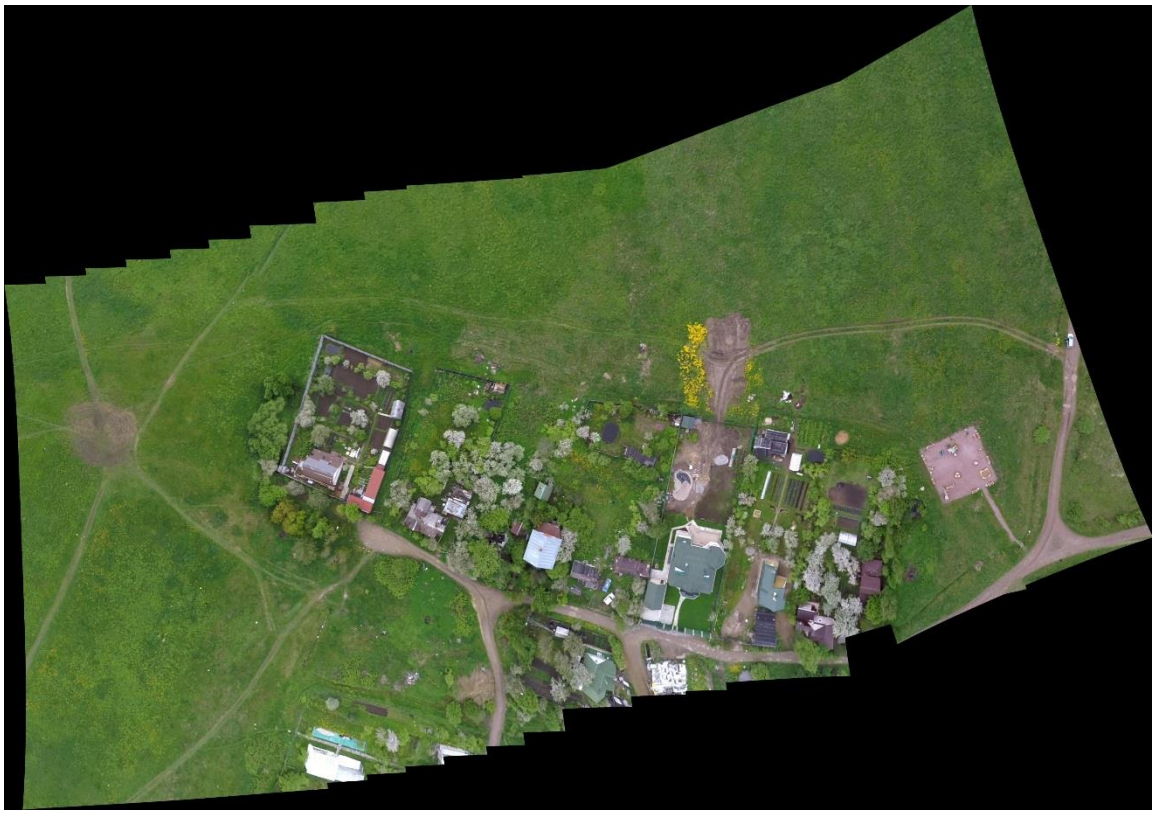

Рис. 24

Список литературы

1. Гонсалес Р., Вудс Р. Цифровая обработка изображений. М.: Техносфера, 2005. 1072 с.
2. Гонсалес Р., Вудс Р., Эддинс С. Цифровая обработка изображений в среде MATLAB. М.: Техносфера, 2006. 616 с.
3. Szeliski R. Computer Vision: Algorithms and Applications. New York: Springer-Verlag, 2010. 812 p.
4. Hartley R., Zisserman A. Multiple View Geometry in Computer Vision. New York: Cambridge University Press, 2004. 670 p.
5. Szeliski R. Image Alignment and Stitching: A Tutorial // Foundations and Trends® in Computer Graphics and Vision, December 2006. Vol. 2, Issue 1. P. 1–104.
6. Brown M., Lowe D. G. Recognising Panoramas // Proceedings of the Ninth IEEE International Conference on Computer Vision, 2003. Vol. 2. P. 1218–1225.
7. Brown M., Lowe D. G. Automatic Panoramic Image Stitching Using Invariant Features // International Journal of Computer Vision, 2007. Vol. 74, Issue 1. P. 59–73.
8. Steedly D., Pal C., Szeliski R. Efficiently Registering Video into Panoramic Mosaics // Tenth IEEE International Conference on Computer Vision, 2005. Vol. 2. P. 1300–1307.
9. Bay H., Ess A., Tuytelaars T., Van Gool L. Speeded-Up Robust Features (SURF) // Computer Vision and Image Understanding, 2008. Vol. 110, Issue 3. P. 346–359.
10. Fischler M. A., Bolles R. C. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography // Communications of the ACM, 1981. Vol. 24, Issue 6. P. 381–395.

11. Rublee E., Rabaud V., Konolige K., Bradski G. ORB: An efficient alternative to SIFT or SURF // Proceedings of the 2011 International Conference on Computer Vision, 2011. P. 2564-2571.
12. Alcantarilla P. F., Nuevo J., Bartoli A. Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces // British Machine Vision Conf. (BMVC). Bristol, UK. 2013

Приложение

Ниже приведена основная функция программы для метода на основе AKAZE.

```
def stitch_akaze(videofile,
                 kps_detect=1000,
                 matches_take=100,
                 ransac_threshold=4.0,
                 ransac_iters=2000,
                 fps=60,
                 delay_in_seconds=0.5):

    images, kps, fts = [None]*2, [None]*2, [None]*2
    transformations = [np.eye(3), None]

    status_min_percent = 100

    assert os.path.exists(videofile)
    stream = cv2.VideoCapture(videofile)
    (grabbed, images[0]) = stream.read()
    videothresh = int(fps * delay_in_seconds)
    framecount, totalframecount, pausecount = 0, 0, 0

    descriptor = cv2.AKAZE_create(nOctaves=1, nOctaveLayers=3)
    matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    (kps[0], fts[0]) = descriptor.detectAndCompute(images[0], mask=None)

    h, w = images[0].shape[:2] # width and height are the same for all images
    ratio, middle_h, middle_w = 0.1, h//2+1, w//2+1
    window_h, window_w = int(h * ratio), int(w * ratio)
    print('Centers (h and w): %d %d' % (middle_h, middle_w))
    print('Windows: height %s px, width %s px' % (window_h, window_w))

    result = images[0].copy()
    while grabbed:
        (grabbed, images[1]) = stream.read()
        framecount += 1
        totalframecount += 1
        if framecount == videothresh:
            framecount = 0
        else:
            continue
        pausecount += 1

        image1 = images[1].copy()
        print('\nProcessing %s' % pausecount)

        (kps[1], fts[1]) = descriptor.detectAndCompute(image1, mask=None)
        print('AKAZE: len(kps) %d' % len(kps))

        matches = matcher.match(fts[1], fts[0])
        print('Total matches num: %d' % len(matches))
```

```
        matches.sort(key=lambda x: x.distance)
        matches = matches[:matches_take]
        print('Only %d matches were taken, max distance %f' % (len(matches),
matches[-1].distance))
        pts0 = np.array([kps[0][match.trainIdx].pt for match in matches])
        pts1 = np.array([kps[1][match.queryIdx].pt for match in matches])


        affine = None
        # if mode == 'scipy':
        affine,   status   =   ransac((pts1,   pts0),   SimilarityTransform,
min_samples=3,
                                      residual_threshold=ransac_threshold,
max_trials=ransac_iters)
        if np.count_nonzero(status)/len(status) < status_min_percent:
            status_min_percent = np.count_nonzero(status)/len(status)

        affine = affine.params
        # elif mode == 'opencv':
        #     while affine is None and len(pts1) > 5:
        #                   affine  =  cv2.estimateRigidTransform(pts1,  pts0,
fullAffine=fullAffine)
        #         pts0, pts1 = pts0[:-10], pts1[:-10]
        #     affine = np.vstack((affine, [0, 0, 1]))
        assert affine is not None
        print('affine transformation: \n %s' % affine)

        transformations[1] = affine

        ### perform cropping image1 and stitching
        min_x,           min_y,           max_x,           max_y           =
border_points(transformations[0].dot(transformations[1]),
*images[1].shape[:2])
        print('Borders: min_x %s (0) min_y %s (0) max_x %s (%s) max_y %s
(%s)' %
              (min_x, min_y, max_x, result.shape[1], max_y, result.shape[0]))

        result  =  cv2.copyMakeBorder(result,  max(0,  -min_y),  max(0,  max_y-
result.shape[0]),
                                      max(0,     -min_x),     max(0,     max_x-
result.shape[1]), cv2.BORDER_CONSTANT, value=(0,0,0))

        # crop the opposite to the largest shift part of image
        #  here  transformations[1]  is  only  affine/aff+proj  to  the  previous
image
        _min_x,  _min_y,  _max_x,  _max_y  =  border_points(transformations[1],
*images[1].shape[:2])
        shifts   =   np.array([-_min_x,   -_min_y,   _max_x-images[0].shape[1],
_max_y-images[0].shape[0]])
        max_shift_ind = np.argmax(shifts)
        whole_image_left = True
        if shifts[max_shift_ind] > 0:
            whole_image_left = False
            if max_shift_ind == 1: image1[middle_h+window_h:, :] = 0
            elif max_shift_ind == 3: image1[:middle_h-window_h, :] = 0
```

```
            elif max_shift_ind == 0: image1[:, middle_w+window_w:] = 0
            elif max_shift_ind == 2: image1[:, :middle_w-window_w] = 0
        if whole_image_left:
            continue
        else:
            transformations[1] = transformations[0].dot(transformations[1])

        ## show
        # temp = imutils.resize(image1, width=min(800, image1.shape[1]))
        # cv2.imshow('image1', temp)
        # cv2.waitKey(0)

        ### translate image
        if min_x < 0 or min_y < 0:
            translation = np.array([[1, 0, max(-min_x, 0)], [0, 1, max(-
min_y, 0)], [0, 0, 1]])
            transformations[1] = translation.dot(transformations[1])

        warped = cv2.warpPerspective(image1, transformations[1],
(result.shape[1], result.shape[0]), result.copy(),
                                     borderMode=cv2.BORDER_TRANSPARENT)
        warped[warped == 0] = result[warped == 0]
        result = warped

        ### before the new iteration
        kps[0], fts[0] = kps[1], fts[1]
        images[0], transformations[0] = images[1], transformations[1]
        # image0 = image1

        if pausecount % 100 == 0:
            # temp = imutils.resize(result, height=min(800, result.shape[1]))
            # cv2.imshow(str(pausecount), temp)
            # cv2.waitKey(0)
            cv2.imwrite('results/final_%s.jpg' % pausecount, result)

    result = cv2.medianBlur(result, 3)
    stream.release()
    print('Total frame count: %d' % totalframecount)
    print('Min status percent: %d' % status_min_percent)
    return result
```