***Отчет о работе программы и используемых в ней алгоритмах***

**Часть I. Совмещение, кадрирование и коррекция изображения**

Программа представляет собой функцию, на вход которой подаются два имени файла (полные пути), а именно файл изображения для обработки, и файл, в который будет записан результат; третьим параметром функции является расширение результирующего файла.

На первом этапе происходит выделение из изображения областей, соответствующих трем цветовым каналам. Выделение происходит «с нахлестом», чтобы не потерять при дальнейшей обработке какую-либо часть картинки. После чего происходит поиск вектора сдвига каналов друг относительно друга для лучшего совмещения. Однако сдвиг ищется для несколько обрезанных изображений (чтобы заранее убрать возможные ошибки от шума по краям изображения).

Поиск вектора сдвига

Основой алгоритма поиска сдвига является подсчет кросскорреляции каналов через преобразование Фурье. Рассмотрим его более подробно. Пусть есть картинка размером AxB, и фрагмент размером CxD, мы хотим найти положение фрагмента относительно картинки.

* Для начала сместим сигналы картинки и фрагмента на их среднее значение, так чтобы сумма всех интенсивностей стала близкой к нулю.
* Далее, создаем две матрицы размером A+C на B+D. Копируем картинку и фрагмент в эти матрицы начиная с верхнего левого угла. Оставшиеся части матрицы заполняем нулями.
* Получаем спектры матриц при помощи FFT
* От спектра фрагмента берем комплексное сопряжение
* Перемножаем поэлементно преобразованные матрицы
* Берем обратное FFT от результата перемножения матриц
* У получившегося сигнала ищем максимум (при этом стоит учитывать, что фрагмент не мог быть сдвинут относительно изображения на размер больший чем A-B и C-D по соответствующим измерениям)
* Координаты полученного максимума и являются координатами вектора сдвига

Описанный выше алгоритм реализован функцией *GetShift*, в которую передаются три цветовых канала, координаты центра «окна», размеры «окна» (на самом деле половины размеров) и значение максимально возможного сдвига. Функция возвращает два вектора сдвига (синего канала относительно зеленого и красного канала относительно зеленого).

Данный алгоритм работает достаточно быстро и показывает хорошие результаты при последующем совмещении каналов. Однако, при попытке обработать с его помощью изображение размером более 1600x1600, 32-битная версия Matlab сталкивается с недостатком памяти. Кроме того, на изображениях высокого разрешения скорость работы алгоритма не так высока, как хотелось бы. Поэтому было принято следующее решение. Для изображений, на которых «активная» область, для которой мы определяем сдвиг, составляет меньше 140000 пикселей, применяется непосредственно описанный выше алгоритм. В случае больших изображений используется алгоритм пирамиды – функция *Pyramid* - (с повышением разрешения вдвое на каждом шаге). На первом шаге максимальное смещение каналов задается довольно широким, на последующих же оно составляет два пикселя. При этом преобразование Фурье на каждом шаге вызывается по отношению к одной и той же области (уже небольшой по сравнению со всем изображением), но на более высоком разрешении. Особенностью пирамидального поиска является то, что вектор сдвига на текущем шаге ограничивается предыдущим. Таким образом, вектор не может резко сместиться в некорректном направлении; а так как на низком разрешении определить вектор смещения довольно просто, то и на высоком разрешении вектор будет найден корректно.

Далее, разумно выбирать такие области изображения для дальнейшего поиска сдвига, для которых совмещение обещает быть «хорошим». Например, они должны лежать в центральной части кадра, картинка в этих областях должна содержать какие-либо объекты (а не просто фон), а также быть довольно контрастной. С этой целью выбирается центральная часть кадра (обрезается по 0.3 с каждой стороны изображения), полученная часть разбивается на несколько (к примеру, 48) одинаковых прямоугольных блоков; для каждого из этих блоков ищется суммарная попарная дисперсия между тремя каналами; после чего составляется вектор из координат центров этих блоков. При этом вектор отсортирован по дисперсии блоков. Таким образом, в начале вектора находятся «хорошие» блоки, в конце – «плохие». Все это выполняется функцией *GetZones*.

Таким образом, при поиске вектора сдвига для больших изображений ищутся вектора сдвига для n областей, полученных на предыдущем шаге работы программы, затем итоговый вектор вычисляется как среднее арифметическое. Этот способ позволяет получить «сфокусированное» изображение. Проверено, что алгоритм хорошо работает при выборе 3-4 областей для подавляющего большинства изображений. Однако, в некоторых случаях, когда каналы картинки плохо совмещаются по всему изображению (некоторые изображения из коллекции сильно искажены и не совмещаются простым сдвигом), искать вектор сдвига следует лишь по одной области; тогда изображение получится хорошо «сфокусированным» в одной точке, что видимо все-таки лучше, чем «везде, но чуть-чуть».

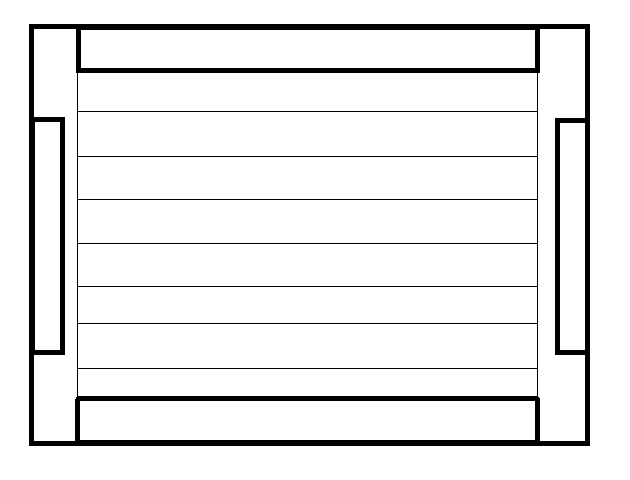
На этом заканчивается этап поиска вектора сдвига. После чего цветовые каналы смещаются друг относительно друга за счет циклического сдвига.

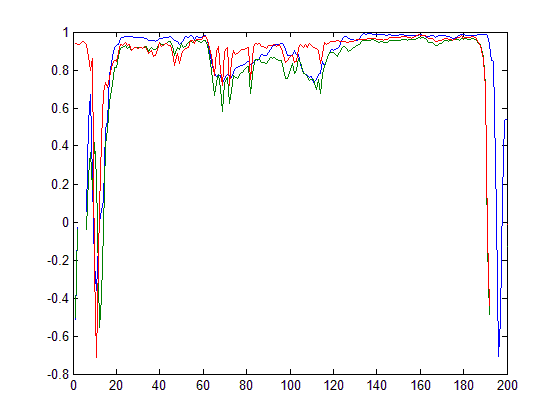
В результате, описанный алгоритм при работе демонстрирует хорошее совмещение и при этом невысокое время работы. При необходимости он может быть вручную откорректирован за счет задания количества зон фокусировки.

Обрезка

После установления величины смещения каналов относительно друг друга и получения изображения данная программа предусматривает возможность последующей обрезки фотографии. На начальном этапе в исходном изображении выделялись три части, соответствующие цветовым каналам. При этом выделение этих частей происходило «с запасом», каждый из каналов мог содержать рамку фотопластины, нежелательный шум на краях изображения, или даже часть изображения с другого канала. В данной задаче был применен следующий метод обрезки картинки.

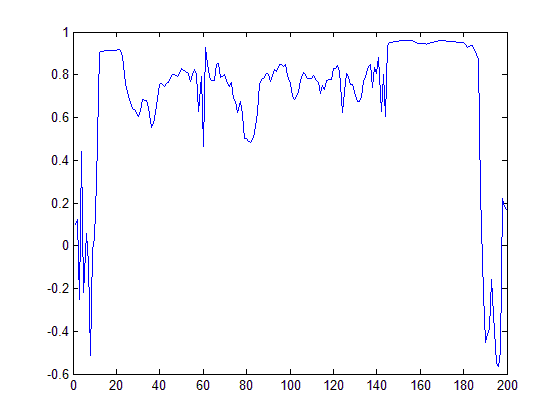
Обрезка проводится функцией *GetBorder*, которая принимает следующие аргументы: три канала изображения G, B, R (уже совмещенные относительно друг друга), *Step* – шаг поиска (в процентах от всего изображения), *Strip* – размер полосы поиска (в процентах от всего изображения), *MaxNoiseStrip* – максимально возможную ширину полосы шума (также в процентах от всего изображения). Процесс обрезки заключается в разбиении всего изображения на полосы шириной *Step* (например, 0.5%) сначала по одному измерению, потом по другому.

Длина полосы определяется параметром *Strip* (например, 80%); полосы отцентрированы относительно изображения. Каждый из краев изображения определяется путем передвижения полосы из стороны «хорошего изображения» в сторону «шума»; как только полоса попадает в область шума, процесс обрезки в этом направлении завершается, соответствующая координата определяется как выходное значение функции. Обрезка ведется по четырем направлениям.

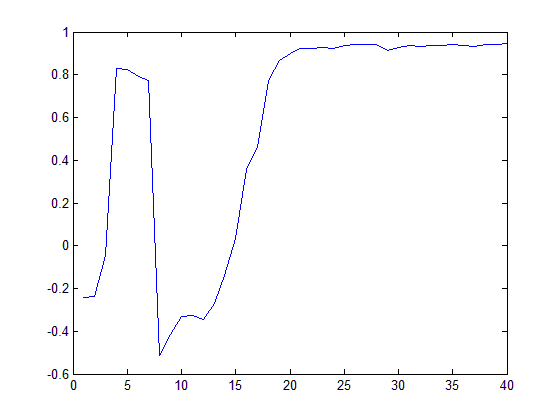
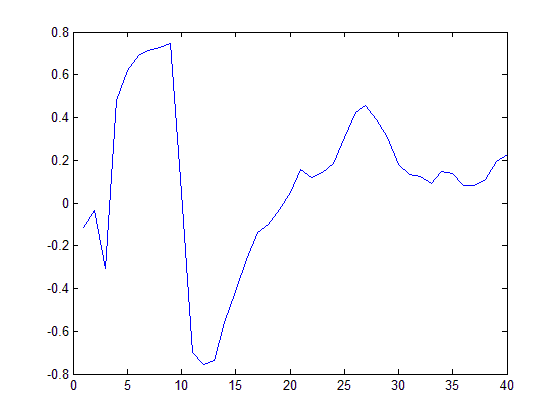


Движению полосы сопоставляется вектор. Длина вектора равна *MaxNoiseStrip/Step*. В каждой ячейке вектора записывается результат функции *StripCorr*, примененной к текущей полосе. Функция вычисляет минимальное значение корреляции выбранной полосы между каналами (синий-зеленый, синий-красный, зеленый-красный).

Корреляция по трем каналам изображения

Например, для одного из изображений, при разбиении всей картинки по вертикали полученный вектор выглядит так:

Отметки 0 и 200 соответствуют границе изображения. Резкие падения функции соответствуют переходу от изображения к шуму. Такие падения присутствуют на всех снимках. И их, соответственно, надо идентифицировать как переход к участку шума.



Полученный вектор корреляции в верхней и нижней частях изображения.

Происходит это следующим образом.

Полученный вектор (в данном примере, от 0 до 40) передается в функцию *GetCutOFF*, которая вычисляет значение элемента вектора, на котором проходит граница раздела. Этапы вычисления:

* находится элемент вектора с наименьшим значением корреляции
* правее этого индекса находится элемент с наибольшим значением корреляции
* через переданный функции коэффициент вычисляется значение корреляции между этими двумя точками (например, при k = 0.5, берется среднее арифметическое)
* находится элемент вектора между выбранными точками, значение корреляции для которого наиболее близко к выше посчитанному.
* полученный индекс возвращается в виде результата функции

Таким образом, для каждого из четырех векторов вычисляется значение индекса, на котором происходит переход от изображения к шуму. Полученные четыре числа пересчитываются в координаты пикселей на изображении и возвращаются исходной функцией *GetBorder* в качестве выходных значений. Затем, по этим координатам программа производит кадрирование изображения. Стоит отметить, что такое кадрирование не отрезает лишних частей изображения и хорошо работает на изображениях высокого разрешения.

Постобработка

На завершающем этапе программа проводит постобработку изображения, которая заключается в увеличении контрастности изображения и корректировки яркости. Эти действия совершаются при помощи функций *AddContrast* и *AddBright*. Обе функции принимают три канала изображения и коэффициент от 0 до 1, возвращают измененные каналы. Яркость изменяется следующим образом. Сначала подсчитывается нормированная сумма интенсивностей пикселей по всем трем каналам. Экспериментальным образом было установлено, что оптимальное значение суммы на подавляющем большинстве фотографий лежит в диапазоне от 1.8 до 2.0. Функция повышения яркости немного «подтягивает» интенсивность пикселей одинаково по всем каналам, чтобы значение суммы стало ближе к указанному диапазону. Сила коррекции яркости зависит от переданного функции коэффициента. Увеличение контрастности основано на использовании стандартной функции *stretchlim*. Полученные с ее помощью пределы могут расширяться за счет коэффициента, переданного в функцию AddContrast. После чего диапазон яркости растягивается до посчитанных пределов функцией *imadjust*.

Заключение

В заключении можно отметить, что данная программа была протестирована не только на трех предоставленных снимках, но и на 16 фотографиях, взятых с сайта библиотеки конгресса США. При этом никаких существенных недостатков при обработке этих изображений выявлено не было (за тем лишь исключением, что для двух снимков пришлось уменьшить число зон, по которым ведется поиск вектора сдвига, до одной).

Программа была протестирована на 32-битной Windows версии MatLab, CPU - Core 2 Duo 2.0 GHz, Ram - DDR2 3Gb. При этом время, затрачиваемое на обработку изображений 400x1000, составляло 0.3 секунды, для изображений 1000x2600 – 2.5 секунды, для изображений 3800x9800 – порядка 15 секунд.

**Часть II. Поиск и устранение дефектов изображения**

Идея алгоритма

Вторая часть работы заключается в написании функции InPaint, осуществляющей поиск и устранение некоторых дефектов на фотографии. На сведенном цветном изображении необходимо найти участки, содержащие какие-либо сколы и царапины хотя бы в одном из цветовых каналов. Затем, эти участки помечаются значением NaN на соответствующем канале, после чего передаются в предоставленную функцию inpaint\_nans, которая уже пытается «восстановить» изображение по окрестности дефекта.

Детектирование дефектных участков происходит следующим образом. Изображение разбивается сеткой на довольно мелкие квадратные блоки, размером net (например, 4 пикселя). Для каждого блока принимается решение о том, принадлежит он зоне дефекта или нет. На каждом выбранном блоке вычисляются значения кросскорреляции и среднеквадратичного отклонения между цветовыми каналами. В случае если кросскорреляция между всеми каналами достаточно велика считается, что этот участок не принадлежит дефекту. В противном случае, возможны две ситуации: либо данный участок содержит дефект, либо он принадлежит фону изображения (в этом случае цветовые каналы тоже плохо коррелируют). Разделение этих ситуаций происходит путем подсчета значений среднеквадратичного отклонения каналов в блоке. В случае фона они весьма малы, в случае реального дефекта достаточно велики. Результатом определения принадлежности блока дефектной зоне является запись единицы или нуля в бинарную матрицу маски. Пороговые значения для кросскорреляции и среднеквадратичного отклонения задаются заранее; они были определены экспериментальным способом. Кроме того, будем считать, что дефекты каждого из цветовых каналов не имеют больших общих участков, так как это маловероятно. Также выделенные дефектные зоны фильтруются – отдельно стоящие участки размером net\*net выкидываются и в дальнейшем уже не учитываются.

Однако, если применять описанный алгоритм непосредственно к выходному изображению функции PTP8, можно столкнуться с некоторыми проблемами. Так как картинка первоначально хорошо совмещена лишь в какой-то части изображения, то функция InPaint пометит все области, на которых расходятся цветовые каналы (то есть очень большие участки изображения); при этом очевидно, что последующее устранение дефектов не приведет к положительному результату.

С целью обойти данную проблему алгоритм был изменен. Все изображение разбивается на довольно крупные квадратные блоки (например, 128х128 пикселей), в каждом из этих блоков происходит совмещение цветовых каналов функцией GetShift2, при этом сдвиг совершается нециклический, а недостающая информация берется из окружающего изображения. Максимальное значение сдвига задано заранее (полагается, что оно не превышает 10 пикселей). Соответственно, в пределах каждого «большого» блока изображение можно считать хорошо совмещенным. Далее, в этих блоках применятся уже описанный алгоритм поиска дефектов, составляется бинарная маска данного блока, которая затем копируется в нужную часть общей маски с учетом полученного вектора совмещения каналов.

При таком подходе функция определения дефектов работает уже гораздо лучше, находит и отмечает большинство дефектов. Опять же, стоит отметить, что пороговые значения детектирования можно изменять.

Пример работы функции обнаружения дефектов

Однако, данный алгоритм помечает преимущественно внутреннюю часть дефекта, без границы, что приводит в дальнейшем лишь к размазыванию картинки внутри дефекта. Поэтому зоны дефектов на бинарной матрице расширяются морфологическим методом. Коэффициент расширения тоже задается заранее.

Так как функция inpaint\_nans размывает картинку по каждому цветовому каналу отдельно, то при неточном определении зоны дефекта возможно появление на итоговом изображении цветовых пятен. Поэтому радиус расширения не стоит выбирать большим, однако и не следует делать его слишком маленьким. Опытным путем было установлено, что на картинках высокого расширения, оптимальное значение радиуса лежит в интервале от 1.5\*net до 2.0\*net.

Результаты, ресурсы и время работы

Функция InPaint достаточно хорошо идентифицирует большую часть дефектов. Так как, фактически, ведется полный пересмотр изображения с сеткой порядка четырех пикселей и в каждом большом блоке дополнительно считается сдвиг каналов, то время работы функции не так мало, как хотелось бы. В частности, на компьютере с описанной выше конфигурацией поиск дефектов на изображениях высокого разрешения занимает 100 - 130 секунд.

К сожалению, на такой конфигурации функцию inpaint\_nans запустить не удалось, так как она требует у MatLab минимум 1Gb памяти для работы, что не возможно в 32-битной версии.

Поэтому, тестирование проводилось по сети на удаленном компьютере со следующими параметрами:

OS - Windows 64bit, CPU – intel Core i3 3.1Ghz, Ram – DDR3 4Gb.

При этом полное время, затраченное функцией InPaint на поиск и последующее замазывание дефектов, составляло от 120 до 350 секунд. Естественно, время работы существенно зависело непосредственно от картинки и от значения радиуса расширения дефектных зон. При тестировании функции на изображениях малого разрешения время выполнения составляло 1-3 секунд.