**ЛАБОРАТОРНАЯ РАБОТА №3**

Распознавание образов на изображении

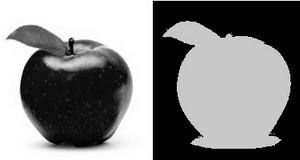
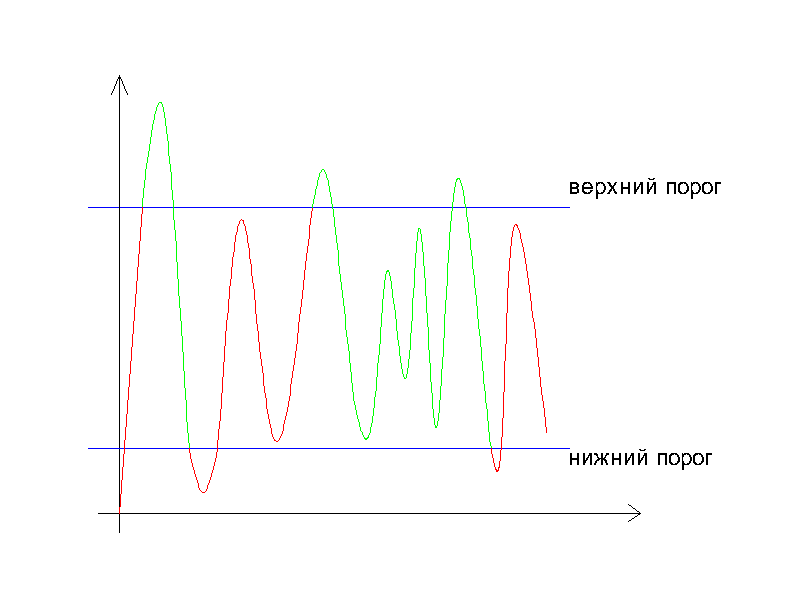
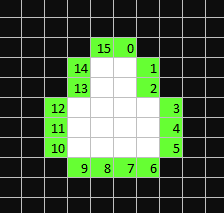
при помощи контурного анализа

Мороз Юрий Сергеевич

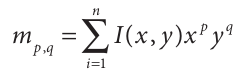
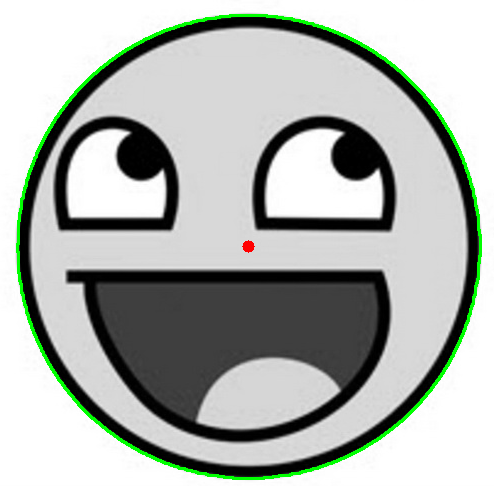
e-mail: [crazysiberianscientist@gmail.com](mailto:crazysiberianscientist@gmail.com)

*Ключевые слова*: HSV, BGR, контур, образ, распознавание, цветовая модель, цветовое пространство, двоичное изображение, пороговая фильтрация, threshold, cvtColor, Canny, findContours, границы, бегунок, интерфейс, открытие файла.

**ТЕОРИЯ**

1. И что же у нас сегодня интересного (а для некоторых и «не») на урок? А у нас на сегодня (а для некоторых на несколько недель) некоторые **простые** методы распознавания образов на изображении при помощи контурного анализа. **Простые,** т.к. большая часть из вас ещё не сдала старые лабы без объяснения причин. Читаем внимательно и не торопимся, но и не спим.
2. Как написано в педивикии: ***Распознавание образов****— это отнесение исходных данных к определенному классу с помощью выделения существенных признаков, характеризующих эти данные, из общей массы несущественных данных* [ru.wikipedia.org/wiki/Теория\_распознавания\_образов]*.* Так вот приступим к познанию священных методов.
3. Предварительно ещё узнаем о нескольких методах для получения ***двоичного*** (где каждый элемент может принимать значение только true или false) изображения, которые нам в дальнейшем пригодятся.
4. **!!!ВАЖНО:** В OpenCV нет отдельного типа для создания двоичного изображения. В качестве двоичного изображения принято считать одноканальное 8-ми битное изображение (т.е. тип CV\_8UC1), элементы которого могут принимать значения только 0(false) или 255(true). Можно задать двоичное изображение как экземпляр шаблонного класса ***Mat\_<bool>***, но функции OpenCV не будут с ним работать.
5. **Пороговая фильтрация**. Данная функция изменяет элементы изображения в зависимости от расположения их значения относительно заданного порога. Порог задаётся пользователем или высчитывается полуавтоматически, например, как в
6. [http://docs.opencv.org/modules/imgproc/doc/miscellaneous\_transformations.html#adaptivethreshold].
7. Есть множество видов пороговой фильтрации
8. [http://docs.opencv.org/doc/tutorials/imgproc/threshold/threshold.html], нам пока нужна только двоичная.
9. Чтобы обработать изображение в OpenCV при помощи пороговой фильтрации с заданным порогом следует использовать функцию:
10. double threshold(InputArray **src**, OutputArray **dst**, double **thresh**, double **maxval**, int **type**), где:
11. InputArray **src –** одноканальное изображение, можно получить из **BGR** изображения, при помощи функции из **главы 3**, с используя константу CV\_BGR2GRAY;
12. double **thresh –** значение порога;
13. double **maxval –** наибольшее значение яркости для элементов, определённых как **true**;
14. int **type** – вид пороговой фильтрации, как его выбрать смотри ниже.
15. Простая двоичная пороговая фильтрация. Константа для выбора её вида: THRESH\_BINARY. Присваивает элементу значение **maxval** (true), если тот выше указанного порога. В противном случае – 0 (false).
16. \texttt{dst} (x,y) =  \fork{\texttt{maxval}}{if $\texttt{src}(x,y) > \texttt{thresh}$}{0}{otherwise}
17. Обратная двоичная пороговая фильтрация. Константа для выбора её вида: THRESH\_BINARY\_INV. Присваивает элементу значение 0(false), если тот выше указанного порога. В противном случае – **maxval** (true).
18. \texttt{dst} (x,y) =  \fork{0}{if $\texttt{src}(x,y) > \texttt{thresh}$}{\texttt{maxval}}{otherwise}
19. Вот пример работы данной функции: 
20. Очевидно (не так как тут <http://s.pikabu.ru/post_img/2014/01/12/10/1389542243_824700135.jpg>), что данную функцию удобно использовать для выделения объектов, которые резко выделяются относительно фона и имеют более менее однородную структуру.
21. **Преобразование цветового пространства изображения из BGR в HSV.** Цветовая модель **HSV(HSB)** используется для удобного выделения объектов по цветовому признаку. **H-Hue(оттенок), S-Saturation(насыщенность), V-Value(Значение, или более понятное как Яркость).**
22. При помощи **HSV-**модели можно удобно производить **цветовую сегментацию. Цветовая сегментация** - выделение объектов на изображении по цветовому признаку.
23. Чем же **HSV** модель удобнее **BGR**? А тем, что она более привычна для восприятия человеком, ведь человеку проще сказать: «фиолетовый(H=3000) насыщенный(S=100%) тёмный(V=42%) дай мне баклажан быстро», чем «тёмно-красно(R=107) – тёмно-синеватый(B=107) баклажан дай мне быстро».
24. Обычно на изображениях нам требуется найти объект одного оттенка, но при этом его яркость может меняться (обычно из-за освещения). Если захочешь все баклажаны с бабкиного прилавка, то тебе достаточно сказать (на псевдокоде): **(290<=H<=310)&&(10<=V<=100)&&(10<=S<=100)***,* и ты получишь баклажаны (возможно с лицами гостей из средней Азии, которые попали на картинку). В модели **BGR** тебе уже будет более проблематично такое сделать. При помощи координат элементов, попавших в заданный диапазон, можно создать **двоичное изображение**, в котором элементы с выше полученными координатами будут равны **true**, а все остальные - **false**.
25. **Надоело читать уже эту воду**? Получай пример как преобразовать цветовое пространство из BGR в HSV:
26. cvtColor(bgrimg, hsvimg, CV\_BGR2HSV);
27. Эта функция позволяет преобразовывать цветовое пространство изображения из одного в другое. Как видно в примере для этого достаточно указать правильную константу (их там много) из какой модели в какую надо сделать преобразование.
28. **!!!ВАЖНО:** Функция cvtColor принимает на вход только изображения типов **8U** и **32F**. После выше приведённого преобразования получим **8-битное 3-канальное** изображение (к элементам обращаемся как к **Vec3b**). Диапазоны каналов HSV-модели в Opencv **H[0;179] S[0;255] V[0;255]**, это немного отличается от стандартного представления **HSV** модели, у которой следующие диапазоны каналов: **H[00;3590] S[0%;100%] V[0%;100%]**. В графических редакторах и простых преобразовалках на сайтах **HSV** палитра представлена именно в стандартном виде, поэтому если ты хочешь подобрать цвет в этих палитрах, а потом использовать в OpenCV, то следует помножить значение каждой компоненты на масштабирующий коэффициент для каждой компоненты (думаю, что догадаешься как их вычислить).
29. Для пороговой фильтрации на любых изображениях советую использовать функцию:
30. void inRange(InputArray src, InputArray lowerb, InputArray upperb, OutputArray dst)
31. InputArray src – любое изображение
32. InputArray lowerb – нижний порог. Имеет тип изображения (в случае с 8U\_C3 имеет тип Vec3b).
33. InputArray upperb – верхний порог.
34. OutputArray dst – двоичное изображение. Те элементы, которые попали в диапазон, равны 255 (true).
35. **Использование алгоритма Кэнни (Canny) для выделения границ.** Данный алгоритм используется для выделения границ между контрастными участками на изображении, на выходе алгоритма получаем двоичное изображение с границами, где все элементы со значением **true –** искомые границы. Для того, чтобы вы поняли о каких границах идёт речь приведём пример на богомерзких рожах:
36. 
37. Чтобы вас особо не напрягать по поводу как работает этот алгоритм (т.к. захотят подробнее про него узнать не только лишь все), то скажу кратко с выделением основных этапов и дам ссылку [ru.wikipedia.org/wiki/Оператор\_Кэнни]:
38. Производится расчёт градиентов изображения (обычно при помощи оператора Собеля, вроде в лекциях вы про него слышали);
39. Производится выделение локальных максимумов на полученном изображении;
40. Подавляются слабые границы при помощи двух порогов (пример их задания можно увидеть на выше приведённой картинке рядом с бегунками). И картинка для наглядности:
41. 
42. [Скопировано из источника выше] Для чего используются два порога: чтобы уменьшить влияние шума для инициализации кривой, используем верхний порог;  
    чтобы «не потерять хвост», используем нижний порог при прослеживании.
43. По моей логике (не смог найти в литературе) самым наибольшим значением порога может быть значение **1530** в случае 8-битного изображения. Потому как элементы изображения, обработанного оператором Собеля [ru.wikipedia.org/wiki/Оператор\_Собеля] по вертикали и горизонтали, могут иметь значение не выше 1530 (когда граница проходит по диагонали получаем: 255\*1+255\*2+255\*1+255\*2). Эта идея была проверена на практике: к изображению с идеальными границами (переход между 0 и 255) применялся алгоритм Кэнни, при установке порога выше **1529**(-1 скорее всего из-за того, что «идеальные» условия не совсем идеальные) все границы пропадали.
44. А теперь функция Кэнни:
45. void Canny( InputArray image, OutputArray edges,double threshold1, double threshold2,int apertureSize = 3, bool L2gradient = false )
46. InputArray image – 8-битное одноканальное изображение;
47. OutputArray edges – двоичное изображение с границами;
48. double threshold1, double threshold2 – нижний и верхний пороги, должны >0, но не ограничены по наибольшему значению ;
49. int apertureSize – размер маски оператора Собеля;
50. bool L2gradient – включить более точное определение амплитуды градиента. Если **=true,** то амплитуда считается как sqrt{(dI/dx)^2 + (dI/dy)^2}, если нет, то |dI/dx|+|dI/dy|. Если включён, то наибольшее значение порога равняется около 1180.
51. **Функция поиска контуров на двоичном изображении.**
52. После получения двоичного изображения могут понадобиться контуры участков из **true**-элементов. Контур – массив точек, которые упорядочены в зависимости от их взаимного положения.
53. 
54. Функция для поиска контуров:
55. void findContours(InputOutputArray **image**, OutputArrayOfArrays **contours**, OutputArray **hierarchy**, int **mode**, int **method**, Point **offset**=Point())
56. [http://docs.opencv.org/modules/imgproc/doc/structural\_analysis\_and\_shape\_descriptors.html#findcontours]
57. InputOutputArray **image –** 8-ми битное одноканальное двоичное изображение. !!!Данная функция изменяет это изображение (изменяет значения элементов ради своей выгоды), поэтому если хочешь использовать исходное изображение дальше, то передавай его копию (**Mat::clone()**) данной функции;
58. OutputArrayOfArrays **contours** – массив из найденных контуров. Хранится как **std::vector<тип>.** Не забываем написать **using namespace std;**, чтобы каждый раз не писать **std::** . **vector<тип>** - шаблонный класс, представляет собой улучшенный динамический массив, позволяет удобно удалять, добавлять, изменять и т.д. элементы динамического массива. Доступ к элементам вектора осуществляется через оператор **[int index], отсчёт от 0.** Пример: Point p=contours[0][4], получаем доступ к 4-ой точке 0-го контура. Чтобы узнать количество элементов в векторе используем метод **vector<type>::size().** Пример: int cnts\_size=cnts.size();
59. OutputArray **hierarchy –** вектор с иерархией контуров, не всегда используется. В нашем случае просто пропускаем его.
60. int **mode** – режим поиска контуров. Несколько режимов: поиск только внешних контуров, которые не находятся внутри другого контура; поиск всех контуров без иерархии; поиск контуров с иерархией (какой внутри какого расположен). Для нашего задания нам нужен только первый режим, его константа CV\_RETR\_EXTERNAL.
61. int **method** – метод аппроксимации(упрощения) контуров, удаляет маловажные (удаление которых не сильно повлияет на форму контура) точки в контуре. В нашем случае не будем аппроксимировать: CV\_CHAIN\_APPROX\_NONE
62. Point **offset** – точка на которую будет сдвинуты все координаты точек контура, не трогаем. Может использоваться, когда ищем контуры на ОИ, которые в дальнейшем требуется наложить на оригинальное изображение. В этом случае логично сделать так, чтобы координаты этой точки равнялись координатам верхнего левого угла прямоугольника ОИ.
63. Пример:

vector<vector<Point>> cnts;

1. findContours(cimg, cnts, CV\_RETR\_EXTERNAL, CV\_CHAIN\_APPROX\_NONE);
2. Чтобы нарисовать контур на изображении можно использовать drawContours, но она немного дурацкая (не может принять на вход отдельный контур, ей надо массив контуров, что иногда неудобно). Поэтому советую использовать функцию для рисования кривых:
3. void polylines(InputOutputArray **img**, InputArrayOfArrays **pts**, bool **isClosed**, const Scalar& **color**, int **thickness**=1, int **lineType**=8)
4. [http://docs.opencv.org/modules/core/doc/drawing\_functions.html?highlight=polylines#polylines]
5. InputArrayOfArrays **pts –** один контур или вектор контуров.
6. bool **isClosed –** замкнуть(true) линии или нет(false);
7. int **lineType –** тип линии, не трогаем. Если интересно, то смотри по ссылке выше.
8. int **thickness –** жирнота линий, ***-1*** – чтобы залить пространство внутри контура.
9. **Характеристики контура.**
10. Нашли мы контуры вот, нарисовали их красиво, и что дальше с ними делать? Можно оставить так и радоваться красивой картинке, а можно найти среди них контур, который описывает искомый объект. У контура можно выделить множество параметров, которые будут характерны только для определённого типа объектов, например: площадь, периметр, их соотношение, форма контура и т.д.
11. **Площадь контура** можно найти при помощи функции:
12. double contourArea( InputArray contour, bool oriented = false )
13. Возвращает значение площади внутри контура;
14. bool oriented – изменять ли знак(+\-) возвращённого значения площади в зависимости от его направления (по часовой стрелке или против).
15. **Моменты контура.** Моменты контура считаются по формуле:
16. 
17. ***I(x,y)*** – значение элемента (для контура =1); ***p,q*** – параметры функции.
18. Имеются также другие виды моментов, такие как центральные, нормальные, Хью(Hu). Он представляют собой усовершенствованные простые моменты. Кому интересно, тот почитает в интернете.
19. Совокупность таких моментов с разными параметрами ***p,q*** позволяет описать контур при помощи нескольких числовых значений, что даёт возможность сравнивать один контур с другим и находить похожие. Метод сравнения контуров по моментам инвариантен к масштабированию, повороту, отражению, но не всегда точен и чувствителен к изменению самой формы объекта. В лабораторной работе его используем, потому что он реализован в OpenCV и прост для использования.
20. Простые моменты, которые нам пригодятся: ***m(0,0)*** - периметр контура; ***m(0,1); m(1,0).*** Они позволят найти центр масс контура.
21. ***Xцм= m(1,0)/m(0,0); Yцм= m(0,1)/m(0,0).***
22. 
23. Красная точка – центр масс зелёного контура.
24. Для хранения моментов используется специальный класс **Moments**, а для их расчёта функция **moments.**
25. Пример (думаю, что тут всё понятно):

Moments mnts = moments(cnt);

1. double m00 = mnts.m00;
2. **Функция сравнения контуров.**
3. Функция производит сравнение контуров при помощи выше описанных моментов.

double matchShapes( InputArray contour1, InputArray contour2, int method, double parameter );

int method – номер метода сравнения моментов, нас интересует константа ShapeMatchModes::CV\_CONTOURS\_MATCH\_I2, при выборе этого метода использовании функция будет возвращать разность моментов контуров, чем ближе к 0, тем контуры более похожи.

double parameter – параметр, который в дальнейшем создатели хотят использовать для метода сравнения, на данный момент не работает, присваиваем ему 0.

1. **Как удобно загружать изображения в программу.**
2. **cv::Directory.** Работа с путями. Не знаю зачем, но разрабы опять выпилили этот полезный класс из 3.0.0 версии. При чём про аналог в 3.0.0. ничего сообщили. Поэтому в комплекте идут два файла, которые необходимо подключить к проекту вашему (если версия OpenCV не ниже 3.0.0).
3. При чём этот класс содержит только методы и работает по сути как область имён (только с ***using*** нельзя использовать). Методы предназначены для получения списка файлов или папок по указанному пути.
4. Там есть три метода:

static std::vector<std::string> GetListFiles(const std::string& path, const std::string & exten = "\*", bool addPath = true);

static std::vector<std::string> GetListFilesR(const std::string& path, const std::string & exten = "\*", bool addPath = true); – рекурсивное получение списка, т.е. получение списка всех файлов, которые содержатся по указанному пути и во всех вложенных папках внутри исходного каталога.

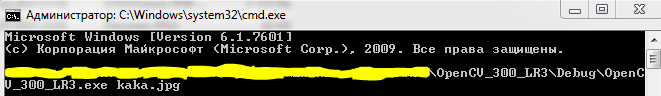
1. static std::vector<std::string> GetListFolders(const std::string& path, const std::string & exten = "\*", bool addPath = true);
2. Я тут думаю, что вам более менее должно быть понятно, т.к. названия переменных и функций говорят сами за себя. Немножко служебных символьных сочетаний (как-то немного криво я их описал):
3. "/" – перейти ко вложенному каталогу;
4. "./" – ставится в начале пути, означает относительный путь, относительно рабочей папки;
5. "../" – перейти к родительскому каталогу
6. Дико, например:
7. vector<string> fnms = Directory::GetListFiles("./");//получить список имён всех файлов в рабочей папке.
8. vector<string> fnms = Directory::GetListFiles("./","\*.jpg",false);//получить имён список всех jpg в рабочей папке без добавления к ним пути.
9. **Аргументы функции main.**  int main(int argc, char\*\* argv) .

Эти аргументы содержат параметры запуска программы, которые могут быть переданы программе во время её запуска. Изменять пользователю их нельзя.

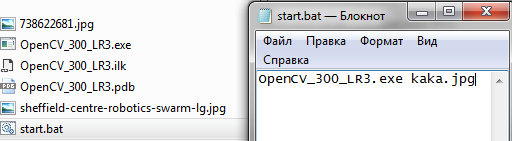
int argc – количество параметров, по умолчанию =1, т.к. программе всегда передаётся её полное имя (т.е. вместе с полным путём к приложению).

char\*\* argv – массив строк (в формате char\*), которые содержат значения параметров. В нашем случае данная переменная может хранить имя картинки, имя которой мы укажем при запуске программы.

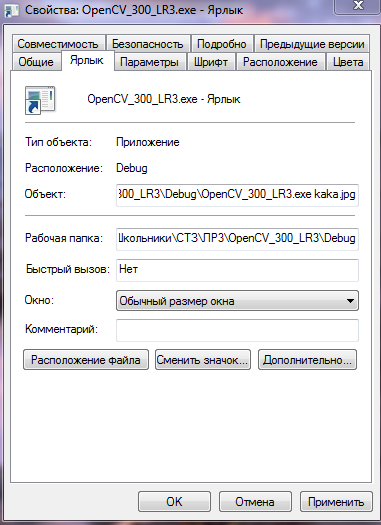
Как присвоить данным параметрам значения:

а) Через командую строку: 

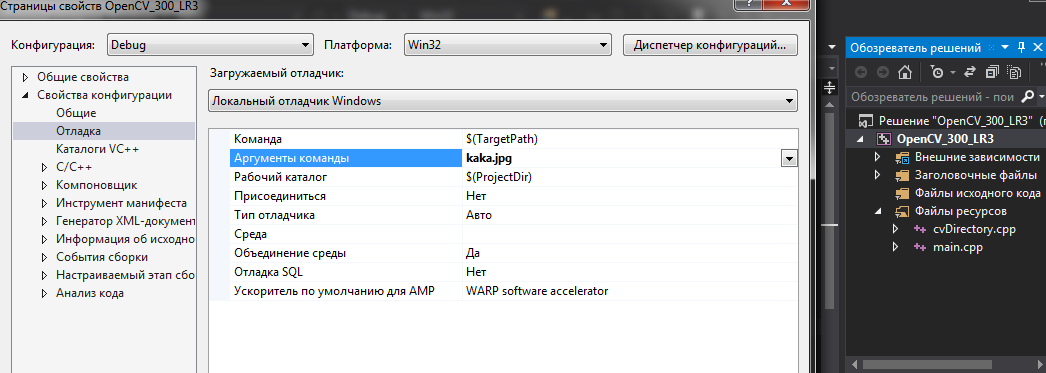
б) Через bat-файл:



в) Через ярлык:

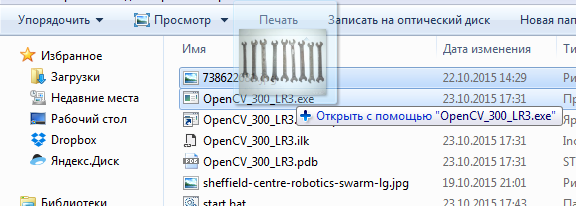


г) Через отладчик:



д) Через «Открыть с помощью» какой-нибудь файл, где в приложениях выбираем нашу программу.

е) Через перетаскивание элемента на исполняемый файл или его ярлык:



Пример:

int main(int argc, char\*\* argv){

string fn;

if (argc>1) fn= argv[1];

else fn = "img.jpg";

img = imread(fn,0);

1. **Создание бегунков в интерфейсе программы.**
2. Чтобы не быть быдлокодером и не компилировать каждый раз заново программу, когда нам надо изменить значения переменных, будем использовать бегунки (как на картинке с богомерзкими рожами).
3. Те, кто жаждет познать остальные возможности примитивного GUI OpenCV, могут почитать по ссылке: [http://docs.opencv.org/modules/highgui/doc/user\_interface.html].

Функция для создания бегунка:

int createTrackbar(const string& **trackbarname**, const string& **winname**, int\* **value**, int **count**, TrackbarCallback **onChange**=0, void\* **userdata**=0)

const string& **winname** – имя окна внутри которого хотим создать бегунок, окно обязательно должно быть заранее создано.

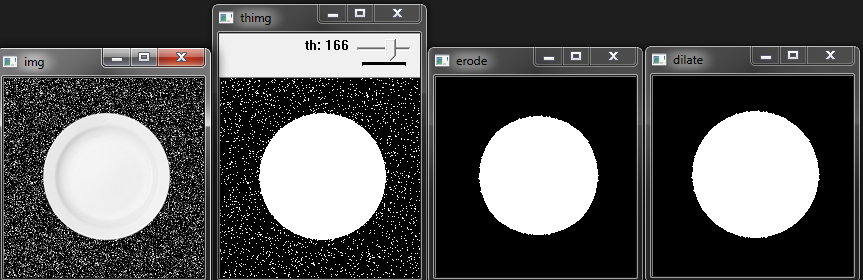
int\* **value** – указатель на переменную, куда будет записываться значение текущего положения бегунка.

int **count** – наибольшее значение положения бегунка.

TrackbarCallback **onChange** – функция отклика, которая вызывается при изменении положения бегунка. Чтобы её не использовать надо написать 0 вместо имени. Её прототип: void Foo(int,void\*); , где первый аргумент – текущее положение бегунка, 2-ой – указатель на данные пользователя.

void\* **userdata** – указатель на данные пользователя.

Файл ***“begunok.cpp”*** содержит пример использования бегунка.

1. **Использование морфологии.**
2. А теперь спустя столько времени я решил рассказать вам о **морфологии**, которая может пригодиться вам для выполнения заданий. Расскажу я об её основных двух функциях.
3. **Эрозия (Erode).** Обычно используется для удаления шума с изображения. Скользящим окном (ядро) обходит всё изображение. На каждом шаге перемещения находит наименьший элемент внутри ядра и присваивает его значение элементу в (по умолчанию) центре ядра. Математически это выглядит так:
4. \texttt{dst} (x,y) =  \min _{(x',y'):  \, \texttt{element} (x',y') \ne0 } \texttt{src} (x+x',y+y')
5. Функция: [<http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#erode>]
6. void erode(InputArray src, OutputArray dst, InputArray kernel, Point anchor=Point(-1,-1), int iterations=1)
7. InputArray src – изображение почти (8U можно) любого типа и с любым количеством каналов.
8. InputArray kernel – маска ядра. Представлена в виде типа **Mat.** Можно нарисовать самому, а можно получить при помощи функции getStructuringElement
9. [http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#getstructuringelement]
10. Point anchor – точка ~~якоря корабельного~~ присваивания, та точка внутри ядра, куда будет записан результат работы фильтра. По умолчанию - в центре.
11. int iterations=1 – количество повторений прохода фильтром по изображению.
12. **Расширение (Dilate).** Обычно используется для восстановления размера участков после эрозии или заполнения небольших промежутков. Скользящим окном (ядро) обходит всё изображение. На каждом шаге перемещения находит наибольший элемент внутри ядра и присваивает его значение элементу в (по умолчанию) центре ядра. Математически это выглядит так:
13. Функция: [http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#dilate]
14. void dilate(InputArray src, OutputArray dst, InputArray kernel, Point anchor=Point(-1,-1), int iterations=1) – всё так же как и в erode.
15. Ты только посмотри! Какой-то шалун рассыпал белый порошок (вроде сахар) в автомат для отбраковки тарелок. Нам лень идти его вытирать (потому что не платят за это), поэтому используем морфологию:
16. 
17. **ЗАДАНИЕ**
18. Товарищ!!! Помоги пилоту бомбардировщика найти демократически умеренных сношателей с животными и с автоматами при помощи тепловизора. Известно, что они любят прятаться в уютных тёплых домиках (которые заметно теплее всего остального) и там же делать свои тёмные делишки. Чтобы пилоту было удобнее навестись тебе надо обозначить примерный центр цели на изображении. См. картинки в папке «allababah».
19. Товарищ! Родина не ждёт! Тебя отправили работать в сверхсекретное КБ, где тебе поручили разработать сверхсекретную систему сверхсекретного наведения на забугорные транспортные средства, которые желают принести нам демократию. Твоя задача - точно навести сверхсекретную пушку на тёпленькое моторное отделение. См. картинки в папке «teplovizor».
20. В том же секретном КБ тебе поручили следить за особыми экспериментальными роботами, которые любят погреться под лампочкой. Твоя задача:
21. На каждом роботе найти его цветную верхнюю крышку и обвести контуром цвета его команды;
22. Найти лампу, обозначить её как-нибудь.
23. Для каждой команды обозначить ближайшего робота к лампе, путём рисования его центра масс.
24. См. картинки в папке «roboti».
25. А теперь тебе поручили разработать систему ТЗ по отбраковке гаечных ключей для сборки тех роботов. Обозначь бракованные и правильные гаечные ключи разными метками. См. картинки в папке «gk».