Aruco (Charuco)

Aruco предтавляет собой модуль OpenCV для автоматической детекции и определения положения визуальных маркеров. Пример представлена на рисунке

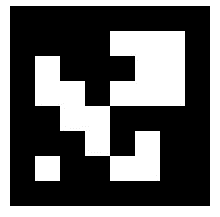


Рис. – Маркер Aruco

Хотя aruco является модулем OpenCV, он не входит в стандартную установку а является частью пакета contributions.

Чтобы использовать Aruco его необходимо скомпилировать вместе с той версией OpenCV, которая будет использоваться в проекте.

Для этого:

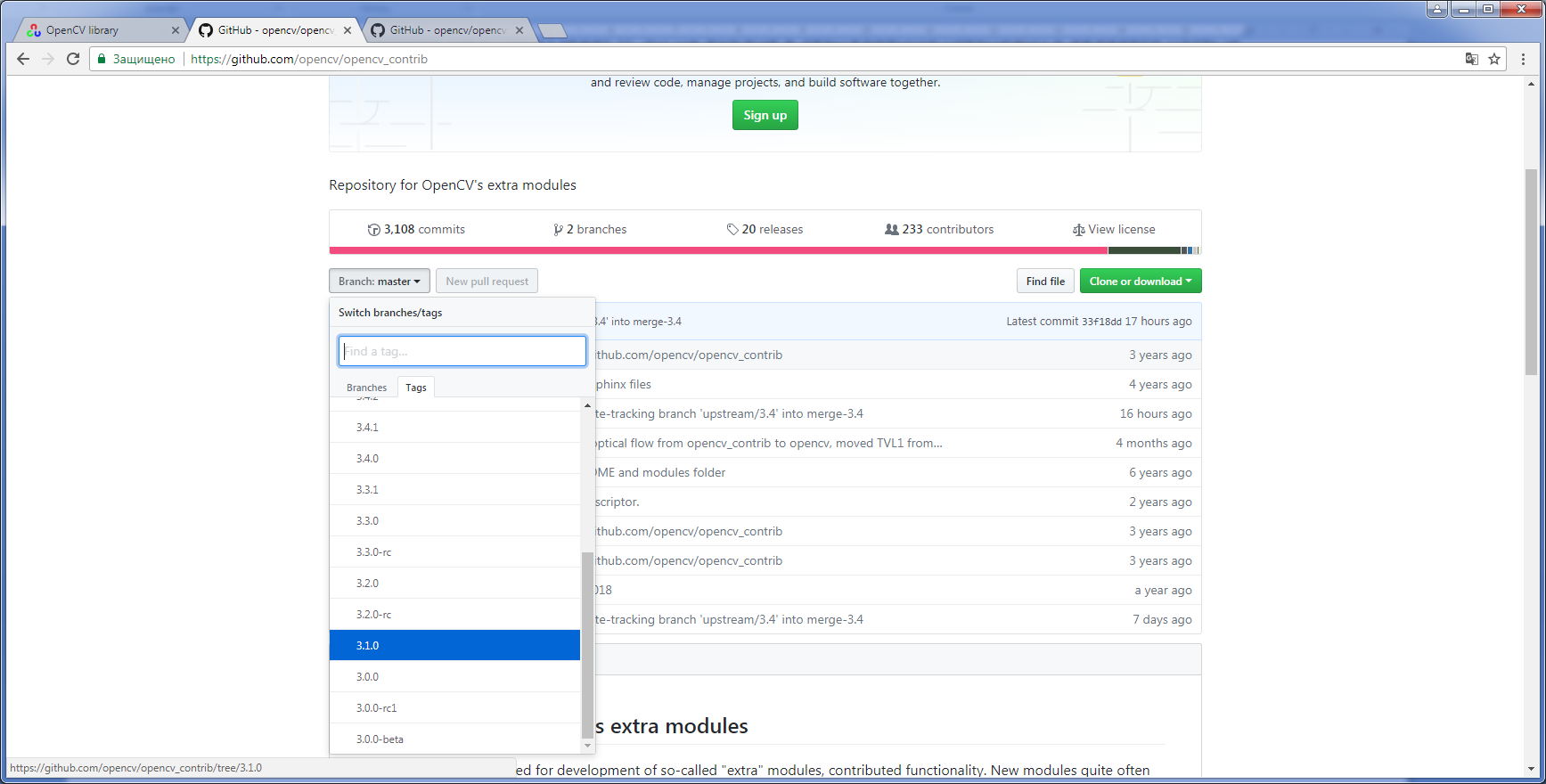
1) Скачивается исходный код с сайта разработчика

https://github.com/opencv/opencv – OpenCV

https://github.com/opencv/opencv\_contrib – пакет contributions

2) Создаем отдельную папку (напримера "<путь к папке без русских букв и странных символов>/opencv\_x.x/") в которую кладем папку source из скачанного репозитория OpenCV

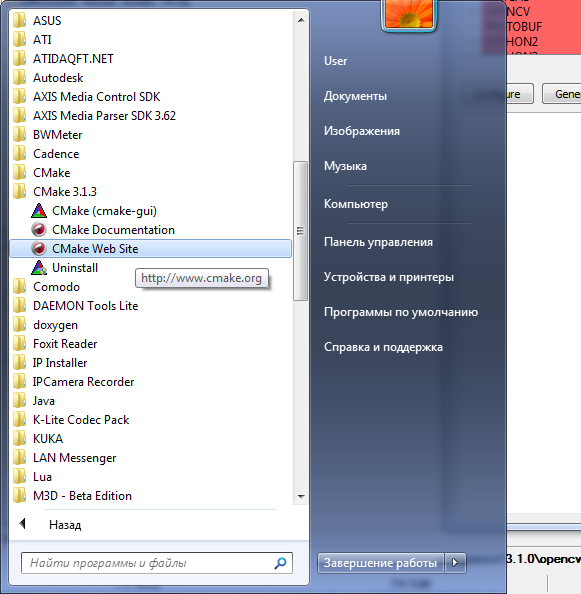
!!!ВАЖНО!!!! версия OpenCV и пакета contributions должны совпадать. Для этого в выпадающем списке branch на сайте разработчика нужно выбрать вкладку Tag и там соответствующую версию (в данном случае 3.1). См. рис.



В папке, где лежит папка source (..../opencv\_x.x) создаем папку contribs, куда копируем содержимое архива opencv\_contrib-x.y.z, скачанного с сайта разработчика.

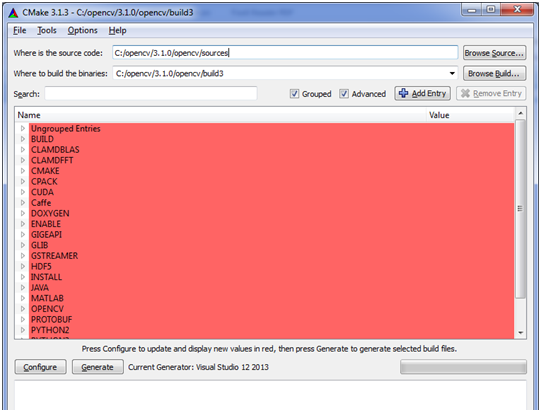
3) Устанавливаем программу CMake с сайта https://cmake.org/

4) Запускаем сmake-gui.exe из меню пуск или из консоли



5) в поле ".. soruce code" задаем папку "..../opencv\_x.x/source"

в поле "Where to build binaries" задаем "..../opencv\_x.x/build " (или другую, если понимаем, что делаем)



Где лежит исходник Opencv

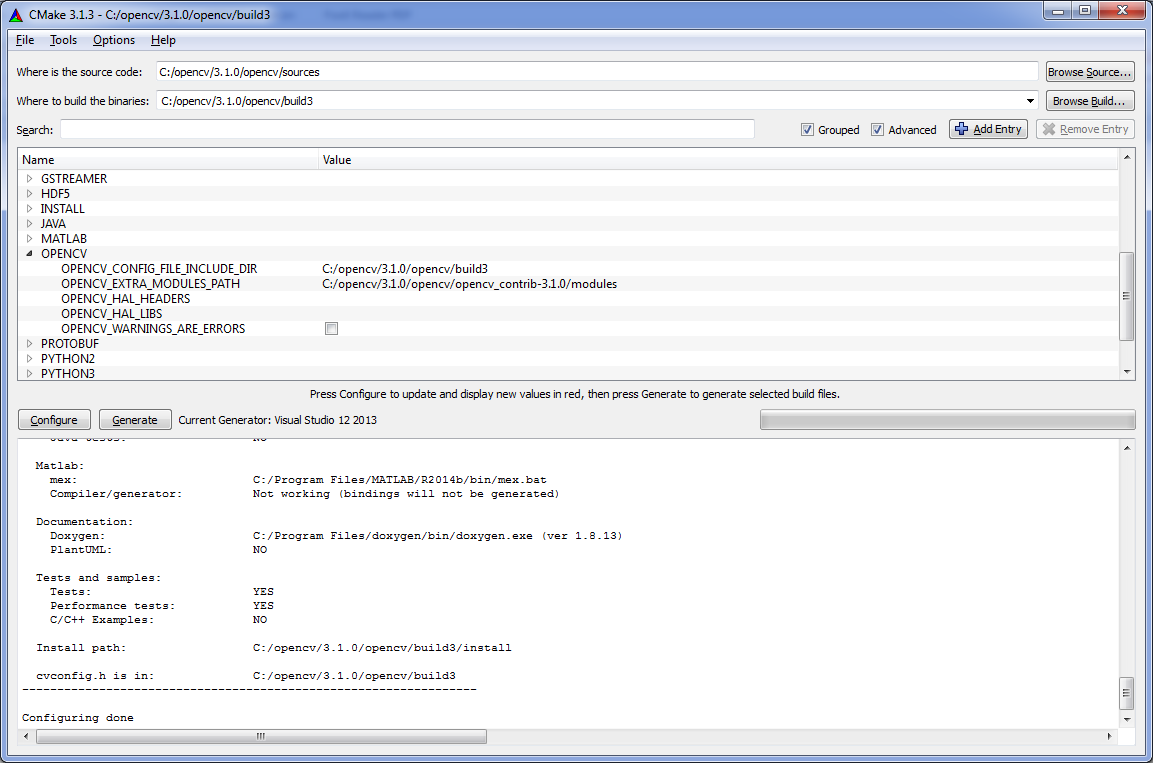
Другая папка, куда будут положены файлы проекта

Нажимаем для конфигурации

Нажимаем для генерации файлов проекта

6) нажимаем кнопку "Configure" и ждем.

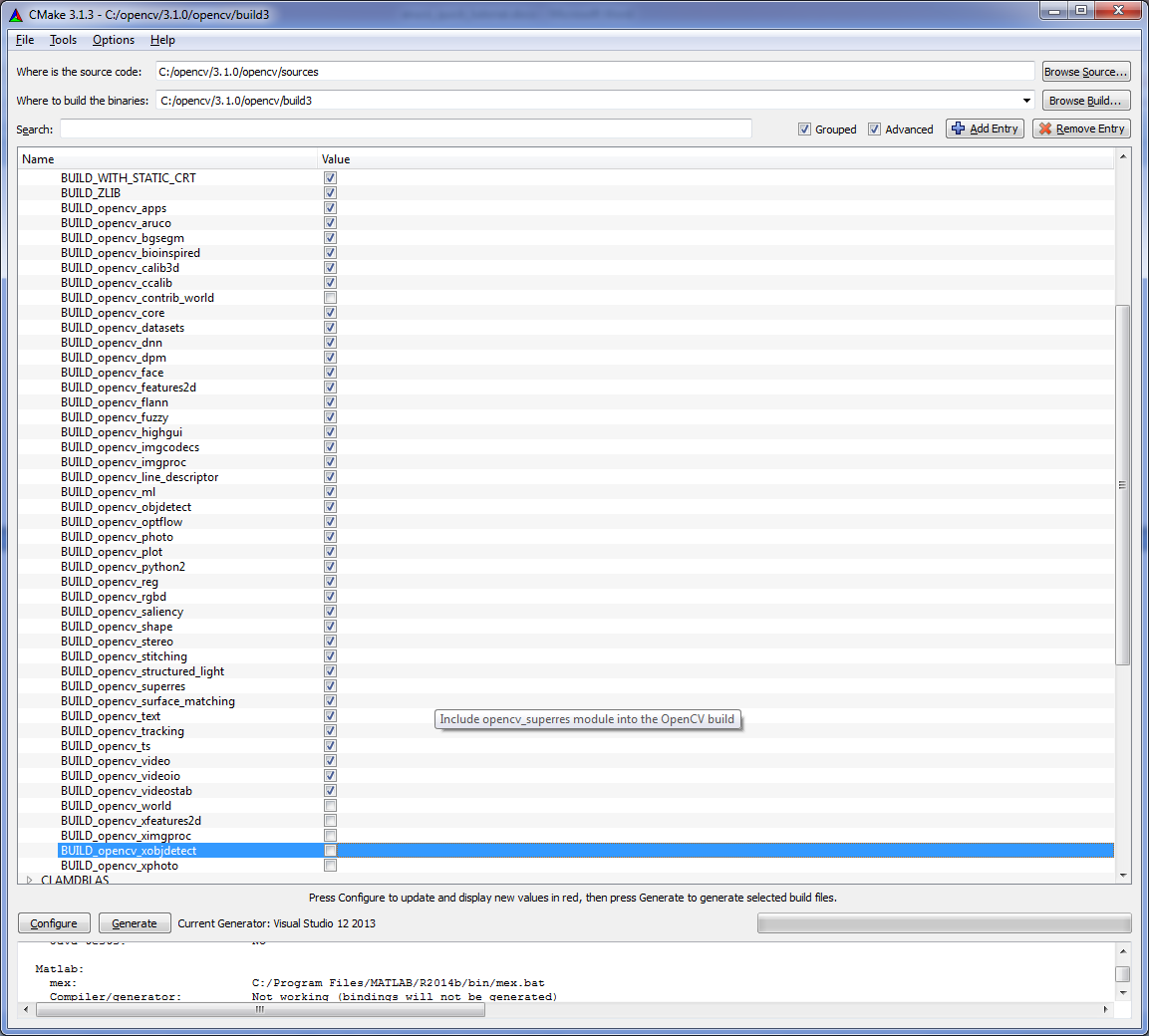
7) по окончанию конфигурации идем в области переменных компиляции в закладку OPENCV и задаем переменную OPENCV\_EXTRA\_MODULES\_PATH путь "..../opencv\_x.x/contribs/modules " (см. рис.)



Снова жмем Configure

8) По завершению конфигурации идем в закладку BUILD в переменных

выбираем все как показано на рисунке

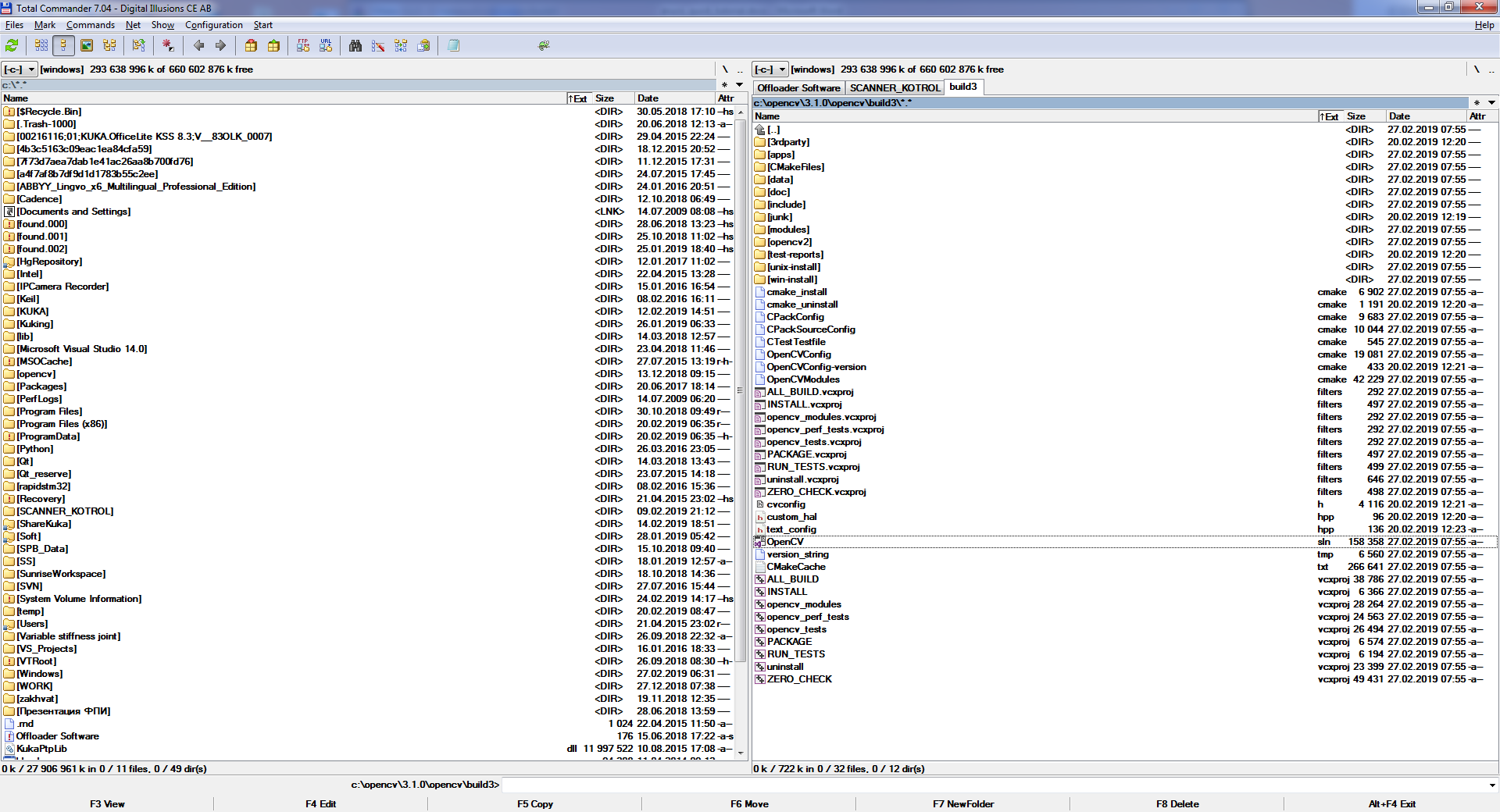


Главное не выбираем xfeatures2d иначе будут проблемы с компиляцией.

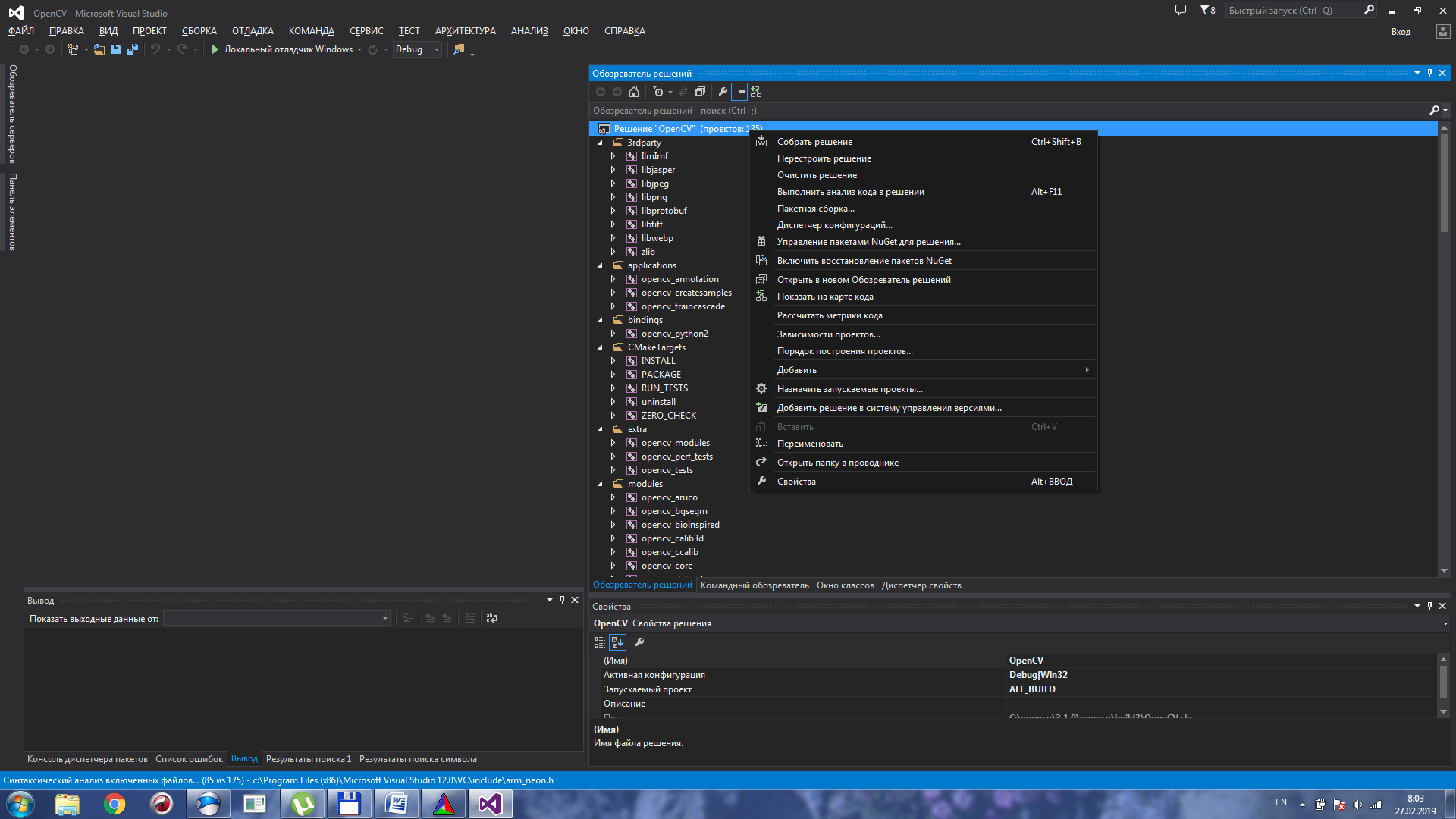
9) Жмем Generate. В высочевшем окне выбираем любимый компилятор (или тот, что имеется). В данном случае MSVS 2013.

Нажимаем продолжить и ждем генерации проекта.

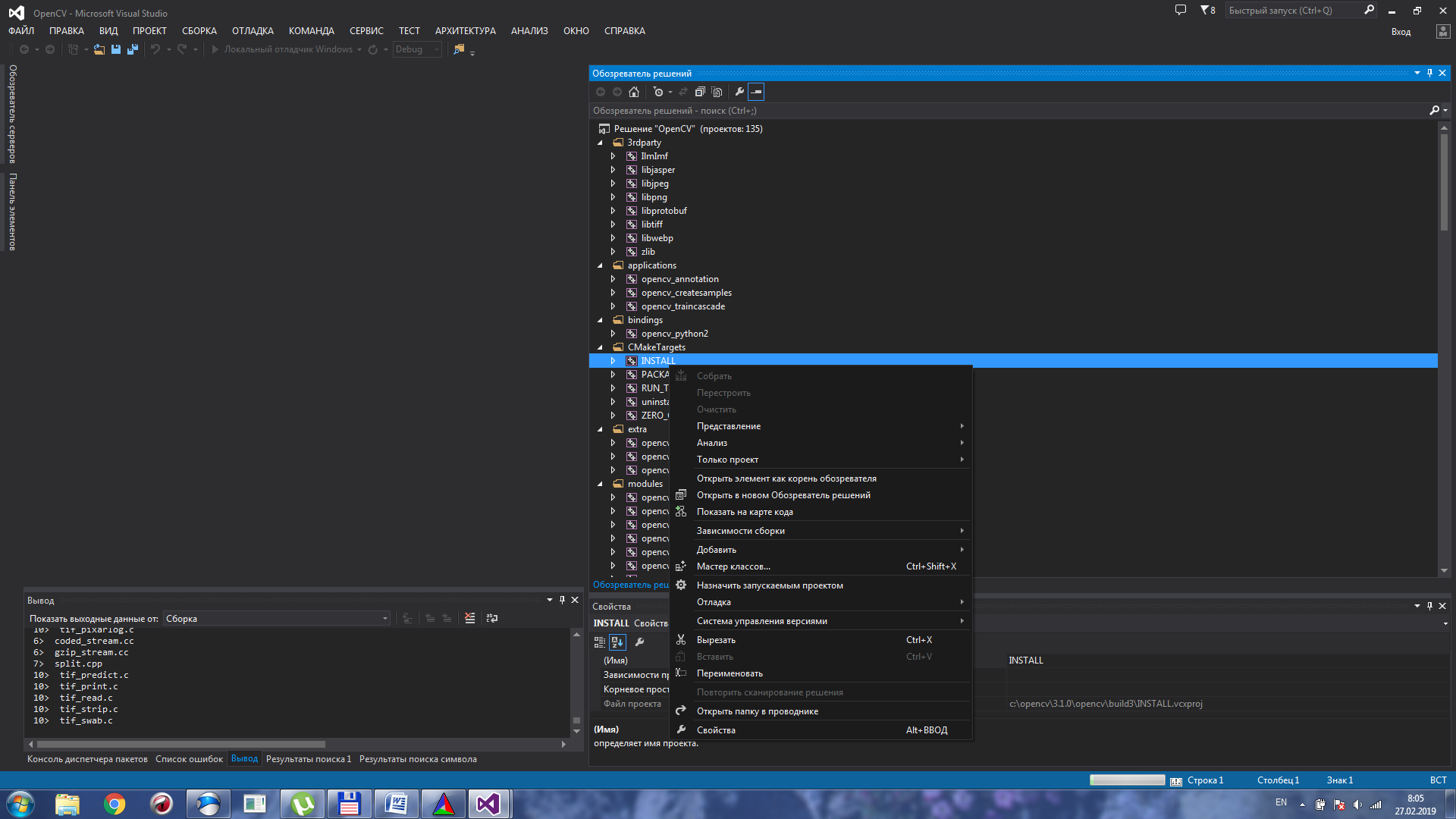
10) Идем в папку "..../opencv\_x.x/build "и открываем файл решения, под которое был сгенерирован проект. В данном случае OpenCV.sln



11) Запускаем сборку решения и ждем...бесконечно долго...



12)После окончания компиляции запускаем и компилируем проекта INSTALL.vcxproj. Если все прошло удачно, то скомпилированный проект и все \*.h файлы будут находиться в поддиректории "install".



Если что-то пошло не так: а) помощь друга  
 б) помощь преподавателя

**Использование aruco: продолжение**

После компиляции в поддиректории "bin\Release" директории решения (solution) будут находиться dll библиотеки и exe-файлы демо-программ. В рамках лаборатоной работы инересны следуюдие программы

* aruco-example-create\_marker.exe - создает и отображает (если указано при вызове) маркер с нужным номером и из нужного словаря
* aruco-example-detect\_markers.exe - демо-программа распознает маркеры из указанного словаря.
* aruco-example-create\_board\_charuco.exe – создает доску для калибровки камеры
* aruco-example-calibrate\_camera.exe - производит калибровку по созданной доске

Для начала необходимо создать маркер. В директории "bin\Release" из коммандной строки запустите следующую команду

*> aruco-example-detect\_markers.exe -d=10 --bb=1 --id=0 -ms=200 --si=true --wb=200 --up=true marker.jpg*

Для того, чтобы подробнее узнать о программе введите в командой строке

*> aruco-example-detect\_markers.exe -h*

Она выведет краткую справку в коммандной строке:

*-d* - номер словаря, из которого создается код маркера. доступны следующие коды словарей: DICT\_4X4\_50=0, DICT\_4X4\_100=1, DICT\_4X4\_250=2,DICT\_4

X4\_1000=3, DICT\_5X5\_50=4, DICT\_5X5\_100=5, DICT\_5X5\_250=6, DICT\_5X5\_1000=7, DICT\_

6X6\_50=8, DICT\_6X6\_100=9, DICT\_6X6\_250=10, DICT\_6X6\_1000=11, DICT\_7X7\_50=12,DICT

\_7X7\_100=13, DICT\_7X7\_250=14, DICT\_7X7\_1000=15, DICT\_ARUCO\_ORIGINAL = 16

Обозначение номера словаря строиться следующим образом "DICT\_@X@\_#", где @ - количество пикселей кода маркера на каждой из сторон (например 50х50), # - количество номеров маркеров в словаре

*--bb* – может быть больше или равно 1 (1 - по-умолчанию), задает колличество бит черной границы маркера.

--id - номер маркера в словаре

--ms - размер маркера в пикселях на геерируемом изображении

output\_file (в примере *marker.jpg*) - имея выходного файла

Для проверки созданного маркера:

1) распечатайте маркер

2) включите вебкамеру ноутбука (или ПК)

3) запустите команду из консоли

*> aruco-example-detect\_markers.exe* *-d=10 --dp="./detector\_params.yml" -c="./calibration.xml" -l=0.022*

В примере команды указаны

-d - номер словаря

--dp - конфирурационый файл с параметрами детектора

-с - файл с каблировочными параметрами камеры

-l - размер стороны маркера

Прежде, чем запустить данную команду нужно откалибровать саму камеру.

Для этого сначала сгенерируем калибровочную доску, запустив команду

*> aruco-example-calibrate\_camera\_charuco.exe -d=10 -h=10 -w=7 -l=50 -s=25 --si --tp=a checkboard\_aruco.jpg*

где

--bb (value:1) - число бит в границе

-d - номер словаря ( одно из значений (см. выше): DICT\_4X4\_50=0, DICT\_4X4\_100=1, DICT\_4X4\_250=2,DICT\_4X4\_1000=3, DICT\_5X5\_50=4, DICT\_5X5\_100=5, DICT\_5X5\_250=6, DICT\_5X5\_1000=7, DICT\_6X6\_50=8, DICT\_6X6\_100=9, DICT\_6X6\_250=10, DICT\_6X6\_1000=11, DICT\_7X7\_50=12,DICT\_7X7\_100=13, DICT\_7X7\_250=14, DICT\_7X7\_1000=15, DICT\_ARUCO\_ORIGINAL = 16)

-h - количество ячеек доски по вертикали

-m - дополнительная граница вокруг в маркеров (обычно равна --sl минус --ml)

--ml - размер маркера в пикселях

--si (true/false) – показать сгенерированное изрбражение

--sl - размер ячейки в пикселях с учетом размера маркера и промежутка между маркерами

-w - количество ячеек по горизонтали

outfile (значение: *checkboard\_aruco.jpg*) - файл, куда сохранить изображение

На выходе должно получиться такое изображение

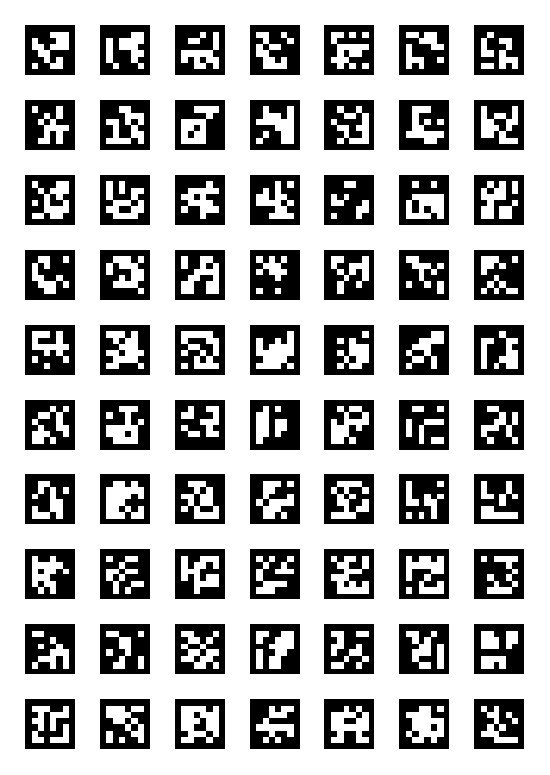


Рис. - Пример доски для калибровки с помощью Aruco

aruco-example-calibrate\_camera\_charuco.exe -d=10 --dp="./detector\_params.yml" -h=10 -w=7 -l=0.022 -s=0.0105 --zt=true calibration.xml

aruco-example-calibrate\_camera\_charuco.exe [params] outfile

-a

Fix aspect ratio (fx/fy) to this value

--ci (value:0)

Camera id if input doesnt come from video (-v)

-d - как и раньше - номер словаря

--dp - файл конфигурации для детекции маркеров

-h – число маркеров по высоте

-w - число маркеров по горизонтали

--zt (по умолчанию:false) - предположить отсутсвие тангенциальной дисторсии (в целях лабораторной поствить true)

-l - размер маркера в метрах

outfile (значене: calibration.xml) – выходной калибровочный файл. Это файл можно в последствии использовать для работы с камерой и определения положения маркеров. Он представляет из себя xml файл следующего содежрания

<?xml version="1.0"?>

<opencv\_storage>

<calibration\_time>"09/19/18 17:33:00"</calibration\_time>

<image\_width>640</image\_width>

<image\_height>480</image\_height>

<flags>8</flags>

<camera\_matrix type\_id="opencv-matrix">

<rows>3</rows>

<cols>3</cols>

<dt>d</dt>

<data>

7.5633851483481260e+002 0. 3.2333943745309250e+002 0.

7.5966100735852513e+002 2.7313186701148140e+002 0. 0. 1.</data></camera\_matrix>

<distortion\_coefficients type\_id="opencv-matrix">

<rows>1</rows>

<cols>5</cols>

<dt>d</dt>

<data>

2.3808046712327176e-002 6.8807433158169329e-001 0. 0.

-3.2517356625880911e+000</data></distortion\_coefficients>

<avg\_reprojection\_error>1.4126995753964209e+000</avg\_reprojection\_error>

</opencv\_storage>

Основными данными здесь являются теги:

camera\_matrix - матрица содержащая в себе фокусные расстояния и смещения оптической оси

distortion\_coefficients – коэффициенты в модели дисторсии

Файл detector\_params.yml содержит в себе настройки алгоритма детекции и распознования маркеров. Его содержание в рамках лабораторной

%YAML:1.0

nmarkers: 1024

adaptiveThreshWinSizeMin: 18

adaptiveThreshWinSizeMax: 25

adaptiveThreshWinSizeStep: 1

adaptiveThreshConstant: 7

minMarkerPerimeterRate: 0.03

maxMarkerPerimeterRate: 4.0

polygonalApproxAccuracyRate: 0.05

minCornerDistance: 10.0

minDistanceToBorder: 3

minMarkerDistance: 10.0

cornerRefinementWinSize: 5

cornerRefinementMaxIterations: 30

cornerRefinementMinAccuracy: 0.1

markerBorderBits: 1

perspectiveRemovePixelPerCell: 8

perspectiveRemoveIgnoredMarginPerCell: 0.13

maxErroneousBitsInBorderRate: 0.04

***Использование в коде***

Создаем дополнительную функцию, загружающую параметры детектора из файла

static bool readDetectorParameters(string filename, aruco::DetectorParameters &params) {

FileStorage fs(filename, FileStorage::READ);

if(!fs.isOpened())

return false;

fs["adaptiveThreshWinSizeMin"] >> params.adaptiveThreshWinSizeMin;

fs["adaptiveThreshWinSizeMax"] >> params.adaptiveThreshWinSizeMax;

fs["adaptiveThreshWinSizeStep"] >> params.adaptiveThreshWinSizeStep;

fs["adaptiveThreshConstant"] >> params.adaptiveThreshConstant;

fs["minMarkerPerimeterRate"] >> params.minMarkerPerimeterRate;

fs["maxMarkerPerimeterRate"] >> params.maxMarkerPerimeterRate;

fs["polygonalApproxAccuracyRate"] >> params.polygonalApproxAccuracyRate;

fs["minCornerDistanceRate"] >> params.minCornerDistanceRate;

fs["minDistanceToBorder"] >> params.minDistanceToBorder;

fs["minMarkerDistanceRate"] >> params.minMarkerDistanceRate;

fs["doCornerRefinement"] >> params.doCornerRefinement;

fs["cornerRefinementWinSize"] >> params.cornerRefinementWinSize;

fs["cornerRefinementMaxIterations"] >> params.cornerRefinementMaxIterations;

fs["cornerRefinementMinAccuracy"] >> params.cornerRefinementMinAccuracy;

fs["markerBorderBits"] >> params.markerBorderBits;

fs["perspectiveRemovePixelPerCell"] >> params.perspectiveRemovePixelPerCell;

fs["perspectiveRemoveIgnoredMarginPerCell"] >> params.perspectiveRemoveIgnoredMarginPerCell;

fs["maxErroneousBitsInBorderRate"] >> params.maxErroneousBitsInBorderRate;

fs["minOtsuStdDev"] >> params.minOtsuStdDev;

fs["errorCorrectionRate"] >> params.errorCorrectionRate;

return true;}

В области кода, где происходит непосредственно детекция, используем следующую конструкцию

aruco::Dictionary dictionary =

aruco::getPredefinedDictionary(aruco::PREDEFINED\_DICTIONARY\_NAME(dictionaryId));

// dictionaryId - номер словаря

Mat camMatrix, distCoeffs;

aruco::DetectorParameters detectorParams;

vector< int > ids;

vector< vector< Point2f > > corners, rejected;

vector< Vec3d > rvecs, tvecs;

// detect markers and estimate pose

aruco::detectMarkers(image, dictionary, corners, ids, detectorParams, rejected);

if(ids.size() > 0)

aruco::estimatePoseSingleMarkers(corners, markerLength, camMatrix, distCoeffs, rvecs, tvecs);

detectMarkers – осуществляет собственно детекцию маркеров. На входе и выходе

* image – собственно изображение с камеры
* dictionary - словарь (см выше в коде)
* corners - вектор векторов точек в (по 4 в каждом подвекторе) углов маркеров (выходная переменная)
* ids - массив найденных номеров маркеров (выходная переменная)
* detectorParams - структура типа aruco::DetectorParameters (заполняется функцией приведенной выше)
* rejected - вектор векторов точек, выброшенных в ходе отбраковки (выходная переменная)

Функция estimatePoseSingleMarkers осущесвляет определение положения маркеров, т.е. находит матрицу однородных преобразований из системы координат маркера к системе координат телекамеры. На входе

* corners - см. предыдущую функцию
* markerLength - размер стороны маркера в мерах
* camMatrix - калибровочная матрица камеры (см. выше)
* distCoeffs - матрица коэффициентов дисторсии (см. выше)
* rvecs - вектор содержащий повороты маркеров относительно камеры в форме "угол-вектор" (каждый элемент вектора содержит в себе трехмерный вектор отражающий по направлению – ось поворота, а по модулю – угол поворота вокруг этой оси; такое представление еще известно как представление Родригеса)
* tvecs – вектор трехмерных векторов смещений начал координат маркеров, относительно начала координат камеры

**Задание:**

1. Откалибровать собственную камеру при помощи доски aruco.

2. Сгенерировать маркер Aruco из понравившегося словоря

3. Написать программу, которая бы детектировала маркер на изображении и рисовать куб с основанием в виде маркера (куб должен быть спроекцирован на плоскость изображения и иметь различные цвета ребер). Используйте захват видеопотока с камеры (VideoCapture), чтобы получать изображения.