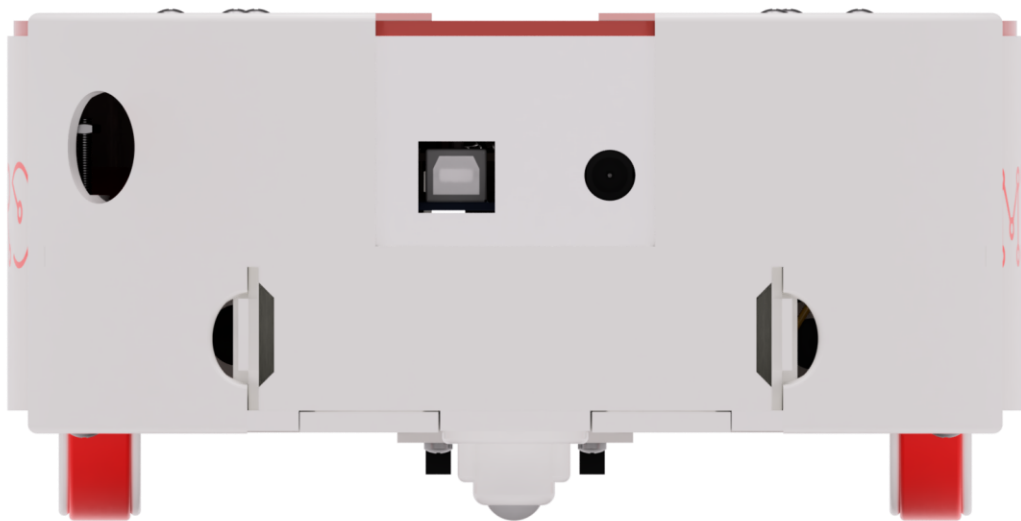


EDUCATION BOOK

LEVEL 3

outdated



УРОК 14. УСТАНОВКА И НАСТРОЙКА RASPBERRY PI

OpenCV и различные вспомогательные пакеты в сумме занимают достаточно много места. Настоятельно рекомендуется использовать SD-карту размером не менее 16 Гб.

Для начала нужно настроить сам RASPBERRY PI, чтобы к нему можно было подключиться. Существует два варианта настройки подключения.

Способ 1. Через HDMI-порт.

Подключаем к RASPBERRY PI монитор через HDMI-кабель, мышь, клавиатуру и запускаем RASPBERRY PI. В правом верхнем углу рабочего стола есть ярлык, обозначающий подключения к сети.



При подключении к сети нужно задать IP, чтобы можно было удобно подключаться к нему. Иначе придется искать IP. Об этом подробнее можно будет узнать при рассмотрении второго способа подключения.

Также нужно включить SSH, чтобы возможно было удаленное подключение. Для этого заходим в командную строку и вводим следующую команду:

```
sudo raspi-config
```

Заходим в пункт Interfacing Options и включаем SSH.

Теперь можно выключать RASPBERRY PI и отключать HDMI-кабель, мышь и клавиатуру.

Способ 2. Используя сетевой кабель.

Включим SSH без консоли. Для того создаем пустой файл без расширения и скидываем его в корень карты памяти SD с установленной RASPBIAN OS. Теперь можно вставлять карту в RASPBERRY PI.

Настройка компьютера для подключения к RASPBERRY PI.

После того как мы настроили, RASPBERRY PI, нужно к нему подключиться. Для этого потребуются следующие программы:

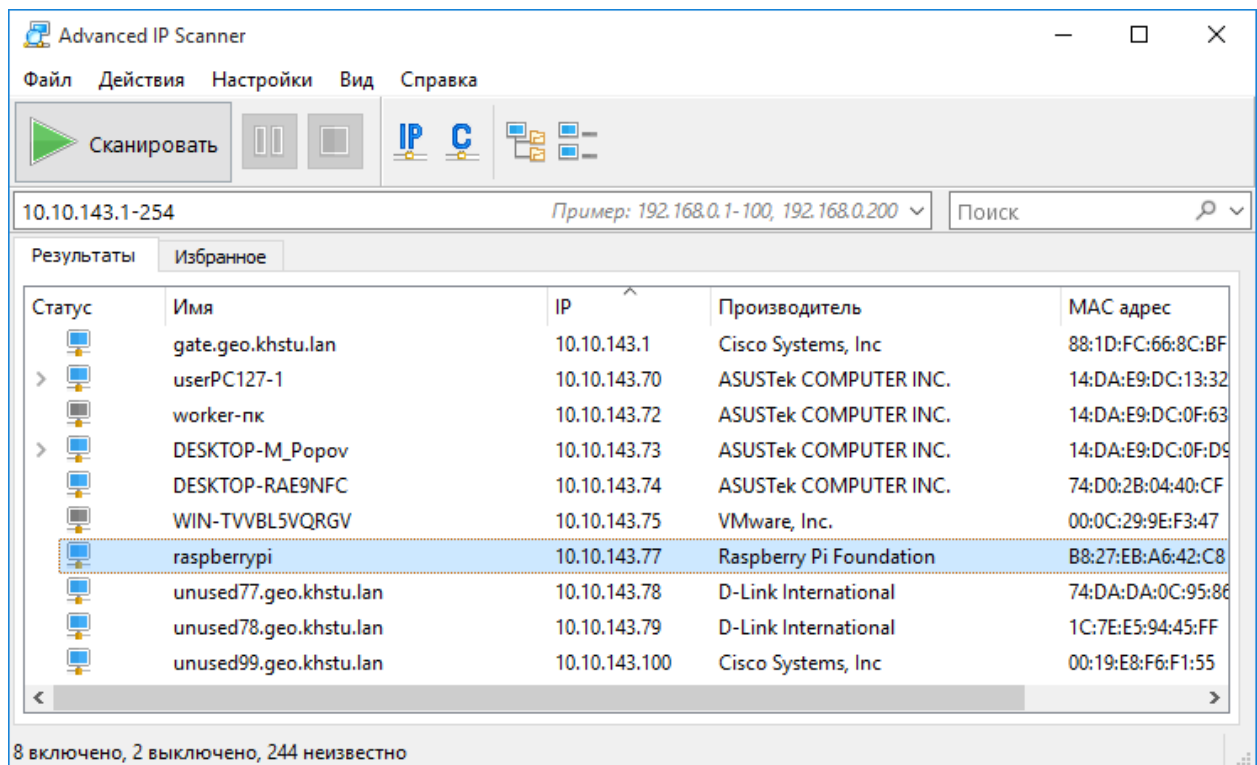
1. [Advanced IP Scanner](#) – сканирует все IP в заданной сети. Потребуется, если мы заранее не знаем IP-адрес RASPBERRY PI.

2. [PuTTY](#) – небольшая, но функциональная терминальная программа, необходима для подключения к RASPBERRY PI по протоколу SSH.

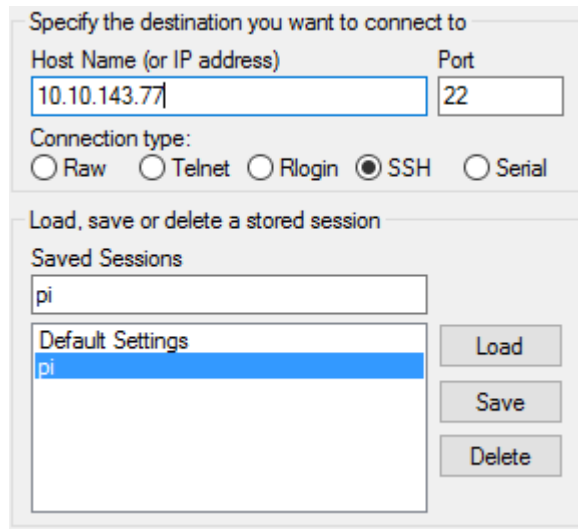
3. [VNC Viewer](#) – для подключения к графической оболочки RASPBIAN OS.

Эти программы требуется скачать и установить на рабочем компьютере.

Скорее всего Вам не известен IP-адрес платы RASPBERRY PI, поэтому запускаем программу Advanced IP Scanner. Вводим в ней диапазон IP-адресов нашей сети и нажимаем кнопку «Сканировать». В списке будут все устройства, подключенные к сети и их адреса. RASPBERRY PI среди них не сложно найти.



Подключимся к RASPBERRY PI по протоколу SSH. Для это запустим программу PuTTY.



Specify the destination you want to connect to

Host Name (or IP address) Port

Connection type:
☐ Raw ☐ Telnet ☐ Rlogin ☒ SSH ☐ Serial

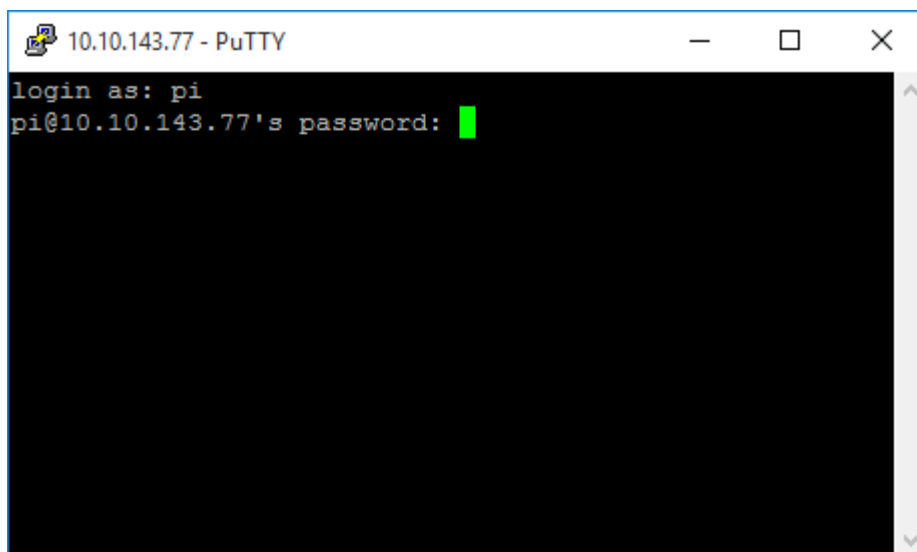
Load, save or delete a stored session

Saved Sessions

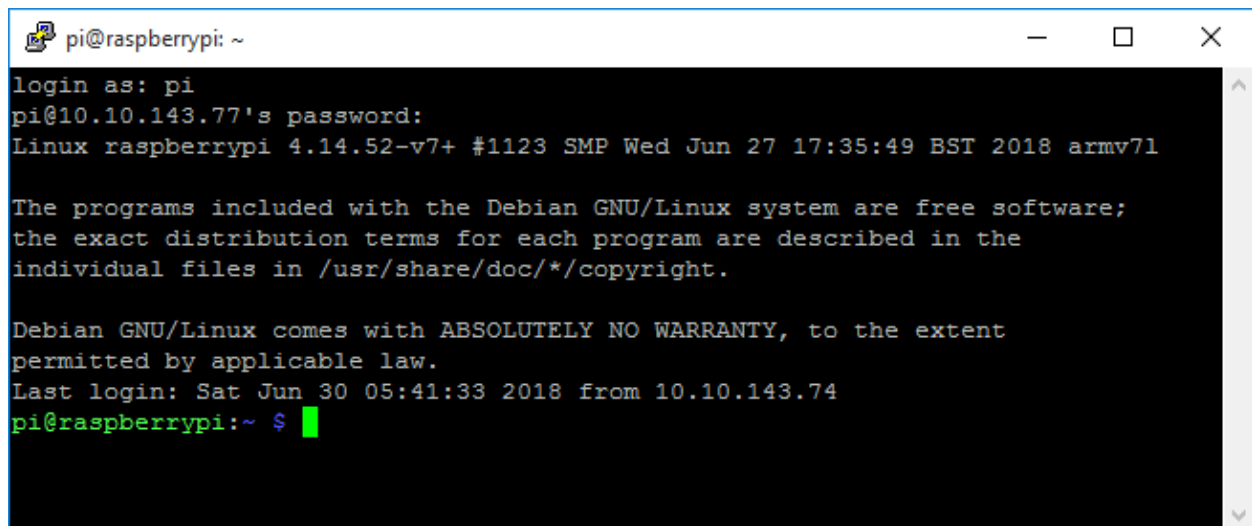
pi
Default Settings
pi

Load Save Delete

Сверху пишем IP-адрес RASPBERRY PI и выбираем тип подключения «SSH». Ниже можно ввести имя подключения и сохранить его, чтобы каждый раз не вводить IP заново. Далее снизу жмем кнопку Open и если все хорошо, то открывается консоль.



Тут требуется ввести логин и пароль. Логин – `pi`, а пароль – `raspberrry`. Если все прошло успешно, то будет видно примерно такое окно.



```
pi@raspberrypi: ~
login as: pi
pi@10.10.143.77's password:
Linux raspberrypi 4.14.52-v7+ #1123 SMP Wed Jun 27 17:35:49 BST 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Jun 30 05:41:33 2018 from 10.10.143.74
pi@raspberrypi:~ $
```

Далее нужно включить VNC и камеру. Заходим в терминал и вводим команду:

```
sudo raspi-config
```

Заходим в пункт Interfacing Options и включаем Camera, VNC. Заодно расширим файловую систему на весь объём SD-карты. Для этого нужно выйти из Interfacing Options и зайти в Advanced Options. Там выбираем опцию Expand Filesystem. Далее нужно перезагрузить RASPBERRY PI. Для этого воспользуемся следующей командой:

```
sudo reboot
```

Установка сервера VNC

После перезагрузки платы и входа в консоль, установку сервера VNC можно осуществить следующей командой:

```
sudo apt install xfce4 xfce4-goodies tightvncserver
```

После установки, запустить сервер VNC можно командой:

```
vncserver
```

Удачный запуск выглядит вот так:

```

pi@raspberrypi: ~
Copyright (C) 2002-2018 RealVNC Ltd.
RealVNC and VNC are trademarks of RealVNC Ltd and are protected by trademark
registrations and/or pending trademark applications in the European Union,
United States of America and other jurisdictions.
Protected by UK patent 2481870; US patent 8760366; EU patent 2652951.
See https://www.realvnc.com for information on VNC.
For third party acknowledgements see:
https://www.realvnc.com/docs/6/foss.html
OS: Linux 4.14.52 armv7l

On some distributions (in particular Red Hat), you may get a better experience
by running vncserver-virtual in conjunction with the system Xorg server, rather
than the old version built-in to Xvnc. More desktop environments and
applications will likely be compatible. For more information on this alternative
implementation, please see: https://www.realvnc.com/doclink/kb-546

Running applications in /etc/vnc/xstartup

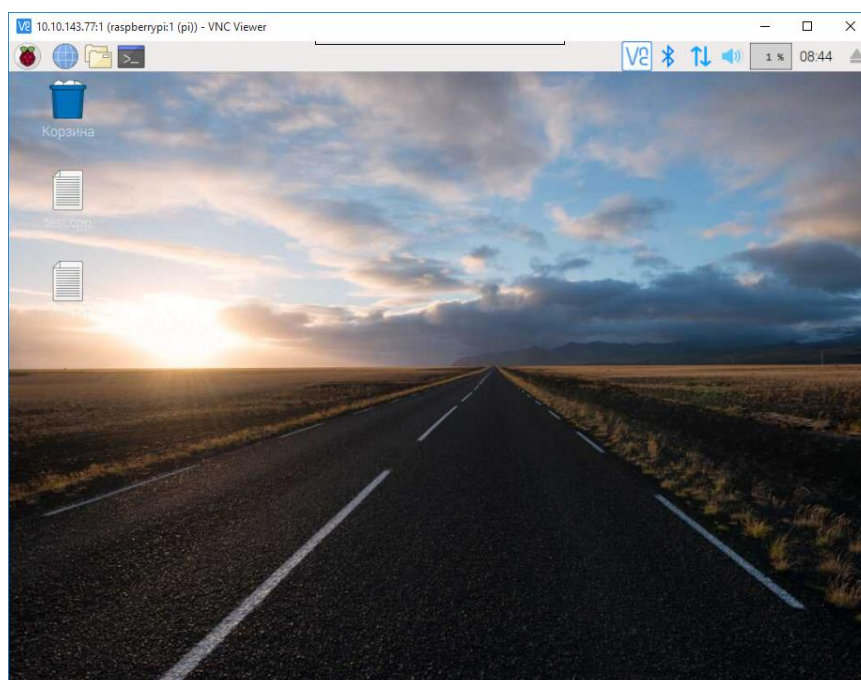
VNC Server catchphrase: "Vodka escape macro. Double April Tarzan."
signature: ed-20-d9-43-a9-5c-93-6f

Log file is /home/pi/.vnc/raspberrypi:1.log
New desktop is raspberrypi:1 (10.10.143.77:1)
pi@raspberrypi:~ $
  
```

Запоминаем снизу строку с IP-адресом и номером рабочего стола (10.10.143.77:1). После этого можно выйти из командной строки:

```
exit
```

Теперь можно запускать VNC Viewer на рабочем компьютере. В верхнем поле вводим IP-адрес и номер рабочего стола. Потребуется ввести тот же самый логин и пароль как при подключении через SSH.



Все настроено. Теперь каждый раз, при перезагрузке платы нужно будет выполнять команду на включение сервера (`vncserver`), чтобы подключиться по VNC.

УРОК 15. УСТАНОВКА OPENCV НА RASPBERRY PI

OpenCV - это популярная библиотека функций машинного зрения, которые позволяют роботам распознавать объекты окружающего мира. OpenCV применяют для навигации, обнаружения препятствий, распознавания лиц и жестов.

На этом уроке мы по шагам разберем установку OpenCV на одноплатный компьютер RASPBERRY PI.

Шаг 1. Свободное место

Если у вас SD-карта размером всего 8Гб, можно удалить что-нибудь лишнее, например пакет wolfram-engine. Делается это командой:

```
$ sudo apt-get purge wolfram-engine
```

Также можно удалить офисный пакет:

```
$ sudo apt-get purge libreoffice*
```

Эти операции освободят дополнительные 900 Мб места.

Шаг 2. Установка зависимостей

Для полноценной работы с OpenCV нам потребуется обновить существующие пакеты и установить ряд новых. Начнем с обновления.

```
$ sudo apt-get update  
$ sudo apt-get upgrade
```

В зависимости от скорости интернета, данные операции займут около 5-10 минут. Далее устанавливаем в систему cmake и еще несколько полезных пакетов:

```
$ sudo apt-get install build-essential cmake pkg-config
```

Следом пакеты для работы с известными форматами изображений:

```
$ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
```

Пакеты для работы с видео:

```
$ sudo apt-get install libxvidcore-dev libx264-dev libopencore-amrnb-dev  
libopencore-amrwb-dev libavresample-dev x264 v4l-utils  
  
$ cd /usr/include/linux  
$ sudo ln -s -f ../libv4l1-videodev.h videodev.h  
$ cd $pwd
```

Пакеты для создания простых экранных форм:

```
$ sudo apt-get install libgtk2.0-dev libgtk-3-dev
```


Специальные ускоренные операции над матрицами.

```
$ sudo apt-get install libatlas-base-dev gfortran
```

Пакеты для работы с камерой и кодирования видео:

```
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev  
libv4l-dev v4l-utils
```

Шаг 3. Загрузка OpenCV из репозитория

Сначала скачаем архив с самим OpenCV. Для этого перейдем в папку /home/pi/Downloads:

```
$ cd ~/Downloads
```

Загрузим архив с помощью wget и распакуем unzip-ом:

```
$ wget -O opencv.zip https://github.com/opencv/opencv/archive/master.zip  
$ unzip opencv.zip
```

Следом скачаем пакет с дополнительными модулями - opencv_contrib.

```
$ wget -O opencv_contrib.zip  
https://github.com/opencv/opencv_contrib/archive/master.zip  
$ unzip opencv_contrib.zip
```

Шаг 4. Компиляция и установка OpenCV

Зайдем в папку с исходными кодами OpenCV и выполним команды:

```
$ cd ~/Downloads/opencv-master  
$ mkdir build  
$ cd build  
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \  
-D CMAKE_INSTALL_PREFIX=/usr/local \  
-D INSTALL_C_EXAMPLES=ON \  
-D BUILD_EXAMPLES=ON \  
-D OPENCV_EXTRA_MODULES_PATH=~/Downloads/opencv_contrib-master/modules \  
-D WITH_CUDA=OFF \  
-D WITH_GTK=ON \  
-D BUILD_TESTS=OFF \  
-D BUILD_PERF_TESTS= OFF ..
```

В конце процедуры появится список компонентов OpenCV, готовых к установке.

Ну а теперь само интересное — сборка бинарных файлов. Не выходя из текущей папки, запускаем команду make:

```
$ make
```

Сборка займет около трех часов и должна завершиться примерно таким вот отчетом:

```

[ 98%] Built target tutorial_npr_demo
Scanning dependencies of target tutorial_planar_tracking
[ 98%] Building CXX object samples/cpp/MakeFiles/tutorial_planar_tracking.dir/tutorial_code/features2D/AKAZE_tracking/plano
r_tracking.cpp.o
Linking CXX executable ../../bin/cpp-tutorial-objectDetection
Linking CXX executable ../../bin/cpp-tutorial-objectDetection2
Linking CXX executable ../../bin/cpp-tutorial-non_linear_svms
[ 98%] Built target tutorial_objectDetection
[ 98%] Built target tutorial_objectDetection2
Scanning dependencies of target tutorial_pointPolygonTest_demo
Scanning dependencies of target tutorial_video-input-psnr-ssim
[ 98%] [ 98%] Building CXX object samples/cpp/MakeFiles/tutorial_pointPolygonTest_demo.dir/tutorial_code/ShapeDescriptors/p
ointPolygonTest_demo.cpp.o
Built target tutorial_non_linear_svms
[ 98%] Building CXX object samples/cpp/MakeFiles/tutorial_video-input-psnr-ssim.dir/tutorial_code/HighGUI/video-input-psnr-
ssim/video-input-psnr-ssim.cpp.o
Scanning dependencies of target tutorial_video-write
[100%] Building CXX object samples/cpp/MakeFiles/tutorial_video-write.dir/tutorial_code/HighGUI/video-write/video-write.cpp
.o
Linking CXX executable ../../bin/cpp-tutorial-pointPolygonTest_demo
Linking CXX executable ../../bin/cpp-tutorial-video-write
[100%] Built target tutorial_pointPolygonTest_demo
[100%] Built target tutorial_video-write
Linking CXX executable ../../bin/cpp-tutorial-video-input-psnr-ssim
Linking CXX executable ../../bin/cpp-tutorial-planar_tracking
[100%] Built target tutorial_video-input-psnr-ssim
[100%] Built target tutorial_planar_tracking

real    72m12.945s
user    228m31.480s
sys     6m20.110s
(cv) pi@raspberrypi:~/opencv-3.1.0/build $

```

Примечание. Сборка такой большой библиотеки – сложная процедура. Вполне возможны зависания RASPBERRY PI в процессе. В этом случае, перезагрузите плату и повторите процесс сборки все той же командой make.

Последнее, что нужно сделать — установить собранные бинарные файлы библиотеки:

```

$ sudo make install
$ sudo ldconfig

```

Вот и всё — OpenCV установлен успешно! В следующих уроках мы будем разбираться с обработкой изображения с видекамеры, применительно к робототехнике.

В уроке использованы материалы:

<http://robotclass.ru/articles/raspberrypi-3-opencv-3-install/>

УРОК 16. ПЕРВАЯ ПРОГРАММА С OPENCV

Для начала нужно создать Makefile. Makefile - это файл, содержащий набор инструкций, используемых утилитой make в инструментарии автоматизации сборки. Этот файл мы будем использовать с каждой программой, поэтому подключим как можно больше библиотек.

Информационный блок

Остановимся подробнее на модулях библиотеки OpenCV:

- core (Core functionality) – основной функционал OpenCV
- imgproc (Image processing) – множество операции над изображением
- higghui (High-level GUI) – управление интерфейсом с помощью мыши и кнопок
- calib3d (Camera Calibration and 3D Reconstruction) – калибровка камеры
- features2d (2D Features Framework) – детектирование ключевых точек.
- imgcodecs (Image file reading and writing) – чтение и запись изображения.
- video (Video Analysis) – анализ движений и отслеживание объекта.
- videoio (Video I/O) – считывание и запись видео
- ml (Machine Learning) – машинное обучение
- objdetect (Object Detection) – обнаружение объектов

Текст Makefile'a:

```
CC:= g++
CFLAGS:= -I/usr/local/include/opencv -L/usr/local/lib
OBJECTS:=
LIBRARIES:= -lopencv_core -lopencv_imgproc -lopencv_highgui -
lopencv_calib3d -lopencv_features2d -lopencv_flann -lopencv_imgcodecs -
lopencv_video -lopencv_videoio -lopencv_ml -lopencv_objdetect

.PHONY: all clean

all: test

test:
$(CC) $(CFLAGS) -o test test.cpp $(LIBRARIES)

clean:
rm -f *.o
```

ВНИМАНИЕ! В Makefile важны символы табуляции. В приведенном выше тексте этого файла есть символ табуляции в начале строк:

```
$(CC) $(CFLAGS) -o test test.cpp $(LIBRARIES)
```

и

```
rm -f *.o
```

Программу пишем в файле `test.cpp`.

Код программы

```
#include <stdlib.h>
#include <stdio.h>

#include <cv.h>
#include <highgui.h>
#include <opencv2/highgui.hpp>

using namespace std;
using namespace cv;

Mat image;

int main(int argc, char* argv[])
{
    // имя картинки
    string filename = "Image0.jpg";

    // получаем картинку
    image = imread(filename, IMREAD_COLOR);

    // окно для отображения картинки
    namedWindow("original", WINDOW_AUTOSIZE);

    // показываем картинку
    imshow("original", image);

    // выводим в консоль информацию о картинке
    printf("[i] image: %s\n", filename);
    printf("[i] channels:  %i\n", image.channels());
    printf("[i] width:      %i pixels\n", image.rows);
    printf("[i] height:      %i pixels\n", image.cols);

    // ждём нажатия клавиши
    waitKey(0);
    // удаляем окно
    destroyWindow("original");
    return 0;
}
```

Вот что происходит в консоли:

```

pi@raspberrypi: ~/example_opencv/Images
Файл Правка Вкладки Справка
pi@raspberrypi:~/example_opencv/Images $ make
g++ -I/usr/local/include/opencv -L/usr/local/lib -o test test.cpp -lopencv_core
-lopencv_imgproc -lopencv_highgui -lopencv_calib3d -lopencv_features2d -lopencv
flann -lopencv_imgcodecs
test.cpp: In function 'int main(int, char**)':
test.cpp:11:26: warning: ISO C++ forbids converting a string constant to 'char*'
[-Wwrite-strings]
    char* filename = "Image.jpg";
                        ^~~~~~
pi@raspberrypi:~/example_opencv/Images $ ./test
[i] image: Image.jpg
[i] channels: 3
[i] pixel depth: 8 bits
[i] width: 800 pixels
[i] height: 500 pixels
[i] image size: 1200000 bytes
[i] width step: 2400 bytes
pi@raspberrypi:~/example_opencv/Images $

```

Пояснения к коду

Mat - тип данных (матрица), для хранения изображений.

Mat imread (const string filename, int flags = IMREAD_COLOR) - функция для загрузки изображения из файла. Первый аргумент имя файла, а второй конвертация изображения в нужный вид. Флаг IMREAD_COLOR означает, что изображение будет загружено в виде в 3-канального цветного изображения BGR.

void namedWindow (const string& name, int flags = WINDOW_AUTOSIZE) - создает окно, в которое будет выводиться изображение. Первый аргумент – имя окна. Второй аргумент – размер окна. WINDOW_AUTOSIZE - позволяет окну подстроиться под размеры изображения.

void imshow (const string& name, InputArray mat) - отображение изображения в окне. Первый аргумент – название окна. Второй аргумент – изображение.

int waitKey(int delay=0) - ожидание. Параметр delay указывается в мс. Если delay==0, то ожидание до нажатия клавиши.

void destroyWindow(const string& name) - эта функция уничтожает окно и освобождает выделенную память.

Практическое задание

1. Загрузите изображения в оттенках серого. Для этого нужно найти нужный флаг для функции imread в официальной [документации](#).

УРОК 17. ВЫВОД ВИДЕО С КАМЕРЫ РОБОТА

Для того чтобы работала камера нужно ее подключить в терминале:

```
$ sudo modprobe bcm2835-v4l2
```

Чтобы не делать это каждый раз при запуске ОС, можно добавить ее в автозапуск. Для этого нужно перейти в файл `/etc/modules-load.d/modules.conf` и добавить строчку с именем камеры `bcm2835-v4l2`. Зайти в режим редактирования файла можно командой:

```
$ sudo nano /etc/modules-load.d/modules.conf
```

Дописываем строку с названием камеры, используем сочетание клавиш `ctrl+x`, затем подтверждаем сохранение файла и жмем `Enter`.

Код программы

```
#include <stdlib.h>
#include <stdio.h>

#include <cv.h>
#include <highgui.h>
#include <opencv2/highgui.hpp>

using namespace std;
using namespace cv;

int main(int argc, char* argv[])
{
    // Получаем любую подключённую камеру
    VideoCapture capture = VideoCapture(0);
    if (!capture.isOpened()) return -1;

    // Можно вручную задать разрешение
    capture.set(CAP_PROP_FRAME_WIDTH, 640);
    capture.set(CAP_PROP_FRAME_HEIGHT, 480);

    namedWindow("capture", WINDOW_AUTOSIZE);

    printf("[i] press Enter for capture image and Esc for quit!\n\n");

    int counter = 0;
    char filename[512];
    Mat frame;

    while (true)
    {
        // Получаем кадр
        capture >> frame;

        // Отображаем кадр
        imshow("capture", frame);

        char c = waitKey(33);
        if (c == 27) // нажата ESC
        {
            break;
        }
    }
}
```

```

    }
    else if (c == 13) // нажата Enter
    {
        // Сохраняем кадр в файл
        sprintf(filename, "Image%d.jpg", counter);
        printf("[i] capture... %s\n", filename);
        imwrite(filename, frame);
        counter++;
    }
}
// Освобождаем ресурсы
capture.release();
destroyWindow("capture");

return 0;
}

```

Пояснения к коду

`VideoCapture` - класс для захвата видео.

`VideoCapture(int index)` конструктор для открытия камеры по индексу. 0 – означает открытие самой первой подключенной камеры.

`bool isOpened` функция позволяет проверить инициализацию видеозахвата.

`bool set(int propid, double value)` функция позволяет задавать различные настройки для видеопотока. В данном случае используется для задания разрешения. Первый параметр задает свойство, а второй значение.

`bool imwrite(const string& filename, InputArray img, const std::vector<int> & params =std::vector<int>())` функция сохраняет изображение в указанный файл. Третий параметр (необязательный) задает специфичные для формата параметры: степень сжатия, качество, яркость, цветность.

`void release()` функция закрывает видеофайл или устройство захвата.

В качестве параметра функции `waitKey(33)` передается число 33 – время ожидания между кадрами (1000 мс делим на 30 кадров/секунду и получаем 33.(3) мс).

Практическое задание

1. Откройте видеопоток с разрешением формата HD.

УРОК 18. ПОИСК ОБЪЕКТА ПО ЦВЕТУ. НАСТРОЙКА ФИЛЬТРА HSV.

Цветовая модель HSV позволяет более точно настраивать цвета, поэтому она популярна в компьютерном зрении. Следующая программа нам позволяет вручную настроить бинарную маску, чтобы можно было распознавать объект по цвету. Для того чтобы фильтр заработал нужно подвигать либо 3 верхние границы, либо 3 нижние.

Код программы

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

#include <opencv/highgui.h>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace std;
using namespace cv;

Mat image;

// для хранения каналов HSV
Mat channels[3];
Mat hsv;
Mat h_plane;
Mat s_plane;
Mat v_plane;

// для хранения каналов HSV после преобразования
Mat h_range;
Mat s_range;
Mat v_range;

// значения для ползунков
int Hmin = 0;
int Hmax = 255;

int Smin = 0;
int Smax = 255;

int Vmin = 0;
int Vmax = 255;

int HSVmax = 255;

//
// функции-обработчики ползунков
//
void myTrackbarHmin(int pos, void*) {
    Hmin = pos;
    inRange(h_plane, Scalar(Hmin), Scalar(Hmax), h_range);
}

void myTrackbarHmax(int pos, void*) {
    Hmax = pos;
    inRange(h_plane, Scalar(Hmin), Scalar(Hmax), h_range);
}
```



```

void myTrackbarSmin(int pos, void*) {
    Smin = pos;
    inRange(s_plane, Scalar(Smin), Scalar(Smax), s_range);
}

void myTrackbarSmax(int pos, void*) {
    Smax = pos;
    inRange(s_plane, Scalar(Smin), Scalar(Smax), s_range);
}

void myTrackbarVmin(int pos, void*) {
    Vmin = pos;
    inRange(v_plane, Scalar(Vmin), Scalar(Vmax), v_range);
}

void myTrackbarVmax(int pos, void*) {
    Vmax = pos;
    inRange(v_plane, Scalar(Vmin), Scalar(Vmax), v_range);
}

int main(int argc, char* argv[])
{
    // имя картинки
    string filename = "Image1.jpg";

    // получаем картинку
    image = imread(filename, IMREAD_COLOR);

    // создаём картинки

    hsv.create(image.rows, image.cols, CV_8UC3);
    h_plane.create(image.rows, image.cols, CV_8UC1);
    s_plane.create(image.rows, image.cols, CV_8UC1);
    v_plane.create(image.rows, image.cols, CV_8UC1);
    h_range.create(image.rows, image.cols, CV_8UC1);
    s_range.create(image.rows, image.cols, CV_8UC1);
    v_range.create(image.rows, image.cols, CV_8UC1);

    // конвертируем в HSV
    cvtColor(image, hsv, CV_BGR2HSV);

    // разбиваем на отдельные каналы
    split(hsv, channels);
    h_plane = channels[0];
    s_plane = channels[1];
    v_plane = channels[2];

    // окна для отображения картинки
    namedWindow("original", WINDOW_AUTOSIZE);
    namedWindow("mask", WINDOW_AUTOSIZE);
    namedWindow("other", WINDOW_AUTOSIZE);

    createTrackbar("Hmin", "other", &Hmin, HSVmax, myTrackbarHmin);
    createTrackbar("Hmax", "other", &Hmax, HSVmax, myTrackbarHmax);
    createTrackbar("Smin", "other", &Smin, HSVmax, myTrackbarSmin);
    createTrackbar("Smax", "other", &Smax, HSVmax, myTrackbarSmax);
    createTrackbar("Vmin", "other", &Vmin, HSVmax, myTrackbarVmin);
    createTrackbar("Vmax", "other", &Vmax, HSVmax, myTrackbarVmax);

    while (true)

```

```
{
    // складываем бинарные изображения слоев HSV
    Mat m; // дополнительная переменная для сложения
    Mat mask;
    h_range.copyTo(m, s_range);
    m.copyTo(mask, v_range);

    Mat result;
    // наложим маску на первоначальное изображение
    image.copyTo(result, mask);

    imshow("other", 0);
    imshow("mask", mask);
    imshow("original", result);

    char c = waitKey(33);
    if (c == 27) { // если нажата ESC - выходим
        break;
    }
}
// удаляем окна
destroyAllWindows();
return 0;
}
```

Пояснения к коду

`void inRange(InputArray src, InputArray lowerb, InputArray upperb, OutputArray dst)` проверяет, что элемент массива лежит между двух скаляров. В данном случае используется, чтобы изменить граничные значения слоя.

`void create(int rows, int cols, int type)` функция используется для создания типа данных `Mat` с заданными параметрами. Задается количество строк и столбцов, а третьим параметром задается битность изображения (глубина цвета) и количество каналов. `CV_8UC3` означает, 8-битное беззнаковое (`unsigned`) изображение с 3-мя каналами. С одним каналом значит, что изображение бинарное, т.е. имеет только белые и черные пиксели.

`void cvtColor(InputArray src, OutputArray dst, int code)` конвертирует изображение из одного цветового пространства в другое: `src` - исходное изображение, `dst` - получаемое изображение, `code` - параметр, указывающий тип конвертации.

`split(InputArray m, OutputArrayOfArrays mv)` разделяет многоканальный массив на несколько одноканальных массивов.

`createTrackbar(const String& trackbarname, const String& winname, int* value, int count, TrackbarCallback)` создание трекбара. Задается имя трекбара, имя окна, в котором он будет, создаваться, ссылка на значение ползунка, максимальное положение ползунка. Последний аргумент - указатель на функцию, которая вызывается по каждому изменению ползунка.

`void copyTo(OutputArray m, InputArray mask)` копирует изображение, которое пишется перед `copyTo` в изображение `m` с наложением маски `mask`. Т.е. в коде мы на слой `h_range` накладываем слой `s_range` и в результирующее изображение накладываем `v_range`. Таким образом, у нас получается маска, в которой сложены все 3 слоя HSV в один. Затем маска накладывается на

исходное изображение, и мы видим только те участки изображения, где маска имеет белый пиксель.

УРОК 19. РАБОТА С МЫШЬЮ В ОКНЕ OPENCV

Следующая программа выводит цвет текущего пикселя в правом верхнем углу. Пиксель выбирается нажатием левой кнопки мыши (ЛКМ).

Код программы:

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

#include <opencv/highgui.h>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace std;
using namespace cv;

Mat image;

void drawRectangle(int x, int y)
{
    Mat copy = image;
    Vec3b color = copy.at<Vec3b>(Point(x, y));
    rectangle(copy, Point(copy.rows - 50, 0), Point(copy.rows, 50),
Scalar(color), -1);
    printf("%d x %d\n", x, y);
}

// обработчик событий от мышки
void myMouseCallback(int event, int x, int y, int flags, void* param)
{
    switch (event) {
        case CV_EVENT_MOUSEMOVE:
            break;

        case CV_EVENT_LBUTTONDOWN:
            drawRectangle(x, y);
            break;

        case CV_EVENT_LBUTTONUP:
            break;
    }
}

int main(int argc, char* argv[])
{
    // имя картинки
    string filename = "Image1.jpg";

    // получаем картинку
    image = imread(filename, IMREAD_COLOR);

    // окно для отображения картинки
    namedWindow("original", WINDOW_AUTOSIZE);

    // задаём обработчик мышки
    setMouseCallback("original", myMouseCallback);

    while (1)
```

```
{  
    // показываем картинку  
    imshow("original", image);  
  
    char c = waitKey(33);  
    if (c == 27) { // если нажата ESC - выходим  
        break;  
    }  
}  
  
// удаляем окно  
destroyWindow("original");  
return 0;  
}
```

Пояснения к коду

`void setMouseCallback(const String& winname, MouseCallback onMouse)` первым параметром задается имя окна, а вторым — функция, которая будет обрабатывать события мыши.

`void MouseCallback (int event, int x, int y, int flags, void* param)` так выглядит структура функции-обработчика для мыши. Первый параметр — это событие мыши (`MouseEventTypes`) туда входят всевозможные действия мыши. Следующие два параметра это координаты мыши в окне, к которому привязана эта функция. `flags` дополнительные опции. `param` дополнительные параметры, которые можно передавать в функцию.

`Vec3b color = copy.at<Vec3b>(Point(x, y));` Эта строчка берет цвет пикселя из изображения `copy` и координат `(x, y)`

`rectangle(InputOutputArray img, Point pt1, Point pt2, const Scalar & color, int thickness)` рисует прямоугольник по 2-м точкам, последний параметр `thickness` указывает толщину линии. Отрицательное значение, означает закрашивание прямоугольника.

Практическое задание

1. Сделайте так, чтобы отрисовка прямоугольника происходила по перемещению мыши.
2. Сделайте так, чтобы отрисовка прямоугольника происходила по отпусканию ЛКМ.

УРОК 20. НАСТРОЙКА МАСКИ ДЛЯ ВЫДЕЛЕНИЯ ОБЛАСТИ ИНТЕРЕСА С ПОМОЩЬЮ МЫШИ

Программа создает бинарную маску по выбранному цвету пикселя. Затем происходит морфологическая операция растягивание, чтобы объекты не разрывались в маске неточностями изображения. После этого выполняется поиск контуров, а потом они выделяются зелеными прямоугольниками.

Код программы

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

#include <opencv/highgui.h>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace std;
using namespace cv;

Mat frame;

//зададим начальные параметры для фильтра
//это примерно оранжевый цвет
Scalar scalMin = Scalar(120, 83, 126);
Scalar scalMax = Scalar(183, 190, 249);

// цвет пикселя
uchar h = 0;
uchar s = 0;
uchar v = 0;

void normal(uchar Nh, uchar Ns, uchar Nv)
{
    if (Nh < 20)
        h = 20;
    if (Ns < 40)
        s = 40;
    if (Nv < 40)
        v = 40;
    if (Nh + 20 > 255)
        h = 255;
    if (Ns + 40 > 255)
        s = 255;
    if (Nv + 40 > 255)
        v = 255;
}

// обработчик событий от мышки
void myMouseCallback(int event, int x, int y, int flags, void* param)
{
    if (event == CV_EVENT_LBUTTONDOWN) {
        Mat copy;
        frame.copyTo(copy);
        cvtColor(frame, copy, CV_BGR2HSV);
        printf("x = %d y = %d \n", x, y);
        Vec3b color = copy.at<Vec3b>(Point(x, y));
        h = color.val[0];
    }
}
```

```

        s = color.val[1];
        v = color.val[2];
        printf("hsv = (%u, %u, %u) \n", h, s, v);
        // нормализация, чтобы не выходило за границы цвета
        normal(h, s, v);
        scalMin = Scalar(h - 20, s - 40, v - 40);
        scalMax = Scalar(h + 20, s + 40, v + 40);
    }
}

Mat treatment(const Mat& original)
{
    Mat copy;
    Mat hsv;
    Mat bin;
    original.copyTo(copy);
    original.copyTo(hsv);
    original.copyTo(bin);

    // конвертируем в HSV
    cvtColor(original, hsv, CV_BGR2HSV);

    // установка значений HSV
    inRange(hsv, scalMin, scalMax, bin);

    // создаём структурирующий элемент
    Mat element = getStructuringElement(MORPH_ELLIPSE, Size(3, 3), Point(1,
1));
    // выполняем операцию растягивание
    dilate(bin, bin, element, Point(-1, -1));

    vector<vector<Point>> > contours;
    // поиск контуров
    findContours(bin, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
    // аппроксимация многоугольников с точностью +-3 и утверждаем, что кривая
должна быть замкнутой
    // после этого мы найдем ограничивающий прямоугольник для каждого
многоугольника и сохраним его для boundRect.
    vector<vector<Point>> > contours_poly(contours.size());
    vector<Rect> boundRect(contours.size());
    for (size_t i = 0; i < contours.size(); i++) {
        approxPolyDP(contours[i], contours_poly[i], 3, true);
        boundRect[i] = boundingRect(contours_poly[i]);
    }
    for (size_t i = 0; i < contours.size(); i++) {
        rectangle(copy, boundRect[i].tl(), boundRect[i].br(), Scalar(0, 255,
0), 2);
    }
    return copy;
}

int main(int argc, char** argv)
{
    VideoCapture capture = VideoCapture(0);
    if (!capture.isOpened()) return -1;

    namedWindow("CAM Capture", WINDOW_AUTOSIZE);
    setMouseCallback("CAM Capture", myMouseCallback);

    char c;
    while (1)
    {
        capture >> frame;

```

```

        //обработка кадра
        frame = treatment(frame);

        imshow("CAM Capture", frame);
        c = waitKey(30);
        if (c == 27) break;
    }
    destroyAllWindows();
}

```

Пояснения к коду

`Mat treatment(const Mat& original)` тут происходит обработка изображения для того, чтобы выделить участки изображения, соответствующие выбранному цвету.

`getStructuringElement(int shape, Size ksize, Point anchor)` Функция конструирует и возвращает элемент структурирования, который может быть далее передан для операций размывтия (`erode`) и расширения (`dilate`).

Первый аргумент – форма элемента (эллипс, крест, прямоугольник). Второй аргумент - размер структурирующего элемента. Третий аргумент – якорная позиция внутри элемента. Значение (1,1) располагает якорь в центре матрицы 3*3.

`void dilate(InputArray src, OutputArray dst, InputArray kernel, Point anchor)` функция расширяет исходное изображение с использованием указанного элемента структурирования, который определяет форму окрестности пикселя. `kernel` ядро, которое создается в функции `getStructuringElement`. `anchor` – якорь у которого значение по умолчанию `Point(-1, -1)` значит, что привязка находится в центре элемента.

`findContours(InputOutputArray image, OutputArrayOfArrays contours, int mode, int method)` функция извлекает контуры из двоичного изображения. Первый параметр – изображение, в котором ищем контуры. Второй параметр – найденные контуры. Они представляют из себя вектор точек. Третий параметр – режим поиска контуров. `RETR_EXTERNAL` извлекает только крайние внешние контуры. Четвертый параметр – метод приближения контуров (аппроксимации). `CHAIN_APPROX_SIMPLE` метод сжимает горизонтальные, вертикальные и диагональные сегменты и оставляет только их конечные точки.

`approxPolyDP(InputArray curve, OutputArray approxCurve, double epsilon, bool closed)` метод аппроксимирует кривую или многоугольник с другой кривой / многоугольником с меньшим количеством вершин, так что расстояние между ними меньше или равно заданной точности. Используется алгоритм [Дугласа-Пьюкера](#). Первый аргумент – входной вектор двумерных точек. Второй аргумент – результат приближения (тип должен соответствовать входным данным). Третий аргумент – параметр, определяющий точность аппроксимации. Четвертый параметр определяет замкнутость кривой. Если `true`, то крайние точки кривой соединены.

`Rect boundingRect(InputArray points)` вычисляет минимальный ограничивающий прямоугольник для входного набора точек. Он используется для последующей отрисовки в функции `rectangle`. `tl()` и `br()` возвращают верхний левый угол и нижний правый угол соответственно.

Практическое задание

1. Выведите в отдельные окна все этапы операций над изображением в функции `treatment`.

Для того чтобы работать с одним выбранным кадром, предварительно можно сделать конструкцию для остановки потокового вывода, как в уроке 17. С помощью конструкции:

```
else if (c == 13){}
```

То есть после нажатия `Enter`, мы будем заходить в этот участок кода. В нем можно сделать вызов функции `treatment`, потом вывод результата в окно «СМ Capture». После, сделать ожидания нажатия клавиши, чтобы можно было изучить полученные результаты (`waitKey(0);`). И сделать закрытие окон, которые мы создадим в функции `treatment`. Функция закрытия отдельного окна выглядит следующим образом:

```
destroyWindow("nameWindow");
```

Первый вывод окна в `treatment` после функции `inRange` — это будет исходное одноканальное изображение. Второй вывод — после функции размытия `dilate`. Третий вывод в конце функции `treatment`. В нём нужно вывести одноканальное изображение, на котором нарисованы прямоугольники (зеленого цвета). Для этого нужно из одноканального изображения сделать 3-х канальное с помощью данной конструкции:

```
Mat channels[3] = { bin, bin, bin };
```

Затем нужно объединить каналы в одно изображение с помощью функции `void merge(const Mat* mv, size_t count, OutputArray dst)`. Первый аргумент — входной массив (в нашем случае `channels`). Второй аргумент — количество входных матриц. Третий аргумент — изображение которое получаем на выходе (`Mat`). Затем в цикл отрисовки прямоугольников добавляем еще одну отрисовку на нашем 3-х канальном изображении, полученным из 3-х одноканальных.

УРОК 21. БИБЛИОТЕКА ARUCO. СОЗДАНИЕ МАРКЕРА.

Для работы с библиотекой ARUCO нам потребуются специальные маркеры. Данная программа создает маркер, сохраняет его в файле и выводит на экран.

Код программы

```
#include <cv.h>
#include <highgui.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/aruco.hpp"
#include "opencv2/aruco/dictionary.hpp"

using namespace std ;
using namespace cv;

int main( int argc, char** argv )
{
    Mat markerImage;
    aruco::Dictionary dictionary =
aruco::getPredefinedDictionary(aruco::DICT_6X6_250);

    // создаем словарь
    dictionary.drawMarker(21, 200, markerImage, 1); // рисуем маркер
    imwrite("Image.png", markerImage); // сохраняем маркер в файл
    namedWindow("CAM Capture", WINDOW_AUTOSIZE ); // создаем окно
    imshow("CAM Capture", markerImage); // показываем окно

    char c;
    while (1)
    {
        c = waitKey(30);
        if ( c == 27 ) break;
    }
    destroyAllWindows();
}
```

Пояснения к коду

`Ptr<Dictionary>` `cv::aruco::getPredefinedDictionary`
(`PREDEFINED_DICTIONARY_NAME` name) возвращает один из предопределенных словарей, определенных в `PREDEFINED_DICTIONARY_NAME`. `DICT_6X6_250` — словарь 6*6 бит, состоящий из 250 слов.

`void drawMarker(int id, int sidePixels, OutputArray img, int borderBits = 1)`. Первый параметр `int id` - номер маркера в словаре. Второй параметр `int sidePixels` - задает размер в пикселах. Третий параметр `OutputArray _img` — маркер, представленный в `Mat`. Четвертый параметр `int borderBits` - задает ширину границы маркера.

Практическое задание

1. Создайте еще пару маркеров с другим id и сравните их.

УРОК 22. БИБЛИОТЕКА ARUCO. ОБНАРУЖЕНИЕ МАРКЕРА

Программа находит на фотографии все маркеры, соответствующие словарю «DICT_6X6_250».

Код программы

```
#include <cv.h>
#include <highgui.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/aruco.hpp"
#include "opencv2/aruco/dictionary.hpp"

using namespace std;
using namespace cv;

int main( int argc, char** argv )
{
    Mat inputImage = imread("Image.png", IMREAD_COLOR); // считываем изображение

    vector< int > markerIds; // хранение ID маркеров
    vector< vector<Point2f> > markerCorners; // углы маркеров
    Ptr<aruco::Dictionary> dictionary =
aruco::getPredefinedDictionary(aruco::DICT_6X6_250);
    aruco::detectMarkers(inputImage, dictionary, markerCorners, markerIds);
    aruco::drawDetectedMarkers(inputImage, markerCorners, markerIds);

    namedWindow("CAM Capture", WINDOW_AUTOSIZE);
    imshow("CAM Capture", inputImage);
    char c;
    while (1)
    {
        c = waitKey(30);
        if (c == 27) break;
    }
    destroyAllWindows();
}
```

Пояснения к коду

`detectMarkers(InputArray image, const Ptr<Dictionary>& dictionary, OutputArrayOfArrays corners, OutputArray ids)` функция ищет маркеры на изображении. Первый параметр – изображение, второй – словарь, третий – вектор углов маркера, четвертый – вектор ID маркеров.

`drawDetectedMarkers(InputOutputArray image, InputArrayOfArrays corners, InputArray ids, Scalar borderColor = Scalar(0, 255, 0))` осуществляет отрисовку обнаруженных маркеров. Первый параметр – изображение, второй – вектор углов маркера, третий – вектор ID маркеров, четвертый – цвет границ маркера (по умолчанию зелёный).

УРОК 23. БИБЛИОТЕКА ARUCO. КАЛИБРОВКА КАМЕРЫ.

Для определения положения маркера в пространстве требуется калибровка камеры. Калибровка выполняется с помощью шахматной доски.

Для того чтобы сделать один снимок доски нужно нажать пробел, после этого будет происходить поиск доски. Если доска будет найдена, то изображение выведется в отдельном окне. После калибровки значения запишутся в файл «calibrateCamera.yml». Количество внутренних углов равно количеству ячеек по горизонтали (вертикали) минус 1.

В конце программа покажет поток видео до калибровки и после неё.

Код программы

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <iostream>

#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/flann/flann.hpp"
#include "opencv2/video.hpp"
#include "opencv2/videoio.hpp"
#include "opencv2/ml.hpp"
#include "opencv2/objdetect.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/aruco.hpp"

using namespace std;
using namespace cv;

int main(int argc, char** argv)
{
    int numBoards = 8; //количество калибровок (снимков доски)
    int numCornersHor = 7; //количество внутренних углов по горизонтали
    int numCornersVer = 7; //количество внутренних углов по вертикали

    int numSquares = numCornersHor * numCornersVer; //количество ячеек
    Size board_sz = Size(numCornersHor, numCornersVer); //представление доски в
виде матрицы
    VideoCapture capture = VideoCapture(0);
    if (!capture.isOpened()) return -1;
    vector<vector<Point3f>> object_points;
    vector<vector<Point2f>> image_points;

    vector<Point2f> corners;
    int successes = 0;

    Mat image;
    Mat gray_image;

    capture >> image;
    vector<Point3f> obj;
    for (int j = 0; j < numSquares; j++)
        obj.push_back(Point3f(j / numCornersHor, j % numCornersHor, 0.0f));
```

```

while (successes < numBoards)
{
    capture >> image;

    cvtColor(image, gray_image, CV_BGR2GRAY);

    imshow("win1", gray_image);

    int key = waitKey(30);
    if (key == 27)
        return 0;

    if (key == ' ')
    {
        bool found = findChessboardCorners(image, board_sz, corners,
CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS);
        if (found)
        {
            cornerSubPix(gray_image, corners, Size(11, 11), Size(-1, -1),
TermCriteria(CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 30, 0.1));
            drawChessboardCorners(gray_image, board_sz, corners, found);
            imshow("win2", gray_image);
            image_points.push_back(corners);
            object_points.push_back(obj);

            printf("Frame %d found!\n", successes);

            successes++;
            if (successes >= numBoards)
                break;
        }
    }
}

Mat cameraMatrix = Mat(3, 3, CV_32FC1);
Mat distCoeffs;
vector<Mat> rvecs;
vector<Mat> tvecs;

cameraMatrix.ptr<float>(0)[0] = 1;
cameraMatrix.ptr<float>(1)[1] = 1;

calibrateCamera(object_points, image_points, image.size(), cameraMatrix,
distCoeffs, rvecs, tvecs);

FileStorage fs("calibrateCamera.yml", FileStorage::WRITE);
fs << "cameraMatrix" << cameraMatrix << "distCoeffs" << distCoeffs;
fs.release();

Mat imageUndistorted;
while (1)
{
    capture >> image;
    undistort(image, imageUndistorted, cameraMatrix, distCoeffs);

    imshow("win1", image);
    imshow("win2", imageUndistorted);
    int key = waitKey(30);

    if (key == 27)
        break;
}

```

```
capture.release();  
return 0;  
}
```

УРОКИ 24. БИБЛИОТЕКА ARUCO. ОПРЕДЕЛЕНИЕ ПОЗИЦИИ МАРКЕРА

Программа определяет положение маркеров относительно камеры.

Код программы

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <iostream>

#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/flann/flann.hpp"
#include "opencv2/video.hpp"
#include "opencv2/videoio.hpp"
#include "opencv2/ml.hpp"
#include "opencv2/aruco.hpp"

using namespace std;
using namespace cv;

bool find_ids(std::vector<int> ids, int number);

int main(int argc, char** argv)
{
    VideoCapture capture = VideoCapture(0);
    if(!capture.isOpened()) return -1;
    Mat image, cameraMatrix, distCoeffs;

    FileStorage fs2("calibrateCamera.yml", FileStorage::READ); //загружаем
калибровку камеры
    fs2["cameraMatrix"] >> cameraMatrix;
    fs2["distCoeffs"] >> distCoeffs;
    fs2.release();

    Ptr<aruco::Dictionary> dictionary =
aruco::getPredefinedDictionary(aruco::DICT_6X6_250);
    while (1)
    {
        capture >> image;

        std::vector<int> ids;
        std::vector<std::vector<cv::Point2f> > corners;
        aruco::detectMarkers(image, dictionary, corners, ids);

        if (ids.size() > 0) //количество маркеров > 0
        {
            aruco::drawDetectedMarkers(image, corners, ids);
            vector< Vec3d > rrvecs, ttvecs;
            aruco::estimatePoseSingleMarkers(corners, 0.05, cameraMatrix, distCoeffs,
rrvecs, ttvecs); //оценка положения маркеров нужна для отрисовке осей

            for (int i = 0; i < ids.size(); i++) {
```



```

        aruco::drawAxis(image, cameraMatrix, distCoeffs, rrvects[i], ttvecs[i],
0.1); // отрисовка осей маркеров (последний параметр – длина в метрах)
    }
    find_ids(ids, 23);
}
imshow("", image);
int key = waitKey(30);
if (key == 27)
    break;
}
capture.release();
destroyAllWindows();
return 0;
}

bool find_ids(std::vector<int> ids, int number)
{
    if (std::find(ids.begin(), ids.end(), number) != ids.end())
    {
        printf("Found %d - id\n", number);
        return true;
    }
    return false;
}

```

Пояснения к коду

void estimatePoseSingleMarkers(InputArrayOfArrays corners, float markerLength, InputArray cameraMatrix, InputArray distCoeffs, OutputArray rrvects, OutputArray ttvecs) corners – массив углов маркера, markerLength – длина стороны маркера в метрах, cameraMatrix – матрица камеры, distCoeffs – вектор коэффициентов искажения, rrvects – массив выходных векторов вращения, ttvecs – массив выходных векторов трансляции.

Функция bool find_ids(std::vector<int> ids, int number) выводит строку с найденным id и возвращает true. Если маркера с заданным id не найдено – false. Первый параметр – вектор идентификаторов, второй – id который нужно найти.