Оглавление

[Введение 5](#_Toc495565399)

[1 Обзор предметной области 7](#_Toc495565400)

[1.1 Алгоритмы распознавания, основанные на дескрипторах ключевых точек 7](#_Toc495565401)

[1.2 Детектор Харриса 10](#_Toc495565402)

[1.3 Детектор Ши-Томаси 11](#_Toc495565403)

[1.4 Детектор SIFT 12](#_Toc495565404)

[1.5 Детектор SURF 15](#_Toc495565405)

[2 Разработка системы 17](#_Toc495565406)

[2.1 Подготовка рабочей среды 17](#_Toc495565407)

[2.1.1 Установка операционной системы 17](#_Toc495565408)

[2.1.2 Настройка удаленного доступа 20](#_Toc495565409)

[2.1.3 Установка OpenCV 23](#_Toc495565410)

[2.2 Описание разработанной программы 26](#_Toc495565411)

[2.2.1 Выбор языка программирования 26](#_Toc495565412)

[2.2.2 Библиотека OpenCV 26](#_Toc495565413)

[2.2.3 Информационная модель программы 28](#_Toc495565414)

[3 Выводы 30](#_Toc495565415)

[Заключение 31](#_Toc495565416)

[Список литературы 32](#_Toc495565417)

[Приложение А 34](#_Toc495565418)

# Введение

Зрение является самым мощным из наших органов чувств. Оно снабжает нас поразительно большим объемом информации о тот, что нас окружает, и дает возможность свободно взаимодействовать с внешним миром, причем все это без непосредственного физического контакта. Благодаря ему мы узнаем расположение объектов, идентифицируем их, соотносим их друг с другом. Неудивительно, что, как только цифровые вычислительные машины оказались достаточно доступными, начали предприниматься попытки наделит их способностью «видеть» [1].

Обработка изображений является одним из важнейших направлений применения современной вычислительной техники. Одной из распространенных задач обработки изображений являются фильтрация и восстановление изображений, сегментация, а также средства сжатия информации. На место классических задач по распознаванию фигур заданной формы на изображении приходят новые задачи распознавания линий и углов на изображении, распознавания краев изображений. Сегодня, с каждым днем появляется всё больше данных, которые уже некогда нельзя будет обработать человеческим трудом, поэтому темы обработки данных, распознавание образов и машинное обучение являются актуальными темами.

По мнению авторов [2], не существует общепринятой точки зрения, где заканчивается обработка изображений и начинаются другие смежные области, такие как анализ изображений и машинное зрение. Естественным этапом перехода от обработки изображений к их анализу выступает автоматическое или интерактивное распознавание отдельных областей или объектов на предварительно улучшенном изображении.

Целью данной работы является разработка системы анализа и классификации объектов на изображении при помощи алгоритмов, основанных на нахождении особых точек, уникальных характеристик объектов.

В ходе выполнения работы необходимо выполнить ряд задач:

* изучение методов нахождения ключевых точек;
* анализ алгоритмов, основанных на ключевых точках;
* построение информационной модели;
* реализация программы.

Объектом исследования является система анализа, обработки и нахождения ключевых точек на изображении.

Предметом исследования является система классификация образов на исходном изображении.

# 1 Обзор предметной области

## 1.1 Алгоритмы распознавания, основанные на дескрипторах ключевых точек

Распознавание образов в полном объеме машинным способом – это задача, которая до сих пор не решена, однако в рамках существенных ограничений, есть методы, позволяющие приблизится к ее решению. Проецируя задачу на человека, можно представить игру-головоломку «Пазл», в которой из мелких деталей нужно собрать цельную картину. Основной алгоритм заключается в том, чтобы найти конкретные шаблоны или уникальные особенности, которые легко отыскать и просто сравнить с эталонным изображением, и далее по собранным частям составить итоговую картину. Мы ищем уникальные части объекта на пазлах, и до завершения работы уже способны составить цельное представление о конечном результате. Что бы понять, как мы легко находим особые детали, достаточно взглянуть на следующее изображение (рисунок 1):



Рисунок 1 – Характеристические признаки

В верхней части изображения даны шесть небольших клочков, задача –найти точное местоположение каждой части в исходном изображении.

A и B – плоские поверхности, распределенные во многих местах. Трудно найти точное местоположение каждого из этих обрывков.

C и D – задача становится намного проще. Части являются краями здания и можно найти приблизительное местоположение, но точное местоположение узнать все еще сложно. Таким образом, край – намного лучший характерный признак по сравнению с равномерной областью, но все равно недостаточно хорош.

И наконец, E и F – некоторые углы здания. Их можно легко узнать, т.к. в углах, как бы не перемещать и прикладывать к изображению этот кусочек, он все равно будет выглядеть иначе. Поэтому углы можно считать хорошим характеристическим признаком.

Для выделения из изображения некоторой информации необходимо привязаться к локальным особенностям изображения. На изображении возможно выделить особые (характеристические) точки. Характеристическая точка (точка интереса) – точка изображения, обладающая высокой локальной информативностью. В качестве численной меры информативности предлагаются различные формальные критерии, называемые операторами интереса. Оператор интереса должен обеспечивать достаточно точное позиционирование точки в плоскости снимка. Необходимо также, чтобы положение точки интереса обладало достаточной устойчивостью к фотометрическим и геометрическим искажениям изображения, включающим неравномерные изменения яркости, сдвиг, поворот, изменение масштаба, ракурсные искажения. Процесс определения особых точек достигается путем использования детектора и дескриптора.

Детектор – метод извлечения особых точек из изображения.

Дескриптор – идентификатор особой точки, выделяющий ее из остального множества особых точек. Дескрипторы должны обеспечивать инвариантность нахождения соответствия между особыми точками относительно аффинных преобразований изображений [3].

В 1992 Haralick и Shapir [4] выделили следующие требования к особым точкам в виде следующих свойств:

* Отличимость (distinctness) – особая точка должна явно выделяться на фоне и быть отличимой (уникальной) в своей окрестности.
* Инвариантность (invariance) – определение особой точки должно быть независимо к аффинным преобразованиям.
* Стабильность (stability) – определение особой точки должно быть устойчиво к шумам и ошибкам.
* Уникальность (uniqueness) – кроме локальной отличимости, особая точка должна обладать глобальной уникальностью для улучшения различимости повторяющихся паттернов.
* Интерпретируемость (interpretability) – особые точки должны определяться так, чтобы их можно было использовать для анализа соответствий и выявления интерпретируемой информации из изображения.

Подходы к определению особых точек могут быть разделены на 3 категории [4]:

1. Основанные на интенсивности изображения.
2. Использующие контуры изображения.
3. На основе использования модели с интенсивность в качестве параметров, которые подстраиваются к изображениям-шаблонам.

На практике для широкого применения наиболее распространены методы, основанные на интенсивности изображения.

# 1.2 Детектор Харриса

Можно заметить, что углы являются областями с большим изменением интенсивности во всех направлениях. Одна из первых попыток найти эти углы была сделана Крисом Харрисом и Майком Стивенсом в их статье «A Combined corner and edge detector» [5] в 1988 году. В основе лежит нахождение разности интенсивности для смещения по всем направлениям. Это выражается следующим образом:

*–* весовая функция (обычно используется функция Гаусса или бинарное окно).

Для максимизации функции для определения угла:

где:

*–* производные своих направлений.

Затем идет основная часть. Создается оценка, в которой определяется, может ли окно содержать угол или нет.

где:

– эмпирическая константа

­– собственные вектора.

Таким образом, если *R* мала, т.е. когда малы, то область однородная. Если *R* < 0, что происходит при или наоборот, то область является краем. Если *R* велика, что происходит, когда велики и , то область является углом (рисунок 2).

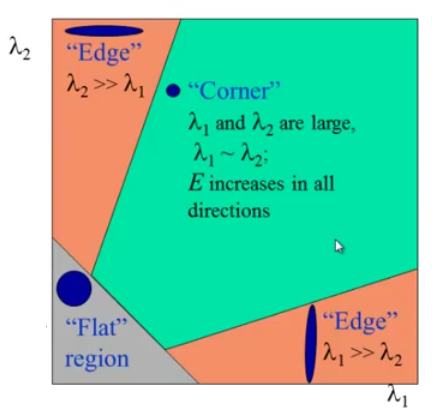


Рисунок 2 – Значения собственных векторов

# 1.3 Детектор Ши-Томаси

Детектор Ши-Томаси, на ряду с детектором Харриса, отличается тем, то мера отклика значения напрямую вычисляется следующей формулой:

В своей работе [6] авторы делают предположение, что поиск углов становится более стабильным.

# 1.4 Детектор SIFT

SIFT (Scale-Invariant Feature Transform) – алгоритм масштаба-независимого преобразования ключевых точке. Предыдущие алгоритмы инвариантны к любым вращениям изображения, т.к. те же углы так же остаются на повернутой картинке, однако при ее масштабировании, угол в маленьком окне может превратится в край (рисунок 3).

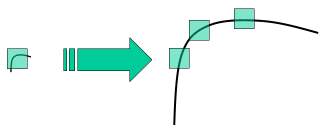


Рисунок 3 – Наложение фрейма на отмасштабированное изображение

В 2004 году, Давидом Ловем из университета British Columbia, в своей статье «Отличительное представление характеристик из масштаба-инвариантных ключевых точек» [7], был представлен новый алгоритм SIFT (Инвариантное масштабирование признаков). Алгоритм SIFT состоит из четырех основных шагов.

1. Обнаружение точек при масштабировании

Из вышеприведенного изображения очевидно, что мы не можем использовать одно и то же окно для обнаружения ключевых точек с разным масштабом. Все в порядке с маленькими углами, но для обнаружения больших углов нужны окна больших размеров. Для этого используется построение пирамиды гауссианов и разностей гауссианов:

где:

– значение гауссиана в точке с координатами (x, y)

– радиус размытия

– гауссово ядро

– значение исходного изображения

– операция свертки.

Difference of Gaussian (DoG):

(7)

Инвариантность достигается благодаря нахождению характеристических точек для первоначального изображения, взятого в разных масштабах. Строится пирамида гауссианов, разбивая все пространство на октавы (рисунок 4):

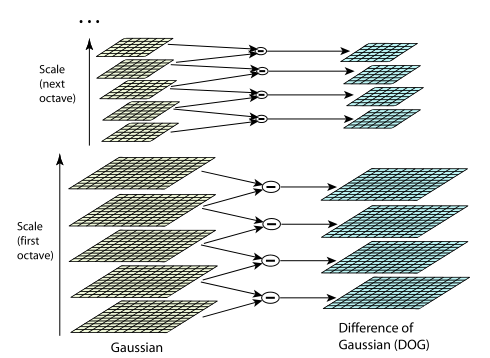


Рисунок 4 – Пирамиды преобразования гауссиана

Как только найдены разности гауссианов, ищутся локальные экстремумы в каждом из DoG изображений. Каждая точка, сравнивается с восемью соседними точками и девятью, расположенными выше и ниже уровнем. Наибольшая или наименьшая по значению точка принимается за экстремум (рисунок 5).

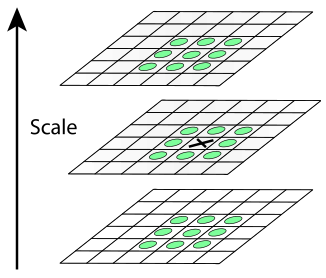


Рисунок 5 – DoG октавы

2. Уточнение ключевых точек

После обнаружения местоположений потенциальных ключевых точек их нужно уточнить, используя ряд Тейлора второго порядка. Если интенсивность на текущем экстремуме меньше порогового значения, то идет сдвижение к соседней точки. Если расчет оказался неудачным и произошел выход за пределы октавы, то точка отвергается. Также идет проверка, лежит ли точка на границе какого-то объекта или плохо освещена. Таким образом устраняются любые низко контрастные точки, оставляя только характеристические точки.

3. Выяснение ориентации ключевой точки

Далее необходимо определить ориентацию каждой ключевой точки для достижения инвариантности к вращению изображения. Берутся соседние с особой точи и вычисляется значение градиента (8) и его направление (9).

4. Формирование дескрипторов

Создается вектор направленных градиентов в окрестности ключевой точки. Берется матрица размером 16х16, окружающая ключевую точку. Далее блок разделяется на шестнадцать подблоков, каждый из которых состоит из восьми компонент гистограммы, покрывающей участок в 360 градусов. Таким образом образуется дескриптор размерности 128 компонент (рисунок 6).

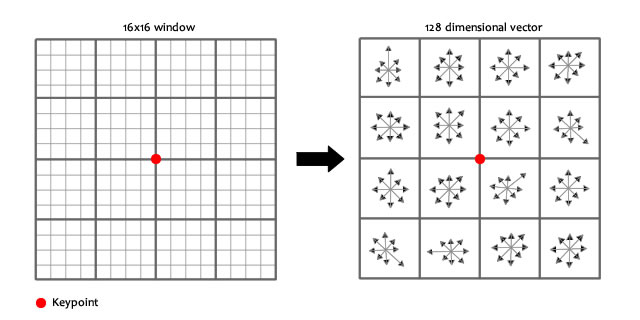


Рисунок 6 – SIFT Дескриптор

# 1.5 Детектор SURF

SURF (Speeded up Robust Features) – алгоритм, разработанный с целью ускорить поиск ключевых точек [9]. Обнаружение характеристических точек основано на вычислении детерминанта матрицы Гессе (гессиана):

где:

– значение текущего пикселя

После чего вычисляется градиент для каждой особой точки применяя фильтры Хаара (рисунок 7) [10].

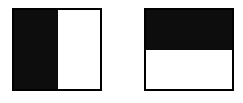


Рисунок 7 – Фильтры Хаара

# 2 Разработка системы

## 2.1 Подготовка рабочей среды

## 2.1.1 Установка операционной системы

Raspberry Pi 3 – одноплатный компьютер со следующими характеристикам (рисунок 8):

* 4х ядерный ARM процессор с частотой 1200 МГц,
* 1 Гб оперативной памяти,
* порты USB, HDMI, LAN, Jack 3.5 mm,
* поддерживается Bluetooth, WiFi;



Рисунок 8 –­ Raspberry Pi 3

Для установки операционной системы скачал образ операционной системы с официального сайта Raspberry Pi. Далее для прошивки операционной системы на SD – карту использовал программу SDFormatter V4.0 со следующими настройками (рисунок 9).

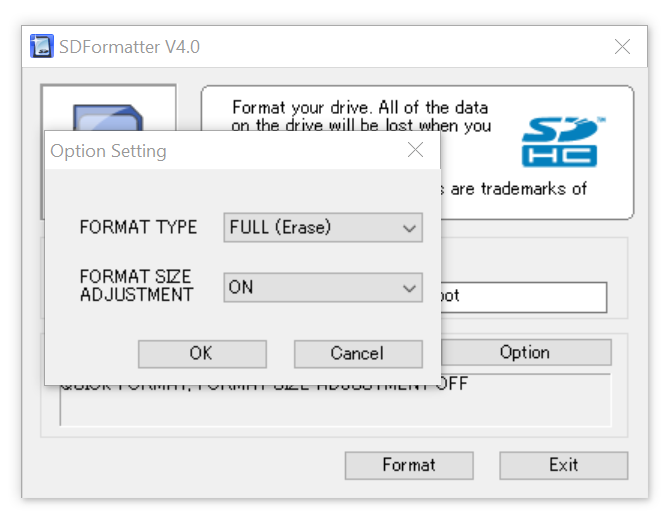


Рисунок 9 – SDFormater

После завершения передачи операционной системы на SD карту, ее следует вставить в задний порт Raspberry Pi, далее запустить, подсоединив устройство питания. При запуске, установил предложенную Raspbian OS (рисуноки 10 – 12).

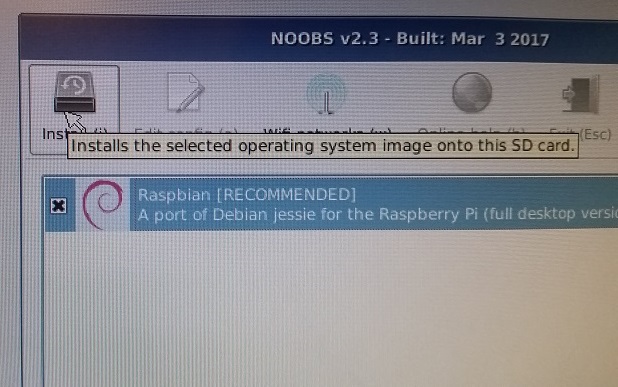


Рисунок 10 – Raspbian OS

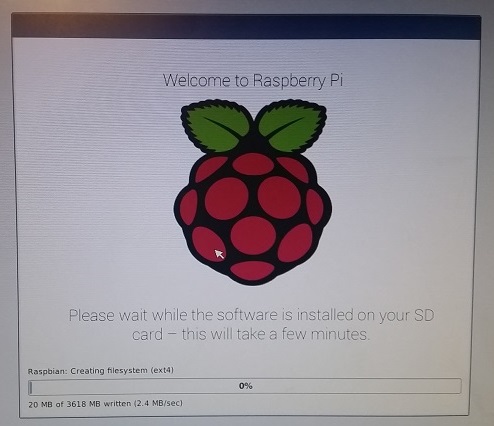


Рисунок 11 – Установка Raspbian OS

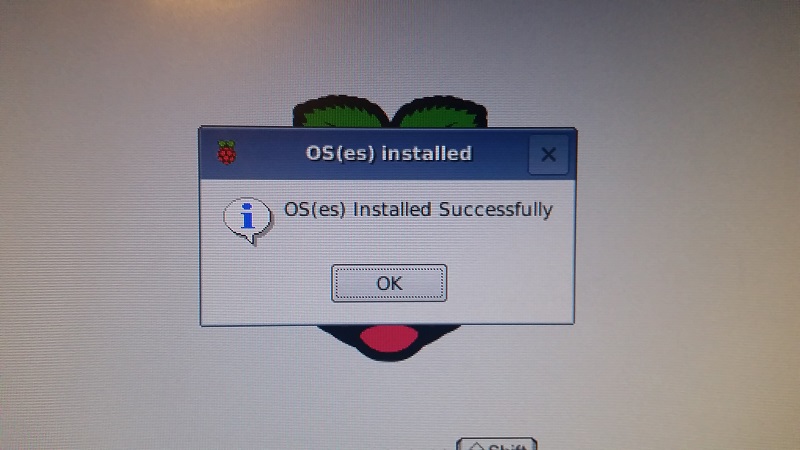


Рисунок 12 – Завершение установки

После перезагрузки компьютера, сперва подключился к домашней WiFi сети, далее сочетанием клавиш ctrl+alt+t запустил командную строку и ввел команды для обновления системы:

* sudo apt-get update
* sudo apt-get upgrade

## 2.1.2 Настройка удаленного доступа

Сперва запустил SSH сервера на Raspberry Pi. Для этого в командной строке ввел sudo raspi-config, после чего открылось меню настоек (рисунок 13).

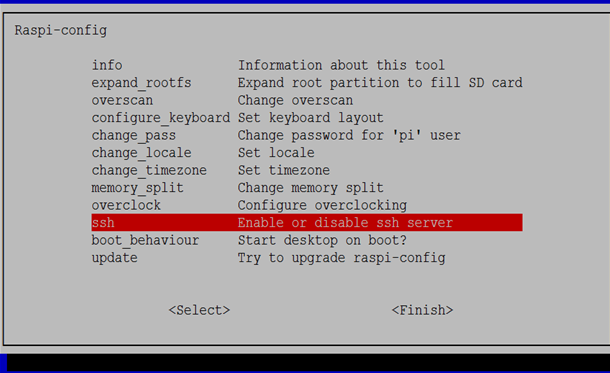


Рисунок 13 –Raspberry Pi

После чего перевел состояние ssh в enable (рисунок 14).



Рисунок 14 – SHH Ebable

Далее настроил программу PuTTY для SSH доступа к консоли Raspberry Pi из Windows. Сперва скачал дистрибутив программы с официального сайта. После запуска программы перешел в раздел Translation и выбрал кодировку UTF–8 (рисунок 15).

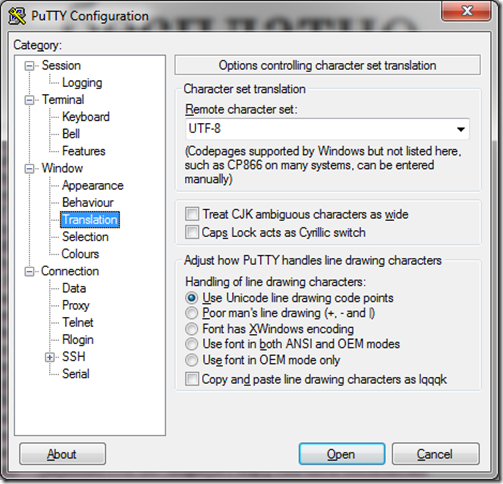


Рисунок 15 – PuTTY настройка кодировки

Используя программу Free IP Scanner, посмотрел список доступных устройств и их IP (рисунок 16).

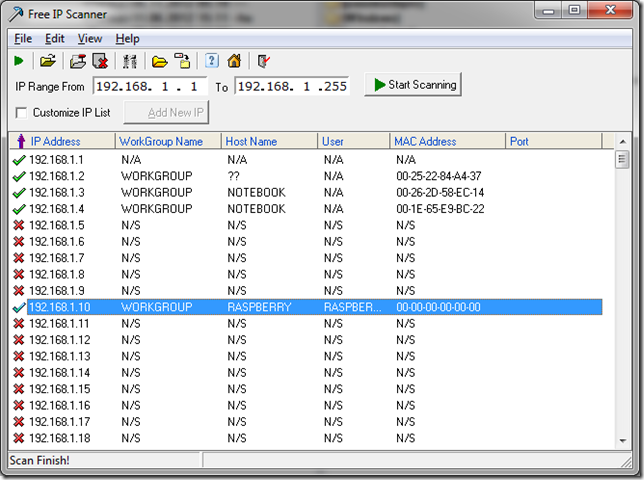


Рисунок 16 – Free IP Scanner

После чего в разделе Session ввел IP, Port сохранил настройки (рисунок 17).

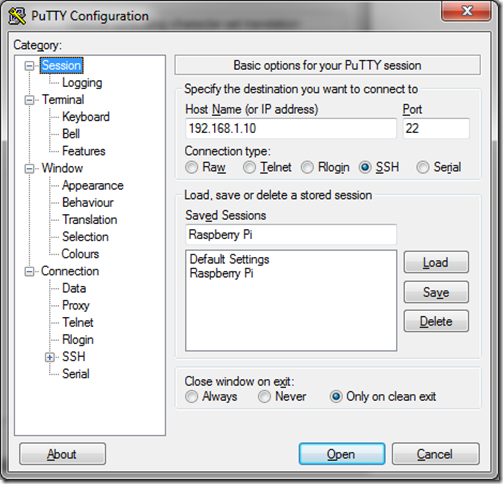


Рисунок 17 – PuTTY конфигурация сессии

Далее нажал кнопку Open и получил удаленный доступ к устройству (рисунок 18).

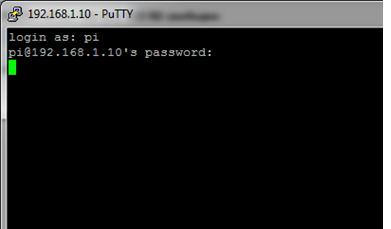


Рисунок 18 – Удаленный доступ

## 2.1.3 Установка OpenCV

OpenCV – это библиотека функций машинного зрения, позволяющая распознавать объекты окружающего мира.

Для полноценной работы с OpenCV обновил и установил новые пакеты. Сперва обновил систему:

* sudo apt-get update
* sudo apt-get upgrade

Далее установил в систему cmake, несколько пакетов для работы с изображениями и видео:

* sudo apt-get install build-essential cmake pkg-config
* sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
* sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv41-dev
* sudo apt-get install libxvidcore-dev libx264-dev

Установил пакеты для создания экранных форм, а также ускоренные операции над матрицами и заголовочные файлы языка python 2.7 и python 3:

* sudo apt-get install libgtk2.0-dev
* sudo apt-get install libatlas-base-dev gfortran
* sudo apt-get install python2.7-ev python3-dev
* После чего скачал архив с opencv в папку Download и распаковал его:
* cd ~/Downloads
* wget -O opencv.zip https://github.com/opencv/opencv/archive/master.zip
* unzip onpecv.zip

Далее скачал пакет с дополнительными функциями, не вошедшими в основной релиз:

* wget -O opencv\_contrib.zip https://github.com/opencv/opencv\_contrib/archive/master.zip
* unzip opencv\_contrib.zip

Перед установкой OpenCV, создал виртуальное окружение для изоляции проекта от системной версии и установленных пакетов python. Сперва установил менеджер пакетов pip:

* wget https://bootstrap.pypa.io/get-pip.py
* sudo python get-pip.py

Затем установил менеджер виртуального окружения:

* sudo pip install virtualenv virtualenvwrapper
* sudo rm -rf ~/.cache/pip

Далее добавил несколько строк в профиль пользователя:

* echo -e “\n# virtualenv and virtualenvwrapper” >> ~/.profile
* echo “export WORKON\_HOME=$HOME/.virtualenvs” >> ~/.profile
* echo “source /usr/local/bin/virtualenvwrapper.sh” >> ~/.profile

После чего перезагрузил Raspberry Pi, в командной строке ввел команду:

* source ~/.profile

Далее создал виртуальное окружение с именем «cv», запустил его и тут же установил в него математический пакет NumPy:

* mkvirtualenv cv -p python3
* workon cv
* pip install numpy

Находясь в виртуальном окружении, скомпилировал и установил OpenCV. Для этого зашёл в папку с исходным кодом библиотеки и выполнил команды:

* cd ~/Downloads/opencv-master
* mkdir build
* cd build
* cmake -D CMAKE\_BUILD\_TYPE=RELEASE \

-D CMAKE\_INSTALL\_PREFIX=/usr/local \

-D INSTALL\_PYTHON\_EXAMPLES=ON \

-D OPENCV\_EXTRA\_MODULES\_PATH=~ \

/Downloads/opencv\_contrib-master/modules \

-D BUILD\_EXAMPLES=ON ..

После подготовки компонентов собрал бинарные файлы командой:

* make -j2, где ключ -j2 обозначает количество ядер использующихся для сборки библиотеки

После нескольких часов сборки установил бинарный файл командой:

* sudo make install
* sudo ldconfig

Далее переименовал появившейся в папке python файл cv2.cpython-34m.so:

* cd /usr/local/lib/python3.4/site-packages/
* sudo mv cv2.cpython-34m.so cv2.so

Затем, чтобы пользоваться OpenCV, находясь в виртуальном окружении, сделал символьную ссылку на получившийся cv2.so файл:

* cd ~/.virtaulev/cv/lib/python3.4/site-packages/
* ln -s /usr/local/lib/python3.4/site-packages/cv2.so cv2.so

Для проверки, запустил виртуальное окружение, интерпретатор, импортировал библиотеку и проверил текущую версию (рисунок 19):

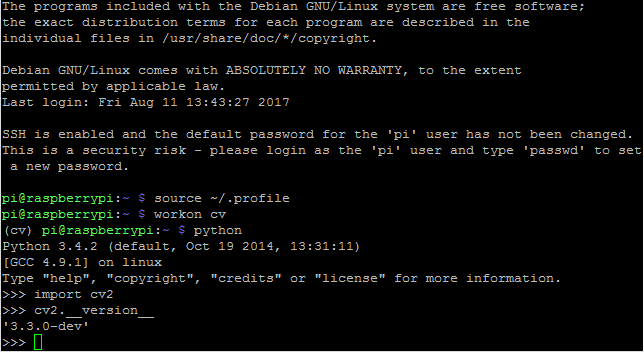


Рисунок 19 – PuTTY connection

## 2.2 Описание разработанной программы

# 2.2.1 Выбор языка программирования

Работа с изображениями, а в частности их обработка и анализ требует постоянных экспериментов. Поиск подходящего решения вовсе не линеен, а напротив, приходится многократно возвращаться назад, пробуя другие сочетания исходных данных и алгоритмов. Именно изыскательская природа этого процесса делает применение Python чрезвычайно уместным. Ведь Python, будучи интерпретируемым языком программирования, как будто специально придуман для опробования разных вариантов. К тому же, он работает быстро. Конечно он медленнее компилируемых языков, но при наличии огромного числа простых в использовании библиотек, зачастую написанных на С, не приходится жертвовать скорость [11].

Среди интерпретируемых языков, Python выделяется тем, что имеет богатый набор математических и аналитических библиотек, а также имеет обертки на различные библиотеки, например, такие как OpenCV, TensorFlow и т.д.

# 2.2.2 Библиотека OpenCV

Одной из целей OpenCV является предоставление простой в использовании библиотеки машинного зрения, которая помогает создавать достаточно сложные приложения в короткие сроки. Поскольку компьютерное зрение и обучение машин идут сильно связаны, то OpenCV включает в себя библиотеку общего назначения MLL (Machine Learning Library — библиотека обучения машин). Она позволяет решать задачи статистического распознавания образов и кластеризации. MLL очень полезна для решения сложных задач, связанных с компьютерным зрением, но является достаточно общей, для того чтобы решать большинство задач обучения машин [12].

Многомерная архитектура проекта OpenCV представлена на (рисунок 20). Библиотека состоит из 16 модулей. Реализованы около 1000 алгоритмов. Функ- 48 циональность доступна на разных языках: C, C++, Python, CUDA, Java. Поддерживаются основные операционные системы: MS Windows, Linux, Mac, Android, iOS и другие.

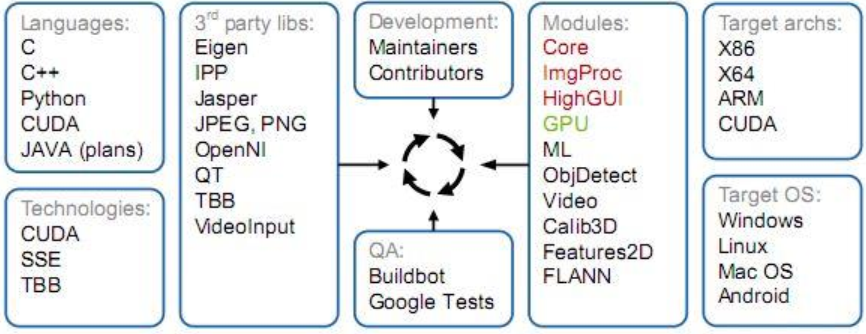


Рисунок 20 – Состав библиотеки OpenCV

Основные модули библиотеки можно отнести к 4 группам (разделам):

* Модули core, highgui, реализующие базовую функциональность (базовые структуры, математические функции, генераторы случайных чисел, линейная алгебра, быстрое преобразование Фурье, ввод/вывод изображений и видео, ввод/вывод в форматах XML, YAML и др.).
* Модули imgproc, features2d для обработки изображений (фильтрация, геометрические преобразования, преобразование цветовых пространств, сегментация, обнаружение особых точек и ребер, контурный анализ и др).
* Модули video, objdetect, calib3d (калибровка камеры, анализ движения и отслеживание объектов, вычисление положения в пространстве, построение карты глубины, детектирование объектов, оптический поток).
* Модуль ml, реализующий алгоритмы машинного обучения. Результаты промежуточных вычислений можно сохранять в xml-файлах, а затем их прочитать, например, в другой программе, что облегчает разработку алгоритма по частям, работу над алгоритмом целой командой.

OpenCV может быть использована в приложениях реального времени, может использовать возможности многоядерных процессоров. Если необходимо получить от библиотеки большую производительность, то необходимо приобрести IPP (Integrated Performance Primitives), представляющий из себя набор тщательно оптимизированных функций для работы с мультимедиа, криптографией, распознанием речи, сжатием данных. OpenCV автоматически использует IPP, 51 если она установлена. При этом можно использовать и другие средства распараллеливания (CUDA, OpenMP).

В ходе работы выяснил достоинства и недостатки OpenCV. К первым можно отнести большое количество алгоритмов и удобство настройки параметров, ко вторым трудоемкость (необходимо знать цепочку вызовов функций). При использовании OpenCV можно выбрать какой язык использовать из большого перечня. Скорость работы программ при этом как правило не зависят от ЯВУ. При этом OpenCV проще всего использовать с Python.

# 2.2.3 Информационная модель программы

Для реализации программы был выбран метод SIFT. Не смотря на меньшею производительность по сравнению с алгоритмом SURF, SIFT способен более точно распознавать характеристические точки на изображении [13].

На рисунке отображена общая схема выполнения программы (рисунок 21).

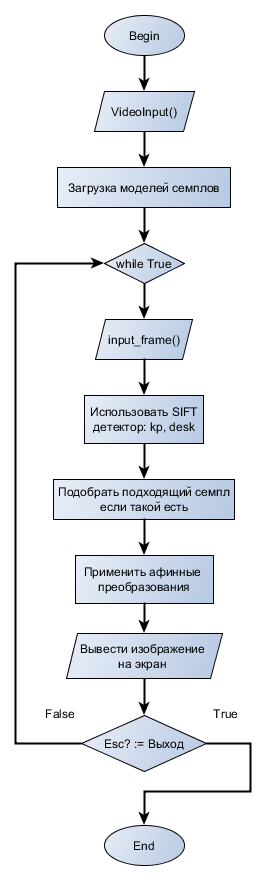


Рисунок 21 – Схема работы программы

# 3 Выводы

Ознакомился и опробовал механизм распознавания образов методом нахождения характеристических точек и построением их дескрипторов алгоритмом SIFT. Воспользовавшись библиотекой OpenCV написал дополнительные модули и основной скрипт для распознавания объектов на подаваемом изображении.

В результате тестирования, программа распознавала гранату F1, обводив ее рамкой и выводила на экран изображение примера класса, к которому относится выделенный объект (рисунок 22, 23).

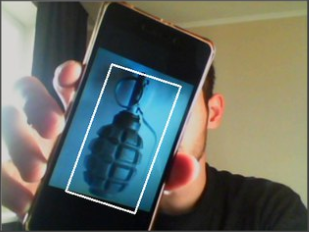


Рисунок 22 - Результат обработки



Рисунок 23 – Пример класса объектов

# Заключение

В ходе выполнения исследования были решены следующие задачи:

1) проанализированы алгоритмы классификации по характеристическим точкам;

2) реализована система распознавания образа.

Таким образом, все поставленные задачи были решены и цель работы, заключающаяся в разработки системы по распознаванию образа, была достигнута.

В следующей работе планируется повысить производительность системы путем перехода на компилируемый язык программирования, а также оптимизации и распараллеливания существующих модулей.

# Список литературы

1. Хорн Б. К. П. Зрение роботов: Пер. с англ. – М.: Мир, 1989. – 487 с., ил. ISBN 5-03-000570-6.
2. Фисенко В. Т., Фисенко Т. Ю. Компьютерная обработка и распознавание изображений: учеб. пособие / В. Т. Фисенко. – СПб: СПбГУ ИТМО, 2008. – 195 с.
3. Конушин А. Слежение за точечными особенностями сцены (Point feature tracking). Компьютерная графика и мультимедиа. Выпуск №1(5)/2003.
4. V. Rodehorst, A. Koschan. Comparison and evaluation of feature point detectors, 2006.
5. Chris Harris, Mike Stephens. A Combined Corner and Edge Detector. Plessey Research Roke Manor, United Kingdom the Plessey Company plc. 1988.
6. Jianbo Shi, Carlo Tomasi. Good Features to Track. IEEE Conference on Computer Vision and Pattern Recognition (CVPR94) Seattle, June 1994.
7. Lowe, D. G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision (2004) 60: 91.
8. Дж. Стокман, Л. Шапиро Компьютерное зрение. Пер. с анг. – М.: БИНОМ. Лаборатория знаний, 2006.
9. Bay H., Ess A., Gool L.V., Tuytelaars T. SURF: speed up robust features. Computer Vision and Image Understanding (CVIU), Vol.110, No.3, 2008. – pp. 3346-359.
10. Дружков П. Н., Золотых Н. Ю., Половинкин А. Н. Параллельная реализация алгоритма предсказания с помощью модели градиентного бустинга деревьев решений. Вестник Южно-Уральского государственного университета. Серия: Математическое моделирование и программирование. 2011. No. 37 (254). С. 82-89.
11. Луис Педро Коэльо, Вилли Ричарт. Построение систем машинного обучения на языке Python. 2-е издание / пер. с англ. Слинкин А. А. – М.: ДМК Пресс, 2016. – 302 с.: ил.
12. Laganiere R. OpenCV 2 Computer Vision Application Programming Cook-book / R. Laganiere –Packt Publishing, 2011. – 298 p.
13. P. M. Panchal, S. R. Panchal, S. K. Shah. A Comparison of SIFT and SURF. International Journal of Innovative Research in Computer and Communication Engineering. ISSN (Print): 2320 – 9798. Vol 1, Issue2, April 2013
14. Mikolajczyk K., Tuytelaars T. Local Invariant Feature Detecors: A Survey. Foundation and Trends in Computer Vision, Vol.3, No. 3, 2007. – pp. 177-280.
15. Понс Ж., Форсайт Д. Компьютерное зрение. Современный подход. М.: Изд. д. Вильямс, 2004. – 465 с.

# Приложение А

**(обязательное)**

**Исходный код обработки изображений на Python**

Листинг А.1 – src.py

import cv2

import os

# Custom import

import videoinput

import utilscv

import objrecogn as orec

def main():

cv2.namedWindow('Features')

str\_src = '0:rows=300:cols=400'

video\_input = videoinput.VideoInput(str\_src)

db\_dict = orec.loadModelsFromDirectory()

while True:

frame = video\_input.read()

detector = cv2.xfeatures2d.SIFT\_create(

nfeatures=250

)

img\_in = cv2.cvtColor(

frame,

cv2.COLOR\_BGR2GRAY

)

img\_out = frame.copy()

kp, desc = detector.detectAndCompute(img\_in, None)

# for debug

print(type(kp))

print(len(desc))

# end for debug

selected\_db = db\_dict['SIFT']

if len(selected\_db) > 0:

img\_match\_mutual = orec.findMatchingMutuosOptimizado(

selected\_db,

desc,

kp

)

min\_inliners = int(20)

projer = float(5)

best\_img, inliners\_web\_cam, inliners\_db = orec.calculateBestImageByNumInliers(

selected\_db,

projer,

min\_inliners

)

if not best\_img is None:

orec.calculateAffinityMatrixAndDraw(

best\_img,

inliners\_db,

inliners\_web\_cam,

img\_out

)

cv2.imshow('Features', img\_out)

ch = cv2.waitKey(5) & 0xFF

if ch == 27:

break

video\_input.close()

cv2.destroyAllWindows()

if \_\_name\_\_ in "\_\_main\_\_":

main()

Листинг А.2 – objrecogn.py

import cv2

import os

import numpy as np

import utilscv

# A Python class has been created, called ImageFeature

# which will contain for each of the images of the database,

# the information needed to compute object recognition.

class ImageFeature(object):

def \_\_init\_\_(self, nameFile, shape, imageBinary, kp, desc):

self.nameFile = nameFile

self.shape = shape

self.imageBinary = imageBinary

self.kp = kp #KeyPoints

self.desc = desc #Descriptors

#Matchings of the image of the database with the image of the webcam

self.matchingWebcam = []

#Matching the webcam with the current image of the database

self.matchingDatabase = []

#Allows to empty previously calculated matching for a new image

def clearMatchingMutuos(self):

self.matchingWebcam = []

self.matchingDatabase = []

# Function for calculating, for each of the methods of calculation of features,

# the features of each of the images of the directory "models"

def loadModelsFromDirectory():

# The method returns a dictionary. The key is the features algorithm

# while the value is a list of objects of type ImageFeature

# where all the data of the features of the images of the

dataBase = dict([

('SIFT', []),

('AKAZE', []),

('SURF', []),

('ORB', []),

('BRISK', [])]

)

#The number of features has been limited to 250 for the algorithm to flow.

sift = cv2.xfeatures2d.SIFT\_create(nfeatures=250)

akaze = cv2.AKAZE\_create()

surf = cv2.xfeatures2d.SURF\_create(800)

orb = cv2.ORB\_create(400)

brisk = cv2.BRISK\_create()

for imageFile in os.listdir("template"):

colorImage = cv2.imread("template/" + str(imageFile))

currentImage = cv2.cvtColor(colorImage, cv2.COLOR\_BGR2GRAY)

#We perform a resize of the image, so that the compared image is equal

kp, desc = sift.detectAndCompute(currentImage, None)

#Features are loaded

dataBase["SIFT"].append(ImageFeature(imageFile, currentImage.shape, colorImage, kp, desc))

kp, desc = akaze.detectAndCompute(currentImage, None)

dataBase["AKAZE"].append(ImageFeature(imageFile, currentImage.shape, colorImage, kp, desc))

kp, desc = surf.detectAndCompute(currentImage, None)

dataBase["SURF"].append(ImageFeature(imageFile, currentImage.shape, colorImage, kp, desc))

kp, desc = orb.detectAndCompute(currentImage, None)

dataBase["ORB"].append(ImageFeature(imageFile, currentImage.shape, colorImage, kp, desc))

kp, desc = brisk.detectAndCompute(currentImage, None)

dataBase["BRISK"].append(ImageFeature(imageFile, currentImage.shape, colorImage, kp, desc))

return dataBase

# Function responsible for calculating mutual matching, but nesting loops

# It is a very slow solution because it does not take advantage of Numpy power

# We do not even put a slider to use this method as it is very slow

def findMatchingMutuos(selectedDataBase, desc, kp):

for imgFeatures in selectedDataBase:

imgFeatures.clearMatchingMutuos()

for i in range(len(desc)):

primerMatching = None

canditatoDataBase = None

matchingSegundo = None

candidateWebCam = None

for j in range(len(imgFeatures.desc)):

valorMatching = np.linalg.norm(desc[i] - imgFeatures.desc[j])

if (primerMatching is None or valorMatching < primerMatching):

primerMatching = valorMatching

canditatoDataBase = j

for k in range(len(desc)):

valorMatching = np.linalg.norm(imgFeatures.desc[canditatoDataBase] - desc[k])

if (matchingSegundo is None or valorMatching < matchingSegundo):

matchingSegundo = valorMatching

candidateWebCam = k

if not candidateWebCam is None and i == candidateWebCam:

imgFeatures.matchingWebcam.append(kp[i].pt)

imgFeatures.matchingDatabase.append(imgFeatures.kp[canditatoDataBase].pt)

return selectedDataBase

# Function responsible for calculating the mutual matching of a webcam image,

# with all the images of the database. Receive as input parameter

# the database based on the method of calculation of features used

# in the image input of the webcam.

def findMatchingMutuosOptimizado(selectedDataBase, desc, kp):

for img in selectedDataBase:

img.clearMatchingMutuos()

for i in range(len(desc)):

# The standard of difference of the current descriptor is calculated, with all

# the image descriptors of the database. We got

# without loops and making use of Numpy broadcasting, all distances

# between the current descriptor with all the descriptors of the current image

distanceListFromWebCam = np.linalg.norm(desc[i] - img.desc, axis=-1)

#Obtain the candidate who is the shortest distance from the current descriptor

candidatoDataBase = distanceListFromWebCam.argmin()

# It is checked if the matching is mutual, that is, if it is in the other direction.

# That is, it is verified that the has the current descriptor as best matching

distanceListFromDataBase = np.linalg.norm(

img.desc[candidatoDataBase] - desc,

axis=-1

)

candidatoWebCam = distanceListFromDataBase.argmin()

#If mutual matching is fulfilled, it is stored for later processing

if (i == candidatoWebCam):

img.matchingWebcam.append(kp[i].pt)

img.matchingDatabase.append(img.kp[candidatoDataBase].pt)

#For convenience they become Numpy ND-Array

img.matchingWebcam = np.array(img.matchingWebcam)

img.matchingDatabase = np.array(img.matchingDatabase)

return selectedDataBase

# This function calculates the best image based on the number of inliers

# which has each image of the database with the image obtained from

# the web camera.

def calculateBestImageByNumInliers(selectedDataBase, projer, minInliers):

if minInliers < 15:

minInliers = 15

bestIndex = None

bestMask = None

numInliers = 0

for index, imgWithMatching in enumerate(selectedDataBase):

#The RANSAC algorithm is computed to calculate the number of inliers

\_, mask = cv2.findHomography(

imgWithMatching.matchingDatabase,

imgWithMatching.matchingWebcam,

cv2.RANSAC,

projer

)

if not mask is None:

# The number of inliers is checked from the mask.

# If the number of inliers exceeds the minimum number of

# and is a maximum (it has more inliers than the image

# then it is considered that it is the image that matches the

# stored in the database.

countNonZero = np.count\_nonzero(mask)

if (countNonZero >= minInliers and countNonZero > numInliers):

numInliers = countNonZero

bestIndex = index

bestMask = (mask >= 1).reshape(-1)

# If an image has been obtained as the best image and, for

# must have a minimum number of inlers, then calculate

# the keypoints that are inliers from the mask obtained in

# and is returned as the best image.

if not bestIndex is None:

bestImage = selectedDataBase[bestIndex]

inliersWebCam = bestImage.matchingWebcam[bestMask]

inliersDataBase = bestImage.matchingDatabase[bestMask]

return bestImage, inliersWebCam, inliersDataBase

return None, None, None

# This function calculates the affinity matrix A, paints a rectangle around

# of the detected object and paints in a new window the image of the database

# corresponding to the recognized object.

def calculateAffinityMatrixAndDraw(bestImage, inliersDataBase, inliersWebCam, imgout):

#The affinity matrix A

A = cv2.estimateRigidTransform(inliersDataBase, inliersWebCam, fullAffine=True)

A = np.vstack((A, [0, 0, 1]))

#Calculate the points of the rectangle occupied by the recognized object

a = np.array([0, 0, 1], np.float)

b = np.array([bestImage.shape[1], 0, 1], np.float)

c = np.array([bestImage.shape[1], bestImage.shape[0], 1], np.float)

d = np.array([0, bestImage.shape[0], 1], np.float)

centro = np.array([float(bestImage.shape[0])/2,

float(bestImage.shape[1])/2, 1], np.float)

# Multiply the points of the virtual space, to convert them into

# real image points

a = np.dot(A, a)

b = np.dot(A, b)

c = np.dot(A, c)

d = np.dot(A, d)

centro = np.dot(A, centro)

areal = (int(a[0]/a[2]), int(a[1]/b[2]))

breal = (int(b[0]/b[2]), int(b[1]/b[2]))

creal = (int(c[0]/c[2]), int(c[1]/c[2]))

dreal = (int(d[0]/d[2]), int(d[1]/d[2]))

centroreal = (int(centro[0]/centro[2]), int(centro[1]/centro[2]))

#The polygon and the file name of the

#image are painted in the center of the polygon

points = np.array([areal, breal, creal, dreal], np.int32)

cv2.polylines(imgout, np.int32([points]),1, (255,255,255), thickness=2)

# For debug

'''

try:

utilscv.draw\_str(imgout, centroreal, bestImage.nameFile.upper())

except:

print("Trouble with darw\_str")

'''

cv2.imshow('ImageDetector', bestImage.imageBinary)

Листинг А.3 – utilscv.py

import cv2

# Function for drawing contrast text in an image:

def draw\_str(dst, x, y, s):

cv2.putText(dst, s, (x+1, y+1), cv2.FONT\_HERSHEY\_PLAIN, 1.0, (0, 0, 0),

thickness=2, lineType=cv2.LINE\_AA)

cv2.putText(dst, s, (x, y), cv2.FONT\_HERSHEY\_PLAIN, 1.0, (255, 255, 255),

lineType=cv2.LINE\_AA)

# Function to trim an ROI over an image of given size:

def fixroi(roi, imshape):

if roi == ((-1, -1), (-1, -1)):

rroi = ((0, 0), (imshape[1], imshape[0]))

else:

rroi = ((max(0, min(roi[0][0], roi[1][0])),

max(0, min(roi[0][1], roi[1][1]))),

(min(imshape[1], max(roi[0][0], roi[1][0])),

min(imshape[0], max(roi[0][1], roi[1][1]))))

return rroi

# Obtain the subpicture given by an ROI:

def subimg(pimg, proi):

return pimg[proi[0][1]:proi[1][1], proi[0][0]:proi[1][0]]

# Write the subpicture given by an ROI:

def setsubimg(img1, img2, proi):

img1[proi[0][1]:proi[1][1], proi[0][0]:proi[1][0]] = img2

Листинг А.4 – videoinput.py

import glob

import cv2

class VideoInput(object):

def \_\_init\_\_(self, str):

largs = map(lambda x: tuple(x.split('=')), str.split(':'))

self.cols, self.rows = 0, 0

self.loop = False

self.type = 'camera'

self.camera = 0

for arg in largs:

if len(arg) == 1:

if arg[0] == 'loop':

self.loop = True

elif arg[0].lower().endswith(('.jpg', '.jpeg',

'.png', '.gif')):

self.type = 'imgfiles'

self.imgfiles = glob.glob(arg[0])

self.imgfiles.sort()

self.curframe = 0

elif arg[0].lower().endswith(('.mpeg', '.mpg', '.dv', '.wmv',

'.avi', '.mp4', '.webm', '.mkv')):

self.type = 'videofile'

self.videofile = arg[0]

self.cap = cv2.VideoCapture(self.videofile)

elif arg[0].isdigit():

self.type = 'camera'

self.camera = int(arg[0])

self.cap = cv2.VideoCapture(self.camera)

else:

var, val = arg

if(var == 'cols'):

self.cols = int(val)

elif(var == 'rows'):

self.rows = int(val)

if self.cols != 0 and self.type == 'camera':

self.cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH, self.cols)

if self.rows != 0 and self.type == 'camera':

self.cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, self.rows)

def read(self):

if self.type in ('camera', 'videofile'):

flag, frame = self.cap.read()

if self.loop and frame is None:

self.cap.set(cv2.CAP\_PROP\_POS\_AVI\_RATIO, 0)

flag, frame = self.cap.read()

elif self.type in ('imgfiles', ):

if self.curframe == len(self.imgfiles):

if self.loop:

self.curframe = 0

else:

return None

frame = cv2.imread(self.imgfiles[self.curframe])

self.curframe += 1

if frame is not None: # Posible escalado

if self.cols != 0 and self.rows != 0:

frame = cv2.resize(frame, (self.cols, self.rows))

elif self.cols != 0:

frame = cv2.resize(frame, (self.cols, frame.shape[0]))

elif self.rows != 0:

frame = cv2.resize(frame, (frame.shape[1], self.rows))

return frame

def close(self):

if self.type in ('videofile', 'camera'):

# try:

self.cap.release()