

E CHALLAN SYSTEM

**E CHALLAN SYSTEM**

**HARDWARE BASE OPEN CV**

**PROJECT**

HARDWARE BASE OPEN CV

PROJECT



**Session 2018-2019**

**Project Advisors**

**Mr. Badar Sami**

**Mr. Muhammad Saeed**

**Name of Co-Supervisor**

**Submitted By**

|  |  |
| --- | --- |
| Summaiya Zulfiqar | MCS-EP-1851048 |
| Aamir Bashir | MCS-EP-1851001 |

**Department of Computer Science**

**University of Karachi**

**Karachi**

MCS

A report submitted to the

FEB 16, 2020

Department of Computer Science

in partial fulfillment of the requirements for the degree of

Masters in

Computer Science

by

Summaiya Zulfiqar

Aamir Bashir

University of Karachi

Dec 30, 2019

Acknowledgements

We truly acknowledge the cooperation and help make by Dr. ABC, Department of Computer Science University of Karachi. He has been a constant source of guidance throughout the course of this project. We would also like to thank Prof. Dr.ABC for his help and guidance in understanding Queuing Theory. We are also thankful to our friends and families whose silent support led us to complete our project.

(Signed)

Summaiya Zulfiqar EP-1851048

Aamir Bashir EP-1851001

Date

Dec 30, 2019

Table Of Contents

Statement Of Submission i

Acknowledgements ii

Abstract iii

Clients Approval Letter iv

List Of Figures v

List Of Tables vi

Table Of Contents vii

Chapter 1: Introduction 1

1.1.1 Introduction 2

1.1.2 Stretegic Objectives 3

1.1.3 Levels Of Identification 5

1.1.4 Environment Of The System 6

1.1.1.A Digital Analysis 7

1.1.1.B Work Plan 8

1.1.1.C Development Of System

1.2 Working Procedure 12

1.2.1 Open CV 13

1.2.1.A Open CV Python 14

1.2.1.B Contributers 15

1.2.2 Steps For Developing Pycharm Installation 22

1.3 Introducton To Pycharm In Python 31

1.3.1 Launching Steps 32

1.3.2 Pycharm KeyMaps 33

Chapter 2: Research Works And Implemented Details 37

2.1 International Conference 8th 38

2.1.1 Scope & Topics 39

2.1.1.A Paper Submission 40

2.1.1.B Important Dates 41

2.2 Research Conducted By Ictia 42

2.2.1.Scope & Topics 43

2.3 Complete View Implementation 44

2.4 Latest Article 45

2.5 Second Article 46

2.5.1 Breif Report US Patent 47

Chapter 3 The E Challan System 48

3.1 Project Description 49

3.2 Goals & Objectives 50

3.3 Main Idea Of The System 51

3.4 Usage Criteria 52

3.4.1.Users 53

3.5 Development & Design 54

3.5.1 G-API High Level Design 55

3.5.2 G-API Framework Architecture 56

3.5.3API Layer 57

3.5.4.Backends Layer 58

3.5.5 Graph Execution 59

3.6 Public Expectattions 60

3.7 Risk Assesment 62

Chapter 4 Analysis & Observations 63

4.1 Examination & Estimation 64

4.1.1 Image Segmentation With Watershed Algorithm 65

4.2 Background Capturing Analysis 66

4.3 Formation Of Color Detection Methodology 67

4.3.1 Open CV & Python Color Detection 68

4.3.2 List & Boundaries 69

4.3.4 Open CV & Python Color Detection 70

4.4 Objects Motion Detection 71

4.4..1 Classes 72

4.4.2.Python 73

4.5 Tesseract Library To Convert Image To Text 74

4.5.1 Applyong OCR With Tesseract & Python 75

4.5.2 Using OCR with Python 76

4.5.3 Summary 77

Chapter 5: Refrences 84

5.1 Refrences 85

5.2 Ref To Chapter 1 86

5.3 Ref To Chapter 2 87

5.4 Ref To Chapter 3 88

5.5 Ref To Chapter 4 38

Appendix A: Glossary Of Terms 139

Appendix B: Ideal Model 140

Appendix C: P-CMM Maturity Level–3

**CHAPTER 1 Introduction**

**Abstract**

The main objective of this project is to design a Quick Response code which contains overall information of the vehicle and to introduce a system which helps us to identify the owner of the vehicle and take fine easily from them. Breaking traffic rules on road is a major issue now a day. Due to large population and traffic congestion it is difficult to identify which vehicle has broken traffic rules. To monitor culprit vehicle manually is very difficult. Therefore, there is need to monitor these vehicles automatically. Hence, we propose a smart monitoring system that will monitor the culprit vehicle. The proposed system will generate detailed data at the time of incident and that data by monitoring the number plate, it will be send to the nearby station’s database which is handling by the authorized person. The existing system is tedious and time consuming, it requires stopping the vehicle, collecting the owner information, checking his license and collecting Fine, giving the acknowledgment and then allowing the vehicle to pass by. The Challan System for vehicle verification is a new system which is designed to enhance the convenience for officers and vehicle owners. It saves lots of time.

**1.1.1 Introduction**

Due to traffic congestion and increasing number of user on road it is difficult to identify every vehicle uniquely. To monitor the vehicle manually is very difficult task. The aim of this research is to automate the identification of the vehicle. In today’s time there are so many people breaking traffic rules without any fear, one of them is jumping red light signals. Sometimes due to this the vehicle meets an accident with another vehicle. In today’s world there are lots of junctions and crossings of road where traffic lights are fitted, but at only few places the duty of traffic police is assigned where they monitor traffic with the ever increasing vehicles on road and the number of users on road, limited resources are available to monitor the vehicle uniquely. Hence, an intelligent monitoring of vehicle uniquely is an important issue to be considered. The traffic monitoring authorities need to find new methods of overcoming this difficulty. In our research, we are designing a system which will tell about Vehicle by just scanning the number plate which is situated on vehicle.

**1.1.2 Strategic Objectives**

The main strategic objectives of the project are to provide the people a convenient way by monitoring the signal light and the vehicles we can easily capture that who is disturbing the traffic rules In this way it will be very easy for the road safety purpose. And for all the people by having a clean and discipline traffic throughout all the roads the purpose of e-challan system is to make sure that everyone is following the traffic signals in a proper way.

A strategic objective of our project is to focus on the selling and buying of the cars and those who are selling definitely they will think about the technology that their car will be captured with the proper name with all the identity having all the documents of the car by the e-challan system this way we can provide our community a fraud free e environment for buying and selling the cars also.

**1.1.3 Levels of Identification**

The levels of identification of the e-challan system or to provide a focus full road safety purposes and a truly clean environment.

The levels includes:

1. Safety
2. cleanliness
3. proper way
4. proper turns
5. proper lines of the vehicles
6. proper road traffic
7. Proper go through ways.

Environment level also includes the identification of the man who is also driving the car exactly, the location and the time of the car which passed by the traffic signal at the time of red light**.**

**1.1.4 Environment Of The System**

Environment of the system is based on roads having a heavy traffic and we are having a cameras on every side of the road covering the zebra crossing of the roads in order to give the proper way. We are providing a space behind the traffic lights so that everyone can see the cameras also and for the security purpose also.

The main part of the system is the On & Off gradually, we are increasing the on off system also the moving system of the camera.

For the environment we are having camera protector that covers the camera and traffic light protrctor that should properly cover the roads including the traffic lights and In this way, it will save the cameras from heavy rain also the traffic lights. There may be environment change for the safety these things are needed.

**1.1.1.A Digital Analysis**

The data analysis stands for analyzing and computing the data which is stored or sent from the database system. There may be some or many data that has to be send from the database in the form of text to the officer police or to the RTO officer that is doing the job or monitoring the cameras or to the nearby the police station in order to cut the challan by observing the E-challan systems call digital data is sent in form of text message or in the form of email or in the form of any kind of message to the nearby computer Or maybe a mobile number that is fixed or entered in the E-challan system’s Module. The number plate text is very helpful that the police officer can get the detail of the person who is driving and any illegal act that is done by the driver. That can also easily monitored.

**1.1.1.B Work Plan**

The work plan includes:

1. The turning on of the camera:
2. the turning on of the signal light to red

When the signal light turns to green the camera won't observe any car that’s crossing the route.

Our plan also includes the sensors installed, the working procedure of the software and the backend process and backup processes that is to handle by the team management. To reduce the risk there will be a hardware system that is collecting information and stores data which can be used for 10 to 20 days. After the limit exceeds this data will be deleted and the process of new data storing would be start. That's the work Plan. Of the project

**1.1.1.C Development Of System**

The development of the system is the measure of the software update.

*All the programs which is observed by the camera in the form of cars, trucks, motorcycles and other vehicles on the road. to observe the new arrival of the cars there would be a proper development program, which is also monitoring the new systems of the cars which are arriving on the route. And by the movement and their shape itself will able to update itself after three to four times of storing data.*

**1.2 Working Procedure**

**1.2.1 OpenCV**

OpenCV was started at Intel in 1999 by Gary Bradsky, and the first release came out in 2000. Vadim Pisarevsky joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle that won the 2005 DARPA Grand Challenge. Later, its active development continued under the support of Willow Garage with Gary Bradsky and Vadim Pisarevsky leading the project. OpenCV now supports a multitude of algorithms related to Computer Vision and Machine Learning and is expanding day by day.

OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS. Interfaces for high-speed GPU operations based on CUDA and OpenCL are also under active development.

OpenCV-Python is the Python API for OpenCV, combining the best qualities of the OpenCV C++ API and the Python language.

**1.2.1A OpenCV-Python**

OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

Python is a general purpose programming language started by Guido van Rossum that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability.

Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second, it easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

**1.2.1 B Contributors**

Below is the list of contributors who submitted tutorials to OpenCV-Python.

Alexander Mordvintsev (GSoC-2013 mentor)

Abid Rahman K. (GSoC-2013 intern)

Additional Resources

 A Quick guide to Python –

1. [A Byte of Python](http://swaroopch.com/notes/python/)
2. [Basic Numpy Tutorials](http://wiki.scipy.org/Tentative_NumPy_Tutorial)
3. [Numpy Examples List](http://wiki.scipy.org/Numpy_Example_List)
4. [OpenCV Documentation](http://docs.opencv.org/)
5. [OpenCV Forum](http://answers.opencv.org/questions/)

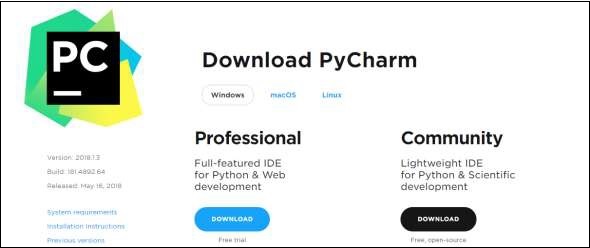
**1.2.2 Steps For Developing Pycharm Installation**

*Steps Involved*

*You will have to follow the steps given below to install PyCharm on your system. These steps show the installation procedure starting from downloading the PyCharm package from its official website to creating a new project.*

*Step 1*

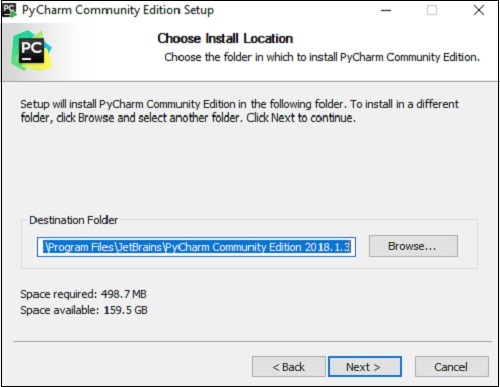
*Download the required package or executable from the official website of PyCharm*[*https://www.jetbrains.com/pycharm/download/#section=windows*](https://www.jetbrains.com/pycharm/download/#section=windows)*Here you will observe two versions of package for Windows as shown in the screenshot given below −*

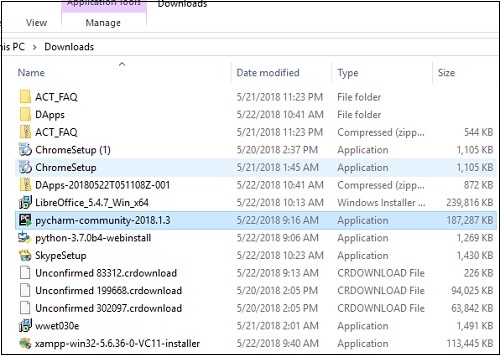
**

*Note that the professional package involves all the advanced features and comes with free trial for few days and the user has to buy a licensed key for activation beyond the trial period. Community package is for free and can be downloaded and installed as and when required. It includes all the basic features needed for installation. Note that we will continue with community package throughout this tutorial.*

*Step 2*

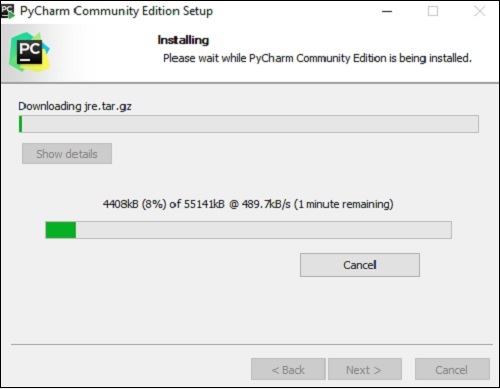
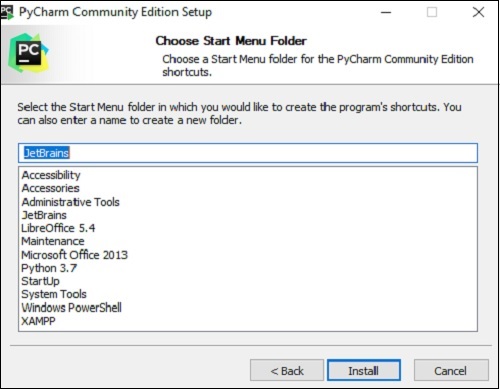
*Download the community package (executable file) onto your system and mention a destination folder as shown below −*

**

**

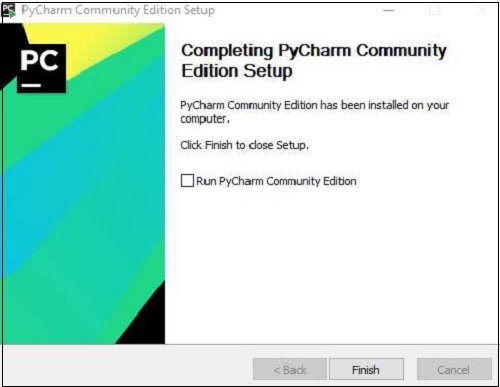
*Step 3*

*Now, begin the installation procedure similar to any other software package.*

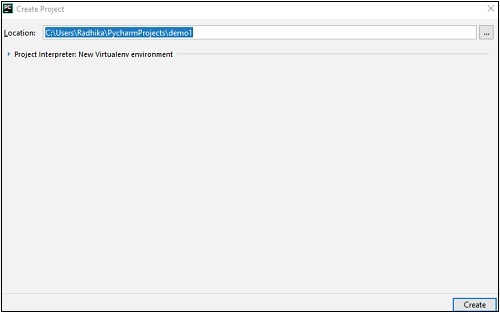
**

*Step 4*

*Once the installation is successful, PyCharm asks you to import settings of the existing package if any.*

**

**

**

*This helps in creating a new project of Python where you can work from the scratch. Note that unlike other IDEs, PyCharm only focusses on working with projects of Python scripting language.*

**1.3 Introduction To Pycharm In Python**

PyCharm is the most popular IDE for Python, and includes great features such as excellent code completion and inspection with advanced debugger and support for web programming and various frameworks. PyCharm is created by Czech company, Jet brains which focusses on creating integrated development environment for various web development languages like JavaScript and PHP. PyCharm is the most popular IDE used for Python scripting language. This chapter will give you an introduction to PyCharm and explains its features.

PyCharm offers some of the best features to its users and developers in the following aspects −

* Code completion and inspection
* Advanced debugging
* Support for web programming and frameworks such as Django and Flask

Features of PyCharm

Besides, a developer will find PyCharm comfortable to work with because of the features mentioned below −

Code Completion

PyCharm enables smoother code completion whether it is for built in or for an external package.

SQLAlchemy as Debugger

You can set a breakpoint, pause in the debugger and can see the SQL representation of the user expression for SQL Language code.

Git Visualization in Editor

When coding in Python, queries are normal for a developer. You can check the last commit easily in PyCharm as it has the blue sections that can define the difference between the last commit and the current one.

Code Coverage in Editor

You can run **.py** files outside PyCharm Editor as well marking it as code coverage details elsewhere in the project tree, in the summary section etc.

Package Management

All the installed packages are displayed with proper visual representation. This includes list of installed packages and the ability to search and add new packages.

Local History

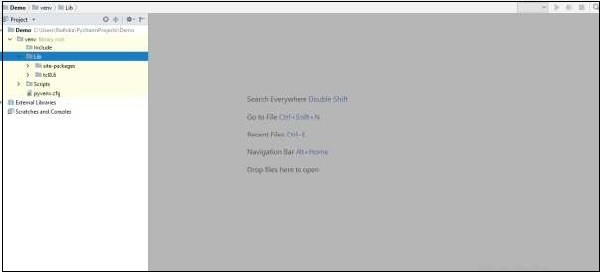
Local History is always keeping track of the changes in a way that complements like Git. Local history in PyCharm gives complete details of what is needed to rollback and what is to be added.

Refactoring

Refactoring is the process of renaming one or more files at a time and PyCharm includes various shortcuts for a smooth refactoring process.

User Interface of PyCharm Editor

The user interface of PyCharm editor is shown in the screenshot given below. Observe that the editor includes various features to create a new project or import from an existing project.



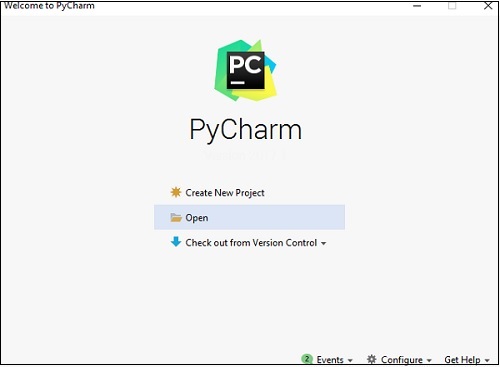
From the screenshot shown above, you can see the newly created project Demo and the **site-packages** folder for package management along with various other folders.

You can download the PyCharm Editor and read its official documentation at this link − <https://www.jetbrains.com/pycharm/>

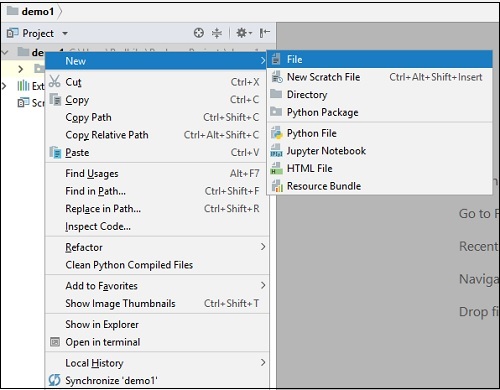
**1.3.1 Launching Steps**

When you launch PyCharm for the first time, you can see a welcome screen with entry points to IDE such as −

* Creating or opening the project
* Checking out the project from version control
* Viewing the documentation
* Configuring the IDE



Recall that in the last chapter, we created a project named **demo1** and we will be referring to the same project throughout this tutorial. Now we will start creating new files in the same project to understand the basics of PyCharm Editor.



The above snapshot describes the project overview of demo1 and the options to create a new file. Let us create a new file called **main.py**.

The code included in main.py is as follows −

y = 3

def print\_stuff():

print ("Calling print\_stuff")

print (y)

z = 4

print (z)

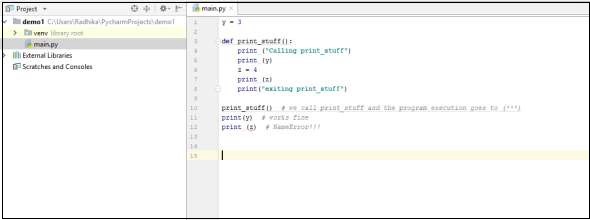
print("exiting print\_stuff")

print\_stuff() # we call print\_stuff and the program execution goes to (\*\*\*)

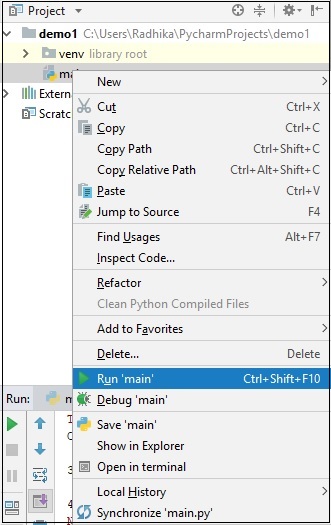
print(y) # works fine

print (z) # NameError!!!

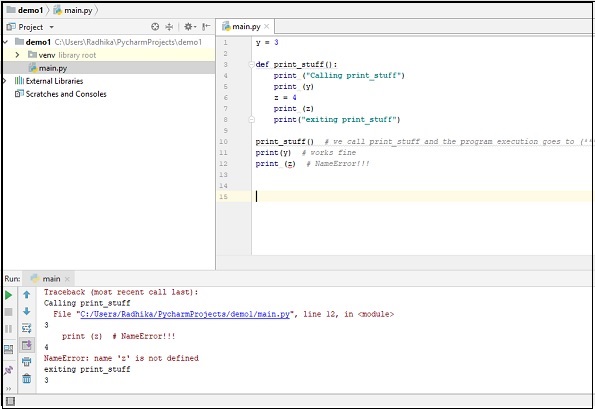
The code created in the file **main.py** using PyCharm Editor is displayed as shown below −



This code can be run within IDE environment. The basic demonstration of running a program is discussed below −



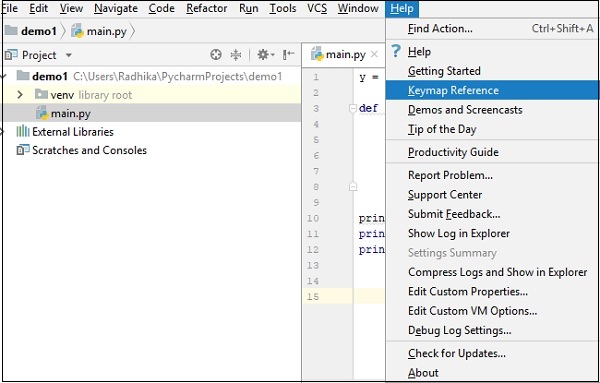
Note that we have included some errors within the specified code such that console can execute the code and display output as the way it is intended to.



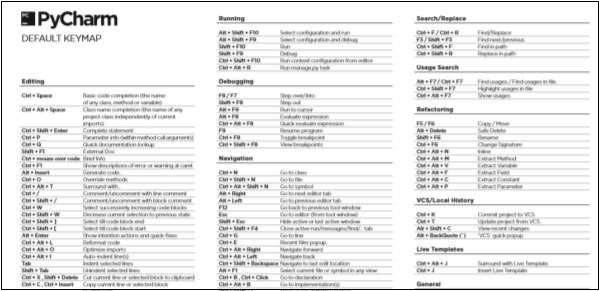
**1.3.2 Pycharm Key Maps**

PyCharm includes various Keymaps to show the most-used commands in the editor. This chapter discusses Keymaps in detail.

You can find the list of Keymaps available in the file menu **Help -> Keymap Reference** as shown in the screenshot given below −

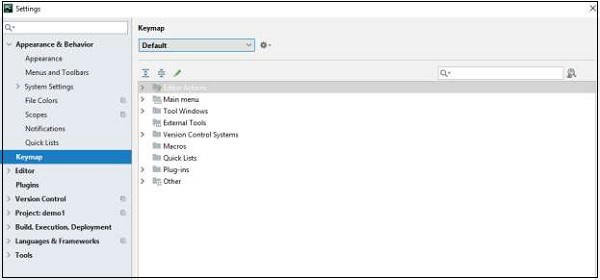


You can find the list of Keymaps and the available shortcuts in PDF format as shown below −



Note − The default Keymap for Windows and Linux operating systems is default, while in Mac OS the default Keymap is OSX 10.5.

You can also view the list of Keymaps available using the **Settings** option in Windows and Linux Operating system (Preferences in Mac OS) as shown in the screenshot given below −



The default Keymap includes various sections for Editor Actions, Main Menu, Tool Windows, External tools, Version Control System, Macros, Quick Lists, Plug-ins and Other options as well.

**Chapter 2: Research Works And Implemented Details**

**2.1 8th International Conference on Signal, Image Processing and Pattern Recognition (SIPP 2020)**

**March 21~22, 2020, Vienna, Austria**

**https://ccsit2020.org/sipp/index.html**

**2.1.1 Scope & Topics**

8th International Conference on Signal, Image Processing and Pattern Recognition (SIPP 2020) is a forum for presenting new advances and research results in the fields of Signal and Image Processing. The conference will bring together leading researchers, engineers and scientists in the domain of interest from around the world.

Authors are solicited to contribute to the conference by submitting articles that illustrate research results, projects, surveying works and industrial experiences. Topics of interest include, but are not limited to, the following:

Topics of interest include, but are not limited to, the following:

* Image Acquisition & Medical Image Processing
* Pattern Recognition and Analysis
* Visualization
* Image Coding and Compression
* Face Recognition & Super Resolution Imaging
* Image Segmentation
* -D and Surface Reconstruction
* 3D and Stereo Imaging
* Analog and Mixed Signal Processing
* Application & Others 
* Applications (Biomedical, Bioinformatics, Genomic, Seismic, Radar, Sonar, Remote Sensing, Positioning, etc.)
* Array Signal Processing
* Audio/Speech Processing and Coding
* Digital & Mobile Signal Processing
* Statistical & Optical Signal Processing
* Data Mining Techniques
* Motion Detection
* Content-Based Image Retrieval
* Video Signal Processing
* Watermarking
* Detection and Estimation of Signal Parameters
* Signal Identification
* Nonlinear Signals and Systems
* Time-Frequency Signal Analysis
* Signal Reconstruction
* Spectral Analysis
* Filter Design and Structures
* FIR, IIR, Adaptive Filters
* Signal Noise Control
* Multiple Filtering and Filter Banks
* Performance Evaluation
* Radar Signal Processing
* Remote Sensing
* Segmentation
* Sensor Array and Multi-Channel Processing
* Biomedical Imaging Technologies
* Biometrics and Pattern Recognition
* Cognitive and Biologically Inspired Vision
* Color and Texture
* Communication Signal processing
* Computer Communication and Networks
* Computer Vision & VR
* Constraint Processing
* Distributed Source Coding
* Document Recognition
* DSP Implementation and Embedded Systems
* Face and Gesture

**2.1.1A Paper Submission**

Authors are invited to submit papers through the conference Submission System by February 08, 2020. Submissions must be original and should not have been published previously or be under consideration for publication while being evaluated for this conference. The proceedings of the conference will be published by Computer Science Conference Proceedings in Computer Science & Information Technology (CS & IT) series (Confirmed).

Selected papers from SIPP 2020, after further revisions, will be published in the special issues of the following journals

* Signal & Image Processing : An International Journal (SIPIJ)
* International Journal Of VLSI Design & Communication Systems (VLSICS)
* International Journal of Embedded Systems and Applications (IJESA)
* The International Journal of Multimedia & Its Applications (IJMA) - ERA Indexed
* Information Technology in Industry (ITII) New- ESCI(WOS) Indexed

**2.1.1B Important Dates**

Second Batch :( Submissions after December 22, 2019)

* Submission Deadline : February 08, 2020
* Authors Notification : March 05, 2020
* Registration & Camera-Ready Paper Due : March 10, 2020

CONTACT US

Here's where you can reach us: sipp@ccsit2020.org or [sipp\_conf@yahoo.com](mailto:sipp_conf@yahoo.com)

**2.2 Research Conducted By Icaita**

**7 th International Conference on Signal and Image Processing (Signal 2020) July 11~12, 2020, Toronto, Canada** [**https://icaita2020.org/signal/index.html**](https://icaita2020.org/signal/index.html)

**2.2.1 Scope & Topics**

7th International Conference on Signal and Image Processing (Signal 2020) is a forum for presenting new advances and research results in the fields of Digital Image Processing. The conference will bring together leading researchers, engineers and scientists in the domain of interest from around the world. The scope of the conference covers all theoretical and practical aspects of the Signal, Image Processing & Pattern Recognition.

Authors are solicited to contribute to the conference by submitting articles that illustrate research results, projects, surveying works and industrial experiences that describe significant advances in the following areas, but are not limited to

Topics of interest include but are not limited to the following:

● 3-D and Surface Reconstruction

● 3D and Stereo Imaging

● Analog and Mixed Signal Processing

● Application & Others

● Applications (Biomedical, Bioinformatics, Genomic, Seismic, Radar, Sonar, Remote Sensing, Positioning, etc.)

● Array Signal Processing

● Audio/Speech Processing and Coding

● Content-based Image Retrieval

● Data Mining Techniques

● Detection and Estimation of Signal Parameters

● Digital & Mobile Signal Processing

● Face Recognition

● Super-Resolution Imaging

● Image Acquisition & Medical Image processing

● Image Coding and Compression

● Image Segmentation

● Motion Detection

● Nonlinear Signals and Systems

● Pattern Recognition and Analysis

● Signal Identification

● Signal Reconstruction

● Spectral Analysis

● Statistical & Optical Signal Processing

● Time-Frequency Signal Analysis

● Image-Based Modeling

● Image and Video Retrieval

● Image Processing & Understanding

● Internet Signal Processing

● Knowledge Representation and High-Level Vision

● Medical Image Analysis

● Multi-view Geometry

● Multidimensional Signal Processing

● Multiple Filtering and Filter Banks

● Neural Networks and Genetic Algorithms

● Object Detection, Recognition and Categorization

● Pattern Recognition in New Modalities

● Video Signal Processing

● Visualization

● Watermarking

● Sonar Signal Processing and Localization

● Speech, Audio and Music Processing

● Statistic Learning & Pattern Recognition

● Text Processing

● Biometrics and Pattern Recognition

● Cognitive and Biologically-Inspired Vision

● Color and Texture

● Communication Signal processing

● Computer Communication and Networks

● Computer Vision & VR Constraint Processing

● Biomedical Imaging Technologies

● DE for Image Processing

● Distributed Source Coding

● Document Recognition

● DSP Implementation and Embedded Systems Face and Gesture

● Filter Design and Structures

● FIR, IIR and Adaptive Filters

● Hardware Implementation for Signal Processing

● Higher Order Spectral Analysis

● Remote Sensing

● Segmentation

● Sensor Array and Multi-channel Processing

● Shape Representation

● Signal Noise Control

● Signal Processing Education

● Time-Frequency/Time-Scale Analysis

● Video Analysis and Event Recognition

● Video compression & Streaming

● Video Surveillance and Monitoring

● Others

Paper Submission Authors are invited to submit papers through the Submission System by January 04, 2020. Submissions must be original and should not have been published previously or be under consideration for publication while being evaluated for this conference. The proceedings of the conference will be published by Computer Science Conference Proceedings in Computer Science & Information Technology (CS&IT) series (Confirmed).

Selected papers from Signal-2020, after further revisions, will be published in the special issue of the following journals

● Signal & Image Processing : An International Journal (SIPIJ)

● International Journal of VLSI Design & Communication Systems (VLSICS)

● International Journal of Embedded Systems and Applications (IJESA)

● International Journal on Organic Electronics (IJOE)

● Information Technology in Industry (ITII)New - ESCI(WOS) Indexed

Important Dates

● Submission Deadline: January 04, 2020

● Authors Notification: February 20, 2020

● Registration & Camera-Ready Paper Due: February 29, 2020

Contact Us: Here’s where you can reach us : signal@icaita2020.org Submission System : https://icaita2020.org/submission/index.php

**2.3 Complete View Implementation**

The new system identifies the violators of the traffic rules with the help of modern Automatic Vehicle Number Plate Recognition (ANPR) cameras, which PSCA installed across different roads of the city.

The E-Challan is sent to the violators’ addresses registered against their vehicle IDs, containing the information regarding how the driver violated the traffic rules with the picture of the vehicle posted on the challan paper.



**2.4 Latest Articles**

# Directional statistics-based quality measure for spotlight color images

* [Authors](https://link.springer.com/article/10.1007/s11760-020-01653-z#authors)
* [Authors and affiliations](https://link.springer.com/article/10.1007/s11760-020-01653-z#authorsandaffiliations)
* F. Kerouh
* D. Ziou
* Q. Jiang

.

Original Paper

**First Online:**11 February 2020

## Abstract

The present paper addresses a new problem related to measuring the quality of color spotlight images. Its primary aim is to raise the limitation problem of available cameras to reproduce the spotlight colors, especially at night. To address this issue, a new reduced reference quality measure is proposed to measure the spotlight color degradation. The idea focuses on transforming the color information into another space where it is defined as an orientation represented on a unit sphere. Then, the directional statistics-based von Mises–Fisher probability density function is used as a deviation measure. To validate the proposed model, a new collection of widely used spotlight color images is constructed. The collection contains a hundred of spotlight colors captured by different cameras in Sherbrooke city as well as available images on the Web. Obtained results are promising.

**2.5 Second Article**

# Signal, Image and Video Processing

**ISSN: 1863-1703 (Print) 1863-1711 (Online)**

## Description

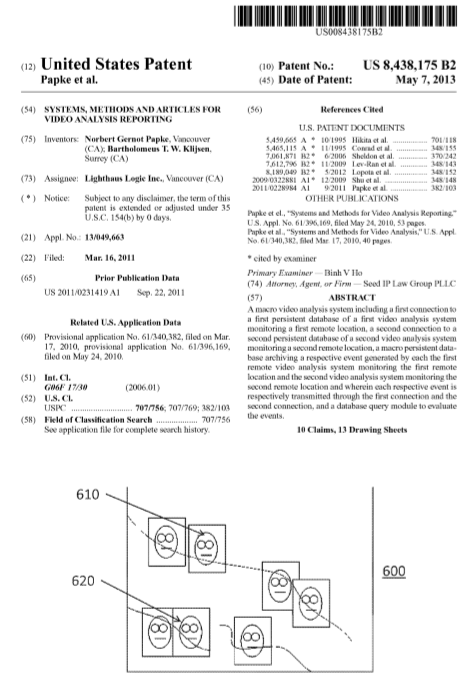
The journal is an interdisciplinary journal presenting the theory and practice of signal, image and video processing.  It aims at:

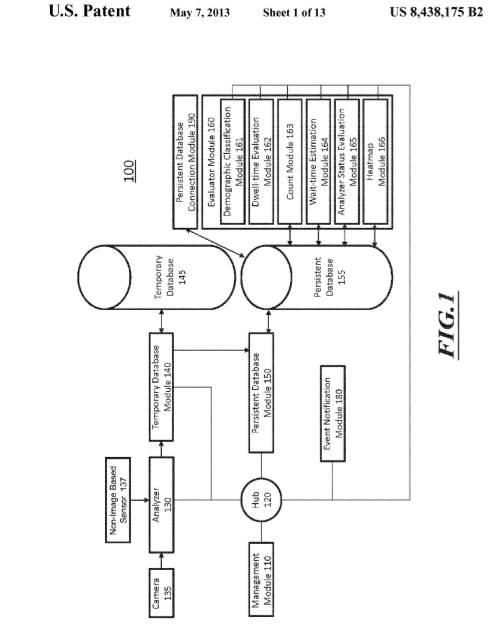
* Disseminating high level research results and engineering developments to all signal, image or video processing researchers and research groups.
* Presenting practical solutions for the current signal, image and video processing problems in Engineering and Science

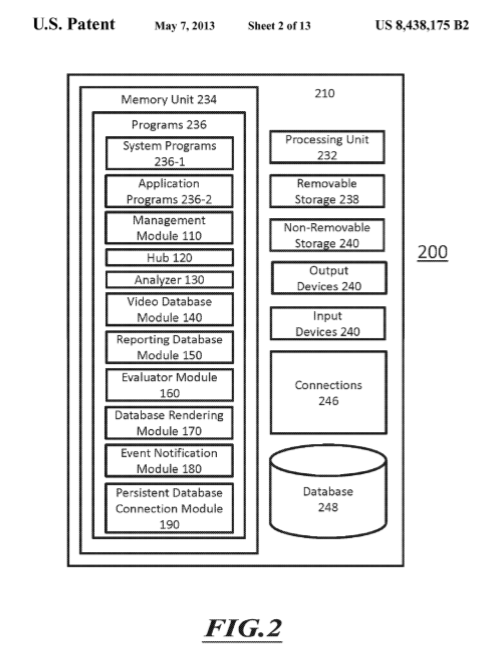
It features original research work, review and tutorial papers and accounts of practical developments.  It is intended for the rapid dissemination of knowledge and experience to Scientists and Engineers working in any area related to or using signal, image and video processing.

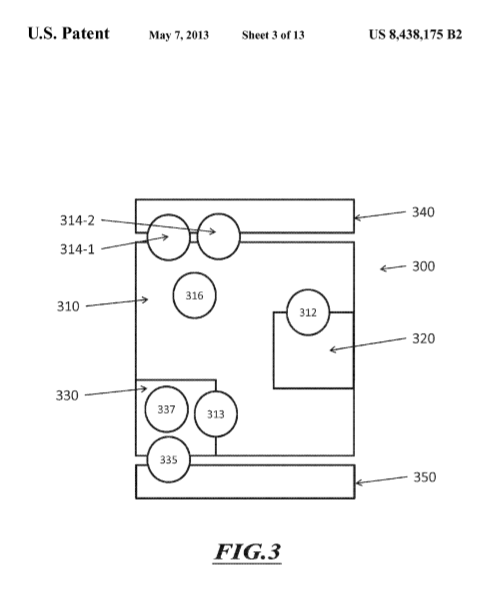
The editorial policy and the technical content of the journal are the responsibility of the Editor-in-Chief and the Editorial Board.  The journal welcomes contributions from every country in the world.  All submissions are peer reviewed by anonymous referees.

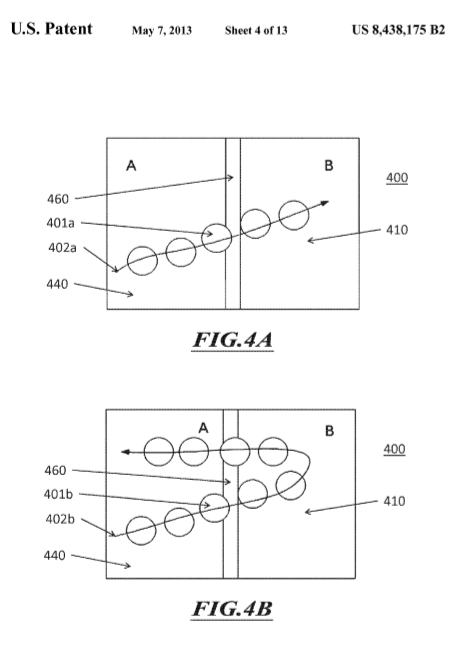
**2.5.1 Brief Report US Patent**

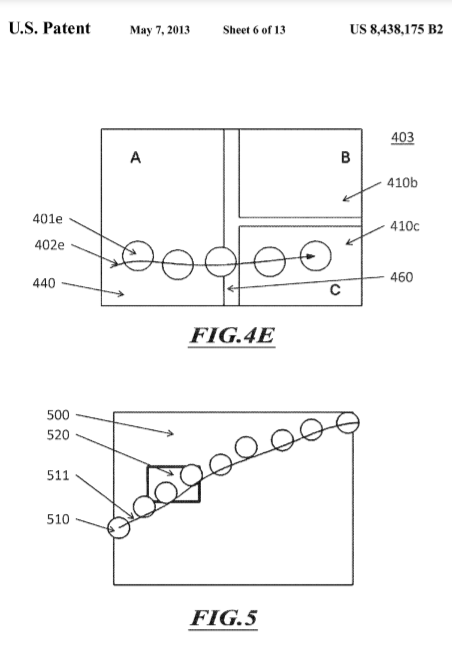


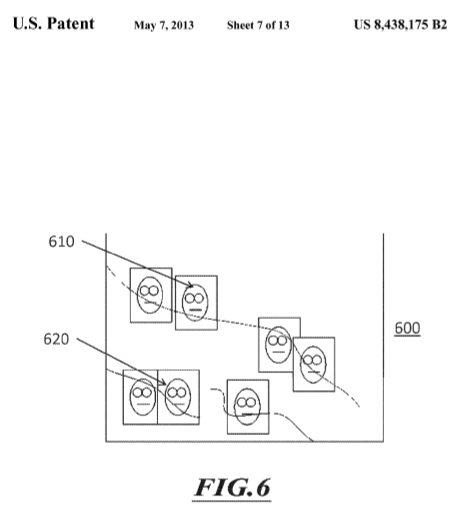


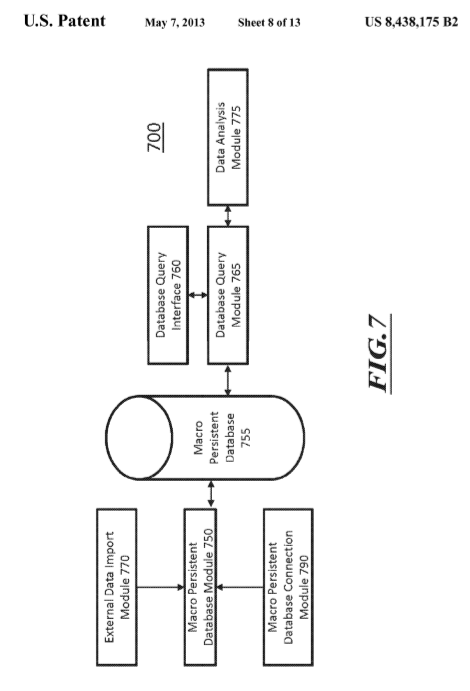


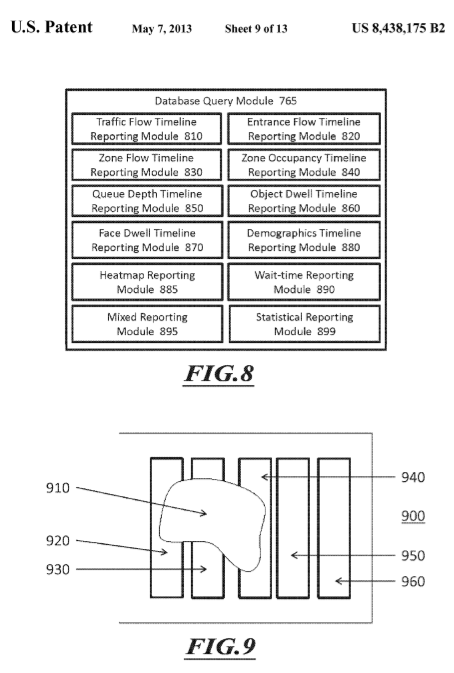


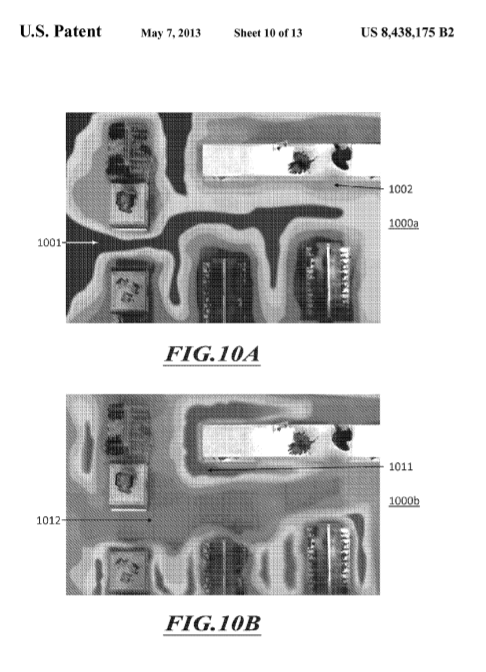












**Chapter 3: The E-Challan System**

**3.1 Project Description**

Traffic Violation reporting System (e-Challan) is a hardware base application that helps the Traffic Police in verifying license status of a person who has violated the traffic laws by tracking vehicle on the spot Information related to the license holder is retrieved using Desktop-base Window Service. Furthermore, It will resolve the issue of fake licenses, which are being used by individuals, making police department more equipped to enforce rule of law.

The project will also help improve the quality of driving, create awareness regarding driving problems, improve traffic standards and effective traffic control or roads, and help decrease the accident ratio.

The project objectives are also aligned with the government’s efforts to curb the menace of corruption from public offices and make the service mechanism easy and transparent.

**3.2 Goals And Objectives**

When a vehicle passes vehicle detector fitted before signal pole, camera captures the image and its registration number is read and instantly checked against database records of vehicle. A record of all vehicles that are breaking signals is stored. Records are shared to police department for further enquiry. Following hardware devices would be required to achieve this system:

* A Vehicle detector.
* A Camera.

Following tasks would be required to achieve this system:

* Develop software that extract plate number of a vehicle from captured image and read data from database against that number.
* Store records of captured vehicles in database.
* Share records with police department.

**3.3 Main Idea of the System**

The main idea of the system is developed at clean and cure environment for the traffic system which is very bad in our country. The special areas like Karachi and other cities which are having heavy traffic, they are facing through many, many problems the problems like breaking traffic rules, signals and not following the zebra crossing, not following the traffic lines and many cases and many accidental cases this has to be stop by some advanced technology and using some advanced system. So we are trying to develop such system which would definitely monitor the traffic breaking person and it will monitor all the system and it will give information about the vehicle owner and this way these things can be handled.

**3.4 Usage Criteria**

Here we are introducing the usage of e challan system the e-challan system. The system will work for a camera which will be connected with the software having all the codes related to the open CV as mentioned in the working procedure of open CV above. Now by using pycharm modules in python the program is prepared by checking all the things including how many cameras are connected with the system (pc or computer) .In this we will need a system that may be a computer system to run the code using Python language or if we talk about the usage criteria when the vehicle on the road crosses the red traffic signal light the system will monitor its number plate and it will convert it into text and it will give or send it to the nearby police station or nearby on duty officer who is monitoring the cameras, in this way the traffic police station can take action against the culprit

**3.4.1 Users**

The users of the system will be those who are working in nearby station. The developers working in station are responsible to monitor the camera correctly and if there, some problem arose they will have to maintain it properly and have to get to it wisely

**3.5 Development And Design**

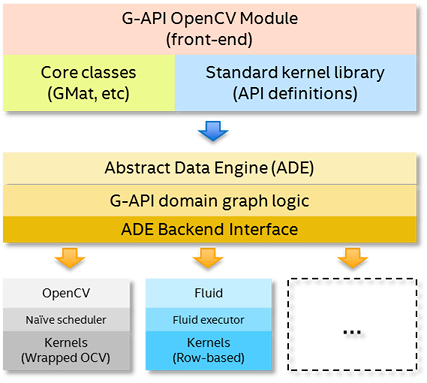
The development and the design of our system will be in process within a year. The system upgrade or software upgrade will be according to the available updates that we will fetch from the cloud or from a well-known company that’s having an updated code and design, in this way the process of betterment can be seen.

Some Updates About The Open CV is as follow

**3.5.1 G-API High-level design overview**

G-API is a heterogeneous framework and provides an unified API to program image processing pipelines with a number of supported backends.

The key design idea is to keep pipeline code itself platform-neutral while specifying which kernels to use and which devices to utilize using extra parameters at graph compile (configuration) time. This requirement has led to the following architecture:



**3.5.2 G-API framework architecture**

There are three layers in this architecture:

* **API Layer** – this is the top layer, which implements G-API public interface, its building blocks and semantics. When user constructs a pipeline with G-API, he interacts with this layer directly, and the entities the user operates on (like [**cv::GMat**](https://docs.opencv.org/master/df/daa/classcv_1_1GMat.html) or [**cv::GComputation**](https://docs.opencv.org/master/d9/dfe/classcv_1_1GComputation.html)) are provided by this layer.
* **Graph Compiler Layer** – this is the intermediate layer which unrolls user computation into a graph and then applies a number of transformations to it (e.g. optimizations). This layer is built atop of [**ADE Framework**](https://docs.opencv.org/master/d4/ddd/gapi_impl.html#gapi_detail_ade).
* **Backends Layer** – this is the lowest level layer, which lists a number of *Backends*. In contrast with the above two layers, backends are highly coupled with low-level platform details, with every backend standing for every platform. A backend operates on a processed graph (coming from the graph compiler) and executes this graph optimally for a specific platform or device.

**3.5.3 API layer**

API layer is what user interacts with when defining and using a pipeline (a Computation in G-API terms). API layer defines a set of G-API *dynamic* objects which can be used as inputs, outputs, and intermediate data objects within a graph:

* [**cv::GMat**](https://docs.opencv.org/master/df/daa/classcv_1_1GMat.html)
* [**cv::GScalar**](https://docs.opencv.org/master/d9/d98/classcv_1_1GScalar.html)
* [**cv::GArray**](https://docs.opencv.org/master/d3/d44/classcv_1_1GArray.html) (template class)

API layer specifies a list of Operations which are defined on these data objects – so called kernels. See G-API [**core**](https://docs.opencv.org/master/df/d1f/group__gapi__core.html) and **[imgproc](https://docs.opencv.org/master/d2/d00/group__gapi__imgproc.html)** namespaces for details on which operations G-API provides by default.

G-API is not limited to these operations only – users can define their own kernels easily using a special macro [**G\_TYPED\_KERNEL()**](https://docs.opencv.org/master/de/d7f/gkernel_8hpp.html#a83d2e73c619a545f69e89bf7b03cbd80).

API layer is also responsible for marshalling and storing operation parameters on pipeline creation. In addition to the aforementioned G-API dynamic objects, operations may also accept arbitrary parameters (more on this [**here**](https://docs.opencv.org/master/d4/ddd/gapi_impl.html#gapi_detail_params)), so API layer captures its values and stores internally upon the moment of execution.

Finally, [**cv::GComputation**](https://docs.opencv.org/master/d9/dfe/classcv_1_1GComputation.html) and [**cv::GCompiled**](https://docs.opencv.org/master/d2/d2c/classcv_1_1GCompiled.html) are the remaining important components of API layer. The former wraps a series of G-API expressions into an object (graph), and the latter is a product of graph *compilation* (see [**this chapter**](https://docs.opencv.org/master/d4/ddd/gapi_impl.html#gapi_detail_compiler) for details).

**Graph compiler layer**

Every G-API computation is compiled before it executes. Compilation process is triggered in two ways:

* *implicitly*, when [**cv::GComputation::apply()**](https://docs.opencv.org/master/d9/dfe/classcv_1_1GComputation.html#aa25bc73f7cba9bb2f9992edaa248f868) is used. In this case, graph compilation is then immediately followed by execution.
* *explicitly*, when [**cv::GComputation::compile()**](https://docs.opencv.org/master/d9/dfe/classcv_1_1GComputation.html#ab11bcc0578f9aa5ec8f8eaf6cda0dfa8) is used. In this case, a [**cv::GCompiled**](https://docs.opencv.org/master/d2/d2c/classcv_1_1GCompiled.html) object is returned which then can be invoked as a C++ functor.

The first way is recommended for cases when input data format is not known in advance – e.g. when it comes from an arbitrary input file. The second way is recommended for deployment (production) scenarios where input data characteristics are usually predefined.

Graph compilation process is built atop of ADE Framework. Initially, a bipartite graph is generated from expressions captured by API layer. This graph contains nodes of two types: *Data* and *Operations*. Graph always starts and ends with a Data node(s), with Operations nodes in-between. Every Operation node has inputs and outputs, both are Data nodes.

After the initial graph is generated, it is actually processed by a number of graph transformations, called *passes*. ADE Framework acts as a compiler pass management engine, and passes are written specifically for G-API.

There are different passes which check graph validity, refine details on operations and data, organize nodes into clusters ("Islands") based on affinity or user-specified regioning[TBD], and more. Backends also are able to inject backend-specific passes into the compilation process, see more on this in the [**dedicated chapter**](https://docs.opencv.org/master/d4/ddd/gapi_impl.html#gapi_detail_meta).

Result of graph compilation is a compiled object, represented by class [**cv::GCompiled**](https://docs.opencv.org/master/d2/d2c/classcv_1_1GCompiled.html). A new [**cv::GCompiled**](https://docs.opencv.org/master/d2/d2c/classcv_1_1GCompiled.html) object is always created regardless if there was an explicit or implicit compilation request (see above). Actual graph execution happens within [**cv::GCompiled**](https://docs.opencv.org/master/d2/d2c/classcv_1_1GCompiled.html) and is determined by backends which participated in the graph compilation.

**See also**

[**cv::GComputation::apply()**](https://docs.opencv.org/master/d9/dfe/classcv_1_1GComputation.html#aa25bc73f7cba9bb2f9992edaa248f868), [**cv::GComputation::compile()**](https://docs.opencv.org/master/d9/dfe/classcv_1_1GComputation.html#ab11bcc0578f9aa5ec8f8eaf6cda0dfa8), [**cv::GCompiled**](https://docs.opencv.org/master/d2/d2c/classcv_1_1GCompiled.html)

**3.5.4 Backends layer**

The above diagram lists two backends, *OpenCV* and *Fluid*. *OpenCV* is so-called "reference backend", which implements G-API operations using plain old OpenCV functions. This backend is useful for prototyping on a familiar development system. *Fluid* is a plugin for cache-efficient execution on CPU – it implements a different execution policy and operates with its own, special kernels. Fluid backend allows to achieve less memory footprint and better memory locality when running on CPU.

There may be more backends available, e.g. Halide, OpenCL, etc. – G-API provides an uniform internal API to develop backends so any enthusiast or a company are free to scale G-API on a new platform or accelerator. In terms of OpenCV infrastructure, every new backend is a new distinct OpenCV module, which extends G-API when build as a part of OpenCV.

**3.5.5 Graph execution**

The way graph executed is defined by backends selected for compilation. In fact, every backend builds its own execution script as the final stage of graph compilation process, when an executable (compiled) object is being generated. For example, in OpenCV backend, this script is just a topologically-sorted sequence of OpenCV functions to call; for Fluid backend, it is a similar thing – a topologically sorted list of *Agents* processing lines of input on every iteration.

Graph execution is triggered in two ways:

* via [**cv::GComputation::apply()**](https://docs.opencv.org/master/d9/dfe/classcv_1_1GComputation.html#aa25bc73f7cba9bb2f9992edaa248f868), with graph compiled in-place exactly for the given input data;
* via [**cv::GCompiled::operator()()**](https://docs.opencv.org/master/d2/d2c/classcv_1_1GCompiled.html#ad2ed48a66be94fae14bc3a2a08ea8e95), when the graph has been precompiled.

Both methods are polimorphic and take a variadic number of arguments, with validity checks performed in runtime. If a number, shapes, and formats of passed data objects differ from expected, a run-time exception is thrown. G-API also provides *typed* wrappers to move these checks to the compile time – see [**cv::GComputationT**](https://docs.opencv.org/master/dc/d07/classcv_1_1GComputationT.html)<>.

G-API graph execution is declared stateless – it means that a compiled functor ([**cv::GCompiled**](https://docs.opencv.org/master/d2/d2c/classcv_1_1GCompiled.html)) acts like a pure C++ function and provides the same result for the same set of input arguments.

**3.6 Public Expectations**

The public Expectations will be monitored by a feedback process, maybe online or having a box nearby at signals. After the feedback we will convince them and will tell them how to follow and how to be aware through the e-challan system

**3.7 Risk Assessment**

By observing all the things that will be zero risk assessment on minimum risk assessment in the e-challan system, but the awareness should be knownt to the people for their own benefit.

**Chapter 4: Analysis and Observations**

**4.1 Examination and Estimation**

**4.1.1 Image Segmentation with Watershed Algorithm**

**Goal**

**In this chapter,**

* We will learn to use marker-based image segmentation using watershed algorithm
* We will see: **cv2.watershed()**

**Theory**

Any grayscale image can be viewed as a topographic surface where high intensity denotes peaks and hills while low intensity denotes valleys. You start filling every isolated valleys (local minima) with different colored water (labels). As the water rises, depending on the peaks (gradients) nearby, water from different valleys, obviously with different colors will start to merge. To avoid that, you build barriers in the locations where water merges. You continue the work of filling water and building barriers until all the peaks are under water. Then the barriers you created gives you the segmentation result. This is the “philosophy” behind the watershed. You can visit the [CMM webpage on watershed](http://cmm.ensmp.fr/~beucher/wtshed.html) to understand it with the help of some animations.

But this approach gives you oversegmented result due to noise or any other irregularities in the image. So OpenCV implemented a marker-based watershed algorithm where you specify which are all valley points are to be merged and which are not. It is an interactive image segmentation. What we do is to give different labels for our object we know. Label the region which we are sure of being the foreground or object with one color (or intensity), label the region which we are sure of being background or non-object with another color and finally the region which we are not sure of anything, label it with 0. That is our marker. Then apply watershed algorithm. Then our marker will be updated with the labels we gave, and the boundaries of objects will have a value of -1.

**Code**

Below we will see an example on how to use the Distance Transform along with watershed to segment mutually touching objects.

Consider the coins image below, the coins are touching each other. Even if you threshold it, it will be touching each other.



We start with finding an approximate estimate of the coins. For that, we can use the Otsu’s binarization.

import numpy as np

import cv2

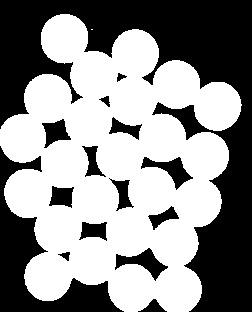
from matplotlib import pyplot as plt

img = cv2.imread('coins.png')

gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

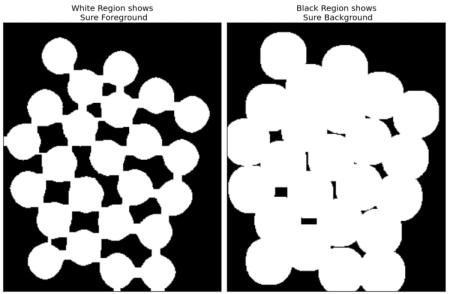
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH\_BINARY\_INV+cv2.THRESH\_OTSU)

Result:



Now we need to remove any small white noises in the image. For that we can use morphological opening. To remove any small holes in the object, we can use morphological closing. So, now we know for sure that region near to center of objects are foreground and region much away from the object are background. Only region we are not sure is the boundary region of coins.

So we need to extract the area which we are sure they are coins. Erosion removes the boundary pixels. So whatever remaining, we can be sure it is coin. That would work if objects were not touching each other. But since they are touching each other, another good option would be to find the distance transform and apply a proper threshold. Next we need to find the area which we are sure they are not coins. For that, we dilate the result. Dilation increases object boundary to background. This way, we can make sure whatever region in background in result is really a background, since boundary region is removed. See the image below.



The remaining regions are those which we don’t have any idea, whether it is coins or background. Watershed algorithm should find it. These areas are normally around the boundaries of coins where foreground and background meet (Or even two different coins meet). We call it border. It can be obtained from subtracting sure\_fg area from sure\_bg area.

# noise removal

kernel = np.ones((3,3),np.uint8)

opening = cv2.morphologyEx(thresh,cv2.MORPH\_OPEN,kernel, iterations = 2)

# sure background area

sure\_bg = cv2.dilate(opening,kernel,iterations=3)

# Finding sure foreground area

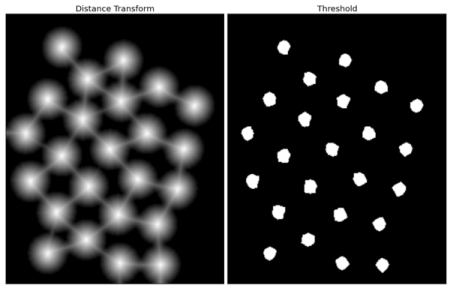
dist\_transform = cv2.distanceTransform(opening,cv2.DIST\_L2,5)ret, sure\_fg = cv2.threshold(dist\_transform,0.7\*dist\_transform.max(),255,0)

# Finding unknown region

sure\_fg = np.uint8(sure\_fg)

unknown = cv2.subtract(sure\_bg,sure\_fg)

See the result. In the thresholded image, we get some regions of coins which we are sure of coins and they are detached now. (In some cases, you may be interested in only foreground segmentation, not in separating the mutually touching objects. In that case, you need not use distance transform, just erosion is sufficient. Erosion is just another method to extract sure foreground area, that’s all.)



Now we know for sure which are region of coins, which are background and all. So we create marker (it is an array of same size as that of original image, but with int32 datatype) and label the regions inside it. The regions we know for sure (whether foreground or background) are labelled with any positive integers, but different integers, and the area we don’t know for sure are just left as zero. For this we use **cv2.connectedComponents()**. It labels background of the image with 0, then other objects are labelled with integers starting from 1.

But we know that if background is marked with 0, watershed will consider it as unknown area. So we want to mark it with different integer. Instead, we will mark unknown region, defined by unknown, with 0.

# Marker labelling

ret, markers = cv2.connectedComponents(sure\_fg)

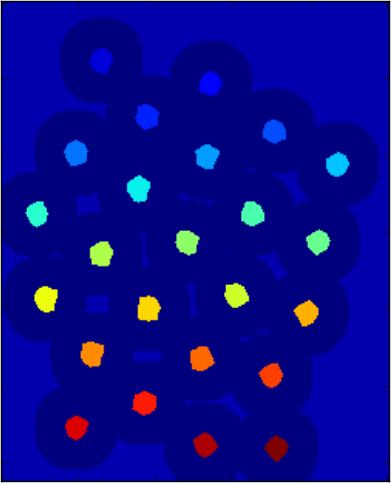
# Add one to all labels so that sure background is not 0, but 1

markers = markers+1

# Now, mark the region of unknown with zero

markers[unknown==255] = 0

See the result shown in JET colormap. The dark blue region shows unknown region. Sure coins are colored with different values. Remaining area which are sure background are shown in lighter blue compared to unknown region.

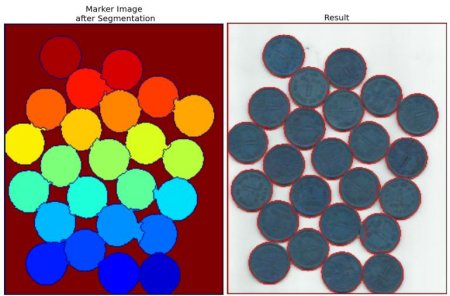


Now our marker is ready. It is time for final step, apply watershed. Then marker image will be modified. The boundary region will be marked with -1.

markers = cv2.watershed(img,markers)

img[markers == -1] = [255,0,0]

See the result below. For some coins, the region where they touch are segmented properly and for some, they are not.



**4.2 Background Capturing Analysis**

**Interactive Foreground Extraction using GrabCut Algorithm**

**Goal**

**In this**

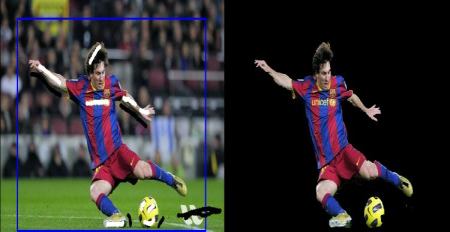
* We will see GrabCut algorithm to extract foreground in images
* We will create an interactive application for this.

**Theory**

GrabCut algorithm was designed by Carsten Rother, Vladimir Kolmogorov & Andrew Blake from Microsoft Research Cambridge, UK. in their paper, [“GrabCut”: interactive foreground extraction using iterated graph cuts](http://dl.acm.org/citation.cfm?id=1015720) . An algorithm was needed for foreground extraction with minimal user interaction, and the result was GrabCut.

How it works from user point of view ? Initially user draws a rectangle around the foreground region (foreground region shoule be completely inside the rectangle). Then algorithm segments it iteratively to get the best result. Done. But in some cases, the segmentation won’t be fine, like, it may have marked some foreground region as background and vice versa. In that case, user need to do fine touch-ups. Just give some strokes on the images where some faulty results are there. Strokes basically says *“Hey, this region should be foreground, you marked it background, correct it in next iteration”* or its opposite for background. Then in the next iteration, you get better results.

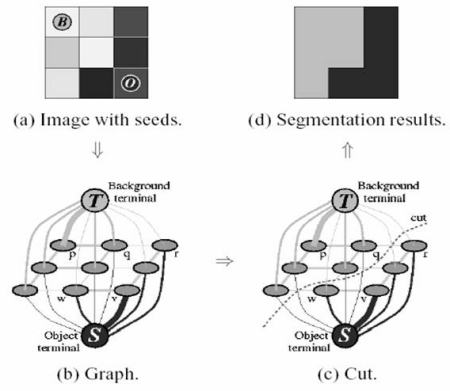
See the image below. First player and football is enclosed in a blue rectangle. Then some final touchups with white strokes (denoting foreground) and black strokes (denoting background) is made. And we get a nice result.



So what happens in background ?

* User inputs the rectangle. Everything outside this rectangle will be taken as sure background (That is the reason it is mentioned before that your rectangle should include all the objects). Everything inside rectangle is unknown. Similarly any user input specifying foreground and background are considered as hard-labelling which means they won’t change in the process.
* Computer does an initial labelling depeding on the data we gave. It labels the foreground and background pixels (or it hard-labels)
* Now a Gaussian Mixture Model(GMM) is used to model the foreground and background.
* Depending on the data we gave, GMM learns and create new pixel distribution. That is, the unknown pixels are labelled either probable foreground or probable background depending on its relation with the other hard-labelled pixels in terms of color statistics (It is just like clustering).
* A graph is built from this pixel distribution. Nodes in the graphs are pixels. Additional two nodes are added, **Source node** and **Sink node**. Every foreground pixel is connected to Source node and every background pixel is connected to Sink node.
* The weights of edges connecting pixels to source node/end node are defined by the probability of a pixel being foreground/background. The weights between the pixels are defined by the edge information or pixel similarity. If there is a large difference in pixel color, the edge between them will get a low weight.
* Then a mincut algorithm is used to segment the graph. It cuts the graph into two separating source node and sink node with minimum cost function. The cost function is the sum of all weights of the edges that are cut. After the cut, all the pixels connected to Source node become foreground and those connected to Sink node become background.
* The process is continued until the classification converges.

It is illustrated in below image (Image Courtesy: <http://www.cs.ru.ac.za/research/g02m1682/>)



**Demo**

Now we go for grabcut algorithm with OpenCV. OpenCV has the function, **cv2.grabCut()** for this. We will see its arguments first:

* *img* - Input image
* *mask* - It is a mask image where we specify which areas are background, foreground or probable background/foreground etc. It is done by the following flags, **cv2.GC\_BGD, cv2.GC\_FGD, cv2.GC\_PR\_BGD, cv2.GC\_PR\_FGD**, or simply pass 0,1,2,3 to image.
* *rect* - It is the coordinates of a rectangle which includes the foreground object in the format (x,y,w,h)
* *bdgModel*, *fgdModel* - These are arrays used by the algorithm internally. You just create two np.float64 type zero arrays of size (1,65).
* *iterCount* - Number of iterations the algorithm should run.
* *mode* - It should be **cv2.GC\_INIT\_WITH\_RECT** or **cv2.GC\_INIT\_WITH\_MASK** or combined which decides whether we are drawing rectangle or final touchup strokes.

First let’s see with rectangular mode. We load the image, create a similar mask image. We create *fgdModel* and *bgdModel*. We give the rectangle parameters. It’s all straight-forward. Let the algorithm run for 5 iterations. Mode should be *cv2.GC\_INIT\_WITH\_RECT* since we are using rectangle. Then run the grabcut. It modifies the mask image. In the new mask image, pixels will be marked with four flags denoting background/foreground as specified above. So we modify the mask such that all 0-pixels and 2-pixels are put to 0 (ie background) and all 1-pixels and 3-pixels are put to 1(ie foreground pixels). Now our final mask is ready. Just multiply it with input image to get the segmented image.

import numpy as np

import cv2

from matplotlib import pyplot as plt

img = cv2.imread('messi5.jpg')

mask = np.zeros(img.shape[:2],np.uint8)

bgdModel = np.zeros((1,65),np.float64)

fgdModel = np.zeros((1,65),np.float64)

rect = (50,50,450,290)

cv2.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv2.GC\_INIT\_WITH\_RECT)

mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')

img = img\*mask2[:,:,np.newaxis]

plt.imshow(img),plt.colorbar(),plt.show()

See the results below:



Oops, Messi’s hair is gone. *Who likes Messi without his hair?* We need to bring it back. So we will give there a fine touchup with 1-pixel (sure foreground). At the same time, Some part of ground has come to picture which we don’t want, and also some logo. We need to remove them. There we give some 0-pixel touchup (sure background). So we modify our resulting mask in previous case as we told now.

*What I actually did is that, I opened input image in paint application and added another layer to the image. Using brush tool in the paint, I marked missed foreground (hair, shoes, ball etc) with white and unwanted background (like logo, ground etc) with black on this new layer. Then filled remaining background with gray. Then loaded that mask image in OpenCV, edited original mask image we got with corresponding values in newly added mask image. Check the code below:*

# newmask is the mask image I manually labelled

newmask = cv2.imread('newmask.png',0)

# whereever it is marked white (sure foreground), change mask=1

# whereever it is marked black (sure background), change mask=0

mask[newmask == 0] = 0

mask[newmask == 255] = 1

mask, bgdModel, fgdModel = cv2.grabCut(img,mask,None,bgdModel,fgdModel,5,cv2.GC\_INIT\_WITH\_MASK)

mask = np.where((mask==2)|(mask==0),0,1).astype('uint8')

img = img\*mask[:,:,np.newaxis]

plt.imshow(img),plt.colorbar(),plt.show()

See the result below:



So that’s it. Here instead of initializing in rect mode, you can directly go into mask mode. Just mark the rectangle area in mask image with 2-pixel or 3-pixel (probable background/foreground). Then mark our sure\_foreground with 1-pixel as we did in second example. Then directly apply the grabCut function with mask mode.

**4.3 Formulation Of Color Detection Methodology**

4.3.1 OpenCV and Python Color Detection

Let’s go ahead and get this started.

Open up your favorite editor and create a file named

detect\_color.py

1. :
2. OpenCV and Python Color Detection
3. # import the necessary packages
4. import numpy as np
5. import argparse
6. import cv2
7. # construct the argument parse and parse the arguments
8. ap = argparse.ArgumentParser()
9. ap.add\_argument("-i", "--image", help = "path to the image")
10. args = vars(ap.parse\_args())
11. # load the image
12. image = cv2.imread(args["image"])

We’ll start by importing our necessary packages on **Lines 2-4**. We’ll use NumPy for numerical processing,

argparse

  to parse our command line arguments, and

cv2

  for our OpenCV bindings.

**Lines 7-9** then handle parsing our command line arguments. We’ll need just a single switch,

--image

 , which is the path to where our image resides on disk.

Then, on **Line 12**, we load our image off disk.

Now, here comes the interesting part.

We want to be able to detect each of the Game Boy cartridges in the image. That means we’ll have to recognize **red**, **blue**,**yellow**, and **gray** colors in the image.

Let’s go ahead and define this list of colors:

1. OpenCV and Python Color Detection
2. # define the list of boundaries
3. boundaries = [
4. ([17, 15, 100], [50, 56, 200]),
5. ([86, 31, 4], [220, 88, 50]),
6. ([25, 146, 190], [62, 174, 250]),
7. ([103, 86, 65], [145, 133, 128])
8. ]

**4.3.2 List & Boundaries**

 in the RGB color space (or rather, BGR, since OpenCV represents images as NumPy arrays in reverse order), where each entry in the list is a tuple with two values: a list of lower limits and a list of upper limits.

For example, let’s take a look at the tuple

([17, 15, 100], [50, 56, 200])

 .

Here, we are saying that all pixels in our image that have a R >= 100, B >= 15, and G >= 17 along with R <= 200, B <= 56, and G <= 50 will be considered **red**.

Now that we have our list of boundaries, we can use the

cv2.inRange

 function to perform the actual color detection.

Let’s take a look:

1. OpenCV and Python Color Detection
2. # loop over the boundaries
3. for (lower, upper) in boundaries:
4. # create NumPy arrays from the boundaries
5. lower = np.array(lower, dtype = "uint8")
6. upper = np.array(upper, dtype = "uint8")
7. # find the colors within the specified boundaries and apply
8. # the mask
9. mask = cv2.inRange(image, lower, upper)
10. output = cv2.bitwise\_and(image, image, mask = mask)
11. # show the images
12. cv2.imshow("images", np.hstack([image, output]))
13. cv2.waitKey(0)

**4.3.3 We start looping over our upper and lower**

Boundaries

 on **Line 23**, then convert the upper and lower limits to NumPy arrays on **Line 25 and 26**. These two lines seem like they can be omitted, but when you are working with OpenCV Python bindings, OpenCV expects these limits to be NumPy arrays. Furthermore, since these are pixel values that fall within the range [0, 256] we can use the unsigned 8-bit integer data type.

To perform the actual color detection using OpenCV, take a look at **Line 29** where we use the

cv2.inRange

 function.

The

cv2.inRange

  function expects three arguments: the first is the

image

  were we are going to perform color detection, the second is the

lower

  limit of the color you want to detect, and the third argument is the

upper

  limit of the color you want to detect.

After calling

cv2.inRange

, a binary mask is returned, where white pixels (255) represent pixels that fall into the upper and lower limit range and black pixels (0) do not.

Note: We are performing color detection in the RGB color space. But you can easily do this in the HSV or L\*a\*b\* color space as well. You would simply need to adjust your upper and lower limits to the respective color space.

To create the output image, we apply our mask on **Line 31**. This line simply makes a call to

cv2.bitwise\_and

, showing only pixels in the

image

 that have a corresponding white (255) value in the

mask

.

Finally, our output images are displayed on **Lines 34 and 35**.

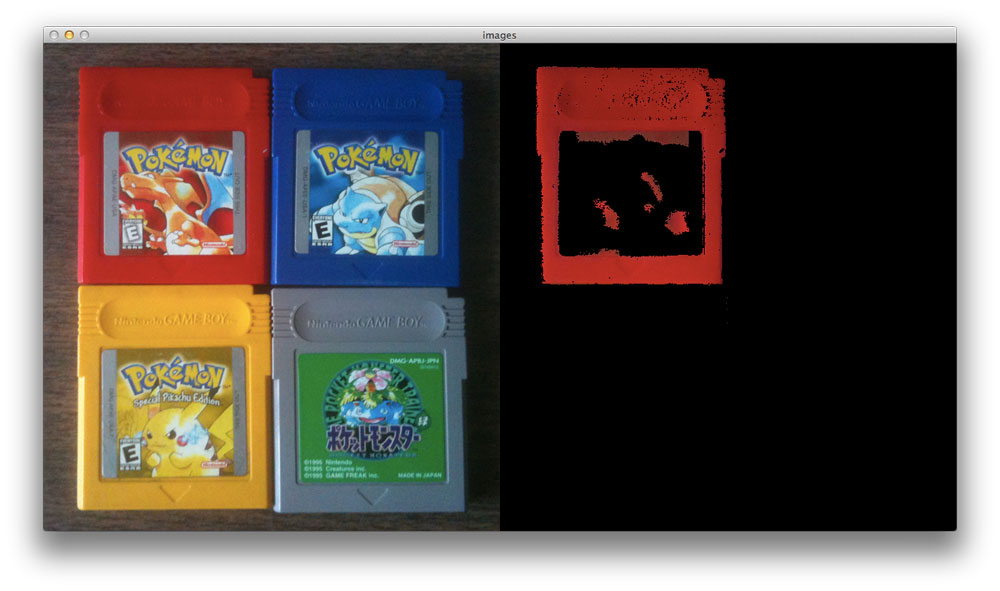
Not bad. Only 35 lines of code, and the vast majority is imports, argument parsing, and comments.

Let’s go ahead and run our script:

**4.3.4 OpenCV and Python Color Detection**

$ python detect\_color.py --image pokemon\_games.png

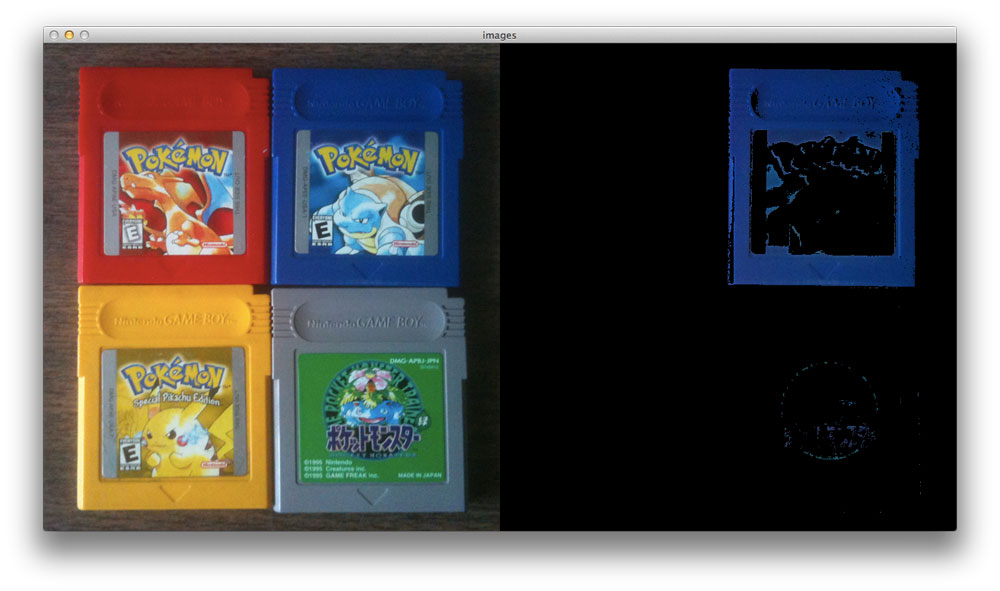
If your environment is configured correctly (meaning you have OpenCV with Python bindings installed), you should see this as your output image:

**[](https://www.pyimagesearch.com/wp-content/uploads/2014/07/color_detection_red_version.jpg)**

**Figure 1:** Detecting the color red in an image using OpenCV and Python.

As you can see, the Red Pokemon cartridge is easily detected!

Now let’s try the blue one:

**[](https://www.pyimagesearch.com/wp-content/uploads/2014/07/color_detection_blue_version.jpg)**

**Figure 2:** Detecting the color blue in an image using OpenCV and Python.

And a similar story for Yellow version:

**[](https://www.pyimagesearch.com/wp-content/uploads/2014/07/color_detection_yellow_version.jpg)**

**Figure 3:** Detecting the color yellow in an image using OpenCV and Python.

Lastly, the outline of the gray Game Boy cartridge is also found:

**[](https://www.pyimagesearch.com/wp-content/uploads/2014/07/color_detection_green_version.jpg)Figure 4:** Detecting gray in an image using OpenCV and Python.

**Summary**

To detect colors in images, the first thing you need to do is define the upper and lower limits for your pixel values.

Once you have defined your upper and lower limits, you then make a call to the

cv2.inRange

 method which returns a mask, specifying which pixels fall into your specified upper and lower range.

Finally, now that you have the mask, you can apply it to your image using the

cv2.bitwise\_and

 function.

**4.4 Object’s Motion Detection Algorithm**

|  |  |
| --- | --- |
| **4.4.1 Classes** | |
| class | [**cv::BackgroundSubtractor**](https://docs.opencv.org/master/d7/df6/classcv_1_1BackgroundSubtractor.html) |
|  | Base class for background/foreground segmentation. : [More...](https://docs.opencv.org/master/d7/df6/classcv_1_1BackgroundSubtractor.html#details) |
|  | |
| class | [**cv::BackgroundSubtractorKNN**](https://docs.opencv.org/master/db/d88/classcv_1_1BackgroundSubtractorKNN.html) |
|  | K-nearest neighbours - based Background/Foreground Segmentation [**Algorithm**](https://docs.opencv.org/master/d3/d46/classcv_1_1Algorithm.html). [More...](https://docs.opencv.org/master/db/d88/classcv_1_1BackgroundSubtractorKNN.html#details) |
|  | |
| class | [**cv::BackgroundSubtractorMOG2**](https://docs.opencv.org/master/d7/d7b/classcv_1_1BackgroundSubtractorMOG2.html) |
|  | Gaussian Mixture-based Background/Foreground Segmentation [**Algorithm**](https://docs.opencv.org/master/d3/d46/classcv_1_1Algorithm.html). [More...](https://docs.opencv.org/master/d7/d7b/classcv_1_1BackgroundSubtractorMOG2.html#details) |
|  | |

|  |  |
| --- | --- |
| Functions | |
| [**Ptr**](https://docs.opencv.org/master/dc/d84/group__core__basic.html#ga6395ca871a678020c4a31fadf7e8cc63)< **[BackgroundSubtractorKNN](https://docs.opencv.org/master/db/d88/classcv_1_1BackgroundSubtractorKNN.html)** > | [**cv::createBackgroundSubtractorKNN**](https://docs.opencv.org/master/de/de1/group__video__motion.html#gac9be925771f805b6fdb614ec2292006d) (int history=500, double dist2Threshold=400.0, bool detectShadows=true) |
|  | Creates KNN Background Subtractor. [More...](https://docs.opencv.org/master/de/de1/group__video__motion.html#gac9be925771f805b6fdb614ec2292006d) |
|  | |
| [**Ptr**](https://docs.opencv.org/master/dc/d84/group__core__basic.html#ga6395ca871a678020c4a31fadf7e8cc63)< [**BackgroundSubtractorMOG2**](https://docs.opencv.org/master/d7/d7b/classcv_1_1BackgroundSubtractorMOG2.html) > | [**cv::createBackgroundSubtractorMOG2**](https://docs.opencv.org/master/de/de1/group__video__motion.html#ga2beb2dee7a073809ccec60f145b6b29c) (int history=500, double varThreshold=16, bool detectShadows=true) |
|  | Creates MOG2 Background Subtractor. [More...](https://docs.opencv.org/master/de/de1/group__video__motion.html#ga2beb2dee7a073809ccec60f145b6b29c) |
|  | |

Detailed Description

Function Documentation

[◆](https://docs.opencv.org/master/de/de1/group__video__motion.html#gac9be925771f805b6fdb614ec2292006d)createBackgroundSubtractorKNN()

|  |  |  |  |
| --- | --- | --- | --- |
| [**Ptr**](https://docs.opencv.org/master/dc/d84/group__core__basic.html#ga6395ca871a678020c4a31fadf7e8cc63)<**[BackgroundSubtractorKNN](https://docs.opencv.org/master/db/d88/classcv_1_1BackgroundSubtractorKNN.html)**> cv::createBackgroundSubtractorKNN | ( | int | history = 500, |
|  |  | double | dist2Threshold = 400.0, |
|  |  | bool | detectShadows = true |
|  | ) |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **4.4.2 Python:** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | retval | = | cv.createBackgroundSubtractorKNN( | [, history[, dist2Threshold[, detectShadows]]] | ) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

#include <[**opencv2/video/background\_segm.hpp**](https://docs.opencv.org/master/db/d73/background__segm_8hpp.html)>

Creates KNN Background Subtractor.

**Parameters**

|  |  |
| --- | --- |
| **history** | Length of the history. |
| **dist2Threshold** | Threshold on the squared distance between the pixel and the sample to decide whether a pixel is close to that sample. This parameter does not affect the background update. |
| **detectShadows** | If true, the algorithm will detect shadows and mark them. It decreases the speed a bit, so if you do not need this feature, set the parameter to false. |

[◆](https://docs.opencv.org/master/de/de1/group__video__motion.html#ga2beb2dee7a073809ccec60f145b6b29c)createBackgroundSubtractorMOG2()

|  |  |  |  |
| --- | --- | --- | --- |
| [**Ptr**](https://docs.opencv.org/master/dc/d84/group__core__basic.html#ga6395ca871a678020c4a31fadf7e8cc63)<[**BackgroundSubtractorMOG2**](https://docs.opencv.org/master/d7/d7b/classcv_1_1BackgroundSubtractorMOG2.html)> cv::createBackgroundSubtractorMOG2 | ( | int | history = 500, |
|  |  | double | varThreshold = 16, |
|  |  | bool | detectShadows = true |
|  | ) |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **4.4.3 Python:** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | retval | = | cv.createBackgroundSubtractorMOG2( | [, history[, varThreshold[, detectShadows]]] | ) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

#include <[**opencv2/video/background\_segm.hpp**](https://docs.opencv.org/master/db/d73/background__segm_8hpp.html)>

Creates MOG2 Background Subtractor.

**Parameters**

|  |  |
| --- | --- |
| **history** | Length of the history. |
| **varThreshold** | Threshold on the squared Mahalanobis distance between the pixel and the model to decide whether a pixel is well described by the background model. This parameter does not affect the background update. |
| **detectShadows** | If true, the algorithm will detect shadows and mark them. It decreases the speed a bit, so if you do not need this feature, set the parameter to false. |

**Examples:**

[**samples/cpp/segment\_objects.cpp**](https://docs.opencv.org/master/d5/de8/samples_2cpp_2segment_objects_8cpp-example.html#a23).

**4.5 Tesseract Library To convert Image To Text**

Using Tesseract OCR with Python

This blog post is divided into three parts.

First, we’ll learn how to install the [**pytesseract package**](https://github.com/madmaze/pytesseract) so that we can access Tesseract via the Python programming language.

Next, we’ll develop a simple Python script to load an image, binarize it, and pass it through the Tesseract OCR system.

Finally, we’ll test our OCR pipeline on some example images and review the results.

To download the source code + example images to this blog post, be sure to use the ***“Downloads”*** section below.

### Installing the Tesseract + Python “bindings”

Let’s begin by getting

pytesseract

  installed. To install

pytesseract

  we’ll take advantage of

pip

 .

If you’re using a virtual environment (which I highly recommend so that you can separate different projects), use the

workon

  command followed by the appropriate virtual environment name. In this case, our virtualenv is named

cv

 .

Using Tesseract OCR with Python

$ workon cv

Next let’s install [**Pillow**](https://python-pillow.org/), a more Python-friendly port of PIL (a dependency) followed by

pytesseract

 .

Using Tesseract OCR with Python

$ pip install pillow

$ pip install pytesseract

***Note***:

*pytesseract*

  does not provide true Python bindings. Rather, it simply provides an interface to the

*tesseract*

  binary. If you [***take a look at the project on GitHub***](https://github.com/madmaze/pytesseract/blob/master/src/pytesseract.py) you’ll see that the library is writing the image to a temporary file on disk followed by calling the

*tesseract*

  binary on the file and capturing the resulting output. This is definitely a bit hackish, but it gets the job done for us.

Let’s move forward by reviewing some code that segments the foreground text from the background and then makes use of our freshly installed

pytesseract

 .

### **4.5.1 Applying OCR with Tesseract and Python**

Let’s begin by creating a new file named

1. ocr.py
2. :
3. Using Tesseract OCR with Python
4. # import the necessary packages
5. from PIL import Image
6. import pytesseract
7. import argparse
8. import cv2
9. import os
10. # construct the argument parse and parse the arguments
11. ap = argparse.ArgumentParser()
12. ap.add\_argument("-i", "--image", required=True,
13. help="path to input image to be OCR'd")
14. ap.add\_argument("-p", "--preprocess", type=str, default="thresh",
15. help="type of preprocessing to be done")
16. args = vars(ap.parse\_args())

**Lines 2-6** handle our imports. The

Image

  class is required so that we can load our input image from disk in PIL format, a requirement when using

pytesseract

 .

Our command line arguments are parsed on **Lines 9-14**. We have two command line arguments:

* --image

 : The path to the image we’re sending through the OCR system.

* --preprocess

 : The preprocessing method. This switch is optional and for this tutorial and can accept two values:

thresh

  (threshold) or

blur

 .

Next we’ll load the image, binarize it, and write it to disk.

**4.5.2 Using Tesseract OCR with Python**

1. # load the example image and convert it to grayscale
2. image = cv2.imread(args["image"])
3. gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)
4. # check to see if we should apply thresholding to preprocess the
5. # image
6. if args["preprocess"] == "thresh":
7. gray = cv2.threshold(gray, 0, 255,
8. cv2.THRESH\_BINARY | cv2.THRESH\_OTSU)[1]
9. # make a check to see if median blurring should be done to remove
10. # noise
11. elif args["preprocess"] == "blur":
12. gray = cv2.medianBlur(gray, 3)
13. # write the grayscale image to disk as a temporary file so we can
14. # apply OCR to it
15. filename = "{}.png".format(os.getpid())
16. cv2.imwrite(filename, gray)

First, we load

--image

  from disk into memory (**Line 17**) followed by converting it to grayscale (**Line 18**).

Next, depending on the pre-processing method specified by our command line argument, we will either threshold or blur the image. This is where you would want to add more advanced pre-processing methods (depending on your specific application of OCR) which are beyond the scope of this blog post.

The

if

  statement and body on **Lines 22-24**perform a threshold in order to segment the foreground from the background. We do this using both

cv2.THRESH\_BINARY

  and

cv2.THRESH\_OTSU

  flags. For details on Otsu’s method, see “Otsu’s Binarization” in the [**official OpenCV documentation**](http://docs.opencv.org/trunk/d7/d4d/tutorial_py_thresholding.html).

We will see later in the results section that this thresholding method can be useful to read dark text that is overlaid upon gray shapes.

Alternatively, a blurring method may be applied. **Lines 28-29** perform a median blur when the

--preprocess

  flag is set to

blur

 . Applying a median blur can help reduce salt and pepper noise, again making it easier for Tesseract to correctly OCR the image.

After pre-processing the image, we use

os.getpid

  to derive a temporary image

filename

 based on the process ID of our Python script (**Line 33**).

The final step before using

pytesseract

 for OCR is to write the pre-processed image,

gray

 , to disk saving it with the

filename

  from above (**Line 34**).

We can finally apply OCR to our image using the Tesseract Python “bindings”:

**4.5.3 Using Tesseract OCR with Python**

1. # load the image as a PIL/Pillow image, apply OCR, and then delete
2. # the temporary file
3. text = pytesseract.image\_to\_string(Image.open(filename))
4. os.remove(filename)
5. print(text)
6. # show the output images
7. cv2.imshow("Image", image)
8. cv2.imshow("Output", gray)
9. cv2.waitKey(0)

Using

pytesseract.image\_to\_string

  on **Line 38** we convert the contents of the image into our desired string,

text

 . Notice that we passed a reference to the temporary image file residing on disk.

This is followed by some cleanup on **Line 39** where we delete the temporary file.

**Line 40** is where we print text to the terminal. In your own applications, you may wish to do some additional processing here such as spellchecking for OCR errors or Natural Language Processing rather than simply printing it to the console as we’ve done in this tutorial.

Finally, **Lines 43 and 44** handle displaying the original image and pre-processed image on the screen in separate windows. The

cv2.waitKey(0)

  on **Line 34** indicates that we should wait until a key on the keyboard is pressed before exiting the script.

Let’s see our handywork in action.

### Tesseract OCR and Python results

Now that

ocr.py

  has been created, it’s time to apply Python + Tesseract to perform OCR on some example input images.

In this section we will try OCR’ing three sample images using the following process:

* First, we will run each image through the Tesseract binary as-is.
* Then we will run each image through

ocr.py

  (which performs pre-processing before sending through Tesseract).

* Finally, we will compare the results of both of these methods and note any errors.

Our first example is a “noisy” image. This image contains our desired foreground black text on a background that is partly white and partly scattered with artificially generated circular blobs. The blobs act as “distractors” to our simple algorithm.

**Figure 1:** Our first example input for Optical Character Recognition using Python.

Using the Tesseract binary, [**as we learned last week**](https://www.pyimagesearch.com/2017/07/03/installing-tesseract-for-ocr/), we can apply OCR to the raw, unprocessed image:

Using Tesseract OCR with Python

1. $ tesseract images/example\_01.png stdout
2. Noisy image
3. to test
4. Tesseract OCR

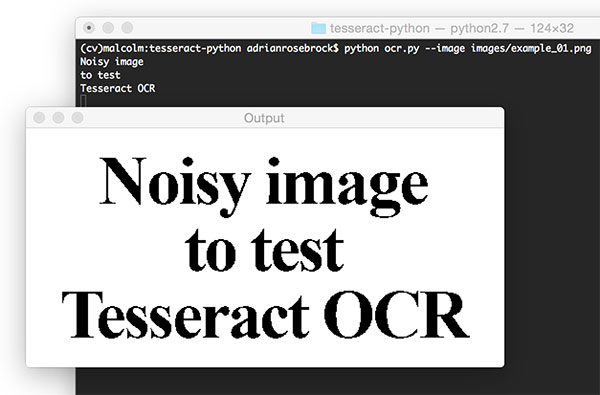
Tesseract performed well with no errors in this case.

Now let’s confirm that our newly made script,

ocr.py

 , also works:

1. Using Tesseract OCR with Python
2. $ python ocr.py --image images/example\_01.png
3. Noisy image
4. to test
5. Tesseract OCR

**Figure 2:** Applying image preprocessing for OCR with Python.

As you can see in this screenshot, the thresholded image is very clear and the background has been removed. Our script correctly prints the contents of the image to the console.

Next, let’s test Tesseract and our pre-processing script on an image with “salt and pepper” noise in the background:

**Figure 3:** An example input image containing noise. This image will “confuse” our OCR algorithm, leading to incorrect OCR results.

We can see the output of the

tesseract

  binary below:

Using Tesseract OCR with Python

$ tesseract images/example\_02.png stdout

Detected 32 diacritics

" Tesséra‘c't Will

Fail With Noisy

Backgrounds

Unfortunately, Tesseract did not successfully OCR the text in the image.

However, by using the

blur

  pre-processing method in

ocr.py

  we can obtain better results:

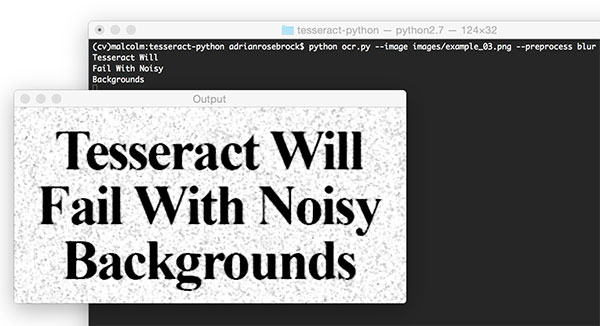
Using Tesseract OCR with Python

$ python ocr.py --image images/example\_02.png --preprocess blur

Tesseract Will

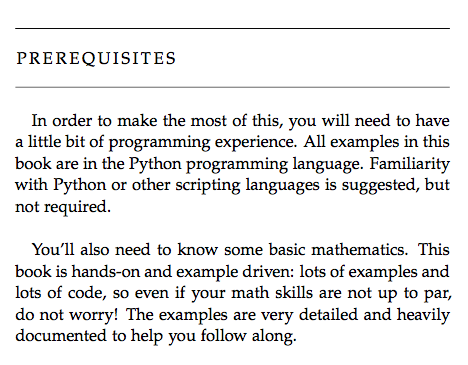
Fail With Noisy

Backgrounds

**Figure 4:** Applying image preprocessing with Python and OpenCV to improve OCR results.

Success! Our blur pre-processing step enabled Tesseract to correctly OCR and output our desired text.

Finally, let’s try another image, this one with more text:



**Figure 5:** Another example input to our Tesseract + Python OCR system.

The above image is a screenshot from the “Prerequisites” section of my book, [***Practical Python and OpenCV***](https://www.pyimagesearch.com/practical-python-opencv/) — let’s see how the Tesseract binary handles this image:

Using Tesseract OCR with Python

$ tesseract images/example\_03.png stdout

PREREQUISITES

In order In make the rnosi of this, you will need (a have

a little bit of pregrarrmung experience. All examples in this

book are in the Python programming language. Familiarity

with Pyihon or other scriphng languages is suggesied, but

mm required.

You'll also need (a know some basic mathematics. This

book is handson and example driven: leis of examples and

lots of code, so even if your math skills are noi up to par.

do noi worry! The examples are very damned and heavily

documented (a help yuu follaw along.

Followed by testing the image with

ocr.py

 :

Using Tesseract OCR with Python

$ python ocr.py --image images/example\_03.png

PREREQUISITES

Lu order to make the most ol this, you will need to have

a little bit ol programming experience. All examples in this

book are in the Python programming language. Familiarity

with Python or other scripting languages is suggested, but

not requixed.

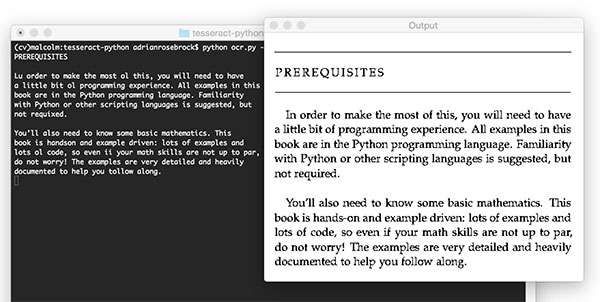
You’ll also need to know some basic mathematics. This

book is handson and example driven: lots of examples and

lots ol code, so even ii your math skills are not up to par,

do not worry! The examples are very detailed and heavily

documented to help you tollow along.

**Figure 6:** Applying Optical Character Recognition (OCR) using Python, OpenCV, and Tesseract.

Notice misspellings in both outputs including, but not limited to, “In”, “of”, “required”, “programming”, and “follow”.

The output for both of these do not match; however, interestingly the pre-processed version has only 8 word errors whereas the non-pre-processed image has 17 word errors **(over twice as many errors)**. Our pre-processing helps even on a clean background!

Python + Tesseract did a reasonable job here, but once again we have demonstrated the limitations of the library as an off-the-shelf classifier.

We may obtain good or acceptable results with Tesseract for OCR, but the best accuracy will come from training custom character classifiers on specific sets of fonts that appear in actual real-world images.

Don’t let the results of Tesseract OCR discourage you — simply manage your expectations and be realistic on Tesseract’s performance. There is no such thing as a true “off-the-shelf” OCR system that will give you perfect results (there are bound to be some errors).

***Note:***If your text is rotated, you may wish to do additional pre-processing as is performed in [***this previous blog post on correcting text skew***](https://www.pyimagesearch.com/2017/02/20/text-skew-correction-opencv-python/). Otherwise, if you’re interested in building a [***mobile document scanner***](https://www.pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/), you now have a reasonably good OCR system to integrate into it.

## 4.5..3 Summary

Here we learned how to apply the Tesseract OCR engine with the Python programming language. This enabled us to apply OCR algorithms from within our Python script.

The biggest downside is with the limitations of Tesseract itself. Tesseract works best when there are extremely clean segmentations of the foreground text from the background.

Furthermore these segmentations need to be as high resolution (DPI) as possible and the characters in the input image cannot appear “pixelated” after segmentation. If characters do appear pixelated then Tesseract will struggle to correctly recognize the text — we found this out even when applying images captured under ideal conditions (a PDF screenshot).

OCR, while no longer a new technology, is still an active area of research in the computer vision literature especially when applying OCR to real-world, unconstrained images. Deep learning and Convolutional Neural Networks (CNNs) are certainly enabling us to obtain higher accuracy, but we are still a long way from seeing “near perfect” OCR systems. Furthermore, as OCR has many applications across many domains, some of the best algorithms used for OCR are commercial and require licensing to be used in your own project

**Chapter 5 References**

**5.1 References**

References are those which are taken from the links on the internet and many of the others are the working procedure which is done in the main project the references about the man project working illustrated and the main body that can be seen without link and all other links which is related to the project is is present

* 1. **Ref To Chapter 1**
  2. [OpenCV: Introduction to OpenCV-Python Tutorials](https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html)docs.opencv.org
  3. <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_intro/py_intro.html>
  4. [Pycharm - Macros - Tutorialspoint](https://www.tutorialspoint.com/pycharm/pycharm_macros.htm)www.tutorialspoint.com
  5. [Pycharm - Omni - Tutorialspoint](https://www.tutorialspoint.com/pycharm/pycharm_omni.htm)www.tutorialspoint.com
  6. [Pycharm - Shortcuts - Tutorialspoint](https://www.tutorialspoint.com/pycharm/pycharm_shortcuts.htm)www.tutorialspoint.com
  7. [OpenCV Tutorial: A Guide to Learn OpenCV - PyImageSearch](https://www.pyimagesearch.com/2018/07/19/opencv-tutorial-a-guide-to-learn-opencv/)www.pyimagesearch.com

**5.3 Ref To Chapter 2**

2.1 [Image Processing with OpenCV Research Papers - Academia.edu](https://www.academia.edu/Documents/in/Image_Processing_with_OpenCV) www.academia.edu

2.2 <https://pdfs.semanticscholar.org/f247/ddb93a92b8b5c82fa771cd34d4b7c9ead0e3.pdfpdfs.semanticscholar.org>

2.3 [Download Opencv Research Paper - ledgys.io](http://ledgys.io/opencv_research_paper.pdf#forward)ledgys.io

2.4 [7th\_International\_Conference\_on\_Signal\_a.pdf](file:///C:\Users\AAMIR%20BASHIR\Downloads\7th_International_Conference_on_Signal_a.pdf)file:

2.5 [Call\_for\_papers\_-\_8th\_International\_Conf.pdf](file:///C:\Users\AAMIR%20BASHIR\Downloads\Call_for_papers_-_8th_International_Conf.pdf)file:

2.6 [International Journal on Organic Electronics (IJOE)](http://airccse.org/journal/IJOE/ijoe.html)airccse.org

**5.4 Ref To Chapter 3**

3.1[OpenCV: High-level design overview](https://docs.opencv.org/master/de/d4d/gapi_hld.html)docs.opencv.org

3.2 [(PDF) A Review on OpenCV](https://www.researchgate.net/publication/280977983_A_Review_on_OpenCV)www.researchgate.net

3.3[ResearchGate](https://www.researchgate.net/publication/280977983_A_Review_on_OpenCV/link/55cf4fd608ae6a881385d975/download)www.researchgate.net

**5.5 Ref To Chapter 3**

4.1 [Design and Development of Automated Intelligent Robot Using OpenCV - IEEE Conference Publication](https://ieeexplore.ieee.org/document/8437096)ieeexplore.ieee.org

4.2 [Head Pose Estimation using OpenCV and Dlib | Learn OpenCV](https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/)www.learnopencv.com

4.3 <https://www.pyimagesearch.com/2017/07/10/using-tesseract-ocr-python/>

4.4 <https://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/>

4.5 <https://docs.opencv.org/master/dc/d2c/tutorial_real_time_pose.html>

4.6 <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_intro/py_intro.html>

4.7 https://www.pyimagesearch.com/2017/07/10/using-tesseract-ocr-python/