

Machine Vision Course

Introduction

Machine vision course aims to build the basic knowledge of Image acquisition techniques such as choosing the Camera, Lens, and Lighting in Single-shot and Video mode. Understanding the image processing technique image transformation, segmentation, feature extraction, image reconstruction, edge detection, object recognition, stereo vision, and machine vision. This course shall be implemented on Open CV (Opensource software).

Table of Contents:

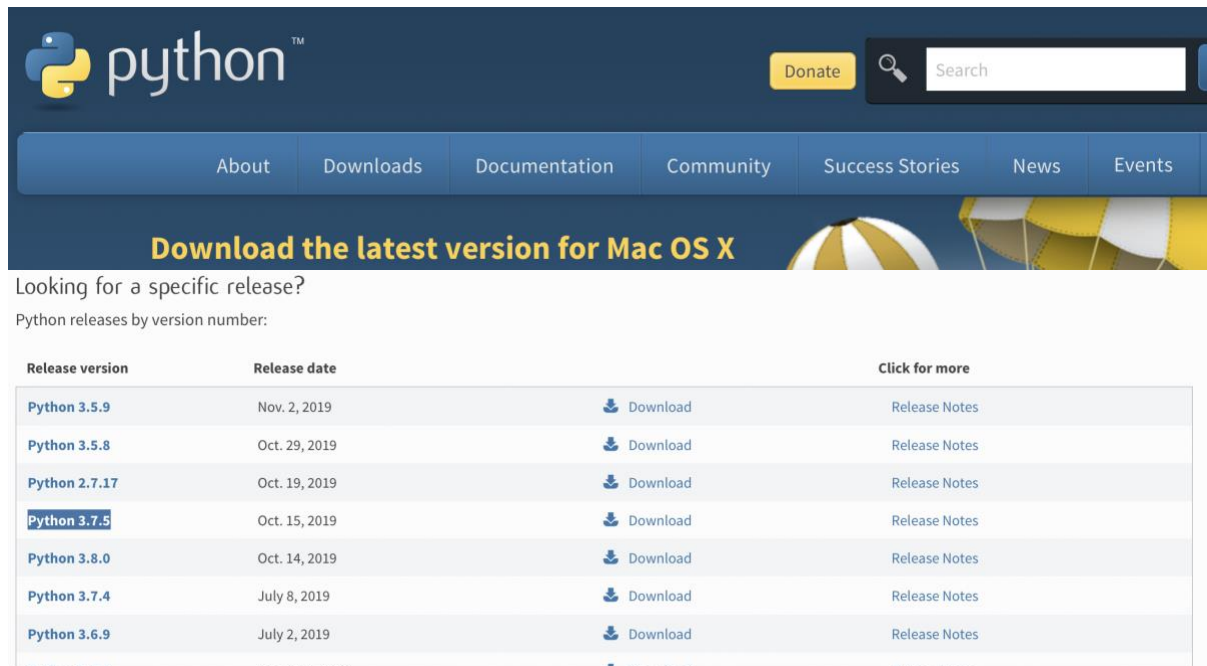
<u>INTRODUCTION</u>	<u>1</u>
<u>LAB 1 DEVELOPMENT ENVIRONMENT SETUP FOR OPENCV PYTHON</u>	<u>2</u>
<u>LAB 2 INTRODUCTION TO PYTHON PROGRAMMING</u>	<u>12</u>
<u>LAB 3 PYTHON PROGRAMMING FOR OPENCV</u>	<u>29</u>
<u>LAB 4 IMAGE OPERATION IN OPENCV</u>	<u>36</u>
<u>LAB 5 IMAGE MANIPULATION IN OPENCV</u>	<u>40</u>
<u>LAB 6 IMAGE PROCESSING IN OPENCV</u>	<u>47</u>
<u>LAB 7 IMAGE PROCESSING IN OPENCV 2</u>	<u>55</u>
<u>LAB 8 TEMPLATE MATCHING AND HOUGH TRANSFORM IN OPENCV</u>	<u>79</u>
<u>LAB 9 OCR USING NEURAL NET LSTM</u>	<u>91</u>
<u>LAB 10 VIDEO ANALYSIS</u>	<u>96</u>

Lab 1 Development Environment setup for OpenCV Python

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

This lab will teach about how to setup the environment of Python, OpenCV and PyCharm for development on your computer.

1. Install Python 3.7.x



The screenshot shows the Python.org website. At the top, there is a navigation bar with links: About, Downloads, Documentation, Community, Success Stories, News, and Events. Below the navigation bar, there is a prominent banner that says "Download the latest version for Mac OS X". Underneath the banner, it says "Looking for a specific release?" and "Python releases by version number:". Below this, there is a table listing various Python releases.

Release version	Release date	Click for more
Python 3.5.9	Nov. 2, 2019	Download Release Notes
Python 3.5.8	Oct. 29, 2019	Download Release Notes
Python 2.7.17	Oct. 19, 2019	Download Release Notes
Python 3.7.5	Oct. 15, 2019	Download Release Notes
Python 3.8.0	Oct. 14, 2019	Download Release Notes
Python 3.7.4	July 8, 2019	Download Release Notes
Python 3.6.9	July 2, 2019	Download Release Notes

Step 1: Download the Python 3 Installer

1. Open a browser window and navigate to the [Download page for Windows](#) at [python.org](#).

2. Underneath the heading at the top that says **Python Releases for Windows**, click on the link for the **Latest Python 3 Release - Python 3.x.x**. (As of this writing, the latest is Python 3.7.x.)
3. Scroll to the bottom and select either **Windows x86-64 executable installer** for 64-bit or **Windows x86 executable installer** for 32-bit. (See below.)

Remark: 32-bit or 64-bit Python?

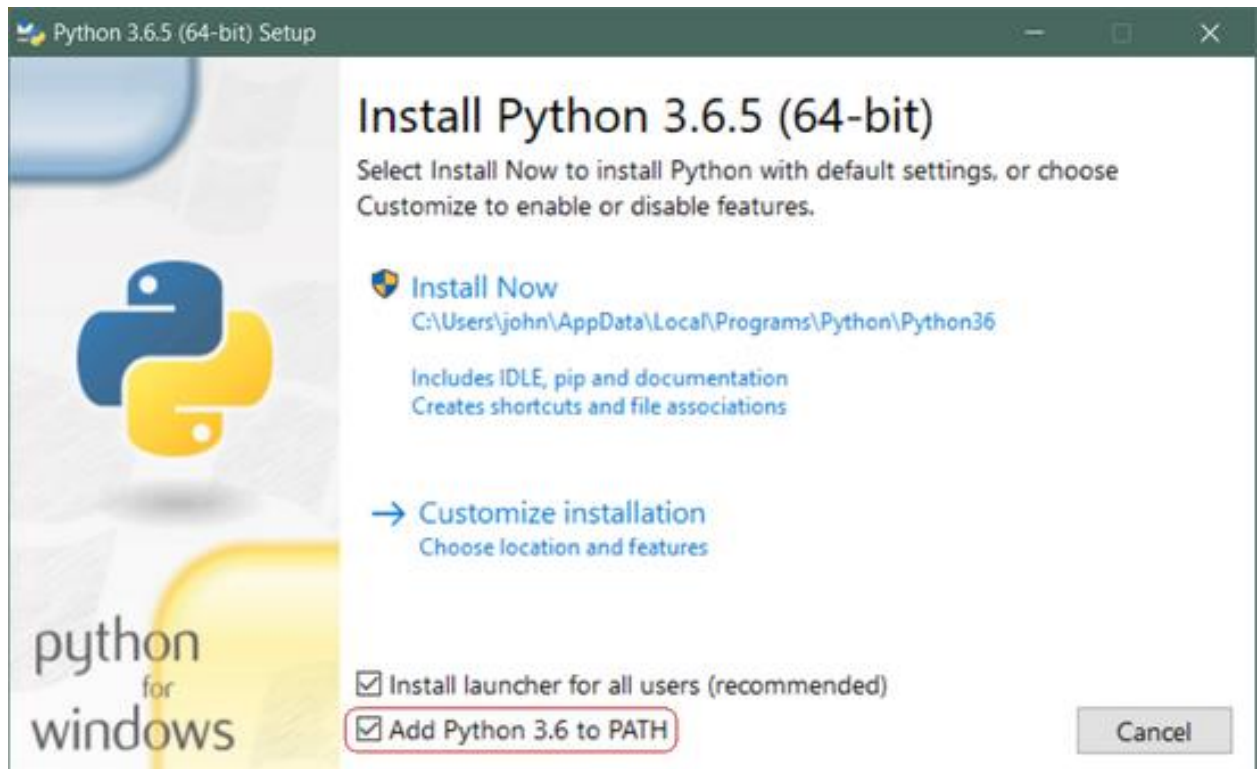
For Windows, you can choose either the 32-bit or 64-bit installer. Here's what the difference between the two comes down to:

- If your system has a 32-bit processor, then you should choose the 32-bit installer.
- On a 64-bit system, either installer will actually work for most purposes. The 32-bit version will generally use less memory, but the 64-bit version performs better for applications with intensive computation.
- If you're unsure which version to pick, go with the 64-bit version.

Note: Remember that if you get this choice “wrong” and would like to switch to another version of Python, you can just uninstall Python and then re-install it by downloading another installer from python.org.

Step 2: Run the Installer

Once you have chosen and downloaded an installer, simply run it by double-clicking on the downloaded file. A dialog should appear that looks something like this:



Important

You want to be sure to check the box that says Add Python 3.x to PATH as shown to ensure that the interpreter will be placed in your execution path.

Then just click Install Now. That should be all there is to it. A few minutes later you should have a working Python 3 installation on your system.

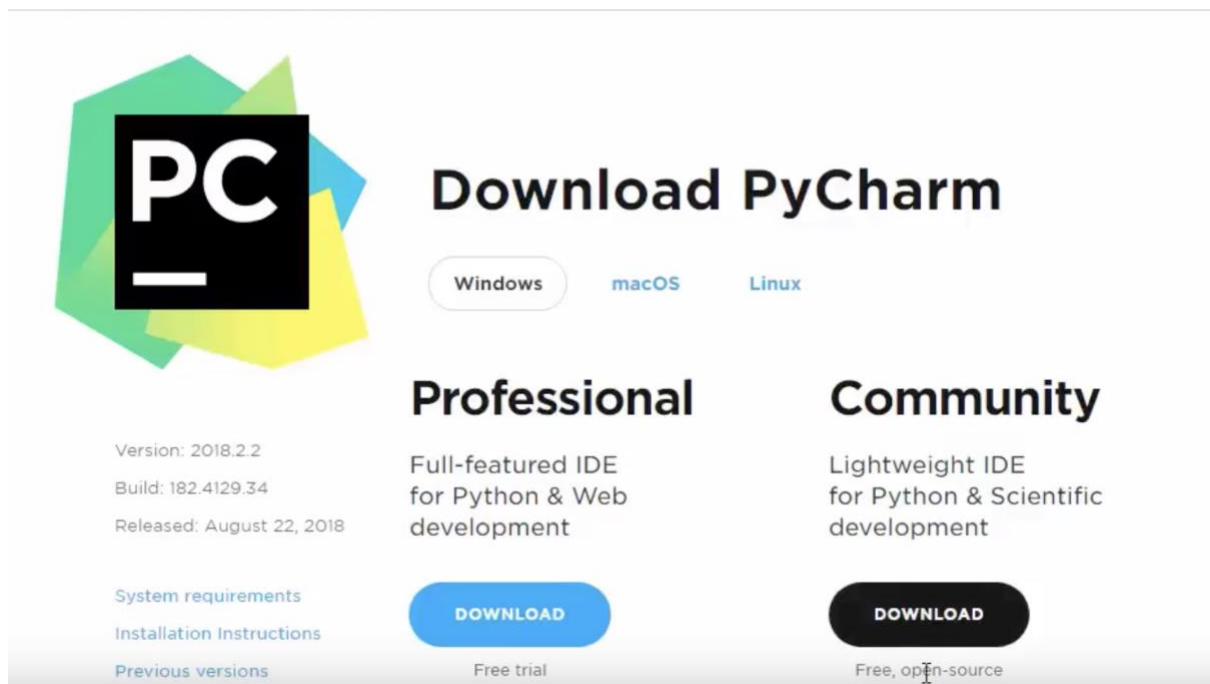
2. Install PyCharm IDE

PyCharm is an integrated development environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django as well as Data Science with Anaconda.

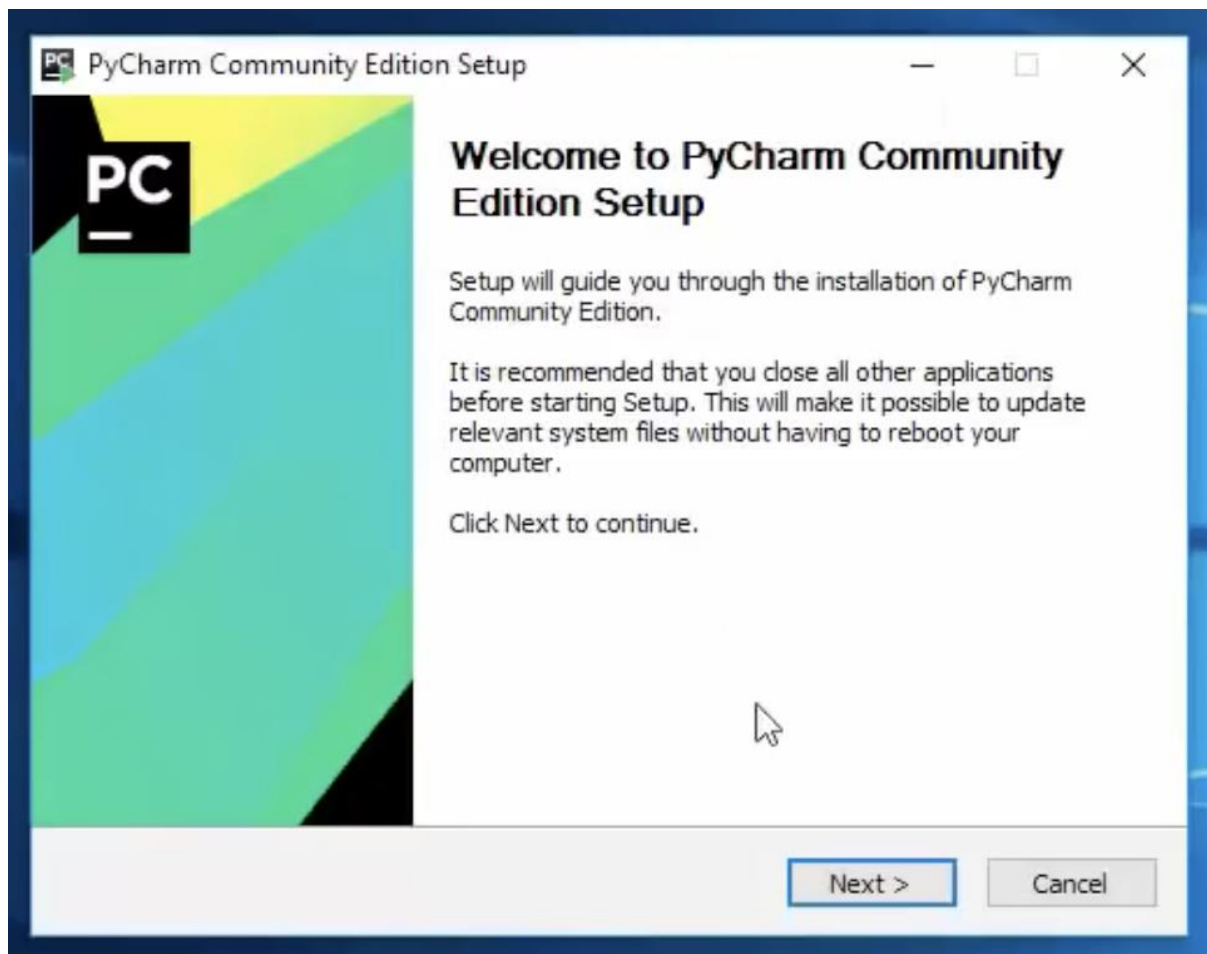
PyCharm is cross-platform, with Windows, macOS and Linux versions. The Community Edition is released under the Apache License, and there is also Professional Edition with extra features – released under a proprietary license.



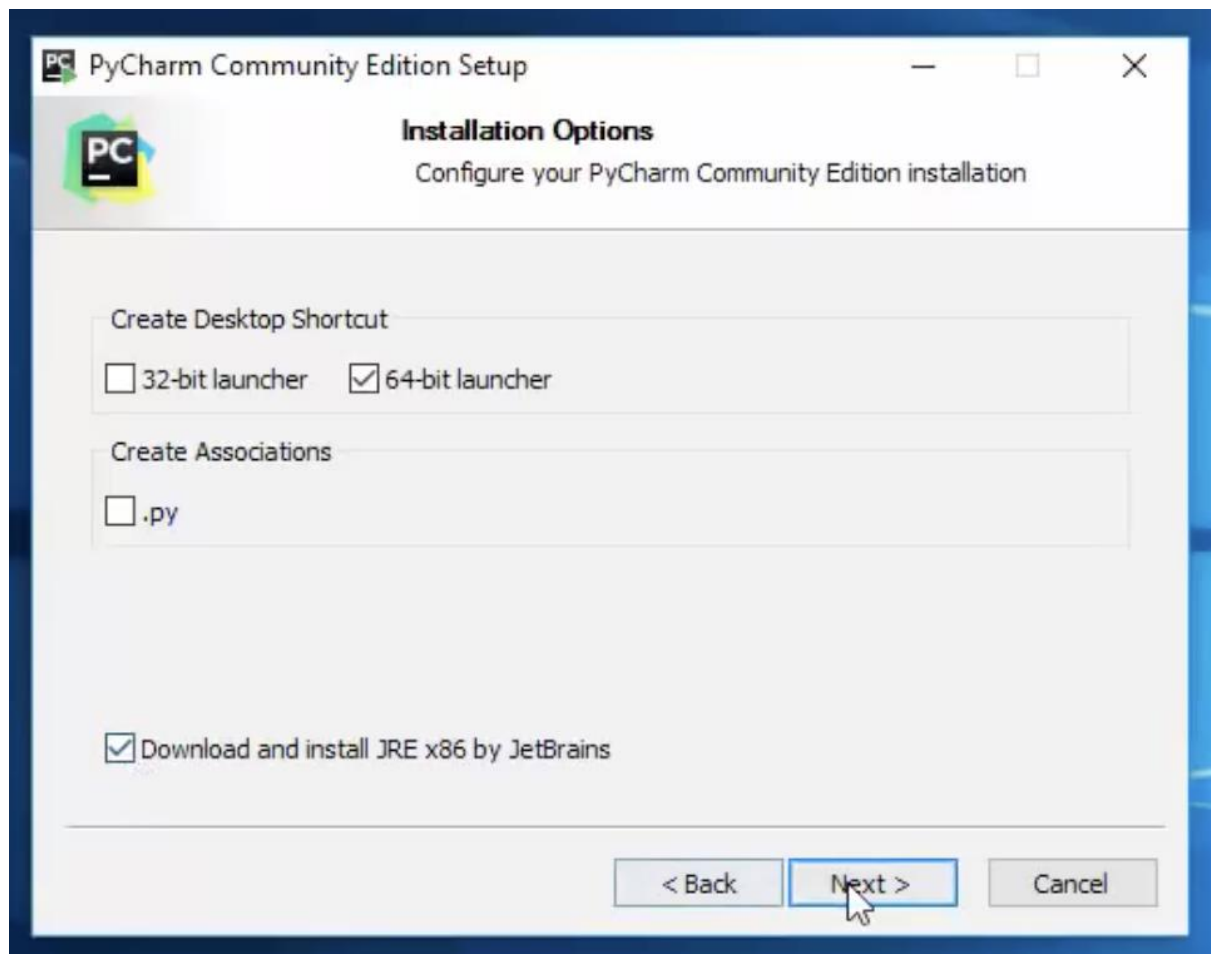
In this lab, we will use PyCharm Community version.



Install PyCharm



Choose 64-bit launcher and Download JRE



Then create new project



PyCharm

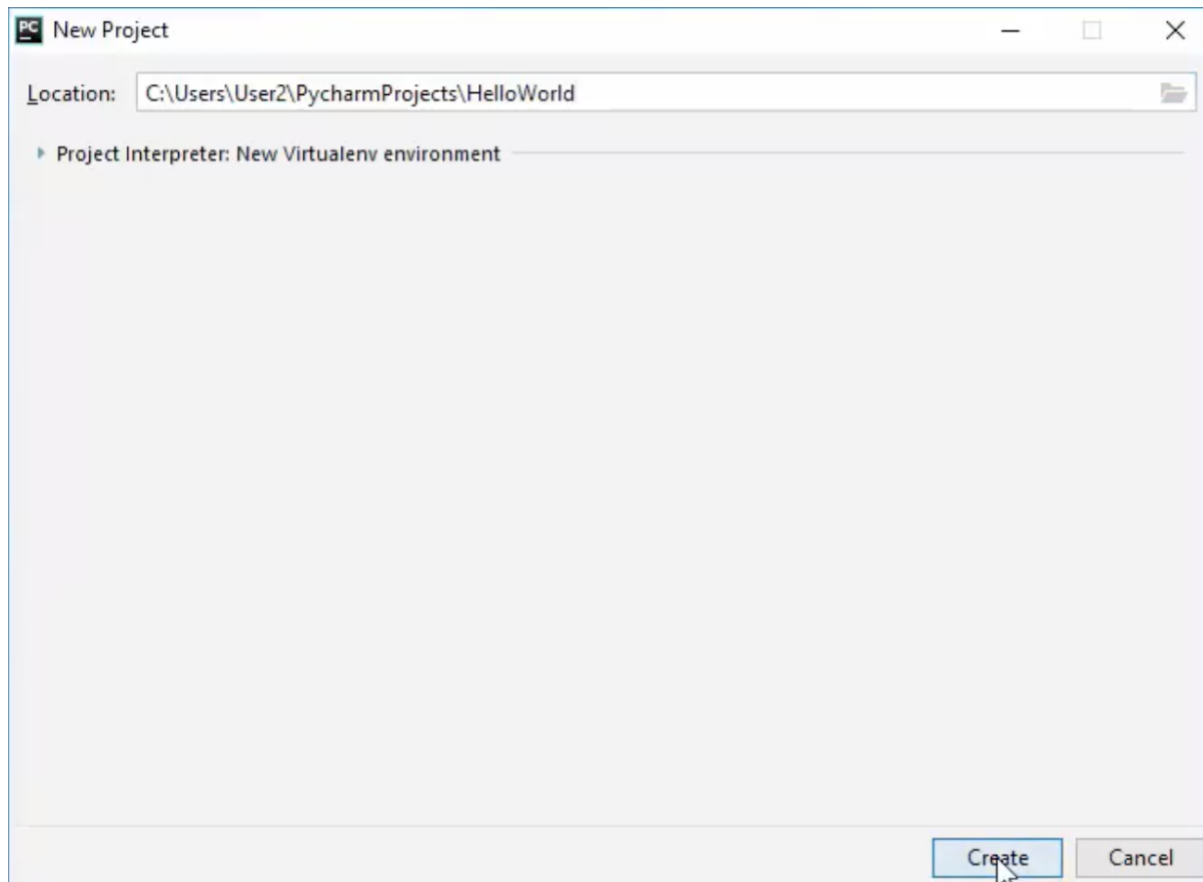
Version 2018.2.2

+ Create New Project

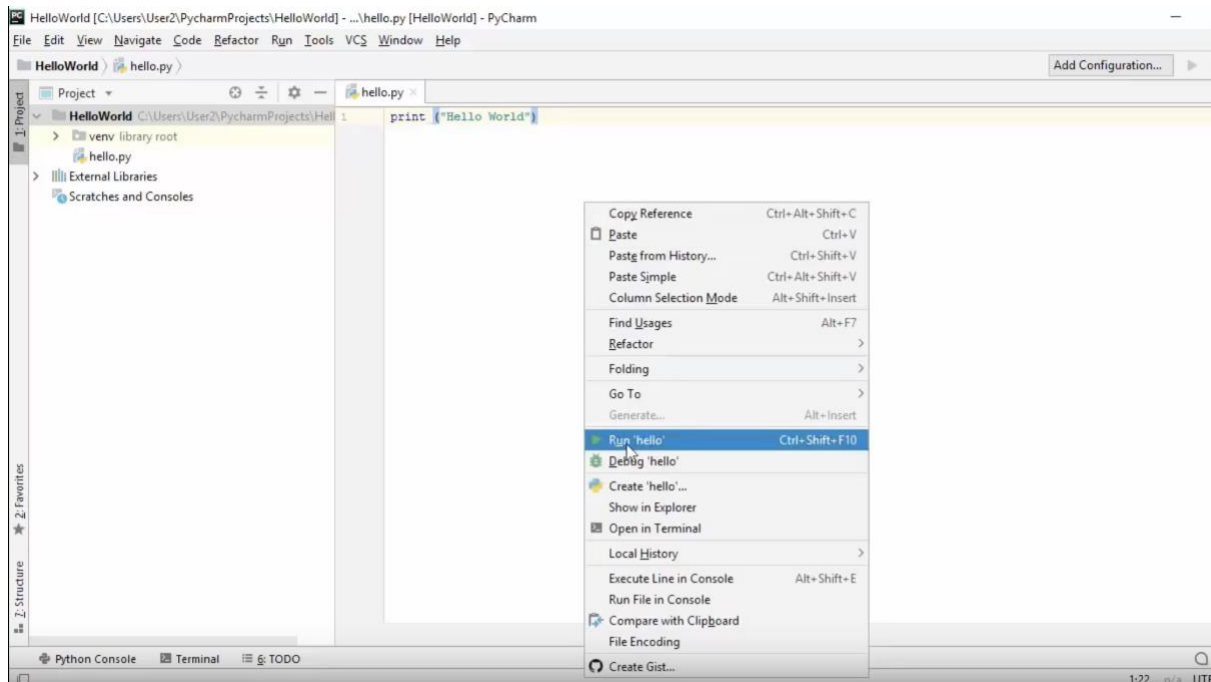
Open

Check out from Version Control ▾

⚙ Configure ▾ Get Help ▾

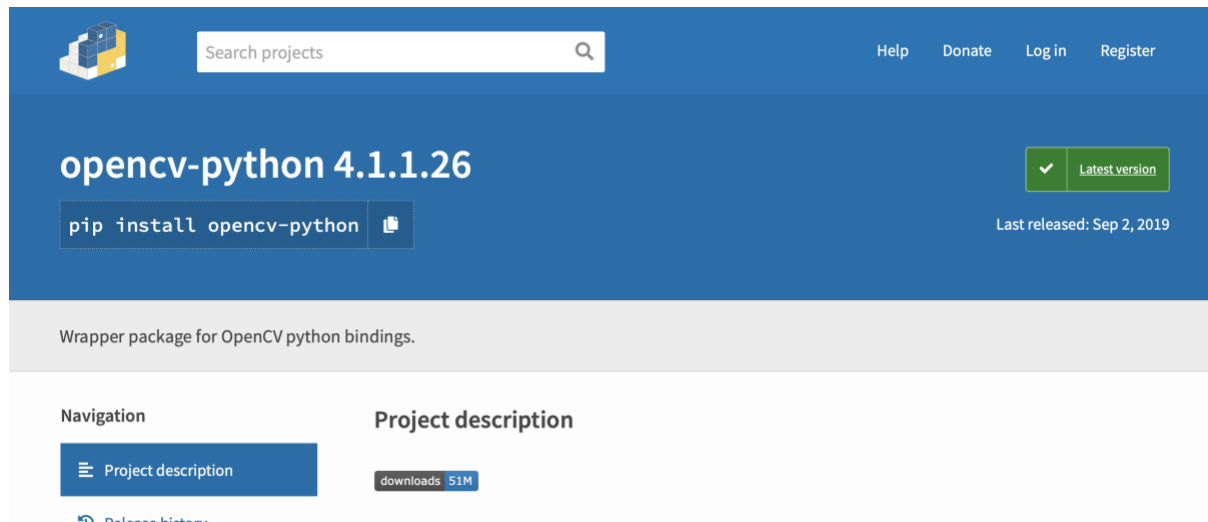


Run your first code



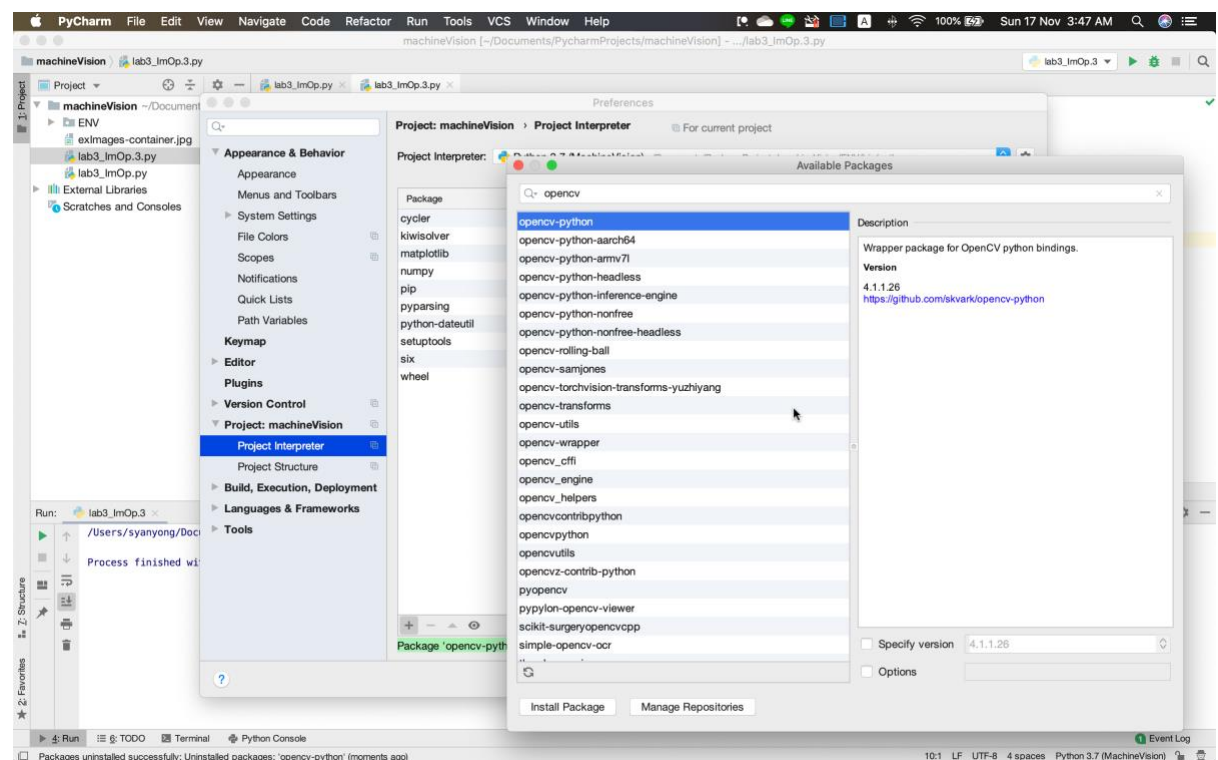
3. Install OpenCV Python

The resource of OpenCV Python is kept in pypi.org repository, so the student can go to website <https://pypi.org/project/opencv-python/> to check the updated version of the library or the other document.



Install from PyCharm IDE

- Go to File → Preferences → Project → Project Interpreter
- Click + Symbol
- Type in the search textbox as “opencv”, so the exactly package name is opencv-python
- Select the package name
- Click Install Package



4. Your first Python Program

Now that we have Python up and running, we can write our first Python program.

Let's create a very simple program called **"Hello World!"**. A "Hello, World!" is a simple program that outputs **Hello, World!** on the screen. Since it's a very simple program, it's often used to introduce a new programming language to a newbie.

Type the following code in any text editor or an IDE and save it as **helloWorld.py**

```
print("Hello world!")
```

Then, run the file. You will get the following output.

Hello world!

Congratulations! You just wrote your first program in Python.

As we can see, it was pretty easy. This is the beauty of Python programming language.

Lab 2 Introduction to Python programming

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python Identifiers

An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.

Rules for writing identifiers

1. Identifiers can be a combination of letters in lowercase (**a to z**) or uppercase (**A to Z**) or digits (**0 to 9**) or an underscore `_`. Names like `myClass`, `var_1` and `print_this_to_screen`, all are valid example.
2. An identifier cannot start with a digit. `1variable` is invalid, but `variable1` is perfectly fine.
3. Keywords cannot be used as identifiers.

```
a.
b. >>> global = 1
c. File "<interactive input>", line 1
d.     global = 1
e.         ^
f. SyntaxError: invalid syntax
```

4. We cannot use special symbols like `!`, `@`, `#`, `$`, `%` etc. in our identifier.

```
a.
b. >>> a@ = 0
c. File "<interactive input>", line 1
d.     a@ = 0
e.         ^
f. SyntaxError: invalid syntax
```

5. Identifier can be of any length.

Things to Remember

Python is a case-sensitive language. This means, `Variable` and `variable` are not the same. Always name identifiers that make sense.

While, `c = 10` is valid. Writing `count = 10` would make more sense and it would be easier to figure out what it does even when you look at your code after a long gap.

Multiple words can be separated using an underscore, `this_is_a_long_variable`.

Python Statement, Indentation and Comments

In this article, you will learn about Python statements, why indentation is important and use of comments in programming.

Python Statement

Instructions that a Python interpreter can execute are called statements. For example, `a = 1` is an assignment statement. `if` statement, `for` statement, `while` statement etc. are other kinds of statements which will be discussed later.

Multi-line statement

In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character (`\`). For example:

```
1. a = 1 + 2 + 3 + \  
2.    4 + 5 + 6 + \  
3.    7 + 8 + 9
```

This is explicit line continuation. In Python, line continuation is implied inside parentheses (), brackets [] and braces { }. For instance, we can implement the above multi-line statement as

```
1. a = (1 + 2 + 3 +  
2.    4 + 5 + 6 +  
3.    7 + 8 + 9)
```

Here, the surrounding parentheses () do the line continuation implicitly. Same is the case with [] and { }. For example:

```
1. colors = ['red',
```

```
2.     'blue',  
3.     'green']
```

We could also put multiple statements in a single line using semicolons, as follows

```
1. a = 1; b = 2; c = 3
```

Python Indentation

Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.

A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.

Generally four whitespaces are used for indentation and is preferred over tabs. Here is an example.

```
for i in range(1,11):  
    print(i)  
    if i == 5:  
        break
```

The enforcement of indentation in Python makes the code look neat and clean. This results into Python programs that look similar and consistent.

Python Comments

Comments are very important while writing a program. It describes what's going on inside a program so that a person looking at the source code does not have a hard time figuring it out. You might forget the key details of the program you just wrote in a month's time. So taking time to explain these concepts in form of comments is always fruitful.

In Python, we use the hash (#) symbol to start writing a comment.

It extends up to the newline character. Comments are for programmers for better understanding of a program. Python Interpreter ignores comment.

```
1. #This is a comment  
2. #print out Hello  
3. print('Hello')
```

Multi-line comments

If we have comments that extend multiple lines, one way of doing it is to use hash (#) in the beginning of each line. For example:

```
1. #This is a long comment
2. #and it extends
3. #to multiple lines
```

Python Variables

A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data which can be changed later throughout programming. For example,

```
1. number = 10
```

Here, we have created a named `number`. We have assigned value 10 to the variable.

You can think variable as a bag to store books in it and those books can be replaced at any time.

```
1. number = 10
2. number = 1.1
3. website = "apple.com"
4.
```

Initially, the value of `number` was 10. Later it's changed to 1.1.

Note: In Python, we don't assign values to the variables, whereas Python gives the reference of the object (value) to the variable.

Constants

A constant is a type of variable whose value cannot be changed. It is helpful to think of constants as containers that hold information which cannot be changed later.

Non technically, you can think of constant as a bag to store some books and those books cannot be replaced once placed inside the bag.

Assigning value to a constant in Python

In Python, constants are usually declared and assigned on a module. Here, the module means a new file containing variables, functions etc which is imported to main file. Inside the module, constants are written in all capital letters and underscores separating the words.

Example 3: Declaring and assigning value to a constant

Create a constant.py

```
1. PI = 3.14
2. GRAVITY = 9.8
```

Create a main.py

```
1. import constant
2.
3. print(constant.PI)
4. print(constant.GRAVITY)
```

When you run the program, the output will be:

```
3.14
9.8
```

In the above program, we create a constant.py module file. Then, we assign the constant value to PI and GRAVITY. After that, we create a main.py file and import the constant module. Finally, we print the constant value.

Note: In reality, we don't use constants in Python. The globals or constants module is used throughout the Python programs.

Rules and Naming Convention for Variables and constants

1. Constant and variable names should have a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_). For example:

```
snake_case
MACRO_CASE
```



```
camelCase
CapWords
```

2. Create a name that makes sense. For example, `vowel` makes more sense than `v`.
3. If you want to create a variable name having two words, use underscore to separate them. For example:

```
my_name
current_salary
```

4. Use capital letters possible to declare a constant. For example:

```
PI
G
MASS
SPEED_OF_LIGHT
TEMP
```

5. Never use special symbols like `!`, `@`, `#`, `$`, `%`, etc.
6. Don't start a variable name with a digit.

Python Data Types

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are listed below.

Python Numbers

Integers, floating point numbers and complex numbers falls under Python numbers category.

They are defined as `int`, `float` and `complex` class in Python.

We can use the `type()` function to know which class a variable or a value belongs to and the `isinstance()` function to check if an object belongs to a particular class.

```
a = 5
print(a, "is of type", type(a))

a = 2.0
print(a, "is of type", type(a))
```

```
a = 1+2j
print(a, "is complex number?", isinstance(1+2j,complex))
```

Python Strings

String is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, `'''` or `"""`.

```
s = 'Hello world!'

# s[4] = 'o'
print("s[4] = ", s[4])

# s[6:11] = 'world'
print("s[6:11] = ", s[6:11])

# Generates error
# Strings are immutable in Python
s[5] = 'd'
```

Python List

List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type.

Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [].

We can use the slicing operator [] to extract an item or a range of items from a list. Index starts from 0 in Python.

```
a = [5,10,15,20,25,30,35,40]

# a[2] = 15
print("a[2] = ", a[2])

# a[0:3] = [5, 10, 15]
print("a[0:3] = ", a[0:3])
```

```
# a[5:] = [30, 35, 40]
print("a[5:] = ", a[5:])
```

Python Dictionary

[Dictionary](#) is an unordered collection of key-value pairs. It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.

In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type.

```
d = {'value':1, 'key':2} # Example of dictionary define
print(type(d))

print("d[1] = ", d[1]);

print("d['key'] = ", d['key']);

# Generates error
print("d[2] = ", d[2]);
```

Python Tuple

[Tuple](#) is an ordered sequence of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified.

Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically.

It is defined within parentheses () where items are separated by commas. We can use the slicing operator [] to extract items but we cannot change its value.

```
t = (5, 'program', 1+3j)

# t[1] = 'program'
print("t[1] = ", t[1])

# t[0:3] = (5, 'program', (1+3j))
print("t[0:3] = ", t[0:3])
```

```
# Generates error
# Tuples are immutable
t[0] = 10
```

Python Output Using print() function

We use the `print()` function to output data to the standard output device (screen).

We can also [output data to a file](#), but this will be discussed later. An example use is given below.

```
print('This sentence is output to the screen')
# Output: This sentence is output to the screen

a = 5

print('The value of a is', a)
# Output: The value of a is 5
```

Output formatting

Sometimes we would like to format our output to make it look attractive. This can be done by using the `str.format()` method. This method is visible to any string object.

```
1. >>> x = 5; y = 10
2. >>> print('The value of x is {} and y is {}'.format(x,y))
3. The value of x is 5 and y is 10
```

Here the curly braces `{ }` are used as placeholders. We can specify the order in which it is printed by using numbers (tuple index).

```
print('I love {0} and {1}'.format('bread','butter'))
```

Output: I love bread and butter

```
print('I love {1} and {0}'.format('bread','butter'))
```

Output: I love butter and bread

We can even use keyword arguments to format the string.

```
1. >>> print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning', name = 'John'))
```

```
2. Hello John, Goodmorning
```

We can even format strings like the old `printf()` style used in [C programming language](#). We use the `%` operator to accomplish this.

```
1. >>> x = 12.3456789
```

```
2. >>> print('The value of x is %3.2f' %x)
```

```
3. The value of x is 12.35
```

```
4. >>> print('The value of x is %3.4f' %x)
```

```
5. The value of x is 12.3457
```

Python Input

Up till now, our programs were static. The value of variables were defined or hard coded into the source code.

To allow flexibility we might want to take the input from the user. In Python, we have the `input()` function to allow this. The syntax for `input()` is

```
input([prompt])
```

where `prompt` is the string we wish to display on the screen. It is optional.

```
1. >>> num = input('Enter a number: ')
2. Enter a number: 10
3. >>> print(num)
4. 10
```

Python Operators

What are operators in python? Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

For example:

```
1. >>> 2+3
2. 5
```

Here, + is the operator that performs addition. 2 and 3 are the operands and 5 is the output of the operation.

Arithmetic operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Operator	Meaning	Example
+	Add two operands or unary plus	$x + y$ +2
-	Subtract right operand from the left or unary minus	$x - y$ -2
*	Multiply two operands	$x * y$

/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	$x \% y$ (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	$x // y$
**	Exponent - left operand raised to the power of right	$x^{**}y$ (x to the power y)

Example #1: Arithmetic operators in Python

```

1. x = 15
2. y = 4
3.
4. # Output: x + y = 19
5. print('x + y =',x+y)
6.
7. # Output: x - y = 11
8. print('x - y =',x-y)
9.
10. # Output: x * y = 60
11. print('x * y =',x*y)
12.
13. # Output: x / y = 3.75
14. print('x / y =',x/y)
15.
16. # Output: x // y = 3
17. print('x // y =',x//y)
18.
19. # Output: x ** y = 50625
20. print('x ** y =',x**y)

```

When you run the program, the output will be:

```
x + y = 19
x - y = 11
x * y = 60
x / y = 3.75
x // y = 3
x ** y = 50625
```

Example #2: Comparison operators in Python

```
1. x = 10
2. y = 12
3.
4. # Output: x > y is False
5. print('x > y is',x>y)
6.
7. # Output: x < y is True
8. print('x < y is',x<y)
9.
10. # Output: x == y is False
11. print('x == y is',x==y)
12.
13. # Output: x != y is True
14. print('x != y is',x!=y)
15.
16. # Output: x >= y is False
17. print('x >= y is',x>=y)
18.
19. # Output: x <= y is True
20. print('x <= y is',x<=y)
```

Example #3: Logical Operators in Python

```
1. x = True
2. y = False
3.
4. # Output: x and y is False
5. print('x and y is',x and y)
6.
7. # Output: x or y is True
8. print('x or y is',x or y)
9.
10. # Output: not x is False
```



```
11. print('not x is',not x)
```

Python Variable Scope

Although there are various unique namespaces defined, we may not be able to access all of them from every part of the program. The concept of scope comes into play.

Scope is the portion of the program from where a namespace can be accessed directly without any prefix.

At any given moment, there are at least three nested scopes.

1. Scope of the current function which has local names
2. Scope of the module which has global names
3. Outermost scope which has built-in names

When a reference is made inside a function, the name is searched in the local namespace, then in the global namespace and finally in the built-in namespace.

If there is a function inside another function, a new scope is nested inside the local scope.

Example of Scope in Python

```
def outer_function():  
    b = 20  
    def inner_func():  
        c = 30  
  
a = 10
```

Here, the variable `a` is in the global namespace. Variable `b` is in the local namespace of `outer_function()` and `c` is in the nested local namespace of `inner_function()`.

When we are in `inner_function()`, `c` is local to us, `b` is nonlocal and `a` is global. We can read as well as assign new values to `c` but can only read `b` and `a` from `inner_function()`.

If we try to assign a value to `b`, a new variable `b` is created in the local namespace which is different than the nonlocal `b`. Same thing happens when we assign a value to `a`.

However, if we declare `a` as global, all the reference and assignment go to the global `a`. Similarly, if we want to rebind the variable `b`, it must be declared as nonlocal. The following example will further clarify this.

```
def outer_function():  
    a = 20  
  
    def inner_function():
```

```
a = 30
print('a =', a)

inner_function()
print('a =', a)

a = 10
outer_function()
print('a =', a)
```

As you can see, the output of this program is

```
a = 30
a = 20
a = 10
```

In this program, three different variables **a** are defined in separate namespaces and accessed accordingly. While in the following program,

```
def outer_function():
    global a
    a = 20

    def inner_function():
        global a
        a = 30
        print('a =', a)

    inner_function()
    print('a =', a)

a = 10
outer_function()
print('a =', a)
```


Lab 3 Python Programming for OpenCV

Python if Statement Syntax

Decision making is required when we want to execute a code only if a certain condition is satisfied. The if...elif...else statement is used in Python for decision making.

Syntax of if statement

```
if test expression:  
    statement(s)
```

Here, the program evaluates the test expression and will execute statement(s) only if the test expression is True.

If the test expression is False, the statement(s) is not executed.

In Python, the body of the if statement is indicated by the indentation. Body starts with an indentation and the first unindented line marks the end.

Python interprets non-zero values as True. None and 0 are interpreted as False.

Python if Statement Flowchart

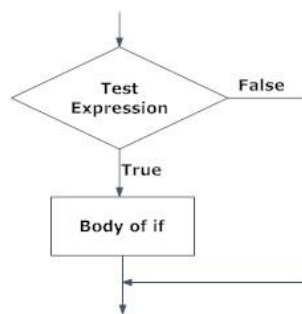


Fig: Operation of if statement

Example of if statement

If the number is positive, we print an appropriate message

```
num = 3  
if num > 0:  
    print(num, "is a positive number.")  
print("This is always printed.")  
  
num = -1  
if num > 0:  
    print(num, "is a positive number.")  
print("This is also always printed.")
```

Output

```
3 is a positive number
This is always printed
This is also always printed.
```

In the above example, `num > 0` is the test expression.

The body of `if` is executed only if this evaluates to `True`.

When variable `num` is equal to 3, test expression is true and body inside body of `if` is executed.

If variable `num` is equal to -1, test expression is false and body inside body of `if` is skipped.

The `print()` statement falls outside of the `if` block (unindented). Hence, it is executed regardless of the test expression.

Python if...else Statement

Syntax of if-else statement

```
if test expression:
```

```
    Body of if
```

```
else:
```

```
    Body of else
```

The `if..else` statement evaluates test expression and will execute body of `if` only when test condition is `True`.

If the condition is `False`, body of `else` is executed. Indentation is used to separate the blocks.

Python if..else Flowchart

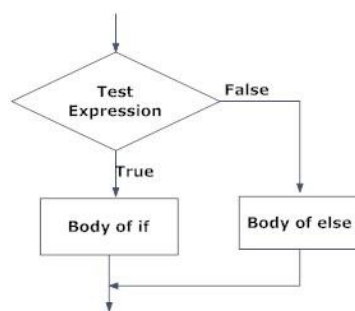


Fig: Operation of if...else statement

Example of if-else statement

```
# Program checks if the number is positive or negative
```

```
# And displays an appropriate message
```

```
num = 3
```

```
# Try these two variations as well.
```

```
# num = -5
```

```
# num = 0
```

```
if num >= 0:
```

```
    print("Positive or Zero")
```

```
else:
```

```
    print("Negative number")
```

```
Output
```

```
Positive or Zero
```

In the above example, when `num` is equal to 3, the test expression is true and body of `if` is executed and body of `else` is skipped.

If `num` is equal to -5, the test expression is false and body of `else` is executed and body of `if` is skipped.

If `num` is equal to 0, the test expression is true and body of `if` is executed and body of `else` is skipped.

Python if...elif...else Statement

```
Syntax of if-else statement
```

```
if test expression:
```

```
    Body of if
```

```
elif test expression:
```

```
    Body of elif
```

```
else:
```

```
    Body of else
```

The `elif` is short for else if. It allows us to check for multiple expressions.

If the condition for `if` is False, it checks the condition of the next `elif` block and so on.

If all the conditions are False, body of `else` is executed.

Only one block among the several `if...elif...else` blocks is executed according to the condition.

The `if` block can have only one `else` block. But it can have multiple `elif` blocks.

Flowchart of if...elif...else

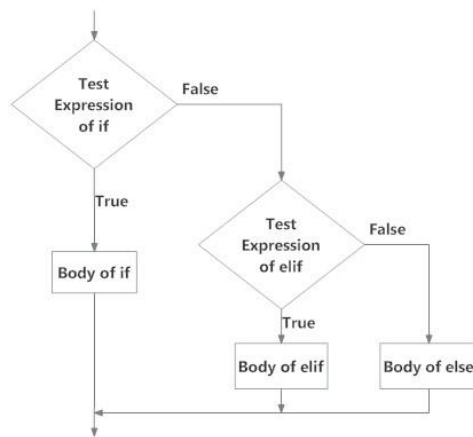


Fig: Operation of if...elif...else statement

Example of if-elif-else statement

In this program,

we check if the number is positive or

negative or zero and

display an appropriate message

`num = 3.4`

Try these two variations as well:

num = 0

num = -4.5

`if num > 0:`

`print("Positive number")`

`elif num == 0:`

`print("Zero")`

`else:`

`print("Negative number")`

Output

Positive number

When variable `num` is positive, Positive number is printed.

If `num` is equal to 0, Zero is printed.

If `num` is negative, Negative number is printed

Python Nested if statements

We can have a `if...elif...else` statement inside another `if...elif...else` statement. This is called nesting in computer programming.

Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if we can.

Example of nested if statement

```
# In this program, we input a number  
# check if the number is positive or  
# negative or zero and display  
# an appropriate message  
# This time we use nested if  
num = float(input("Enter a number: "))  
if num >= 0:  
    if num == 0:  
        print("Zero")  
    else:  
        print("Positive number")  
else:  
    print("Negative number")
```

Output 1

```
Enter a number: 5  
Positive number
```

Output 2

```
Enter a number: -1  
Negative number
```

Output 3

```
Enter a number: 0  
Zero
```

Python for loop

In this article, you'll learn to iterate over a sequence of elements using the different variations of for loop.

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

Syntax of for loop

```
for val in sequence:
```

```
    Body of for
```

Here, **val** is the variable that takes the value of the item inside the sequence on each iteration.

Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

Flowchart of for Loop

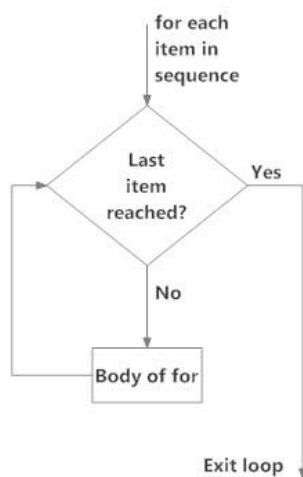


Fig: operation of for loop

Python while Loop

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

We generally use this loop when we don't know beforehand, the number of times to iterate.

Syntax of while loop

```
while test_expression:
```

```
    Body of while
```

Flowchart of while Loop

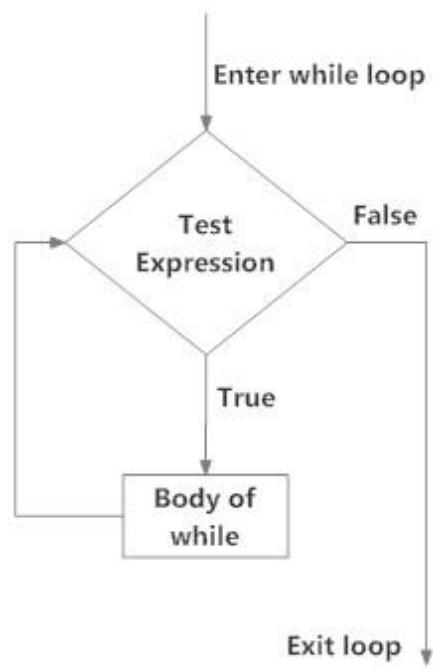


Fig: operation of while loop

Lab 4 Image Operation in OpenCV

Goals

- Here, you will learn how to read an image, how to display it and how to save it back
- You will learn these functions : `cv2.imread()`, `cv2.imshow()` , `cv2.imwrite()`

Read an image

Use the function `cv2.imread()` to read an image. The image should be in the working directory or a full path of image should be given.

The color plane in array of OpenCV are Blue, Green and Red (BGR), so the color plane is different from RGB.

Second argument is a flag which specifies the way image should be read.

- `cv2.IMREAD_COLOR` : Loads a color image. Any transparency of image will be neglected. It is the default flag.
- `cv2.IMREAD_GRAYSCALE` : Loads image in grayscale mode
- `cv2.IMREAD_UNCHANGED` : Loads image as such including alpha channel

See the code below:

```
import numpy as np
import cv2

# Load an color image in grayscale
img = cv2.imread("exImages-container.jpg", cv2.IMREAD_COLOR)
```

Warning: Even if the image path is wrong, it won't throw any error, but print `img` will give you `None`

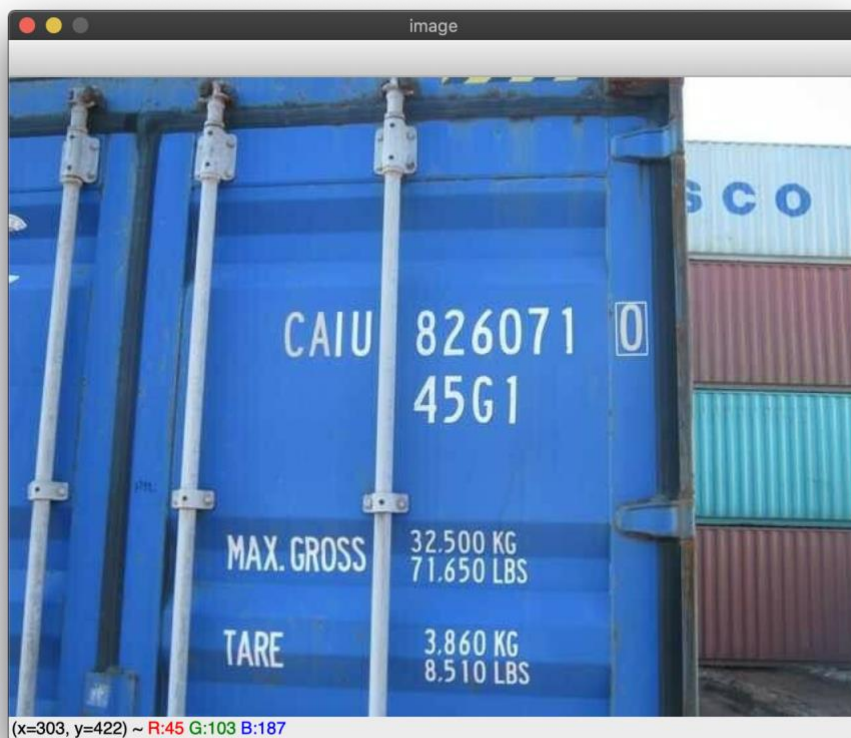
Display an image

Use the function `cv2.imshow()` to display an image in a window. The window automatically fits to the image size.

First argument is a window name which is a string. second argument is our image. You can create as many windows as you wish, but with different window names.

```
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

A screenshot of the window will look like this'



cv2.waitKey() is a keyboard binding function. Its argument is the time in milliseconds. The function waits for specified milliseconds for any keyboard event. If you press any key in that time, the program continues. If **0** is passed, it waits indefinitely for a key stroke. It can also be set to detect specific key strokes like, if key *a* is pressed etc which we will discuss below.

cv2.destroyAllWindows() simply destroys all the windows we created. If you want to destroy any specific window, use the function **cv2.destroyWindow()** where you pass the exact window name as the argument.

Note: There is a special case where you can already create a window and load image to it later. In that case, you can specify whether window is resizable or not. It is done with the function `cv2.namedWindow()`. By default, the flag is `cv2.WINDOW_AUTOSIZE`. But if you specify flag to be `cv2.WINDOW_NORMAL`, you can resize window. It will be helpful when image is too large in dimension and adding track bar to windows.

See the code below:

```
cv2.namedWindow('image', cv2.WINDOW_NORMAL)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Write an image

Use the function `cv2.imwrite()` to save an image.

First argument is the file name, second argument is the image you want to save.

```
cv2.imwrite("exImages-container2.jpg", img)
```

This will save the image in PNG format in the working directory.

Sum it up

Below program loads an image in grayscale, displays it, save the image if you press 's' and exit, or simply exit without saving if you press *ESC* key.

```
import numpy as np
import cv2

img = cv2.imread("exImages-container.jpg", cv2.IMREAD_GRAYSCALE)
cv2.imshow('image',img)
k = cv2.waitKey(0)
if k == 27:          # wait for ESC key to exit
    cv2.destroyAllWindows()
elif k == ord('s'): # wait for 's' key to save and exit
    cv2.imwrite("exImages-container2.jpg",img)
    cv2.destroyAllWindows()
```

Warning: If you are using a 64-bit machine, you will have to modify `k = cv2.waitKey(0)` line as follows : `k = cv2.waitKey(0) & 0xFF`

Using Matplotlib

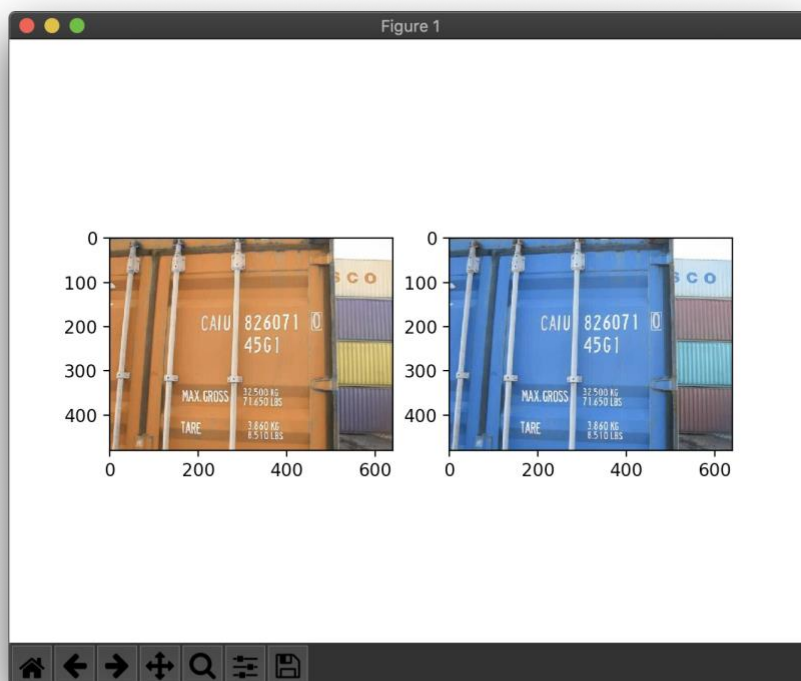
Matplotlib is a plotting library for Python which gives you wide variety of plotting methods. You will see them in coming articles. Here, you will learn how to display image with Matplotlib. You can zoom images, save it etc using Matplotlib.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('exImages-container.jpg')
b,g,r = cv2.split(img)
img2 = cv2.merge([r,g,b])
plt.subplot(121);plt.imshow(img) # expects distorted color
plt.subplot(122);plt.imshow(img2) # expect true color
plt.show()

cv2.imshow('bgr image',img) # expects true color
cv2.imshow('rgb image',img2) # expects distorted color
cv2.waitKey(0)
cv2.destroyAllWindows()
```

A screen-shot of the window will look like this :



See also

Plenty of plotting options are available in Matplotlib. Please refer to Matplotlib docs for more details. Some, we will see on the way.

Warning

Color image loaded by OpenCV is in BGR mode. But Matplotlib displays in RGB mode. So color images will not be displayed correctly in Matplotlib if image is read with OpenCV. Please see the exercises for more details.

Lab 5 Image Manipulation in OpenCV

Rotate Image

Python's OpenCV handles images as NumPy array ndarray. There are functions for rotating or flipping images (= ndarray) in OpenCV and NumPy, either of which can be used.

The OpenCV function that rotates the image (= ndarray) is **cv2.rotate()**.

Specify the original ndarray as the first argument and the constant indicating the rotation angle and direction as the second argument rotateCode.

The following three constants can be specified in rotateCode.

- cv2.ROTATE_90_CLOCKWISE
- cv2.ROTATE_90_COUNTERCLOCKWISE
- cv2.ROTATE_180

Sample code and results are below.

```
import cv2

img = cv2.imread('exImages-container.jpg', cv2.IMREAD_COLOR)
print(type(img))
print(img.shape)
cv2.imshow("Original Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()

img_rotate_90_clockwise = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)
```



```
cv2.imshow("Rotate 90 Clockwise Image", img_rotate_90_clockwise)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Flip Image

The image is flipped according to the value of flipCode as follows:

- flipcode = 0: flip vertically
- flipcode > 0: flip horizontally
- flipcode < 0: flip vertically and horizontally

```
import cv2
img = cv2.imread('exImages-container.jpg', cv2.IMREAD_COLOR)
print(type(img))
print(img.shape)
cv2.imshow("Original Image", img)
```

```

cv2.waitKey(0)
cv2.destroyAllWindows()

img_flip = cv2.flip(img, 0)
cv2.imshow("Flip Image", img_flip)
cv2.waitKey(0)
cv2.destroyAllWindows()

```



Crop Image

It's very simple to crop image in Python, so numpy slicing can be used. In Python programming language, the part of array can slice by this syntax `array_name[start : end]`.

```

import cv2
img = cv2.imread('exImages-container.jpg', cv2.IMREAD_COLOR)
# print(type(img))
print(img.shape)
print(img.shape[0])

crop = img[0:200, 0:400] # Crop size is starting from y=0 to 200, x=0 to 400

cv2.imshow("Original Image", img)
cv2.imshow("Crop Image", crop)

```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Accessing and Modify pixel values

You can access a pixel value by its row and column coordinates. For BGR image, it returns an array of Blue, Green, Red values. For grayscale image, just corresponding intensity is returned. And, you can also modify the pixel values the same way.

Warning

Numpy is a optimized library for fast array calculations. So simply accessing each and every pixel values and modifying it will be very slow and it is discouraged.

Note

Above mentioned method is normally used for selecting a region of array, say first 5 rows and last 3 columns like that. For individual pixel access, Numpy array methods, array.item() and array.itemset() is considered to be better. But it always returns a scalar. So if you want to access all B,G,R values, you need to call array.item()separately for all.

```
import cv2
img = cv2.imread('exImages-container.jpg', cv2.IMREAD_COLOR)
px = img[100,100]
print(px)
```

```
cv2.imshow("Original Image", img)
```

```
# accessing only blue pixel
```

```
blue = img[100,100,0]
```

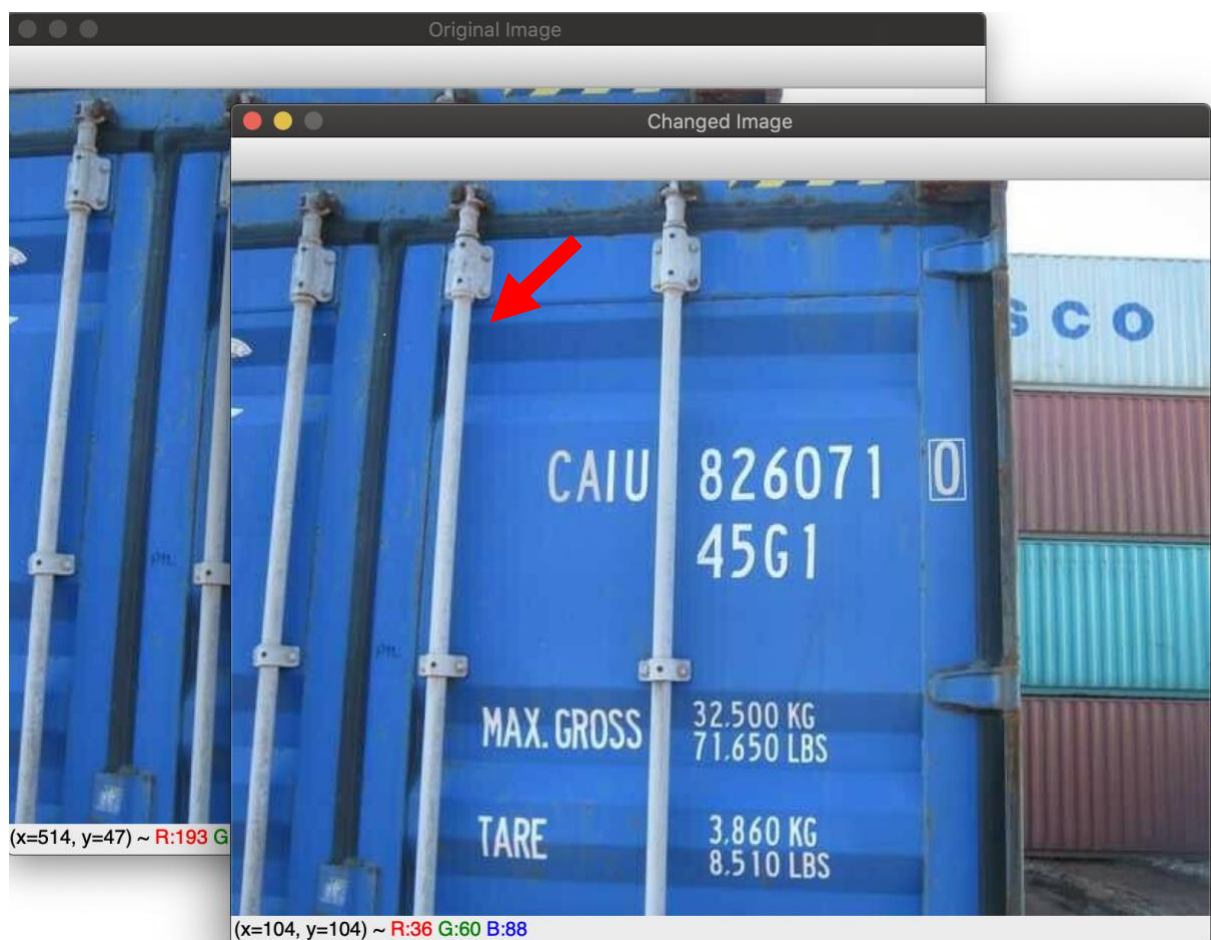
```
print(blue)
```

```
img[100,100] = [255,255,255]
```

```
cv2.imshow("Updated Image", img)
```

```
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```



Accessing Image Properties

Image properties include number of rows, columns and channels, type of image data, number of pixels etc.

Shape of image is accessed by `img.shape`. It returns a tuple of number of rows, columns and channels (if image is color).

Total number of pixels is accessed by `img.size`.

Image datatype is obtained by `img.dtype`.

```
print(img.shape) # Get shape of image
# > (342, 548, 3)
print(img.size) # Get size of image
# > 562248
print(img.dtype) # Get datatype of image
# > uint8
```

`img.dtype` is very important while debugging because a large number of errors in OpenCV-Python code is caused by invalid datatype.

Image ROI

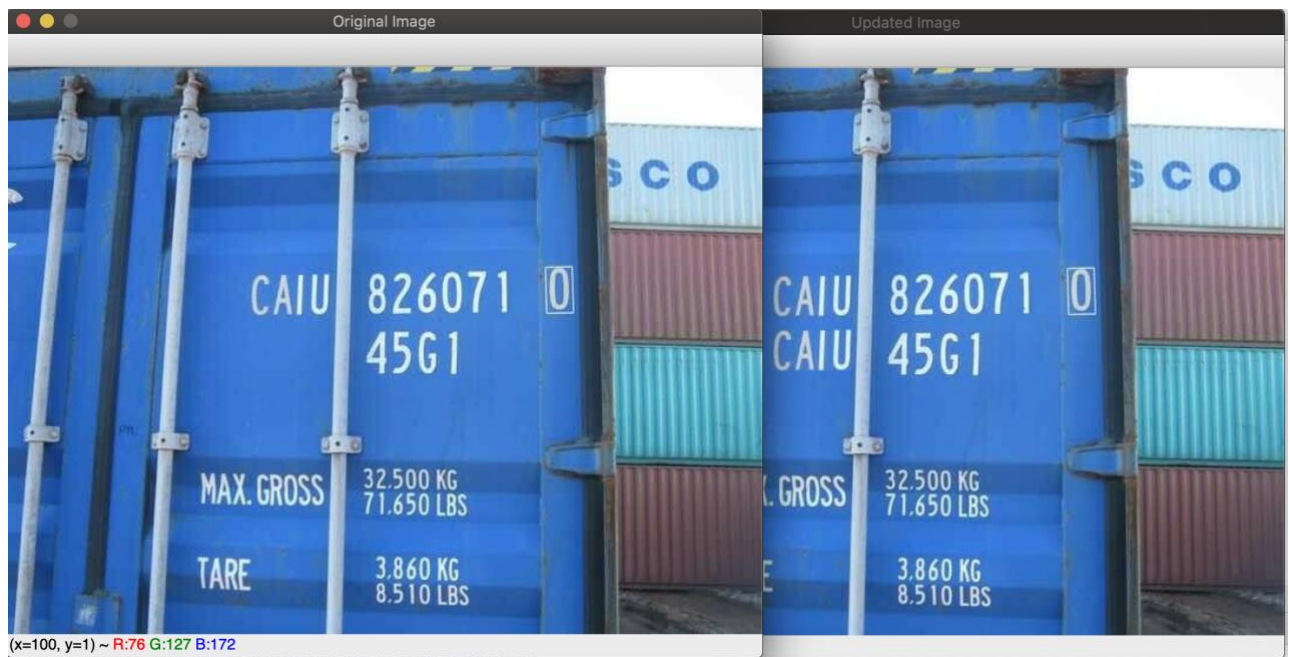
Sometimes, you will have to play with certain region of images. For eye detection in images, first perform face detection over the image until the face is found, then search within the face region for eyes. This approach improves accuracy (because eyes are always on faces) and performance (because we search for a small area).

ROI is again obtained using Numpy indexing. Here I am selecting the ball and copying it to another region in the image:

```
import cv2
img = cv2.imread('exImages-container.jpg', cv2.IMREAD_COLOR)
cv2.imshow("Original Image", img)

label_copy = img[174:210, 202:274]
img[220:256, 202:274] = label_copy
cv2.imshow("Updated Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Check the results below:



Lab 6 Image Processing in OpenCV

Changing Colourspaces

Goal

- In this tutorial, you will learn how to convert images from one color-space to another, like BGR ↔ Gray, BGR ↔ HSV etc.
- In addition to that, we will create an application which extracts a colored object in a video
- You will learn following functions : `cv2.cvtColor()`, `cv2.inRange()` etc.

Changing Color-space

There are more than 150 color-space conversion methods available in OpenCV. But we will look into only two which are most widely used ones, BGR ↔ Gray and BGR ↔ HSV.

For color conversion, we use the

function `cv2.cvtColor(input_image, flag)` where `flag` determines the type of conversion.

For BGR → Gray conversion we use the flags `cv2.COLOR_BGR2GRAY`. Similarly for BGR → HSV, we use the flag `cv2.COLOR_BGR2HSV`. To get other flags, just run following commands in your Python terminal :

```
>>> import cv2
>>> flags = [i for i in dir(cv2) if i.startswith('COLOR_')]
>>> print flags
```

Note

For HSV, Hue range is [0,179], Saturation range is [0,255] and Value range is [0,255]. Different softwares use different scales. So if you are comparing OpenCV values with them, you need to normalize these ranges.

Object Tracking

Now we know how to convert BGR image to HSV, we can use this to extract a colored object. In HSV, it is more easier to represent a color than RGB color-space. In our application, we will try to extract a blue colored object. So here is the method:

- Take each frame of the video
- Convert from BGR to HSV color-space
- We threshold the HSV image for a range of blue color
- Now extract the blue object alone, we can do whatever on that image we want.

Below is the code which are commented in detail :

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while(1):
    # Take each frame
    _, frame = cap.read()

    # Convert BGR to HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # define range of blue color in HSV
    lower_blue = np.array([110,50,50])
    upper_blue = np.array([130,255,255])

    # Threshold the HSV image to get only blue colors
    mask = cv2.inRange(hsv, lower_blue, upper_blue)

    # Bitwise-AND mask and original image
    res = cv2.bitwise_and(frame,frame, mask= mask)

    cv2.imshow('frame',frame)
    cv2.imshow('mask',mask)
    cv2.imshow('res',res)
    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
```

Below image shows tracking of the blue object:



Note There are some noises in the image. We will see how to remove them in later chapters. This is the simplest method in object tracking. Once you learn functions of contours, you can do plenty of things like find centroid of this object and use it to track the object, draw diagrams just by moving your hand in front of camera and many other funny stuffs.

How to find HSV values to track?

This is a common question found in stackoverflow.com. It is very simple and you can use the same function, `cv2.cvtColor()`. Instead of passing an image, you just pass the BGR values you want. For example, to find the HSV value of Green, try following commands in Python terminal:

```
>>> green = np.uint8([[0,255,0 ]])
>>> hsv_green = cv2.cvtColor(green,cv2.COLOR_BGR2HSV)
>>> print hsv_green
[[[ 60 255 255]]]
```

Now you take `[H-10, 100,100]` and `[H+10, 255, 255]` as lower bound and upper bound respectively. Apart from this method, you can use any image editing tools like GIMP or any online converters to find these values, but don't forget to adjust the HSV ranges.

Exercises

Try to find a way to extract more than one colored objects, for eg, extract red, blue, green objects simultaneously.

Image Thresholding

Goal

- In this tutorial, you will learn Simple thresholding, Adaptive thresholding, Otsu's thresholding etc.
- You will learn these functions : `cv2.threshold`, `cv2.adaptiveThreshold` etc.

Simple Thresholding

Here, the matter is straight forward. If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). The

function used is **cv2.threshold**. First argument is the source image, which **should be a grayscale image**. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value. OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function. Different types are:

- cv2.THRESH_BINARY
- cv2.THRESH_BINARY_INV
- cv2.THRESH_TRUNC
- cv2.THRESH_TOZERO
- cv2.THRESH_TOZERO_INV

Documentation clearly explain what each type is meant for. Please check out the documentation.

Two outputs are obtained. First one is a **retval** which will be explained later. Second output is our **thresholded image**.

Code :

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('gradient.png',0)
ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

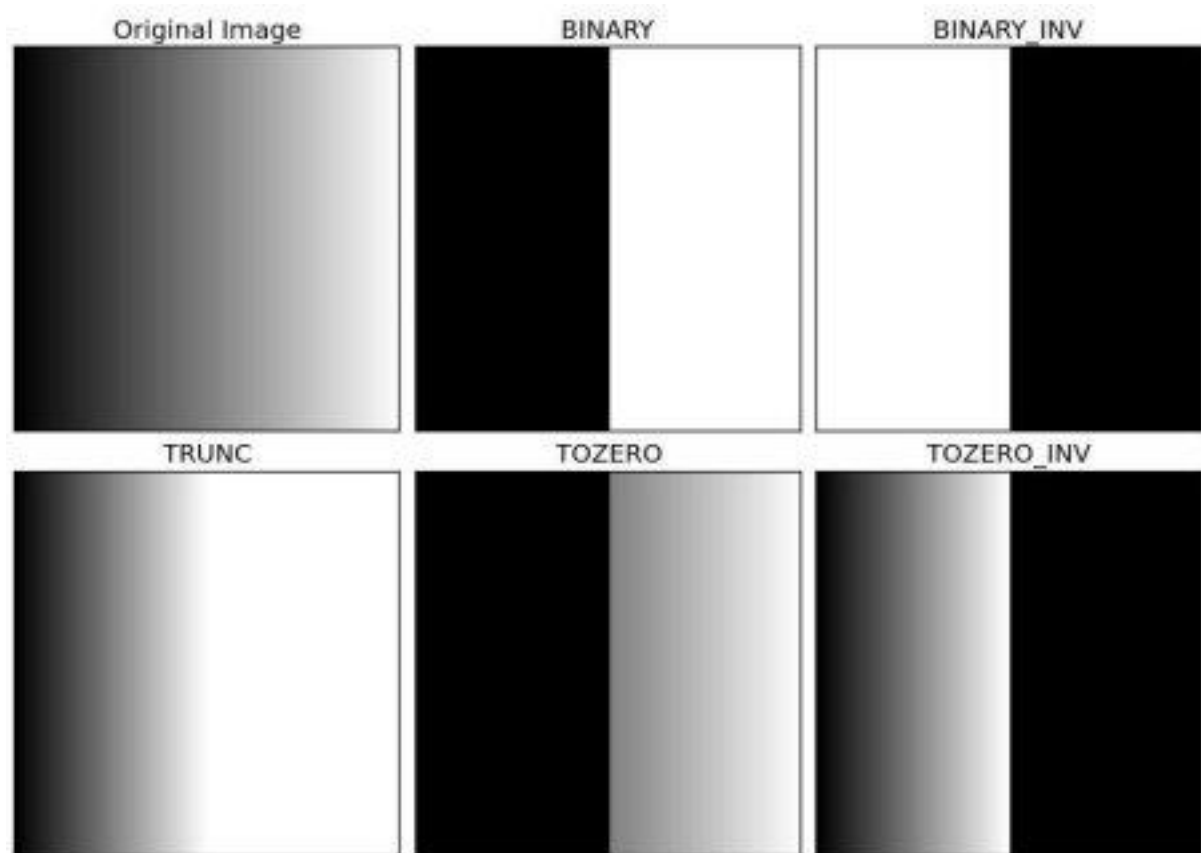
titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in xrange(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([],plt.yticks([]))

plt.show()
```

NOTES: To plot multiple images, we have used *plt.subplot()* function. Please checkout Matplotlib docs for more details.

Result is given below :



Adaptive Thresholding

In the previous section, we used a global value as threshold value. But it may not be good in all the conditions where image has different lighting conditions in different areas. In that case, we go for adaptive thresholding. In this, the algorithm calculate the threshold for a small regions of the image. So we get different thresholds for different regions of the same image and it gives us better results for images with varying illumination.

It has three 'special' input params and only one output argument.

Adaptive Method - It decides how thresholding value is calculated.

- `cv2.ADAPTIVE_THRESH_MEAN_C` : threshold value is the mean of neighbourhood area.
- `cv2.ADAPTIVE_THRESH_GAUSSIAN_C` : threshold value is the weighted sum of neighbourhood values where weights are a gaussian window.

Block Size - It decides the size of neighbourhood area.

C - It is just a constant which is subtracted from the mean or weighted mean calculated.

Below piece of code compares global thresholding and adaptive thresholding for an image with varying illumination:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('dave.jpg',0)
img = cv2.medianBlur(img,5)

ret, th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,\
    cv2.THRESH_BINARY,11,2)
th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
    cv2.THRESH_BINARY,11,2)

titles = ['Original Image', 'Global Thresholding (v = 127)',
    'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
images = [img, th1, th2, th3]

for i in xrange(4):
    plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([],plt.yticks([]))
plt.show()
```

Result

Original Image



Global Thresholding ($v = 127$)



Adaptive Mean Thresholding



Adaptive Gaussian Thresholding



Lab 7 Image Processing in OpenCV 2

Smoothing Images

Goals

Learn to:

- Blur images with various low pass filters
- Apply custom-made filters to images (2D convolution)

2D Convolution (Image Filtering)

As for one-dimensional signals, images also can be filtered with various low-pass filters (LPF), high-pass filters (HPF), etc. A LPF helps in removing noise, or blurring the image. A HPF filters helps in finding edges in an image.

OpenCV provides a function, `cv2.filter2D()`, to convolve a kernel with an image. As an example, we will try an averaging filter on an image. A 5x5 averaging filter kernel can be defined as follows:

$$K = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Filtering with the above kernel results in the following being performed: for each pixel, a 5x5 window is centered on this pixel, all pixels falling within this window are summed up, and the result is then divided by 25. This equates to computing the average of the pixel values inside that window. This operation is performed for all the pixels in the image to produce the output filtered image. Try this code and check the result:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

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

kernel = np.ones((5,5),np.float32)/25
dst = cv2.filter2D(img,-1,kernel)

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([], plt.yticks([]))
plt.subplot(122),plt.imshow(dst),plt.title('Averaging')
plt.xticks([], plt.yticks([]))
plt.show()
```

Result:

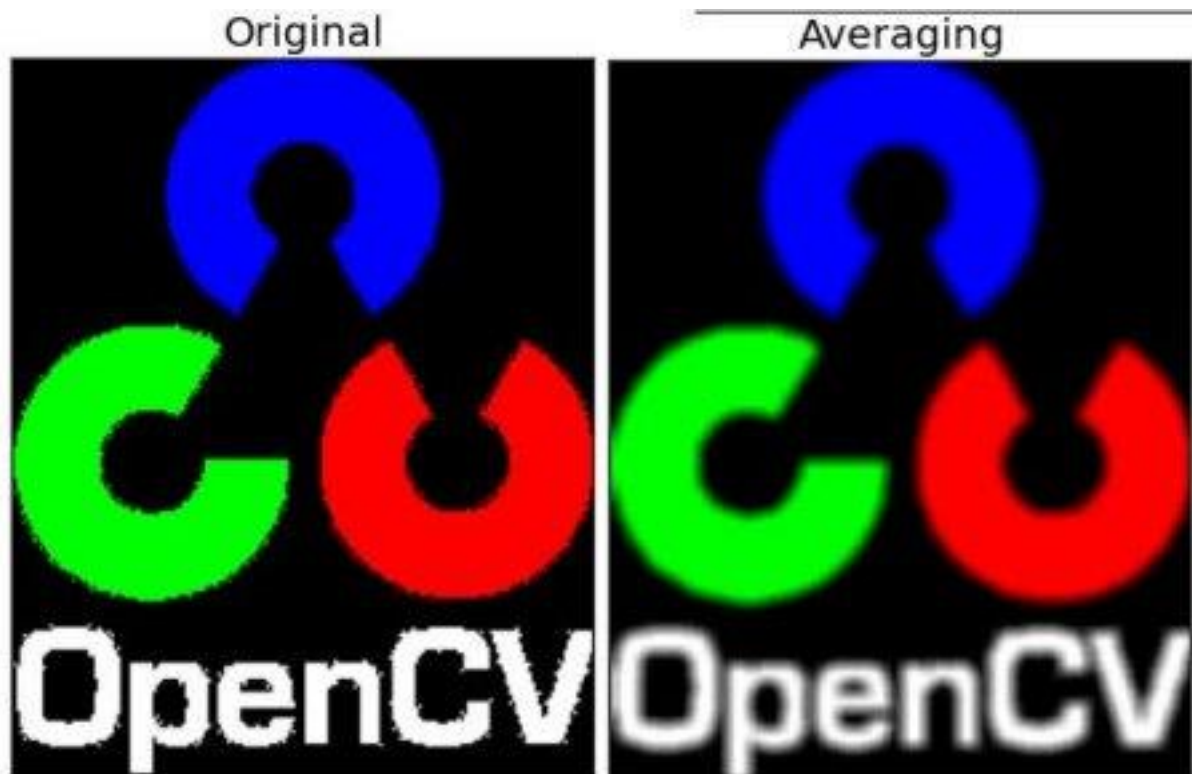


Image Blurring (Image Smoothing)

Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It actually removes high frequency content (e.g: noise, edges) from the image resulting in edges being blurred when this filter is applied. (Well, there are blurring techniques which do not blur edges). OpenCV provides mainly four types of blurring techniques.

1. Averaging

This is done by convolving the image with a normalized box filter. It simply takes the average of all the pixels under kernel area and replaces the central element with this average. This is done by the function `cv2.blur()` or `cv2.boxFilter()`. Check the docs for more details about the kernel. We should specify the width and height of kernel. A 3x3 normalized box filter would look like this:

$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Note

If you don't want to use a normalized box filter, use `cv2.boxFilter()` and pass the argument `normalize=False` to the function.

Check the sample demo below with a kernel of 5x5 size:

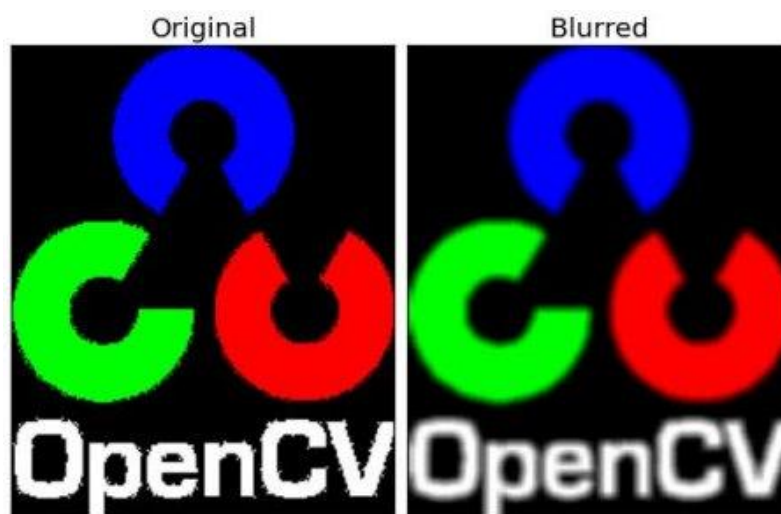
```
import cv2
import numpy as np
from matplotlib import pyplot as plt

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

blur = cv2.blur(img,(5,5))

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([], plt.yticks([]))
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
plt.xticks([], plt.yticks([]))
plt.show()
```

Result:



2. Gaussian Filtering

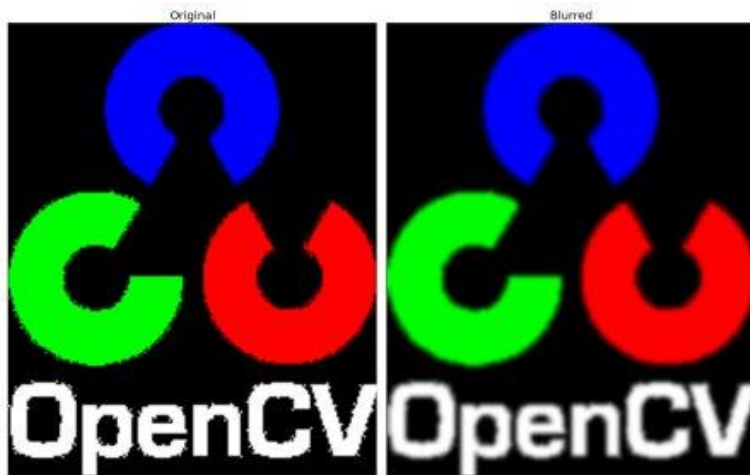
In this approach, instead of a box filter consisting of equal filter coefficients, a Gaussian kernel is used. It is done with the function, `cv2.GaussianBlur()`. We should specify the width and height of the kernel which should be positive and odd. We also should specify the standard deviation in the X and Y directions, `sigmaX` and `sigmaY` respectively. If only `sigmaX` is specified, `sigmaY` is taken as equal to `sigmaX`. If both are given as zeros, they are calculated from the kernel size. Gaussian filtering is highly effective in removing Gaussian noise from the image.

If you want, you can create a Gaussian kernel with the function, `cv2.getGaussianKernel()`.

The above code can be modified for Gaussian blurring:

```
blur = cv2.GaussianBlur(img,(5,5),0)
```

Result:



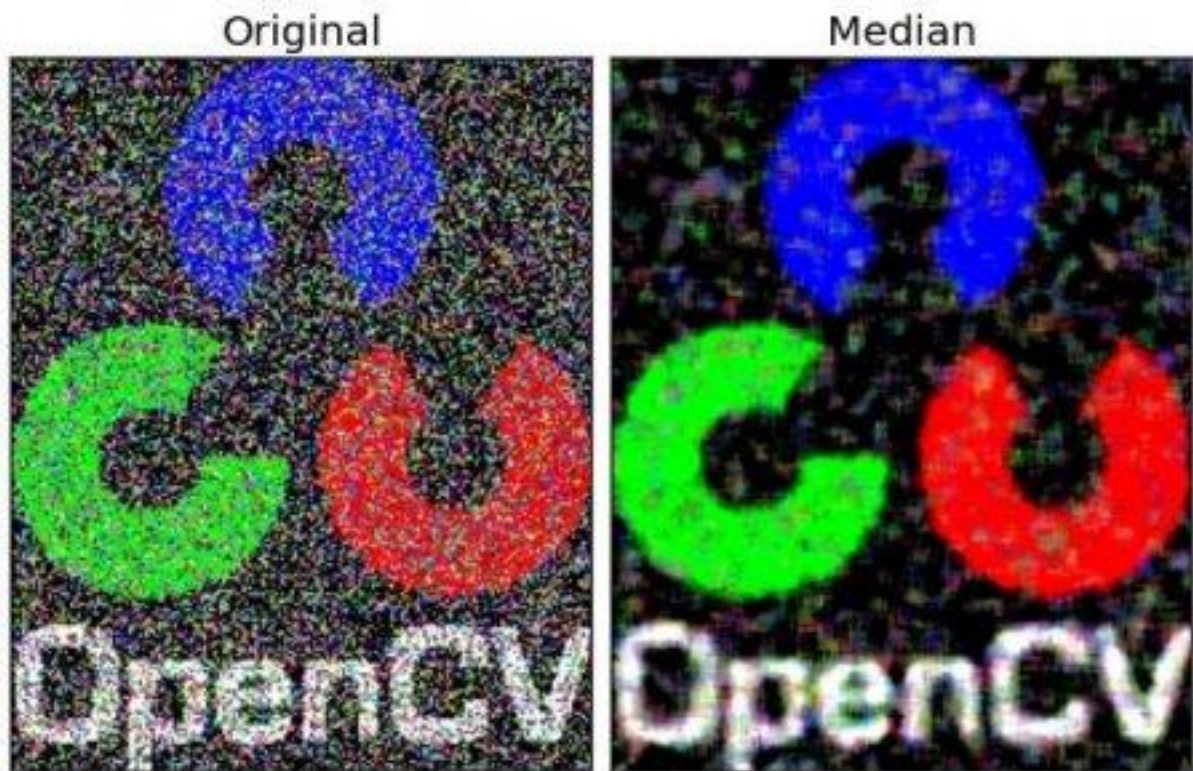
3. Median Filtering

Here, the function `cv2.medianBlur()` computes the median of all the pixels under the kernel window and the central pixel is replaced with this median value. This is highly effective in removing salt-and-pepper noise. One interesting thing to note is that, in the Gaussian and box filters, the filtered value for the central element can be a value which may not exist in the original image. However this is not the case in median filtering, since the central element is always replaced by some pixel value in the image. This reduces the noise effectively. The kernel size must be a positive odd integer.

In this demo, we add a 50% noise to our original image and use a median filter. Check the result:

```
median = cv2.medianBlur(img,5)
```

Result:



4. Bilateral Filtering

As we noted, the filters we presented earlier tend to blur edges. This is not the case for the bilateral filter, `cv2.bilateralFilter()`, which was defined for, and is highly effective at noise removal while preserving edges. But the operation is slower compared to other filters. We already saw that a Gaussian filter takes the a neighborhood around the pixel and finds its Gaussian weighted average. This Gaussian filter is a function of space alone, that is, nearby pixels are considered while filtering. It does not consider whether pixels have almost the same intensity value and does not consider whether the pixel lies on an edge or not. The resulting effect is that Gaussian filters tend to blur edges, which is undesirable.

The bilateral filter also uses a Gaussian filter in the space domain, but it also uses one more (multiplicative) Gaussian filter component which is a function of pixel intensity differences. The Gaussian function of space makes sure that only pixels are ‘spatial neighbors’ are considered for filtering, while the Gaussian component applied in the intensity domain (a Gaussian function of intensity differences) ensures that only those pixels with intensities similar to that of the central pixel (‘intensity neighbors’) are included to compute the blurred intensity value. As a result, this method preserves edges, since for pixels lying near edges, neighboring pixels placed on the other side of the edge, and therefore exhibiting large intensity variations when compared to the central pixel, will not be included for blurring.

The sample below demonstrates the use of bilateral filtering (For details on arguments, see the OpenCV docs).

```
blur = cv2.bilateralFilter(img,9,75,75)
```

Result:



Note that the texture on the surface is gone, but edges are still preserved.

Exercises

Take an image, add Gaussian noise and salt and pepper noise, compare the effect of blurring via box, Gaussian, median and bilateral filters for both noisy images, as you change the level of noise.

Morphological Transformations

Goal

In this chapter,

- We will learn different morphological operations like Erosion, Dilation, Opening, Closing etc.
- We will see different functions like
: `cv2.erode()`, `cv2.dilate()`, `cv2.morphologyEx()` etc.

Theory

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called **structuring element** or **kernel** which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc also comes into play. We will see them one-by-one with help of following image:



1. Erosion

The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white). So what does it do? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

So what happens is that, all the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply

white region decreases in the image. It is useful for removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.

Here, as an example, I would use a 5x5 kernel with full of ones. Let's see it how it works:

```
import cv2
import numpy as np

img = cv2.imread('j.png',0)
kernel = np.ones((5,5),np.uint8)
erosion = cv2.erode(img,kernel,iterations = 1)
```

Result:



2. Dilation

It is just opposite of erosion. Here, a pixel element is '1' if atleast one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

```
dilation = cv2.dilate(img,kernel,iterations = 1)
```

Result:



3. Opening

Opening is just another name of **erosion followed by dilation**. It is useful in removing noise, as we explained above. Here we use the function, `cv2.morphologyEx()`

```
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
```

Result:



4. Closing

Closing is reverse of Opening, Dilation followed by Erosion. It is useful in closing small holes inside the foreground objects, or small black points on the object.

```
closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
```

Result



5. Morphological Gradient

It is the difference between dilation and erosion of an image.

The result will look like the outline of the object.

```
gradient = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)
```

Result



6. Top Hat

It is the difference between input image and Opening of the image. Below example is done for a 9x9 kernel.

```
tophat = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel)
```

Result



7. Black Hat

It is the difference between the closing of the input image and input image.

```
blackhat = cv2.morphologyEx(img, cv2.MORPH_BLACKHAT, kernel)
```

Result



Structuring Element

We manually created a structuring elements in the previous examples with help of Numpy. It is rectangular shape. But in some cases, you may need elliptical/circular shaped kernels. So for this purpose, OpenCV has a function, `cv2.getStructuringElement()`. You just pass the shape and size of the kernel, you get the desired kernel.

```
# Rectangular Kernel
>>> cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]], dtype=uint8)

# Elliptical Kernel
>>> cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
array([[0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0]], dtype=uint8)

# Cross-shaped Kernel
>>> cv2.getStructuringElement(cv2.MORPH_CROSS,(5,5))
array([[0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0]], dtype=uint8)
```

Image Gradients

Goal

In this chapter, we will learn to:

- Find Image gradients, edges etc
- We will see following functions
: `cv2.Sobel()`, `cv2.Scharr()`, `cv2.Laplacian()` etc

Theory

OpenCV provides three types of gradient filters or High-pass filters, Sobel, Scharr and Laplacian. We will see each one of them.

1. Sobel and Scharr Derivatives

Sobel operators is a joint Gaussian smoothing plus differentiation operation, so it is more resistant to noise. You can specify the direction of derivatives to be taken, vertical or horizontal (by the arguments, `yorder` and `xorder` respectively). You can also specify the size of kernel by the argument `ksize`. If `ksize = -1`, a 3x3 Scharr filter is used which gives better results than 3x3 Sobel filter. Please see the docs for kernels used.

2. Laplacian Derivatives

It calculates the Laplacian of the image given by the

relation, $\Delta src = \frac{\partial^2 src}{\partial x^2} + \frac{\partial^2 src}{\partial y^2}$ where each derivative is found using Sobel derivatives. If `ksize = -1`, then following kernel is used for filtering:

$$kernel = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Code

Below code shows all operators in a single diagram. All kernels are of 5x5 size. Depth of output image is passed -1 to get the result in np.uint8 type.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('dave.jpg',0)

laplacian = cv2.Laplacian(img,cv2.CV_64F)
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)

plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([], plt.yticks([]))
plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([], plt.yticks([]))
plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([], plt.yticks([]))
plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([], plt.yticks([]))

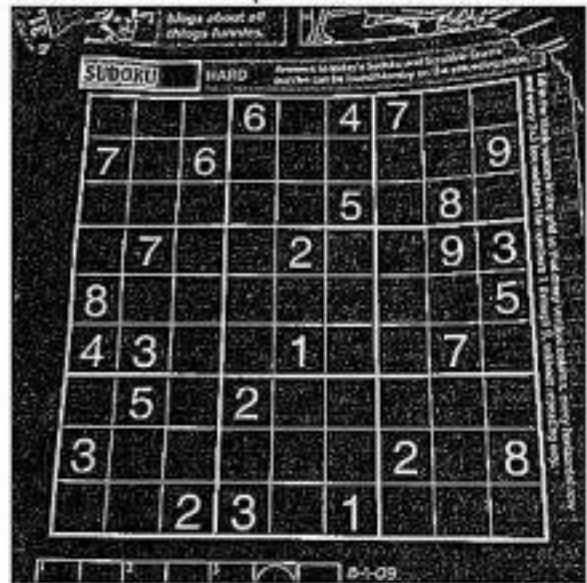
plt.show()
```

Result

Original



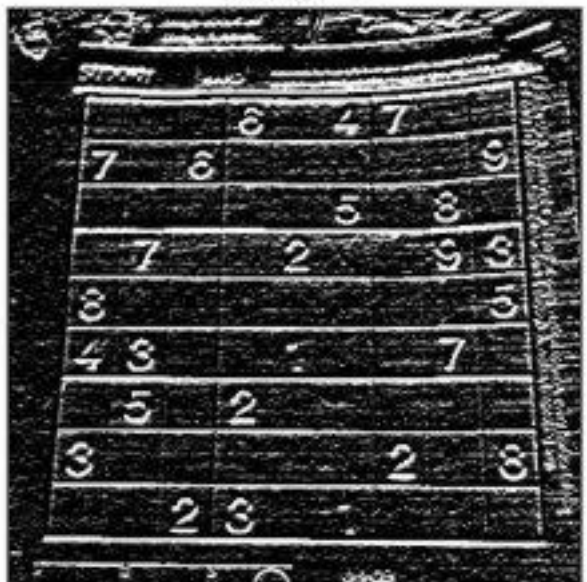
Laplacian



Sobel X



Sobel Y



One Important Matter!

In our last example, output datatype is `cv2.CV_8U` or `np.uint8`. But there is a slight problem with that. Black-to-White transition is taken as Positive slope (it has a positive value) while White-to-Black transition is taken as a Negative slope (It has negative value). So when you convert data to `np.uint8`, all negative slopes are made zero. In simple words, you miss that edge.

If you want to detect both edges, better option is to keep the output datatype to some higher forms, like `cv2.CV_16S`, `cv2.CV_64F` etc, take its absolute value and then convert back

to cv2.CV_8U. Below code demonstrates this procedure for a horizontal Sobel filter and difference in results.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('box.png',0)

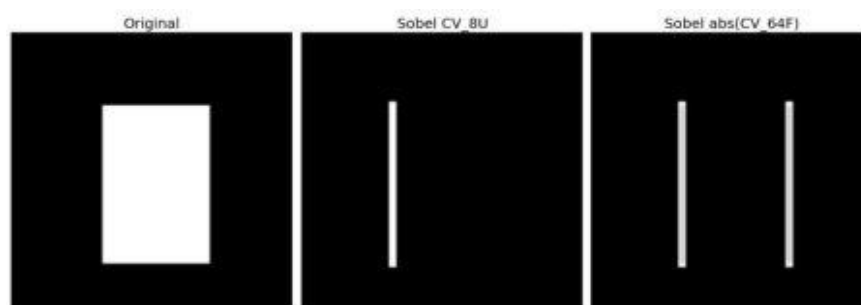
# Output dtype = cv2.CV_8U
sobelx8u = cv2.Sobel(img,cv2.CV_8U,1,0,ksize=5)

# Output dtype = cv2.CV_64F. Then take its absolute and convert to cv2.CV_8U
sobelx64f = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
abs_sobel64f = np.absolute(sobelx64f)
sobel_8u = np.uint8(abs_sobel64f)

plt.subplot(1,3,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(1,3,2),plt.imshow(sobelx8u,cmap = 'gray')
plt.title('Sobel CV_8U'), plt.xticks([]), plt.yticks([])
plt.subplot(1,3,3),plt.imshow(sobel_8u,cmap = 'gray')
plt.title('Sobel abs(CV_64F)'), plt.xticks([]), plt.yticks([])

plt.show()
```

Check the result below:



Canny Edge Detection in OpenCV

Goal

In this chapter, we will learn about

- Concept of Canny edge detection
- OpenCV functions for that : `cv2.Canny()`

Theory

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in 1986. It is a multi-stage algorithm and we will go through each stages.

1. Noise Reduction

Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter. We have already seen this in previous chapters.

2. Finding Intensity Gradient of the Image

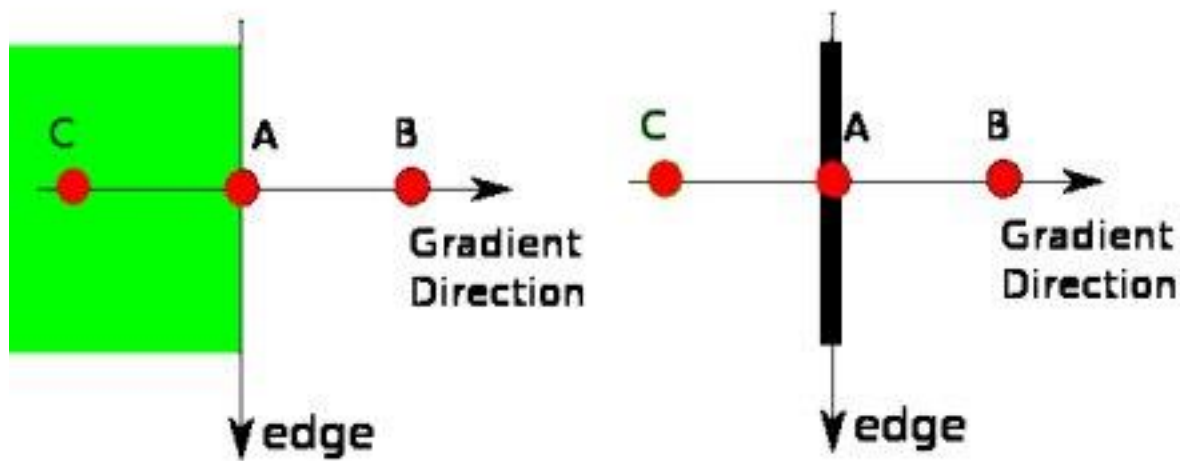
Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (G_x) and vertical direction (G_y). From these two images, we can find edge gradient and direction for each pixel as follows:

$$\text{Edge_Gradient } (G) = \sqrt{G_x^2 + G_y^2}$$
$$\text{Angle } (\theta) = \tan^{-1} \left(\frac{G_y}{G_x} \right)$$

Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

3. Non-maximum Suppression

After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient. Check the image below:

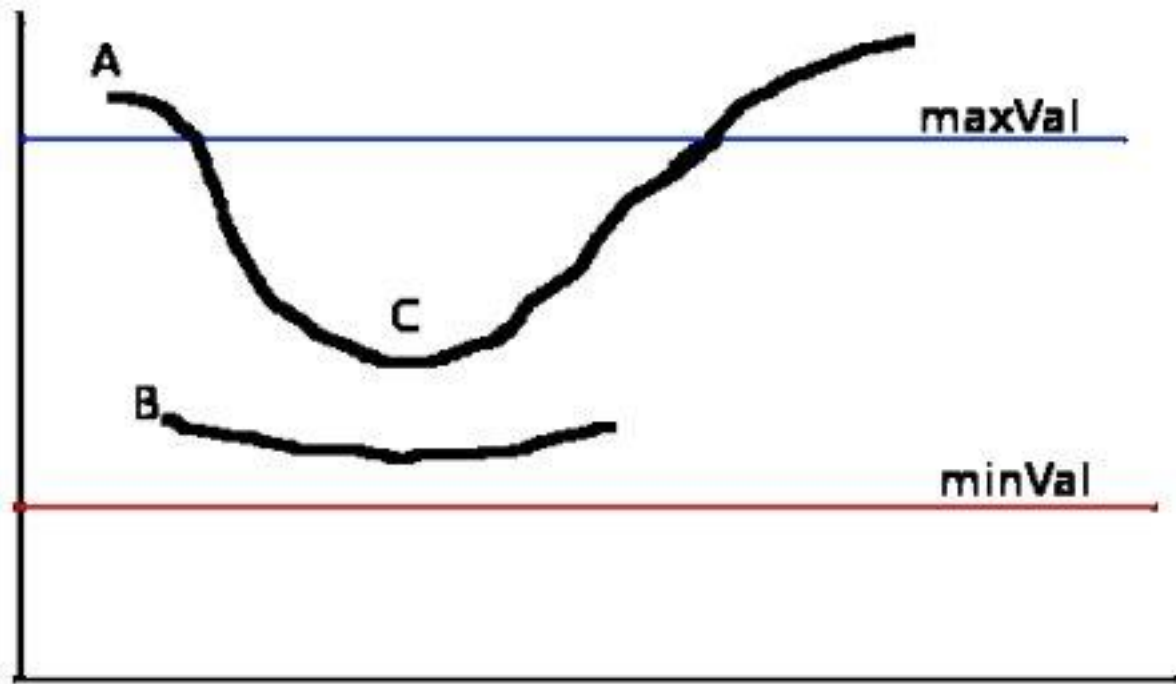


Point A is on the edge (in vertical direction). Gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed (put to zero).

In short, the result you get is a binary image with “thin edges”.

4. Hysteresis Thresholding

This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, *minVal* and *maxVal*. Any edges with intensity gradient more than *maxVal* are sure to be edges and those below *minVal* are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to “sure-edge” pixels, they are considered to be part of edges. Otherwise, they are also discarded. See the image below:



The edge A is above the *maxVal*, so considered as “sure-edge”. Although edge C is below *maxVal*, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above *minVal* and is in same region as that of edge C, it is not connected to any “sure-edge”, so that is discarded. So it is very important that we have to select *minVal* and *maxVal* accordingly to get the correct result.

This stage also removes small pixels noises on the assumption that edges are long lines.

So what we finally get is strong edges in the image.

Canny Edge Detection in OpenCV

OpenCV puts all the above in single function, `cv2.Canny()`. We will see how to use it. First argument is our input image. Second and third arguments are our *minVal* and *maxVal* respectively. Third argument is *aperture_size*. It is the size of Sobel kernel used for find image gradients. By default it is 3. Last argument is *L2gradient* which specifies the equation for finding gradient magnitude. If it is `True`, it uses the equation mentioned above which is more accurate, otherwise it uses this

function: $Edge_Gradient (G) = |G_x| + |G_y|$. By default, it is `False`.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('messi5.jpg',0)
edges = cv2.Canny(img,100,200)

plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])

plt.show()
```

See the result below:



Exercises

Write a small application to find the Canny edge detection whose threshold values can be varied using two trackbars. This way, you can understand the effect of threshold values.

Histograms in OpenCV

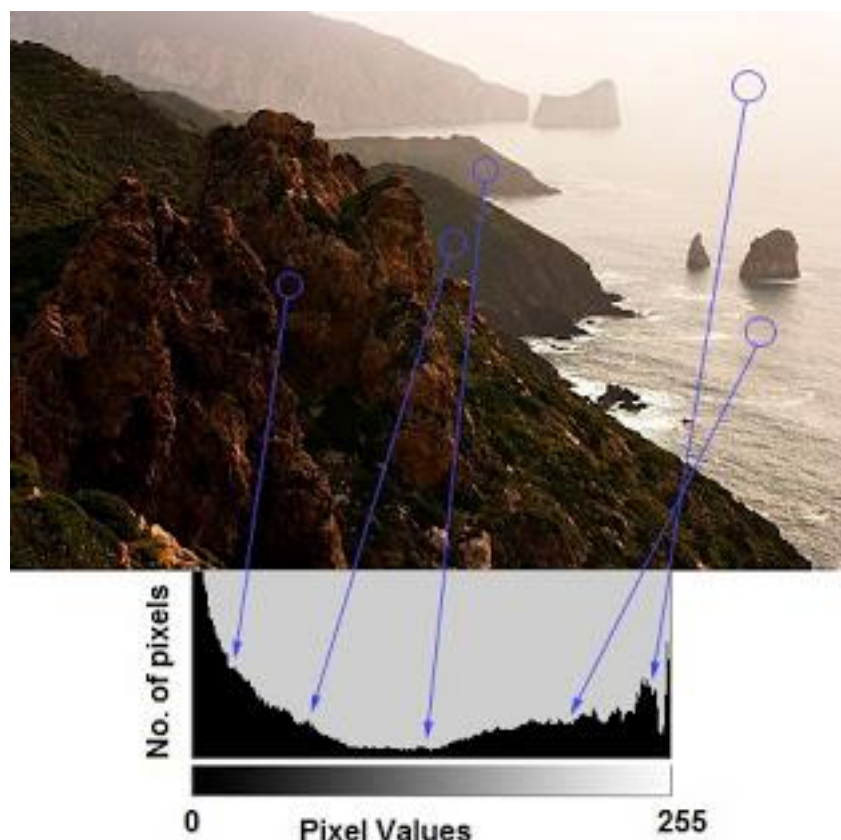
Goal

- Find histograms, using both OpenCV and Numpy functions
- Plot histograms, using OpenCV and Matplotlib functions
- You will see these functions : `cv2.calcHist()`, `np.histogram()` etc.

Theory

So what is histogram ? You can consider histogram as a graph or plot, which gives you an overall idea about the intensity distribution of an image. It is a plot with pixel values (ranging from 0 to 255, not always) in X-axis and corresponding number of pixels in the image on Y-axis.

It is just another way of understanding the image. By looking at the histogram of an image, you get intuition about contrast, brightness, intensity distribution etc of that image. Almost all image processing tools today, provides features on histogram. Below is an image from [Cambridge in Color website](#), and I recommend you to visit the site for more details.



You can see the image and its histogram. (Remember, this histogram is drawn for grayscale image, not color image). Left region of histogram shows the amount of darker pixels in image and right region shows the amount of brighter pixels. From the histogram, you can see dark

region is more than brighter region, and amount of midtones (pixel values in mid-range, say around 127) are very less.

Find Histogram

Now we have an idea on what is histogram, we can look into how to find this. Both OpenCV and Numpy come with in-built function for this. Before using those functions, we need to understand some terminologies related with histograms.

BINS: The above histogram shows the number of pixels for every pixel value, ie from 0 to 255. ie you need 256 values to show the above histogram. But consider, what if you need not find the number of pixels for all pixel values separately, but number of pixels in a interval of pixel values? say for example, you need to find the number of pixels lying between 0 to 15, then 16 to 31, ..., 240 to 255. You will need only 16 values to represent the histogram. And that is what is shown in example given in [OpenCV Tutorials on histograms](#).

So, what you do is simply split the whole histogram to 16 sub-parts and value of each sub-part is the sum of all pixel count in it. This each sub-part is called “BIN”. In first case, number of bins where 256 (one for each pixel) while in second case, it is only 16. BINS is represented by the term **histSize** in OpenCV docs.

DIMS: It is the number of parameters for which we collect the data. In this case, we collect data regarding only one thing, intensity value. So here it is 1.

RANGE: It is the range of intensity values you want to measure. Normally, it is [0,256], ie all intensity values.

1. Histogram Calculation in OpenCV

So now we use **cv2.calcHist()** function to find the histogram. Let's familiarize with the function and its parameters :

cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])

1. **images** : it is the source image of type uint8 or float32. it should be given in square brackets, ie, “[img]”.
2. **channels** : it is also given in square brackets. It the index of channel for which we calculate histogram. For example, if input is grayscale image, its value is [0]. For color

image, you can pass [0],[1] or [2] to calculate histogram of blue,green or red channel respectively.

3. mask : mask image. To find histogram of full image, it is given as “None”. But if you want to find histogram of particular region of image, you have to create a mask image for that and give it as mask. (I will show an example later.)
4. histSize : this represents our BIN count. Need to be given in square brackets. For full scale, we pass [256].
5. ranges : this is our RANGE. Normally, it is [0,256].

So let's start with a sample image. Simply load an image in grayscale mode and find its full histogram.

```
img = cv2.imread('home.jpg',0)
hist = cv2.calcHist([img],[0],None,[256],[0,256])
```

hist is a 256x1 array, each value corresponds to number of pixels in that image with its corresponding pixel value.

Plotting Histograms using Matplotlib

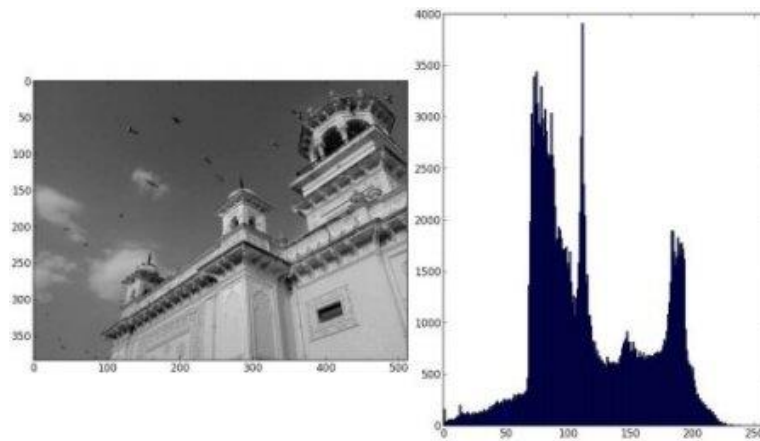
Matplotlib comes with a histogram plotting function: matplotlib.pyplot.hist()

It directly finds the histogram and plot it. You need not use calcHist() or np.histogram() function to find the histogram. See the code below:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('home.jpg',0)
plt.hist(img.ravel(),256,[0,256]); plt.show()
```

You will get a plot as below:

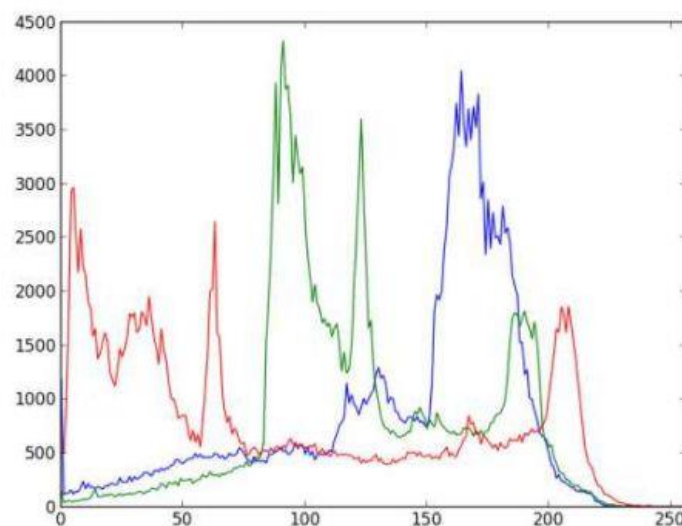


Or you can use normal plot of matplotlib, which would be good for BGR plot. For that, you need to find the histogram data first. Try below code:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('home.jpg')
color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv2.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show()
```

Result



You can deduce from the above graph that, blue has some high value areas in the image (obviously it should be due to the sky)

Lab 8 Template matching and Hough Transform in OpenCV

Template Matching

Goals

- To find objects in an image using Template Matching
- You will see these functions : `cv2.matchTemplate()`, `cv2.minMaxLoc()`

Theory

Template Matching is a method for searching and finding the location of a template image in a larger image. OpenCV comes with a function `cv2.matchTemplate()` for this purpose. It simply slides the template image over the input image (as in 2D convolution) and compares the template and patch of input image under the template image. Several comparison methods are implemented in OpenCV. (You can check docs for more details). It returns a grayscale image, where each pixel denotes how much does the neighbourhood of that pixel match with template.

If input image is of size $(W \times H)$ and template image is of size $(w \times h)$, output image will have a size of $(W-w+1, H-h+1)$. Once you got the result, you can use `cv2.minMaxLoc()` function to find where is the maximum/minimum value. Take it as the top-left corner of rectangle and take (w, h) as width and height of the rectangle. That rectangle is your region of template.

Note! If you are using `cv2.TM_SQDIFF` as comparison method, minimum value gives the best match.

Here, as an example, we will search for Messi's face in his photo. So I created a template as below:



We will try all the comparison methods so that we can see how their results look like:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('messi5.jpg',0)
img2 = img.copy()
```

```

template = cv2.imread('template.jpg',0)
w, h = template.shape[::-1]

# All the 6 methods for comparison in a list
methods = ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR',
           'cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF', 'cv2.TM_SQDIFF_NORMED']

for meth in methods:
    img = img2.copy()
    method = eval(meth)

    # Apply template Matching
    res = cv2.matchTemplate(img,template,method)
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

    # If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum
    if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
        top_left = min_loc
    else:
        top_left = max_loc
    bottom_right = (top_left[0] + w, top_left[1] + h)

    cv2.rectangle(img,top_left, bottom_right, 255, 2)

    plt.subplot(121),plt.imshow(res,cmap = 'gray')
    plt.title('Matching Result'), plt.xticks([], plt.yticks([]))
    plt.subplot(122),plt.imshow(img,cmap = 'gray')
    plt.title('Detected Point'), plt.xticks([], plt.yticks([]))
    plt.suptitle(meth)

plt.show()

```

See the results below:

- cv2.TM_CCOEFF



- cv2.TM_CCOEFF_NORMED

Matching Result



Detected Point



- `cv2.TM_CCORR`

Matching Result



Detected Point



- `cv2.TM_CCORR_NORMED`

Matching Result



Detected Point



- `cv2.TM_SQDIFF`

Matching Result



Detected Point



- `cv2.TM_SQDIFF_NORMED`

Matching Result



Detected Point



You can see that the result using `cv2.TM_CCORR` is not good as we expected.

Hough Line Transform

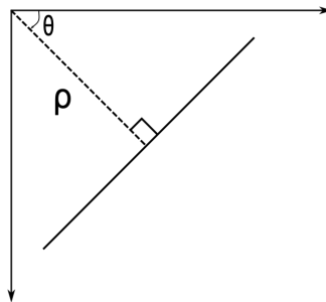
Goal

- We will understand the concept of Hough Transform.
- We will see how to use it to detect lines in an image.
- We will see the following functions: `cv2.HoughLines()`, `cv2.HoughLinesP()`

Theory

Hough Transform is a popular technique to detect any shape, if you can represent that shape in mathematical form. It can detect the shape even if it is broken or distorted a little bit. We will see how it works for a line.

A line can be represented as $y = mx + c$ or in parametric form, as $\rho = x \cos \theta + y \sin \theta$ where ρ is the perpendicular distance from origin to the line, and θ is the angle formed by this perpendicular line and horizontal axis measured in counter-clockwise (That direction varies on how you represent the coordinate system. This representation is used in OpenCV). Check below image:



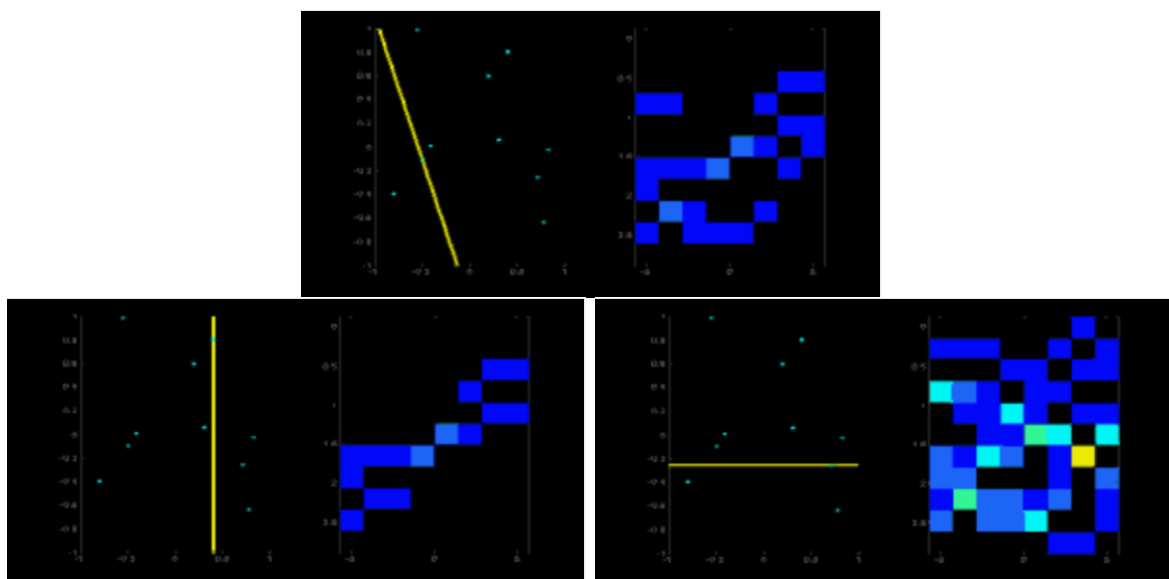
So if a line is passing below the origin, it will have a positive rho and angle less than 180. If it is going above the origin, instead of taking angle greater than 180, angle is taken less than 180, and rho is taken negative. Any vertical line will have 0 degree and horizontal lines will have 90 degree.

Now let's see how Hough Transform works for lines. Any line can be represented in these two terms, (ρ, θ) . So first it creates a 2D array or accumulator (to hold values of two parameters) and it is set to 0 initially. Let rows denote the ρ and columns denote the θ .

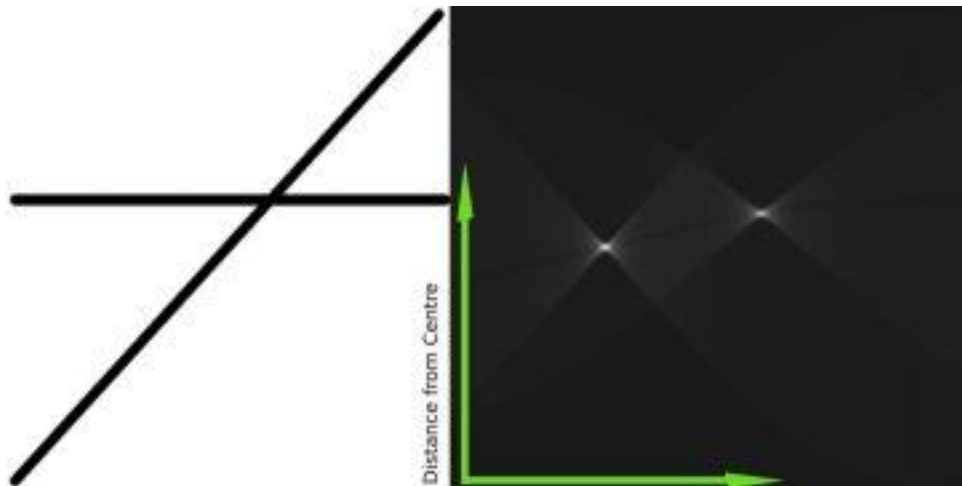
Size of array depends on the accuracy you need. Suppose you want the accuracy of angles to be 1 degree, you need 180 columns. For ρ , the maximum distance possible is the diagonal length of the image. So taking one pixel accuracy, number of rows can be diagonal length of the image.

Consider a 100x100 image with a horizontal line at the middle. Take the first point of the line. You know its (x,y) values. Now in the line equation, put the values $\theta = 0, 1, 2, \dots, 180$ and check the ρ you get. For every (ρ, θ) pair, you increment value by one in our accumulator in its corresponding (ρ, θ) cells. So now in accumulator, the cell $(50,90) = 1$ along with some other cells.

Now take the second point on the line. Do the same as above. Increment the values in the cells corresponding to (ρ, θ) you got. This time, the cell $(50,90) = 2$. What you actually do is voting the (ρ, θ) values. You continue this process for every point on the line. At each point, the cell $(50,90)$ will be incremented or voted up, while other cells may or may not be voted up. This way, at the end, the cell $(50,90)$ will have maximum votes. So if you search the accumulator for maximum votes, you get the value $(50,90)$ which says, there is a line in this image at distance 50 from origin and at angle 90 degrees. It is well shown in below animation (Image Courtesy: [Amos Storkey](#))



This is how hough transform for lines works. It is simple, and may be you can implement it using Numpy on your own. Below is an image which shows the accumulator. Bright spots at some locations denotes they are the parameters of possible lines in the image. (Image courtesy: [Wikipedia](#))



Everything explained above is encapsulated in the OpenCV function, `cv2.HoughLines()`. It simply returns an array of (ρ, θ) values. ρ is measured in pixels and θ is measured in radians. First parameter, Input image should be a binary image, so apply threshold or use canny edge detection before finding applying hough transform. Second and third parameters are ρ and θ accuracies respectively. Fourth argument is the *threshold*, which means minimum vote it should get for it to be considered as a line. Remember, number of votes depend upon number of points on the line. So it represents the minimum length of line that should be detected.

```
import cv2
import numpy as np

img = cv2.imread('dave.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150,apertureSize = 3)

lines = cv2.HoughLines(edges,1,np.pi/180,200)
for rho,theta in lines[0]:
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
```

```

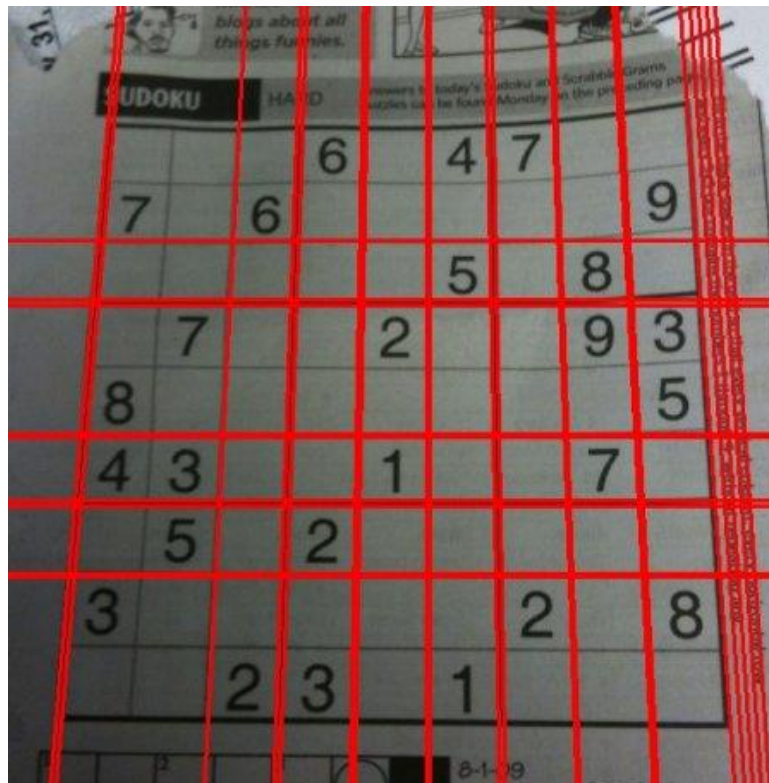
x2 = int(x0 - 1000*(-b))
y2 = int(y0 - 1000*(a))

cv2.line(img, (x1,y1), (x2,y2), (0,0,255), 2)

cv2.imwrite('houghlines3.jpg', img)

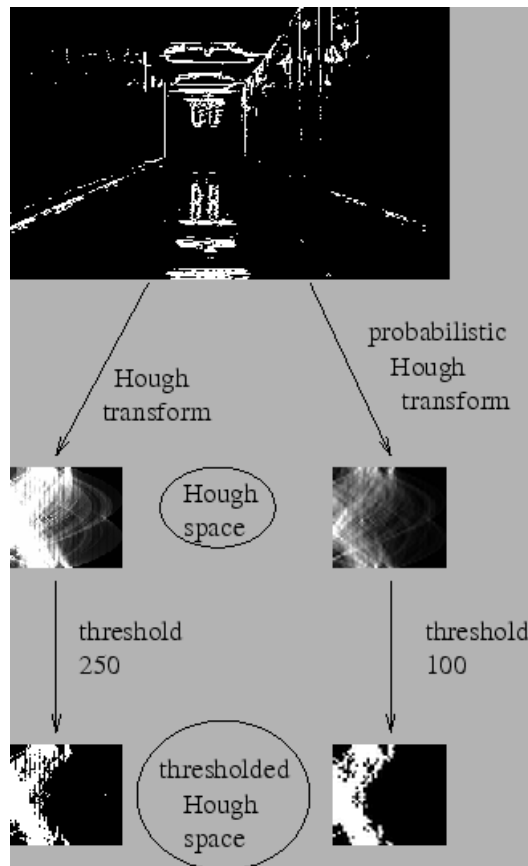
```

Check the results below:



Probabilistic Hough Transform

In the hough transform, you can see that even for a line with two arguments, it takes a lot of computation. Probabilistic Hough Transform is an optimization of Hough Transform we saw. It doesn't take all the points into consideration, instead take only a random subset of points and that is sufficient for line detection. Just we have to decrease the threshold. See below image which compare Hough Transform and Probabilistic Hough Transform in hough space. (Image Courtesy : [Franck Bettinger's home page](#))



OpenCV implementation is based on Robust Detection of Lines Using the Progressive Probabilistic Hough Transform by Matas, J. and Galambos, C. and Kittler, J.V.. The function used is `cv2.HoughLinesP()`. It has two new arguments.

- **minLineLength** - Minimum length of line. Line segments shorter than this are rejected.
- **maxLineGap** - Maximum allowed gap between line segments to treat them as single line.

Best thing is that, it directly returns the two endpoints of lines. In previous case, you got only the parameters of lines, and you had to find all the points. Here, everything is direct and simple.

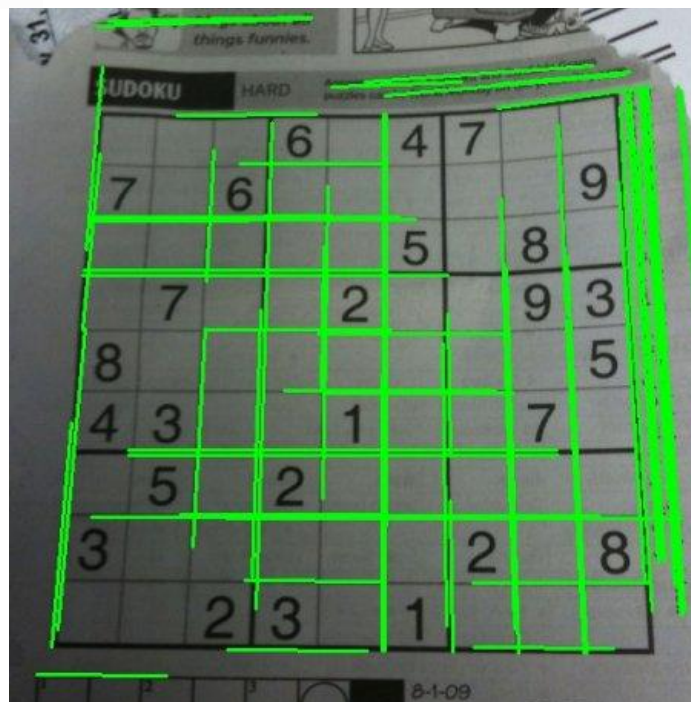
```
import cv2
import numpy as np

img = cv2.imread('dave.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150,apertureSize = 3)
minLineLength = 100
maxLineGap = 10
lines = cv2.HoughLinesP(edges,1,np.pi/180,100,minLineLength,maxLineGap)
for x1,y1,x2,y2 in lines[0]:
    cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)
```



```
cv2.imwrite('houghlines5.jpg',img)
```

See the results below:



Hough Circle Transform

Goal

- We will learn to use Hough Transform to find circles in an image.
- We will see these functions: `cv2.HoughCircles()`

Theory

A circle is represented mathematically as $(x - x_{center})^2 + (y - y_{center})^2 = r^2$ where (x_{center}, y_{center}) is the center of the circle, and r is the radius of the circle. From equation, we can see we have 3 parameters, so we need a 3D accumulator for hough transform, which would be highly ineffective. So OpenCV uses more trickier method, **Hough Gradient Method** which uses the gradient information of edges.

The function we use here is `cv2.HoughCircles()`. It has plenty of arguments which are well explained in the documentation. So we directly go to the code.

```
import cv2
import numpy as np

img = cv2.imread('opencv_logo.png',0)
img = cv2.medianBlur(img,5)
cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)

circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,20,
                           param1=50,param2=30,minRadius=0,maxRadius=0)

circles = np.uint16(np.around(circles))
for i in circles[0,:]:
    # draw the outer circle
    cv2.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
    # draw the center of the circle
    cv2.circle(cimg,(i[0],i[1]),2,(0,0,255),3)

cv2.imshow('detected circles',cimg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Result is shown below:



Lab 9 OCR using Neural net LSTM

Optical character recognition (OCR) refers to the process of electronically extracting text from images (printed or handwritten) or documents in PDF form. This process is also known as text recognition. This lab will teach you how to use OCR with Tesseract OCR Engine.

Tesseract is a tool originally developed by Hewlett Packard between 1985 and 1994, with some changes made in 1996 to port to Windows, and some C++ in 1998. Tesseract became open source by HP in 2005, and Google has been further developing it since 2006.

Currently, Tesseract become version 4 that adds a new neural net Long Short-Term Memory (LSTM) based OCR engine which is focused online recognition. And, the library supports more than 116 languages. Tesseract is also used for text detection on mobile devices, in video, and in Gmail image spam detection.



The repository is kept in **GitHub**. <https://github.com/tesseract-ocr/tesseract>

Tesseract OCR Installation

Regarding the Tesseract OCR is Opensource software, Tesseract is available directly from many Linux distributions such as Ubuntu OS, CentOS and etc. The package is generally called '**tesseract**' or '**tesseract-ocr**' - search your distribution's repositories to find it. Thus you can install Tesseract 4.x and its developer tools on Ubuntu 18.x bionic by simply running:

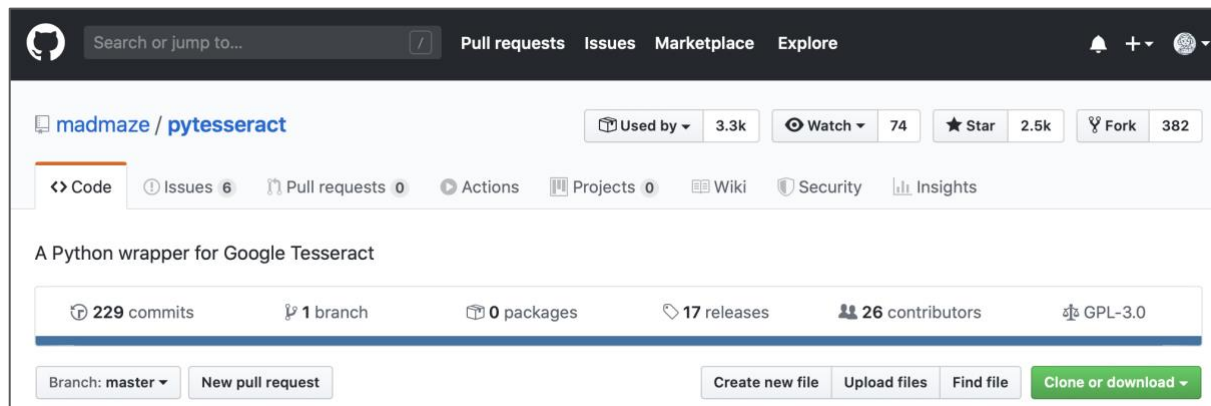
```
Tesseract Installation on Ubuntu 18.x
sudo apt install tesseract-ocr
sudo apt install libtesseract-dev
```

For the other operating system, the instruction can be found in the link below.

<https://github.com/tesseract-ocr/tesseract/wiki#macos>

<https://github.com/tesseract-ocr/tesseract/wiki#windows>

Tesseract Python OpenCV wrapper Installation



Python-tesseract is an optical character recognition (OCR) tool for python. That is, it will recognize and "read" the text embedded in images.

Python-tesseract is a wrapper for [Google's Tesseract-OCR Engine](https://tesseract-ocr.github.io/). It is also useful as a stand-alone invocation script to tesseract, as it can read all image types supported by the Pillow and Leptonica imaging libraries, including jpeg, png, gif, bmp, tiff, and others. Additionally, if used as a script, Python-tesseract will print the recognized text instead of writing it to a file.

We pick this library because Python-tesseract is the famous opensource project that is used by 3,300 projects. The opensource project can be found from Github as below.

<https://github.com/madmaze/pytesseract>

Before using the pytesseract library, it should be installed by "pip".

```
Install pytesseract  
pip install pytesseract
```

Integration of Tesseract OCR and OpenCV Python

Goals

- The students can use Tesseract command line to extract character from Image.
- The students can use integrated OCR Engine with OpenCV to extract character from Image or Camera.

Check Tesseract version

```
Command Line Interface  
tesseract --version
```

We have already tested with Tesseract version 4.1.0.

Manually read character from Image

Command Line Interface

```
tesseract exImages-container_label.jpg result
```

“exImages-container_label.jpg” is image name, and “result” is output from OCR Engine.

After you use tesseract with Command Line Interface, the step is to use OCR Engine from Python OpenCV.

- 1) Import “pytesseract” and “cv2” library

Python script

```
import cv2
import pytesseract
```

- 2) You can check the Tesseract OCR Engine from Python to check the completion of the library by the following scripts.

Python scripts

```
import cv2
import pytesseract
print(pytesseract.get_tesseract_version())
```

Output

```
4.1.0
```

- 3) Read the image from file

Python script

```
import cv2
import pytesseract

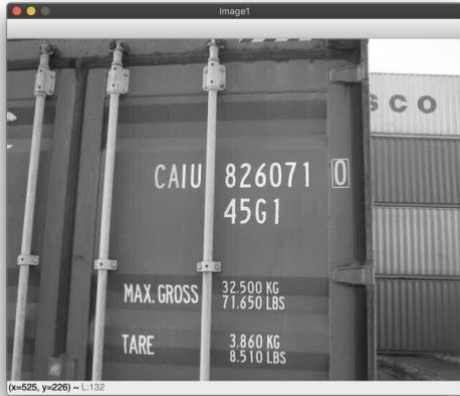
img = cv2.imread('exImages-container.jpg', cv2.IMREAD_GRAYSCALE)
cv2.imshow("Image1", img)
```

- 4) Show image

Python scripts

```
cv2.imshow("Image1 (Original)", img)
```

Output



5) Create ROI

Python script

```
x1=290; y1=150
x2=450; y2=220
img2 = img[y1:y2,x1:x2]
```

6) Show image from ROI

Python scripts

```
cv2.imshow("Image2 (ROI)", img2)
```

Output



7) Call “image_to_string” function

Python scripts

```
print(pyesseract.image_to_string(img2))
```

Output

826071

Complete source code

Python script

```
import cv2
import pytesseract

img = cv2.imread('exImages-container.jpg', cv2.IMREAD_GRAYSCALE)

x1=290; y1=150
x2=450; y2=220
img2 = img[y1:y2,x1:x2]
cv2.imshow("Image1", img)
cv2.imshow("Image2", img2)
print(pytesseract.image_to_string(img2))
```

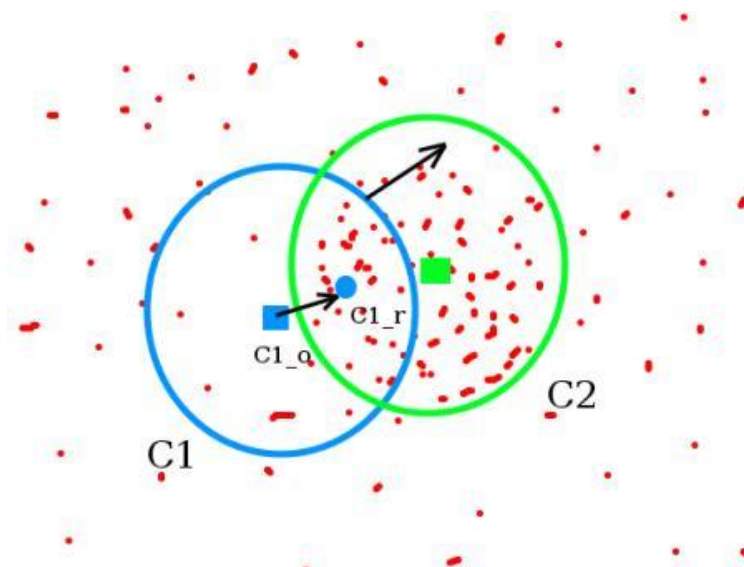
Lab 10 Video analysis

Goal

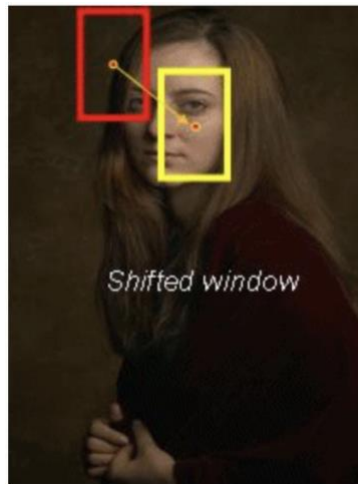
- We will learn about Meanshift and Camshift algorithms to find and track objects in videos.

Meanshift

The intuition behind the meanshift is simple. Consider you have a set of points. (It can be a pixel distribution like histogram backprojection). You are given a small window (may be a circle) and you have to move that window to the area of maximum pixel density (or maximum number of points). It is illustrated in the simple image given below:



The initial window is shown in blue circle with the name “C1”. Its original center is marked in blue rectangle, named “C1_o”. But if you find the centroid of the points inside that window, you will get the point “C1_r” (marked in small blue circle) which is the real centroid of window. Surely they don’t match. So move your window such that circle of the new window matches with previous centroid. Again find the new centroid. Most probably, it won’t match. So move it again, and continue the iterations such that center of window and its centroid falls on the same location (or with a small desired error). So finally what you obtain is a window with maximum pixel distribution. It is marked with green circle, named “C2”. As you can see in image, it has maximum number of points. The whole process is demonstrated on a static image below:



So we normally pass the histogram backprojected image and initial target location. When the object moves, obviously the movement is reflected in histogram backprojected image. As a result, meanshift algorithm moves our window to the new location with maximum density.

Meanshift in OpenCV

To use meanshift in OpenCV, first we need to setup the target, find its histogram so that we can backproject the target on each frame for calculation of meanshift. We also need to provide initial location of window. For histogram, only Hue is considered here. Also, to avoid false values due to low light, low light values are discarded using `cv2.inRange()` function.

Python script

```
# exImages-slow.mp4
import numpy as np
import cv2

cap = cv2.VideoCapture('exImages-slow.mp4')

# take first frame of the video
ret,frame = cap.read()

# setup initial location of window
r,h,c,w = 200,90,220,125 # simply hardcoded the values
track_window = (c,r,w,h)
```

```

# set up the ROI for tracking

roi = frame[r:r+h, c:c+w]
hsv_roi = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi, np.array((0., 60.,32.)), np.array((180.,255.,255.)))
roi_hist = cv2.calcHist([hsv_roi],[0],mask,[180],[0,180])
cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)

# cv2.imshow('img2',roi)

# Setup the termination criteria, either 10 iteration or move by atleast 1 pt
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )

#
while(1):
    ret ,frame = cap.read()

    if ret == True:
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)

        # apply meanshift to get the new location
        ret, track_window = cv2.meanShift(dst, track_window, term_crit)

        # Draw it on image
        x,y,w,h = track_window
        img2 = cv2.rectangle(frame, (x,y), (x+w,y+h), 255,2)
        cv2.imshow("img2",img2)

        k = cv2.waitKey(60) & 0xff
        if k == 27:
            break
        else:
            cv2.imwrite(chr(k)+".jpg",img2)

    else:
        break

# cv2.waitKey(0)
cv2.destroyAllWindows()
cap.release()

```

Three frames in a video I used is given below



Camshift in OpenCV

It is almost same as meanshift, but it returns a rotated rectangle (that is our result) and box parameters (used to be passed as search window in next iteration). See the code below:

Python script

```
import numpy as np
import cv2

cap = cv2.VideoCapture('exImages-slow.mp4')

# take first frame of the video
ret,frame = cap.read()

# setup initial location of window
r,h,c,w = 200,90,220,125 # simply hardcoded the values
track_window = (c,r,w,h)

# set up the ROI for tracking
```

```

roi = frame[r:r+h, c:c+w]
hsv_roi = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi, np.array((0., 60.,32.)), np.array((180.,255.,255.)))
roi_hist = cv2.calcHist([hsv_roi],[0],mask,[180],[0,180])
cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)

# Setup the termination criteria, either 10 iteration or move by atleast 1 pt
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )

while(1):
    ret,frame = cap.read()

    if ret == True:
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)

        # apply meanshift to get the new location
        ret, track_window = cv2.CamShift(dst, track_window, term_crit)

        # Draw it on image
        pts = cv2.boxPoints(ret)
        pts = np.int0(pts)
        img2 = cv2.polylines(frame,[pts],True, 255,2)
        cv2.imshow('img2',img2)

        k = cv2.waitKey(60) & 0xff
        if k == 27:
            break
        else:
            cv2.imwrite(chr(k)+".jpg",img2)

    else:
        break

cv2.destroyAllWindows()
cap.release()

```

Three frames of the result is shown below:



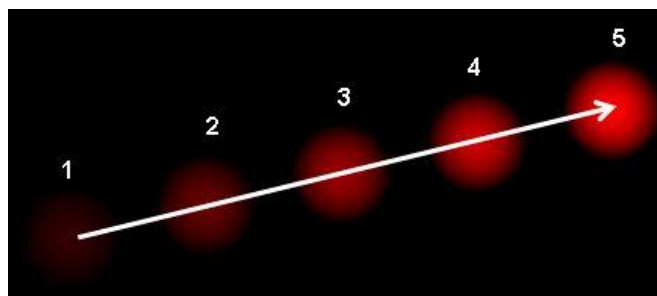
Optical Flow in OpenCV

Goal

- We will understand the concepts of optical flow and its estimation using Lucas-Kanade method.
- We will use functions like `cv2.calcOpticalFlowPyrLK()` to track feature points in a video.

Theory

Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera. It is 2D vector field where each vector is a displacement vector showing the movement of points from first frame to second. Consider the image below (Image Courtesy: [Wikipedia article on Optical Flow](#)).



It shows a ball moving in 5 consecutive frames. The arrow shows its displacement vector.

Optical flow has many applications in areas like :

- Structure from Motion
- Video Compression
- Video Stabilization ...

Optical flow works on several assumptions:

1. The pixel intensities of an object do not change between consecutive frames.
2. Neighbouring pixels have similar motion.

Consider a pixel $I(x, y, t)$ in first frame (Check a new dimension, time, is added here. Earlier we were working with images only, so no need of time). It moves by distance (dx, dy) in next frame taken after dt time. So since those pixels are the same and intensity does not change, we can say,

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

Then take Taylor series approximation of right-hand side, remove common terms and divide by dt to get the following equation:

$$f_x u + f_y v + f_t = 0$$

where:

$$f_x = \frac{\partial f}{\partial x}; f_y = \frac{\partial f}{\partial y}$$

$$u = \frac{dx}{dt}; v = \frac{dy}{dt}$$

Above equation is called Optical Flow equation. In it, we can find f_x and f_y , they are image gradients. Similarly f_t is the gradient along time. But (u, v) is unknown. We cannot solve this one equation with two unknown variables. So several methods are provided to solve this problem and one of them is Lucas-Kanade.

Lucas-Kanade method

We have seen an assumption before, that all the neighbouring pixels will have similar motion. Lucas-Kanade method takes a 3x3 patch around the point. So all the 9 points have the same motion. We can find (f_x, f_y, f_t) for these 9 points. So now our problem becomes solving 9 equations with two unknown variables which is over-determined. A better solution is obtained with least square fit method. Below is the final solution which is two equation-two unknown problem and solve to get the solution.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_{x_i}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i f_{y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix}$$

(Check similarity of inverse matrix with Harris corner detector. It denotes that corners are better points to be tracked.)

So from user point of view, idea is simple, we give some points to track, we receive the optical flow vectors of those points. But again there are some problems. Until now, we were dealing with small motions. So it fails when there is large motion. So again we go for pyramids. When we go up in the pyramid, small motions are removed and large motions becomes small motions. So applying Lucas-Kanade there, we get optical flow along with the scale.

Lucas-Kanade Optical Flow in OpenCV

provides all these in a single function, `cv2.calcOpticalFlowPyrLK()`. Here, we create a simple application which tracks some points in a video. To decide the points, we use `cv2.goodFeaturesToTrack()`. We take the first frame, detect some Shi-Tomasi corner points in it, then we iteratively track those points using Lucas-Kanade optical flow. For the function `cv2.calcOpticalFlowPyrLK()` we pass the previous frame, previous points and next frame. It returns next points along with some status numbers which has a value of 1 if next point is found, else zero. We iteratively pass these next points as previous points in next step. See the code below:

Python script

```
import numpy as np
import cv2
```

```

cap = cv2.VideoCapture('exImages-slow.mp4')

# params for ShiTomasi corner detection
feature_params = dict( maxCorners = 100,
                        qualityLevel = 0.3,
                        minDistance = 7,
                        blockSize = 7 )

# Parameters for lucas kanade optical flow
lk_params = dict( winSize = (15,15),
                  maxLevel = 2,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# Create some random colors
color = np.random.randint(0,255,(100,3))

# Take first frame and find corners in it
ret, old_frame = cap.read()
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(old_gray, mask = None, **feature_params)

# Create a mask image for drawing purposes
mask = np.zeros_like(old_frame)

while(1):
    ret, frame = cap.read()
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # calculate optical flow
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)

    # Select good points
    good_new = p1[st==1]
    good_old = p0[st==1]

    # draw the tracks
    for i,(new,old) in enumerate(zip(good_new,good_old)):
        a,b = new.ravel()

```



```

c,d = old.ravel()
mask = cv2.line(mask, (a,b),(c,d), color[i].tolist(), 2)
frame = cv2.circle(frame,(a,b),5,color[i].tolist(),-1)
img = cv2.add(frame,mask)

cv2.imshow('frame',img)
k = cv2.waitKey(30) & 0xff
if k == 27:
    break

# Now update the previous frame and previous points
old_gray = frame_gray.copy()
p0 = good_new.reshape(-1,1,2)

cv2.destroyAllWindows()
cap.release()

```

(This code doesn't check how correct are the next keypoints. So even if any feature point disappears in image, there is a chance that optical flow finds the next point which may look close to it. So actually for a robust tracking, corner points should be detected in particular intervals. OpenCV samples comes up with such a sample which finds the feature points at every 5 frames. It also run a backward-check of the optical flow points got to select only good ones. Check [samples/python2/lk_track.py](#)).

See the results we got:



