**INTRUDER DETECTION SYSTEM USING PYTHON AND OpenCV**

**Abstract**

Security and surveillance systems play a pivotal role in safeguarding our surroundings, be it homes, commercial establishments, or public spaces. Intruder detection forms the core of such systems, offering real-time protection against unauthorized access and potential threats. This project leverages the power of OpenCV, a popular computer vision library, to develop an Intruder Detection System for security applications. Through a combination of video feed processing, object detection, and motion analysis, this system aims to identify and alert security personnel or system administrators about the presence of intruders.

This abstract highlights the relevance and significance of using OpenCV for intruder detection in security fields, underlining the potential benefits in terms of enhanced security and surveillance.

**EXISTING SYSTEM**

1. **Burglar Alarm Systems:** Traditional burglar alarm systems are widely used in residential and commercial settings. They rely on sensors such as motion detectors, door/window contact sensors, and glass break detectors to trigger alarms when intruders are detected.
2. **Perimeter Security Systems:** These systems are used to protect the perimeters of properties. They include technologies like infrared sensors, laser detectors, and electrified fences to detect intruders attempting to breach a physical boundary.
3. **Access Control Systems:** Access control systems regulate entry into secure areas. They use methods such as key cards, biometrics, and PIN codes to grant access to authorized personnel while denying access to intruders.
4. **Glass Break Detectors**: These devices can detect the sound or vibration of breaking glass, which can be an indicator of a break-in.
5. **Acoustic Intrusion Detection**: Some systems use acoustic sensors to listen for unusual sounds associated with intrusions, such as breaking glass or forced entry.

**PROPOSED SYSTEM**

1. **Video Feed Input**: The system will take input from one or more cameras or video sources. This can include CCTV cameras, webcams, or IP cameras.
2. **Preprocessing**: The video frames will undergo preprocessing to optimize them for analysis. This can include resizing, noise reduction, and conversion to grayscale.
3. **Object Detection**: OpenCV will be used to implement object detection models to recognize and identify objects in the video frames. This could involve pedestrian detection, vehicle detection, or custom object recognition.
4. **Background Subtraction**: To identify moving objects, the system will utilize background subtraction techniques to distinguish between static and moving elements within the frame.
5. **Contour Detection**: The system will detect contours around moving objects, allowing for precise tracking and identification of intruders.
6. **Motion Analysis**: The motion of objects within the frame will be analyzed to differentiate between expected and unexpected movements. Algorithms like optical flow can be used to track object movement.
7. **Intruder Detection Algorithm**: A customized intruder detection algorithm will be implemented to determine if detected movements qualify as an intrusion. This algorithm might take into account factors such as object size, speed, and proximity to restricted areas.
8. **Alert Mechanism**: When an intruder is detected, an alert mechanism will be triggered. This could include sending notifications, activating alarms, or triggering automated responses, such as alerting security personnel or recording video evidence.
9. **Logging and Data Storage**: The system will maintain logs of intruder detection events, including timestamps, images, and video recordings for post-incident analysis.

**SOFTWARE REQUIREMENTS:**

Processor: Pentium IV

Hard Disk: 512GB or more

RAM: 8GB or more

Operating System: Windows 7, 10, 11,Linux

Programming Language: Python, HTML, CSS, Js Bootstrap, Django

IDE/Workbench: Pycharm, visual Studio code

**PYTHON**

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

**Python Features**

Python has few keywords, simple structure, and a clearly defined syntax. Python code is more clearly defined and visible to the eyes. Python's source code is fairly easy-to-maintaining. Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh. Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Portable Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

**Extendable**

It allows to add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

**Databases**

Python provides interfaces to all major commercial databases.

**GUI Programming**

Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

**Scalable**

Python provides a better structure and support for large programs than shell scripting.

**Object-Oriented Approach**

One of the key aspects of Python is its object-oriented approach. This basically means that Python recognizes the concept of class and object encapsulation thus allowing programs to be efficient in the long run.

**Highly Dynamic**

Python is one of the most dynamic languages available in the industry today. There isno need to specify the type of the variable during coding, thus saving time and increasing efficiency.

**Extensive Array of Libraries**

Python comes inbuilt with many libraries that can be imported at any instance and be used in a specific program.

**Open Source and Free**

Python is an open-source programming language which means that anyone can create and contribute to its development. Python is free to download and use in any operating system, like Windows, Mac or Lin.

**MODULES**

A module allows you to logically organize your Python code. Grouping related code into a module makes

the code easier to understand and use. A module is a Python object with arbitrarily named attributes that

you can bind and reference. Simply, a module is a file consisting of Python code. A module can define

functions, classes and variables. A module can also include runnable code.

Example:

The Python code for a module named aname normally resides in a file named aname.py. Here's an

example of a simple module, support.py

def print\_func( par ):

print "Hello : ", par

return

The importStatement

The import has the following syntax:

import module1[, module2[,... moduleN]

When the interpreter encounters an import statement, it imports the module if the module is present in the

search path. A search path is a list of directories that the interpreter searches before importing a module.

For example, to import the module support.py, you need to put the following command at the top of the

script −

A module is loaded only once, regardless of the number of times it is imported. This prevents the module

execution from happening over and over again if multiple imports occur.

**Packages in Python**

A package is a hierarchical file directory structure that defines a single Python application environment

that consists of modules and sub packages and sub-sub packages.

Consider a file Pots.py available in Phone directory. This file has following line of source code −

def Pots():

print "I'm Pots Phone"

Similar way, we have another two files having different functions with the same name as above −

• Phone/Isdn.py file having function Isdn()

• Phone/G3.py file having function G3()

Now, create one more file \_\_init\_\_.py in Phone directory −

• Phone/\_\_init\_\_.py

To make all of your functions available when you've imported Phone,to put explicit import statements in

\_\_init\_\_.py as follows −

from Pots import Pots

from Isdn import Isdn

from G3 import G3

After you add these lines to \_\_init\_\_.py, you have all of these classes available when you import the

Phone package.

# Now import your Phone Package.

import Phone

Phone.Pots()

Phone.Isdn()

Phone.G3()

RESULT:

I'm Pots Phone

I'm 3G Phone

I'm ISDN Phone

In the above example, we have taken example of a single functions in each file, but you can keep multiple

functions in your files. You can also define different Python classes in those files and then you can create

your packages out of those classes.

**PYTHON FILES I/O**

This chapter covers all the basic I/O functions available in Python.

PRINTING TO THE SCREEN

The simplest way to produce output is using the print statement where you can pass zero or more

expressions separated by commas. This function converts the expressions you pass into a string and

writes the result to standard output as follows −

print "Python is really a great language,", "isn't it?"

Result:

Python is really a great language, isn't it?

READING KEYBOARD INPUT

Python provides two built-in functions to read a line of text from standard input, which by default comes

from the keyboard. These functions are −

• raw\_input

• input

Theraw\_inputFunction

The raw\_input([prompt]) function reads one line from standard input and returns it as a string (removing

the trailing newline).

str = raw\_input("Enter your input: ");

print "Received input is : ", str

This prompts you to enter any string and it would display same string on the screen. When I typed "Hello

Python!", its output is like this −

Enter your input: Hello Python

Received input is : Hello Python

22 | P a g e

The input Function

The input([prompt]) function is equivalent to raw\_input, except that it assumes the input is a valid Python

expression and returns the evaluated result to you.

str = input("Enter your input: ");

print "Received input is : ", str

This would produce the following result against the entered input −

Enter your input: [x\*5 for x in range(2,10,2)]

Recieved input is : [10, 20, 30, 40]

**Class and Object**

Python has been an object-oriented language since it existed. Because of this, creating and using classes

and objects are downright easy. This chapter helps you become an expert in using Python's objectoriented programming support.

If you do not have any previous experience with object-oriented (OO) programming, you may want to

consult an introductory course on it or at least a tutorial of some sort so that you have a grasp of the basic

concepts.

However, here is small introduction of Object-Oriented Programming (OOP) to bring you at speed −

Overview of OOP Terminology

• Class: A user-defined prototype for an object that defines a set of attributes that characterize

any object of the class. The attributes are data members (class variables and instance

variables) and methods, accessed via dot notation.

• Class variable: A variable that is shared by all instances of a class. Class variables are defined

within a class but outside any of the class's methods. Class variables are not used as frequently

as instance variables are.

• Data member: A class variable or instance variable that holds data associated with a class

and its objects.

• Function overloading: The assignment of more than one behavior to a particular function.

The operation performed varies by the types of objects or argument

• Instance variable: A variable that is defined inside a method and belongs only to the current

instance of a class.

• Inheritance: The transfer of the characteristics of a class to other classes that are derived from

it.

• Instance: An individual object of a certain class. An object obj that belongs to a class Circle,

for example, is an instance of the class Circle.

• Instantiation: The creation of an instance of a class.

• Method : A special kind of function that is defined in a class definition.

Object: A unique instance of a data structure that's defined by its class. An object comprises

both data members (class variables and instance variables) and methods.

• Operator overloading: The assignment of more than one function to a particular operator.

Creating Classes

The class statement creates a new class definition. The name of the class immediately follows the

keyword class followed by a colon as follows −

class ClassName:

'Optional class documentation string'

class\_suite

• The class has a documentation string, which can be accessed via ClassName.\_\_doc\_\_.

• The class\_suite consists of all the component statements defining class members, data

attributes and functions.

Class Inheritance

Instead of starting from scratch, you can create a class by deriving it from a preexisting class by listing

the parent class in parentheses after the new class name.

The child class inherits the attributes of its parent class, and you can use those attributes as if they were

defined in the child class. A child class can also override data members and methods from the parent.

Syntax

Derived classes are declared much like their parent class; however, a list of base classes to inherit from

is given after the class name −

class SubClassName (ParentClass1[, ParentClass2, ...]):

'Optional class documentation string'

class\_suite

Overriding Methods

You can always override your parent class methods. One reason for overriding parent's methods is

because you may want special or different functionality in your subclass.

Example

class Parent: # define parent class

def myMethod(self):

print 'Calling parent method'

class Child(Parent): # define child class

def myMethod(self):

print 'Calling child method'

c = Child() # instance of child

c.myMethod() # child calls overridden method

When the above code is executed, it produces the following result −

Calling child method

Base Overloading Methods

Following table lists some generic functionality that you can override in your own classes −

SN Method, Description & Sample Call

1 \_\_init\_\_ ( self [,args...] )

Constructor (with any optional arguments)

Sample Call : obj = className(args)

2 \_\_del\_\_( self )

Destructor, deletes an object

Sample Call : del obj

3 \_\_repr\_\_( self )

Evaluatable string representation

Sample Call : repr(obj)

4 \_\_str\_\_( self )

Printable string representation

Sample Call : str(obj)

5 \_\_cmp\_\_ ( self, x )

Object comparison

Sample Call : cmp(obj, x)

Overloading Operators

Suppose you have created a Vector class to represent two-dimensional vectors, what happens when you

use the plus operator to add them? Most likely Python will yell at you.

You could, however, define the \_\_add\_\_ method in your class to perform vector addition and then the

plus operator would behave as per expectation −

class Vector:

def \_\_init\_\_(self, a, b):

self.a = a

self.b = b

def \_\_str\_\_(self):

return 'Vector (%d, %d)' % (self.a, self.b)

def \_\_add\_\_(self,other):

return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)

v2 = Vector(5,-2)

print v1 + v2

When the above code is executed, it produces the following result –

Data Hiding

An object's attributes may or may not be visible outside the class definition. You need to name attributes

with a double underscore prefix, and those attributes then are not be directly visible to outsiders.

lass JustCounter:

\_\_secretCount = 0

def count(self):

self.\_\_secretCount += 1

print self.\_\_secretCount

counter = JustCounter()

counter.count()

counter.count()

print counter.\_\_secretCount

1

2

Traceback (most recent call last):

File "test.py", line 12, in <module>

print counter.\_\_secretCount

AttributeError: JustCounter instance has no attribute '\_\_secretCount'

Python protects those members by internally changing the name to include the class name. You can

access such attributes as object.\_className\_\_attrName. If you would replace your last line as following,

then it works for you –

**NumPy**

”NumPy is a library for the Python programming language, adding support for large, multi- dimensional arrays and matrices, along with a large collection of high-level mathematical func- tions to operate on these arrays”. The previous similar programming of NumPy is Numeric, and this language was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. [12] It is an open source library and free of cost.

**Pandas**

Pandas is also a library or a data analysis tool in python which is written in python program- ming language. It is mostly used for data analysis and data manipulation. It is also used for data structures and time series.

We can see the application of python in many fields such as - Economics, Recommendation Systems - Spotify, Netflix and Amazon, Stock Prediction, Neuro science, Statistics, Advertising, Analytics, Natural Language Processing. Data can be analyzed in pandas in two ways -

Data frames - In this data is two dimensional and consist of multiple series. Data is always represented in rectangular table.

Series - In this data is one dimensional and consist of single list with index.

**Matplotlib**

”Matplotlib is a plotting library for the Python programming language and its numerical math- ematics extension NumPy”[11]. Matlab provides an application that is used in graphical user interface tool kits. Another such libraby is pylab which is almost same as MATLAB.

It is a library for 2D graphics, it finds its application in web application servers, graphical user interface toolkit and shell.

This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

**OpenCV**

OpenCV (Open Source Computer Vision Library) is an open-source library that includes several hundreds of computer vision algorithms. The document describes the so-called OpenCV 2.x API, which is essentially a C++ API, as opposed to the C-based OpenCV 1.x API (C API is deprecated and not tested with "C" compiler since OpenCV 2.4 releases)

OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

* **Core functionality** (**core**) - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
* **Image Processing** (**imgproc**) - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
* **Video Analysis** (**video**) - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
* **Camera Calibration and 3D Reconstruction** (**calib3d**) - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
* **2D Features Framework** (**features2d**) - salient feature detectors, descriptors, and descriptor matchers.
* **Object Detection** (**objdetect**) - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
* **High-level GUI** (**highgui**) - an easy-to-use interface to simple UI capabilities.
* **Video I/O** (**videoio**) - an easy-to-use interface to video capturing and video codecs.
* ... some other helper modules, such as FLANN and Google test wrappers, Python bindings, and others.

The further chapters of the document describe functionality of each module. But first, make sure to get familiar with the common API concepts used thoroughly in the library.

**API Concepts**

**cv Namespace**

All the OpenCV classes and functions are placed into the cv namespace. Therefore, to access this functionality from your code, use the cv:: specifier or using namespace cv; directive:

#include "opencv2/core.hpp"

...

cv::Mat H = cv::findHomography(points1, points2, cv::RANSAC, 5);

...

or :

#include "opencv2/core.hpp"

using namespace [cv](https://docs.opencv.org/4.x/d2/d75/namespacecv.html);

...

Mat H = findHomography(points1, points2, RANSAC, 5 );

...

Some of the current or future OpenCV external names may conflict with STL or other libraries. In this case, use explicit namespace specifiers to resolve the name conflicts:

Mat a(100, 100, CV\_32F);

randu(a, Scalar::all(1), Scalar::all(std::rand()));

cv::log(a, a);

a /= std::log(2.);

**Automatic Memory Management**

OpenCV handles all the memory automatically.

First of all, std::vector, **cv::Mat**, and other data structures used by the functions and methods have destructors that deallocate the underlying memory buffers when needed. This means that the destructors do not always deallocate the buffers as in case of Mat. They take into account possible data sharing. A destructor decrements the reference counter associated with the matrix data buffer. The buffer is deallocated if and only if the reference counter reaches zero, that is, when no other structures refer to the same buffer. Similarly, when a Mat instance is copied, no actual data is really copied. Instead, the reference counter is incremented to memorize that there is another owner of the same data. There is also the **cv::Mat::clone** method that creates a full copy of the matrix data. See the example below:

// create a big 8Mb matrix

Mat A(1000, 1000, CV\_64F);

// create another header for the same matrix;

// this is an instant operation, regardless of the matrix size.

Mat B = A;

// create another header for the 3-rd row of A; no data is copied either

Mat C = B.row(3);

// now create a separate copy of the matrix

Mat D = B.clone();

// copy the 5-th row of B to C, that is, copy the 5-th row of A

// to the 3-rd row of A.

B.row(5).copyTo(C);

// now let A and D share the data; after that the modified version

// of A is still referenced by B and C.

A = D;

// now make B an empty matrix (which references no memory buffers),

// but the modified version of A will still be referenced by C,

// despite that C is just a single row of the original A

B.release();

// finally, make a full copy of C. As a result, the big modified

// matrix will be deallocated, since it is not referenced by anyone

C = C.clone();

You see that the use of Mat and other basic structures is simple. But what about high-level classes or even user data types created without taking automatic memory management into account? For them, OpenCV offers the **cv::Ptr** template class that is similar to std::shared\_ptr from C++11. So, instead of using plain pointers:

T\* ptr = new T(...);

you can use:

Ptr<T> ptr(new T(...));

or:

Ptr<T> ptr = makePtr<T>(...);

Ptr<T> encapsulates a pointer to a T instance and a reference counter associated with the pointer. See the **cv::Ptr** description for details.

**Automatic Allocation of the Output Data**

OpenCV deallocates the memory automatically, as well as automatically allocates the memory for output function parameters most of the time. So, if a function has one or more input arrays (**cv::Mat** instances) and some output arrays, the output arrays are automatically allocated or reallocated. The size and type of the output arrays are determined from the size and type of input arrays. If needed, the functions take extra parameters that help to figure out the output array properties.

Example:

#include "opencv2/imgproc.hpp"

#include "opencv2/highgui.hpp"

using namespace cv;

int main(int, char\*\*)

{

VideoCapture cap(0);

if(!cap.isOpened()) return -1;

Mat frame, edges;

namedWindow("edges", WINDOW\_AUTOSIZE);

for(;;)

{

cap >> frame;

cvtColor(frame, edges, COLOR\_BGR2GRAY);

GaussianBlur(edges, edges, Size(7,7), 1.5, 1.5);

Canny(edges, edges, 0, 30, 3);

imshow("edges", edges);

if(waitKey(30) >= 0) break;

}

return 0;

}

The array frame is automatically allocated by the >> operator since the video frame resolution and the bit-depth is known to the video capturing module. The array edges is automatically allocated by the cvtColor function. It has the same size and the bit-depth as the input array. The number of channels is 1 because the color conversion code **cv::COLOR\_BGR2GRAY** is passed, which means a color to grayscale conversion. Note that frame and edges are allocated only once during the first execution of the loop body since all the next video frames have the same resolution. If you somehow change the video resolution, the arrays are automatically reallocated.

The key component of this technology is the **cv::Mat::create** method. It takes the desired array size and type. If the array already has the specified size and type, the method does nothing. Otherwise, it releases the previously allocated data, if any (this part involves decrementing the reference counter and comparing it with zero), and then allocates a new buffer of the required size. Most functions call the **cv::Mat::create** method for each output array, and so the automatic output data allocation is implemented.

Some notable exceptions from this scheme are **cv::mixChannels**, **cv::RNG::fill**, and a few other functions and methods. They are not able to allocate the output array, so you have to do this in advance.

**Saturation Arithmetics**

As a computer vision library, OpenCV deals a lot with image pixels that are often encoded in a compact, 8- or 16-bit per channel, form and thus have a limited value range. Furthermore, certain operations on images, like color space conversions, brightness/contrast adjustments, sharpening, complex interpolation (bi-cubic, Lanczos) can produce values out of the available range. If you just store the lowest 8 (16) bits of the result, this results in visual artifacts and may affect a further image analysis. To solve this problem, the so-called *saturation* arithmetics is used. For example, to store r, the result of an operation, to an 8-bit image, you find the nearest value within the 0..255 range:

I(x,y)=min(max(round(r),0),255)

Similar rules are applied to 8-bit signed, 16-bit signed and unsigned types. This semantics is used everywhere in the library. In C++ code, it is done using the **cv::saturate\_cast**<> functions that resemble standard C++ cast operations. See below the implementation of the formula provided above:

I.at<uchar>(y, x) = saturate\_cast<uchar>(r);

where **cv::uchar** is an OpenCV 8-bit unsigned integer type. In the optimized SIMD code, such SSE2 instructions as paddusb, packuswb, and so on are used. They help achieve exactly the same behavior as in C++ code.

**Note**

Saturation is not applied when the result is 32-bit integer.

**Fixed Pixel Types. Limited Use of Templates**

Templates is a great feature of C++ that enables implementation of very powerful, efficient and yet safe data structures and algorithms. However, the extensive use of templates may dramatically increase compilation time and code size. Besides, it is difficult to separate an interface and implementation when templates are used exclusively. This could be fine for basic algorithms but not good for computer vision libraries where a single algorithm may span thousands lines of code. Because of this and also to simplify development of bindings for other languages, like Python, Java, Matlab that do not have templates at all or have limited template capabilities, the current OpenCV implementation is based on polymorphism and runtime dispatching over templates. In those places where runtime dispatching would be too slow (like pixel access operators), impossible (generic **cv::Ptr**<> implementation), or just very inconvenient (**cv::saturate\_cast**<>()) the current implementation introduces small template classes, methods, and functions. Anywhere else in the current OpenCV version the use of templates is limited.

Consequently, there is a limited fixed set of primitive data types the library can operate on. That is, array elements should have one of the following types:

* 8-bit unsigned integer (uchar)
* 8-bit signed integer (schar)
* 16-bit unsigned integer (ushort)
* 16-bit signed integer (short)
* 32-bit signed integer (int)
* 32-bit floating-point number (float)
* 64-bit floating-point number (double)
* a tuple of several elements where all elements have the same type (one of the above). An array whose elements are such tuples, are called multi-channel arrays, as opposite to the single-channel arrays, whose elements are scalar values. The maximum possible number of channels is defined by the **CV\_CN\_MAX** constant, which is currently set to 512.

For these basic types, the following enumeration is applied:

enum { CV\_8U=0, CV\_8S=1, CV\_16U=2, CV\_16S=3, CV\_32S=4, CV\_32F=5, CV\_64F=6 };

Multi-channel (n-channel) types can be specified using the following options:

* **CV\_8UC1** ... **CV\_64FC4** constants (for a number of channels from 1 to 4)
* **CV\_8UC(n)** ... **CV\_64FC(n)** or **CV\_MAKETYPE(CV\_8U, n)** ... **CV\_MAKETYPE(CV\_64F, n)** macros when the number of channels is more than 4 or unknown at the compilation time.

**Note**

**CV\_32FC1** == **CV\_32F**, **CV\_32FC2** == **CV\_32FC(2)** == **CV\_MAKETYPE(CV\_32F, 2)**, and **CV\_MAKETYPE(depth, n)** == ((depth&7) + ((n-1)<<3). This means that the constant type is formed from the depth, taking the lowest 3 bits, and the number of channels minus 1, taking the next log2(CV\_CN\_MAX) bits.

Examples:

Mat mtx(3, 3, [CV\_32F](https://docs.opencv.org/4.x/d1/d1b/group__core__hal__interface.html#ga4a3def5d72b74bed31f5f8ab7676099c)); // make a 3x3 floating-point matrix

Mat cmtx(10, 1, [CV\_64FC2](https://docs.opencv.org/4.x/d1/d1b/group__core__hal__interface.html#ga90505db617283cb4ac14f0870ef57021)); // make a 10x1 2-channel floating-point

// matrix (10-element complex vector)

Mat img([Size](https://docs.opencv.org/4.x/dc/d84/group__core__basic.html#ga346f563897249351a34549137c8532a0)(1920, 1080), [CV\_8UC3](https://docs.opencv.org/4.x/d1/d1b/group__core__hal__interface.html#ga88c4cd9de76f678f33928ef1e3f96047)); // make a 3-channel (color) image

// of 1920 columns and 1080 rows.

Mat grayscale(img.size(), [CV\_MAKETYPE](https://docs.opencv.org/4.x/d1/d1b/group__core__hal__interface.html#gab2ebca36079fd923483abee99d7ff40d)(img.depth(), 1)); // make a 1-channel image of

// the same size and same

// channel type as img

Arrays with more complex elements cannot be constructed or processed using OpenCV. Furthermore, each function or method can handle only a subset of all possible array types. Usually, the more complex the algorithm is, the smaller the supported subset of formats is. See below typical examples of such limitations:

* The face detection algorithm only works with 8-bit grayscale or color images.
* Linear algebra functions and most of the machine learning algorithms work with floating-point arrays only.
* Basic functions, such as [**cv::add**](https://docs.opencv.org/4.x/d2/de8/group__core__array.html#ga10ac1bfb180e2cfda1701d06c24fdbd6), support all types.
* Color space conversion functions support 8-bit unsigned, 16-bit unsigned, and 32-bit floating-point types.

The subset of supported types for each function has been defined from practical needs and could be extended in future based on user requests.

**InputArray and OutputArray**

Many OpenCV functions process dense 2-dimensional or multi-dimensional numerical arrays. Usually, such functions take [**cv::Mat**](https://docs.opencv.org/4.x/d3/d63/classcv_1_1Mat.html) as parameters, but in some cases it's more convenient to use std::vector<> (for a point set, for example) or [**cv::Matx**](https://docs.opencv.org/4.x/de/de1/classcv_1_1Matx.html)<> (for 3x3 homography matrix and such). To avoid many duplicates in the API, special "proxy" classes have been introduced. The base "proxy" class is [**cv::InputArray**](https://docs.opencv.org/4.x/dc/d84/group__core__basic.html#ga353a9de602fe76c709e12074a6f362ba). It is used for passing read-only arrays on a function input. The derived from InputArray class [**cv::OutputArray**](https://docs.opencv.org/4.x/dc/d84/group__core__basic.html#gaad17fda1d0f0d1ee069aebb1df2913c0) is used to specify an output array for a function. Normally, you should not care of those intermediate types (and you should not declare variables of those types explicitly) - it will all just work automatically. You can assume that instead of InputArray/OutputArray you can always use [**cv::Mat**](https://docs.opencv.org/4.x/d3/d63/classcv_1_1Mat.html), std::vector<>, [**cv::Matx**](https://docs.opencv.org/4.x/de/de1/classcv_1_1Matx.html)<>, [**cv::Vec**](https://docs.opencv.org/4.x/d6/dcf/classcv_1_1Vec.html)<> or [**cv::Scalar**](https://docs.opencv.org/4.x/dc/d84/group__core__basic.html#ga599fe92e910c027be274233eccad7beb). When a function has an optional input or output array, and you do not have or do not want one, pass [**cv::noArray()**](https://docs.opencv.org/4.x/dc/d84/group__core__basic.html#gad9287b23bba2fed753b36ef561ae7346).

**Error Handling**

OpenCV uses exceptions to signal critical errors. When the input data has a correct format and belongs to the specified value range, but the algorithm cannot succeed for some reason (for example, the optimization algorithm did not converge), it returns a special error code (typically, just a boolean variable).

The exceptions can be instances of the [**cv::Exception**](https://docs.opencv.org/4.x/d1/dee/classcv_1_1Exception.html) class or its derivatives. In its turn, [**cv::Exception**](https://docs.opencv.org/4.x/d1/dee/classcv_1_1Exception.html) is a derivative of std::exception. So it can be gracefully handled in the code using other standard C++ library components.

The exception is typically thrown either using the **[CV\_Error(errcode, description)](https://docs.opencv.org/4.x/db/de0/group__core__utils.html" \l "ga5b48c333c777666e076bd7052799f891" \o "Call the error handler. )** macro, or its printf-like **[CV\_Error\_](https://docs.opencv.org/4.x/db/de0/group__core__utils.html" \l "ga1c0cd6e5bd9a5f915c6cab9c0632f969" \o "Call the error handler. )**(errcode, (printf-spec, printf-args)) variant, or using the **[CV\_Assert(condition)](https://docs.opencv.org/4.x/db/de0/group__core__utils.html" \l "gaf62bcd90f70e275191ab95136d85906b" \o "Checks a condition at runtime and throws exception if it fails. )** macro that checks the condition and throws an exception when it is not satisfied. For performance-critical code, there is **[CV\_DbgAssert(condition)](https://docs.opencv.org/4.x/db/de0/group__core__utils.html" \l "gafbcb487cba05bd288dbe18c433de4f6f)** that is only retained in the Debug configuration. Due to the automatic memory management, all the intermediate buffers are automatically deallocated in case of a sudden error. You only need to add a try statement to catch exceptions, if needed:

try

{

... // call OpenCV

}

catch (const [cv::Exception](https://docs.opencv.org/4.x/d1/dee/classcv_1_1Exception.html)& e)

{

const char\* err\_msg = e.[what](https://docs.opencv.org/4.x/d1/dee/classcv_1_1Exception.html#a9c1e692401016807255e0e6ad562ece9)();

std::cout << "exception caught: " << err\_msg << std::endl;

}

**Multi-threading and Re-enterability**

The current OpenCV implementation is fully re-enterable. That is, the same function or the same methods of different class instances can be called from different threads. Also, the same Mat can be used in different threads because the reference-counting operations use the architecture-specific atomic instructions.

Installation

If you already installed NumPy and Scipy, following are the two easiest ways to install scikit-learn −

Using pip

Following command can be used to install scikit-learn via pip −

pip install -U scikit-learn

Using conda

Following command can be used to install scikit-learn via conda −

conda install scikit-learn

On the other hand, if NumPy and Scipy is not yet installed on your Python workstation then, you can install them by using either pip or conda.

Another option to use scikit-learn is to use Python distributions like Canopy and Anaconda because they both ship the latest version of scikit-learn.

Features

Rather than focusing on loading, manipulating and summarising data, Scikit-learn library is focused on modeling the data. Some of the most popular groups of models provided by Sklearn are as follows −

Supervised Learning algorithms − Almost all the popular supervised learning algorithms, like Linear Regression, Support Vector Machine (SVM), Decision Tree etc., are the part of scikit-learn.

Unsupervised Learning algorithms − On the other hand, it also has all the popular unsupervised learning algorithms from clustering, factor analysis, PCA (Principal Component Analysis) to unsupervised neural networks.

Clustering − This model is used for grouping unlabeled data.

Cross Validation − It is used to check the accuracy of supervised models on unseen data.

Dimensionality Reduction − It is used for reducing the number of attributes in data which can be further used for summarisation, visualisation and feature selection.

Ensemble methods − As name suggest, it is used for combining the predictions of multiple supervised models.

Feature extraction − It is used to extract the features from data to define the attributes in image and text data.

Dataset Loading

A collection of data is called dataset. It is having the following two components −

Features − The variables of data are called its features. They are also known as predictors, inputs or attributes.

Feature matrix − It is the collection of features, in case there are more than one.

Feature Names − It is the list of all the names of the features.

Response − It is the output variable that basically depends upon the feature variables. They are also known as target, label or output.

Response Vector − It is used to represent response column. Generally, we have just one response column.

Target Names − It represent the possible values taken by a response vector.

Scikit-learn have few example datasets like iris and digits for classification and the Boston house prices for regression.

**SYSTEM DESIGN**

The degree of interest in each concept has varied over the year, each has stood the test of time. Each provides the software designer with a foundation from which more sophisticated design methods can be applied. Fundamental design concepts provide the necessary framework for “getting it right”. During the design process the software requirements model is transformed into design models that describe the details of the data structures, system architecture, interface, and components. Each design product is reviewed for quality before moving to the next phase of software development.

**INPUT DESIGN**

The design of input focus on controlling the amount of dataset as input required, avoiding delay and keeping the process simple. The input is designed in such a way to provide security. Input design will consider the following steps:

1. The dataset should be given as input.
2. The dataset should be arranged.
3. Methods for preparing input validations.

**OUTPUT DESIGN**

A quality output is one, which meets the requirement of the user and presents the information clearly. In output design, it is determined how the information is to be displayed for immediate need. Designing computer output should proceed in an organized, well thought out manner the right output must be developed while ensuring that each output element is designed so that the user will find the system can be used easily and effectively.

**DATASET DESIGN**

This phase contains the attributes of the dataset which are maintained in the database table. The dataset collection can be of two types namely train dataset and test dataset.

**FEASIBILITY STUDY**

A feasibility analysis is used to determine the viability of an idea, such as ensuring a project is legally and technically feasible as well as economically justifiable. Feasibility study lets the developer to foresee the project and the usefulness of the system proposal as per its workability. It impacts the organization, ability to meet the user needs and effective use of resource. Thus, when a new application is proposed it normally goes through a feasibility study before it is approved for development.

Three key consideration involved in the feasibility analysis are,

1. TECHNICAL FEASIBILITY
2. OPERATIONAL FEASIBILITY
3. ECONOMIC FEASIBILITY

TECHNICAL FEASIBILITY

This phase focuses on the technical resources available to the organization. It helps organizations determine whether the technical resources meet capacity and whether the ideas can be converted into working system model. Technical feasibility also involves the evaluation of the hardware, software, and other technical requirements of

the proposed system.

OPERATIONAL FEASIBILITY

This phase involves undertaking a study to analyse and determine how well the organization’s needs can be met by completing the project. Operational feasibility study also examines how a project plan satisfies the requirements that are needed for the phase of system development.

ECONOMIC FEASIBILITY

This phase typically involves a cost benefits analysis of the project and help the organization to determine the viability, cost-benefits associated with a project before financial resources are allocated. It also serves as an independent project assessment and enhances project credibility. It helps the decision-makers to determine the positive economic benefits of the organization that the proposed project will provide

**SYSTEM TESTING**

System testing is the stage of implementation that is aimed at ensuring that the system works accurately and efficiently before live operation commences. Testing is vital to the success of the system. System testing makes logical assumption that if all the parts of the system are correct, then the goal will be successfully achieved. System testing involves user training system testing and successful running of the developed proposed system. The user tests the developed system and changes are made per their needs. The testing phase involves the testing of developed system using various kinds of data. While testing, errors are noted and the corrections are made. The corrections are also noted for the future use.

**UNIT TESTING**

Unit testing focuses verification effort on the smallest unit of software design, software component or module. Using the component level design description as a control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and the errors those uncover is limited by the constrained scope established for unit testing. The unit test focuses on the internal processing logic and data structures within the boundaries of a component. This is normally considered as an adjunct to the coding step. The design of unit tests can be performed before coding begins.

**BLACK BOX TESTING**

Black box testing also called behavioural testing, focuses on the functional requirement of the software. This testing enables to derive set of input conditions of all functional requirements for a program. This technique focuses on the information domain of the software, deriving test cases by partitioning the input and output of a program.

**WHITE BOX TESTING**

White box testing also called as glass box testing, is a test case design that uses the control structures described as part of component level design to derive test cases. This test case is derived to ensure all statements in the program have been executed at least once during the testing and that all logical conditions have been exercised.

**INTEGRATION TESTING**

Integration testing is a systematic technique for constructing the software architecture to conduct errors associated with interfacing. Top-down integration testing is an incremental approach to construction of the software architecture. Modules are integrated by movingdownward through the control hierarchy, beginning with the main control module. Bottom-up integration testing begins the construction and testing with atomic modules. Because components are integrated from the bottom up,

processing required for components subordinate to a given level is always available.

**VALIDATION TESTING**

Validation testing begins at the culmination of integration testing, when individual components have been exercised, the software is completely assembled as a package. The testing focuses on user visible actions and user recognizable output from the system. The testing has been conducted on possible condition such as the function characteristic conforms the specification and a deviation or error is uncovered. The alpha test and beta test is conducted at the developer site by end-users.

**CONCLUSION**

In conclusion, "Intruder Detection in Security Fields Using OpenCV" represents a powerful and versatile approach to enhancing security and surveillance in various domains. OpenCV, with its extensive capabilities in computer vision, image processing, and object detection, provides a robust foundation for the development of effective intruder detection systems. This technology can significantly contribute to maintaining the safety and security of both public and private spaces.

The proposed system leverages OpenCV to capture and process video feeds, detect objects and movements, and implement a customized intruder detection algorithm. It offers real-time monitoring and alerts to security personnel, thereby ensuring a prompt response to potential security threats.

The benefits of such a system extend to applications in residential, commercial, industrial, and public settings. Its adaptability to specific security requirements and the potential for customization make it a valuable tool for safeguarding assets and protecting individuals.

As technology continues to advance, the integration of OpenCV in security fields exemplifies the innovative solutions that can be created to address modern security challenges. However, it's crucial to consider the ethical and privacy implications of such systems and implement them responsibly.

In summary, "Intruder Detection in Security Fields Using OpenCV" is a forward-looking approach that aligns with the growing need for intelligent and efficient security systems, and it underscores the importance of leveraging computer vision technology to enhance security across diverse environments.