



Graduation project



Team Work

Elsaeed Reda Elzeiny
Ziad Mahmoud Herat
Abdel-Rahman Feter
Abdel-Rahman Samlk
Rawda Gamal Saleh

Mayada Farrag
Nada Abo A'lya
Lamiaa Shahin
Aya Tariq

Supervisor

DR- AbdullFattah Al-Adl

January 2019

Index

Chapter (1) Introduction: an overview to the project

(Smart glasses for blind .. I See / Blinds guide) 4

Chapter (2) Software and programming language .

Introduction 12

Python programming language 13

Libraries 16

Chapter (3) Optical Character Recognition

Introduction 20

Applications and types 22

How does OCR work 23

Code 24

Chapter (4) Bill Detection

Introduction (SIFT Transform) 27

SIFT in OpenCV 31

Bill Detection Code 32

Chapter (6) Face Recognition:



Chapter 1

Introduction



Chapter 1

Introduction

The term "technology" generally refers to the means and devices used by man to guide life. Information technology is the search for the best means of providing information and exchanging information and facilitating its rapid and effective access to its clients And the information revolution.

Its importance is due to its central role in achieving the objectives of the libraries and information centers and learning resources in facilitating the availability and access to accurate and rapid information for the general public, especially those with visual disabilities. Equal to the opportunities available to the visible beneficiary groups, and this change has been expressed in the phrase "Assistive Technology for the Visually Impaired". Assistive Technology "This umbrella includes aids, adaptation and rehabilitation for people with visual disabilities, where AT enhances the availability and independence of visually impaired people by performing tasks they were previously unable to perform.



This new technology has become the keys for visually impaired people to allow knowledge and free circulation of information and resources. It has enabled them to achieve tasks and meet scientific needs. It has become a distant dream especially in the field of scientific research and self-learning. One million people around the world are visually impaired, of whom 45 million are completely blind. Problems are exacerbated when 87% of this total is found in developing countries and women and persons over 50 years of age are most vulnerable to this disability (Friend, C. 2009). These increased

The percentage of Arab countries in the Arab Spring revolutions in Tunisia and Egypt has increased significantly since the beginning of the Arab spring revolutions in Egypt. Many citizens have suffered severe eye injuries and have faced many problems in accessing their information needs for scientific research purposes.



1.1. Daily Life Problems Faced by Blind People

A few problems that blind people face in their day-to-day lives. Many of the assumptions that we make about blind people are really not problems — and then there are many other things that we do not imagine as problems — but they are! Let's find out!

Blindness is one of the most, if not the most, misunderstood type of disability. The general masses have their own pre-conceived notions about the blind people that they firmly believe to be true without even getting in touch with a blind person. Most of the members of the non-blind community believe that the blind people cannot do their work or live a normal life. 'My Son will not be a Beggar be' by Ved Mehta is a perfect example of the contradiction of society's perspective and the reality of a blind person's life.

Blind people do lead a NORMAL LIFE with their own style of doing things. But, they definitely face troubles due to inaccessible infrastructure and social challenges. Let us have an empathetic look at some of the daily life problems faced by the blind people.

1.1.1. Navigating Around the Places

The biggest challenge for a blind person, especially the one with the complete loss of vision, is to navigate around places. Obviously, blind people roam easily around their house without any help because they know the position of everything in the house. People living with and visiting blind people must make sure not to move things around without informing or asking the blind person.

Commercial places can be made easily accessible for the blinds with tactile tiles. But, unfortunately, this is not done in most of the places. This creates a big problem for blind people who might want to visit the place.



1.1.2. Finding Reading Materials

Blind people have a tough time finding good reading materials in accessible formats. Millions of people in India are blind but we do not have even the proper textbooks in braille, leave alone the novels and other leisure reading materials. Internet, the treasure trove of information and reading materials, too is mostly inaccessible for the blind people. Even though a blind person can use screen reading software but it does not make the surfing experience very smooth if the sites are not designed accordingly. Blind person depends on the image description for understanding whatever is represented through pictures. But most of the time, websites do not provide clear image description.

1.1.3. Arranging Clothes

As most of the blind people depend on the objects' shape and texture, arranging the laundry becomes a challenging task. Although a majority of blind people device their own technique to recognize and arrange at least their own clothes but it still is a challenging chore. This becomes a daredevil task if it's about pairing and arranging the socks. All this is because recognizing colors is almost impossible for the persons with total blindness.

1.1.4. Getting Things of Independence

The most valuable thing for a disabled person is gaining independence. A blind person can lead an independent life with some specifically designed adaptive things for them. There are lots of adaptive equipment that can enable a blind person to live their life independently but they are not easily available in the local shops or markets. A blind person needs to hunt and put much effort to get each equipment that can take them one step closer towards independence.

Everyone faces challenges in their life... blind people face a lot more. But, this certainly does not mean that you can show sympathy to blind persons. They too, just like any individual, take up life's challenges and live a normal life, even if it does not seem normal to the sighted individuals.



1.2. Suggested Solutions

In this section, we will suggest a set of solutions of the blind people problems, and how to help them to have a normal live and do what they want alone with out the help of people.

We found that the best solution is to design Smart Glasses which contain a high resolution camera that can detect and recognize faces , colors , Text , objects . The glass can also define the Qibla direction , time ,weather state and it can be used as a remote control to control the electrical devices as TV and air condition, and convert Them to sound output so that the blind person can practice his life easier,



Figure (1.1) Project Components



1.3. Project

1.3.1. Python Software for camera

A high resolution camera that is programmed using python so that it can detect and recognize faces , colors , text , objects .. etc

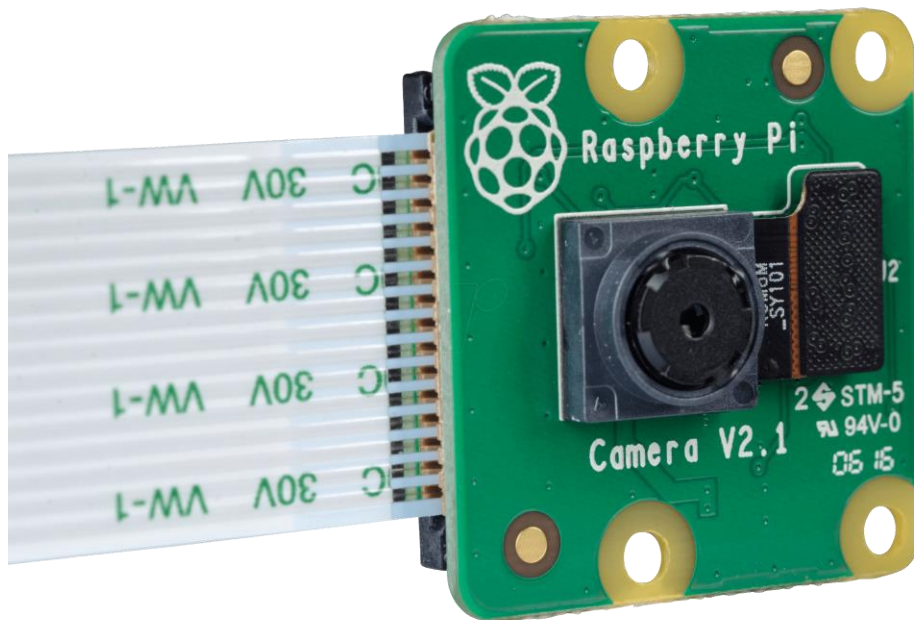


Figure (1.2) Raspberry pi camera



1.3.2 Glasses which contains raspberry pi and the camera

The glasses that contain raspberry pi that controls the camera and the sound is generated using its operating system. It also contains a lot of bush buttons that performing the features.

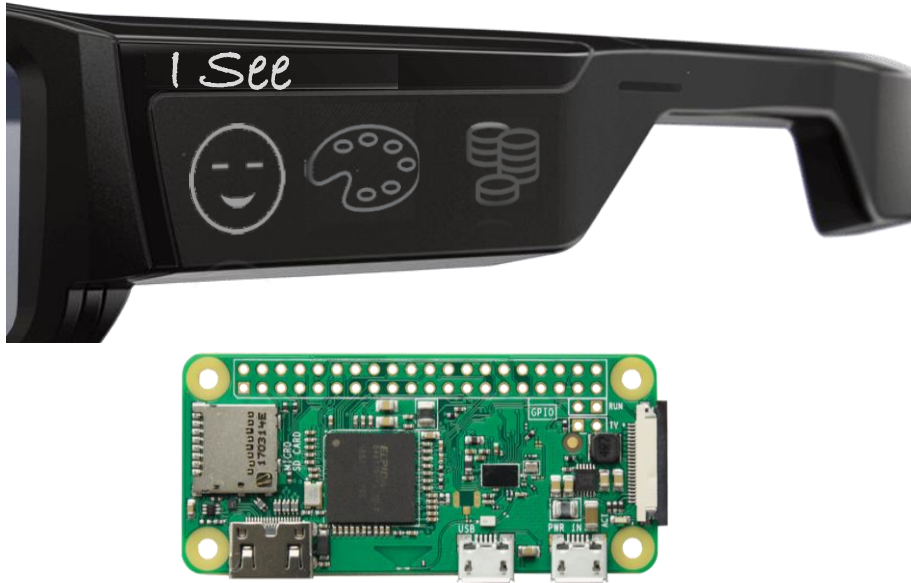


Figure (1.3) Raspberry pi zero

1.3.3 Power Bank

So that we can generate power to raspberry pi and run its system and camera

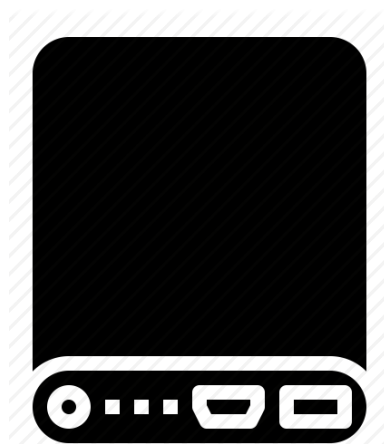


Figure (1.4) Power Bank



Chapter 2

Software & Programming Language



Chapter 2

Software & Programming Language

2.1 introduction

People often amused by wars over programming languages. People have very strong opinions about which one is better. The truth is that a programmer should chose the language appropriate for their task. Want to rapidly build a web app ? Try Ruby on Rails or Django. Want to write high performance code for an embedded device. Try C.

In computer vision, we are faced with similar choices. Which tool should a computer vision engineer / programmer learn — OpenCV using C++, OpenCV using Python, or MATLAB ? The good news is that today we have a few options to choose from! A decade back there were no good libraries for computer vision. If you wanted to learn, you just picked up a book and started coding your own mini library of computer vision algorithms. Thankfully, things are much better now.

If you are a beginner, programmers would suggest using the path of least resistance and picking a tool you are familiar with. If you are a python programmer, use OpenCV with Python. If you know C++, use C++ with OpenCV. The same holds true for MATLAB. That said in a few months you will no longer be a beginner. You may be looking to apply this newly acquired knowledge to the real world. You could be thinking of a new side project, or looking for a new job in this area.



2.2. Python Programming Language

2.2.1. Why Learn Python?

Python is a general-purpose language, which means it can be used to build just about anything, which will be made easy with the right tools/libraries.

Professionally, Python is great for backend web development, data analysis, artificial intelligence, and scientific computing. Many developers have also used Python to build productivity tools, games, and desktop apps, so there are plenty of resources to help you learn how to do those as well.

2.2.2. Beginner Friendliness

Python was designed to be easy to understand and fun to use (its name came from Monty Python so a lot of its beginner tutorials reference it). Fun is a great motivator, and since you'll be able to build prototypes and tools quickly with Python, many find coding in Python a satisfying experience. Thus, Python has gained popularity for being a beginner-friendly language, and it has replaced Java as the most popular introductory language at Top U.S. Universities.

2.2.3. Easy to Understand

Being a very high level language, Python reads like English, which takes a lot of syntax-learning stress off coding beginners. Python handles a lot of complexity for you, so it is very beginner-friendly in that it allows beginners to focus on learning programming concepts and not have to worry about too much details.

2.2.4. Very Flexible

As a dynamically typed language, Python is really flexible. This means there are no hard rules on how to build features, and you'll have more flexibility solving problems using different methods (though the Python philosophy encourages using the obvious way to solve things). Furthermore, Python is also more forgiving of errors, so you'll still be able to compile and run your program until you hit the problematic part.



Python can do



Desktop apps & Web apps



Data mining



Scientific computing

Figure (2.1) Python can do

2.2.5. Scalability (*Not Easy to Maintain*)

Because Python is a dynamically typed language, the same thing can easily mean something different depending on the context. As a Python app grows larger and more complex, this may get difficult to maintain as errors will become difficult to track down and fix, so it will take experience and insight to know how to design your code or write unit tests to ease maintainability.

2.2.6. Slow

As a dynamically typed language, Python is slow because it is too flexible and the machine would need to do a lot of referencing to make sure what the definition of something is, and this slows Python performance down.

At any rate, there are alternatives such as PyPy that are faster implementations of Python. While they might still not be as fast as Java, for example, it certainly improves the speed greatly.

2.2.7. Community

As you step into the programming world, you'll soon understand how vital support is, as the developer community is all about giving and receiving help. The larger a community, the more likely you'd get help and the more people will be building useful tools to ease the process of development.



2.2.8. 5th Largest StackOverflow Community

StackOverflow is a programming Q&A site you will no doubt become intimate with as a coding beginner. Python has 85.9k followers, with over 500k Python questions. Python questions are also the 3rd most likely to be answered when compared to other popular programming languages.

2.2.9. 3rd Largest Meetup Community

At meetups, you can generally network and learn from fellow developers. Meetups often offer mentorship to those who want it as well. There are 1300+ Python groups on Meetup.com, totaling 608k+ members. Thus, in terms of programming languages, Python is the 3rd largest community.

2.2.10. 4th Most-Used Language at GitHub

The more useful projects there are, the more likely someone has already built a function you need and built it well, which will greatly speed up your development process. Over 950 Python projects have over 500 stars.

Python is also known to have an abundance of libraries that assist with data analysis and scientific computing. In addition, PyGames is a neat game engine to build games with if you want to make simple games.



2.3. Libraries

2.3.1. OpenCv

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.



Figure (2.2) OpenCV



2.3.2. PIL

The **Python Imaging Library** adds image processing capabilities to your Python interpreter. This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.

The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

The library contains basic image processing functionality, including point operations, filtering with a set of built-in convolution kernels, and colour space conversions. It also supports image resizing, rotation and arbitrary affine transforms.

2.3.3. NumPy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.

Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.



2.3.4. OS

This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see **open()**, if you want to manipulate paths, see the **os.path** module, and if you want to read all the lines in all the files on the command line see the **fileinput** module. For creating temporary files and directories see the **tempfile** module, and for high-level file and directory handling see the **shutil** module.

2.3.5. gTTs

Text to Speech API commonly known as the gTTS API. gTTS is a very easy to use tool which converts the text entered, into audio which can be saved as a mp3 file. The gTTS API supports several languages including English, Hindi, Tamil, French, German and many more. The speech can be delivered in any one of the two available audio speeds, fast or slow. However, as of the latest update, it is not possible to change the voice of the generated audio.



Figure (2.3) Text to Speech API



Chapter 3

Optical Character Recognition (**OCR**)



Chapter 3

Optical Character Recognition

(OCR)

3.1. Introduction

Optical character recognition (also optical character reader, OCR) is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document.

Early versions needed to be trained with images of each character, and worked on one font at a time. Advanced systems capable of producing a high degree of recognition accuracy for most fonts are now common, and with support for a variety of digital image file format inputs (PNG , JPG ... etc).

Some systems are capable of reproducing formatted output that closely approximates the original page including images, columns, and other non-textual components.

3.2. History

Early optical character recognition may be traced to technologies involving telegraphy and creating reading devices for the blind.

In 1914, **Emanuel Goldberg** developed a machine called the **Optophone**, a handheld scanner that when moved across a printed page, produced tones that corresponded to specific letters or characters.

With the advent of smart-phones and smart glasses, OCR can be used in internet connected mobile device applications that extract text captured using the device's camera and The OCR API can returns the extracted text to speech.



3.2. Blind and visually impaired users

In 1974, Ray Kurzweil started the company Kurzweil Computer Products, Inc. and continued development of omni-font OCR, which could recognise text printed in virtually any font (Kurzweil is often credited with inventing omni-font OCR, but it was in use by companies, including CompuScan, in the late 1960s and 1970s). Kurzweil decided that the best application of this technology would be to create a reading machine for the blind, which would allow blind people to have a computer read text to them out loud. This device required the invention of two enabling technologies – the CCD flatbed scanner and the text-to-speech synthesiser. On January 13, 1976, the successful finished product was unveiled during a widely reported news conference headed by Kurzweil and the leaders of the National Federation of the Blind.

In 1978, Kurzweil Computer Products began selling a commercial version of the optical character recognition computer program. LexisNexis was one of the first customers, and bought the program to upload legal paper and news documents onto its nascent online databases. Two years later, Kurzweil sold his company to Xerox, which had an interest in further commercialising paper-to-computer text conversion. Xerox eventually spun it off as Scansoft, which merged with Nuance Communications.[citation needed] The research group headed by A. G. Ramakrishnan at the Medical intelligence and language engineering lab, Indian Institute of Science, has developed PrintToBraille tool, an open source GUI frontend[8] that can be used by any OCR to convert scanned images of printed books to Braille books.

In the 2000s, OCR was made available online as a service (WebOCR), in a cloud computing environment, and in mobile applications like real-time translation of foreign-language signs on a smartphone.

Various commercial and open source OCR systems are available for most common writing systems, including Latin, Cyrillic, Arabic, Hebrew, Indic, Bengali (Bangla), Devanagari, Tamil, Chinese, Japanese, and Korean characters.



3.3. Applications

OCR engines have been developed into many kinds of domain-specific OCR applications, such as receipt OCR, invoice OCR, check OCR, legal billing document OCR.

They can be used for:

- Data entry for business documents, e.g. check, passport, invoice, bank statement and receipt
- Automatic number plate recognition
- In airports, for passport recognition and information extraction
- Automatic insurance documents key information extraction
- Extracting business card information into a contact list
- More quickly make textual versions of printed documents, e.g. book scanning for Project Gutenberg
- Make electronic images of printed documents searchable, e.g. Google Books
- Converting handwriting in real time to control a computer (pen computing)
- Defeating CAPTCHA anti-bot systems, though these are specifically designed to prevent OCR. The purpose can also be to test the robustness of CAPTCHA anti-bot systems.
- Assistive technology for blind and visually impaired users

3.4. Types

- Optical character recognition (OCR) – targets typewritten text, one glyph or character at a time.
- Optical word recognition – targets typewritten text, one word at a time (for languages that use a space as a word divider). (Usually just called "OCR".)
- Intelligent character recognition (ICR) – also targets handwritten printscript or cursive text one glyph or character at a time, usually involving machine learning.
- Intelligent word recognition (IWR) – also targets handwritten printscript or cursive text, one word at a time. This is especially useful for languages where glyphs are not separated in cursive script.

OCR is generally an "offline" process, which analyses a static document. Handwriting movement analysis can be used as input to handwriting recognition. Instead of merely using the shapes of glyphs and words, this technique is able to capture motions, such as the order in which segments are drawn, the direction, and the pattern of putting the pen down and lifting it. This additional information can make the end-to-end process more accurate. This technology is also known as "on-line character recognition", "dynamic character recognition", "real-time character recognition", and "intelligent character recognition".



3.5. How does OCR work

Pattern recognition, works by identifying the character as a whole. We can identify a line of text by looking for rows of white pixels with rows of black pixels in between. In the same way, we can identify where an individual character begins and ends. Next, we convert the image of the character into a binary matrix where white pixels are 0s and black pixels are 1s.



Figure (3.1) Detecting the Character A

3.5. Tesseract OCR library

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. It can read all image types supported by the Python Imaging Library, including jpeg, png, gif, bmp, tiff, and others. It can read different types of languages including English , French , etc. also it can read and convert mathematical formulas



3.5.OCR-Code

3.5.1.

```
#To read all images format
from PIL import Image
#OCR Library
import pytesseract
import argparse
import cv2
# This module is imported so that we can
# play the converted audio
import os
# Import the required module for text
# to speech conversion
from gtts import gTTS
import numpy as np

cap = cv2.VideoCapture(0)
# (0) is for opening Default Camera
# If (1) USB Camera will run

while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Our operations on the frame come here
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Display the resulting frame
    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        cv2.imwrite('pic1.png',gray)
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

Figure (3.2) OCR-Code Part 1



3.5.2.

```
image = cv2.imread('pic1.png',0)
cv2.imshow("Image", image)
filename = "ss.png".format(os.getpid())
#cv2.imshow("Image", filename)
cv2.imwrite(filename, image)
pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files (x86)\Tess
mytext = pytesseract.image_to_string(Image.open(filename))
os.remove(filename)
# The text that you want to convert to audio
print(mytext)

# Language in which you want to convert
language = 'en'

# Passing the text and language to the engine,
# here we have marked slow=False. Which tells
# the module that the converted audio should
# have a high speed

myobj = gTTS(text=mytext, lang=language, slow=False)

# Saving the converted audio in a Wav file named
# welcome2
myobj.save("welcome2.wav")
# Playing the converted file
```

Figure (3.3) OCR-Code Part 2



Chapter 4

Bill Detection



Chapter 4

Bill Detection

4.1. Introduction

Banknote detection program Is a program that checks banknotes That came into the picture as much as a banknote This program was developed with OpenCV 3.0 using the Python language for development.

Banknote detection program uses the SIFT (Scale Invariant Feature Transform) technique to help detect and compare images that have been entered in OpenCV 3.0 in normal installation. Therefore, you need to install Extra modules of OpenCV first.

4.2. SIFT (Scale-Invariant Feature Transform)

4.2.1.Theory

In last couple of chapters, we saw some corner detectors like Harris etc. They are rotation-invariant, which means, even if the image is rotated, we can find the same corners. It is obvious because corners remain corners in rotated image also. But what about scaling? A corner may not be a corner if the image is scaled. For example, check a simple image below. A corner in a small image within a small window is flat when it is zoomed in the same window. So Harris corner is not scale invariant.

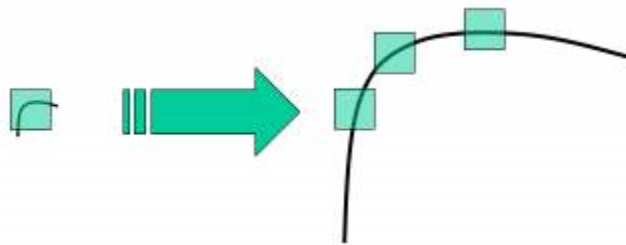


Figure (4.1) SIFT Theory



So, in 2004, D.Lowe, University of British Columbia, came up with a new algorithm, Scale Invariant Feature Transform (SIFT) in his paper, Distinctive Image Features from Scale-Invariant Keypoints, which extract keypoints and compute its descriptors. (This paper is easy to understand and considered to be best material available on SIFT. So this explanation is just a short summary of this paper).

There are mainly four steps involved in SIFT algorithm. We will see them one-by-one.

4.2.2. Scale-space Extrema Detection

From the image above, it is obvious that we can't use the same window to detect keypoints with different scale. It is OK with small corner. But to detect larger corners we need larger windows. For this, scale-space filtering is used. In it, Laplacian of Gaussian is found for the image with various σ values. LoG acts as a blob detector which detects blobs in various sizes due to change in σ . In short, σ acts as a scaling parameter. For eg, in the above image, gaussian kernel with low σ gives high value for small corner while gaussian kernel with high σ fits well for larger corner. So, we can find the local maxima across the scale and space which gives us a list of (x, y, σ) values which means there is a potential keypoint at (x, y) at σ scale.

But this LoG is a little costly, so SIFT algorithm uses Difference of Gaussians which is an approximation of LoG. Difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different σ , let it be σ and $k\sigma$. This process is done for different octaves of the image in Gaussian Pyramid. It is represented in below image:

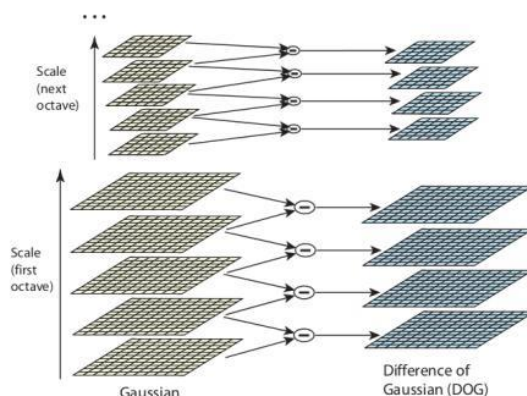


Figure (4.2) Process



Once this DoG are found, images are searched for local extrema over scale and space. For eg, one pixel in an image is compared with its 8 neighbours as well as 9 pixels in next scale and 9 pixels in previous scales. If it is a local extrema, it is a potential keypoint. It basically means that keypoint is best represented in that scale. It is shown in below image:

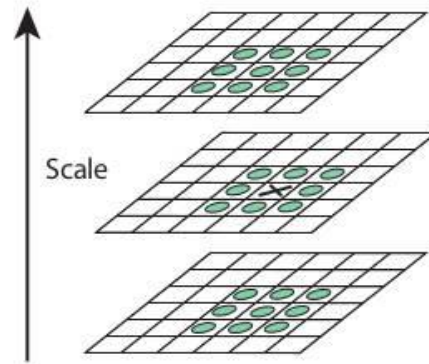


Figure (4.3) potential keypoint

Regarding different parameters, the paper gives some empirical data which can be summarized as, number of octaves = 4, number of scale levels = 5, initial $\sigma=1.6$, $k=\sqrt{2}$ etc as optimal values.

4.2.3. Keypoint Localization

Once potential keypoints locations are found, they have to be refined to get more accurate results. They used Taylor series expansion of scale space to get more accurate location of extrema, and if the intensity at this extrema is less than a threshold value (0.03 as per the paper), it is rejected. This threshold is called `contrastThreshold` in OpenCV.

DoG has higher response for edges, so edges also need to be removed. For this, a concept similar to Harris corner detector is used. They used a 2×2 Hessian matrix (H) to compute the principal curvature. We know from Harris corner detector that for edges, one eigen value is larger than the other. So here they used a simple function. If this ratio is greater than a threshold, called `edgeThreshold` in OpenCV, that keypoint is discarded. It is given as 10 in paper.

So it eliminates any low-contrast keypoints and edge keypoints and what remains is strong interest points.



4.2.4. Orientation Assignment

Now an orientation is assigned to each keypoint to achieve invariance to image rotation. A neighbourhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. (It is weighted by gradient magnitude and gaussian-weighted circular window with σ equal to 1.5 times the scale of keypoint. The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation. It creates keypoints with same location and scale, but different directions. It contribute to stability of matching.

4.2.5. Keypoint Descriptor

Now keypoint descriptor is created. A 16x16 neighbourhood around the keypoint is taken. It is divided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form keypoint descriptor. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation etc.

4.2.6. Keypoint Matching

Keypoints between two images are matched by identifying their nearest neighbours. But in some cases, the second closest-match may be very near to the first. It may happen due to noise or some other reasons. In that case, ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected. It eliminates around 90% of false matches while discards only 5% correct matches, as per the paper.

So this is a summary of SIFT algorithm. For more details and understanding, reading the original paper is highly recommended. Remember one thing, this algorithm is patented. So this algorithm is included in Non-free module in OpenCV.



4.3.SIFT in OpenCV

So now let's see SIFT functionalities available in OpenCV. Let's start with keypoint detection and draw them. First we have to construct a SIFT object. We can pass different parameters to it which are optional and they are well explained in docs.

`sift.detect()` function finds the keypoint in the images. You can pass a mask if you want to search only a part of image. Each keypoint is a special structure which has many attributes like its (x,y) coordinates, size of the meaningful neighbourhood, angle which specifies its orientation, response that specifies strength of keypoints etc.

OpenCV also provides `cv2.drawKeyPoints()` function which draws the small circles on the locations of keypoints. If you pass a flag, `cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS` to it, it will draw a circle with size of keypoint and it will even show its orientation .

See the two results below:



Figure (4.4) circles using OpenCV

Now to calculate the descriptor, OpenCV provides two methods:

1-Since you already found keypoints, you can call `sift.compute()` which computes the descriptors from the keypoints we have found. Eg: `kp,des = sift.compute(gray,kp)`

2-If you didn't find keypoints, directly find keypoints and descriptors in a single step with the function, `sift.detectAndCompute()`.



4.4. Bill Detection Code

init_feature: function sets the initial value and algorithm in the search feature.

MIN_POIN: Used to determine the minimum number of features that must be detected.

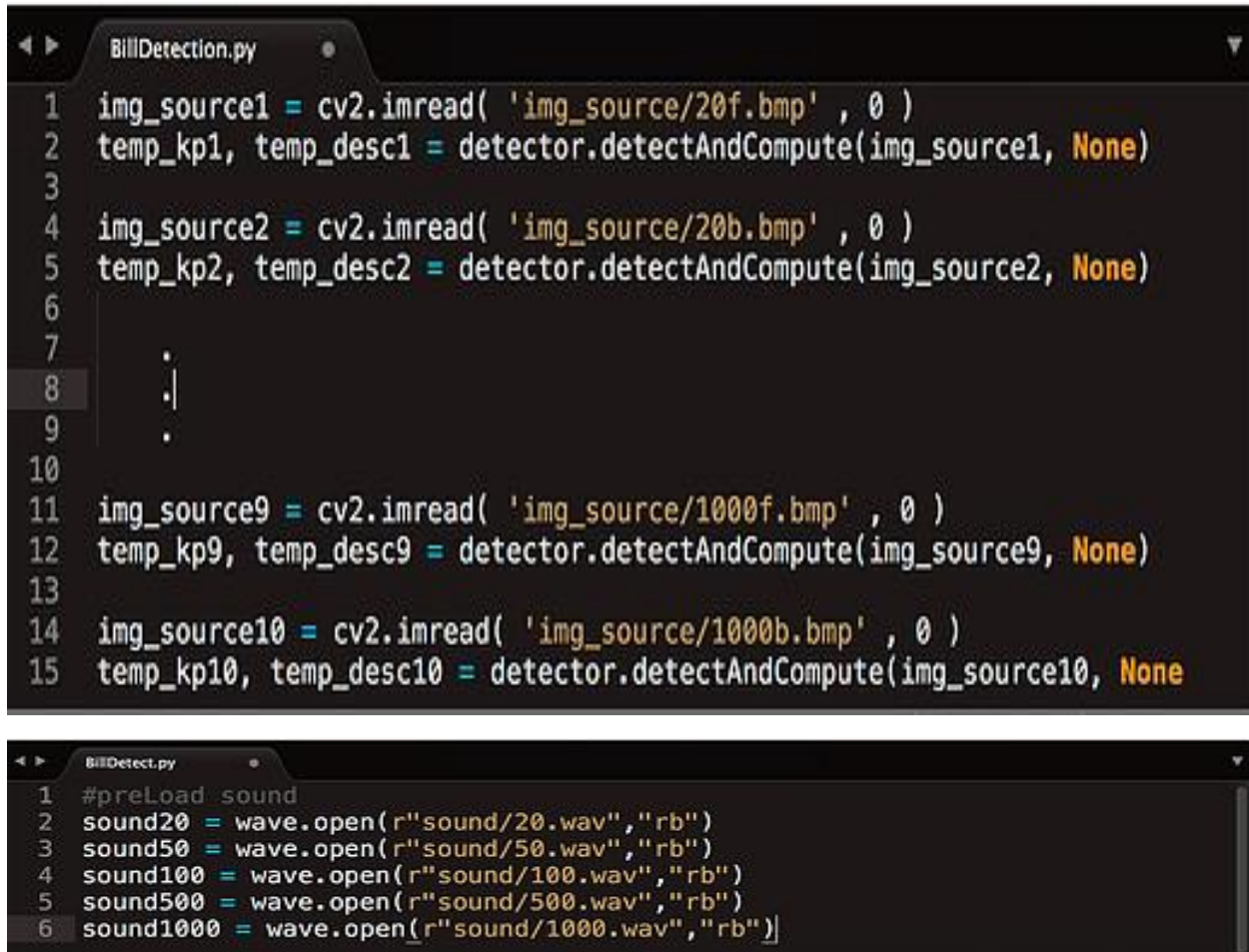
FLANN_INDEX_KDTREE: Use the algorithm to determine the Nearest Neighbors

```
BillDetection.py
1 import numpy as np
2 import cv2
3 import common
4 import time
5 import threading
6 import sys, getopt
7 from common import anorm, getsize
8
9 def init_feature():
10     detector = cv2.xfeatures2d.SIFT_create()
11     norm = cv2.NORM_L1
12
13     flann_params = dict(algorithm = FLANN_INDEX_KDTREE,
14                         trees = MIN_POINT)
15     matcher = cv2.BFMatcher(norm)
16     return detector, matcher
17
18 cv2.useOptimized()
19
20 MIN_POINT = 15
21
22 FLANN_INDEX_KDTREE = 5
23 FLANN_INDEX_LSH = 20
24
25 cap = cv2.VideoCapture(0)
26
27 detector, matcher = init_feature()
28
29 found = False
30 searchIndex = 1
```

Figure (4.5) Bill Detection Code 1



In the main part of the program We will read the prototype image and calculate the feature that will be used to search before entering the loop of image processing from the camcorder. Because it helps to increase the calculation speed fairly.



```
BillDetection.py
1 img_source1 = cv2.imread( 'img_source/20f.bmp' , 0 )
2 temp_kp1, temp_desc1 = detector.detectAndCompute(img_source1, None)
3
4 img_source2 = cv2.imread( 'img_source/20b.bmp' , 0 )
5 temp_kp2, temp_desc2 = detector.detectAndCompute(img_source2, None)
6
7 .
8 .|
9 .
10
11 img_source9 = cv2.imread( 'img_source/1000f.bmp' , 0 )
12 temp_kp9, temp_desc9 = detector.detectAndCompute(img_source9, None)
13
14 img_source10 = cv2.imread( 'img_source/1000b.bmp' , 0 )
15 temp_kp10, temp_desc10 = detector.detectAndCompute(img_source10, None)

BillDetect.py
1 #preLoad sound
2 sound20 = wave.open( r"sound/20.wav", "rb" )
3 sound50 = wave.open( r"sound/50.wav", "rb" )
4 sound100 = wave.open( r"sound/100.wav", "rb" )
5 sound500 = wave.open( r"sound/500.wav", "rb" )
6 sound1000 = wave.open( r"sound/1000.wav", "rb" )]
```

Figure (4.6) Bill Detection Code 2



Within the loop used to receive images from video cameras In one frame, one master bank will be used in the search. As soon as the next frame is changed, it will continue to use the next prototype banknote. The next frame will not change the original banknote.

```

1 while(True):
2     t1 = cv2.getTickCount()
3     p = pyaudio.PyAudio()
4     #switch template
5     if not found:
6         if searchIndex <= 10:
7             if searchIndex == 1:
8                 img1 = img_source1
9                 kp1 = temp_kp1
10                desc1 = temp_desc1
11                showText = '20'
12            elif searchIndex == 2:
13                img1 = img_source2
14                kp1 = temp_kp2
15                desc1 = temp_desc2
16                showText = '20'
17                .
18                .
19                .
20            elif searchIndex == 9:
21                img1 = img_source9
22                kp1 = temp_kp9
23                desc1 = temp_desc9
24                showText = '1000'
25            elif searchIndex == 10:
26                img1 = img_source10
27                kp1 = temp_kp10
28                desc1 = temp_desc10
29                showText = '1000'
30
31            searchIndex = searchIndex+1
32        else:
33            searchIndex = 1
34            img1 = img_source1

```

Figure (4.7) Bill Detection Code 3



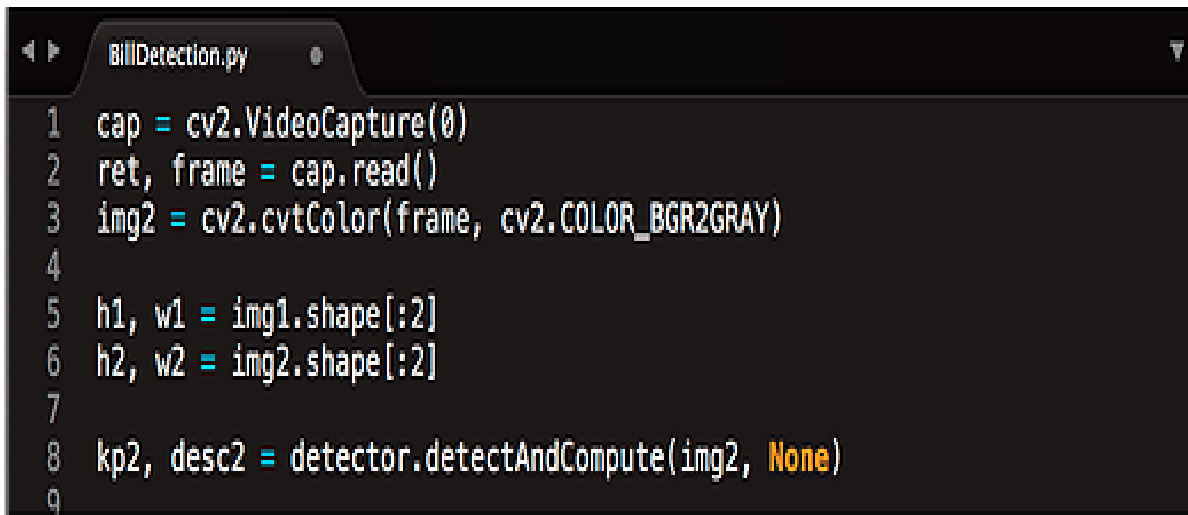
`cv2.cvtColor (frame, cv2.COLOR_BGR2GRAY)`

Get one video camera And changed from color to black and white

`h1, w1 = img1.shape [: 2]`

find the width and length of the prototype image And images from the
camcorder `detector.detectAndCompute`

Calculate the feature in the image from the camcorder to search



```

1 cap = cv2.VideoCapture(0)
2 ret, frame = cap.read()
3 img2 = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
4
5 h1, w1 = img1.shape[:2]
6 h2, w2 = img2.shape[:2]
7
8 kp2, desc2 = detector.detectAndCompute(img2, None)
9

```

Figure (4.8) Bill Detection Code 4

`cv2.cvtColor (frame, cv2.COLOR_BGR2GRAY)`

Get one video camera And changed from color to black and white

`h1, w1 = img1.shape [: 2]`

find the width and length of the prototype image And images from the
camcorder

`detector.detectAndCompute`

Calculate the feature in the image from the camcorder to search



```

1 cap = cv2.VideoCapture(0)
2 ret, frame = cap.read()
3 img2 = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
4
5 h1, w1 = img1.shape[:2]
6 h2, w2 = img2.shape[:2]
7
8 kp2, desc2 = detector.detectAndCompute(img2, None)
9

```

Figure (4.9) Bill Detection Code 5



After that, start matching between the feature of the original image and the feature of the image received from the camcorder by calling the function `match_and_draw`

While the function `getTrickCount ()` is used to timer that one How much time is calculated (In seconds) and calculated as fps for optimizing the algorithm to work as quickly as possible

```

1  found, checksound, count = match_and_draw('find_obj',
2      checksound, found, count)
3
4  t2 = cv2.getTickCount()
5  #calculate fps
6  time = (t2 - t1)/ cv2.getTickFrequency()
7  print 'FPS = ',1/time
8
9  if cv2.waitKey(1) & 0xFF == 27:
10     break

```

Figure (4.10) Bill Detection Code 6

Call function `match_and_draw` To start the search and get the true value back. If you find it and get false back if you can't find it To be used to stop changing the prototype image in the next frame search

```

1  def match_and_draw(win, checksound, found , count):
2      if(len(kp2) > 0):
3          #matching feature
4          raw_matches = matcher.knnMatch(desc1, trainDescriptors = desc2, k =
5              p1, p2, kp_pairs = filter_matches(kp1, kp2, raw_matches)
6          if len(p1) >= MIN_POINT:
7              if not found:
8                  checksound = True
9                  found = True
10                 count = count + 1
11                 H, status = cv2.findHomography(p1, p2, cv2.RANSAC, 5.0)
12                 vis = explore_match(win, img1, img2, kp_pairs, status, H)
13                 #print '%d / %d  inliers/matched' % (np.sum(status), len(status))
14
15             else:
16                 found = False
17                 checksound = False
18                 H, status = None, None
19                 count = 0
20                 #print '%d matches found, not enough for homography estimation
21                 vis = np.zeros((max(h1, h2), w1+w2), np.uint8)
22                 vis[:h1, :w1] = img1
23                 vis[h2, w1:w1+w2] = img2
24
25                 cv2.imshow('find_obj', vis)
26                 if(count > 3):
27                     if checksound :
28                         checksound = False
29                         play_Sound(showText)
30                         count = 0
31
32     return found, checksound ,count

```

Figure (4.11) Bill Detection Code 7



function `filter_matches` is called by function `match_and_draw` Will search and match the features found

```

1 def filter_matches(kp1, kp2, matches, ratio = 0.75): #ratio = 0.75
2     mkp1, mkp2 = [], []
3     for m in matches:
4         if len(m) == 2 and m[0].distance < m[1].distance * ratio:
5             m = m[0]
6             mkp1.append( kp1[m.queryIdx] )
7             mkp2.append( kp2[m.trainIdx] )
8     p1 = np.float32([kp.pt for kp in mkp1])
9     p2 = np.float32([kp.pt for kp in mkp2])
10    kp_pairs = zip(mkp1, mkp2)
11    return p1, p2, kp_pairs

```

The `explore_match` function is called by the `match_and_draw` function. Find the corresponding feature between the original image and the image captured by the camera. And if found Will surround the part of the image detected Along with drawing the points of the corresponding feature

```

1 def explore_match(win, img1, img2, kp_pairs, status = None, H = None):
2     vis = np.zeros((max(h1, h2), w1+w2), np.uint8)
3     vis[:h1, :w1] = img1
4     vis[:h2, w1:w1+w2] = img2
5     vis = cv2.cvtColor(vis, cv2.COLOR_GRAY2BGR)
6     if H is not None :
7         corners = np.float32([[0, 0], [w1, 0], [w1, h1], [0, h1]])
8         corners = np.int32( cv2.perspectiveTransform(
9             corners.reshape(1, -1, 2), H).reshape(-1, 2) + (w1, 0) )
10        cv2.polylines(vis, [corners], True, (0, 255, 0))
11        font = cv2.FONT_HERSHEY_SIMPLEX
12        cv2.putText(vis, showText, (corners[0][0], corners[0][1]), font,
13            1, (0, 0, 255), 2)
14    if status is None:
15        status = np.ones(len(kp_pairs), np.bool_)
16
17    p1 = np.int32([kpp[0].pt for kpp in kp_pairs])
18    p2 = np.int32([kpp[1].pt for kpp in kp_pairs]) + (w1, 0)
19    for (x1, y1), (x2, y2), inlier in zip(p1, p2, status):
20        if inlier:
21            col = (0, 255, 0)
22            cv2.circle(vis, (x1, y1), 2, col, -1)
23            cv2.circle(vis, (x2, y2), 2, col, -1)
24    return vis

```

Result :

The results will be satisfying . The program can detect correctly. But there are times when it can not be detected Because the image quality may not be sharp enough or if there is similarity in the different bills .



Chapter 5

Face Recognition



