# Mini-project-Report

**(OpenCV Development)**

**TITLE:** Design an application for object detection.

## OBJECTIVES:

1. To study and understand the open CV system functionalities to develop application.
2. To understand how to use open CV libraries to perform recognition.
3. To develop open CV program using python Concepts.

**PROBLEM STATEMENT:** To develop application project, to detect objects using open CV in python.

## OUTCOMES:

1. Use of appropriate method and classes to develop a application to detect objects using open CV in python.
2. Use of appropriate data structure for implementation.

## SOFTWARE & HARDWARE REQUIREMENTS:

**Software Requirements:**

1. Operating System : Ubuntu or Fedora
2. Python 2.7
3. Open cv, numpy

## Hardware Requirements:

1. 64 bit machine
2. 4GB or 8 GB RAM
3. 500 GB or 1TB HDD

## THEORY-CONCEPT:

## OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it’s free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.

**SIFT:**

The **scale-invariant feature transform** (**SIFT**) is a feature detection algorithm in computer vision to detect and describe local features in images. It was patented in Canada by the University of British Columbia and published by David Lowe in 1999. Applications include object recognition, robotic mapping and navigation, image stitching, 3D modeling, gesture recognition, video tracking, individual identification of wildlife and match moving.

SIFT keypoints of objects are first extracted from a set of reference images[[2]](https://en.wikipedia.org/wiki/Scale-invariant_feature_transform#cite_note-Lowe1999-2) and stored in a database. An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on Euclidean distance of their feature vectors. From the full set of matches, subsets of keypoints that agree on the object and its location, scale, and orientation in the new image are identified to filter out good matches. The determination of consistent clusters is performed rapidly by using an efficient hash table implementation of the generalised Hough transform. Each cluster of 3 or more features that agree on an object and its pose is then subject to further detailed model verification and subsequently outliers are discarded. Finally the probability that a particular set of features indicates the presence of an object is computed, given the accuracy of fit and number of probable false matches. Object matches that pass all these tests can be identified as correct with high confidence.

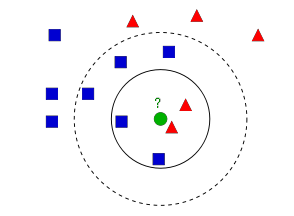
## What is FLANN?

FLANN is a library for performing fast approximate nearest neighbor searches in high dimensional spaces. It contains a collection of algorithms we found to work best for nearest neighbor search and a system for automatically choosing the best algorithm and optimum parameters depending on the dataset.

FLANN is written in C++ and contains bindings for the following languages: C, MATLAB and Python.

**KNN:**

kNN is one of the simplest of classification algorithms available for supervised learning. The idea is to search for closest match of the test data in feature space. We will look into it with below image.



In the image, there are two families, *Blue Squares and Red Triangles*. We call each family as **Class**. Their houses are shown in their town map which we call *feature space*. *(You can consider a feature space as a space where all datas are projected. For example, consider a 2D coordinate space. Each data has two features, x and y coordinates. You can represent this data in your 2D coordinate space, right? Now imagine if there are three features, you need 3D space. Now consider N features, where you need N-dimensional space, right? This N-dimensional space is its feature space. In our image, you can consider it as a 2D case with two features)*.

Now a new member comes into the town and creates a new home, which is shown as green circle. He should be added to one of these Blue/Red families. We call that process, **Classification**. What we do? Since we are dealing with kNN, let us apply this algorithm.

One method is to check who is his nearest neighbour. From the image, it is clear it is the Red Triangle family. So he is also added into Red Triangle. This method is called simply **Nearest Neighbour**, because classification depends only on the nearest neighbour.

But there is a problem with that. Red Triangle may be the nearest. But what if there are lot of Blue Squares near to him? Then Blue Squares have more strength in that locality than Red Triangle. So just checking nearest one is not sufficient. Instead we check some *k* nearest families. Then whoever is majority in them, the new guy belongs to that family. In our image, let’s take *k=3*, ie 3 nearest families. He has two Red and one Blue (there are two Blues equidistant, but since k=3, we take only one of them), so again he should be added to Red family. But what if we take *k=7*? Then he has 5 Blue families and 2 Red families. Great!! Now he should be added to Blue family. So it all changes with value of k. More funny thing is, what if *k = 4*? He has 2 Red and 2 Blue neighbours. It is a tie !!! So better take k as an odd number. So this method is called **k-Nearest Neighbour** since classification depends on k nearest neighbours.

Again, in kNN, it is true we are considering k neighbours, but we are giving equal importance to all, right? Is it justice? For example, take the case of *k=4*. We told it is a tie. But see, the 2 Red families are more closer to him than the other 2 Blue families. So he is more eligible to be added to Red. So how do we mathematically explain that? We give some weights to each family depending on their distance to the new-comer. For those who are near to him get higher weights while those are far away get lower weights. Then we add total weights of each family separately. Whoever gets highest total weights, new-comer goes to that family. This is called **modified kNN**.

So what are some important things you see here?

* You need to have information about all the houses in town, right? Because, we have to check the distance from new-comer to all the existing houses to find the nearest neighbour. If there are plenty of houses and families, it takes lots of memory, and more time for calculation also.
* There is almost zero time for any kind of training or preparation.

**CONCLUSION:**

Demonstration of android mobile application project, to fetch music files from internal or external storage and play music progress using SeekBar & ListView is successfully implemented using android-studio.

## DATE OF COMPLETITION:

**ASSESSMENT GRADE/MARKS AND ASSESSOR'S SIGN:**