RBE 549 Computer Vision

Course Project on

Object Recognition

<u>Team Leprechaun</u>

Akshat Jain

Shlok Agarwal

Rohit Salem

Apoorva Gupta

INDEX

1. <u>Abstract</u>

This project aims at developing algorithms using computer vision techniques to recognize a given object (which in our case, happens to be a Leprechaun soft toy shown in Figure 1).

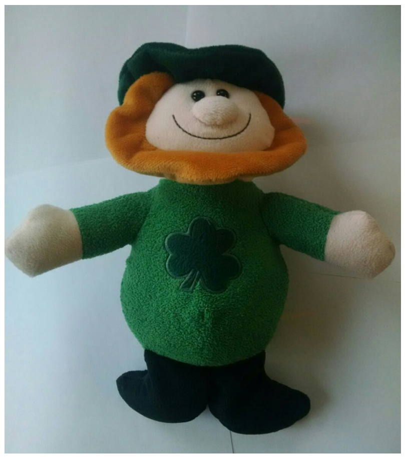


Figure 1: Given Object

Since the given object has a lot of information, many techniques can be implemented for tracking it. We implemented a total of four techniques to track this object: (a) Camshift, (b) SURF, (c) Neural Network and (d) Decision Tree.

2. Introduction

There are a number of ways to do real time object recognition and in this project we aim at using four completely different approaches to compare and come up with results. The project is challenging because the recognition has to be done real time with varying environmental conditions. The conditions here refer to varying lighting effects, amount of clustering in the environment and occluded view of the object. The object chosen for this project is a Leprechaun, a soft toy.

The first technique used is Camshift (Continuously Adaptive Meanshift Algorithm) is a technique based on the meanshift algorithm. It is a general purpose object tracking approach where we do not make any assumptions about the object to be tracked.

The second technique used is SURF(speeded up robust features) which is a very robust and common approach used in the industry for object recognition.

The third technique we used was a neural network approach. This approach does not use any computer vision technique. We implemented this technique only to demonstrate it as one of the techniques that may be used.

The fourth technique used is a decision tree approach. We train the classifier to detect the object. Features are extracted using Oriented FAST and Rotated BRIEF (ORB) descriptors.

3. Methodology

We implemented the following algorithms:

a. Camshift
b. SURF(Speeded Up Robust Features)
c. Neural Network
d. Decision Tree

A. Camshift:

Camshift stands for Continuously Adaptive Meanshift algorithm.
We'll first discuss the Meanshift algorithm. Consider a set of points shown in Figure 2.

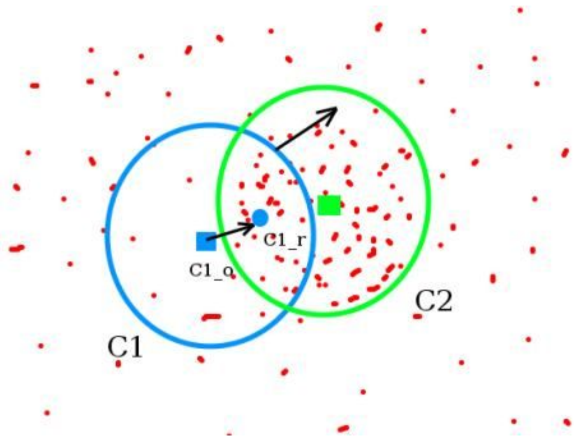


Figure 2: Shows a set of points and circular windows

We start with a circular window C1 that can be randomly placed anywhere in the figure as shown. Next, we compute the centroid of the points lying inside this window (this is shown as point C1_r in the figure). We then shift the center of the window (C1_q) to the calculated centroid (C1_r). This process is repeated till we reach a stage where the center of our window and the centroid of the points match (like in C2). This is known as the Meanshift algorithm.

This technique works great if we filter out our object and pass the filtered image as an input to the algorithm. As our object moves, its mean also moves and in turn our tracking window will move as it tries to match the center to centroid. However, the Meanshift algorithm has some limitations. Irrespective of the distance of the object to the camera, the window size always remains constant. Also, the window doesn't rotate with rotation of the object. This is shown in Figure 3.

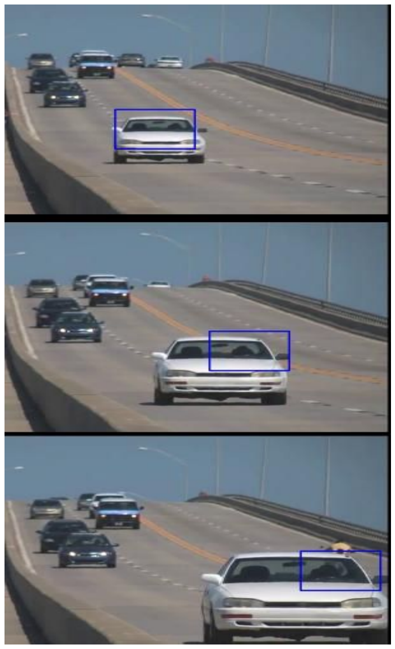
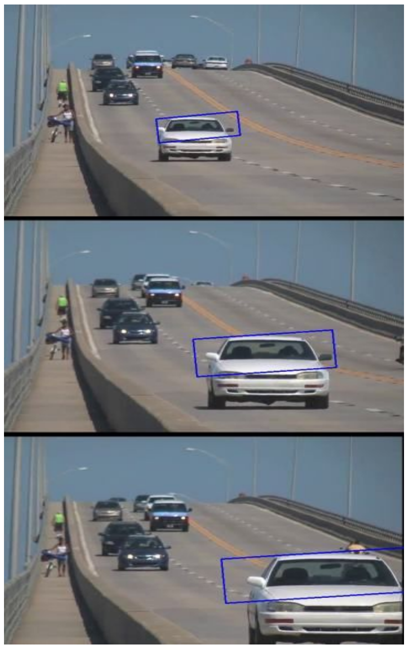



Figure 3: Output of Meanshift Figure 4: Output of Camshift

Camshift is an adaptation of the Meanshift algorithm. It fixes the window size and rotation problems. It first applies meanshift. And once, it converges, it updates the size of the window using the formula:

$$s = 2 \times \sqrt{\frac{M_{00}}{256}}$$

It also calculates the orientation of the best fitting ellipse. This process is repeated as the object moves (example shown in Figure 4).

<u>Pseudo Code:</u>

```
while(True):
    Let user select object to track using the mouse by click and drag.
    If user selects object:
        Selected object = Region of Interest (RoI)
        Dynamic Thresholding of HSV values using RoI
        Apply erosion and dilation
        Get binary mask of object to be tracked
        Start Camshift
        Draw ellipse around object
    If user decides to exit:
        Break
end
```

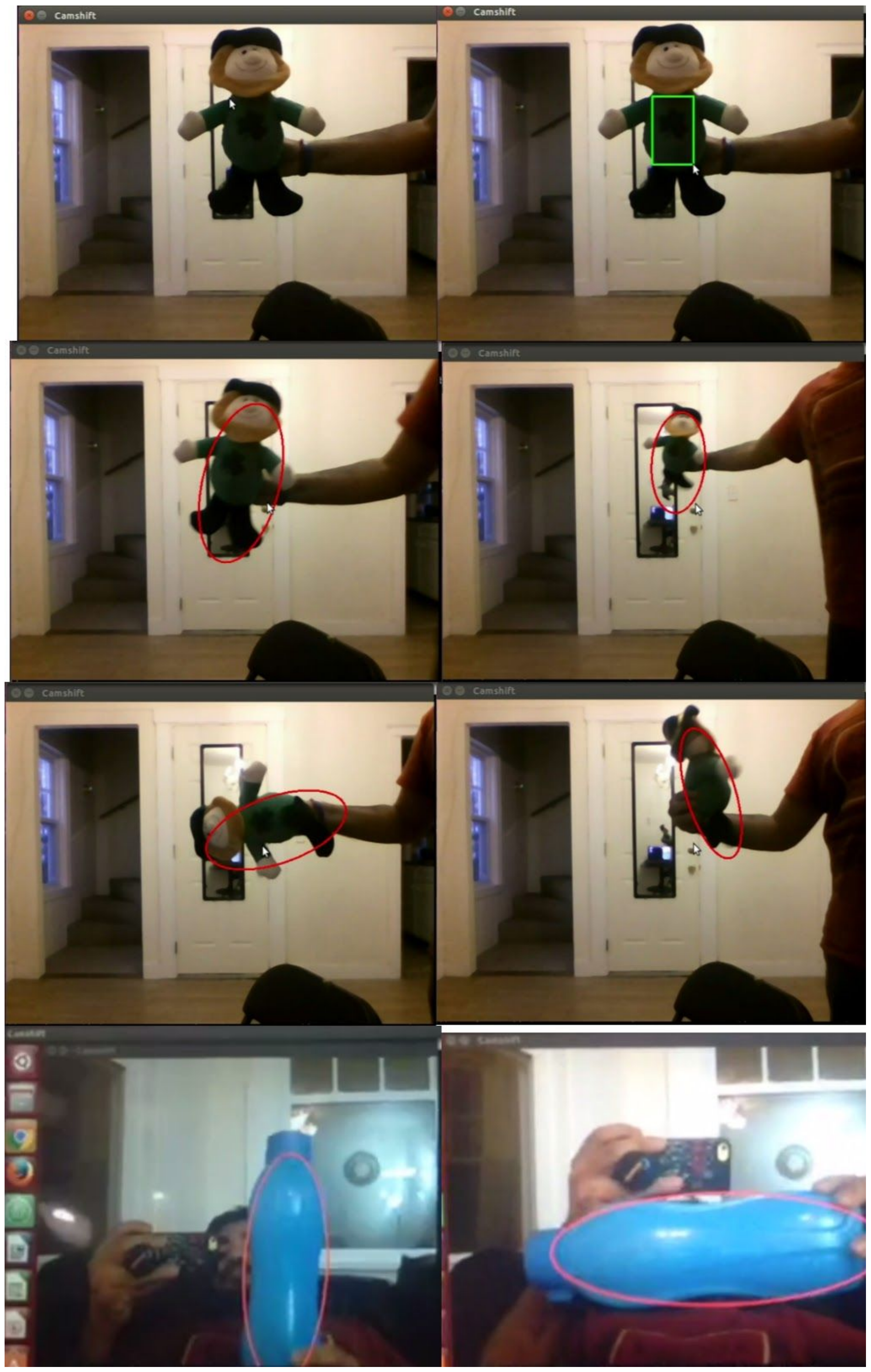


Figure 5: First row shows how the object is selected. Second and third row show the scale and rotation invariant nature of Camshift respectively. Fourth row shows the generic nature of the algorithm as same code works for any object.

Advantages:

- Invariant to translation, scale and rotation.
- Generic algorithm that works for any object as long as the selected object/part of the object has a uniform color that can be treated as our RoI.
- Dynamic on-the-go thresholding makes it robust to different lighting conditions.
- Works for partial occlusion.
- Takes only about ~10ms per cycle.

Disadvantages:

- Currently doesn’t work for total occlusion but techniques like adaptive template, etc can be implemented to make it work for total occlusion.
- Fails for a clustered background with similar colored objects.

B. SURF:

Speeded up robust features(SURF) is a patented feature detector and descriptor used for object recognition. It is an extension on the Shift Invariant Feature Transform (SIFT) proposed by D. Lowe in 1999. It is reported to be three times faster than SIFT and almost as robust.

The SIFT algorithm approximates the Laplacian of Gaussian with difference of Gaussians for scale spacing. SURF approximates the Laplacian of Gaussian with the Box Filter. This speeds up the process as convolution with box filter can be calculated using Integral Images.

Even though OpenCV has inbuilt functions and good support for SIFT and SURF, the technology is patented and cannot be used for commercial applications without permission.

There are three main steps involved in using SURF:

1. Interest Point Detection
2. Interest Point Description Feature Vector Extraction
3. Feature Vector Matching Between Two Images

There are a number of ways of extracting interest points. A suitable choice of technique could be made using each of them individually with varying thresholds. The objective should be to get minimal number of keypoints identifying unique features in the object. It would also help to experiment with a variety of images taken at different angles. A plain background would give minimal or no keypoints in the background which would help while matching.

After the keypoints are identified, the next step is to create a feature vector which would contain the information of the keypoints. This would be used for matching the features.

To get started with using this technique, the keypoints and feature vector is identified for the training image. Ideally, for real time object recognition, the feature vector of the training image is matched with the feature vector of the image received from camera feed.

When using the ideal method, we were not getting accurate results, so we used simple hacks to get a better results. The algorithm we used is described in the table below:

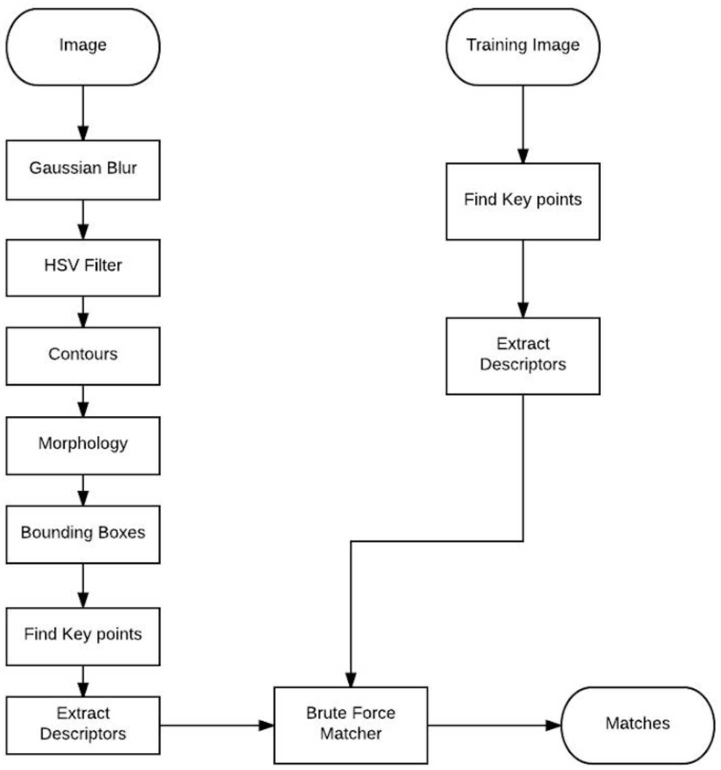
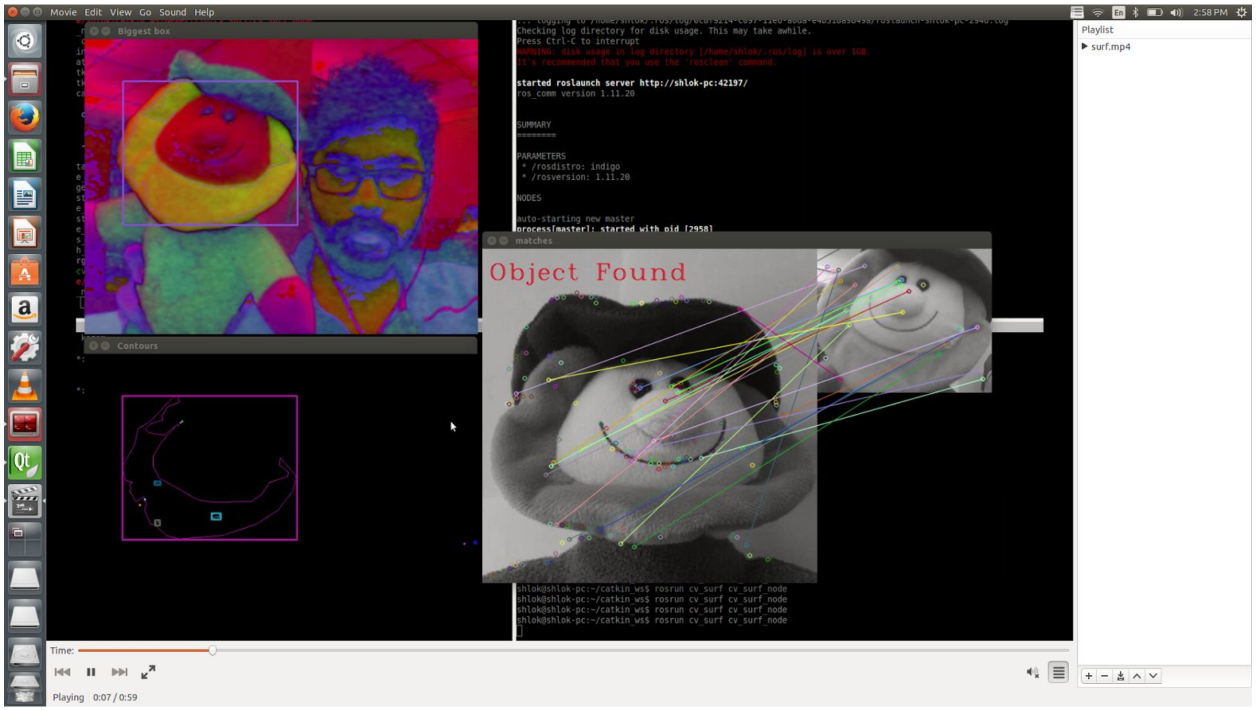


Figure 6. Flowchart of working of the algorithm

The image received from the camera is blurred to filter sharp edges. Then we use a HSV filter to further filter the regions with almost similar HSV values as the training image. We then find contours of the filtered image and use morphology to close the open edges. Based on the geometry of the contours, we create bounding boxes which are used to identify keypoints and feature vector. So in a nutshell, we compare the highly filtered image with the training image to get smooth and accurate results. Following this approach we got very robust recognition which is invariant to translation, scale, rotation and noise. The frame rates were 40-70 ms which made this method ideal for real time object recognition.A sample image from our testing could be seen below.



```
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
WARNING: disk usage in log directory [/home/shlok/.ros/log] is over 1GB.
It's recommended that you use the 'rosclean' command.

started roslaunch server http://shlok-pc:42197/
ros_comm version 1.11.20


SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.20

NODES

auto-starting new master
process[master]: started with pid [2958]
```

```
shlok@shlok-pc:~/catkin_ws$ rosrun cv_surf cv_surf_node
shlok@shlok-pc:~/catkin_ws$ rosrun cv_surf cv_surf_node
shlok@shlok-pc:~/catkin_ws$ rosrun cv_surf cv_surf_node
shlok@shlok-pc:~/catkin_ws$ rosrun cv_surf cv_surf_node
```

Figure 7: Output of SURF

C. Neural Network:

A typical Neural Network consists of layers which can be divided into three categories: Input, Hidden and output layers (fig 8). These layers are made up of number of interconnected nodes with an activation function. The input to these functions is the weighted values of the previous layer as seen in the figure 9. The output of this activation functions is input to the next layers in this case the output of the hidden layer is the output of the neural network which is a score. Based on the score and the confidence level of the objects can be classified in a classification problem.

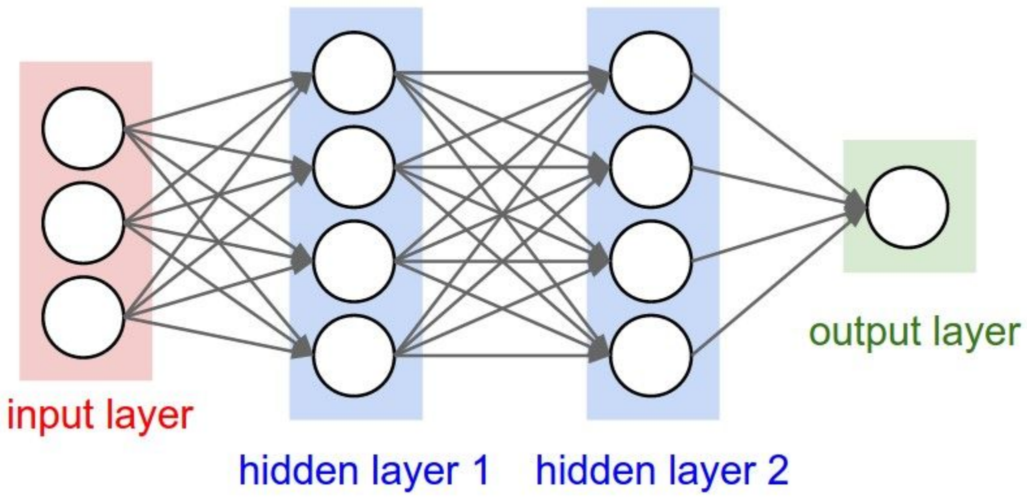


Figure 8: A typical neural network

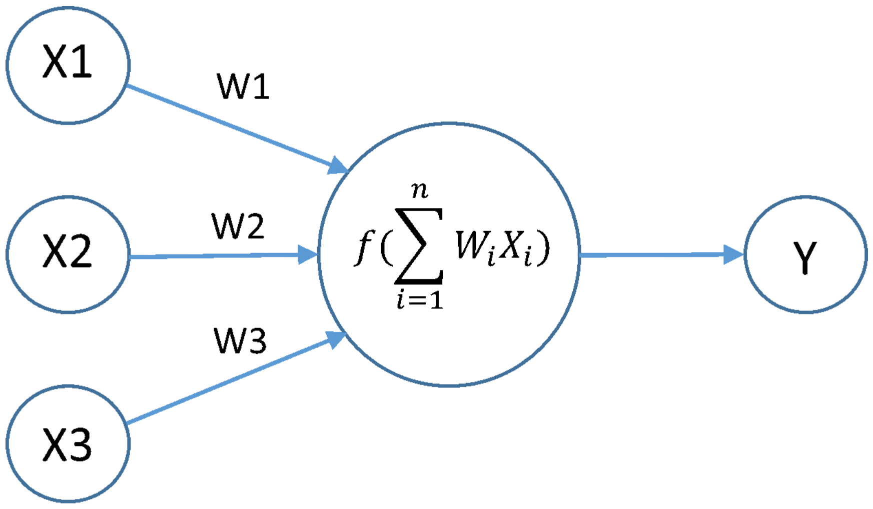


Figure 9: Shows how input is mapped to output for each neuron

Data set:

The dataset for the problem was created using the one minute videos of the object taken in different orientations in three different backgrounds and these videos were converted into images which served as the dataset for training.

Methodology

Our problem statement is to detect an object and therefore there could be only one possible output which is a score. The input to the network is the pixel values of the images that belong to the dataset. While training, the input images are fed into the network and weights corresponding to layers the are modified using backpropagation to maximize the score of the object, Here as there is only one value for the output we set a threshold for the score, if the output value is greater than the threshold then the object is detected.In our case we have used standard VGG-16 model as the network architecture. This model uses convolutional layers to predict the score. The activation function is ReLU (Linear Rectified Unit).

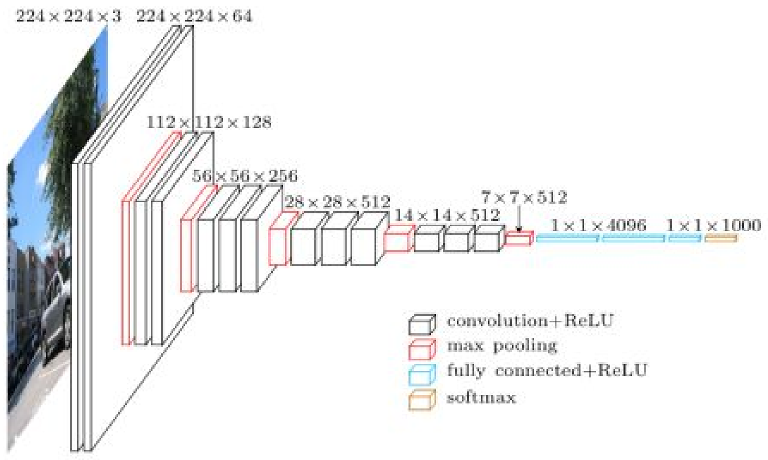


Figure 10: VGG-16 Network Architecture

D. Decision Tree:

Decision Tree Algorithm was implemented to train the classifier to detect the object. Features were extracted using Oriented FAST and Rotated BRIEF(ORB) descriptors.The problem was converted into a binary classification by giving object class as one and background as zero.K-means was used to reduce the number of false positives in Decision Tree’s prediction.

Working of BRIEF:
Given the key-points from CenSurE, the BRIEF algorithm performs pair-wise intensity comparisons over a 16 x 16 neighborhood of these key-points using a test function defined as

$$\begin{aligned} t(p;x,y) &= 1 \ \mathit{if}\ p(x) < p(y) \\ &= 0 \ \mathit{otherwise} \end{aligned}$$

The test function when performed on each interest point returns a 256 bit i.e. 32 byte codeword called a “Bitstring” which then forms the descriptor array of size 1 x 32 for that corresponding key-point.

Since CenSurE detects N such key-points, the entire descriptor matrix is an N x 32 matrix, where each row corresponds to the descriptor corresponding to a key-point and each column is 8 bits out of the 256-bit bitstring.

Flow Diagram:
Two approaches were tried out. The first was to extract key points and descriptors and use Brute Force Matching. The results of this almost plug-and-play approach were not satisfactory and hence, a second approach of using machine learning techniques such as classification and clustering to detect the object using key-points from ORB and descriptors computed from BRIEF was implemented. This object detection technique was tried out on still images the results of the second approach proved to be far better than the first.

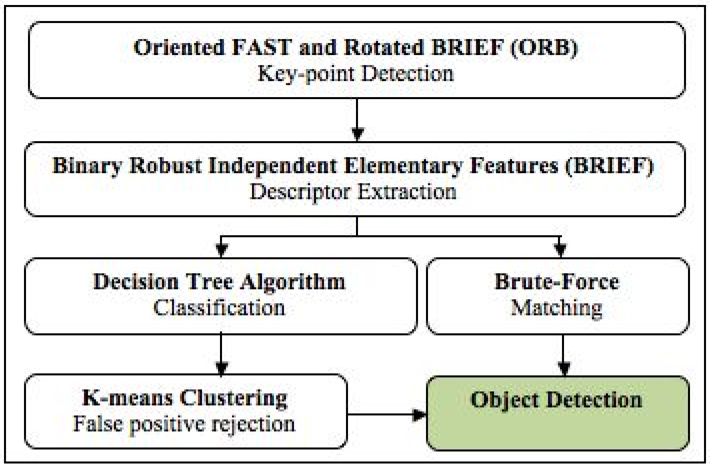


Figure 11: Flow Diagram

During Brute Force matching, the number of descriptors shoots up for images under different conditions and the sheer size of database will reduce the efficiency of the algorithm. So, to overcome this problem, machine learning algorithm called Decision Trees was implemented. The sheer size of the data would not pose a problem as Decision Trees requires a large database to produce good results and as a result the system becomes more robust and adaptable to different conditions.

The integration of Decision Tree was implemented by:

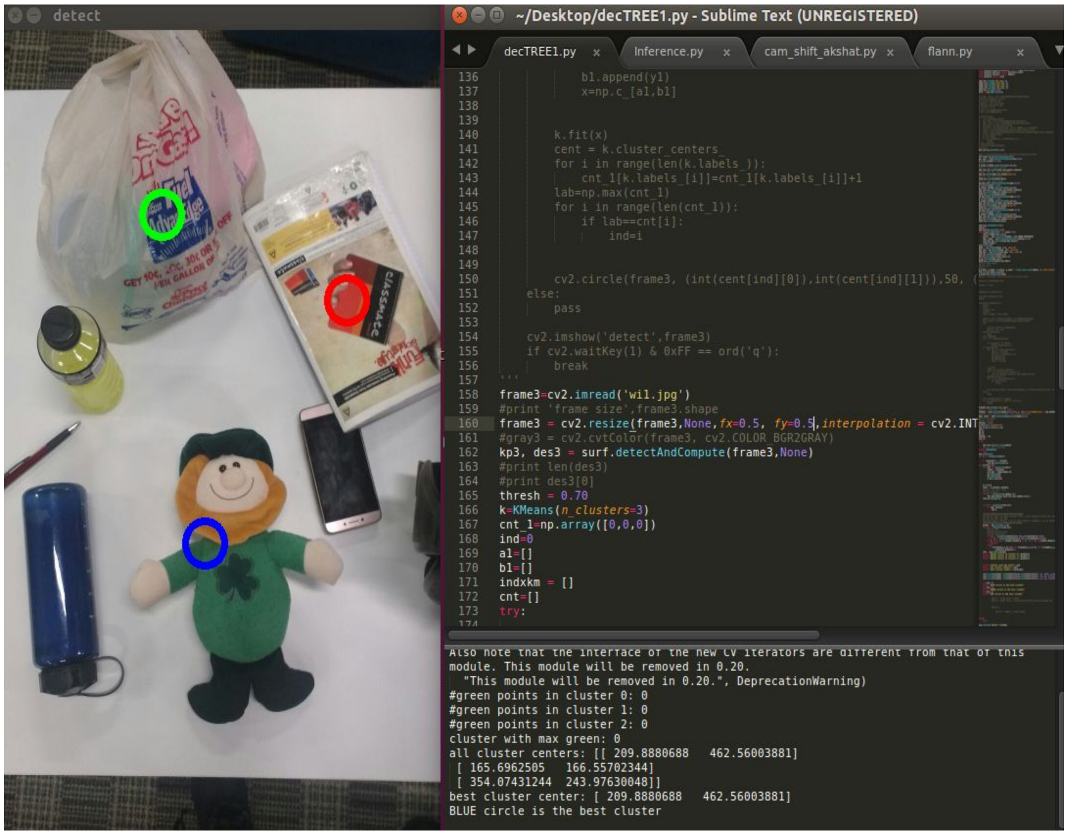
1. Training the Decision Trees model with descriptors of the Leprechaun, taken in different orientations and lighting, as the class of label 1.
2. Training the Decision Trees model with descriptors of the environment as the class of label 0.
3. Creating a probability index upon classification of a key-point that has maximum number of green pixels so that key-points with less probability can be ignored.

K-Means Clustering:

In order to further reduce the number of false positives, k-means clustering is used under the assumption that the maximum number of key-points will be generated at the centroid around which the number of green colored pixels are maximum. So by ignoring the remaining key-points outside the maximum number of green clusters, the number of false positives is further reduced.

Observations:

The number of clusters formed from K-Means was three each calculates the maximum number of green pixels around 100x100 pixels. If green pixel was inside a cluster it Increases the count of green pixels. The cluster with maximum number of green pixels was selected to be the best one. Hence,The Blue Cluster was the best cluster as it had maximum number of Green Pixels.



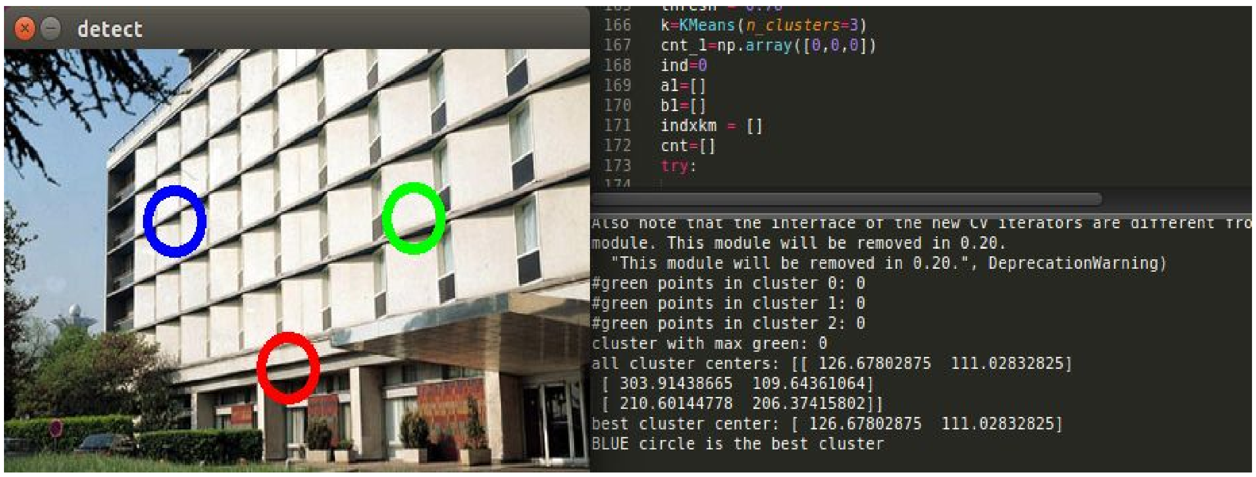
```
                b1.append(y1)
                x=np.c_[a1,b1]


            k.fit(x)
            cent = k.cluster_centers_
            for i in range(len(k.labels_)):
                cnt_1[k.labels_[i]]=cnt_1[k.labels_[i]]+1
            lab=np.max(cnt_1)
            for i in range(len(cnt_1)):
                if lab==cnt[i]:
                    ind=i


            cv2.circle(frame3, (int(cent[ind][0]),int(cent[ind][1])),50, (
        else:
            pass

        cv2.imshow('detect',frame3)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
'''
frame3=cv2.imread('wil.jpg')
#print 'frame size',frame3.shape
frame3 = cv2.resize(frame3,None,fx=0.5, fy=0.5,interpolation = cv2.INT
#gray3 = cv2.cvtColor(frame3, cv2.COLOR_BGR2GRAY)
kp3, des3 = surf.detectAndCompute(frame3,None)
#print len(des3)
#print des3[0]
thresh = 0.70
k=KMeans(n_clusters=3)
cnt_1=np.array([0,0,0])
ind=0
a1=[]
b1=[]
indxkm = []
cnt=[]
try:
```

```
Also note that the interface of the new CV iterators are different from that of this
module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
#green points in cluster 0: 0
#green points in cluster 1: 0
#green points in cluster 2: 0
cluster with max green: 0
all cluster centers: [[ 209.8880688   462.56003881]
 [ 165.6962505   166.55702344]
 [ 354.07431244  243.97630048]]
best cluster center: [ 209.8880688   462.56003881]
BLUE circle is the best cluster
```

Figure 12: Max cluster is found at the blue circle for green object

Drawbacks:

Since K-Means always form clusters, some clusters would always appear even if the
Object is not there. Even though we were able to reduce the false positives we were not able to completely remove it. Even when the number of green pixel is zero for each of the Cluster, it is giving output as blue is the best cluster which is one random decision. This was the major drawback of this technique and hence it was discarded.



```
k=KMeans(n_clusters=3)
cnt_1=np.array([0,0,0])
ind=0
a1=[]
b1=[]
indxkm = []
cnt=[]
try:
```

```
Also note that the interface of the new CV iterators are different fro
module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
#green points in cluster 0: 0
#green points in cluster 1: 0
#green points in cluster 2: 0
cluster with max green: 0
all cluster centers: [[ 126.67802875  111.02832825]
 [ 303.91438665  109.64361064]
 [ 210.60144778  206.37415802]]
best cluster center: [ 126.67802875  111.02832825]
BLUE circle is the best cluster
```

Figure 13: Shows how this method always finds clusters even when the object is not present in the frame.

4. Results

Technique	**Rotation Invariance**	**Scale Invariance**	**Partial Occlusion**	**Total Occlusion**	**Computation Time**
Camshift	Yes	Yes	Yes	No	~10ms
Decision Tree	Yes	No	No	No	~26-35 ms
SURF	Yes	Yes	Yes but number of matches decrease	No	~40-70ms depending on the number of matches
Neural Network	Yes	Yes	Confidence decreases	No	~85-95ms prediction time after training

5. Conclusion

We see that all the techniques have their own advantages and disadvantages. However, we see that the computation time for Camshift is much faster compared to others.

There are many people who worked on making camshift robust to occlusion. We can also try making these techniques robust to occlusion and clustered background to some extent.

Each technique can be implemented and improved depending on the requirements of the application.

6. References

- OpenCV Documentation - docs.opencv.org
- sklearn.tree.DecisionTreeClassifier on Scikit-learn
- Keras Documentation - keras.io , CS231 github.
- VGG16 model - https://gist.github.com/baraldilorenzo/

7. Appendix

Appendix A: Camshift Code:

```
#This code uses the CamShift algorithm to track any object selected by the user during run time.
#Usage: Run the code using python cam_shift_akshat.py.
#The camera feed will open up. Select the object you want to0 track using the mouse (click, drag, release using left click).
#You may press 'p' to pause the frame if you cannot select the object with live feed.
#You may press 'd' to finalise the selection
#You may press 'r' to reset and select again

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

#Create a VideoCapture object to get live feed from webcam
cam = cv2.VideoCapture(0)

#Initialize Variables
refPt = []
track_window = ()
Mode = 0
ranges = [[0, 180], [0, 255], [0, 255]]
thresh = [0, 0, 0]
kernel = np.ones((3,3),np.uint8)
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )

#Function gets RoI when user clicks and drags on object to be tracked using the mouse
def get_roi(event, x, y, flags, param):
    global refPt, track_window

    if event == cv2.EVENT_LBUTTONDOWN:
        refPt = [(x, y)]

    elif event == cv2.EVENT_LBUTTONUP:
        refPt.append((x, y))

        cv2.rectangle(img, refPt[0], refPt[1], (0, 255, 0), 2)
        cv2.imshow('Camshift', img)
```

```
cv2.namedWindow('Camshift')
cv2.setMouseCallback('Camshift', get_roi)

while(1):
    #Read the incoming camera frame
    ret, img = cam.read()

    #Add median blur
    img = cv2.medianBlur(img,5)

    #Duplicate for backup
    duplicate = img.copy()

    #Convert to HSV
    hsv_img = cv2.cvtColor(img.copy(), cv2.COLOR_BGR2HSV)

    #Mode = 0 until the user doesn't select an object
    if Mode == 0:
        key = cv2.waitKey(1) & 0xFF
        if key == ord('p'): #pause to select object
            key = cv2.waitKey(0) & 0xFF
        if key == ord('r'): #Reset
            img = duplicate.copy()
            refPt = []
        elif key == ord('d'): #Done selecting
            Mode = 1

            #Define Region of Interest
            roi = duplicate[refPt[0][1]:refPt[1][1], refPt[0][0]:refPt[1][0]]
            cv2.imshow('ROI', roi)

            #Initialize a track_window value
            track_window = (refPt[0][0], refPt[0][1], refPt[1][0], refPt[1][1])

            hsv = cv2.cvtColor(roi.copy(), cv2.COLOR_BGR2HSV)

            #Get the Histogram, normalize and get the dynamic threshold value
            for i in range(3):
                histr = cv2.calcHist([hsv],[i],None,[ranges[i][1]],ranges[i])
                if i==0:
                    histrr = histr
                cv2.normalize(histr, histr, ranges[i][0], ranges[i][1], cv2.NORM_MINMAX);
            thresh[i], max_value = max(enumerate(histr), key=operator.itemgetter(1))

    if Mode == 1:
        #Creating arrays to store the min and max HSV ranges
        lower = np.array([thresh[0]-30, thresh[1]-30, thresh[2]-40])
        upper = np.array([thresh[0]+30, thresh[1]+30, thresh[2]+40])

        #Create binary mask of in range object
        mask = cv2.inRange(hsv_img, lower, upper)

        #Morphology
        erosion = cv2.erode(mask,kernel,iterations = 2)
        dilation = cv2.dilate(erosion,kernel,iterations = 3)
        opening = cv2.dilate(dilation,kernel,iterations = 3)

        #Calculate Back Propogation
        dst = cv2.calcBackProject([hsv_img],[0],histrr,[0,180],1)

        #CamShift
        ret, track_window = cv2.CamShift(dst, track_window, term_crit)
        duplicate[opening == 0] = 0

        if ret[1][0] >= ret[1][1]:
            ret = (ret[0], (abs(ret[1][0]*1.5), abs(ret[1][1]*1.3)), ret[2])
        else:
            ret = (ret[0], (abs(ret[1][0]*1.3), abs(ret[1][1]*1.5)), ret[2])

        #Draw an ellipse
        cv2.ellipse(img, ret, (0, 0, 255), 2)

        cv2.imshow('Object', opening)
        k = cv2.waitKey(1) & 0xFF
        if(k==27):
            break
    cv2.imshow('Camshift', img)
```

Appendix B: SURF Code

```
//Include statements
#include <iostream>
#include <fstream>
#include <string>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/nonfree/nonfree.hpp"
#include <ros/package.h>
#include <string>
#include <ros/ros.h>
#include "std_msgs/String.h"
#include <chrono>
#include <ctime>


//Name spaces used
using namespace cv;
using namespace std;

int morph_elem = 1;
int morph_size = 5;

Mat canny_output;
vector<vector<Point> > contours;
vector<Vec4i> hierarchy;
Mat threshold_output;
int thresh=120;
RNG rng(12345);
vector<KeyPoint> kpVidImage;
Mat desImage,img_matches;

int main()
{

   //load training image
```

```
        Mat    object    =    imread("/home/shlok/catkin_ws/src/cv_surf/input_image_cropped.jpg",
CV_LOAD_IMAGE_GRAYSCALE);
  if (!object.data){
    cout<<"Can't open image";
    return -1;
  }

  //SURF Detector, and descriptor parameters
  vector<KeyPoint> kpObject;
  Mat desObject,kpObjectImage;

  //SURF Detector, and descriptor parameters, match object initialization
  int minHess=400;
  //Detect training interest points

 // namedWindow("Keypoints_SURF",CV_WINDOW_NORMAL);
 // resizeWindow("Keypoints_SURF",800,800);

  SurfFeatureDetector detector(minHess);
  detector.detect(object, kpObject);
                              drawKeypoints(object,kpObject,kpObjectImage,Scalar::all(-1),
DrawMatchesFlags::DEFAULT);
 // imshow("Keypoints_SURF",kpObjectImage);

  //Extract training interest point descriptors
  SurfDescriptorExtractor extractor;
  extractor.compute(object, kpObject, desObject);

  BFMatcher matcher(NORM_L2, true);
  std::vector<DMatch> matches;

  VideoCapture cap(0); // open the default camera
  if(!cap.isOpened())  // check if we succeeded
    return -1;

  Mat edges,segImage,morphImage,framegray;
  Mat element = getStructuringElement( morph_elem, Size( 2*morph_size + 1, 2*morph_size+1
), Point( morph_size, morph_size ) );

  int sensitivity=20;
 // namedWindow("edges",1);
  std::chrono::system_clock::time_point t1 = std::chrono::system_clock::now();
  for(;;)
  {

    t1 = std::chrono::system_clock::now();
    Mat frame;
    cap >> frame; // get a new frame from camera
    GaussianBlur(frame, edges, Size(7,7), 1.5, 1.5);
    cvtColor(edges, edges, CV_BGR2HSV);
    cvtColor(frame,framegray,CV_RGB2GRAY);
    inRange(edges,Scalar(0,100,150),Scalar(60+sensitivity,255,255),segImage);
    morphologyEx( segImage, morphImage, 3, element );
    //imshow("Morphed Image",morphImage);

    //imshow("edges", edges);
    //imshow("SegImage",segImage);

    /// Detect edges using Threshold
    threshold( segImage, threshold_output, thresh, 255, THRESH_BINARY );
    /// Find contours
                findContours( threshold_output, contours, hierarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );

    /// Approximate contours to polygons + get bounding rects and circles
    vector<vector<Point> > contours_poly( contours.size() );
    vector<Rect> boundRect( contours.size() );
    vector<Rect> boundRectLarge( contours.size() );
    vector<Point2f>center( contours.size() );
    vector<float>radius( contours.size() );

    for( int i = 0; i < contours.size(); i++ )
    { approxPolyDP( Mat(contours[i]), contours_poly[i], 3, true );
      boundRect[i] = boundingRect( Mat(contours_poly[i]) );


      //minEnclosingCircle( (Mat)contours_poly[i], center[i], radius[i] );
    }


    /// Draw polygonal contour + bonding rects + circles
    Mat drawing = Mat::zeros( threshold_output.size(), CV_8UC3 );
    for( int i = 0; i< contours.size(); i++ )
    {
      //if (contourArea( contours[i],false)>20)
      //   {// Find the area of contour
```

```
        Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
        drawContours( drawing, contours_poly, i, color, 1, 8, vector<Vec4i>(), 0, Point() );
        rectangle( drawing, boundRect[i].tl(), boundRect[i].br(), color, 2, 8, 0 );

        // }
    }

    // cout<<contours.size()<<endl;

    /// Show in a window
    namedWindow( "Contours", CV_WINDOW_AUTOSIZE );
    imshow( "Contours", drawing );

    double largest_area=1;
    int largest_contour_index=1;

    for( int i = 0; i< contours.size(); i++ ) // iterate through each contour.
    {
        double a=contourArea( contours[i],false);  //  Find the area of contour
        if(a>largest_area){
            largest_area=a;
            largest_contour_index=i;                //Store the index of largest contour

        }

    }
    Mat dst;
    if(largest_contour_index!=0 && largest_contour_index!=1)
    {
        // Scalar color( 255,255,255);
        Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
                drawContours( edges, contours_poly, largest_contour_index, color, 1, 8,
vector<Vec4i>(), 0, Point() );
                        rectangle( edges, boundRect[largest_contour_index].tl(),
boundRect[largest_contour_index].br(), color, 2, 8, 0 );
        dst = framegray(Rect(boundRect[largest_contour_index])).clone();
                                                                        //dst =
framegray(Rect(boundRect[largest_contour_index].tl().x,boundRect[largest_contour_index].tl().y
,boundRect[largest_contour_index].br().x,boundRect[largest_contour_index].br().y)).clone();
        //imshow("SegmentedCameraImage",dst);
    }
    namedWindow( "Biggest box", CV_WINDOW_AUTOSIZE );
    imshow( "Biggest box", edges );
```

```
        namedWindow( "matches", CV_WINDOW_AUTOSIZE );


        if (!dst.empty())
        {  //cout<<"entered loop"<<endl;
            detector.detect(dst,kpVidImage);
            extractor.compute(dst,kpVidImage,desImage);
            if(!desImage.empty())
            {
                matcher.match(desObject, desImage, matches);
                drawMatches(object, kpObject, dst, kpVidImage, matches, img_matches);
                if(matches.size()>10)
                {
                                                                    putText(img_matches,    "Object    Found",
cvPoint(10,50),FONT_HERSHEY_COMPLEX_SMALL, 2, cvScalar(0,0,250), 1, CV_AA);
                }
                imshow("matches", img_matches);
            }
        }


        char k=waitKey(30);
        if (k=='p')
        {
            imshow("grabbed frame",drawing);
            waitKey(0);

        }
        if (k=='q')
        {
            break;
        }

                                                                std::cout        <<        "time        "        <<
std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now()          -
t1).count() << "ms" <<std::endl;
    }

    return 0;
}
```

Appendix C: Neural Network:

Code for VGG_16 Model:
This code saves the model into h5 and json files along with the weights which can be loaded in the inference code for testing.

```
from __future__ import print_function
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D, ZeroPadding2D
from keras.optimizers import SGD
from keras.utils import np_utils
import numpy as np

batch_size = 20
nClasses = 2
dataAugmentation = False

###################################################
train_datagen = ImageDataGenerator()
test_datagen = ImageDataGenerator()

train_generator = train_datagen.flow_from_directory(
        '/home/rohit/cv/Training_Dataset/',
        target_size=(224, 224),
        batch_size=batch_size)

validation_generator = test_datagen.flow_from_directory(
        '/home/rohit/cv/Validation_Dataset/',
        target_size=(224, 224),
        batch_size=batch_size)
###################################################

model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(3,224,224)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(nClasses))

#model.load_weights("vgg16_model_11-20_epochs.h5")

#sgd = SGD(lr=0.001, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mse',
          optimizer='adam',
          metrics=['accuracy'])
```

```
history = model.fit_generator(
          train_generator,
          samples_per_epoch=2000,
          nb_epoch=2,
          validation_data=validation_generator,
          nb_val_samples=400,verbose = 1)

file = open("training_history.txt", "w")
file.write(str(history.history['acc']))
file.write(',')
file.write(str(history.history['val_acc']))
file.write(',')
file.write(str(history.history['loss']))
file.write(',')
file.write(str(history.history['val_loss']))
file.write(',')
file.close()

# serialize model to JSON
model_json = model.to_json()
with open("VGG16_Model.json", "w") as json_file:
          json_file.write(model_json)

# serialize weights to HDF5
model.save_weights("VGG16_Model.h5")
print("Saved model to disk")
```

Inference code:
This code loads the model and the weights and takes the input to predict the output

```
from keras.models import Sequential
from keras.layers import Dense
from keras.models import model_from_json
from keras.optimizers import SGD
import numpy as np
import os
import cv2
import time

# load json and create model
json_file = open('VGG16_Model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)

# load weights into new model
loaded_model.load_weights("VGG16_Model.h5")
print("Loaded model from disk")

# evaluate loaded model on test data
sgd = SGD(lr=0.001, decay=1e-6, momentum=0.9, nesterov=True)
loaded_model.compile(loss='mse',
        optimizer=sgd,
        metrics=['accuracy'])

#loaded_model.compile(loss='mse', optimizer='rmsprop', metrics=['accuracy'])

# Code to load the image from webcam
cam = cv2.VideoCapture(0)

print 'Press <q> to exit this code!'

while(1):
        start_time = time.time()
        ret, img = cam.read()

        # Code to load the image from local directory
        #im = cv2.imread('test_images/5.jpeg')
        im = img.copy()
        im = cv2.resize(im, (224, 224)).astype(np.float32)
```

```
    im = im.transpose((2,0,1))
    im = np.expand_dims(im, axis=0)

    out = loaded_model.predict(im)
    print out[0]

    if out[0][0] > 0.8 and out[0][1] > 0.022:
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(img,'Object Detected', (20,150), font, 2, (255,255,255), 2, cv2.CV_AA)

    cv2.imshow('Input', img)

    k = cv2.waitKey(33)

    print time.time() - start_time

    if k == 1048689:
cam.release()
break
```

Appendix D: Decision Tree Code

```
import cv2
from matplotlib import pyplot as plt
from sklearn import svm
import numpy as np
import random
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.cluster import KMeans
from sklearn import tree

img=cv2.imread('w1n.jpg',0)
img1=cv2.imread('w2n.jpg',0)
img2=cv2.imread('w3n.jpg',0)
img3=cv2.imread('w4n.jpg',0)
des0=np.zeros((1,32))
surf = cv2.ORB_create()
des_val=np.zeros((1,32))

####Training of Decision Tree Classifier

clf=tree.DecisionTreeClassifier()

####Making feature vector through descriptors
####Feature vector from images will be assigned class 1

kp, des = surf.detectAndCompute(img,None)
k=int(len(des)*0.25);
k_index=random.sample(range(0,k),k);
des_val=np.vstack((des_val,des[k_index]))
des=np.delete(des,(k_index),axis=0)
des0=np.vstack((des0,des))
kp,des=surf.detectAndCompute(img1,None)
k=int(len(des)*0.25);
k_index=random.sample(range(0,k),k);
des_val=np.vstack((des_val,des[k_index]))
des=np.delete(des,(k_index),axis=0)
des0=np.vstack((des0,des))
kp,des=surf.detectAndCompute(img2,None)
k=int(len(des)*0.25);
k_index=random.sample(range(0,k),k);
```

```
des_val=np.vstack((des_val,des[k_index]))
des=np.delete(des,(k_index),axis=0)
des0=np.vstack((des0,des))
kp,des=surf.detectAndCompute(img3,None)
k=int(len(des)*0.25);
k_index=random.sample(range(0,k),k);
des_val=np.vstack((des_val,des[k_index]))
des=np.delete(des,(k_index),axis=0)
des0=np.vstack((des0,des))

####Background is captured by switching on the camera for 20secs
####The features captured will be assigned class 0

vid=cv2.VideoCapture(0)
count=0
new_des=np.zeros((1,32))
while(vid.isOpened() and count<20):
   ret1,frame1=vid.read()
   ngray = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
   kp2, des2 = surf.detectAndCompute(ngray,None)
   new_des=np.vstack((new_des,des2))
   count=count+1
vid.release()
des0 = np.delete(des0, (0), axis=0)
des_val= np.delete(des0, (0), axis=0)
new_des = np.delete(new_des, (0), axis=0)
one=np.ones((len(des0),1))
zer=np.zeros((len(new_des),1))
o=np.vstack((one,zer))
des0=np.vstack((des0,new_des))

####To test accuracy
####Training and test data are split

X_train, X_test, y_train, y_test = train_test_split(des0, o, test_size=0.20)
clf.fit(X_train,y_train)
y_pred=clf.predict(X_test)
print 'accuracy',accuracy_score(y_test,y_pred)

####K-Means Implementation
frame3=cv2.imread('test_image2.jpg')
frame3 = cv2.resize(frame3,None,fx=0.2, fy=0.2,interpolation = cv2.INTER_CUBIC)
kp3, des3 = surf.detectAndCompute(frame3,None)
```

```
thresh = 0.70
k=KMeans(n_clusters=3)
cnt_1=np.array([0,0,0])
ind=0
a1=[]
b1=[]
indxkm = []
cnt=[]
try:

   out=clf.predict_proba(des3)

except Exception:
   pass
cnt.append(1)
for i in range(len(out)):

   if out[i][1] >= thresh:
      cnt=np.append(cnt,i)
if len(cnt)>10:
   for mat in range(len(cnt)):
      index = cnt[mat]
      (x1,y1) = kp3[index].pt
      a1.append(x1)
      b1.append(y1)
      x=np.c_[a1,b1]


   k.fit(x)
   cent = k.cluster_centers_

   for i in range(len(k.labels_)):
      cnt_1[k.labels_[i]]=cnt_1[k.labels_[i]]+1
   lab=np.max(cnt_1)


   for i in range(len(cnt_1)):
      if lab==cnt[i]:
         ind=i

   ########################################
   ## OpenCV takes the Blue channel first, green channel second and red channel third.
   ## its not RGB. Its BGR
```

```
    ## which means frame3[:,:,0] is blue channel, frame3[:,:,1] is green channel, frame3[:,:,2] is
    red channel.!!!
    ######################################
    green = np.array([0,0,0])
    for i in range(len(cnt_1)):

        for x in range(int(cent[i][0]-50),int(cent[i][0]+50)):
            for y in range(int(cent[i][1]-50),int(cent[i][1]+50)):
                    if x < 0 or x >= frame3.shape[0] or y < 0 or y >= frame3.shape[1]:
                      continue
                    else:
                        if  ((frame3[x,y,0]>30)  and  (frame3[x,y,0]<50)  and  (frame3[x,y,1]>119)  and
(frame3[x,y,2]<25)) :
                                green[i]=green[i]+1
    ind = np.argmax(green)
    print "#green points in cluster 0:",green[0]
    print "#green points in cluster 1:",green[1]
    print "#green points in cluster 2:",green[2]


    print "cluster with max green:",ind
    print "all cluster centers:",cent
    print "best cluster center:",cent[ind]

    cv2.circle(frame3, (int(cent[0][0]),int(cent[0][1])), 20, (255, 0,0), 5)   # Blue circle
    cv2.circle(frame3, (int(cent[1][0]),int(cent[1][1])), 20, (0, 255,0), 5)   # Green Circle
    cv2.circle(frame3, (int(cent[2][0]),int(cent[2][1])), 20, (0, 0,255), 5)   # Red circle

    if ind==0:
          print "BLUE circle is the best cluster"
    if ind==1:
          print "GREEN circle is the best cluster"
    if ind==2:
          print "RED circle is the best cluster"




else:
    pass

cv2.imshow('detect',frame3)
cv2.waitKey(0)
```