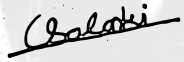
CS/RBE 549 Computer Vision
Fall 2015
Final Project Report

"Where's the Pokari Can ?"

Team Jaguar

Member	Signature	Contribution (%)
Aman Rana		25%
Rohan Kothari		25%
Syamprasad Karyattuparambil Rajagopalan		25%
Ulkesh Solanki		25%

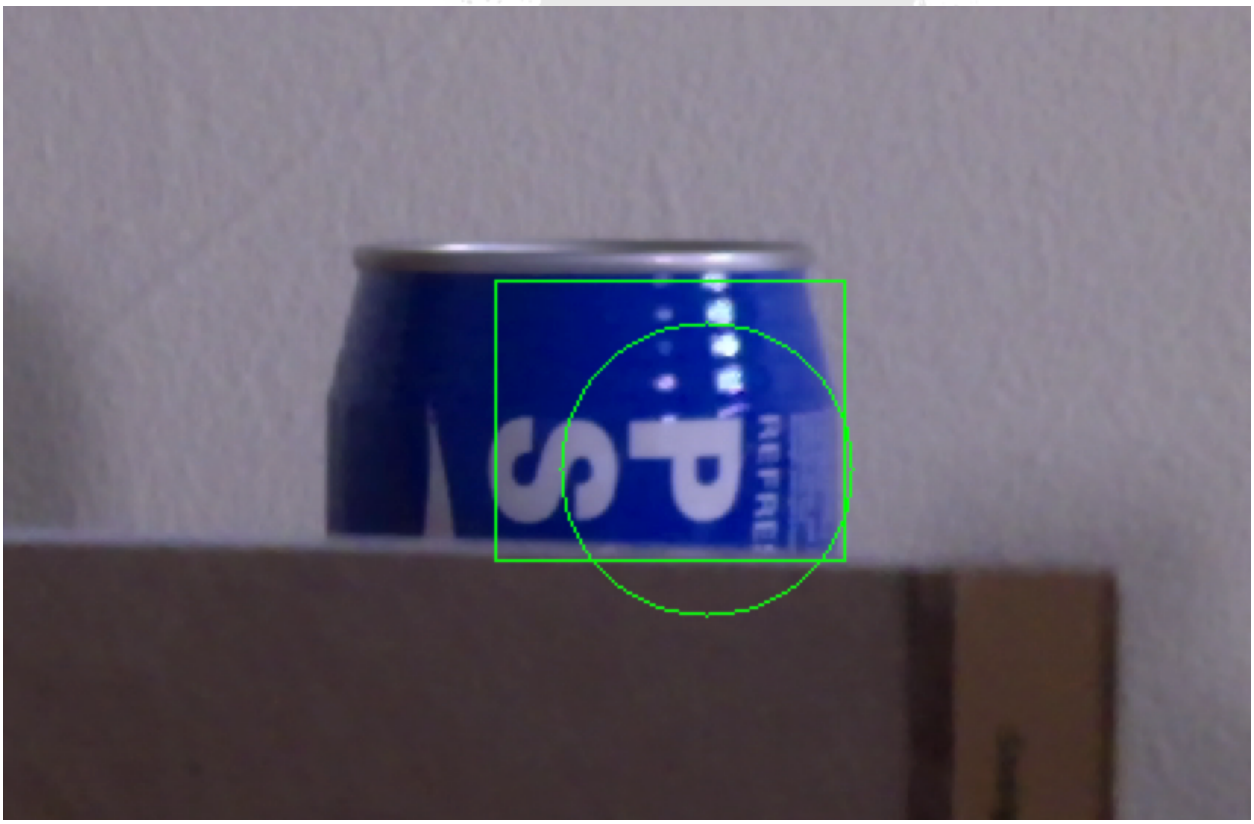
Grading:		
	Approach	_____/20
	Justification	_____/15
	Analysis	_____/20
	Testing & examples	_____/15
	Documentation	_____/15
	Presentation	_____/10
	Difficulty	_____/5
	Total	_____/100

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

This project involved the design and implementation of a computer vision algorithm to be used for object detection – one of the major problems in computer vision. The primary aim of the project was to detect a Soda Can provided to us in various environment. Initially various available techniques were discussed to find the perfect match for the task at hand. SURF Algorithm seemed to be the best for our purpose. Program was written using openCV, implementing SURF. While testing it was found that the object was detected using surf successfully but only in an environment which is less cluttered and didn't have much noise. Its performance reduced a lot as the background became more chaotic. In order to overcome this, we decided to use cascade of 2 algorithms, Color detection in HSV space and SURF detection. The HSV detects all the blue (which is the color of the CAN) objects in surrounding. On these detected objects, SURF is implemented. The above described leads to a tremendous increase in accuracy as compared to using only SURF. We were able to detect the object even in the case of partial occlusion. The results can be seen in the Images below. The limitation in our approach is the lighting conditions, if there is drastic change in lighting conditions then HSV filter might not detect the can and thus SURF will not detect the object. In drastic it condition we will need to modify the filter manually for the Algorithm to give good accuracy.



1. INTRODUCTION

Objective

The primary objective is to detect the given blue colored ‘Pokari Sweat’ can, high accuracy, under the following conditions:

. a) In arbitrary orientation
. b) While the can is partially occluded and
. c) In a cluttered environment.

The detection of an object can be made possible through multiple techniques. The most suitable technique for the detection must be chosen based on the features available in the object. Fig1 shows the object in a fairly white background. The can is primarily of blue color with writings in white on it.

Fig 1.1. Pokari Sweat Can to be detected

The project has been implemented on OpenCV 2.4.x with C++ using color based detection and Speeded Up Robust Features (SURF) algorithm.

2. AVAILABLE OBJECT DETECTION TECHNIQUES

Potential Detection Technique

The detection techniques under consideration for the project were:

- **Edge based Detection**

  The object is capable of providing good edges that can be easily detected. But the disadvantage of the approach is that this algorithm will not be able to distinguish the given can from other similar cans, since they all have similar shape and edges.

- **Hough Transform**

  The Hough Transform algorithm works well in detection of lines and circles. Again, this algorithm has the same disadvantage as that of the first approach and is altogether not suitable for the given object.

- **Machine Learning Approach**

  The machine learning approach trains the computer system using a large number of images that may or may not contain object. This approach requires a set of positive and negative images. A positive image is an image which contains the object against some background. A negative image is an image which does not contain the image. One the training approaches that can be used for this approach is the Haar training method, available in OpenCV. The disadvantage of this method is that it requires a large database of images; also, this is primarily a machine-learning based approach.

- **Color Detection**

  One of the dominating features of the given object is its color: blue. The obvious disadvantage with this method is that it would detect any blue object present in the scene.

- **Speeded Up Robust Features (SURF)**

  The SURF algorithm provides good detection based on the features extracted from the reference image of the object. The disadvantage of this algorithm is that it may detect a lot of unwanted features depending on the input scene. Also it requires higher computational resources to search for the features.

3. APPROACH

Software Platform

The first consideration for the project was the platform to be used. The two popular options available were MATLAB and OpenCV. MATLAB has extensive documentation on the topic and was easier to implement the project. Unfortunately, this platform is not open source. OpenCV is open source and provides two options for coding: Python and C++. The disadvantage of OpenCV is limited supporting resources, compared to MATLAB. Considering the future prospects and the educational value, OpenCV 2.4.x with C++ was chosen as the platform.

The team started with detection using SURF. But soon found that there were lot of false detections. Not happy with the results, it was decided to use a combination of color detection and SURF algorithm. Initially, the color based detection will identify the blue colored regions in the scene, which will naturally include blue objects. Then a sub-frame will be extracted that bounds these blue objects. These sub-frames will be passed to the SURF algorithm. Now the SURF will match the features of each of these objects with that of the reference image to detect the can. If the object is available in the su-frame, the main() function will draw a rectangle around the can sub-frame. This approach eliminates the disadvantages of the two algorithms as discussed above.

Color Based Detection

The most commonly used color model is RGB (used as BGR in OpenCV), where three channels of primary colors red, green and blue are used in image formation. For the color detection, this RGB model is converted to HSV (Hue, Saturation and Value) model. The Hue depends on the dominating color present in the object while saturation and value may depend on the environment and lighting conditions. Once the HSV model is obtained, the hue value for BLUE can be set to detect the blue colored objects present in the environment. This value falls in the range 75-130. The Saturation and Value should be adjusted to suit the environment and lighting conditions. This can be done using a blue colored test object.

SURF

The SURF algorithm requires a reference image which contains only the object. Fig2 shows the reference image used for this project. The algorithm first detects the key points (interest points) in the image. Now the surf feature descriptor is created such that it will provide a unique and robust description of the feature. They are created based on the response of Haar wavelet in the neighborhood of point of interest. An example of a descriptor is the variation of intensity values in the neighborhood of interest point. These reference image descriptors are stored in the system. When the computer receives an input frame/scene image, the surf algorithm repeats the process and creates the descriptor for the input. These feature descriptors are matched with those of the reference; if a good number of match is obtained, the algorithm will be able to detect the object.

Fig 3.1. Reference Image to detect the CAN

4. OBJECT DETECTION USING SURF

SETUP

Reference image of the POKARI SWEAT CAN is fed to the program. And camera is used to capture the image of the surrounding. We need to detect the object from the surroundings. Can is placed in various surroundings such as white background, blue background, cluttered environment etc.

RESULTS

As we saw that surf is a very powerful algorithm. It uses various features of the object and stores them as descriptors. These descriptors are matched with descriptors in the image which is captured from live feed. If the descriptors match above certain threshold value then we say that the object is detected. As we humans remember a face with help of certain features such skin tone, facial hair, eyes, nose etc. similarly the features of the object are stored and used for detection.

The program was working well enough in the white background and the Can was detected easily. But results began to depreciate when the surroundings became more and more cluttered.

As the background became more cluttered the number of false detections increased. As Seen in the image below the features of the can are detected in the background. These lead to false classification and thus we were not able to detect the CAN.

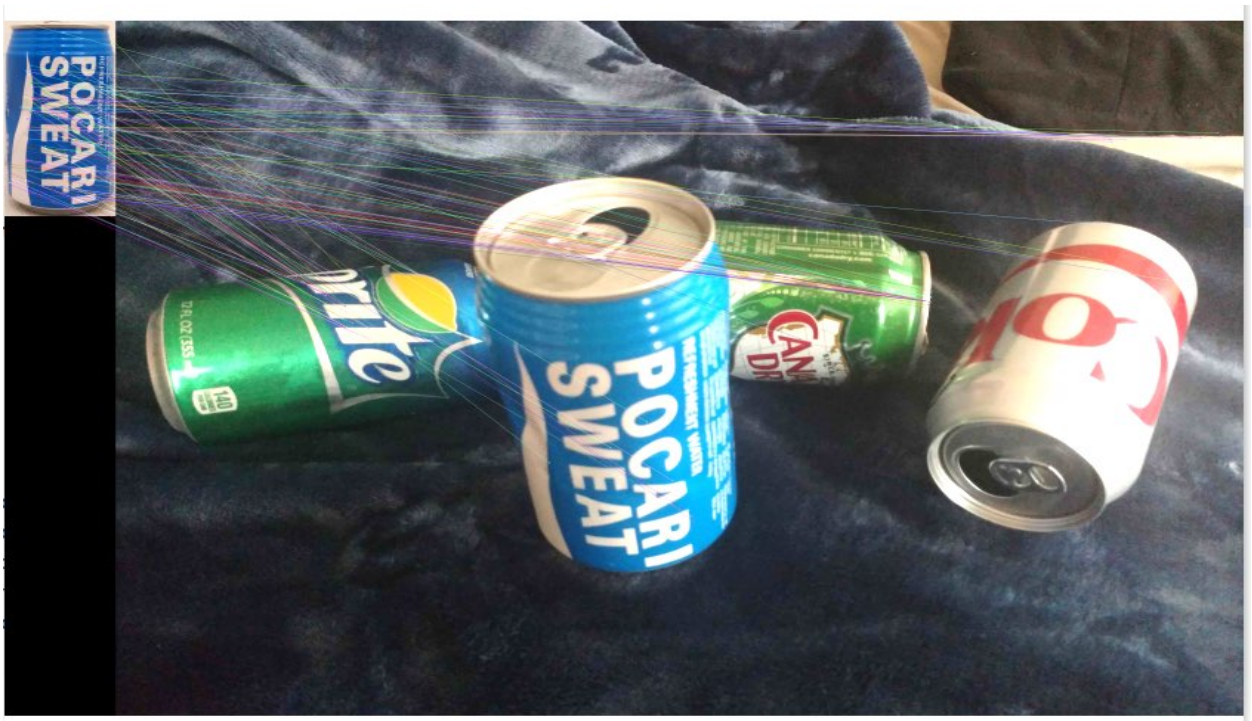


Fig 4.1. SURF false detections

REASONS FOR FAILURE

The major cause of failure was that, the algorithm is run over entire image i.e. too many false detections. The image consists of background which is the largest portion in most of the cases. So the possibility of having a false detection increases.

SOLUTION

The basic problem of false detection is due to the background. Our solution is to try and remove the background using the properties of the Can. The property that we exploited was the color. The blue color of the can was detected in HSV color space. HSV space was chosen because HSV separates the image intensity, from the color information. We tried with different threshold values and were able to reach an optimum threshold which would detect blue color in different lighting conditions. So the program would detect any blue object in the image as shown in the image below.

Fig 4.2. Reference Image to detect the CAN

So now SURF algorithm is performed only on the objects which are blue in color as shown in the figure below. Thus the possibility of a feature being false detected reduces. Hence improving the overall efficiency of the program.

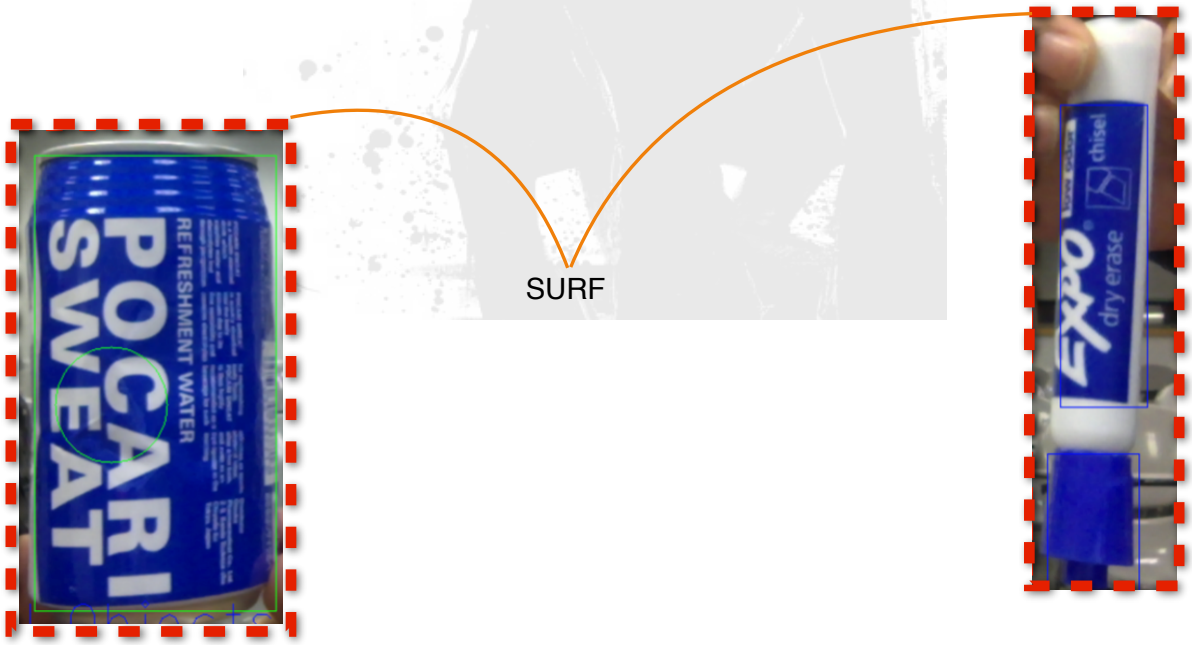


Fig 4.3. Extracted Sub-Frames

5. GETTING THE SUB-FRAMES

a. Get a frame from the video feed

Fig 5.1. Reference Image for webcam

b. Convert the frame to HSV color mode

The inRange function in openCV was used to convert the BGR image to HSV mode.

```
cvtColor(frame, frame_hsv, COLOR_BGR2HSV);
```

Fig 5.2. BGR to HSV mode

c. Image Thresholding

The *imgRange* function is used to threshold the image between HSV_{min} and HSV_{max} values. We have chosen the HSV min and max values to segment only the blue objects.

```
inRange(frame_gray, Scalar(h_min, s_min, v_min), Scalar(h_max, s_max, v_max), frame_gray);
```

This feature allows us to detect multiple blue objects and take multiple sub-frames containing blue objects.

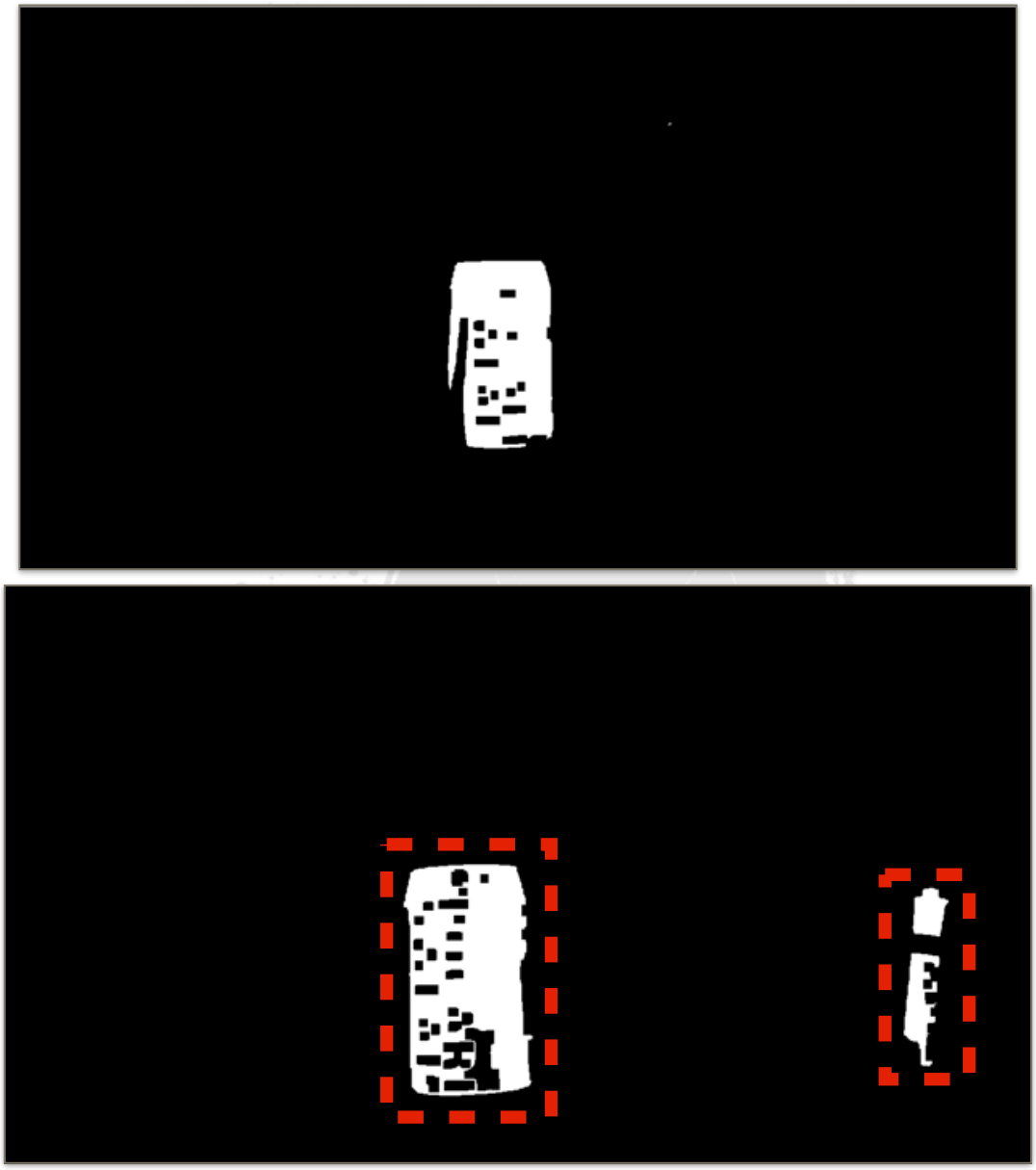


Fig 5.3. Thresholded images and bounding rectangles

d. Image Morphology

Here we performing closing (dilation and erosion) three times to eliminate all the holes in the thresholded frame.

```
Mat strucElement = getStructuringElement(0, Size(10,10));
dilate(frame_gray, frame_gray, strucElement);
erode(frame_gray, frame_gray, strucElement);
```

} X 3

e. Find Contours

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

In OpenCV, finding contours is like finding white object from black background.

```
findContours(frame_gray, contours, hierarchy, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
```

f. Getting the bounding rectangle around the contour

We aim to get the minimum size of the rectangle bounding the contour (i.e. surrounding the white part in the thresholded image). This is accomplished using the boundingRect(contours) function in openCV. This function takes the contours as an argument and returns the properties of the bounding rectangle around the contour.

```
Rect boundingRectData = boundingRect(contours[i]);
```

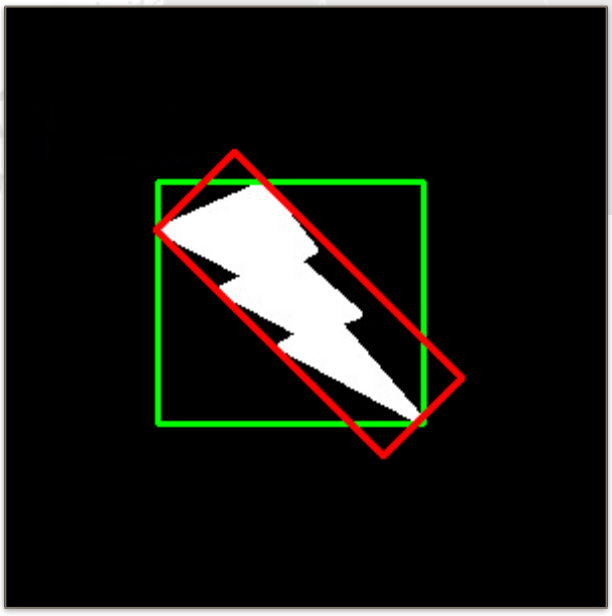


Fig 5.4. Bounding rectangles

g. Extracting the sub-frame using the bounding rectangle

Having found the bounding rectangle, we can now extract the sub-frame using the corner coordinates of the bounding rectangle.

```
Mat temp = frame(Rect( boundingRectData.tl().x, boundingRectData.tl().y, boundingRectData.br().x - boundingRectData.tl().x, boundingRectData.br().y - boundingRectData.tl().y ) );
```

Here, temp is a matrix containing only that part of the main frame that has the blue object.

h. Passing the sub-frame to SURF function

We pass this sub-frame matrix to the SURF function and expect a boolean value telling us whether the frame contains the Pokari Sweat can or not. If the returned value is TRUE, a green rectangle is drawn around the frame containing the Pokari Sweat Can. If the return value is FALSE, then a blue rectangle is drawn around the sub-frame indicating potential object which is blue in color. There are also 2 other integer values expected. If the can is detected in the sub-frame, then the centroid is also calculated by the SURF function and returned as a tuple.

```
tuple<int, int, bool> SurfReturnData = SurfMain(temp);

if POKARI CAN present in SUB-FRAME:
    rectangle(frame, Point(boundingRectData.x - ractangleBoundsConstant, boundingRectData.y - ractangleBoundsConstant), Point(boundingRectData.x + boundingRectData.width + ractangleBoundsConstant, boundingRectData.y + boundingRectData.height + ractangleBoundsConstant), Scalar(0,255,0), 2); // draws rectangle in green color
else
    rectangle(frame, Point(boundingRectData.x - ractangleBoundsConstant, boundingRectData.y - ractangleBoundsConstant), Point(boundingRectData.x + boundingRectData.width + ractangleBoundsConstant, boundingRectData.y + boundingRectData.height + ractangleBoundsConstant), Scalar(255,0,0), 2); // draws rectangle in blue colour
```

6. OBJECT DETECTION IN VARIOUS CONDITIONS

The environmental condition in which the object is to be detected is never known. So it is better to design an algorithm which can detect the desired object in any environmental condition. We have tried the same thing. Multiple modifications were done in the basic code to make it work under vulnerable conditions. In this section let's have a look how our algorithm reacts in various conditions

- **Cluttered Environment**

  Humans may also struggle sometimes to find the desired object in a cluttered environment. But our algorithm can easily detect the object in such conditions. Successful object detection in a cluttered environment can be seen in figure 1.

Fig. 6.1 Object detection in cluttered environment

In this case according to the flow of the algorithm, initially the thresholded (blue) sub-frames are extracted from the image.

- **Partial Occlusion**

As SURF is included in our algorithm partially occluded objects can also be detected. It is possible to detect the object which is occluded as much as 60%. Based on the key points matched in the partially occluded image, SURF is able to detect the object.

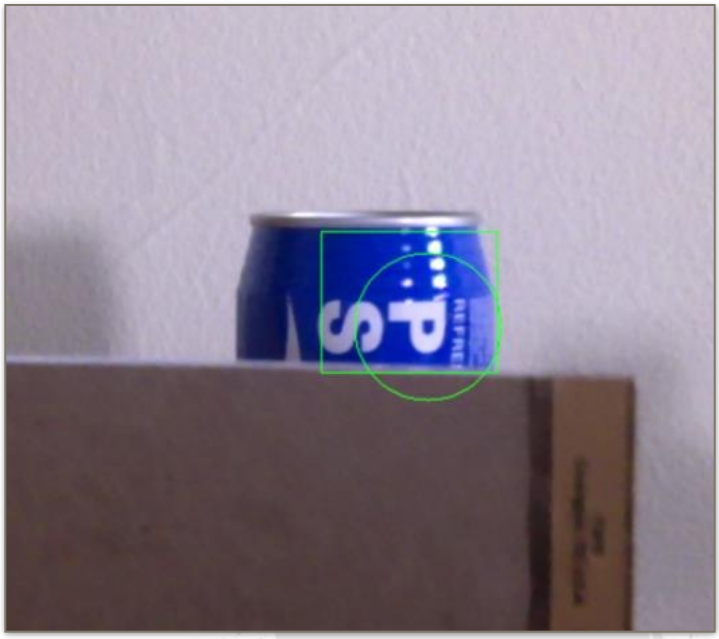


Fig. 6.2. 75 % occlusion.

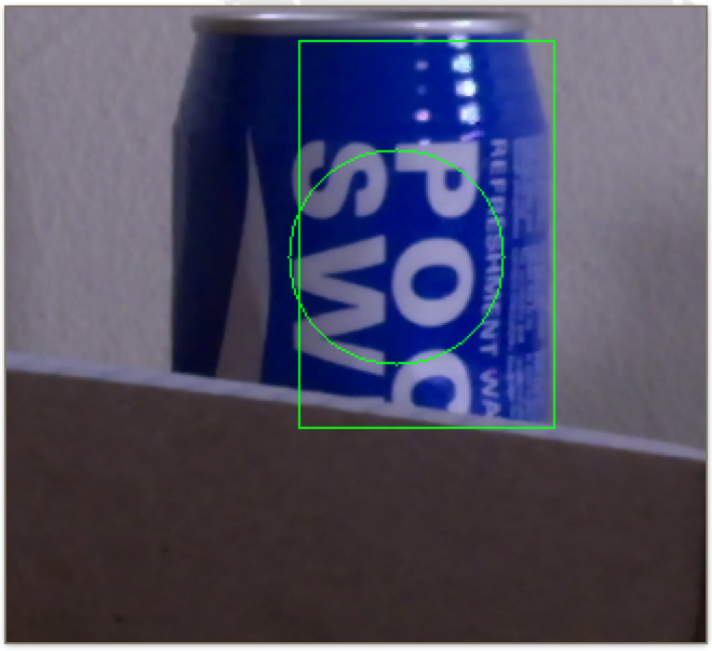


Fig. 6.3. 50 % occlusion.

- **Presence of other blue objects**

The HSV thresholding based approach work fine when there are no blue objects other than the desired object. Object detection in presence of other blue objects is shown in figure 3. When there are other blue objects in the image, initially the thresholded (blue) sub-frames are extracted from the image. All these sub-frames are treated as potential objects as they are similar in color to the desired object. These potential objects are enclosed in a blue rectangle. Further SURF is performed over these sub-frames and key-points are detected. Based on the maximum matches found with the key-points of the reference image, the sub-frame is treated as object and is enclosed in a green rectangle.

Fig. 6.4. Object detection in presence of other blue objects

- **Presence of blue background**

  It becomes challenging to detect the object in presence of the blue background of same shade as that of the object. When the object is surrounded by the blue background, the entire background along with the object is treated as a single sub-frame. Here a problem arises as we cannot extract out the object from the sub-frame according to the algorithm discussed so far. But a small extension to our algorithm can detect the object, even in such a hard condition. Object detection in presence of other blue objects is shown in figure 4. As discussed earlier, initially the HSV thresholded (blue) sub-frame is extracted out of the image. In this case the entire blue background is been extracted. Further SURF detects that there is the desired object in this sub-frame and encloses it in a green rectangle. But to be more specific we need to locate the object within this huge sub-frame. To do so we came up with the concept of detection based on centroid of the key-points. According to this concept the average coordinates of both x and y components of all the key-points with respect to the sub-frame are calculated. Ultimately we get centroid of all the key-points and finally this centroid is located by enclosing it in a green circle. In this way we have successfully detected the object in the presence of the blue background.

Fig. 6.5. Object detection in presence of other blue objects

7. LIMITATIONS

Our algorithm can detect the object in almost any conditions. But still it has some limitations.

- We cannot detect our object successfully when it is rotated about its own axis (i.e. axis of the cylindrical body of the can). This is because of the less number of key-points detection in other orientations.
- Sometimes, the program draws a green rectangle around a sub-frame which doesn't contain the frame.

We plan to include more than one reference images from different angles of the can -front, sides and back.

8. CONCLUSION

We have achieved the goals of detecting our object:

- in any arbitrary orientation about one axis
- under partially occluded state
- in the presence of other object similar in shape or color
- and also in a cluttered environment.

with reasonably good efficiency.

Fig. 8.1. Object detected

REFERENCES

- https://en.wikipedia.org/wiki/Speeded_up_robust_features
- http://opencv.org/documentation.html
- H. Bay, A. Ess, T. Tuytelaars and L. van Gool, Speeded-Up Robust Features (SURF), Computer Vision and Image Understanding 110 (2008) 346–359.
  https://www.vision.ee.ethz.ch/en/publications/papers/articles/eth_biwi_00517.pdf
- http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html
- http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html

APPENDIX - CODE

To keep the program simple to understand and debug, it has been divided into separate files:

- main.cpp
- TrackerBar.cpp
- TrackerBar.hpp
- Misc.cpp
- Misc.hpp
- SurfDetection.cpp
- SurfDetection.hpp

9.1. main.cpp

```
#include <iostream>
#include <tuple>
#include "TrackerBar.hpp"
#include "Misc.hpp"
#include "SurfDetection.hpp"

using namespace cv;
using namespace std;

const int FRAME_WIDTH = 640;
const int FRAME_HEIGHT = 480;
const int MIN_OBJECT_AREA = 30 * 30;
const int ractangleBoundsConstant = 0;

// Hue values range
int h_min = 0;
int h_max = 180;

// saturation values range
int s_min = 0;
int s_max = 255;

// Value values range
int v_min = 0;
int v_max = 255;

vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
Mat frame, frame_gray;
```

```
bool changePermission = true; // this allows the user to change
the HSV-min and HSV-max values using trackerBar.

// GLOBAL VARIABLES END

int main() {

    getRefImageAndPerformSURF();

    VideoCapture cap(0);

    cap.set(CV_CAP_PROP_FRAME_WIDTH, FRAME_WIDTH);
    cap.set(CV_CAP_PROP_FRAME_HEIGHT, FRAME_HEIGHT);

    createAWindow();
    createTheTrackerbar();

    for(;;) {

        cap >> frame;
        cvtColor(frame, frame_gray, COLOR_BGR2HSV);
        imshow("HSV image", frame_gray);
           // Getting the thresholded image between the HSV-min
values and HSV-max values.
               inRange(frame_gray, Scalar(h_min, s_min, v_min),
Scalar(h_max, s_max, v_max), frame_gray);

        // dilates and closes the thresholded image 3 times.
        closeImg();
        closeImg();
        closeImg();

        //if (changePermission == true) {
            imshow("B/W Image", frame_gray);
        //}

        // Runs only if the program is not in configeration mode
        if (changePermission != true) {

              // Finds the contours in each frame taken from the
webcam
                  findContours(frame_gray, contours, hierarchy,
CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);

             // Sorting the contour array in descending order of
area (i.e. Largest first, smallest last)
```

```
                        sort(contours.begin(), contours.end(),
compareContourAreas);

            unsigned long int contourSize = contours.size();

            if (contourSize > 0) {

                //cout << (int)contourSize << endl;

                for (int i= 0; i < contourSize; i++) {

                                    if (contourArea(contours[i]) >=
MIN_OBJECT_AREA) {

                                // this functions automatically creates
a rectangle bounding the contour and then we can extract the
top-left point and the bottom-right point.
                                                Rect boundingRectData =
boundingRect(contours[i]);

                                // Passing the subFrame to perform SURF
detection
                                                        Mat temp =
frame(Rect(boundingRectData.tl().x, boundingRectData.tl().y,
boundingRectData.br().x - boundingRectData.tl().x,
boundingRectData.br().y - boundingRectData.tl().y));

                                    // Passing the subFrame(-> temp) to
perform SURF detection and if it returns true, then draw the
rectangle around the object.
                                tuple<int, int, bool> SurfReturnData =
SurfMain(temp);
                            if (get<2>(SurfReturnData) == true) {

                                    putText(frame, "Object in the green
rectangle. Exactly located by the circle.", Point(10,50), 1,
2.0, Scalar(0,255,0), 2);
                                                        rectangle(frame,
Point(boundingRectData.x - ractangleBoundsConstant,
boundingRectData.y -    ractangleBoundsConstant),
Point(boundingRectData.x + boundingRectData.width +
ractangleBoundsConstant, boundingRectData.y +
boundingRectData.height + ractangleBoundsConstant),
Scalar(0,255,0), 2); // draws rectangle in green colour

                                int x = get<0>(SurfReturnData);
```

```
                            int y = get<1>(SurfReturnData);

                            if ((x != 0 && y != 0)) {
                                                circle(frame,
Point(boundingRectData.x + x, boundingRectData.y + y), 50,
Scalar(0,255,0), 2);
                            }

                         } // end of if (SurfMain(temp, frame) ==
true)
                        else {

                            // This gets printed to the frame
                             putText(frame, "Potential Objects in
BLUE.", Point(10, frame.rows - 50), 1, 2.0, Scalar(255,0,0), 2);

                                                rectangle(frame,
Point(boundingRectData.x - ractangleBoundsConstant,
boundingRectData.y -    ractangleBoundsConstant),
Point(boundingRectData.x + boundingRectData.width +
ractangleBoundsConstant, boundingRectData.y +
boundingRectData.height + ractangleBoundsConstant),
Scalar(255,0,0), 2); // draws rectangle in blue colour
                                        // this text is written for
debugging purposes. This gets printed in the console.
                                    cout << "SURF Returned false. ->
Object not in frame -> OR -> wrong object passed to SURF
function.." << endl; // DEBUGGING


                        } // end of else

                    } // end of if (contourArea(contours[0]) >=
MIN_OBJECT_AREA)

                } // end of for (int i= 0; i < contourSize; i++)

                //imshow("Threshold Image", frame);

            } // end of if (contourSize > 0)

            imshow("Threshold Image", frame);

        } // and of if (changePermission != true)

        imshow("Threshold Image", frame);
```

```
        // CONFIGURATION OPTIONS
        int wait = waitKey(30);
        if(wait >= 0) {

            if (wait == 99) {

                 cout << h_min << " " << s_min << " " << v_min <<
" " << h_max << " " << s_max << " " << v_max << endl;
                changePermission = !changePermission;

                destroyTheWindow();
                destroyWindow("Thresholded Image");
                createAWindow();

                if (changePermission == false) {
                    cap.set(CV_CAP_PROP_FRAME_WIDTH, FRAME_WIDTH
* 2);
                                cap.set(CV_CAP_PROP_FRAME_HEIGHT,
FRAME_HEIGHT * 1.5);
                }

                if (changePermission == true) {
                    createTheTrackerbar();
                    loadValuesToTrackBar();
                                 cap.set(CV_CAP_PROP_FRAME_WIDTH,
FRAME_WIDTH);
                                cap.set(CV_CAP_PROP_FRAME_HEIGHT,
FRAME_HEIGHT);

                } // end of if (changePermission == true)


            } // end of if (wait == 99)
            else {
                 cout << h_min << " " << s_min << " " << v_min <<
" " << h_max << " " << s_max << " " << v_max << endl;
                break;
            } // end of else

            // CONFIGURATION OPTIONS END

        } // end of if(wait >= 0)
    } // end of for(;;)

    destroyAllWindows();
```

```
    return 0;
}
```

9.2. TrackerBar.cpp

```
#include "TrackerBar.hpp"

// GLOBAL VARIABLES
extern int thresholdValue;

// Hue values range
extern int h_min;
extern int h_max;

// saturation values range
extern int s_min;
extern int s_max;

// Value values range
extern int v_min;
extern int v_max;

// GLOBAL VARIABLES END

void hueMinValueChange(int value, void* userInfo) {

    h_min = value;
}

void hueMaxValueChange(int value, void* userInfo) {

    h_max = value;
}

void saturationMinValueChange(int value, void* userInfo) {

    s_min = value;
}

void saturationMaxValueChange(int value, void* userInfo) {

    s_max = value;
}
```

```
void valueMinValueChange(int value, void* userInfo) {

    v_min = value;
}

void valueMaxValueChange(int value, void* userInfo) {

    v_max = value;
}

void loadValuesToTrackBar() {

    setTrackbarPos("Hue Min:", "Threshold Image", h_min);
    setTrackbarPos("Hue Max:", "Threshold Image", h_max);
    setTrackbarPos("Saturation Min:", "Threshold Image", s_min);
    setTrackbarPos("Saturation Max:", "Threshold Image", s_max);
    setTrackbarPos("Value Min:", "Threshold Image", v_min);
    setTrackbarPos("Value Max:", "Threshold Image", v_max);
}

void createTheTrackerbar() {

         createTrackbar("Hue  Min:",  "Threshold  Image",  0,  180,
hueMinValueChange);
         createTrackbar("Hue  Max:",  "Threshold  Image",  0,  180,
hueMaxValueChange);

     createTrackbar("Saturation Min:", "Threshold Image", 0, 255,
saturationMinValueChange);
     createTrackbar("Saturation Max:", "Threshold Image", 0, 255,
saturationMaxValueChange);

        createTrackbar("Value  Min:",  "Threshold  Image",  0,  255,
valueMinValueChange);
        createTrackbar("Value  Max:",  "Threshold  Image",  0,  255,
valueMaxValueChange);

}
```

9.3. TrackerBar.hpp

```
#ifndef TrackerBar_hpp
#define TrackerBar_hpp

#include <stdio.h>
#include <stdio.h>
#include <iostream>
#include "opencv2/highgui/highgui.hpp"

using namespace cv;
using namespace std;

void hueMinValueChange(int value, void* userInfo);
void hueMaxValueChange(int value, void* userInfo);
void saturationMinValueChange(int value, void* userInfo);
void saturationMaxValueChange(int value, void* userInfo);
void valueMinValueChange(int value, void* userInfo);
void valueMaxValueChange(int value, void* userInfo);
void loadValuesToTrackBar();
void createTheTrackerbar();



#endif /* TrackerBar_hpp */
```

9.4. Misc.cpp

```
#include "Misc.hpp"

void closeImg() {
    Mat strucElement = getStructuringElement(0, Size(10,10));
    dilate(frame_gray, frame_gray, strucElement);
    erode(frame_gray, frame_gray, strucElement);
}

bool  compareContourAreas(vector<Point>  contour1,  vector<Point>
contour2) {
    double i = fabs( contourArea(cv::Mat(contour1)) );
    double j = fabs( contourArea(cv::Mat(contour2)) );
    return ( i > j );  // sorts in descending order
    //return true;
}
```

```
//int* findLargeContourAreas(vector<vector<Point>> _contours) {
//    int largestArea = 0, pos[5];
//        sort(_contours.begin(), _contours.end(),
compareContourAreas);
//    pos.
//    return pos;
//}

void createAWindow() {
    namedWindow("Threshold Image", CV_WINDOW_OPENGL);
}

void destroyTheWindow() {
    destroyWindow("Threshold Image");
}
```

9.5. Misc.hpp

```
#ifndef Misc_hpp
#define Misc_hpp

#include <stdio.h>
#include <iostream>
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/nonfree/nonfree.hpp"

using namespace cv;
using namespace std;

extern Mat frame_gray;

void closeImg();
//int* findLargeContourAreas(vector<vector<Point>> _contours);
bool compareContourAreas(vector<Point> contour1, vector<Point>
contour2);
void createAWindow();
void destroyTheWindow();

#endif /* Misc_hpp */
```

9.6. SurfDetection.cpp

```
#include "SurfDetection.hpp"

// global variables - SURF
Mat refImage, refDescriptors;
Mat H;
Mat img_matches;
vector<KeyPoint> refKeyPoints, targetKeyPoints;
std::vector<Point2f> ref_obj_corners(4);
Mat outputImage, targetDescriptors;
int i;

extern Mat frame;

// VARIABLES TO FINETUNE THE OBJECT DETECTION
int matchThreshold = 1;
int numberOfMatchingPoints =15;    // This is the size of
'good_matches' vector array, obtained by taking the best matches
from 'matches' vector array.

void getReferenceImage() {

    /*
      // Setting the videoCapture object 'cap' to receive feed
from the webcam i.e. (0).
    VideoCapture cap(0);

    bool condition = true;      // This condition causes the loop
to run continuously till it is false

    // Capturing the Reference image from the webcam
    while (condition == true) {

        Mat boxImage1;
        cap >> boxImage1;
        imshow("Select an object to detect", boxImage1);
        cvtColor(boxImage1, boxImage1, CV_BGR2GRAY);


        if (waitKey(30) >= 0) {

            refImage = boxImage1;
            resize(refImage, refImage, Size(700, 437));
            condition = false;
        } // end if (waitKey(30) >= 0)
```

```
    } // end while(condition == true)
    */

    refImage = imread("/Users/amanrana/Desktop/refImage.jpg");
    if (refImage.empty() == false) {
        cvtColor(refImage, refImage, CV_RGB2GRAY);
    }
     cout << "Reference image has been captured successfully !"
<< endl << endl;

}

tuple<int, int, bool> performSurfDetection(Mat image, bool
referenceOrNot) {

    //bool goodMatchesDetected;
     vector<KeyPoint> keyPoints; // Vector definition to be used
in the SURF algorithm.

    // Defining SurfFeatureDetector
    int hessian = 2000; // setting the hessian to 2500
    SurfFeatureDetector surf(hessian);

      // Detecting the keypoints and storing them in keyPoints
array
    surf.detect(image, keyPoints);

     // drawing the keypoints to outputImage using the image and
keyPoints
         drawKeypoints(image, keyPoints, outputImage,
Scalar(255,255,255), DrawMatchesFlags::DRAW_RICH_KEYPOINTS);

    // Extracting Descriptors from images
    SurfDescriptorExtractor surfDesc;
    Mat descriptors;
    surfDesc.compute(image, keyPoints, descriptors);

    if (referenceOrNot == true) {
        refKeyPoints = keyPoints;
        refDescriptors = descriptors;
              //imshow("Key Points in frame1", outputImage); //
TESTING - DELETE AFTER USE OVER
    } else {
        targetKeyPoints = keyPoints;
        targetDescriptors = descriptors;
```

```
         //imshow("Key Points in frame", outputImage); // TESTING
- DELETE AFTER USE OVER
        return matchFeatures(image);
    }
    return false;
}

void getRefImageAndPerformSURF() {
    // Get the reference image and store it in (Mat) refImage.
    getReferenceImage();
     performSurfDetection(refImage, true); // TRUE will tell the
performSurfDetection function whether this is the reference
image or not
}

tuple<int, int, bool> matchFeatures(Mat _frame) {

    // Matching descriptor vectors with a brute force matcher
    BFMatcher matcher(NORM_L2);
    vector<DMatch> matches, good_matches;
    matcher.match(targetDescriptors, refDescriptors, matches);

    unsigned long int matchSize = matches.size();

     if (matchSize > 0) { // numberOfMatchingPoints changed to 0
-> See if it works.

        for (int i = 0; i < matchSize; i++) {

                        // TESTING -> Delete if the project runs
successfully 95% of the time.
                  //cout << int(matches[i].queryIdx) << " " <<
int(matches[i].trainIdx) << " " << int(matches[i].trainIdx) << "
" << int(matches[i].queryIdx) << endl;

                           if (( abs( int(matches[i].queryIdx) -
int(matches[i].trainIdx) ) <= matchThreshold) ||
( abs(int(matches[i].trainIdx) - int(matches[i].queryIdx)) <=
matchThreshold)) {

                good_matches.push_back(matches[i]);

                                    if (good_matches.size() >=
numberOfMatchingPoints) {
                    break;
                }
```

```
            }
        }

        if (good_matches.size() > 0) {

              tuple<int, int> XAndYCoordinatesOfDetectedObject =
drawLinesAroundDetectedObject(good_matches, _frame);
                                                          return
make_tuple((get<0>(XAndYCoordinatesOfDetectedObject)),
(get<1>(XAndYCoordinatesOfDetectedObject)), true);
                  //drawMatches(refImage, refKeyPoints, _frame,
targetPoints, good_matches, img_matches, Scalar(255,255,255));

        } // end of if (good_matches.size() > 0)
    }
    else {

         cout << "good_matches.size < 0. Exiting..." << endl; //
TESTING
        return false;
    } // end of else
    return false;

}

tuple<int, int> drawLinesAroundDetectedObject(vector<DMatch>
good_matches, Mat _frame) {

    std::vector<Point2f> obj;
    std::vector<Point2f> scene;

    for( int i = 0; i < good_matches.size(); i++ ) {
        //-- Get the keypoints from the good matches

obj.push_back( refKeyPoints[ good_matches[i].queryIdx ].pt );

scene.push_back( targetKeyPoints[ good_matches[i].trainIdx ].pt
);
    }

    int centerX = 0, centerY = 0;

    for (i = 0; i < good_matches.size(); i++) {

         centerX = (int)targetKeyPoints[ good_matches[i].trainIdx
].pt.x + centerX;
```

```
        centerY = (int)targetKeyPoints[ good_matches[i].trainIdx
].pt.y + centerY;

    }

    centerX = int(centerX / i);  // averaging the X values after
summation
    centerY = int(centerY / i);  // averaging the Y values after
summation

    return make_tuple(centerX, centerY);
        //rectangle(_frame, Point2i(centerX -
int(refImage.size().height / 8), centerY -
int(refImage.size().width / 8)), Point2i(centerX +
int(refImage.size().height / 6), centerY +
int(refImage.size().width / 6)), Scalar(0,255,0), 2);


}


tuple<int, int, bool> SurfMain(Mat detectedFrame) {

    if (detectedFrame.cols > 0 && detectedFrame.rows > 0 ) {

        // Converting to grayscale
        cvtColor(detectedFrame, detectedFrame, CV_BGR2GRAY);

        // Resize the frame to (700,437)
        //resize(detectedFrame, detectedFrame, Size(700, 437));

        // Perform SURF Detection on every frame
           return performSurfDetection(detectedFrame, false); //
false - because the frame is not the reference Image

     } else { cout << "detected frame has size zero.. exiting.."
<< endl; return false; }
} // end of int main()
```

9.6. SurfDetection.hpp

```
#ifndef SurfDetection_hpp
#define SurfDetection_hpp

#include <stdio.h>
#include <iostream>
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/nonfree/nonfree.hpp"

// Using namespace cv and std so we dont have to write them
again in the program.
using namespace cv;
using namespace std;

// FUNCTIONS IN SURF FILE
void getReferenceImage();
tuple<int, int, bool> matchFeatures(Mat _frame);//,
vector<KeyPoint> targetPoints, Mat targetDesc);
//tuple<int, int> drawLinesAroundDetectedObject(vector<DMatch>
good_matches, Mat _frame);
void getDefaultReferenceCoordinates();
tuple<int, int> drawLinesAroundDetectedObject(vector<DMatch>
good_matches, Mat _frame);
void getDefaultReferenceCoordinates();
tuple<int, int, bool> performSurfDetection(Mat image,bool
referenceOrNot);
void getRefImageAndPerformSURF();
tuple<int, int, bool> SurfMain(Mat detectedFrame);

#endif /* SurfDetection_hpp */
```