UCCC 2513 Mini Project

Motion detection and tracking for visual surveillance application using some benchmark video databases

Proposal

By: Team 22

Course	Computer Science
Group Member (Student ID)	Chai Jun Hui 1303417 Chan Man Ling 1303796 Lee Wen Dick 1401837
Lecturer	Prof Dr Chen, Zen

CHAPTER 1 INTRODUCTION 3

1.1 Problem Statement 3

1.2 Project Scope 3

1.3 Problem Objective 3

CHAPTER 2 LITERATURE REVIEW 5

2.1 Codebook algorithm 5

2.1.1 Construction of codebook 5

2.1.2 Algorithm for Codebook Update 6

2.1.3 Algorithm for Background Subtraction 6

CHAPTER 3 METHODOLOGY AND SYSTEM DESIGN 7

3.1 Modification and Combination of Algorithms 7

3.2 Algorithm 7

3.2.1 Construction of Codebook elements 7

3.2.2 Algorithm for Codebook Update 8

3.2.3 Overall algorithms for the system 10

3.3 Database, training subset and testing subset 11

3.4 Expected output and system merits 11

3.5 Milestones and Timeline 12

CHAPTER 4 SYSTEM IMPLEMENTATION AND TESTING 13

4.1 Description of hardware and software environments 13

4.1.1 Required system software environment 13

4.2 Testing result presentation 18

4.3 Evaluation and discussions on the results obtained 18

CHAPTER 5 CONCLUSIONS 20

5.1 Problems tackled and achievements in project objectives 20

5.2 Novelties and Contributions 20

5.3 Future Work 20

REFERENCES 22

APPENDIX 23

CHAPTER 1 INTRODUCTION

1.1 Problem Statement

In present age, surveillance system is very common and affordable in the world where surveillance system are to be seen in normal civilians houses and also in shopping centers with the videos recorded being temporarily stored. Detection of any moving object is needed without involving any human to monitor the system constantly.

However, some problem might surface during motion detection. For example, presence of inconsistent lighting and dynamic background movement, weather or illumination changes will reduce the performance and accuracy of the system.

In this project, we are given a video clip which was captured outdoor to record busy moving traffics and walking pedestrians. We are going to detect motion and perform motion detection to them.

1.2 Project Scope

In our system, we are implementing Codebook algorithm for constructing background model. Codebook algorithm adopts a basic model of the background over few seconds or minutes to construct the background model by fitting a time-series model to each pixel or group of pixels. In background subtraction step, the values of the pixel are compared to the range threshold in the Codebook element. In the background subtraction algorithm, background updating is performed periodically by update the matched codeword. In this way, we are able to track the moving objects in a fast manner.

1.3 Problem Objective

There are existing Codebook algorithm proposed to perform the detection and tracking of moving object in different situations and circumstances. The proposed Codebook algorithm uses RGB color scheme to perform the motion detection. Practically, the choice of RGB is not particularly optimal. It is almost always better to use a color space which the axis is aligned with brightness such as YUV color space since empirically, most of the variation in

background tends to be along the brightness axis. Furthermore, we improve the Codebook by simplifying the model without tedious statistical calculation to reduce the computation time needed. Besides, we perform erosion and dilation techniques, cleaning up the foreground segmentation mask and smoothing the edges using polygonal approximation of the segmentation or convex hull of segmentation to obtain the center of the object by calculating the moment of the contour for motion detection purpose.

CHAPTER 2 LITERATURE REVIEW

2.1 Codebook algorithm

Algorithm and technique mentioned in "Real-time foreground-background segmentation using codebook model" are modified.

2.1.1 Construction of codebook

The algorithm is described for color imagery, but it can also be used for gray-scale imagery with minor modifications. Let $\mathcal{X}$ be a training sequence for a single pixel consisting of N RGB-vectors: $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$. Let $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_L\}$ represent the codebook for the pixel consisting of L codewords. Each pixel has a different codebook size based on its sample variation.

Each codeword $\mathbf{c}_i$, $i = 1 \ldots L$, consists of an RGB vector $\mathbf{v}_i = (\bar{R}_i, \bar{G}_i, \bar{B}_i)$ and a 6-tuple $\mathbf{aux}_i = \langle \check{I}_i, \hat{I}_i, f_i, \lambda_i, p_i, q_i \rangle$. The tuple $\mathbf{aux}_i$ contains intensity (brightness) values and temporal variables described below:

$\check{I}$, $\hat{I}$	the *min* and *max* brightness, respectively, of all pixels assigned to this codeword
f	the *frequency* with which the codeword has occurred
λ	the *maximum negative run-length* (MNRL) defined as the longest interval during the training period that the codeword has NOT recurred
p, q	the *first* and *last* access times, respectively, that the codeword has occurred

Figure 1: Construction of initial codebook

In our system, we store a number of Codebook entries in the Codebook. In each Codebook entries, we preserved the last access time, maximum negative run-length and the brightness(which can be obtained from YUV color scheme) from this codebook model. In addition, we introduced high and low side threshold for learning, high and low side box boundary. Outside the Codebook, we track how many codebook entries we have in *numEntries*. We initialize a variable *t* counts the number of access we have accumulated since the start or the last clear operation.

2.1.2 Algorithm for Codebook Update

I. $L \leftarrow 0^1$, $\mathscr{C} \leftarrow \emptyset$ (empty set)
II. **for** $t = 1$ to N **do**
(i) $\mathbf{x}_t = (R, G, B)$, $I \leftarrow \sqrt{R^2 + G^2 + B^2}$
(ii) Find the codeword $\mathbf{c}_m$ in $\mathscr{C} = \{\mathbf{c}_i | 1 \leqslant i \leqslant L\}$ matching to $\mathbf{x}_t$ based on two conditions (a) and (b).
(a) $colordist(\mathbf{x}_t, \mathbf{v}_m) \leqslant \varepsilon_1$
(b) $brightness(I, \langle \check{I}_m, \hat{I}_m \rangle) =$ **true**
(iii) If $\mathscr{C} = \emptyset$ or there is no match, then $L \leftarrow L + 1$. Create a new codeword $\mathbf{c}_L$ by setting
• $\mathbf{v}_L \leftarrow (R, G, B)$
• $\mathbf{aux}_L \leftarrow \langle I, I, 1, t-1, t, t \rangle$.
(iv) Otherwise, update the matched codeword $\mathbf{c}_m$, consisting of
$\mathbf{v}_m = (\bar{R}_m, \bar{G}_m, \bar{B}_m)$ and $\mathbf{aux}_m = \langle \check{I}_m, \hat{I}_m, f_m, \lambda_m, p_m, q_m \rangle$, by setting
• $\mathbf{v}_m \leftarrow \left(\frac{f_m \bar{R}_m + R}{f_m + 1}, \frac{f_m \bar{G}_m + G}{f_m + 1}, \frac{f_m \bar{B}_m + B}{f_m + 1} \right)$
• $\mathbf{aux}_m \leftarrow \langle \min\{I, \check{I}_m\}, \max\{I, \hat{I}_m\}, f_m + 1, \max\{\lambda_m, t - q_m\}, p_m, t \rangle$.
end for
III. For each codeword $\mathbf{c}_i$, $i = 1, \ldots, L$, wrap around λ_i by setting $\lambda_i \leftarrow \max\{\lambda_i, (N - q_i + p_i - 1)\}$.

Figure 2: Algorithm for Codebook construction

We introduced a simplified codebook updating algorithm to reduce the computation time of the system. Therefore, we did not implement the Codebook Update algorithm in this paper.

2.1.3 Algorithm for Background Subtraction

I. $\mathbf{x} = (R, G, B)$, $I \leftarrow \sqrt{R^2 + G^2 + B^2}$
II. For all codewords in $\mathscr{M}$ in Eq. (1), find the codeword $\mathbf{c}_m$ matching to $\mathbf{x}$ based on two conditions:
• $colordist(\mathbf{x}, \mathbf{c}_m) \leqslant \varepsilon_2$
• $brightness(I, \langle \check{I}_m, \hat{I}_m \rangle) =$ **true**
Update the matched codeword as in Step II (iv) in the algorithm of codebook construction.
III. $BGS(\mathbf{x}) = \begin{cases} \textbf{foreground} & \text{if there is no match} \\ \textbf{background} & \text{otherwise.} \end{cases}$

ε_2 is the detection threshold. The pixel is detected as foreground if no acceptable matching codeword exists. Otherwise it is classified as background.

Figure 3: Algorithm for Background subtraction

We introduced our own background subtraction method which we classified the pixel as foreground pixel when the YUV values fall within the box boundary of the Codebook, and classified the pixel as background pixel otherwise.

CHAPTER 3 METHODOLOGY AND SYSTEM DESIGN

3.1 Modification and Combination of Algorithms

Our system are expected to detect motion from moving objects such as car and pedestrian in parallel operating with the video played. Modified Codebook Algorithm is implemented which is fast, and we converted the color scheme from RGB to YUV every time before processing the image. Besides, the system will be able to perform smoothing and noise deletion using connected component analysis with polygonal approximation and detects motion with a box labeled.

3.2 Algorithm

3.2.1 Construction of Codebook

```
Structure name: code_book
      code_element **cb //Codebook entries
      int numEntries //track number of Codebook entries
      int t //counts number of access accumulated since the start or
last clear operation
```

3.2.1 Construction of Codebook elements

```
Structure name: ce
      uchar learnHigh[3] //High side threshold for learning
      uchar learnLow[3] //Low side threshold for learning
      uchar max[3] // High side of box boundary
      uchar min[3] // Low side of box boundary
      int t_last_update // To kill stale entries
      int stale // max negative run
```

The high and low side of box boundary consist of the three raw values of YUV color scheme. The high and low side thread for learning are the threshold that trigger the generation of new Codebook element. In short, a new generation of element will be constructed if the pixel does

not lie between min - learnLow and max + learnHigh in each channels. The time to last update (t_last_update) and max negative run are used to delete the seldom-used codebook entries created during the learning process.

3.2.3 Algorithm for stale cleaning entries

```
Function name: clearStale
INPUT:
c: Codebook for the particular pixel
OUTPUT: Number of entries deleted
int stale = (c.t)/2 //Divisor value can be tuned
For n in each codebook entries
     If max negative run > stale
          Delete the entry
Reset the stale tracking
Refresh time last update for each entries in the codebook
```

The variable *stale* is hardcoded to be half the total running time which can be adjusted freely to favour the system on the video database. Therefore, the codebook entry that is not accessed beyond the half total running time will be deleted. In such way, the noises that are segmented as foreground pixel can be removed and the memory can be saved.

3.2.2 Algorithm for Codebook Update

```
Function name: updateCB
INPUT:
p: Values of YUV pixel
c: Codebook for this pixel
cBound: Learning bounds for codebook which can be tuned in the
system
OUTPUT: Index of the modified Codebook
For n in each channels in the pixel
     High_n = p_n + cBound_n
```

If ($High_n$ > 255) Set $High_n$ = 255 //To prevent overshoot

Low_n = p_n + $cBound_n$

If (Low_n < 0) Set Low_n = 0 //To prevent overshoot

If all channels of this pixel falls within the learning bound

Update last_update time

Adjust the codeword max and min range

Break

Check and update stale entries

If a new codeword is needed

Create new codeword //set learnHigh, learnLow, min and max learning bound

Slowly adjust learning bound //increasing the value of current learnHigh by 1 and decreasing the current value of learnLow by 1.

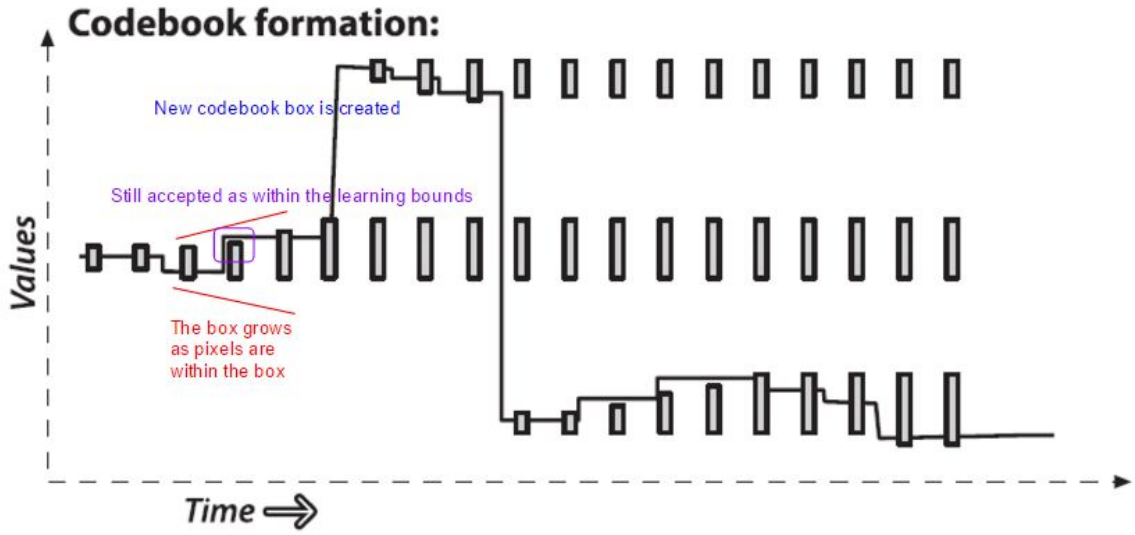


Figure 1: Time series model

Figure 1 is used to further explain the algorithm for the algorithm of Codebook updating.

The function grows or adds a codebook entry when the pixel falls outside the existing codebook boxes. If the pixel falls within *cBounds* of an existing box, the boxes will grows, else a new codebook box will be created. In short, Codebooks are just boxes delimiting intensity values which that box is formed to cover a new value and slowly grows to cover

nearby values. Lastly, it should be noted that the box contains YUV values that must be satisfied for all of them for a pixel to be accepted in the box.

3.2.5 Algorithm for Background Subtraction

```
Function name: bgSubtraction
INPUT:
p: Values of YUV pixel
c: Codebook for this pixel
minLevel: Minimum range //Can be tuned in system
maxLevel: Maximum range //Can be tuned in system
If every channel in the pixel is between (low side of box boundary –
minLevel) && (high side of box boundary + maxLevel)
           Return foreground
Return background
```

3.2.6 Algorithm for finding the contours around the object

```
Function name: drawBox
```

1. Clean up raw mask.
2. Find contours around bigger regions.
3. Construct contour using polygonal approximation.
4. Determine the center point of the contour object with moments.
5. Find the edge point of the rectangle.
6. Draw the box around the object.

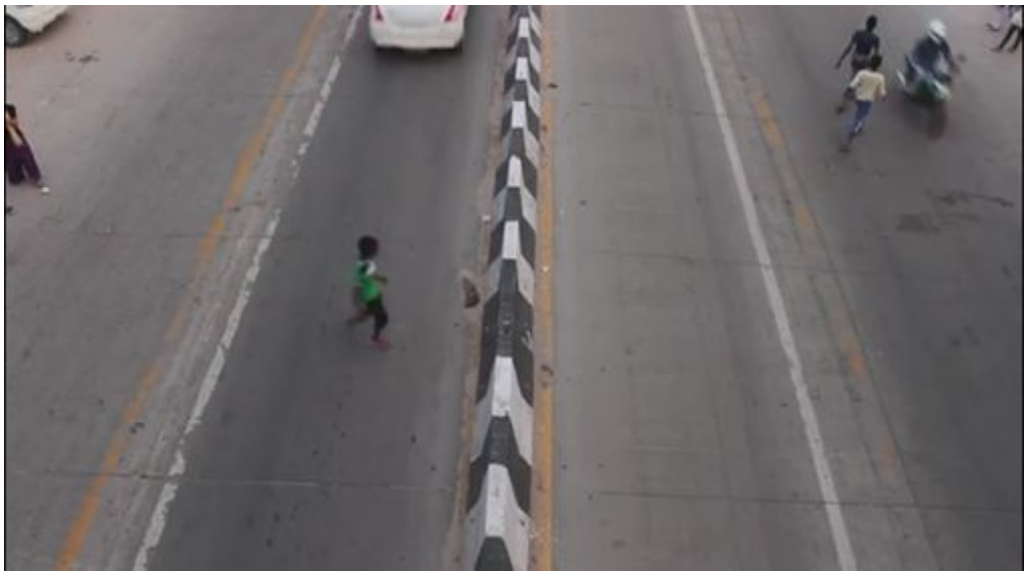
3.2.3 Overall algorithms for the system

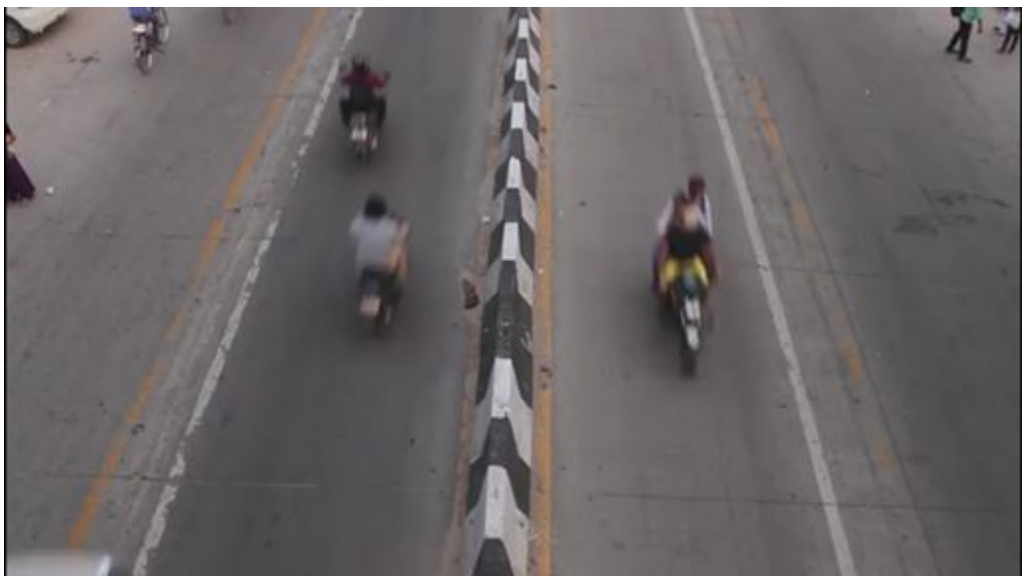
1. Learn a basic background model for *n* frames using updateCB().
2. Clean out stale entries using clearStale().
3. Adjust the threshold to best segment the known foreground.
4. Use the learned model to segment foreground from background using bgSubtraction().
5. Draw boxes around detected motion objects using drawBox().

6. Periodically update codebook using updateCB().

3.3 Database, training subset and testing subset

The video do not have intensive dynamic background therefore a short training sub-set can be acceptable by the system. We tested our system with 100 frames of training and it comes out to be good except there were noises around the result. Noises can be eliminated using erosion.





3.4 Expected output and system merits

System will distinguish between different motion objects by drawing a red box surrounding them.

3.5 Milestones and Timeline

ID	Task Name	Start	Finish	Duration
1	Choose the topic of project	30/5/2016	12/6/2016	2w
2	Research of literature papers	13/6/2016	26/6/2016	2w
3	Write literature reviews	27/6/2016	3/7/2016	1w
4	Review various algorithm and techniques	4/7/2016	10/7/2016	1w
5	**Write the proposal**	**11/7/2016**	**23/7/2016**	**1w 6d**
6	Write problem statement and objectives	11/7/2016	12/7/2016	2d
7	Modify literature reviews	13/7/2016	14/7/2016	2d
8	Modify the existing algorithm	15/7/2016	19/7/2016	5d
9	Construct the flow chart for the system	20/7/2016	21/7/2016	2d
10	Choose the video for the project	22/7/2016	22/7/2016	1d
11	Expect the output for the system	23/7/2016	23/7/2016	1d
12	Submission of the proposal	24/7/2016	24/7/2016	1d
13	**Design and implement the prototype**	**26/7/2016**	**4/8/2016**	**1w 3d**
14	Implement codebook element and codebook structures	26/7/2016	27/7/2016	2d
15	Implement codebook construction	28/7/2016	29/7/2016	2d
16	Implement background subtraction	30/7/2016	31/7/2016	2d
17	Implement connected component analysis	1/8/2016	1/8/2016	1d
18	Implement motion tracking and create boxes for motion detection	2/8/2016	3/8/2016	2d
19	Evaluate the performance	4/8/2016	4/8/2016	1d
20	Modify the proposal	5/8/2016	8/8/2016	4d
21	Submission of report 1	9/8/2016	9/8/2016	1d
22	Testing and debugging the prototype	10/8/2016	15/8/2016	6d
23	Submission of first milestone result	16/8/2016	16/8/2016	1d
24	Implement the final system	17/8/2016	21/8/2016	5d
25	Testing and debugging the final system	22/8/2016	23/8/2016	2d
26	Finalize the system	24/8/2016	24/8/2016	1d
27	Write the report 2	25/8/2016	28/8/2016	4d
28	Prepare presentation slide	29/8/2016	29/8/2016	1d
29	Design the poster	29/8/2016	29/8/2016	1d
30	Presentation of the system and submission of poster	30/8/2016	30/8/2016	1d
31	Submission of the softcopy of the project	30/8/2016	30/8/2016	1d
32	Finalize the report 2	31/8/2016	1/9/2016	2d
33	Submission of the report 2	2/9/2016	2/9/2016	1d

CHAPTER 4 SYSTEM IMPLEMENTATION AND TESTING

4.1 Description of hardware and software environments

4.1.1 Required system hardware environment

The system is running smoothly with a 4GB of RAM tested, with an Intel i-5 processor. Keyboard can be used to adjust the parameter in the program to tune the performance for the system.

4.1.1 Required system software environment

The system is compatible with Windows Operating System, tested with Windows 7, 8, 8.1 and 10. The development is performed in Microsoft Visual Studio 2012 with OpenCV 3.0 installed.

4.2 Experimental design and verification

Most of the research included sophisticated mathematical formula and defination in order to achieve a good quality of motion detection in the trade off of the runtime complexity of their algorithm, lead to a slow system designed that merely support real time surveillance system. In our system, we implemented a simple modified codebook model which can maintain the precision of motion detection and low computational cost of it. Besides, we converted the color space from RGB to YUV to obtain the brightness of the pixel in "Y" which is not considered in the research paper from our literature review.

The system components correctness can be checked by selecting different set of training data and justify if the detected motions are in satisfactory condition.

There are several parameters of the setting can be tuned over the system performance on the particular given video. The number of learning frames are adjusted to minimize the learning time on the video but maximize the correct foreground and background detection on the testing set. Then, we can adjust the background updating period to an optimal interval value so that the system will run as smooth as possible with proper segmentation of foreground and background pixels in the testing set. After performing the background subtraction method ,we perform erosion and dilation on it to clean up the noises produced and amplify the eroded

pixels. In this step, we can tune the number of iteration of erosion and dilation to be done so that the number erosion is enough to clean up the noises but not too much to destroy the object pixels and the dilation of the pixel will not oversize the motion object. Lastly, only the contours that are large enough will be connected to form a polygonal object whose the size can be determined by our system parameter too.

4.2 Testing result presentation

Tested using street pedestrian video **with** shadow effect

Training data set (Number of frames)	Total sum of number of object in motion detected per frame in whole testing set (Start from 150th frames)
25	3081
50	3061
75	3076
100	3064
125	3070

From the table above, the total sum of number of object in motion detected in each training data set are mostly close to each other. However, when there are lesser number of frames taken as training data set, such as 25 frames and 50 frames, we found that there are a lot of false positive object detected. 50 frames and 100 frames have the lowest number of motion detected in overall testing set. However, 100 frames of training data set is preferable as when we justify the motion detection, we realize there is a number of false positive happening after 25 frames of training, which can be shown in Figure 1 below.

Figure 1: False positive object detections

However, due to the constantly background update of our system, the occurrence of false positive object detection only happen in a while after the learning phase, which only make the training of 50 frames inaccurate for only a couple for seconds. In order to achieve the best result, we decided to choose a 100 frames of training to the system so that the inaccuracy of the detection can be minimized. Besides, only 100 frames of training will be needed as we mentioned previously that the system will always perform a constant background update to ensure the most update segmentation between foreground and background.

To verify that our system is constantly update the background, another video is used as examples, which the parked car is treated as background in the learning phase, but soon when the car is moved away, the area of the parked car will become the foreground just for merely few seconds before our system updates it to become background again. (See Figure 2)

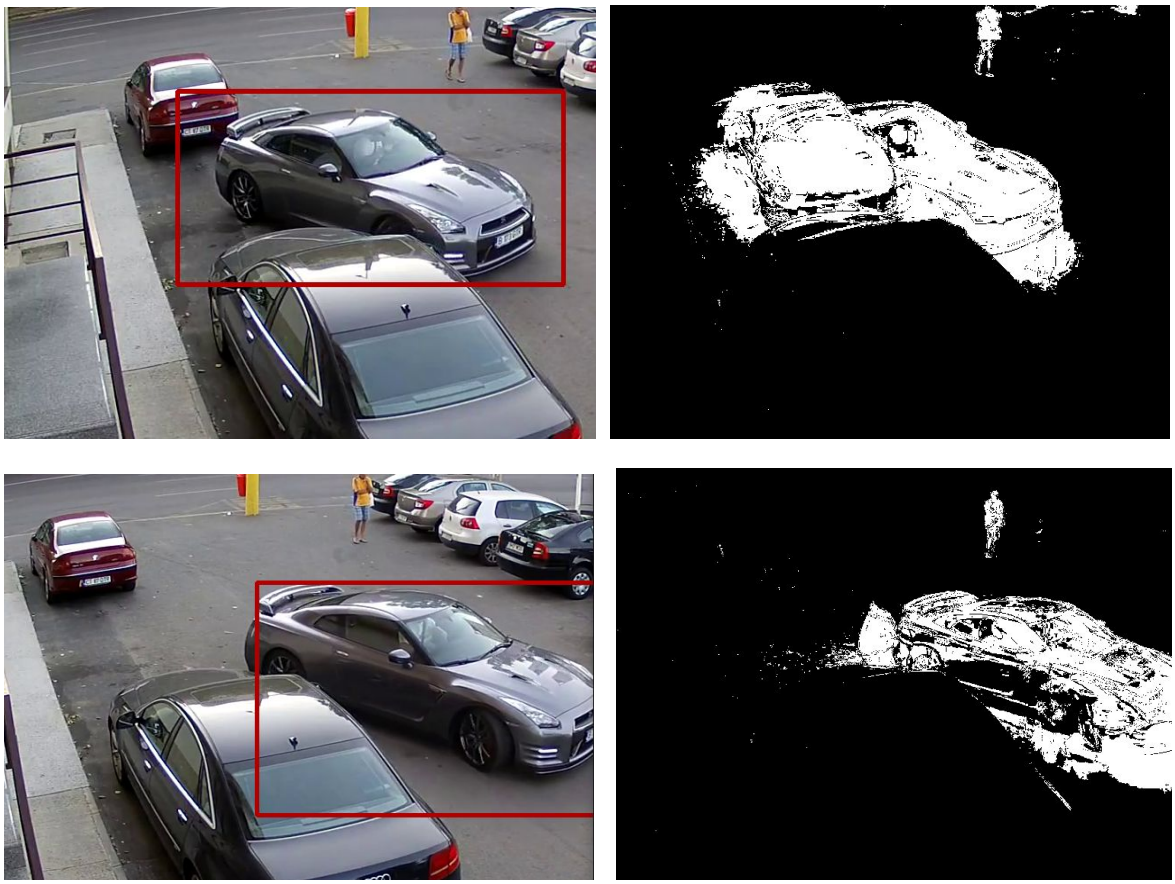


Figure 2: Foreground to background update

Beside outdoor scene, our system can be adapted well with indoor scene. Due to the constant background updating, our system also can adapt to the foreground objects that are

temporarily being the background, which is leaved statically for some times, and then become foreground again when moved. To show it, we used another indoor video to demonstrate our system capability. (See Figure 3)

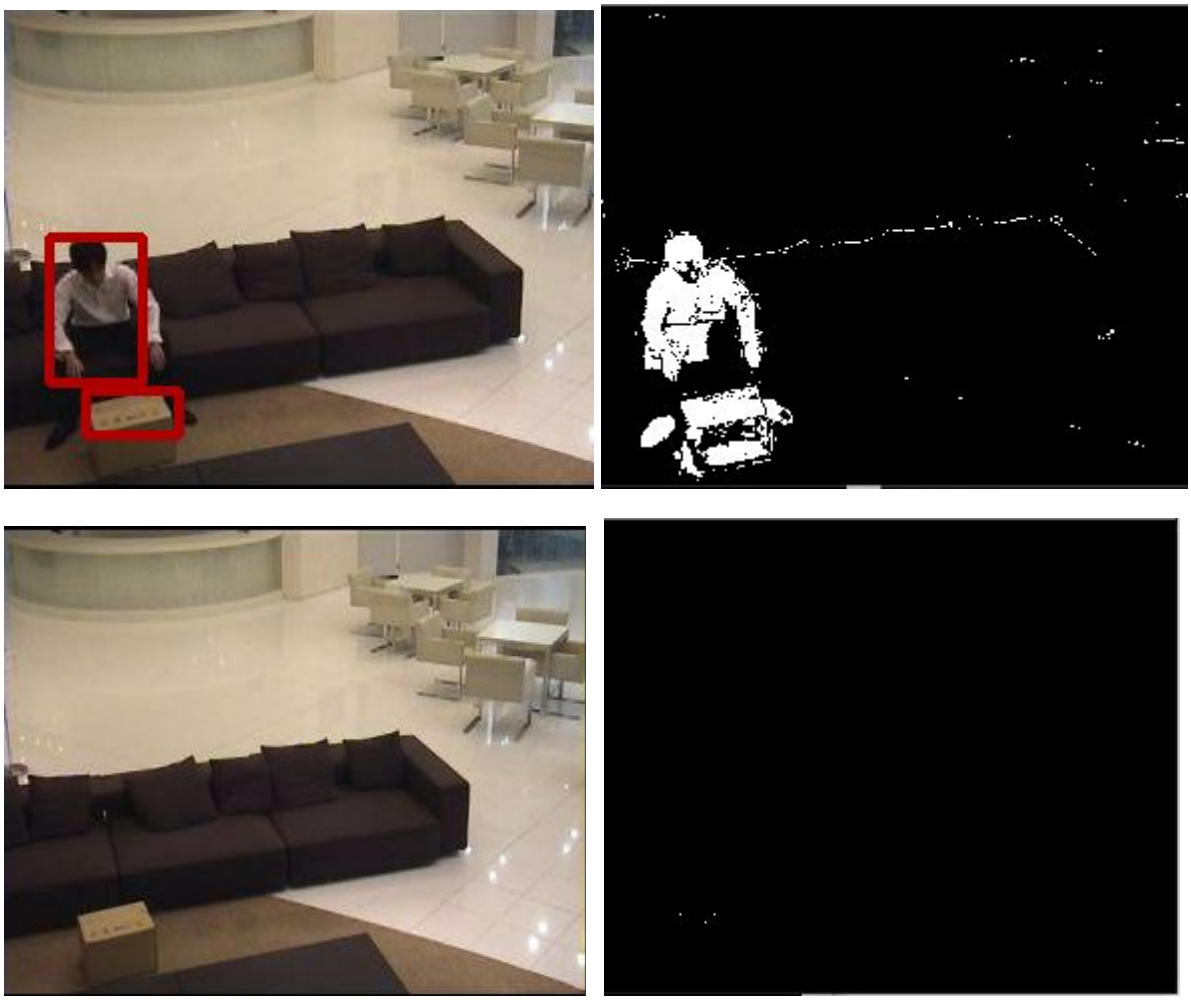


Figure 3: Foreground object classified as background

In video above, there are a lot of foreground to background and background to foreground events as the person inside the video clip placed their belongings in static for a while. Our system can cope with such events without any trouble.

Figure 4: 3 static objects are classified as background as soon as they remain static

In order to clean the raw mask to remove noises and obtain the contour of the objects to form a red box around them, we perform erosion and dilation first and then we will connect them with cvApproxPoly() function which is to perform polygonal approximation. The moments of the objects will lead us to identify those necessary points to form the rectangular red boxes.

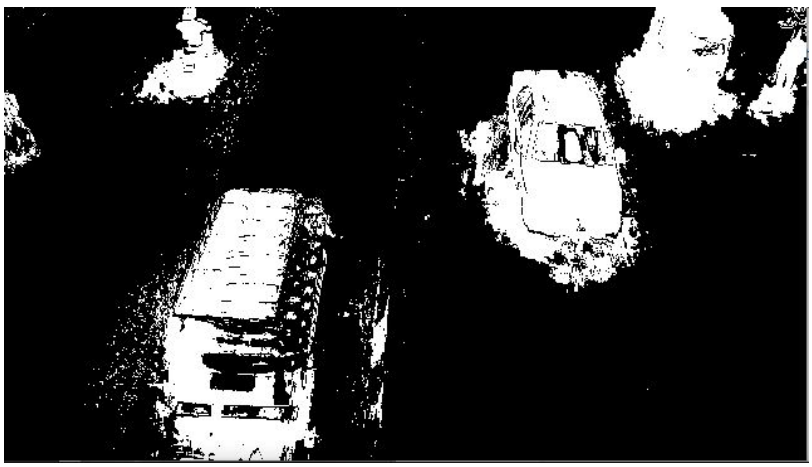


Figure 5: Before clean the raw background subtraction mask

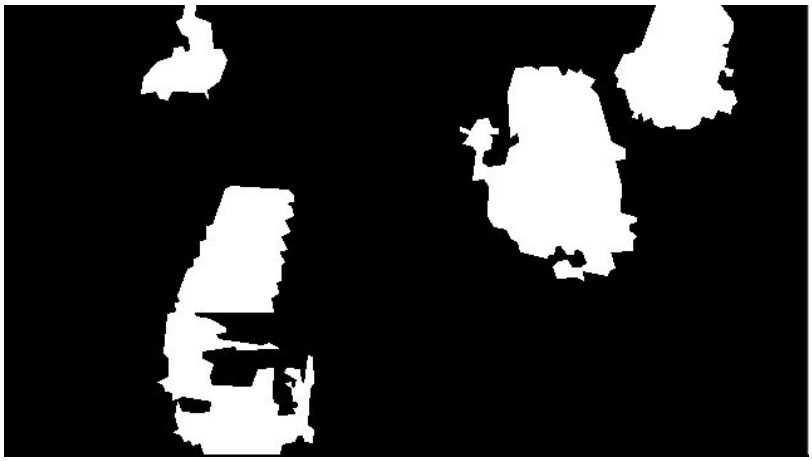


Figure 6: After erosion, dilation and polygonal approximation

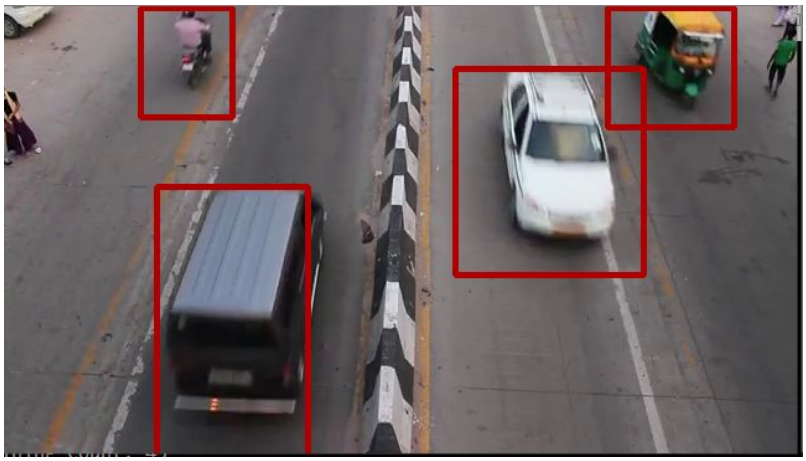
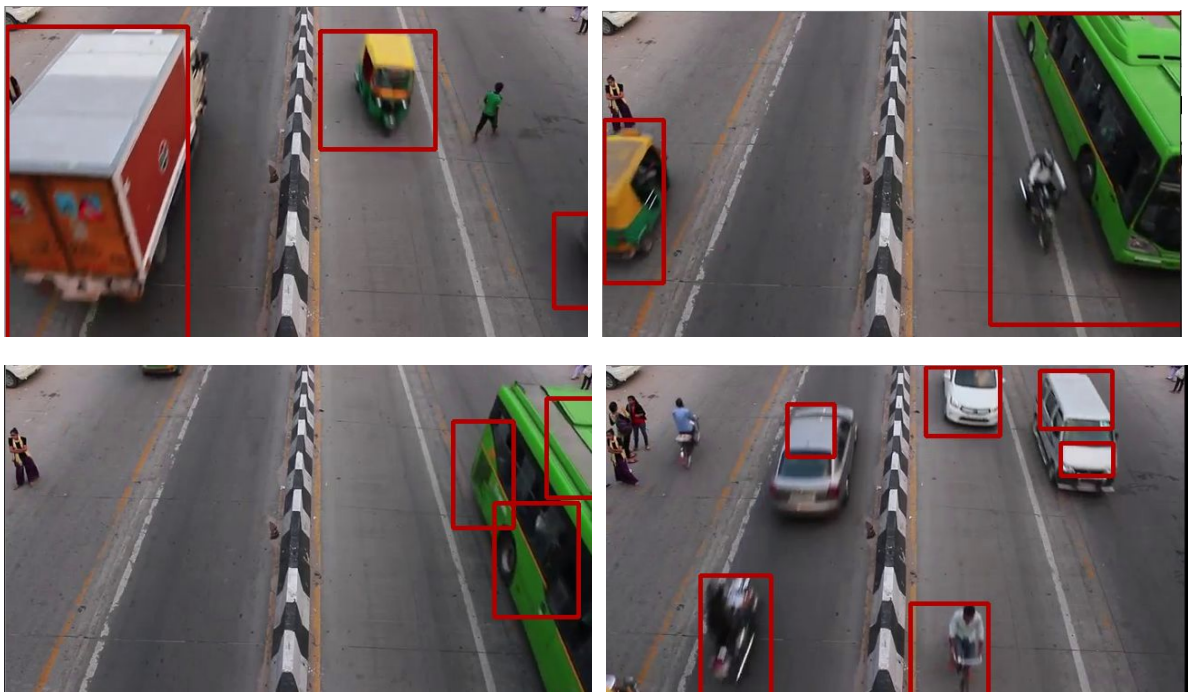


Figure 7: Final result

The rest of the screenshots are illustrated to show our final outcome and performance of our system on the precision of the object motion tracking.



4.3 Evaluation and discussions on the results obtained

We solved the slow computational problem by simplified the codebook model with satisfactory motion detection result. Besides, we included the object motion detection function which are not mentioned in the research paper we reviewed. The system runs smoothly as we expected, and we achieved an acceptable object motion detection on some of

the videos provided. In some videos, our system failed to detect the objects in motion which are too small. Besides, due to the time adjusted for periodically updating background, it takes a few seconds to adapt the leaving object that are originally classified as the background, such as parked car, which is to change the background of the leaving object from foreground to background. Lastly, our system do not cope with shadow, which included it as a foreground object, making the object motion detection box larger than the object, including the shadow of the object as well.

CHAPTER 5 CONCLUSIONS

5.1 Problems tackled and achievements in project objectives

For this project, we've implement modified codebook algorithm to construct the background model. The original codebook algorithm uses RGB color scheme which is not particular optimal without brightness value, so in we've modified the codebook to use YUV color space which the axis is aligned with brightness, because most of the variation in the background tends to be along the brightness axis.

To reduce the computational time needed, we've improved the codebook by simplifying the model without tedious statistical calculation. Besides, we implemented erosion and dilation techniques, cleaning up the foreground segmentation mask and smoothing the edges using polygonal approximation to obtain the center of the object by calculating the moment of the contour for motion detection purpose.

5.2 Novelties and Contributions

Modified Codebook Algorithm is implemented which is faster, and we converted the color scheme from RGB to YUV every time before processing the image. Besides, the system will be able to perform smoothing and noise deletion using dilation and erosion techniques with polygonal approximation, to find the center of point of the detected motion object and label them with a red box.

Moreover, we've added motion objects tracking function whereby, whenever an object in motion is detected, a red box is drawn surrounding to it.

5.3 Future Work

Hopefully, in the future we can add a few more functions like object tracking with ROI. Which is when an object is detected, a box is drawn to indicate that is an object. Other than that, we could also do shadow removal as "A shadow is normally an area that is not or only

partially irradiated or illuminated because of the interception of radiation by an opaque object between the area and the source of radiation. Assuming that the irradiation consists only of white light, the chromaticity in a shadowed region should be the same as when it is directly illuminated." (*Shadow Removal with Morphological Reconstruction,* n.d.).

Lastly, we hope that in the future, we could improve the algorithm to have a more precise detected with the speed maintained.

REFERENCES

Basic C Structures and Operations, n.d. Available from: http://docs.opencv.org/2.4/modules/core/doc/old_basic_structures.html

Real-time foreground–background segmentation using codebook model, n.d. Available from: http://www.umiacs.umd.edu/users/knkim/paper/Kim-RTI2005-FinalPublished.pdf

RGB color model - Wikipedia, n.d. Available from: https://en.wikipedia.org/wiki/RGB_color_model

YUV - Wikipedia, n.d. Available from: https://en.wikipedia.org/wiki/YUV

APPENDIX

```
#include <cv.h>
#include <highgui.h>
#include <cxcore.h>

int CVCONTOUR_APPROX_LEVEL = 2;   // Approx.threshold - the bigger, the simpler the
boundary
int CVCLOSE_ITR = 1;              //Iterations of erosion and dilation
#define WHITE CV_RGB(255,255,255)
#define BLACK CV_RGB(0,0,0)
#define RED CV_RGB(0,255,0)
#define CHL 3
int num = 5; //Arbitrarily
CvRect bbs[20];
CvPoint centers[20];
int Counter = 0, Global = 0;
typedef struct ce {
        uchar   learnHigh[CHL];    // High side threshold for learning
        uchar   learnLow[CHL];     // Low side threshold for learning
        uchar   max[CHL];          // High side of box boundary
        uchar   min[CHL];          // Low side of box boundary
        int     t_last_update;     // To kill stale entries
        int     stale;             // max negative run
}code_element;

typedef struct code_book {
        code_element    **cb;
        int             numEntries;
        int             t;
}codeBook;

int updateCB(uchar *p, codeBook &c, unsigned *cbBounds, int numCHL){
        if (c.numEntries == 0) c.t = 0;
        c.t += 1;
        int n;
        unsigned int high[3], low[3];

        //Learning bounds
        for (n = 0; n<numCHL; n++){
                high[n] = *(p + n) + *(cbBounds + n);
                if (high[n] > 255) high[n] = 255;
                low[n] = *(p + n) - *(cbBounds + n);
                if (low[n] < 0) low[n] = 0;
        }

        int matchChannel;
        int i;
        for (i = 0; i<c.numEntries; i++){
                matchChannel = 0;
                for (n = 0; n<numCHL; n++){//numCHL = 3
                        //Found an entry for this channel
```

```
                        if ((c.cb[i]->learnLow[n] <= *(p + n)) && (*(p + n) <=
c.cb[i]->learnHigh[n]))  matchChannel++;
                }

                if (matchChannel == numCHL){
                        c.cb[i]->t_last_update = c.t;
                        for (n = 0; n<numCHL; n++){
                                if (c.cb[i]->max[n] < *(p + n))   c.cb[i]->max[n] = *(p + n);
                                else if (c.cb[i]->min[n] > *(p + n))   c.cb[i]->min[n] = *(p +
n);
                        }
                        break;
                }
        }

        //Have to add new entry
        if (i == c.numEntries){
                code_element **temp = new code_element*[c.numEntries + 1];
                for (int j = 0; j < c.numEntries; j++) temp[j] = c.cb[j];
                temp[c.numEntries] = new code_element;
                if (c.numEntries) delete[] c.cb;
                c.cb = temp;
                for (n = 0; n<numCHL; n++){
                        c.cb[c.numEntries]->learnHigh[n] = high[n]; //Update local learning
bound to the learnHigh in codebook element
                        c.cb[c.numEntries]->learnLow[n] = low[n]; //Update local learning
bound to the learnLow in codebook element
                        c.cb[c.numEntries]->max[n] = *(p + n); //Exact value of nth chanel to
max
                        c.cb[c.numEntries]->min[n] = *(p + n); //Exact value of ntgh channel
to min
                }
                c.cb[c.numEntries]->t_last_update = c.t;
                c.cb[c.numEntries]->stale = 0;
                c.numEntries += 1;
        }

        for (int k = 0; k < c.numEntries; k++){
                int negRun = c.t - c.cb[k]->t_last_update;
                if (c.cb[k]->stale < negRun) c.cb[k]->stale = negRun;
        }

        //ADJUST LEARNING BOUNDS
        for (n = 0; n < numCHL; n++){
                if (c.cb[i]->learnHigh[n] < high[n]) c.cb[i]->learnHigh[n]++;
                if (c.cb[i]->learnLow[n] > low[n]) c.cb[i]->learnLow[n]--;
        }
        return(i);
}


uchar bgSubtraction(uchar *p, codeBook &c, int numCHL, int *minMod, int *maxMod){
        int matchChl, i;
        for (i = 0; i<c.numEntries; i++){
```

```
                matchChl = 0;
                for (int n = 0; n < numCHL; n++){ //numCHL = 3
                        if ((c.cb[i]->min[n] - minMod[n] <= *(p + n)) && (*(p + n) <=
c.cb[i]->max[n] + maxMod[n]))
                                matchChl++;
                        else break;
                }
                if (matchChl == numCHL) break;
        }
        if (i == c.numEntries) return(255);
        return(0);
}

int clearStale(codeBook &c){
        int staleThresh = c.t >> 2;
        int *keep = new int[c.numEntries];
        int keepCounter = 0;

        for (int i = 0; i<c.numEntries; i++){
                if (c.cb[i]->stale > staleThresh) keep[i] = 0;
                else{
                        keep[i] = 1;
                        keepCounter += 1;
                }
        }

        c.t = 0;
        code_element **temp = new code_element*[keepCounter];
        int k = 0;
        for (int i = 0; i<c.numEntries; i++){
                if (keep[i]){
                        temp[k] = c.cb[i];
                        temp[k]->stale = 0;
                        temp[k]->t_last_update = 0;
                        k++;
                }
        }
        //Clean memory
        delete[] c.cb;
        delete[] keep;
        c.cb = temp;
        int clear = c.numEntries - keepCounter;
        c.numEntries = keepCounter;
        return(clear);
}

void drawBox(IplImage *mask, IplImage *fmask, float perimScale, CvRect *bbs, CvPoint
*centers){
        static CvMemStorage* mem_store = NULL;
        static CvSeq* contours = NULL;
        cvMorphologyEx(mask, mask, NULL, NULL, CV_MOP_OPEN, CVCLOSE_ITR);
        cvMorphologyEx(mask, mask, NULL, NULL, CV_MOP_CLOSE, CVCLOSE_ITR );

        if (mem_store == NULL) mem_store = cvCreateMemStorage(0);
```

```
        else cvClearMemStorage(mem_store);

        CvContourScanner scanner = cvStartFindContours(mask, mem_store, sizeof(CvContour),
CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);

        CvSeq* c;
        int numCont = 0;
        while ((c = cvFindNextContour(scanner)) != NULL){
                double length = cvContourPerimeter(c);
                double q = (mask->height + mask->width) / perimScale;
                if (length < q) cvSubstituteContour(scanner, NULL);
                else{
                        CvSeq* c_new;
                        c_new = cvApproxPoly(c, sizeof(CvContour), mem_store,
CV_POLY_APPROX_DP, CVCONTOUR_APPROX_LEVEL, 1);
                        cvSubstituteContour(scanner, c_new);
                        numCont++;
                }
        }
        contours = cvEndFindContours(&scanner);
        cvZero(mask);
        IplImage *maskTemp;
        int i = 0;
        CvMoments moments;
        double M00, M01, M10;
        maskTemp = cvCloneImage(mask);

        for (i = 0, c = contours; c != NULL; c = c->h_next, i++){
                cvDrawContours(maskTemp, c, WHITE, WHITE, -1, CV_FILLED, 8);
                cvMoments(maskTemp, &moments, 1);
                M00 = cvGetSpatialMoment(&moments, 0, 0);
                M10 = cvGetSpatialMoment(&moments, 1, 0);
                M01 = cvGetSpatialMoment(&moments, 0, 1);
                centers[i].x = (int)(M10 / M00);
                centers[i].y = (int)(M01 / M00);
                bbs[i] = cvBoundingRect(c);
                int distanceX = abs(centers[i].x - bbs[i].x);
                int distanceY = abs(centers[i].y - bbs[i].y);
                cvRectangle(fmask, cvPoint(centers[i].x - distanceX, centers[i].y -
distanceY), cvPoint(centers[i].x + distanceX, centers[i].y + distanceY), RED, 4, 8);
                if(Global >= 150) Counter++;
                cvZero(maskTemp);
                cvDrawContours(mask, c, WHITE, WHITE, -1, CV_FILLED, 8);
        }
        cvReleaseImage(&maskTemp);
}

int main()
{
        CvCapture*  capture = NULL;
        IplImage*   yuvImage = NULL;
        IplImage*   rawImage = NULL;
        IplImage*   ImaskCodeBook = NULL;
        IplImage*   ImaskCodeBookCC = NULL;
```

```
        codeBook*   cB = NULL;
        unsigned    cbBounds[CHL];
        uchar*      pColor = NULL; //YUV pointer
        int         imageSize = 0, nCHL = CHL, minMod[CHL], maxMod[CHL];

        capture = cvCreateFileCapture("C:\\Users\\Mervyn\\Desktop\\Road.mp4");
        cvNamedWindow("Raw");
        cvNamedWindow("CodeBook");
        //cvNamedWindow("Filtered");

        if (!capture){
                printf("Fail to open the file\n");
                return -1;
        }

        rawImage = cvQueryFrame(capture);
        yuvImage = cvCreateImage(cvGetSize(rawImage), 8, 3); //8 bit depth and 3 CHL
        ImaskCodeBook = cvCreateImage(cvGetSize(rawImage), IPL_DEPTH_8U, 1);
        ImaskCodeBookCC = cvCreateImage(cvGetSize(rawImage), IPL_DEPTH_8U, 1);

        cvSet(ImaskCodeBook, cvScalar(255));
        imageSize = cvGetSize(rawImage).height * cvGetSize(rawImage).width;
        cB = new codeBook[imageSize];

        for (int i = 0; i<imageSize; i++) cB[i].numEntries = 0;

        for (int i = 0; i<nCHL; i++){
                cbBounds[i] = 7;
                minMod[i] = 10;
                maxMod[i] = 10;
        }

        for (int i = 0;; i++){
                Global = i;
                int learningFrame = 100;
                cvCvtColor(rawImage, yuvImage, CV_BGR2YCrCb); //Convert to YUV in raw to
yuvImage
                if (i <= learningFrame){
                        printf("Training count: %d\n", i);
                        pColor = (uchar *)(yuvImage->imageData);
                        for (int c = 0; c<imageSize; c++){
                                updateCB(pColor, cB[c], cbBounds, nCHL);
                                pColor += 3;
                        }
                        if (i == learningFrame) for (int c = 0; c<imageSize; c++)
clearStale(cB[c]);
                }
                else{
                        uchar maskPixelCodeBook;
                        pColor = (uchar *)((yuvImage)->imageData); //3 channel yuv image
                        uchar *pMask = (uchar *)((ImaskCodeBook)->imageData); //1 channel
image
                        for (int c = 0; c<imageSize; c++){
```

```
                                maskPixelCodeBook = bgSubtraction(pColor, cB[c], nCHL, minMod,
maxMod);
                                if (i%100 == 0) {
                                        updateCB(pColor, cB[c], cbBounds, nCHL);
                                }
                                *pMask++ = maskPixelCodeBook;
                                pColor += 3;
                        }

                        cvCopy(ImaskCodeBook, ImaskCodeBookCC);
                        drawBox(ImaskCodeBookCC, yuvImage, 4.0, bbs, centers);

                }
                cvCvtColor(yuvImage, yuvImage, CV_YCrCb2BGR);
                if (!(rawImage = cvQueryFrame(capture))) break;
                cvShowImage("Raw", yuvImage);
                cvShowImage("CodeBook", ImaskCodeBook);
                cvShowImage("Filtered", ImaskCodeBookCC);

                if (cvWaitKey(30) == 27) break;
        }
        printf("%d\n", Counter);
        cvReleaseCapture(&capture);
        if (yuvImage) cvReleaseImage(&yuvImage);
        if (ImaskCodeBook) cvReleaseImage(&ImaskCodeBook);
        cvDestroyAllWindows();
        delete[] cB;
        return 0;
}
```