UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY INFORMATION, COMMUNICATION

AND TECHNOLOGY (FICT)

UCCC2513:

Mini Project Assignment 2 Report

COURSEWORK TRIMESTER MAY YEAR 2018

Instructor's name: Prof. Dr. Zen Chen

Team #: 12

Tutorial Day (Mon/Tue/Thu/Fri): Mon

Student Name (Role C/A/T)	Student ID #	Job Division
Ooi Zhi Xuan (Role A)	15ACB02725	1/3
Tan Huei Lie (Role T)	15ACB01982	1/3
Wan Kar Ming (Role C)	15ACB04750	1/3

Abstract

Vision-based motion detection and tracking by using camera to locate moving objects over time has several applications. For instance, traffic control, video communication and compression, surveillance and security and medical imaging. The use of motion detection as part of the security systems has been increasing, to detect and classify as well to track human motion and object with high precision. However, moving objects detection and video tracking are considered to be time consuming processes because of the large data amount in the consecutive video frames. Furthermore, the association between the tracked objects with fast moving targets relative to the frame rate or the objects that have been tracking change their orientation over a period of time will be complicated. Thus, video analysis and processing are required. To compare digital video processing with processing of still images, video processing has a large amount of temporal correlation between the frames. By using video processing techniques, many approaches have been proposed during the last decades. This is because computer vision has let human to manipulate digital image sequences to extract useful data contained in a video stream.

The system we develop aims at detecting and tracking moving objects. The making of a visual-based motion detection and tracking system require reliable, fast and robust algorithms. Background subtraction which detecting the moving objects in the foreground has been included in the algorithm in the image sequences. The first task of moving object detection in a video is the background subtraction and then is the foreground mask sampling. Detailed analysis is carried out on the performance of the visual-based motion detection and tracking system on various test videos.

Table of contents

Chapter 1: Introduction

1.1 Background

Background subtraction is a popular approach to retrieve moving object in a static camera or video, so that the background model obtained will be more consistent. The general idea of background subtracting is by differencing the input frames with the background model constructed and the foreground is constructed if the difference exceed the threshold. So algorithm to construct the background model will affect the result (foreground) obtained. Currently there are different algorithms for background model construction, for example, Mixture of Gaussian, Running average, median filter, and dominant color.

1.2 Problem Statement

However, background subtraction might encounter some problem shown below:

- Dynamic Background
- Lighting (Shadows or gradual/sudden illumination changes)
- Camouflage
- Ghost problem

1.3 Motivation

Our motivation behind this project is to solve the problem in background subtraction as much as possible so that background subtraction can be implement under different condition.

1.4 Project Scope

In this report, a program that used to detect moving objects from an input video will be developed. The program developed are used to detect the moving objects under a static camera and background. The method proposed in this report consist of two phases, which is training and testing phase. In training phase, a background model will be constructed by using mean filter. Whilst, in testing phase, background subtraction technique will be used to extract the moving objects out from the input video.

1.5 Project Objectives

One of the objectives of this project is to create a vision-based motion detection and tracking program using background subtraction technique. The project is aim to reduce problem set for further processing and segment the image into foreground and background. However, there is flaw in using existing techniques to perform video motion detection-based background subtraction process. For examples, the lightning including shadows, sudden illumination changes and camouflage appear of the video will alter the precisions of the detection process by using current existing techniques. Therefore, this project is aim to develop an effective video motion detection scheme which can deal with the flaws in existing system to product a better outcome of vision-based motion detection and tracking program. At the same time, this project also aims to deal with the dynamic background, moving background object and static moving object during the detection process as well.

1.6 Proposed approach

In our project, the motion tracking and detection is done by background subtraction using multiple color and their respective quantity (occurrences) to identify the background and dynamic color. This is because it is believe that background and dynamic color will occurs more frequently, therefore, the higher the quantity of the color, the higher the possibility of the color being background or dynamic color.

In training phase, we store the color occur in each pixel of video frame to a colorList in ColorBox. Each ColorBox represent a pixel, ColorBox box contain a colorList which is use to store the colors occur in that pixel. The background being trained by keep track what color occur in each of the pixel and the quantity of each pixel. The, a background model can be generate by using the most occurred color (dominant color of colorList).

In testing phase, few color from the colorList in each of the ColorBox which total combining weightage are not greater than 75% will be use to compare with the color of each pixel in testing frame. If one of their differences less than threshold value, then it would be set as background object. If none of their differences are less than threshold, then it would be set as foreground object.

1.7 Highlight of what have been achieved

In the previous mini project 1, we detect the moving object from the input video by using mean filter. However, this method give us some issue dealing with dynamic background. Therefore, in this mini project 2, we use dominant color for our background subtraction. Throughout this project, we are able to achieve our objective to dealing with the dynamic background. We are also able to track the moving object by draw out a boundary box on the moving object.

Chapter 2: Literature Review

Detection of moving objects in a video scene plays an important role in computer vision field. Many application such as traffic monitoring system, human detection systems and surveillance systems are based on motion detection. Background subtraction is a simple and effective approach to detect and segment motion in video sequences. Different techniques have been proposed by researchers on detecting moving objects by using background subtraction recently.

2.1 Real-time foreground–background segmentation using codebook model

Kyungnam Kim et. al use a model of background subtraction built from codebook. The authors quantize the background value of each pixel into group of code words. The number of code words is different following the activities of the pixels.

The codebooks idea gives the possibility to learn more about the model in the training period. In the codebook algorithm, each pixel is represented by a codebook, which is a compressed form of background model for a long image sequence. The authors proposed that codebook are expected to capture background motion over a long period of time with limited amount of memory. Thus, codebook are learned from a long training period. The authors improve their basic algorithm by layered modelling/detection multiple background layers and adaptive codebook updating to handle backgrounds change in order to make their method more suitable in a visual surveillance system.

Besides that, the authors claim to cope with unstable information of the dark pixels, however, it still can be problematic when dealing with the low and the high intensity regions.

2.2 Background Subtraction Using Running Gaussian Average and Frame Difference (Tang Z. et al, 2007)

The main idea of this paper proposed by Tang Z. et al. is improve the accuracy of the method of Running Gaussian Average by combining with the frame difference technique.

1. Running Gaussian Average
    a. Background model is constructed by ideally fitting a Gaussian probability density (pdf) on the last *n* pixel’s value.

b. For each pixel, a running average $\boldsymbol{m_t}$ and a standard deviation $\boldsymbol{\sigma_t}$ is updated and store for each color channel.

c. Subtract the running average from a new incoming frame and produces a difference image $\boldsymbol{D_t}$.

d. After normalization step for every color channel, the image is converted into gray scale format and binary mask $\boldsymbol{B^c_t}$ is derived.

2. Frame Difference Method

a. Result obtained based on consecutive two or three frames.

b. Moving objects can be simply extracted by the difference of the current frame and previous frame.

c. An intersection step is performed on the consecutive two binarilized difference images, $\boldsymbol{B^c_t} \cap \boldsymbol{B^c_{t-1}}$, in order to produce $\boldsymbol{B^f_t}$. The purpose of doing this is to eliminate the ghost in the difference images.

By combining the both methods above, the mask produced can compensate each other. So the authors integrate $\boldsymbol{B^f_t}$ with $\boldsymbol{B^c_t}$ and obtain a more reliable mask with the equation below:

$$B_t = \begin{cases} 0 & B^c{}_t + B^f{}_t < 0 \\ B^c{}_t + B^f{}_t & 0 \le B^c{}_t + B^f{}_t \le 255 \\ 255 & B^c{}_t + B^f{}_t > 255 \end{cases}$$

Thus, $\boldsymbol{B_t}$ is the new foreground mask. However, the authors mention that $\boldsymbol{B_t}$ has some small gaps or big holes needed to be fill after filtering the noises. Experiment results of their proposed method is shown below:

(a) $B^c{}_t$ (b) $B^f{}_t$ (c) B_t (d) after filling the holes

2.3 Adaptive background mixture models for real time tracking

The methodology is proposed by Stauffer and Grimson in 1999 which have been worked for the development of an adaptive mixture of Gaussians that can model a changing scene. This model has significant improvement in term of stability and reliability for video surveillance systems compared to other approaches. The authors of the paper have compare their technique to:

- Improvement of modeling each pixel with a single Gaussian
  - Has well indoor performance
  - Not tested for outdoor scenes which has repetitive changes.
- Improvement of modeling each pixel with a Kalman filter
  - Not suitable for backgrounds with repetitive changes.
  - Takes too much time to reestablish the background.
- Averaging images over time to threshold-out foreground pixels
  - Not steady which having many moving objects
  - Has only single threshold for the entire scene.

Real-time tracking technique will be modeling each pixel as a mixture of Gaussians to determine whether or not a pixel is part of the background. There is two main parts for this methodology:

Two main parts:

1. Probabilistic model for separating the background and foreground.

   - Adaptive mixture of multi-modal Gaussians per pixel.

Let *{ X*1 *, ... , Xt }* be a pixel process for *X*. The probability of observing a *X* at frame *t* is as follows, where *K* is the number of Gaussians in the mixture.

$$P(X_t) = \sum_{i=1}^{K} weight_{i,t} * GaussianPDF(X_t, Means_{i,t}, Covariance_{i,t})$$

On the next frame, there will have a new pixel *Xt* where *t* = *t* + 1. Probabilistic model must as follow. By looking at each Gaussian in the mixture of pixel *Xt*., if *Xt* ≤ 2.5 standard deviations from the mean then label matched and stop and any Gaussian not matched is labeled unmatched.

- Method for updating the Gaussian parameters.

From the labeling of each Gaussian, if *Gaussiani,t* is marked as matched. Then there is the need to increase the weight, adjust the mean closer to *Xt*. and decrease the variance.

If *Gaussiani,t* is marked as unmatched. Then the weight has to be decreased.

If all the Gaussians in the mixture for pixel *Xt* are unmatched. *Xt* as a foreground pixel has to be marked. Least probable Gaussian in the mixture has to be find and set *Mean* = *Xt, Va riance* as a high value *weight* as a low value.

Equations for the update:

- $\omega_{i,t} = (1-\alpha) * \omega_{i,t-1} + \alpha * M_{i,t}$
- $\mu_t = (1-\rho) * \mu_{t-1} + \rho * X_t$
- $\sigma_t^2 = (1-\rho) * \sigma_{t-1}^2 + \rho * (X_t - \mu_t)^T * (X_t - \mu_t)$
- $\rho = \alpha * \eta(X_t, \mu_{t-1}, \Sigma_{t-1})$

α is the learning parameter.

- Heuristic for determining the background.

Gaussians from the mixtures which represent the background pixels have to be determined to distribute which pixels have high weight and low variance. Gaussians of each mixture is ordered by weight/standard deviation. Then, the weights in this ordering is summed until the sum is greater than a pre-set threshold T. As last, the means from each Gaussian in the sum to represent the background is used.

2. Technique for tracking objects in the foreground.

- Algorithm to label connected components.

- Method for tracking each connected component.

After experimentation, this methodology is proved to robust against rain, snow, sleet hail and overcast. It can handle motion from swaying branches, rippling water and noise. Furthermore, it also works under day and night cycles. However, it has difficulty in full sun due to the appearance of long shadows of objects and windy and cloudy days. Object overlapping and lighting changes are the issues in this methodology. There is problem during the implementation of this methodology as well. The variance will decrease for stable pixels as time increase. The noise from the camera is marked as foreground pixels if the variance become too small. There is possibility the mixtures may adapt too fast to slow moving objects with same color, which result incorporation of foreground object into background.

Chapter 3: System Design

3.1 System Flow

3.2 Pseudocode of System

In our system, we create 2 class object to store the color of each pixel in a frame. The 2 classes are:

1. Color

Color
- colorIndex: int
- quantity: int
- weightage: double

2. ColorBox

ColorBox
-vector<Color> colorList

Step 1: Get the video clip properties such as CV_CAP_PROP_FRAME_COUNT(total frame),CV_CAP_PROP_FRAME_HEIGHT, CV_CAP_PROP_FRAME_WIDTH and display them.

Step 2: Read in the first frame of the video clip for ROI selection. The ROI width, height and area will be store.

Step 3: Initialize a vector of ColorBox ($V_{ColorBox}$) call background, for each ColorBox is representing a pixel in a frame. ColorBox store what color has been occurred in that pixel, the quantity of that color occurred and the weightage of each color in that pixel (highest weightage is dominant color for that pixel).

Step 4: First 30% of total frame will be used as training frame (ending frame=total frame*0.03), which we call as training sequence.

Step 5: Read in frames in the training sequence one by one from the first frame to the ending frame. For each of the frame read in, crop down the selected ROI area in Step 2 and only the selected ROI area will be needed. Each pixel in the frame represent a ColorBox, color occurred in each pixel will store in a colorList in the ColorBox. If the color have already occur in that colorList before, increase its frequency, else, store that color. All the ColorBox will store in $V_{ColorBox}$.

Step 6: After finished read in from first frame to ending frame, we will have a complete vector of ColorBox. Loop through each of the ColorBox, sort the colorList according to the quantity of the color and compute the weightage (number of percent occurred in the ColorBox) for each of the color in the colorList. Get the first most occur color(dominant color) in each of the ColorBox, generate a background model.

Step 7: Show the background model generated.

Step 8: After finish training, test each frame from the beginning until the end of the video clip with the background model generated. The frames of the video clip in this testing phase we call testing sequence.

for *a*:=1 **to** total_frame **step** 1 **do**

Read in a testing_frame from testing sequence

Crop testing_frame with selected ROI area, store back into testing_frame

Show testing_frame

Generate foreground_mask

for *i*:=0 **to** height of testing_frame **step** 1 **do**

for *j*:=0 **to** width of testing_frame **step** 1 **do**

for *k*:=1 **to** accumulate < 0.75 **do**

if difference**s** of color of testing frame at (*i*, *j*) with color at *k* of the colorList in ColorBox of background at (*i*, *j*) < threshold

this pixel will treat as background object

endif

accumulate += weightage of color at *k* of colorList

endfor

endfor

endfor

Show foreground_mask

Get structure element and apply morphology operations to the current foreground_mask by closing and opening it.

Output the current morphology result in a window.

Create connected component with stats. If the object have an area greater than 80, then connect them together and draw a boundary box. Each frame might have more than one objects. After that output this frame to a window.

endfor

Chapter 4: Implementation and Testing

4.1 System Code Layout

<u>GLOBAL VARIABLES</u>

```
/** 2. Global Variables **/
bool ldown = false; // Left Mouse Button Down (is clicking)
bool lup = false; // Left Mouse Button Up (not clicking)
Point corner1; // Use to store Point A (x1, y1)
Point corner2; // Use to store Point B (x2, y2)
Mat ROI; // Use to store ROI image frame
Rect box; //Rectangle of ROI
bool doneCropping = false;
const int NBIN = 16;
const int BINSIZE = 256 / NBIN;
const int THRES = 50;

string filename = "parking.mp4";
//string filename = "contrysideTraffic.mp4";
//string filename = "MAQ00626.MP4";
//string filename = "Home_Intrusion.mp4";
//string filename = "WavingTrees.avi";
//string filename = "highway1.avi";
//string filename = "fountain01.avi";

VideoCapture video(filename);
```

For Region of Interest (ROI) retrieval, consists of *bool* type variables for flags: *ldown, lup, doneCropping*, *Point* type variable to store the ROI's vertex point: *corner1*, *corner2*, *Rect* type: *box* to store the rectangle of ROI and *Mat* type to store the ROI image frame. While, *int const* type: *NBIN* (Number of Bin to divide), *BINSIZE* (Each Bin's size), *THRES* (Threshold to identifying moving object). *String* type variable *filename* to indicate which video file to be read.

int main(void)

```
/** 5. Main Functions **/
int main(void)
{
    system("Color F0"); // Set white background, black font

                        //-- 1. Check whether video exist or not
                        // If not, then exit the program
    if (!video.isOpened())
    {
        cout << "Can't find video file named - " << filename << "." << endl;
        system("PAUSE");
        return -1;
    }

    int total_frame = (int)video.get(CV_CAP_PROP_FRAME_COUNT);
    int video_height = (int)video.get(CV_CAP_PROP_FRAME_HEIGHT);
    int video_width = (int)video.get(CV_CAP_PROP_FRAME_WIDTH);

    //-- 2. Show the properties of the Video
    cout << "-----// Video Information //-----" << endl;
    cout << "Total Frame: " << total_frame << endl;
    cout << "Video Height: " << video_height << "px" << endl;
    cout << "Video Width: " << video_width << "px" << endl << endl;

    cout << "Press and hold mouse left button to select." << endl;
    cout << "\tor" << endl;
    cout << "Press 'q' to exit" << endl << endl;
```

```
    video.read(ROI);
    imshow("Select ROI", ROI);
    setMouseCallback("Select ROI", mouse_callback);
    while (char(waitKey(1)) != 'q' && !doneCropping) {/*Critical Section*/ }

    //-- 3. Training Phase
    vector<ColorBox> background(box.area(), ColorBox());
    Mat background_model = training(0.3, background); // train only 30% of the frame

                                            //-- 4. Testing Phase
    video.open(filename);
    testing(background);

    cout << "\nDone" << endl;
    system("PAUSE");
    return 0;
}
```

This is the main function, at this function, the video is loaded from the file system. Location of the video is based on the *filename* global variable. The metadata of the video such as total frame in the video, the height of the video and the width of the video is obtained and display to the console for viewing. Then we read a frame from the video to *ROI* and by using mouse, ROI is obtained and storing the information in *corner1, corncer2, box* for further reference. The function initialize *vector<ColorBox> background (box.area(), ColorBox())*, this is the background model which store a list of color in each pixel. Then, proceed to *training (0.3, background)* to train the model, the first parameter of the *training* function is in the data type of *double* (0.3 means train 30% of the total frame), and we pass the background model thru second parameter. After finish training, we proceed to *testing(background)* where foreground mask is produced, the first parameter is used to pass in the background model.

<u>Mat training(double percentage, vector<ColorBox> &background)</u>

```
//-- b. Training Phase
Mat training(double percentage, vector<ColorBox>& background)
{
    if (percentage <= 0.0 || percentage > 1.0)
    {
        percentage = 0.3;
    }

    Mat training_frame;
    int total_frame = (int)video.get(CV_CAP_PROP_FRAME_COUNT);
    int total_train_frame = int(total_frame * percentage);

    cout << "\n-----// Training Phase //-----" << endl;
    cout << "Train " << percentage * 100 << "% of total " << total_frame << " frames." << endl;
    cout << "Learning Frame 1 to Frame " << total_train_frame - 1 << "." << endl;

    for (int i = 1; i <= total_train_frame - 1; i++)
    {
        video.read(training_frame);
        training_frame = training_frame(box);
        imshow("Learning Progress", training_frame);
        waitKey(1);

        put_frame_to_background(training_frame, background);

        cout << "\r" << (i * 100 / total_train_frame) + 1 << "%";
    }
```

```
    Mat background_model = get_background_model(background);
    imshow("Background Trained", background_model);
    waitKey(1);

    for (int i = 0; i < background.size(); i++)
    {
        for (int j = 0; j < background[i].getColorList().size(); j++)
        {
            background[i].getColorList().at(j).setWeightage(total_train_frame);

        }
    }

    cout << endl;
    Sleep(1000); // Sleep for 1 second
    destroyWindow("Select ROI");
    destroyWindow("Learning Progress");
    return background_model;
}
```

If the *percentage* set is less than 0 or more than 1, then reset it to 0.3 to prevent error. Initialize *Mat training_frame* and get the *total_train_frame* by multiplying *percentage* with total frame of the video. Then train all the frame within the *for loop*, then display the background image by *get_background_model* function. At the end, assign weightage to all the color in the colorList in each pixel in the background model using *setWeightage* function in *Color* class.

void put_frame_to_background

```
void put_frame_to_background(Mat& training_frame, vector<ColorBox>& background)
{
    int cols = box.width;
    int rows = box.height;

    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            background[i*cols + j].addColor(training_frame.at<Vec3b>(i, j));
            background[i*cols + j].sortColorList();
        }
    }
}
```

Update the background model using this function, where the *training_frame* act as input. It used *addColor* function in *ColorBox* class to consider the color in the specific pixel into background model (If the color already exist in the list then update its frequency, else add a new color in the list). Then, it used the *sortColorList* function in *ColorBox* class to sort the ColorList by their perspective *quantity*. This action continues until all the pixel in the training frame is considered.

Mat get_background_model(vector<ColorBox> &background)

```
Mat get_background_model(vector<ColorBox>& background)
{
    int cols = box.width;
    int rows = box.height;

    Mat background_model(rows, cols, CV_8UC3);
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            background_model.at<Vec3b>(i, j) = background[i*cols + j].getDominantColor();
        }
    }
    return background_model;
}
```

Return *Mat* type variable which is the background image to be display. This function will construct the background image by using the most occurrences color (highest weightage, highest quantity).

void testing(vector<ColorBox> &background)

```
void testing(vector<ColorBox>& background)
{
    moveWindow("Background Trained", 100, 100);

    Mat testing_frame;
    Mat foreground_mask, display;
    int frame_counter = 1;
    int total_frame = (int)video.get(CV_CAP_PROP_FRAME_COUNT);

    cout << "\n-----// Testing Phase //-----" << endl;
    cout << "Testing " << total_frame << "frames." << endl;

    for (int i = 1; i <= total_frame - 1; i++)
    {
        video.read(testing_frame);
        testing_frame = testing_frame(box);
        imshow("Original Video", testing_frame);
        moveWindow("Original Video", 300, 100);

        foreground_mask = get_foreground_mask(testing_frame, background);
        bitwise_not(foreground_mask, display);
        imshow("Foreground Mask", display);
        moveWindow("Foreground Mask", 100, 300);

        Mat structElement = getStructuringElement(MORPH_ELLIPSE, Size(2, 2));
        morphologyEx(foreground_mask, foreground_mask, MORPH_OPEN, structElement);
        structElement = getStructuringElement(MORPH_ELLIPSE, Size(5, 5));
        morphologyEx(foreground_mask, foreground_mask, MORPH_CLOSE, structElement);
        bitwise_not(foreground_mask, display);
        imshow("Morphology Transformations", display);
        moveWindow("Morphology Transformations", 300, 300);
```

```
        Mat labels, stats, centroids;
        int nLabels = connectedComponentsWithStats(foreground_mask, labels, stats, centroids);
        vector<ForegroundObject> f_objects;
        for (int i = 0; i < nLabels; i++)
        {
            if (stats.at<int>(i, CC_STAT_AREA) < 80)
            {
                continue;
            }
            int x = stats.at<int>(i, CC_STAT_LEFT);
            int y = stats.at<int>(i, CC_STAT_TOP);
            int height = stats.at<int>(i, CC_STAT_HEIGHT);
            int width = stats.at<int>(i, CC_STAT_WIDTH);

            ForegroundObject current_object(Point(centroids.at<double>(i, 0),
                centroids.at<double>(i, 1)), stats.at<int>(i, CC_STAT_AREA), x, y, height, width);
            f_objects.push_back(current_object);
        }

        for (int i = 0; i < f_objects.size(); i++)
        {
            f_objects[i].draw(testing_frame);
        }

        imshow("Testing Result", testing_frame);
        moveWindow("Testing Result", 500, 300);

        cout << "\r" << (i * 100 / total_frame) + 1 << "%";
        waitKey(1);
    }
```

Mat type variable: *testing_frame, foreground_mask, display* is initiated. The foreground mask is computed by using the function *get_foreground_mask* and displayed. The *foreground_mask* is used as input for a series of opening and closing morphological using a kernel of 2x2 and 5x5 to reduce the noise and better visualization purpose. Then, run connected component algorithm (*connectedComponentWithStats)* on the morphological image. With this, the system will be able to identify moving object, by using the *CC_STAT_AREA* filter the small component away and obtains larger component with stats such as *CC_STAT_LEFT*, *CC_STAT_TOP*, *CC_STAT_HEIGHT and CC_STAT WIDTH* to store it in *vector<ForegroundObject> f_objects* for further operation such as drawing boxes and centroid using *draw* function in *ForegroundObject* class for easier tracking of human eyes.

Mat get_foreground_mask(Mat &testing_frame, vector<ColorBox> &background

```
Mat get_foreground_mask(Mat& testing_frame, vector<ColorBox>& background)
{
    int cols = box.width;
    int rows = box.height;

    Mat foreground_mask(rows, cols, CV_8UC1);
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            foreground_mask.at<uchar>(i, j) = 255 *
                !compare_color(testing_frame.at<Vec3b>(i, j), background[i*cols + j].getColorList());
        }
    }
    return foreground_mask;
}
```

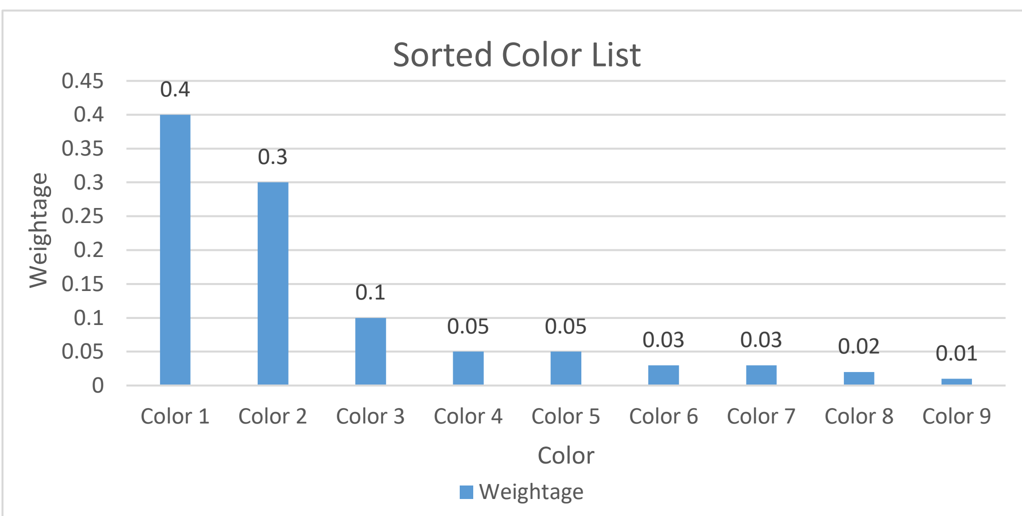
Construct foreground mask (data type: CV_8UC1) by comparing the color in the specific pixel from the *testing_frame* as input with *background* which is the background model for every pixel in the *testing_frame*. The comparison method will be discuss in *compare_color* function. The foreground mask constructed is then returned.

<u>bool compare_color(Vec3b a, vector<Color> &colorList)</u>

```
bool compare_color(Vec3b a, vector<Color>& colorList)
{
    double accumulate = 0;
    for (int i = 0; accumulate < 0.75; i++) {

        if (matchColor(a, colorList[i].getColor()))
        {
            return true;
        }
        accumulate += colorList[i].getWeightage();
    }
    return false;
}
```

This function compare the input color *a* with a colorList in the same position of pixel in the frame with *a*. An *integer* type variable *accumulate* is initiated, it is used to limit the colors used to compare in the color list by updating it using the weightage of each color in the color list that have been compared. The *accumulate* can also be considered as coverage percentage of total color occurs. For example, using 100 frames to training, the *accumulate* (limit) of 90% (0.9) is 90 frames, the first n^{th} color where their total quantity under 90 is considered as the dynamic model (background model). For our case, we converted the quantity to weightage which is the percentage of the total frame used to train. The purpose of this is to differentiate the dynamic and background color from the moving color.



Color 1 + Color 2 + Color 3 + Color 4 = 0.85 → Background and Dynamic Color

Color 5, Color 6…→ Not used in comparison, most probably moving color

bool matchColor(Vec3b a, Vec3b)

```
bool matchColor(Vec3b a, Vec3b b)
{
    for (int i = 0; i < 3; i++)
    {
        if (abs(a.val[i] - b.val[i]) > THRES)
        {
            return false;
        }
    }
    return true;
}
```

Comparison by differencing both color with the use of *abs()* function after subtraction in calculation for each color channel (For bgr, number of channels=3). If the result is more than *THRES* then considered as different color, return false. Else, continue comparing the color in other channel until all the channel is done comparing then return true.

Color class

```
/** 3. Object Classes **/
class Color
{
private:
    int colorIndex;
    int quantity;
    double weightage;

public:
    Color(int colorIndex)
    {
        this->colorIndex = colorIndex;
        this->quantity = 1;
        this->weightage = 0.0;
    }

    int getColorIndex()
    {
        return colorIndex;
    }

    Vec3b getColor()
    {
        int r = colorIndex % (NBIN * NBIN) % (NBIN)* BINSIZE + BINSIZE / 2;
        int g = colorIndex % (NBIN * NBIN) / NBIN * BINSIZE + BINSIZE / 2;
        int b = colorIndex / (NBIN * NBIN) * BINSIZE + BINSIZE / 2;

        return Vec3b(b, g, r);
    }
```

```
    int getQuantity()
    {
        return quantity;
    }

    double getWeightage()
    {
        return weightage;
    }

    void increaseQuantity()
    {
        quantity++;
    }

    void setWeightage(int& total_train_frame)
    {
        weightage = quantity / (double)total_train_frame;
    }

    bool operator< (Color& c)
    {
        return c.quantity < this->quantity;
    }
};
```

The class consists of *integer* variable of *colorIndex* (single integer value to represent bgr value), *quantity* (number of occurrence of the color), *and weightage* (percentage respective to total training frame).

Color::Color(int colorIndex)

Constructor to initialize variable in Color, use the first parameter to initial colorIndex in Color, quantity=1, and weightage=0.0.

int Color::getColorIndex()

return colorIndex.

Vec3b Color::getColor()

Use the colorIndex and calculate back the r, g, b value and return back in data type of Vec3b.

int Color::getQuantity()

return number of color occurrence.

double Color::getWeightage()

return the weightage of the color.

void Color::increaseQuantity()

Increase the quantity when meet the same color.

void Color::setWeightage(int &total_train_frame)

set the weightage of the color based on the quantity with the total training frame.

bool operator < (Color &c)

Overloading the operator for sorting function by using the quantity.

ColorBox Class

```
class ColorBox
{
private:
    vector<Color> colorList;

public:
    ColorBox() {}

    vector<Color>& getColorList()
    {
        return colorList;
    }

    int findIndex(int index)
    {
        for (int i = 0; i < colorList.size(); i++)
        {
            if (colorList[i].getColorIndex() == index)
            {
                return i;
            }
        }
        return -1;
    }
```

```
    void addColor(Vec3b color)
    {
        int index = 0;

        index += (color[0] / BINSIZE) * NBIN * NBIN;
        index += color[1] / BINSIZE * NBIN;
        index += color[2] / BINSIZE;

        int found = findIndex(index);

        if (found == -1)
        {
            Color newColor(index);
            colorList.push_back(newColor);
        }
        else
        {
            colorList[found].increaseQuantity();
        }
    }

    void sortColorList()
    {
        sort(colorList.begin(), colorList.end());
    }

    Vec3b getDominantColor()
    {
        return colorList[0].getColor();
    }
};
```

The class consist of *vector<Color> colorList* used to store colors in a pixel.

int ColorBox::findIndex(int index)

The first parameter pass in the index which represent the color and search in the list (*vector<Color> colorList*) to return the position of the color in the color list.

void ColorBox::addColor(Vec3b color)

Function used to update the list where the first parameter is the input color. It will convert the input color into index form and find whether the color already existed in the list. If the color doesn't exist in the list then a new color is added, else the color is found in the list and the quantity of the color is updated (increased).

void ColorBox::sortColorList()

Function used to sort the list based on the color's quantity, from highest quantity to lowest quantity.

Vec3b ColorBox::getDominantColor()

This function will return the first color in the sorted list which the color is the most occurrence (dominant color in the pixel).

<u>ForegroundObject Class</u>

```
struct ForegroundObject
{
    Point center;
    int area;
    int x;
    int y;
    int height;
    int width;

    ForegroundObject(Point center, int area, int x, int y, int h, int w)
    {
        this->center = center;
        this->area = area;
        this->x = x;
        this->y = y;
        this->height = h;
        this->width = w;
    }

    void draw(Mat& frame)
    {
        rectangle(frame, Point(x, y), Point(x + width, y + height), Scalar(0, 0, 255), 1);
        circle(frame, center, 2, Scalar(255, 0, 0), -1);
    }
};
```

This class consist of a *Point* which store the centroid of the component named *center*. Besides, *integer* type variables such as *area*, *x*, *y*, *height*, *width* of the component is also stored.

ForegroundObject::ForegroundObject(Point center, int area, int x, int y, int h, int w)

This constructor is used when there is component identified as moving object is pass in with the parameters which is *center, area, x, y, h* (height), *w* (width) to be stored as foreground object.

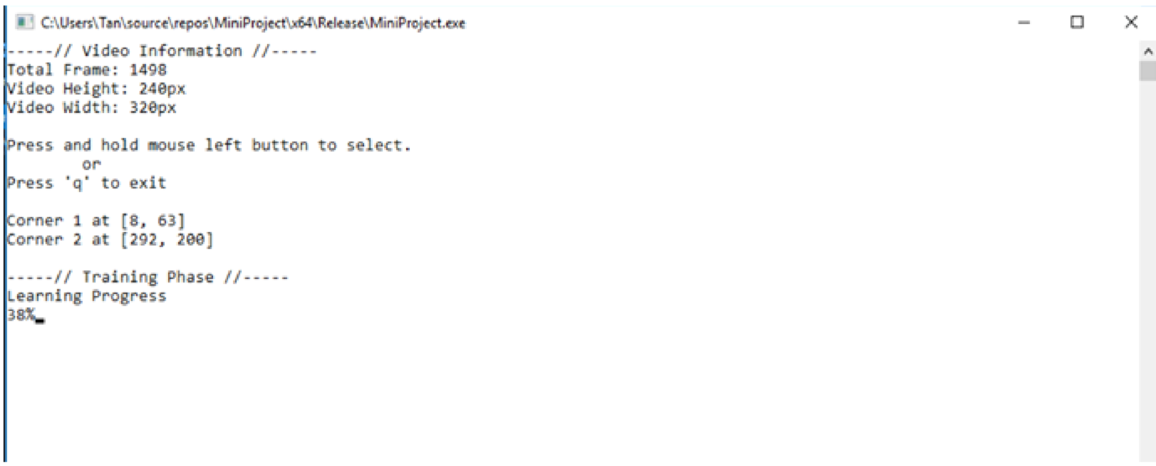
Void ForegroundObject::draw(Mat &frame)

This function is used to draw the rectangle to boundary the moving object and a circle dot to identify the moving object centroid on the frame (The first parameter: &frame).

Chapter 5: Result Analysis and Comparison

5.1 Performance between Mini Project 1 and Mini Project 2

The first dataset that we use for our simple system is “parking.mp4”. Before the program start the training phase, user is requiring to select an ROI for training. Figure below shows the console screen for our simple system.

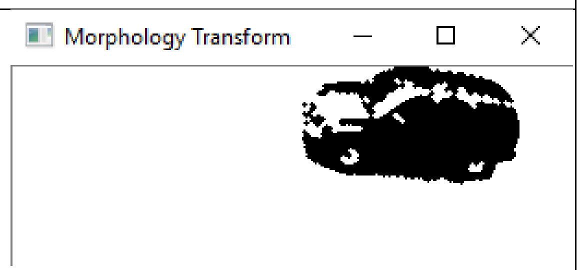
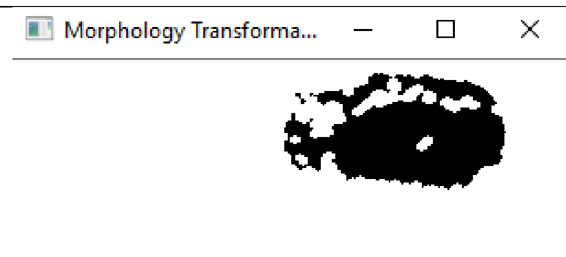


Running console screen in Mini Project 2

In Mini Project 1, we learn the background by updating their mean value for each of the pixel location in training phase. After the training phase, the background model is construct by using the mean color value for each of the pixel location. The figure below shows the learning outcome of our simple system.

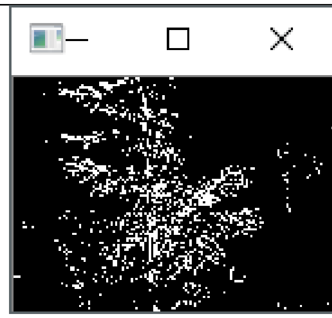
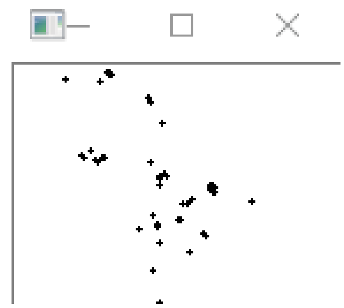
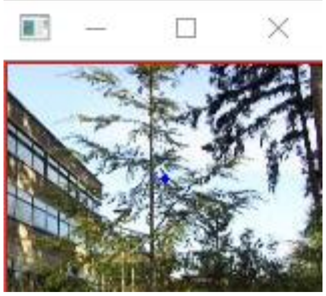
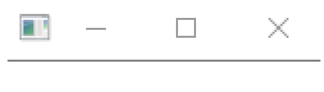
<table>
<tr><td colspan="2">Select ROI</td></tr>
<tr><td colspan="2">
</td></tr>
<tr><td>Learning Progress</td><td>Learned Background</td></tr>
<tr><td>
</td><td>
</td></tr>
</table>

Result of training phase for parking.mp4

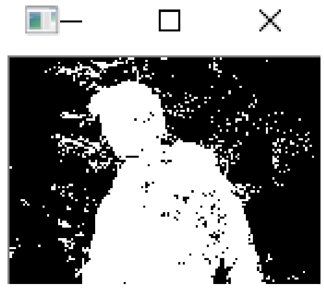
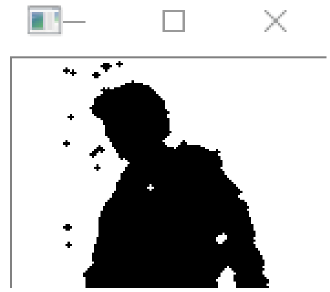
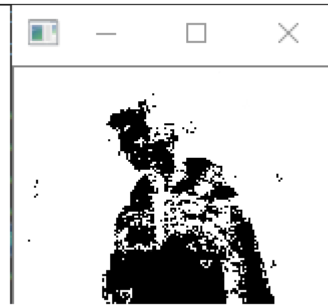
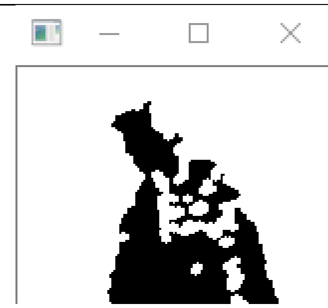
<table>
<tr><td colspan="2">Original Video</td></tr>
<tr><td colspan="2">
</td></tr>
<tr><td colspan="2">Result of testing phase in Mini Project 1</td></tr>
<tr><td>Foreground Mask</td><td>Morphology Transform</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td colspan="2">Result of testing phase in Mini Project 2</td></tr>
<tr><td>Foreground Mask</td><td>Morphology Transform</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>Testing Result</td><td></td></tr>
<tr><td>
</td><td></td></tr>
</table>

Result of testing phase for parking.mp4

The second dataset that we are going to test is “WavingTrees_.avi”. The second dataset we use to test our system is different from the first dataset as this video is having a dynamic background. Figures below show the result of testing phase.

Result of testing phase in Mini Project 1		
Original Video	Foreground Mask	Morphology Transform
		
Result of testing phase in Mini Project 2		
		

Result of testing phase for WavingTrees_.avi (1)

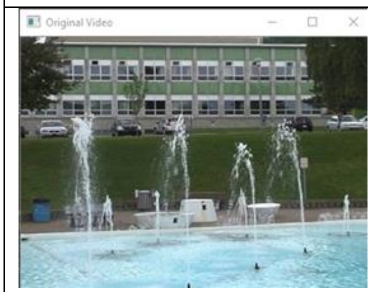
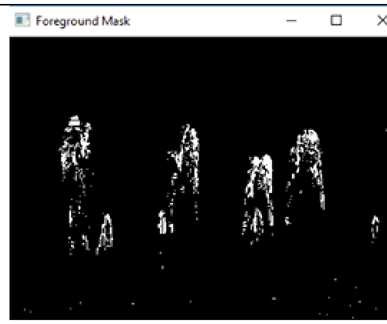
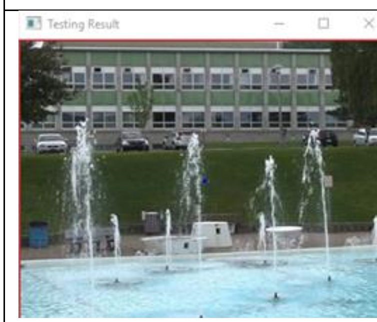
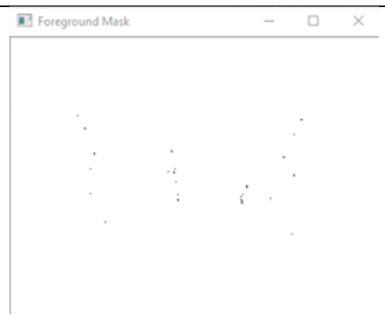
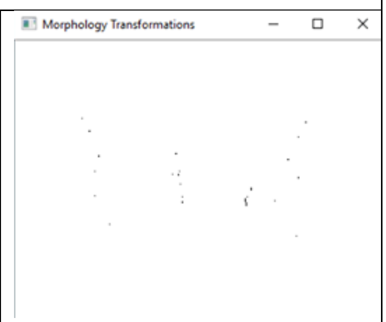
Result of testing phase in Mini Project 1		
Original Video	Foreground Mask	Morphology Transform
		
Result of testing phase in Mini Project 2		
		

Result of testing phase for WavingTrees_.avi (2)

The proposed method we used in Mini Project 1 which is mean filter has failed to process the video with dynamic background. Our simple system treated the waving trees as foreground objects which we are not interested in. Some noise is still leaving behind after a morphology transformation have been carry out to clean out the noise in foreground mask.

Whereas using the proposed method we used in Mini Project 2 which is dominant color are managed to decrease the noise by differentiate the waving tree as a background successfully.

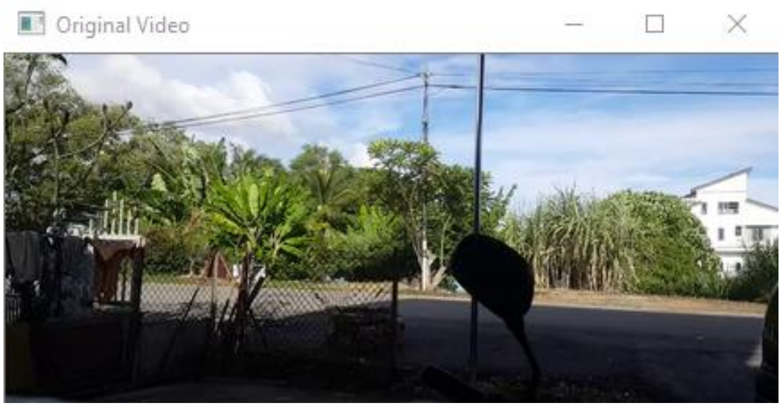
We had tested another video that consist dynamic background which is "fountain01.avi" that having water movement in the entire video.

Result of testing phase in Mini Project 1		
Original Video	Foreground Mask	Morphology Transform
		
Result of testing phase in Mini Project 2		
		

Result of testing phase for fountain01.avi

The result works well as the previous dataset, which the noise from the proposed method in Mini Project 1 has significantly decrease in the proposed method in Mini Project 2. This has shown that dominant color is good in dealing video that has dynamic background.

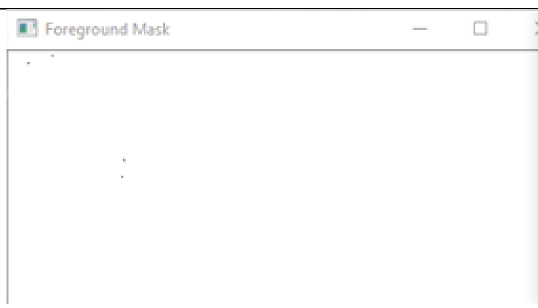
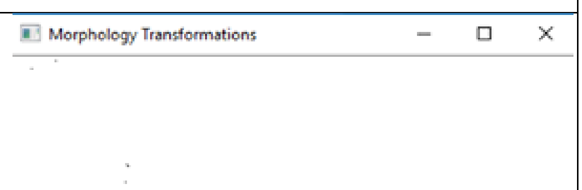
Original Video



Result of testing phase in Mini Project 1

Foreground Mask	Morphology Transform
	

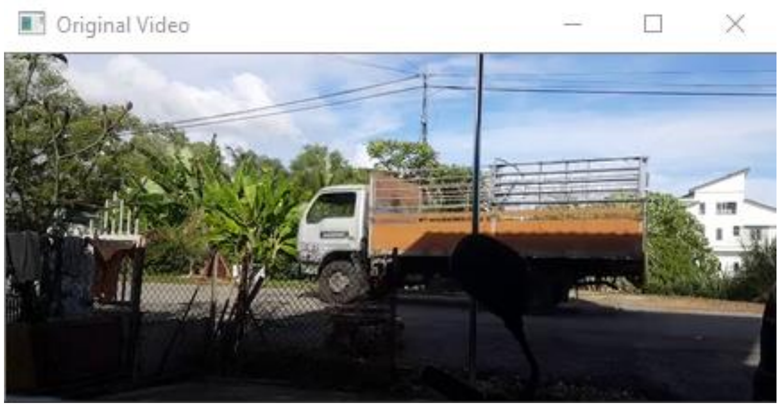
Result of testing phase in Mini Project 2

Foreground Mask	Morphology Transform
	

Testing Result

Result of testing phase for contrysideTraffic.mp4 (1)

Original Video



Result of testing phase in Mini Project 1

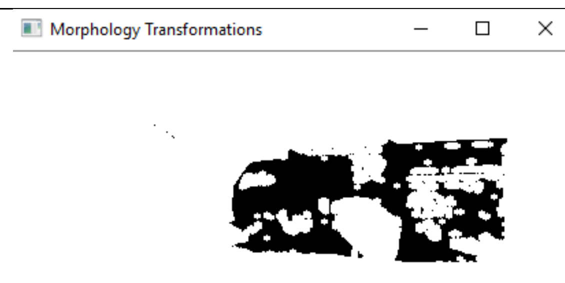
Foreground Mask

Morphology Transform

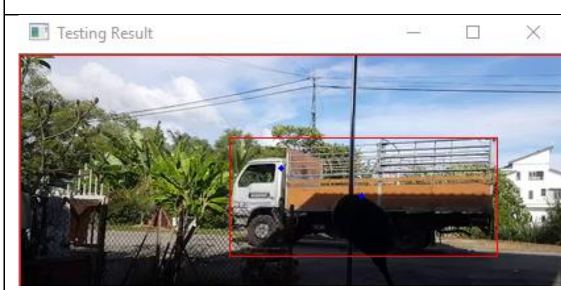
Result of testing phase in Mini Project 2

Foreground Mask

Morphology Transform



Testing Result

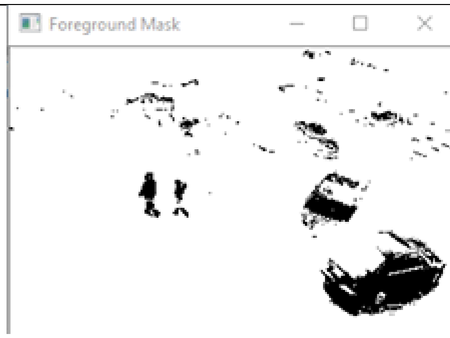


Result of testing phase for contrysideTraffic.mp4 (2)

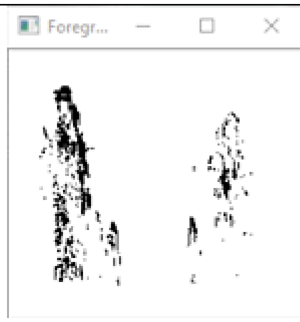
By using the mean method approach in Mini Project 1, the outcome is performed badly when we tested it with fourth dataset – “contrysideTraffic.mp4” due to dynamic background. The waving trees behind are being treated as foreground objects which we are not interested in also. The problem has fixed in our proposed method in Mini Project 2.

However, dominant color method is still poor in dealing with the shadow and reflection as mean filter method. Due to sudden change in color intensity, program treated shadow and reflection as moving objects as well.

5.2 Output Result with Different System Parameters Setting

<table>
<tr><th colspan="2">Result of training frame with value = 0.1</th></tr>
<tr><td>Foreground Mask</td><td>Testing Result</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><th colspan="2">Result of training frame with value = 0.9</th></tr>
<tr><td>Foreground Mask</td><td>Testing Result</td></tr>
<tr><td>
</td><td>
</td></tr>
</table>

Parameter	**Increase**	**Decrease**	**Optimal Value**
training frame	Training phase takes longer time to complete. Might treat a temporary resting foreground as background.	Training phase takes shorter time to complete. Might take a background as foreground. Increase in noise appearance.	0.3

Result of accumulate with value = 10	
Foreground Mask	Testing Result
	
Result of accumulate with value = 90	
Foreground Mask	Testing Result
	

Parameter	**Increase**	**Decrease**	**Optimal Value**
accumulate	Mistakenly treat foreground object as background object.	Background object might mistakenly classify as foreground object especially when there is dynamic background.	0.75

Chapter 6: Conclusion

6.1 Project Review and Discussions

This project has provided an object motion detection program which involved the algorithm that includes background subtraction of the video and foreground detection of objects. The object motion detection system finds its applications where real time surveillance is required such as car parking, traffic monitoring, roadside etc. Motion tracking is performed in the context of higher level applications since it requires the location and shape of the object in every frame to be defined. In the recent years, object motion detection has become significant in the video analysis and processing fields. It concerned with monitoring and tracking an object or multiple objects in a video sequence. Several challenges in objects tracking process can be arise due to the changing of the appearance of the object structures and the camera motion.

In this project, a vision-based motion detection and tracking which done by background subtraction using dominant colour has been proposed. This idea is able to detect moving object, similar as the idea we used in the Mini Project 1 which is mean filter and able to deal with dynamic background which is what mean filter unable to do at the same time. In both ideas, we have used morphology transform to clean out the excess noise in the foreground mask for a better detection of moving objects by segment out individual moving object from a cluster of moving objects. Whereas in Mini Project 2, we have added bounding box on moving object for a better looking.

6.2 Highlight any novelties and contributions the project has achieved

- Our proposed system used background subtraction by using dominant color instead of MOG approaches which works not so well when a video having a dynamic background.
- Our proposed system able to track moving objects from the video only while eliminate most of the noises of a video with dynamic background at the same time.
- Our proposed system computes the connected component in the foreground mask and obtain the bounding boxes and centroid of each moving object to make it better

for visualization. By connecting the centroids of moving object, we would be able to track the movement of the moving objects.

6.3 Future works

- **Find efficient algorithm to reduce computational cost and to decrease the time required for tracking the object for variety of videos.**

  The current algorithm used in this project allowed user to select ROI. However, if the size of ROI is too large, the time to complete training phase and testing phase will be longer.

- **Extend our system by replacing the BGR color space to LAB color space**.

  The reason that we choose to use LAB color space is because LAB color space is a color-opponent space with dimensions L for lightness. By using LAB color space instead of BGR color space, we expect to solve the reflection and shadow problems that occur in our current system.

Bibliography

Kim, K. C. T. H. D. a. D. L., 2005. Real-time foreground–background segmentation using codebook model. *Real-time imaging,* 11(3), pp. 172-185.

Stauffer, C. & Grimson, W. E. L., 1999. Learning patterns of activity using real-time tracking. *Computer Vision and Pattern Recognition,* Volume 2, pp. 252-258.

Tang, Z., Miao, Z. & Wan, Y., 2007. *Background Subtraction Using Running Gaussian Average and Frame Difference,* Springer, Berlin, Heidelberg: Entertainment Computing-ICEC.

Appendices

```
/*------------------- Team 12 -------------------
|      CONTENTS                                                |
|      1. Libraries                                     |
|      2. Global Variables                              |
|             a. You can change video filename here     |
|      3. Object Classes                                |
|             a. Color                                          |
|             b. ColorBox                                       |
|      4. Functions Headers                     |
|      5. Main Function                                 |
|      6. Functions                                     |
|             a. ROI
       |
|             b. Training Phase                         |
|             c. Testing Phase                          |
-------------------------------------------------*/

/** 1. Libraries **/
#include <opencv2/opencv.hpp>
#include <iostream>
#include <algorithm>
#include <vector>
#include <conio.h>
#include <Windows.h>

using namespace cv;
using namespace std;

/** 2. Global Variables **/
bool ldown = false; // Left Mouse Button Down (is clicking)
bool lup = false; // Left Mouse Button Up (not clicking)
Point corner1; // Use to store Point A (x1, y1)
Point corner2; // Use to store Point B (x2, y2)
Mat ROI; // Use to store ROI image frame
Rect box; //Rectangle of ROI
bool doneCropping = false;
const int NBIN = 16;
const int BINSIZE = 256 / NBIN;
const int THRES = 50;

string filename = "parking.mp4";
//string filename = "contrysideTraffic.mp4";
//string filename = "MAQ00626.MP4";
//string filename = "Home_Intrusion.mp4";
//string filename = "WavingTrees.avi";
//string filename = "highway1.avi";
//string filename = "fountain01.avi";

VideoCapture video(filename);

/** 3. Object Classes **/
class Color
{
private:
	int colorIndex;
```

```
	int quantity;
	double weightage;

public:
	Color(int colorIndex)
	{
		this->colorIndex = colorIndex;
		this->quantity = 1;
		this->weightage = 0.0;
	}

	int getColorIndex()
	{
		return colorIndex;
	}

	Vec3b getColor()
	{
		int r = colorIndex % (NBIN * NBIN) % (NBIN)* BINSIZE + BINSIZE / 2;
		int g = colorIndex % (NBIN * NBIN) / NBIN * BINSIZE + BINSIZE / 2;
		int b = colorIndex / (NBIN * NBIN) * BINSIZE + BINSIZE / 2;

		return Vec3b(b, g, r);
	}

	int getQuantity()
	{
		return quantity;
	}

	double getWeightage()
	{
		return weightage;
	}

	void increaseQuantity()
	{
		quantity++;
	}

	void setWeightage(int& total_train_frame)
	{
		weightage = quantity / (double)total_train_frame;
	}

	bool operator< (Color& c)
	{
		return c.quantity < this->quantity;
	}
};

class ColorBox
{
private:
	vector<Color> colorList;

public:
```

```
	ColorBox() {}

	vector<Color>& getColorList()
	{
		return colorList;
	}

	int findIndex(int index)
	{
		for (int i = 0; i < colorList.size(); i++)
		{
			if (colorList[i].getColorIndex() == index)
			{
				return i;
			}
		}
		return -1;
	}

	void addColor(Vec3b color)
	{
		int index = 0;

		index += (color[0] / BINSIZE) * NBIN * NBIN;
		index += color[1] / BINSIZE * NBIN;
		index += color[2] / BINSIZE;

		int found = findIndex(index);

		if (found == -1)
		{
			Color newColor(index);
			colorList.push_back(newColor);
		}
		else
		{
			colorList[found].increaseQuantity();
		}
	}

	void sortColorList()
	{
		sort(colorList.begin(), colorList.end());
	}

	Vec3b getDominantColor()
	{
		return colorList[0].getColor();
	}
};

struct ForegroundObject
{
	Point center;
	int area;
	int x;
	int y;
```

```
	int height;
	int width;

	ForegroundObject(Point center, int area, int x, int y, int h, int w)
	{
		this->center = center;
		this->area = area;
		this->x = x;
		this->y = y;
		this->height = h;
		this->width = w;
	}

	void draw(Mat& frame)
	{
		rectangle(frame, Point(x, y), Point(x + width, y + height),
Scalar(0, 0, 255), 1);
		circle(frame, center, 2, Scalar(255, 0, 0), -1);
	}
};

/** 4. Function Headers **/
//-- a. ROI
static void mouse_callback(int event, int x, int y, int, void *);
//-- b. Training Phase
Mat training(double percentage, vector<ColorBox>& background);
void put_frame_to_background(Mat& training_frame, vector<ColorBox>& background);
Mat get_background_model(vector<ColorBox>& background);
//-- c. Testing Phase
void testing(vector<ColorBox>& background);
Mat get_foreground_mask(Mat& training_frame, vector<ColorBox>& background);
bool compare_color(Vec3b a, vector<Color>& colorList);
bool matchColor(Vec3b a, Vec3b b);

/** 5. Main Functions **/
int main(void)
{
	system("Color F0"); // Set white background, black font

	//-- 1. Check whether video exist or not
	// If not, then exit the program
	if (!video.isOpened())
	{
		cout << "Can't find video file named - " << filename << "." << endl;
		system("PAUSE");
		return -1;
	}

	int total_frame = (int)video.get(CV_CAP_PROP_FRAME_COUNT);
	int video_height = (int)video.get(CV_CAP_PROP_FRAME_HEIGHT);
	int video_width = (int)video.get(CV_CAP_PROP_FRAME_WIDTH);

	//-- 2. Show the properties of the Video
	cout << "-----// Video Information //-----" << endl;
	cout << "Total Frame: " << total_frame << endl;
	cout << "Video Height: " << video_height << "px" << endl;
	cout << "Video Width: " << video_width << "px" << endl << endl;
```

```
	cout << "Press and hold mouse left button to select." << endl;
	cout << "\tor" << endl;
	cout << "Press 'q' to exit" << endl << endl;

	video.read(ROI);
	imshow("Select ROI", ROI);
	setMouseCallback("Select ROI", mouse_callback);
	while (char(waitKey(1)) != 'q' && !doneCropping) {/*Critical Section*/ }

	//-- 3. Training Phase
	vector<ColorBox> background(box.area(), ColorBox());
	Mat background_model = training(0.3, background); // train only 30% of the
frame

	//-- 4. Testing Phase
	video.open(filename);
	testing(background);

	cout << "\nDone" << endl;
	system("PAUSE");
	return 0;
}

/** 6. Functions **/
//-- a. ROI
void mouse_callback(int event, int x, int y, int, void *)
{
	if (event == EVENT_LBUTTONDOWN)
	{
		ldown = true;
		corner1.x = x;
		corner1.y = y;
		cout << "Corner 1 at " << corner1 << endl;
	}

	if (event == EVENT_LBUTTONUP)
	{
		if (abs(x - corner1.x) > 10 && abs(y - corner1.y) > 10)
		{
			lup = true;

			corner2.x = x;
			corner2.y = y;

			if (y > ROI.size().height) corner2.y = ROI.size().height;
			if (x > ROI.size().width) corner2.x = ROI.size().width;

			if (y < 0) corner2.y = 0;
			if (x < 0) corner2.x = 0;

			cout << "Corner 2 at " << corner2 << endl;
		}
		else
		{
			cout << "Please select a bigger region" << endl;
			ldown = false;
```

```
		}
	}

	if (ldown == true && lup == false)
	{
		Point pt;
		pt.x = x;
		pt.y = y;

		Mat locale_img = ROI.clone();

		if (y > ROI.size().height) corner2.y = ROI.size().height;
		if (x > ROI.size().width) corner2.x = ROI.size().width;

		if (y < 0) pt.y = 0;
		if (x < 0) pt.x = 0;

		rectangle(locale_img, corner1, pt, Scalar(0, 0, 255), 2);

		imshow("Select ROI", locale_img);
	}

	if (ldown == true && lup == true)
	{
		ldown = false;
		lup = false;

		box.width = abs(corner1.x - corner2.x);
		box.height = abs(corner1.y - corner2.y);

		box.x = min(corner1.x, corner2.x);
		box.y = min(corner1.y, corner2.y);

		doneCropping = true;
	}
}

//-- b. Training Phase
Mat training(double percentage, vector<ColorBox>& background)
{
	if (percentage <= 0.0 || percentage > 1.0)
	{
		percentage = 0.3;
	}

	Mat training_frame;
	int total_frame = (int)video.get(CV_CAP_PROP_FRAME_COUNT);
	int total_train_frame = int(total_frame * percentage);

	cout << "\n-----// Training Phase //-----" << endl;
	cout << "Train " << percentage * 100 << "% of total " << total_frame << " 
frames." << endl;
	cout << "Learning Frame 1 to Frame " << total_train_frame - 1 << "." << 
endl;

	for (int i = 1; i <= total_train_frame - 1; i++)
	{
```

```
		video.read(training_frame);
		training_frame = training_frame(box);
		imshow("Learning Progress", training_frame);
		waitKey(1);

		put_frame_to_background(training_frame, background);

		cout << "\r" << (i * 100 / total_train_frame) + 1 << "%";
	}
	Mat background_model = get_background_model(background);
	imshow("Background Trained", background_model);
	waitKey(1);

	for (int i = 0; i < background.size(); i++)
	{
		for (int j = 0; j < background[i].getColorList().size(); j++)
		{
	background[i].getColorList().at(j).setWeightage(total_train_frame);

		}
	}

	cout << endl;
	Sleep(1000); // Sleep for 1 second
	destroyWindow("Select ROI");
	destroyWindow("Learning Progress");
	return background_model;
}

void put_frame_to_background(Mat& training_frame, vector<ColorBox>& background)
{
	int cols = box.width;
	int rows = box.height;

	for (int i = 0; i < rows; i++)
	{
		for (int j = 0; j < cols; j++)
		{
			background[i*cols + j].addColor(training_frame.at<Vec3b>(i,
j));
			background[i*cols + j].sortColorList();
		}
	}
}

Mat get_background_model(vector<ColorBox>& background)
{
	int cols = box.width;
	int rows = box.height;

	Mat background_model(rows, cols, CV_8UC3);
	for (int i = 0; i < rows; i++)
	{
		for (int j = 0; j < cols; j++)
		{
```

```
                    background_model.at<Vec3b>(i, j) = background[i*cols +
j].getDominantColor();
              }
       }
       return background_model;
}

//-- c. Testing Phase
void testing(vector<ColorBox>& background)
{
       moveWindow("Background Trained", 100, 100);

       Mat testing_frame;
       Mat foreground_mask, display;
       int frame_counter = 1;
       int total_frame = (int)video.get(CV_CAP_PROP_FRAME_COUNT);

       cout << "\n-----// Testing Phase //-----" << endl;
       cout << "Testing " << total_frame << "frames." << endl;

       for (int i = 1; i <= total_frame - 1; i++)
       {
              video.read(testing_frame);
              testing_frame = testing_frame(box);
              imshow("Original Video", testing_frame);
              moveWindow("Original Video", 300, 100);

              foreground_mask = get_foreground_mask(testing_frame, background);
              bitwise_not(foreground_mask, display);
              imshow("Foreground Mask", display);
              moveWindow("Foreground Mask", 100, 300);

              Mat structElement = getStructuringElement(MORPH_ELLIPSE, Size(2,
2));
              morphologyEx(foreground_mask, foreground_mask, MORPH_OPEN,
structElement);
              structElement = getStructuringElement(MORPH_ELLIPSE, Size(5, 5));
              morphologyEx(foreground_mask, foreground_mask, MORPH_CLOSE,
structElement);
              //morphologyEx(foreground_mask, foreground_mask, MORPH_CLOSE,
structElement);
              bitwise_not(foreground_mask, display);
              imshow("Morphology Transformations", display);
              moveWindow("Morphology Transformations", 300, 300);

              Mat labels, stats, centroids;
              int nLabels = connectedComponentsWithStats(foreground_mask, labels,
stats, centroids);
              vector<ForegroundObject> f_objects;
              for (int i = 0; i < nLabels; i++)
              {
                     if (stats.at<int>(i, CC_STAT_AREA) < 80)
                     {
                            continue;
                     }
                     int x = stats.at<int>(i, CC_STAT_LEFT);
                     int y = stats.at<int>(i, CC_STAT_TOP);
```

```
                        int height = stats.at<int>(i, CC_STAT_HEIGHT);
                        int width = stats.at<int>(i, CC_STAT_WIDTH);

                        ForegroundObject current_object(Point(centroids.at<double>(i,
0), centroids.at<double>(i, 1)), stats.at<int>(i, CC_STAT_AREA), x, y, height,
width);
                        f_objects.push_back(current_object);
                }

                for (int i = 0; i < f_objects.size(); i++)
                {
                        f_objects[i].draw(testing_frame);
                }

                imshow("Testing Result", testing_frame);
                moveWindow("Testing Result", 500, 300);

                cout << "\r" << (i * 100 / total_frame) + 1 << "%";
                waitKey(1);
        }

        cout << endl;
}

Mat get_foreground_mask(Mat& testing_frame, vector<ColorBox>& background)
{
        int cols = box.width;
        int rows = box.height;

        Mat foreground_mask(rows, cols, CV_8UC1);
        for (int i = 0; i < rows; i++)
        {
                for (int j = 0; j < cols; j++)
                {
                        foreground_mask.at<uchar>(i, j) = 255 *
                                !compare_color(testing_frame.at<Vec3b>(i, j),
background[i*cols + j].getColorList());
                }
        }
        return foreground_mask;
}

bool compare_color(Vec3b a, vector<Color>& colorList)
{
        double accumulate = 0;
        for (int i = 0; accumulate < 0.75; i++) {

                if (matchColor(a, colorList[i].getColor()))
                {
                        return true;
                }
                accumulate += colorList[i].getWeightage();
        }
        return false;
}

bool matchColor(Vec3b a, Vec3b b)
```

```
{
	for (int i = 0; i < 3; i++)
	{
		if (abs(a.val[i] - b.val[i]) > THRES)
		{
			return false;
		}
	}
	return true;
}
```