UNIVERSITI TUNKU ABDUL RAHMAN

Faculty of Information and Communication Technology (FICT)

UCCC2513 Mini Projects

Title: Mini project report 2
Team ID: 13

No.	Name	Student ID	Programme
1	Lim Yun Kai	1402083	CS
2	Kelvin Ong Chun Yauw	1506094	CS
3	Jackson Tan Zhe Sheng	1402698	CS

TABLE OF CONTENT

Chapter 1: Introduction

1.1 Motivation and Problem Statement

Background subtraction is the most commonly used approach in setups with static cameras. It consists in using a model of the scene background in order to detect foreground objects by differencing incoming frames with the model. Background subtraction is mostly fast and has low computational demands. However, it can be distracted by the dynamic background or even affect by the sudden or gradual illumination changes, shadow, bootstrapping, and small camera motions as, e.g., vibrations.

Then it is an interesting task to make the computer to analyze the object moving detection. Thus, we are going to develop a video motion detection based background subtraction system that is able to retrieve images manipulation to differentiate the foreground from the background efficiency and effectiveness.

1.2 Problem scope

Our system involves at least the following research questions:

- Color quantization
- Color histogram computation
- Histogram intersection distance
- Dominant color extraction for background substraction
- Noice reduction using Morphological Transformations
- Dynamic background model
- Object tracking and trajectory determination

1.3 Project Objective

The main focus of this thesis is the real-time detection of objects in unrestricted environments monitored with static video cameras. The objects of interest are moving as well as new static objects. The video analytics system is not provided with any previous knowledge neither of the observed scene nor of the visual appearance of the objects to be detected. The main application in mind of the developed algorithms is the detection of abandoned objects in public spaces,

which has gained an important attention in the security domain, since abandoned objects might be often considered as a threat to the public security. The final system has to provide on-line alerts to human operators. Furthermore, the detected moving objects should be provided to higher-level analysis tools in order to recognize further actions and behaviour of interest typical of surveillance systems for public spaces

We shall combine and modify existing techniques to perform the video motion detection based background subtraction process in a different and better way compared with the existing systems. Our aim is to develop an efficient and effective video motion detection scheme which is insensitive to small variations in the image orientation and size as well as able to deal with the dynamic background and to predict the direction of the moving objects of the foreground. In order to increase the efficiency and speed of the system, the acquisition image is pre-processed and enhanced. This enables us to use a background subtraction approaches so that able identify for each frame the set of pixels that are significantly different from the previous frames in given an image sequences.

1.4 Impact, Significance and Contribution of the Project

The surveillance system developed will provide a robust detection feature in real-time manner. The surveillance purpose to monitor and detect moving object automatically without assistance from human and able to perform multiple object tracking. The system developed shall classify the scene captured into outside area (public space) or inside area (restricted zone). The system developed significantly increases personal safety and comfort in every time and everywhere. The system developed shall able to solve problem of sensitive to sudden or gradual illumination changes, shadow, bootstrapping, and small camera motions as, e.g., vibrations, and so on with applies some techniques such as background subtraction. The system developed shall able to let users felt desirable and reliable within the system.

1.5 Background Information

Background subtraction methods are widely exploited for object moving detection in a video in many applications such as video surveillance, traffic monitoring, and human gesture recognition. How to correctly and efficiently model and update the background model and how to deal with shadows are the most challenging aspects of such approaches.

The principle of this method is to build a model of the static scene (i.e. without moving objects) called background, and then compare every frame of the sequence to this background in order to discriminate the region of motion, called foreground (the moving objects). This approach requires images manipulation to differentiate the foreground from the background. For more easy to understanding, let assume we have 2 images X and Y, then we are manipulating these images to obtain image Z.

Background subtraction method is effective to improve the effect of moving object detection. So, any motion detection system based on background subtraction needs to proper handle for some critical situations such as noise in the image, due to poor quality image or video source, small movement of non-static objects such as tree branches and bushes blowing in the wind, variations in lighting conditions in different parts of the same object, gradual and sudden changes in the light conditions or object moving so fast that they are captured in only a single frame of the whole scene will affect the accuracy of the system. Different colour space also greatly affects in result of detection where if the colour of moving objects and background is almost similar, then that it could not able to detected the moving object. In order to get a prefect background model in a video, the capturing process is important where the camera must not be shaking to ensure the differences between frames are able to minimize as possible.

Chapter 2: Literature Review

2.1 Adaptive background mixture models for real-time tracking

Main ideas/concepts of the paper

- Usage of adaptive model against traditional non-adaptive model so that the background is able to change (for example, from morning to night the background colour change from bright to dark)
- Using a Gaussian distribution on a single pixel over time can model a single background
- The author used mixture of multiple Gaussian distribution to model the background so that the background can have many layers where each Gaussian distribution represent 1 layer of the background, hence, temporary static object can be absorbed as the background and when the object is removed, the true background can be restored easily
- If a current pixel value cannot fit into the background Gaussian model, then it is the foreground pixel
- The foreground pixels are connected to from a region
- The foreground region from current frame to next frame are tracked by using the Kalman filter

Important techniques disclosed

Properties of the distribution:

- The Gaussian distribution has to track the background lightning changes
- More recent pixel value is more important in the background estimation

Model for each pixel using Mixture of K Gaussian distribution (typically K is between 3 to 5)

In Mixture of K Gaussian distribution, the probability of observing a new pixel value X (X is a vector if the image is a RGB image) at time t is

$$P(X) = \sum_{i=1}^{K} \omega_i \eta(X, \mu_i, \Sigma_i)$$

where η is a Gaussian probability density function

$$\eta(X, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{1}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)}$$

and

Σ_i is the covariance matrix of the i^{th} Gaussian distribution ($\Sigma_i = \sigma_i^2 \boldsymbol{I}$) at time t

μ_i is the mean of the i^{th} Gaussian distribution at time t

σ_i^2 is the variance of the i^{th} Gaussian distribution at time t

ω_i is the estimation of the weight of i^{th} Gaussian distribution at time t

with the assumption of

red, green, and blue pixel values are independent and have the same variances.

Updating the Model for a new coming pixel value using K-means approximations

According to the standard process, the model should be updated using expectation maximization (known as EM algorithm). However, using EM algorithm to update the model for each pixel would result in a performance issue. Hence, the author uses K-means approximations to update the model. In K-means approximations, we try to fit a new coming pixel value into one of the existing K Gaussian distribution. If the fitting is successful, then we updating the Gaussian model only. The exact algorithm is as below:

- Check the new coming pixel value X against each of the K Gaussian distribution, one by one until a match (X is within 2.5 standard deviation of the distribution) is found
- If a match is found, update the matched Gaussian distribution with the following formula
    - $\mu_{new} = (1-\rho)\mu_{old} + \rho X$
    - $\sigma_{new}^2 = (1-\rho)\sigma_{old}^2 + \rho(X-\mu_{new})^T(X-\mu_{new})$
    - where the second learning rate, $\rho = \alpha\eta(X|\mu_{old}, \sigma_{old})$
    - and α, is the learning rate
- If no match is found, the least probable of the K Gaussian distribution is discarded and replace with a Gaussian distribution where
    - $\mu = X$
    - $\sigma^2 =$ a high initial value

    - $\omega =$ a low initial value
- After that for all K Gaussian distribution, the weight is adjusted as below
    - $\omega_{i,new} = (1-\alpha)\omega_{i,old} + \alpha(M_i)$
    - where α is the learning rate
    - M_i is 1 for the distribution which matched and 0 for the other distribution

Choose the background distribution

For a distribution to be chosen as a background model, it should have the most supporting evidence and the least variance. Hence, it is sensible to use the heuristic value ω/σ. Higher ω/σ means that the distribution is the most probable background model. Thus, we first sort the distributing from highest ω/σ value to lowest. Then, chosen the first B distribution as the background model where the sum of the weight of the first B distribution **FIRST** exceed T, where T is a measure of the minimum portion of the data that should be accounted for by the background. In others words, the first B distribution is chosen such as sum of first B-1 distribution does not exceed T and sum of first B distribution exceed T. Formally,

$$B = argmin_b \left(\sum_{i=1}^{B} \omega_i > T \right)$$

This takes the "best" distributions until a certain portion, T, of the recent data has been accounted for.

Form Connected Component

After the background model had been determined, we can know a pixel is a foreground pixel if it does not fit into the background model. After filtering out all the foreground pixel, we connect all the neighbouring foreground pixel to form foreground region by running the connected component algorithm.

Tracking the Component

In order to track the component, each component on current frame has to be related to the next frame. Hence, the author uses a Kalman filter for each component to track the component. At current frame, author created a pool of Kalman model form by Kalman model of each

component. At next frame, a new pool of Kalman model is form similarly. Then, for each Kalman model in the new pool, it is matched with the old pool. If a match is obtained, it indicates that it is the same component.

2.2 Real-time foreground-background segmentation using codebook model

Main ideas/concepts of the paper

- The paper proposed real-time algorithm to segment the foreground from the background by quantized the sampled values of each pixel from the background into codebooks that act as the compressed background model for a long image sequence.
- By using the codebook implementation, it further improved the efficiency in memory and speed than the other background modelling techniques.
- 2 addition features of the algorithm are proposed:
    - Layered modelling / detection
    - Adaptive codebook updating

Important techniques disclosed

The key features of the codebook background subtraction algorithm:

- Structural background motion extraction over a long period without making parametric assumptions and massive memory consumption which allow the encoding of the dynamic or multiple changing backgrounds.
- Ability to handle both of the local and global illumination changes.
- Moving foreground objects are allowed due the unconstrained training of the initial period.
- Multiple background layers that represented by different layers of background with the layered modelling and detection

The algorithm is described for color imagery, but it can also be used for gray-scale imagery with minor modifications. Let $\mathscr{X}$ be a training sequence for a single pixel consisting of N RGB-vectors: $\mathscr{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$. Let $\mathscr{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_L\}$ 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

Algorithm for Codebook construction

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

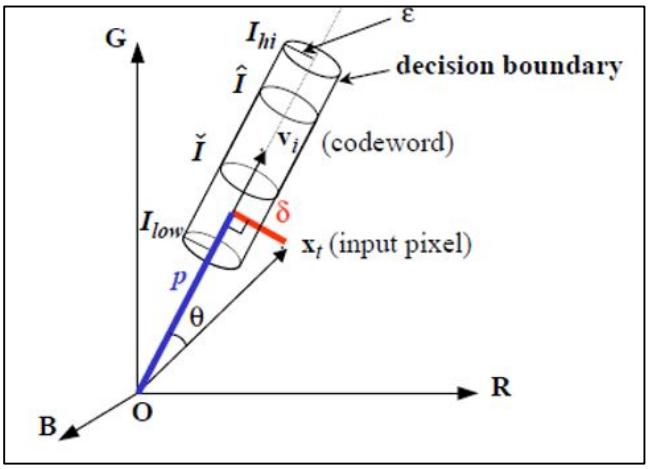
$\mathbf{v}_m = (\bar{R}_m, \bar{G}_m, \bar{B}_m)$ and $\mathbf{aux}_m = \langle \check{I}_m, \hat{I}_m, f_m, \lambda_m, p_m, q_m \rangle$, by setting

- $\mathbf{v}_m \leftarrow \left(\frac{f_m \bar{R}_m + R}{f_m + 1}, \frac{f_m \bar{G}_m + G}{f_m + 1}, \frac{f_m \bar{B}_m + B}{f_m + 1} \right)$
- $\mathbf{aux}_m \leftarrow \langle \min\{I, \check{I}_m\}, \max\{I, \hat{I}_m\}, f_m + 1, \max\{\lambda_m, t - q_m\}, p_m, t \rangle$.

end for

III. For each codeword $\mathbf{c}_i$, $i = 1, \ldots, L$, wrap around λ_i by setting $\lambda_i \leftarrow \max\{\lambda_i, (N - q_i + p_i - 1)\}$.

The algorithm for codebook construction is used to determine the best matched codeword as the sample's encoding approximation by using the colour distortion measure and brightness bounds by referring the steps given above.



The proposed colour model is based on the divided evaluation of the colour and brightness distortion in order to handle the local and global illumination changes such as:

- Shadows
- Highlights

Algorithm for Background subtraction

I. $\mathbf{x} = (R, G, B)$, $I \leftarrow \sqrt{R^2 + G^2 + B^2}$

II. For all codewords in $\mathcal{M}$ 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. $$BGS(\mathbf{x}) = \begin{cases} \textbf{foreground} & \text{if there is no match} \\ \textbf{background} & \text{otherwise.} \end{cases}$$

The foreground detection is done by computing the distance of the sample from nearest cluster mean instead of using the probability calculation to compute the probabilities with the floating point operations which are very costly. This will further reduce the computation time and having a very similar result with the estimated probability.

2.3 An Adaptive Background Subtraction Method Based on Kernel Density Estimation

by Jeisung Lee and Mignon Park

Jeisung. L at el (2012) proposed the kernel density estimation (KDE) method, a non-parametric approach that can effectively adapt to a dynamic background. In each pixel, the KDE is calculated by the following equation at time index *t.*

$$p(x) = \frac{1}{n}\sum_{i=1}^{n} K(x - x_t)$$

where

n is the number of total observed frames

xt is the observed value at time index t.

p(x) is an average of normal densities centered at the sample x.

The kernel function K(x) should satisfy the following conditions: $\int K(x)dx=1$, $\int xK(x)dx = 0$, $and\ K(x) > 0$. Typically, the normal distribution N (0, 1) is used as the kernel function. In an investigation conducted by Park et al., Many frames are collected before estimating Gaussian background models and therefore, a large memory space is required. To overcome this weakness, Jeisung. L at el (2012) modify the original KDE method and propose a scheme that uses the first frame to start the KDE background model. In the first frame, most pixels represent the background, and there is a foreground in some other pixels. Although Jeisung. L at el (2012) use the first frame to start a background model, background information will be reduced and remain only background information by updating the process because the background value is more frequent than the foreground value at the pixel level. The KDE Gaussian model is then updated in each frame by controlling learning rate according to the situation. The probability $pt\ (x)$ is based on each pixel and can be expressed as:

$$p_t(x) = \hat{p}_{t-1}(x) + \frac{1}{G_t\sqrt{2\pi\sigma^2}} \exp\left[\frac{1}{2}\left(\frac{x - x_t}{\sigma}\right)^2\right]$$

Each pixel has a probability model. The probability obtained by the KDE method is added to the prior probability density at every frame. In second equation, G_t is used as the learning rate at time t and can be changed depending on factors such as time and illumination changes. Since the probability should satisfy $\int pt(x)dx = 1$, $pt(x)$ is normalized as follows:

$$\hat{p}_t(x) = p_t(x) \,/\, \sum_{x=0}^{N} p_t\,(x)$$

where

$pt(x)$ is a normal density at the sample x and at time index t.

$\hat{pt}(x)$ is a normalized normal density and N is the total number of samples.

A new probability background model is obtained through the above process. This updating method improves memory effectiveness as it does not require many images to be stored to start a probability background model. The method of updating automatically reduces the probability of non-essential backgrounds that do not appear over a long period of time by adding additional probabilities and performing normal steps. For example, when a parked car for a long period of time moves or disappears, the proposed method continues to update the environment. As a result, new background information appears and the probability of a previously unimportant background associated with the car is automatically downgraded by updating the background model. Jeisung. L at el (2012) use Gt as a parameter to control learning rates. If Gt increases, new information is slowly studied and previous information is slowly lost. If Gt is reduced, algorithms quickly adapt to the environment and quickly delete old information. In the initial stage, the background model should adapt to a new environment and, when the time comes, the background should have a stable updating process. For this reason, Gt has been used as a sigmoid function which can be stated as follows:

$$G_t = Gain * \frac{2}{1 + \exp(-(cnt - \beta)/\lambda}$$

The value of G_t over time is shown in Figure 1.

With applies the equation, the value of *cnt* increases proportionally with respect to time and can be used to initialize the background by initializing or control the learning rate through the environment by initializing the value of *cnt*. The inflection point is controlled by β and *Gain*, while the gradient can be changed by λ. The learning rate of the proposed method is affected by the *Gain* parameter. If the *Gain* parameter increases, the learning rate of the algorithm will decrease and *vice versa*. For example, β was set to 100, the *Gain* was 300, and λ s 20.

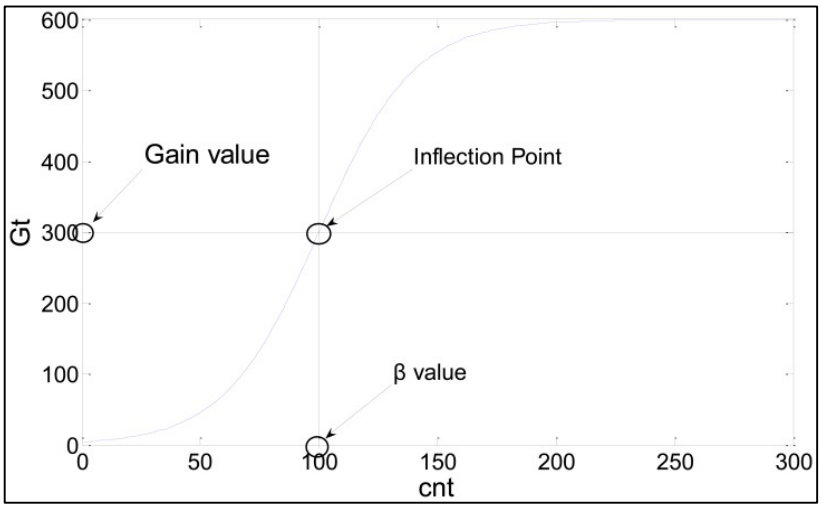


Figure 1: Example of the value of Gt over time

A several of the problems associated with the non-parametric kernel density estimation approach are the undesirably long processing time and the large memory requirement. Jeisung. L at el (2012) stated that they can reduce the complexity and memory requirement using histogram approximation. The Gaussian probability and an example of histogram approximation. Bd is the width of the histograms along dimension d, Ck is the center of each histogram, and k is the histogram number? The parameter Bd can be calculated according to the following equation.

$$B_d = \frac{\max(x^d) - \min(x^d)}{N_d} \qquad d = 1,2,3$$

where

Nd represents the number of bins for each dimension *d* and

x^d is the value of a pixel in the *d* dimension.

A general image has three dimensions: R, G, and B. Thus, the range of *d* is $1 \leq d \leq 3$. The change in the kernel density estimation by histogram approximation may be expressed as follows:

$$p_t{}^d(C_k) = \hat{p}_{t-1}{}^d(C_k) + \frac{1}{G_t\sqrt{2\pi(B_d/2)^2}} \exp\left(-\frac{1}{2}\left(\frac{C_k - x_t^d}{B_d/2}\right)^2\right) \qquad k = 1,2\ldots.N_d\, d = 1,2,3 \tag{6}$$

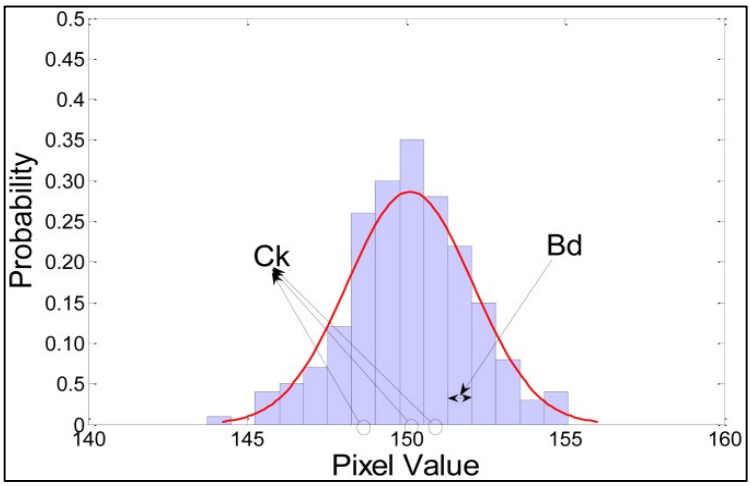


Figure 2: Gaussian probability and an example of histogram approximation; B_d is the width of the histograms in dimension d and C_k is the center of each histogram.

To reduce the complexity, by taking the integer part after dividing the input with the width of the bin, they may directly find the bin number which the current input belongs to. The *floor* function means that *floor*(A) rounds the elements of A to the nearest integers less than or equal to A:

$$k = floor(x_t^d / B_d)$$

where x_t^d is input value, B_d is the width of the bin, and the k = 0,2, 3, …, N_d -1.

For instance, if the input sequences have values in [0 255] and take a bin width of B_d = 4, the bin numbers k of the histogram have values in [0 63]. If the input value is 150, they may find the bin number using the equation above, k = *floor* (150/4) = 37. So the input belongs to the 37th histogram. By using this method, they can be avoided to search the bins one by one which means a reduction to the complexity.

To update the probability histogram, Jeisung. L at el (2012) applied a Gaussian whose mean value is the input as Equation (6). They can ignore the inference which the input gives to the remote bins, since the Gaussian value quickly falls off towards plus/minus infinity. They calculate the probability of the KDE background model not in whole bins but only for the back and forth less than or equal to $B_d/2$ bins. For example, if the closest center of the input x_t^d was C_k and the B_d was 4, then they only update the background probability histogram

$P_t^d(C_{(k-2)})$~$P_t^d(C_{(k+2)})$, with the Equation (6) using $C_{(k-2)}$~$C_{(k+2.}$ Figure 3 shows an example of this.

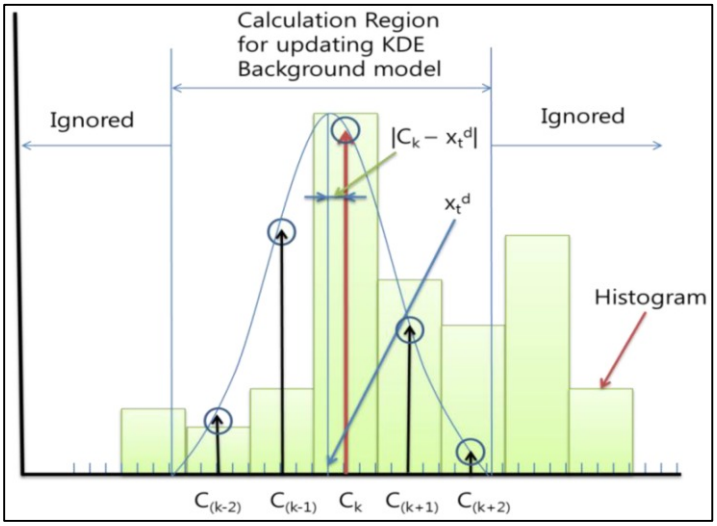


Figure 3: An example of how the histogram was updated.

When they update the $P_t^d(C_{(k-2)})$~$P_t^d(C_{(k+2)})$, they need to find the Gaussian value of each bin. If they calculate the following equation, which is a computational component of Equation (6), every time and in every pixel, it takes too much time. So to reduce the computational cost, they previously calculated and saved the Gaussian probability results according to the case of the difference between the closest center C_k and input value x_t^d:

$$Pre_p = \frac{1}{\sqrt{2\pi(B_d/2)^2}} \exp\left(-\frac{1}{2}\left(\frac{C_k - x_t{}^d}{B_d/2}\right)^2\right)$$

Jeisung. L at el (2012) has consider the case in the example before. If the B_d is 4, there are only four possible Gaussians according to the input value and they only need to save $B_d + 1 = 5$ values for each Gaussian in each bin as Figures 3 and Figure 4 (the total of the points are $B_d \times (B_d + 1) = 20$). It can cover all the cases they will meet on the updating process. However, the B_d is not usually integer, so if they set $G = floor(B_d/2)$, they are updated the background probability histogram $P_t^d(C_{(k-G)})$~$P_t^d(C_{(k+G)})$. The total of the points will be $ceil(B_d) \times (floor(B_d) + 1)$, where *ceil*(A) rounds the elements of A to the nearest integers greater than or equal to A and *floor*(A) rounds the elements of A to the nearest integers less than or equal to A.

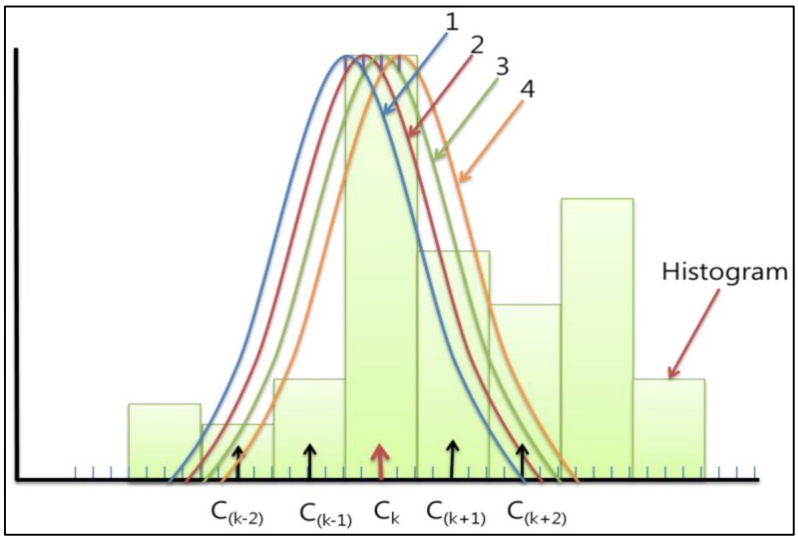


Figure 4: An example of possible Gaussians in a bin.

Chapter 3: Proposed Method/Approached for Simple System

3.1 Project topic title

Our project topic title is Video Motion Detection Based on Background Subtraction. We need an efficient algorithm for computer-vision program to track moving object in a video. Although many algorithm and method has been proposed, most of them have their shortcoming in terms of their accuracy, performance, memory requirement. There are 3 most common methods in detecting motion in a video, namely, background subtraction, frame subtraction, and optical flow. We are going to use the background subtraction method.

3.2 Methodology

Normally, background subtraction method is used in static camera, hence, the background will be static (not moving). Background subtraction involved in building a background model. Then we can determine for each pixel, it is a background pixel or a foreground pixel. All the foreground pixel can be classified as a "moving" object (because the background is static, hence the foreground can be said as "moving"). Hence, we are able to detect motion through this method.

3.3 Implementation of Simple System

In our implementation, we assign a (or a set of) dominant colour to each pixel location. In training phase, we learn the background through accumulation of quantized colour bin value across each frame of the training video for each pixel location. After training phase, the background model can be from using the colour that occurs the most across all the frame of training video for each pixel location.

At testing phase, for each pixel location, we obtain an absolute different of colour value between the background model and video frame. If the different exceed certain threshold, it is marked as a foreground. The foreground is identified as "moving object". After we obtain a foreground mask, we run a connected component algorithm on the foreground mask to obtain the bounding rectangle of moving objects and then draw the bounding rectangle on the original video frame for better visualization.

3.4 Result of the Simple System

The sample video that we used for the project is the 'parking.mp4' file which provided by our lecturer, Prof. Dr Zen Chen that able to be downloaded from the WBLE in order to train and test our program. The screenshots below indicated the intermediate result and the final output of our proposed system.

The first step is training phase which is to obtain the background model from the video. The figure below shows our simple program intermediate output.

Learning Progress	Learned Background
	

Then, after the background model had obtained from the training phase, we compute a foreground mask for each frame from the video by subtracting the background model in testing phase.

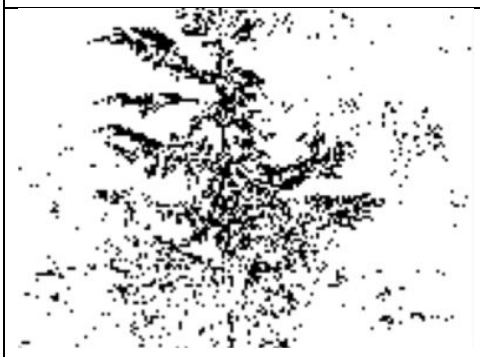
Foreground Mask	Original Video
	

Then, based on the computed foreground mask, we computed the connected component in the foreground mask and obtain the bounding boxes and centroid of each object.

Original Video	Video with Bounding Box
	

The image above shows the final detection of our simple system. It can be seen that, the moving objects of the foreground such as the cars and people of the sample video are being detected and tracked by using a red bounded rectangle and green centroid which indicated each centre of the moving object.

3.5 Implementation Issues and Challenges of the Simple System

Foreground Mask	Original Video
	

There are some issues for the simple system such as it failed to process the video with the dynamic background as shown as the table. The dataset above is the “WavingTrees.avi” which consisted of a tree that shake through the entire video which further caused the system unable to difference between the background (waving tree) and foreground (moving object such as a person passing by). The simple system is also unable to process the video which consist of a lot of noises such as the shadows and reflection in the datasets of the “CampusStreet.avi” and “contrysideTraffic.mp4”.

3.6 Method to Overcome Challenges in Simple System

In order to solve the dynamic background problem, we need to make use of multiple code word as the background code book. In order to achieve this, we introduction one more parameter to the code book for each code word which is the maximum negative run length of each code word, lambda. The lambda is defined as the maximum number of consecutive frames during which the particular code word does not appear. Also, in order to calculate lambda, we need to keep track of the frame number of the last code word appearance too.

The lambda and last frame number will be initialized to 0. When a particular code word appears, we update the lambda as the maximum of lambda and current frame - last frame, and also update the last frame as current frame accordingly.

After we have obtained the lambda of each code word, we select those code word that has a quite high frequency and relatively small lambda as the dynamic background code word.

3.7 Additional Features for Advance System

In advance system, we will implement the above method which introduce lambda to enable dynamic background model to solve our simple system weakness.

Furthermore, to achieve object tracking, we will visualize the trajectory estimation of the object. To estimate the trajectory of the object, we give each object an attribute called velocity. First of all, we try to associate a detected object in current frame with a detected object in previous frame. We associate the object to each other by using the heuristic: “If the centroid of both object is close to each other across the frame, then they are the same object”. An association of the object indicates that both are the same object.

The velocity can then be computed by the following formula, velocity = current position - previous position. However, after some testing, we decided to introduce some heuristic for a more accurate velocity estimation. By using the fact that, previous object also carries a velocity, we must take account into previous object velocity too when determining the current object velocity. Hence, we use the following formula for determining current object velocity.

$$\text{appearance velocity} = \text{current position} - \text{previous position}$$

$$\text{current velocity} = \alpha \times \text{appearance velocity} + (1 - \alpha) \times \text{previous velocity}$$

where α is the learning rate (we set it to 0.2 in our implementation).

Once we obtained a more accurate velocity estimation for each tracked object, then we can draw out the trajectory estimation for each object by using arrowed line as visualization.

3.8 Technical Implementation of Advance System

First in **training()**, we read the video, loop through each frame and used the frames to learn the background model by using **background.learn()**.

In **background.learn()**, we loop through each pixel of the training frame and update code book using **pixelCodeBook.update()**. In **pixelCodeBook.update()**, we increase the code word quantity, compute the lambda and update the last appeared frame.

After learning the background, **background.get_background_model()** is used to obtain the background image and it can be displayed for visualization purpose.

Then, at testing phase, the video is read and each frame is looped through for testing. For each frame, **background.get_foreground_mask()** is used to obtain the foreground masking of the testing frame based on the learned background model. In **background.get_foreground_mask()**, we obtain the foreground mask by looping through each pixel, then compare the current pixel value with the background model by using **pixelCodeBook.compare_color()**. Then, assign a white pixel value to indicate foreground and black otherwise.

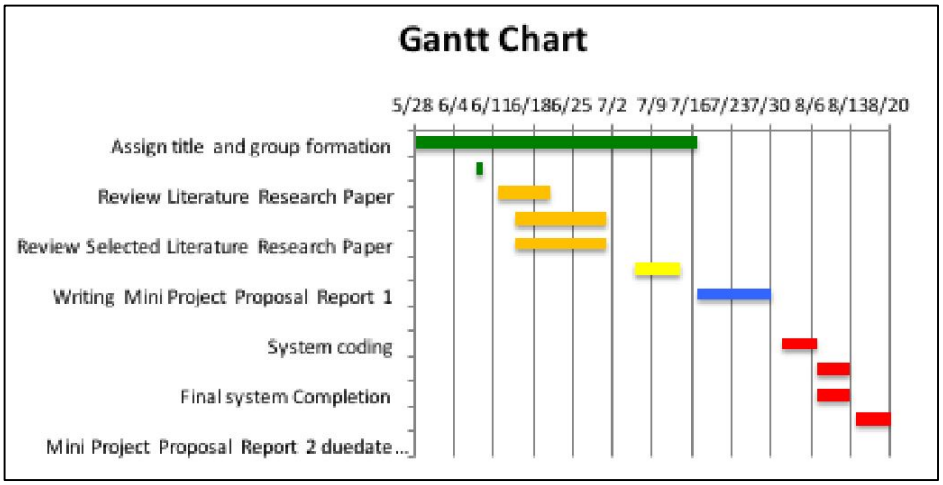
In **pixelCodeBook.compare_color()**, we first tried to match the incoming pixel value with the static background model, if it matches the static background model, then return true (which indicate it is a background pixel). It the static background matching fails, then it will proceed with trying to match the dynamic background model. If the dynamic background matching fails, then it will return false (which indicate it is a foreground pixel) and true otherwise.

After obtaining the foreground mask, the noise in the foreground mask is reduced by a series of opening and closing morphological transformation. Then, the connected component is obtained where each connected component indicates one detected object. Each detected object is stored into a vector which later will be used to perform trajectory estimation. After each object is stored into a vector, we compute the velocity of each object by using the **trajectoryModel.compute_velocity()** function.

In **trajectoryModel.compute_velocity()**, we first try to associate an object from previous frame to current object. The association is done by using the centroid of both object, if their centroid is close to each other, it is said to be the same object. After associate the object, the velocity can be computed using the centroid position difference.

Finally, by using the function **objects.draw()** function, the object information such as centroid, the bounding box and trajectory estimation is draw on the original video frame for better visualization.

3.9 Project Timeline and Milestones for Whole System Implementation



Task Name	Start	End	Duration (days)
Assign title and group formation	5/28/2017	7/17/2017	50
Introduce OpenCV installation environment	6/8/2017	6/9/2017	1
Review Literature Research Paper	6/12/2017	6/21/2017	9
Identifies Project Scope	6/15/2017	7/1/2017	16
Review Selected Literature Research Paper	6/15/2017	7/1/2017	16
Design and Propose a new solution	7/6/2017	7/14/2017	8
Writing Mini Project Proposal Report 1	7/17/2017	7/30/2017	13
Mini Project Proposal Report 1 duedate submission	7/30/2017	7/30/2017	0
System coding	8/1/2017	8/7/2017	6
Testing error and debugging	8/7/2017	8/13/2017	6
Final system Completion	8/7/2017	8/13/2017	6
Writing Mini Project Proposal Report 2	8/14/2017	8/20/2017	6
Mini Project Proposal Report 2 duedate submission	8/20/2017	8/20/2017	0

Planning Phase

Week 1 to week 2

- Defining the problems, the objectives and the resources
- Studying the ability of proposing alternative solution with team member.
- Studying how to make our system better than other existing system.
- Research and do more relevant reading of past work to understand how a similar system is implemented.

Analysis Phase

Week 3 to week 5

- Conduct the literature reviews from the relevant papers.
- To determined and documented what our expectations for the system and how it will be perform.

Design Phase

Week 6 to week 9

- Defines the elements of our system, the modules, architecture and searching for suitable database.
- Produces the proposal for the whole project and starts to plan for whole system.

Implement Phase

Week 10 to week12

- Implement the full system that can fulfil the requirements from the proposal.
- Start to deploy and running our system, testing the accuracy of system. After testing, start debugging and fix the error.

Future work

- Continue maintaining and enhancing the system.

Project Milestone

We will be using the following milestones to track project progress and accomplishments:

1. Working backbone of system that considered commonly used to detect foreground objects by differencing incoming frames with the model
2. Finish implementation of colour quantization to reduce the number of colours in the image frame
3. Finish partition of the image frame and then compute colour histogram of the image frame and store the value as image index.
4. Finish implementation of find the similarity computation based on integration of shape, colour and texture for each frame the set of pixels that are significantly different from the previous frames in given an image sequences.
5. Finish Full system implementation. So our system will be able to perform images manipulation to differentiate the foreground from the background and show out the result.

Chapter 4: System Implementation, Testing, and Verification for Advanced system

4.1 Description of hardware and software environments used for project implementations

Required system hardware environment

Operating System: Microsoft Windows 10 Home
System Type: x64-based PC
Processor: Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz, 2301Mhz, 2 Core(s)
Graphical Processing Unit (GPU): Nvidia GeForce GT 740M
Total Physical Memory: 7.78GB

Required system software environment

OpenCV-3.1.0
Microsoft Visual Studio 2017

Datasets Used

1. parking.mp4

   Type of file: MP4 File (.mp4)
   Size: 5.61 MB
   Length: 3 minutes 7 seconds
   Frame width: 320
   Frame height: 240
   Frame rate: 8 frames/second
   Description:

2. fountain01.mp4

   Type of file: AVI File (.avi)
   Size: 27.6 MB
   Length: 39 seconds
   Frame width: 432
   Frame height: 288
   Frame rate: 30 frames/second

3. WavingTrees.avi

   Type of file: AVI File (.avi)

Size: 2.42 MB

Length: 39 seconds

Frame width: 160

Frame height: 120

Frame rate: 30 frames/second

4. contrysideTraffic.mp4

   Type of file: MP4 File (.mp4)

   Size: 4.68 MB

   Length: 1 minute 2 seconds

   Frame width: 480

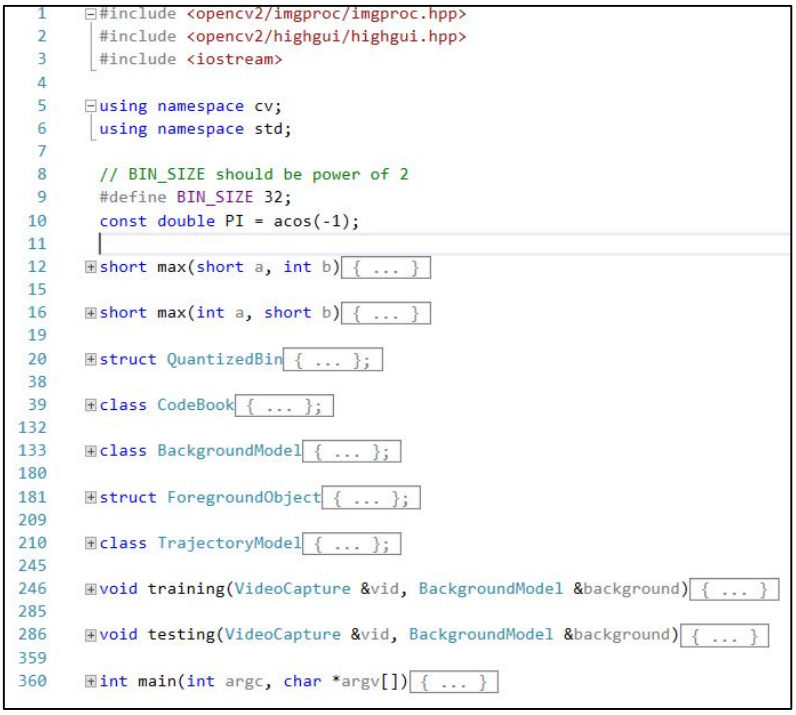
   Frame height: 270

   Frame rate: 29 frames/second

5. contrysideTraffic.avi
   - Type of file: MP4 File (.mp4)
   - Size: 4.68 MB
   - Length: 1 minute 2 seconds
   - Frame width: 480
   - Frame height: 270
   - Frame rate: 29 frames/second
6. WetSnow.mp4
   - Type of file: MP4 File (.mp4)
   - Size: 26.3 MB
   - Length: 56 seconds
   - Frame width: 720
   - Frame height: 540
   - Frame rate: 30 frames/second

4.2 System Code Implementation

Code Documentation



```
1   #include <opencv2/imgproc/imgproc.hpp>
2   #include <opencv2/highgui/highgui.hpp>
3   #include <iostream>
4
5   using namespace cv;
6   using namespace std;
7
8    // BIN_SIZE should be power of 2
9    #define BIN_SIZE 32;
10   const double PI = acos(-1);
11
12  short max(short a, int b) { ... }
15
16  short max(int a, short b) { ... }
19
20  struct QuantizedBin { ... };
38
39  class CodeBook { ... };
132
133 class BackgroundModel { ... };
180
181 struct ForegroundObject { ... };
209
210 class TrajectoryModel { ... };
245
246 void training(VideoCapture &vid, BackgroundModel &background) { ... }
285
286 void testing(VideoCapture &vid, BackgroundModel &background) { ... }
359
360 int main(int argc, char *argv[]) { ... }
```

The program of our proposed system is implemented by using the Object-Oriented based concept which further divided into 9 parts that consist of 2 structures, 3 classes and the 4 functions which included the main, 2 overloaded max methods, training and testing functions:

Functions:

- Main function
- Overloaded max methods
- training
- testing

Structures:

- QuantizedBin
- ForegroundObject

Classes:

- BackgroundModel
- CodeBook
- TrajectoryModel

main()

```
int main(int argc, char *argv[]){
    String filename = "parking.mp4";
    if(argc > 1){
        filename = argv[1];
    }
    VideoCapture vid(filename);
    if(!vid.isOpened() || !vid.grab())
        return -1;

    int v_height = vid.get(CV_CAP_PROP_FRAME_HEIGHT);
    int v_width = vid.get(CV_CAP_PROP_FRAME_WIDTH);
    int total_frame = vid.get(CV_CAP_PROP_FRAME_COUNT);
    double fps = vid.get(CV_CAP_PROP_FPS);

    cout << endl;
    cout << "---------------------- Video Information ----------------------" << endl;
    cout << "Video Name: " << filename << endl;
    cout << "Video Frame: " << total_frame << endl;
    cout << "Video FPS: " << fps << endl;
    cout << "Video Length: "; printf("%d min %d sec\n", (int)(total_frame/fps/60), (int)(total_frame/fps)%60);
    cout << "Video Dimension: " << v_width << " x " << v_height << endl;
    cout << endl;

    BackgroundModel background(v_height, v_width);

    training(vid, background);
    cout << endl << endl;

    vid.open(filename);
    if(!vid.isOpened() || !vid.grab())
        return -1;

    testing(vid, background);

    return 0;
}
```

The first part of the main function, pre-training phase. At here, the video is first loaded from the file system, and the metadata of the video is read and output to the console for user information. Then, it enters the training phase.

training()

```
void training(VideoCapture &vid, BackgroundModel &background){
    cout << "---------------------- Training Phase ------------------------" << endl << endl;
    cout << "  Training Progress    Processing Speed          ETA" << endl;

    long long t0 = cv::getTickCount(), t1;
    int total_frame = vid.get(CV_CAP_PROP_FRAME_COUNT);
    double fps = vid.get(CV_CAP_PROP_FPS);

    int f = 0;
    while(vid.grab()){
        Mat frame;
        vid.retrieve(frame);

        background.learn(frame);
        imshow("Learning Progress", frame);
        waitKey(1);
        f++;

        if(f%10 == 0){
            double cur_fps = 10.0/(cv::getTickCount()-t1)*cv::getTickFrequency();
            int eta = (total_frame-f)/cur_fps;
            printf("\r         %d %%  \t\t   %0.2lf FPS \t     %d min %d sec         ",
                f*100/total_frame, cur_fps, eta/60, eta%60);
            fflush(stdout);
            t1 = cv::getTickCount();
        }
    }

    vid.release();
    imshow("Learned Background", background.get_background_model());

    cout << endl << endl;
    cout << "Training Phase Completed\nTotal Elapse Time: ";
    int elapsetime = (int)((cv::getTickCount()-t0)/cv::getTickFrequency());
    printf("%d min %d sec \n\n", elapsetime/60, elapsetime%60);

    cout << "Press Any Key to Continue" << endl;
    waitKey(0);
}
```

In training phase, each video frame is looped through and the background model is being learn by using the function background.learn() function. While learning, some information such as the progress, processing speed and estimated time to complete are displayed to the console for user information. After the training is completed, the background model is displayed. After that it enters the testing phase.

testing()

```
void testing(VideoCapture &vid, BackgroundModel &background){
    cout << "----------------------- Testing Phase -----------------------" << endl << endl;
    cout << "   Testing Progress    Processing Speed          ETA" << endl;

    int total_frame = vid.get(CV_CAP_PROP_FRAME_COUNT);
    double fps = vid.get(CV_CAP_PROP_FPS);
    long long t0 = cv::getTickCount(), t1;
    TrajectoryModel trajectoryModel;
    int f = 0;
    while(vid.grab()){
        Mat frame;
        vid.retrieve(frame);
        imshow("Original Video", frame);

        Mat foregroundMask = background.get_foreground_mask(frame);
        imshow("Forground Mask", 255-foregroundMask);

        Mat labels, stats, centroids;
        Mat strucElement = getStructuringElement(MORPH_ELLIPSE, Size(2, 2));
        morphologyEx(foregroundMask, foregroundMask, MORPH_CLOSE, strucElement);
        morphologyEx(foregroundMask, foregroundMask, MORPH_OPEN, strucElement);
        morphologyEx(foregroundMask, foregroundMask, MORPH_OPEN, strucElement);
        strucElement = getStructuringElement(MORPH_ELLIPSE, Size(4, 4));
        morphologyEx(foregroundMask, foregroundMask, MORPH_CLOSE, strucElement);
        imshow("Morphology Transform", 255-foregroundMask);
        int nLabels = connectedComponentsWithStats(foregroundMask, labels, stats, centroids);
```

At the first part of testing phase, the foreground mask is computed by using the function background.get_foreground_mask(). Then, a series of opening and closing morphological transformation are applied on the foreground mask to reduce the noise. All intermediary images are displayed for debugging and better visualization purpose. Then, run connected component algorithm on the noise reduced foreground mask to obtain the region of detected object.

```
    vector<ForegroundObject> objects;
    for(int i = 1; i < nLabels; i++){
        if(stats.at<int>(i, CC_STAT_AREA) < 20)
            continue;

        int x = stats.at<int>(i, CC_STAT_LEFT);
        int y = stats.at<int>(i, CC_STAT_TOP);
        int w = stats.at<int>(i, CC_STAT_WIDTH);
        int h = stats.at<int>(i, CC_STAT_HEIGHT);

        ForegroundObject curObject(Point(centroids.at<double>(i, 0), centroids.at<double>(i, 1)),
            stats.at<int>(i, CC_STAT_AREA), x, y, h, w);

        objects.push_back(curObject);
    }

    trajectoryModel.computeVelocity(objects);

    for(int i = 0; i < objects.size(); i++)
        objects[i].draw(frame);

    imshow("Original Video with Bounding Box", frame);

    double sec = (cv::getTickCount()-t0)/cv::getTickFrequency();
    if(sec < 1.0/fps)
        waitKey(1000.0/fps - sec*1000 + 1);
    else
        waitKey(1);

    f++;
    if(f%10 == 0){
        double cur_fps = 10.0/(cv::getTickCount()-t1)*cv::getTickFrequency();
        int eta = (total_frame-f)/cur_fps;
        printf("\r         %d %%  \t\t   %0.2lf FPS \t     %d min %d sec         ",
            f*100/total_frame, cur_fps, eta/60, eta%60);
        fflush(stdout);
        t1 = cv::getTickCount();
    }
}
```

After the connected component algorithm, each component is looped through and the object is stored inside a vector for trajectory computing. After that, trajectoryModel.computeVelocity() function is used to compute the velocity for each object. After the velocity had been computed, the information of each object such as centroid, bounding box, estimated trajectory are drawn using the function objects.draw(). Information such as progress, processing speed and estimated time to complete are displayed to the console for user information.

QuantizedBin

```
struct QuantizedBin{
    short quantity, lambda;
    Vec3b color;
    vector<short> frame_index;

    QuantizedBin(){}
    QuantizedBin(Vec3b color){
        lambda = 0;
        frame_index.reserve(1);
        frame_index.push_back(-1);
        quantity = 0;
        this->color = color;
    }

    bool operator< (const QuantizedBin x) const {
        return this->quantity < x.quantity;
    }
};
```

The 'QuantizedBin' structure consists of the 4 variables which are the quantity, lambda, last and colour with the types of integer, integer, integer and Veb3b respectively. It used to store the frequency of this bin, lambda of this bin, last frame number that this bin appeared and the colour that the bin represents. It consists of the overloaded less than operator for the sorting algorithm.

QuantizedBin::QuantizedBin()

```
QuantizedBin(){}
```

Parameters:

None

It is an empty constructor with no parameter which used to construct, initialize the object and its variables.

QuantizedBin::QuantizedBin(Vec3b color)

```
QuantizedBin(Vec3b color){
    lambda = 0;
    frame_index.reserve(1);
    frame_index.push_back(-1);
    quantity = 0;
    this->color = color;
}
```

Parameters:

color – It is a Veb3b type variable that used to initialize the color of the 'QuantizedBin' structure.

It is a constructor with a single parameter which used to construct object as well as initialize its quantity to 0, frame_index and color by using the pass-in parameter.

BackgroundModel

The class of 'BackgroundModel' is used to create objects that able to build and store the background model for a video and be further used to obtain the foreground mask. It consists of 3 variables which are private as well as 1 constructor and 3 methods which are public.

```
class BackgroundModel{
  private:
    // every pixel has a CodeBook
    int rows, cols;
    vector<CodeBook> pixelCodeBook;
```

The 'BackgroundModel' class consists of 2 integer variables and a vector of CodeBook object which are the rows, cols and pixelCodeBook which can only be accessed within the class.

BackgroundModel::BackgroundModel (int rows, int cols)

```
public:
  BackgroundModel(int rows, int cols){
      this->rows = rows;
      this->cols = cols;

      pixelCodeBook.assign(rows*cols, CodeBook());
  }
```

Parameters:

rows - the number of rows of the background model should contain

cols - the number of columns of the background model should contain

The constructor which used to construct the object and initialize the object attribute. It assigns an empty code book for each pixel of the video frame.

void BackgroundModel::learn(Mat frame)

```
void learn(Mat frame){
    for(int i = 0; i < rows; i++){
        for(int j = 0; j < cols; j++){
            // mat.at<>(r, c);
            pixelCodeBook[i*cols+j].update(frame.at<Vec3b>(i, j));
        }
    }
}
```

Parameters:

frame - It is Mat object that indicated the frame of the video which background model to be extracted.

The learn() method learn the background model for the input frame and provide all frame of the video to this method to learn the background model and return void after executed. This is achieved by looping through each pixel and updating the CodeBook at the corresponding pixel with current pixel color.

Mat BackgroundModel::get_background_model()

```
Mat get_background_model(){
    Mat background(rows, cols, CV_8UC3);
    for(int i = 0; i < rows; i++){
        for(int j = 0; j < cols; j++){
            // mat.at<>(r, c);
            background.at<Vec3b>(i, j) = pixelCodeBook[i*cols+j].get_color();
        }
    }
    return background;
}
```

Parameters:

None

This method is used to get and return the learned background model after providing all video frame to the learn() method. The function first created an empty picture, then loop through each pixel and assign the pixel with its dominant color retrieved from the CodeBook.

Mat BackgroundModel::get_foreground_mask(Mat frame)

```
Mat get_foreground_mask(Mat frame){
    Mat foreground(rows, cols, CV_8UC1);
    for(int i = 0; i < rows; i++){
        for(int j = 0; j < cols; j++){
            // mat.at<>(r, c);
            // compare color return true(1) false(0), multiple 255 to get binary foreground image
            foreground.at<uchar>(i, j) = 255 *
                !pixelCodeBook[i*cols+j].compare_color(frame.at<Vec3b>(i, j));
        }
    }
    return foreground;
}
```

Parameters:

frame - It is Mat object that indicated the frame which the foreground to be detected.

This method to get the foreground mask of the frame provided as a parameter after the background model is learned. The method returns a binary image where white color (pixel value 255) represent the foreground and black color (pixel value 0) represent the background. The function first creates an empty binary image, then loop through each pixel and compare current frame pixel color value against the CodeBook.

CodeBook

The class CodeBook is used to create objects that able to store the code book of a single pixel. Mainly use by BackgroundModel. It consists of 3 constants, 4 variables and 2 functions which are private as well as 1 constructor and 3 methods which are public.

```
class CodeBook{
  private:
    static const short sbin = BIN_SIZE;
    static const short nbin = 256/sbin;
    static const short tbin = nbin*nbin*nbin;

    short cur;
    bool isSorted;
    vector<QuantizedBin> colorBin;
    // colorBinIdx track the where the colorBin is after sorting
    vector<short> colorBinIdx;

    void binSort(){
        // sort reverse order
        sort(colorBin.rbegin(), colorBin.rend());

        for(int i = 0; i < colorBin.size(); i++){
            colorBinIdx[colorToIdx(colorBin[i].color)] = i;
        }
        isSorted = true;
    }
    int colorToIdx(Vec3b color){
        int b = color[0]/sbin, g = color[1]/sbin, r = color[2]/sbin;
        return nbin*nbin*b + nbin*g + r;
    }
```

The class consists of 3 static constant integer which store the size of the bin, number of bin and total number of bin. Also consist of vector of QuantizedBin object which store all the code word in this code book. Other variable and function are main helper of ther class.

CodeBook::CodeBook()

```
public:
  CodeBook(){
      cur = 0;

      colorBin.resize(tbin);
      colorBinIdx.resize(tbin);
      for(int i = 0; i < tbin; i++){
          int bin = i;
          int r = bin%nbin; bin /= nbin;
          int g = bin%nbin; bin /= nbin;
          int b = bin%nbin;

          colorBin[i] = QuantizedBin(Vec3b(b*sbin+sbin/2, g*sbin+sbin/2, r*sbin+sbin/2));
          colorBinIdx[i] = i;
      }
      isSorted = true;
  }
```

Parameters:

None

It is an empty constructor with no parameter which used to construct, initialize the object and its variables.

void CodeBook::update(Vec3b color)

```
void update(Vec3b color){
    isSorted = false;
    int idx = colorBinIdx[colorToIdx(color)];

    // check colorBin[idx].color ==  color
    colorBin[idx].quantity++;
    colorBin[idx].lambda = max(colorBin[idx].lambda, cur-colorBin[idx].frame_index.back());
    colorBin[idx].frame_index.push_back(cur);
    cur++;
}
```

Parameters:

color – It is a Veb3b type variable that indicated the new frame's pixel color in order to be updated in this code book.

It is used by the BackgroundModel to update the pixel color in the CodeBook and return void after executed. It increases the corresponding quantized color bin frequency by 1. And also compute the lambda and last of the color bin.

Vec3b CodeBook::get_color()

```
Vec3b get_color(){
    if(!isSorted)
        binSort();
    return colorBin[0].color;
}
```

Parameters:

None

It is used by the BackgroundModel to obtain and return the dominant color of current CodeBook. It first sort the quantized color bin by frequency, then return the color that has highest frequency.

bool CodeBook::compare_color(Vec3b color)

```
bool compare_color(Vec3b color){
    Vec3b dominant = this->get_color();

    const int thres = 40;
    bool match = true;
    for(int i = 0; i < 3; i++){
        if(abs(dominant[i]-color[i]) > thres)
            match = false;
    }
    if(match)
        return true;

    if(!isSorted)
        binSort();

    // variable !!!
    for(int i = 1; i < 10; i++){
        colorBin[i].lambda = max(colorBin[i].lambda, cur-colorBin[i].frame_index.back());
        if(colorBin[i].lambda > 300)
            continue;
        // a dynamic backgruond code word
        bool match = true;
        for(int j = 0; j < 3; j++){
            if(abs(colorBin[i].color[j]-color[j]) > thres)
                match = false;
        }
        if(match)
            return true;
    }
    return false;
}
```

Parameters:

color - It is a Veb3b type variable that indicated the color value that used to compare with this CodeBook.

It used to compare the color against the current CodeBook and returns true if the color belongs to the background color and false otherwise. This is achieved by obtaining the absolute different of each 3 color channel value between the color and input color. If any of the difference exceed the threshold value, then they are different color and hence, is a foreground image. The comparing is done by first compare with the static background model, if it fails then it compares with the dynamic background model, if it fails again then it is a foreground pixel.

ForegroundObject

The structure is used to store the detected object. The object properties such as center, area, x, y, h, w, and velocity are stored. It consists of a constructor to store initialize the attribute and a method which draw the object information on the image for visualization purpose.

```
struct ForegroundObject{
    Point center;
    int area, x, y, h, w;
    double velocity_x, velocity_y;
    vector<Point> path;

    ForegroundObject(Point center, int area, int x, int y, int h, int w){
        velocity_x = velocity_y = 0;
        this->center = center;
        this->area = area;
        this->x = x;
        this->y = y;
        this->h = h;
        this->w = w;
    }

    void draw(Mat &img){
        rectangle(img, Point(x, y), Point(x+w, y+h), Scalar(0, 0, 255), 1);
        circle(img, center, 2, Scalar(255, 0, 0), -1);
        arrowedLine(img, center, Point(center.x + velocity_x*7, center.y + velocity_y*7),
            Scalar(255, 255, 0), 2, CV_AA, 0, 0.2);

        // draw path
        for(int i = 0; i < (int)path.size()-1; i++){
            line(img, path[i], path[i+1], Scalar(255, 255, 0), 1, CV_AA);
        }
    }
```

void ForegroundObject::draw(Mat &img)

Parameters:

img - the image which the object information to be draw on

It draws the object bounding box, centroid and trajectory estimation on the img.

TrajectoryModel

```
class TrajectoryModel{
  private:
    vector<ForegroundObject> objects;

  public:
    void computeVelocity(vector<ForegroundObject> &curObjects){
        // for each curObject, try to match with a object
        for(int i = 0; i < curObjects.size(); i++){
            for(int j = 0; j < objects.size(); j++){
                if( abs(objects[j].center.x + objects[j].velocity_x - curObjects[i].center.x) < 9 &&
                    abs(objects[j].center.y + objects[j].velocity_y - curObjects[i].center.y) < 9){
                    // matched expected center

                    double v_x = curObjects[i].center.x - objects[j].center.x;
                    double v_y = curObjects[i].center.y - objects[j].center.y;

                    // check direction, angle should be less than 120 degree
                    double angle_rad = atan2(v_x, v_y)-atan2(objects[j].velocity_x, objects[j].velocity_y);
                    if(abs(angle_rad) > 3*PI/4)
                        continue;

                    // update velocity
                    curObjects[i].velocity_x = 0.8*objects[j].velocity_x + 0.2*v_x;
                    curObjects[i].velocity_y = 0.8*objects[j].velocity_y + 0.2*v_y;
                    // update path
                    curObjects[i].path = objects[j].path;
                    curObjects[i].path.push_back(curObjects[i].center);
                    break;
                }
            }
        }

        objects = curObjects;
    }
};
```

The trajectory model mainly consists of a method which can be used to compute the velocity of current objects. This is done by, associate the object from the previous frame to current frame by using the heuristic: “If the centroid of both object is close to each other across the frame, then they are the same object”. An association of the object indicates that both are the same object.

The velocity can then be computed by the following formula, velocity = current position - previous position. However, after some testing, we decided to introduce some heuristic for a more accurate velocity estimation. By using the fact that, previous object also carries a velocity, we must take account into previous object velocity too when determining the current object velocity. Hence, we use the following formula for determining current object velocity.

$$\text{appearance velocity} = \text{current position} - \text{previous position}$$

$$\text{current velocity} = \alpha \times \text{appearance velocity} + (1 - \alpha) \times \text{previous velocity}$$

where α is the learning rate (we set it to 0.2 in our implementation).

void TrajectoryModel::computeVelocity(vector<ForegroundObject> &curObjects)

Parameters:

curObjects - all the detected object in the current frame

It computes the velocity of all the object in curObjects using the method described above.

```
short max(short a, int b){
    return max(int(a), b);
}

short max(int a, short b){
    return max(b, a);
}
```

The 2 functions are overloaded functions which having the same name and return type but different parameters.

short max(short a, int b)

Parameters:

a – a short variable

b – an integer variable

It compared and return the maximum value between a and b.

short max(int a, short b)

Parameters:

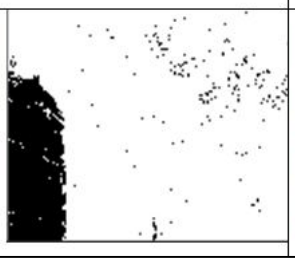
a – an integer variable

b – a short variable

It compared and return the maximum value between a and b.

4.3 System code testing and verification

By setting the solution configuration to the release mode, we able to get a shorter computation time for the code testing and verification purpose. So we go for the full sample data instead of using some simple data or just considering about the Region of Interest (ROI) for the verification purpose.

Original Video	Foreground Mask	Morphology Transformation	Results with boxes and directions
			

Chapter 5: System Analysis and Performance Evaluation

5.1 Different application environments or scenarios

5.1.1 Parking.mp4

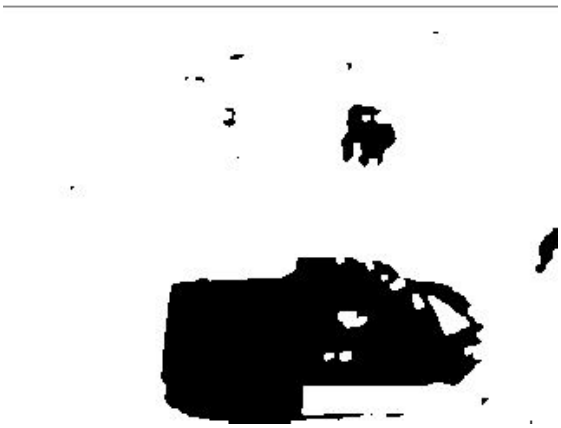
Video information and progresses:

```
---------------------- Video Information ------------------
Video Name: parking.mp4
Video Frame: 1498
Video FPS: 8
Video Length: 3 min 7 sec
Video Dimension: 320 x 240

---------------------- Training Phase ---------------------

  Training Progress    Processing Speed          ETA
         99 %              25.58 FPS         0 min 0 sec

Training Phase Completed
Total Elapse Time: 0 min 56 sec

Press Any Key to Continue


----------------------- Testing Phase ---------------------

   Testing Progress    Processing Speed          ETA
         99 %              15.27 FPS         0 min 0 sec

Testing Phase Completed
Total Elapse Time: 0 min 51 sec

Press any key to continue . . .
```

Learning Progress	Learned Background
	

Testing Results:

Learned Background


Original Video	Foreground Mask
	
Morphology Transformation	**Results with boxes and directions**
	

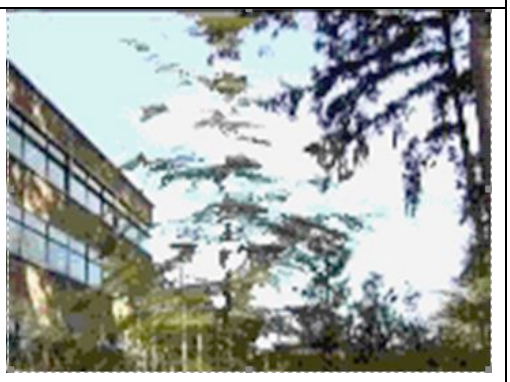
Original Video	Foreground Mask
	
Morphology Transformation	**Results with boxes and directions**
	

Our system worked fine on the “parking.mp4” due to its static background. By using the morphology, some noises are eliminated. The final results illustrated the detection of the moving objects within the foreground such as the people and vehicles by using the red-bounded box. The cyan arrow of the image indicated the predicted direction of the moving objects.

5.1.2 WavingTrees.avi

Video information and progresses:

```
---------------------- Video Information ----------------------
Video Name: WavingTrees.avi
Video Frame: 288
Video FPS: 30
Video Length: 0 min 9 sec
Video Dimension: 160 x 120

----------------------- Training Phase -----------------------

  Training Progress    Processing Speed         ETA
         97 %              64.14 FPS         0 min 0 sec

Training Phase Completed
Total Elapse Time: 0 min 5 sec

Press Any Key to Continue


----------------------- Testing Phase ------------------------

   Testing Progress    Processing Speed         ETA
         97 %              63.89 FPS         0 min 0 sec

Testing Phase Completed
Total Elapse Time: 0 min 4 sec

Press any key to continue . . .
```

Learning Progress	Learned Background
	

Testing Results:

Learned Background


Original Video	Foreground Mask
	
Morphology Transformation	**Results with boxes and directions**
	

Original Video	Foreground Mask
	
Morphology Transformation	**Results with boxes and directions**
	

The “WavingTrees.avi” consist of the dynamic background that having a tree waving throughout the video which increase the difficulties of background subtraction but with the lambda setting, our system managed to detect the moving object and predict its direction without any issue as shown as the table above.

5.1.3 fountain01.avi

Video information and progresses:

```
---------------------- Video Information ----------------------
Video Name: fountain01.avi
Video Frame: 1184
Video FPS: 30
Video Length: 0 min 39 sec
Video Dimension: 432 x 288

----------------------- Training Phase ------------------------

  Training Progress    Processing Speed          ETA
         99 %               17.37 FPS        0 min 0 sec

Training Phase Completed
Total Elapse Time: 1 min 4 sec

Press Any Key to Continue


------------------------ Testing Phase ------------------------

   Testing Progress    Processing Speed          ETA
         99 %               21.34 FPS        0 min 0 sec

Testing Phase Completed
Total Elapse Time: 1 min 3 sec

Press any key to continue . . .
```

Learning Progress	Learned Background
	

Testing Results:

Learned Background


Original Video	Foreground Mask
	
Morphology Transformation	**Results with boxes and directions**
	

Original Video	Foreground Mask
	
Morphology Transformation	**Results with boxes and directions**
	

The “fountain01.avi” also consist of the dynamic background that having a fountain that increase the difficulties of background subtraction due to its water movement, our system managed to detect the moving object such as the vehicle behind the fountain and predict its direction but it is still affected by its dynamic background.

5.1.4 contrysideTraffic.mp4

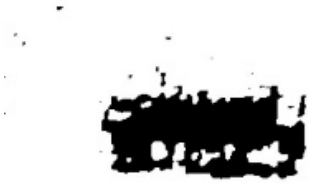
Video information and progresses:

```
---------------------- Video Information ------------------
Video Name: contrysideTraffic.mp4
Video Frame: 1883
Video FPS: 29.9818
Video Length: 1 min 2 sec
Video Dimension: 480 x 270

----------------------- Training Phase --------------------

  Training Progress    Processing Speed          ETA
         99 %              17.47 FPS         0 min 0 sec

Training Phase Completed
Total Elapse Time: 2 min 0 sec

Press Any Key to Continue


------------------------ Testing Phase --------------------

   Testing Progress    Processing Speed          ETA
         99 %              24.67 FPS         0 min 0 sec

Testing Phase Completed
Total Elapse Time: 1 min 42 sec

Press any key to continue . . .
```

Learning Progress	Learned Background
	

Testing Results:

Learned Background


Original Video	Foreground Mask
	
Morphology Transformation	**Results with boxes and directions**
	

Original Video	Foreground Mask
	
Morphology Transformation	**Results with boxes and directions**
	

The “contrysideTraffic.mp4” also having a dynamic background which are the waving trees that increase the difficulties of background subtraction due to its wind movement, our system managed to detect the moving objects such as the vehicle and people as well as predict their directions. However, our proposed system is using the RGB color representation which unable to perform well enough within the video that contained the lighting changing and shadows.

5.1.5 WetSnow.mp4

Video information and progresses:

```
---------------------- Video Information -----------------
Video Name: WetSnow.mp4
Video Frame: 1694
Video FPS: 30
Video Length: 0 min 56 sec
Video Dimension: 720 x 540

----------------------- Training Phase -------------------

  Training Progress    Processing Speed         ETA
         99 %               5.87 FPS         0 min 0 sec

Training Phase Completed
Total Elapse Time: 4 min 31 sec

Press Any Key to Continue


------------------------ Testing Phase -------------------

   Testing Progress    Processing Speed         ETA
         99 %               9.39 FPS         0 min 0 sec

Testing Phase Completed
Total Elapse Time: 3 min 49 sec

Press any key to continue . . .
```

Learning Progress	Learned Background
	

Testing Results:

Learned Background


Original Video	Foreground Mask
	
Morphology Transformation	**Results with boxes and directions**
	

Original Video	Foreground Mask
	
Morphology Transformation	**Results with boxes and directions**
	

The “WetSnow.mp4” converted from the multiple “WetSnow” frames which having a bigger and longer sequence compared to other datasets. It consists of a lot of challenging components within the video such as the snowing scene and water drops which form the dynamic background and noises as well as the sleeping moving objects which is a car that remain static for a certain moment but our system managed to detect the sleeping moving object and predict its direction even it is still slightly affected by its dynamic background.

5.1.6 CampusStreet.avi

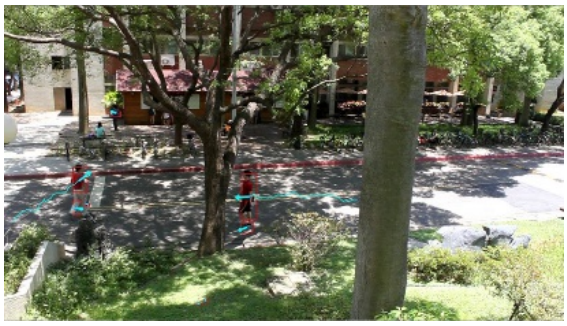
Video information and progresses:

```
---------- Video Information ----------
Video Name: CampusStreet.avi
Video Frame: 535
Video FPS: 20
Video Length: 0 min 26 sec
Video Dimension: 856 x 480

---------- Training Phase ----------

  Training Progress    Processing Speed          ETA
         97 %              11.07 FPS        0 min 1 sec

Training Phase Completed
Total Elapse Time: 1 min 42 sec

Press Any Key to Continue


---------- Testing Phase ----------

   Testing Progress    Processing Speed          ETA
         97 %              9.72 FPS         0 min 1 sec

Testing Phase Completed
Total Elapse Time: 1 min 53 sec

Press any key to continue . . .
```

Learning Progress	Learned Background
	

Testing Results:

Learned Background


Original Video	Foreground Mask
	
Morphology Transformation	**Results with boxes and directions**
	

Original Video	Foreground Mask
	
Morphology Transformation	**Results with boxes and directions**
	

The “CampusStreet.avi” also consist of the lighting changes and shadows. During the initial stage of the testing phase, our system unable to eliminate the noises effectively. However, the performance increased as more noises are being eliminated after some periods.

5.1.7 Implementation Issues and Challenges for Advanced System

With Frame Index implementation	Without Frame Index implementation
---------- Video Information ---------- Video Name: WetSnow.mp4 Video Frame: 1694 Video FPS: 30 Video Length: 0 min 56 sec Video Dimension: 720 x 540 ---------- Training Phase ---------- Training Progress Processing Speed ETA 99 % 1.68 FPS 0 min 2 sec Training Phase Completed Total Elapse Time: 37 min 34 sec Press Any Key to Continue ---------- Testing Phase ---------- Testing Progress Processing Speed ETA 99 % 7.70 FPS 0 min 0 sec Testing Phase Completed Total Elapse Time: 7 min 5 sec	---------- Video Information ---------- Video Name: WetSnow.mp4 Video Frame: 1694 Video FPS: 30 Video Length: 0 min 56 sec Video Dimension: 720 x 540 ---------- Training Phase ---------- Training Progress Processing Speed ETA 99 % 5.87 FPS 0 min 0 sec Training Phase Completed Total Elapse Time: 4 min 31 sec Press Any Key to Continue ---------- Testing Phase ---------- Testing Progress Processing Speed ETA 99 % 9.39 FPS 0 min 0 sec Testing Phase Completed Total Elapse Time: 3 min 49 sec

With Frame Index implementation	Without Frame Index implementation
---------- Video Information ---------- Video Name: CampusStreet.avi Video Frame: 535 Video FPS: 20 Video Length: 0 min 26 sec Video Dimension: 856 x 480 ---------- Training Phase ---------- Training Progress Processing Speed ETA 97 % 0.30 FPS 0 min 50 sec Training Phase Completed Total Elapse Time: 30 min 41 sec Press Any Key to Continue ---------- Testing Phase ---------- Testing Progress Processing Speed ETA 97 % 4.74 FPS 0 min 3 sec Testing Phase Completed Total Elapse Time: 2 min 49 sec	---------- Video Information ---------- Video Name: CampusStreet.avi Video Frame: 535 Video FPS: 20 Video Length: 0 min 26 sec Video Dimension: 856 x 480 ---------- Training Phase ---------- Training Progress Processing Speed ETA 97 % 11.07 FPS 0 min 1 sec Training Phase Completed Total Elapse Time: 1 min 42 sec Press Any Key to Continue ---------- Testing Phase ---------- Testing Progress Processing Speed ETA 97 % 9.72 FPS 0 min 1 sec Testing Phase Completed Total Elapse Time: 1 min 53 sec

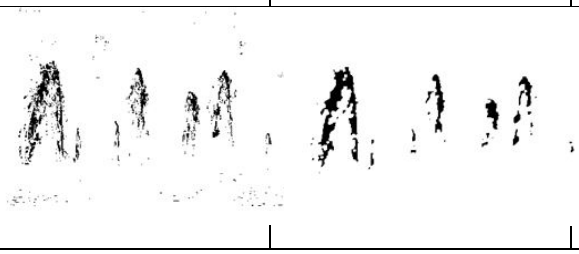
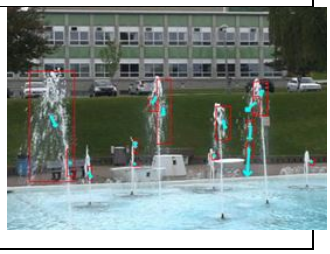
From the tables above, we can see that the coding with the frame index implementation required more time for the training phase. The memory consumption has also increased significantly with the implementation.

The “WetSnow.mp4” and “CampusStreet.avi” videos have a more obvious delays and memory consumptions during the training phase compared to other videos due their sizes.

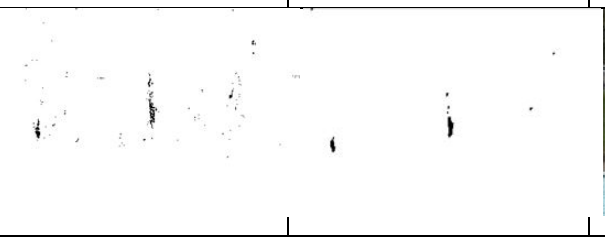
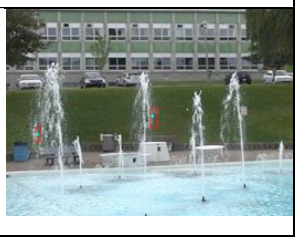
5.2 Differences of the output results under different system parameter setting

By the changing the setting of the lambda

(lambda > 50)

Original Video	Foreground Mask	Morphology Transformation	Results with boxes and directions
			

(lambda > 300)

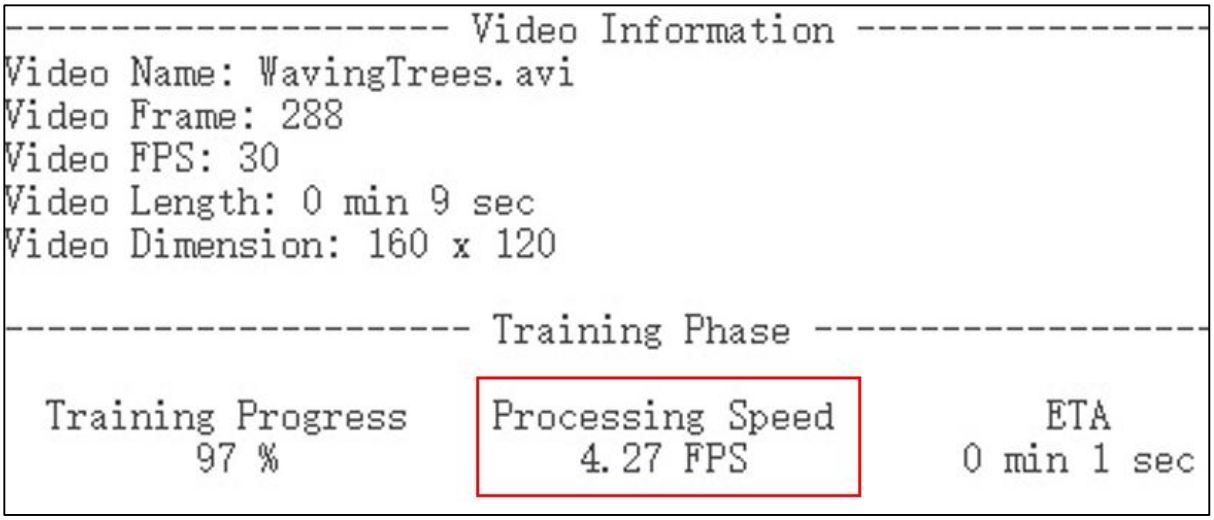
Original Video	Foreground Mask	Morphology Transformation	Results with boxes and directions
			

By increase the size of the maximum negative run length, lambda (from 50 to 300), we can observe that the water moments in the foreground mask and morphed foreground mask (noises) has been significantly decreased by treated the water splits as the background instead of the moving objects of the foreground.

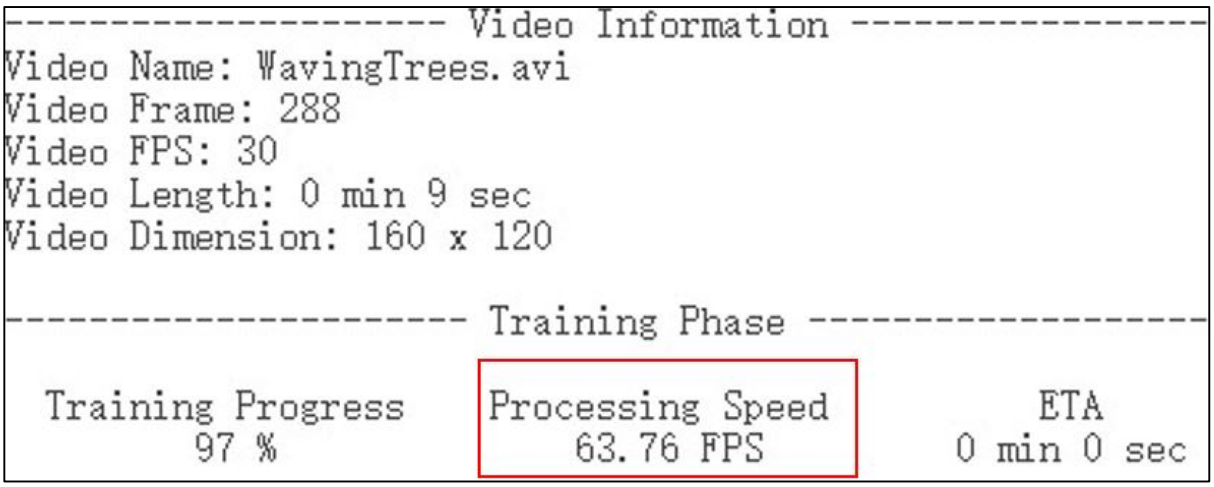
5.3 Show the differences in the output results when using different system configurations

Solution Configuration

(Debug Mode)



(Release Mode)



Debug and release mode are 2 different types of the solution configurations where the debug consist of the debug information for debugging purpose while the release usually be optimized without the completed debugging information.

As the results above shown the processing speed in term of frame per second (FPS) of the 2 different modes, as the debug mode is more time consuming due to its low processing of 4.27 FPS while the release mode having the higher processing speed of 63.76 FPS which is around 16 times higher than the debug mode and result in faster computation speed.

5.4 Strengths and weaknesses the system implemented

Strengths

- Able extract the background model from the given input video
- Able to perform moving objects detection and tracking of the foreground
- Able to extract and detect of the sleeping moving objects
- Able to predict and track the moving direction of the objects by using the motion direction and trajectories
- Able to reduce the noises by applying the morphology techniques towards the foreground mask

Weaknesses

- Unable to handle the lighting changes and shadow
- Lack of the uses of advanced color representations such as the HSV and LAB

Chapter 6: Conclusion

6.1 Summary on the research problem that have been considered and solved

(a) Multiple layer codebooks

- The system will contain multiple layers of the codebooks to deal with different type of the background or foreground such as the static and dynamic backgrounds.

(b) Codebooks to consider the problem of sleeping moving object

- The codebook should be able to detect and track the foreground moving objects which remain as temporarily static and started to move afterward.

(c) Motion direction and motion trajectory of the moving objects

- The system should be able to track the motion direction and motion trajectory of the moving objects within the foreground of the input videos

6.2 Highlight on any novelties and contributions the project has achieved

- Our proposed system performs the background subtraction by using the dominant colors instead of the MOG and KDE approaches which used by the current existing system
- Our proposed system able to outperform the MOG approach which consist of a lot of the noises just like the KDE approach did.

6.3 Future works

- Applied HSV or LAB instead of using the RGB
  - To ensure the system having a better immunity to the effect of dynamic background factors such as the shadow and reflection.
- Included other kind of datasets
  - To further evaluate the system with different variety of the datasets
- Deeper comparison against other approaches
  - Detail comparison with other existing approaches and methods will be in term of their efficiency and performance.

Biography / References

C. Stauffer and W. Grimson. "Adaptive background mixture models for real-time tracking", 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition.

Kyungnam Kim, Thanarat H. Chalidabhongse, David Harwood, and Larry Davis, "Real-time foreground-background segmentation using codebook model", Computer vision lab, Department of Computer Science, University of Maryland, College Park, MD 20742, USA, 2005.

Jeisung Lee and Mignon Park, (2012). *An Adaptive Background Subtraction Method Based on Kernel Density Estimation*.

Appendix: The Source Code of the Program

```
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>

using namespace cv;
using namespace std;

// BIN_SIZE should be power of 2
#define BIN_SIZE 32;
const double PI = acos(-1);

short max(short a, int b){
   return max(int(a), b);
}

short max(int a, short b){
   return max(b, a);
}

struct QuantizedBin{
   short quantity, lambda;
   Vec3b color;
   vector<short> frame_index;

   QuantizedBin(){}
   QuantizedBin(Vec3b color){
      lambda = 0;
      frame_index.reserve(1);
      frame_index.push_back(-1);
      quantity = 0;
      this->color = color;
   }
```

```
    bool operator< (const QuantizedBin x) const {
        return this->quantity < x.quantity;
    }
};

class CodeBook{
  private:
    static const short sbin = BIN_SIZE;
    static const short nbin = 256/sbin;
    static const short tbin = nbin*nbin*nbin;

    short cur;
    bool isSorted;
    vector<QuantizedBin> colorBin;
    // colorBinIdx track the where the colorBin is after sorting
    vector<short> colorBinIdx;

    void binSort(){
        // sort reverse order
        sort(colorBin.rbegin(), colorBin.rend());

        for(int i = 0; i < colorBin.size(); i++){
            colorBinIdx[colorToIdx(colorBin[i].color)] = i;
        }
        isSorted = true;
    }
    int colorToIdx(Vec3b color){
        int b = color[0]/sbin, g = color[1]/sbin, r = color[2]/sbin;
        return nbin*nbin*b + nbin*g + r;
    }
```

```
public:
  CodeBook(){
    cur = 0;

    colorBin.resize(tbin);
    colorBinIdx.resize(tbin);
    for(int i = 0; i < tbin; i++){
      int bin = i;
      int r = bin%nbin; bin /= nbin;
      int g = bin%nbin; bin /= nbin;
      int b = bin%nbin;

      colorBin[i] = QuantizedBin(Vec3b(b*sbin+sbin/2, g*sbin+sbin/2, r*sbin+sbin/2));
      colorBinIdx[i] = i;
    }
    isSorted = true;
  }

  void update(Vec3b color){
    isSorted = false;
    int idx = colorBinIdx[colorToIdx(color)];

    // check colorBin[idx].color ==  color
    colorBin[idx].quantity++;
    colorBin[idx].lambda = max(colorBin[idx].lambda, cur-
colorBin[idx].frame_index.back());
    colorBin[idx].frame_index.push_back(cur);
    cur++;
  }

  Vec3b get_color(){
    if(!isSorted)
      binSort();
```

```
    return colorBin[0].color;
}

bool compare_color(Vec3b color){
    Vec3b dominant = this->get_color();

    const int thres = 40;
    bool match = true;
    for(int i = 0; i < 3; i++){
        if(abs(dominant[i]-color[i]) > thres)
            match = false;
    }
    if(match)
        return true;

    if(!isSorted)
        binSort();

    // variable !!!
    for(int i = 1; i < 10; i++){
        colorBin[i].lambda = max(colorBin[i].lambda, cur-colorBin[i].frame_index.back());
        if(colorBin[i].lambda > 300)
            continue;
        // a dynamic backgruond code word
        bool match = true;
        for(int j = 0; j < 3; j++){
            if(abs(colorBin[i].color[j]-color[j]) > thres)
                match = false;
        }
        if(match)
            return true;
    }
```

```
        return false;
    }
};

class BackgroundModel{
  private:
    // every pixel has a CodeBook
    int rows, cols;
    vector<CodeBook> pixelCodeBook;

  public:
    BackgroundModel(int rows, int cols){
        this->rows = rows;
        this->cols = cols;

        pixelCodeBook.assign(rows*cols, CodeBook());
    }

    void learn(Mat frame){
        for(int i = 0; i < rows; i++){
            for(int j = 0; j < cols; j++){
                // mat.at<>(r, c);
                pixelCodeBook[i*cols+j].update(frame.at<Vec3b>(i, j));
            }
        }
    }

    Mat get_background_model(){
        Mat background(rows, cols, CV_8UC3);
        for(int i = 0; i < rows; i++){
            for(int j = 0; j < cols; j++){
                // mat.at<>(r, c);
```

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

Mat get_foreground_mask(Mat frame){
    Mat foreground(rows, cols, CV_8UC1);
    for(int i = 0; i < rows; i++){
        for(int j = 0; j < cols; j++){
            // mat.at<>(r, c);
            // compare color return true(1) false(0), multiple 255 to get binary foreground image
            foreground.at<uchar>(i, j) = 255 *
                !pixelCodeBook[i*cols+j].compare_color(frame.at<Vec3b>(i, j));
        }
    }
    return foreground;
}
};

struct ForegroundObject{
    Point center;
    int area, x, y, h, w;
    double velocity_x, velocity_y;
    vector<Point> path;

    ForegroundObject(Point center, int area, int x, int y, int h, int w){
        velocity_x = velocity_y = 0;
        this->center = center;
        this->area = area;
        this->x = x;
        this->y = y;
```

```
        this->h = h;
        this->w = w;
    }

    void draw(Mat &img){
        rectangle(img, Point(x, y), Point(x+w, y+h), Scalar(0, 0, 255), 1);
        circle(img, center, 2, Scalar(255, 0, 0), -1);
        arrowedLine(img, center, Point(center.x + velocity_x*7, center.y + velocity_y*7),
            Scalar(255, 255, 0), 2, CV_AA, 0, 0.2);

        // draw path
        for(int i = 0; i < (int)path.size()-1; i++){
            line(img, path[i], path[i+1], Scalar(255, 255, 0), 1, CV_AA);
        }
    }
};

class TrajectoryModel{
  private:
    vector<ForegroundObject> objects;

  public:
    void computeVelocity(vector<ForegroundObject> &curObjects){
        // for each curObject, try to match with a object
        for(int i = 0; i < curObjects.size(); i++){
            for(int j = 0; j < objects.size(); j++){
                if( abs(objects[j].center.x + objects[j].velocity_x - curObjects[i].center.x) < 9 &&
                    abs(objects[j].center.y + objects[j].velocity_y - curObjects[i].center.y) < 9){
                    // matched expected center

                    double v_x = curObjects[i].center.x - objects[j].center.x;
                    double v_y = curObjects[i].center.y - objects[j].center.y;
```

```
            // check direction, angle should be less than 120 degree
            double angle_rad = atan2(v_x, v_y)-atan2(objects[j].velocity_x,
objects[j].velocity_y);
            if(abs(angle_rad) > 3*PI/4)
               continue;

            // update velocity
            curObjects[i].velocity_x = 0.8*objects[j].velocity_x + 0.2*v_x;
            curObjects[i].velocity_y = 0.8*objects[j].velocity_y + 0.2*v_y;
            // update path
            curObjects[i].path = objects[j].path;
            curObjects[i].path.push_back(curObjects[i].center);
            break;
          }
        }
      }

      objects = curObjects;
   }
};

void training(VideoCapture &vid, BackgroundModel &background){
   cout << "---------------------- Training Phase ------------------------" << endl << endl;
   cout << "  Training Progress    Processing Speed         ETA" << endl;

   long long t0 = cv::getTickCount(), t1;
   int total_frame = vid.get(CV_CAP_PROP_FRAME_COUNT);
   double fps = vid.get(CV_CAP_PROP_FPS);

   int f = 0;
   while(vid.grab()){
      Mat frame;
```

```
        vid.retrieve(frame);

        background.learn(frame);
        imshow("Learning Progress", frame);
        waitKey(1);
        f++;

        if(f%10 == 0){
            double cur_fps = 10.0/(cv::getTickCount()-t1)*cv::getTickFrequency();
            int eta = (total_frame-f)/cur_fps;
            printf("\r         %d %%  \t\t   %0.2lf FPS \t     %d min %d sec         ",
                f*100/total_frame, cur_fps, eta/60, eta%60);
            fflush(stdout);
            t1 = cv::getTickCount();
        }
    }

    vid.release();
    imshow("Learned Background", background.get_background_model());

    cout << endl << endl;
    cout << "Training Phase Completed\nTotal Elapse Time: ";
    int elapsetime = (int)((cv::getTickCount()-t0)/cv::getTickFrequency());
    printf("%d min %d sec \n\n", elapsetime/60, elapsetime%60);

    cout << "Press Any Key to Continue" << endl;
    waitKey(0);
}

void testing(VideoCapture &vid, BackgroundModel &background){
    cout << "---------------------- Testing Phase ------------------------" << endl << endl;
    cout << "    Testing Progress    Processing Speed          ETA" << endl;
```

```
int total_frame = vid.get(CV_CAP_PROP_FRAME_COUNT);
double fps = vid.get(CV_CAP_PROP_FPS);
long long t0 = cv::getTickCount(), t1;
TrajectoryModel trajectoryModel;
int f = 0;
while(vid.grab()){
    Mat frame;
    vid.retrieve(frame);
    imshow("Original Video", frame);

    Mat foregroundMask = background.get_foreground_mask(frame);
    imshow("Forground Mask", 255-foregroundMask);

    Mat labels, stats, centroids;
    Mat strucElement = getStructuringElement(MORPH_ELLIPSE, Size(2, 2));
    morphologyEx(foregroundMask, foregroundMask, MORPH_CLOSE, strucElement);
    morphologyEx(foregroundMask, foregroundMask, MORPH_OPEN, strucElement);
    morphologyEx(foregroundMask, foregroundMask, MORPH_OPEN, strucElement);
    strucElement = getStructuringElement(MORPH_ELLIPSE, Size(4, 4));
    morphologyEx(foregroundMask, foregroundMask, MORPH_CLOSE, strucElement);
    imshow("Morphology Transform", 255-foregroundMask);
    int nLabels = connectedComponentsWithStats(foregroundMask, labels, stats, centroids);

    vector<ForegroundObject> objects;
    for(int i = 1; i < nLabels; i++){
        if(stats.at<int>(i, CC_STAT_AREA) < 20)
            continue;

        int x = stats.at<int>(i, CC_STAT_LEFT);
        int y = stats.at<int>(i, CC_STAT_TOP);
        int w = stats.at<int>(i, CC_STAT_WIDTH);
```

```
            int h = stats.at<int>(i, CC_STAT_HEIGHT);

            ForegroundObject curObject(Point(centroids.at<double>(i, 0), centroids.at<double>(i,
1)),
                stats.at<int>(i, CC_STAT_AREA), x, y, h, w);

            objects.push_back(curObject);
        }

        trajectoryModel.computeVelocity(objects);

        for(int i = 0; i < objects.size(); i++)
            objects[i].draw(frame);

        imshow("Original Video with Bounding Box", frame);

        double sec = (cv::getTickCount()-t0)/cv::getTickFrequency();
        if(sec < 1.0/fps)
            waitKey(1000.0/fps - sec*1000 + 1);
        else
            waitKey(1);

        f++;
        if(f%10 == 0){
            double cur_fps = 10.0/(cv::getTickCount()-t1)*cv::getTickFrequency();
            int eta = (total_frame-f)/cur_fps;
            printf("\r         %d %%  \t\t   %0.2lf FPS \t     %d min %d sec          ",
                f*100/total_frame, cur_fps, eta/60, eta%60);
            fflush(stdout);
            t1 = cv::getTickCount();
        }
    }
```

```
    cout << endl << endl;
    cout << "Testing Phase Completed\nTotal Elapse Time: ";
    int elapsetime = (int)((cv::getTickCount()-t0)/cv::getTickFrequency());
    printf("%d min %d sec \n\n", elapsetime/60, elapsetime%60);

}

int main(int argc, char *argv[]){
    String filename = "CampusStreet.avi";
    if(argc > 1){
        filename = argv[1];
    }
    VideoCapture vid(filename);
    if(!vid.isOpened() || !vid.grab())
        return -1;

    int v_height = vid.get(CV_CAP_PROP_FRAME_HEIGHT);
    int v_width = vid.get(CV_CAP_PROP_FRAME_WIDTH);
    int total_frame = vid.get(CV_CAP_PROP_FRAME_COUNT);
    double fps = vid.get(CV_CAP_PROP_FPS);

    cout << endl;
    cout << "---------- Video Information ----------" << endl;
    cout << "Video Name: " << filename << endl;
    cout << "Video Frame: " << total_frame << endl;
    cout << "Video FPS: " << fps << endl;
    cout << "Video Length: "; printf("%d min %d sec\n", (int)(total_frame/fps/60), (int)(total_frame/fps)%60);
    cout << "Video Dimension: " << v_width << " x " << v_height << endl;
    cout << endl;

    BackgroundModel background(v_height, v_width);
```

```
    training(vid, background);
    cout << endl << endl;

    vid.open(filename);
    if(!vid.isOpened() || !vid.grab())
        return -1;

    testing(vid, background);

    return 0;
}
```