Video Object Avoidance Implementation on Embedded Platform

By

Yeong Ming Keat

16469

Dissertation submitted in partial fulfilment of
the requirements for the
Bachelor of Engineering (Hons)
(Electrical and Electronics)

JANUARY 2015

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Video Object Avoidance Implementation On Embedded Platform

by

Yeong Ming Keat
16469

Dissertation submitted in partial fulfillment of
the requirements for the
Bachelor of Engineering (Hons)
(Electrical and Electronics)

Approved by,

____
Mr Patrick Sebastian

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK
JAN 2015

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

YEONG MING KEAT

ABSTRACT

Motion detection is fundamental in various computer vision related applications. In this project, there are two motion detection techniques being studied, namely optical flow and motion templates. This is to detect the moving obstacles as well as to classify the direction of the moving obstacles. Optical flow is the computation to approximate the image motion, while motion templates use the motion-history-image (MHI) to keep track of the most recent movement with the timestamp. Besides, this project also covers the static object detection, where HSV color model classification technique is used to detect the static obstacles. This technique is based on filtration of color, which depending on the HSV values of the static objects. Both motion and static detection algorithms will be tested in Window Visual Studio 2010, before implementing them into the embedded platform, which is Raspberry Pi. Meanwhile, OpenCV is used as the computer vision library throughout the project. At the end of this project object, motion templates is selected as a more suitable motion detection techniques due to its extra information, which is the angle. The HSV technique can detect the static objects but limited to the calibrated color only.

ACKNOWLEDGEMENT

I would like to thanks my respectable supervisor, Mr Patrick Sebastian, for all his helps and guidance given to me in the process of completing my Final Year Project.

I had learnt a lot of priceless experience through the working on OpenCV, which is focusing on the development of the image and video algorithm to detect the static and moving objects.

As the saying goes, “No pain, go gain”. I feel grateful that at the end I am able to implement the algorithms into the Raspberry Pi and I really enjoy throughout the learning process in exploring the embedded system.

Once again, I appreciate the chance and trust given to me in completing this Final Year Project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

CHAPTER 1
INTRODUCTION

1.1 Background

Object avoidance is important for any moving objects such as vehicles, to avoid collision. In this project, object avoidance is implemented by using video and image processing approaches on embedded platform. The obstacles will be detected and captured by a camera. Besides, the obstacles detected will be analyzed and classified into 2 categories, which is either static or moving. The moving objects will be further classified into two categories, which are moving to the left and moving to the right. Throughout this project, the camera which is used to capture the image is static. The entire accuracy of the static and moving objects detection will be affected if the camera itself is also moving, which the problem is known as motion blur.

1.2 Problem Statement

Nowadays, most of the motion detection algorithms implemented on the vehicles only show the moving objects detected, but it does not acknowledge the drivers about the direction of the moving obstacles, which can be either crossing to the left or right on the road.

Thus, in this project the algorithms are developed to run the video and image processing which can analyze the obstacle whether it is static or moving and further classifying the moving obstacles into either moving to the left or right, within the vision of the camera. There are several situations for the condition of obstacles:

i. The obstacle is static.
ii. The obstacle is moving while the camera itself is static.
iii. The obstacle is moving while the camera is also moving.

Besides, the obstacle may be crossing the road, thus the design of the width of the camera's vision should be wide enough to capture the potential obstacle. For the camera, the light intensity of the captured environment can be classified into 3 categories:

i. Dark
ii. Dim
ii. Bright

The camera may face challenges when capturing the pictures of the real environment, under different light intensities.

1.3 Objectives and Scope of Study

1.3.1 Objectives

i. To implement real-time image and video processing algorithm on embedded computing platform.

ii To implement object avoidance algorithm which is capable of classifying the obstacles' behavior, whether they are moving or static.

1.3.2 Scope of Study

This project will cover the embedded system which is focusing on C language programming in Raspberry Pi's Raspbian Linux platform. Real-time image and video processing will be studied and applied throughout this project. Besides, OpenCV library will be used as a computer vision library while optical flow is used as the approach in performing the video processing. The image and video inputs will be acquired through the Raspberry Pi Camera, which can capture 5MP resolution images and deliver HD 1080p HD video recording [4]. The camera used will be fixed at a static location throughout the project.

CHAPTER 2
LITERATURE REVIEW

In this project, the video object avoidance system consists of 4 functional blocks, namely the image and video inputs, obstacle detection, classification of obstacles' behaviors and object avoidance. The Raspberry Pi Board is used as a computer while the HD Pi camera is used to capture the obstacle's images and videos.

One of the open sources for the operating system is used in this project, which is known as Raspbian. Raspbian is an Linux operating system which is customized for Raspberry Pi [5]. Before that, a user friendly NOOBS (New Out Of Box Software) had to be installed into the formatted SD card to create a bootable SD card [6]. The bootable SD card will enable the launching of the Raspberry Pi system when it is plugged into the Raspberry Pi board, while the power is turned on. The HDMI LED screen will display the Raspbian Linux Operating System.

On the other hand, this video object avoidance system on embedded platform will be designed by using OpenCV library. OpenCV is an open source computer vision library for C or C++ programming language and it is suitable to be used in the Raspbian Linux platform [7]. While the optical flow is a computer vision technique which comparing two successive image frames to detect the motion of the obstacles [8]. This is important in video analysis which can apply to the object tracking, motion detection or robot navigation [9]. The main reason of studying about the optical flow in this project is because of the capability of optical flow in detecting moving objects.

<u>Optical Flow</u>

The general principle of optical flow is to make use of the projected two dimensional (2D) image motion on an image plane, from the three dimensional (3D) motion of an object which is relative to the visual sensor. Through the sequencing of images in time order, the two dimensional image motion can be estimated as the discrete images displacement or instantaneous image velocity [3]. There are basically three main models developed for the motion and scene structure, which are used to perform estimation on the image motion, namely velocity, disparity and intensity.

Based on the 2D velocity field, by relating the parameters of motion and structure with the optical flow, 3D motion and the scene structure can be deduced. These parameters include the instantaneous displacement, the rate of rotation, the environment surface parameters and relative depth. While the disparity is used to get the 3D translational vectors, the matrices of rotation and surface attribute. Disparity can be established through the local correlation or image feature correspondence. Lastly, the intensity is used to compute the parameters of motion and structure [3].

However, the motion field and optical flow are not equivalent. For instance, a sphere which fixed at a point is rotating under a light source. We can observe that the shading part of sphere is constant due to the static light source, but the sphere is actually rotating. In this case, the optical flow is zero but motion field is not zero. If now the sphere is completely stationary, the light source is moving. Now, we can observe that the shading part of the sphere is moving due to the moving light source. In this case, the motion field is zero, but the optical flow is not zero. Nonetheless, most of the case, assumption is made that the motion field is similar to the optical flow [10].

There are two estimation method for optical flow analysis, namely Horn-Schunck method and Lucas-Kanade's method. These two gradient-based methods are performed through calculation of brightness gradient of image locally [9]. The Lucas-Kanade's method is a local method that applying a local constraint for each pixel [10]. First assumption needed is the intensity of an image pixel remains the same through the transition in time [10]. Second, the transition of image between frames is assumed to be small, about one pixel per frame of sub-pixel order. This is to enhance the accuracy of the optical flow as its detection sensitivity will drop significantly with the increasing in the image displacement [9]. The displacement of the image can be made smaller by reducing the resolution of the image [11]. In addition, pyramid based Lucas Kanade is used to extend the estimation of motion from corners to edges and inner region [12].

Meanwhile, Horn-Schunck's method is a global method which introduces a global smoothness constraint. The assumption made in Horn-Schunck's method is that the variation of optical flow is smooth or in other words, not too large. It is justified by claiming that the neighboring velocities which are similar to the same object surface should be identical [3]. This is a conventional standard for the whole image [10].

Thus, according to the first assumption, let's I(x,t) be the image intensity function,

$I(x,t) \approx I(x+\delta x, t+\delta t)$, Equation (1)

where,

δx is the local image region displacement at (x, t), after time δt.

Another important equation which is derived from Equation (1) is known as optical flow constraint equation [3], which defines the single local constraint on image motion, as shown below:

$\nabla I.\mathrm{v} + I_t = 0,$ Equation (2)

where, $\nabla I = (I_x, I_y)$ is the spatial intensity gradient, v = (u, v) is the velocity of image.

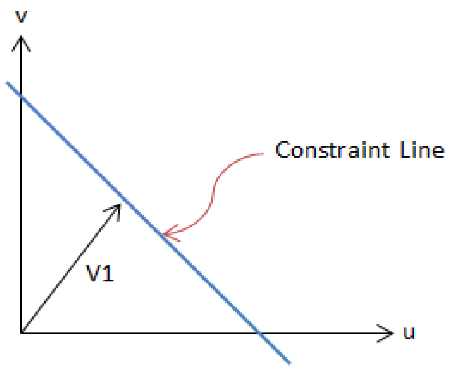


Figure (1)

Figure (1) shows the equation (2) defines a line in the velocity space. [3]

From Figure (1), V_L is the vector which is perpendicular to the constraint line, but it is not sufficient to obtain both v components. This means that $V_L$which is located in the direction of the local gradient of the image intensity function can be estimated. This situation is also known as aperture problem [3]. This means unless the motion component is at the intensity structure which is sufficient, the motion component cannot be fully estimated with the Equation (2), which is the optical flow constraint equation.

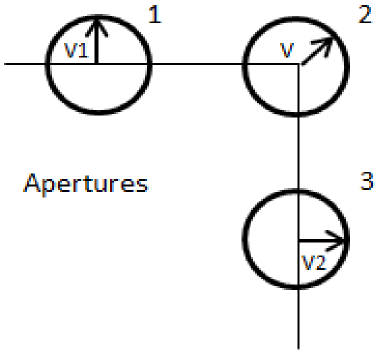


Figure (2)

Figure (2) In aperture 1 and 3, due to the lack of local structure, only the normal motions of the edges which can form the square can be estimated. In the aperture 2 which is located at the corner has sufficient local structure, thus both normal motions are visible [3].

The optical flow's problem or known as aperture problem when the optical flow is not able to estimate motion or flow, which is perpendicular to the image gradient. Optical flow can only measure its components which is in the direction of the intensity gradient, but not the components that are tangential to the intensity gradient [2]. In this condition, an assumption is made that the optical flow sees somewhere at the corner and the flow is smooth [11]. This is the situation that needs some constraints, which includes the global method, Horn-Schunck's method and local method, Locus-Kanade's method [10]. Besides, the spatial integration is required in the computing the optical flow due to the aperture problem as well as the noise at the local signals [12].

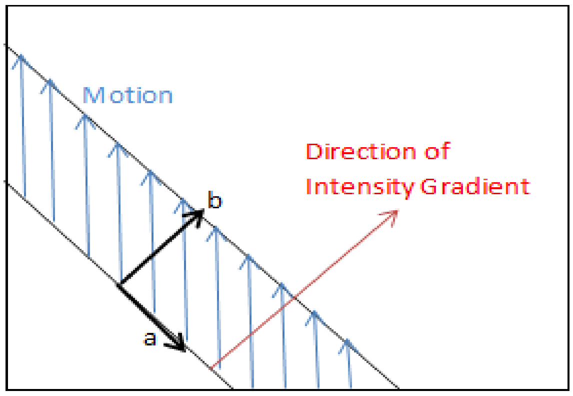


Figure (3)

Figure (3) above shows the aperture problem: We can only measure the component b which is in the direction of intensity gradient [2].

Another algorithm which is known as Shi-Tomasi Algorithm is being used as the corner detector. It is originated from Harris corner detector, but a slight modification has been made on the corner selection criteria. This makes Shi-Tomasi algorithm is better than the Harris corner detector algorithm. Shi-Tomasi proposed that only eigenvalues should be used to check whether the pixel is a corner and it is calculated by using the equation below:

$$R = min(\lambda_1, \lambda_2) \qquad \text{Equation (3)}$$

Equation (3): Shi-Tomasi Algorithm Equation [1]

where λ = eigenvalues,

Based on the equation above, if R is larger than the set value, the pixel can be marked as a corner [1]. All the corners which are below the set value will be rejected. Thus, Shi-Tomasi will only take the strongest corner. The overall objective of Shi—Tomasi algorithm is to choose the good features to track and enhance the tracking accuracy. A good feature will have big eigenvalues which implies two qualities, namely texture and corner. Lacking of texture and corner will lead to ambiguity in tracking and aperture problem respectively [13].

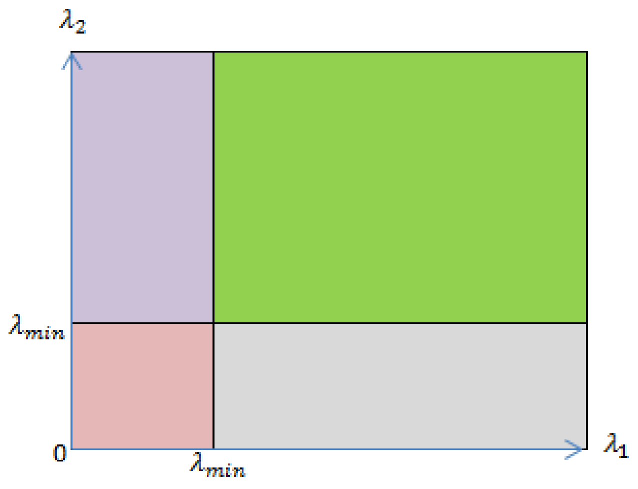


Figure (4)

Figure (4) above shows that a corner is found in the green region where both λ_1 *and* λ_2 are greater than a certain value, which is λ_{min}. [1].

From Figure (4), those eigenvalues, λ_1 and λ_2 with large positive values are indicated as the corner. For eigenvalues, λ_1 and λ_2 which have some positive values, but either one is less than the λ_{min}, for instance the purple and grey regions are considered as “edge” , while the eigenvalues, λ_1 and λ_2 which are both below λ_{min} , for instance the red region, has no features of interest or considered as “flat” area, in Harris corner detector [1].

The Lucas-Kanade and Shi-Tomasi methods are used in performing sparse optical flow, where only some pixels which have good features are analyzed. This causes the inaccuracy when measuring the motion. In this case, there is another algorithm that compute the optical flow for each pixel in the frame, which is known as dense optical flow or Farnaback algorithm [14]. Farneback algorithm is based on the two frame motion estimation based on polynomial expansion [14].

Besides, the obstacles detected need further analysis to find out whether they are stationary or dynamic. One of the techniques to classify the motion is though the statistic of optical flow orientation [15]. The prerequisite of this technique is to get the dense optical flow. Through the dense optical flow, the region of coherent flow will be grouped together in the RoI (Region of Interest). From these selected subset of samples, Motion Orientation Histogram (MOH) is calculated, which normally comprised 32 directions. Then, the criteria of determining an important motion is the norm of the flow vector. The level of the importance of a certain motion is proportional to this criterion. At the end, the features of each direction are classified by the motion descriptor via simple statistics on the temporal series to capture the nature of the motion [15].

Thus, constructing the coherent motion field is essential for identifying the coherent motion which can then be used to group and classify the coherent motion [16]. One technique which can produce the precise coherent motion field is Thermal-Diffusion-Based Approach. This technique involves the transfer of the motion input to the thermal energy field (TEF) which can encode the motion correlation and its trends among the particles [16]. Then, the semantic regions are found by using the two clustering techniques based on the relationships between the coherent motions. In short, thermal energy field (TEF) is a more accurate motion field compared to the original motion field from the dense optical flow. Basically, calculation on each pixel for its motion vector, which is a sub pixel of x and y movement has to be performed [17].

Motion Templates

Besides optical flow, another technique known as “Motion Templates” can be used to detect the motion and its direction [18]. Motion Templates technique consisted of four parts, namely updating the motion history image (MHI), computing the gradient orientation of the motion history image, computing the global orientation of the motion in the region of interest and segmentation of motion [18]. The motion history image is updated via moving silhouettes. Silhouette is the essential requirement in motion template for detecting the motion as it can provide the geometric information for the computer vision analysis [19].

Silhouette is created by extracting the exact difference between the two consecutive images from the buffer [20]. This output is then converted to binary image. As the time passing, the number of silhouette created will increase as long as there are movements in the images. In OpenCV, a function known as "timestamp" is used to measure the recency of the image. The most recent silhouette and the older silhouette will be compared to figure out the motion detection. Thus, the motion history image (MHI) is the sequence of silhouettes along with the timestamp [20].

The overall motion can be detected by calculating the gradient of the MHI. In OpenCV, the gradient of MHI is classified based on the upper limit and lower limit of the gradient. Thus, the unacceptable gradient such as the high gradient at the edge can be eliminated. By having the information of MHI, upper and lower limit of MHI gradient and the variable aperture size which depend on the size of the gradient operator, the OpenCV function "cvCalcMotionGradient" can output a mask, which contains the valid gradient and also the direction angle of the detected motion, which is known as the orientation [20]. From this output, the global orientation can also be calculated.

In OpenCV, Motion Templates can also calculate the local orientation. This was done by taking the most recent silhouette from MHI to find its perimeter. "Floodfill", which is a function to fill the connected region is done to highlight the motion found [21]. From this finding, the local motion gradient direction can be computed. Consequently, the motion is displayed on the image.

Cascade Classifier

As the name implies, the motion detection techniques are used to detect the moving objects, but not the static object. Thus, one of the techniques which can be used is the object detection method, for instance, cascade classifier class which also supports HAAR cascade classifier in the form of cross link. In this technique, the classifier is trained with hundreds of views of sample of a target object, for example, faces, cars and building, then scale them into the same size. This kind of sample views of an object is known as positive example. Besides, the classifier also trained with the negative examples, which can consist of any samples which don't have the target object. After the training procedure, the classifier can be applied to an input image, towards the region of interest (ROI). The classifier will output "1" to indicate the existence of the target object in the region, otherwise "0" will be the output [22].

HSV Color Model Classification Techniques

Another method for object detection is to use the HSV (Hue, Saturation and Value) color model classification. “Hue” represents the number from 0 to 360 degrees, where the hues of red starts at 0 degree, yellow starts at 60 degrees, green starts at 120 degrees, cyan starts at 180 degrees, blue starts at 240 degrees, and magenta starts at 300 degrees. “Saturation” represents the amount of gray in the color while “Value” describes the color brightness [23]. HSV describes the relationships among the color in three dimensions. HSV is like a cone, the center axis comprises the white at the top to black at the bottom, with other neutral colors in between. The angle which extending from the center axis represents the hue, while the distance from the center axis describes the saturation, and the distance along the axis is the value [24].

Static and Moving Camera

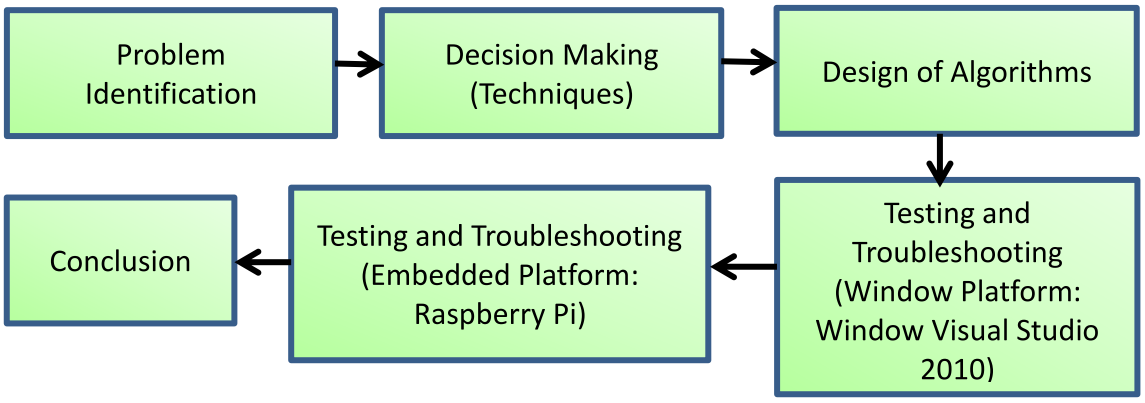
Currently, stationary sensors are being used in most of the surveillance system [25]. However, this project takes the motion of the camera itself into consideration to make this obstacles detection system more stable. Thus, one of the challenges in this project is to solve the problem of motion blur. This problem occurs when capturing the images of moving obstacles, while the camera itself is also moving simultaneously. Consequently, this motion blur distorts the image captured [26]. One of the methods of solving this is through the technology of motion stabilization [27].

The motion stabilization has two approaches in solving motion blur, which are through correction and prevention. In this project, the prevention approach of motion stabilization is applied, which is known as multi-frame image stabilization. This involves the multiple capturing of short exposed images for the same object and then fusing them together to produce zero motion blur image [27]. On the other hand, video stabilization method is used to de-blur the distorted video captured due to the camera shaking. This method involves the processing of many video images and data extraction from the unstable video captured, for instance, the features of the video images [28].

CHAPTER 3

METHODOLOGY

3.1 Overall Project Methodology


Problem Identification
Decision Making (Techniques)
Design of Algorithms
Testing and Troubleshooting (Window Platform: Window Visual Studio 2010)
Testing and Troubleshooting (Embedded Platform: Raspberry Pi)
Conclusion

Below is the list of tools and software which are needed in this project.

i. OpenCV software

ii. Raspberry Pi

iii. Pi Camera

iv. HDMI Monitor

v. Window Visual Studio

The methodology for implementing the algorithm into Raspberry Pi is as shown below:

i. A bootable 16 GB microsd card for Raspberry Pi is set up by using the NOOBS. Pi Camera and OpenCV computer vision library is installed into Raspberry Pi.

ii. The algorithm is developed which can detect the objects in front of the Pi camera. The techniques of image processing and video processing are applied.

iii. The algorithm is developed which can classify objects in front of the Pi Camera, whether they are moving or static.

iv. Write the code to enable the camera to adapt to unstable condition, for instance, the camera itself is moving while capturing the objects or obstacles.

3.2 Motion Objects Detection

3.2.1 Optical Flow Methodology

Step 1: Open Input Video

CvCapture *input_video = cvCsptureFromFile("filename".avi);

This is to get a video file input for optical flow analysis. "filename" is the name of the input video. This step will not be successful if the file is not exist in the computer or the AVI video uses a codec that OpenCV cannot read.

Codec is a program which is capable of compressing a video or audio file when storing it into the disk file and able to decompress the video or audio file when it is being played. The objective is to minimize the storage space for the video or audio file [29].

Step 2: Get a Video Frame

cvQueryFrame(input_video);

This step is to look into the internal information of the AVI video. To do this, OpenCV needs to get a video frame first.

Step 3: Read AVI Video Properties

CvSize Frame_size;

Frame_size.height = cvGetCaptureProperty(input_video, CV_CAP_PROP_FRAME_HEIGHT);

This is to get the width and number of frames of the avi video.

Step 4: Create a Window.

cvNamedWindow("Optical Flow", CV_WINDOW_AUTOSIZE);

This is the place to display the output of the optical flow analysis and for visualization and debugging purposes.

Step 5: Loop Through Frames

Go to certain number of frame, Frame N.

cvSetCaptureProperty(input_video, CV_CAP_PROP_POS_FRAMES, N);

Get Frame N:

IpImage *frame = cvQueryFrame(Input_video);

This is important to note that cvQueryFrame always returns a pointer to the same location in memory.

Step 6: Convert/Allocate

Convert input frame to 8-bit monochrome. This is image format that most of the time OpenCV algorithms operate on.

Step 7: Run Shi and Tomasi

```
CvPoint2D32  frame1_features[N];
cvGoodFeaturesToTrack(
frame1, eig_image, temp_image,
frame1_features, &N, .01, .01, NULL);
```

The “frame1_1C" is the input image. "eig_image" and "temp_image" are just workspace for the algorithm.

These algorithms will return “frame1_features" which contains the feature points.

Step 8: Run Optical Flow

This step is to perform the optical flow analysis.

Step 9: Visualize the Output

The direction of displacement of image points will be drawn by using the arrows.

Step 10: Make an AVI Output

Finally, the output will be displayed in a window.

3.2.2 Motion Templates Methodology

Step 1: Finding Object Silhouettes

This is done by taking the exact difference between two images frames, by using “cvAbsDiff” function of OpenCV. The images is then converted to binary form via “cvThreshold” function.

Step 2: Motion History Image (MHI)

“Timestamp” is used to identify the most recent silhouette image which will be compared with the older silhouette image, to perform motion detection. The sequence of silhouette images and the record of the preceding motion is known as MHI. The function that updates the MHI is “cvUpdateMotionHistory”.

Step 3: Calculation of Motion Gradient

Overall motion is detected through the MHI gradient. High gradient which is unacceptable will be rejected by referring to the upper boundary and lower boundary of the MHI gradient set. By having these gradients as inputs, orientation of the motion can be generated which will show the information of the gradients’ direction angle. The OpenCV function that calculates the motion gradient is “cvCalcMotionGradient”.

Step 4: Calculation of Global Orientation

The orientation of the motion can be classified into two, namely local and global. Global orientation can be computed based on the output from the previous step, from the “cvCalcMotionGradient” function of OpenCV.

In this step, “cvCalcGlobalOrientation” function which uses the output from Step 3 produce the global orientation. The input for this function includes orientation, mask along the timestamp and computation time for producing a MHI template and the MHI.

Step 5: Detecting Local motion using Segmentation

Besides the global orientation, local motion is calculated within the regions of interest (ROI) through segmentation.

This is done via “cvSegmentMotion” function from OpenCV, which requires the inputs of MHI, timestamp, segmentation threshold and the storage object. The output will be the segmentation of each motion. While for every local segments, motion is calculated using “cvCalcGlobalOrientation” function of OpenCV.

3.3 Static Objects Detection

3.3.1 HSV Color Model Classification Methodology

Step 1: Calibration of color threshold

The colour of the target object is determined for its values in Hue, Saturation and Value respectively.

Step 2:Classification of objects based on colour threshold.

The target object is detected through the HSV colour model filtration from the other objects. This is done through the calibrating the values of each H (Hue), S (Saturation) and V (Value).

Step 3: Classification based on salient properties.

Look for objects with same salient properties and group them together.

Step 4: Allocation of vector for similar objects.

Push back similar objects into a C++ vector.

Step 5: Hardcore the setting of the color threshold.

This is to make the program run without calibration of the color threshold again every time when running the program.

Step 6: Display the found objects.

Unpack the vector to display the found objects stored inside it.

Step 7: Track multiple objects.

Repeat Step 1 to Step 6 to track different target objects with different color thresholds.

.

CHAPTER 4
RESULT AND DISCUSSION

4.1 Optical Flow (Window Visual Studio 2010)

4.1.1 Sparse Optical Flow

Before implementing the video processing in Raspberry Pi, the optical flow analysis is implemented on window platform, through the Window Visual Studio which acts as the compiler for the OpenCV.

The following figure shows the result of the simulation of the optical flow.

Figure (5)

Figure (5) shows the original first frame of the AVI video.

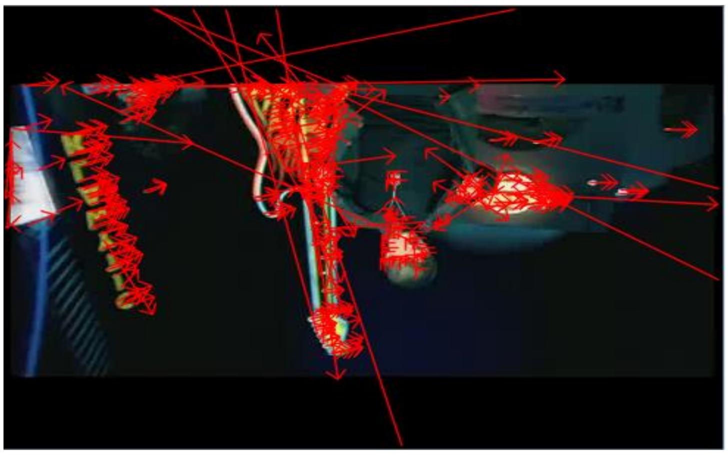


Figure (6)

Figure (6) shows that the output window of the optical flow, in Window Visual Studio is actually flipped vertically. This is because the OpenCV reads the AVI upside-down by default.

Figure (7)

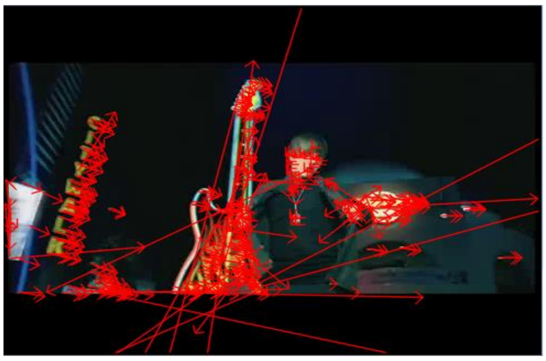


Figure (8)

Figures (7) and (8) above shows the comparison of the original first frame of the AVI input and the output window of the optical flow in Window Visual Studio respectively. Figure (8) of the output window of the optical flow has been flipped vertically for this comparison purpose.

Through this comparison, we know that the arrows in Figure (viii) is actually pointing towards the direction of the displacement of each image points on the AVI video frame that is making some displacement, as the frame makes the transition to the next frame.

In optical flow analysis, given the set of points in an image, it will find those points in another image. Through this tracking of the points, or features of the image pixel, we can find any object from one image to the other and determine the direction of the object moved. This serves the purpose of this project, which is to detect any object or obstacle, whether it is crossing over the road or static in front of the camera.

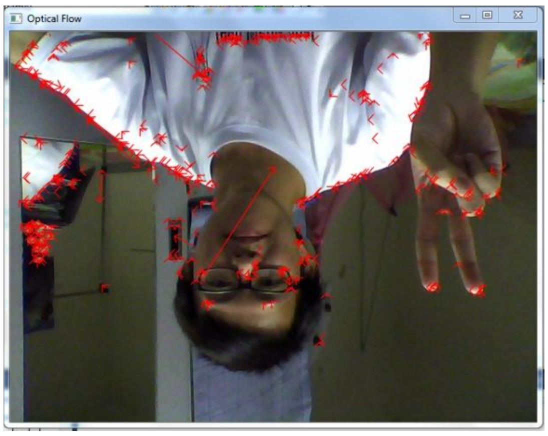
The output window in the Window Visual Studio is the result of comparing the displacement taken place from the first frame to the second frame of the AVI video.

To make the program more flexible, some delay has been added to allow the user to have sufficient time to look at the image. In OpenCV, the delay can be implemented through “**cvWaitKey(x)**”, where x is the argument for the duration of delay.

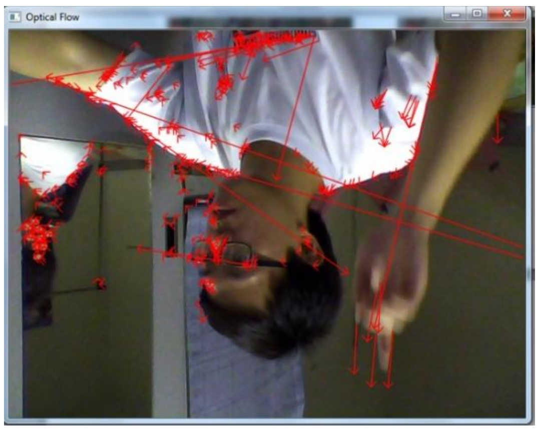
When the argument is 0, the program will wait forever, in this case, the output frame will forever stay at second frame, as the output frame shown in the output window is the result of comparison between the first frame and second frame of the input video. Otherwise, if $x > 0$, the frame will be forwarding to the next frame after time $= x$ until the final frame of the input video.

To call a video which is the stored in laptop to this OpenCV program, the function of “**CvCapture *input_video = cvCaptureFromFile("D:\\\\opticalFlow.avi")**;” is used, where the input_video is the name of the video input variable and “**D:\\\\opticalFlow.avi”** is the memory location in the laptop where the video named as “opticalFlow” is stored. In OpenCV, “CvCapture” is used as a parameter for video capturing function while “cvCaptureFromFile” is to initialize the captured video from the laptop [30].

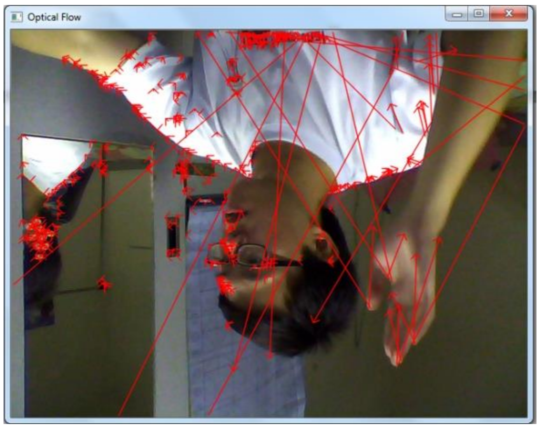
Now, to change the input video from the laptop to the life video input, the function of “cvCaptureFromCAM(CV_CAP_ANY)” is used instead of “cvCaptureFromFile”. The result of using the life video input which is captured from the webcam of the laptop is as shown below.



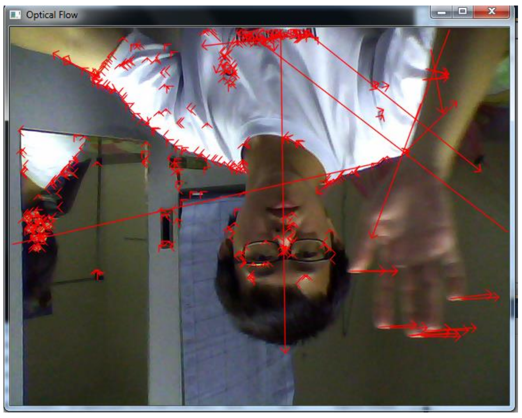
Figure(9)



Figure(10)



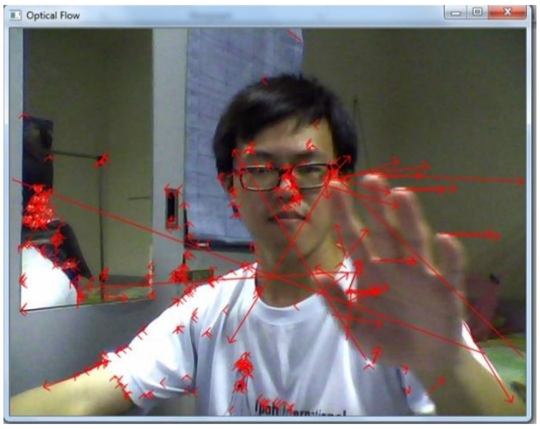
Figure(11)



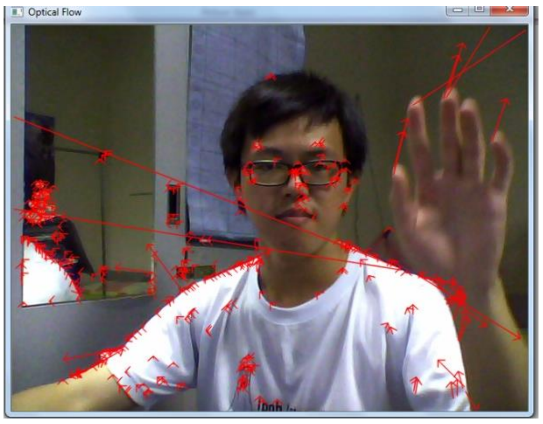
Figure(12)

Since the life video input is no longer of avi format, thus just need to change the argument in the OpenCV function "cvConvertImage" to make the capture image unflipped. (Format: void cvConvertImage(const CvArr* src, CvArr* dst, int flags=0);)

Therefore, original "cvConvertImage(frame, frame1_1C, CV_CVTIMG_FLIP);" is changed to "cvConvertImage(frame, frame1_1C, 0); ", which means the operation flag is equal to zero indicates that no flipping operation is required. The result of no flipping to the captured image is as shown below.



Figure(13)

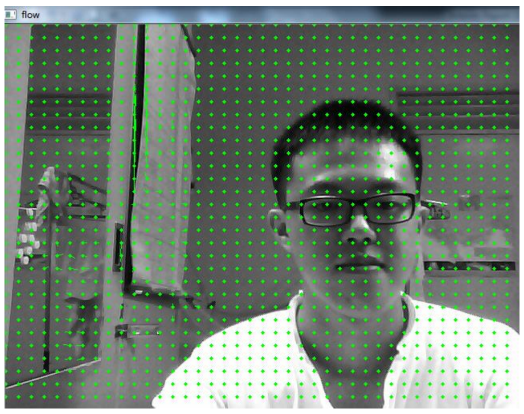


Figure(14)

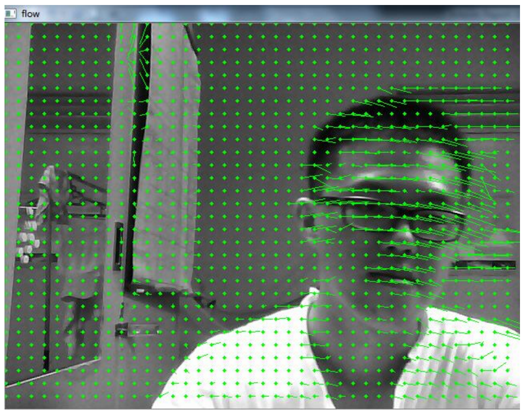
However, the optical flow shown by these results are scattering throughout the whole image and seem to be difficult when further motion classification has to be carried out. From the observation of the images shown above, some unnecessary optical flow are displayed for the very small motion, for example the optical flow for the little vibrations on clothes. Part of the reasons for this is because the optical flow method used is implementing Lucas-Kanade and Shi-Tomasi, where only some pixels from the image are being processed, via the processes of features identification, tracking and extraction from the images. This kind of optical flow approach is known as sparse optical flow.

4.1.2 Dense Optical Flow

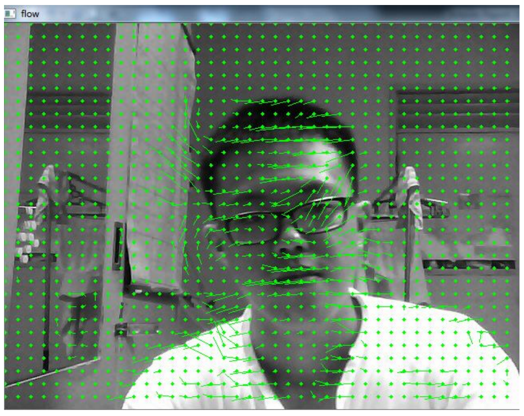
Thus, another method has been attempted, which is the dense optical flow (or Farneback optical flow). The main different between the sparse optical flow and dense optical flow is that the dense optical flow analyse every pixel of the capture image while the sparse optical flow only selecting specific good feature to detect the motion in the capture image. Therefore, the dense optical flow can provide greater sensitivity to motion detection of the objects or obstacles. The figures below show the computation of dense optical flow by Window Visual Studio.



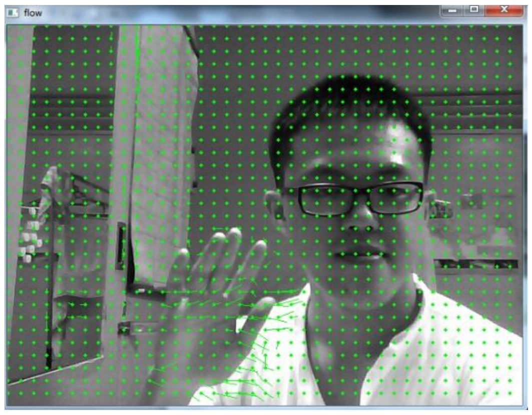
Figure(15)



Figure(16)

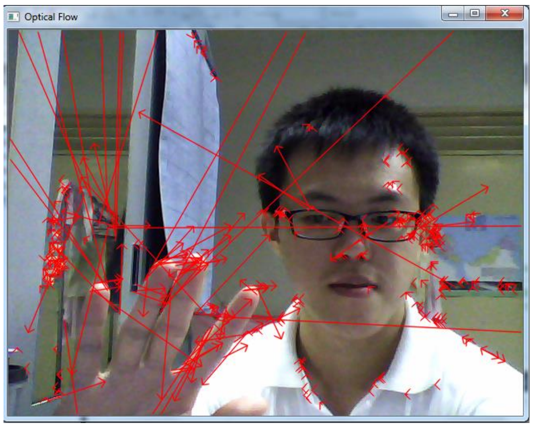


Figure(17)

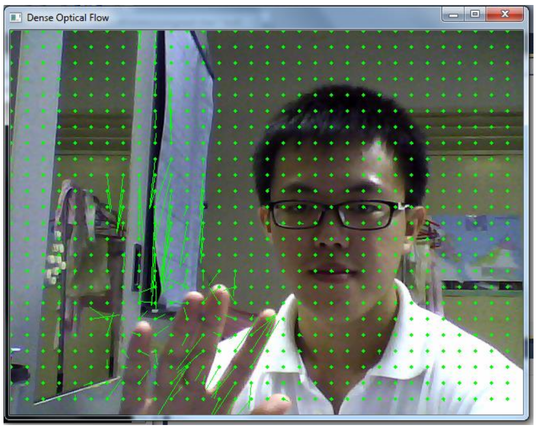


Figure(18)

However, the drawback of dense optical flow is that it has a longer computation time compare to sparse optical flow. This may be due to the increased computation work of dense optical flow compared to the sparse optical flow. This is because the dense optical flow detects the motion at every pixel of the image, while the sparse optical flow selects the certain good features to track and detect the motion at that particular region. The comparison between Sparse Optical Flow and Dense Optical Flow is as shown in Figure (19) and Figure (20).



Figure(19): Sparse Optical Flow which only selects good features to track and detects the motion in that region.

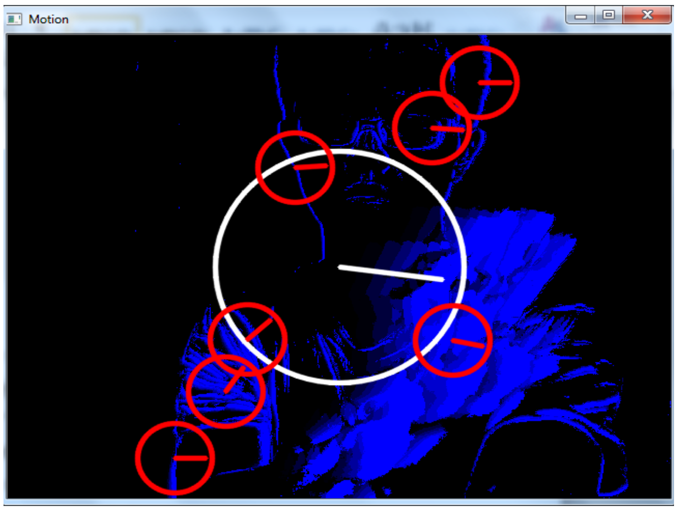


Figure(20): Dense Optical Flow will detect the motion in each pixel of the image. This dense optical flow is modified that the output image is in RGB form as compared to earlier version in the discussion which shows the threshold image.

4.2 Motion Templates (Window Visual Studio 2010)

Later on, another technique which is known as "Motion Templates" also has the capability to detect the motion. By comparing Motion Templates to the Sparse and Dense Optical Flow, Motion Templates has an added advantage as this technique can display the direction of the motion by circling the motion detected and displaying the angle of each motion in red colour, as shown in Figure (21). The bigger white colour circle and angle displays are belonging to the global orientation, which takes the average of the all motion detected to give the information of overall motion and the direction in the image.

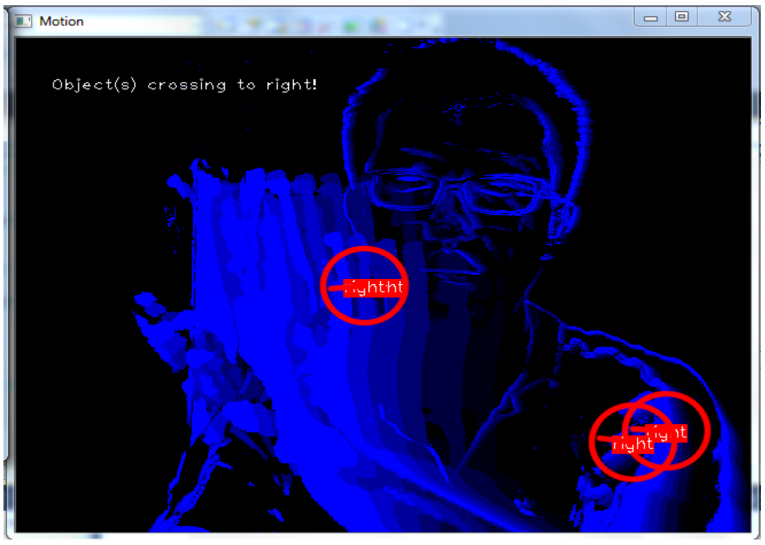
Thus, this method is useful in the case of classifying the moving obstacles because the moving objects or obstacles in front of the camera can be either crossing to the left or right. The C codes "motempl.c" in the OpenCV samples file is being used for detecting motion as well as classifying the moving object's orientation. Figure (21) shows the computation of motion templates in Window Visual Studio.



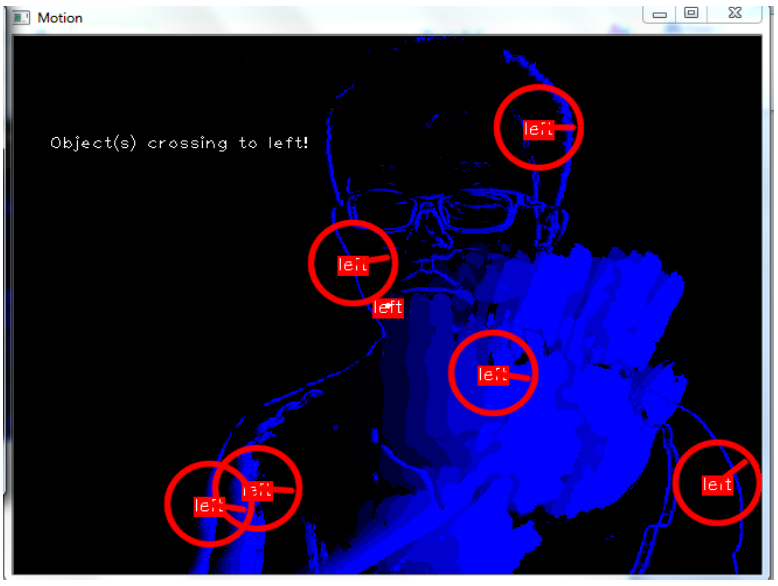
Figure(21)

However, the original "motemp1.c" has no labeling about the direction of the moving objects. Thus, modification has been made to label the direction on the detected motion of the obstacles or objects. Compared to the original motion template, now the label of "left" or "right" is indicated in each red circle that surrounding the moving objects. On the top left side of the window, the labels of "Object(s) moving to the left" or/and "Object(s) moving to the right" are shown to inform users about the existence of the moving objects and their directions.

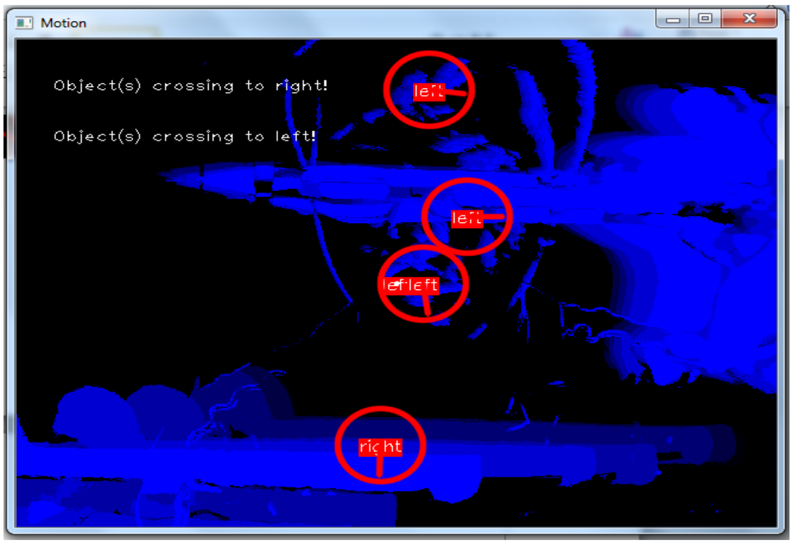
With the labeling added to the original Motion Templates codes, now the user can know the information of the motion orientation clearly, as shown in Figure (22), Figure (23) and Figure (24).



Figure(22): Objects/potential obstacles moving to the right.

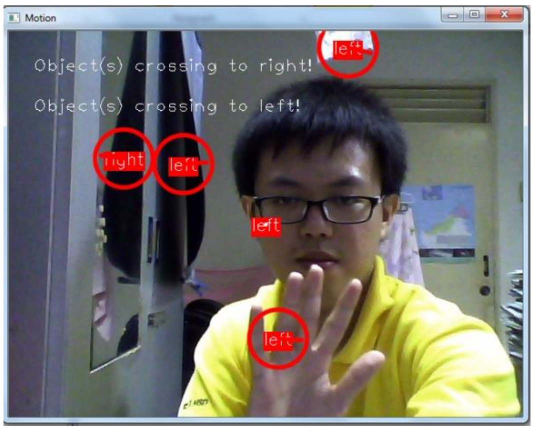


Figure(23): Objects/potential obstacles moving to the left.

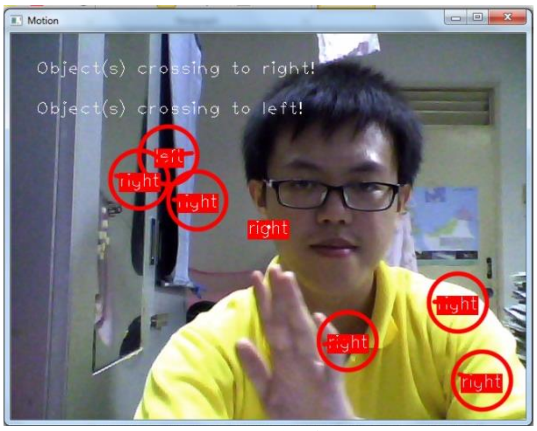


Figure(24): There are objects/potential obstacles moving to the right and left.

In order to make the display of Motion Templates in RGB (Red, Green and Blue), instead of black and blue, some modification had been done. With reference to the coding of the motion templates, the original frame captured by the camera is used, instead of black background. This is done by replacing "cvZero(dst)" with the "cvCopy(img, dst, NULL)". Besides, the blue mask is not drawn by removing the call to cvMerge() [31]. The result of this modification is as shown in Figure (25) and Figure (26).



Figure(25)



Figure(26)

Overall, all the three techniques mentioned earlier, namely sparse and dense optical flow and the motion templates can perform the motion detection as well as the motion direction.

However, the real time computation of motion estimation and its direction is very slow for the dense optical flow technique, compared to the sparse optical flow and the motion templates, due to the large computation of each motion at every pixel of the frame.

With the labeling of the direction on each moving object in the modified version of Motion Templates, this make the user easier and efficient in analyzing the direction, compared to both dense and sparse optical flow techniques. Table 1 below shows the comparison of three motion detection techniques, namely sparse and dense optical flow and the motion templates.

4.3 Comparison between Optical Flow and Motion Templates

<u>Comparison of Three Different Techniques of Motion Detection In OpenCV</u>

	Sparse Optical Flow	**Dense Optical Flow**	**Motion Templates**
Algorithm	<u>i. Lucas Kanade</u> To determine the features of one frame in another frame. <u>ii. Shi-Tomasi</u> To select the good features to track.	<u>i. Gunner Farneback</u> To compute the optical flow for every point in the frame	Update Motion History image (MHI), calculate motion gradient, global orientation and motion segmentation.
Characteristics	Only compute the motion ay some pixels on the frame.	All the motion at each pixel of the frame is computed.	Only moving object(s) will update MHI.
Functionality	Motion Detection and Computation of the Motion Direction		
Assumption	i. The pixel intensities of an object is constant between consecutive frames. ii. The adjacent pixels have similar motion.		No

Table 1

<u>Comparison of Build Time for each technique in Window Visual Studio</u>

Techniques / **Program Testing**	**Sparse Optical Flow**	**Dense Optical Flow**	**Motion Templates**
1	4.93 seconds	4.63 seconds	3.84 seconds
2	5.36 seconds	4.59 seconds	3.63 seconds
3	4.58 seconds	4.85 seconds	3.48 seconds
4	5.22 seconds	4.44 seconds	3.68 seconds
5	4.80seconds	4.81 seconds	3.54 seconds
Average Build Time for five testing for each technique	**4.98 seconds**	**4.66 seconds**	**3.63 seconds**

Table 2

Building is a process for creating one or more new files, known as “target”, from the source code text file, so that they can be used when the application (the program or technique) is run [33]. In this case, the build time is used as a parameter to compare the speed of the building process for the techniques mentioned above.

For each technique, its program is test run for 5 times, the build time for each test run is recorded and the average of the build time is calculated. This is to get the mean of its build time of the technique’s source codes. The building time is computed by the Window Visual Studio and displayed at the output window after the source codes of the technique is built.

From the results, this is obvious that the computation time for the Motion Templates technique is the lowest. In short, Motion Templates is the technique which can perform the motion analysis in the shortest time.

In short, due to the more advantages of motion templates technique compared to the optical flow techniques, motion templates technique is chosen as the method to accomplish the task of detecting the moving objects, as well as to classify the moving objects into either moving to the left or moving to the right.

4.4 Static Object Detection (Window Visual Studio 2010)

Previous techniques which including the optical flow and motion templates are for motion detection. This means that the detection of static objects or obstacles is not within the functionality of these motion detection techniques.

In this project, a technique known as HSV (Hue, Saturation and Value) is used to detect the static object. This is done through the classification of the color of the static object. Once the color threshold of a target object is set, it can be filtered out from the rest objects in an image. Then, any object with this kind of color threshold can be detected and grouped into a vector. This is to enable the detection of more than one of the similar objects with that particular color threshold. At the end, the vector just needs to be unpacked to display the found objects at the output window [32].

The same methodology can be applied to other target objects with other color thresholds. Thus, this HSV color model classification concept is suitable to be applied to detect the static objects. This can compensate the limited functionality of the motion detection techniques discussed earlier, which is to detect only the moving objects. Although the HSV color model classification technique can also be used to track the targets objects in case they are moving, it will not be able to describe the moving objects as detail as the motion detection techniques, for instance Motion Templates.

The figures below show the simulation of the HSV color model classification in Window Visual Studio 2010. For Figure(27) to Figure (29), the left and right picture show the output window of the real time image of the static obstacles detected and the threshold image of the object filtered through calibration of HSV value of the object respectively.

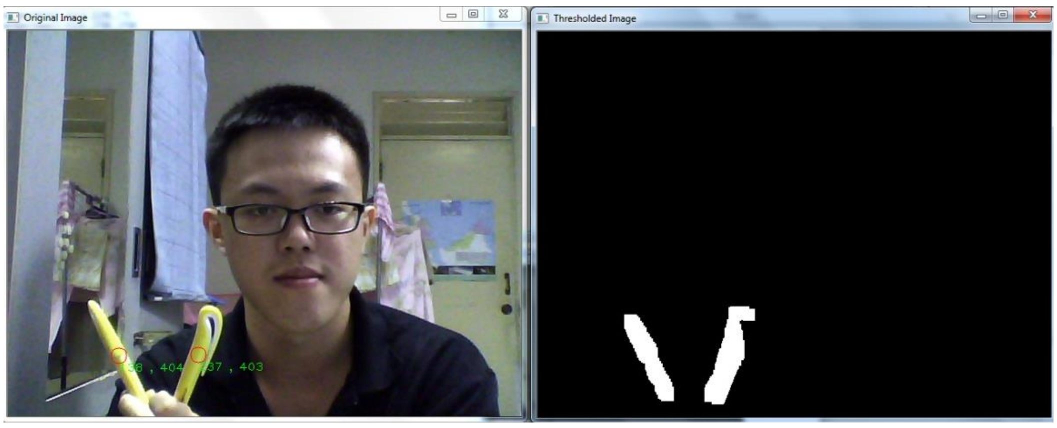


Figure (27) : Calibration for yellow color to detect yellow static obstacles. The value of HSV for yellow: minimum HSV= (26,81,153), maximum HSV=(91,186,256).

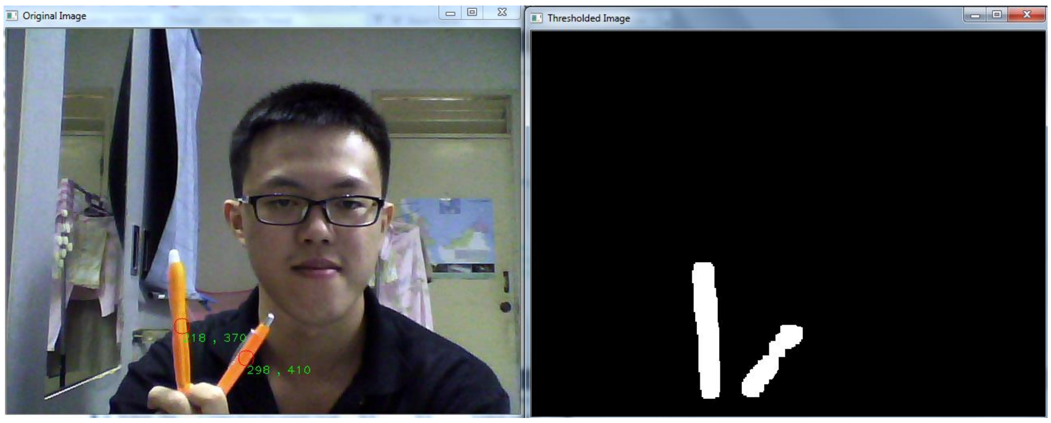


Figure (28) : Calibration for orange color to detect orange static obstacles. The value of HSV for orange: minimum HSV= (5,163,117), maximum HSV=(23,231,256).

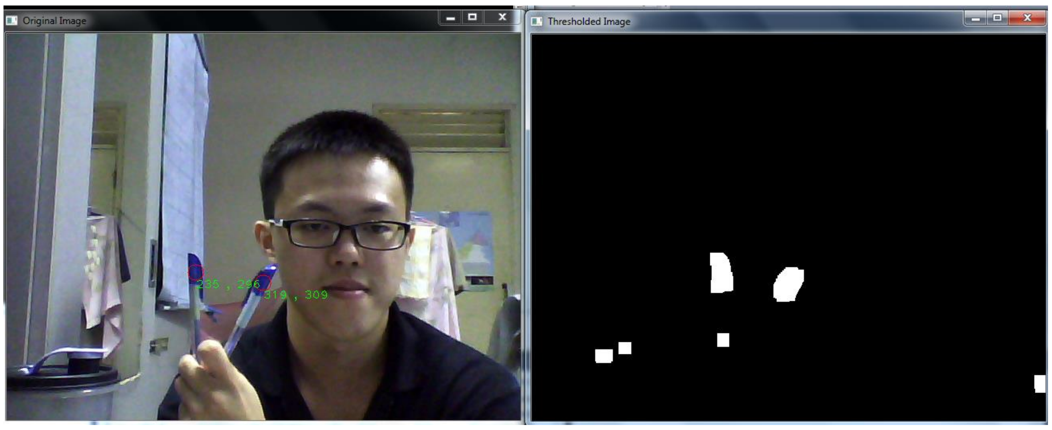


Figure (29): Calibration for blue color to detect blue static obstacles. The value of HSV for blue: minimum HSV= (100,84,68), maximum HSV=(128,256,168).

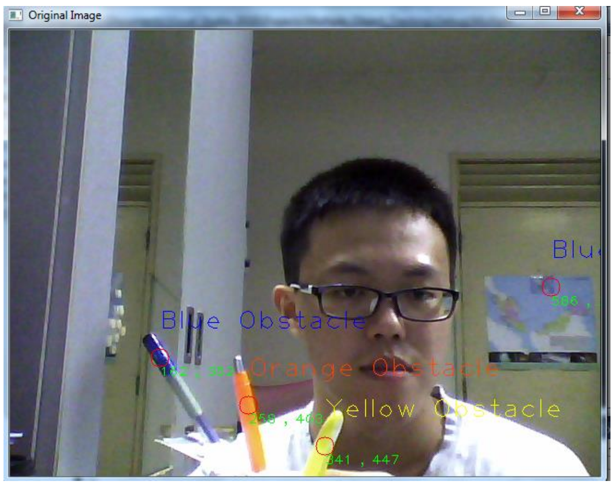


Figure (30): Hard code the calibration made to the HSV color model of the target objects to detect multiple static obstacles simultaneously.

4.5 Raspberry Pi (Embedded System in Linux Platform)

Once the C++ coding of the motion detection techniques is checked and verified that no error occurred in Window Visual Studio 2010, the code is then transferred to the embedded platform, which is Raspberry Pi. The camera used is the 5MP, 1080HD Pi camera.

The code will not function straightaway in Raspberry Pi as the Window Visual Studio 2010 did. Parts of the reason is that in Window Visual Studio 2010, the code is running at the window platform, the compiler will search for the default camera in the laptop, which is the built-in webcam. While in the Raspberry Pi which is using the Linux platform, the user has to acknowledge the compiler so that it can link the code with the Pi camera.

To do this, a library which is known as Raspicam library is needed to be installed to the Raspberry Pi. In this project, the latest version of Raspicam-0.1.6 had been installed into the Raspberry Pi. By using this Raspicam library, the code can run with or without the OpenCV library and Raspicam library does its job of linking to the Pi camera quite well. The user just needs to include the Raspicam library header library to the motion templates code, which is being transferred from Window Visual Studio 2010.

Before running the code, another software “CMake” needed to be installed in Raspberry Pi, through the command “sudo apt-get install cmake”. CMake is an open source cross platform build system that allow the developers to work on different platforms, rather than just compiling the codes with Window Visual Studio in window platform or GNU Make in Linux platform [34]. CMake is controlled by the “CMakeLists.txt” text file which contains a set of instruction. These instruction will be used as the input to the CMake for building the codes [35].

Then, a new directory, named as “build” is required to be created under the project directory. The CMake is called in the “build” directory to point the top-level CMakeLists.txt, which is in the project directory. CMake will process the CMakeLists.txt files, linking to the location of all libraries and include paths and producing the configuration information, for instance, Makefile in the build directory. The next operation will be compiling the codes in the make system [36]. After that, by writing the command "./(the project_code_name)" in the build directory, the motion templates code will run just the same as in Window Visual Studio 2010.

However, there is a problem in displaying the result of the Motion Templates in Raspberry Pi. This is because the angles are displayed on the output window are always zero. The OpenCV function which calculates the angle of the moving objects is "cvCalcGlobalOrientation()". Thus, a hypothesis is made that the problem of zero angle for every moving object is caused by the malfunction of "cvCalcGlobalOrientation()" function in Linux platform. Further studies are required to figure out the root cause of this issue in running the code of Motion Templates in Raspberry Pi, which is Linux platform. Figure (xxvii) and Figure (xxviii) show the Motion Templates which is running in Raspberry Pi.

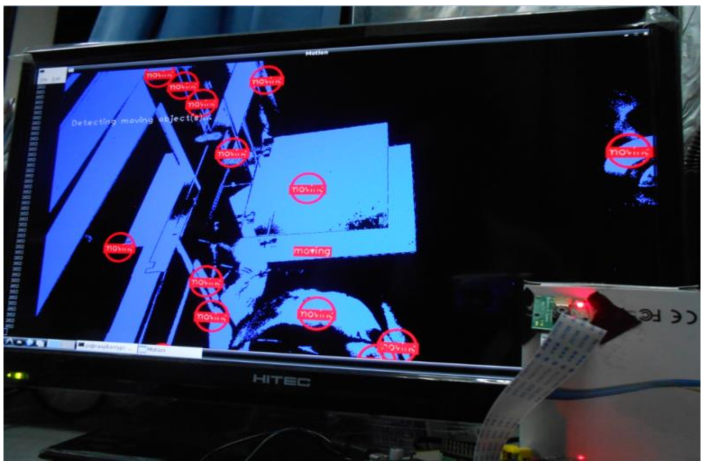


Figure (31): Modified Motion Templates with labeling which displayed in blue and black background.

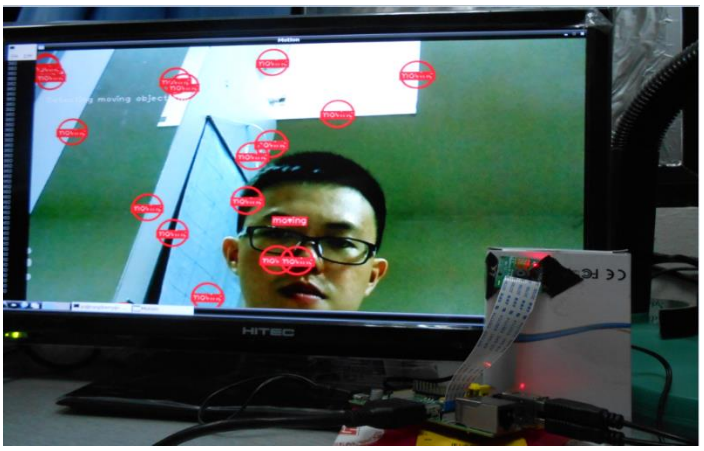


Figure (32): Modified Motion Templates with labeling which displayed in normal color image.

Gantt Chart 1

FYP1 Project Flow	Week (September 2014- December 2014)													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Decide which project to be chosen	🟧	🟧												
Consultation with supervisor	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
Research and data collection	🟦	🟦	🟦	🟦										
Identification of potential problem					🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥		
Prepare Extended report				🟦	🟦	🟦								
Submit Extend Proposal/ Report (31/10)						🟥								
FYP Proposal Defense Preparation (17-21/10)						🟩	🟩	🟩	🟩					
Presentation using power point												🟧	🟧	🟧
Draft Interim Report (Hardcopy)									🟪	🟪	🟪	🟪	🟪	
Interim report												🟫	🟫	🟫

Table 3

Gantt Chart 2

FYP2 Project Flow	Week (January 2015 – May 2015)													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Optical flow analysis in continuous image frames	🟧	🟧	🟧											
Optical flow analysis with life video input (Window Visual Studio)	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
Optical flow analysis with life video input (Raspberry Pi)					🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Identification of potential problem	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫		
Prepare and submit Progress Report	🟦	🟦	🟦	🟦	🟦	🟦	🟦							
Prepare and submit Technical Paper						🟥	🟥	🟥	🟥	🟥	🟥	🟥		
Prepare and submit Dissertation						🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	
FYP Viva											🟪	🟪	🟪	🟪

Table 4

CHAPTER 5
CONCLUSION AND RECOMMENDATION

5.1 Conclusion

At the end of this project, the following objectives are achieved:

i. Real-time image and video processing algorithm can be implemented on an embedded computing platform, which is Raspberry Pi.

ii. Classification of detected object into static and moving is done and can be applied to the object avoidance or collision avoidance system.

After comparing the advantages and the disadvantages of the motion detection techniques, Motion Templates technique is being selected as the appropriate method to accomplish the objective of this object, which is to detect and classify the objects or obstacles into stationary and dynamic, while the dynamic objects detected are further classified into the left and right movement. By comparing the Motion Templates to the Optical Flow technique, Motion Templates have extra information at the output, which is the angle. This makes Motion Templates technique more user friendly as the angle information can be used directly to indicate the direction of the moving objects.

Although the optical flow technique does not have the angle in its output information, the moving objects detected with the direction arrows can be segmented through the motion segmentation technique. This allow optical flow technique to define the direction of the moving objects and classify the moving objects in more detail, for instance the moving objects can be human, bus, cars, bicycles and others. However, this approach will be more complicated than directly using the angle information of the motion templates to describe the moving objects. This approach can be explored in future to further describing the objects detected.

For the static object detection, the HSV color model classification technique used is working well but more calibration of different colors are necessary to increase the sensitivity of the static object detection. The approach can also be applied in detecting the moving objects since once the color of the target object is calibrated for its HSV value, the target object will be marked and tracked, regardless of whether it is moving or static. In spite of that, HSV color model classification technique will not able to describe the moving objects in detail, for example their moving direction. Thus, this HSV color model classification techniques will be more appropriate to be used in detecting the static objects.

The difficulties in implementing the algorithms of static and motion object detection techniques, namely optical flow, motion templates and HSV color model classification is the linking between the Pi Camera with the Raspberry Pi. This is because the platform of Window and Linux are completely different. In the Window Visual Studio 2010, the C++ codes of the techniques discussed above will search for the default webcam in the laptop. This will not happen in Raspberry Pi as the C++ codes running in Linux platform of Raspberry Pi need extra instruction so that the compiler can link the C++ code with the Pi camera. Luckily, there is one library which is known as Raspicam can be installed into the Raspberry Pi. The Raspicam library speeds up the complicated procedures in linking the Pi Camera with the compiler. Thus, by including the Raspicam library into the C++ codes in Raspberry Pi, it will run just the same as the Window Visual Studio 2010.

Nonetheless, there is a mystery in running the Motion Templates code in Raspberry Pi which remains unsolved till now. The motion templates works the same as in Window Visual Studio 2010, but the angles returned for the detected moving objects are always zero. The OpenCV function which responsible in computing the angle is the cvCalGlobalOrientation(). As the hypothesis for this issue, this OpenCV function is not functioning correctly when it is run in Linux platform.

5.2 Recommendation

Throughout this project, the camera which is the main tool to capture the objects is static. Due to the time constraint, this project has not cover the situation where the camera itself is also moving. Besides, to find out the better algorithm for static objects detection, the current static object detection technique, which is HSV color model classification can be compared with another object detection techniques, which is known as cascade classifier. In short, future studies can improve the current project by implementing the algorithm which is more robust in detecting the moving and static objects and capable to overcome the problem caused by the moving camera or so-called camera shaking, for instance motion blur.

REFERENCE

[1] U. Sinha, "The Shi-Tomasi Corner Detector."
[2] R. Owens, "Optical flow," 1997.
[3] J. L. B. S. S. Beauchemin, "The Computation of Optical Flow," *ACM Computing Surveys*, vol. 27 (3), pp. 433-467, September 1995 1995.
[4] MODMYPI, "Raspberry Pi Camera Board (5MP, 1080p, v1.3)."
[5] A. M. Milenkovic, I. M. Markovic, D. S. Jankovic, and P. J. Rajkovic, "Using of Raspberry Pi for data acquisition from biochemical analyzers," in *Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS), 2013 11th International Conference on*, 2013, pp. 389-392.
[6] R. Connect. Raspberry Connect – Noobs – New Out of The Box Software [Online]. Available: http://www.raspberryconnect.com/operating-system-s/item/143-noobs-new-out-of-the-box-software
[7] M. Marengoni and D. Stringhini, "High Level Computer Vision Using OpenCV," in *Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2011 24th SIBGRAPI Conference on*, 2011, pp. 11-24.
[8] A. E. Ortiz and N. Neogi, "Color Optic Flow: A Computer Vision Approach for Object Detection on UAVs," in *25th Digital Avionics Systems Conference, 2006 IEEE/AIAA*, 2006, pp. 1-12.
[9] C. Lei, Y. Hua, T. Takaki, and I. Ishii, "Real-time frame-straddling-based optical flow detection," in *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, 2011, pp. 2447-2452.
[10] Y. Wu, "Optical Flow and Motion Analysis," pp. 1-12.
[11] "Lecture 17: Optical Flow," 2010.
[12] X. Ren, "Local Grouping for Optical Flow."
[13] S. K. Min Sun, "Optical Flow."
[14] O. D. Team. Optical Flow [Online]. Available: http://docs.opencv.org/trunk/doc/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html
[15] A. M. Fabio Mart ńez1, Eduardo Romero, "A motion descriptor based on statistics of optical flow

orientations for action classification in video-surveillance," 2011.
[16] W. L. Weiyue Wang, , Yuanzhe Chen, Jianxin Wu, and a. B. S. Jingdong Wang, "Finding Coherent Motions

and Semantic Regions in Crowd Scenes:

A Diffusion and Clustering Approach," pp. 756-771, 2014.
[17] M. Seymour, "Art of Optical Flow," ed, 2006.
[18] Motion Templates [Online]. Available: http://docs.opencv.org/trunk/modules/optflow/doc/motion_templates.html
[19] L. Q. Gang ZENG. SILHOUETTE EXTRACTION

FROM MULTIPLE IMAGES OF AN UNKNOWN BACKGROUND [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.126.2993&rep=rep1&type=pdf
[20] P. K. P. N. OpenCV: Implementing Motion Detection using Motion Templates [Online]. Available: http://tech-

lightenment.blogspot.com/2012/05/implementing-motion-detection-using.html
[21] Miscellaneous Image Transformations [Online]. Available: http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html
[22] O. D. Team. (2014). *Cascade Classification*. Available: http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html
[23] J. H. Bear. (2015). *HSV*. Available: http://desktoppub.about.com/od/glossary/g/HSV.htm
[24] *Color Theory*. Available: http://learn.colorotate.org/color-models/#.VSsTGfmUfSk
[25] C. Chung-Hao, C. Chang, D. Page, A. Koschan, and M. Abidi, "A Moving Object Tracked by A Mobile Robot with Real-Time Obstacles Avoidance Capacity," in *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, 2006, pp. 1091-1094.
[26] M. Tico and M. Vehvilainen, "Robust method of digital image stabilization," in *Communications, Control and Signal Processing, 2008. ISCCSP 2008. 3rd International Symposium on*, 2008, pp. 316-321.
[27] M. Tico, "Adaptive block-based approach to image stabilization," in *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, 2008, pp. 521-524.
[28] Z. Gang, Y. Luming, and W. Wenlong, "Video stabilization algorithm based on video object segmentation," in *Future Computer and Communication (ICFCC), 2010 2nd International Conference on*, 2010, pp. V2-509-V2-512.
[29] TechTerms.com, "Codec," 2014.
[30] "HighGUI Reference Manual."
[31] ButterCookies. (2011, 10 February 2011). *Motion Templates*. Available: http://experienceopencv.blogspot.com/2011/02/motion-templates.html
[32] K. Hounslow, "OpenCV Tutorial: Multiple Object Tracking in Real Time," ed, 2013.
[33] Getting Started with Visual Studio [Online]. Available: http://www.cs.tufts.edu/research/graphics/resources/vs_getting_started/vs_getting_started.htm
[34] J. Lamp. (2014). *CMake Tutorial*. Available: https://www.johnlamp.net/cmake-tutorial.html
[35] O. S. R. Foundation. (2015). *catkin and CMakeLists.txt*. Available: http://wiki.ros.org/catkin/CMakeLists.txt
[36] (2011). *Cmake*. Available: http://www.cs.swarthmore.edu/~adanner/tips/cmake.php

APPENDICES

Appendix 1: Sparse Optical Flow

```
#include "opencv/highgui.h"
#include "opencv/cv.h"
#include "stdio.h"
#include <math.h>

static const double pi = 3.14159265358979323846;

inline static double square(int a)
{
        return a * a;
}

/* This is just an inline that allocates images.  I did this to reduce clutter in the
 * actual computer vision algorithmic code.  Basically it allocates the requested
image
 * unless that image is already non-NULL.  It always leaves a non-NULL image as-is
even
 * if that image's size, depth, and/or channels are different than the request.
 */

inline static void allocateOnDemand( IplImage **img, CvSize size, int depth, int
channels )
{
        if ( *img != NULL )    return;

        *img = cvCreateImage( size, depth, channels );
        if ( *img == NULL )
        {
                fprintf(stderr, "Error: Couldn't allocate image.  Out of memory?\n");
                exit(-1);
        }
}

/*int main (int argc, char** argv) {    */

int main(void){

        IplImage* frame = 0;
        //cvNamedWindow("Example2", CV_WINDOW_AUTOSIZE);
        CvCapture* input_video =0;
        input_video = cvCaptureFromCAM(CV_CAP_ANY);


                /* Read the video's frame size out of the AVI. */
        CvSize frame_size;
        frame_size.height =
                (int) cvGetCaptureProperty( input_video, CV_CAP_PROP_FRAME_HEIGHT );
        frame_size.width =
                (int) cvGetCaptureProperty( input_video, CV_CAP_PROP_FRAME_WIDTH );

        /* Determine the number of frames in the AVI. */
        long number_of_frames;
        /* Go to the end of the AVI (ie: the fraction is "1") */
        cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_AVI_RATIO, 1. );
        /* Now that we're at the end, read the AVI position in frames */
        number_of_frames = (int) cvGetCaptureProperty( input_video,
CV_CAP_PROP_POS_FRAMES );
        /* Return to the beginning */
        cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_FRAMES, 0. );

        /* Create a windows called "Optical Flow" for visualizing the output.
         * Have the window automatically change its size to match the output.
         */
        cvNamedWindow("Optical Flow", CV_WINDOW_AUTOSIZE);

        long current_frame = 0;
        while(true)
        {
                static IplImage *frame = NULL, *frame1 = NULL, *frame1_1C = NULL,
*frame2_1C = NULL, *eig_image = NULL, *temp_image = NULL, *pyramid1 = NULL, *pyramid2
= NULL;

                /* Go to the frame we want.  Important if multiple frames are queried
in
                 * the loop which they of course are for optical flow.  Note that the
very
                 * first call to this is actually not needed. (Because the correct
position
                 * is set outsite the for() loop.)
                 */
                cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_FRAMES,
current_frame );

                /* Get the next frame of the video.
                 * IMPORTANT!  cvQueryFrame() always returns a pointer to the _same_
                 * memory location.  So successive calls:
                 * frame1 = cvQueryFrame();
                 * frame2 = cvQueryFrame();
                 * frame3 = cvQueryFrame();
                 * will result in (frame1 == frame2 && frame2 == frame3) being true.
                 * The solution is to make a copy of the cvQueryFrame() output.
                 */


                frame = cvQueryFrame( input_video );
                if (frame == NULL)
                {
                        /* Why did we get a NULL frame?  We shouldn't be at the end. */
                        fprintf(stderr, "Error: Hmm. The end came sooner than we
thought.\n");
                        return -1;
                }
                /* Allocate another image if not already allocated.
                 * Image has ONE channel of color (ie: monochrome) with 8-bit "color"
depth.
                 * This is the image format OpenCV algorithms actually operate on
(mostly).
                 */
                allocateOnDemand( &frame1_1C, frame_size, IPL_DEPTH_8U, 1 );
                /* Convert whatever the AVI image format is into OpenCV's preferred
format.
                 * AND flip the image vertically.  Flip is a shameless hack.  OpenCV
reads
                 * in AVIs upside-down by default.  (No comment :-))
                 */

                //cvConvertImage(frame, frame1_1C, CV_CVTIMG_FLIP);
                cvConvertImage(frame, frame1_1C, 0);

                /* We'll make a full color backup of this frame so that we can draw on
it.
                 * (It's not the best idea to draw on the static memory space of
cvQueryFrame().)
                 */
                allocateOnDemand( &frame1, frame_size, IPL_DEPTH_8U, 3 );
                //cvConvertImage(frame, frame1, CV_CVTIMG_FLIP);
                cvConvertImage(frame, frame1, 0);

                /* Get the second frame of video.  Same principles as the first. */
                frame = cvQueryFrame( input_video );
                if (frame == NULL)
                {
                        fprintf(stderr, "Error: Hmm. The end came sooner than we
thought.\n");
                        return -1;
                }
                allocateOnDemand( &frame2_1C, frame_size, IPL_DEPTH_8U, 1 );

                //cvConvertImage(frame, frame2_1C, CV_CVTIMG_FLIP);
                cvConvertImage(frame, frame2_1C, 0);
                /* Shi and Tomasi Feature Tracking! */

                /* Preparation: Allocate the necessary storage. */
                allocateOnDemand( &eig_image, frame_size, IPL_DEPTH_32F, 1 );
                allocateOnDemand( &temp_image, frame_size, IPL_DEPTH_32F, 1 );

                /* Preparation: This array will contain the features found in frame 1.
*/
                CvPoint2D32f frame1_features[400];

                /* Preparation: BEFORE the function call this variable is the array
size
                 * (or the maximum number of features to find).  AFTER the function
call
                 * this variable is the number of features actually found.
                 */
                int number_of_features;

                /* I'm hardcoding this at 400.  But you should make this a #define so
that you can
                 * change the number of features you use for an accuracy/speed tradeoff
analysis.
                 */
                number_of_features = 400;

                /* Actually run the Shi and Tomasi algorithm!!
                 * "frame1_1C" is the input image.
                 * "eig_image" and "temp_image" are just workspace for the algorithm.
                 * The first ".01" specifies the minimum quality of the features (based
on the eigenvalues).
                 * The second ".01" specifies the minimum Euclidean distance between
features.
                 * "NULL" means use the entire input image.  You could point to a part
of the image.
                 * WHEN THE ALGORITHM RETURNS:
                 * "frame1_features" will contain the feature points.
                 * "number_of_features" will be set to a value <= 400 indicating the
number of feature points found.
                 */
               cvGoodFeaturesToTrack(frame1_1C, eig_image, temp_image, frame1_features,
&number_of_features, .01, .01, NULL);

                /* Pyramidal Lucas Kanade Optical Flow! */

                /* This array will contain the locations of the points from frame 1 in
frame 2. */
                CvPoint2D32f frame2_features[400];

                /* The i-th element of this array will be non-zero if and only if the
i-th feature of
                 * frame 1 was found in frame 2.
                 */
                char optical_flow_found_feature[400];

                /* The i-th element of this array is the error in the optical flow for
the i-th feature
                 * of frame1 as found in frame 2.  If the i-th feature was not found
(see the array above)
                 * I think the i-th entry in this array is undefined.
                 */
                float optical_flow_feature_error[400];

                /* This is the window size to use to avoid the aperture problem (see
slide "Optical Flow: Overview"). */
                CvSize optical_flow_window = cvSize(3,3);

                /* This termination criteria tells the algorithm to stop when it has
either done 20 iterations or when
                 * epsilon is better than .3.  You can play with these parameters for
speed vs. accuracy but these values
                 * work pretty well in many situations.
                 */
                CvTermCriteria optical_flow_termination_criteria
                        = cvTermCriteria( CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 20, .3 );

                /* This is some workspace for the algorithm.
                 * (The algorithm actually carves the image into pyramids of different
resolutions.)
                 */
                allocateOnDemand( &pyramid1, frame_size, IPL_DEPTH_8U, 1 );
                allocateOnDemand( &pyramid2, frame_size, IPL_DEPTH_8U, 1 );

                /* Actually run Pyramidal Lucas Kanade Optical Flow!!
                 * "frame1_1C" is the first frame with the known features.
                * "frame2_1C" is the second frame where we want to find the first
frame's features.
                * "pyramid1" and "pyramid2" are workspace for the algorithm.
                * "frame1_features" are the features from the first frame.
                * "frame2_features" is the (outputted) locations of those features in
the second frame.
                * "number_of_features" is the number of features in the
frame1_features array.
                * "optical_flow_window" is the size of the window to use to avoid the
aperture problem.
                * "5" is the maximum number of pyramids to use.  0 would be just one
level.
                * "optical_flow_found_feature" is as described above (non-zero iff
feature found by the flow).
                * "optical_flow_feature_error" is as described above (error in the
flow for this feature).
                * "optical_flow_termination_criteria" is as described above (how long
the algorithm should look).
                * "0" means disable enhancements.  (For example, the second array
isn't pre-initialized with guesses.)
                */
               cvCalcOpticalFlowPyrLK(frame1_1C, frame2_1C, pyramid1, pyramid2,
frame1_features, frame2_features, number_of_features, optical_flow_window, 5,
optical_flow_found_feature, optical_flow_feature_error,
optical_flow_termination_criteria, 0 );

               /* For fun (and debugging :)), let's draw the flow field. */
               for(int i = 0; i < number_of_features; i++)
               {
                       /* If Pyramidal Lucas Kanade didn't really find the feature,
skip it. */
                       if ( optical_flow_found_feature[i] == 0 )     continue;

                       int line_thickness;                         line_thickness = 1;
                       /* CV_RGB(red, green, blue) is the red, green, and blue
components
                        * of the color you want, each out of 255.
                        */
                       CvScalar line_color;                  line_color =
CV_RGB(255,0,0);

                       /* Let's make the flow field look nice with arrows. */

                       /* The arrows will be a bit too short for a nice visualization
because of the high framerate
                        * (ie: there's not much motion between the frames).  So let's
lengthen them by a factor of 3.
                        */
                       CvPoint p,q;
                       p.x = (int) frame1_features[i].x;
                       p.y = (int) frame1_features[i].y;
                       q.x = (int) frame2_features[i].x;
                       q.y = (int) frame2_features[i].y;

                       double angle;          angle = atan2( (double) p.y - q.y,
(double) p.x - q.x );
                       double hypotenuse;     hypotenuse = sqrt( square(p.y - q.y) +
square(p.x - q.x) );
```

```
                        /* Here we lengthen the arrow by a factor of three. */
                        q.x = (int) (p.x - 3 * hypotenuse * cos(angle));
                        q.y = (int) (p.y - 3 * hypotenuse * sin(angle));

                        /* Now we draw the main line of the arrow. */
                        /* "frame1" is the frame to draw on.
                         * "p" is the point where the line begins.
                         * "q" is the point where the line stops.
                         * "CV_AA" means antialiased drawing.
                         * "0" means no fractional bits in the center cooridinate or
radius.
                         */
                        cvLine( frame1, p, q, line_color, line_thickness, CV_AA, 0 );
                        /* Now draw the tips of the arrow.  I do some scaling so that
the
                         * tips look proportional to the main line of the arrow.
                         */
                        p.x = (int) (q.x + 9 * cos(angle + pi / 4));
                        p.y = (int) (q.y + 9 * sin(angle + pi / 4));
                        cvLine( frame1, p, q, line_color, line_thickness, CV_AA, 0 );
                        p.x = (int) (q.x + 9 * cos(angle - pi / 4));
                        p.y = (int) (q.y + 9 * sin(angle - pi / 4));
                        cvLine( frame1, p, q, line_color, line_thickness, CV_AA, 0 );
                }
                /* Now display the image we drew on.  Recall that "Optical Flow" is the
name of
                 * the window we created above.
                 */
                cvShowImage("Optical Flow", frame1);
                /* And wait for the user to press a key (so the user has time to look
at the image).
                 * If the argument is 0 then it waits forever otherwise it waits that
number of milliseconds.
                 * The return value is the key the user pressed.
                 */
                int key_pressed;
                key_pressed = cvWaitKey(33);

                /* If the users pushes "b" or "B" go back one frame.
                 * Otherwise go forward one frame.
                 */
                if (key_pressed == 'b' || key_pressed == 'B') current_frame--;
                else
                current_frame++;
                /* Don't run past the front/end of the AVI. */
                if (current_frame < 0)
        current_frame = 0;
                if (current_frame >= number_of_frames - 1)    current_frame =
number_of_frames - 2;
        }
}
```

Appendix 2: Dense Optical Flow

```
#include <opencv2/opencv.hpp>
#include "opencv2/video/tracking.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc_c.h"
#include <time.h>
#include <stdio.h>
#include <ctype.h>

using namespace cv;


void drawOptFlowMap(const cv::Mat& flow,
                    cv::Mat& cflowmap,
                    int step,
                    const cv::Scalar& color
                   )
{
    for(int y = 0; y < cflowmap.rows; y += step)
        for(int x = 0; x < cflowmap.cols; x += step)
        {
            const cv::Point2f& fxy = flow.at<cv::Point2f>(y, x);
            cv::line(cflowmap,
                             cv::Point(x,y),
                             cv::Point(cvRound(x+fxy.x),cvRound(y+fxy.y)),
                 color);
            cv::circle(cflowmap, cv::Point(x,y), 2, color, -1);
        }
}
*/
int main(int argc, char **argv) {


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

   Mat newFrame, newGray, prevGray;

  // Mat NEWFRAME;

    cap >> newFrame; // get a new frame from camera, for fback.cpp

        //cap >> NEWFRAME;
        //CvCapture* NEWFRAME = 0;// for Motempl.cpp

        CvCapture* capture = 0;   //for Motempl.cpp not yet merged with the fback.cpp

        //NEWFRAME = cvCaptureFromCAM(CV_CAP_ANY);

    cvtColor(newFrame, newGray, CV_BGR2GRAY);
    prevGray = newGray.clone();

    double pyr_scale = 0.5;
    int levels = 3;
    int winsize = 5;
    int iterations = 5;
    int poly_n = 5;
    double poly_sigma = 1.1;
    int flags = 0;


    while(1) {

               Mat forcoherent;
        cap >> newFrame;
        if(newFrame.empty()) break;
        cvtColor(newFrame, newGray, CV_BGR2GRAY);
               cvtColor(newFrame, forcoherent, CV_BGR2GRAY);

        Mat flow = Mat(newGray.size(), CV_32FC2);
               Mat flowa = Mat(forcoherent.size(), CV_32FC2);

        /* Do optical flow computation */
       calcOpticalFlowFarneback(
            prevGray,
            newGray,
            flow,
            pyr_scale,
            levels,
            winsize,
            iterations,
            poly_n,
            poly_sigma,
            flags
            );


        drawOptFlowMap(flow, newFrame, 20, CV_RGB(0,255,0));

               namedWindow("Dense Optical Flow", 1);
               imshow("Dense Optical Flow", newFrame);
               waitKey(1);
               prevGray = newGray.clone();


        }
//      }

    return 0;
}



#ifdef _EiC
main(1,"motempl.c");
#endif
```

Appendix 3: Motion Templates

```
#include "opencv2/video/tracking.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc_c.h"
#include <time.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <iostream>
#include <sstream>
#include <string>
//#include <raspicam/raspicam_cv.h>   //include this if run the code in Raspberry Pi
//#include <raspicam/raspicam.h>      //include this if run the code in Raspberry Pi

using namespace cv;

static void help(void)
{
    printf(
            "\nThis program demonstrated the use of motion templates -- basically
using the gradients\n"
            "of thresholded layers of decaying frame differencing. New movements are
stamped on top with floating system\n"
            "time code and motions too old are thresholded away. This is the 'motion
history file'. The program reads from the camera of your choice or from\n"
            "a file. Gradients of motion history are used to detect direction of
motoin etc\n"
            "Usage :\n"
            "./motempl [camera number 0-n or file name, default is camera 0]\n"
            );
}
// various tracking parameters (in seconds)
const double MHI_DURATION = 1;
const double MAX_TIME_DELTA = 0.5;
const double MIN_TIME_DELTA = 0.05;
// number of cyclic frame buffer used for motion detection
// (should, probably, depend on FPS)
const int N = 4;

// ring image buffer
IplImage **buf = 0;
int last = 0;

// temporary images
IplImage *mhi = 0; // MHI
IplImage *orient = 0; // orientation
IplImage *mask = 0; // valid orientation mask
IplImage *segmask = 0; // motion segmentation map
CvMemStorage* storage = 0; // temporary storage

Mat im;


//***************** Helper function to put text in the center of a
rectangle*********************
static void set_label(cv::Mat& im, cv::Rect r, const std::string label)
```

```
{

        //IplImage* img;
        //Mat im(img);
    int fontface = cv::FONT_HERSHEY_SIMPLEX;
    double scale = 0.7;
    int thickness = 1;
    int baseline = 0;

    cv::Size text = cv::getTextSize(label, fontface, scale, thickness, &baseline);
    cv::Point pt(r.x + (r.width-text.width)/2, r.y + (r.height+text.height)/2);

    cv::rectangle(
        im,
        pt + cv::Point(0, baseline),
        pt + cv::Point(text.width, -text.height),
        CV_RGB(255,0,0), CV_FILLED
    );

    cv::putText(im, label, pt, fontface, scale, CV_RGB(255,255,255), thickness, 8);
}

//************************for labelling purpose on
cv::Rec*******************************
// parameters:
//  img - input video frame
//  dst - resultant motion picture
//  args - optional parameters
static void  update_mhi( IplImage* img, IplImage* dst, int diff_threshold )
{
    double timestamp = (double)clock()/CLOCKS_PER_SEC; // get current time in seconds
    CvSize size = cvSize(img->width,img->height); // get current frame size
    int i, idx1 = last, idx2;
    IplImage* silh;
    CvSeq* seq;
    CvRect comp_rect;
    double count;
    double angle;
    CvPoint center;
    double magnitude;
    CvScalar color;

        Mat images(dst);   //convert IplImage to Mat iamge
    // allocate images at the beginning or
    // reallocate them if the frame size is changed
    if( !mhi || mhi->width != size.width || mhi->height != size.height ) {
        if( buf == 0 ) {
            buf = (IplImage**)malloc(N*sizeof(buf[0]));
            memset( buf, 0, N*sizeof(buf[0]));
        }

        for( i = 0; i < N; i++ ) {
            cvReleaseImage( &buf[i] );
            buf[i] = cvCreateImage( size, IPL_DEPTH_8U, 1 );
            cvZero( buf[i] );
        }
        cvReleaseImage( &mhi );
        cvReleaseImage( &orient );
        cvReleaseImage( &segmask );
        cvReleaseImage( &mask );

        mhi = cvCreateImage( size, IPL_DEPTH_32F, 1 );
        cvZero( mhi ); // clear MHI at the beginning
        orient = cvCreateImage( size, IPL_DEPTH_32F, 1 );
        segmask = cvCreateImage( size, IPL_DEPTH_32F, 1 );
        mask = cvCreateImage( size, IPL_DEPTH_8U, 1 );
    }

    cvCvtColor( img, buf[last], CV_BGR2GRAY ); // convert frame to grayscale

    idx2 = (last + 1) % N; // index of (last - (N-1))th frame
    last = idx2;

    silh = buf[idx2];
    cvAbsDiff( buf[idx1], buf[idx2], silh ); // get difference between frames

    cvThreshold( silh, silh, diff_threshold, 1, CV_THRESH_BINARY ); // and threshold
it
    cvUpdateMotionHistory( silh, mhi, timestamp, MHI_DURATION ); // update MHI

    // convert MHI to blue 8u image
    cvCvtScale( mhi, mask, 255./MHI_DURATION,
                (MHI_DURATION - timestamp)*255./MHI_DURATION );
    //cvZero( dst ); //updated 30032015, Monday, 539pm.
    cvCopy(img, dst, NULL);

        //cvMerge( mask, 0, 0, 0, dst );

    // calculate motion gradient orientation and valid orientation mask
    cvCalcMotionGradient( mhi, mask, orient, MAX_TIME_DELTA, MIN_TIME_DELTA, 3 );

    if( !storage )
        storage = cvCreateMemStorage(0);
    else
        cvClearMemStorage(storage);

    // segment motion: get sequence of motion components
    // segmask is marked motion components map. It is not used further
    seq = cvSegmentMotion( mhi, segmask, storage, timestamp, MAX_TIME_DELTA );

    // iterate through the motion components,
    // One more iteration (i == -1) corresponds to the whole image (global motion)
    for( i = -1; i < seq->total; i++ ) {

        if( i < 0 ) { // case of the whole image
            comp_rect = cvRect( 0, 0, size.width, size.height );
            color = CV_RGB(255,255,255);
            magnitude = 0;
        }
        else { // i-th motion component
            comp_rect = ((CvConnectedComp*)cvGetSeqElem( seq, i ))->rect;
            if( comp_rect.width + comp_rect.height < 100 ) // reject very small
components
                continue;
                       //putText(images, "testing", Point2f(x,y), FONT_HERSHEY_PLAIN,
3,  Scalar(255,255,255,255));
```

```
            color = CV_RGB(255,0,0);
            magnitude = 30;

        }

        // select component ROI
        cvSetImageROI( silh, comp_rect );
        cvSetImageROI( mhi, comp_rect );
        cvSetImageROI( orient, comp_rect );
        cvSetImageROI( mask, comp_rect );

        // calculate orientation
        angle = cvCalcGlobalOrientation( orient, mask, mhi, timestamp, MHI_DURATION);
        angle = 360.0 - angle;  // adjust for images with top-left origin
               if(angle>90 && angle<270){
                      set_label(images, comp_rect,  "right");
                     putText(images, "Object(s) crossing to right! ", Point2f(30,50),
FONT_HERSHEY_PLAIN, 1.5,  Scalar(255,255,255,255));
               }

               else if((angle>270 && angle<360)|| (angle<90 && angle>0)){
                      set_label(images, comp_rect,  "left");
                      putText(images, "Object(s) crossing to left!", Point2f(30,100),
FONT_HERSHEY_PLAIN, 1.5,  Scalar(255,255,255,255));
               }

               else{
                      set_label(images, comp_rect,  "No moving object!");}
        count = cvNorm( silh, 0, CV_L1, 0 ); // calculate number of points within
silhouette ROI

        cvResetImageROI( mhi );
        cvResetImageROI( orient );
        cvResetImageROI( mask );
        cvResetImageROI( silh );

        // check for the case of little motion
        if( count < comp_rect.width*comp_rect.height * 0.05 )
            continue;

        // draw a clock with arrow indicating the direction
        center = cvPoint( (comp_rect.x + comp_rect.width/2),
                          (comp_rect.y + comp_rect.height/2) );

        cvCircle( dst, center, cvRound(magnitude*1.2), color, 3, CV_AA, 0 );
        cvLine( dst, center, cvPoint( cvRound( center.x +
magnitude*cos(angle*CV_PI/180)),
                cvRound( center.y - magnitude*sin(angle*CV_PI/180))), color, 3, CV_AA,
0 );
    }
}


int main(int argc, char** argv)
{
    IplImage* motion = 0;
    CvCapture* capture = 0;
    help();

    if( argc == 1 || (argc == 2 && strlen(argv[1]) == 1 && isdigit(argv[1][0])))
        capture = cvCaptureFromCAM( argc == 2 ? argv[1][0] - '0' : 0 );
    else if( argc == 2 )
        capture = cvCaptureFromFile( argv[1] );

    if( capture )
    {
        cvNamedWindow( "Motion", 1 );
               //cvResizeWindow("Motion",500,500);

        for(;;)
        {
            IplImage* image = cvQueryFrame( capture );
            if( !image )
                break;

            if( !motion )
            {
                motion = cvCreateImage( cvSize(image->width,image->height), 8, 3 );
                cvZero( motion );
                //cvCopy(img, dst, NULL)


                              motion->origin = image->origin;
            }

            update_mhi( image, motion, 30 );
            cvShowImage( "Motion", motion );

            if( cvWaitKey(10) >= 0 )
                break;
        }
        cvReleaseCapture( &capture );
        cvDestroyWindow( "Motion" );
    }

    return 0;
}

#ifdef _EiC
main(1,"motempl.c");
#endif
```

Appendix 4: HSV Color Model Classification (Fruit.h)

```
#pragma once
#include <string>
#include <opencv\cv.h>
#include <opencv\highgui.h>
using namespace std;
using namespace cv;   //since declared, the following "cv::" can be removed.

class Fruit
{
public:
	Fruit(void);
	~Fruit(void);

	Fruit(string name);

	int getXPos();   //redefined the function
	void setXPos(int x);

	int getYPos();   //redefined the function
	void setYPos(int y);

	//cv::Scalar getHSVmin();
	//cv::Scalar getHSVmax();
	Scalar getHSVmin();
	Scalar getHSVmax();

	void setHSVmin(Scalar min);
	void setHSVmax(Scalar max);

	string getType(){ return type;}
	void setType(string t){ type = t;}

	Scalar getColour(){
	    return Colour;
	}

	void setColour(Scalar c){
		Colour = c;
	}

private:

	int xPos, yPos;
	string type;
	//cv::Scalar HSVmin, HSVmax;
	Scalar HSVmin, HSVmax;
	Scalar Colour;


};
```

Appendix 5: HSV Color Model Classification (Fruit.cpp (Main codes 1))

```
#include "Fruit.h"


Fruit::Fruit(void)
{
}

Fruit::Fruit(string name){

	setType(name);
	if(name=="Orange Obstacle"){

		setHSVmin(Scalar(5,163,117));
		setHSVmax(Scalar(23,231,256));

		setColour(Scalar(0,69,255));  //RGB value, in opencv, it is BGR
	}

	if(name=="Yellow Obstacle"){

		setHSVmin(Scalar(26,81,153));
		setHSVmax(Scalar(91,186,256));

		setColour(Scalar(0,255,255));  //RGB value, in opencv, it is BGR
	}

	if(name=="Blue Obstacle"){

		setHSVmin(Scalar(100, 84, 68));
		setHSVmax(Scalar(128,256,168));

		setColour(Scalar(255,0,0));  //RGB value, in opencv, it is BGR
	}



}



Fruit::~Fruit(void)
{
}

int Fruit::getXPos(){

	return Fruit::xPos;


}

void Fruit::setXPos (int x) {

	Fruit::xPos = x;
```

```
        xPos = x;
}

int Fruit::getYPos(){

        return Fruit::yPos;


}

void Fruit::setYPos (int y) {

        Fruit::yPos = y;

        yPos = y;
}

Scalar Fruit::getHSVmin(){

        return Fruit::HSVmin;

        }
Scalar Fruit::getHSVmax(){

        return Fruit::HSVmax;
}

void Fruit::setHSVmin(Scalar min){

        Fruit::HSVmin = min;

}

void Fruit::setHSVmax(Scalar max){

        Fruit::HSVmax = max;

}
```

Appendix 6: HSV Color Model Classification (Main codes 2)

```
#include <sstream>
#include <string>
#include <iostream>
#include <vector>
//#include <opencv\highgui.h>// transfered to Fruit.h
//#include <opencv\cv.h>   //since this is already declared in Fruit.h
#include "Multiple_Object_Tracking\Fruit.h"
//#include "Fruit.h"


//using namespace cv;  //transfered to the Fruit.h
//initial min and max HSV filter values.
//these will be changed using trackbars
int H_MIN = 0;
int H_MAX = 256;
int S_MIN = 0;
int S_MAX = 256;
int V_MIN = 0;
int V_MAX = 256;
//default capture width and height
const int FRAME_WIDTH = 640;
const int FRAME_HEIGHT = 480;
//max number of objects to be detected in frame
const int MAX_NUM_OBJECTS=50;
//minimum and maximum object area
//const int MIN_OBJECT_AREA = 40*40;
const int MIN_OBJECT_AREA = 20*20;
const int MAX_OBJECT_AREA = FRAME_HEIGHT*FRAME_WIDTH/1.5;
//names that will appear at the top of each window
const string windowName = "Original Image";
const string windowName1 = "HSV Image";
const string windowName2 = "Thresholded Image";
const string windowName3 = "After Morphological Operations";
const string trackbarWindowName = "Trackbars";
void on_trackbar( int, void* )
{//This function gets called whenever a
        // trackbar position is changed

}
string intToString(int number){

        std::stringstream ss;
        ss << number;
        return ss.str();
}
void createTrackbars(){
        //create window for trackbars


        namedWindow(trackbarWindowName,0);
        //create memory to store trackbar name on window
        char TrackbarName[50];
        sprintf( TrackbarName, "H_MIN", H_MIN);
        sprintf( TrackbarName, "H_MAX", H_MAX);
        sprintf( TrackbarName, "S_MIN", S_MIN);
        sprintf( TrackbarName, "S_MAX", S_MAX);
        sprintf( TrackbarName, "V_MIN", V_MIN);
        sprintf( TrackbarName, "V_MAX", V_MAX);
        //create trackbars and insert them into window
        //3 parameters are: the address of the variable that is changing when the 
trackbar is moved(eg.H_LOW),
        //the max value the trackbar can move (eg. H_HIGH), 
        //and the function that is called whenever the trackbar is moved(eg. 
on_trackbar)
        //                                  ---->    ---->     ---->      
        createTrackbar( "H_MIN", trackbarWindowName, &H_MIN, H_MAX, on_trackbar );
        createTrackbar( "H_MAX", trackbarWindowName, &H_MAX, H_MAX, on_trackbar );
        createTrackbar( "S_MIN", trackbarWindowName, &S_MIN, S_MAX, on_trackbar );
        createTrackbar( "S_MAX", trackbarWindowName, &S_MAX, S_MAX, on_trackbar );
        createTrackbar( "V_MIN", trackbarWindowName, &V_MIN, V_MAX, on_trackbar );
        createTrackbar( "V_MAX", trackbarWindowName, &V_MAX, V_MAX, on_trackbar );


}
//void drawObject(int x,int y,Mat &frame){
void drawObject(vector<Fruit> theFruits,Mat &frame){

        //theFruit.getXPos()
        for(int i=0; i<theFruits.size();i++){

                //theFruits.at(i).getXPos()

        cv::circle(frame,cv::Point(theFruits.at(i).getXPos(),theFruits.at(i).getYPos()
),10,cv::Scalar(0,0,255));
        cv::putText(frame,intToString(theFruits.at(i).getXPos())+ " , " + 
intToString(theFruits.at(i).getYPos()),cv::Point(theFruits.at(i).getXPos(),theFruits.a
t(i).getYPos()+20),1,1,Scalar(0,255,0));
        cv::putText(frame, 
theFruits.at(i).getType(),cv::Point(theFruits.at(i).getXPos(),theFruits.at(i).getYPos(
)-30),1,2, theFruits.at(i).getColour());


        //cv::putText(frame,intToString(x)+ " , " + 
intToString(y),cv::Point(x,y+20),1,1,Scalar(0,255,0));
        }
        }
void morphOps(Mat &thresh){

        //create structuring element that will be used to "dilate" and "erode" image.
        //the element chosen here is a 3px by 3px rectangle

        Mat erodeElement = getStructuringElement( MORPH_RECT,Size(3,3));
        //dilate with larger element so make sure object is nicely visible
        Mat dilateElement = getStructuringElement( MORPH_RECT,Size(8,8));

        erode(thresh,thresh,erodeElement);
        erode(thresh,thresh,erodeElement);


        dilate(thresh,thresh,dilateElement);
        dilate(thresh,thresh,dilateElement);
}
void trackFilteredObject(Mat threshold,Mat HSV, Mat &cameraFeed){

	//int x,y;

	//Fruit apple;

	vector <Fruit> apples;

	Mat temp;
	threshold.copyTo(temp);
	//these two vectors needed for output of findContours
	vector< vector<Point> > contours;
	vector<Vec4i> hierarchy;
	//find contours of filtered image using openCV findContours function
	findContours(temp,contours,hierarchy,CV_RETR_CCOMP,CV_CHAIN_APPROX_SIMPLE );
	//use moments method to find our filtered object
	double refArea = 0;
	bool objectFound = false;
	if (hierarchy.size() > 0) {
		int numObjects = hierarchy.size();
		//if number of objects greater than MAX_NUM_OBJECTS we have a noisy
filter
		if(numObjects<MAX_NUM_OBJECTS){
			for (int index = 0; index >= 0; index = hierarchy[index][0]) {

				Moments moment = moments((cv::Mat)contours[index]);
				double area = moment.m00;

				if(area>MIN_OBJECT_AREA){

					Fruit apple;
					apple.setXPos(moment.m10/area);
					apple.setYPos(moment.m01/area);
					//x = moment.m10/area;
					//y = moment.m01/area;

					//apple.setX(x);
					//apple.setY(y);
					apples.push_back(apple);

					objectFound = true;

				}else objectFound = false;


			}
			//let user know you found an object
			if(objectFound ==true){
				//draw object location on screen
				//drawObject(x,y,cameraFeed);}
				drawObject(apples,cameraFeed);}

		}else putText(cameraFeed,"TOO MUCH NOISE! ADJUST
FILTER",Point(0,50),1,2,Scalar(0,0,255),2);
	}
}
```

```
void trackFilteredObject(Fruit theFruit,Mat threshold,Mat HSV, Mat &cameraFeed){

        //int x,y;

        //Fruit apple;

        vector <Fruit> apples;
        vector <Fruit> bananas;
        vector <Fruit> cherrys;

        Mat temp;
        threshold.copyTo(temp);
        //these two vectors needed for output of findContours
        vector< vector<Point> > contours;
        vector<Vec4i> hierarchy;
        //find contours of filtered image using openCV findContours function
        findContours(temp,contours,hierarchy,CV_RETR_CCOMP,CV_CHAIN_APPROX_SIMPLE );
        //use moments method to find our filtered object
        double refArea = 0;
        bool objectFound = false;
        if (hierarchy.size() > 0) {
                int numObjects = hierarchy.size();
                //if number of objects greater than MAX_NUM_OBJECTS we have a noisy
filter
                if(numObjects<MAX_NUM_OBJECTS){
                        for (int index = 0; index >= 0; index = hierarchy[index][0]) {

                                Moments moment = moments((cv::Mat)contours[index]);
                                double area = moment.m00;


                                if(area>MIN_OBJECT_AREA){

                                        Fruit apple;
                                        apple.setXPos(moment.m10/area);
                                        apple.setYPos(moment.m01/area);
                                        apple.setType(theFruit.getType());
                                        apple.setColour(theFruit.getColour());

                                        Fruit banana;
                                        banana.setXPos(moment.m10/area);
                                        banana.setYPos(moment.m01/area);
                                        banana.setType(theFruit.getType());
                                        banana.setColour(theFruit.getColour());

                                        Fruit cherry;
                                        cherry.setXPos(moment.m10/area);
                                        cherry.setYPos(moment.m01/area);
                                        cherry.setType(theFruit.getType());
                                        cherry.setColour(theFruit.getColour());

                                        //x = moment.m10/area;
                                        //y = moment.m01/area;

                                        //apple.setX(x);
                                        //apple.setY(y);
                                        apples.push_back(apple);
                                        bananas.push_back(banana);
```

```
					cherrys.push_back(cherry);

					objectFound = true;

				}else objectFound = false;


			}
			//let user know you found an object
			if(objectFound ==true){
				//draw object location on screen
				//drawObject(x,y,cameraFeed);}
				drawObject(apples,cameraFeed);
			 drawObject(bananas,cameraFeed);
				drawObject(cherrys,cameraFeed);


			}

		}else putText(cameraFeed,"TOO MUCH NOISE! ADJUST 
FILTER",Point(0,50),1,2,Scalar(0,0,255),2);
	}
}
int main(int argc, char* argv[])
{
	//if we would like to calibrate our filter values, set to true.
	//bool calibrationMode = true;    //to calibrate the colour of the object
	bool calibrationMode = false; // to hardcode the colour of the object

	//Matrix to store each frame of the webcam feed
	Mat cameraFeed;
	Mat threshold;
	Mat HSV;

	if(calibrationMode){
		//create slider bars for HSV filtering
		createTrackbars();
	}
	//video capture object to acquire webcam feed
	VideoCapture capture;
	//open capture object at location zero (default location for webcam)
	capture.open(0);
	//set height and width of capture frame
	capture.set(CV_CAP_PROP_FRAME_WIDTH,FRAME_WIDTH);
	capture.set(CV_CAP_PROP_FRAME_HEIGHT,FRAME_HEIGHT);
	//start an infinite loop where webcam feed is copied to cameraFeed matrix
	//all of our operations will be performed within this loop
	while(1){
		//store image to matrix
		capture.read(cameraFeed);
		//convert frame from BGR to HSV colorspace
		cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);

		if(calibrationMode==true){
		//if in calibration mode, we track objects based on the HSV slider 
values.
		cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);
```

```
        inRange(HSV,Scalar(H_MIN,S_MIN,V_MIN),Scalar(H_MAX,S_MAX,V_MAX),threshold);
                morphOps(threshold);
                imshow(windowName2,threshold);
                trackFilteredObject(threshold,HSV,cameraFeed);
                }else{

        Fruit apple("Orange Obstacle"), banana("Yellow Obstacle"), cherry("Blue
Obstacle");

                //apple.setHSVmin(Scalar(0,0,0));
                //apple.setHSVmax(Scalar(255,255,255));

                apple.setHSVmin(Scalar(5,163,117));
                apple.setHSVmax(Scalar(23,231,256));

                banana.setHSVmin(Scalar(26,81,153));
                banana.setHSVmax(Scalar(91,186,256));

                cherry.setHSVmin(Scalar(100, 84, 68));
                cherry.setHSVmax(Scalar(128,256,168));



                cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);
                inRange(HSV,apple.getHSVmin(),apple.getHSVmax(),threshold);
                morphOps(threshold);
                //imshow(windowName2,threshold);
                trackFilteredObject(apple,threshold,HSV,cameraFeed);

                cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);
                inRange(HSV,banana.getHSVmin(),banana.getHSVmax(),threshold);
                morphOps(threshold);
                //imshow(windowName2,threshold);
                trackFilteredObject(banana,threshold,HSV,cameraFeed);

                cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);
                inRange(HSV,cherry.getHSVmin(),cherry.getHSVmax(),threshold);
                morphOps(threshold);
                //imshow(windowName2,threshold);
                trackFilteredObject(cherry,threshold,HSV,cameraFeed);


                }

                //show frames
                //imshow(windowName2,threshold);

                imshow(windowName,cameraFeed);
                //imshow(windowName1,HSV);


                //delay 30ms so that screen can refresh.
                //image will not appear without this waitKey() command
                waitKey(30);
        }
        return 0;
}
```

Appendix 7: CMakeLists.txt (For the compilation of Motion Templates codes)

```
cmake_minimum_required (VERSION 2.8)

project (raspicam_test)

SET( CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH}
/usr/local/lib/cmake/)

find_package(raspicam REQUIRED)

find_package(OpenCV)

IF  ( OpenCV_FOUND AND raspicam_CV_FOUND)

MESSAGE(STATUS "COMPILING OPENCV TESTS")

add_executable (videotest1  videotest1.cpp)

target_link_libraries (videotest1   ${raspicam_CV_LIBS})

ELSE()

MESSAGE(FATAL_ERROR "OPENCV NOT FOUND IN YOUR SYSTEM")

ENDIF()
```