TARGET TRACKING FOR AUDIO BEAMING SYSTEM

SUBMITTED

BY

RISAL MUHAMMED KIZHAKEL NASAR

SCHOOL OF ELECTRICAL & ELECTRONIC ENGINEERING

A final year project report

presented to

Nanyang Technological University

in partial fulfillment of the requirements for the

Degree of Bachelor of Engineering (Electrical & Electronic Engineering)

Nanyang Technological University

Academic Year 2009/2010

Contents

Abstract

The Audio Beaming System developed in the Digital Signal Processing Laboratory at Nanyang Technological University is an innovative technology providing directional sound. This project serves as a value addition to the Audio Beaming System by designing and building a system to track a target as it moves around and thereby making it possible to beam directional audio continuously at the target.

This project involves making a Target Tracking Application using a web camera, a host computer, a motorized pan and tilt system and a microcontroller. The application takes in live video input from the web camera and run computer vision codes on it. The computer vision codes analyzing the live video will track potential targets on the scene. The corresponding position data of the target will be handed over to the microcontroller. The microcontroller uses this data to control the pan and tilt system. The pan and tilt system will position the Audio Beaming System on the target. Only the moving target on the scene will be able to hear the audio being beamed.

The Audio Beaming System with Target Tracking has tremendous applications in the gaming industry. Other areas of application are in marketing and advertisement, customer support, museums, crowd control, etc.

Acknowledgement

The author expresses his deepest gratitude to the following people for offering their kind help and support which made the author's Final Year Project (FYP) a highly enriching and enjoyable learning experience.

Associate Professor Gan Woon Seng

The author is highly grateful to Associate Professor Gan Woon Seng, the author's FYP co-supervisor, for the tremendous motivation, support, advice and direction given to the author throughout the course of the project. Professor Gan motivated the author to do his best and also provided various platforms to showcase the author's project, some of them being the iJam session organized by the Media Development Authority of Singapore and the Discover Engineering @ NTU session held at the NTU Open House.

Associate Professor Chong Yong Kim

The author would like to thank Professor Chong Yong Kim, the author's FYP supervisor, for the support and direction given to the author throughout the project.

Mr. Tan Ee Leng, Research Associate

The author expresses his sincere gratitude to Mr. Tan Ee Leng for the ideas he suggested all through the project. Mr. Ee Leng was very approachable, friendly and vibrant. He always helped the author by brainstorming ideas to be incorporated into the project.

Mr. Yeo Sung Kheng & Mr. Ong Say Cheng, Technical Staff, DSP Lab

The author would like to thank Mr. Yeo and Mr. Ong for their kind support in providing the hardware necessary for the implementation of the applications developed by the author. They were approachable and very friendly and generated a feeling of welcome in the laboratory which was instrumental in the smooth execution of the project.

All the staff and fellow FYP students in DSP Lab

The author thanks the staff and FYP students in the DSP lab for the help and cooperation provided during the course of the project. Special thanks to Mr. Ji Wei, Dr. Ji Peifeng, Mr. Reuben Johannas and Mr. Wang Tong Wei for their support, especially during the project exhibitions.

List of Figures

List of Tables

1. Introduction

1.1 Audio Beaming System

The Audio Beaming System was developed in the Digital Signal Processing Laboratory at Nanyang Technological University by Professor Gan Woon Seng and his group of researchers.

Using the Audio Beaming system, audio sound can be delivered to desired locations with precision. This technique uses an ultrasound carrier to deliver audible sound to a private listening space or create a virtual speaker.

The audio message is first modulated with the ultrasonic carrier and audible sound is generated in air as a byproduct of nonlinear interaction of ultrasound with air. Figure 1.1 shows the propagation of directional sound in air.

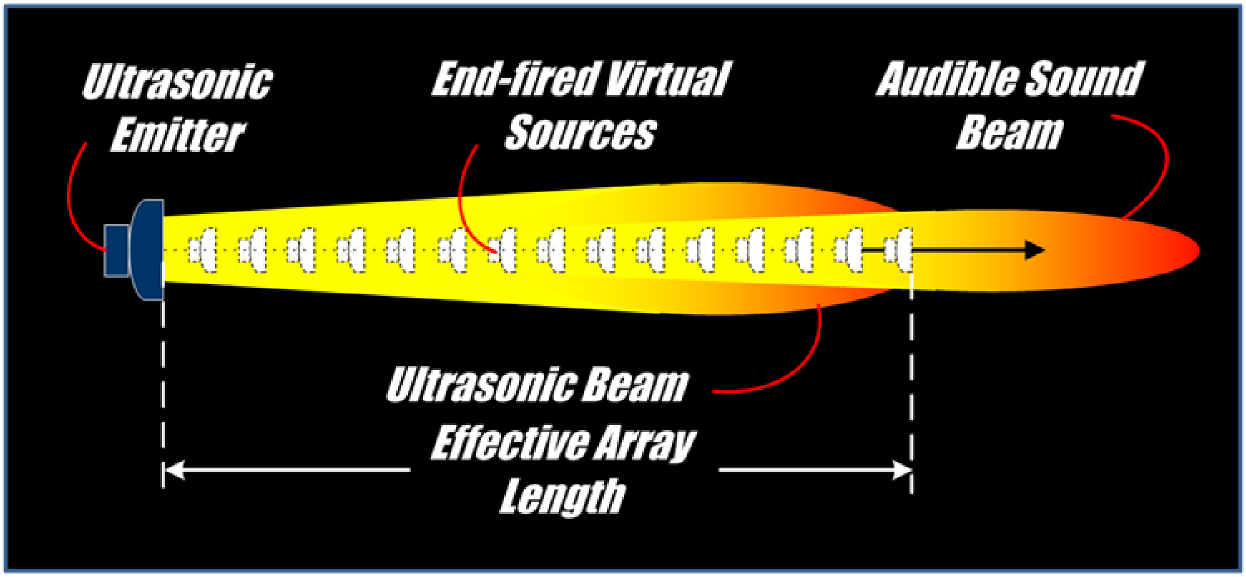


Figure 1. 1: Directional Audio Propagation

Due to the high directivity (or short wavelength) of the ultrasound wave, it is able to carry the audible sound in a narrow beam, just like a beam of light. A comparison between an audio beam and a conventional loudspeaker is shown in Figure 1.2.

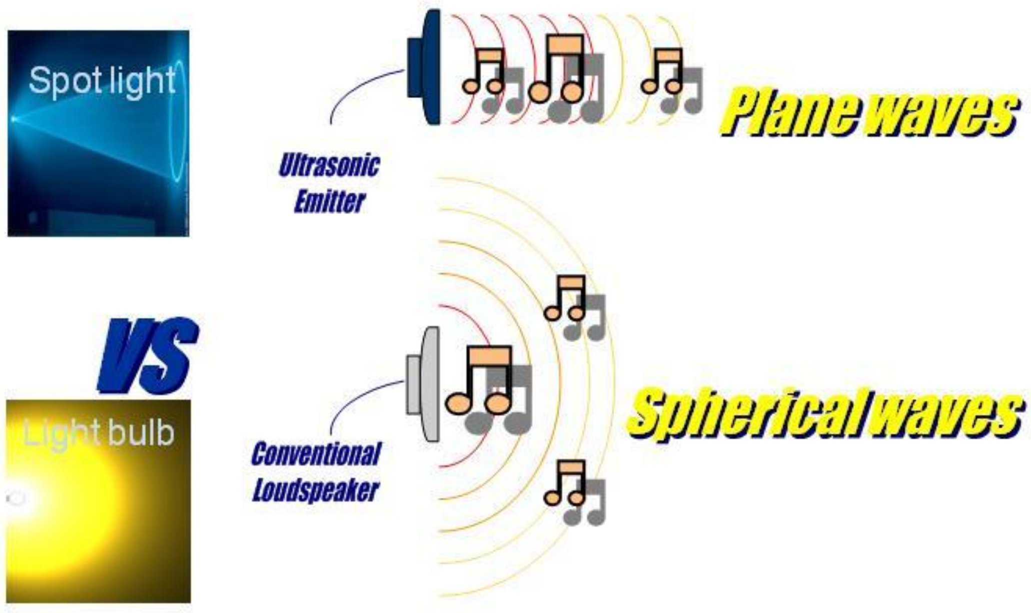


Figure 1. 2: Distinctive Difference Between Loudspeaker and ABS

Two types of directional audio speakers are used in this project. They are the Audio Beaming System (ABS) developed in the Digital Signal Processing Laboratory at Nanyang Technological University and the commercially available HSS speaker from American Technology Corporation (ATC). Figure 1.3 shows the Audio Beaming System (ABS). Figure 1.4 shows the HSS speaker.

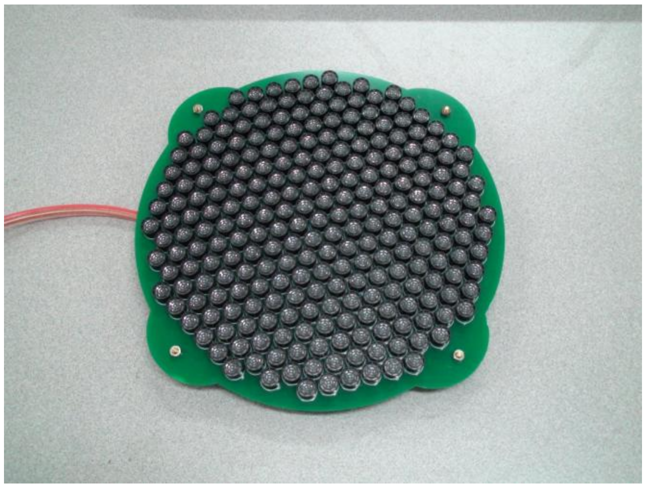


Figure 1. 3: Audio Beaming System (ABS)

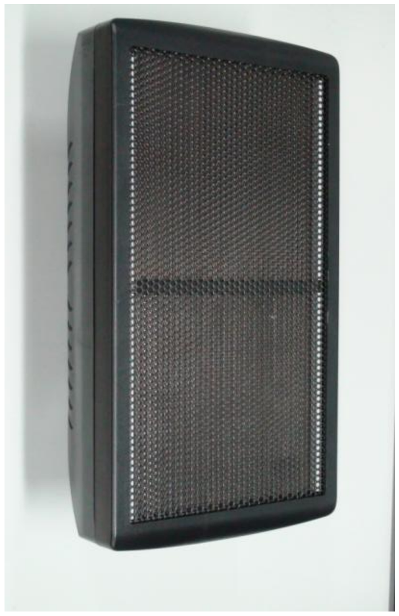


Figure 1. 4: HSS Directional Audio Speaker

1.2 Project Requirements

The project requirement entitled on the author was to build a Target Tracking System. The Target Tracking System should be able to detect targets (eg: a human face) in front of the ABS using a camera and to robustly track the target as it moves around. The ABS will be mounted on the Target Tracking System and will direct the audio towards the moving target on the scene.

1.3 Target Tracking System – "Blue Print"

The target tracking system consists of a web camera, target tracking program running on a host computer, a microcontroller and a motorized pan and tilt system.

The target tracking program takes in video input from the web camera and executes a video processing algorithm on it. This algorithm analyzes the live video and tracks targets in the scene. The corresponding positioning data from the algorithm is sent to the microcontroller. Using the positioning data, the microcontroller produces the control signals to drive the motorized pan and tilt system.

By mounting the Audio Beaming System (ABS) on the pan and tilt mechanism, the audio beam can be delivered to a moving target in the scene.

Figure 1.5 shows the functional block diagram of the Target Tracking System.

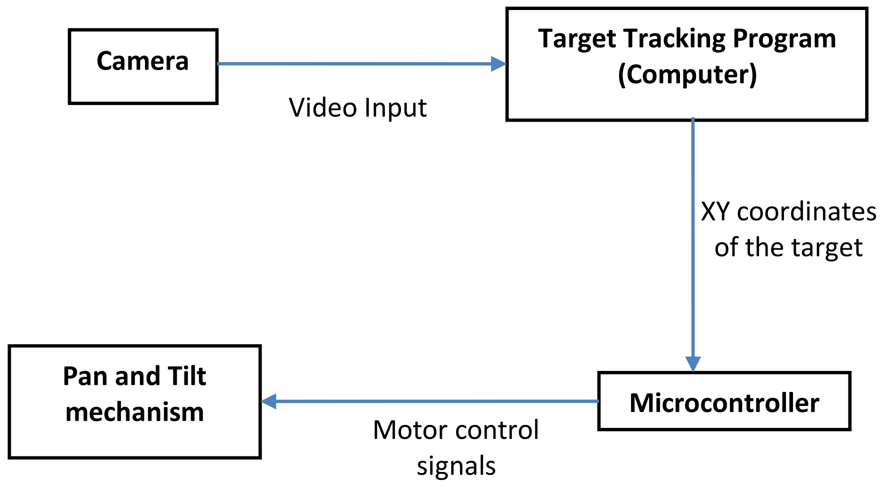


Figure 1. 5: Functional Block Diagram of the Target Tracking System

2. Potential Applications of the Target Tracking with Audio Beaming System

2.1 Gaming

The Target Tracking System with the Audio Beaming System can be using in the field of gaming. In many multiplayer games, each of the players is engaged in a different environment and requires a completely different set of audio cues. Using the Target Tracking with Audio Beam System, each and every player in the same room can be provided with a completely different set of private audio cues audible only to them.

Figure 2.1 shows a first person shooter game which is an example of games in which private audio cues can be utilized. [1]

Figure 2. 1: An Xbox first person shooter game

Project Natal has been initiated by Microsoft to provide a "controller-free gaming and entertainment experience". By integrating the Target Tracking and Audio Beaming System with Project Natal, a completely new immersive sound experience for gaming and entertainment can be created.

Figure 2.2 shows the Project Natal Logo. Figure 2.3 shows a basic idea of how the Target Tracking and Audio Beaming System can be integrated with Project Natal. [2][3]

Figure 2. 2: Project Natal Logo (Microsoft)

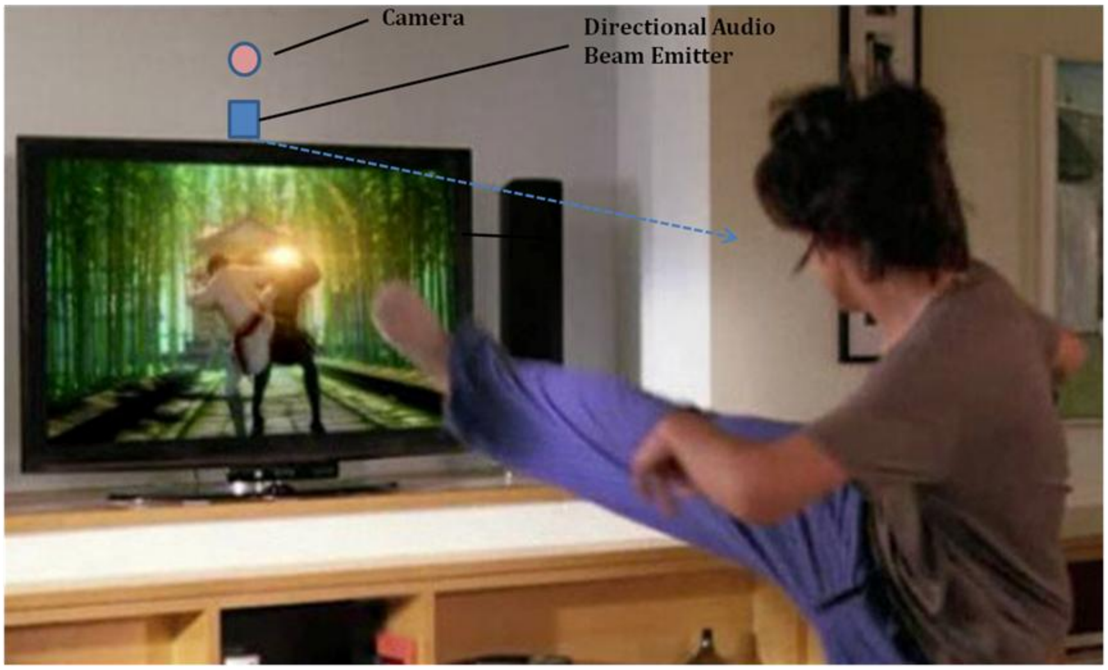


Figure 2. 3: Integrating Target Tracking and Audio Beaming System with Project Natal

2.2 Marketing and Advertisement

The Audio Beaming System with Target Tracking can be employed in the field of marketing and advertisement. In a shop, a customer checking out a particular product can be given the extra information which will help him to assess the product in a better manner. The information is only beamed to the interested customer and so, other bystanders would not find the marketing as a nuisance. Figure 2.4 shows how the Target Tracking and Audio Beaming System can be used for advertising a product. [4]

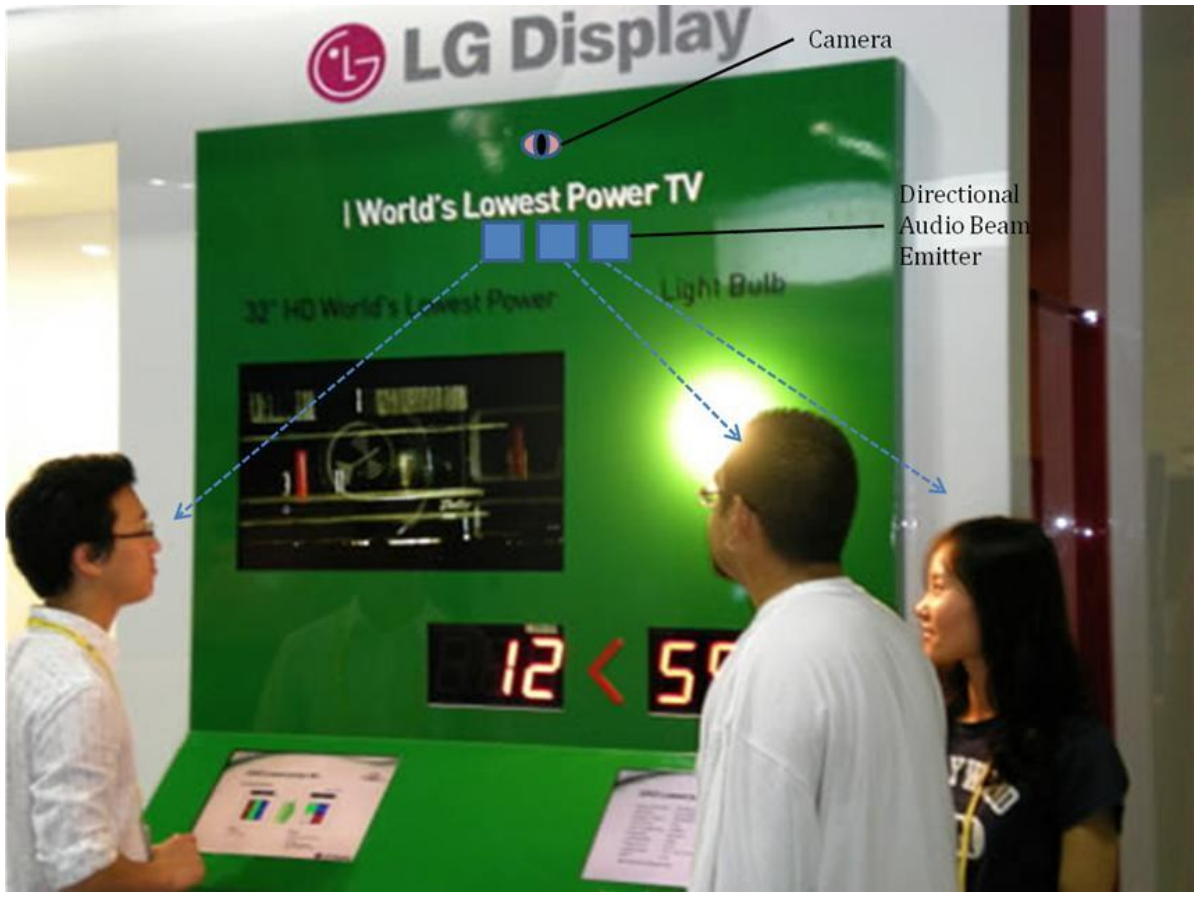


Figure 2. 4: Target Tracking and Audio Beaming System used for Advertising a product

2.3 Museum

Historical information regarding a specific artifact can be beamed to a group of visitors standing in front of the artifact. This would help visitors to navigate the museum without a guide accompanying them. Figure 2.5 shows the Target Tracking and Audio Beaming System used in museums. [5]

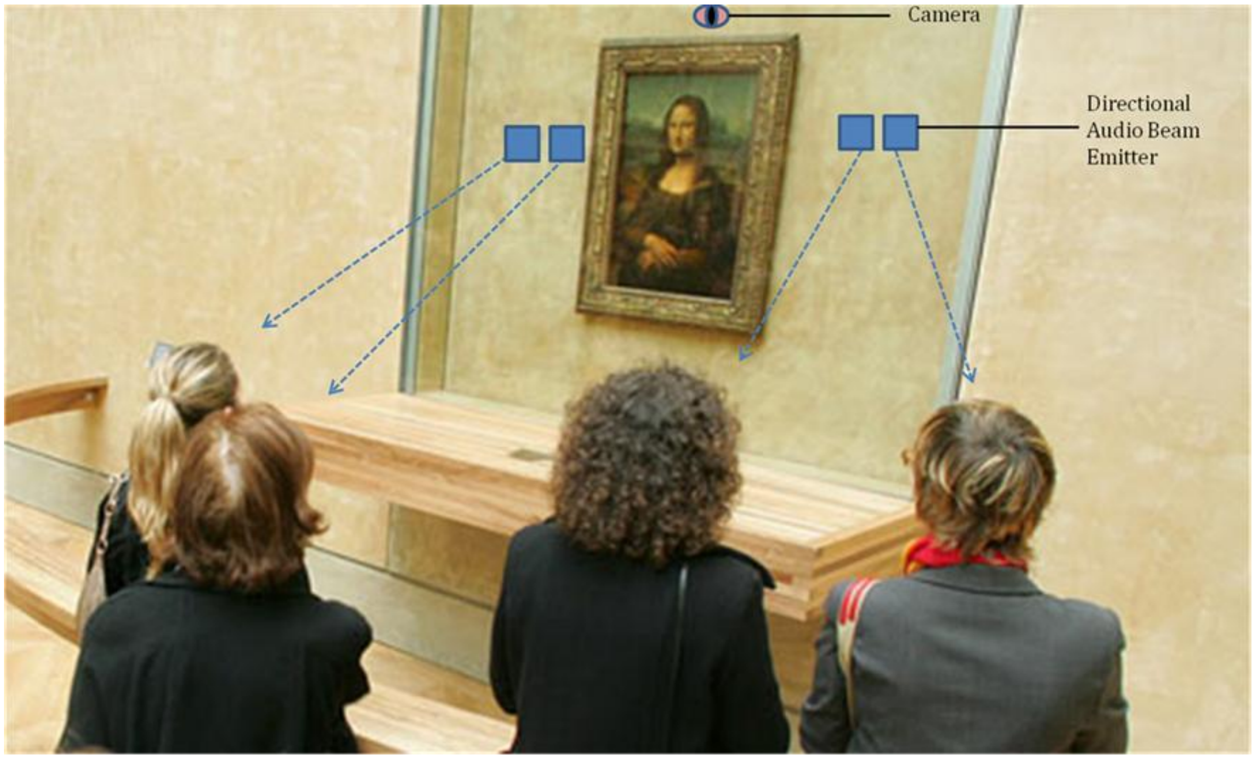


Figure 2. 5: Target Tracking and Audio Beaming System used in a Museum

2.4 Crowd Control

Directional Audio Beams with Target Tracking System can also be used to control crowd in public places. For an example, in an MRT station, audio beam can be used to warn passengers crossing the yellow line. This could help avoid accidents.

3. Selection of Target Tracking Program Development Platform and Camera

3.1 OpenCV Library

OpenCV is a computer vision library originally developed by Intel. It is written in C/C++. It is free for commercial and research use under the open source BSD license. It is cross platform and can be used with Linux or Windows operating systems. It includes various computer vision functions and provides an abstract layer which can be used to develop computer vision programs. It mainly focuses on real-time image processing. [6]

3.2 Choosing the Operating System and Compiler Platform for Application Development

OpenCV can be installed in Linux, Windows or Mac Operating Systems. It will work with C/C++ compilers like Visual Studio 2005 and 2008 editions and MinGW on Windows, GCC 4.x on Linux, MacOSX and other Unix like systems. [7]

Developing code in the Linux platform using the OpenCV library will require building the OpenCV library as well as elaborate steps to include the path to OpenCV files in every program that is being made.

The Microsoft Visual C++ (MVC++) platform is included with the Microsoft Visual Studio Suite in the Windows Operating System. Under the MVC++, path to the OpenCV files can be easily included from the Project options selection process and it is less tedious. Windows XP was chosen as the development Operating System and Microsoft Visual Studio 2005 as the C/C++ compiler platform. Since MSVC++ is a stable platform and access to resources is easily available, it proved to be a better platform to work with.

Programs can be written in VC++ utilizing the computer vision functions provided by the OpenCV Library. VC++ also gives access to the Microsoft Foundation Class (MFC) Library. MFC is a library that wraps portions of the windows Application Programming Interface (API) in C++ classes, including functionality that enables them to use the default application framework. Classes are defined for mainly the handle-managed Windows objects and also for predefined windows and common controls. [8]

The snapshots showing the inclusion of the path to the OpenCV files are shown in Appendix A.

3.3 OpenCV version selection

Initially development was conducted using OpenCV 1.0. But using this version, the camera resolution remains at the default value (640 X 480). The resolution could not be controlled from the program itself. The newer version of OpenCV which is 1.1 was installed to solve the problem. With OpenCV1.1, resolutions such as 1280 X 720 and 1600 X 900 can be set directly from the program. But the camera frame rate still cannot be set from the program since it is not implemented even in version 1.1 of OpenCV for Windows. Given next is the code to set camera capture resolution from the program.

```
// set video properties
cvSetCaptureProperty( capture, CV_CAP_PROP_FRAME_WIDTH, 1600 );
cvSetCaptureProperty( capture, CV_CAP_PROP_FRAME_HEIGHT, 900 );
cvSetCaptureProperty( capture, CV_CAP_PROP_FPS, 30);
```

The 'Visual C++ redistribute package for x86' needed to be installed for the working of OpenCV 1.1 since it otherwise gives the error "cv110.dll could not be found". [9]

3.4 Selection of Web Camera

A web camera will take as input the real-time video of the area where the target is supposed to be present. Some of the specifications of the web camera are:

- Compatible with OpenCV
- High video resolution
- Ability to adjust settings when lighting change to provide good video quality

The two web cameras which were used in the project are the Logitech QuickCam Orbit AF and the Logitech Webcam C905. They provide the following functionalities:

- ➢ 2 Megapixel Sensor
- ➢ Auto Focus
- ➢ HD Video
- ➢ Ultra-Wide Angle Lens
- ➢ Right Light technology (Automatically adjusts with changes in lighting)
- ➢ Carl Zeiss optics (Sharp images)
- ➢ USB connection

Figure 3. 1: Logitech QuickCam Orbit AF

Figure 3. 2: Logitech Webcam C905

The Logitech Webcam C905 is the preferred web camera for the project because of its small size and better video quality.

3.5 Host Computer

A host computer is connected to the web camera via USB port. It takes in the video input and run computer vision codes on it.

4. Exploring various Image Processing Algorithms in OpenCV

4.1 Tracking Algorithm Based on Camshift

OpenCV includes a sample face tracking program which uses Camshift algorithm to follow a specific target in the video frame. Tracking is based on the color representation of the target. The four steps in Camshift are : [10]

1. Create a color histogram to represent the face
2. Calculate a "face probability" for each pixel in the incoming video frames
3. Shift the location of the face ellipse in each video frame
4. Calculate the size and angle of the ellipse covering the face

4.1.1 Advantages

Camshift tracking is a fast tracking algorithm. It can resize the tracking ellipse when the target moves closer to or away from the camera.

4.1.2 Disadvantages

The tracking is based on color features. So the tracking is not efficient enough to track a target that is having similar color tones to materials in the background.

The Figures 4.1 and 4.2 show Camshift algorithm applied for face tracking. The pictures clearly show how the tracker can falsely track an arm (instead of the face) having similar color tone to the face, even when the face was the initial target selected.

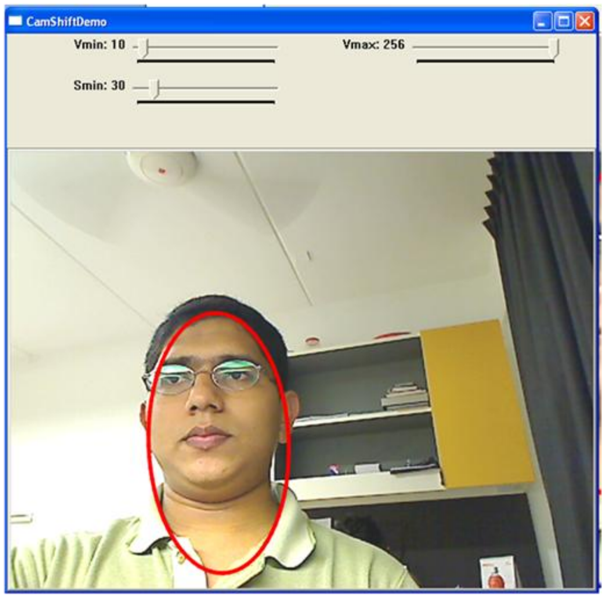


Figure 4. 1: Camshift Tracking 1

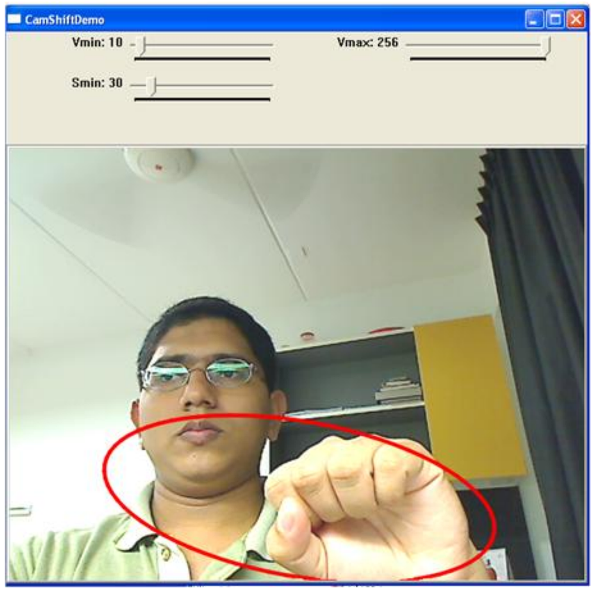


Figure 4. 2: Camshift Tracking 2

4.2 Background Subtraction and Blob Tracking (Video Surveillance)

The OpenCV Video Surveillance facility, also called Blob Tracker is a facility intended to track moving foreground objects against a relatively static background. Conceptually it consists of a three-stage video processing pipeline: [11]

1. A foreground/background discriminator which labels each pixel as either foreground or background
2. A blob detector which groups adjacent "foreground" pixels into blobs, flood-fill style
3. A blob tracker which assigns ID numbers to blobs and tracks their motion frame-to-frame

Almost all the CPU time is devoted to the first stage.

4.2.1 Disadvantages

The execution is computationally very intensive especially when the video resolution is high. The background is assumed to be static for the operation of the algorithm which may not be the case always. Even small changes in lighting affect the performance of the foreground/background discriminator. Moreover, this project involves a continuously moving camera thereby resulting in continuously changing background.

The figures 4.3 and 4.4 show the Blob Tracking as well as Background Subtraction taken during one of the experiments.

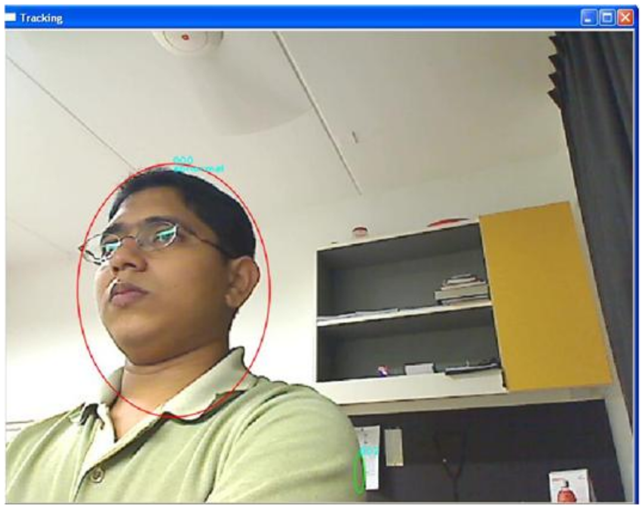


Figure 4. 3: Blob Tracking

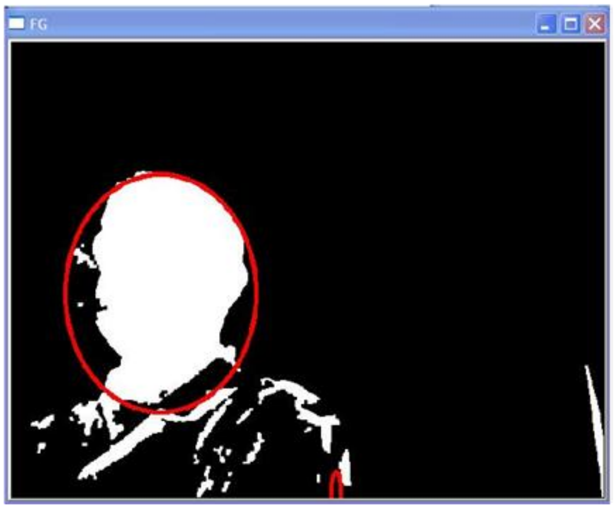


Figure 4. 4: Background Subtraction

4.3 Optical Flow and Motion Detection

Optical Flow is the pattern of apparent motion of objects, surfaces and edges in a visual scene caused by the relative motion between the camera and the scene. Optical Flow techniques can be used for detecting the motion of a particular point in a video frame. [12]

4.3.1 Disadvantages

Optical Flow techniques only detect the motion of a particular point on the video frame, but are not able to detect objects as such. Objects moving in front of the target can drag the Optical Flow points together with it. This is demonstrated in the figures 4.5 and 4.6.

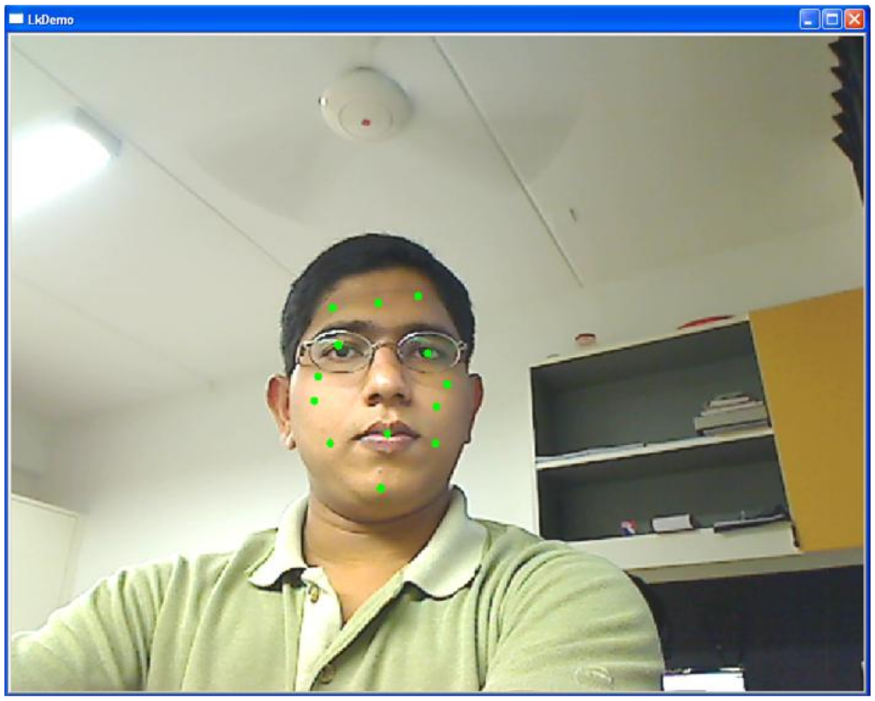


Figure 4. 5: Motion Tracking using Optical Flow 1

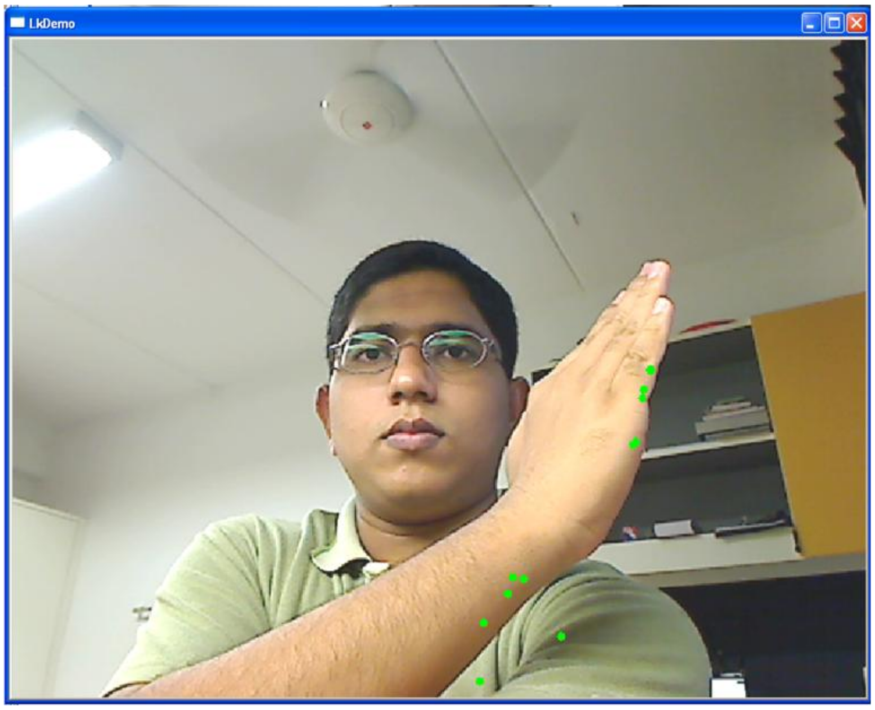


Figure 4. 6: Motion Tracking using Optical Flow 2

4.4 Face Recognition using Eigenfaces or Principal Component Analysis (PCA)

Face Recognition can be performed using the method of Principal Component Analysis (PCA). Principal component analysis (PCA), based on information theory concepts, seeks a computational model that best describes a face by extracting the most relevant information contained in that face. The Eigenfaces approach is a PCA method, in which a small set of characteristic pictures, are used to describe the variation between face images. The goal is to find the eigenvectors (eigenfaces) of the covariance matrix of the distribution, spanned by training a set of face images. Later, every face image is represented by a linear combination of these eigenvectors. Recognition is performed by projecting a new image onto the subspace spanned by the eigenfaces and then classifying the face by comparing its position in the face space with the positions of known individuals. [13][14]

Many pre-processing steps are to be performed in order to utilize the PCA method for recognition: [13]

1. Face detection and extraction
2. Illumination Normalization
3. Scaling

4.4.1 Advantages

PCA is fast and it only requires lesser amount of memory.

4.4.2 Disadvantages

1. Tilt variant: When the faces are tilted or transformed by muscle movement, PCA would not recognize the face
2. Scale variant: Same faces which are scaled are difficult to be recognized
3. Background variant: Same faces may not be recognized under different backgrounds
4. Lighting variant: If the lighting intensity changes, the face would not be recognized accurately

It is highly probable that the lighting conditions, the scale of the face, the facial expressions and background of the video frame are susceptible to changes. So Face Recognition using PCA would not add much of a value to the project compared to the computational costs which would be incurred in implementing the algorithm.

4.5 Skin Extraction

Extracting the skin area within an image would give higher probability of tracking down a human target face. Skin extraction would be possible based on a decision rule which discriminates between skin and non-skin pixels based on color components. In order to obtain adequate distinction between skin and non-skin regions, color transformations effectively separating luminance from chrominance is needed. [15]

In this project, the HSV and YCbCr color spaces are utilized in extracting the skin regions in a video image.

4.5.1 HSV (Hue, Saturation, Value) Color Space

HSV color space is based on the human color perception. The Value component is related to the color luminance. Hue is generally related to the wavelength of a light. Saturation is a component that measures the “colorfulness” in HSV space. The intuitiveness of the color space components and explicit discrimination between luminance and chrominance properties made this color space popular for skin extraction. [15]

$$H = \begin{cases} 2 - \cos^{-1}\left\{ \dfrac{[(R-G)+(R-B)]}{2\sqrt{(R-G)^2+(R-B)(G-B)}} \right. , B > G \\ \cos^{-1}\left\{ \dfrac{[(R-G)+(R-B)]}{2\sqrt{(R-G)^2+(R-G)(G-B)}} \right. , otherwise \end{cases}$$

$$S = 1 - \frac{3 * \min(R,G,B)}{I} \quad ; \quad V = \max(R,G,B)$$

In experimenting with the HSV color space, Hue was identified as a strong color factor to extract skin regions in the real time video input from camera. A pixel is labeled as skin if:

Hmin < H < Hmax;

H is the Hue component of a pixel;
Hmin is the minimum Hue threshold;
Hmax is the maximum Hue threshold;

It was also noticed that the Hue range within which a skin area was expected, changes under different lighting conditions. So the Hue range is made dynamically changeable through a track bar even when the program is running.

4.5.2 YCbCr Color Space

Y is the luminance component and Cb and Cr are the blue and red chrominance components respectively. The explicit separation of the luminance and chrominance components makes this color space attractive for skin color detection. [15]

$$Y = 0.299R + 0.587G + 0.114B, \quad Cb = 128 - 0.168736R - 0.331264 + 0.5B$$
$$Cr = 128 + 0.5R - 0.418688G - 0.081312B \quad R, G, B, Y, Cb, Cr \in \{0,1,\ldots,255\}$$

A pixel is labeled as skin if it satisfies the following constraint:

Ymin < Y < Ymax;
Crmin < Cr < Crmax;
Cbmin < Cb < Cbmax;

As in the previous case with HSV color space, the color component ranges in the YCbCr space is also made dynamically changeable to adjust with the changing environment and lighting conditions.

Figure 4.7 given shows skin extraction performed through OpenCV.

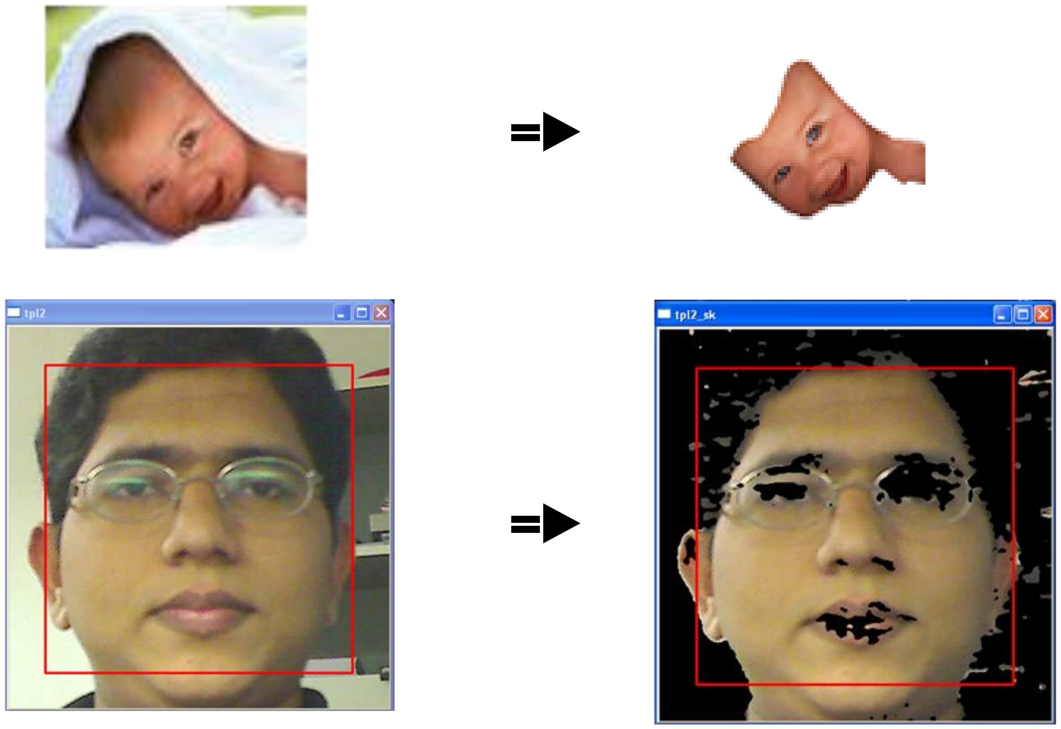


Figure 4. 7: Skin Extraction

4.6 Contour Detection

Contours are sequence of points defining a line/curve in an image. Contour matching can be used to classify image objects. To find contours of skin, the following procedure was followed: [16][17]

1. Convert the color space from RGB to HSV
2. Split the Hue Component of the video frame
3. Find region between two threshold levels (which matches the skin)
4. Find and draw contours

Figure 4.8 show contours detected in one of the experiments.

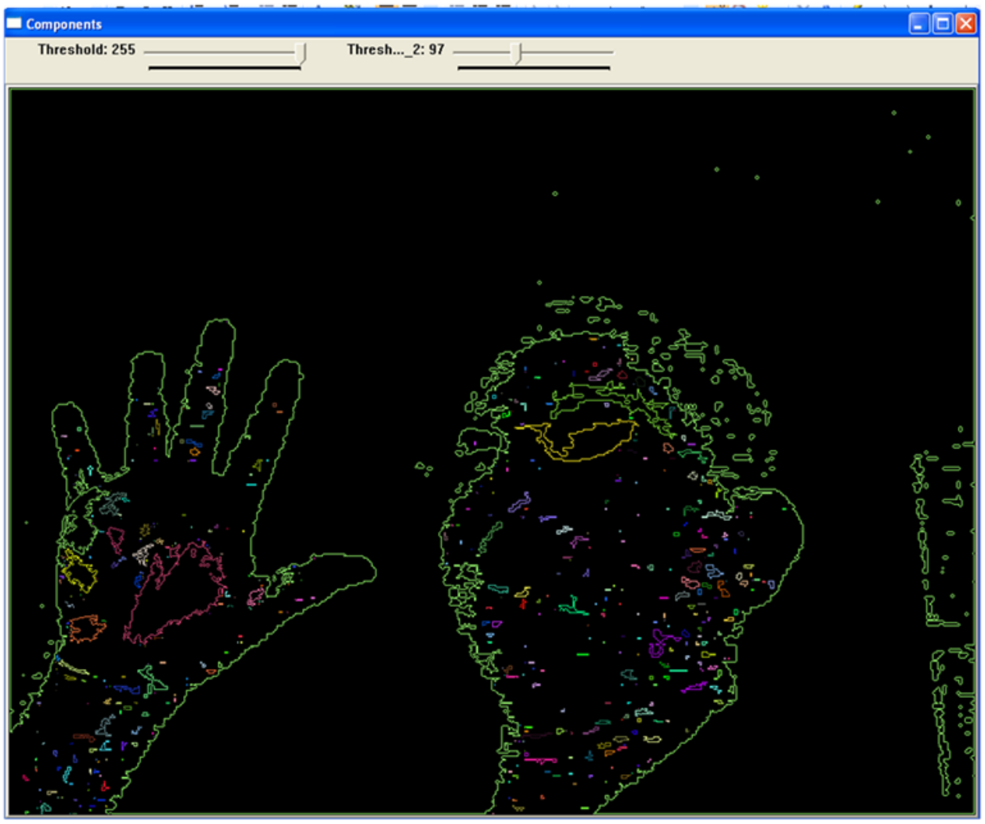


Figure 4. 8: Contour Detection

4.7 Face Detection using Haar Feature Classifier

Haar-like features are digital image features used in object recognition. Haar-like features are so called because they are computed similar to the coefficients in Haar wavelet transforms. Historically, object detection was carried out by working with only image intensities (RGB values at every pixel of the image) and so the task was computationally expensive. Research on this area has lead to the working with alternate feature set instead of the usual image intensities. This feature set considers rectangular regions of the image and sums up the pixels in this region. This sum is used to categorize images. Thus it is possible to categorize all images whose Haar-like feature in this rectangular region is in a certain range of values as one category and those falling out of this range in another category. [18][19]

OpenCV provides functions which can apply Haar Classifiers to an image to detect a particular object. The Haar Classifers are cascades of boosted classifiers working with Haar-like features. The Haar Classifiers are trained with a few hundreds of sample views of a particular target object that are scaled to the same size, called positive samples and with negative samples (arbitrary images of the same size, with the object to be detected not present in the image). [19]

After a classifier is trained, it can be applied to a region of interest (of the same size as used during the training) in an input image. The classifier outputs a "1" if the region is likely to show the object (i.e., face/car), and "0" otherwise. To search for the object in the whole image one can move the search window across the image and check every location using the classifier. The classifier is designed so that it can be easily "resized" in order to be able to find the objects of interest at different sizes, which is more efficient than resizing the image itself. So, to find an object of an unknown size in the image the scan procedure should be done several times at different scales. [19]

In this project the Haar-Classifiers which are trained to detect human faces are used. Figure 4.9 shows a face detected using the haar classifier method.

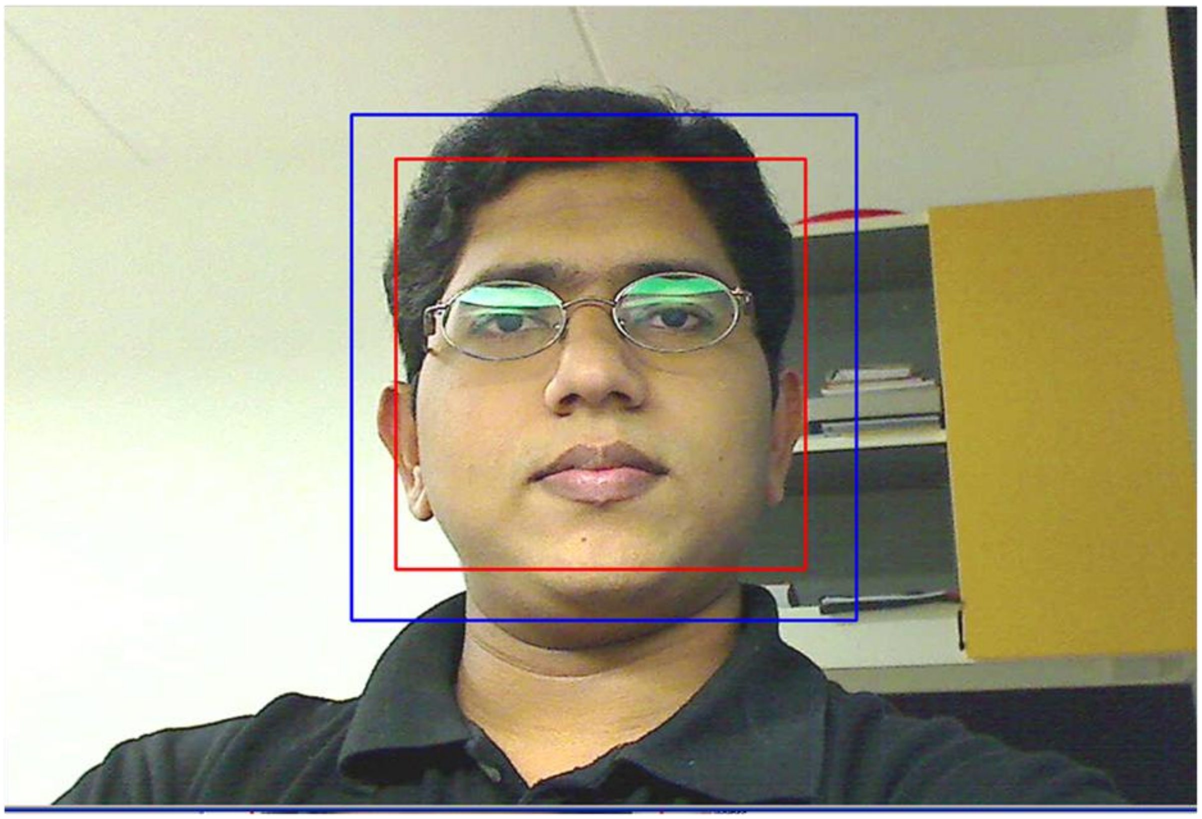


Figure 4. 9: Face Detection using

4.8 Defining the Region Of Interest (ROI)

Region of Interest is a rectangular area in an image, to segment the area for further processing. OpenCV supports setting of ROI. Once the ROI is set, most of the OpenCV functions will be performed only on the particular region highlighted by the ROI. Figure 4.10 shows an ROI (subimage) within an image. [20]

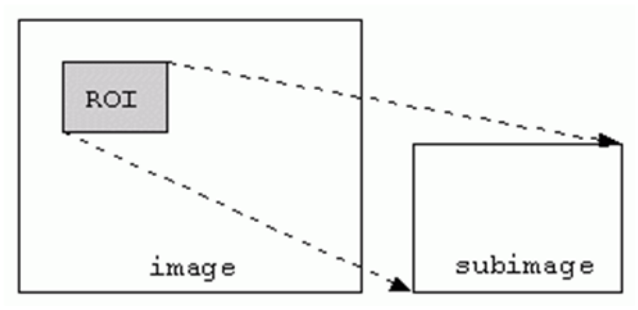


Figure 4. 10: Region Of Interest (ROI)

ROI greatly helps computer vision codes by concentrating only on specific areas of interest within an image where the target is expected to be present. Setting ROI speeds up the algorithm to a great extent and so ROI setting functions are indispensable tools in OpenCV.

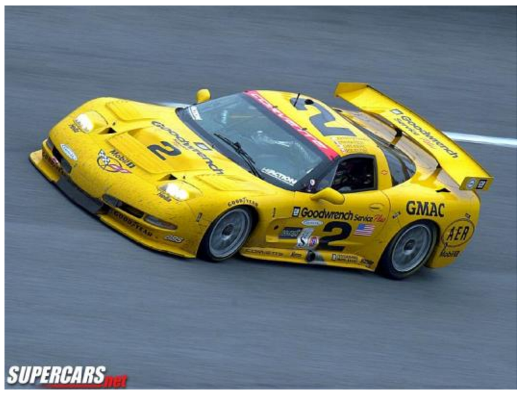
4.9 Template Matching

Template matching is a technique in digital image processing for finding small parts of an image which match a template image. The template matching procedure involves sliding the template from the top left to the bottom right of the search image, and comparing for the best match with template. The template dimension should be equal or smaller than the reference/search image. [21]

OpenCV has built in functions which can perform template matching and they are made use of in the project. Figure 4.11 shows an example of template matching.

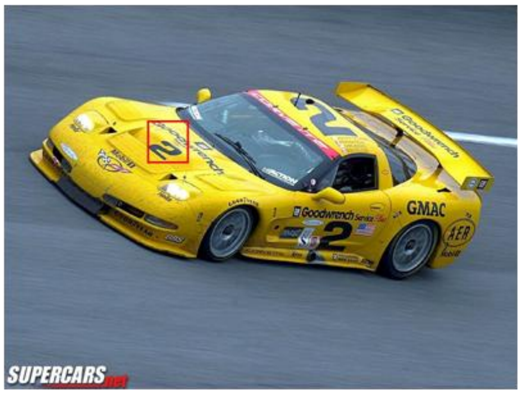
Template Image

+



Search Image

=



Result Image

Figure 4. 11: Template Matching

5. Developing the Target Tracking Program Frame Work Using MVC++ and OpenCV Library (Solely based on Template Matching)

There are two approaches in which the template matching algorithm can be implemented:

- Static Template Matching
- Dynamic Template Matching

5.1 Static Template Matching

The normal method in which Template Matching is used for Target Tracking involves creating a static model template of a specific dimension at the beginning of the program and searching for a match of the same template in the subsequent frames of the video.

One limitation of static template matching is that a match cannot be found if the target's features (shape, color, etc) changes slightly due to movement or due to changes in lighting.

5.2 Dynamic Template Matching

In this method, the model template will get replaced with the match region found on the incoming frames from the video camera. The advantage of using Dynamic Template matching is that it will detect the target even when the target's features (shape, color, etc) changes slightly. Figure 5.1 shows the model template changing when the features of the target changes.

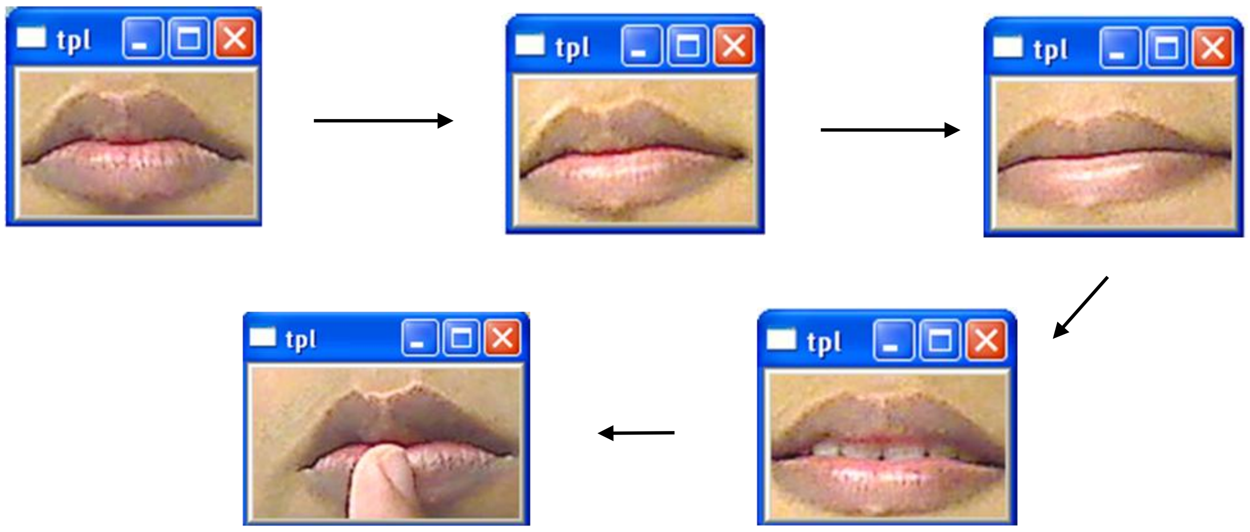


Figure 5. 1: Dynamic Template Matching

One limitation of dynamic template matching is that if any object covers the target very slowly, the dynamic model template will get replaced with the wrong object and the program will no longer track the target.

Another limitation of Template Matching is that the size of the target in the frame will change, when it comes closer or goes farther away from the camera. To counter this, certain constraints had to be imposed regarding the position of the camera. The camera should be placed nearer to the ceiling of the room, looking downwards in a slanting position. From this position, the target will appear bigger when the height of the target from the bottom of the frame decreases. This effect was taken into consideration to add in the template resizing codes into the program. The template will resize according to the height of the template from the bottom of the frame.

The effect of changing the template size according to its height from the bottom of the frame is shown in figure 5.2.

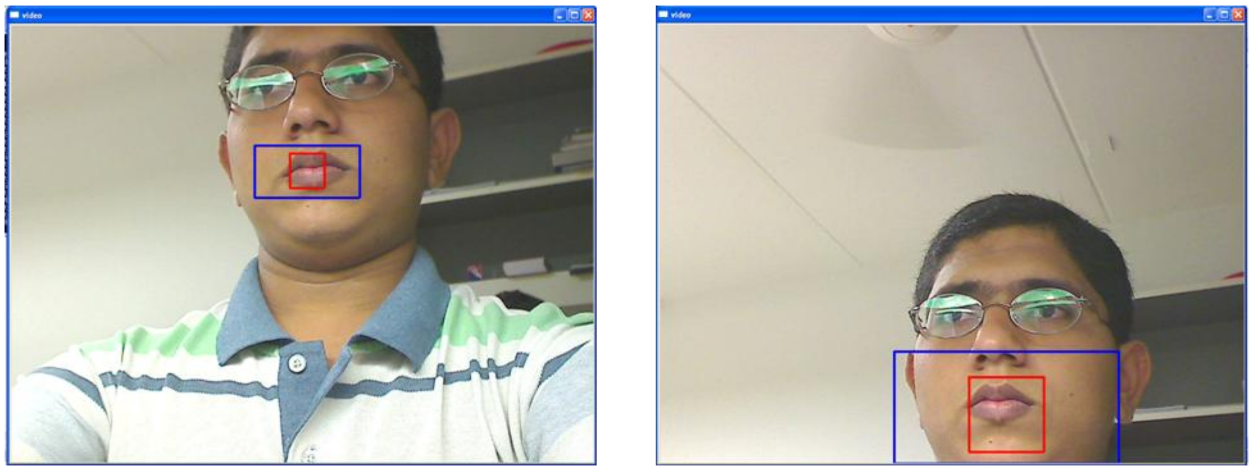


Figure 5. 2: Template Resizing with Height

5.3 Boundary Constraints

One problem facing the dynamic template tracking is that the template gets replaced when the target disappears at the boundary of the video frame. To counter this, boundary constraints were added to the program. Using the boundary constraints, the template of the target being tracked is stored at the boundary and this template will be used for template matching at the boundary regions.

5.4 Integrated Template Matching

The two approaches of static and dynamic template matching were integrated together to get better results. Under this approach the program will search for a match of the static model template in the current frame. If a match cannot be found, then the program searches for a match of the dynamic model template. If a match to the static template is found in the current frame, the dynamic model template gets replaced with the frame portion where the static template match was found.

5.5 Search Area (ROI – Region Of Interest)

When a target match can be found in the current frames being processed, the ROI is kept as small as possible to provide fast processing of the frames and execution of the program. But when a match cannot be found, the ROI is slowly widened to counter the possibility that the target might have moved out of the search area in a very rapid motion.

5.6 Color Signature and Template Selection

A strong colored signature or pattern (cap, design on shirt, etc) is to be included in the model template along with the face of the person (target) being tracked for better results.

The selected template should not include any background. Including background will result in poor tracking. The difference between a good and bad template selection is figures 5.3 and 5.4.

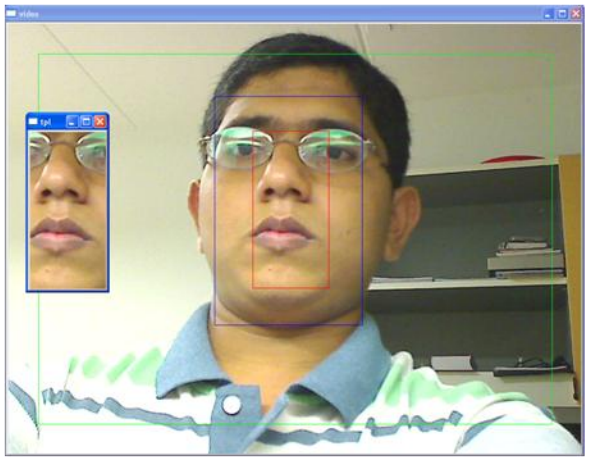


Figure 5. 3: Good Template Selection

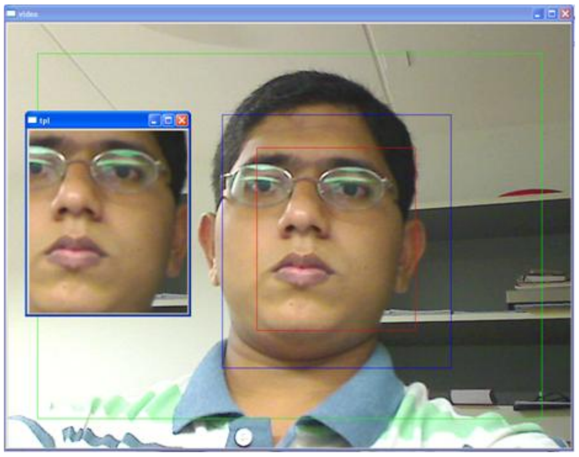


Figure 5. 4: Bad Template Selection

5.7 Disadvantages of a solely Template Matching based Tracking Program

The algorithms developed using template matching proved to be inefficient since:

1. The initial template should be chosen by the user (not automatic)
2. There is a chance that the target might get lost because of changes in lighting, change in expressions of the target face, etc

To counter the problem of manual selection of target template, the face tracking algorithm possible by haar classifiers was introduced into the program.

The haar-classifers gives much false detection if the frame being handled has many objects of the same size as that of a probable face. To reduce the number of false detections, the concept of skin extraction was incorporated into the program. The haar classifier will run on top of the skin region identified by the skin extractor.

Since the haar classifier algorithm along with skin extraction will be computationally very intensive, a sophisticated algorithm utilizing the ROI was to be developed to make the ultimate algorithm as robust as possible.

6. Developing the Target Tracking Program Frame Work Using MVC++ and OpenCV Library (Robust Program combining Multiple Algorithms)

6.1 Arriving at a Robust Program combining Multiple Algorithms

A robust target tracking program was developed combining various different algorithms namely:

- Face detection using haar classifier
- Skin extraction
- Applying Region of Interests (ROI)
- Template Matching

The flow charts showing the basic frame work of the target tracking program is shown in the next few pages.

The program after initialization enters into a loop where each video frame from the camera will be analyzed in a series of steps to detect, locate and track the target face. The programs consist of four states of analyzing the video frame depending upon the status of tracking.

When the program is in State 3 and 4, the target tracking is being carried out. So the program calculates the coordinates of the target face and sends it to the COM port of the computer. Any device which controls the pan and tilt system can be connected to the COM port of the computer to utilize the coordinates of the face calculated by the image processing algorithm running on the host computer. Figure 6.1 shows the flow chart of the overview of the program.

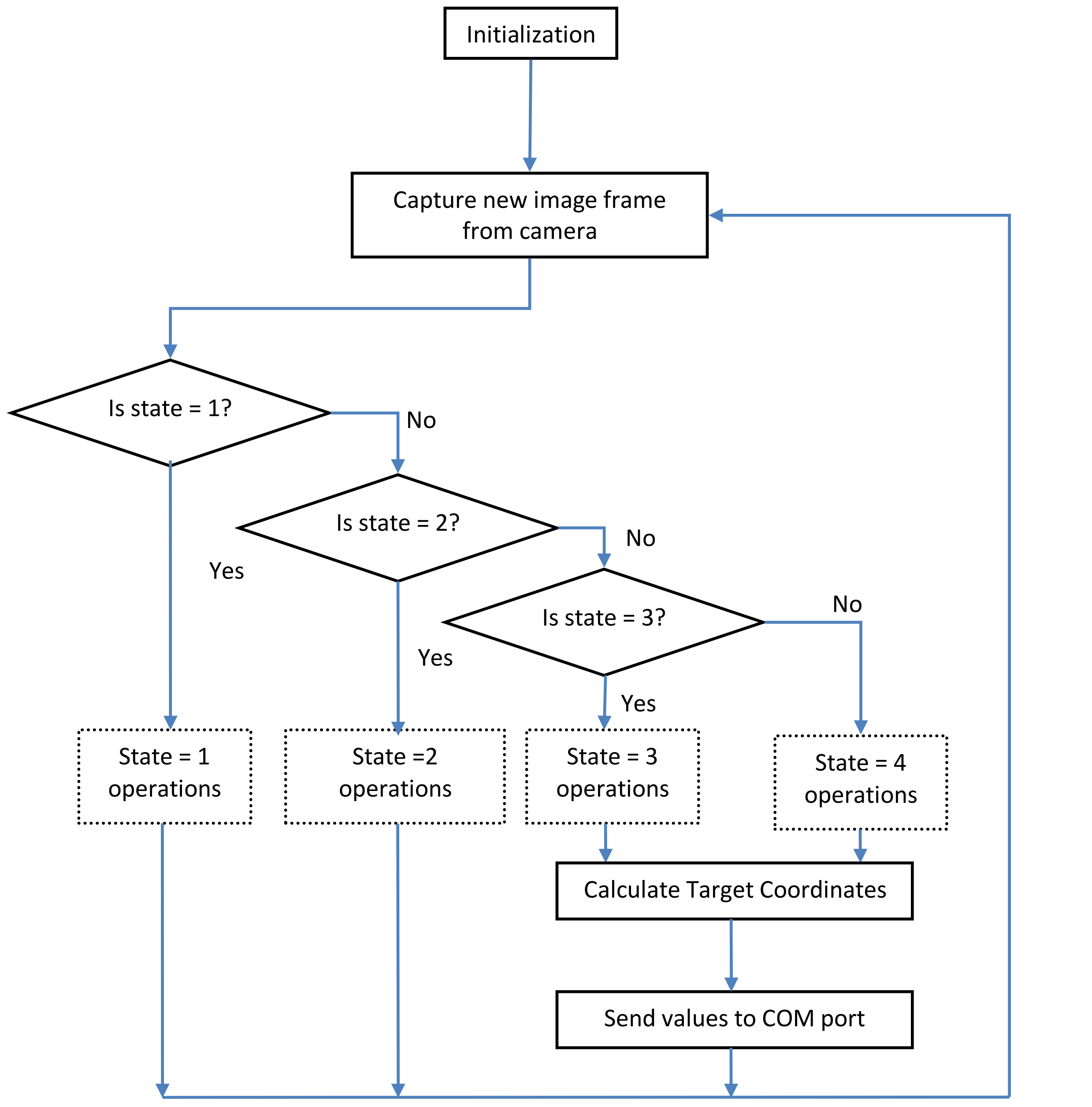


Figure 6. 1: Flow Chart – Program Overview

6.2 State 1

The program existing in this state would mean than no face is detected, or face which was being tracked is lost.

Program detects the skin area within the entire incoming video frame by applying the skin extractor algorithm. Within the skin area detected it searches for face using the haar classifier.

Passing the incoming frame through the two algorithms improves the probability of identifying a true face.

If a face is found, the program calculates the Region of Interest (ROI) surrounding the detection area and moves into state 2. If no face is found the program stays in state 1. Figure 6.2 shows the flowchart of state 1 operations.

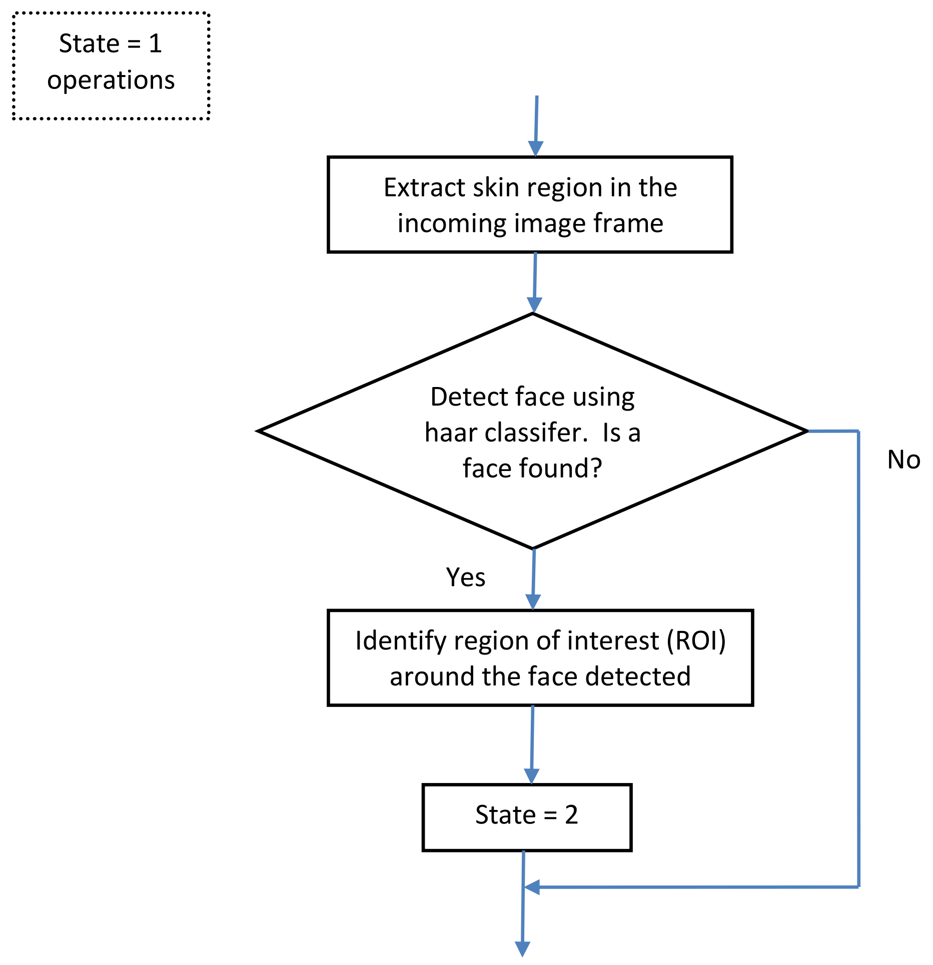

State = 1 operations
Extract skin region in the incoming image frame
Detect face using haar classifer. Is a face found?
No
Yes
Identify region of interest (ROI) around the face detected
State = 2

Figure 6. 2: Flow Chart – State 1

6.3 State 2

In state 2, the program searches for a face within the same Region Of Interest (ROI) where a face was detected in state 1. If a face is found, the program confirms that the target detected is in fact a face and the program moves to state 3. The program in state 2 acts as a double confirmation for the face detected in state 1.

If a face cannot be found, the program returns to state 1. Figure 6.3 shows flow chart of the state 2 operations.

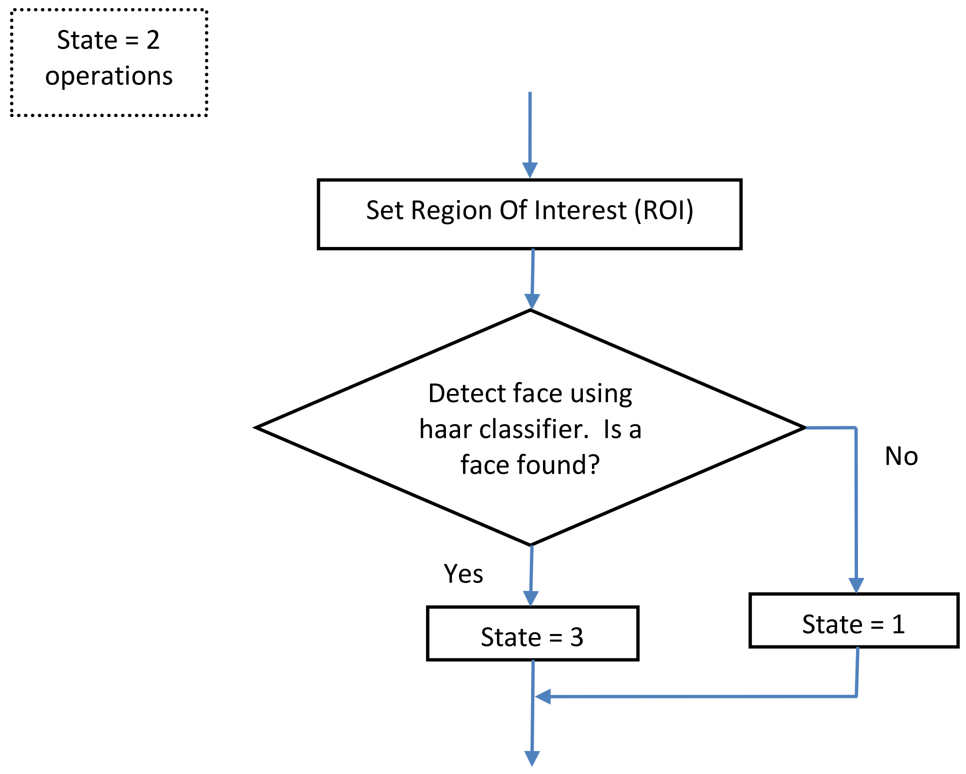

State = 2 operations
Set Region Of Interest (ROI)
Detect face using haar classifier. Is a face found?
No
Yes
State = 3
State = 1

Figure 6. 3: Flow Chart – State 2

6.4 State 3

In state 3, tracking of target face is in progress. The confirmed target face is being detected and followed using haar classifier within the Region of Interest (ROI). The 'count_no_haar_detection' parameter is used to direct the program to either state 1 or state 4 according to different conditions. Count_no_haar_detection is incremented until 5 if a face cannot be detected in state 3. The program in state 3 prepares the model template for template matching carried out in state 4 when count_no_haar_detection equates to 1. If a face cannot be found in continuous 5 rounds of state 3 (count_no_haar_detection = 5), the program moves into state 1. If a face is found then the parameter count_no_haar_detection is equated to zero. If a face is not found and count_no_haar_detection is less than 5, then the program moves into state 4. Figure 6.4 shows the flow chart of state 3 operations.

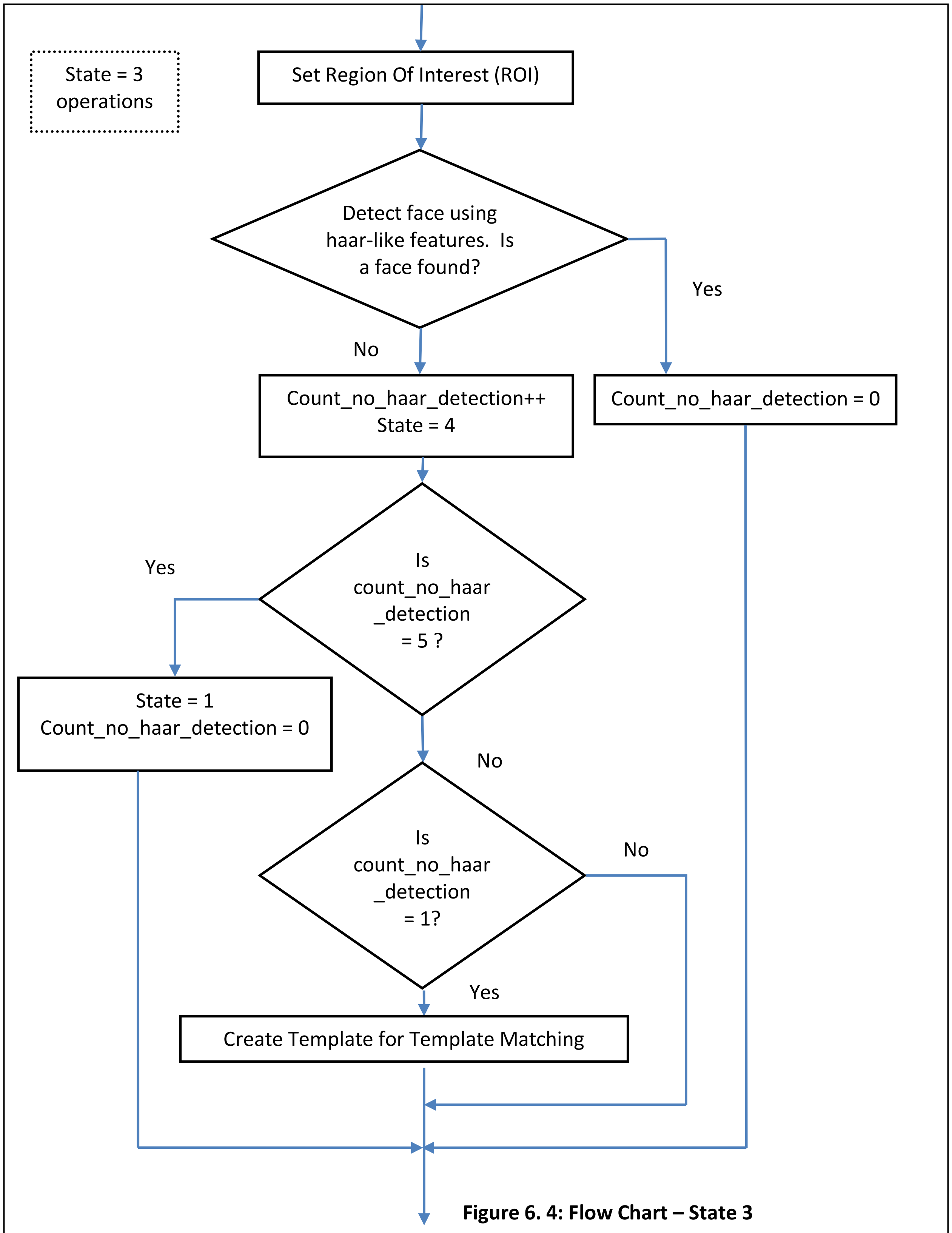

State = 3
operations
Set Region Of Interest (ROI)
Detect face using haar-like features. Is a face found?
Yes
No
Count_no_haar_detection++
State = 4
Count_no_haar_detection = 0
Yes
Is count_no_haar _detection = 5 ?
State = 1
Count_no_haar_detection = 0
No
No
Is count_no_haar _detection = 1?
Yes
Create Template for Template Matching

Figure 6. 4: Flow Chart – State 3

6.5 State 4

In state 4, the program is still tracking the target face detected. Previously in state 3, haar classifier could not find a face within the ROI. In state 4, the program uses template matching to track the face within the ROI. When in state 4, the program executes 5 loops at a time before reverting back to state 3. This is counted by the parameter 'count_template_matching_rounds'. Template matching just returns the area within the ROI showing the maximum degree of match to the template. The concept of dynamic template matching is used which means that the model template gets replaced with the area of maximum match found within the current video frame. Figure 6.5 shows flow chart of state 4 operations.

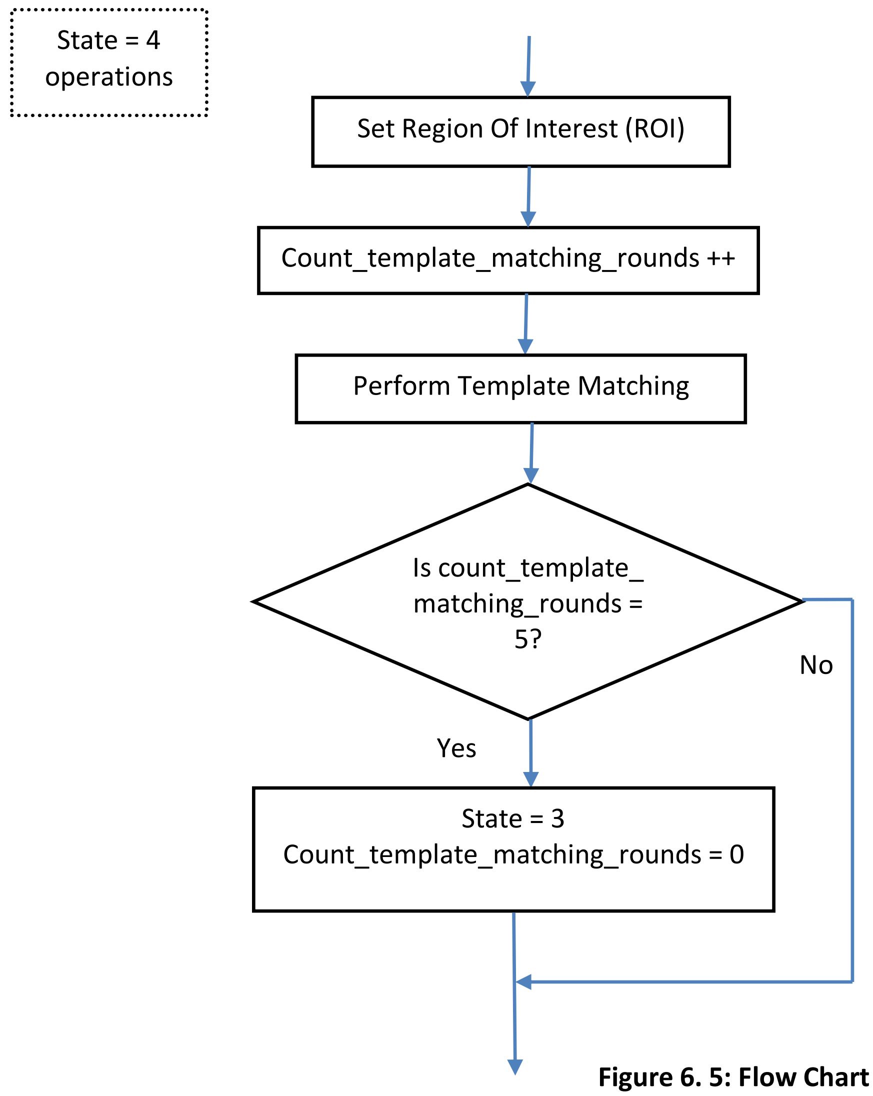
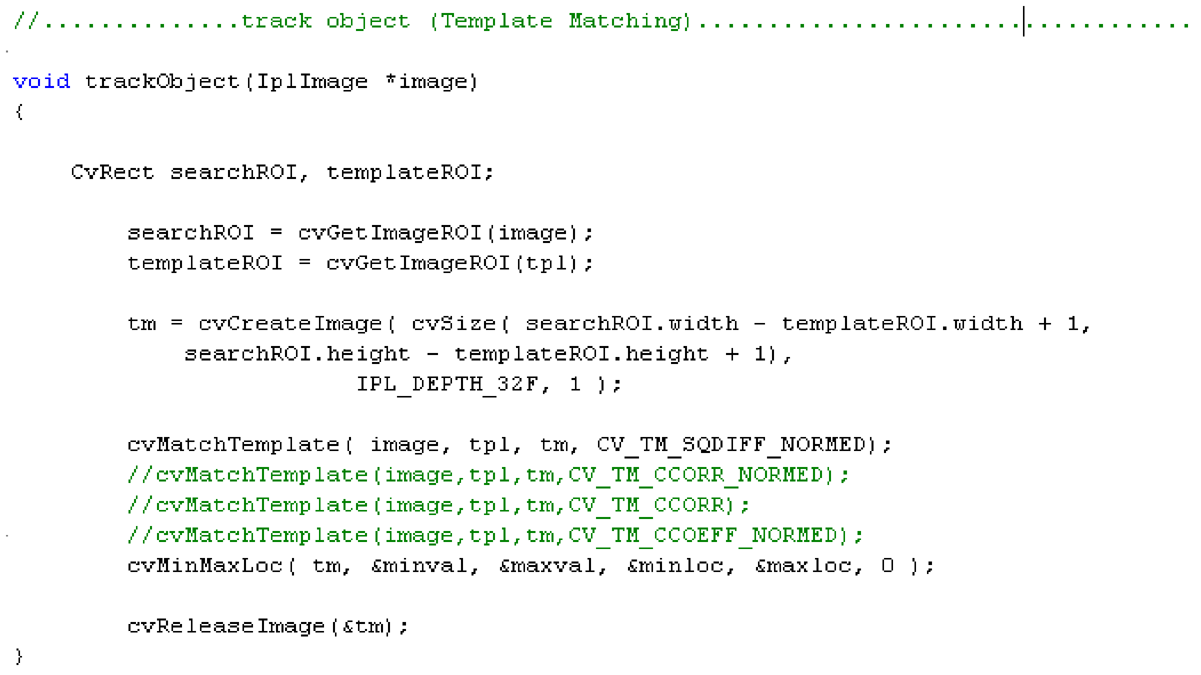

State = 4 operations
Set Region Of Interest (ROI)
Count_template_matching_rounds ++
Perform Template Matching
Is count_template_ matching_rounds = 5?
No
Yes
State = 3
Count_template_matching_rounds = 0

Figure 6. 5: Flow Chart – State 4

6.6 Template Matching Function - trackObject()

This function performs template matching on the video frame 'image' passed to it using the model template 'tpl'. The main function performing the template matching is cvMatchTemplate(). The function cvMinMaxLoc() locates the maximum and minimum values of the match and also their corresponding locations. Given next is the code for the function used in the program.



```
//.............track object (Template Matching).................................

void trackObject(IplImage *image)
{

     CvRect searchROI, templateROI;

         searchROI = cvGetImageROI(image);
         templateROI = cvGetImageROI(tpl);

         tm = cvCreateImage( cvSize( searchROI.width - templateROI.width + 1,
             searchROI.height - templateROI.height + 1),
                         IPL_DEPTH_32F, 1 );

         cvMatchTemplate( image, tpl, tm, CV_TM_SQDIFF_NORMED);
         //cvMatchTemplate(image,tpl,tm,CV_TM_CCORR_NORMED);
         //cvMatchTemplate(image,tpl,tm,CV_TM_CCORR);
         //cvMatchTemplate(image,tpl,tm,CV_TM_CCOEFF_NORMED);
         cvMinMaxLoc( tm, &minval, &maxval, &minloc, &maxloc, 0 );

         cvReleaseImage(&tm);
}
```

The cvMatchTemplate() function is explained in more detail below. [22][23]

void cvMatchTemplate(const CvArr* *image*, const CvArr* *templ*, CvArr* *result*, int *method*)

Compares a template against overlapped image regions.

image – Image where the search is running; should be 8-bit or 32-bit floating-point

templ – Searched template; must be not greater than the source image and should be the same data type as the image

result – A map of comparison results; single-channel 32-bit floating-point. If *image* is $W \times H$ and *templ* is $w \times h$ then *result* must be $(W - w + 1) \times (H - h + 1)$

method – Specifies the way the template must be compared with the image regions (explained below)

The function slides through *image*, compares the overlapped patches of size $w \times h$ against *templ* using the specified *method* and stores the comparison results to *result*. Here are the formulas for the different comparison methods one may use (I denotes *image*, T *template*, R *result*). The summation is done over template and/or the image patch: $x' = 0...w-1, y' = 0...h-1$

method = CV_TM_SQDIFF *

$$R(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2$$

method = CV_TM_SQDIFF_NORMED *

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

method = CV_TM_CCORR *

$$R(x,y) = \sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))$$

method = CV_TM_CCORR_NORMED *

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

method = CV_TM_CCOEFF *

$$R(x,y) = \sum_{x',y'} (T'(x',y') \cdot I(x+x',y+y'))$$

where

$$T'(x',y') = T(x',y') - 1/(w \cdot h) \cdot \textstyle\sum_{x'',y''} T(x",y")$$
$$I'(x+x',y+y') = I(x+x',y+y') - 1/(w \cdot h) \cdot \textstyle\sum_{x'',y''} I(x+x",y+y")$$

method = CV_TM_CCOEFF_NORMED *

$$R(x,y) = \frac{\sum_{x',y'} (T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

After the function finishes the comparison, the best matches can be found as global minimums (for CV_TM_SQDIFF) or maximums (for CV_TM_CCORR and CV_TM_CCOEFF) using the MinMaxLoc() function. In the case of a color image, template summation in the numerator and each sum in the denominator is done over all of the channels (and separate mean values are used for each channel).

In the program the template matching algorithm uses the method CV_TM_SQDIFF_NORMED since it has been shown to give better results.

6.7 Face Detection Function - detect_faces()

This function performs the detection of target faces in the incoming video frame 'image'. The haar cascade used in the program is 'haarcascade_frontalface_alt_tree.xml'. In the function, the incoming image is downsized for faster operation at the cost of only a little performance degradation. The downsizing is performed using the function cvPyrDown(). To perform the downsizing, the parameter 'do_pyramids' is equated to 1.

The cvHaarDetectObjects() function is able to detect all the probable face regions within the incoming video frame. But the pan and tilt system can direct the audio beam only to a particular target at a time. So the program is only concerned about a single target on the frame. Hence the operation performed by the cvHaarDetectObjects() function is 'CV_HAAR_FIND_BIGGEST_OBJECT' which returns only the biggest face region found in the image frame.

Given next is the codes which implements the detect_faces() function in the program.

```
// ................detect faces (haar classifier) ......................................................

int detect_faces( IplImage* image,CvHaarClassifierCascade* cascade,int do_pyramids)
{
    IplImage* small_image = image;
    CvMemStorage* storage = cvCreateMemStorage(0);
    CvSeq* faces;
    int i, scale = 1;
    int temp = 0;

    // if the flag is specified, down-scale the input image to get a performance boost w/o loosing quality
    if( do_pyramids )
    {
        small_image = cvCreateImage( cvSize(image->width/2,image->height/2), IPL_DEPTH_8U, 3 );
        cvPyrDown( image, small_image, CV_GAUSSIAN_5x5 );
        scale = 2;
    }

    faces = cvHaarDetectObjects( small_image, cascade, storage,
                                 1.2,                          // Scale factor
                                 2,                            // Minimum number of neighbours
                                 CV_HAAR_FIND_BIGGEST_OBJECT,   // Operation flag
                                 cvSize(20,20));                // Minimum size of the face


    /* draw all the rectangles */
    for( i = 0; i < faces->total; i++ )
    {
        // extract the rectanlges only
        CvRect face_rect = *(CvRect*)cvGetSeqElem( faces, i );

        face.height = face_rect.height*scale;
        face.width  = face_rect.width*scale;
        face.x = face_rect.x*scale;
        face.y = face_rect.y*scale;

        temp = faces->total;
    }

    if( small_image != image )
        cvReleaseImage( &small_image );
    cvReleaseMemStorage( &storage );
    return temp;
}
```

A more detailed explanation of the cvHaarDetectObjects() function is given below. [24][25]

```
CvSeq* cvHaarDetectObjects(const CvArr* image,
                           CvHaarClassifierCascade* cascade,
                           CvMemStorage* storage,
                           double scale_factor=1.1,
                           int min_neighbors=3,
                           int flags=0,
                           CvSize min_size=cvSize(0, 0))
```

image – Image to detect objects in
cascade – Haar classifier cascade in internal representation

storage – Memory storage to store the resultant sequence of the object candidate rectangles

scale_factor – The factor by which the search window is scaled between the subsequent scans, 1.1 means increasing window by 10%

min_neighbors – Minimum number (minus 1) of neighbor rectangles that makes up an object. All the groups of a smaller number of rectangles than *min_neighbors*-1 are rejected. If *min_neighbors* is 0, the function does not perform any grouping at all and returns all the detected candidate rectangles, which may be useful if the user wants to apply a customized grouping procedure

flags – Mode of operation

- 0: If this flag is set, then the function finds and returns all possible face regions within the frame.

- CV_HAAR_DO_CANNY_PRUNING: If it is set, the function uses Canny edge detector to reject some image regions that contain too few or too much edges and thus can not contain the searched object. The particular threshold values are tuned for face detection and in this case the pruning speeds up the processing.

- CV_HAAR_FIND_BIGGEST_OBJECT: If this flag is set, the function returns only the biggest face region found in the image frame.

min_size – Minimum window size. By default, it is set to the size of samples the classifier has been trained on ($\sim 20 \times 20$ for face detection)

The function cvHaarDetectObjects() finds rectangular regions in the given image that are likely to contain objects the cascade has been trained for and returns those regions as a sequence of rectangles. The function scans the image several times at different scales. Each time it considers overlapping regions in the image and applies the classifiers to the regions using *RunHaarClassifierCascade*(). It may also apply some heuristics to reduce number of analyzed regions, such as Canny prunning. After it has proceeded and collected the candidate rectangles (regions that passed the classifier cascade), it groups them and returns a sequence of average rectangles for each large enough group. The default parameters (scale_factor =1.1,min_neighbors =3, flags =0) are tuned for accurate yet slow object detection. For a faster operation on real video images the settings are:scale_factor =1.2, min_neighbors =2, flags =:cmacro:*CV_HAAR_DO_CANNY_PRUNING*, min_size =* minimum possible face size* (for example, ~1/4 to 1/16 of the image area in the case of video conferencing).

6.8 Skin Extraction Function – GetSkinMask()

The GetSkinMask() function creates a mask which can be used to extract skin from the original video frame. The skin mask should be ANDed with the original image to get the skin extracted image.

The figure 6.6 shows the flow chart of the basic skin mask creation process.

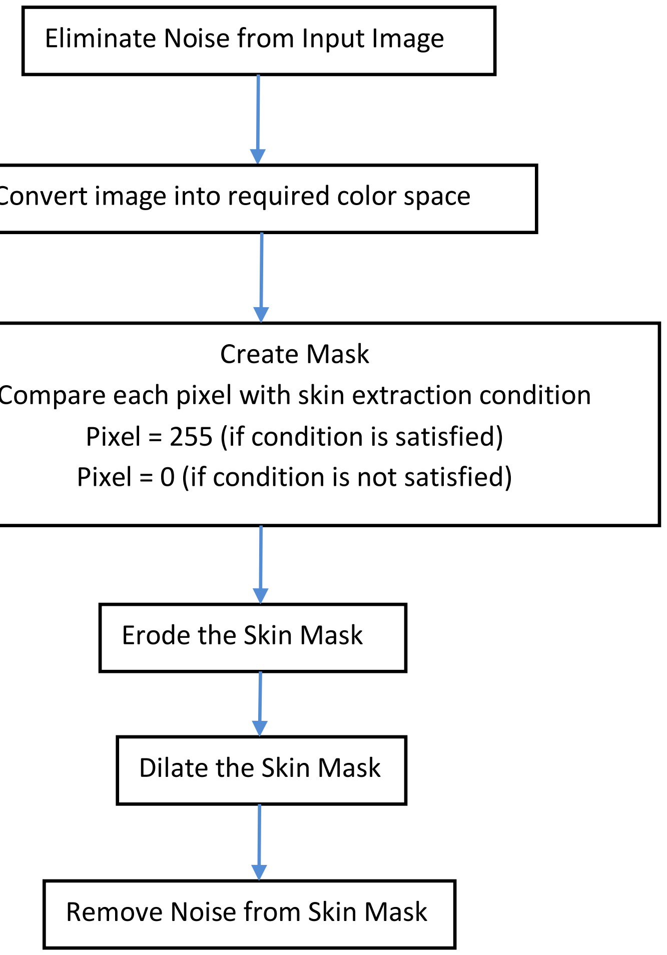

Eliminate Noise from Input Image
Convert image into required color space
Create Mask
Compare each pixel with skin extraction condition
Pixel = 255 (if condition is satisfied)
Pixel = 0 (if condition is not satisfied)
Erode the Skin Mask
Dilate the Skin Mask
Remove Noise from Skin Mask

Figure 6. 6: Skin Mask Creation

The initial module performs the elimination of noise from the input image. This is carried out by downsampling followed by upsampling the input image. The cvPyrDown() function performs downsampling step of the Gaussian Pyramid Decomposition. It convolutes the source image with the specified filter (CV_GAUSSIAN_5X5) and then downsamples the image by rejecting even rows and columns. The function cvPyrUp() performs up-sampling step of Gaussian pyramid decomposition. First it upsamples the source image by injecting even zero rows and columns and then convolves result with the specified filter multiplied by 4 for interpolation. The final image after performing downsampling and upsampling is more error free. [26]

The image is then converted to a chosen color space. The different color spaces which are used are HSV, YCbCr and RGB. The selection of specific color spaces can be performed from the program itself. The cvCvtColor() function performs the conversion of color spaces.

After conversion into the required color space, the program loops through two for-loops, in which each pixel value will be subjected to inspection against a certain criterion which shows whether it represents a skin pixel or not. If the pixel satisfies the condition, the corresponding pixel in the skin mask being created is given the value 255 (maximum intensity - white). If the condition is not satisfied by the pixel, the corresponding pixel in the mask is labeled as 0 (zero intensity - black). Carrying out this procedure for all the pixels on the frame leads to the formation of the entire skin mask.

The next couple of steps are to perform erosion and dilation on the skin mask. The functions are described below. [27]

The cvErode() function erodes the skin mask using a 3X3 rectangular structuring element that determines the shape of the pixel neighborhood over which the minimum is taken:

$$\min_{(x',y')\, in\, \texttt{element}} src(x+x', y+y')$$

Erosion is iterated many number of times as specified by the parameter ‘erosions’.

The cvDilate() function dilates the skin mask using a 3X3 rectangular structuring element that determines the shape of the pixel neighborhood over which the maximum is taken:

$$\max_{(x',y')\, in\, \texttt{element}} src(x+x', y+y')$$

Dilation is iterated many number of times as specified by the parameter ‘dilations’.

The final step is a noise removal process by applying a smoothing filter on the skin mask. The cvSmooth() function performs the smoothing operation. The filter used here is a median filter (CV_MEDIAN) with a 5X5 square aperture.

The codes used to implement the GetSkinMask() function is given next.

```
//....................get skin mask ..............................................................

void GetSkinMask(IplImage * src_RGB, IplImage * mask_BW, int erosions, int dilations)
{
    CvSize sz = cvSize( src_RGB->width & -2, src_RGB->height & -2);
    //get the size of input_image (src_RGB)

    IplImage* pyr = cvCreateImage( cvSize(sz.width/2, sz.height/2), 8, 3 ); //create 2 temp-images
    // pyr is needed to downsample the source image, which is required for noise reduction.

    IplImage* src = cvCreateImage(cvGetSize(src_RGB), IPL_DEPTH_8U ,3);
    cvCopyImage(src_RGB, src);

    IplImage* tmpHSV = cvCreateImage(cvGetSize(src), IPL_DEPTH_8U , 3);

    //remove noise from input image = cvPyrDown() + cvPyrUp() using the CV_GAUSSIAN_5X5

    cvPyrDown( src, pyr, 7 );
    // Downsamples an image. Function performs downsampling step of the Gaussian Pyramid Decomposition.
    // Convolutes the source image with the specified filter (CV_GAUSSIAN_5X5 = 7) and then downsamples
    // the image by rejecting even rows and columns.

    cvPyrUp( pyr, src, 7 );
    /* Performs up-sampling step of Gaussian pyramid decomposition.First it upsamples the source image
    by injecting even zero rows and columns and then convolves result with the specified filtermultiplied
    by 4 for interpolation. So the destination image is four times larger than the source image.
    */

    if(skin_detection_method == 2)
        cvCvtColor(src ,tmpHSV , CV_RGB2YCrCb);     // RGB to YCrCb color conversion
    else if(skin_detection_method == 1)
        cvCvtColor(src, tmpHSV, CV_RGB2HSV);
    else if(skin_detection_method == 3)
        cvCopyImage(src,tmpHSV);

    uchar H;
    uchar S;
    uchar V;

    uchar Y;
    uchar Cb;
    uchar Cr;
uchar R;
uchar G;
uchar B;


CvPixelPosition8u pos_src;      // To move through the pixels easily
CvPixelPosition8u pos_dst;

int x =0;
int y =0;

//Initializing the pixel position structure
CV_INIT_PIXEL_POS(pos_src,(unsigned char *) tmpHSV->imageData,
    tmpHSV->widthStep,cvGetSize(tmpHSV),x,y,tmpHSV->origin);

CV_INIT_PIXEL_POS(pos_dst,(unsigned char *) mask_BW->imageData,
    mask_BW->widthStep,cvGetSize(mask_BW),x,y,mask_BW->origin);



uchar * ptr_src;        // A single pixel
uchar * ptr_dst;


for( y=0;y<src-> height; y++)
{

    for ( x=0; x<src->width; x++)
    {

        ptr_src = CV_MOVE_TO(pos_src,x,y,3);        // Move to the position
        ptr_dst = CV_MOVE_TO(pos_dst,x,y,3);

        H = ptr_src[0];     // Y, Cb, Cr extraction from source pixel
        S = ptr_src[1];
        V = ptr_src[2];

        Y = H;
        Cr = S;
        Cb = V;

        R = H;
        G = S;
        B = V;

        // Color condition and detection
    if(skin_detection_method == 1){
        if((H > Hmin) && (H < Hmax))
        {
            ptr_dst[0] = 255;       // Mask creation
            ptr_dst[1] = 255;
            ptr_dst[2] = 255;
        }
        else
        {
            ptr_dst[0] = 0;
            ptr_dst[1] = 0;
            ptr_dst[2] = 0;
        }
    }
    else if(skin_detection_method == 2){
        if((Y > Ymin) && (Y < Ymax) && (Cb > Cbmin) && (Cb < Cbmax) && (Cr > Crmin) && (Cr < Crmax))
        {
            ptr_dst[0] = 255;       // Mask creation
            ptr_dst[1] = 255;
            ptr_dst[2] = 255;
        }
        else
        {
            ptr_dst[0] = 0;
            ptr_dst[1] = 0;
            ptr_dst[2] = 0;
        }
    }
    else if(skin_detection_method == 3){
        if((R < Redmax) && (G < Greenmax) && (B < Bluemax) && (R > Redmin) &&(G > Greenmin) && (B > Bluemin))
        {
            ptr_dst[0] = 255;       // Mask creation
            ptr_dst[1] = 255;
            ptr_dst[2] = 255;
        }
        else
        {
            ptr_dst[0] = 0;
            ptr_dst[1] = 0;
            ptr_dst[2] = 0;
        }
    }
}
```

```
    if(erosions>0)
        cvErode(mask_BW,mask_BW,0,erosions);     // Erode the mask
    if (dilations>0)
        cvDilate(mask_BW,mask_BW,0,dilations);   // Dialate the mask

    //Apply an averaging filter to remove noise from mask_BW image
    cvSmooth(mask_BW, mask_BW, CV_MEDIAN, 5, 5, 0, 0);

    cvReleaseImage(&pyr);
    cvReleaseImage(&tmpHSV);
    cvReleaseImage(&src);
}
```

6.9 Function Calculating the Center Coordinates of the Target – calculate_cordinates()

The function calculate_cordinates() calculates the center point of the target face being followed. The frame axis is also converted as shown in figure 6.7 below by this function. (Here it is assumed that the video frame is of resolution 960 X 720).

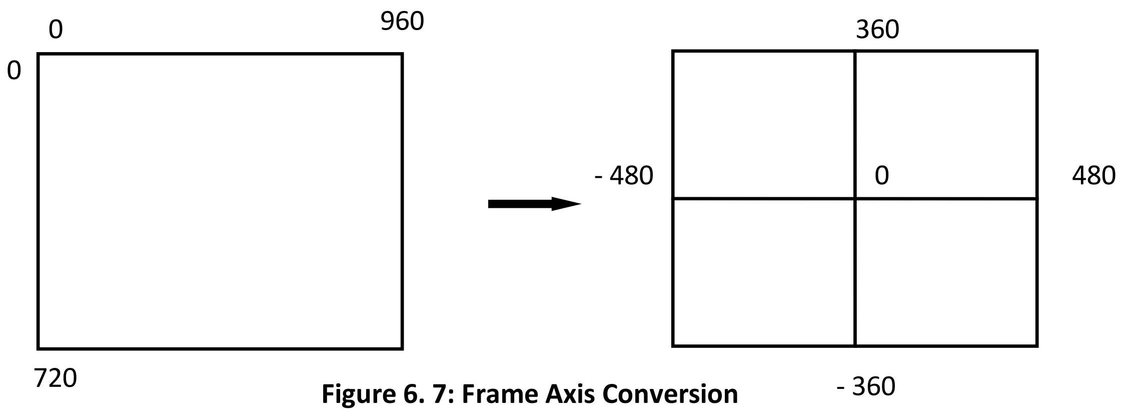


Figure 6. 7: Frame Axis Conversion

The code implementing this function is given next.

```
//..................calculate cordinates.................................
void calculate_cordinates(void){

    center_x = object_x0 + tpl_width/2;
    center_y = object_y0 + tpl_height/2;

    center_x = frameW - center_x;

    center_x = center_x - frameW/2; // center_x >= -480 and  <= 480
    center_y = center_y - frameH/2; // center_y >= -360 and  <= 360

}
```

6.10 Function Calculating the Degree of Pan and Tilt Needed – calculate_pan_and_tilt()

This function calculates the degree of pan and tilt bursts necessary in order to bring the target object back to the centre point (0,0) of the video frame. The function rules than the degree of pan or tilt should be higher if the target is farther away from the centre. Given next is the code implementing this function.

```
//................calculate pan and tilt .......................
void calculate_pan_and_tilt(void){

    if(center_x > 200)
        pan_value = pan_value + 4;
    else if(center_x > 100)
        pan_value = pan_value + 1;
    else if(center_x < -200)
        pan_value = pan_value - 4;
    else if(center_x < -100)
        pan_value = pan_value - 1;

    if(center_y > 200)
        tilt_value = tilt_value + 4;
    else if(center_y > 100)
        tilt_value = tilt_value + 1;
    else if(center_y < -200)
        tilt_value = tilt_value - 4;
    else if(center_y < -100)
        tilt_value = tilt_value - 1;

}
```

6.11 Function to Convert Pan and Tilt Values to Sending Format – convert_values()

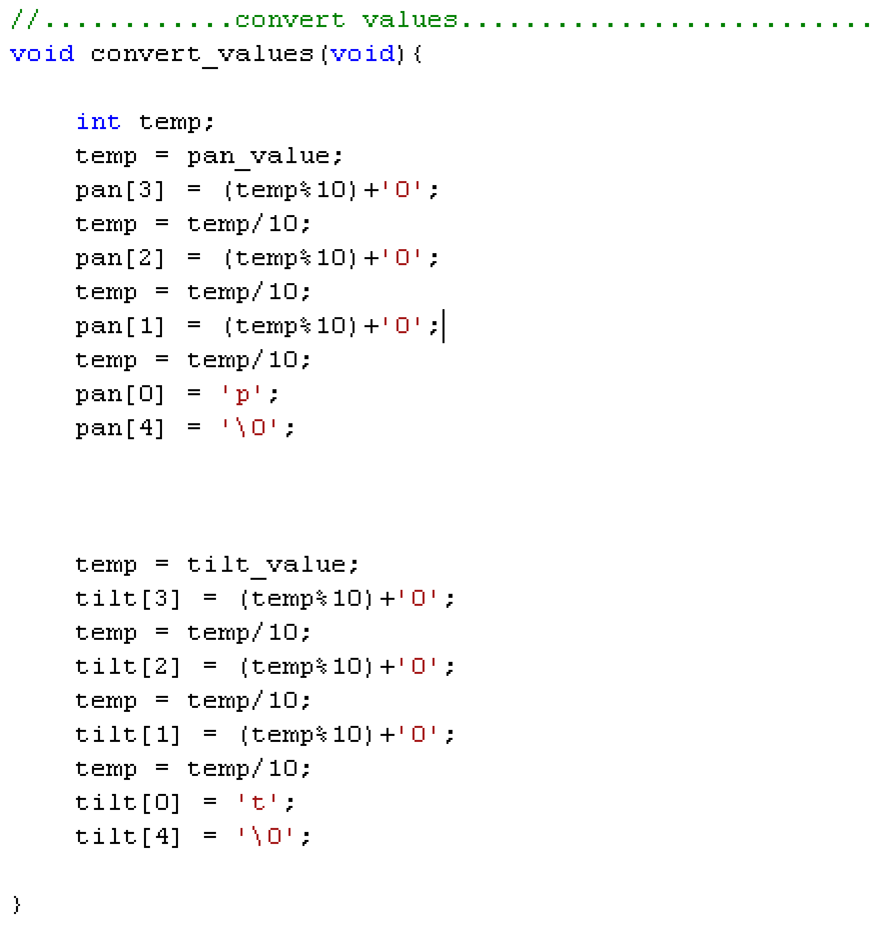
Values send to the COM port of the host computer should be in ‘char’ format for devices connected to the COM port to pick them up in a common standard format. So pan and tilt values should be converted to a standard character format before being send to the COM port.

Standard Sending Format: Five characters (ex: “p123\0”) used to represent one value of pan or tilt. The first character is either a ‘p’ or ’t’ representing ‘pan’ or ‘tilt’ respectively. The next three characters are numerical values and they represent the degree value of rotation in the pan or tilt direction. The last character is the ‘\0’ which is used to represent the end of a string. Essentially only four characters are sent to the COM port since the character ‘\0’ is used only to denote end of the string internally within the VC++ program.

An example of a conversion performed by the function convert_values() is:

pan_value = 750 → pan = “p750\0”.

The code implementing the convert_values() function is given next.



```
//............convert values..........................
void convert_values(void){

     int temp;
     temp = pan_value;
     pan[3]  =  (temp%10)+'0';
     temp = temp/10;
     pan[2]  =  (temp%10)+'0';
     temp = temp/10;
     pan[1]  =  (temp%10)+'0';
     temp = temp/10;
     pan[0]  =  'p';
     pan[4]  =  '\0';



     temp = tilt_value;
     tilt[3]  =  (temp%10)+'0';
     temp = temp/10;
     tilt[2]  =  (temp%10)+'0';
     temp = temp/10;
     tilt[1]  =  (temp%10)+'0';
     temp = temp/10;
     tilt[0]  =  't';
     tilt[4]  =  '\0';

}
```

6.12 Handling the Serial Port in MVC++

From MVC++, the serial ports can be accessed through the Application Programming Interface (API) provided by the MFC class libraries. For this the MFC class libraries should be included in the project as shown in Appendix B.

The COM port 4 is opened as the serial port for communication accessible through the MVC++. This is specified by the parameter 'PortSpecifier'. The CreateFile() structure creates a handle to the COM port that can be used to write data to the port. After setting the state of the port to meet the baud rate requirements, the WriteFile() function can be used to send data to the port.

The port settings are:

Baud Rate = 9600 bits per second
Byte Size = 8 bits
Parity = No
Stop Bits = 1 Stop Bit

The code in the program which initializes the COM port for communication is given next.

```
if(enable_port)
{
    // Initializing the Port
    PortSpecifier = "\\\\.\\COM4";
    hPort = CreateFile(
        PortSpecifier,
        GENERIC_WRITE,
        0,
        NULL,
        OPEN_EXISTING,
        0,
        NULL
        );
    port_opened = true;

    if (!GetCommState(hPort,&dcb)){
        port_opened = false;
    }
    else{
        dcb.BaudRate = CBR_9600; //9600 Baud
        dcb.ByteSize = 8; //8 data bits
        dcb.Parity = NOPARITY; //no parity
        dcb.StopBits = ONESTOPBIT; //1 stop
    }
    if (!SetCommState(hPort,&dcb)){
        port_opened = false;
    }

    if(!port_opened){
        printf("Port could not be opened!\n");
        return 3;
    }
    else{
        printf("Port opened\n");
    }
}
```

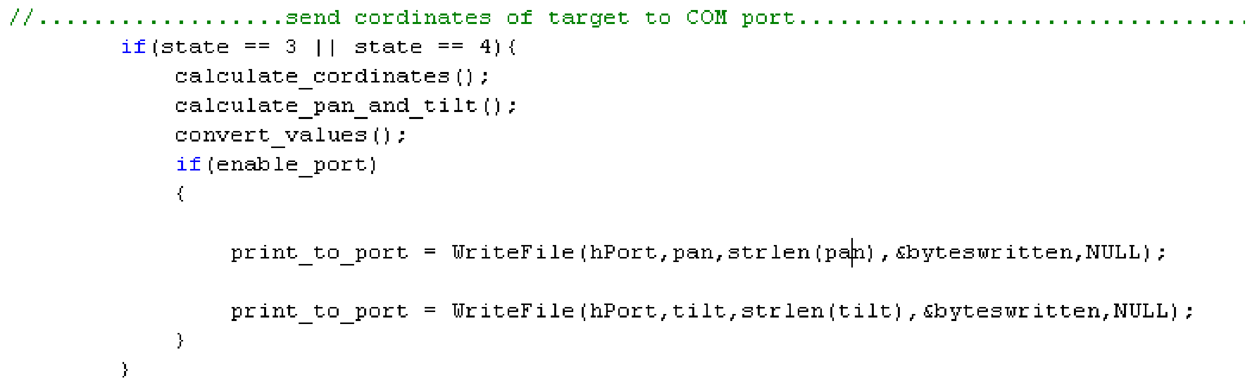
6.13 Method of Sending the Coordinates of the Target to the COM port

Coordinates of the target are sent to the COM port when the program is in state 3 or 4. To send the pan and tilt values to the COM port, a series of functions are executed.

→ Calculate_cordinates() → Calculate_pan_and_tilt() → Convert_values() → WriteFile()

The parameters used with the WriteFile() function are the file handle (hPort), the data to be written as a string, the number of bytes to be written, and a place to store the number of bytes actually written to the port.

The RS232 of the host computer was connected to the RS232 of a second computer, using a null modem cable. The hyperterminal of the second computer was activated for testing. The hyperterminal showed that the values sent from the host computer are in fact received by the second computer. Given next is the code which shows how the program sends pan and tilt values to the COM port.



```
//..................send cordinates of target to COM port...................................
        if(state == 3 || state == 4){
            calculate_cordinates();
            calculate_pan_and_tilt();
            convert_values();
            if(enable_port)
            {

                print_to_port = WriteFile(hPort,pan,strlen(pan),&byteswritten,NULL);

                print_to_port = WriteFile(hPort,tilt,strlen(tilt),&byteswritten,NULL);
            }
        }
```

6.14 Mouse Handler Function – mouseHandler()

The mouse handler function in OpenCV enables the use of different mouse events to input parameters into the running program. [28]

The different variables, events and flags which can be handled in the mouse handler function are given below.

x,y:
pixel coordinates with respect to the upper left corner of the screen

event:
CV_EVENT_LBUTTONDOWN, CV_EVENT_RBUTTONDOWN, CV_EVENT_MBUTTONDOWN,
CV_EVENT_LBUTTONUP, CV_EVENT_RBUTTONUP, CV_EVENT_MBUTTONUP,
CV_EVENT_LBUTTONDBLCLK, CV_EVENT_RBUTTONDBLCLK, CV_EVENT_MBUTTONDBLCLK,
CV_EVENT_MOUSEMOVE:

flags:
CV_EVENT_FLAG_CTRLKEY, CV_EVENT_FLAG_SHIFTKEY, CV_EVENT_FLAG_ALTKEY,
CV_EVENT_FLAG_LBUTTON, CV_EVENT_FLAG_RBUTTON, CV_EVENT_FLAG_MBUTTON

The mouse handler has been used in this project for the selection of a template from the video frame itself for tracking purposes. The code in the program is given next.

```
//...............mouse handler ..........................................

void mouseHandler( int event, int x, int y, int flags, void *param )
{

    if(event == CV_EVENT_LBUTTONDOWN){
        pos_x1 = x;
        pos_y1 = y;

    }
    else if( event == CV_EVENT_LBUTTONUP ) {
        //printf("x : %d, y : %d\t",x,y);
        pos_x2 = x;
        pos_y2 = y;
        if((pos_x2 > pos_x1)&&(pos_y2>pos_y1))
        accept_mouse_input = 1;

    }

}
```

6.15 Restricting ROI within the Frame Dimensions

The ROI should be restricted to be always within the frame boundaries throughout the run time of the program. If the logical values of ROI move out of the frame dimensions, it results in "out of bounds error".

The figure 6.8 shows ROI going out of bounds. The code given next is used in several places within the program to ensure that the ROI is always within the frame boundaries.

```
    if(win_x0<=0)
        win_x0 = 0;
    if(win_y0<0)
        win_y0 = 0;
    if(window_width>frameW)
        window_width = frameW;
    if(window_height> frameH)
        window_height = frameH ;
    if((win_x0+window_width)> frameW)
        win_x0 = frameW-window_width;
    if((win_y0+window_height)>frameH)
        win_y0 = (frameH-window_height);
```

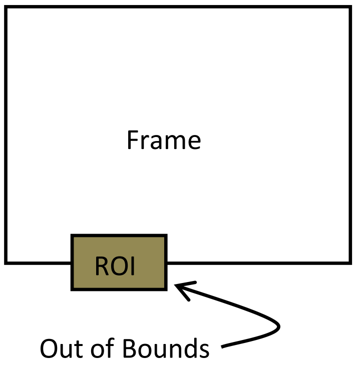


Figure 6. 8: ROI going Out of Bounds

6.16 Avoiding Memory Leak

Running the program continuously with the video frames being stored in pointers will result in a lot of memory being used in storing the images from the live video. If the memory is not freed as soon as the images stored become devoid of any use, it will result in "out of memory" errors.

So codes similar to the one given next were used throughout the program to free up memory whose contents become devoid of any use.

```
// create template image
if(tpl != NULL)
    cvReleaseImage(&tpl); // Release image to free memory
tpl = cvCreateImage( cvSize( tpl_width, tpl_height ),frame->depth, frame->nChannels );
```

7. Pan and Tilt System

7.1 Selecting a Pan and Tilt System

A pan and tilt system is necessary to mount the audio beaming system and to point it at the target on the scene. The audio beaming system is of circular shape and can be of variable size and weight between 5 to 10 Kg. The pan and tilt system should provide at least 180^0angle coverage for both panning and tilting. These are the specifications which decided the type of pan and tilt system to be used in the project.

The Pan and Tilt System used in the project is the 785 Series System from ServoCity. The 785 Series System is made of aluminium body, comes with two powerful servo motors and can handle a payload of 11.5 Kg. The rotation possible for panning and tilting is full 360^0. Figure 7.1 shows the 785 Series of Pan and Tilt System.

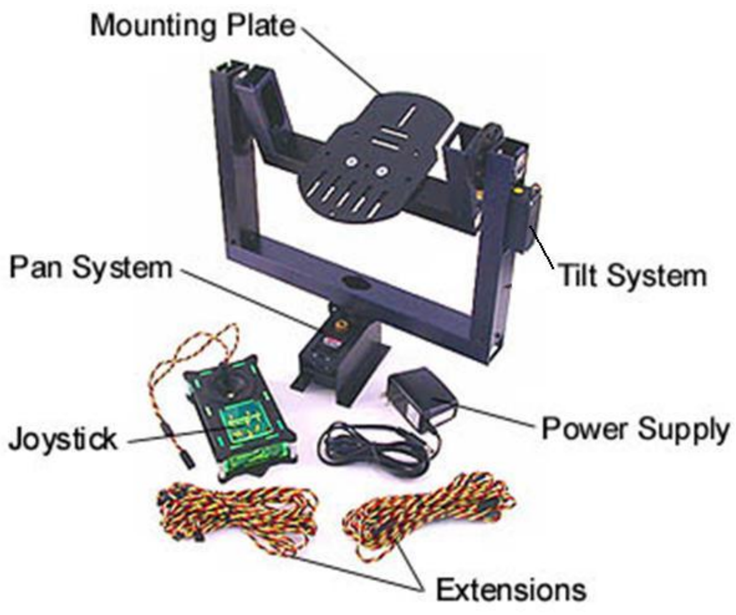


Figure 7. 1: 785 Series Pan and Tilt System from Servocity

The joystick provides two channels to control the pan and tilt servo motors. Each channel has three wires to drive the servo connected to it. Two of them are for power supply (6V) and the other one provides the control signal. The control signal is a 20 millisecond Pulse Width Modulated (PWM) signal and controls the relative position (angle of rotation) of the servo motor.

7.2 PWM control signals to the Pan and Tilt System

The figures 7.2 and 7.3 show the control signal from the joystick to the pan and tilt system plotted using the oscilloscope. Figure 7.4 shows the PWM control signal with measurements marked.

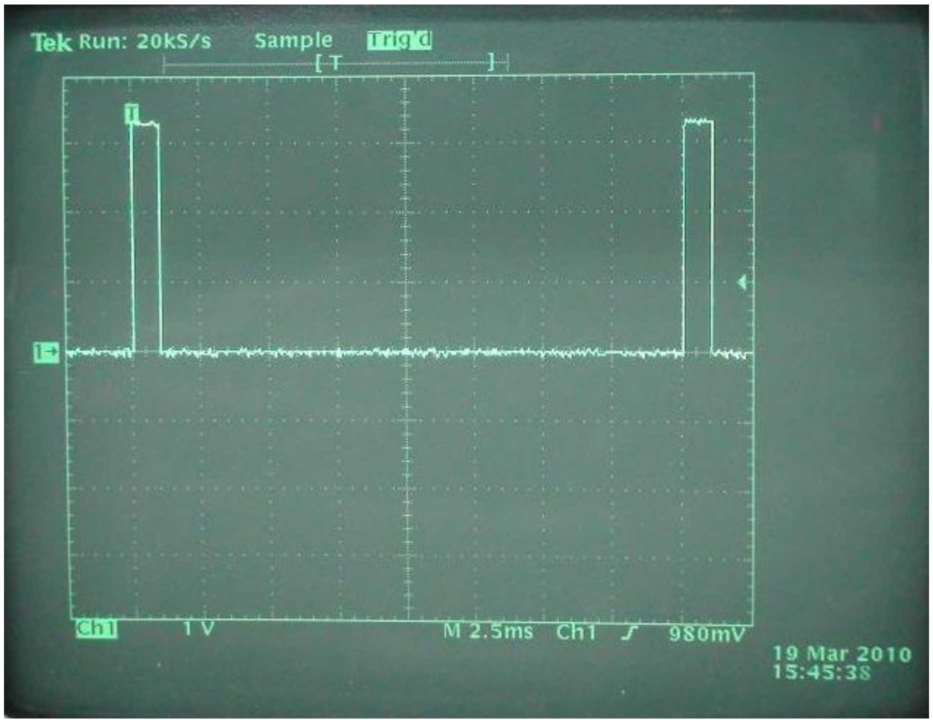


Figure 7. 2: PWM Control Signal (5% Duty Cycle)

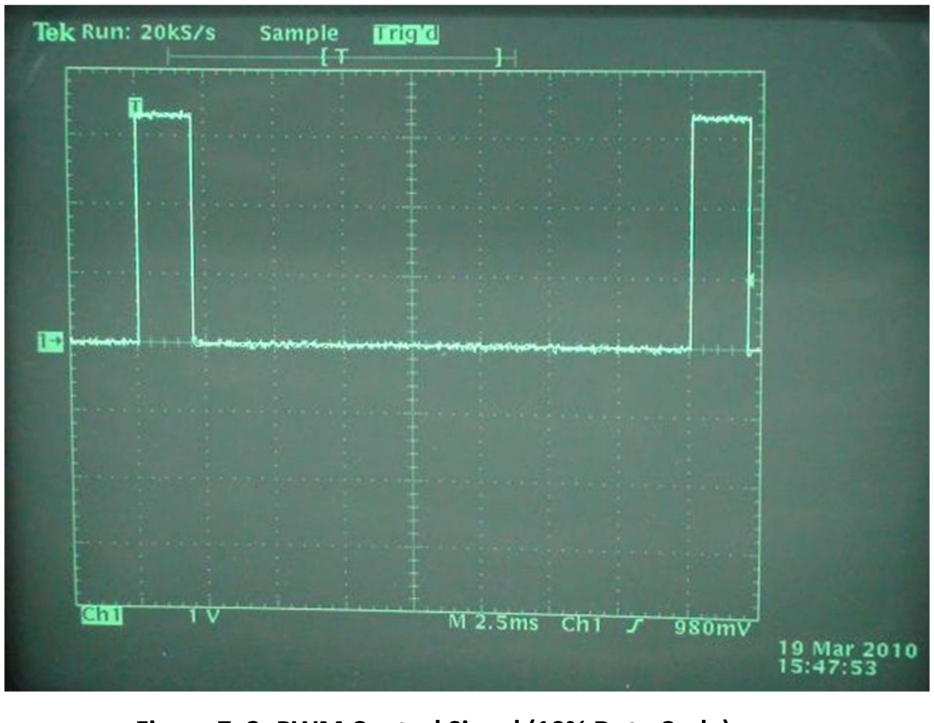


Figure 7. 3: PWM Control Signal (10% Duty Cycle)

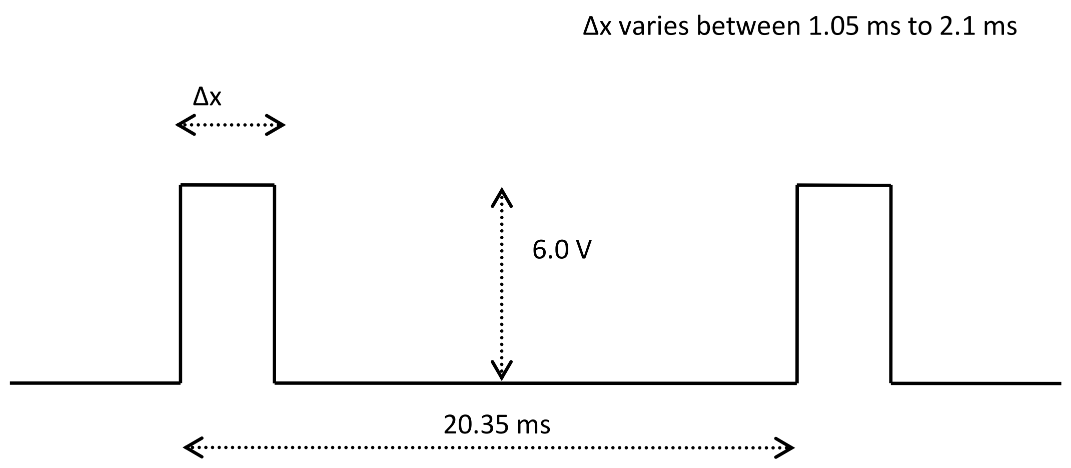


Figure 7. 4: Pulse Width Modulated (PWM) Control Signal to the Pan and Tilt System

The duty cycle of the PWM signal varies from 5 to 10 %. The signal's time period is nearly 20 ms. The duty cycle is directly mapped to angle of rotation of the servo motors and covers full 360^o.

8. Microcontroller

8.1 Need for a Microcontroller and its Selection

The video processing program running on the host computer can track the target faces on the scene, calculate the coordinates of the target and send it to the COM port. But this values should be translated to the PWM control signals which the pan and tilt system can understand. So an electronic circuit board, essentially consisting of a microcontroller, which can perform this translation, is necessary.

General Requirements of the Electronic Circuit Board:

1. Should be able to be connected to the COM port of the host computer
2. Should have a microcontroller UART port
3. Should contain modules for the generation of PWM signals
4. Two sets of control signals (3 wires- red, yellow, black) which is nearly 6 V should be able to be sourced from the circuit board

8.2 Microcontroller Circuit Board

The Explorer 16 Development Board and the PIC24FJ128GA010 Plug-In Module (PIM) was used as the microcontroller. It can be programmed and debugged using the MBLAB ICD 2 connector.

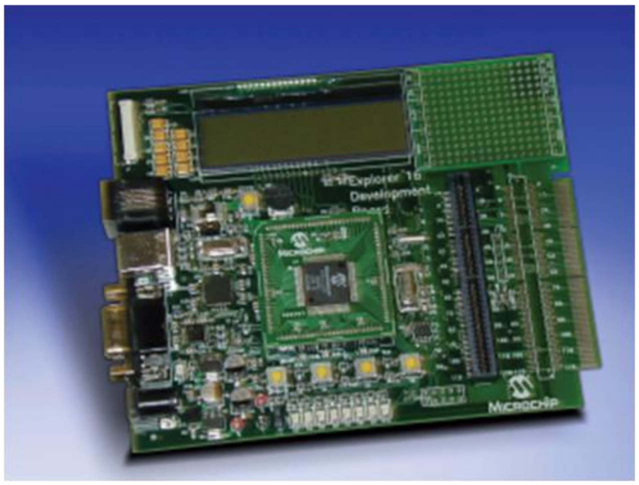


Figure 8. 1: Explorer 16 Development Board with PIC24FJ128GA010 PIM

Figure 8. 2: Microchip MPLAB ICD 2

Some of the features of the Explorer 16 Development board which lead to its selection for the project are:

- RS232 serial port and associated hardware (MAX232 module to convert signals between RS232 +-12 V to microcontroller level 0-5V)
- In-Circuit Debugger (ICD) connector for connections to the MPLAB ICD 2 programmer /debugger module
- Eight indicator LEDs
- Independent crystals for precision microcontroller clocking (8 MHz) and RTCC operation (32.768 KHz)
- Prototype area for developing custom applications

Features of the PIC24FJ128GA010 microcontroller used in the project are:

- Flash Program Memory
- Interrupt Controller
- Oscillators
- I/O Ports
- Timers
- Pulse Width Modulation (PWM) module
- Universal Asynchronous Receiver Transmitter (UART)

The microcontroller connects to the host computer through the RS232 and accepts data representing the target's position. The microcontroller produces the PWM control signal needed to drive the servo motors according to the positional data received from the program running on the host computer.

8.3 Replacing the Joystick Controller with the Microcontroller

The Joystick provided two channels to control the two servo motors in the pan and tilt system. Each channel consisted of three wires.

Red – 6 V DC power supply
Black – Ground Wire 0 V
Yellow – PWM Control Signal 6V

The circuit board including the microcontroller, which replaces the joystick, should provide similar channels to control the servo motors.

To reproduce the power supply to the servo motors, an external 5V power supply was used. Since the microcontroller runs on 3.3 V DC, the PWM control signal can only be 3.3V without any voltage translation. Two channels were soldered onto the expansion bred board provided with the Explorer 16 board. The figure 8.3 and 8.4 shows how the two PWM channels are created on the Explorer 16 board.

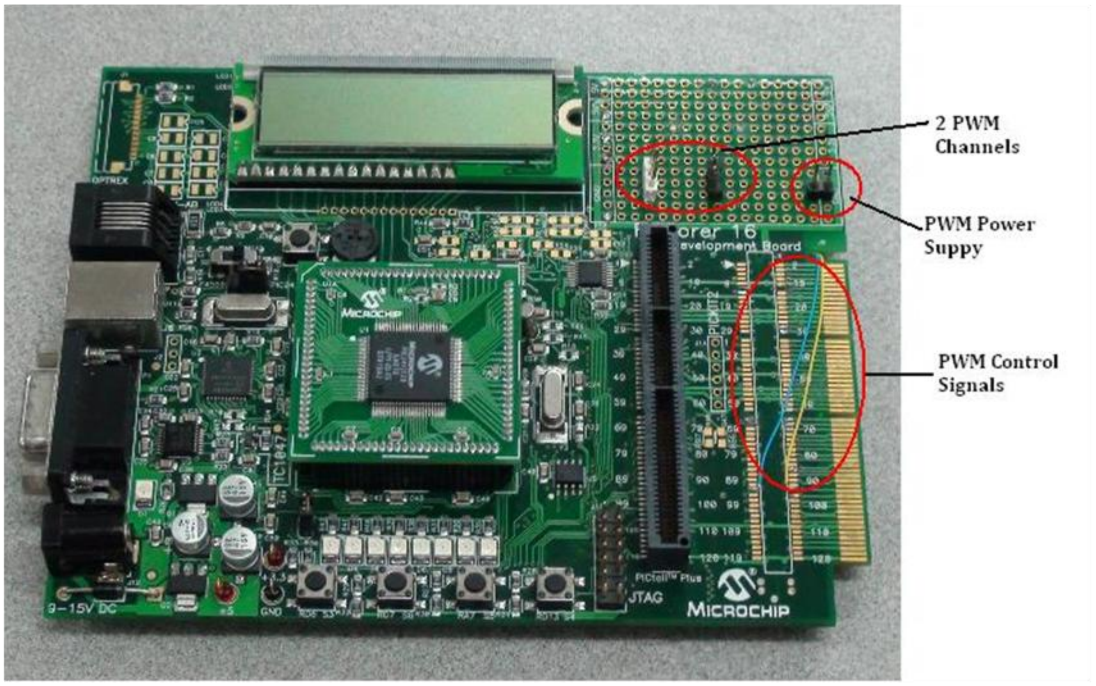


Figure 8. 3: PWM Power Supply and Channels on the Explorer 16 Board

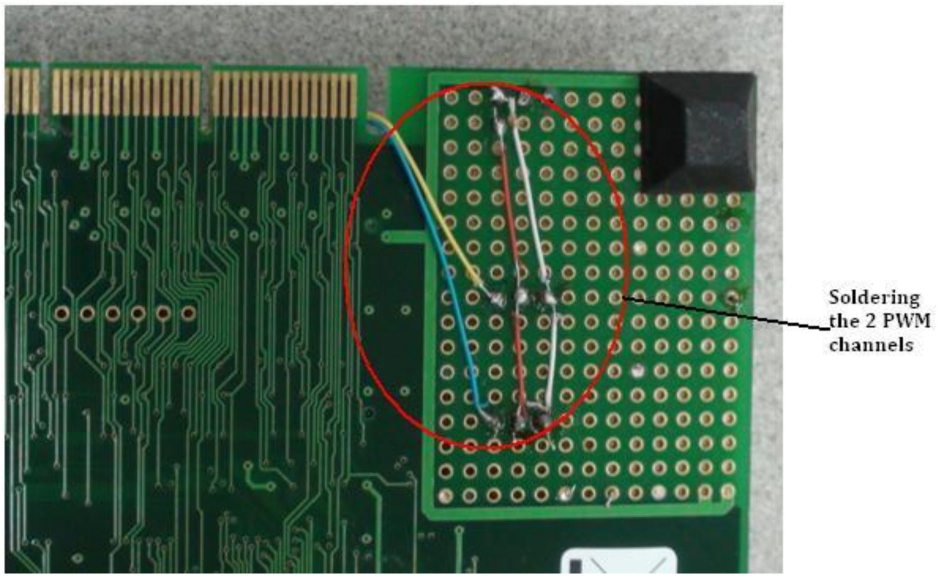


Figure 8. 4: Soldering the PWM channels onto the Explorer 16 Board

The pan and tilt system was tested with the new channels where the power supply is only 5V instead of 6V and the PWM control signal is only 3.3 V instead of 6V. It was ensured that the pan and tilt system is working fine since the changed input voltages are still within the operating ranges of the servo motors.

Figure 8.5 and 8.6 shows the Joystick and the Explorer 16 board respectively with their two servo motor driver channels.

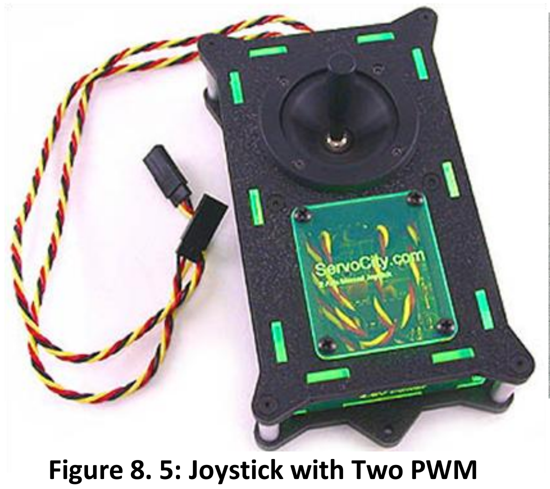


Figure 8. 5: Joystick with Two PWM Servo Motor Channels

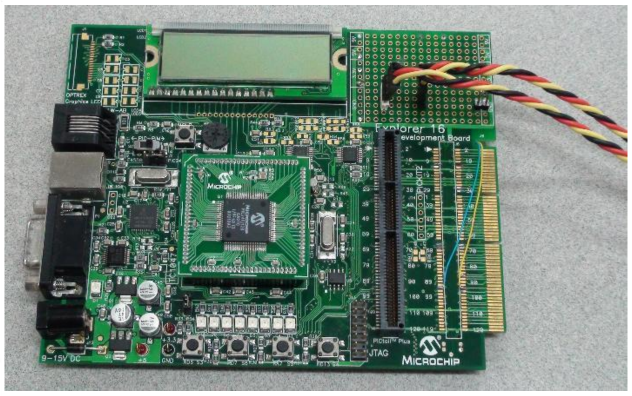


Figure 8. 6: Explorer 16 Board with Two PWM Servo Motor Channels

8.4 MPLAB Integrated Development Environment (IDE)

MPLAB IDE is a free, integrated toolset for the development of embedded applications employing microchip's PIC and dsPIC microcontrollers. The MPLAB IDE runs as a 32-bit application on Microsoft Windows, and includes several free software components for application development, hardware emulation and debugging. MPLAB IDE also serves as a single, unified graphical user interface for additional Microchip and third-party software and hardware development tools. Both Assembly and C programming languages can be used with MPLAB IDE. Others may be supported through the use of third party programs. [29]

The C30 Educational Suite compiler platform was installed together with MPLAB IDE to write programs for the PIC24 microcontroller. [30]

The MPLAB IDE provides two modes namely Debugger and Programmer. The Debugger mode provides debugging capabilities without permanently writing the program into the flash memory of the microcontroller. The Programmer mode writes the tested program permanently into the flash memory of the microcontroller.

8.5 Microcontroller Program

8.5.1 Setting Configuration Bits

Microcontrollers from Microchip contain a series of configurable parameters that are arranged outside of programming. These are more or less special directives built into the program instructions that tell the microcontroller what to do on the most basic level. These may include the parameters for selecting which programming pins to use, alternate pin configurations, watch dog timers, JTAG programming configurations, oscillator selection and many more, depending on the chip. [31]

In the PIC24FJ128GA010 microcontroller, the following configurations shown next are made.

```
//Setting Configuration bits
_CONFIG1( JTAGEN_OFF & GCP_OFF & GWRP_OFF & COE_OFF & FWDTEN_OFF & ICS_PGx2)
_CONFIG2( IESO_OFF & FNOSC_PRI &  FCKSM_CSDCMD & OSCIOFNC_OFF & POSCMOD_XT)
```

The function of each selection is explained below.

JTAGEN_OFF: JTAG Disabled. The JTAG interface provides boundary scan device testing and in-circuit programming. There features are not utilized in this project.

GCP_OFF: Code Protect Disabled. The on-chip memory space is treated as a single block. If Code Protect is enabled, external read and write to the program memory is inhibited.

GWRP_OFF: Write Protect Disabled. When Enabled, internal write and erase operation to program memory are blocked.

COE_OFF: Background Debugger Disabled

FWDTEN_OFF: Watchdog Timer Disabled

ICS_PGx2: ICD pin select – EMUC/EMUD share PGC2/PGD2

IESO_OFF: Two Speed Start-up Disabled

FNOSC_PRI: Oscillator Selection – Primary Oscillator (XT, HS, EC)

FCKSM_CSDCMD: Clock Switching and Clock Monitor – Both Disabled

OSCIOFNC_OFF: OSCO/RC15 function – OSCO or Fosc/2

POSCMOD_XT: XT Oscillator Selected

8.5.2 Setting Ports

The Port A, bits 0 through 7 to which the LEDs are connected in the Explorer 16 board is set as output. The 'TRISA' register controls the operating mode (input or output) of Port A. Using the 'LATA' (Latch A) register, the output values can be controlled. [32]

8.5.3 Oscillator Settings

The 'OSCCON' and 'CLKDIV' registers are used to control the settings of the oscillator. The primary oscillator (XT, HS, EC) is selected to run the microcontroller. The different options possible within the primary oscillator are: [33]

- XT – 3.5 MHz to 10 MHz crystal
- HS – 10 MHz to 32 MHz crystal
- EC – External Clock Input (0 to 32 MHz)

Of these options, XT is selected. The primary oscillator is run at a frequency (Fosc) of 8MHz. The instruction cycle clock frequency (Fy = Fosc/2) is 4MHz.

8.5.4 Timer Settings

The program requires the timer to act as a source for the generation of PWM signals to control the pan and tilt system. The PWM signals have a time period of 20 milli seconds. So the time period of the timer in the microcontroller should be 20 milli seconds.

The Timer2 is selected as the timer used in the microcontroller. The Time Period of the timer is calculated as given below: [34]

Time Period = 20 milli seconds = (PR2 + 1) X Tcy X TMRy pre-scale value; Where:

PR2 $\rightarrow$ 'PR2' is the period register value
Tcy $\rightarrow$ Tcy = 1/Fcy; Fcy is the instruction cycle clock frequency. => Tcy = 1/4MHz = 250 ns.
TMRy pre-scale value $\rightarrow$ Set through the register 'T2CON'. The pre-scale value is 8.

Putting all values, the PR2 value is found out to be = 9999;

The Timer2 interrupt can be used to continuously toggle the 7th LED to give a visual flag that the microcontroller is up and running. The Timer2 interrupt should be enabled to make this happen.

The Timer2 interrupt is enabled, cleared and the priority is set. This is performed in the program using the register bits '_T2IF', '_T2IE'and '_T2IP'.

Selection of the particular clock to run the timer, setting of the pre-scale value and starting the timer is performed using the register 'T2CON'. The internal clock (Fosc/2) is selected to run the Timer2, pre-scale value is set as 8 and the timer is started for enabling further operations (TON = 1).

8.5.5 PWM Settings

The two output compare modules present in the microcontroller are set to generate PWM signals. The 'OC1CON' and 'OC2CON' registers set the Output Compare Module 1 and 2 to work in PWM mode. The clock source is set as Timer2. [35]

The 'OC1R' and 'OC2R' registers control the initial duty cycle of the PWM signals. After the PWM generation has been initiated, any further changes in the duty cycle can be made by manipulating the values of the register 'OC1RS' and 'OC2RS'.

The initial duty cycle of the PWM signals are set at 7.5%. So the initial values of the 'OC1R', 'OC2R', 'OC1RS' and 'OC2RS' registers are set at 750. (750 / 9999 is approximately equal to 7.5%)

The duty cycle values are allowed to change between 5% and 10 % during normal operation. This means the duty cycle register values can change between 500 and 1000.

Given next is the code to perform the initialization of the microcontroller.

```
//.....Oscillator Settings.............................................................
    OSCCON = 0x2200;
                        // Current OScillator Selection = Primary Oscillator (XT, HS, EC).
                        // New Oscillator Selection = Primary Oscillator (XT, HS, EC).
    CLKDIV = 0X0000;    // No Divide
                        //Fosc = 8MHz; Fy (instruction cycle clock) = Fosc/2 = 4MHz

//....Port A Settings..................................................................
                        // Connect to LEDs RA0 to RA7 of PORTA
    TRISA = 0xff00;     // PORTA bits 0 through 7 set as output

//....Initialize the UART2 serial port.................................................
    UART2Init();

//....Initialize Timer 2...............................................................
                        //Set timer period
    PR2 = 9999;         // Timer time period =
                        // (PRy + 1)*Tcy*TMRy prescale value = 20 milli seconds
                        // Tcy = 1/4MHz
                        // TMRy prescale value = 8


//.......PWM settings..................................................................
    OC1RS = 750;         // Intial PWM 1 duty cycle = 7.5%
    OC1R = 750;

    OC2RS = 750;         // Intial PWM 2 duty cycle = 7.5%
    OC2R = 750;

                         // Timer 2 Interrupt enable
    _T2IF = 0;           // Clear Interrupt Flag
    _T2IE = 1;           // Enable Interrupt
    _T2IP = 4;           // Interrupt priority


    _IPL = 0;            // Set processor priority level; default value

    TMR2 = 0;            // Clear the timer

                         //Output compare module 1 - set PWM
    OC1CON = 0x0006;     // Timer 2 is the clock source for output compare;
                         // PWM mode on OC1; Fault pin is disabled
                         // OC1 is multiplexed with pin RD0

                         //Output compare module 2 - set PWM
    OC2CON = 0x0006;     //  Timer 2 is the clock source for the output compare module;
                         // PWM mode on OC2; Fault pin is disabled;
                         // OC2 is multiplexed with pin RD1


                         // Start Timer
    T2CON = 0x8010;      // TON = 1;Prescale value = 8; Internal clock (Fosc/2) selected
```

8.5.6 UART Initialization Function – UART2Init()

The UART2 module is connected to the RS232 translation circuitry in the Explorer 16 board. So UART2 is set in the microcontroller to receive transmitted messages from the host computer. The registers 'U2BRG', 'U2MODE' and 'U2STA' are used to perform the UART2 settings.

The UART2 module is turned on, the receiver and transmitter interrupts are enabled and low baud rate is selected (BRGH = 0). [36]

The baud rate calculations are given below:

Baud Rate = 9600 bits per second (to match the baud rate settings at the host computer side)

Baud Rate = $\frac{Fcy}{16\ X\ (U2BRG+1)}$ = 9600

Therefore, U2BRG = $\frac{Fcy}{16\ X\ Baud\ Rate}$ - 1

Here, Fcy denotes the instruction clock cycle frequency which is 4MHz.

Putting values, U2BRG = 25.0417 approximately equal to 25 (integer value). Because of this approximation, there is an error in the resultant baud rate.

Resultant baud rate = $\frac{Fcy}{16\ X\ (U2BRG+1)}$ = 4 M/(16 X (25 + 1)) = 9615.4 bits per second

Baud Rate Error = $\frac{\text{(Resultant Baud rate – Desired Baud Rate)}}{\text{Desired Baud Rate}} = \frac{(9615.4 - 9600)}{9600}$ = 0.16 % (negligible)

Given next is the code of the function which performs UART initialization.

```
//UART2 initialization function
void UART2Init(){

//Set up registers

U2BRG =  25;          //Baud Rate Speed
U2MODE = 0x8000;      //Turn on uart2 module; BRGH = 0(low baud rate select)
U2STA = 0x8400;       //Set Receive and Transmit Interrupts;UART2 transmitter enabled

//Reset RX interrupt flag
_U2RXIF = 0;

}
```

8.5.7 UART Receive Character Function – UART2GetChar()

This function receives each character which arrives at the UART of the microcontroller from the host computer's RS232 port and passes it for processing.

The function waits for the receive interrupt to be triggered (_U2RXIF = 1). When the interrupt is triggered it copies the character received by the UART2 register 'U2RXREG' to a character variable

(Ch) which is then passed on to the main program. The interrupt which was triggered is cleared after this operation (_U2RXIF = 0). Given next is the code implementation of the function.

```
//UART2 receive function; Returns the value received
char UART2GetChar (){

char Ch;

//wait for Receive buffer to fill up; wait for receive interrupt
while (_U2RXIF == 0);
Ch = U2RXREG;

//Reset the interrupt
_U2RXIF = 0;

//Return the byte received
return(Ch);

}
```

8.5.8 Microcontroller Program Main Loop

The main loop in the microcontroller program performs operations based on the different types of input characters received from the host computer. It waits until the UART receives a character and then performs the following operations on the PWM channels:

Received Character	
‘p’ ‘t’	GetDataFromUART2() $\rightarrow$ PassDataToPanTiltSystem()
‘a’ ‘s’ ‘d’ ‘f’ ‘g’	Calibrate PWM channel 1 (Tilt Servo Motor)
‘z’ ‘x’ ‘c’ ‘v’ ‘b’	Calibrate PWM channel 2 (Pan Servo Motor)

Given next is the code implementing the main loop in the program.

```
 while(1){
     Ch[0] = UART2GetChar();
     if(Ch[0] == 'p' || Ch[0] == 't'){
          GetDataFromUART2();
          PassDataToPanTiltSystem();
     }
     else if(Ch[0] == 'a'){
          oc1rs_value = 300;
          OC1RS = oc1rs_value;
     }
     else if(Ch[0] == 's'){
          oc1rs_value = 310;
          OC1RS = oc1rs_value;
     }
     else if(Ch[0] == 'd'){
          oc1rs_value = 750;
          OC1RS = oc1rs_value;
     }
     else if(Ch[0] == 'f'){
          oc1rs_value = 1350;
          OC1RS = oc1rs_value;
     }
     else if(Ch[0] == 'g'){
          oc1rs_value = 1370;
          OC1RS = oc1rs_value;
     }

    else if(Ch[0] == 'z'){
         oc2rs_value = 290;
         OC2RS = oc2rs_value;
    }
    else if(Ch[0] == 'x'){
         oc2rs_value = 300;
         OC2RS = oc2rs_value;
    }
    else if(Ch[0] == 'c'){
         oc2rs_value = 750;
         OC2RS = oc2rs_value;
    }
    else if(Ch[0] == 'v'){
         oc2rs_value = 1330;
         OC2RS = oc2rs_value;
    }
    else if(Ch[0] == 'b'){
         oc2rs_value = 1345;
         OC2RS = oc2rs_value;
    }


}
```

8.5.9 Timer2 Interrupt Function

The Timer2 Interrupt is used to toggle the 7th LED. This will give a visual cue that the Timer is in fact running. The interrupt is cleared after each operation of the function. The code implementing the function is given next.

```
void _ISR _T2Interrupt(void){


    led_mask = led_mask ^ 0b10000000;
    LATA = led_mask;

    _T2IF = 0; // Clear interrupt flag
}
```

8.5.10 Get Pan and Tilt Value From Host – GetDataFromUART2()

The format in which the values are sent from the host computer was discussed previously. If the UART2 of the microcontroller receives a ‘p’ or ‘t’, it calls the GetDataFromUART2() function where the UART collects the next three numerical characters and calculates the actual position values in number format (int). The code implementing the function is given next.

```
void GetDataFromUART2(void){


    Ch[1] = UART2GetChar();
    Ch[2] = UART2GetChar();
    Ch[3] = UART2GetChar();


    position_value = Ch[3] - '0';
    position_value = position_value + ((Ch[2] - '0')*10);
    position_value = position_value + ((Ch[1] - '0')*100);

}
```

8.5.11 Pass Position Value to Pan and Tilt System – PassDataToPanTiltSystem()

This function passes position values to the pan and tilt system. It ensures that the position value remains between 500 and 1000. If it is within the allowed value range, the position value is written into the ‘OC1RS’ or ‘OC2RS’ registers according to whether the value follows a ‘t’ (Tilt) or ‘p’ (Pan).

The first or second LEDs are toggled to show the reception of a Pan or Tilt position value respectively. The code is given next.

```
void PassDataToPanTiltSystem(void){

    if(Ch[0] == 't'){
        if(position_value > 500 && position_value < 1000)
        {
            oc1rs_value = position_value;
            OC1RS = oc1rs_value;
            led_mask = led_mask ^ 0b00000001;
        }
    }
    else if(Ch[0] == 'p'){
        if(position_value > 500 && position_value < 1000)
        {
            oc2rs_value = position_value;
            OC2RS = oc2rs_value;
            led_mask = led_mask ^ 0b00000010;
        }

    }

}
```

8.6 Mapping the PWM Duty Cycle to the Servo Motor's Angle of Rotation

The PWM duty cycle is allowed to exist in the range of 5% to 10%. Within this range, each duty cycle value is mapped to specific angular positions of the servo motor. The mapping is as shown in table 8.1:

Table 8. 1: Mapping PWM Duty Cylce to the Servo Motor's Angle of Rotation

Duty Cycle (%)	**OCxRS**	**Angle in degrees (relative)**
5	500	0
6.25	625	90
7.5	750	180
8.75	875	270
10	1000	360

The angle of rotation is relative to the initial calibrated position.

9. MVC++ Program Additional Functions

9.1 Calibration of the Pan and Tilt System

The calibration of pan and tilt can be accessed from the main program in MVC++. In the calibration mode, the following actions shown by the tables 9.1 and 9.2 can be performed to align both the pan and tilt servo motors in an initial position which satisfies the project requirements. The figure 9.1 shows the direction of anti-clockwise rotation of both the pan and tilt systems.

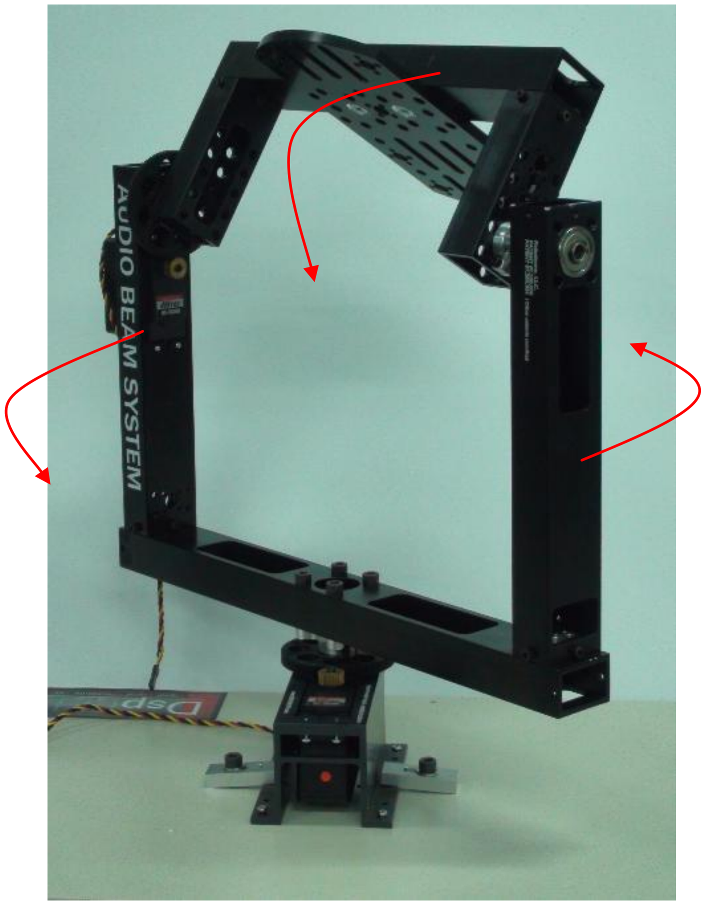


Figure 9. 1: Direction of Anti-Clockwise Rotation of the Pan and Tilt System

Calibrate Tilt Servo Motor

Table 9. 1: Calibrate Tilt Servo Motor

Character Pressed	**Duty Cycle (%)**	**OC1RS**	**Action**
‘a’	3	300	Rotate Clock Wise (Not positional mapping)
‘s’	3.1	310	Minimum Positional Mapping
‘d’	7.5	750	180 degree position (relative – positional mapping)
‘f’	13.5	1350	Maximum Positional Mapping
‘g’	13.7	1370	Rotate Counter Clock Wise (Not positional mapping)

Calibrate Pan Servo Motor

Table 9. 2: Calibrate Pan Servo Motor

Character Pressed	**Duty Cycle (%)**	**OC2RS**	**Action**
‘z’	2.9	290	Rotate Clock Wise (Not positional mapping)
‘x’	3	300	Minimum Positional Mapping
‘c’	7.5	750	180 degree position (relative – positional mapping)
‘v’	13.3	1330	Maximum Positional Mapping
‘b’	13.45	1345	Rotate Counter Clock Wise (Not positional mapping)

Given next is the code making it possible to calibrate the pan and tilt servo motors from the MVC++ program running on the host computer.

```
//....................................Calibrate Pan and Tilt...............................
        if(selection == 4){
            char key[2];
            key[0] = '1';
            key[1] = '\0';
            if(!enable_port){
                printf("COM port is not enabled ... !\n");
                key[0] = '0';
            }
            while(key[0] != '0'){

                if((key[0] == 'a') || (key[0] == 's') || (key[0] == 'd') || (key[0] == 'f') ||
                    (key[0] == 'g') || (key[0] == 'z') ||(key[0] == 'x') || (key[0] == 'c') ||
                    (key[0] == 'v') || (key[0] == 'b'))
                    print_to_port = WriteFile(hPort,key,strlen(key),&byteswritten,NULL);

                key[0] = cvWaitKey(0);
            }
            state = 1;
            selection = 0;
            printf("Normal Program Enabled\n");

        }
```

9.2 Manual Control of the Pan and Tilt System

The pan and tilt system can be manually controlled from the MVC++ program. After entering into the manual control mode, the actions shown by the table 9.3 can be performed.

Table 9. 3: Manual Control of the Pan and Tilt System

Character Pressed	**Action**
‘a’	Pan Counter Clock Wise
‘s’	Pan Clock Wise
‘d’	Tilt Counter Clock Wise
‘f’	Tilt Clock Wise

The codes implementing the manual control in the program developed in MVC++ is given next.

```
//.........................................Manual control of Pan and Tilt...................
        if(selection == 3){
            int key = '1';
            if(!enable_port){
                printf("COM port is not enabled ... !\n");
                key = '0';
            }
            while(key != '0'){
                if(key == 'a')
                    pan_value = pan_value + 1;
                else if(key == 's')
                    pan_value = pan_value - 1;
                else if(key == 'd')
                    tilt_value = tilt_value + 1;
                else if(key == 'f')
                    tilt_value = tilt_value - 1;
                if((key == 'a')||(key == 's')||(key == 'd')||(key == 'f')){
                    convert_values();
                    print_to_port = WriteFile(hPort,pan,strlen(pan),&byteswritten,NULL);
                    print_to_port = WriteFile(hPort,tilt,strlen(tilt),&byteswritten,NULL);

                }
                key = cvWaitKey(0);
            }
            state = 1;
            selection = 0;
            printf("Normal Program Enabled\n");
        }
```

9.3 Different Modes of Operation of the MVC++ Program

Different functionalities can be achieved by changing the mode of operation of the program running on the host computer.

Pressing ‘0’ selects the ‘Normal Mode’ of operation where the normal target tracking program is running. Other selectable options are given in the table 9.4.

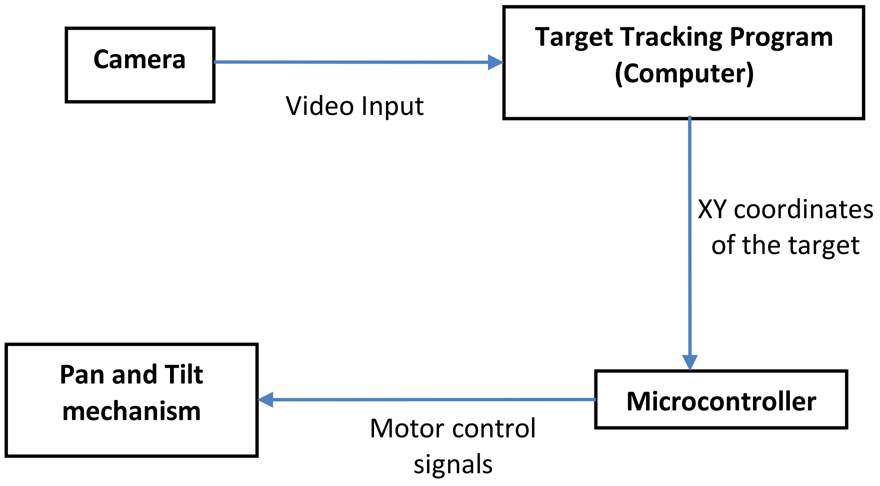
Table 9. 4: Different Modes of Operation of the MVC++ Program

Character Pressed	**Action**
‘1’	Face detection on entire frame is enabled
‘2’	Face detection on ROI only, is enabled
‘3’	Manual control of pan and tilt system is enabled
‘4’	Calibration mode of pan and tilt system is enabled
‘5’	Template matching only, is enabled
‘b’	Brings the pan and tilt system back to initial position

10. Conclusion

10.1 Completed Target Tracking with Audio Beaming System

Figure 10.1 show both the functional block diagram and the diagram of the entire system after the completion of the project.



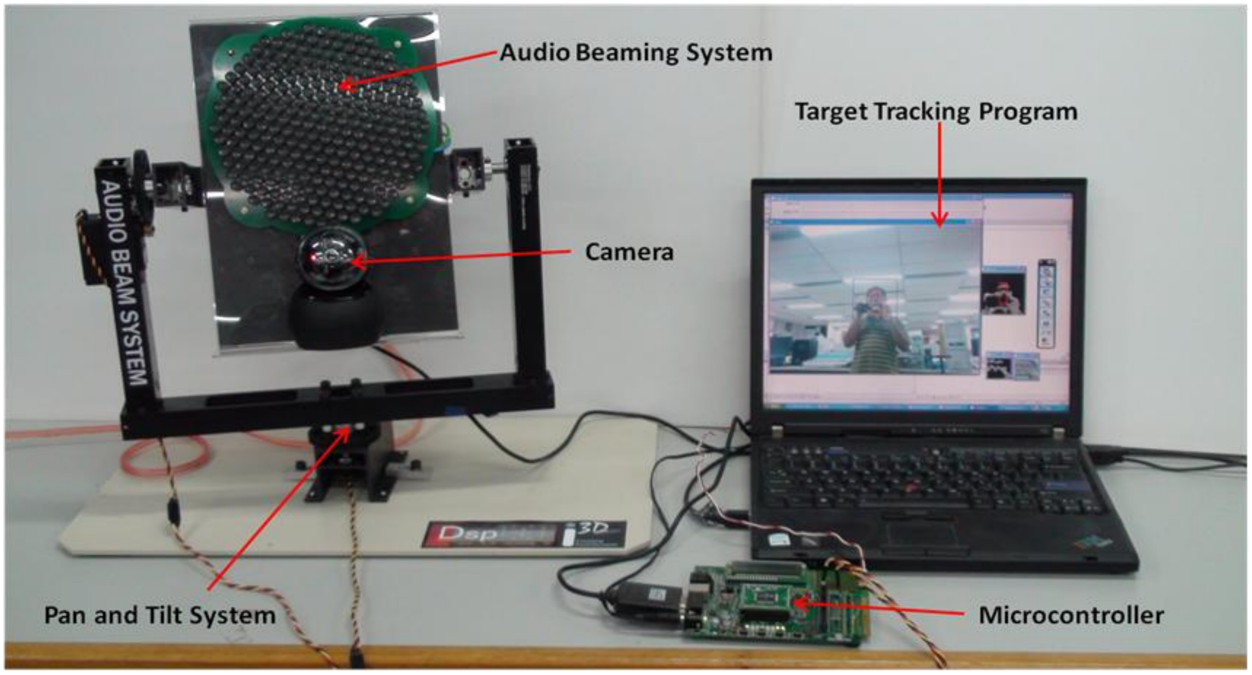


Figure 10. 1: Target Tracking with Audio Beaming System after Completion of Project

The Target Tracking Program runs on the host computer. The program takes in live video input from the web camera and run computer vision codes on it. The computer vision codes analyzing the live video will detect and track a target face on the scene. The corresponding position data will be handed over to the microcontroller through the COM port. The microcontroller produces PWM control signals to drive the motorized pan and tilt system.

By mounting the Audio Beaming System on the pan and tilt mechanism, the audio beam can be pointed continuously at the moving target on the scene.

10.2 Challenges

The main challenges in the execution of the project were:

- **The coding of a robust program for face detection and tracking on a real time video**
- **The selection of the right type of pan and tilt system**
- **The selection of the microcontroller to generate the PWM control signals**
- **The interfacing of the microcontroller with the host computer for the transmission of control signals from the Microsoft VC++ program to the microcontroller**

10.3 Project Exhibition and Demonstration

This project was exhibited in the iJam session held at Fusionopolis, Singapore, on the 25th of January, 2010. iJam is the flagship event organized by the Media Development Authority (MDA) of Singapore.

This project was also exhibited at the Discover Engineering @ NTU event, held as a part of the NTU Open House, at Nanyang Auditorium on the 13th of March, 2010.

10.4 Limitations

The program used for the face detection and tracking is based on the open source computer vision library, OpenCV. It has the limitation of having slower computational speeds than proprietary alternatives.

The pan and tilt system uses feedback from the face tracking program to position the audio beaming system on the target face. There is a significant time lag between the capture of a video frame by the camera and the passing of target positions after frame processing to the microcontroller. At fairly high rotation speeds of the servo motor, the pan and tilt system tend to oscillate before damping the target face to the centre of the frame due to the high time lag. This puts a limitation in the speed with which the pan and tilt servo motors can be rotated.

10.5 Future Improvements

10.5.1 Incorporate the Intel IPP into the Project

The computational speeds of the face detection and tracking codes can be improved significantly by incorporating proprietary multimedia libraries like the Intel Performance Primitives (IPP).

The Intel IPP is a threaded library of functions for multimedia and data processing applications, produced by Intel. The library supports Intel and compatible processors and is available for Windows. Using Intel IPP optimization functions together with OpenCV, program execution times can be improved tremendously. [37][38]

This can solve the oscillation problem experienced with the pan and tilt motors at fairly large speeds. Hence fast detection and tracking of the target faces and fast following by the pan and tilt system will be possible.

10.5.2 Use Camera with Optical Zoom Capability

Currently OpenCV supports only selected web cameras. When the OpenCV support increases fairly large to include cameras with optical zoom capabilities, they can be used in the project instead of normal web cameras. Cameras with optical zoom capabilities can tremendously improve the target face detection and tracking process.

References

[1] http://pcmedia.gamespy.com/pc/image/article/561/561380/shadow-ops-red-mercury-20041028003252925-000.jpg

[2] http://microsoftfeed.com/wp-content/uploads/2009/10/ProjectNatal.jpg

[3] http://www.dustinkirk.com/blogpicsBig/Project_Natal.jpg

[4] http://www.oled-display.net/images/lgdisplay/lg-display-world-lowest-power-tv.jpg

[5] http://static.guim.co.uk/sys-images/Arts/Arts_/Pictures/2009/8/11/1250004374020/Leonardos-Mona-Lisa-at-th-002.jpg

[6] http://en.wikipedia.org/wiki/OpenCV

[7] http://opencv.willowgarage.com/wiki/InstallGuide

[8] http://en.wikipedia.org/wiki/Microsoft_Foundation_Class_Library

[9] http://n2.nabble.com/Problems-on-Win7-SxS-issue-Missing-dependency-td3420025.html

[10] http://www.cognotics.com/opencv/servo_2007_series/part_3/sidebar.html

[11] http://opencv.willowgarage.com/wiki/VideoSurveillance

[12] http://en.wikipedia.org/wiki/Optical_flow

[13] http://opencv.willowgarage.com/wiki/FaceRecognition

[14] http://www.cognotics.com/opencv/servo_2007_series/part_4/index.html

[15] http://www.science.az/cyber/pci2008/2/2-26_1.doc

[16] http://dasl.mem.drexel.edu/~noahKuntz/openCVTut7.html#Step%202

[17] http://www.andol.info/wp-content/uploads/2009/03/handdetection.cpp

[18] http://en.wikipedia.org/wiki/Haar-like_features

[19] http://opencv.willowgarage.com/wiki/FaceDetection

[20] http://nashruddin.com/OpenCV_Region_of_Interest_(ROI)

[21] http://nashruddin.com/template-matching-in-opencv-with-example.html

[22] http://software.intel.com/sites/oss/pdfs/OpenCVreferencemanual.pdf

[23] http://opencv.willowgarage.com/documentation/object_detection.html#cvHaarDetectObjects

[24] http://opencv.willowgarage.com/documentation/object_detection.html#cvHaarDetectObjects

[25] http://cognotics.com/opencv/servo_2007_series/part_2/page_2.html

[26] http://www710.univ-lyon1.fr/~bouakaz/OpenCV-0.9.5/docs/ref/OpenCVRef_ImageProcessing.htm

[27] http://opencv.willowgarage.com/documentation/image_filtering.html

[28] http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html

[29] http://en.wikipedia.org/wiki/MPLAB

[30]http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en535364

[31] http://ww1.microchip.com/downloads/en/DeviceDoc/39747e.pdf

[32] http://ww1.microchip.com/downloads/en/DeviceDoc/39711b.pdf

[33] http://ww1.microchip.com/downloads/en/DeviceDoc/39700b.pdf

[34] http://ww1.microchip.com/downloads/en/DeviceDoc/39704a.pdf

[35] http://ww1.microchip.com/downloads/en/DeviceDoc/39706a.pdf

[36] http://ww1.microchip.com/downloads/en/DeviceDoc/en026583.pdf

[37] http://software.intel.com/en-us/articles/intel-integrated-performance-primitives-intel-ipp-open-source-computer-vision-library-opencv-faq/

[38] http://en.wikipedia.org/wiki/Integrated_Performance_Primitives

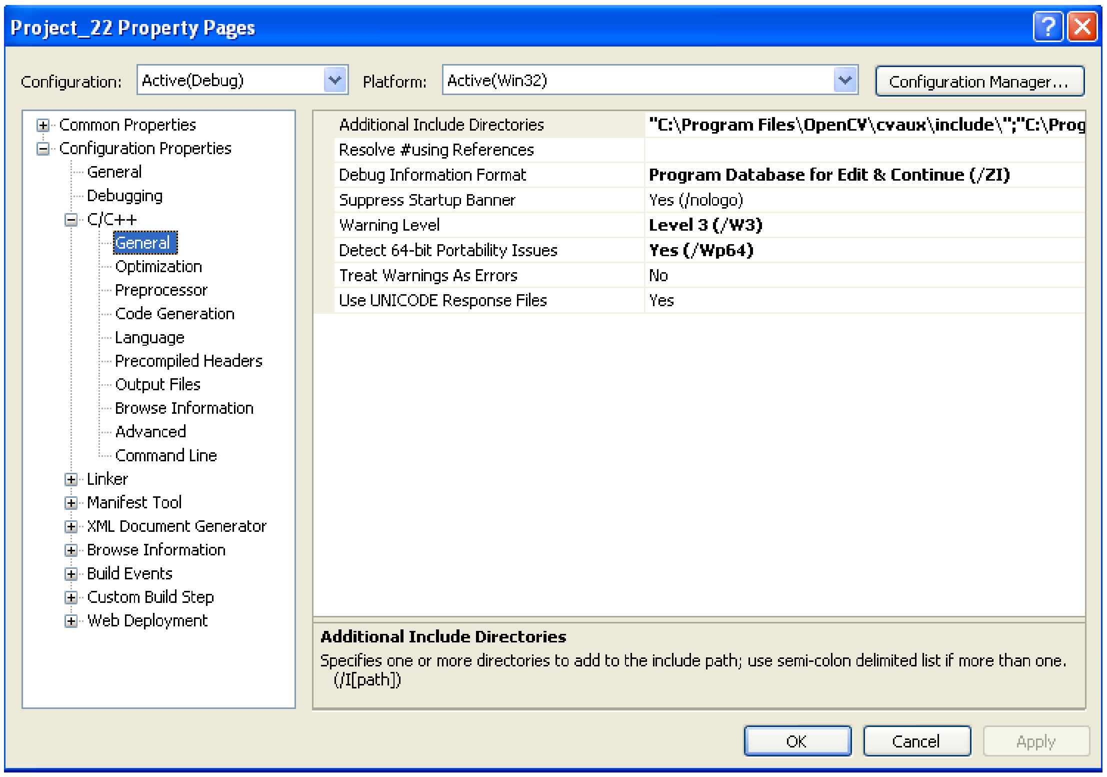
Appendix A

Inclusion of path to OpenCV in Microsoft Visual C++

Select Project -> Configuration Properties -> C/C++ -> General -> Additional Include Directories

The values shown below should be pasted into the selected field.

```
"C:\Program Files\OpenCV\cvaux\include\";"C:\Program
Files\OpenCV\cxcore\include\";"C:\Program
Files\OpenCV\cv\include\";"C:\Program Files\OpenCV\otherlibs\highgui\";
```



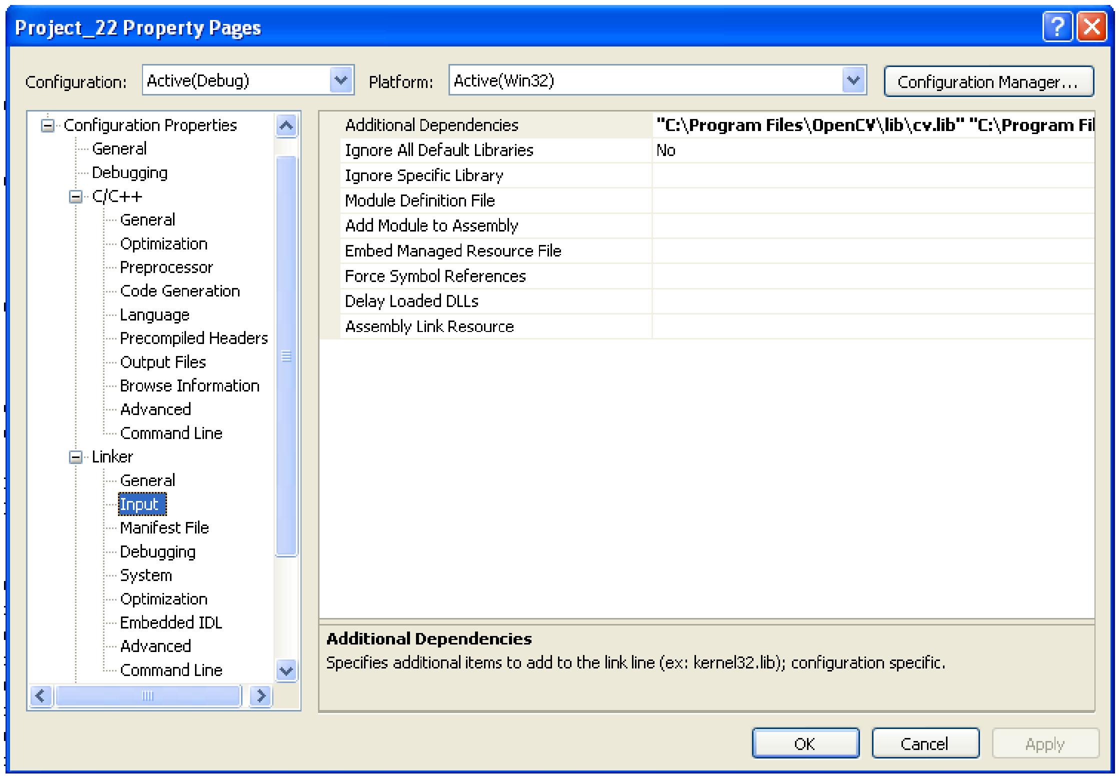
Additional Include Directories	"C:\Program Files\OpenCV\cvaux\include\";"C:\Prog
Resolve #using References	
Debug Information Format	Program Database for Edit & Continue (/ZI)
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level 3 (/W3)
Detect 64-bit Portability Issues	Yes (/Wp64)
Treat Warnings As Errors	No
Use UNICODE Response Files	Yes

Additional Include Directories
Specifies one or more directories to add to the include path; use semi-colon delimited list if more than one. (/I[path])

Select Project -> Configuration Properties -> Linker -> Input -> Additional Dependencies

The values shown below should be pasted into the selected field.

```
"C:\Program Files\OpenCV\lib\cv.lib" "C:\Program Files\OpenCV\lib\cvaux.lib"
"C:\Program Files\OpenCV\lib\cxcore.lib" "C:\Program
Files\OpenCV\lib\highgui.lib"
```



Additional Dependencies	"C:\Program Files\OpenCV\lib\cv.lib" "C:\Program Fil
Ignore All Default Libraries	No
Ignore Specific Library	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

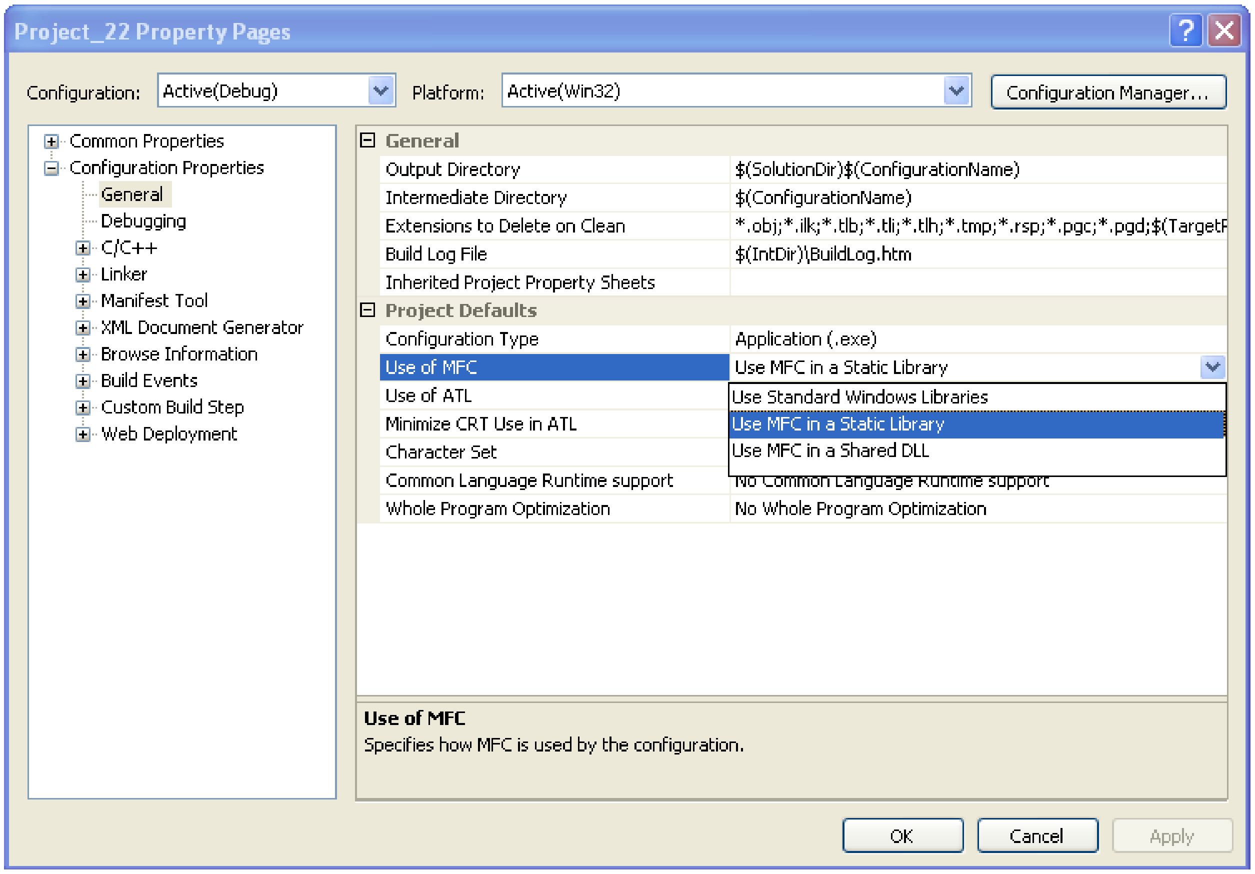
Appendix B

Inclusion of Microsoft Foundation Classes (MFC) in the MVC++ Project

`To use the Microsoft Foundation Classes (MFC)`

`Select Project -> Configuration Properties -> General -> Project Defaults -> Use of MFC`

`Select either "Use MFC in a Static Library" or "Use MFC in a Shared DLL"`



General	
Output Directory	$(SolutionDir)$(ConfigurationName)
Intermediate Directory	$(ConfigurationName)
Extensions to Delete on Clean	*.obj;*.ilk;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.pgc;*.pgd;$(TargetF
Build Log File	$(IntDir)\BuildLog.htm
Inherited Project Property Sheets	
Project Defaults	
Configuration Type	Application (.exe)
Use of MFC	Use MFC in a Static Library
Use of ATL	Use Standard Windows Libraries
Minimize CRT Use in ATL	Use MFC in a Static Library
Character Set	Use MFC in a Shared DLL
Common Language Runtime support	No Common Language Runtime support
Whole Program Optimization	No Whole Program Optimization

Use of MFC
Specifies how MFC is used by the configuration.