**VIDEO LOCALIZATION USING ARRAY OF MICORPHONES**

By

YASIR SALIH OSMAN

FINAL PROJECT REPORT

Submitted to the Electrical & Electronics Engineering Programme
in Partial Fulfillment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Universiti Teknologi Petronas
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

# CERTIFICATION OF APPROVAL

**VIDEO LOCALIZATION USING ARRAY OF MICROPHONES**

by

Yaisr Salih Osman

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Approved:

_____

Mr. Patrick Sebastian
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK

December 2009

## CERTIFICATION OF ORIGINALITY

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

_____

Yasir Salih Osman

# ABSTRACT

*Object localization helps to improve the surveillance system and provide extra information from the camera. In this study DCS 5220 PT camera and array of microphones are used to achieve camera localization based on audio inputs, the system consists of microphones, multiplexing circuit, computer and PT camera. The results obtains clearly demonstrates that this techniques is usable for video-discussion applications.*

*The localization algorithm is based on comparing the amplitude level of the signals from all microphones and then selects the one with higher amplitude, strong noise presence is one of the limitations of this techniques, however the type of noise add can be clearly identified in the frequency spectrum, and for this reason signals are compared in their frequency spectrum after the noise is been trimmed off. The DCS 5220 PT camera is IP camera manufactured by Dlink, it can pan and tilt by using URL command; to utilize the camera a network is been created by using DR 300 wireless router with data rate up to 54 Mbps and can accommodate up to 4 network elements.*

*The system is been developed for closed room localization but it can also be extended to outdoor applications by using suitable sensors and suitable cameras, in video discussions the system can be used in conferences and in parliament halls where such a system can be used as an automatic cameraman.*

# ACKNOWLEDGEMENTS

By the end of this project first of all I thank the almighty GOD, whom I believe without his blessings this project will not be at this stage. I have not forgetting also my parents who despite the distance always cheer me and encourage me to work more.

Special thanks to my supervisors Dr. Yap and Mr. Patrick Sebastian who were always there whenever I need any help or clarifications. I will not forget also to thank my fellow friends who stood by during the long nights of this project, special M. Tag, Huzaifa, Muhand, Adel Aziz and all others who I enjoyed their company during my four years degree.

Special thanks also dedicated to all the provider of open source code which I used in this project; Jeff Morton, Andrew Kirillov and microchip solution demo codes. Lastly I will not forget to thank Universiti Teknologi PETRONAS for providing me the environment to develop such project and to pursue knowledge for my future betterment.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

PTZ: pan tilt and zoom.

PT: pan and tilt.

CCTV: closed-circuit television.

IR: infrared.

BPP: bits per pixel.

SDK: software development kit.

RGB: red, green and blue.

HSV: hue, saturation and value.

CPU: central processing unit.

I/O: input/output.

PIC: peripheral interface controller.

USB: universal serial bus.

SIE: serial interface engine.

FFT: fast Fourier transforms.

IFFT: inverse fast Fourier transform.

# CHAPTER 1
# INTRODUCTION

In current videoconferencing environment a set of human-controlled cameras been used in different locations to cover the active speaker in an interactive discussion room, this tedious work involve the use of professional camera operators to be dedicated to track the active speaker as they contribute to the discussion. Reducing this costly process can be achieved by using automatic video localization system that depends on array of microphones allocated in front of the speakers; such a system is capable of located the active speaker in 3D- space and therefore; it eliminate the needs of professional camera operators as well as the need for multiple cameras, and this results in reducing the total cost involve and makes the system more accurate [13].

## 1.1 Background of study

This study was initially designed to investigate the use of audio sensors to monitor an outdoor car park, using vehicles sound, however due to the fact that most vehicles today produce low amplitude audio signals and also there is a strong presence of noise and interference in the parking are; the objectives of the study has been redefined to video discussion rooms.

There are many localization techniques available and each has its own features and limitations that suits different applications such car park, discussion rooms busy seen and many other applications. In discussion rooms audio is a clue for activeness and presence of new event, this project is evaluation the use of audio sensor (microphones) and pan, tilt (PT) cameras for localization purposes. Speaker localization in video discussions is obtain by using profession camera operators who point their cameras at the speaker and display him/her in the project screen for the audience, in outdoor applications vehicle localization helps to monitoring the car park and inspects any suspicious movements; different applications used different localization techniques based on what are the activeness clues available at that situation.

## 1.2 Problem statement

In interactive video discussions such as conference halls uses human controlled camera to identify the location of the speaker and highlight him/her to the audience in the projector screen, this is a costly work and it involves the use of multiple cameras and professional camera operators who are highly paid. In addition to the above sometimes it is very difficult for the camera operator to gauge where the sound is coming from.

## 1.3 Objectives and scope of the study

The objectives of this study are:

- Develop a surveillance system using the method used in video conferencing applications by utilizing PT camera and microphones.
- Study the possibility of using this technique in outdoor environment.
- Develop a robust audio processing system in order to use microphones as motion sensors.

The scope of the study focuses on the audio processing part as a key of the system success; audio process encloses all the filtering, noise suppression, multiplexing and de-multiplexing operations, in addition to that there is USB communication link for synchronization purposes. Beside the audio processing the study cover the controlling of a PT camera to cover desired locations where active events and specious motions take place.

# CHAPTER 2
# LITERATURE REVIEW

This chapter describes the main reading materials that help in developing this project; the materials were from technical papers as well as websites. The materials describe the USB interface, audio capturing, audio visualization, audio processing, Fourier transformations as well as IP camera acquisition and visualization. In addition to that there are many open source application been used for the audio capturing, audio processing and also about the GUI design.

## 2.1 Camera Vision- Video Surveillance on C#

In this application Mr. Andrew Kirillov developed a system that can handle most of the existing IP cameras, including Dlink cameras which are used in this project, the application suits single camera viewing as well as multiple cameras viewing simultaneously. It allows simultaneous view of not only several cameras from a single video server, but allows many different cameras of different manufacturers at a time. The range of supported video sources by the application such as [7]:

- Continuously updating JPEG source;
- MJPEG (motion JPEG) streams;
- Some Axis cameras and video servers (205, 206, 2100, 2110, 2120, 2130R, 2400, 2401, 2460);
- D-Link cameras (JPEG support only);
- Panasonic cameras;
- PiXORD cameras;
- StarDot cameras (NetCam, Express 6);
- Local devices, which support the DirectShow interface;
- MMS (Microsoft Media Services) streams.

The article Andrew posts in Code project website [7] describes the application in details with it coding and it describes the video formats of each type of these camera and how his program can retrieve images from it. Figure 1, shows the user interface of the application, this application helps in developing the user interface and communicating with the Dlink camera.

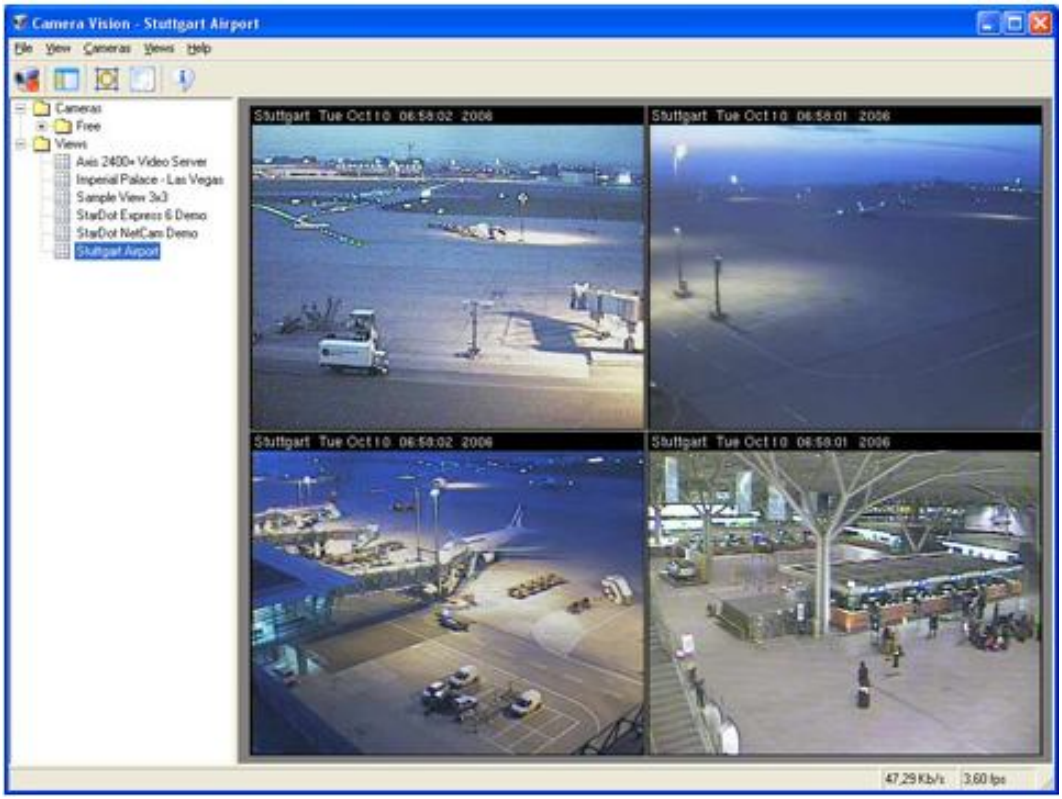


**Figure 1: video surveillance system developed by Andrew Kirillov [7].**

### 2.2 Sound Activated Recorder with Spectrum in C#

This application is been developed by Jeff Morton [6], this application can capture audio data from a microphone and display it in time domain and frequency domain, the application is very robust and the audio data parameters such as sampling rate and number of bits per one sample can be modified during the run time of the application. This application is the base for the audio processing system used in this project; figure 2 shows the user interface of the system showing the audio signal for a dual channel microphone in both time domain and frequency spectrum.

The application contains an FFT function to transform the signal into the frequency

spectrum, in addition to that the user interface contains extra feature to change the audio data settings such as the sampling rate, number of bit per one sample and the number of channels used. Moreover the user can save the recorder audio data and can replay it back, also the user can use internally generated test signals inform of sinusoid waves and square waves to test the system robustness.

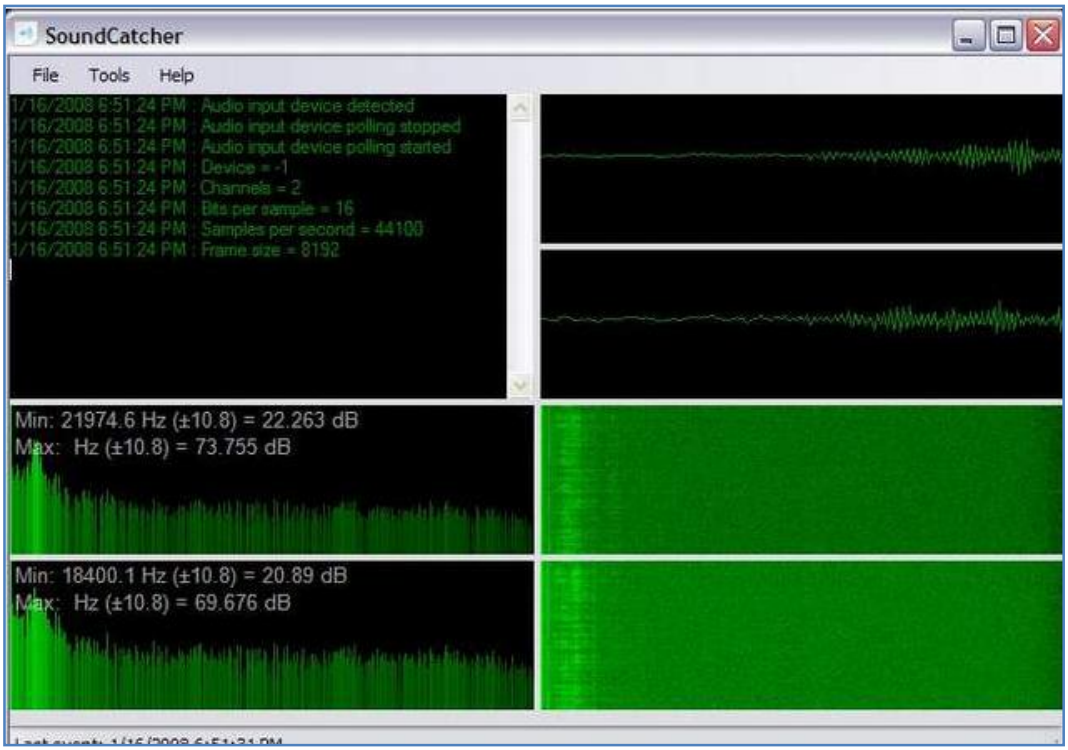


**Figure 2: User interface for the SoundCatcher application developed by Jeff Morton [6].**

### 2.3 Video sources:

In order to localize video images a video source is required. Video sources are cameras as well as video files; however for localization purpose only cameras are valid because localization uses real time images. There are different types of camera available for different purposes; figure 3 shows different cameras and its associated images such as [14]:

- analogue cameras
- PTZ cameras the camera is used to cover wide area by pan, tilt and zoom.
- Wide angle cameras is used to cover wide area in a single image
- IR cameras is used for night vision
- Thermal cameras is used to detect the temperature of the seen object

- Webcams is used for beginner and simple purposes

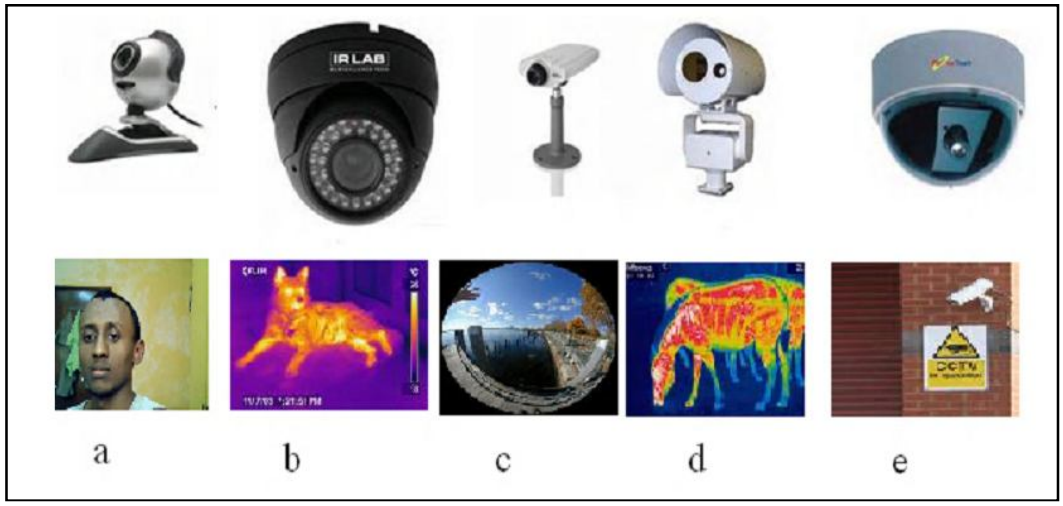


**Figure 3: different types of cameras with its images, (a) webcam, (b) IR camera, (c) fish-eye Len camera, (d) thermal camera, and (e) dome CCTV camera [14].**

## 2.4 Dlink DCS 5220 PT camera

The D-Link SECURICAM Network DCS-5220 Wireless PT Internet Camera is a powerful surveillance system that connects wirelessly to an 802.11b/g network at a rate of up to 54Mbps. The DCS-5220 differs from a conventional PC Camera because it is a standalone system with a built-in CPU and Web server. The camera has Snapshot feature, light sensitivity of 1 lux as well as digital zooming up to 4x [9][15].

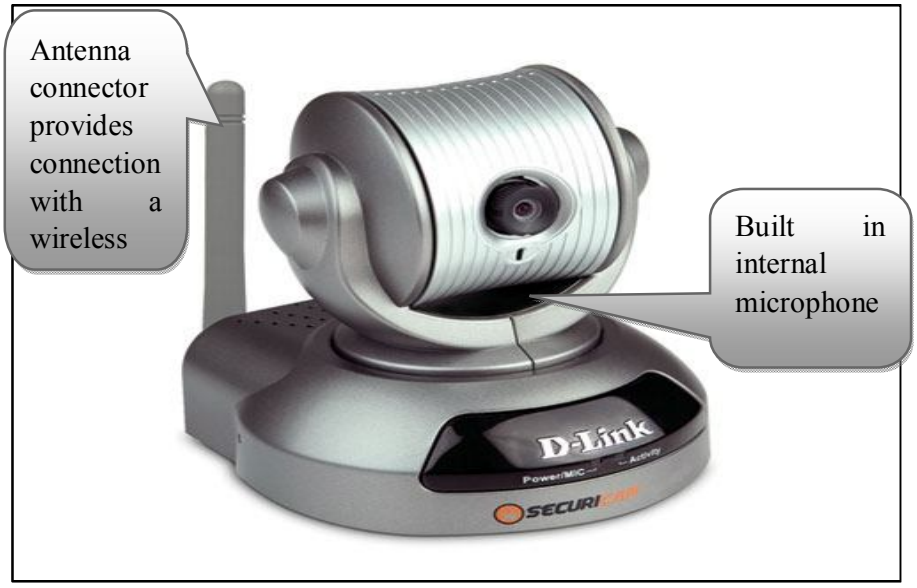


**Figure 4: Dlink DCS 5220 PT camera [9].**

## 2.5 PIC microcontroller:

Microcontroller is a small computer on a single integrated circuit consisting of a relatively simple CPU combined with support functions such as a crystal oscillator, timers, watchdog, serial communication and analog I/O. Microcontrollers are designed for small application that requires small processing capability and uses less memory. A very popular type of microcontroller is the microchip PIC which is known as MCU. There are different types of microcontroller available in the market that suit different applications [18].

### *2.5.1 Microcontroller programming*

Most microcontrollers support the use of different programming languages, such as assembly, C, Basic and Pascal. All programming language generates a .HEX file to be loaded on the microcontroller. For microchip microcontrollers; the general layout of any program can be summarized in the following steps [18]:

1. Include the microcontroller header file, which define all its registers and peripherals.
2. Set the main configurations of the PIC such as the crystal frequency, the watchdog status and other configurations settings.
3. In the main functions, initialize the ports of the PIC by setting its appropriate directions as well as specifying weather its analog input of digital input.
4. The rest of the program depends on the user application.
5. If the programmer decided to use any peripheral or communication module; it has to be properly configured prior its use.

Microcontroller can be programmed using different programming languages, such as assembly and all the structured programming languages like C, FORTRAN, Pascal and Basic. C language is the most common programming language used to program Microchip microcontroller and there are different C compiler used such as:

a. C18 compiler: owned by microchip used to program the 18F series.
b. Hi-tech compiler: which also owned by microchip it can program different microcontroller series.
c. CCS compiler: can be used to program different microcontroller series.

### *2.5.2 Communication between microcontrollers and computers:*

In many applications a communication link is required between microcontrollers and a computer, most PIC supports varies types of communication links; parallel port, serial port as well as USB communication. Parallel port communication is not favorable because it requires the use of multiple pins of the PIC I/O; most likely people use serial communication, specially the RS-232. However USB is getting more popular these days because of its fast speed and high accuracy [11, 16, 8, 17].

#### *2.5.2.1 Parallel interface between PIC and computers*

Old computers are equipped with a parallel interface port that can be used to communicate with a microcontroller, parallel interface send data in bytes by using 8 of the microcontroller pins.

Advantages:

1. It has higher data rate.
2. It does not require specific module at the microcontroller.

Disadvantages:

1. It uses more pins from the microcontroller.
2. It has a limited distance.
3. Most new computers do not support parallel interfaces.

#### *2.5.2.2 RS232 communication between PIC and computers*

It is an asynchronous serial communication protocol been developed by Philips Inc, it is a three wire communication transmission, reception and ground wires as it has been shown in figure 5 [11,16].

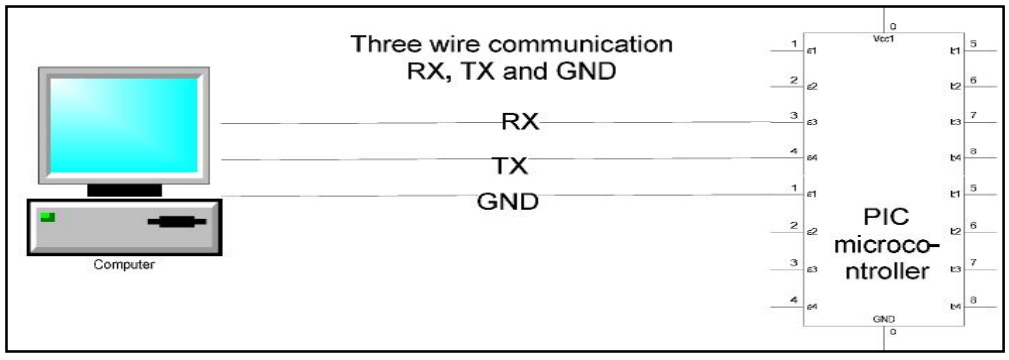


**Figure 5: Generic diagram for RS-232 communication between PIC and host computer.**

Advantages of RS232:

1. It uses only two pins from the PIC.
2. Easy to use and to configure.

Disadvantages of RS232:

1. The voltage level at the PIC and the computer is not compatible which requires the use of some interfacing devices such as MAX-232 and MAX-233.
2. Require the use of coupling capacitors.
3. It has a considerable error rate.

*2.5.2.3 USB communication between PIC and computers:*

USB stands for Universal Serial Board; it is a standard communication system used in enormous applications between computer devices as well as computers and other devices such PICs. The PIC 18FX455/X550 device families contains a full speed and a low-speed compatible USB Serial interface Engine (SIE) that allows fast communications between USB host and the PIC microcontroller. USB is considered as a three wire communication (D+, D- and GND), and instead of having TX and RX like other serial devices it send the signal differentially through positive and negative data terminals. Microchip Inc provides the general layout of microcontroller USB communication and it is available as open source at microchip website [16][12][8].

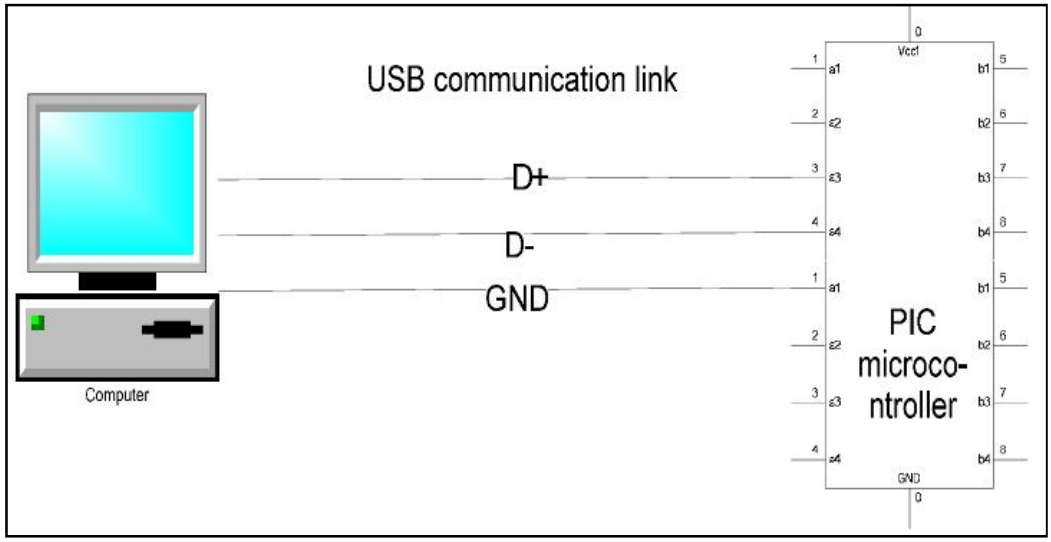


**Figure 6:** Generic diagram for USB communication between PIC and host computer.

*Advantages of USB communication*

1. High speed serial communication
2. Accurate and reliable link.
3. It uses a fewer pins form the PIC.
4. Compatible voltage level between PIC and host computer.

*Disadvantages of USB communication*

1. Require more complex coding.
2. Require a pre-knowledge about USB communication.

## 2.6 Microchip Communication Driver Class (CDC)

Microchip communication class driver or (CDC) is a free demo code provided by Microchip to help developers develops USB applications. CDC comes along with other demo codes in Microchip solution library, the code defines the USB board as a COM board, which makes it easier for user to use it, in addition the code is also equipped with the necessary INF file that define the USB device to the computer [11]. In this project the CDC program was used to develop the USB communication between the computer and the microcontroller after modifying some function to suit the application requirements.

# CHAPTER 3
# METHODOLOGY

This chapter describes the work related to this project in details. There are three major work areas in this project works on; audio processing, image visualization and communication between microcontroller and computer. This chapter will investigate the methods used in each one of these areas.

## 3.1 Audio processing:

Audio inputs are microphones, four microphone are used this study, there is no amplification stage and the audio data is directly linked to the computer. Normal computer has one sound card and can accommodate one audio input; therefore to accommodate all these sources into one computer a multiplexing technique is used and all the microphone are connected to a multiplexer which connect to the computer. Multiplexer allows the microphones data to be fed sequentially to the computer based on predetermined sequence or via instructions given by the computer.

At the computer side the received signal is de-multiplexed to reconstruct the original signals; hence it is very essential to synchronize the multiplexing and the de-multiplexing process so the computer can be able to reconstruct the correct signals. Figure 7 shows a generic model for the audio processing system, the model consist of 4 microphones placed apart and an analog multiplexer to multiplex the microphones. In addition to that there is a PIC microcontroller to issue the multiplexer control signals; the output of the analog multiplexer is connected to computer, which manages the multiplexing process by sending the multiplexing periods to the PIC microcontroller.

The multiplexing process goes in sequential follow; at first the computer send command to the microcontroller requesting specific microphone to be opened, then the microcontroller set the appropriate signal on the analog multiplexer control pins

in order to open the required microphone, after that the required microphone is connected to the computer while the others are closed, the computer waits for a propagation time delay and after that it receives the audio signal, this process is repeated for all the microphones rotationally until the system is stopped by the user or the operator.

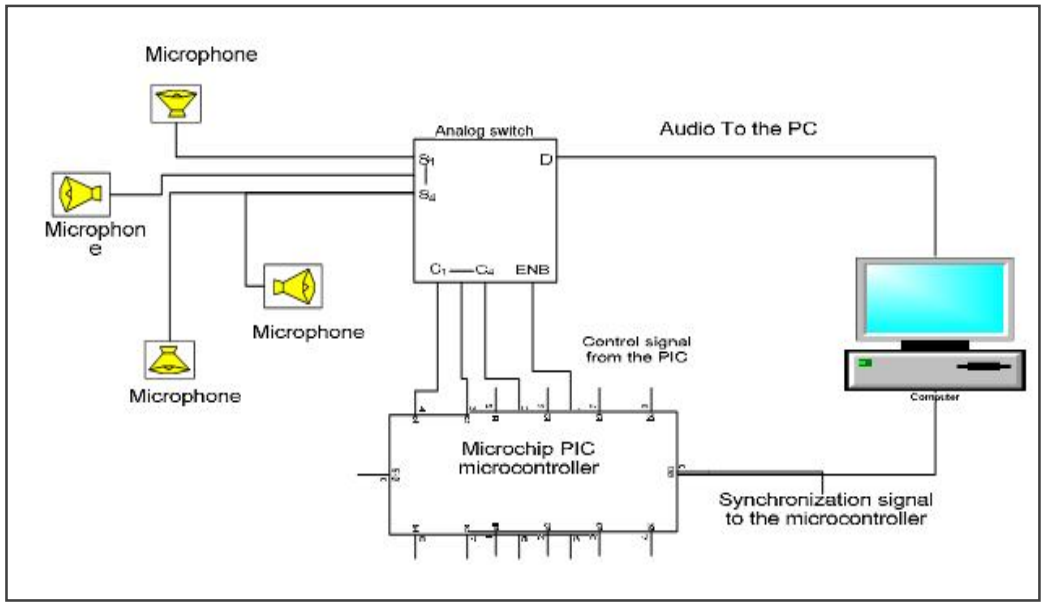


**Figure 7:** Generic diagram of the audio processing system.

### *3.1.1 Hardware part:*

The microphones been used are single channel microphones, the ground PIN of all the microphones is connected together and the signal pin is connected through the multiplexer as shown in the schematic diagram of figure 8. The CD4097 [5] is a dual channel multiplexer, and it can take up to eight inputs at each channel, the multiplexer is controlled by three digital signals C1 – C3(figure 7). Figure 9, illustrates the schematic diagram of the multiplexing audio multiplexing system. The microcontroller been used is the microchip PIC18f4550 and it is function is to issue the multiplexer control signals at a specific time interval given by the computer. These signals are just numbers representing the microphone ID, based on the number given the microcontroller sets the appropriate value of the three control signals of the analog multiplexer as depicted in Table 1, this table is stored in the microcontroller and whenever a microphone ID is received from the computer, the PIC will convert the ID number into appropriate control signals based on this lookup table.

The multiplexer control switches are connected to three pins of port B and are externally pulled up to by 220ohms resistors; this ensures the flow of sufficient current to the multiplexer control pins. The audio output from the multiplexer is pulled down by a small resistor to remove impulse spikes which due to the switching effect in the multiplexer.

**Table 1: decoding the recieved control signal from the computer at the microcontroller.**

| Microphone ID number | Multiplexer control signals | | |
|---|---|---|---|
| | A | B | C |
| 1 | 1 | 1 | 1 |
| 2 | | 1 | 0 |
| 3 | 1 | 0 | 1 |
| 4 | 1 | 0 | 0 |

Microcontroller pins C6 and C7 are connected to the USB socket for USB interface, the whole system is powered up by from the USB port of the computer. Figure 9 shows the complete circuit diagram for the multiplexing system.

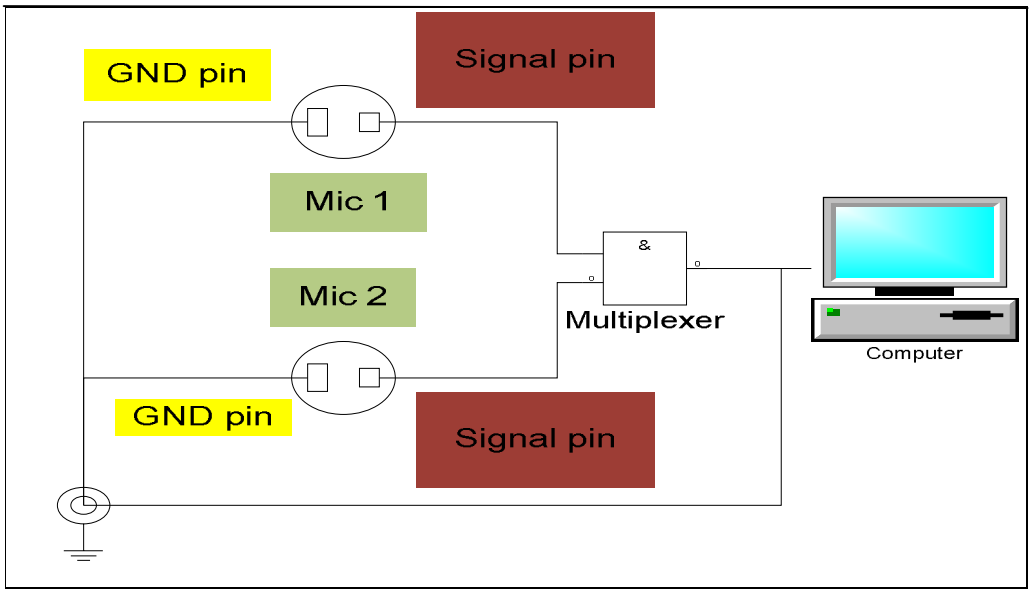


**Figure 8: how microphone is connected to the multiplexer.**

The PIC microcontroller is been programmed using Microchip's C18 compiler, the main code component is the USB interface firmware which been developed by microchip, the Microchip Communication Device Class (CDC), the coding is built for Microchip board the *PICDEM FS USB Demo Board* (based on PIC18F4550) [11], this program contains the main components of how microcontroller and computer can interface. After understanding the basic components of the code, the code is modified to suit the multiplexing requirements; in addition to that there are 5 LED indicators have been to indicate the multiplexing status.

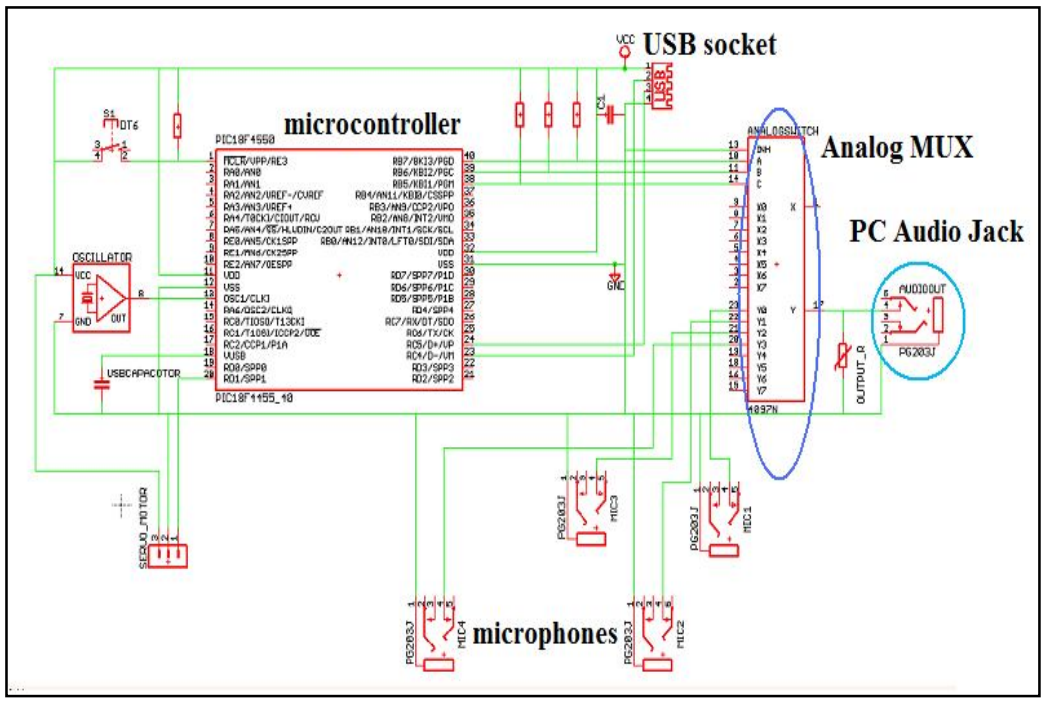


**Figure 9: Schematic diagram for the multiplexing circuit.**

### *3.1.2 Software part:*

The computer receives the input data of all microphones as if it is coming from a single audio source; then it de-multiplexes or resample the original four audio sources, the computer reconstruct four waveforms form the given one based on a known time sequencing, the data is received in a form of audio frames each frame contains 8912 samples and the sampling rate is 192000 sample/second which means there is 23 frame/second, table 2 shows all the predefined setting for the received data, all these values can be modified by the user even during the run time of the program.

**Table 2: Predefined settings for microphones data.**

| Attribute | Value |
|---|---|
| Sampling rate | 192000 samples/second |
| Samples per frame | 4096 samples of 16 bit samples |
| Frame rate | 23 frame per second |
| Maximum allowed frequency | 10KHz |
| Minimum allowed frequency | 50Hz |
| Bits per sample | 16 bits |
| Number of audio channels | 1 |
| Multiplexing rate | 250 milliseconds |
| Propagation delay | 50 milliseconds |

Figure 10 shows the flow of multiplexing program, it start by issuing a multiplexing signal to the PIC through the USB communication link, the signal contains the ID of the microphone that should be connected. Since there is going to be a propagation delay from the time the multiplexing signal is been issued until the audio data arrives; any data comes during this time will be discarded. The propagation delay time is a combination of different timing delays; the time taken by the computer to issue the command on the USB link the time by the PIC to process the commands and convert it into control signals for the multiplexer, the time taken at the multiplexer to switch between the microphone and the time taken by the computer to receive the audio input signal. After the propagation delay period is completed the program receives the data and allocates it to the intended microphone.

In the next step the data go through a filtering process as shown in the flow chart of figure 10. The time domain filter is intended to remove the impulse noise that propagate due to the switching effect. Impulse usually has very high amplitude and short duration; this is the property been used to detect it. The role of the frequency domain filter is to suppress all unwanted frequencies which are defined by the user as table 2 shows; since the project is dealing with merely human speech the minimum frequency and the maximum frequency are set to be the boundaries of the human speech range of frequencies as in table 2. Finally the system compare waveforms based on their amplitude level and select the one with highest amplitude as the desired one; the comparison is done in there frequency domain because it is easier to

distinguish and remove the noise in frequency domain rather than in time domain in addition to that the type of noise present are not regular as it will be shown it the next chapter.

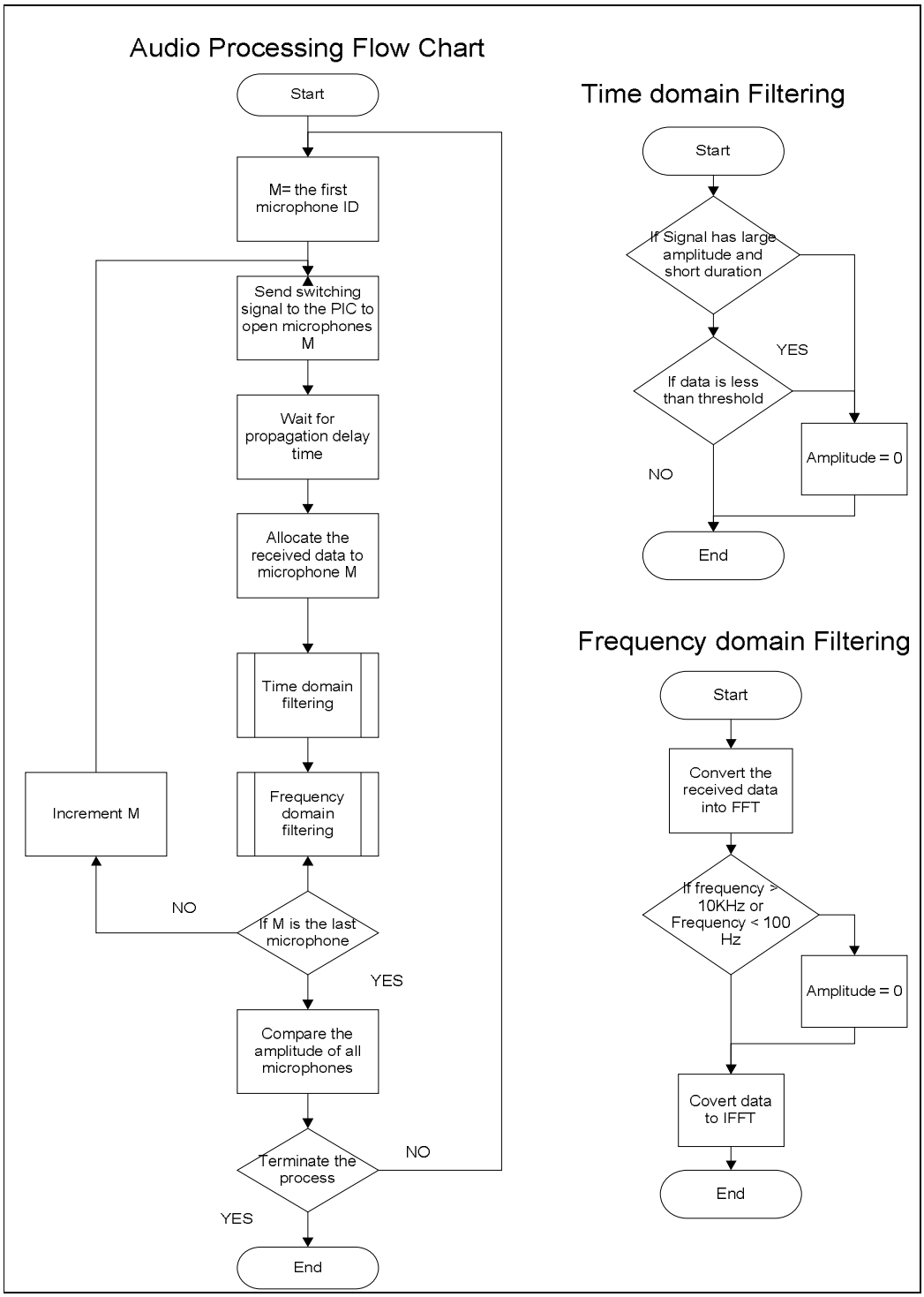


**Figure 10: flow diagram for audio processing at the computer.**

When comparing the frequency spectrum of signal with each other not the entire spectrum is used but rather a narrow range that represents the like human speech frequency is compared in order to eliminate noise presence, Figure 11 shows the selected frequency of interest. The minimum frequency is selected at 300 Hz although there is some human speeches bellow this level but the frequency lower than 300 Hz is very much affected by noise, therefore it is easier to consider it as noise and avoid the need for extra filtering process. For the higher limit which is at 6.0 KHz human speech, beside frequency filtering selecting specific range of frequency a threshold is imposed on the audio signal that should be fulfilled, the threshold will help removing the white noise that present on all frequencies. The threshold level can be selected by the user based on the signal to noise ratio the lower the signal to noise ratio is the higher the threshold value is.

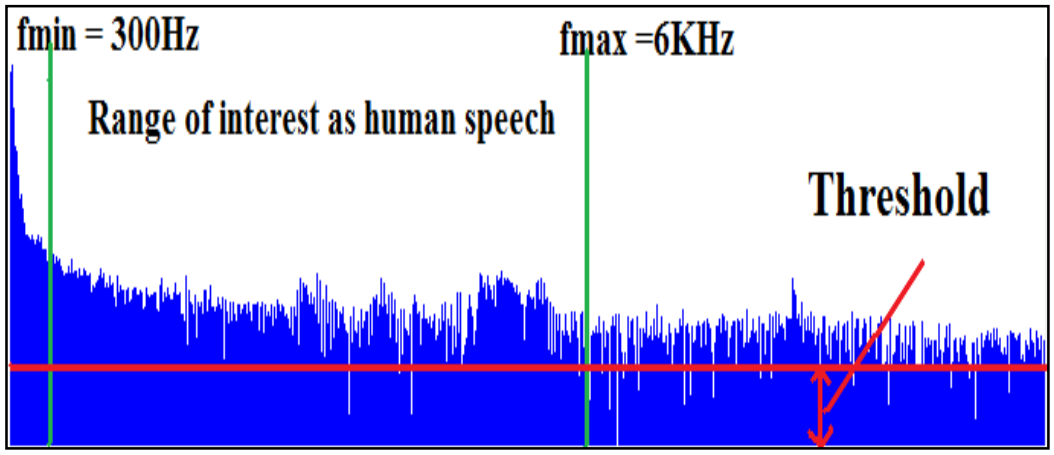


**Figure 11: Selecting the desired frequency range.**

### *3.1.3 Audio multiplexing synchronization*

Synchronization is necessary because the computer needs to know which microphone is been captured at any portion of time so it can allocate the audio input correctly to its correct microphone. Synchronization is achieved through USB communication between the PIC and the computer; which means the computer will issue the multiplexing signal to the PIC first and then it receives the signal after considering the propagation required for both the audio signal and the USB signals.

#### *3.1.3.1 At the computer side*

In the main program the USB port is considered as a normal serial port, this is

because the PIC 18F4550 associated INF defines the USB port as a serial port. INF file is a plain text file used by Microsoft Windows to install drivers and devices. The information in the INF file includes:

- Driver name and location
- Driver version information
- Registry information

when the user plug-in the PIC 18F4550 first time, the New Hardware Wizard of Windows system will pop up and request the user to specify a driver, the user will specify the INF file location, as mentioned before the INF file is not the driver but it will specify to the system which driver to be used. In general installing a USB device takes the following steps:

1. After the device is been plug-in, it will introduce itself to the computer by sending its name, its PIN ID and its vender ID.
2. Then New Hardware Wizard will pop up and ask the user to specify the driver.
3. The INF file in the driver should have the same Pin ID and vender ID as the one given by the devices.
4. If the above parameter matches the system will installed the driver.
5. If these parameters are not matching the installation process will result in failure.

The associated INF file can be found in Appendix B, the Pin ID and the Vender ID are written the this format *USB\VID_xxxx&PID_yyyy*. For this program the value are set to *USB\VID_04D8&PID_000A*. These values can be modified, however same Pin ID and vender ID must be used at the INF file as well as the USB descriptor code in PIC coding; **usbdsc.c** is the file contains all the descriptors of the USB device, more information about the USB descriptor code is available in Appendix C.

#### *3.1.3.2 At the PIC side*

When it comes to coding USB programs are very complicated and requires a full knowledge about USB protocols, fortunately Microchip provides some USB firmware's for USB communications. For this project case Communication Driver

Class (CDC) was used, CDC was modified a new function was added to handle the multiplexing work. The CDC program start by initializing the USB engine and after the above step is successful it read it USB data and pass it to a ServiceRequest() function, the ServiceRequest() function read the data from the USB packet and pass it to the multiplexer function, this data is simple the required microphone ID [11][16][17].

The Service request function is very useful when there are multiple tasks that can be performed by the computer, each task will have its own request ID, so when the computer need the PIC to blink its LEDs it will send specific request accompanied with the value of the LEDs the same goes when it needs the PIC to perform the multiplexing task it will send the service request ID of the multiplexer function accompanied with the desired microphone ID.

## 3.2 Video source:

In this project the camera been used is the Dlink DCS 5220 Pan/ Tilt camera or PT camera. DCS 5220 is an IP camera that can be accessed both through LAN cable as well as wirelessly. In order to be able to use any IP device it has to be connected to a network.

### *3.2.1 Setting up a network*

To set up a network we need to use a router, DR 300 is a Dlink wireless router with the capacity of 4 devices; it can setup a wired or wireless network by using easily identified steps. DR300 is used in this project as shown in figure 12, it construct a LAN between the computer and the camera. The router assigns fixed IP address to the camera and the computer in order to eliminate the need for IP camera search system.

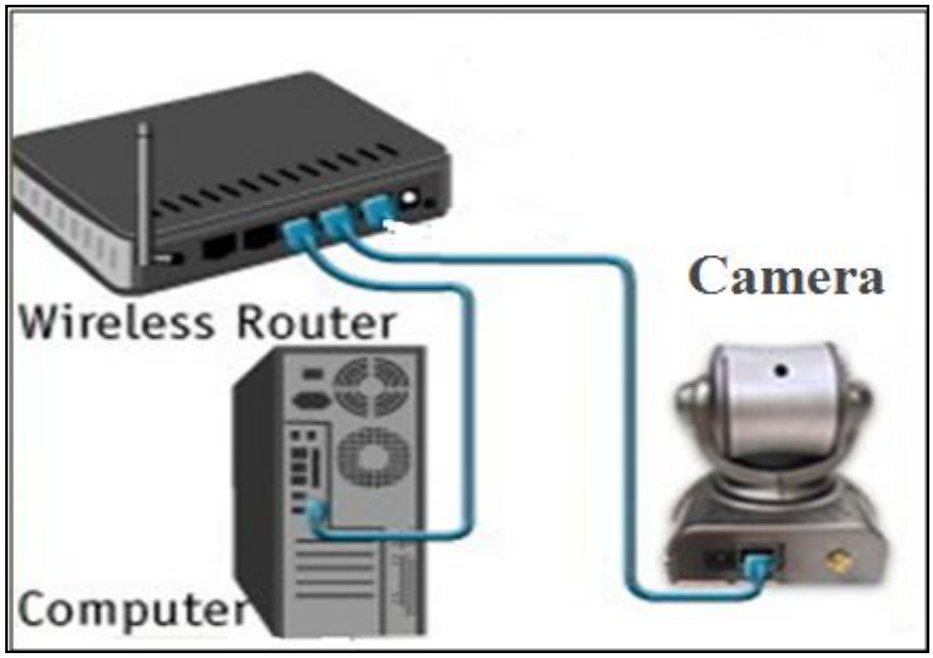


**Figure 12: Setting up a network for the camera system**

### *3.2.2 Communicating with the IP camera*

Because of the IP connection the computer can retrieve images from the camera, the LAN can be wired or wireless LAN. The steps of how to setup a wired or a wireless LAN are available in the manual provided with the router. The images retrieved from the camera are in JPEG format. Open source codes provided by codeproject website [7] had been used to access the camera and retrieve the JEPG images.

Steering the PT camera requires the use of URL special commands, these commands are used to perform panning tilting and digital zooming operations. Pan and tilt means the camera steers up/down or left/right. Usually these commands specified by the camera manufacturer and provided in the user manual, unfortunately Dlink do not provide these commands with user manual, but it can be found in the company website. Below is a generic format for the URL request used to control the Dlink PT camera?

```
http://<servername>/cgi-bin/camctrl.cgi? [move=<value>][&focus=<value>]
[&iris=<value>][&speedpan=<value>][&speedtilt=<value>][&speedzoom=<value>]
[&speedapp=<value>][&auto=<value>][&zoom=<value>][&return=<return page>]
```

In the above format, <servername> is the camera IP address, other parameter are explained in details in Appendix C; for example the HTTP request *http://<192.168.0.102>/cgi-bin/camctrl.cgi?[&move=right][&speedpan=5]* steers the camera to the right at maximum speed, pan and tilt speeds values and movements values are shown in table 3, like in the previous HTTP request the user can specify the commands he/she wishes to use the others can be left blank, other request be side move, speedpan and speedtilt are used for other purposes not required in this project, they are explained in the table of Appendix C. Table 3 contain the value for move, speedpan and speedtilt commands, move represents the direction of movement can be up/down or left/right. Speedpan and speedtilt are the speed of the motors during the movement; the camera uses two motors one for pan operations and the other one for tilt operations.

**Table 3: List of URL commands used to manueover the DCS 5220 camera.**

| Parameter | Value | Description |
|---|---|---|
| move | home | Move the camera to the home position |
| | left | Move the camera left to the home position |
| | right | Move the camera right to the home position |
| | up | Move the camera above to the home position |
| | down | Move the camera under the home position |
| speedpan | -5 ~ 5 | The camera panning speed |
| speedtilt | -5 ~ 5 | The camera tilting speed |

DCS 5220 can be accessed as a wireless camera and all the previous setting will be the same as the wired one and it has a data rate up to 54Mbps.

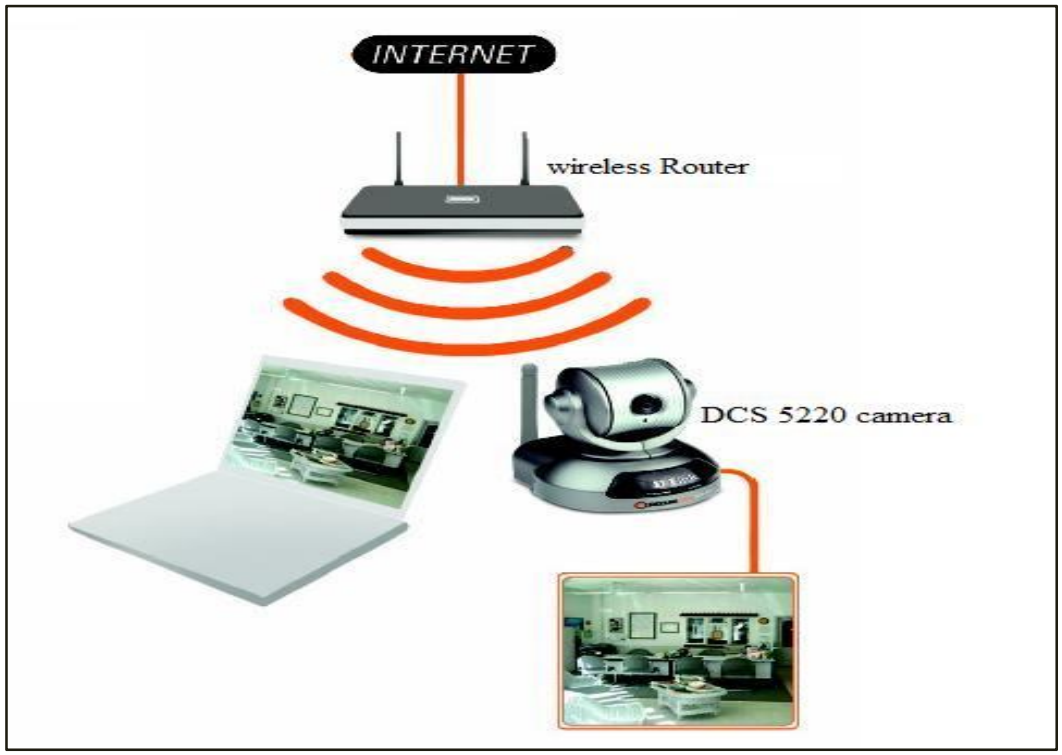


**Figure 13: wireless network for the DCS 5220.**

### *3.2.3 Steering the camera based on audio input*

The main program processes the audio data and decided which microphone the camera should steer to. The locations of the microphones are predefined to the camera and once the system decided when microphone to steer to the camera knows where the active microphone is currently is and then it steers to its location. Figure 14 shows the flow diagram for camera steering.

As shown in the figure below the camera location is not updated every time but it rather has a periodic updates every 3 seconds the reason for choosing this time is to sustain the camera image stable and moreover the camera motors take times to move and it has been experimentally found that 2 seconds are just enough for the camera to pan to farthest location from its current location, using a larger delay will make the image more stable but it will introduce a time lag which is not desirable. The selected microphone is the one that sustain to have the highest amplitude level during this period.

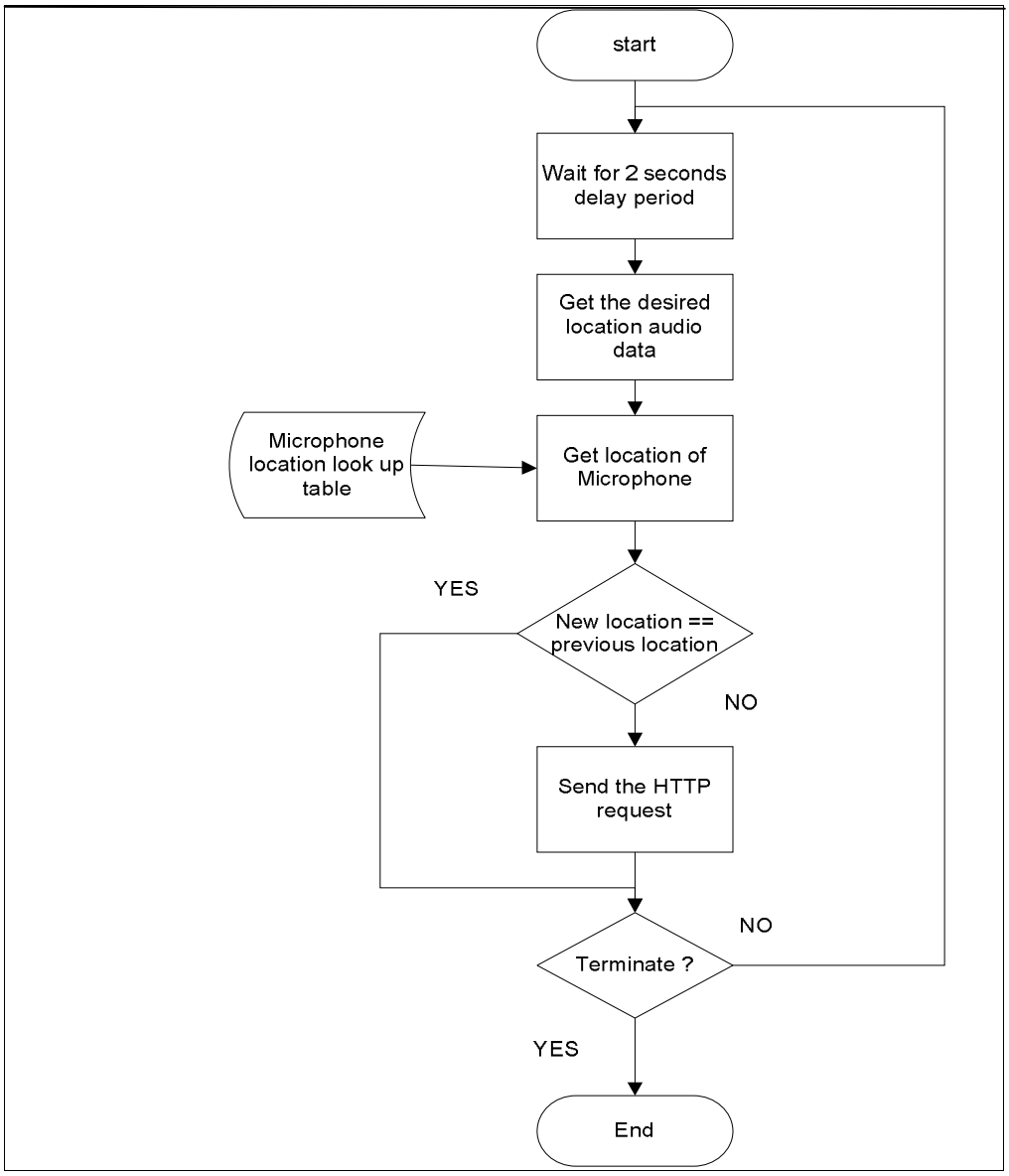


**Figure 14: camera steering system.**

### 3.3 Project milestones

By the time this report was written the project is 100% completed, the concept of localizing video images by using audio is been successfully proven. The completion of the project include developing the audio processing system, constructing the multiplexing circuits and the PIC coding, completing the USB communication, setting up a network for computer and the camera to communicate and lastly developing a user graphical interface for the system operator.

## 3.4 Tools and software:

### *3.4.1 Tools:*

- Philips SHM1000 sensitive PC microphones.
- Microchip's PIC 18F4550 microcontroller.
- Analog Devices CD4097 analog multiplexer.
- Dlink DCS 5220 PT wireless IP camera.
- Dlink DIR-300 wireless router.
- Personal computer.

### *3.4.2 Software:*

- Visual Studio 2008 for the software part programming which uses C# 3.0.
- Mplab V8.20 with microchip C18 compiler for the hardware side program.

# CHAPTER 4
# RESULTS AND DISCUSSIONS

## 4.1 Results:

As discussed in the key milestones, the audio processing part is 100% completed, this includes construction the multiplexing circuit, coding the PIC program, developing the computer program that perform the audio processing and filtering as well as PIC-computer synchronization through a USB communication link is completed successfully. On top of that this part is capable of identifying the correct sound source and steering the DCS 5220 camera to the correct location. Camera steering, image capturing and visualizing are also completed.

### *4.1.1 Multiplexing circuit:*

The multiplexing circuit is been completed first on a breadboard, and after that it has been soldered properly on a Vero board as shown in figure 15, the circuit consist of PIC 18F4550 microcontroller, analog multiplexer and LED indicators. The circuit is been powered up from the USB host, however it can also be powered up by independent power supply.

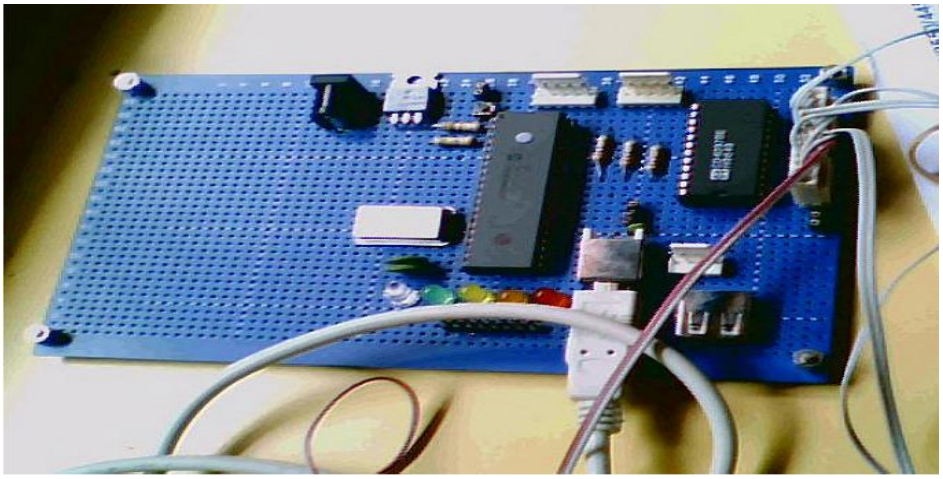

**Figure 15: the multiplexing circuit.**

### *4.1.2 Audio processing system:*

Audio processing system is the main component of this project; signals have been captured from four microphones and can be displayed in both time domain and frequency domain.

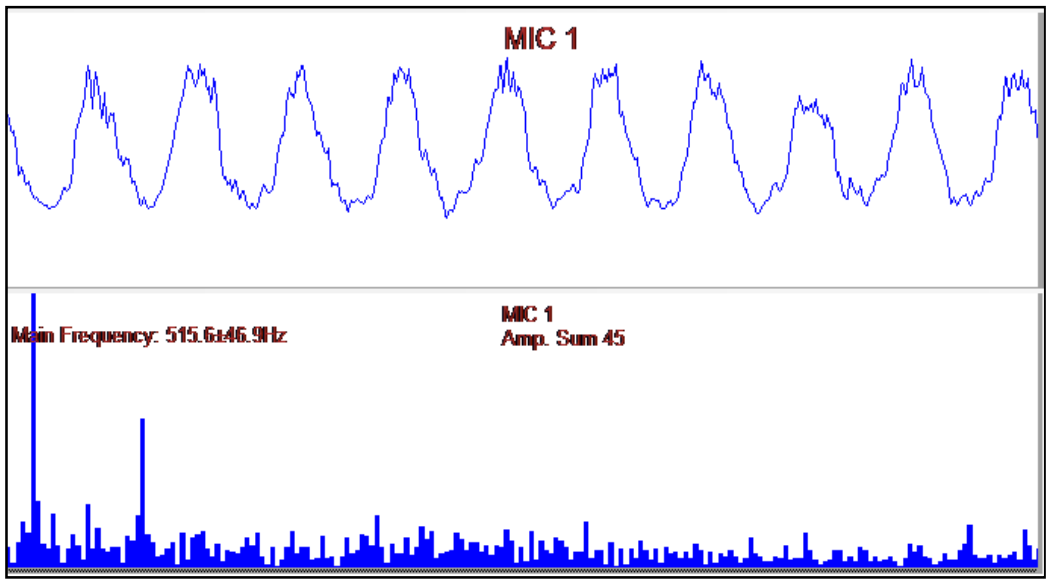


**Figure 16: microphone signal been presented in time and frequency domains.**

The audio processing user interface allows the user to adjust the parameters of the input audio signal, as shown in figure 16, the input data sampling rate can be adjust to one of the values of the combo box, the same can be applied for the other parameters such as the number of bits per sample and the upper and lower frequency values.

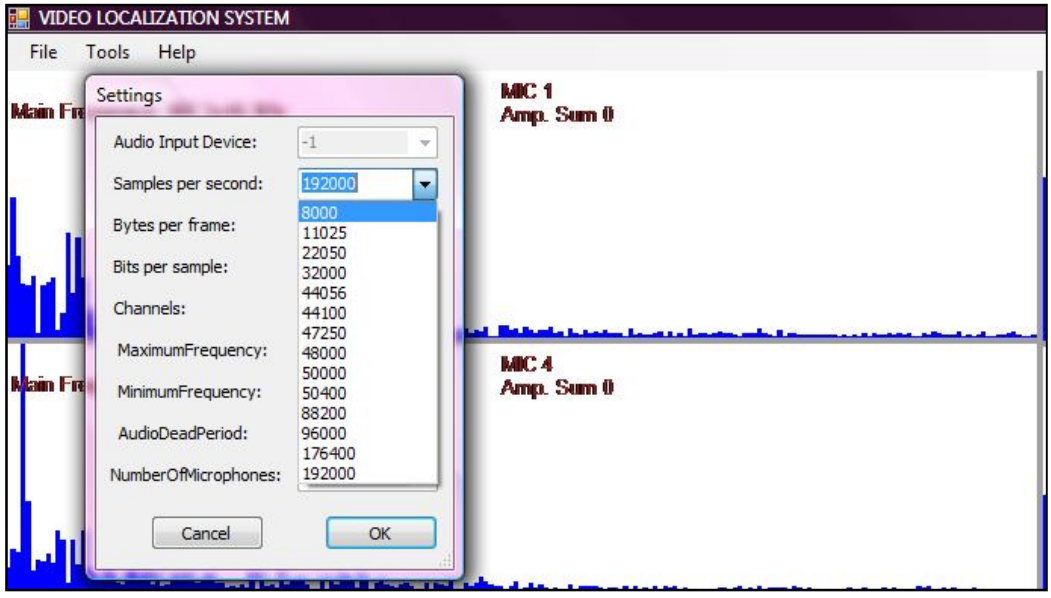


**Figure 17: adjusting the audio signal parameters during the run time.**

### *4.1.3 Localization results*

Figure 17 through figure 20 shows the final results of the localization system whenever audio data is detected at any microphone location and its amplitude is proven to be above the noise threshold level. The camera will move the microphone location and tag that microphone as an active microphone.

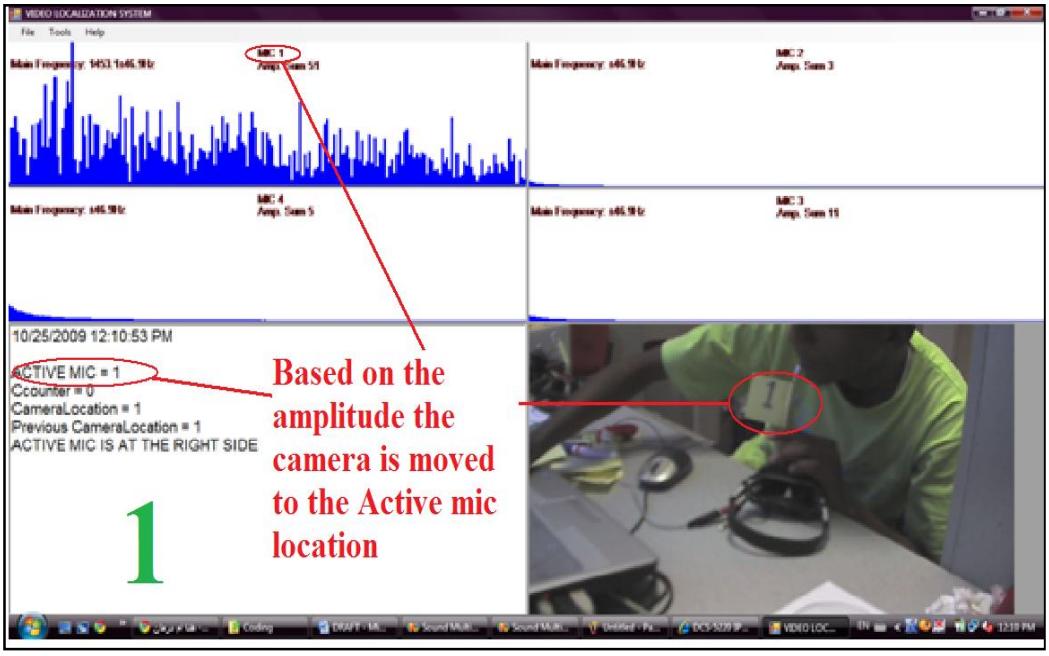


**Figure 18: Localization results shown for MIC 1.**

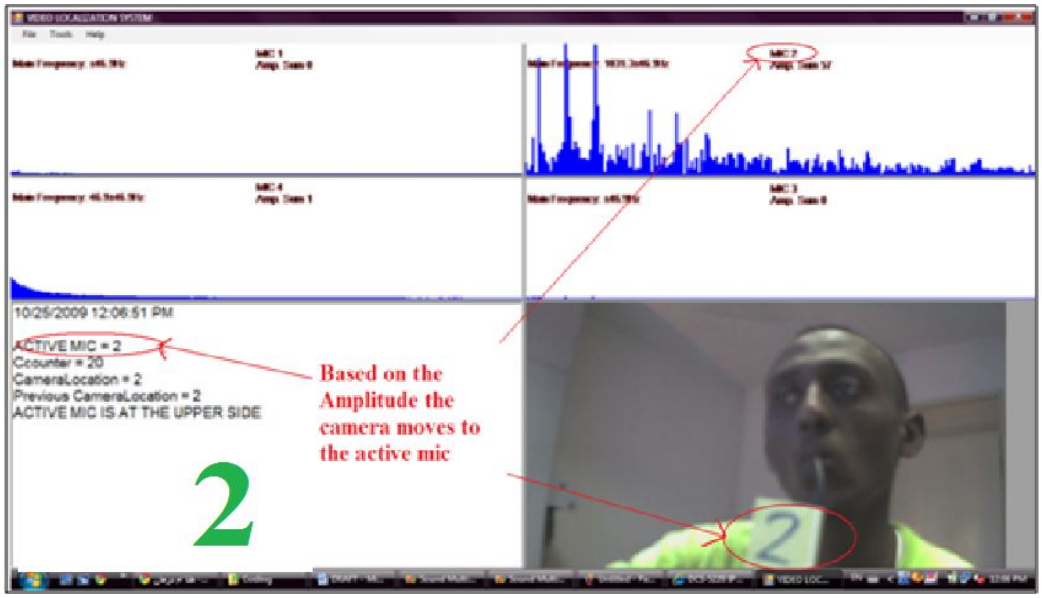


**Figure 19: Localization results shown for MIC 2.**

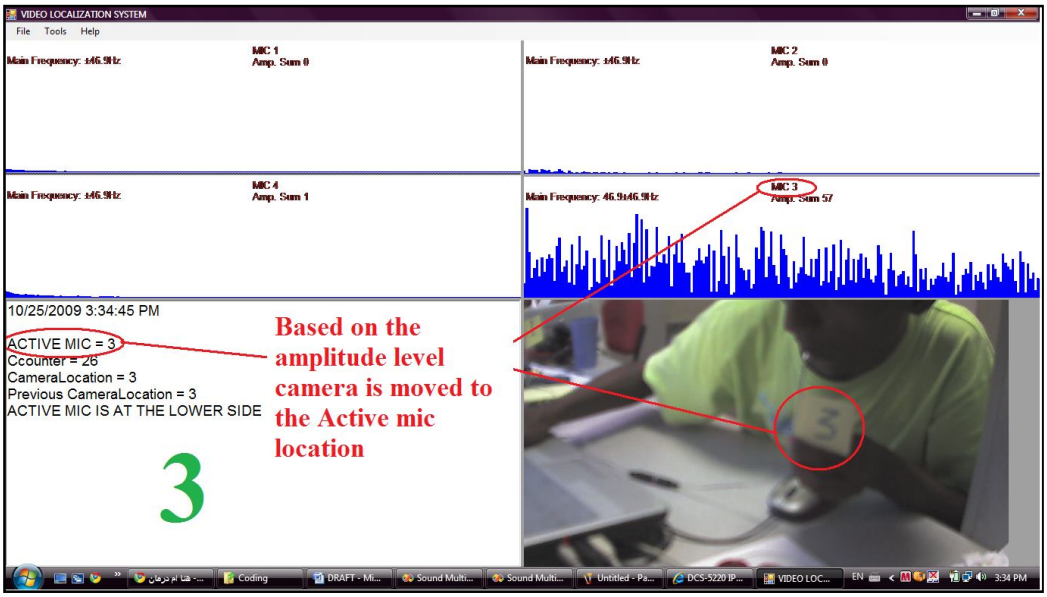


**Figure 20: Localization results shown for MIC 3.**

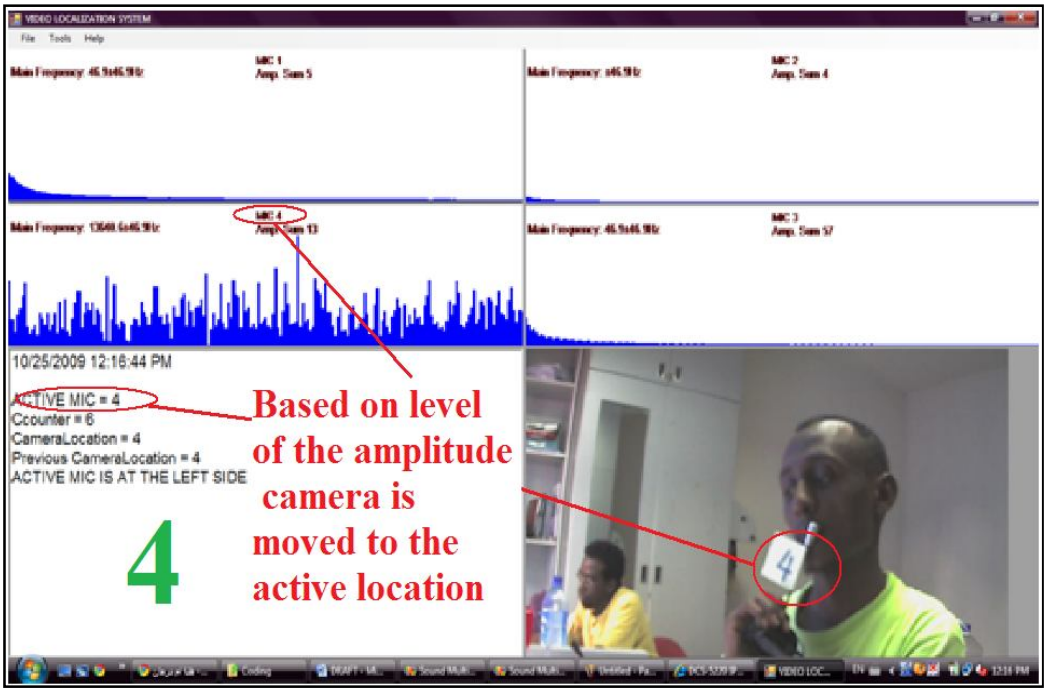


**Figure 21: Localization results shown for MIC 4.**

In the figures shown before the upper side shows the frequency spectrum of the four microphones and it is clear that there is only one active microphone at any time in all these case, the lower part of the window shows the image of the speaker which is the desired output of the system. Beside that it shows the selected microphone and the amplitude counter as well as the camera pointing location in a text format.

### *4.1.4 Camera movements*

The camera can be steered in different locations using URL commands as been explained earlier, figure 22 shows 9 different positioning for the DCS5220 PT camera.

**Figure 22: maneuvering the PT camera in different locations.**

Figure 23 shows the exact system components been used, DCS 5220 camera, DR 300 wireless router, microphones and the multiplexing circuit.

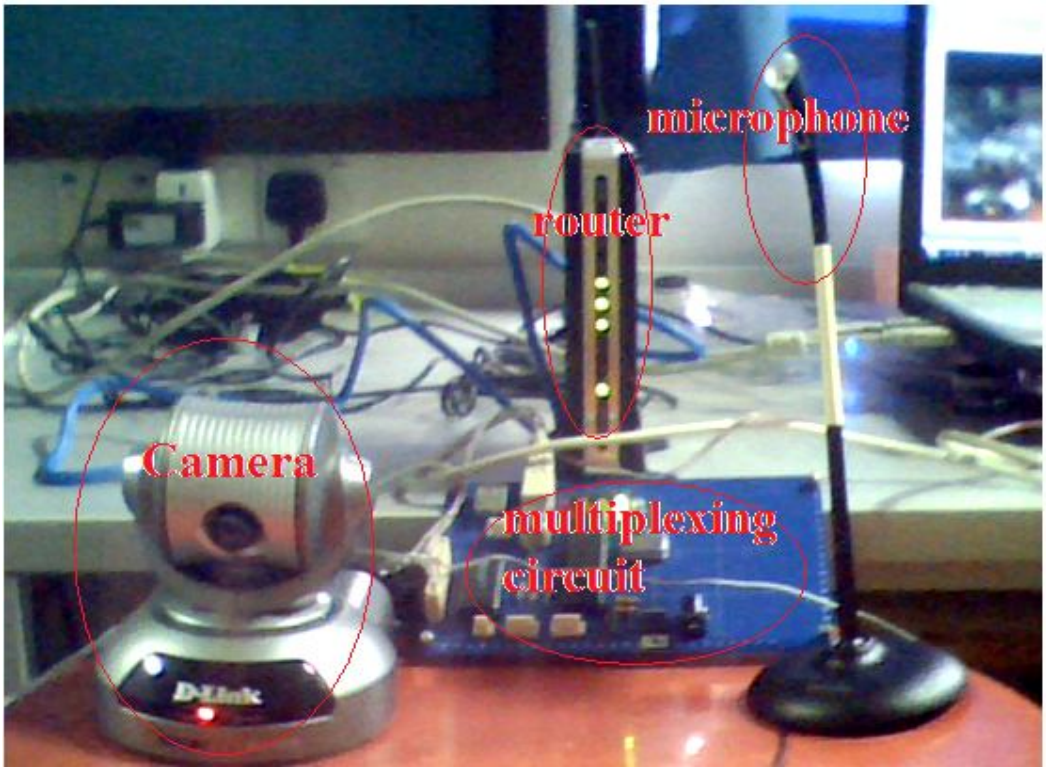


**Figure 23: System components that been used in this project.**

## 4.2 Discussions

This section will analyze the performance of each part of the project components and what are the problems if any that reduces the system performance.

### *4.2.1 The multiplexing circuit*

The multiplexing circuit uses analog multiplexer which is capable of multiplexing of four microphones as maximum inputs, beside that since it is form of switching technique it add noise to the signal. Figure 23 and figure 24 shows the time domain waveform and the frequency spectrum respectively for the microphone inputs when there is no audio source present at the microphones, in time domain the switching appears as impulses with long discharge time, while in frequency domain it appears as low frequency components; therefore it is easier to suppress this type of noise in the frequency domain by designing a low pass filter, while in time domain it is difficult to gauge at what time the impulse appears and moreover its discharge time is longer than normal impulses; which makes it difficult to be removed based on impulse signal characteristics.

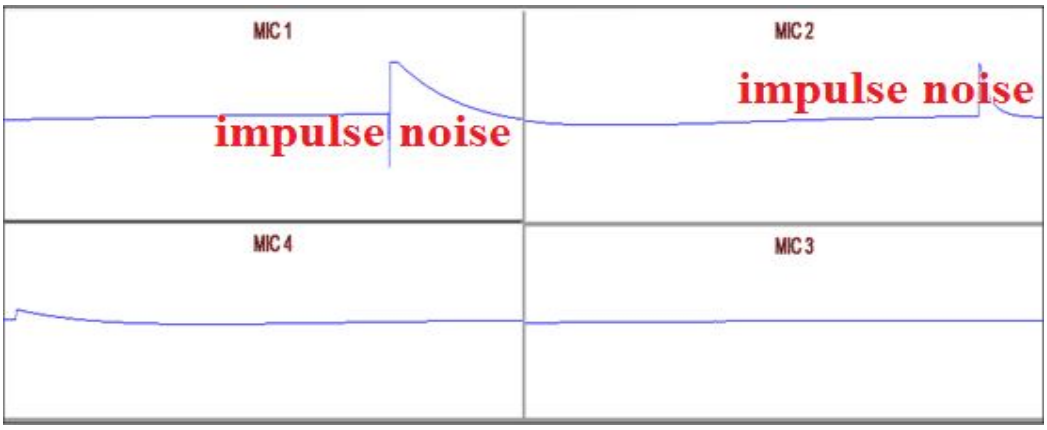


**Figure 24: Effect of switching on the audio data when there is no audio source present at the microphones in the time domain.**

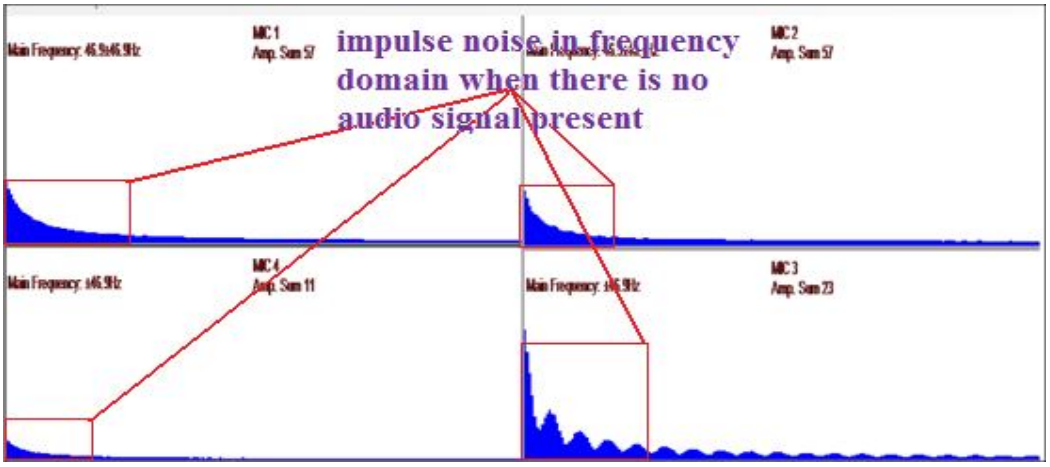


**Figure 25: Effect of switching on the audio data when there is no audio source present at the microphones in frequency spectrum.**

### *4.2.2 Audio processing*

The chosen multiplexing frequency is set to be 0.5 Hz this is the optimum multiplexing rate to avoid signal overlap between the microphones, although this rate conflict with Nyquist criterion for sampling which says that the lowest sampling rate must be larger than twice the maximum frequency which mean is shown be > 20KHz (assume max frequency is 10KHz). However this rate is still acceptable because the audio average amplitude level is the point of interest not the exact meaning of audio data. Increasing the multiplexing rate bend 0.5 Hz will result in increase on the impulse and distortion of the signal. Figure 26 illustrates the effect of increasing the multiplexing rate bend 0.5 Hz, on the time domain the effect is very severe and the

signal is totally distorted. On frequency spectrum the effect is not very significant and it appears as low frequency component but the signal overlaps between the microphones.

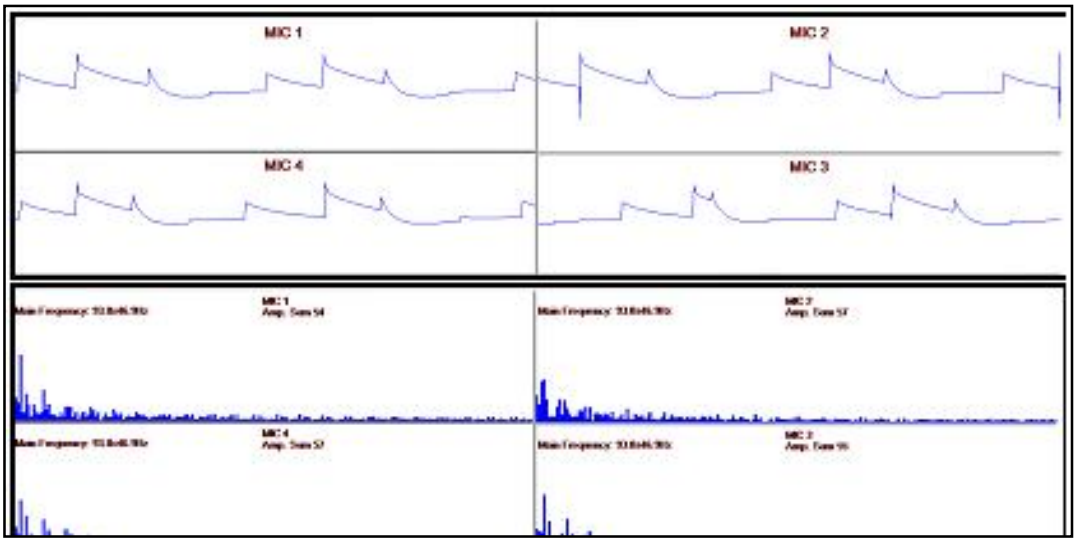


**Figure 26: the effect of increasing the multiplexing rate bend 0.5 Hz on both time domain and frequency domain signals.**

Because of the fact that noise presence in frequency spectrum is less significant than in time domain, the compression between audio signals is been conducted on its frequency spectrum amplitude level after removing the low frequency components from it, however similar results could be obtained by using time domain comparison as shown in figure 27. But it is more challenging to perform the comparison in time domain rather than frequency domain because of noise.

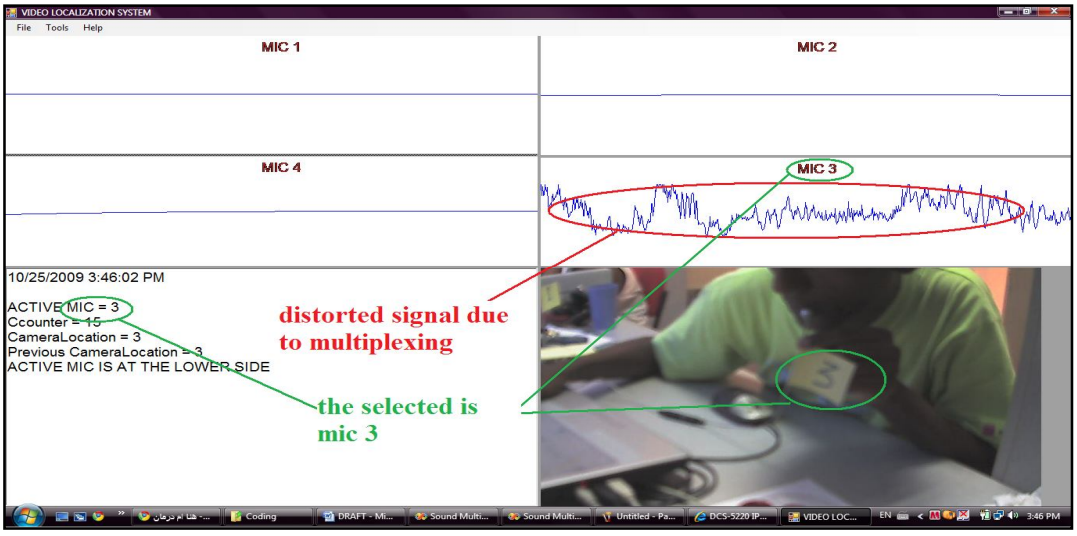


**Figure 27: localization using time domain comparison.**

### *4.2.3 Camera movements*

Camera steering is achieved by sending URL commands, these commands contains the direction and the speed of the motion, the speed of the motion determine whether the camera reach the destination faster or slower, that make the destination for one URL command fixed despite the motion speed. However when steering the camera sometimes user requires the camera to steer to a far distance or sometime to a near one, thus it is necessary to use multiple URL call if one URL call is not enough to reach to the desired destination.

When using multiple URL call it is very important to send the second call after the first one is been completed and this will introduce a considerable delay to the camera movements. Lastly it has to be considered that when a combination of pan and tilt commands are in the same URL call the camera will process only the last command in that call whether it is pan call or tilt call, thus it is necessary to put only one command in one URL call.

### *4.2.4 Overall system analysis*

The previous discussions illustrated some issues with different components of the localization system. Never the least the reliability of the system is very much dependent on the audio processing system as well as the camera steering system. As shown in figure 28, audio signal should be present as the beginning in order for the system to operate; presence of noises and interference will have significant impact on the system performance. In the next step microphones captures the audio data, how sensitive are these microphones is an important feature for a better system performance; therefore it is desirable to have high sensitivity unidirectional microphones equipped with amplifiers. After that microphones are multiplexed using analog multiplexer; in this step significant noise is been added by the multiplexer and also there is a significant voltage drop from the audio data, a low pass filter may be added at this stage and it will have its effect on the resultant audio data. The synchronization components are present in this stage; most of the propagation delay time is generated at this stage it is very important to practice good programming technique in order to reduce the delay taken at the PIC program.

For the next stage the signal is received by the computer, at this stage signal is

demultiplexed, filtered, and analyzed. The demultiplexing and construction of the original signals is affected by the propagation delay and the presence of synchronizing system. To design good filters signals should be modeled properly and all types of noise should by identified in order to suppress it, however not all noises are mathematically modeled specially the noises added by the analog multiplexer. At the last stage the camera translates the computer commands into movements (pan and tilt), camera response time and how easy to steer the camera is an important features for this stage.

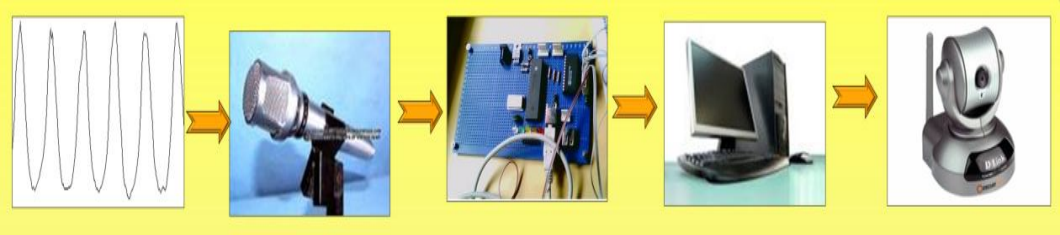

**Figure 28: General flow for the localization system.**

Factors affecting the system performance of the system:

1. Presence of noise and interferences.
2. Sensitivity of the microphone and the presence of amplifiers at the source.
3. Type and performance of the multiplexer used in the audio multiplexing.
4. Using a synchronization system between multiplexer and computer.
5. Propagation delay time.
6. Analyzing algorithm used in audio data analysis.
7. Type of camera used, resolution of image and response time of the camera.

# CHAPTER 5
# CONCLUSIONS AND RECOMMENDATIONS

## 5.1 Conclusion:

The study successful proves that camera can be localized using audio sensors in an indoor environment. Microphones are good sensors for videoconferencing application because it can sense the present of audio data, using multiple microphones require the use of multiplexing circuit which introduces noises to the data. For outdoor applications microphones are not a reliable sensors because of the presence of very strong noise that leads to a week signal to noise ratio (SNR < 1). The audio processing system successfully suppress some of the noise and interference present due to the multiplexing, a band pass filter is used to removes frequency components outside the band of 200Hz to 10 KHz.

USB communication is the best synchronization technique for computer PIC communication. USB synchronization enables the computer to control the multiplexing at the PIC circuit, however, the propagation delay taken from the time the computer issues the multiplexing signal until it receives the audio data is very considerable and it limits the multiplexing rate to about 0.5 Hz, increasing the rate will result in signal overlapping.

DCS 5220 camera is suitable for small area surveillance such as small conference rooms; it has a resolution of 760 X 480 at maximum and the frame rate is around 15 frames per seconds. The camera can pan and tilt in all directions. Despite that, the pan and tilt operations are not linear and varying the speed will make the movement faster or slower but it will not affect how far the camera can pan or tilt. DCS 5220 is an IP camera thus, DR 300 wireless router is used to setup a network in order to access the camera form the computer, DR 300 is not consistent and frequently logout which result in image hanging.

The system assumes only one microphone is active at any time, and if more than one

microphone is active at the same time the system will select the one that has the highest amplitude, that why it is necessary to place the microphones in distances from each other so each one capture different audio pattern form the others.

## 5.2 Recommendations:

For the system to improve its performance it is recommended that is more sensitive microphone is been used, in addition to that the multiplexing could be performed using DSP board which may have better noise performance and the system can be expanded to more than four audio inputs, analog multiplexer have hard switching technique which reduces the signal level and introduce noise because it is all analog devices, while when using a DSP board the audio signal is digitized first and after that it is been multiplexed, with digital signal no worries of signal level reduction.

DCS 5220 pan tilt control is nonlinear and it has low resolution; for a better performance it is strongly recommended to use high resolution camera such as Axis communication PTZ cameras which has higher resolution and linear movements, in addition DCS 5220 does not support optical zooming which is very necessary in large rooms while Axis communication cameras has optical and digital zooming and moreover it has better customer support.

The system could be improved to suit other application such as determine the speaker location when the microphone is not fixed; for this project all microphones are prefixed and the computer know their locations in advance, in TV show speaker carry wireless microphone and maneuver on the stage; thus by equipping the microphone with a localizing device such as accelerometer the system can determined the microphone location and follow the speaker movements.

So far the system is used for closed room application because audio is controlled over there, never the least this system can be applied in outdoor application where the audio sensors can be replaced with IR or ultrasound sensors; IR / ultrasound can be installed in car parks in different segments of a large parking system, so whenever a car crosses the IR/ ultrasound beam this segment will be activated and the camera will point to this segment and highlight the suspicious action in that area.

# REFERENCES


[1]. Robert N. Charette, October 2007, Smart Parking Systems Make It Easier to Find a Parking Space , IEEE spectrum.

[2]. M. Y. Idress, L. L. Leong and others, January 2009, car park system: a review of smart system and its technology, 2nd Edition, University of Malaya.

[3]. B. R. Abidi, N. R. Aragam, and others, December 2008, Survey and Analysis of Multimodal Sensor Planning and Integration for Wide Area Surveillance, university of Tennessee, ACM computing survey.

[4]. Huadong Wu, Mel Segel and Bradeep Khosola, May 1998, Vehicle Sound Signature Recognition by Frequency Vector Principal Component Analysis, Carnegie Mellon University, Pittsburgh PA USA.

[5]. CD 4097 dual channel analog multiplexer/ de-multiplexer datasheets.

[6]. Jeff Morton**,** Sound *Activated Recorder with Spectrogram in C#*, code project website http://www.codeproject.com/KB/audio-video/SoundCatcher.aspx, last visit 24/10/09.

[7]. Andrew Kirillov, Camera Vision - video surveillance on C#, code project website http://www.codeproject.com/KB/audio-video/cameraviewer.aspx, last visit 24/10/2009.

[8]. Microchip PIC 18FX455/X 550 family reference manual, http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010300, last visit 24/10/2009.

[9]. D-Link DCS 5220 User Manual, http://www.dlink.com.my/. Last visit 15/9/2009.

[10]. Code project website http://codeproject.com/ last visit 20/8/2009.

[11]. Microchip Inc website PICDEM Full Speed USB Board http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en021940/ last visit 3/7/2009.

[12]. USB overview, http://www.ganssle.com/articles/usb.htm, last visit 19/10/2009.

[13]. Yitneg Huang, *microphone array for video camera steering*, center for signal and image processing, Georgia Institute for technology.

[14]. YASIR SALIH OSMAN ALI, industrial internship final report, June 2008 – January 2009 at MIMOS BERHAD, Electrical and electronic engineering department Universiti Teknologi PETRONAS, Malaysia.

[15]. URL commands for DCS-5220, DCS-2120, http://www.dlink.com.my/, Version 1.0 B, 6/6/2007.

[16]. USB serial Board on 18F4550, Microchipc website http://www.microchipc.com/sourcecode/index.php#pic18f4550usb, last visit 24/10/2009.

[17]. Basic USB - Using Microchip Stack and C#.Net, PIC coder website http://www.piccoder.co.uk/content/view/42/26/, last visit 24/10/2009.

[18]. Mr. Patrick Sebastian, *Microcontroller course lecture notes*, Universiti Teknologi PETRONAS (UTP), July 2009.

# APPENDICES

Appendix A: Microchip INF files

Appendix B: PIC 18F4550 descriptor

Appendix C: DCS5220 PT camera control commands

Appendix D: PIC code

Appendix E: PC code

# APPENDIX A

# MICROCHIP INF FILE

```
; Windows USB CDC ACM Setup File
; Copyright (c) 2000 Microsoft Corporation
; Copyright (C) 2007 Microchip Technology Inc.

[Version]
Signature="$Windows NT$"
Class=Ports
ClassGuid={4D36E978-E325-11CE-BFC1-
08002BE10318}
Provider=%MFGNAME%
LayoutFile=layout.inf
CatalogFile=%MFGFILENAME%.cat
DriverVer=11/15/2007,5.1.2600.0

[Manufacturer]
%MFGNAME%=DeviceList, NTamd64

[DestinationDirs]
DefaultDestDir=12


;-----------------------------------------------------------
----------------
;  Windows 2000/XP/Vista-32bit Sections
;-----------------------------------------------------------
----------------

[DriverInstall.nt]
include=mdmcpq.inf
CopyFiles=DriverCopyFiles.nt
AddReg=DriverInstall.nt.AddReg

[DriverCopyFiles.nt]
usbser.sys,,,0x20

[DriverInstall.nt.AddReg]
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,%DRIVERFILENAME%.sys
HKR,,EnumPropPages32,,"MsPorts.dll,SerialPortPropPag
eProvider"

[DriverInstall.nt.Services]
AddService=usbser, 0x00000002, DriverService.nt

[DriverService.nt]
DisplayName=%SERVICE%
ServiceType=1
StartType=3
ErrorControl=1
ServiceBinary=%12%\%DRIVERFILENAME%.sys

;-----------------------------------------------------------
----------------
;  Vista-64bit Sections
;-----------------------------------------------------------
----------------

[DriverInstall.NTamd64]
include=mdmcpq.inf
CopyFiles=DriverCopyFiles.NTamd64
AddReg=DriverInstall.NTamd64.AddReg

[DriverCopyFiles.NTamd64]
%DRIVERFILENAME%.sys,,,0x20

[DriverInstall.NTamd64.AddReg]
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,%DRIVERFILENAME%.sys
HKR,,EnumPropPages32,,"MsPorts.dll,SerialPortPropPag
eProvider"

[DriverInstall.NTamd64.Services]
AddService=usbser,                    0x00000002,
DriverService.NTamd64

[DriverService.NTamd64]
DisplayName=%SERVICE%
ServiceType=1
StartType=3
ErrorControl=1
ServiceBinary=%12%\%DRIVERFILENAME%.sys


;-----------------------------------------------------------
----------------
;  Vendor and Product ID Definitions
;-----------------------------------------------------------
----------------
; When developing your USB device, the VID and PID
used in the PC side
; application program and the firmware on the
microcontroller must match.
; Modify the below line to use your VID and PID.  Use
the format as shown below.
; Note: One INF file can be used for multiple devices
with different VID and PIDs.
;     For each supported device, append
",USB\VID_xxxx&PID_yyyy" to the end of the line.
;-----------------------------------------------------------
----------------
[SourceDisksFiles]
[SourceDisksNames]
[DeviceList]
%DESCRIPTION%=DriverInstall,
USB\VID_04D8&PID_000A

[DeviceList.NTamd64]
%DESCRIPTION%=DriverInstall,
USB\VID_04D8&PID_000A


;-----------------------------------------------------------
----------------
;  String Definitions
;-----------------------------------------------------------
----------------
;Modify these strings to customize your device
;-----------------------------------------------------------
----------------
[Strings]
MFGFILENAME="mchpcdc"
DRIVERFILENAME ="usbser"
MFGNAME="Microchip Technology, Inc."
INSTDISK="Microchip Technology, Inc. Installation
Disc"
DESCRIPTION="Communications Port"
SERVICE="USB RS-232 Emulation Driver"
```

# APPENDIX B

# PIC 18F4550 DEVICE DESCRIPTOR

```
 * ------------------------------------------------------------------
 * The look-up table scheme also applies to the configuration
 * descriptor. A USB device may have multiple configuration
 * descriptors, i.e. CFG01, CFG02, etc. To add a configuration
 * descriptor, user must implement a structure similar to CFG01.
 * The next step is to add the configuration descriptor name, i.e.
 * cfg01, cfg02,.., to the look-up table USB_CD_Ptr. USB_CD_Ptr[0]
 * is a dummy place holder since configuration 0 is the un-configured
 * state according to the definition in the USB specification.
 *******************************************************************/
/*******************************************************************
 * Descriptor specific type definitions are defined in:
 * system\usb\usbdefs\usbdefs_std_dsc.h
 * Configuration information is defined in:
 * autofiles\usbcfg.h
 *******************************************************************/
/** I N C L U D E S **************************************************/
#include "system\typedefs.h"
#include "system\usb\usb.h"

/** C O N S T A N T S ************************************************/
#pragma romdata

/* Device Descriptor */
rom USB_DEV_DSC device_dsc=
{
   sizeof(USB_DEV_DSC),    // Size of this descriptor in bytes
   DSC_DEV,                // DEVICE descriptor type
   0x0200,                 // USB Spec Release Number in BCD format
   0x00,                   // Class Code
   0x00,                   // Subclass code
   0x00,                   // Protocol code
   EP0_BUFF_SIZE,          // Max packet size for EP0, see usbcfg.h
   0x04D8,                 // Vendor ID
   0x0000,                 // Product ID: PICDEM FS USB (DEMO Mode)
   0x0001,                 // Device release number in BCD format
   0x01,                   // Manufacturer string index
   0x02,                   // Product string index
   0x00,                   // Device serial number string index
   0x01                    // Number of possible configurations
};

/* Configuration 1 Descriptor */
CFG01=
{
   /* Configuration Descriptor */
   sizeof(USB_CFG_DSC),    // Size of this descriptor in bytes
   DSC_CFG,                // CONFIGURATION descriptor type
   sizeof(cfg01),          // Total length of data for this cfg
   1,                      // Number of interfaces in this cfg
   1,                      // Index value of this configuration
   0,                      // Configuration string index
   _DEFAULT|_SELF,                 // Attributes, see usbdefs_std_dsc.h
   50,                     // Max power consumption (2X mA)

   /* Interface Descriptor */
   sizeof(USB_INTF_DSC),   // Size of this descriptor in bytes
   DSC_INTF,               // INTERFACE descriptor type
   0,                      // Interface Number
   0,                      // Alternate Setting Number
   2,                      // Number of endpoints in this intf
   0x00,                   // Class code
   0x00,                   // Subclass code
   0x00,                   // Protocol code
   0,                      // Interface string index
```

```
    /* Endpoint Descriptors */
    sizeof(USB_EP_DSC),DSC_EP,_EP01_OUT,_INT,USBGEN_EP_SIZE,32,
    sizeof(USB_EP_DSC),DSC_EP,_EP01_IN,_INT,USBGEN_EP_SIZE,32
};

rom struct{byte bLength;byte bDscType;word string[1];}sd000={
sizeof(sd000),DSC_STR,0x0409};

rom struct{byte bLength;byte bDscType;word string[25];}sd001={
sizeof(sd001),DSC_STR,
'M','i','c','r','o','c','h','i','p',' ',
'T','e','c','h','n','o','l','o','g','y',' ','I','n','c','.'};

rom struct{byte bLength;byte bDscType;word string[33];}sd002={
sizeof(sd002),DSC_STR,
/*'M','i','c','r','o','c','h','i','p',' ','U','S','B',' ',
'D','e','m','o',' ','B','o','a','r','d',' ','(','C',')',
' ','2','0','0','7'};*/
{'F','Y','P'}
};

rom const unsigned char *rom USB_CD_Ptr[]=
{
    (rom const unsigned char *rom)&cfg01,
    (rom const unsigned char *rom)&cfg01
};
rom const unsigned char *rom USB_SD_Ptr[]=
{
    (rom const unsigned char *rom)&sd000,
    (rom const unsigned char *rom)&sd001,
    (rom const unsigned char *rom)&sd002
};

#pragma code

/** EOF usbdsc.c ************************************************/
```

# APPENDIX C

# DCS 5220 CAMERA CONTROL COMMANDS

**NOTE:** this request require camera control access privilege.

Syntax:

http://<servername>/cgi-bin/camctrl.cgi? [&move=<value>][&focus=<value>]
[&iris=<value>][&speedpan=<value>][&speedtilt=<value>][&speedzoom=<value>]
[&speedapp=<value>][&auto=<value>][&zoom=<value>][&return=<*return page*>]

| parameter | value | Description |
|---|---|---|
| move | home | home Move to camera to home position |
| | up | Move camera up |
| | down | Move camera down |
| | left | Move camera left |
| | right | Move camera right |
| speedpan | -5 ~ 5 | Set the pan speed |
| speedtilt | -5 ~ 5 | Set the tilt speed |
| speedzoom | -5 ~ 5 | Set the zoom speed |
| speedapp | 5 ~ 5 | Set the auto pan/patrol speed |
| auto | pan | Auto pan |
| | patrol | Auto patrol |
| | stop | Auto stop |
| zoom | wide | To zoom for larger view with current speed |
| | tele | To zoom for farer view with current speed |
| focus | auto | To do auto focus |
| | far | To focus on farer distance |
| | stop | To focus on a nearer distance |
| iris | auto | Let the Network Camera control iris size |
| | open | Manually control the iris for bigger size |
| | stop | Manually control the iris for smaller size |
| return | <return page> | Redirect to the page <return page> after the parameter is assigned. The <return page> |

| | | |
|---|---|---|
| | | can be a full URL path or relative path according to the current path. If you omit this parameter, it will redirect to an empty page. |
| recall | Text string less than 30 characters | One of the present positions to recall. |

# APPENDIX D

# PIC CODE

The PIC microcontroller coding are from http://www.microchipc.com/ it is form the microchip CDC program which can be found at microchip website. The CDC program contain many class and it is the standard PIC-USB communication codes, for this application the other only modified one c files which is the user.c other files are available in Microchip communication class driver (CDC). Due to space limit only the user.c codes are list in this appendix and for the other you can refer to the CDC class coding.

```
//
/********************************************************************
 *
 *   Microchip USB C18 Firmware Version 1.0
 *
********************************************************************
 * FileName:        user.c
 * Dependencies:
 * Processor:       PIC18
 * Compiler:        C18 2.30.01+
 * Company:         Microchip Technology, Inc.
 * Software License Agreement
 * Author:  Rawin Rojvanit
 * Date:        11/19/04
 * Comment : Original.
 * modified by: Yasir Salih Osman
 * Date: 15/7/2009
 ********************************************************************/

/** I N C L U D E S **************************************************/
#include <p18cxxx.h>
#include <usart.h>
#include "system\typedefs.h"

#include "system\usb\usb.h"

#include "io_cfg.h"             // I/O pin mapping
#include "user\user.h"

/** V A R I A B L E S ************************************************/
#pragma udata

byte counter;
byte trf_state;

DATA_PACKET dataPacket;

/** P R I V A T E  P R O T O T Y P E S *******************************/

void BlinkUSBStatus(void);
void ServiceRequests(void);
int Multiplexer(int MicNo);

/** D E C L A R A T I O N S *****************************************/
#pragma code
void UserInit(void)
{
   mInitAllLEDs();

}//end UserInit


/********************************************************************
```

```
 * Function:        void ProcessIO(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This function is a place holder for other user routines.
 *                It is a mixture of both USB and non-USB tasks.
 *
 * Note:            None
 ********************************************************************/
void ProcessIO(void)
{
   //BlinkUSBStatus();
   // User Application USB tasks
   if((usb_device_state < CONFIGURED_STATE)||(UCONbits.SUSPND==1)) return;

   ServiceRequests();

}//end ProcessIO


void ServiceRequests(void)
{
   byte index;
   mLED_3_On();mLED_4_On();
   if(USBGenRead((byte*)&dataPacket,sizeof(dataPacket)))
   {
      counter = 0;
      switch(dataPacket.CMD)
      {
         case READ_VERSION:
            //dataPacket._byte[1] is len
            dataPacket._byte[2] = MINOR_VERSION;
            dataPacket._byte[3] = MAJOR_VERSION;
            counter=0x04;
            break;

         case ID_BOARD:
            counter = 0x01;
            if(dataPacket.ID == 0)
            {
               mLED_3_Off();mLED_4_Off();
            }
            else if(dataPacket.ID == 1)
            {
               mLED_3_Off();mLED_4_On();
            }
            else if(dataPacket.ID == 2)
            {
               mLED_3_On();mLED_4_Off();
            }
            else if(dataPacket.ID == 3)
            {
               mLED_3_On();mLED_4_On();
            }
            else
               counter = 0x00;
            break;

         case MULTIPLEX:
            // LED1 & LED2 are used as USB event indicators.
            LATD = dataPacket.led_num;
            if(dataPacket.led_num == 3)
            {
               mLED_3 = dataPacket.led_status;
               counter = 0x01;
            }//end if
            else if(dataPacket.led_num == 4)
```

```
            {
                mLED_4 = dataPacket.led_status;
                counter = 0x01;
            }//end if else
            break;

        case DO_SERVO:
         //Return the sum of the numbers, note no overflow protection present, to keep simplicity.
         dataPacket._byte[1]=Multiplexer(dataPacket._byte[1]);
            counter=0x02;
         break;

        case RESET:
            Reset();
            break;
        default:
            break;
      }//end switch()
      if(counter != 0)
      {
         if(!mUSBGenTxIsBusy())
            USBGenWrite((byte*)&dataPacket,counter);
      }//end if
   }//end if

}//end ServiceRequests

/********************************************************************************
 * Function:        void BlinkUSBStatus(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        BlinkUSBStatus turns on and off LEDs corresponding to
 *                the USB device state.
 *
 * Note:            mLED macros can be found in io_cfg.h
 *                usb_device_state is declared in usbmmap.c and is modified
 *                in usbdrv.c, usbctrltrf.c, and usb9.c
 *******************************************************************************/
void BlinkUSBStatus(void)
{
   static word led_count=0;

   if(led_count == 0)led_count = 10000U;
   led_count--;

   #define mLED_Both_Off()         {mLED_1_Off();mLED_2_Off();}
   #define mLED_Both_On()          {mLED_1_On();mLED_2_On();}
   #define mLED_Only_1_On()        {mLED_1_On();mLED_2_Off();}
   #define mLED_Only_2_On()        {mLED_1_Off();mLED_2_On();}

   if(UCONbits.SUSPND == 1)
   {
      if(led_count==0)
      {
         mLED_1_Toggle();
         mLED_2 = mLED_1;       // Both blink at the same time
      }//end if
   }
   else
   {
      if(usb_device_state == DETACHED_STATE)
      {
         mLED_Both_Off();

      }
      else if(usb_device_state == ATTACHED_STATE)
```

```
        {
            mLED_Both_On();
        }
        else if(usb_device_state == POWERED_STATE)
        {
            mLED_Only_1_On();
        }
        else if(usb_device_state == DEFAULT_STATE)
        {
            mLED_Only_2_On();
        }
        else if(usb_device_state == ADDRESS_STATE)
        {
            if(led_count == 0)
            {
                mLED_1_Toggle();
                mLED_2_Off();
            }//end if
        }
        else if(usb_device_state == CONFIGURED_STATE)
        {
            if(led_count==0)
            {
                mLED_1_Toggle();
                mLED_2 = !mLED_1;       // Alternate blink
            }//end if
        }//end if(...)
    }//end if(UCONbits.SUSPND...)

}//end BlinkUSBStatus

///the core multiplexing function
int Multiplexer(int MicNo)
{
          switch (MicNo)
          {
                    case 1:
                              PINA_On();
                              PINB_On();
                              PINC_On();
                              mLED_1_On();
                              mLED_2_Off();
                              mLED_3_Off();
                              mLED_4_Off();
                              mLED_5_Off();
                              return 1;
                    break;

                    case 2:
                              PINA_Off();
                              PINB_On();
                              PINC_On();
                              mLED_1_Off();
                              mLED_2_On();
                              mLED_3_Off();
                              mLED_4_Off();
                              mLED_5_Off();
                              return 2;
                    break;

                    case 3:
                              PINA_On();
                              PINB_Off();
                              PINC_On();
                              mLED_1_Off();
                              mLED_2_Off();
                              mLED_3_On();
                              mLED_4_Off();
                              mLED_5_Off();
                              return 3;
                    break;

                    case 4:
```

```
                        PINA_Off();
                        PINB_Off();
                        PINC_On();
                        mLED_1_Off();
                        mLED_2_Off();
                        mLED_3_Off();
                        mLED_4_On();
                        mLED_5_Off();
                        return 4;
                break;

                default :
                        PINA_Off();
                        PINB_Off();
                        PINC_Off();
                        mLED_1_Off();
                        mLED_2_Off();
                        mLED_3_Off();
                        mLED_4_Off();
                        mLED_5_On();
                break;
        }
}

/**     EOF     user.c     ***************************************************************/
```

# APPENDIX E
# PC CODING

The coding at the computer part are developed using C# and it contains numerous coding, some of it have been developed for this project and others are obtained from open sources such as code project website; there are two open source used here

I. Camera vision- Video surveillance on C# by Andrew Kirillov it can be found at this link http://www.codeproject.com/KB/audio-video/cameraviewer.aspx. C# file used from this code are:
    a. ByteArray Utilts.cs
    b. Camera.cs
    c. CameraEvents.cs
    d. IVideoSource.cs
    e. JPEGStream.cs
    f. MPEGStream.cs
    g. VideoFileSource.cs
    h. VideoStream.cs
    i. Beside these file there are many DLLs from the same source been used such as AForge.dll, dlink.dll, TigerVideo.VFW.dll and dshow.dll

II. Sound Activated Recorder with Spectrum in C# by Jeff Morton, it can be found at this link http://www.codeproject.com/KB/audio-video/SoundCatcher.aspx. files obtain from this source are all related to audio capturing, audio processing and GUI design some the codes used from this source without modifications WaveIn.cs and WaveNative.cs, other files are modified to suit this applications these files are
    a. FormAboutDialog.cs
    b. FormObtionDialog.cs
    c. FormSettingsDialog.cs
    d. FourierTransfom.cs

III. New file created for this project are:
    a. Form1.cs //this is the main entry for the project
    b. SpectrumAnalyzer.cs // this contains the core audio processing components.

c. MicsClass.cs // the class the define the microphones

d. USBPIC // handle the USB communications

In this appendix only the new files and the modified file are listed others can be found at the specified websites.

1. Form1.cs this the main entry point of the project

```
//this is the main entry of the project it contain the function to perform the image capturing, and it
//receive the audio signal and pass the audio processing units


using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Threading;
using System.Windows.Forms;
using System.IO;
using System.Net;

namespace Sound_Multiplexer
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void LoadMainForm(object sender, EventArgs e)
      {
         string Dlink_Source;
         samples = 44100;
         PrePoint = new Point(CameraWindow.Width / 2, CameraWindow.Height / 2);
         Dlink_Source = "http://192.168.0.101/cgi-bin/video.jpg";
         JPEGStream Source = new JPEGStream();
         Source.VideoSource = Dlink_Source;
         OpenVideoSource(Source);

         if (WaveNative.waveInGetNumDevs() == 0)
            PropertiesWindow.AppendText("No audio device been detected");
         else
         {
            MicsNo = Properties.Settings.Default.NumberOfMicrophones;
            MuxPeriod = Properties.Settings.Default.MicMuxPeriod;
            Mic = new MicsClass[MicsNo];
            for (int i = 0; i < MicsNo; i++)
            {
               Mic[i] = new MicsClass();
            }

            PropertiesWindow.AppendText("Audio device is been detected");
            PropertiesWindow.AppendText(Time().ToString());
            MuxThread = new Thread(Multiplexing);
            Start();
         }
      }

      private void ResizeMainForm(object sender, EventArgs e)
      {
         if (_isShown&this.WindowState == FormWindowState.Minimized)
         {
            foreach (Form f in this.MdiChildren)
            {
```

```
                f.WindowState = FormWindowState.Normal;
            }
            this.ShowInTaskbar = false;
            this.Visible = false;
            _isShown = false;
        }

    }


    private void ClosingMainForm(object sender, FormClosingEventArgs e)
    {
        Stop();
        this.Dispose(true);
    }

    private void Start()
    {
        Stop();
        MuxThread = new Thread(Multiplexing);
        MuxThread.Start();
        try
        {
            m_WaveFormat = new WaveFormat(samples, BitperSample, Channels);
            m_WaveInRecorder = new
WaveInRecorder(Properties.Settings.Default.SettingAudioInputDevice, m_WaveFormat,
BytesPerFrame *Channels, 3, new BufferDoneEventHandler(m_DataArrived));
        }
        catch (Exception ex)
        {
            PropertiesWindow.AppendText(DateTime.Now + " : " + ex.InnerException.ToString()
+ "\r\n");
        }

    }

    private void Stop()
    {
        if (m_WaveInRecorder != null)
            try
            {
                m_WaveInRecorder.Dispose();
            }
            finally
            {
                m_WaveInRecorder = null;
            }
        if(MuxThread !=null)
            try
            {
                MuxThread.Abort();
            }
            finally
            {
                MuxThread = null;
            }
    }

    #region Process vedio sources
    /// <summary>
    /// Get the frame sequence form the given vedio source
    /// </summary>
    /// <param name="source"></param>
    private void OpenVideoSource(IVideoSource source)
    {
        CloseAllVedioSources();

        // create camera
        camera1 = new Camera(source);
        // start camera
        camera1.Start();

        // attach camera to camera window
```

```
            //cameraWindow1.Camera = camera;

            // set event handlers
            camera1.NewFrame += new EventHandler(camera_NewFrame);

        }

        /// <summary>
        /// Close any exiting veido source
        /// </summary>
        private void CloseAllVedioSources()
        {
            if (camera1 != null)
            {
                // signal camera to stop
                camera1.SignalToStop();
                // wait for the camera
                camera1.WaitForStop();

                camera1 = null;
            }
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void camera_NewFrame(object sender, System.EventArgs e)
        {
            string URLCommand;
            CameraWindow.Image = camera1.LastFrame;
            if ((Time() - m_TimeLastDetection) > 1000)
            {
                if (CameraLocation != PreLocation)
                {
                    //CameraControl("[&move=home]");
                    switch (CameraLocation)
                    {
                        case 0:
                            URLCommand = "[&move=]";
                            CameraControl(URLCommand);
                            break;
                        case 1: // move right
                            URLCommand = "[&move=home]";
                            CameraControl(URLCommand);
                            URLCommand = "[&move=right][&speedpan=" + Math.Abs(5).ToString() +
"]";
                            CameraControl(URLCommand);
                            break;
                        case 2:  //move up
                            URLCommand = "[&move=home]";
                            CameraControl(URLCommand);
                            URLCommand = "[&move=up][&speedtilt=" + Math.Abs(1).ToString() + "]";
                            CameraControl(URLCommand);
                            break;
                        case 3:   // move down
                            URLCommand = "[&move=home]";
                            CameraControl(URLCommand);
                            URLCommand = "[&move=down][&speedtilt=" + Math.Abs(5).ToString() +
"]";
                            CameraControl(URLCommand);
                            break;
                        case 4: // move left
                            URLCommand = "[&move=home]";
                            CameraControl(URLCommand);
                            URLCommand = "[&move=left][&speedpan=" + Math.Abs(5).ToString() +
"]";
                            CameraControl(URLCommand);
                            break;
                        default:
                            URLCommand = "";
                            break;
```

```
                }
                PreLocation = CameraLocation;
                m_TimeLastDetection = Time();
            }
        }
    }


    private void CameraControl(string urlcommand)
    {
        HttpWebRequest req = null;
        HttpWebResponse resp = null;

        try
        {
            req = (HttpWebRequest)HttpWebRequest.Create("http://192.168.0.101/cgi-
bin/camctrl.cgi?" + urlcommand);
            resp = (HttpWebResponse)req.GetResponse();
        }
        catch (Exception ex)
        {
            System.Diagnostics.Debug.WriteLine("==============: " + ex.Message);
        }
        finally
        {
            // abort request
            if (req != null)
            {
                req.Abort();
                req = null;
            }
            // close response
            if (resp != null)
            {
                resp.Close();
                resp = null;
            }
        }
    }

    private void MouseClick(object sender, MouseEventArgs e)
    {
        string URLCommand;
        Point p = new Point(e.X, e.Y);
        int X, Y;
        int width, height;
        PropertiesWindow.AppendText("e.X = " + e.X.ToString() + "    e.Y = " + e.Y.ToString()
+ "\n");
        width = CameraWindow.Width / 2;
        height = CameraWindow.Height / 2;
        X = 5 * (p.X - width) / width;
        Y = 5 * (p.Y - height) / height;
        PropertiesWindow.AppendText("X = " + X.ToString() + "    Y = " + Y.ToString() +
"\n\n");
        //x axis movements
        if (((e.X / width) > 0.8) && ((e.X / width) < 1.2) && ((e.Y / height) > 0.8) && ((e.Y /
height) < 1.2))
        {
            URLCommand = "[&move=home]";
            CameraControl(URLCommand);
        }
        else if (X > 0)
        {
            URLCommand = "[&move=left][&speedpan=" + Math.Abs(X).ToString() + "]";
            CameraControl(URLCommand);
        }
        else if (X < 0)
        {
            URLCommand = "[&move=right][&speedpan=" + Math.Abs(X).ToString() + "]";
            CameraControl(URLCommand);
        }
```

```
            //y axis movements
            if (Y > 0)
            {
                URLCommand = "[&move=down][&speedtilt=" + Math.Abs(X).ToString() + "]";
                CameraControl(URLCommand);
            }
            else if (Y < 0)
            {
                URLCommand = "[&move=up][&speedtilt=" + Math.Abs(X).ToString() + "]";
                CameraControl(URLCommand);
            }

        }
        #endregion process vedio sources
        /// <summary>
        /// multiplexing core function
        /// this callback function multiplex the audio input to 4 different wave forms
        /// </summary>
        /// <param name="data"></param>
        /// <param name="size"></param>
        private void m_DataArrived(IntPtr data, int size)
        {
            m_TimeCurrentDetection = Time();
            if (Isfirst)
            {
                StartTime = Time();
                m_TimeCounter = Time();
                for (int i = 0; i < MicsNo; i++)
                {
                    if (Mic[i].m_MicData == null || Mic[i].m_MicData.Length != size)
                        Mic[i].m_MicData = new byte[size];
                    if (Mic[i].m_MicData != null)
                    {
                        System.Runtime.InteropServices.Marshal.Copy(data, Mic[i].m_MicData, 0,
size);
                        Mic[i].Sectrumanalyzer = new SectrumAnalyzer();
                        Mic[i].Sectrumanalyzer.SectraAnalyzer(ref Mic[i].m_MicData,0);
                        Mic[MuxIndex].Status = 0;
                        Mic[i].MicRefTime = Time();
                    }
                }

                Isfirst = false;
            }
            else
            {
                MuxIndex = Port.LiveMic;
                if ((Mic[MuxIndex].m_MicData != null) && (Port.DeadFlag))
                {
                    Mic[MuxIndex].Sectrumanalyzer.MicFlag = Port.DeadFlag;
                    System.Runtime.InteropServices.Marshal.Copy(data, Mic[MuxIndex].m_MicData,
0, size);
                    Mic[MuxIndex].Sectrumanalyzer.SectraAnalyzer(ref
Mic[MuxIndex].m_MicData,MuxIndex);
                    switch (MuxIndex)
                    {
                        case 0:
                            Mic[MuxIndex].Sectrumanalyzer.Draw(ref WaveViewerMic1);
                            break;
                        case 1:
                            Mic[MuxIndex].Sectrumanalyzer.Draw(ref WaveViewerMic2);
                            break;
                        case 2:
                            Mic[MuxIndex].Sectrumanalyzer.Draw(ref WaveViewerMic3);
                            break;
                        case 3:
                            Mic[MuxIndex].Sectrumanalyzer.Draw(ref WaveViewerMic4);
                            break;
                    }
                    Mic[MuxIndex].MicRefTime = Time();
                }

                if ((Time() - m_TimeCounter) > 2000)
```

```
            {
                //Invoke(new MethodInvoker(AmplitudeEvent));
                int temp2 = 0;
                temp = 5;
                for (int i = 0; i < MicsNo; i++)
                {
                    if ((Mic[i].Sectrumanalyzer.Counter >
temp2)&&(Mic[i].Sectrumanalyzer.Counter > 1))
                    {
                        temp = i;
                        temp2 = Mic[i].Sectrumanalyzer.Counter;
                    }
                }
                for (int i = 0; i < MicsNo; i++)
                {
                    if (i == temp)
                    {
                        CameraLocation = i + 1;
                        Mic[i].MuxFlag = true;
                    }
                    else
                    {
                        Mic[i].MuxFlag = false;
                    }
                }
                if (temp == 5)
                    CameraLocation = 0;

                for (int i = 0; i < MicsNo; i++)
                {
                    Mic[i].Sectrumanalyzer.Counter = 0;
                }
                m_TimeCounter = Time();
            }

            Invoke(new MethodInvoker(AmplitudeEvent));
        }
    }

    /// <summary>
    /// time delay
    /// </summary>
    /// <param name="Period"></param>
    private void DelayMS(int Period)
    {
        long time = Time();
        while ((Time() - time) > Period)
            continue;
    }

    /// <summary>
    /// massage invoker
    /// </summary>
    private void AmplitudeEvent()
    {
        for (int i = 1; i < MicsNo+1; i++)
        {
            //PropertiesWindow.AppendText(("Ccounter = "+ Mic[i -
1].Sectrumanalyzer.Counter.ToString() + "\n"));
            if (Mic[i - 1].MuxFlag)
            {

                Font f = new Font("Arial", 14);
                PropertiesWindow.Clear();
                PropertiesWindow.AppendText(DateTime.Now.ToString() + "\n");
                PropertiesWindow.Font = f;
                PropertiesWindow.AppendText("\n");
                PropertiesWindow.AppendText("ACTIVE MIC = " + i.ToString() + "\n");
                PropertiesWindow.AppendText("Ccounter = "+ Mic[i -
1].Sectrumanalyzer.Counter.ToString() + "\n");
                PropertiesWindow.AppendText("CameraLocation = " + i.ToString() + "\n");
                PropertiesWindow.AppendText("Previous CameraLocation = " +
CameraLocation.ToString() + "\n");
```

```
                switch (i)
                {
                    case 1:
                        PropertiesWindow.AppendText("ACTIVE MIC IS AT THE RIGHT SIDE");
                        break;
                    case 2:
                        PropertiesWindow.AppendText("ACTIVE MIC IS AT THE UPPER SIDE");
                        break;
                    case 3:
                        PropertiesWindow.AppendText("ACTIVE MIC IS AT THE LOWER SIDE");
                        break;
                    case 4:
                        PropertiesWindow.AppendText("ACTIVE MIC IS AT THE LEFT SIDE");
                        break;
                    default:
                        break;
                }
            }
        }
    }

    /// <summary>
    /// this function gives the current time in millisecond format
    /// </summary>
    /// <returns></returns>
    private long Time()
    {
        return DateTime.Now.Millisecond + DateTime.Now.Second * 1000 +
DateTime.Now.Minute * 60000+ DateTime.Now.Hour *3600*1000;
        //return DateTime.Now.Ticks;
    }

    private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
    {
        FormAboutDialog form = new FormAboutDialog();
        form.Show();
    }
    private void exitToolStripMenuItem_Click(object sender, EventArgs e)
    {
        this.Close();
    }
    private void optionsToolStripMenuItem_Click(object sender, EventArgs e)
    {
        FormOptionsDialog form = new FormOptionsDialog();
        if (form.ShowDialog() == DialogResult.OK)
        {
            for (int i = 0; i < MicsNo; i++)
            {

            }
        }
    }
    private void settingsToolStripMenuItem_Click(object sender, EventArgs e)
    {
        FormSettingsDialog form = new FormSettingsDialog();
        if (form.ShowDialog() == DialogResult.OK)
        {
            Stop();

            Start();
        }
    }

    private void Multiplexing()
    {
        int mics = MicsNo;
        Port = new USBPIC();
        while (true)
        {

            Port.UpdateMics(mics, 1);
            MuxIndex = Port.LiveMic;
            Mic[MuxIndex].MuxFlag = Port.DeadFlag;
```

```
        }
    }

    private WaveFormat m_WaveFormat;
    private WaveInRecorder m_WaveInRecorder;
    private long m_TimeLastDetection = 0;
    private long m_TimeCurrentDetection = 0;
    private long m_TimeCounter = 0;
    private MicsClass [] Mic;
    private bool _isShown = true;
    private byte MicsNo;
    private int MuxPeriod;
    private int temp;
    private long TimeSpan;
    private long StartTime;
    private int MuxIndex;
    private bool Isfirst = true;
    private int samples = Properties.Settings.Default.SettingSamplesPerSecond;
    private byte Channels = 1;
    private byte BitperSample = Properties.Settings.Default.SettingBitsPerSample;
    private int ComparisonPeriod = Properties.Settings.Default.ComparisonPeriod;
    private int BytesPerFrame = Properties.Settings.Default.SettingBytesPerFrame;
    private int ComparisonThreshold = Properties.Settings.Default.CamparisonThreshod;
    private Thread MuxThread;
    USBPIC Port;
    private Point PrePoint;
    private Camera camera1;
    private int CameraLocation = 0;
    private int PreLocation = 0;
  }
}
```

2. SpectrumAnalayzer the part contain the main audio processing and visualizaiton component

```
using System;
using System.Collections;
using System.Drawing;
using System.Drawing.Imaging;
using System.Windows.Forms;

namespace Sound_Multiplexer
{
  class SectrumAnalyzer
  {

    /// <summary>
    /// defaut constructor
    /// </summary>
    public SectrumAnalyzer()
    {
    }

    /// <summary>
    ///default structure
    /// </summary>
    ~SectrumAnalyzer()
    {
    }

    /// <summary>
    /// Main fuction in the class
    /// </summary>
    public void SectraAnalyzer(ref byte[] InputData, int id)
    {
      double [] Data;
      ID = id;
      SplitInputData(ref InputData);
      Data = new double[waveLeft.Length];
      Data = TimeDomainFilter(ref waveLeft);
      //waveLeft = SpectrumFilter(ref Data);
```

```
    waveLeft = FourierTransform.FFT(ref Data);
    //waveLeft = FourierTransform.IFFT(ref waveLeft);
}

/// <summary>
/// draw from outside the class
/// </summary>
/// <param name="Box"></param>
public void Draw(ref PictureBox Box)
{
  //RenderTimeDomain(ref waveOut, ref Box);
  RenderFrequencyDomain(ref waveLeft, ref Box, samples);
  CalculateAmp(ref waveLeft);
}

/// <summary>
/// Split 16 bit sample
/// </summary>
/// <param name="wave"></param>
public void SplitInputData(ref byte[] wave)
{
    waveLeft = new double[wave.Length / ((BitperSample/8) * Channels)];
    waveOut = new double[waveLeft.Length];

    // Split out channels from sample
    int h = 0;
    for (int i = 0; i < wave.Length; i += ((BitperSample/8) * Channels))
    {
        //only applicabel for 16 bit operations
        waveLeft[h] = (double)BitConverter.ToInt16(wave, i);
        waveOut[h] = (double)BitConverter.ToInt16(wave, i);
        h++;
    }
}


/// <summary>
/// copy the data from A parameter to B paramerer
/// </summary>
/// <param name="A"></param>
/// <param name="B"></param>
public void CopyAtoB(ref double[] A, ref double [] B)
{
    for (int i = 0; i < B.Length; i++)
    {
          A[i] = B[i];
    }
}

/// <summary>
/// Calcuate the average amplutiude of the given wave
/// </summary>
/// <param name="wave"></param>
/// <returns></returns>
private double CalculateAmp(ref double[] wave)
{
    double sum = 0;
    for (int i = 3; i < 60; i++)
    {
        //sum += Math.Abs(wave[i]);
         //sum += Math.Round((wave[i]/wave.Length)/1000);
        if (wave[i] > 100000)
            sum++;
    }
    //sum /= wave.Length;
    if (sum > 30)
    {
        micFlag = true;
        counter++;
    }
    return sum;
}
```

```
        /// <summary>
        /// Generate the frequency spectrum for both Left and Right waves
        /// </summary>
        /// <param name="WaveLeft"></param>
        /// <param name="WaveRight"></param>
        public double[] FftGenerator(ref double[] Wave)
        {
            double[] fft = new double[Wave.Length];
            // Generate frequency domain data in decibels
            fft = FourierTransform.FFT(ref Wave);
            return fft;
        }


        /// <summary>
        /// Generate the time domain signal using the inverse
        /// IFFT algorithm
        /// </summary>
        /// <param name="Wave"></param>
        /// <returns></returns>
        public double[] IfftGenerator(ref double[] Wave)
        {
            double[] ifft = new double[Wave.Length];
            // Generate frequency domain data in decibels
            ifft = FourierTransform.IFFT(ref Wave);
            return ifft;
        }


        /// <summary>
        /// Analyze the time domain and remove noise from it
        /// </summary>
        /// <param name="Wave"></param>
        /// <returns></returns>
        public double[] TimeDomainFilter(ref double[] Wave)
        {
            double[] InputWave = new double[Wave.Length];
            for (int x = 0; x < Wave.Length; x++)
            {
                if (!micFlag)
                {
                    InputWave[x] = 0;
                }
                else if (Wave[x] == double.NegativeInfinity || Wave[x] == double.PositiveInfinity ||
Wave[x] < double.MinValue/3 || Wave[x] > double.MaxValue/3)
                {
                    InputWave[x] = 0;
                }
                else
                {
                    InputWave[x] = Wave[x];
                }
            }
            return InputWave;
        }

        /// <summary>
        /// analyze the spectrum and remove noise
        /// </summary>
        /// <param name="FftLeft"></param>
        /// <param name="FFTRight"></param>
        public double [] SpectrumFilter(ref double[] Wave)
        {
            double[] InputFft = new double[Wave.Length];
            InputFft = FftGenerator(ref Wave);

            double scaleHz = (double)(samples /2) / (double)Wave.Length;
            double FreqMin = (double)(double)(MinFrequency / scaleHz);
            double FreqMax = (double)(double)(MaxFrequency / scaleHz);

            // Spectrum Filter
            for (int x = 0; x < InputFft.Length; x++)
            {
```

```
            if(InputFft[x] < 5000)
            {
               InputFft[x] = 0;
            }
            if (x > FreqMax)
            {
               InputFft[x] = 0;
            }
            if (x < FreqMin)
            {
               InputFft[x] = 0;
            }
            if (InputFft[x] == double.NegativeInfinity || InputFft[x] == double.PositiveInfinity ||
InputFft[x] == double.MinValue || InputFft[x] == double.MaxValue)
            {
              InputFft[x] = 0;
            }
         }
         // call an inverse FFT function before the return
         //InputFft = IfftGenerator(ref InputFft);
         return InputFft;
      }

      /// <summary>
      /// the output audio waveform
      /// </summary>
      public double[] WaveOut
      {
         get { return waveOut; }
         set { waveOut = value; }
      }

      /// <summary>
      /// determine noise data or not
      /// </summary>
      public bool MicFlag
      {
         get { return micFlag; }
         set { micFlag = value; }
      }

      public int Counter
      {
         get { return counter; }
         set {counter = value;}
      }

      public double Amplitude
      {
         get
         {
            amplitude = CalculateAmp(ref waveLeft);
            return amplitude;
         }
         set
         { amplitude = value;}
      }


      /// <summary>
      /// Render time domain to PictureBox
      /// </summary>
      /// <param name="Wave"></param>
      /// <param name="pictureBox"></param>
      public void RenderTimeDomain(ref double [] Wave,ref PictureBox pictureBox)
      {
         // Set up for drawing
         Bitmap canvas = new Bitmap(pictureBox.Width, pictureBox.Height);
         Graphics offScreenDC = Graphics.FromImage(canvas);
         Pen pen = new System.Drawing.Pen(Color.WhiteSmoke);
         Font font = new Font("Arial", 14);
         SolidBrush brush = new System.Drawing.SolidBrush(Color.Brown);
```

```
            // Determine channnel boundries
            int width = canvas.Width;
            int height = canvas.Height;
            double center = height / 2;
            double sum = CalculateAmp(ref Wave);
            // Draw left channel
            double scale = 0.5 * height / 32768;  // a 16 bit sample has values from -32768 to 32767
            int xPrev = 0, yPrev = 0;
            for (int x = 0; x < width; x++)
            {
               int y = (int)(center + (Wave[Wave.Length / width * x] * scale));
               if (x == 0)
               {
                  xPrev = 0;
                  yPrev = y;
               }
               else
               {
                  pen.Color = Color.Blue;
                  offScreenDC.DrawLine(pen, xPrev, yPrev, x, y);
                  xPrev = x;
                  yPrev = y;
               }
            }
            /*for (int x = 0; x < width; x++)
            {
               int y = (int)(center + (waveLeft[waveLeft.Length / width * x] * scale));
               if (x == 0)
               {
                  xPrev = 0;
                  yPrev = y;
               }
               else
               {
                  pen.Color = Color.Red;
                  offScreenDC.DrawLine(pen, xPrev, yPrev, x, y);
                  xPrev = x;
                  yPrev = y;
               }
            }*/
            ID++;
            offScreenDC.DrawString("MIC " +ID.ToString(), font, brush, canvas.Width/2 - 15, 5);
            //offScreenDC.DrawString("Sum " + sum.ToString(), font, brush, canvas.Width / 2 - 15, 10);
            ID--;
            // Clean up
            pictureBox.Image = canvas;
            offScreenDC.Dispose();
         }

         /// <summary>
         /// Render frequency domain to PictureBox
         /// </summary>
         /// <param name="FFTSpect"></param>
         /// <param name="pictureBox"></param>
         /// <param name="samples"></param>
         public void RenderFrequencyDomain(ref double [] FFTSpect, ref PictureBox pictureBox, int
samples)
         {
            // Set up for drawing
            Bitmap canvas = new Bitmap(pictureBox.Width, pictureBox.Height);
            Graphics offScreenDC = Graphics.FromImage(canvas);
            SolidBrush brush = new System.Drawing.SolidBrush(Color.Brown);
            Pen pen = new System.Drawing.Pen(Color.Green);
            Font font = new Font("Arial", 10);
            //samples = 44100;

            // Determine channnel boundries
            int width = canvas.Width;
            int height = canvas.Height;

            double min = double.MaxValue;
            double minHz = 0;
            double max = double.MinValue;
```

```
            double maxHz = 0;
            double range = 0;
            double scale = 0;
            double scaleHz = (double)(samples / 2) / (double)FFTSpect.Length;
            double FreqMin = (double)(double)(MinFrequency / scaleHz);
            double FreqMax = (double)(double)(MaxFrequency/ scaleHz);
            double sum = CalculateAmp(ref FFTSpect);

            // get min/max
            for (int x = 0; x < FFTSpect.Length; x++)
            {
                double amplitude = FFTSpect[x];
                if (min > amplitude)
                {
                    min = amplitude;
                    minHz = (double)x * scaleHz;
                }
                if (max < amplitude)
                {
                    max = amplitude;
                    maxHz = (double)x * scaleHz;
                }
            }

            // get range
            if (min < 0 || max < 0)
                if (min < 0 && max < 0)
                    range = max - min;
                else
                    range = Math.Abs(min) + max;
            else
                range = max - min;
            scale = range / height;

            // draw channel
            for (int xAxis = 0; xAxis < width; xAxis++)
            {
                double amplitude = (double)FFTSpect[(int)(FreqMin + ((double)(FreqMax - FreqMin) /
(double)(width)) * xAxis)];
                if (amplitude == double.NegativeInfinity || amplitude == double.PositiveInfinity ||
amplitude == double.MinValue || amplitude == double.MaxValue)
                    amplitude = 0;
                int yAxis;
                if (amplitude < 0)
                    yAxis = (int)(height - ((amplitude - min) / scale));
                else
                    yAxis = (int)(0 + ((max - amplitude) / scale));
                if (yAxis < 0)
                    yAxis = 0;
                if (yAxis > height)
                    yAxis = height;
                //pen.Color = Color.FromArgb(0, GetColor(min, max, range, amplitude), 0);
                pen.Color = Color.Blue;
                offScreenDC.DrawLine(pen, xAxis, height, xAxis, yAxis);
            }
            offScreenDC.DrawString("Main Frequency: " + maxHz.ToString(".#") + "±" +
scaleHz.ToString(".#") + "Hz" + " ", font, brush, 0 + 1, 0 + 18);
            offScreenDC.DrawString("Amp. Sum " + sum.ToString(), font, brush, canvas.Width / 2 - 15,
20);
            ID++;
            offScreenDC.DrawString("MIC " + ID.ToString(), font, brush, canvas.Width / 2 - 15, 5);
            ID--;

            // Clean up
            pictureBox.Image = canvas;
            offScreenDC.Dispose();
        }

        /// <summary>
        /// Get color in the range of 0-255 for amplitude sample
        /// </summary>
        /// <param name="min"></param>
        /// <param name="max"></param>
```

```
        /// <param name="range"></param>
        /// <param name="amplitude"></param>
        /// <returns></returns>
        private static int GetColor(double min, double max, double range, double amplitude)
        {
            double color;
            if (min != double.NegativeInfinity && min != double.MaxValue & max !=
double.PositiveInfinity && max != double.MinValue && range != 0)
            {
                if (min < 0 || max < 0)
                    if (min < 0 && max < 0)
                        color = (255 / range) * (Math.Abs(min) - Math.Abs(amplitude));
                    else
                        if (amplitude < 0)
                            color = (255 / range) * (Math.Abs(min) - Math.Abs(amplitude));
                        else
                            color = (255 / range) * (amplitude + Math.Abs(min));
                else
                    color = (255 / range) * (amplitude - min);
            }
            else
                color = 0;
            return (int)color;
        }


        private double[] waveLeft;
        private double[] waveRight;
        private double[] waveOut;
        private double[] spectLeft;
        private double[] spectRight;
        private double[] spectOut;
        private double amplitude;
        int samples = Properties.Settings.Default.SettingSamplesPerSecond;
        int MaxFrequency = Properties.Settings.Default.MaximumFrequency;
        int MinFrequency = Properties.Settings.Default.MinimumFrequency;
        int AmplitudeThreshold = Properties.Settings.Default.SettingAmplitudeThreshold;
        private byte Channels = Properties.Settings.Default.SettingChannels;
        private byte BitperSample = Properties.Settings.Default.SettingBitsPerSample;
        private bool micFlag = true;
        private int ID;
        private int counter = 0;
    }
}
```

3. MicsClass.cs this part define the microphone and its associated parameters

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Sound_Multiplexer
{
    class MicsClass
    {
        /// <summary>
        /// default constructor of the class
        /// </summary>
        public MicsClass()
        {
        }
        /// <summary>
        /// default distructor
        /// </summary>
        ~MicsClass()
        {
        }

        /// <summary>
        /// microphone multiplexing time property
        /// this property features as a reference time
```

```
/// for the multiplexing process
/// </summary>
public long MicRefTime
{
   get
   { return m_MicRefTime; }
   set
   { m_MicRefTime = value; }
}

/// <summary>
/// the average amplitude of the mic for a
/// 10 milliseconds sampling time
/// </summary>
public double MicAvgAmp
{
   get
   { return CalcAvgAmp(m_MicData); }
   set
   { m_MicAvgAmp = value; }
}

/// <summary>
/// multiplexing flag
/// </summary>
public bool MuxFlag
{
   get
   { return m_MuxFlag; }
   set
   { m_MuxFlag = value; }
}

/// <summary>
/// Microphone status counter
/// </summary>
public int Status
{
   get
   { return m_Status; }
   set
   { m_Status = value; }
}

/// <summary>
/// Inherit Spectral analyzer class;
/// </summary>
public SectrumAnalyzer Sectrumanalyzer
{
   get
   { return m_SectrumAnalyzer; }
   set
   { m_SectrumAnalyzer = value; }
}

/// <summary>
/// this method calcultes the average amplitude for
/// one sample
/// </summary>
/// <param name="wave"></param>
/// <returns></returns>
private double CalcAvgAmp(byte[] wave)
{
   lock (this)
   {

   }

   return m_MicAvgAmp;
}

//protected override void Dispose()
//{
```

```
        //}

        private long m_MicRefTime;
        private double m_MicAvgAmp;
        public byte[] m_MicData;
        private SectrumAnalyzer m_SectrumAnalyzer;
        private bool m_MuxFlag = false;
        private int m_Status = 0;
    }
}
```

4. USBPIC.cs this part handles the USB communication part

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO.Ports;

namespace Sound_Multiplexer
{
    class USBPIC
    {
        /// <summary>
        /// default class constructor
        /// </summary>
        public USBPIC()
        {
            USB_Port = new SerialPort();
            USB_Port.PortName = "COM3";
            USB_Port.Open();
        }

        /// <summary>
        /// default distructor
        /// </summary>
        ~USBPIC()
        {
            USB_Port.Close();
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="NoMics"></param>
        /// <param name="CamLocation"></param>
        public void UpdateMics(int NoMics, int CamLocation)
        {
            for (int i = 1; i < NoMics+1; i++)
            {
                switch (i)
                {
                    case 1:
                        buffer[0] = 0x01;
                        break;
                    case 2:
                        buffer[0] = 0x02;
                        break;
                    case 3:
                        buffer[0] = 0x03;
                        break;
                    case 4:
                        buffer[0] = 0x04;
                        break;
                }
                liveMic = i - 1;
                USB_Port.Write(buffer, 0, 1);
                DelayMs(500, 100);
            }

        }
```

```
        /// <summary>
        /// Milliscond delay function
        /// </summary>
        /// <param name="DelayPeriod"></param>
        private void DelayMs(long DelayPeriod, long DeadPeriod)
        {
            long x = Time();
            while (Time() < (x + DelayPeriod))
            {
                if ((Time() > (x + 1*DeadPeriod))&&(Time() < (x + 3*DeadPeriod)))
                {
                    deadFlag = true;
                }
                else
                {
                    deadFlag = false;
                }
            }
        }


        /// <summary>
        /// time clacuation function
        /// </summary>
        /// <returns></returns>
        private long Time()
        {
            return DateTime.Now.Millisecond + DateTime.Now.Second * 1000 + DateTime.Now.Minute *
60000 + DateTime.Now.Hour * 3600 * 1000;
            //return DateTime.Now.Ticks;
        }

        /// <summary>
        /// get the Mic been multiplexed currently
        /// </summary>
        public int LiveMic
        {
            get
            {
                return liveMic;
            }
            set
            {
                liveMic = value;
            }
        }

        /// <summary>
        /// return the current location of the camera
        /// </summary>
        public int CameraLocation
        {
            get
            {
                return ServoLocation;
            }
            set
            {
                ServoLocation = value;
            }

        }

        /// <summary>
        /// Return dead period status
        /// </summary>
        public bool DeadFlag
        {
            get
            {
                return deadFlag;
            }
            set
```

```
            {
                deadFlag = value;
            }

        }

        private int liveMic;
        private int ServoLocation;
        private bool deadFlag;
        private byte[] buffer = {0x00};
        public SerialPort USB_Port;

    }
}
```

5. FourierTransform.cs  this founction perform the FFT and the IFFT

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Sound_Multiplexer
{
    public class FourierTransform
    {
        static private double[] fftspect;
        static private double[] ifftwave;
        static private double[] Temp;
        static private int n, nu;

        /// <summary>
        /// Revese the input bits
        /// e.g: 0b0100011  will be 0b1100010
        /// </summary>
        /// <param name="j"></param>
        /// <returns></returns>
        static private int BitReverse(int j)
        {
            int j2;
            int j1 = j;
            int k = 0;
            for (int i = 1; i <= nu; i++)
            {
                j2 = j1 / 2;
                k = 2 * k + j1 - 2 * j2;
                j1 = j2;
            }
            return k;
        }

        /// <summary>
        /// Calculate the fast Fourier Transform of the given input
        /// </summary>
        /// <param name="x"></param>
        /// <returns></returns>
        static public double[] FFT(ref double[] x)
        {
            // Assume n is a power of 2
            n = x.Length;
            nu = (int)(Math.Log(n) / Math.Log(2));
            int n2 = n / 2;
            int nu1 = nu - 1;
            double[] xre = new double[n];
            double[] xim = new double[n];
            double[] magnitude = new double[n2];
            double tr, ti, p, arg, c, s;
            for (int i = 0; i < n; i++)
            {
                xre[i] = x[i];
                xim[i] = 0.0f;
            }
            int k = 0;
```

```
    for (int l = 1; l <= nu; l++)
    {
       while (k < n)
       {
          for (int i = 1; i <= n2; i++)
          {
             p = BitReverse(k >> nu1);
             arg = 2 * (double)Math.PI * p / n;
             c = (double)Math.Cos(arg);
             s = (double)Math.Sin(arg);
             tr = xre[k + n2] * c + xim[k + n2] * s;
             ti = xim[k + n2] * c - xre[k + n2] * s;
             xre[k + n2] = xre[k] - tr;
             xim[k + n2] = xim[k] - ti;
             xre[k] += tr;
             xim[k] += ti;
             k++;
          }
          k += n2;
       }
       k = 0;
       nu1--;
       n2 = n2 / 2;
    }
    k = 0;
    int r;
    while (k < n)
    {
       r = BitReverse(k);
       if (r > k)
       {
          tr = xre[k];
          ti = xim[k];
          xre[k] = xre[r];
          xim[k] = xim[r];
          xre[r] = tr;
          xim[r] = ti;
       }
       k++;
    }
    for (int i = 0; i < n /2; i++)
       magnitude[i] = (float)(Math.Sqrt((xre[i] * xre[i]) + (xim[i] * xim[i])));
    //return magnitude;
    return magnitude;

}

/// <summary>
/// Calculate the Iverse of the fast Fourier Transform
/// </summary>
/// <param name="comp"></param>
/// <returns></returns>
static public double[] IFFT(ref double [] X)
{
    n = X.Length;
    nu = (int)(Math.Log(n) / Math.Log(2));
    int n2 = n / 2;
    int nu1 = nu - 1;
    double[] xre = new double[n];
    double[] xim = new double[n];
    double tr, ti, p, arg, c, s;
    for (int i = 0; i < n; i++)
    {
       xre[i] = X[i];
       xim[i] = 0.0f;
    }
    int k = 0;
    for (int l = 1; l <= nu; l++)
    {
       while (k < n)
       {
          for (int i = 1; i <= n2; i++)
          {
```

```
                p = BitReverse(k << nu1);
                arg = - 2 * (double)Math.PI * p / n;
                c = (double)Math.Cos(arg);
                s = (double)Math.Sin(arg);
                tr = xre[k + n2] * c + xim[k + n2] * s;
                ti = xim[k + n2] * c - xre[k + n2] * s;
                xre[k + n2] = xre[k] - tr;
                xim[k + n2] = xim[k] - ti;
                xre[k] += tr;
                xim[k] += ti;
                k++;
             }
             k += n2;
          }
          k = 0;
          nu1--;
          n2 = n2 / 2;
       }
       k = 0;
       int r;
       while (k < n)
       {
          r = BitReverse(k);
          if (r > k)
          {
             tr = xre[k];
             ti = xim[k];
             xre[k] = xre[r];
             xim[k] = xim[r];
             xre[r] = tr;
             xim[r] = ti;
          }
          k++;
       }
       for (int i = 0; i < xre.Length; i++)
       {

          xre[i] = (xre[i] / xre.Length)*2;
       }
       return xre;

    }


    static public double[] fft(ref double[] x)
    {
       fftspect = new double[x.Length];
       Temp = new double[x.Length];
       fft_rec(ref x, ref fftspect, ref Temp, 0, 1,x.Length);
       return fftspect;
    }

    static private void fft_rec(ref double[] x, ref double[] X, ref double[] Temp, int offset, int
delta,int N)
    {
       int N2 = N/2;
       int k;
       int k00, k01, k10, k11;
       double cs, ss, tr, ti;
       if (N != 2)
       {
          fft_rec(ref x, ref Temp, ref X, offset, 2 * delta, N2);
          fft_rec(ref x, ref Temp, ref X, offset + delta, 2 * delta, N2);

          for (k = 0; k < N2; k++)
          {
             k00 = offset + k * delta;
             k01 = k00 + N2 * delta;
             k10 = offset + 2 * k * delta;
             k11 = k10 + delta;
             double angle = 2 * Math.PI * k /N;
             cs = Math.Cos(angle);
             ss = Math.Sin(angle);
```

```
            if ((k11 + N2) < (X.Length - 1))
            {
               tr = Temp[k11] * cs + Temp[k11 + N2] * ss;
               ti = Temp[k11 + N2] * cs - Temp[k11] * ss;
               X[k01] = Temp[k10] - tr;
               if ((k01 + N2) < (X.Length - 1))
                  X[k01 + N2] = Temp[k10 + N2] - ti;
               X[k00] = Temp[k10] + tr;
               if ((k10 + N2) < (X.Length - 1))
                  X[k00 + N2] = Temp[k10 + N2] + ti;
            }

         }
      }
      else
      {
         k00 = offset;
         k01 = k00 + delta;
         X[k01] = x[k00] - x[k01];
         if ((k01 + N2) < (X.Length - 1))
            X[k01 + N2] = x[k00 + N2] - x[k01 + N2];
         X[k00] = x[k00] + x[k01];
         if (((k00 + N2) < (X.Length - 1))&&((k01 + N2) < (X.Length - 1)))
         X[k00 + N2] = x[k00 + N2] + x[k01 + N2];
      }
   }

   static public double[] ifft(ref double[] X)
   {
      ifftwave = new double[X.Length];
      Temp = new double[X.Length];
      int N = X.Length;
      int N2 = N / 2;
      int i;
      double tr, ti;
      ifftwave = fft(ref X);
      ifftwave[0] = ifftwave[0] / N;
      ifftwave[N2] = ifftwave[N2] / N;
      for (i = 1; i < N2; i++)
      {
         tr = ifftwave[i] / N;
         ti = ifftwave[i + N2] / N;
         ifftwave[i] = ifftwave[N - i] / N;
         ifftwave[i + N2] = ifftwave[N - (i + N2)] / N;
         ifftwave[N - i] = tr;
         ifftwave[N - (i + N2)] = ti;
      }
      return ifftwave;
   }

}

public class Complex
{
   /// <summary>
   /// real part
   /// </summary>
   public double[] Real
   {
      get { return Re; }
      set { Re = value; }
   }

   /// <summary>
   /// imaginary part
   /// </summary>
   public double[] Imagiary
   {
      get { return Im; }
      set { Im = value; }
   }

   /// <summary>
```

```
        ///
        /// </summary>
        /// <param name="real"></param>
        /// <param name="img"></param>
        Complex(double[] real, double[] img)
        {
            Re = new double[real.Length];
            Im = new double[img.Length];
            Re = real;
            Im = img;
        }

        /// <summary>
        ///
        /// </summary>
        ~Complex()
        {
        }
        private double[] Re;
        private double[] Im;
    }
}
```

5. MainForm user interface, this is a view of how the GUI for the main application look like, the window is been designed by the aid of open source codes.

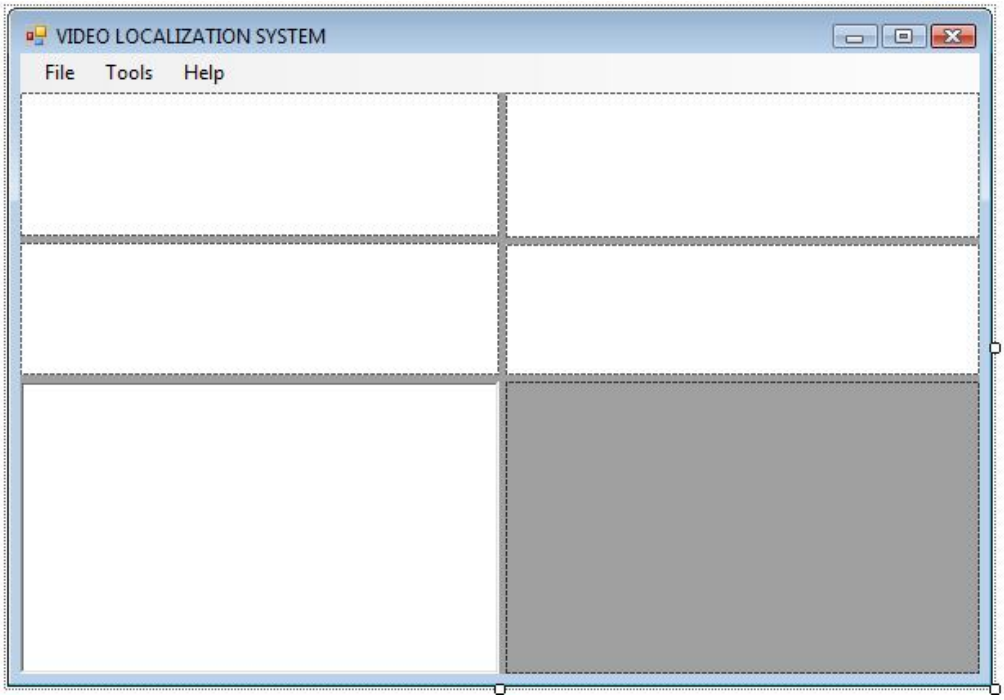


**Figure 29: Main User interface.**

6. Settings modification user interface, this is a video setting modifier user interface window, this window is been designed by the help of open source sources

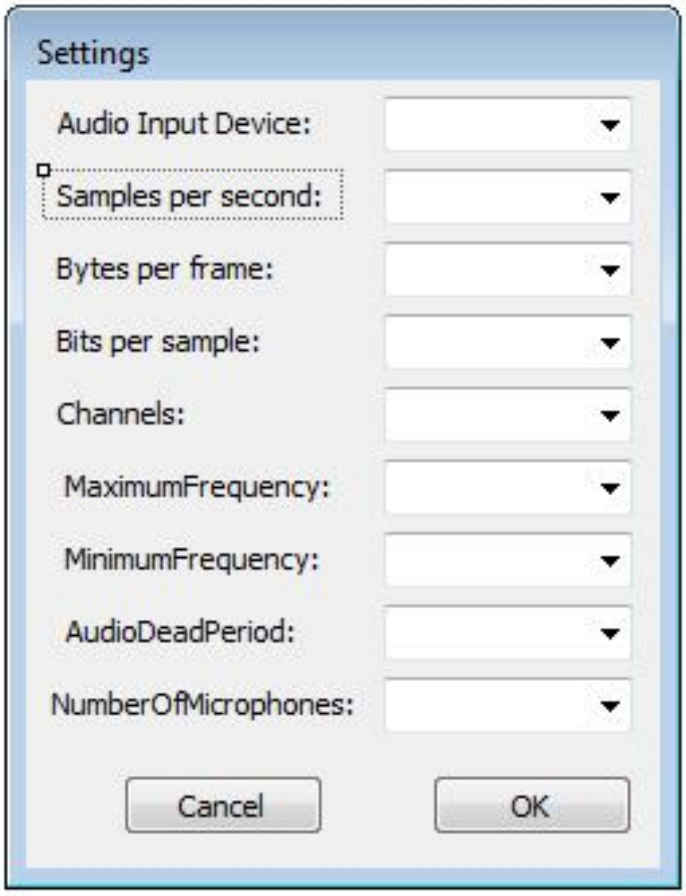


**Figure 30: Setting modification user interface.**

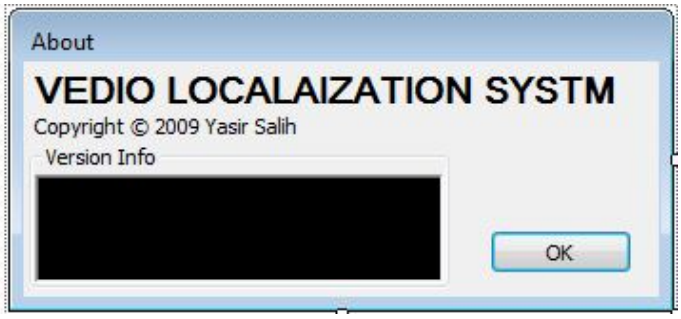


**Figure 31: Project descriptor user interface.**