ECE 445

Spring 2013

Final Paper

2/27/13

Voice tracing video camera designed for meeting recording

Team #42

Jiehan Yao, Yuxiao Lu, Qi Yang

Prof. Scott Carney

TA: Dennis Yuan

Table of Contents

1. Introduction

1.1 Statement of Purpose

Recording an important meeting by video frequently happens in workplaces. Under usual circumstances, a person will be hired to record the entire process of the meeting, where issues of cost, convenience and security arise. Thus we will design a system that would automatically record the meeting in a medium sized room by tracing the voice of the speakers.

Furthermore, we believe during a meeting, arguments, discusses and other forms of information exchange between different speakers are worth capturing. Thus we will provide a mechanism that automatically locates and records the two most important speakers in the ongoing conversation.

1.2 Objective

1) Goals:

For this project we want to implement a voice tracking camera that can select and record up to two active speakers at any time during meetings. The product should improve the quality of meeting recording by being able to capture two speakers at the same time when necessary; it should be highly automated so not much manual adjusting is needed; it should produce high quality recording in a moderate size meeting room. Moreover, the product should be affordable and should be at an acceptable size.

2) Functions / Features:

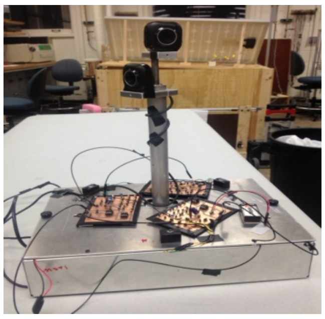
- High-quality video recording
- Each camera is able to rotate to the 24 different directions.
- The two cameras can capture two speakers at a given time.
- Two camera work independently (i.e. no interference).
- Auto-generate splits- screens when both camera are shooting
- Smartly choose the two most important speakers when multiple people participating the conversation.

3) Benefits

- Recording meeting process without cameraman and at a lower cost.
- Provide a solution to best capture information exchanges (e.g. Q&A, argument or discuss) between two people during the meeting.
- 100% automated program. Plug and use.

4) Structure Overview

Two cameras are mounted on 2 rotational rods. The microphones are mounted on the surface of the base. Motors and microphones are powered by 9V batteries which are placed inside the box. All calculation is done by the Arduino Uno board which is powered by USB. The video outputs of the cameras are sent to computer through USB cables. The C++ client program will process the data and display the image.



2. Design Overview

2.1 Block Diagram

1) Function-units data flow

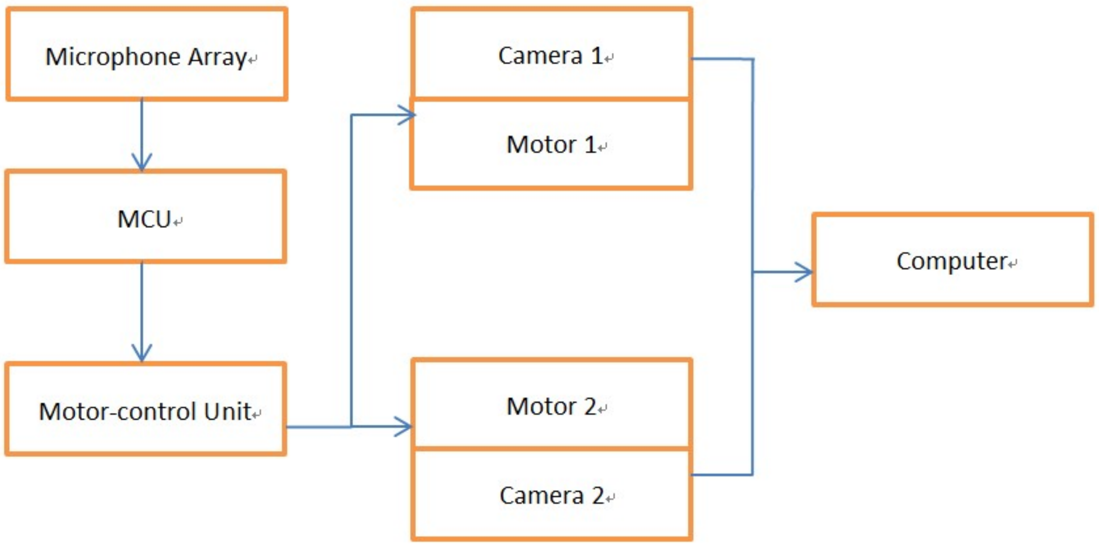


Figure1. Data flow diagram

2.2 Block Description and Design Methodology

1) Microphone Array

This module consists of 3 microphones (MIC2343) located on the vertices of an equilateral triangle. The information of the Microphone Array would be collected and analyzed by the Micro-Controller Unit (the MCU module) to estimate the direction of the speaker. We choose omnidirectional microphones so that they can receive sound information from each direction equally. The signals would need to be amplified because the signals directly coming from the microphones are too small to be processed by the MCU. They would also need to be filtered to limit the frequency within the range of human voice and to eliminate background noise.

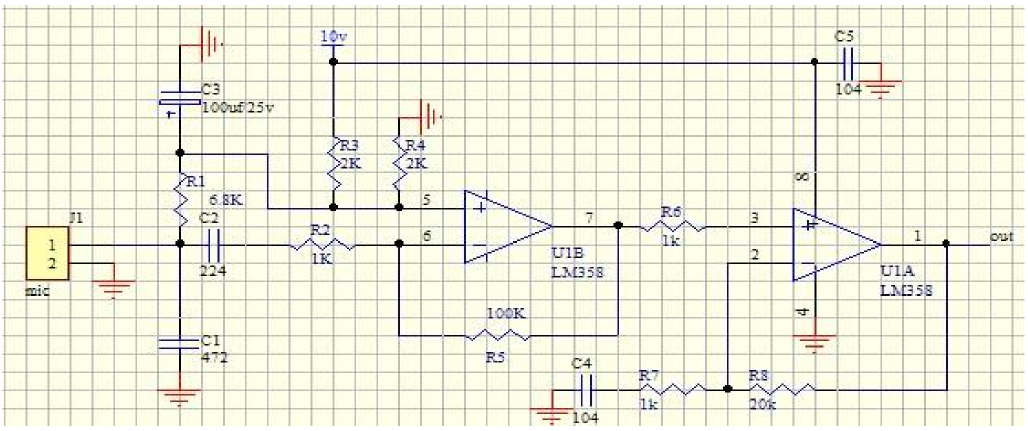


Figure 2 - Microphone unit circuit

Below is a circuit diagram of the Microphone unit. For noise filtering, C2 works as a high pass filter and C1 works as a low pass filter. For amplification, we used two-stage amplification to achieve a desired voltage gain. The total gain is controlled by two-sets of resistors (i.e. R2&R5, R7&R8).

After testing, we decide to increase the gain to 135 from our proposed value (40), in order to make the output of microphone large enough to be analyzed by MCU. Thus we choose the following resistor values: R5 = 30K, R2 = 1K, R7 = 1K, R8 = 150K. The total gain measured is roughly 135.

As the Arduino analog pins only allow 0-5V input, we later add a 5V-diode between microphone-unit output and ground to limit the voltage in below 5V.

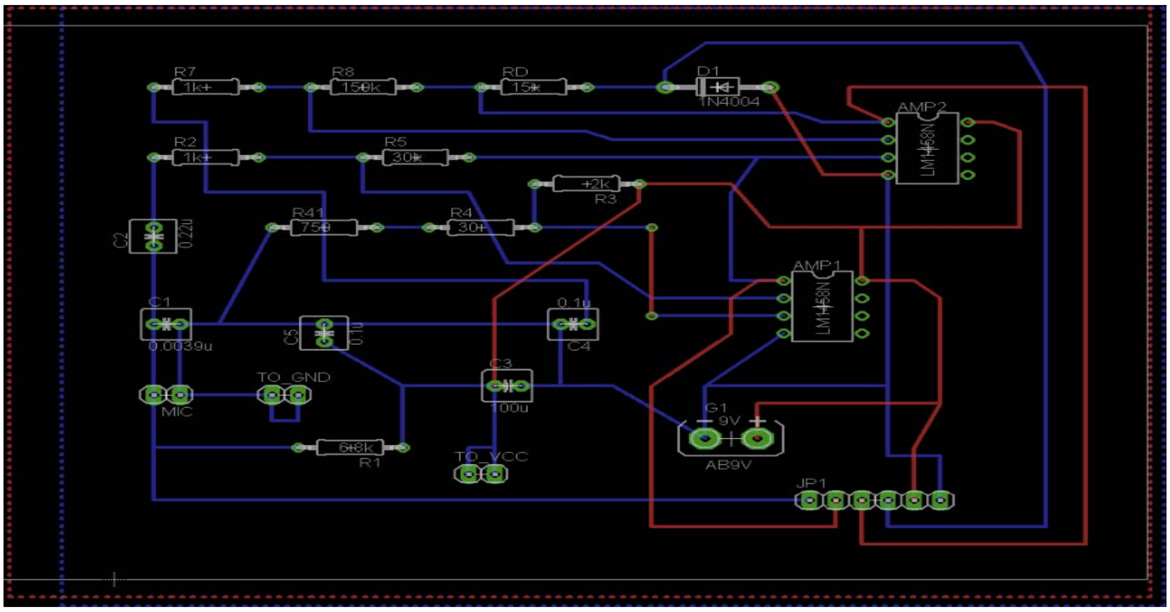


Figure 3 – PCB layout

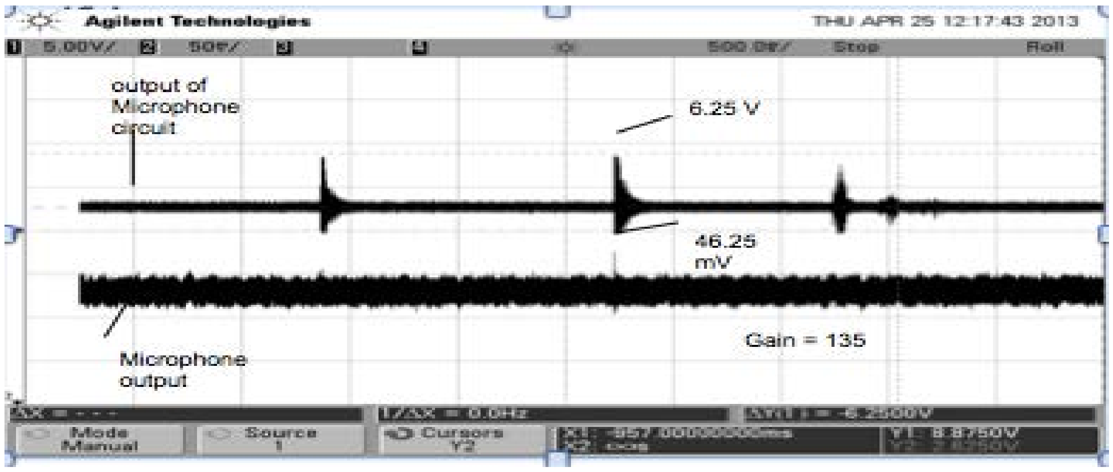


Figure 4 – Amplification

In figure 4, the upper waveform represents the output of microphone circuit and the below one is the output of the microphone. Comparing the peak to peak voltage of microphone output and amplified output, we can observe a gain of around 135.

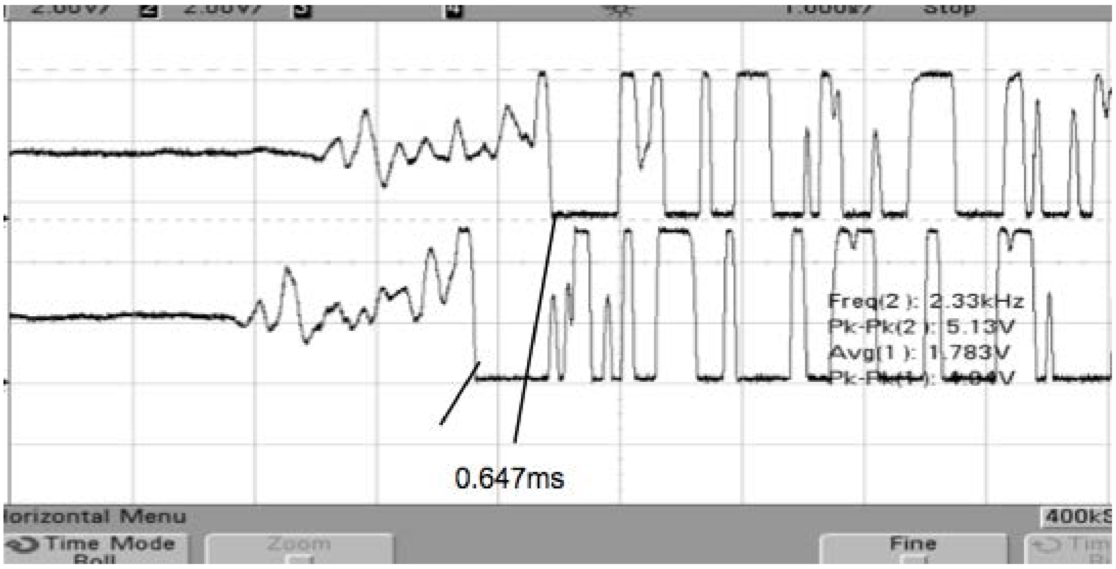


Figure 5 – Maximum Delay

Since the direction determination done is by comparing the signal delay between microphone pairs, we want to make sure the circuitry will not introduce a large phase shift to the output signal. We compare the circuit output of two microphones which are 22cm apart, and we find the maximum delay equals to 0.647. The delay, in principle, should equal 22cm/(342m/s)=0.64ms (distance/speed of sound), which is close to out testing result. Thus we conclude that the microphone unit circuit does not introduce a significant delay.

We can calculate the total power of microphone unit by the equation P = 3V*I=3 *9v*0.04A=1.08 W. Actually, the 9v battery is adequate. Our testing result shows that he microphone unit can work for at least 2 hours with one 9V battery power supply.

2) MCU (Arduino ATMega328)

This module is the central control unit of the system. It samples the microphone output at 17000Hz, estimates the voice source direction, decides camera directions and instructs motor to rotate. Also, it communicates with the C++ client program to generate split screen if two speakers are detected.

A learning algorithm will be running on the MCU to determine the main speakers (for this project, up to 2 main speaker will be located). The Calculation section describes the detailed voice source detection algorithm which runs on the MCU

The board we use is the Arduino ATMega328. The Arduino ATMega328 board has 6 analog inputs, which are sufficient for our implementation (we need only 3 of them). We can use the AnalogRead() function to perform analog to digital conversion. We were able to adjust the sampling frequency up to 20000Hz by setting the pre-scales.

The actually sampling frequency we use is 17000Hz which is fast enough to yield a good result in voice source detection calculation. An output of the AnalogRead() at 17000Hz is shown below (the waveform is generated using Processing) :

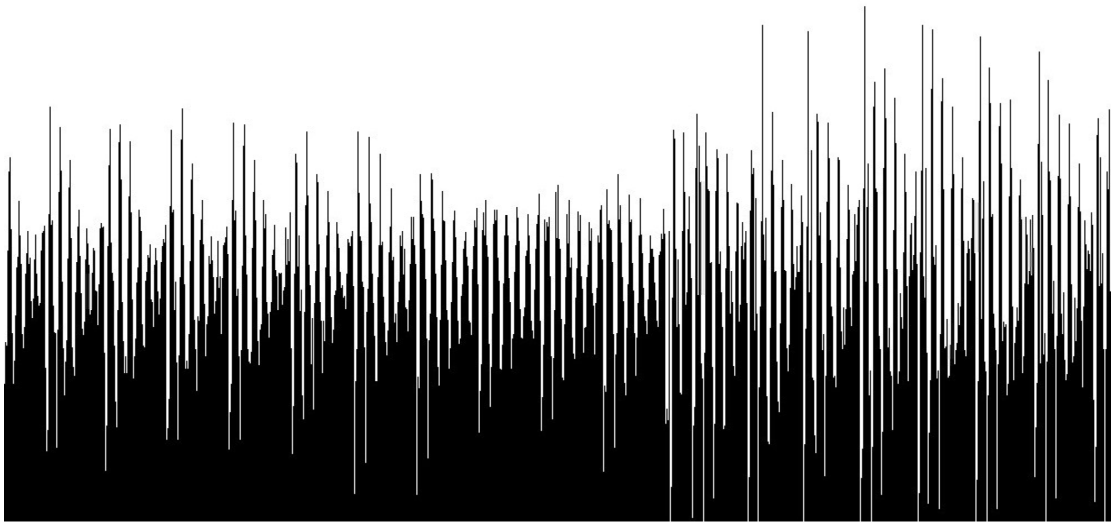


Figure 6 – sampling at 17000Hz

4 of the digital pin of the Arduino are connected to the input pins of Motor driver as shown below. When the program decides one of the camera need to rotate to a new direction, the Arduino will send pulses to the motor driver to trigger the motor rotation.

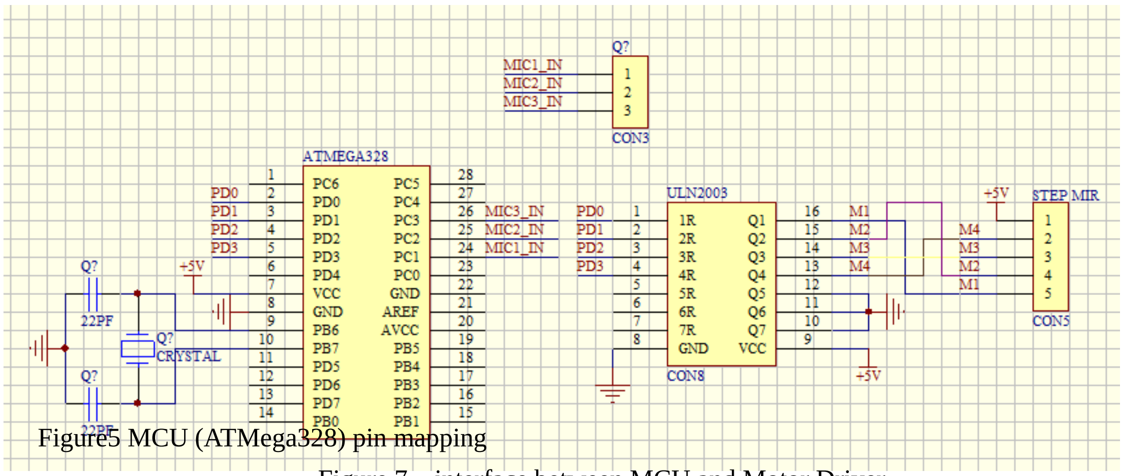


Figure5 MCU (ATMega328) pin mapping

Figure 7 – interface between MCU and Motor Driver

3) Motor Driver Unit

This module receives data from the MCU module and controls the two motors to move to a certain direction so that the camera would be directed at the speaker.

We use the chip ULN2003 to drive the motor. It gets pulse inputs from the Micro Controller and amplifies the signals. It will output voltages high enough to drive the stepper motor. The pull-up resistors are used to ensure the inputs to the chip settle at expected logic levels when high-impedance from the Micro Controller is introduced.

4) Stepper Motor Unit:

This module would be controlled by the Motor-control Unit. Both motors can rotate 360 degrees and in both clockwise and counterclockwise as needed. They will individually carry a camera (Camera 1 and Camera 2) to direct in an appropriate direction to record the speaker.

We choose the stepper motor 28BYJ48 that is 5 lines and 4 phases. It is a gear motor of 1/64. The stepper motor will get inputs from the ULN2003 chip and rotate accordingly. The two motors are controlled independently from two Motor Drivers. The two motors will also individually carry a camera and control its rotational movement in order to track the voice source. The motors can rotate clockwise or counterclockwise according to the input pulses as specified below.

Line	1	2	3	4	5	6	7	8
5	+	+	+	+	+	+	+	+
4	-	-						-
3		-	-	-				
2				-	-	-		
1						-	-	-

If the pulse goes as 1 2 3 4 5 6 7 8, it will rotate in one direction; if it goes like 8 7 6 5 4 3 2 1, it will rotate in the opposite direction. We would need this feature because we do not allow the motor to rotate continuously in one direction. In that way, the USB cables for the camera would intertwine together and limit the further rotation of the motor. We have 24 directions, each labeled as 0, 1, … 23. For example, if we want to rotate from 23 to 0, we could not make it rotate directly from 23 to 0. Instead, we would need it to rotate backwards, namely 23, 22, 21, … 0.

The circuit diagram and pin specification are as follows. (Both for the motor driver unit and motor unit)

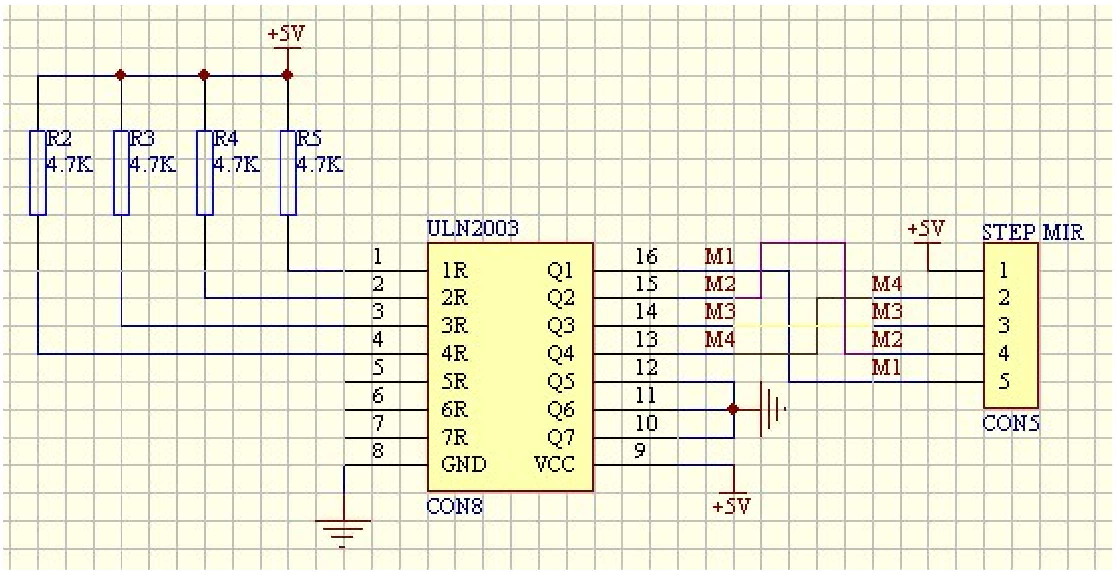


Figure8 - Motor Controller pin mapping

5) Camera 1/Camera 2

We use Microsoft LifeCam VX-2000 for the video recording. The webcam has a 55 degree field of view, and provides a 640x480 pixel resolution. The device is small enough to be mounted on the rotational rod. The data is transferred to computer through USB connection, and the device is powered by USB port.

6) C++ client program

The client program written in C++ monitors user inputs and implements the split screen function. It first establishes connection with Arduino program by opening the serial port. Then, user can input “go”, “halt” and “end”. The client program will instruct the Arduino program to start, halt or end correspondingly. If the Arduino program decides a split screen should be produced, it will send a special message to client program. The client program will turn the 2nd camera on and generate split screen after receiving the message. Below is a screen shot of the split screen.

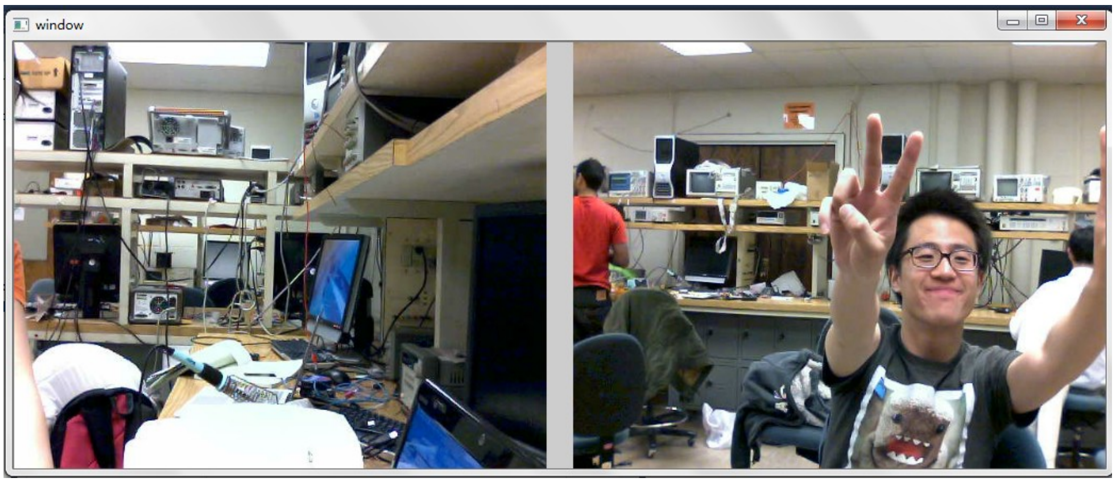


Figure 9 – split screen

7) Power Supply

This module handles the power supply of the Microphone Array module, the MCU module, the Motor-control Unit module, the motors and the cameras. The microphone circuit would need 9V DC. The Aduino Uno, ULN 2003 chip, and the stepper motors will all be operated at 5V DC.

We use 9V battery for our system. The battery could be directly used in our microphone circuit. In order to keep the Vcc the same for all three circuits, we use only one battery and connect those three microphone circuits in parallel with the batter. For the motor unit, we use a regulator to convert the 9V down to 5V.

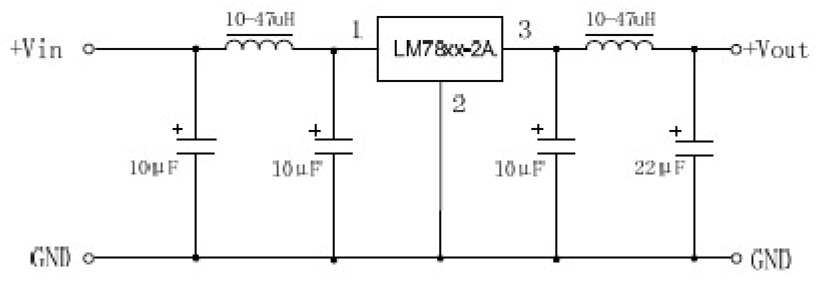


Figure 10- voltage regulator

8) Mechanical Part

For the mechanical part, we have three microphones on three vertices of an equilateral triangle. The distance between each microphone is 22 cm. To independent motors are controlling the rotation of two cameras.

2.3 Voice source detection algorithm

a. Microphone array geometry

The geometry we use is an equilateral triangle with 1 microphone on each vertex. For each pair of microphones, the distance between them is 30cm.

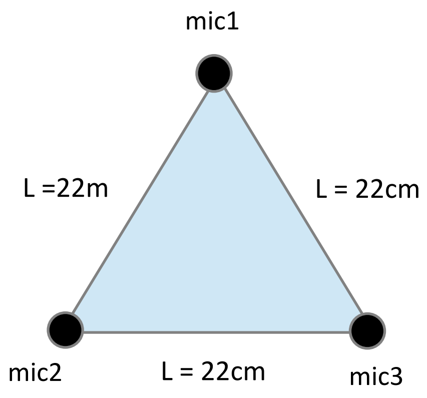
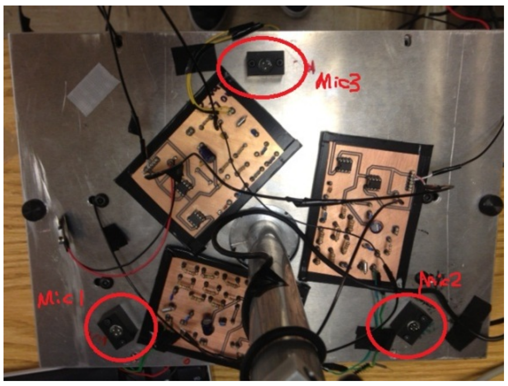
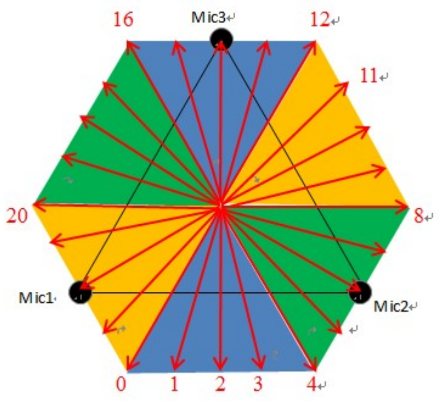


Figure8. Microphone array geometry





1). Basic Voice Direction Detection Algorithm

We adopted the voice source detection algorithm from Knudsen & Lessans' project paper [1]. The 2D-space is equally divided into 6 segments (figure 5.1), each with 60 degree angle centering at the midpoint of the triangular.

Let sig1, sig2, sig3 denote the voice signals received at mic1, mic2 and mic3 respectively.

An example will illustrate how to roughly determine the target angle of camera rotation: Say sig1 and sig2 has smallest phase difference, the voice source is in the blue region in the graph since the blue region is most perpendicular to the line connecting mic1 and mic2. Then, say sig3 lags behind both sig1 and sig2, and the source must be in the top blue section. If sig1 leads sig2, then the source is in the top-left blue segment; otherwise, the source is in the top-right blue segment. The maximum error is limited to 30-degree in this case. For detailed analysis of the algorithm, please reference the Knudsen & Lessans' paper.

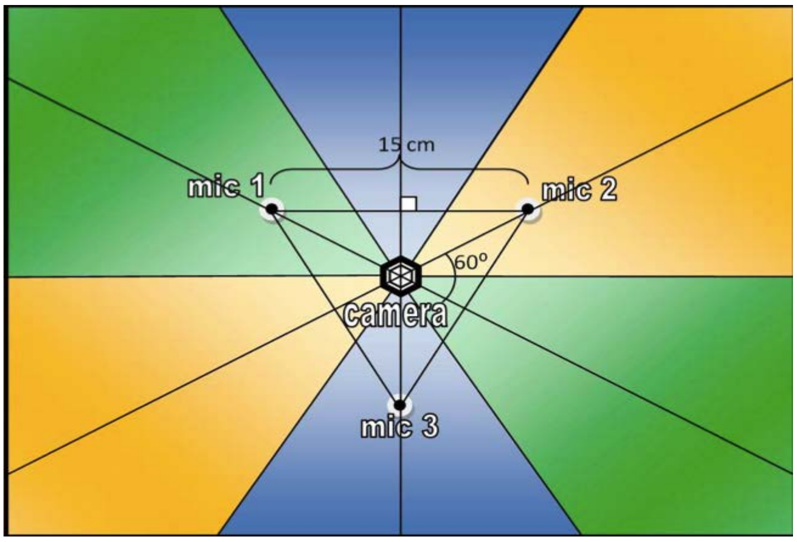


Figure11. Separation of space [2]

2) Further modification to improve resolution

For simplicity, we decided to use discrete values for camera targeting direction. Based on the value of phase difference, we can manually divide each of the 6 segments into even smaller sub-segments. In the below figure, all 24 red arrows indicates all 24 possible camera target direction. We choose to use discrete values for target direction based on the following observation:

1. easy to implement motor control unit using a step motor.
2. easy to implement 2 source tracking (will discuss later)
3. Most web cam would give at least 30 degree of view. A resolution of 24 is sufficient to ensure the quality of the video (i.e. speaker is in the camera frame, but not necessarily centered).

How to divide each region into smaller sub-regions will be discussed later when we introduce the cross correlation and "m value".

``

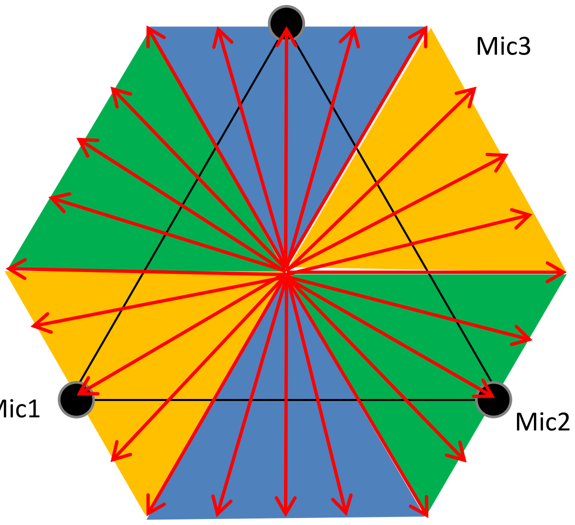


Figure12. Discrete targeting angles

3) Signal delay calculation

The relative delay between each pair of mic-signal is estimated using cross correlation. Let $S_1(n)$ and $S_2(n-m)$ denotes the sample of sig1 and shifted sig2 (shift to the left by m samples) at some sample frequency f.

$$\boldsymbol{C_{1,2}^{m} = \frac{1}{N}\sum_{n=0}^{N-1} S_1(n) * S_2(n-m)}$$

C value indicates the similarity between the two waves. In this case, a large C value for some m indicates a high similarity between sig1 and sig2 shifted by m. Thus, we can estimate the delay between sig1 and sig2 by finding the m which produces largest C. The range of possible m can be calculated as follow:

$L = 0.22$m, $V_{sound} = 342$m/s

p = 0.22/342 = 0.64ms – maximum propagation delay between a pair of microphone

f = 17000Hz, t = 1/f = 0.059ms

m_{max} = p / t = 0.64/0.0059=12 samples ->maximum delay

The maximum delay between a pair of microphone signals are 12 samples (can be either leading or lagging). Thus m∈[-12, 12], with a sample frequency of 17000Hz.

The flow chart below provide more precise information about the delay estimation

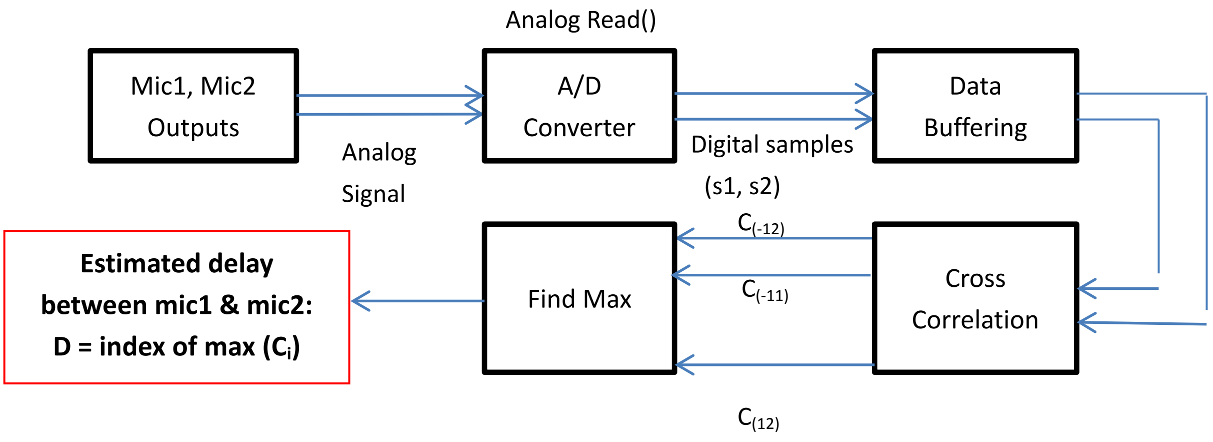


Figure13. Data flow to determine phase difference

The audio signal will be sampling at 17000Hz, and uses every 190 samples as one group.

If we plot the C value against different m values for a mic pair, a single peak would normally suggest a single voice source, or 2 sources but close enough so that they merges into one peak. A threshold is applied to the C value so that the peak produced by background noise when no people are speaking will not be considered as voice sources.

The threshold value needs to be determined and verified by manually testing in the future to insure the background noise to be discarded.

Multiple peaks on the C-m graph suggest multiple speakers speaking within the same time interval. We realize it is hardly possible to separate these voice sources only using the current available information, thus only the peak with largest C-value (and is above threshold) will be accepted into the voice source calculation.

We will get 3 phase differences for 3 microphone pairs (i.e. M_{12}, M_{23}, and M_{31}). Then we can use Knudsen & Lessans' algorithm to roughly locate the voice source in a colored region. Again assume the source is in top blue region of figure1.3. Then we look at the value of M_{12}:

M_{12}	Camera Direction (figure1.5)
-5, -4	A
-3, -2	B
0, -1, 1	C
2, 3	D
4, 5	E

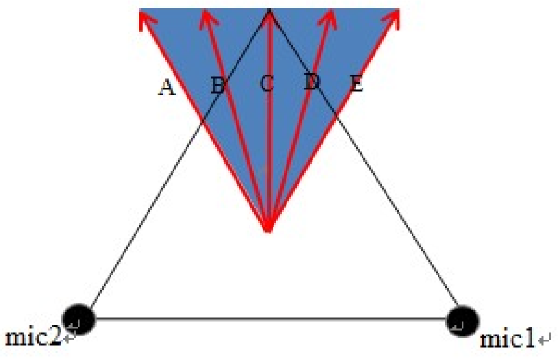


Figure14.

Note: The accuracy of this estimation need to be verified in the future testing.

4) Two speakers tracking:

The idea is to separate different speakers sitting at different directions and find how active each speaker is during the on-going conversation; select the two most active speakers if there are any, and face the camera towards them.

As discussed above, we split the 360 degree into 24 discrete targeting angles, and index them through 1 to 24. An array (let's call it Y-heap) of 24 elements will be initialized to all 0s.
For each 0.5 seconds, the voice detection algorithm will produce a number from 1 to 24 that indicates the current speaker (or 0 if silence). The corresponding array element will increase its value using the forgetting function:

$Y_{new} = a*Y_{old} + (1-a)*C$ Where C is a constant

All other element will be updated using:

$Y_{new} = a*Y_{old}$

The weight of old record will exponentially decrease. The 'a' value needs to be carefully picked so that the Y value can roughly reflect the level of participation of each direction in the on-going conversation.
Each time, we can select the 2 elements with largest Y value (which also need to be greater than some threshold) and have the 2 cameras rotated to the corresponding angles.

State Diagram for MCU

Idx_m1: the index with largest Y

Idx_m2: the index with second large Y

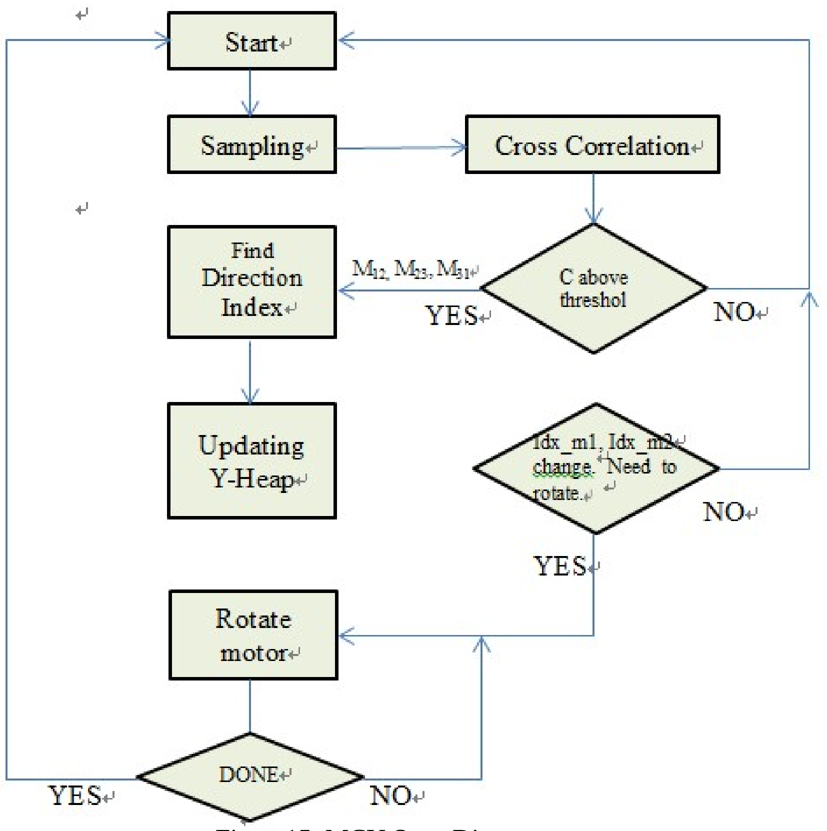


Figure15. MCU State Diagram

3. Requirements and Verification

3.1 Testing procedures

Microphone Unit

Requirements	Verification
(1) The microphones are able to accurately cross-correlate human voice while ignoring the background noise. a. Microphone test with supplied with 1.5V can pick up voltage signal around ± 50 mV when speaker is at 10 feet away. b. The 3dB cut off frequency of high pass filter should be 100Hz and the cut off frequency of low pass filter is close to 3KHZ.	(1) a. Connect output of a microphone to oscilloscope. The waveform of the oscilloscope should be strong and clear when the microphone is at 10 feet away from a speaker. (passed) b. Connect the output of function generate (100Hz or 3KHZ 5V) to the input of the circuit, then observe the amplitude of waveform on oscilloscope showed the output of the amplifier circuit should be close to $5\times 0.707 = 3.537$V. (passed)
(2) The output (amplified) waveform should be clear and strong when the speaker is10 feet away from microphones. Amplifier is able to expand 50 mV to 2 V. a. The gain of amplifier is around 40.	(2) Connect the output of function generate (50 mv) to the input of the above amplifier circuit., then Observe the amplitude of waveform on oscilloscope showed the output of the amplifier circuit should be 2v. (passed)
(3) The range output voltage of microphone circuit is required from 0 to 5V.	(3) Add a 2.5 DC source in series, then obtain the range of voltage waveform on oscilloscope is from 0 to 5V. (passed)
(4) When the distance between two microphones is 30cm, the phase delay close to 0.88ms ($\pm$0.5ms) (.30m/340ms^{-1}).	(4) Connect the 2 outputs to the 2 channels of the oscilloscope and read the delay of the 2 waveforms. (passed)

Voice Source Detection / MCU

Requirement	Verification
(1) The data sample rate should be close to 17000Hz. So all the calculation holds. 2) The C peaks produced by the background noise should be discarded. 3) When multiple C peaks appear in C-m plot, the frame (data within 0.5s interval) should be	1) Count the number of samples over some time interval and find average sampling frequency. The frequency should be roughly around 17000Hz (+-200hz). (passed) 2) Generate some background noise; Find the minimum noise level that creates a peak above the threshold. The min noise level should be

discarded. 4) The direction index calculated should be accurate enough so that the speaker can be captured in frame. 5) The forgetting coefficient 'a' should be properly chosen so that neither the camera frequently change target nor the camera seldom change target. 6) Each digital output pin of the MCU can give a digital pulse of magnitude 5$\pm$0.5V that will be used drive the motor in both directions. The pulse width will be 5ms ($\pm 1ms$) 7) MCU can generate sequence of pulses as described in Chart1. The length of each sequence should be at most 50ms.	considerably large (30 - 35dB). We can use speakers to generate this noise and by adjusting the sound level to modify noise level. (passed) 3) Print out the C-m plot, and mark which frame was used and all frames should only have 1 peak that is above the threshold. (passed) 4) Create some random M values, and check whether the algorithm produces correct direction indices based on each input set. (passed) 5) The coefficient value determines how reactive the systems will be. We will test out a range of values (range from 0 to 1) and see which one serves us the best. (passed) 6) Hook up the output of MCU to an oscilloscope and measure the width and magnitude of the pulse. The test should be done on all 4 output pins. (passed) 7) Connect 4 output pins to 4 LEDs. Issue 100 rotate instructions, and observe the pulse sequence on LEDs. Should see LEDs flashing with the correct pattern described in Chart1. And the LEDs should flash no longer than 5 seconds. Test should be done for both clockwise and counter-clockwise rotation. (passed).

Motor Driver Unit:

Requirement	Verification
(1) It can control the motor to rotate by a certain number of steps. (2) It can control the motor to rotate both clockwise and counterclockwise. (3) When it takes input from MCU, it should output current high enough to drive the motor. Specifically, it should output current at 500 mA (+/-30 mA).	(1) Write some codes to generate input for the driver. Connect the MCU output to the four inputs of the pin 1, 2, 3, 4 of the motor driver. If it goes through 64*8 cycles, the motor should experience exactly one revolution. (passed) (2) Set the codes to go two directions of the cycle described above and see whether the motor rotate in opposite direction. (passed) (3) Connect the MCU to the chip. Use the millimeter to measure the current output to make sure it stays within a certain range.

	(passed)

Stepper Motor Unit:

Requirement	Verification
(1) Motor should rotate exactly the steps it is instructed to by the motor driver. Specifically, to meet our design requirements, we want the motor to rotate all multiples of 15 degrees with zero error tolerance. (2) Motor should rotate in the direction as it is instructed to.	(1) Write code to the MCU to instruct the motor to rotate all multiples of 15 degrees, namely, 15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180, 195, 210, 225, 240, 255, 270, 285, 300, 315, 330, 345, 360. (passed) (4) Write the code to instruct the motor to rotate clockwise or counterclockwise. See if it rotates in the direction correctly. See detailed instruction to make the motor rotate in both directions in the section of stepper motor unit. (passed)

3.2 Tolerance Analyses

The tolerance analysis mainly focuses on:

-The error of voice source detection caused by the geometry of the microphone array and the factor of noise. The goal is to have the difference between actual source and calculated source to be within 5% (i.e. within 20 degrees), so that the camera will still be able capture the speaker in the frame if not centered in the frame. We will test the tolerance by setting up all possible scenarios:

1. Only one speaker
2. Two speakers talking simultaneously / separately.
2. Two speakers sitting close to each other / away from each other.
3. Two speakers with some noise (machine noise or noise from other people).
4. Multiple speakers.

-The error of camera rotation:

We will analyze the cause of error of camera rotation if there is any. Again, our goal is to limit the error within 5% (less than 20 degrees) for the same reason stated above. We will measure the tolerance by feeding the motor control with pre-calculated values so we know where the camera should be pointing at after the rotation.

3.3 Ethical Issues

3) To be honest and realistic in stating claims or estimates based on available data.

We will be true and honest to ourselves and present results and data of our experiment as they are originally without any modification or cheating. We will also ensure to use our own result and don’t plagiarize from others.

5) To improve the understanding of technology, its appropriate application, and potential consequences.

We will explore the most versatile possible version of our product and try to optimize its performance to serve the customers most.

7) To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others.

We will acknowledge the portion that we take reference from others. And we will be responsible about the possible errors we make.

3.4 Safety

Our product contains potentially harmful voltage levels. So we will carefully isolate the inner circuits by shielding them with proper shells. Since this product will always be used inside the room, we do not need to consider issues such as rain and extreme temperature changes.

When building our product, we should also be aware of the lab safety procedures and avoid doing anything that we are not sure about its safety issues. Testing and debugging should also call for carefulness in terms of the possible high voltage source we will use.

4 Costs and Schedule

4.1 Cost Analysis:

Parts name	Quantity	Unit price ($)	Total price ($)
Arduino Uno	1	30	30
Microsoft Webcam VX2000	2	40	80
USB cables	3	5	15
9V Battery	4	2	8
9V Battery clip	4	0.5	2
PCB	3	15	45
Vector Board	2	3	6
motor driver (ULN2003)	2	0.5	1
Stepper Motor (28BYJ48-5V)	2	5	10
Microphone (MIC2343)	3	0.5	1.5
Metal Station	1	30	30
Resistors / Capacitors	50	0.1	5
Grand Total		$233.5	

Labor

Name	Hour Rate	Total Hours Invested	Total
Qi Yang	$35.00	150	$13,125
Yuxiao Lu	$35.00	150	$13,125
Jiehan Yao	$35.00	150	$13,125

Total		450	$39,375

Grand Total

Section	Total
Labor	$39,375
Parts	$233.5
Total	$39608.5

5. Successes and Challenges

Our project has been able to achieve all the proposed functions and features. Our camera could be directed to 24 different directions equally spaced around a circle. When two people are speaking, two cameras would be turned on, each directing towards two speakers, and split-screen can be correctly generated.

However, it should be admitted that our system has not been perfect. For one thing, we have not been able to decide a general correlation threshold for all speakers with voice of different loudness and frequency or a better way to get around this problem. For another thing, the distance between the speaker and the station would affect our decision making about the direction. We would also need to work on this problem. Besides, another problem existing is that the effect of reflection of sound should be taken into account so that it would not generate unnecessary camera rotation.

6. Future Work

We have achieved all of our design goals, but improvement is still needed.

1. Rely less on the specific distance for direction determination.
   Use higher sensitivity microphones instead of MIC2343.
2. More accurate direction determination
   Use larger storage MCU. Since the sampling points are related to memory, we can increase the memory of MCU to increase the sampling points for accurate direction determination
3. More long-lasting and robust power supply
   Use wall jack instead of 9v battery
4. Integrate all the PCBs into box
   Make a larger size box to put all stuff into the box.

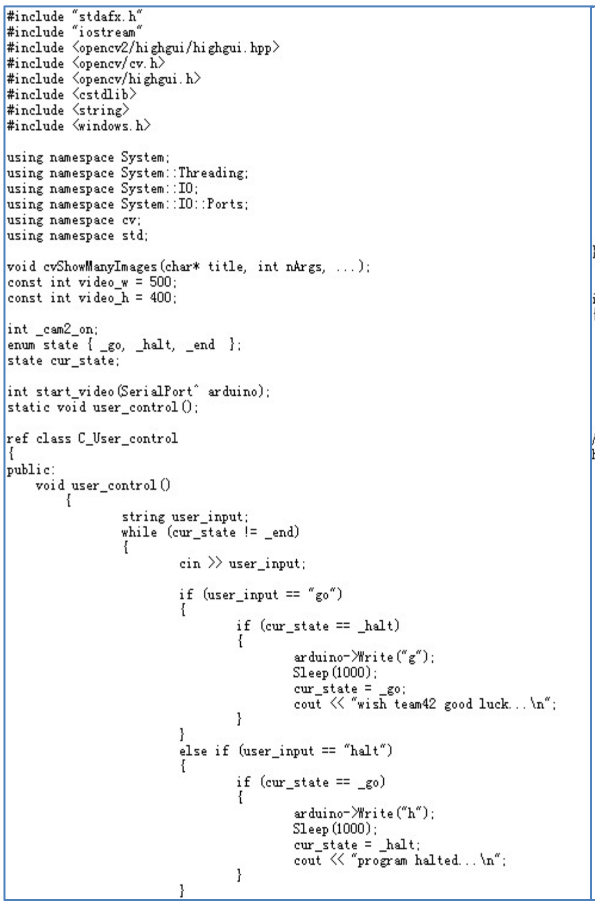
7. Reference

[1][2] Voice Tracking camera-Video conference of the future, Knudsen, Lessans, and Scholl, http://ese.wustl.edu/ContentFiles/Research/UndergraduateResearch/CompletedProjects/WebPages/sp11/JasonZachMichael/ESE%20498%20WriteUp%20Zach,%20Jason,%20Michael.pdf.

[3] Image from the datasheet of LM78xx-2A

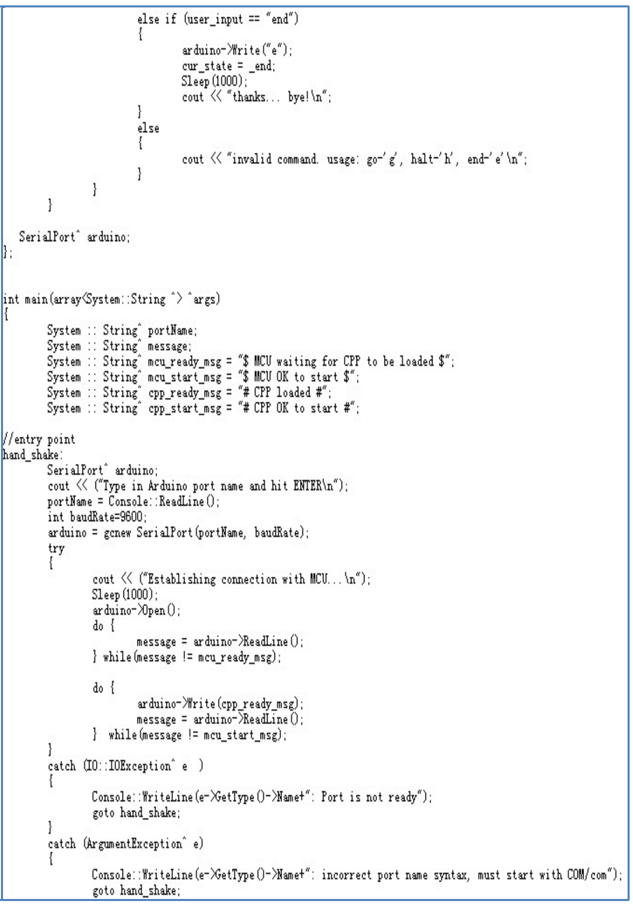
8. Appendix - C++ client program source code

1.



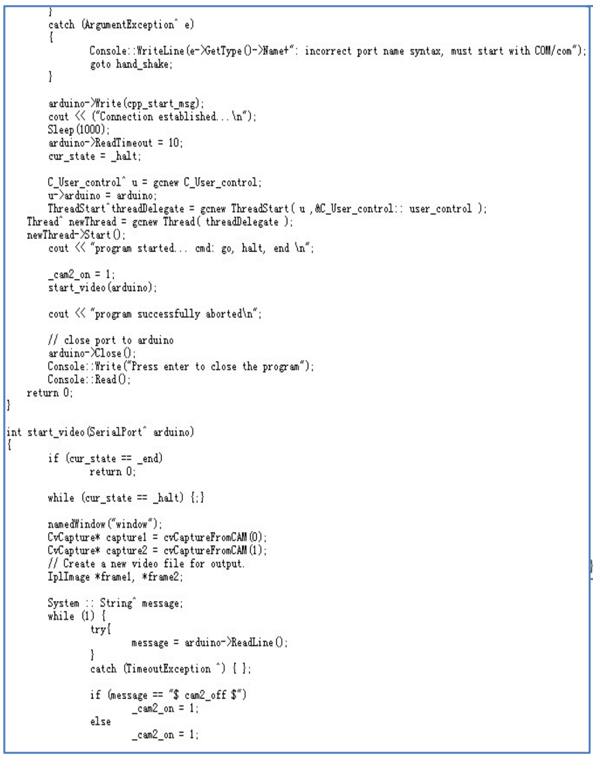
```
#include “stdafx.h”
#include “iostream”
#include <opencv2/highgui/highgui.hpp>
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <cstdlib>
#include <string>
#include <windows.h>

using namespace System;
using namespace System::Threading;
using namespace System::IO;
using namespace System::IO::Ports;
using namespace cv;
using namespace std;

void cvShowManyImages(char* title, int nArgs, ...);
const int video_w = 500;
const int video_h = 400;

int _cam2_on;
enum state { _go, _halt, _end };
state cur_state;

int start_video(SerialPort^ arduino);
static void user_control();

ref class C_User_control
{
public:
    void user_control()
        {
                string user_input;
                while (cur_state != _end)
                {
                        cin >> user_input;

                        if (user_input == “go”)
                        {
                                if (cur_state == _halt)
                                {
                                        arduino->Write(“g”);
                                        Sleep(1000);
                                        cur_state = _go;
                                        cout << “wish team42 good luck...\n”;
                                }
                        }
                        else if (user_input == “halt”)
                        {
                                if (cur_state == _go)
                                {
                                        arduino->Write(“h”);
                                        Sleep(1000);
                                        cur_state = _halt;
                                        cout << “program halted...\n”;
                                }
                        }
```

2.



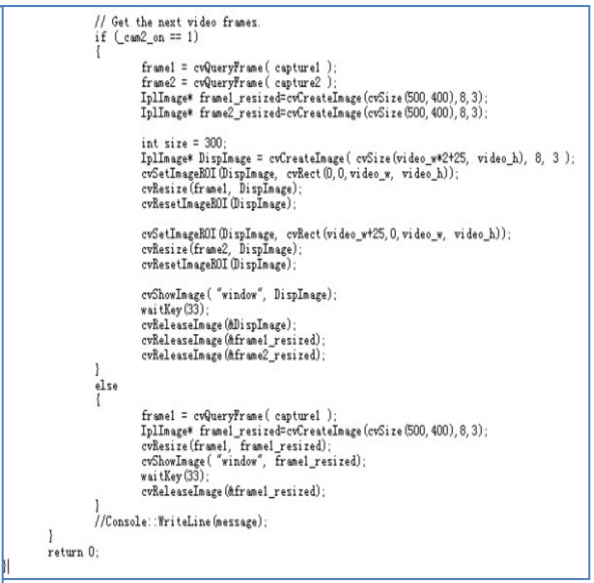
```
                        else if (user_input == “end”)
                        {
                                arduino->Write(“e”);
                                cur_state = _end;
                                Sleep(1000);
                                cout << “thanks... bye!\n”;
                        }
                        else
                        {
                                cout << “invalid command. usage: go-‘g’, halt-‘h’, end-‘e’\n”;
                        }
                }
        }

   SerialPort^ arduino;
};


int main(array<System::String ^> ^args)
{
        System :: String^ portName;
        System :: String^ message;
        System :: String^ mcu_ready_msg = “$ MCU waiting for CPP to be loaded $”;
        System :: String^ mcu_start_msg = “$ MCU OK to start $”;
        System :: String^ cpp_ready_msg = “# CPP loaded #”;
        System :: String^ cpp_start_msg = “# CPP OK to start #”;

//entry point
hand_shake:
        SerialPort^ arduino;
        cout << (“Type in Arduino port name and hit ENTER\n”);
        portName = Console::ReadLine();
        int baudRate=9600;
        arduino = gcnew SerialPort(portName, baudRate);
        try
        {
                cout << (“Establishing connection with MCU...\n”);
                Sleep(1000);
                arduino->Open();
                do {
                        message = arduino->ReadLine();
                } while(message != mcu_ready_msg);

                do {
                        arduino->Write(cpp_ready_msg);
                        message = arduino->ReadLine();
                }  while(message != mcu_start_msg);
        }
        catch (IO::IOException^ e  )
        {
                Console::WriteLine(e->GetType()->Name+”: Port is not ready”);
                goto hand_shake;
        }
        catch (ArgumentException^ e)
        {
                Console::WriteLine(e->GetType()->Name+”: incorrect port name syntax, must start with COM/com”);
                goto hand_shake;
```

3.



```
        }
        catch (ArgumentException^ e)
        {
                Console::WriteLine(e->GetType()->Name+”: incorrect port name syntax, must start with COM/com”);
                goto hand_shake;
        }

        arduino->Write(cpp_start_msg);
        cout << (“Connection established...\n”);
        Sleep(1000);
        arduino->ReadTimeout = 10;
        cur_state = _halt;

        C_User_control^ u = gcnew C_User_control;
        u->arduino = arduino;
        ThreadStart^threadDelegate = gcnew ThreadStart( u ,&C_User_control:: user_control );
    Thread^ newThread = gcnew Thread( threadDelegate );
    newThread->Start();
        cout << “program started... cmd: go, halt, end \n”;

        _cam2_on = 1;
        start_video(arduino);

        cout << “program successfully aborted\n”;

        // close port to arduino
        arduino->Close();
        Console::Write(“Press enter to close the program”);
        Console::Read();
    return 0;
}

int start_video(SerialPort^ arduino)
{
        if (cur_state == _end)
                return 0;

        while (cur_state == _halt) {;}

        namedWindow(“window”);
        CvCapture* capture1 = cvCaptureFromCAM(0);
        CvCapture* capture2 = cvCaptureFromCAM(1);
        // Create a new video file for output.
        IplImage *frame1, *frame2;

        System :: String^ message;
        while (1) {
                try{
                        message = arduino->ReadLine();
                }
                catch (TimeoutException ^) { };

                if (message == “$ cam2_off $”)
                        _cam2_on = 1;
                else
                        _cam2_on = 1;
```

4.



```
                // Get the next video frames.
                if (_cam2_on == 1)
                {
                        frame1 = cvQueryFrame( capture1 );
                        frame2 = cvQueryFrame( capture2 );
                        IplImage* frame1_resized=cvCreateImage(cvSize(500,400),8,3);
                        IplImage* frame2_resized=cvCreateImage(cvSize(500,400),8,3);

                        int size = 300;
                        IplImage* DispImage = cvCreateImage( cvSize(video_w*2+25, video_h), 8, 3 );
                        cvSetImageROI(DispImage, cvRect(0,0,video_w, video_h));
                        cvResize(frame1, DispImage);
                        cvResetImageROI(DispImage);

                        cvSetImageROI(DispImage, cvRect(video_w+25,0,video_w, video_h));
                        cvResize(frame2, DispImage);
                        cvResetImageROI(DispImage);

                        cvShowImage( “window”, DispImage);
                        waitKey(33);
                        cvReleaseImage(&DispImage);
                        cvReleaseImage(&frame1_resized);
                        cvReleaseImage(&frame2_resized);
                }
                else
                {
                        frame1 = cvQueryFrame( capture1 );
                        IplImage* frame1_resized=cvCreateImage(cvSize(500,400),8,3);
                        cvResize(frame1, frame1_resized);
                        cvShowImage( “window”, frame1_resized);
                        waitKey(33);
                        cvReleaseImage(&frame1_resized);
                }
                //Console::WriteLine(message);
        }
        return 0;
}
```