**ABSTRACT**

**BOT CONTROL USING HUMAN INTERACTION**

The aim of our project is to enable bot control using human interaction techniques such as speech recognition and hand gestures (if time permits). Our project is divided over a 12 week period with 4 phases.

The first phase will consist of enabling microphone access and recording our audio for the training of the speech recognition system. The audio files have to be stored in a database.

The second phase will consist of extracting required features of the audio to enable recognition and compare with that present in the database.

The third phase will consist of finding and implementing an algorithm to do the comparison and getting a final score which can be used to identify the command. We will also compile the entire code into 1 C++ file and do offline tests.

The fourth phase will consist of enabling serial communication with the bot. Further tests including increasing the database and correcting any errors will also be done in this phase.

Optional fifth phase will consist of controlling the movement of the bot using hand gestures utilizing an image processing library in open CV.

**Block Diagram –**

Any Recognition process includes the following steps –

1. Training Phase -

Segmentation of Region of Interest (ROI)

Eg. Face, Hand, Speech

Create Training Database

Preoprocess ROI

Extract Features from ROI for Classification

1. Testing/Implementation Phase -

Segmentation of Region of Interest (ROI)

Eg. Face, Hand, Speech

Compare with Training Database

Extract Features from ROI for Testing

Preoprocess ROI

For Speech Recognition we have the following Block Diagram –

1. Training Phase -

Remove Frequency Components outside the vocal range and store it as a .WAV File

Record Sound from the Mic

Create Training Database of Features

Extract MFCC Features used for Classification

1. Testing/Implementation Phase -

Compare MFCC Features of the newly recorded sound with those in the database

Extract MFCC Features for Comparison

Remove Frequency Components outside the vocal range and store it as a .WAV File

Record Sound from the Mic

Send the word recognized to the Bot and Move it

Find the closest match using DTW Algorithm

**Background Knowledge -**

The background knowledge we have is basic C/C++ programming, basics of image processing in open CV, and motor control using AVR Atmega8/Aurdino Uno board. We will be grateful if you become our mentor and guide us through the course of this project. Our team consists of-

1. Tushar Dobhal-11EE93

2. Rachit Mundhra-11EE62

3. Aritro Banerjee-11EE106

4. Vivswan Shitole-11EE104

**Report 1**

-

1) Wave vs Mp3 -

Out of the available formats we chose the .WAV file.

|  |  |
| --- | --- |
| WAVE | MP3 |
|  |  |
| 1. It is uncompressed and hence file size is large. | 1. It is compressed and hence file size is smaller. |
| 2. It has a much larger spectrum and hence includes all frequencies. | 2. It has a smaller spectrum and contains only limited number of frequencies. |
| 3. WAV files are lossless and are used for TV, radio, DVD or any other media requiring top audio quality. | 3. MP3 files are lossy and contain limited information and are used in web pages, web videos and songs. |

2) Recording from the Mic -

We have used portaudio library for recording the sound from mic and sndfile c++ library for storing it in a wave file. A general algorithm for recording the sound is as follows:-

Start

Wait for Space Key

If c == ‘ ‘

No

Yes

Initialize Parameters  
 (No. of channels, seconds, Sampling frequency)

Open Mic for n seconds using PortAudio Lib.

False

If no error occurs

After n sec

Store the values in an array

Save the values as a .WAV File using LibSndfile Lib.

True

Stop

Recording sound from the mic is important for two things –

1. Creating the database of sound clips
2. Recording for comparison with those in the database

**Report 2**

* Out of Arduino and AVR Atmega8 microcontrollers, we have decided to use AVR Atmega8 microcontroller for controlling the motors of the bot because of simplicity and past experience in using the board.
* We have studied the basics of the Atmega8 board including the pins and the functions and have burnt a demo motor driver program in the board which will enable the bot to move forward, backward, left and right on its own.
* Out of the available features, we have decided to use the mel frequency cepstral coefficient (mfcc) features as a major feature extraction criteria. These mfcc features are widely used in speech and speaker recognition systems.
* We have started writing the code for mfcc features extraction.

**AVR board parts identification**

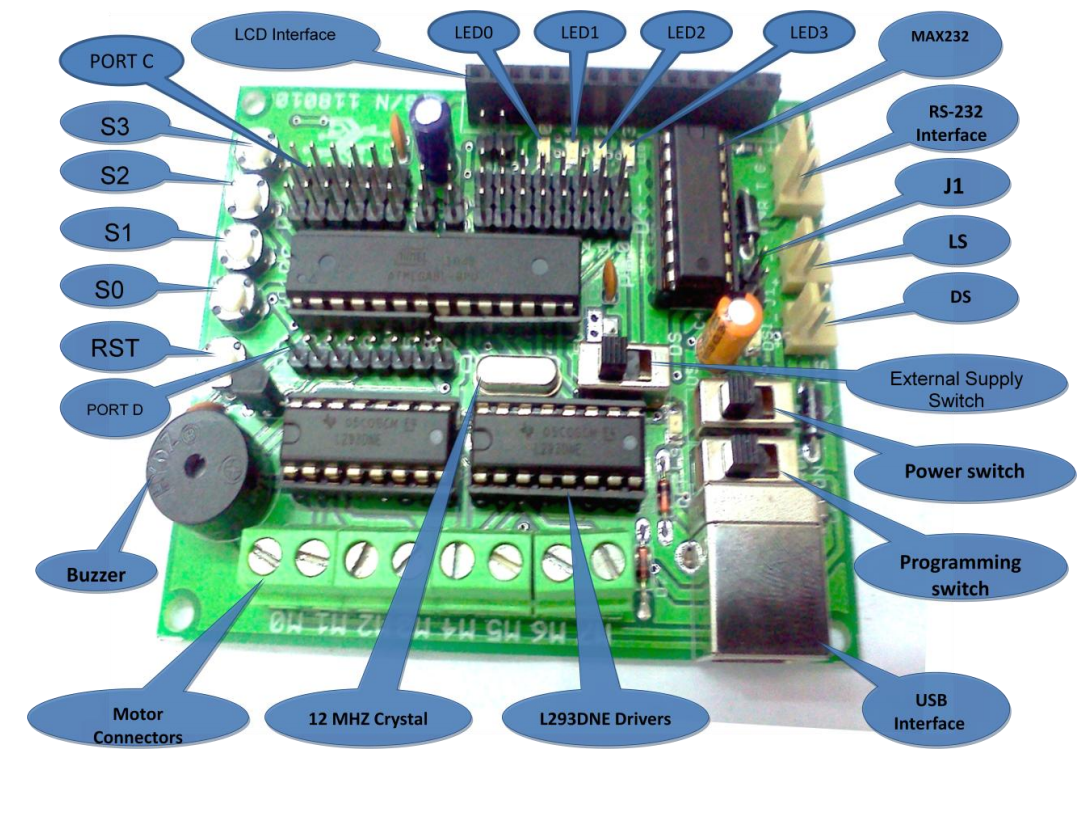


Fig. 1 AVR Atmega8 and its Components

The code is burnt in the AVR ATMEGA8 board as follows:

1. The batteries are connected to the logic supply for supply of voltage to the board and to the driving supply for the supply of voltage to the motor.
2. The PTOG(power toggle) switch is switched to the ‘ext’ mode so that the motor gets power from the external battery and not from the board.
3. The power switch is switched to USB and the PROG(program switch) is turned on to burn the code in the bot through the software HD Boot Flash.
4. After the code is burnt the power switch id again switched to logic supply(LS) mode.
5. The motor IC – L293DNE is controlled by the port b and port d as follows:

M0 - PortB0

M1 - PortB1

M2 - PortB2

M3 - PortB3

M4 - PortD4

M5 - PortD5

M6 - PortD6

M7 - PortD7

Since we use only two motors, we use code only pin b for driving the motor.

1. As per the connections of the motor the wires are connected in such a way that the input :

1010 – drives the motor forward

1001 – drives the motor left

0110 – drives the motor right

0000 – stops the motor

Following is the flow chart of the code burnt in the bot:

start

While(1)

The bot moves forward

Input in port b as 0x0a

2 second delay

The bot moves left

Input in port b as 0x09

2 second delay

Input in pin b as 0x06

The bot moves right

2 seconds delay

Input in pin b as 0x00

The bot stops

2 seconds delay

Report 3

* We have been able to implement serial communication with the bot. In serial communication mode, the bot moves according to the commands sent by the user in real time. In our case, when the user enters
* “w”, the bot moves forward
* “a”, the bot turns left
* “d”, the bot turns right
* “s”, the bot stops
* We found an algorithm (Dynamic Time Warping) for comparing two time series. Since speech is also a function of time, we propose that the algorithm can be used in our scenario as well. Wikipedia link-<http://en.wikipedia.org/wiki/Dynamic_time_warping>

Details of Serial Communication-

* RS232 is an asynchronous serial communication protocol widely used in computers and digital systems. It is called asynchronous because there is no separate synchronizing clock signal as there are in other serial protocols like SPI and I2C. In RS232 there are two data lines RX and TX. TX is the wire in which data is sent out to other device. RX is the line in which other device puts the data it needs to send to the device.
* As there is no "clock" line so for synchronization accurate timing is required so transmissions are carried out with **certain standard speeds**. The speeds are measured in bits per second. Number of bits transmitted is also known as **baud rate.**
* The data transmission is done in following ways
* When there is no transmission the TX line sits HIGH (-12V)
* When the device needs to send data it pulls the TX line low for 104uS (This is the start bit which is always 0)
* Then it send each bits with duration = 104uS
* Finally it sets TX lines to HIGH for at least 104uS (This is stop bits and is always 1). I said "at least" because after you send the stop bit you can either start new transmission by sending a start bit or you let the TX line remain HIGH till next transmission begin in this case the last bit is more than 104uS.
* The data reception is done in following ways
* The receiving device is waiting for the start bit i.e. the RX line to go LOW (+12V)
* When it gets start bit it waits for half bit time i.e. 104/2 = 51uS now it is in middle of start bit it reads it again to make sure it is a valid start bit not a spike.
* Then it waits for 104uS and now it is in middle of first bit it now reads the value of RX line.
* In same way it reads all 8 bits
* Now the receiver has the data.

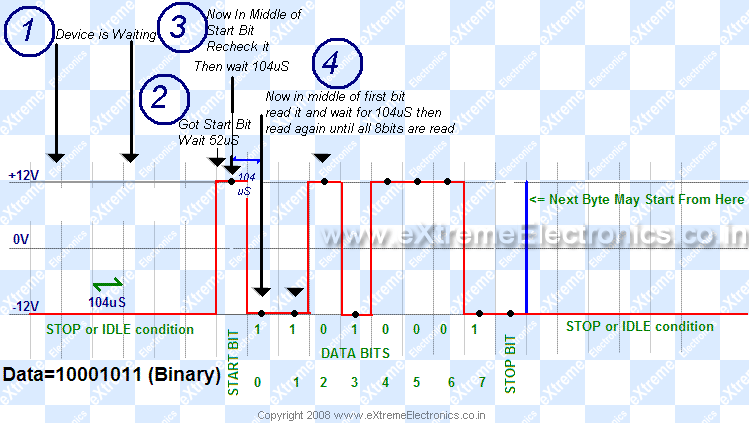


Fig. 1 RS-232 Voltage Levels

* A level converter will convert RS232 level signals (HIGH=-12V LOW=+12V) from PC to TTL level signal (HIGH=+5V LOW=0V) to be fed to MCU and **also the opposite.** As RS232 is such a common protocol there is a dedicated IC designed for this purpose of "Level Conversion". This IC is **MAX232 from Maxim**. By using charge pumps it generates high voltages (12V) and negative voltages (-12V).
* Like many microcontrollers AVR also has a dedicated hardware for serial communication this part is called the **USART - Universal Synchronous Asynchronous Receiver Transmitter**. The USART of the AVR is connected to the CPU by the following six registers.
* **UDR - USART Data Register**: Actually this is not one but two register but when you read it you will get the data stored in receive buffer and when you write data to it goes into the transmitter’s buffer.
* **UCSRA - USART Control and status Register A**: As the name suggests it is used to configure the USART and it also stores some status about the USART. There are two more of these kinds, the **UCSRB and UCSRC**.
* **UBRRH and UBRRH**: This is the USART Baud rate register, it is 16BIT wide so UBRRH is the High Byte and UBRRL is Low byte. But as we are using C language it is directly available as UBRR and compiler manages the 16BIT access.

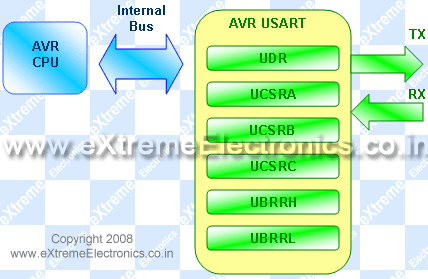


Fig. 2 Communication inside AVR

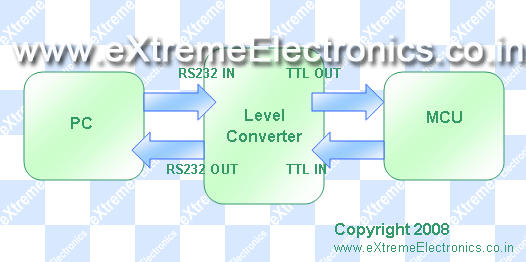


Fig. 3 Communication between PC and AVR

Initialize Registers

Start

NO

NO

NO

NO

NO

YES

o

O

YES

YES

YES

YES

STOP

Move

Right

Move

Left

Move

Forward

IF ch==’s’

IF ch==’d’

IF ch==’a’

IF ch==’w’

‘

IF character Recieved

R

R

Wait for

Character Input

Above is a flow chart diagram of the code burnt in the bot for serial communication.

**Report 4**

Details of MFCC-

• Mel Frequency Cepstral Coefficients (MFCC) are a feature widely used in automatic speech and speaker recognition. They are the extracted features or components of the audio signal which are good for identifying the linguistic content. Thus these components are our best resources to identify the input voice command with the database wav files.

Block Diagram depicting the process of obtaining MFCC of a speech signal:

**Mel Filterbank**

**Speech Signal sampled at 8KHz**

**Take Logarithm**

**Pre-emphasis**

**DCT**

**Windowing**

**MFCC Coefficients**

**Periodigram Estimates**

• Steps to obtain MFCC of an audio signal :

1. Pre – Emphasis - Frame the signal into short frames
2. Windowing – Frame is multiplied with hamming window
3. Periodogram - For each frame calculate the power spectrum
4. Mel Filterbanks - Apply the mel filterbank to the power spectra, sum the energy in each filter
5. Logarithm - Take the logarithm of all filterbank energies
6. DCT - Take the DCT of the log filterbank energies
7. MFCC Coefficients - Keep 13 DCT Coefficients, discard the rest

• Explanation of steps :

* 1. Pre – Emphasis - An audio signal is constantly changing, so to simplify things we assume that on short time scales the audio signal doesn't change much. This is why we frame the signal into 20-40ms frames.
  2. Windowing - These frames are then multiplied by a hamming window to smoothen the ends of the chopped signal.
  3. Periodogram - The next step is to calculate the power spectrum of each frame. The power spectrum is then used to estimate the periodogram to identify which frequencies are present in the frame.

First Discrete Fourier Transform is taken. Its formula is given by –

http://practicalcryptography.com/media/latex/c970f070a776e4c900bd3e8a2082a2971236b013-11pt.png

Where  http://practicalcryptography.com/media/latex/dc52f5afe0e59b4ed07b222ebcb7832bfa6579ad-11pt.png is an http://practicalcryptography.com/media/latex/50ec1aff7eaddddda99094b6cc7c602dc3c98245-11pt.png sample long analysis window (e.g. hamming window), and http://practicalcryptography.com/media/latex/6eee515a89159b28568cb0744e3a8d5a68ab8d26-11pt.png is the length of the DFT. The periodogram-based power spectral estimate for the speech frame http://practicalcryptography.com/media/latex/9b3dc4b224cc0c356406dbbdacd54dc5ab3cae9a-11pt.png is given by:

http://practicalcryptography.com/media/latex/c526edb9d52e631812798237ea3f2beea496d181-11pt.png

This is called the Periodogram estimate of the power spectrum. We take the absolute value of the complex fourier transform, and square the result. We would generally perform a 512 point FFT and keep only the first 257 coefficients.

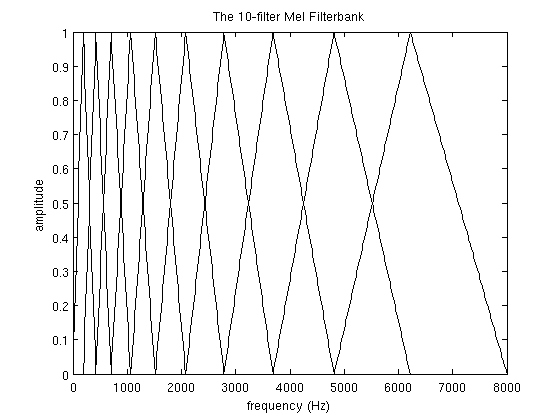
* 1. Mel Filterbanks - Compute the Mel-spaced filterbank. This is a set of 20-40 (26 is standard) triangular filters that we apply to the periodogram power spectral estimate from step 2. Our filterbank comes in the form of 26 vectors of length 257 (assuming the FFT settings fom step 2). Each vector is mostly zeros, but is non-zero for a certain section of the spectrum. To calculate filterbank energies we multiply each filterbank with the power spectrum, then add up the coefficents. Once this is performed we are left with 26 numbers that give us an indication of how much energy was in each filterbank.

The formula for converting from frequency to Mel scale is:

http://practicalcryptography.com/media/latex/369d64804e572729863c874aaa092e582bf5eb56-11pt.png

To go from Mel’s back to frequency:

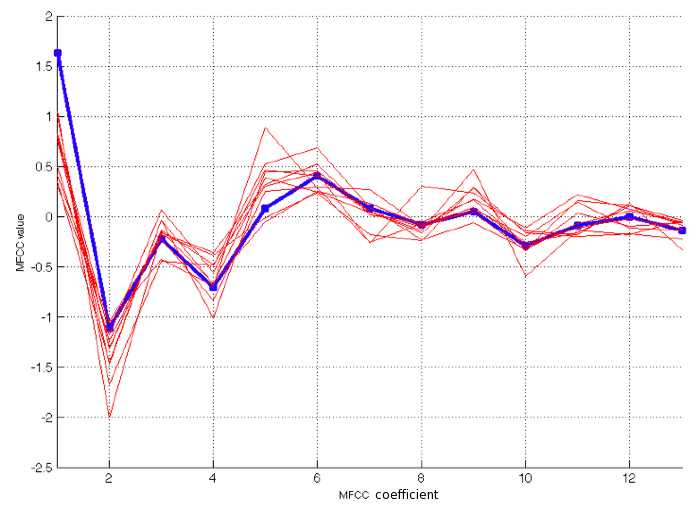
http://practicalcryptography.com/media/latex/05d74bc31f4c2a9c375dd9c95d4642d558f455a0-11pt.png



* 1. Logarithm - Once we have the filterbank energies (energies for different frequency ranges), we take the logarithm of them to get the decibels. This motivated by human hearing: we don't hear loudness on a linear scale. Large variations in energy may not sound all that different if the sound is loud to begin with. Thus logarithmic compression operation makes our features match more closely what humans actually hear.
  2. DCT - The final step is to compute the DCT (Discrete Cosine Transform) of the log filterbank energies. Main reason this is performed is because our filterbanks (frequency ranges) are all overlapping, the filterbank energies are quite correlated with each other. The DCT decorrelates the energies.

X_k = \sum_{n=0}^{2N-1} x_n \cos \left[\frac{\pi}{N} \left(n+\frac{1}{2}+\frac{N}{2}\right) \left(k+\frac{1}{2}\right) \right]

* 1. MFCC Coefficients - But only 12 of the 26 DCT coefficients are kept. This is because the higher DCT coefficients represent fast changes in the filterbank energies and it turns out that these fast changes actually degrade speech recognition performance, so we get a small improvement by dropping them.

 Sample MFCC coefficients for a general audio sign

**Report 5**

**Report 5**

The algorithm to compare the features is implemented. Dynamic Time Warping (DTW) which is used, is an algorithm for measuring similarity between two temporal sequences which may vary in time or speed. In general, DTW is a method that calculates an optimal match between two given sequences (e.g. time series) with certain restrictions. The sequences are "warped" non-linearly in the time dimension to determine a measure of their similarity independent of certain non-linear variations in the time dimension.  
DTW has been applied to temporal sequences of video, audio, and graphics data — indeed, any data which can be turned into a linear sequence can be analyzed with DTW. A well-known application has been automatic speech recognition, to cope with different speaking speeds.

The following is the algorithm –

int DTWDistance(s: array [1..n], t: array [1..m]) {

DTW := array [0..n, 0..m]

for i := 1 to n

DTW[i, 0] := infinity

for i := 1 to m

DTW[0, i] := infinity

DTW[0, 0] := 0

for i := 1 to n

for j := 1 to m

cost:= d(s[i], t[j])

DTW[i, j] := cost + minimum(DTW[i-1, j ], // insertion

DTW[i , j-1], // deletion

DTW[i-1, j-1]) // match

return DTW[n, m]

}

The following is the diagram which represents the working of DTW. It creates a distance matrix between two signals by computing the distance between the two signals. It moves from the bottom leftmost point to the top rightmost point and the minimum distance in traversing is the sum of the distance of in the current position and the minimum of the distance of the positions it crossed previously. The last value of this matrix gives the sum.

\begin{figure}\begin{center}
\leavevmode \epsfxsize =5in \epsfbox{dtw.eps}\par\centering\centering\end{center}\end{figure}

Fig. 1 – DTW curves and Search Area

**Report 6**

1. Preprocessing and Hand Detection :

Input RGB image from webcam

Remove static background by subtracting from previous images

Convert to YCbCr format

Detect faces from image and subtract them

Get regions of skin color (with faces blackened these represent hand region )

Find biggest contour in the image and get its topmost point

Orientation (angle) of this point is calculated in consecutive frames

Vectorize the angle from 0 to 3

1. Training and implementation of HMM :

Send appropriate command to the bot

Identify the gesture performed using the trained matrices

Test with the trained database using Forward-Backward algorithm

Obtain orientation of hand from input image for testing

Maximize the output parameterized by theta

Obtain the theta (three matrices) using Baum Welch algorithm

HMM model is trained from training data

Controlling the bot through Hand Gesture Recognition uses Hidden Markov Models which comprises of the following steps –

1) Training Phase -

Compute the Orientation of the tracked point in consecutive frames

Detect and Segment hand and Track the topmost hand point

Quantize the angles obtained for training

Train the classifier using Baum-Welch Algorithm

(for n gestures)

2) Testing/Implementation Phase -

Test it with the database to get the best matching HMM model using Forward – Backward Algorithm

Quantize the angles obtained for training

Compute the Orientation of the tracked point in consecutive frames

Get the i/p image and track the topmost hand point

Send the gesture recognized to the bot and move it

I Training Phase –

(a) Segmentation of hand – The first step is to detect and segment the hand from rest of the image. This is done in two steps –

(1) Face Detection – From the input image face is detected and removed. This is done by using Haar Cascades/Viola Jones Classifier. Only one face is detected and blackened out from the image so that it does not interfere with hand detection.

(2) Hand Detection – Hand is detected by getting those areas in the image which correspond to skin color. Out of these areas, the one with the largest contour size is detested and tracked. In the hand that is detected, the coordinates of the topmost point are recorded and the angle of the topmost point in consecutive frames is tracked.

(b) Vector Quanntization – The angles that are obtained have to be converted into codeword of size 4 i.e. 0, 1, 2 and 3 before training. This is done by rounding off the angle to a multiple of 90. For eg., an angle of 160° will be rounded off to 180°. Then the number of states for each model is chosen. This can be done by looking at the gesture and then selecting the number of states on the basis of the complexity of the gesture.

135° 90° (2) 45°

180° (3) 0° (1)

225° 270° (3) 315°

Fig. Vector Quantization

(c) Training of the Classifier – Based on the training set, three different matrices namely Initialization, Emission and Transmission matrices are computed using Baum-Welch Algorithm.

The Baum–Welch algorithm uses the well-known EM algorithm to find the maximum likelihood estimate of the parameters of a hidden Markov model given a set of observed feature vectors.

Let X\_t be a discrete hidden random variable with N possible values. We assume the P(X\_t|X\_{t-1}) is independent of time t. We can present this information as a time independent stochastic transition matrix A=\{a\_{ij}\}=P(X\_t=j|X\_{t-1}=i)

The initial state distribution (i.e. when t=1) is given by \pi\_i=P(X\_1 = i)

The observation variables Y\_t can take one of K possible values. The probability of a certain observation vector at time t for state j is given by: b\_j(y\_t)=P(Y\_t=y\_t|X\_t=j) B=\{b\_{ij}\} is a K by N matrix.

An observation sequence is given by Y= (Y\_1=y\_1,Y\_2=y\_2,...,Y\_t=y\_t)

Thus we can describe a hidden Markov chain by \theta = (A,B,\pi). The Baum–Welch algorithm finds \theta^\* = \max\_{\theta} P(Y|\theta). (i.e. the HMM parameters \theta that maximize the probability of the observation.)

II Testing Phase –

The hand is tracked, the angle of the topmost point is tracked and quantized and then sent for testing with the database. This is done using forward-backward algorithm. The forward–backward algorithm is an inference algorithm for hidden Markov models which computes the posterior marginals of all hidden state variables given a sequence of observations/emissions o\_{1:t}:= o\_1,\dots, o\_t, i.e. it computes, for all hidden state variables X\_k \in \{X\_1, \dots, X\_t\}, the distribution P(X\_k\ |\ o\_{1:t}). This finds an optimal match with the HMM gesture model in the database. The appropriate gesture is recognized and the associated command is sent to the bot.

III Hidden Markov Model (HMM)

Consider the example. There is a man in a room and he is given three coins to flip. The room is sealed and there are no windows so no one can see inside. However, outside the room is a display which shows the result of the coin flips, that is, Heads or Tails. The man may flip the coins in any order and therefore we may get a result like this, HHTHHTTTHTHHTTTH etc. This is one of the several thousand possibilities and it is impossible to predict the outcome. The outcome is termed as the "Observation Sequence". Now, imagine that the third coin is highly biased to produce heads. Then its obvious that the output sequence will show more heads than tails. Next, lets assume that the chance of flipping the third coin after the first and second coin is nearly zero. Which means the chances of transition from the first and second coin to the third coin is very less and thus we would see much less heads if the man starts with the first or second coin. Lastly assume that each of the three coins have a probability associated with them that indicates the chance that the man will start the process with that coin. Now lets look back at the three assumptions made. The first one is called the "Emission probability" and indicates the probability that a particular observation is seen at a specific state. The second is called the "Transition Probability" and represents the probability of transition from one change to the other. The third assumption indicated the "Initial Probability" associated with each state. These three matrices totally define the system. In our example, the coins can be thought of as "States" and Heads/Tail sequence is the "Observation Sequence".

Formally HMM is defined as –

The set of states S = {s1, s2, . . . , sNs}, And a set of parameters: λ = { Π, A, B}

* Transition probabilities A = {aij = P(qj at t +1 | qi at t)},where P(a | b) is the conditional probability of a given b, t = 1, . . . , T is time, and qi in Q. Informally, A is the probability that the next state is qj given that the current state is qi.
* Observations (symbols) O = { Ok }, k = 1, . . . , M .
* Emission probabilities B is: B = { bik = bi(Ok) = P(Ok | qi) }, where ok in O. Informally, B is the probability that the output is ok given that the current state is qi.
* Initial state probabilities Π = {pi = P(qi at t = 1)}.

Take another example. Lets consider a city with three kinds of weathers: Sunny, Cloudy and Rainy and depending on what the weather is you might consider Staying home, Talking a Walk, Shopping. Now although its not easy to predict weather but given the assumption that the weather doesn't change during the course of the day and that you have been given a log of the actions performed over the past few days and the relation between the action and the specific weather(emission probabilities) it is possible to fairly estimate the weather on the following day. This can be achieved using the markov model.

In the gesture recognition program, a total of n HMM models are trained for n gestures to be recognized. For each model, a double array of quantized features is used as the training set and consists of m number of states which are hidden. For each model, the three matrices are computed. For testing, the feature extracted is checked with each model to obtain an optimal match.

**Qualitative Analysis Of Project**

Problems faced/Scope for improvement-

1. The extracting of mfcc features was a time consuming computational process. This led to a significant delay between the command and subsequent action. This also meant that consecutive commands with less time gap couldn’t be handled. **Solution**- HMM which was used for hand gesture recognition could be used even in the case of speech recognition to reduce the computation time.

2. Sometimes a particular command wouldn’t get associated to its correct action. This was mainly because of background noise or difference in microphone levels. **Solution**- An even larger database and proper pre-processing techniques such as addition of a low pass noise filter would solve this problem.

3. Since skin colour was used as a dominant feature for detecting hand, sometimes background conditions resembling this feature was also detected. **Solution**- Use of a RGB-D cameras like Microsoft Kinect or Asus XION which also have a feature of depth (distance from camera) along with colour, would help to give us an efficient tracking of the hand.

4. Apart from the above problems, hardware issues were faced at times. This included malfunctioning of the bot due to loose connections, some issues with usb ports, and other unexplainable phenomenon. **Solution**- Generally they could be solved using basic techniques of circuit debugging, restarting the laptop, burning the bot’s program on the board again, etc.