Team Varna

Breath Enabled Assistive Communication Device

Table Of Contents

Basic Idea:

Life is characterized by the ability to breathe and our idea was to make use of this eternally natural tendency of every living being as a way for communication.

Our main idea is based on using one's breath as a mode of communication targeting audience who have lost the ability to speak normally. The idea is to segregate breaths into three distinct types: Long(-), Short(.) and Absent(0), and then code the alphabets, digits and symbols in terms of these 'dots' and 'dashes', popularly known as Morse Code. Useful information can be analysed using appropriate encoding-decoding techniques to get human readable text which can then also be translated to computer generated voice using appropriate software packages.

Team:

Varna (वर्ण)

Team Members	Roll Number	Email ID
Ajinkya Werulkar (Team Leader)	170260002	ajinkyawerulkar@gmail.com
Sanjoli Narang	17D100013	sanjoli.yg@gmail.com
Raunak Dutta	170260026	raunakdutta99@gmail.com

Motivation:

Motivation comes from the ardent desire to serve society, especially those people who lose their senses in accidents and become severely disabled. The main aim is to increase the quality of life of these people by developing alternative and simpler ways which they can use to give words to their thoughts and needs without relying on others for support.

But the currently available assistive communication devices are very expensive and bulky, hence cannot be made to reach the doorstep of common man. Thus, there exists an immediate need for creating viable devices that are cheaper and easier to use.

An initial initiative was taken in the production of "TALK", but taking this project, we want to make it even more user friendly and faster without compromising accuracy.

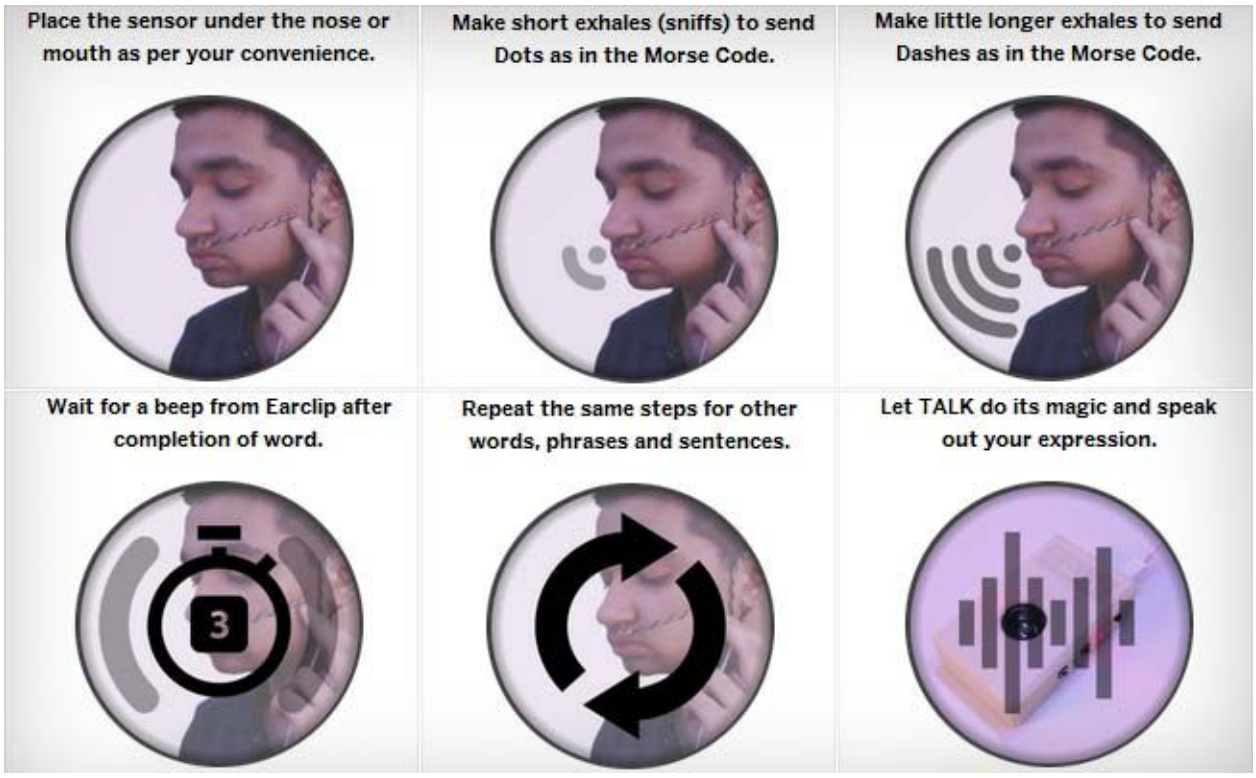
Every human whether handicapped or not share commonalities with their fellow humans, like we all breathe, everyone has a beating heart and running nerves. The main idea of this project is to use one such common trait, breathing, as a way of developing a device that can allow disabled people to communicate easily and efficiently. Since every human can control his breathe, it can be used even by severely handicapped people.

Working on such projects allow us to understand the problems in society and device novel ideas to resolve them, and provides ample space for innovation. It would also introduce us to the methods of data and time management to make the process fast and at the same time error precise. Meanwhile, we would also learn how TTS libraries are created and what determines their quality, which protocols (I2C, RS etc) provide the greatest speed and immunity to noise in sensors and how to filter useful data of sensors.

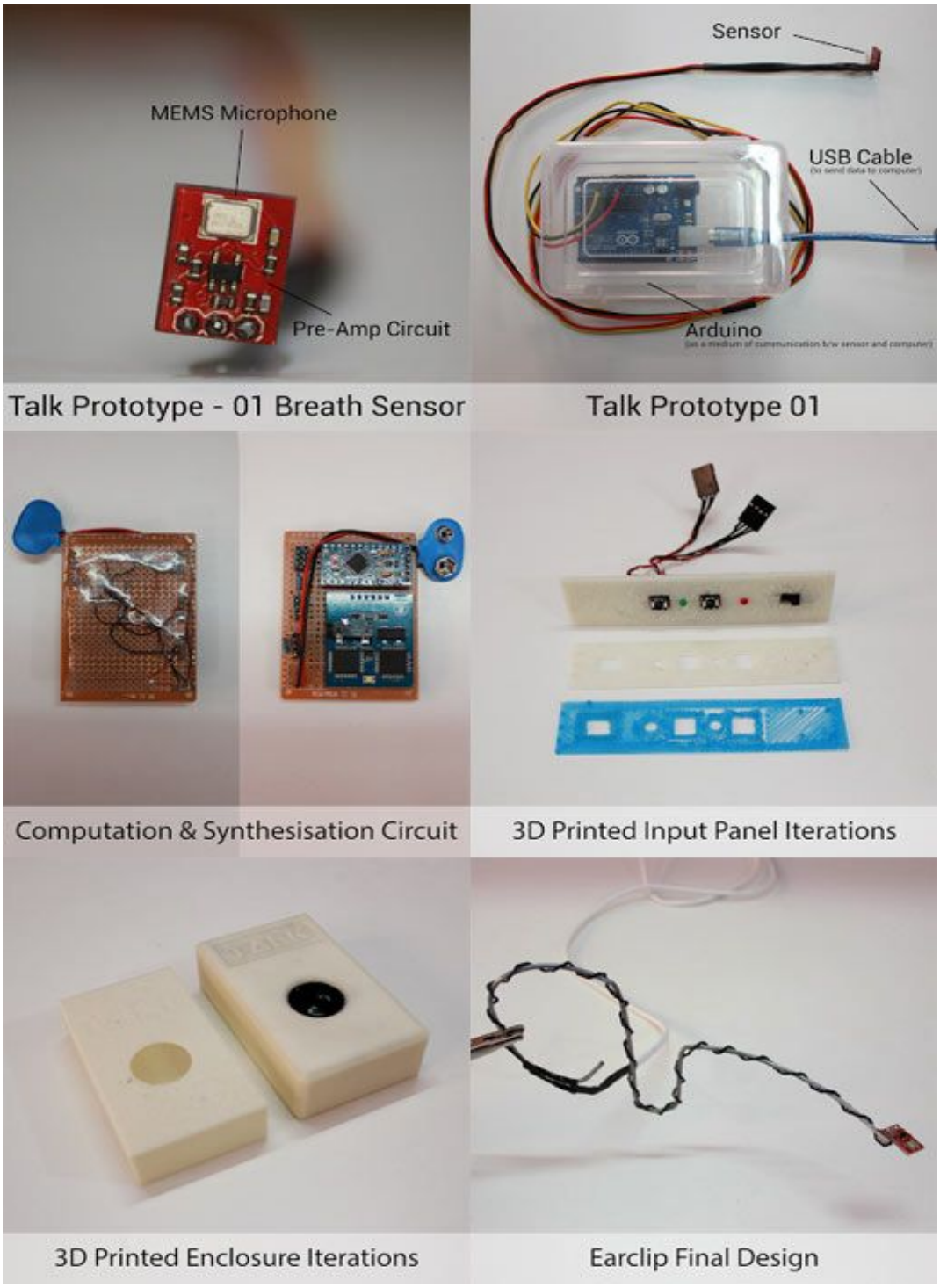
Reference:

Inspired from the alternative communication device, “TALK” developed by Arsh Shah Dilbagi as a tool of assistive communication.
The project is based on how “TALK” was implemented as a viable and cheaper device for the people compared to the alternates available in the market and it can be made accessible to everyone because of low cost and portability.

Explanation of TALK:



Components of TALK:



Detailed Implementation and Technical Aspects:

The breathe durations are used to generate a binary coded pattern that is further processed to generate meaningful expressions in English language.

Data Extraction:

The first step was to take the required data from breathe. There were three choices to get duration data from the breaths.

Pressure sensor:

A pressure sensor was the most natural choice to employ but the human breath pressure range lies in millibars, so we needed high precision as well as accuracy in the sensor. BMP180 pressure sensor module was chosen because of its reasonable cost, its interface with the Arduino, its own data exchange library in AVR and its adjustable resolution.
But, it was found that, even in millibars, the breath pressure was changing the sensor values only by 0.1 or 0.2 units. So we had to edit its code to read the decimal part and get maximum resolution, and on testing it gave good results.
The sensor module needs a specific orientation for capturing the pressure change correctly, which we didn't realise early on for quite some time, and hence, we were getting frivolous data in the beginning.
The numbers we got finally on setting everything correctly changes by nearly 50 units (with respect to the atmosphere) upon getting a breathe.

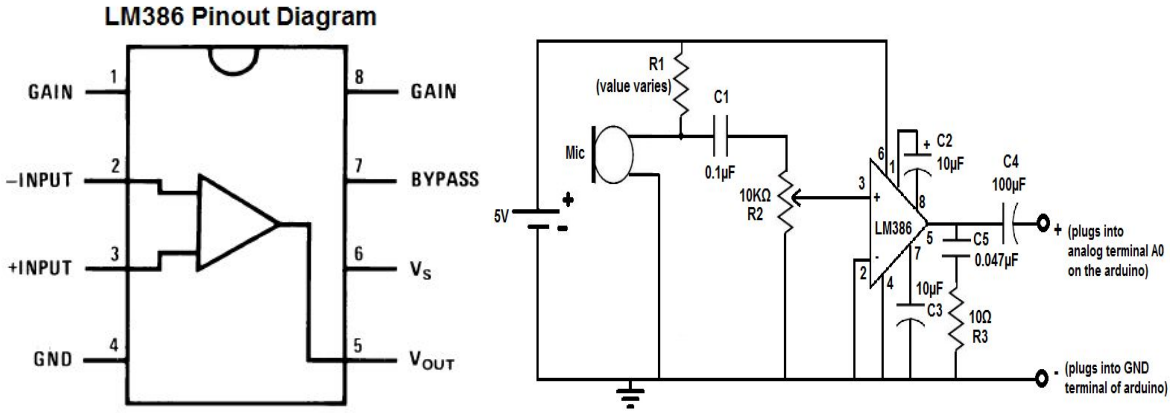
Temperature sensor:

Human breathe is warmer in accordance with the current body temperature.

Initially, while we were having troubles using the pressure sensor, we tried the temp sensor, but couldn't get good output, This is because a thermistor based temperature sensor doesn't change its value fastly with change in temp, and like the pressure change, the temp changes are also very small.

Microphone:

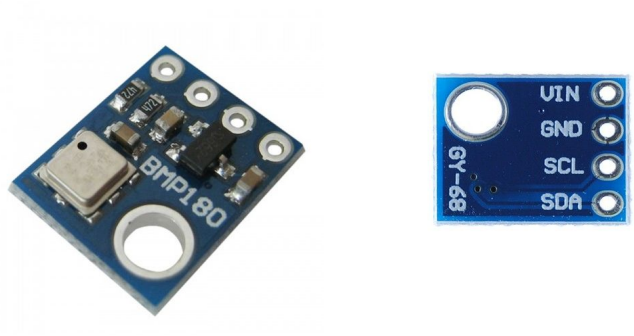
We also tried to use a highly sensitive microphone. But the main problem with it was that it used to give output as an AC signal (vibrations of a material inside it) in millivolts, so we also needed an amplifier (op amp) for it which introduces a lot of errors and reduces sensitivity considerably. Since it is made for detecting sound, faint breathe signals didn't give us appropriate data.



(OpAmp LM 386)

BMP180 Pressure Sensor Module:

The module uses I^2C to interface with Arduino, and exchanges data by SDA (serial data) line that goes to A4 and an SCL (serial clock) line that goes to A5 for synchronous communication.
It employs a bit of calibration and mathematics to calculate temperature, pressure and altitude. Go through its datasheet for further reference.
Its library can be easily downloaded from the Arduino IDE ("Manage libraries").



Getting Duration Pattern (Arduino Code):

Instead of using binary pattern (1/0) (1 for long breathe and 0 for short), we decided to use a ternary pattern (2/1/0) (2 for long, 1 for short and 0 for no breathe detection with respect to the environment). Here, breathe implies that the breathe pressure has exceeded the threshold set by us.
So, in normal breathing (below a certain threshold), the sensor would continue to fetch data but that won't be further processed. The user needs to put a slight push in the breathe to exceed the threshold. Once it is done, the code will process the data, integrate it in the pattern described, convert the pattern into a number using a function and then return that number, which would correspond to a particular expression say an alphabet or a word.

The code to classify a forced breathe as long or short was the most challenging one (setting the time duration to differentiate between the two). There were many iterations of the code and the main aim was to increase its flexibility and user friendliness. In simple words, we didn't want the code to force too long a breathe for a specific duration. We had to take hundreds of tests for comfortable breathing and adjusting the range accordingly.

The Main Code:

The first iteration of the code which completely detects alphabets using the modified Morse code and organizes them in sentences is here.

This version of code gives user great flexibility in the sense that the user can pause the program at any point if he is tired. Eg., While creating words from alphabets, the user can make the program wait until he is ready to give the input for the next alphabet.

We have also compiled a video demonstrating the first stage described in this section showing formation of alphabets, words, sentences and using the built in word/sentence database. It can be found here.

Functions and logic explanation:

All the basic functions used in the main code are described below:

1. readPressure(): reads relative pressure and returns a float.
2. function1(): Takes data from the pressure sensor using readPressure() and shows each of breathe value above threshold in terms of u,w,z,y,x (dummy variables) and returns an int.
3. ConvertToAlphabet(int): Takes input as an int corresponding to the previous function, and in return,displays the alphabet associated to it.
4. GetWord(): Uses ConvertToAlphabet(function1()) until it returns a space, and compiles it into a word with a front space.

5. GetLine(): Uses GetWord() function and returns a line when GetWord() returns a full stop.
6. GetMode(): Returns an int, indicative of the device's current mode, based on function1().
7. ConvertToLine(): Takes int from function1 and returns a sentence, can also change the mode.

Logic Explanation of the Ternary code to integer (and therefore, alphabet) conversion:

Letter	**Code**	**Conversion**		**integer**
A	0 0 1 2	2+2	=	4
B	2 1 1 1	1+2+4+16	=	23
C	2 1 2 1	1+4+4+16	=	25
D	0 2 1 1	1+2+8	=	11
.	0 0 0 1	1	=	1
F	1 1 2 1	1+4+4+8	=	17
G	0 2 2 1	1+4+8	=	13
H	1 1 1 1	1+2+4+8	=	15
I	0 0 1 1	1+2	=	3
J	1 2 2 2	2+4+8+8	=	22
K	0 2 1 2	2+2+8	=	12
L	1 2 1 1	1+2+8+8	=	19
M	0 0 2 2	2+4	=	6
N	0 0 2 1	1+4	=	5
O	0 2 2 2	2+4+8	=	14
P	1 2 2 1	1+4+8+8	=	21
Q	2 2 1 2	2+2+8+16	=	28
R	0 1 2 1	1+4+4	=	9
S	0 1 1 1	1+2+4	=	7
-	0 0 0 2	2	=	2

U	0 1 1 2	2+2+4	=	8
V	1 1 1 2	2+2+4+8	=	16
W	0 1 2 2	2+4+4	=	10
X	2 1 1 2	2+2+4+16	=	24
Y	2 1 2 2	2+4+4+16	=	26
Z	2 2 1 1	1+2+8+16	=	27
?	1 1 2 2	2+4+4+8	=	18
!	1 2 1 2	2+2+8+8	=	20
E	2 2 2 1	1+4+8+16	=	29
T	2 2 2 2	2+4+8+16	=	30

Note that the pattern is converted to number using normal binary to decimal conversion. Here are a few examples:

1. Air 0012 0011 0121
2. Water 0122 0012 2222 2221 0121
3. Food 1121 0222 0222 0211
4. I know it 0011 0002 0212 0021 0222 0122 0002 0011 2222 0002
5. Hi man 1111 0011 0002 0022 0012 0021 0002

Pre-Defined Database:

As it might be visible, generating sentences from letters can become a serious headache and might tend to be exhaustive for a user. Further we also needed to add codes to enable the user to change specific properties of the TTS module like volume, voice type modulation, etc. Hence we decided to shift to a method involving a pre-defined sentence database.

Our final code supports direct generation of 62 common built-in long sentences and 62 short phrases, generation of any number from digits input by the user, 30 kinds of alarming sounds and ringtones, and change

of volume, speed, voice type, pitch of speech as described below in TTS module section.

Text To Speech (TTS Module XFS5152CE):

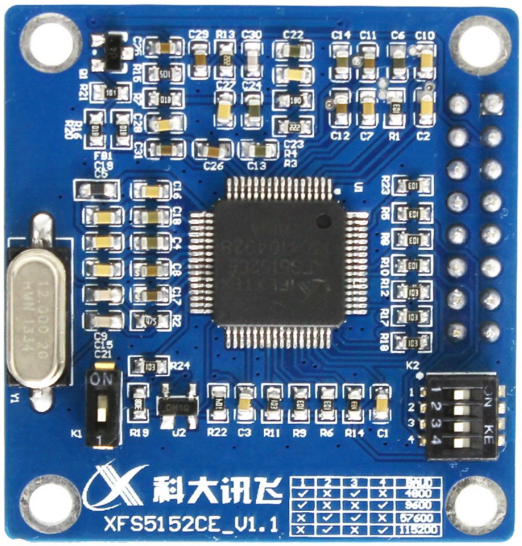
For Text to Speech conversion, we went through different choices:

1. **Use of arduino itself** : This suggestion was discarded very early due to the poor quality of sound produced (even after adding our own filter and amplifier) which is quite natural to expected from a microcontroller like AtMega which has limited PWM capability. Moreover, the TTS library for arduino also created memory limit issues.
2. **TTS modules compatible with arduino**: There are various great TTS modules in the market like EMIC2 series but they all tend to be very costly (around ₹10,000 to ship). The search for an appropriate TTS device with good properties took a lot of time. Finally, we found a TTS module **XFS5152CE** , at reasonable cost, but all its documentation is largely in Chinese which made it a little difficult to understand its working. Moreover it was to be shipped from the chinese market, making its shipping, especially getting it through customs a real headache. But it has a lot of features that had us totally mesmerised.

About the module:

This arduino compatible TTS module can communicate via any of UART, SPI or I2C interfaces among which we used UART after certain considerations and feasibility, understood its messaging format and commands from the datasheet that had to be found in English (English Version of the datasheet), and other sources and integrated everything with Pressure sensor data, giving the users the following controls over TTS:

1. Changing the Volume.
2. Changing the Speed of speech.
3. Changing the Pitch of speech.
4. Switching between 5 kinds of voices: 2 male types , 2 female types and the Donald Duck type.
5. Generating alarms and tones.
6. Stopping, pausing and resuming speech.
7. Printing the status of the TTS module.



Modified Implementation:

After obtaining the TTS Module, we made a second iteration of the code, which can be found here. This iteration of the code was designed to incorporate TTS Module communication as well as improve the efficiency of the previous iterations.
A second video explaining all the prowess of the TTS module (in sync with the VARNA device) and other features of the device can be found here.
A third video consists of a demo conversation written to highlight all the features that can be used in day to day life and can be found here.
Lastly, to ease users into the interface of the device, we have written and would like to present, the User Manual.

Components Used:

The following table explains all the components we had to get our hands on to get the project working.

Sr. No.	Components List	Source	Cost (in ₹)
1	Arduino Uno & Jumper Wires	Amazon	716
2	BMP180 Pressure Sensor	Amazon	489
3	Arduino Mega & Battery & Wires & Speaker	Mangaldeep	1097
4	TTS Voice Module	Amazon.com	4759

Total: Rs. 7061