Imran Hafeez

Speech Controlled RC Car

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Electronics

Thesis

12 May 2014

Author(s) Title Number of Pages Date	Imran Hafeez Speech Controlled RC Car 39 pages + 1 appendix 12 May 2014
Degree	Bachelor of Engineering
Degree Programme	Electronics
Specialisation option	
Instructor	Janne Mäntykoski, Senior Lecturer

This thesis is based on Final Year Project, which falls under the category of embedded systems. The Aim of this project was to modify a small remote control (RC) car and drive it by using speech commands based on EasyVR module and Arduino microcontroller.

Theoretical background is provided for the reader to understand the basics of embedded system and its categories. It also provides the understanding of interface connection, communication modes between the input/output components of microcontroller and external I/O devices.

It is described in the report about the hardware components involved in the making of the system such as Arduino microcontroller board, motor shield, Easy VR speech recognition module, DC motors of small RC (remote control) car. Then it is described that how the hardware is integrated and which programming language was used to get the desired output. Coding done for the system is provided in the appendix.

Result of the project is satisfactory but it also has its limitations. Reasons are mentioned for the limitations of the system along with the suggestions for the future enhancements and the conclusion for this FYP.

Keywords	Speech commands, Easy VR, Arduino Microcontroller

Contents

1 Introduction

The idea for the final year project (FYP) emerged by mutual discussion with the supervisor Janne Mäntykoski. It was intriguing to drive a remote control car with speech commands. The basic theme of FYP is to drive a small toy car, which has 2 dc motors, one is dedicated to reverse and forward direction. The Second motor is mounted in the front for changing the direction to right and left. Other components include Arduino microcontroller, Arduino motor shield, and EasyVR module.

The reason for choosing this topic for FYP is the ease of working with Arduino microntroller. Arduino board is very famous among embedded system designers and plenty of literature is available in the internet.

In research phase of the FYP, the main emphasis was to get the knowledge about the workings of EasyVR module. EasyVR module is best suited for the home automation projects like switching on the lights, lifting the curtains, opening the door and changing TV (television). EasyVR module was challenging to understand.

There are enough examples for the interface of motor controller with Arduino microcontroller given in internet, but examples for the interface EasyVR module with Arduino is lacking. Practical experience from the work placement provided enough knowledge on the usages of Arduino microcontroller and how to interfaces it with I/O (input-output) components such as temperature sensor and Zigbee module for wireless communication.

The programming for Arduino microcontroller was done in C language by using the Arduino IDE (integrated devolvement environment). EasyVR commander software was used for the programming of EasyVR module, which generates the source code automatically after the training of commands.

2 Theory of Embedded Speech Recognition

2.1 What is an Embedded System

Before going into defining embedded system, it is better that the word system is defined. System is an arrangement in which all units or components works together in a synchronized way by following a plan. An example is a time-display system. It consists of needles, battery, dial, and chassis. These parts are assembled in way to show the real time with the passage of second and updating the time. The system of time-display updates the display by using three needles after every second. These are the rules which a time-display is following. [1, 3].

1. The three needles only move in clockwise direction 2. Among them the long and slim needle moves after every second and comes back to same position after one minute 3. Second needle, which indicates minute moves every minute and comes to the same position after an hour 4. Needle which is short among three needles rotates every hour and it comes to the same position after twelve hours. [1, 3].

Now let’s move to the computer system which consists of the components given below. [1, 3].

1) Microprocessor
2) Memory (Random Access Memory, Read only memory) are termed as primary memory. Secondary memory includes hard disks, USB
3) Keyboard, Mouse, and Digitizer, etc are termed as input units
4) Monitor, printer are termed as output units or devices
5) Units for networking are Ethernet card, modems, routers, etc.

A system can be called as embedded system which consists of computer hardware with software embedded in it as one its integral part. This can be an independent system or can be a part of bigger system. Software usually embeds in ROM (Read Only Memory) and it does not need secondary memory types. An embedded system consists of three primary components given below. Figure 1 shows the units in the hardware of an embedded system. [1, 4.]

1) It contains main application software, which perform multiple tasks.
2) It has a real time operating system (RTOS) that handles the application software. RTOS gives definition for the system to work. Clear rules are set for the execution of application software.

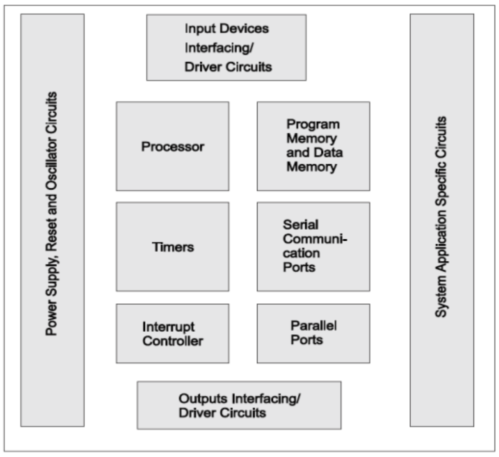

Input Devices Interfacing/ Driver Circuits
Power Supply, Reset and Oscillator Circuits
Processor
Program Memory and Data Memory
Timers
Serial Communi-cation Ports
Interrupt Controller
Parallel Ports
Outputs Interfacing/ Driver Circuits
System Application Specific Circuits

Figure 1. Components of embedded system hardware [1,5]

2.1.1 Classification of an Embedded Systems

Embedded systems can be classified into three types, given below.

1) Small Scale embedded systems

Single 8 or 16 bit microcontrollers are used for small scale embedded systems. They contain little hardware and usually board level programming is used in design. C programming language is usually used. The C code is compiled into the assembly and executable codes are allocated in the microcontroller memory. Designer must keep account of the limitations of memory and power dissipation during the operation of the system. [1, 5.]

2) Medium Scale Embedded Systems

These can be built with single or several 16- or 32-bit microcontrollers. To deal with the complexities of software design, these tools could be used. RTOS, Source code engineering tool and Integrated Development Environment. Complexities involving hardware can be solved by the programming tools. For different functions, we can use ASSPs (application-specific standard product) which is a semiconductor device integrated circuit (IC). [1, 6.]

3) Sophisticated Embedded Systems

These systems contain a lot of hardware and software complexities. It may require scalable processor and programmable logic arrays. Scalable Processor is a 32- and 64-bit microprocessor from Sun Microsystems that is based on reduced instruction set computing (RISC). [1, 6.]

2.2 Processor in a System

Processor has an utmost importance in embedded systems. In depth knowledge of microprocessors and microcontrollers is a prerequisite for designing an embedded system. A processor consists of two units: Program Flow Control Unit (CU) and Execution Unit (EU). Control Unit is a part of computers Central Processing Unit (CPU), which instructs the operation of the processor. Communication and coordination of Input/Output devices are controlled by CPU. [3, 8.]

The control unit is a component of a computer's central processing unit (CPU) which directs operation of the processor. It controls communication and co-ordination between input/output devices. It reads and interprets instructions and determines the sequence for processing the data. Circuits of EU implement the instructions related to data transfer and data conversion. Arithmetic and Logical Unit (ALU), circuits that execute instructions for a program control task such as halt, interrupt are parts of EU. [3, 8.]

Processor runs the instructions which are defined in the processor instruction set. Instructions are executed sequentially from the memory during operation. The following are the important points to consider for selecting a processor. 1) Instruction set 2) Number of bits in a single arithmetic or logical operation 3) Clock frequency in MHz [3, 8].

2.2.1 Microprocessor and Microcontroller

In many mechatronic systems, certain control tasks are based on complex relationship among many inputs and outputs for which strict hardware implementation is not efficient. A better approach would be the use of microcontroller based system to implement a software solution. [2,239.]

A microprocessor is based on a very-large-scale-integration (VLSI) chip made up of many digital circuits that carry out several important functions such as arithmetic, logic, and control. Microprocessor is also known a central processing unit (CPU) is where basic computation and system control takes place. The Arithmetic logic unit (ALU) is a part of the CPU that executes mathematical functions. The job of instruction decoder is to interpret instructions which are fetched and decoded sequentially from the memory. By looking at the figure 2 we can see that the bus is collection of communication lines and the data is shared via these lines with other components. [2,239-240.]

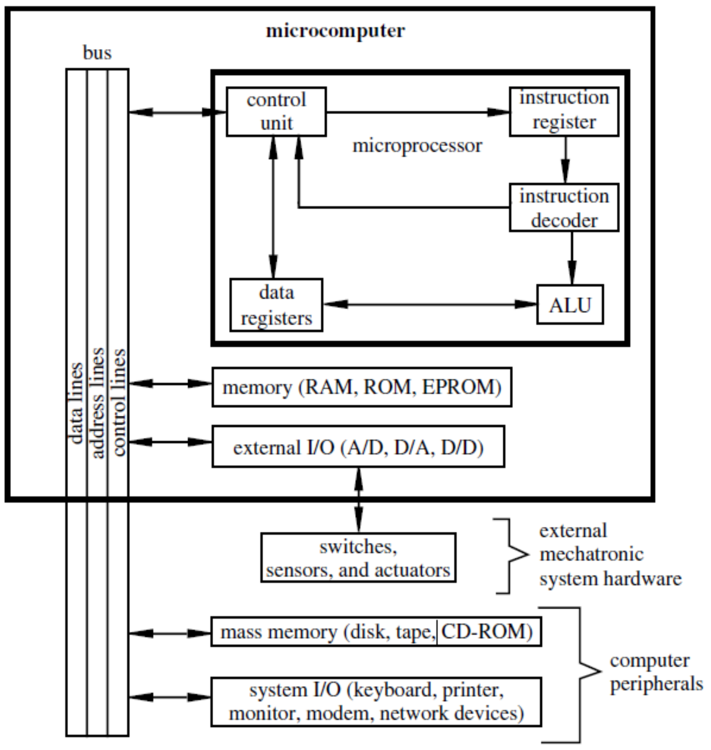

microcomputer
bus
control unit
instruction register
microprocessor
instruction decoder
data registers
ALU
data lines
address lines
control lines
memory (RAM, ROM, EPROM)
external I/O (A/D, D/A, D/D)
switches, sensors, and actuators
external mechatronic system hardware
mass memory (disk, tape, CD-ROM)
system I/O (keyboard, printer, monitor, modem, network devices)
computer peripherals

Figure 2. Microcontroller architecture [2,240]

The job of data lines is to communicate the data of registers stored in various components such as memory. Selection of devices on the bus or data locations within memory is done by address lines. Transmissions of various signals such as read and write, system clock signal and system interrupts is done by the **control lines.** [3,240.]

CPU operations are heavily depended on the memory devices. Types of memory include **read-only-memory (ROM)**, which is used for permanent storage of data and **random-access-memory (RAM)** is used for temporary storage of data. Types of ROM includes **Mask (ROM)** and **One-Time Programmable ROM.** In Mask(ROM) Data bits are permanently programmed into a microchip by the manufacturer of the external MROM chip. MROM design is based on bipolar transistor-based circuits. [3,231.]

RAM is divided into **static RAM (SRAM)** and **dynamic (RAM).** SRAM memory is made up of transistor-based flip-flop circuit that typically stores its data by using bistable latching. DRAM works by storing each bit in a storage cell consisting of a capacitor and a transistor. Capacitors tend to lose their charge rather quickly and needs to be recharged.

Microprocessor use several input and output I/O devices for communication. These will be discussed in the later sections. [2, 240-241.]

Microcontroller, is based on a single IC consists of specialized circuits and functions and are used in variety of applications such as home appliances, telecommunication equipment, office equipment. Single IC contains microprocessor, memory, and Input/output capabilities. Examples of microcontrollers are AVR, Microchips PIC and Motorola's 68HC11. Low cost, ease of programming and small size is the main motivating factors behind the development. For mechatronics system design, it is very attractive due to its size, cost, and vast functionality that can be easily combined in a system to get the desired control functions. [2,242.]

Let's have a look at figure 3, which shows a block diagram of a microcontroller.

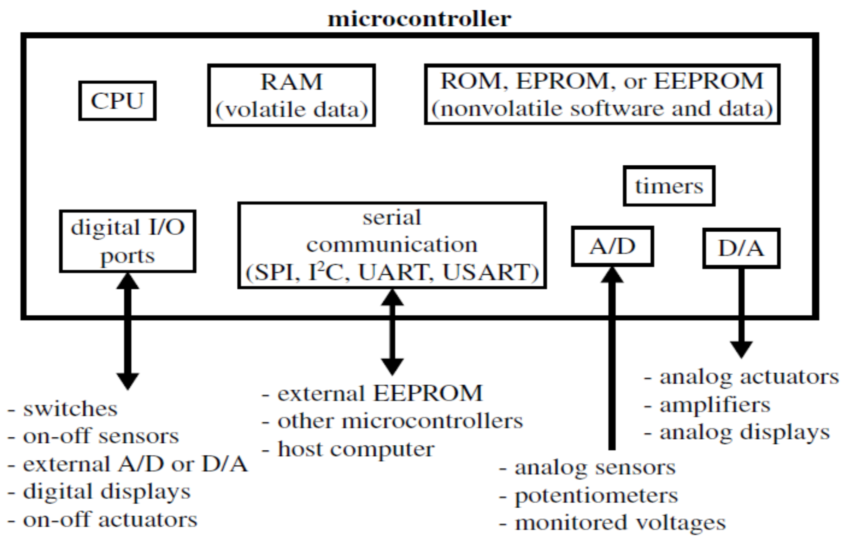


Figure 3. Components of microcontroller [2,243]

The components of microcontroller consists of the CPU, RAM, ROM, digital input/output ports, serial communication interface, timers A/D and D/A converters. The block diagram in figure 3 also shows other devices which use depends on the nature of the system design. Execution of software is done by CPU after retrieving it from ROM and controls all the microcontroller components. The Job of RAM is to provide temporary storage to the setting and values used by software in operation. Software which is permanently stored in ROM is known as firmware. [2, 242.]

Microcontroller’s memory ranges from 1 kilobyte to several tens of kilobytes. This is much less than microprocessors whose RAM memory ranges from megabytes to gigabytes. Beside this microcontroller clock speeds are slower as compared to microprocessor. For certain application microcontroller memory is a constraint and can be solved by adding wide range of products to satisfy the need. [2, 244.]

2.3 Role of Input and Output Components

The movement of the information takes place through the Input/Output (I/O) components located on board to the I/O components connected to an embedded system. The function of the board Input components, is to brings information from an input device to the CPU and the output components take information out from CPU to an output device. Figure 4 shows the graphical representation of I/O. [3,253.]

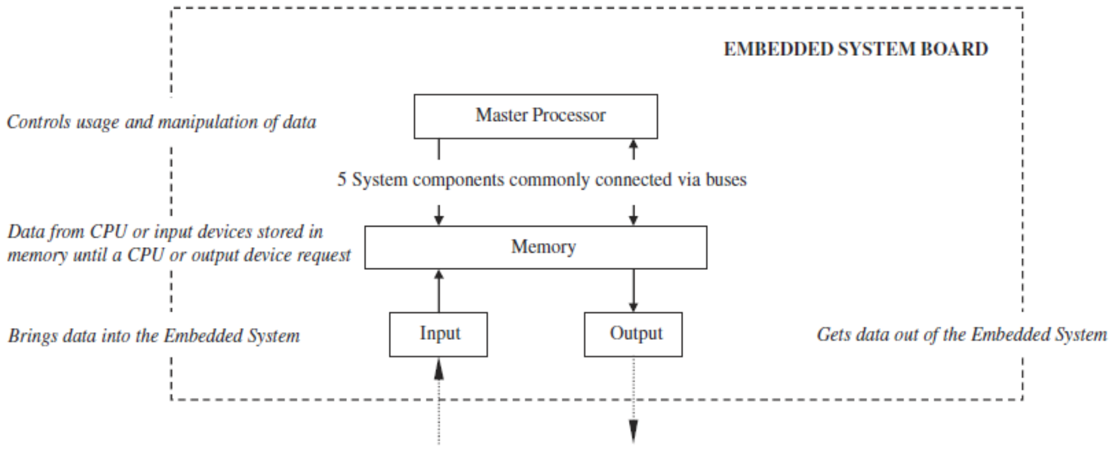

EMBEDDED SYSTEM BOARD
Controls usage and manipulation of data
Master Processor
5 System components commonly connected via buses
Data from CPU or input devices stored in memory until a CPU or output device request
Memory
Brings data into the Embedded System
Input
Output
Gets data out of the Embedded System

Figure 4. I/O block diagram [3,253]

All electromechanical systems, can be connected to a microcontroller and behave as an I/O device. I/O can be divided into subcategories of output devices, input devices and devices that have properties of both input and output devices. Printers, Display and Led are examples of an output device, which receives data from the microcontroller board I/O components. Input devices of an embedded system are keyboard, mouse transfer data to the microcontroller I/O components. [3,254.]

Serial data transmission is a form of data transmission where by bits of characters are sent one at a time along a communication path. Serial data transmissions travel over a single wire in one direction and are often compared to data transmission. They each follow an 8 bit character.

2.3.1 Serial vs. Parallel I/O

The Input/output components of microcontroller can transfer and receive data by using serial data transmission. This means that data is transferred and received one bit at a time along the transmission line. Subdivision of serial data transmission includes serial port and a serial interface. There are three ways in which data communication can take place between two devices. [3,257.]

1) Simplex

Simplex is a mode of communication that occurs only in one direction. Figure 5 is provided for more clarification.

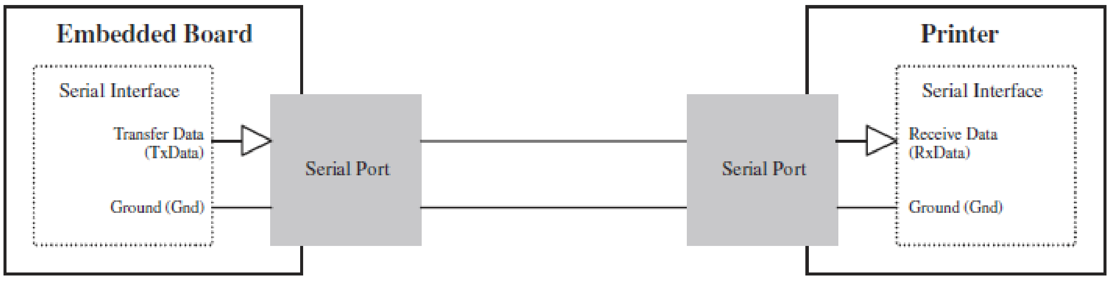


Figure 5. Simplex transmission [3,257]

2) A half-duplex

In this mode of communication, data is transmitted and received in both direction but only one direction at a time not simultaneously. Figure 6 is provided for further clarification.

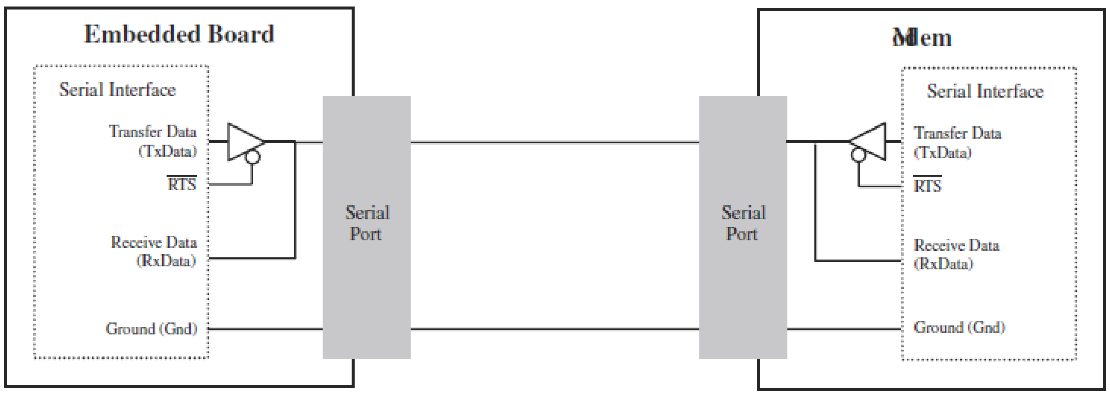


Figure 6. Half-duplex [3,257]

3) A full duplex

In this mode of data communication, data can be transmitted and received simultaneously. See figure 7 given below.

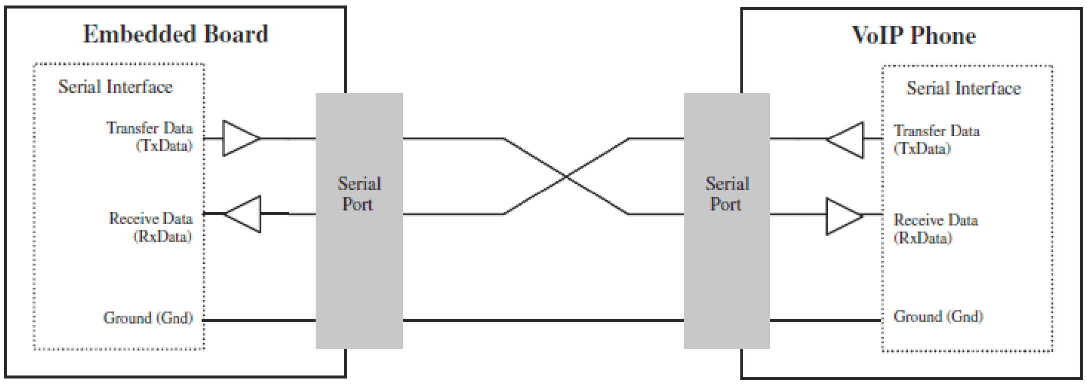


Figure 7. Full duplex transmission [3,258]

There are also components in embedded systems which are considered as parallel I/O devices which can transmit and receive data simultaneously. Parallel I/O uses parallel port and communication interface called as parallel interface. Management of data communication in between the CPU and I/O devices are done by parallel interface. In the context of parallel data transmission, it is differing from serial data transmission on the basis of data transmission direction and on the process how the data bits are processed. Both serial and parallel I/O uses simplex, half-duplex and full-duplex for determining data transmission direction. [3,267-268.]

2.3.2 Interfacing the I/O components

As discussed in the earlier section, microcontroller hardware is made up of integrated components such as CPU, controllers, communication ports, communication interfaces, I/O busses and a transmission medium. Figure 8 is provided for I/O subsystem. This shows clearly the interconnected components, data communication port, and external I/O devices. [3,277.]

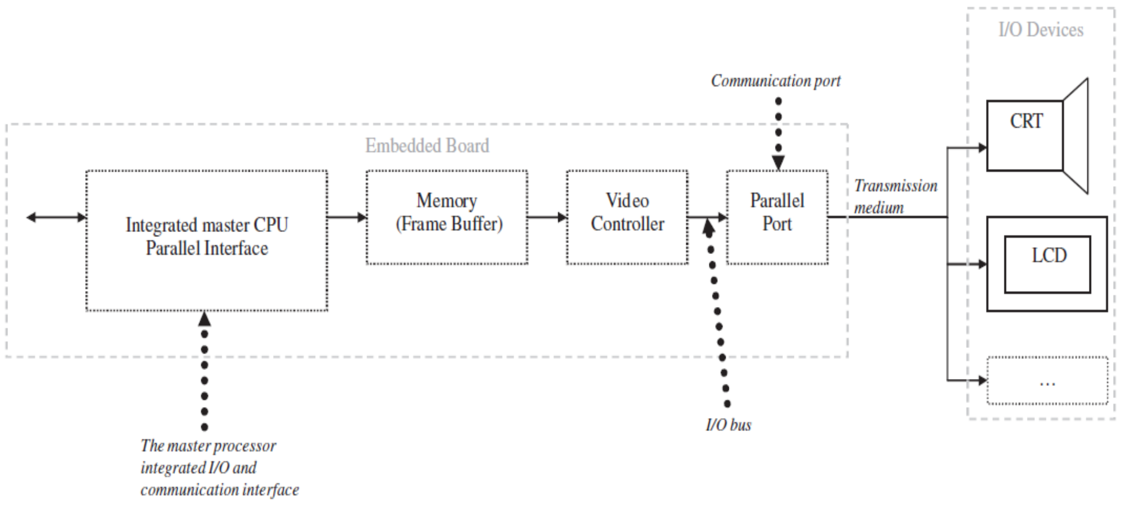


Figure 8. I/O subsystem [3,277]

2.3.3 Connecting the I/O device to the Microcontroller

External I/O devices, such as keyboards, mouse; printers can be connected to the microcontroller by using the transmission medium and communication port. Transmission medium, wireless or wired plays an important role upon the interface schemes of I/O devices to the microcontroller board [4,277]. Figure 9 shows the wired medium transmission.

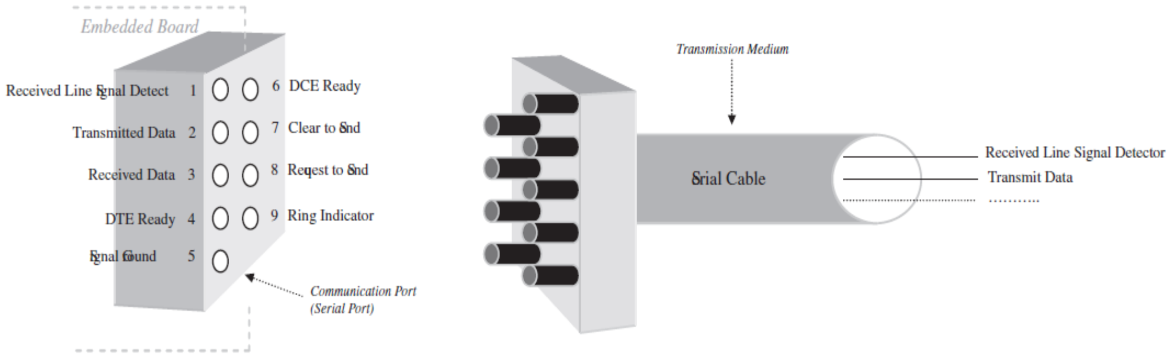


Figure 9. Wired transmission medium [3,277]

Figure 9 is showing the wired mode of transmission between the microcontroller and I/O device is based on cable with the connector head to the microcontroller board. Please consider the example of wireless transmission medium as follows. Remote control is a device that uses the wireless transmission medium for data transmission. In order to understand this mode of transmission, knowledge of infrared wireless communication is needed. Remote control emits electromagnetic waves and it is intercepted by the infrared receiver (IR) on the microcontroller [3,277].

In brief, an I/O device connection can be made to the microcontroller via I/O ports, if the I/O devices are sited on the board or an indirect connection can be made by using a communication interface and the communication port. For the connection of external I/O devices, microcontroller I/O busses play an important role.

2.4 DC Motors

Dc motors are based on well-established technology, with efficiency, wide spread availability and ease of control. Well established means that electric motors history goes back to 1821, when the motor was first built by famous British scientist Michael Faraday, can be seen in figure 10. Faraday placed a magnet in a disk of shallow disk of mercury which acted as a conductor and above this, he suspended a wire. One end of the wire was connected the positive terminal of battery and other end was connected to the negative pole. Mercury acted as a liquid commutator brush, and current flowed through the wire. As it happened, the resultant magnetic field created by flowing current, pushed the wire out and started to revolve around the permanent magnet. [6, 7.]

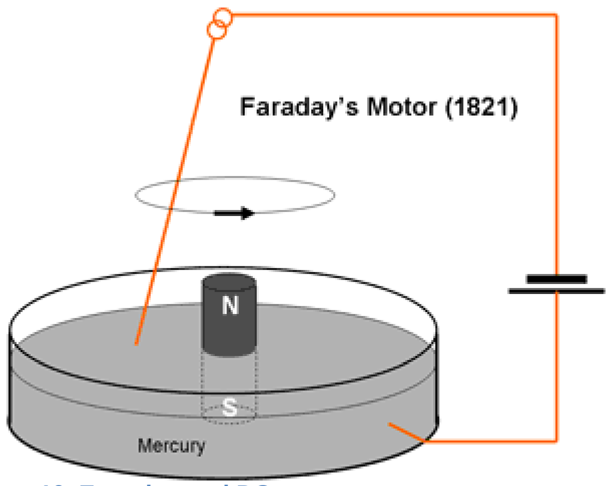


Figure 10. Experimental DC motor

Although this single pole motor, could produce circular motion in the wire. Farday moved to other endeavours and this concept was left to others to make it something more refined. By 1832, William Sturgeon, British inventor of electromagnet, came up with four pole motor that drove a roasting spit, thus known as creating the first modern kitchen appliance. He is best known for inventing the brush commutator, which continuous to be an integral part of most DC motors. In the US, Thomas Davenport used his prototype motors to power a model electric train and sundry shop equipment. Patent was granted to him for his work in 1837. The use of electric motor was restricted by the relative scarcity of electricity. At the time, acid based batteries were almost the exclusive source of electric energy. [6, 8.]

2.4.1 How Electric Motor Works

Electricity is converted to motion by magnetism in an electric motor. If we place the magnets on the table, facing each other causes the magnets to attract or pull, which depends how the magnets are positioned. Magnet can attract with enough potency to drag the other magnet over the surface. This attraction can be enhanced by adding a third magnet. When magnets are mounted around a pole, this combination of pulling and pushing can result in a rotating motion. See figure 11 of electric motor, a magnet on the shaft attached and repelled simultaneously by another magnet mounted on the other side. As the shaft rotates due to pulling, the shaft magnet flips the polarity and starts pushing away. Flow of electricity creates a magnetic field so instead of physically flipping over the magnet to change from attract to repel, the flow of electricity can do this mechanism of changing states. [4, 223.]

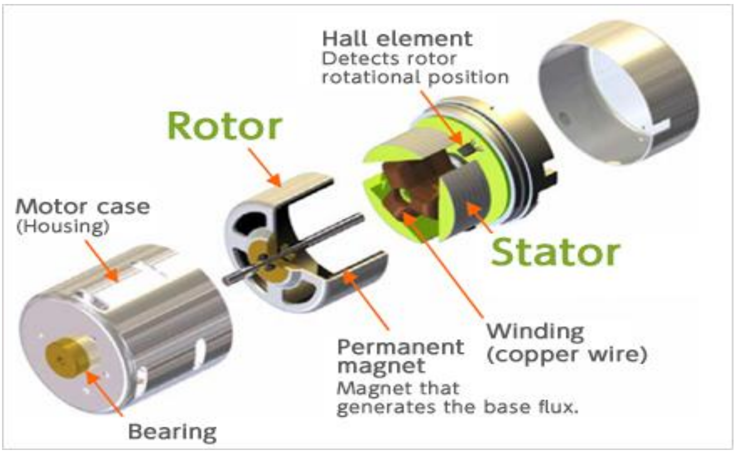


Figure 11. Parts of DC motor

2.4.2 Stator

Figure12 is showing the parts of the classic stator, which includes two permanent magnets located opposite to each other in a metal can. The term permanent magnet means that the magnets remain magnetized even when electricity is turned off. Due to the flow of electricity, magnetic field comes into existence and will push and pull against these two permanent magnets.

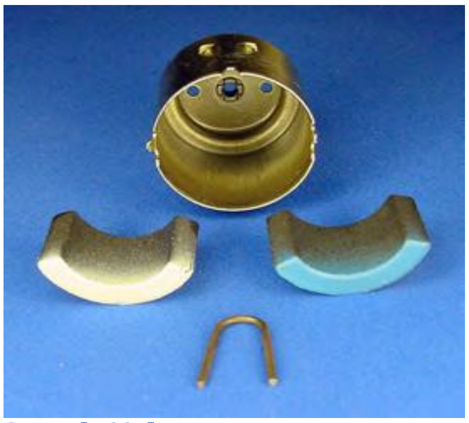


Figure 12. Stator [4,224]

During high temperatures, permanent magnets tend to lose their magnetic field, resulting in sloppy performance or can move to a complete failure. In order to avoid this, the preventive measures such as adequate ventilation and installing the motor body against other metal objects, will allow the extra heat to escape. Magnetic field leakage can be reduced in nearby components by the metal container that makes up the body of motor that acts as the return path for the magnetic field. [4, 224-225.]

2.4.3 Rotor

Rotor is the rotating part of electric motor, built around a shaft. The shaft comes out at the end of motor body so that other components such as wheels, belts, fan blades, or gears can be attached to it. For limiting the friction, only a small part of the rotor touches the motor body. Various sizes of high quality motors often includes ball bearing at the concerned location, which improves carrying strength and reduces friction. [4, 225.]

2.4.4 Controlling Speed

Speed of DC motor can be controlled by using a technique known as Pulse Width Modulation (PWM). When (PWM) is used, power is not supplied in continuous manner, it is provided as a square wave form with a frequency ranges from around 60 Hz - 50 KHz . By changing the pulse width, power supply to the motor can be adjusted. [6, 14.]

The pictorial representation of (PWM) is given below in figure 13.

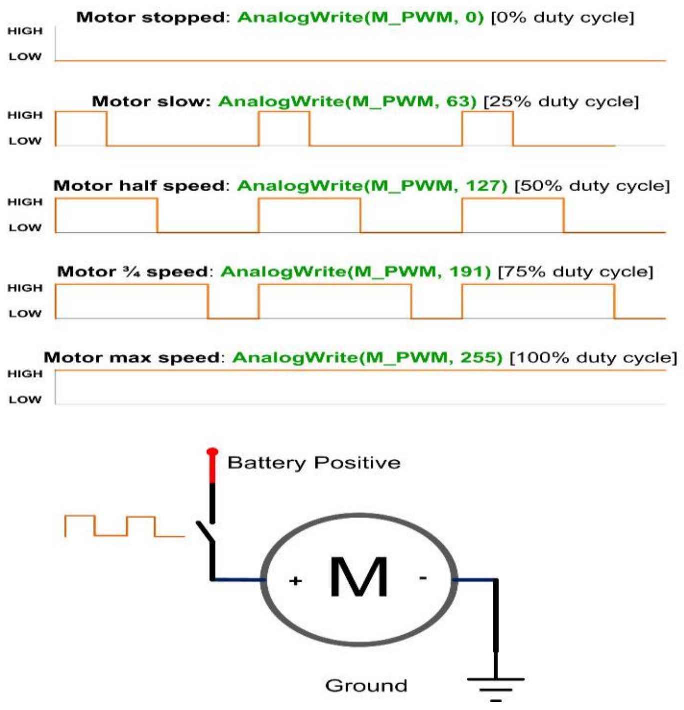

Motor stopped: AnalogWrite(M_PWM, 0) [0% duty cycle]
HIGH
LOW
Motor slow: AnalogWrite(M_PWM, 63) [25% duty cycle]
HIGH
LOW
Motor half speed: AnalogWrite(M_PWM, 127) [50% duty cycle]
HIGH
LOW
Motor ¾ speed: AnalogWrite(M_PWM, 191) [75% duty cycle]
HIGH
LOW
Motor max speed: AnalogWrite(M_PWM, 255) [100% duty cycle]
HIGH
LOW
Battery Positive
+
M
-
Ground

Figure 13. Controlling motor by using pulse width modulation [7,109]

If the duty cycle is at 0 percent, the motor is at rest. But by changing it to 100 percent, motor starts to rotate. The percentage of the (PWM) depends on the application in hand. With microcontroller/microprocessor, generation of PWM signals are easier. [6,15.]

2.4.5 H-Bridge

It is a circuit made of transistors, which is used to control the direction and for applying brakes to motors. In figure 14, working of H-bridge is shown clearly. Transistors of H-bridge act as switches [7,107]. When all the switches are open, mean no flow of current and the motor stays in its idle state.

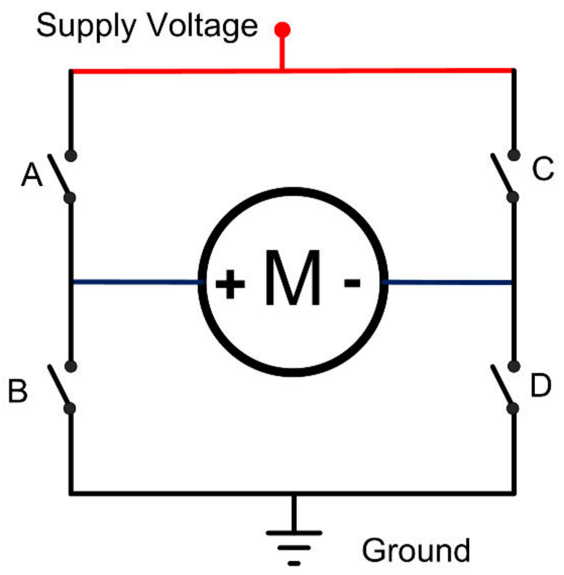


Figure 14. H-Bridge [7,107]

Next figure 15 shows that the two switches are closed, which is causing the motor to go in forward direction.

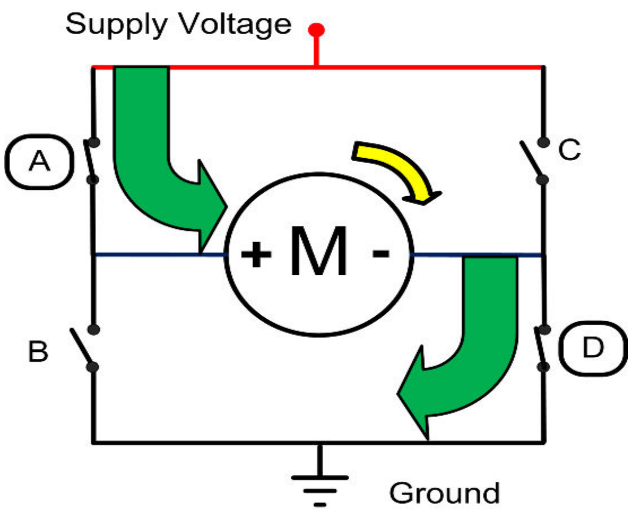


Figure 15. H-Bridge with motor running forward [7,107]

As can be seen in above given figure 15, switch A is connecting the positive terminal to positive side of battery. Switch D is connecting negative terminal of motor to the ground.

Figure 16 below shows the reverse direction of motor. By using switch C, which is connected to negative side of the motor and switch D, which is connected to negative terminal of motor, makes the motor to run in reverse.

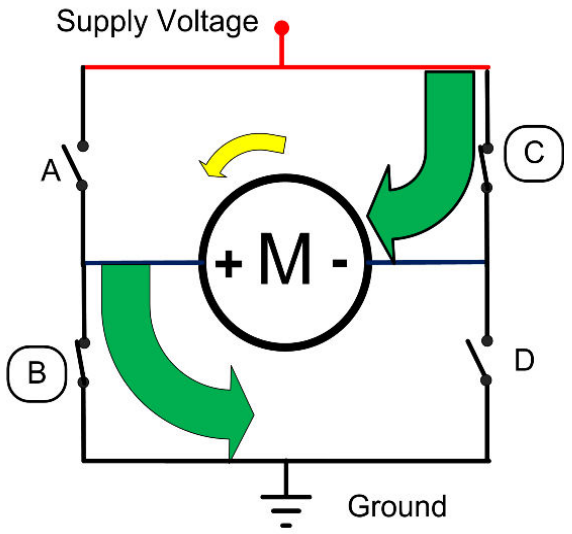


Figure 16. H-Bridge with motor running in reverse [7,108]

In figure 17 given below, it can be seen that switches B and D are in closed position. Connections of the motors are also given in figure 17. When this mode is applied, with the help H-bridge, the motor stops rotating.

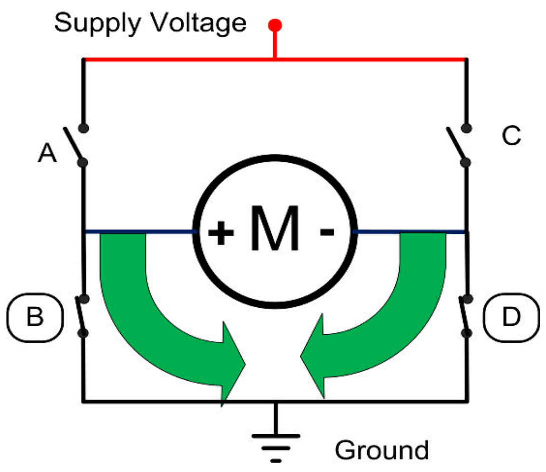


Figure 17. H-Bridge with motor brake [7,108]

2.5 Automatic Speech Recognition

In the domain of computer science and electrical engineering, speech recognition (SR) is the translation of spoken words into text. This is also known as **automatic speech recognition (ASR)** or computer speech recognition (CSR). In today's environment where computers are dominating in every domain of our life, input devices such as keyboard, stylus and small screen are disruptive. Speech dependent user interface is becoming a choice for interaction. **Speech recognition (SR)** is the promising technology and interest of deploying this technology is growing in user devices such as mobiles, tablets and gaming equipment. [5, 2.]

Figure 18 shows the architecture of automatic speech recognition, which includes the components such as speech signal capturing, front end feature extraction, and in the back end speech decoder. In first stage feature vectors are extracted from the speech signals and then it is transferred to the ASR decoder. The function of the decoder is search for the most likely word sequence that matches the feature vectors on the basis of three models. [5, 4.] 1. The acoustic model 2. Lexicon 3. Language model

The difference between the ASR components is well defined and sharp. Speech is always captured in the user device and the application can be either located in the user device or in the server. The decision on where to keep the rest of the ASR components, leads to three approaches given below.

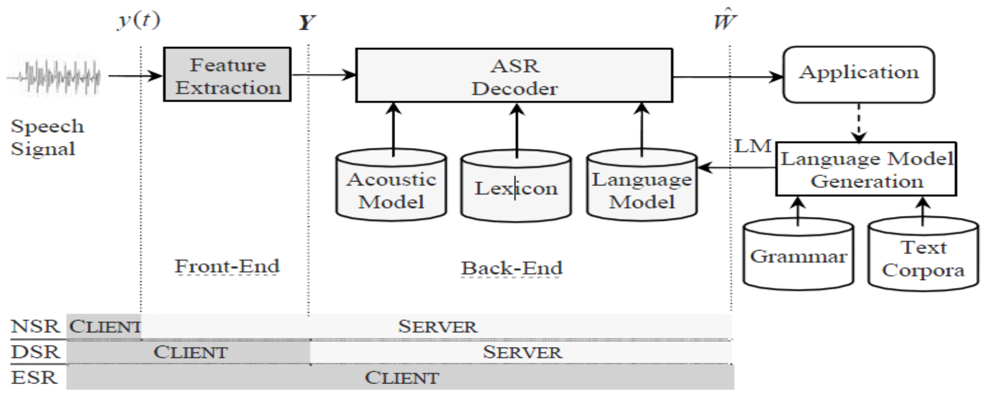


Figure 18. Architecture of an ASR system [5,4]

1. Network speech recognition (NSR)
2. Distributed speech recognition (DSR)
3. Embedded speech recognition (ESR)

In this report, emphasis is upon ESR because of the topic of this report.

2.5.1 Architecture of ESR

The system architecture of ESR is considered to be a simplest approach for implementing speech recognition. It was difficult to run the complex algorithm of (ASR) on a comparatively low resourced user device in terms of processor speed and memory size. Due to rapid advancement in semiconductor technology, computing power has increased dramatically. These advancement are narrowing the gap for (ASR) based applications to be run on user devices. [5, 16.]

2.5.2 Applications of ESR

User devices such as mobile phones, gaming controllers and car kits are very attractive for the implementation of speech recognition. Examples of applications within the car environment as follows, continuous digit and name dialling with hands-free kits, command and control for menus and navigation systems. Speech recognition applications in mobile phone implement speaker dependent (trained) name dialling, in which the user has to train the command before implementation. [5, 16.]

ESR may be implemented by using a general purpose microprocessor/microcontroller located in the user devices or can take advantage of specialised IC to run speech recognition exclusively. [5, 17.]

2.5.3 Fixed –Point- Arithmetic

For low cost and for low power usage, the implementation of fixed-point processor is an important element for consumer devices. If a general purpose processor or a specifically designed IC applied for ESR applications influences the techniques of optimisation which also embody software and hardware design. [5, 17.]

An easy way to make ASR software is by using the programming languages such as C or C++ in floating-point so that to have a reference code. After floating-point the conversion should be done to fixed point, which serves as a foundation for assembler implementation on the processor CPU.

3 System Design

3.1 Overview

In these sections, Major component of the system developed for the FYP is discussed. This also includes the working of components and how they are integrated to get the desired result. Details are given for the programming environment of Arduino microcontroller and EasyVR.

3.2 Arduino Microcontroller Board

Arduino is small computer that can be programmed to process inputs and outputs between the device and external components that can be attached to it. Arduino, also known as embedded computing platforms, means that it is an interactive system that can be used to interact with the environment with the help of hardware and software.
Arduino board has 14 digital input/output pins; out of them 6 can be used as PWM outputs, 6 analog inputs. Moreover Arduino board has a 16 MHz ceramic resonator, USB

connector, power jack, ICSP header, and a reset button. The board itself contains all the features needed to support the microcontroller. Simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.
For example we can attach a led with the Arduino and programme it to blink for 30 seconds. We can enhance this concept to by connecting a temperature sensor, to turn on the led, when temperature reaches to 30 C.

Arduino can be connected to computer, a network or even to the internet to retrieve and send data to from the Arduino and then act on that data. In other words, it can send the data to website after getting the data from sensors connected to it.

Numerous components can be attached to Arduino such as Led's, dot matrix displays, buttons, switches, temperature sensors, pressure sensors, distance sensors, GPS receivers, and Ethernet modules. A simple search on the internet will bring up a wealth of projects where Arduino has been used to read data from or control the vast array of devices.

3.2.1 Connecting to PC

Connecting Arduino Board to windows based PC (personal computer) is extremely easy. Figure 19 shows the hardware structure of Arduino UNO board.

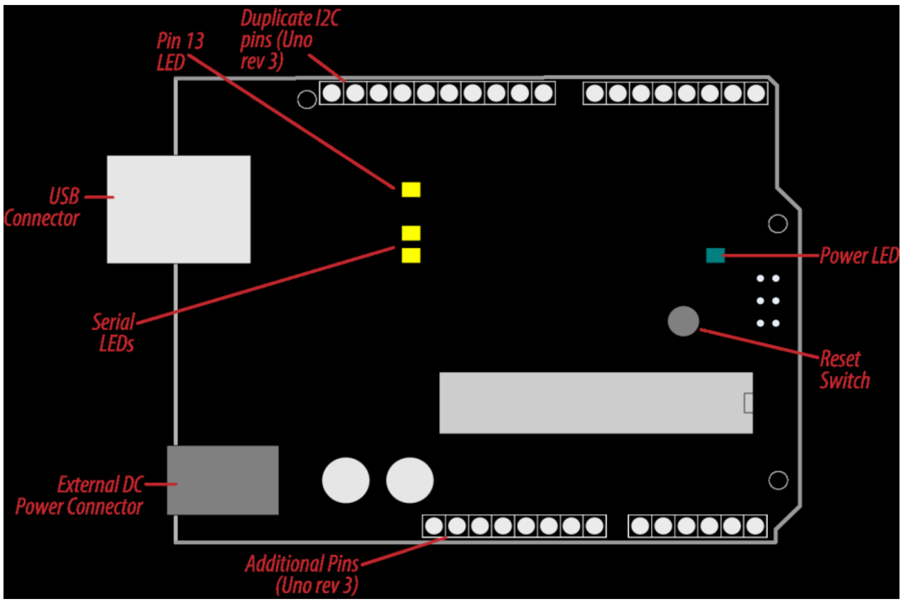

Pin 13 LED
Duplicate I2C pins (Uno rev 3)
USB Connector
Power LED
Serial LEDs
Reset Switch
External DC Power Connector
Additional Pins (Uno rev 3)

Figure 19. Basic Arduino board [8,8]

Universal Serial Bus (USB) cable is required to plug the connection of Arduino with the computer. Figure 18 shows the USB connector, just plug the USB cable there and the other part of USB cable to computer. An orange LED should start blink after the connection with computer.

3.2.2 Integrated Development Environment

For the development of software for Arduino, an integrated devolvement environment is available. IDE runs on different platforms such Windows and Linux. Arduino IDE is helpful in creating, modifying the sketches that define what the board will do after uploading the sketch. Programme written using Arduino is called a sketch. Figure 20 shows the IDE editor where sketch is prepared by using C language.

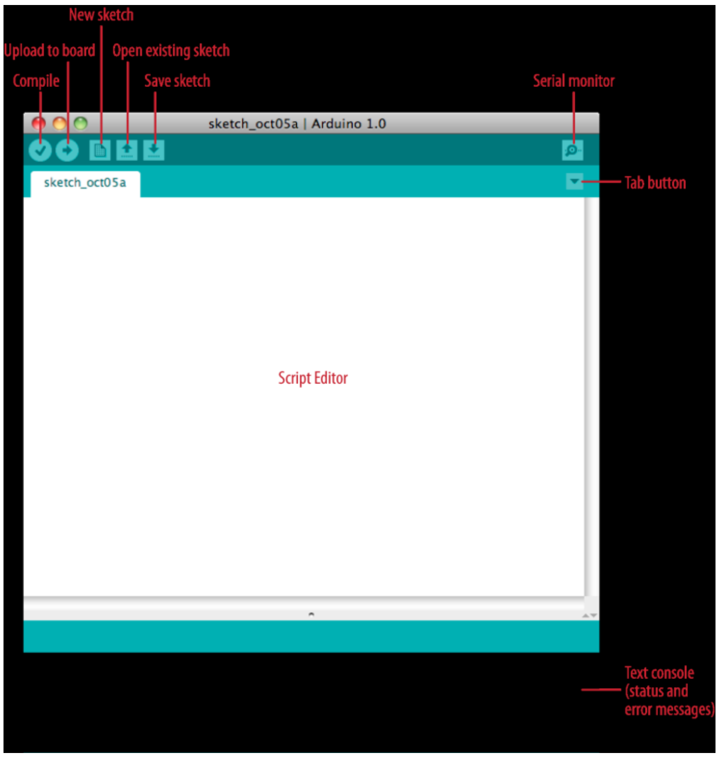


Figure 20. IDE environment [8,11]

Buttons are shown in the above figure 20 clearly showing the showing the function of each button. After writing the sketch, compile button should be pressed and errors will appear in the text console, if any. Upload the sketch after successful compilation and desired results can be seen on the board.

3.3 Connecting DC motors to Arduino

During the construction of project connection between Arduino and motors were made by using a motor Controller shield, whose design is based on H-bridge. Theoretical foundation has been laid earlier for H-bridge. Figure 21 shows motor controller shield.

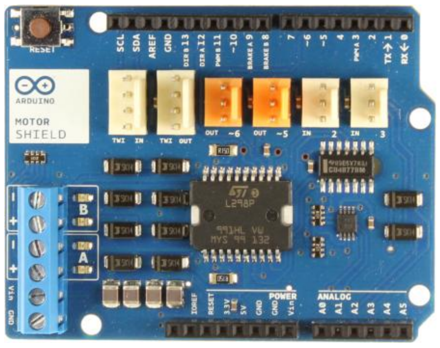


Figure 21. Arduino Motor Shield R3 Front

Motor shield can be very easily inserted in the pins of Arduino board. Wires from DC motors goes to the blue connectors, can be seen in above figure 21.

3.3.1 Coding for Motor Controller

IDE was used for writing the source code for motor controller. Figure 21 above shows the pins used by motor shield. Pins13 and12 are used for the change of direction of motors. Brakes are applied to motors by using pins 8 and 9. PWM signals are generated by the pins 3 and 11, which help in controlling the speed of motors. In the IDE, above given pins are illustrated in listing 1.

Listing 1. Pins initialization in IDE

```
const int // Type of variable
// For back Motor
PWM_A   = 3, // For PWM signals at pin 3
DIR_A   = 12,// For setting the direction of motor at pin 12
BRAKE_A = 9, // For applying brake
SNS_A   = A0,// For current sensing

// For front motor
PWM_B   = 11, //For PWM signals at pin 11
```

```
DIR_B   = 13, //For setting the direction of motor at pin 13
BRAKE_B = 8, // For applying brake
SNS_B   = A1; // For current sensing
```

Listing 2 shows the coding done in IDE for handling the motors with remarks. This will help a reader to understand and modify the code for their needs.

Listing 2. IDE commands for controlling motors

```
digitalWrite(BRAKE_A, LOW);// setting brake LOW for brake

digitalWrite(DIR_A, HIGH); // setting direction to HIGH the

analogWrite(PWM_A, 130);// Set the speed of the motor, 255 is
the maximum value

delay(5000);// Keeps the motor at full speed for 5 seconds
```

Commands provided in listing 2 are used in FYP along with the code for EasyVR module. Next sections provide details on how to use the EasyVR module for speech recognition and the use of EasyVR commander software for the generation of code. In appendix 1 complete code is provided for the reader.

3.4 EasyVR Module

The module EasyVR module shown in figure 22 was used for this FYP. The manual of the EasyVR module provides good introduction, product specifications, connection modes and how to programme it. EasyVR is designed to provide cost effective speech recognition capabilities to a variety of applications. EasyVR module can be used with any device with an UART interface, which can be powered at 3.3V - 5V. In this Arduino or PIC boards are good candidates. Examples where EasyVR can be used are home automation projects, such as voice controlled switches, locks, curtains or kitchen appliances.

Figure 22. EasyVR module

3.4.1 Technical Specification

Figure 23 given below shows the physical dimension and pin assignment of an EasyVR module.

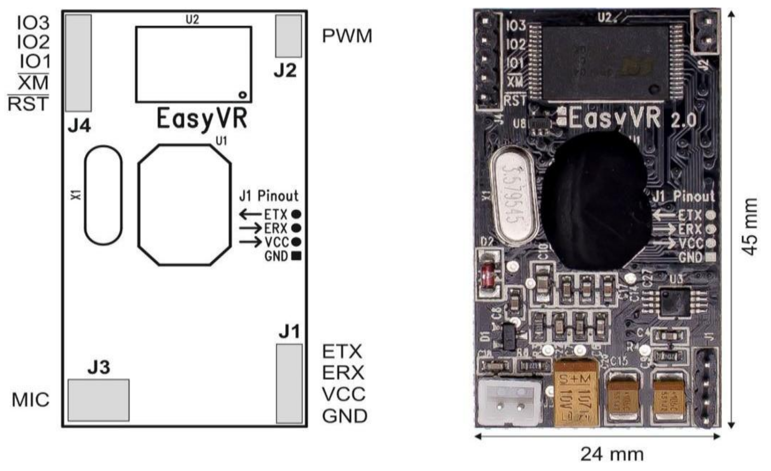


Figure 23. EasyVR connectors

Table 1 shows the description of the connectors, type of data involved.

Table 1. Showing the functions of EasyVR connectors

Connector	Number	Name	Type	Description
J1	1	GND	-	Ground
	2	VCC	I	Voltage DC input
	3	ERX	I	Serial Data Receive (TTL level)
	4	ETX	O	Serial Data Transmit (TTL level)
J2	1-2	PWM	O	Differential audio output (can directly drive 8Ω speaker)
J3	1	MIC_RET	-	Microphone reference ground
	2	MIC_IN	I	Microphone input signal
J4	1	/RST	I	Active low asynchronous reset (internal 100K pull-up)
	2	/XM	I	Boot select (internal 1K pull-down)
	3	IO1	I/O	General purpose I/O (**3.0 VDC** TTL level)
	4	IO2	I/O	General purpose I/O (**3.0 VDC** TTL level)
	5	IO3	I/O	General purpose I/O (**3.0 VDC** TTL level)

3.4.2 Serial Interface

In embedded system EasyVR module works as a salve and communicates by using asynchronous serial interface (UART). Baud Rate, by default is 9600, or other baud rates are 19200, 38700, 57600, 115200. Frame consists of 8 Data bits, No parity, 1 Stop bit ERX is used as input date line by receiver and ETX is used as data output line by the transmitter.

3.4.3 Microphone

EasyVR module has a microphone attached to it, which is an unidirectional electrets condenser microphone (Horn EM9745P-382). Following are the features of micro-phone.

1) It has a sensitivity of – 38 dB
2) Load Impedance is 2.2K
3) Operating Voltage is 3V
4) Frequency response in the range of 100HZ- 20 KHz

Other kinds of microphones are not supported by EasyVR module. It is important to remember that the vocal commands should be given at about 60 cm from microphone.

3.5 EasyVR development Board

EasyVR module also has development board called as DevBoard. This can used to programme commands and sounds into an EasyVR module. Before doing programming for the EasyVR module. It should be checked that the 8 ohms speaker is connected, microphone is connected to its proper connector and the EasyVR module is placed above the DevBoard. Devolvement board consists of a Freescale JS8 microcontroller, which is programmed as a USB-Serial adapter to convert data sent between computer and EasyVR. Test programme can be written on user PC by using the serial port as could be done on a microcontroller. It provides a possibility to upload new firmware and sound-tables. Jumper settings are not required because DevBoard manage it automatically.

3.5.1 Hardware

Let's have a look at the hardware interfaces of DevBoard. It consists of 3 connectors. J1 connector is a screw terminals block for audio output and an 8 Ω speaker is required. J6 connector provides the audio input interface for connecting the microphone which comes with module. Connector J5 on the DevKit board gives the copy of general purpose input/output on the EasyVR as given in the table 2 below.

Table 2. Description of pins and connector

Connector	Pin	Name	Type	Description
J5 **(DevKit)**	**1**	GND	-	Ground
	2	IO1	I/O	General purpose I/O (**3.0 VDC** TTL level)
	3	IO2	I/O	General purpose I/O (**3.0 VDC** TTL level)
	4	IO3	I/O	General purpose I/O (**3.0 VDC** TTL level)

3.6 Connecting EasyVR to Arduino

EasyVR can be connected in two ways to an Arduino microcontroller board.

3.6.1 Bridge Mode

According to the manual that bridge mode is preferable as this allows the communication with both the Arduino board and the computer. Figure 24 given below is showing bridge mode. During construction of system, bridge mode was used.

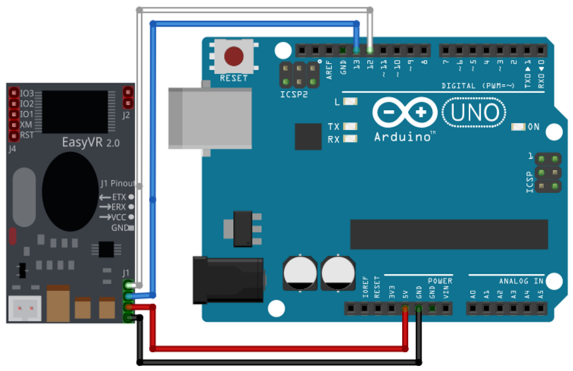


Figure 24. Connection to arduino by using bridge mode

3.6.2 Adapter Mode

This mode has the advantage, if used with any Arduino board that has an on-board USB/Serial adapter. Sound Table can be downloaded to the EasyVR module by using this mode. Figure 25 shows clearly the connection between Arduino and EasyVR by using adapter mode.

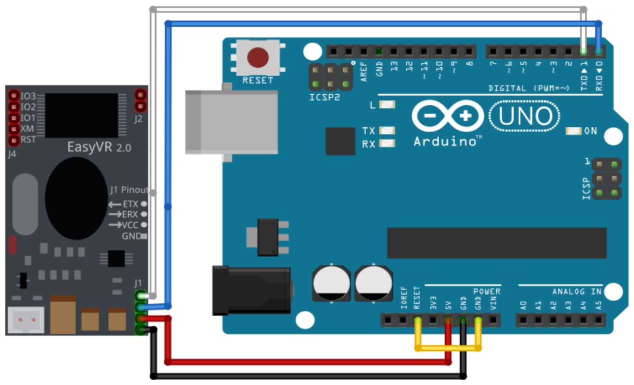


Figure 25. Connection to arduino by using adapter mode

3.7 EasyVR Commander

EasyVR commander software allows the users to configure the module connected to Arduino board in a bridge mode. In software we can define groups of commands or passwords. This software automatically generates the code and contains all the functions to handle the speech recognition tasks.

3.7.1 Coding for EasyVR Module

EasyVR commander was used to for speech recognition commands. Bridge mode was used for the connection with Arduino. In figure 24 of bridge mode, blue wire goes to pin13 and white wire goes to pin12 of Arduino. Figure 26 shows the main application window of EasyVR commander. There are four kinds of commands in the software.

1. Trigger

This is a special group, where user use built-in word “Robot” or can add user-defined trigger word. Trigger words are used to start the recognition process.

2. Group

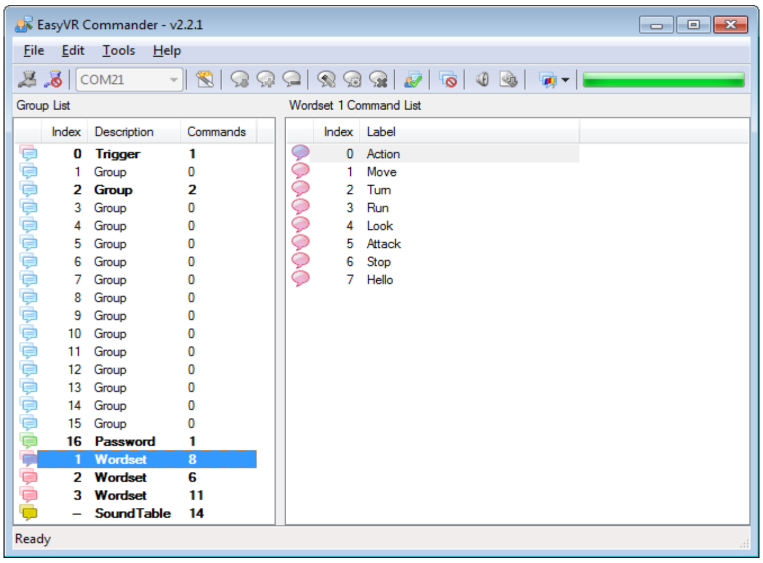
User can define their own command words. In this project, following words are defined for the purpose of FYP. The words are 1. Forward 2. Back 3. Right 4. Left

3. Password

This is a special group for “vocal passwords” which can be used for up to five words by using Speaker Verification (SV) technology.

4. Word set

Word set consists of in-built commands. Figure 26 shows eight commands in the first word set.



Group List

	Index	Description	Commands
	0	Trigger	1
	1	Group	0
	2	Group	2
	3	Group	0
	4	Group	0
	5	Group	0
	6	Group	0
	7	Group	0
	8	Group	0
	9	Group	0
	10	Group	0
	11	Group	0
	12	Group	0
	13	Group	0
	14	Group	0
	15	Group	0
	16	Password	1
	1	Wordset	8
	2	Wordset	6
	3	Wordset	11
	–	SoundTable	14

Wordset 1 Command List

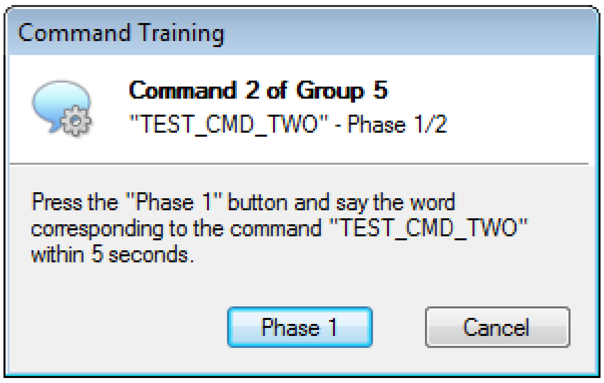
	Index	Label
	0	Action
	1	Move
	2	Turn
	3	Run
	4	Look
	5	Attack
	6	Stop
	7	Hello

Ready

Figure 26. EasyVR commander software

Speech recognition function of the EasyVR works on a single group at a time. That is why during working on FYP all the required commands were placed in Group 1.
When EasyVR commander connects to the module, it read back all the user-defined commands and groups, which EasyVR module stores in the non-volatile memory.

A new command can be added by first selecting the group in which the commands need to be created and then using the toolbar icon. A command should be given a label and then it should be trained twice with the user´s voice. The user will be guided throughout the process. Figure 27 shows the command training process.


Command Training

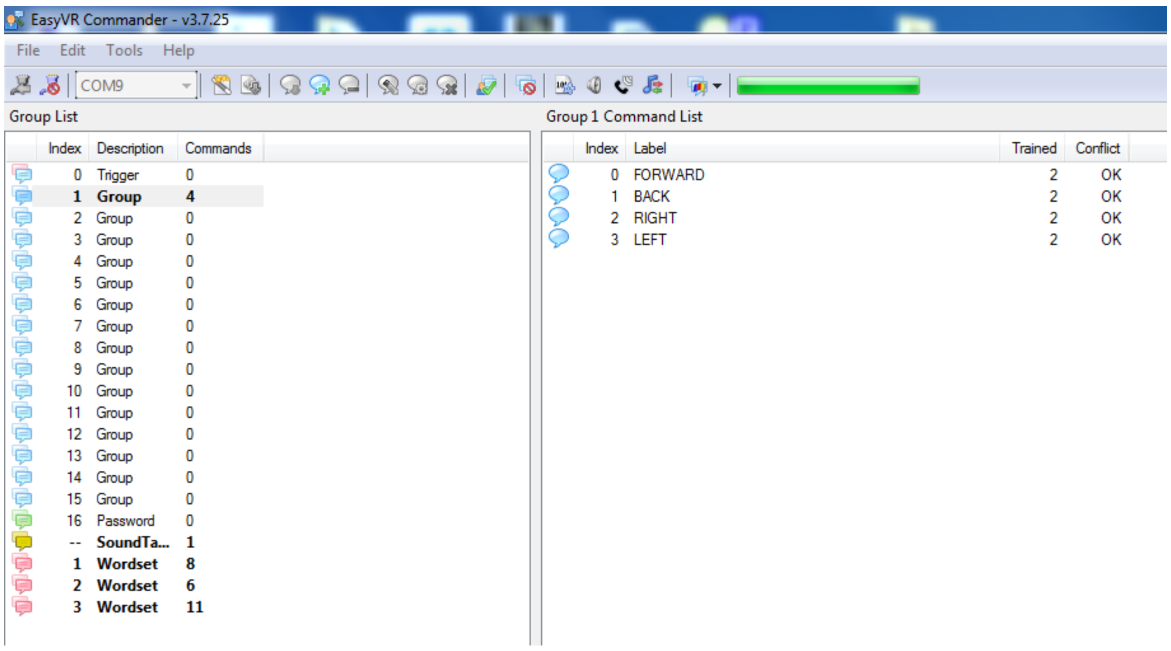
Command 2 of Group 5
"TEST_CMD_TWO" - Phase 1/2

Press the "Phase 1" button and say the word corresponding to the command "TEST_CMD_TWO" within 5 seconds.

Figure 27. Guided training dialog

If any error message appears, command training will be cancelled. Errors may happen when the user´s voice is not recognized correctly, there is too much background noise or when the second word heard is different from the first one.

For this FYP, commands were defined and trained in the Group 1 as can be seen in figure 28.



Group List

Index	Description	Commands
0	Trigger	0
1	**Group**	**4**
2	Group	0
3	Group	0
4	Group	0
5	Group	0
6	Group	0
7	Group	0
8	Group	0
9	Group	0
10	Group	0
11	Group	0
12	Group	0
13	Group	0
14	Group	0
15	Group	0
16	Password	0
--	**SoundTa...**	**1**
1	**Wordset**	**8**
2	**Wordset**	**6**
3	**Wordset**	**11**

Group 1 Command List

Index	Label	Trained	Conflict
0	FORWARD	2	OK
1	BACK	2	OK
2	RIGHT	2	OK
3	LEFT	2	OK

Figure 28. Commands for FYP

3.7.2 Modification of Code

Listing 3 shows the part of code generated by EasyVR commander for this FYP after defining and training of commands. The default pins for connecting EasyVR to Arduino are pin12 and pin13, which can be seen in the figure 24. They were changed to pin2 and pin4 because of motor shield. Motor shield use pin13 and pin12 for changing the direction of motor. PWM signals are provided by pin11 and pin3 to control the speed of motors. Brakes are applied to DC motors are done by utilizing pin8 and pin9.

Listing 3. Modification in Auto-generated code

```
#if defined(ARDUINO) && ARDUINO >= 100
  #include "Arduino.h"
  #include "SoftwareSerial.h"
  SoftwareSerial port(2,4); // Here the changes were made for
the pins
#else // Arduino 0022 - use modified NewSoftSerial
```

```
  #include "WProgram.h"
  #include "SoftwareSerial.h"
NewSoftSerial port(2,4);// Changes made in pins to be used by
EasyVR
#endif
```

Listing 4 shows the part of code where change has been made. This change was crucial for the recognition of speech recognition commands.

Listing 4. Minor change in setup code

```
group =1; // this line was introduced to work with speech com-
mands

group = EasyVR::TRIGGER; // this part of excluded from the code
of interference with the speech command written for FYP
```

3.8 Integrating the code

The code was for motor shield and EasyVR module was integrated which is provided for the reader in the appendix 1. Listing 5 shows the part of code, this was integrated with EasyVR speech commands. Remarks are given in front of codes for the reader to understand the function of code.

Listing 5. Integrated code.

```
void action()
{
    switch (group)
    {
    case GROUP_1:
      switch (idx)
      {
      case G1_FORWARD:
digitalWrite(BRAKE_A, LOW);  // setting brake LOW disable motor
brake
```

```
digitalWrite(DIR_A, HIGH); // for change in direction

analogWrite(PWM_A, 100);  // for setting speed of the motor, 255
is the maximum value

delay(3000); // time delay of 3 seconds

analogWrite(PWM_A, 0);//after time delay motor will turned off

break;
```

When the system is initialized, it wait for the commands and after recognition of commands, the system perform accordingly. For example if user give speech command forward as listing 5 shows, pin12 will be set to HIGH and pin3 which is for PWM signal will be set to 100. The RC car will start moving in the forward direction and will stop after 3 seconds and can be seen in listing that PWM value sets to 0. Appendix 1 has all the code prepared for the system.

4 System Outlook

4.1 Overview

In section 3, the components which are used for system design are not only discussed but also shown how the components are combined to build the project. Here in this section, description is given about the aftermath of system integration. Moreover, future enhancements and possible constraints are discussed.

4.2 Speech Controlled RC Car

In the figure 29, speech controlled car is illustrated. It can be seen from the figure 29 that the cover from the small RC car is removed and integrated hardware components of the system are placed on the skeleton of the car. Due to this placement of components, there is an increase in weight on motors. In practical operation of this system it was noted that motors are running slower as compared to operation of motors before placement of components.

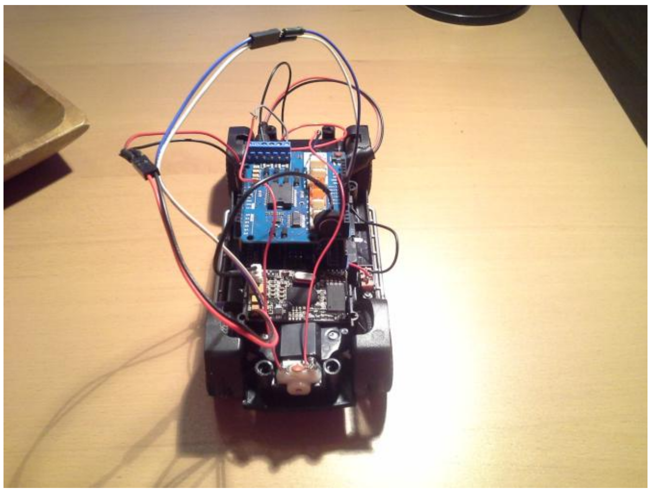


Figure 29. Speech controlled RC car

4.3 Limitations and Future Enhancements

No system is perfect and it also goes with the system which is developed in this FYP. Following are the limitations involved.

1. Due to small size of the RC car, there is a strain on motors, when hardware components are placed over it.

2. Noise robustness is an important issue because the acoustic environment in system usage is very different from the lab. EasyVR module worked very well in lab environment but faced issues in speech recognition while using it outdoors.

3. A 9V battery was depleted after few trials.

4. Financial aspects was a constraint for buying more components

Future enhancements should be based on the following suggestions.

1. RC car with big motors and more space to place the hardware components above it.

2. Another speech module should be considered, which can recognise speech commands in relatively noisy environment.

3. Perhaps lipoly battery should be used for better performance.

5 Conclusion

This project provided good experience of the complexity involved in designing embedded system. This project provided a chance to apply the theory learned in the classroom. It is good to realize that memorizing the theory and passing exams are not enough. In real life, uncertainties are always involved. One event of uncertainty occurred while dealing with EasyVR module, which is not given in the manual is how to make a change in the main programme for the speech commands to be recognized. Solution was discovered with the method of trial and error or in other words learning from failures.

The basic aim of the FYP to build the embedded system based on the hardware and programming was achieved and testing was done in the presence of the advisor, Janne Mäntykoski. Hopefully future enhancements will be made based on the recommendations given above.

References

1. Kamal Raj. Embedded Systems Second Edition. New Delhi: Tata McGraw-Hill Publishing Company Limited; 2008

2. Alciatore David G, Histand Michael B. Introduction to Mechatronics and Measurement Systems Fourth Edition. United States of America: McGraw-Hill Companies, Inc; 2012

3. Noergaard Tammy. Embedded Systems Architecture, A Comprehensive Guide for Engineers and Programmers. United States of America: Elsevier Inc; 2005

4. Cook David. Robot Building for Beginners second edition. United States of America: Springer-Verlag New York, Inc; 2009

5. Zheng-Hua Tan, Børge Lindberg .Automatic Speech Recognition on Mobile Communication Networks. London: Springer-Verlag London Limited; 2008

6. Clark Dennis, Owings Michael. Building Robot Drive Trains. United States of America: McGraw-Hill Companies, Inc; 2003

7. Margolis Michael, Make an Arduino-Controlled Robot. United States of America: O'Reilly Media, Inc; 2013

8. Margolis Michael**,** Arduino Cook Book Second Edition, United States of America: O'Reilly Media, Inc; 2012

Source code for FYP

```
#if defined(ARDUINO) && ARDUINO >= 100
  #include "Arduino.h"
  #include "SoftwareSerial.h"
  SoftwareSerial port(2,4);
#else // Arduino 0022 - use modified NewSoftSerial
  #include "WProgram.h"
  #include "SoftwareSerial.h"
  NewSoftSerial port(2,4);
#endif
#include "EasyVR.h"
EasyVR easyvr(port);
//Groups and Commands
enum Groups
{
  GROUP_1  = 1,
};

enum Group1
{
  G1_FORWARD = 0,
  G1_BACK = 1,
  G1_RIGHT = 2,
  G1_LEFT = 3,
};


EasyVRBridge bridge;

int8_t group, idx;

const int

// For back Motor
PWM_A   = 3,
DIR_A   = 12,
BRAKE_A = 9,
SNS_A   = A0,

// for front motor
PWM_B   = 11,
DIR_B   = 13,
BRAKE_B = 8,
SNS_B   = A1;

void setup()
{
  // bridge mode?
  if (bridge.check())
  {
    cli();
    bridge.loop(0, 1, 12, 13);
  }
  // run normally
  Serial.begin(9600);
  port.begin(9600);

  if (!easyvr.detect())
  {
    Serial.println("EasyVR not detected!");
    for (;;);
  }

  easyvr.setPinOutput(EasyVR::IO1, LOW);
  Serial.println("EasyVR detected!");
  easyvr.setTimeout(5);
  easyvr.setLanguage(0);
  group =1;
  //group = EasyVR::TRIGGER; //<-- start group (customize)
}

void action();

void loop()
{
  easyvr.setPinOutput(EasyVR::IO1, HIGH); // LED on (listening)

  Serial.print("Say a command in Group ");
  Serial.println(group);
  easyvr.recognizeCommand(group);

  do
  {
    // can do some processing while waiting for a spoken command
  }
  while (!easyvr.hasFinished());

  easyvr.setPinOutput(EasyVR::IO1, LOW); // LED off

  idx = easyvr.getWord();
  if (idx >= 0)
  {
    // built-in trigger (ROBOT)
    // group = GROUP_X; <-- jump to another group X
    return;
  }
  idx = easyvr.getCommand();
  if (idx >= 0)
  {
    // print debug message
    uint8_t train = 0;
    char name[32];
    Serial.print("Command: ");
    Serial.print(idx);
    if (easyvr.dumpCommand(group, idx, name, train))
    {
      Serial.print(" = ");
      Serial.println(name);
    }
    else
      Serial.println();
    easyvr.playSound(0, EasyVR::VOL_FULL);
    // perform some action
    action();
  }
  else // errors or timeout
  {
    if (easyvr.isTimeout())
      Serial.println("Timed out, try again...");
    int16_t err = easyvr.getError();
    if (err >= 0)
    {
      Serial.print("Error ");
      Serial.println(err, HEX);
    }
  }
}

void action()
{
    switch (group)
    {
    case GROUP_1:
      switch (idx)
      {
      case G1_FORWARD:
        digitalWrite(BRAKE_A, LOW);  // setting brake LOW disable motor brake
        digitalWrite(DIR_A, HIGH); // for direction
        analogWrite(PWM_A, 100);      // Set the speed of the motor, 255 is the maximum
value
        delay(3000); // time
        analogWrite(PWM_A, 0);

        // write your action code here
        // group = GROUP_X; <-- or jump to another group X for composite commands
        break;
      case G1_BACK:


         digitalWrite(BRAKE_A, LOW);  // setting again the brake LOW to disable motor
brake
         digitalWrite(DIR_A, LOW);      // now change the direction to backward setting
LOW the DIR_A pin
         analogWrite(PWM_A, 100);
         delay(3000);
         analogWrite(PWM_A, 0);
        // write your action code here
        // group = GROUP_X; <-- or jump to another group X for composite commands
        break;
      case G1_RIGHT:

        digitalWrite(BRAKE_B, LOW);  // setting brake LOW disable motor brake
        digitalWrite(DIR_B, HIGH);
        digitalWrite(BRAKE_A, LOW);  // setting brake LOW disable motor brake
        digitalWrite(DIR_A, HIGH);
 analogWrite(PWM_A, 100); //By changing the value from 0-255, speed can be ad-
justed of motor B

        analogWrite(PWM_B, 255);     // By changing the value from 0-255, speed can be
adjusted of motor B
        delay(3000);
        analogWrite(PWM_A, 0);
        analogWrite(PWM_B, 0);
        // write your action code here
        // group = GROUP_X; <-- or jump to another group X for composite commands
        break;
      case G1_LEFT:
      digitalWrite(BRAKE_A, LOW);  // To disable motor brake
      digitalWrite(DIR_A, HIGH);
      digitalWrite(BRAKE_B, LOW);  // To disable motor brake
      digitalWrite(DIR_B, LOW);
      analogWrite(PWM_A, 100);
      analogWrite(PWM_B, 255);

       delay(3000);
       analogWrite(PWM_A, 0);
        analogWrite(PWM_B, 0);
        // write your action code here
        // group = GROUP_X; <-- or jump to another group X for composite commands
        break;
      }
      break;
    }
}
```