DOUBLE DEGREE PROGRAM IN

INFORMATION AND COMMUNICATION TECHNOLOGIES

ENGINEERING

GRADUATION PROJECT REPORT

BRAIN COMPUTER INTERFACE SYSTEM

Advisers

Prof. Amr El-said

Prof. Claudio Fornaro

Students

Mostafa Ahmed Yousef

Mostafa Ezz EL-din Mohamed

Academic year 2011/12

Acknowledgment

First of all,We thank our parents, for advising, praying and supporting in every thing in our lives and specially for the 5 years we studying Engineering.
We would like to thank our university Helwan and Uninettuno university for learning us communication engineering and for all professors and doctors.
We deeply thank our Prof. Amr El Said, whose help, advice and supervision was invaluable.
We also thank Prof. Claudio Fornaro, with whom We discussed with him this project and supervision.
Finally, special thanks for Eng. Hazem Khaled, who help us how to program with C# and using Arduino .

Indication and jobs:

Mostafa Ahmed Yousef :

- Write the Project description.
- Preface and History.
- Introduction to BCI.
- Chapter 1 Neruosky.
- Chapter 4 the interface and forms of GUI to the system by C#.

Mostafa Ezz Eldin Mohmed :

- Chapter 2 Arduino.
- Chapter 3 the Implementation of the Robot.
- Chapter 4 the software of Arduino IDE.
- Conclusion and future work.

Table of Contents:

List of Figure

Project Description

The ultimate purpose of a direct brain computer interface (BCI) is to allow an individual with severe motor disabilities to have effective control over devices.
A BCI system detects the presence of specific patterns in a person's ongoing brain activity that relates to the person's intention to initiate control.

In our project we use a robotic car as a prototype for this idea we will control the robotic car by the attention ratio of the Nerousky mindwave as the speed of the robotic car and we use the line detector to make the robotic car in the terminal or the road.
We make all the system full wireless and we use the Bluetooth shield with arduino to connect the hardware (Robotic car) with Neruosky(Bluetooth dongle) and the Laptop.

Figure P-1 proto type of the project

We divide our project to many tasks to reach the final proto type :

- The first task was dealing with the brain signals by using the thinkgear packet and creates the interface of the brain signals (Alpha- beta- gamma- theta-ratio of attention and mediation).
- The second task was using arduino to control the rate of light of LED because it is similar to control the dc motor and we use the attention ration as the rate of the light.
- The third task was building the robotic car and we make a small test to control it by a simple program in arduino to move forward and take a delay and move to the right and take a delay and move to the left then take a delay to move again to forward .
- The fourth task was connecting the Bluetooth shield in the arduino to control it wireless.
- The fifth task was about a small program by the C# to control the robot by form of the direction as the keyboard and the controlling was full wireless.
- The final task was use the first program in the first task to control the speed of the dc motor by divide the speed to 3 parts (small – medium-high).

Summary

The ultimate purpose of a direct brain computer interface (BCI) is to allow an individual with severe motor disabilities to have effective control over devices.
A BCI system detects the presence of specific patterns in a person's ongoing brain activity that relates to the person's intention to initiate control.
In our project we use a mind-controlled robot as a prototype for this idea.

The bot is easy to use. You put on a headband and when you concentrate, the bot moves. Focus more and it goes faster. And it's a real robot too; it avoids edges so that it stays on the table.

The robot part was based on soccer bot from Make: Arduino Bots and Gadgets (O'Reilly, 2011). We read the EEG with a NeuroSky MindWave. The early model had touse a computer as a gateway between Arduino and MindWave, because we were running the Mind Wave software and our own Python program on the computer.

EEG in Your Living Room Control a computer with just your mind. On one hand, it sounds almost like a sci-fi fantasy. On the other, EEG (electroencephalography) was first used in the early 20th century. What kept you waiting for the future?.
EEG is the recording of electrical activity of the brain from the scalp, produced by neurons firing in the brain. The brain cortex produces tiny electrical.

voltages (1–100 μV on the scalp). EEG doesn't read your thoughts, but it can tell your general state. For example, EEG can show if you are paying attention or meditating.The tiny voltages are easily masked by electrical noise from muscles and ambient sources. EEG currents are measured in microvolts (μV), which are millionths of a volt:
1 μV = 0.001 mV = 10^-6 V.

Noise from muscle and eye movement can be quite powerful compared to this. In normal buildings, the electrical main's current radiates a 50Hz or 60Hz electromagnetic field. In a laboratory setting, EEG is usually measured in a room that has less interference. At home, the EEG unit must filter out the troublesome signals.

EEG devices used to be prohibitively expensive and annoying to connect, and the data required expert knowledge to interpret. For many years, a starting price for the cheapest EEG units was thousands of dollars. They required conductive gel to connect. Having very clean hair and skin was recommended. Most units used at least 19 electrodes. EEG results were printed on paper and doctors had to take a course to be able to analyze them.

Now EEGs are cheap, starting from $100 (USD). Devices are available in shops and by mail order. Consumer-level EEG units are manufactured by NeuroSky and Emotiv. (OCZ used to make a similar device.) With the Open- EEG project, you can even build an EEG device yourself.

NeuroSky's units are the cheapest option, starting from $100 for the Mind-Wave. The headband is fast to attach and works on dry skin without any gels. It only needs electrical contact on your forehead and earlobe. NeuroSky devices measure attention and meditation as well as the raw brainwave data.

Emotiv EPOC promises to recognize multiple visualized thoughts. At $300, it's not very expensive. The Emotive EPOC headset also measures head tilt and muscle activity.
OCZ used to make the mOCZ Neural Impulse Actuator which cost about $100. It made multiple measurements, mostly concentrating on muscle activity.

The OpenEEG project provides instructions on how to build your own EEG device. It's the most expensive option, costing about $500. Building the device requires both technical skills and understanding of safety issues. After all, EEG involves connecting wires on different sides of your head.!

You need to know the basics of programming Arduino. You should make sure you can run a 'hello world' or 'blink' example on your Arduino before you try anything else. This means that you should also have the Arduino IDE installed. If you need help with this, see "Starting with Arduino" (page 18) in MABG. For hand-holding walkthrough code examples, see any of the projects in that book. You may also want to look at Massimo Banzi's Getting Started with Arduino (O'Reilly , 2011) if you need a beginner's introduction. However, as a prospective robot builder , you will find the projects in MABG an excellent complement to the one in this book.

You should have basic mechanical building skills. You'll solder wires to a DC Motor and your own connections to MindWave to build the robot platform.

Introduction

Brain Computer Interface (**BCI**): often called a **mind-machine interface** (**MMI**) , is a direct communication pathway between the brain and an external device. BCIs are often directed at assisting, augmenting, or repairing human cognitive or sensory-motor functions.

Research on BCIs began in the 1970s at the University of California Los Angeles (UCLA) under a grant from the National Science Foundiation followed by a contract from DARPA.

The papers published after this research also mark the first appearance of the expression *brain–computer interface* in scientific literature.

The field of BCI research and development has since focused primarily on neuroprosthetics applications that aim at restoring damaged hearing, sight and movement. Thanks to the remarkable cortical plasticity of the brain, signals from implanted prostheses can, after adaptation, be handled by the brain like natural sensor or effector channels. Following years of animal experimentation, the first neuroprosthetic devices implanted in humans appeared in the mid-1990s.

History :

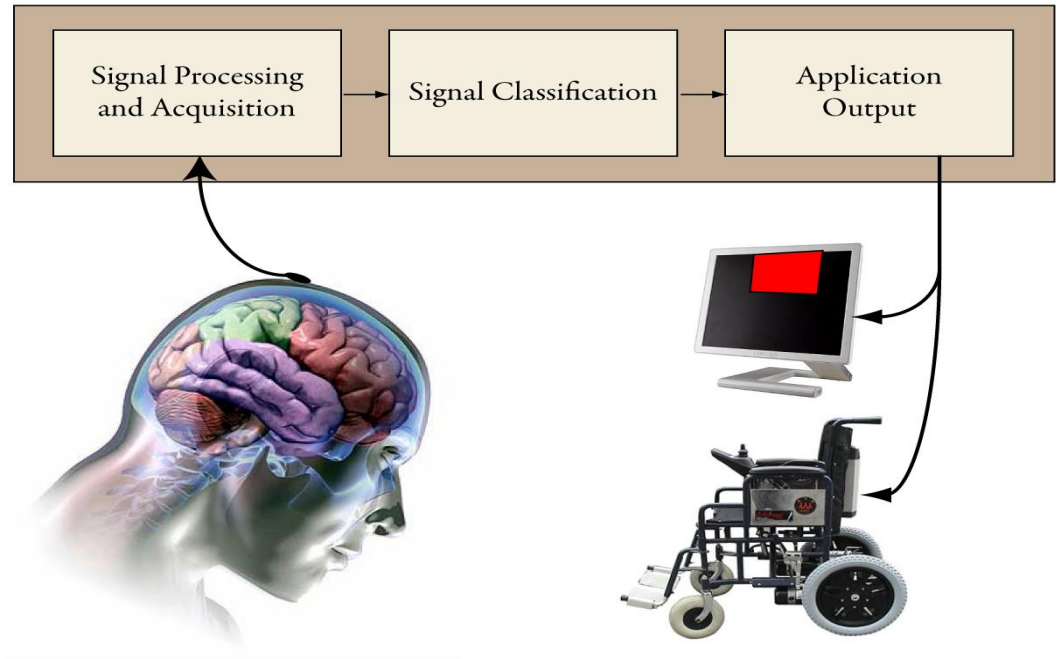
The history of brain–computer interfaces (BCIs) starts with Hans Berger's discovery of the electrical activity of the human brain and the development of electroencephalography (EEG). In 1924 Berger was the first to record human brain activity by means of EEG. By analyzing EEG traces, Berger was able to identify oscillatory activity in the brain, such as the alpha wave (8–12 Hz), also known as Berger's wave.

Berger's first recording device was very rudimentary. He inserted silver wires under the scalps of his patients. These were later replaced by silver foils attached to the patients' head by rubber bandages. Berger connected these sensors to a Lippmann capillary electrometer, with disappointing results. More sophisticated measuring devices, such as the Siemens double-coil recording galvanometer, which displayed electric voltages as small as one ten thousandth of a volt, led to success.

Berger analyzed the interrelation of alternations in his EEG wave diagrams with brain diseases. EEGs permitted completely new possibilities for the research of human brain activities.

What is EEG :

An electroencephalogram is a measure of the brain's voltage fluctuations as detected from scalp electrodes.It is an approximation of the cumulative electrical activity of neurons. Electroencephalography (EEG) is the most studied potential non-invasive interface, mainly due to its fine temporal resolution, ease of use, portability and low set-up cost. But as well as the technology's susceptibility to noise, another substantial barrier to using EEG as a brain–computer interface is the extensive training required before users can work the technology. For example, in experiments beginning in the mid-1990s, Niels Birbaumer at theUniversity of Tübingen in Germany trained severely paralysed people to self-regulate the *slow cortical potentials* in their EEG to such an extent that these signals could be used as a binary signal to control a computer cursor. (Birbaumer had earlier trained epileptics to prevent impending fits by controlling this low voltage wave.) The experiment saw ten patients trained to move a computer cursor by controlling their brainwaves. The process was slow, requiring more than an hour for patients to write 100 characters with the cursor, while training often took many months.



FigureP-2 General Structure of BCI system

Physical Mechanisms:

EEGs require electrodes attached to the scalp with sticky gel . Require physical connection to the machine.

Electrode Placement:

- Standard “10-20 System”
- Spaced apart 10-20%
- Letter for region
  - F - Frontal Lobe
  - T - Temporal Lobe
  - C - Center
  - O - Occipital Lobe
- Number for exact position
  - Odd numbers - left
  - Even numbers - right

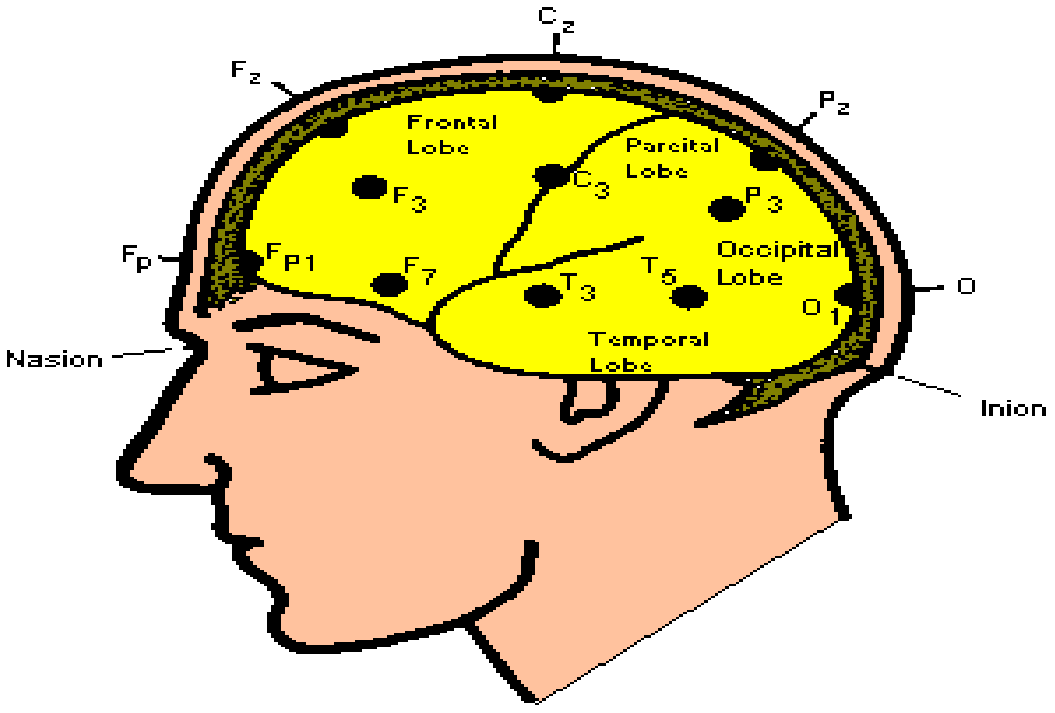


Figure P-2 the distribution of the sensors on the Cortex

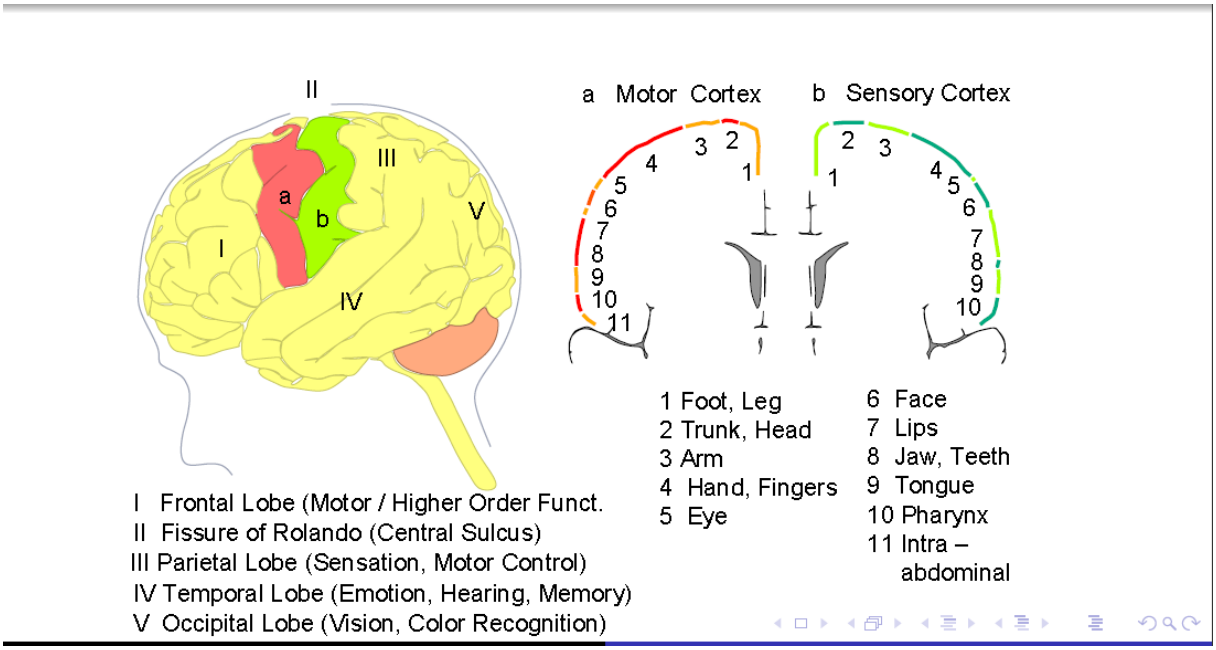


Figure P-3 the function of the brain

Brain Features:

User must be able to control the output:

Use a feature of the continuous EEG output that the user can reliably modify (waves), or

Evoke an EEG response with an external stimulus (evoked potential).

Brain Waves:

Generally grouped by frequency: (amplitudes are about 100µV max).

Type	Frequency	Location	Use
Delta	<4 Hz	everywhere	occur during sleep, coma
Theta	4-7 Hz	temporal and parietal	correlated with emotional stress (frustration & disappointment)
Alpha	8-12 Hz	occipital and parietal	reduce amplitude with sensory stimulation or **mental imagery**
Beta	12-36 Hz	parietal and frontal	can increase amplitude during intense **mental activity**
Mu	9-11 Hz	frontal (motor cortex)	diminishes with movement or **intention of movement**
Lambda	sharp, jagged	occipital	correlated with visual attention
Vertex			higher incidence in patients with epilepsy or encephalopathy

Figure P-4 the Brain Waves

Figure 1-1 NeuroSky MindSet

The NeuroSky MindSet is a brainwave sensing headset which uses a ‘medical grade’ probe to capture brain patterns and translate them into stuff you can do with a computer.

NeuroSky Technology Overview:

Brainwaves:

The last century of neuroscience research has greatly increased our knowledge about the brain and particularly, the electrical signals emitted by neurons firing in the brain. the patterns and frequencies of these electrical signals can be measured by placing a sensor on the scalp. the MindSet contains NeuroSky thinkGear™ technology, which measures the analog electrical signals, commonly referred to as brainwaves, and processes them into digital signals to make the measurements available to games and applications. the table below gives a general synopsis of some of the commonly-recognized frequencies that tend to be generated by different types of activity in the brain:

BrainWave Type	**Frequency range**	**Mental states and conditions**
Delta	0.1 Hz to 3 Hz	Deep, dreamless sleep, non-REM sleep, unconscious
Theta	4 Hz to 7 Hz	Intuitive, creative, recall, fantasy, imaginary, dream
Alpha	8 Hz to 12 Hz	Relaxed, but not drowsy, tranquil, conscious
Low Beta	12 Hz to 15 Hz	Formerly SMR, relaxed yet focused, integrated
Midrange Beta	16 Hz to 20 Hz	□inking, aware of self & surroundings
High Beta	21 Hz to 30 Hz	Alertness, agitation

ThinkGear:

thinkGear is the technology inside every NeuroSky product or partner product that enables a device to interface with the wearers' brainwaves. It includes the sensor that touches the forehead, the contact and reference points located on the ear pad, and the on-board chip that processes all of the data. Both the raw brainwaves and the eSense Meters (Attention and Meditation) are calculated on the thinkGear chip.

ThinkGear Data Values:

POOR_SIGNAL Quality:

this unsigned one-byte integer value describes how poor the signal measured by the thinkGear is. It ranges in value from 0 to 200. Any non-zero value indicates that some sort of noise contamination is detected. the higher the number, the more noise is detected. A value of 200 has a special meaning, specifically that the thinkGear contacts are not touching the user's skin. this value is typically output every second, and indicates the poorness of the most recent measurements. Poor signal may be caused by a number of different things. In order of severity, they are:

• Sensor, ground, or reference contacts not being on a person's head (i.e. when nobody is wearing the thinkGear).

• Poor contact of the sensor, ground, or reference contacts to a person's skin (i.e. hair in the way, or headset which does not properly fit a person's head, or headset not properly placed on the head).

• Excessive motion of the wearer (i.e. moving head or body excessively, jostling the headset).

• Excessive environmental electrostatic noise (some environments have strong electric signals or static electricity buildup in the person wearing the sensor).

• Excessive non-EEG biometric noise (i.e. EMG, EKG/ECG, EOG, etc)

A certain amount of noise is unavoidable in normal usage of □inkGear, and both NeuroSky's filtering technology and eSense™ algorithm have been designed to detect, correct, compensate for, account for, and tolerate many types of non-EEG noise.

Most typical users who are only interested in using the eSense values, such as Attention and Meditation, do not need to worry too much about the POOR_SIGNAL Quality value, except to note that the Attention and Meditation values will not be updated while POOR_SIGNAL is detected. the POOR_SIGNAL Quality value is more useful to some applications which need to be more sensitive to noise (such as some medical or research applications), or applications which need to know right away when there is even minor noise detected. By default, output of this Data Value is enabled. It is typically output once a second.

eSense:

eSense™ is a NeuroSky's proprietary algorithm for characterizing mental states. To calculate eSense, the NeuroSky thinkGear technology amplifies the raw brainwave signal and removes the ambient noise and muscle movement. the eSense algorithm is then applied to the remaining signal, resulting in the interpreted eSense meter values. Please note that eSense meter values do not describe an exact number, but instead describe ranges of activity

eSense Meter - General Information:

the eSense meters are a way to show how effectively the user is engaging Attention (similar to concentration) or Meditation (similar to relaxation).

Like exercising an unfamiliar muscle, it may take some time to gain full proficiency with each of the eSense™ meters. In many cases, people tend to be better at one eSense than the other when they first begin. We recommend trying different tactics until you are successful with one. Once you see a reaction on the screen from your efforts, you will be able to duplicate the action more easily with additional practice.

Generally, Attention can be controlled through a visual focus. Focus on a singular idea. Try to “funnel” your concentration and focus your train of thought towards pushing up the meter. Other suggestions include picking a point on the screen to stare at or imagining the action you are trying to accomplish happening. For example, look at the Attention eSense meter and imagine the dial moving towards higher numbers.

For Meditation, it typically helps to try to relax yourself. Connect to a sense of peace and calm by clearing your mind of thoughts and distractions. If you are having difficulty engaging Meditation, close your eyes, wait a number of seconds, and then open your eyes to see how the meter has responded.

If you have trouble at first in controlling your eSense meter levels, be patient. Try different techniques and practice. Also be sure to read and try to understand the Technical Description in order to get a better idea about how eSense actually works under the hood.

eSense Meter - Technical Description:

For each different type of eSense (i.e. Attention, Meditation), the meter value is reported on a relative eSense scale of 1 to 100. On this scale, a value between 40 to 60 at any given moment in time is considered “neutral” and is similar in notion to “baselines” that are established in conventional brainwave measurement techniques (though the method for determining a thinkGear baseline is proprietary and may differ from conventional brainwaves).

A value from 60 to 80 is considered “slightly elevated”, and may be interpreted as levels tending to be higher than normal (levels of Attention or Meditation that may be higher than normal for a given person). Values from 80 to 100 are considered “elevated”, meaning they are strongly indicative of heightened levels of that eSense.

Similarly, on the other end of the scale, a value between 20 to 40 indicates “reduced” levels of the eSense, while a value between 1 to 20 indicates “strongly lowered” levels of the eSense. these levels may indicate states of distraction, agitation, or abnormality, according to the opposite of each eSense. the reason for the somewhat wide ranges for each interpretation is that some parts of the eSense algorithm are dynamically learning and at times employ some “slow-adaptive” algorithms to adjust to natural fluctuations and trends of each user, accounting for and compensating for the fact that brainwaves in the human brain are subject to normal ranges of variance and fluctuation. this is part of the reason why thinkGear sensors are able to operate on a wide range of individuals under an extremely wide range of personal and environmental conditions, while still giving good accuracy and reliability.

ATTENTION eSense

the eSense Attention meter indicates the intensity of a user's level of mental “focus” or “attention”, such as that which occurs during intense concentration and directed (but stable) mental activity. Its value ranges from 0 to 100. Distractions, wandering thoughts, lack of focus, or anxiety may lower the Attention meter level.

MEDITATION eSense

the eSense Meditation meter indicates the level of a user's mental “calmness” or “relaxation”. Its value ranges from 0 to 100. Note that Meditation is a measure of a person's mental states, not physical levels, so simply relaxing all the muscles of the body may not immediately result in a heightened Meditation level. However, for most people in most normal circumstances, relaxing the body often helps the mind to relax as well. Meditation is related to reduced activity by the active mental processes in the brain.
It has long been an observed effect that closing one's eyes turns off the mental activities which process images from the eyes. So closing the eyes is often an effective method for increasing the Meditation meter level. Distractions, wandering thoughts, anxiety, agitation, and sensory stimuli may lower the Meditation meter levels.

RAW Wave Value (16-bit):

this Data Value consists of two bytes, and represents a single raw wave sample. Its value is a signed 16-bit integer that ranges from -32768 to 32767. the first byte of the Value represents the high-order bits of the twos-compliment value, while the second byte represents the low-order bits. To reconstruct the full raw wave value, simply shift the first byte left by 8 bits, and bitwise-or with the second byte:

```
short raw = (Value[0]<<8) | Value[1];
```

where Value[0] is the high-order byte, and Value[1] is the low-order byte.
In systems or languages where bit operations are inconvenient, the following arithmetic operations may be substituted instead:

```
raw = Value[0]*256 + Value[1];
if( raw >= 32768 ) raw = raw - 65536;
```

where raw is of any signed number type in the language that can represent all the numbers from -32768 to 32767.

Each thinkGear model reports its raw wave information in only certain areas of the full -32768 to 32767 range. For example, MindSet reports raw waves that fall between approximately -2048 to 2047.
By default, output of this Data Value is enabled, and is outputed 512 times a second, or approximately once every 2ms.

ASIC_EEG_POWER:
this Data Value represents the current magnitude of 8 commonly-recognized types of EEG (brainwaves). this Data Value is output as a series of eight 3-byte unsigned integers in little-endian format. the eight EEG powers are output in the following order: delta (0.5 - 2.75Hz), theta (3.5 - 6.75Hz), low-alpha (7.5 - 9.25Hz), high-alpha (10 - 11.75Hz), low-beta (13 - 16.75Hz), high-beta (18 - 29.75Hz), low-gamma (31 - 39.75Hz), and mid-gamma (41 - 49.75Hz). these values have no units and therefore are only meaningful compared to each other and to themselves, to consider relative quantity and temporal fluctuations. By default, output of this Data Value is enabled, and is typically output once a second.
Blink Strength:
this unsigned one byte value reports the intensity of the user's most recent eye blink. Its value ranges from 1 to 255 and it is reported whenever an eye blink is detected. the value indicates the relative intensity of the blink, and has no units.

ThinkGear Packets:

thinkGear components deliver their digital data as an asynchronous serial stream of bytes. the serial stream must be parsed and interpreted as thinkGear Packets in order to properly extract and interpret the thinkGear Data Values described in the chapter above.
A thinkGear Packet is a packet format consisting of 3 parts:
1. Packet Header
2. Packet Payload
3. Payload Checksum

thinkGear Packets are used to deliver Data Values (described in the previous chapter) from a thinkGear module to an arbitrary receiver (a PC, another microprocessor, or any other device that can receive a serial stream of bytes). Since serial I/O programming APIs are different on every platform, operating system, and language, it is outside the scope of this document (see your platform's documentation for serial I/O programming). this chapter will only cover how to interpret the serial stream of bytes into thinkGear Packets, Payloads, and finally into the meaningful Data Values described in the previous chapter. the Packet format is designed primarily to be robust and flexible: Combined, the Header and Checksum provide data stream synchronization and data integrity checks, while the format of the Data Payload ensures that new data fields can be added to (or existing data fields removed from) the Packet in the future without breaking any Packet parsers in any existing applications/devices. this means that any application that implements a thinkGear Packet parser properly will be able to use newer models of thinkGear modules most likely without having to change their parsers or application at all, even if the newer thinkGear hardware includes new data fields or rearranges the order of the data fields.

Packet Structure:

Packets are sent as an asynchronous serial stream of bytes. the transport medium may be UART, serial COM, USB, bluetooth, file, or any other mechanism which can stream bytes. Each Packet begins with its Header, followed by its Data Payload, and ends with the Payload's Checksum Byte, as follows:

[SYNC] [SYNC] [PLENGTH] [PAYLOAD...] [CHKSUM]

_____ _____ _____

^^^^^^^^(Header)^^^^^^^ ^^(Payload)^^ ^(Checksum)^

the [PAYLOAD…] section is allowed to be up to 169 bytes long, while each of [SYNC], [PLENGTH], and [CHKSUM] are a single byte each. this means that a complete, valid Packet is a minimum of 4 bytes long (possible if the Data Payload is zero bytes long, i.e. empty) and a maximum of 173 bytes long (possible if the Data Payload is the maximum 169 bytes long).

Packet Header:

the Header of a Packet consists of 3 bytes: two synchronization [SYNC] bytes (0xAA 0xAA), followed by a [PLENGTH] (Payload length) byte:

[SYNC] [SYNC] [PLENGTH]

_____

^^^^^^^^(Header)^^^^^^^

the two [SYNC] bytes are used to signal the beginning of a new arriving Packet and are bytes with the value 0xAA (decimal 170). Synchronization is two bytes long, instead of only one, to reduce the chance that [SYNC] (0xAA) bytes occurring within the Packet could be mistaken for the beginning of a Packet. Although it is still possible for two consecutive [SYNC] bytes to appear *within* a Packet (leading to a parser attempting to begin parsing the middle of a Packet as the beginning of a Packet) the [PLENGTH] and [CHKSUM] combined ensure that such a "mis-sync'd Packet" will never be accidentally interpreted as a valid packet (see Payload Checksum below for more details). the [PLENGTH] byte indicates the length, in bytes, of the Packet's Data Payload [PAYLOAD…] section, and may be any value from 0 up to 169. Any higher value indicates an error (PLENGTH TOO LARGE). Be sure to note that [PLENGTH] is the length of the Packet's **Data Payload**, NOT of the entire Packet. the Packet's complete length will always be [PLENGTH] + 4.

Data Payload:

the Data Payload of a Packet is simply a series of bytes. the number of Data Payload bytes in the Packet is given by the [PLENGTH] byte from the Packet Header. the interpretation of the Data Payload bytes into the thinkGear Data Values is defined in detail in the Data Payload Structure section below. Note that parsing of the Data Payload **typically should not even be**
attempted until **after** the Payload Checksum Byte [CHKSUM] is verified as described in the following
section.

Payload Checksum

the [CHKSUM] Byte must be used to verify the integrity of the Packet's Data Payload. the Payload's Checksum is defined as:

1. summing all the bytes of the Packet's Data Payload
2. taking the lowest 8 bits of the sum
3. performing the bit inverse (one's compliment inverse) on those lowest 8 bits

A receiver receiving a Packet must use those 3 steps to calculate the checksum for the Data Payload they received, and then compare it to the [CHKSUM] Checksum Byte received with the Packet. If the calculated payload checksum and received [CHKSUM] values do not match, the entire Packet should be discarded as invalid. If they do match, then the receiver may procede to parse the Data Payload as described in the "Data Payload Structure" section below.

Setting Up Your MindSet

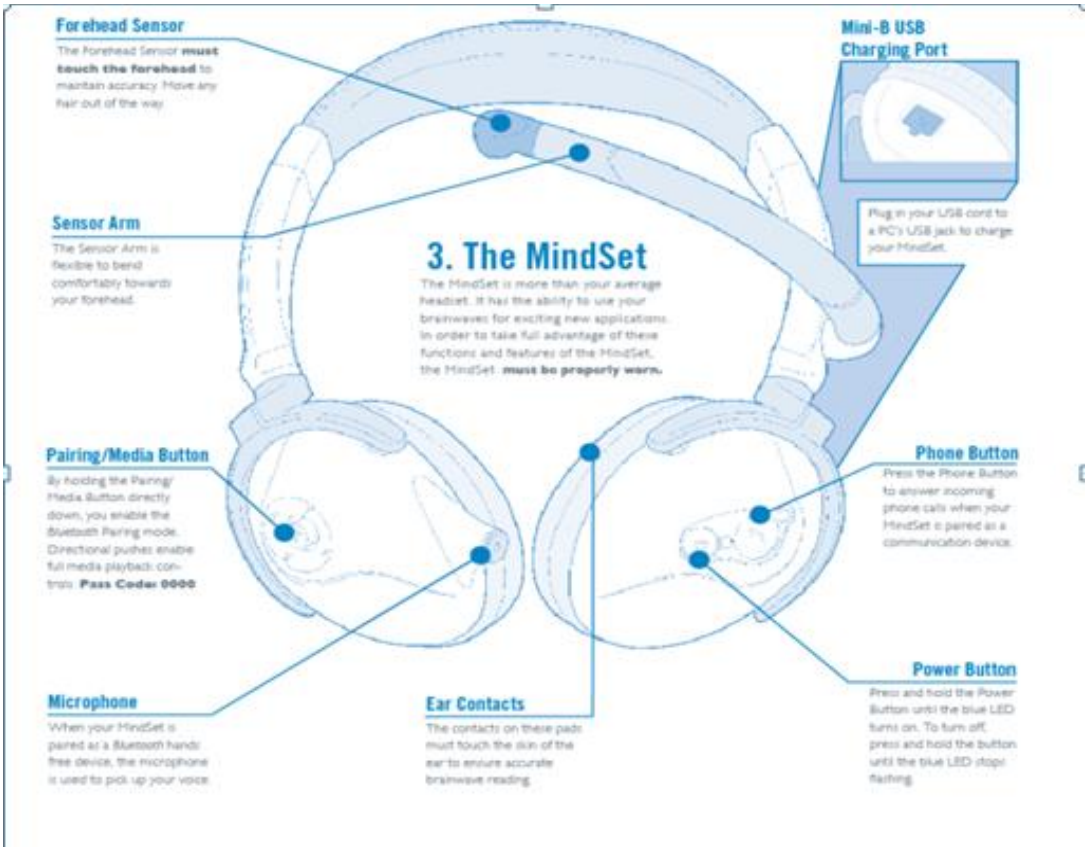


Figure 1-2 Overview of the MindSet

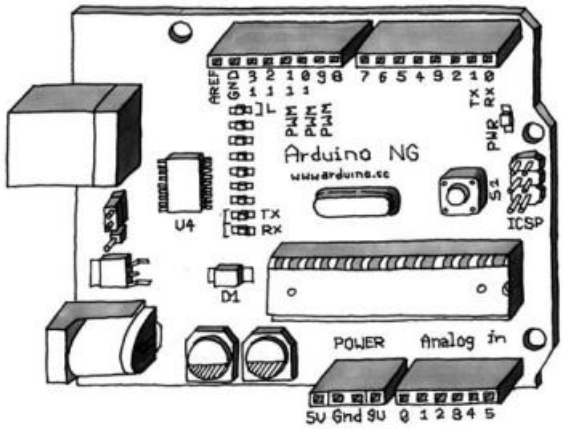


Figure 2-1 Arduino

Arduino is an open source physical computing platform based on a simple input/output (I/O) board and a development environment that implements the Processing language (www.processing.org) . Arduino can be used to develop standalone interactive objects or can be connected to software on your computer (such as Flash, Processing, VVVV, or Max/MSP). The boards can be assembled by hand or purchased preassembled; the open source IDE (Integrated Development Environment) can be downloaded for free from www.arduino.cc.

Arduino is defferent from other platforms on the market because of these features:

» It is a multiplatform environment; it can run on Windows, Macintosh, and Linux.

» It is based on the Processing programming IDE, an easy-to-use development environment used by artists and designers.

» You program it via a USB cable, not a serial port. This feature is useful, because many modern computers don't have serial ports.

» It is open source hardware and software—if you wish, you can download the circuit diagram, buy all the components, and make your own, without paying anything to the makers of Arduino.

» The hardware is cheap. The USB board costs about €20 (currently, about US$35) and replacing a burnt-out chip on the board is easy and costs no more than €5 or US$4. So you can afford to make mistakes.

» There is an active community of users, so there are plenty of people who can help you.
» The Arduino Project was developed in an educational environment and is therefore great for newcomers to get things working quickly.

After Arduino started to become popular, I realised how experimenters, hobbyists, and hackers of all sorts were starting to use it to create beautiful and crazy objects. I realised that you're all artists and designers in your own right, so this book is for you as well.

Arduino was born to teach Interaction Design, a design discipline that puts prototyping at the centre of its methodology. There are many definitions of Interaction Design, but the one that I prefer is:

Interaction Design is the design of any interactive experience:

In today's world, Interaction Design is concerned with the creation of meaningful experiences between us (humans) and objects. It is a good way to explore the creation of beautiful—and maybe even controversial— experiences between us and technology. Interaction Design encourages design through an iterative process based on prototypes of ever-increasing fidelity. This approach—also part of some types of "conventional" design—can be extended to include prototyping with technology; in particular, prototyping with electronics

The specific field of Interaction Design involved with Arduino is Physical Computing (or Physical Interaction Design).

What Is Physical Computing?

Physical Computing uses electronics to prototype new materials for designers and artists.
It involves the design of interactive objects that can communicate with humans using sensors and actuators controlled by a behaviour implemented as software running inside a microcontroller (a small computer on a single chip).
In the past, using electronics meant having to deal with engineers all the time, and building circuits one small component at the time; these issues kept creative people from playing around with the medium directly. Most of the tools were meant for engineers and required extensive knowledge.
In recent years, microcontrollers have become cheaper and easier to use, allowing the creation of better tools.
The progress that we have made with Arduino is to bring these tools one step closer to the novice, allowing people to start building stuff after only two or three days of a workshop.
With Arduino, a designer or artist can get to know the basics of electronics and sensors very quickly and can start building prototypes with very little investment.

The Arduino Platform

Arduino is composed of two major parts:

1) Arduino board, which is the piece of hardware you work on when you build your objects.
2) Arduino IDE, the piece of software you run on your computer. You use the IDE to create a sketch (a little computer program) that you upload to the Arduino board.

The sketch tells the board what to do Not too long ago, working on hardware meant building circuits from scratch, using hundreds of different components with strange names like resistor, capacitor, inductor, transistor, and so on.

Every circuit was “wired” to do one specific application, and making changes required you to cut wires, solder connections, and more.
With the appearance of digital technologies and microprocessors, these functions, which were once implemented with wires, were replaced by software programs Software is easier to modify than hardware. With a few keypresses, you can radically change the logic of a device and try two or three versions in the same amount of time that it would take you to solder a couple of resistors.

.

The Arduino Hardware:

The Arduino board is a small microcontroller board,(which is a small circuit). (the board that contains a whole computer on a small chip the microcontroller) This computer is at least a thousand times less powerful than the MacBook I'm using to write this, but it's a lot cheaper and very useful to build interesting devices. Look at the Arduino board: you'll see a black chip with 28 "legs"—that chip is the ATmega328, the heart of your board.

We (the Arduino team) have placed on this board all the components that are required for this microcontroller to work properly and to communicate with your computer. There are many versions of this board; the one we'll use throughout this book is the Arduino Uno, which is the simplest one to use and the best one for learning on.

.

In those illustrations, you see the Arduino board. At first, all those connectors might be a little confusing. Here is an explanation of what every element of the board does:

Digital I/O

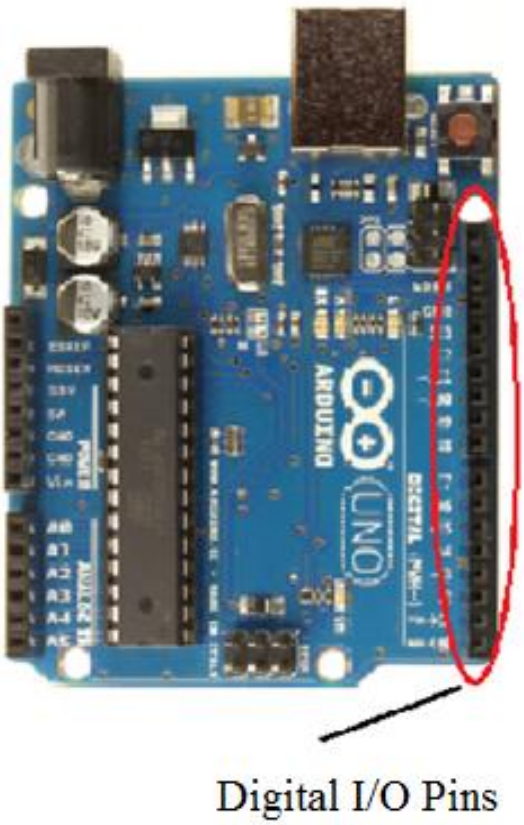


Figure 2-2 Digital I/O Pins

•Input/Output (I/O) is done through pins.

•Input = read physical world data.

•Output = write data to the physical world.

•You insert a wire into a pin and connect it to something else.

•We can program a pin to be input **OR** output.

•Digital pins have two values: high (5 Volts) or low (0 Volt).

Analog I/O

Figure 2-3 Analog I/O Pins

•Analog I/O has a range of numbers.

•Output : 0 ... 255 (256 voltage steps from 0 to 5V).

•Input: 0 ... 1023 (1024 voltage steps from 0 to 5V).

•Analog input pin: we can use it for example to determine the distance of an object via infra-red sensor.

•Analog output pin: we can use it for example to set speed of a motor or the brightness of a LED

Pulse Width Modulation

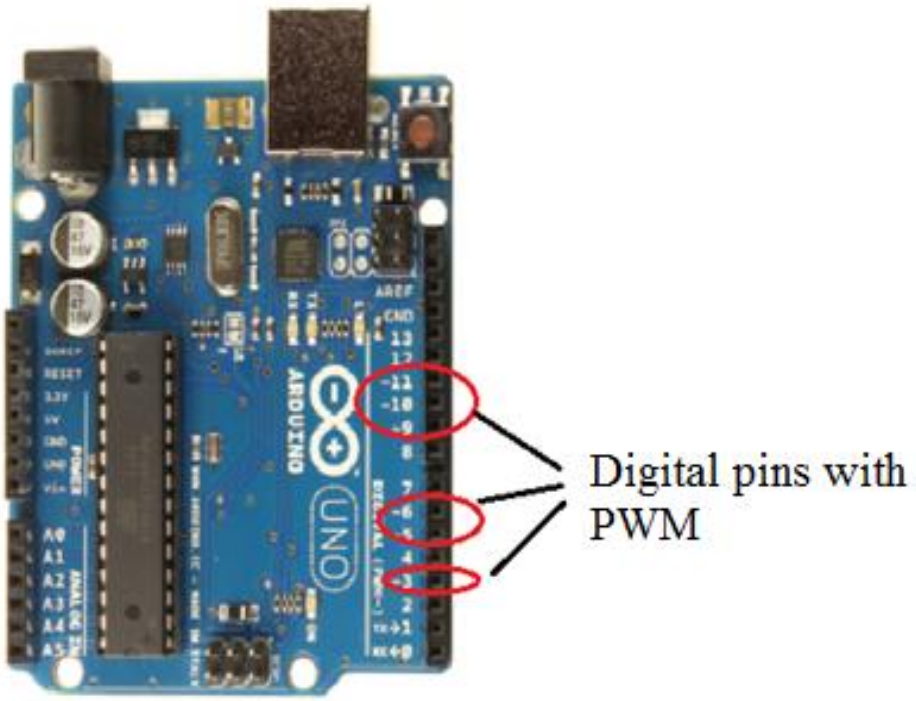


Figure 2-4 Digital Pins with PWM

•Some digital ports can be programmed to output analog signals.

•We enable those ports for output with pulse width modulation (PWM).

•PWM is obtained by varying between HIGH and LOW at the appropriate interval of time.

Other board components

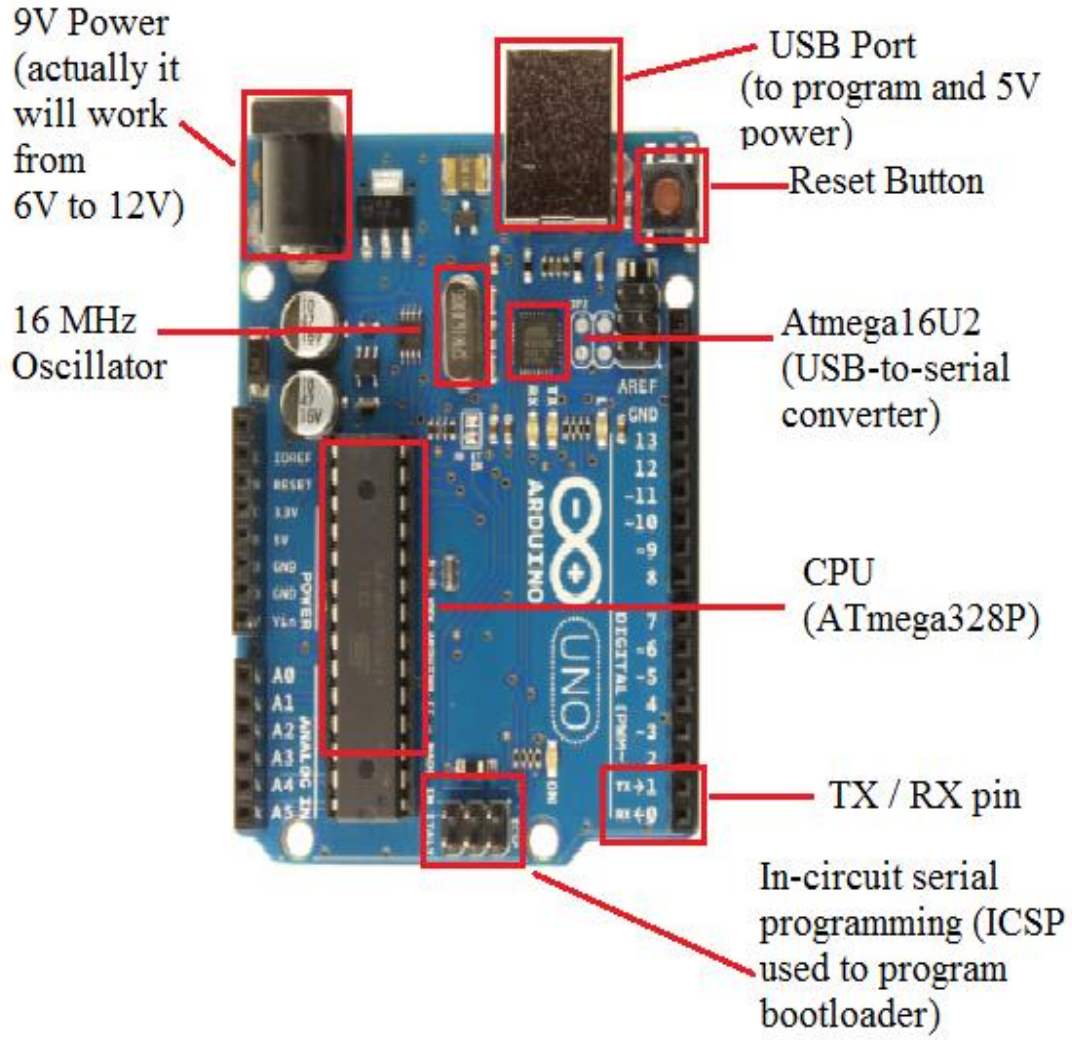


Figure 2-5 Other board components

Arduino Uno specification

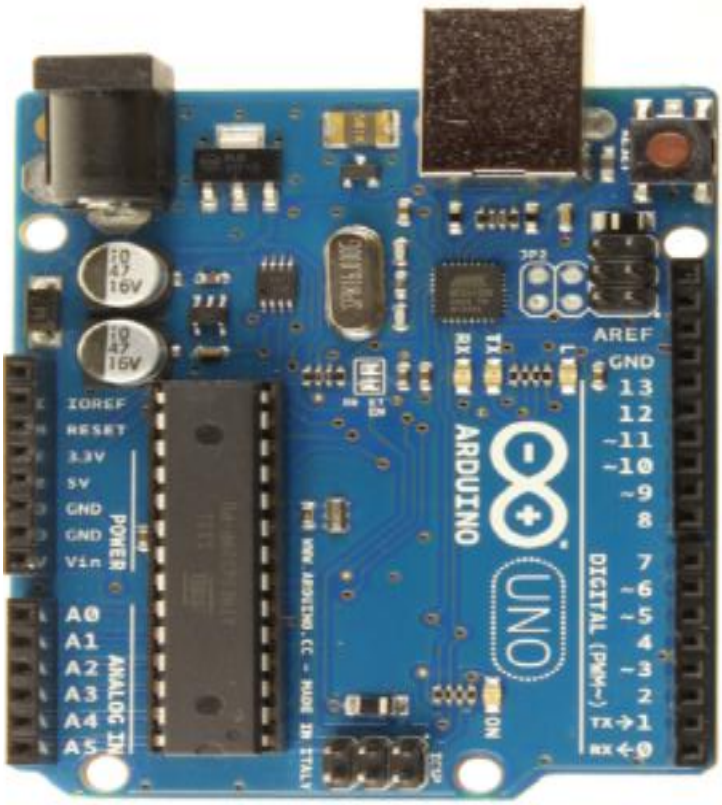


Figure 2-6 Arduino Uno specification

Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Bootloader

•The bootloader is a small piece of software that is burned onto the chips that come with your Arduino boards. It allows the user to upload programs to the board without external hardware.

•When you reset the Arduino board, it runs the bootloader (if present) and digital pin 13 is pulsed.

•The bootloader waits a few seconds for data to arrive from the the USB port. Seconds later, the bootloader launches the newly-uploaded program. If no data arrives from the USB port, the bootloader launches whatever program was last uploaded onto the chip.

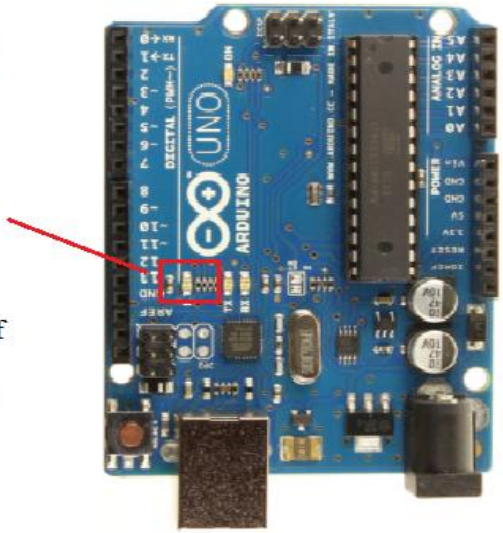


Figure 2-7 Bootloader

Power pins

Figure 2-8 Power Pins

•**VIN** - You may provide a regulated input voltage to the Arduino board as opposed to 5 volts from the USB connection or other regulated power source.

•**5V** - When power is provided to the board, this pin has 5V (reference to **GND** pin)

•**3V3** - A 3.3 volt supply generated by the on-board regulator (reference to **GND** pin). Maximum current draw is 50 mA.

Other pins

Figure 2-9 Other pins

•Digital Pin 2 and 3 can be used as **external interrupts** (trigger an interrupt on a low value, a rising or falling edge, or a change in value).

•**AREF** - Reference voltage for the analog inputs.

•**Reset** - Bring this line LOW to reset the microcontroller.

Polyfuse

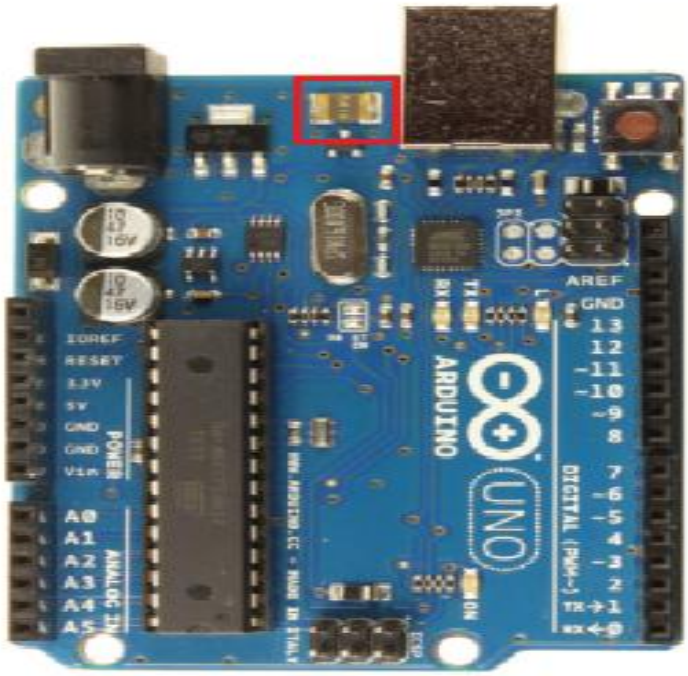


Figure 2-10 Polyfuse

•What is that funny looking chip?

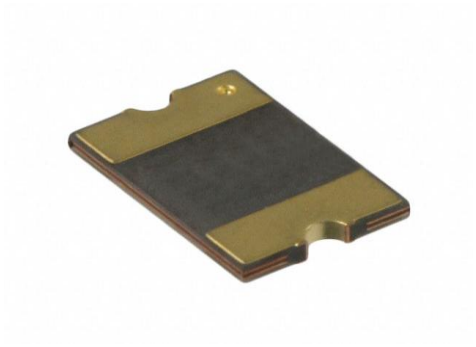


Figure 2-11 chip of Polyfuse

•Its a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection.

The Software (IDE)

The IDE (Integrated Development Environment) is a special program running on your computer that allows you to write sketches for the Arduino board in a simple language modeled after the Processing (www.processing.org) language. The magic happens when you press the button that uploads the sketch to the board: the code that you have written is translated into the C language (which is generally quite hard for a beginner to use), and is passed to the avr-gcc compiler, an important piece of open source software that makes the final translation into the language understood by the microcontroller. This last step is quite important, because it's where Arduino makes your life simple by hiding away as much as possible of the complexities of programming microcontrollers.

Features of the Arduino (IDE) :

• Simple to use
• Open Source (Free)
• Programming style similar to C
• You can download it from www.arduino.cc

The programming cycle on Arduino is basically as follows:

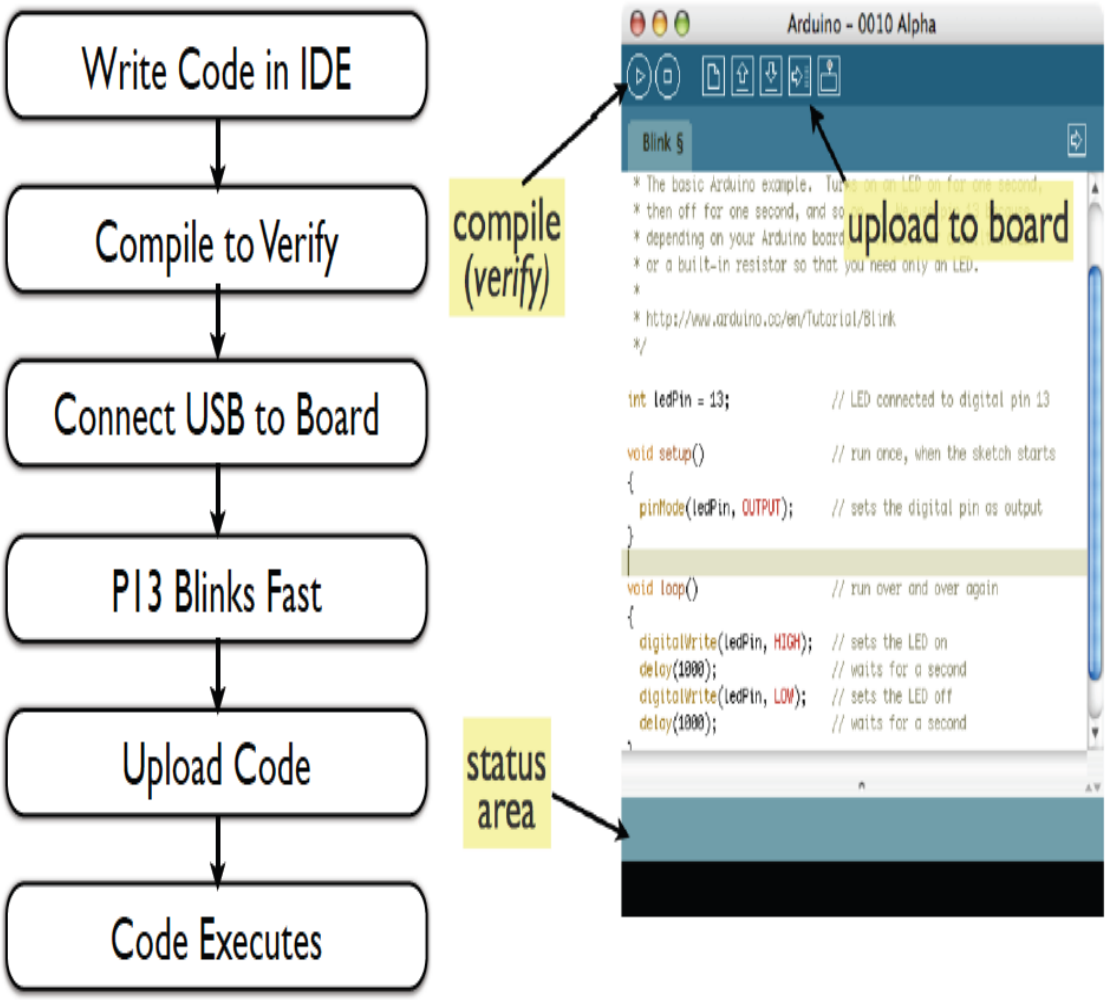


Figure 2-12 The programming cycle on Arduino

Anatomy of a “sketch” :

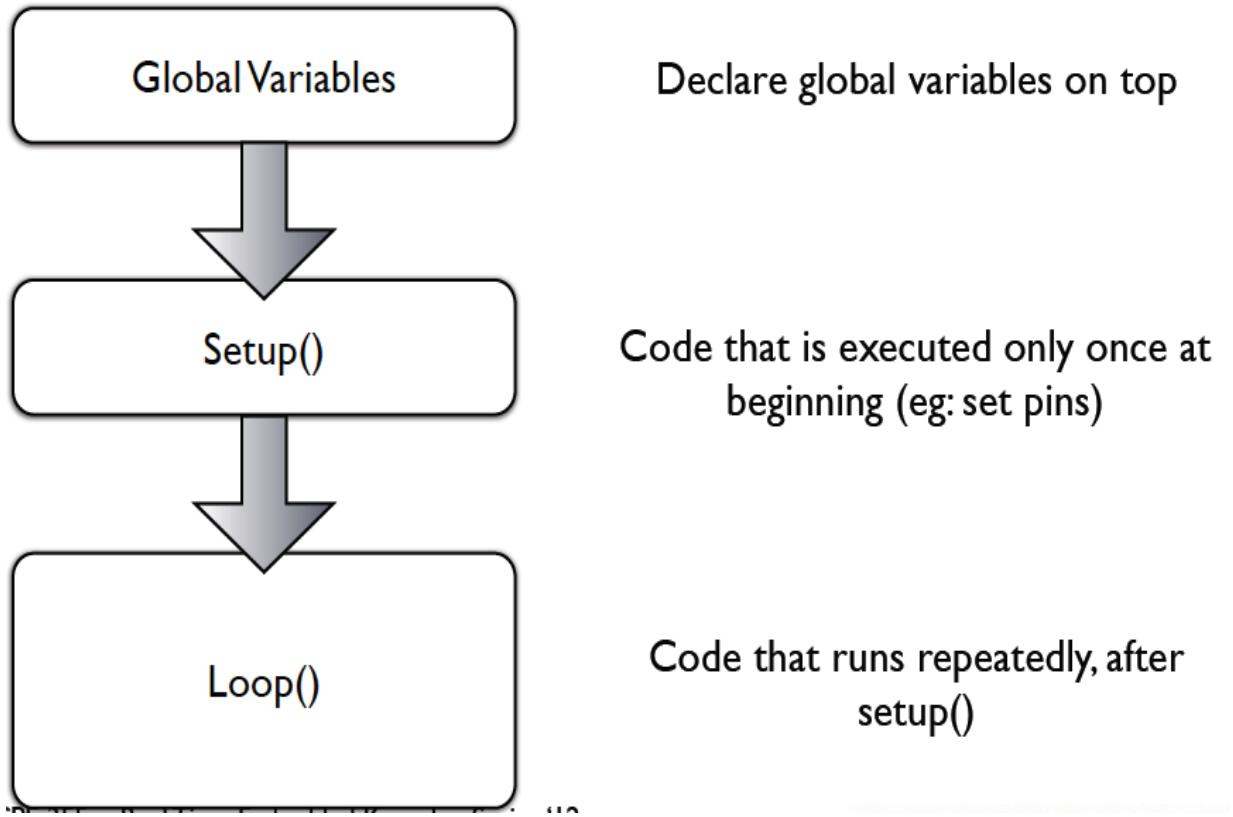


Figure 2-13 Anatomy of a “sketch”

Installing Arduino on Your Computer

To program the Arduino board :

- you must first download the development environment (the IDE) from here: www.arduino.cc/en/Main/Software.
- Choose the right version for your operating system.
- Download the file and double-click on it to open it ; on Windows or Linux, this creates a folder named *arduino-[version]*, such as *arduino-1.0*.
- Drag the folder to wherever you want it: your desktop, your *Program Files* folder (on Windows), etc. On the Mac, double-clicking it will open a disk image with an Arduino application (drag it to your Applications folder).
- Now whenever you want to run the Arduino IDE, you'll open up the *arduino* (Windows and Linux) or *Applications* folder (Mac), and double-click the Arduino icon. Don't do this yet, though; there is one more step.
- Now you must install the drivers that allow your computer to talk to your board through the USB port.

Installing Drivers: Macintosh

- The Arduino Uno on a Mac uses the drivers provided by the operating system, so the procedure is quite simple. Plug the board into your computer.
- The PWR light on the board should come on and the yellow LED labelled "L" should start blinking.
- You might see a popup window telling you that a new network interface has been detected.
- If that happens, Click "Network Preferences...", and when it opens, click "Apply". The Uno will show up as "Not Configured", but it's working properly.
- Quit System Preferences.

If you have an older Arduino board, look for instructions here: www.arduino.cc/en/Guide/MacOSX.

Installing Drivers: Windows

- Plug the Arduino board into the computer; when the Found New Hardware Wizard window comes up, Windows will first try to find the driver on the Windows Update site.
- Windows XP will ask you whether to check Windows Update; if you don't want to use Windows Update, select the "No, not at this time" option and click Next.
- On the next screen, choose "Install from a list or specific location" and click Next.
- Navigate to and select the Uno's driver file, named *ArduinoUNO.inf*, located in the "Drivers" folder of the Arduino Software download (not the "FTDI USB Drivers" sub-directory). Windows will finish up the driver installation from there.

If you have an older board, look for instructions here: www.arduino.cc/en/Guide/Windows.

Once the drivers are installed, you can launch the Arduino IDE and start using Arduino.

.

Sensors and Actuators

Sensors and actuators are electronic components that allow a piece of electronics to interact with the world.

As the microcontroller is a very simple computer, it can process only electric signals (a bit like the electric pulses that are sent between neurons in our brains). For it to sense light, temperature, or other physical quantities, it needs something that can convert them into electricity. In our body, for example, the eye converts light into signals that get sent to the brain using nerves. In electronics, we can use a simple device called a light-dependent resistor (an LDR or photoresistor) that can measure the amount of light that hits it and report it as a signal that can be understood by the microcontroller.

Once the sensors have been read, the device has the information needed to decide how to react. The decision-making process is handled by the microcontroller, and the reaction is performed by actuators. In our bodies, for example, muscles receive electric signals from the brain and convert them into a movement. In the electronic world, these functions could be performed by a light or an electric motor.

In the following sections, you will learn how to read sensors of different types and control different kinds of actuators.

Chapter 3 Building the Robot

Before programming anything, we'll build the chassis for the robot. Basically it's a traditional rover robot structure with four dc motors . To make it suitable for mind-controlling needs, we'll add a line detector and motor driver kit to communicate the motors with arduino. We use a solder less breadboard and Bluetooth shield for the Arduino, to make the system full wireless and communicate the ardunio with Neruosky by Bluetooth .We can adding components and wirers easy to arduino after we put the Bluetooth shield.

Here's how all the major components will work together to create a working robot:

Arduino*:*

This is the brains of the project. It is essentially a small embedded computer With a brain (a microcontroller), as well as header pins that can Connect to inputs (sensors) and outputs (actuators).

4WD Chassis*:*

This holds everything together. It's essentially the platform for the Robot.

Motors:

These are motors that can be connected to (Motor Driver FD04A) and then connect the motor driver to the Arduino. Arduino communicates with them by sending pulses to control speed and direction.

Motor Driver kit:

This kit used to connect the 4 dc motors with arduino to communicate the speed and direction.

Line Detector:

With the line detector, your robot will avoid a black line, which makes it stay in the arena (helpful for keeping it from falling off a table).

Mind Wave:

Mind Wave measures your brainwaves and transmits the results for the Arduino. We have to add a Bluetooth shield to connect the arduino with Mind wave by Bluetooth dongle. We want to connect it directly to Arduino instead of a computer's USB port.

Tools and Parts

Here we list the parts and tools needed to make the robot. Feel free to improvise if you don't find the exact matches.

Figure 3-1 shows the parts & tools of the robot

1. Rechargeable battery (we used a 4 AA battery enercell 2500mAh 4.8v).
2. Vanson compact charger for the battery.
3. Chargable battery Energizer 5000mAh 9v for the motor driver kit.
4. Battery holder for Energizer 9v.
5. Nerousky Mind wave.
6. Line-detecting sensor.
7. Connection wire for the line-detecting sensor.
8. Small solderless breadboard.
9. Arduino Uno.
10. Motor driver 4 channel (we used Flexibot Driver FD04A).
11. Bluetooth shield for Arduino.
12. Ribbon cable or assorted wire in four different colors.
13. Jumpers Wires.
14. Battery holder tie rap.

The 4WD Chassis parts:

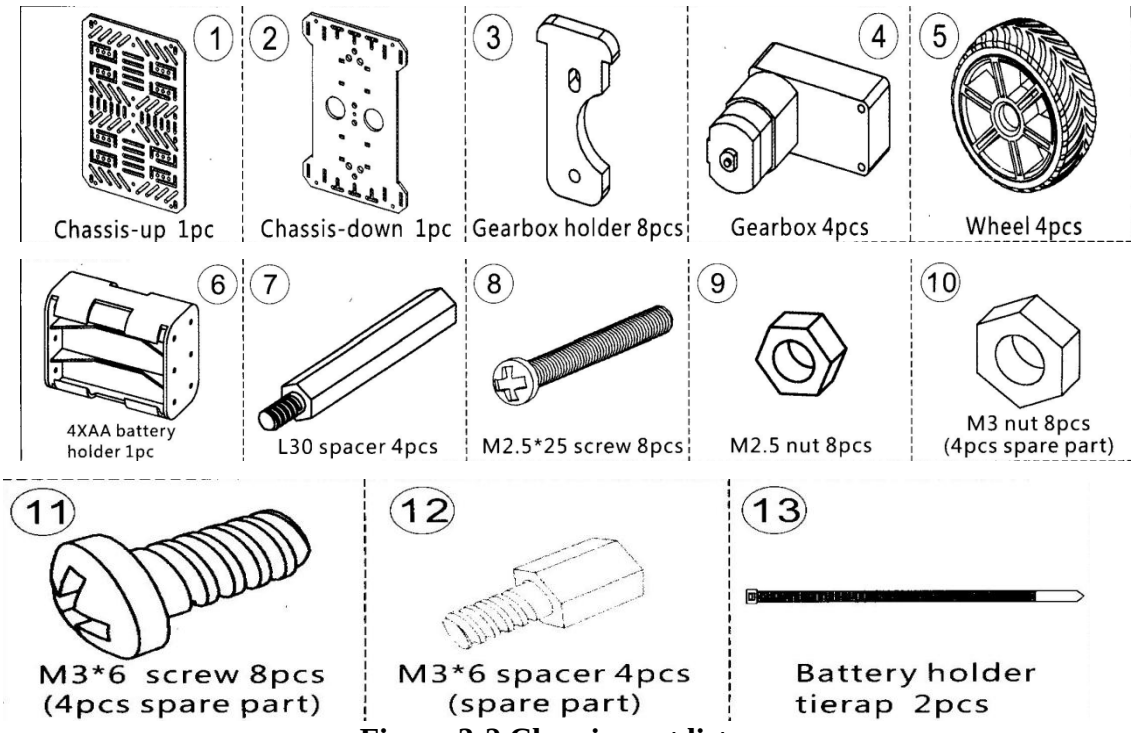


Figure 3-2 Chassis part list

1. Chassis-up 1 pc.
2. Chassis-down 1 pc.
3. Gearbox holder 8 pcs.
4. Gearbox 4 pcs (DC motors).
5. Wheel 4 pcs.
6. 4X AA battery holder 1 pc.
7. L30 spacer 4 pcs.
8. M2.5*25 screw 8pcs.
9. M 2.5 nut 8 pcs.
10. M3 nut 8 pcs (4 pcs spare part).
11. M 3*6 screw 8 pcs (4 pcs spare part).
12. M3*6 spacer 4 pcs (spare part).

Tools:

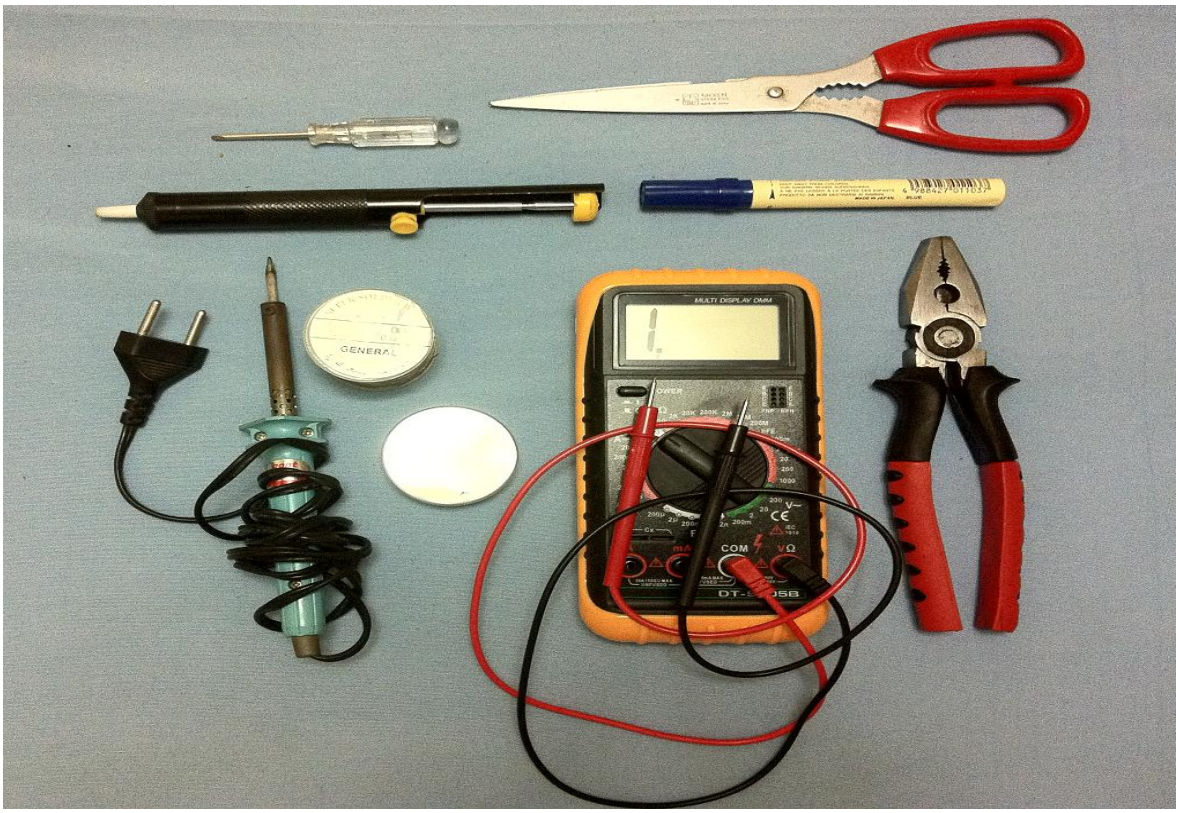


Figure 3-3 Tools

1. Scissors.
2. Soldering iron and solder.
3. Pliers.
4. Phillips screwdriver.
5. Marker.
6. Torch or lighter.
7. Diagonal cutter pliers.
8.Avometer.

Gearbox dc motor:

Dc motor motors will be moving the wheels of our robot. The most usual type of Dc motors have limited rotation. They are used when you need to turn the motor to a specific angle. In our robot, we only need to control speed and direction. And, of course, the motor needs to be able to turn freely. Continuous rotation Dc motors are made for this. Almost any Dc motor can be modified to continuous rotation, but it's easier to buy a ready-made version.
The Parallax (Futaba) continuous rotation Dc motor is perfect for our needs. It has an external potentiometer adjustment screw, which allows identical centering of two Dc motors effortlessly. You'll notice how handy this is later when we program the movements for the robot.

Chassis:

Now we will implement the main body of the robot the chassis it's easy tom implement it just follow the instruction below and read it carefully.

Step 1:

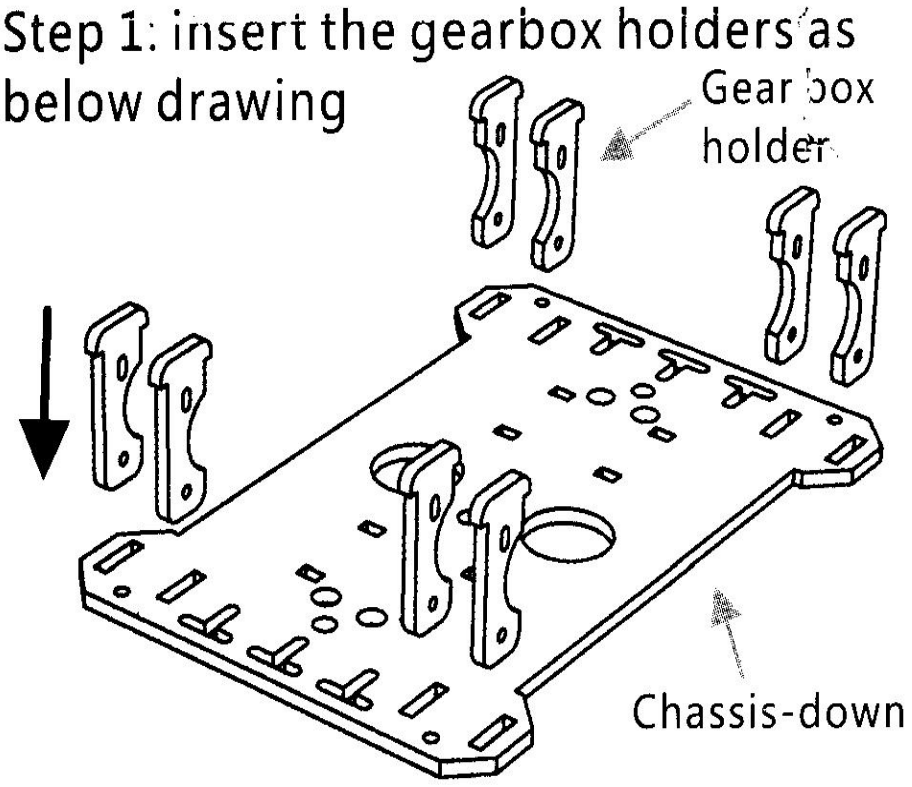


Figure 3-4 Assembly instruction

Step2:

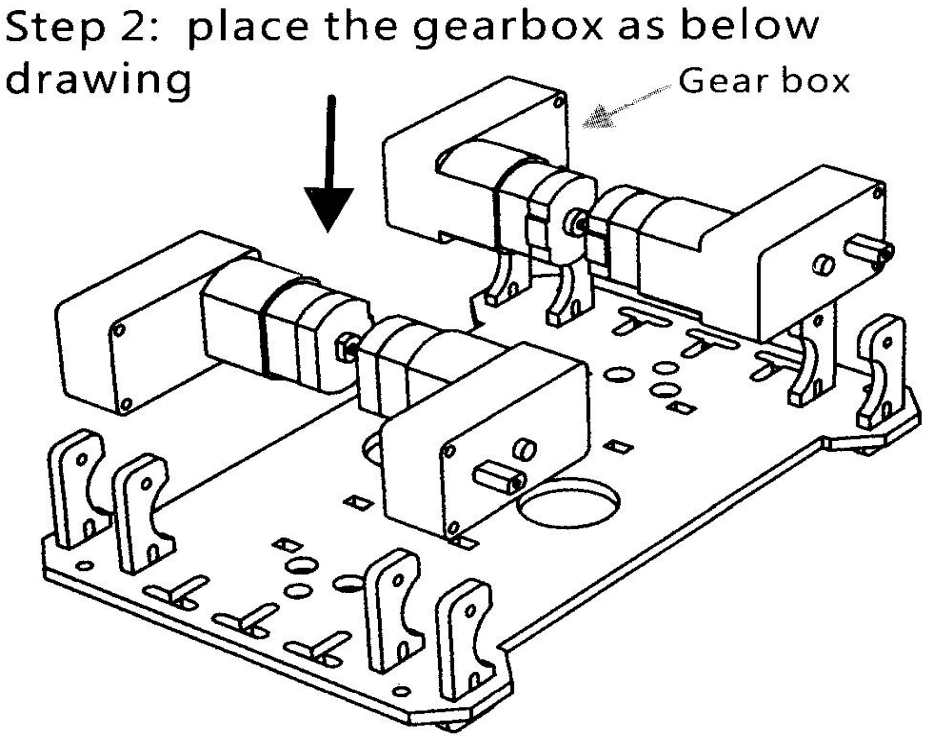


Figure 3-5 Assembly instruction

Now we can use the Soldering iron to solder the dc motor and jumpers because you need to connect the dc motor with motor driver like figure below.

Figure 3-6 Soldering the wires with dc motor

Step3:

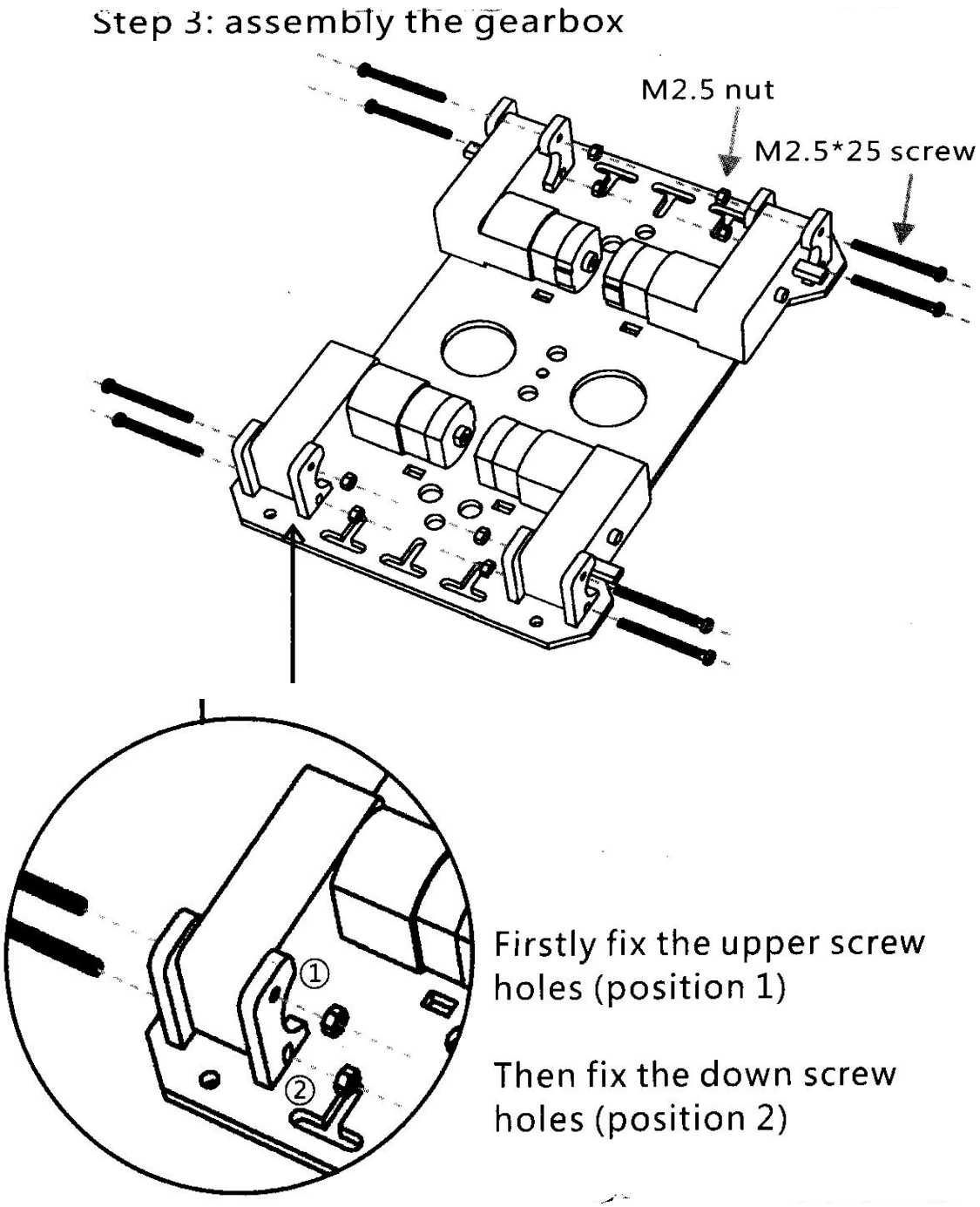


Figure 3-7 Assembly the gear box

Step 4: Assemble the wheel with dc motor and soldering each one with jumpers like figure below:

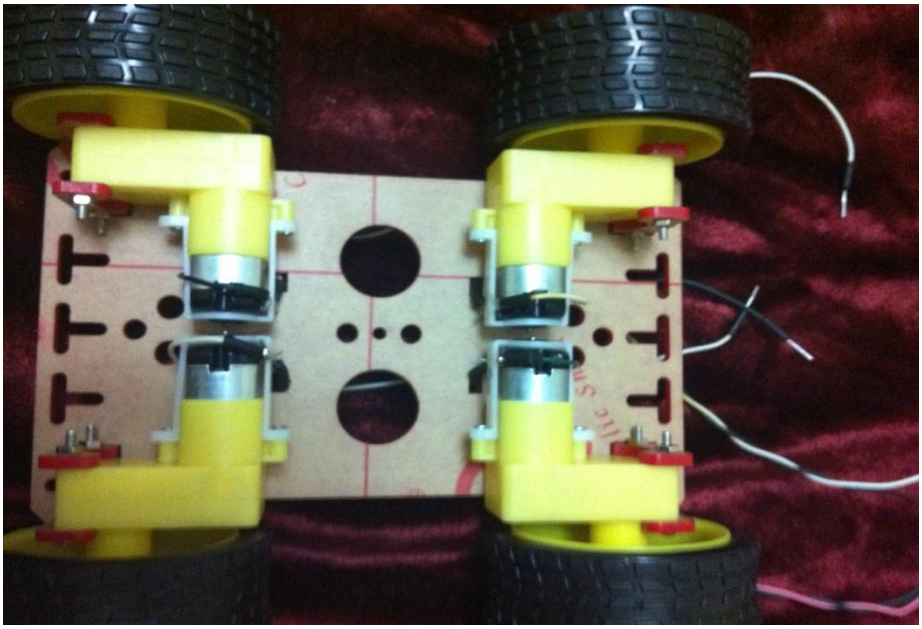


Figure 3-8 assemble the wheel

Step 5: assembly the chassis up to put the kit & shields over it :

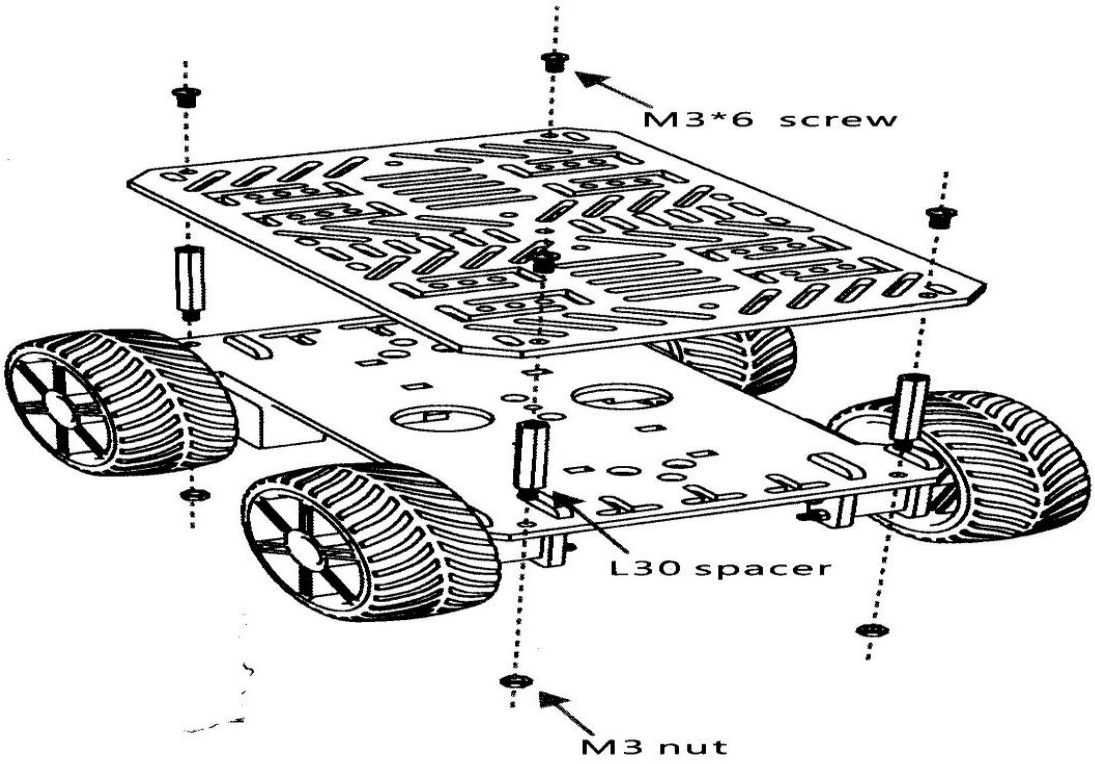


Figure 3-9 assemble the chassis up

Finally, we finished from implemented the chassis of the robot the next task now we want to put every part of the hardware above the chassis and connect it with each other to start the movement.

Attaching Arduino:

Before attaching the Arduino to the robot, cover the bottom of the Arduino with a tie rap to attach it with upper chassis but be carefully from the short circuits that could happen if the Arduino touched metal parts of the boot. We put the arduino in the center of the upper chassis of the robot.

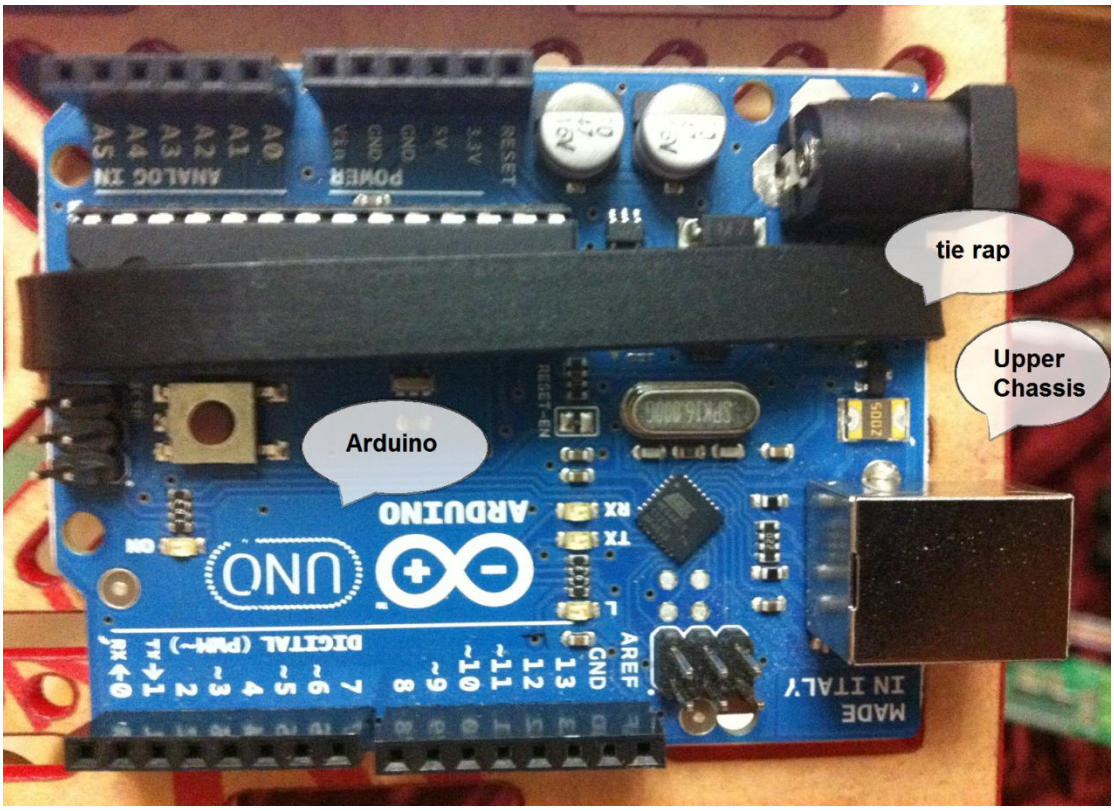


Figure 3-10 attach the arduino over the upper chassis

Attaching Solderless Breadboard:

Remove the adhesive cover from the bottom of the breadboard and stick it in the Backward of the upper chassis robot.

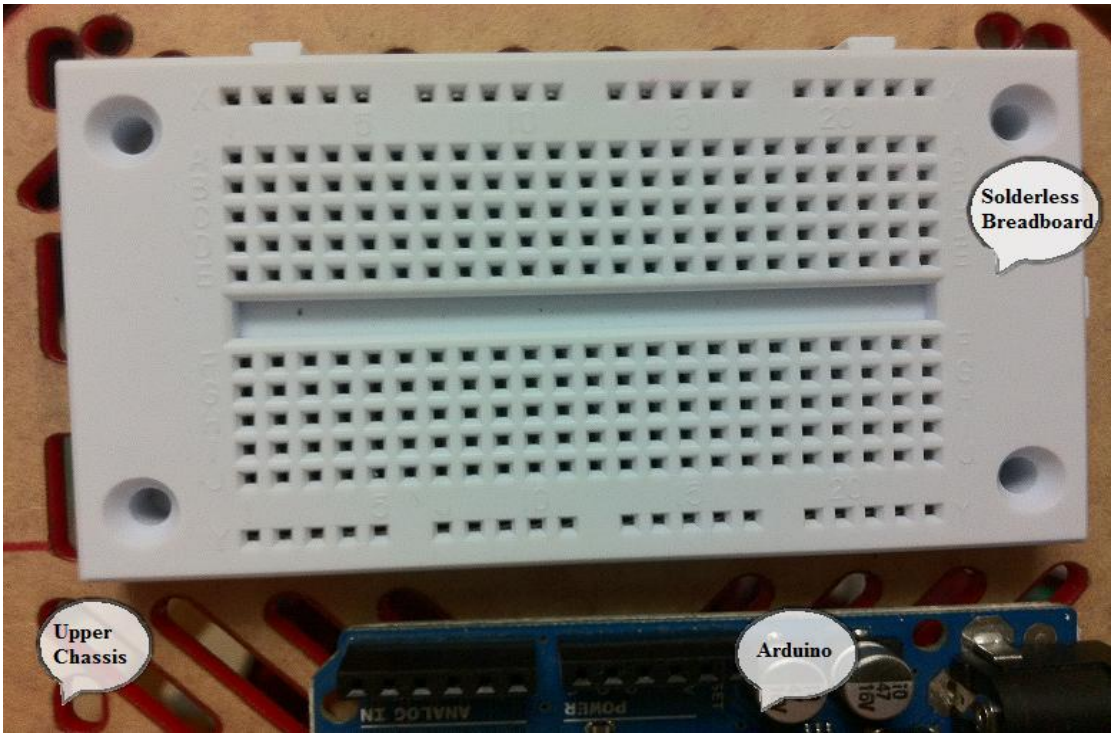


Figure 3-11 attaching the breadboard

Battery Holder:

Use a battery holder tie rap to stick the battery holder in the center of upper chassis beside the arduino.

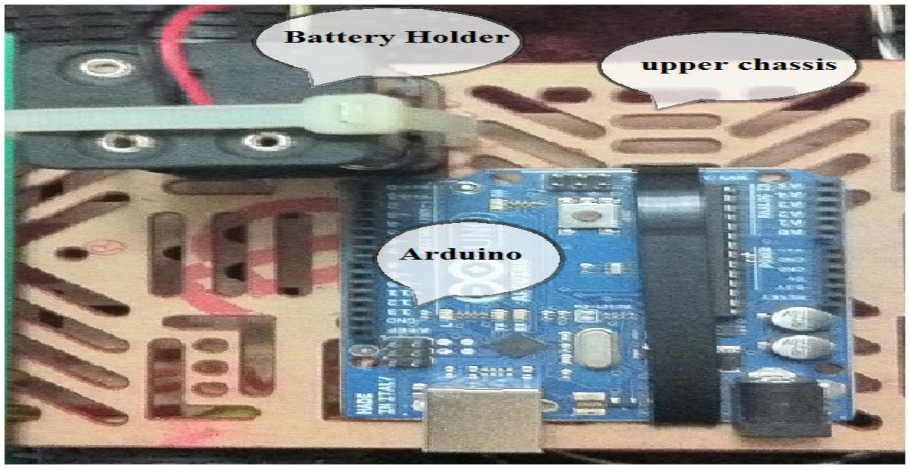


Figure 3-12 the battery holder

Attaching the motor driver:

Use 2 of M 3*6 screws to stick the driver over M 3*6 spacer located in the upper chassis it will be the front of the robot.

Now we finish from implement and put the main parts of the hardware in the robot.

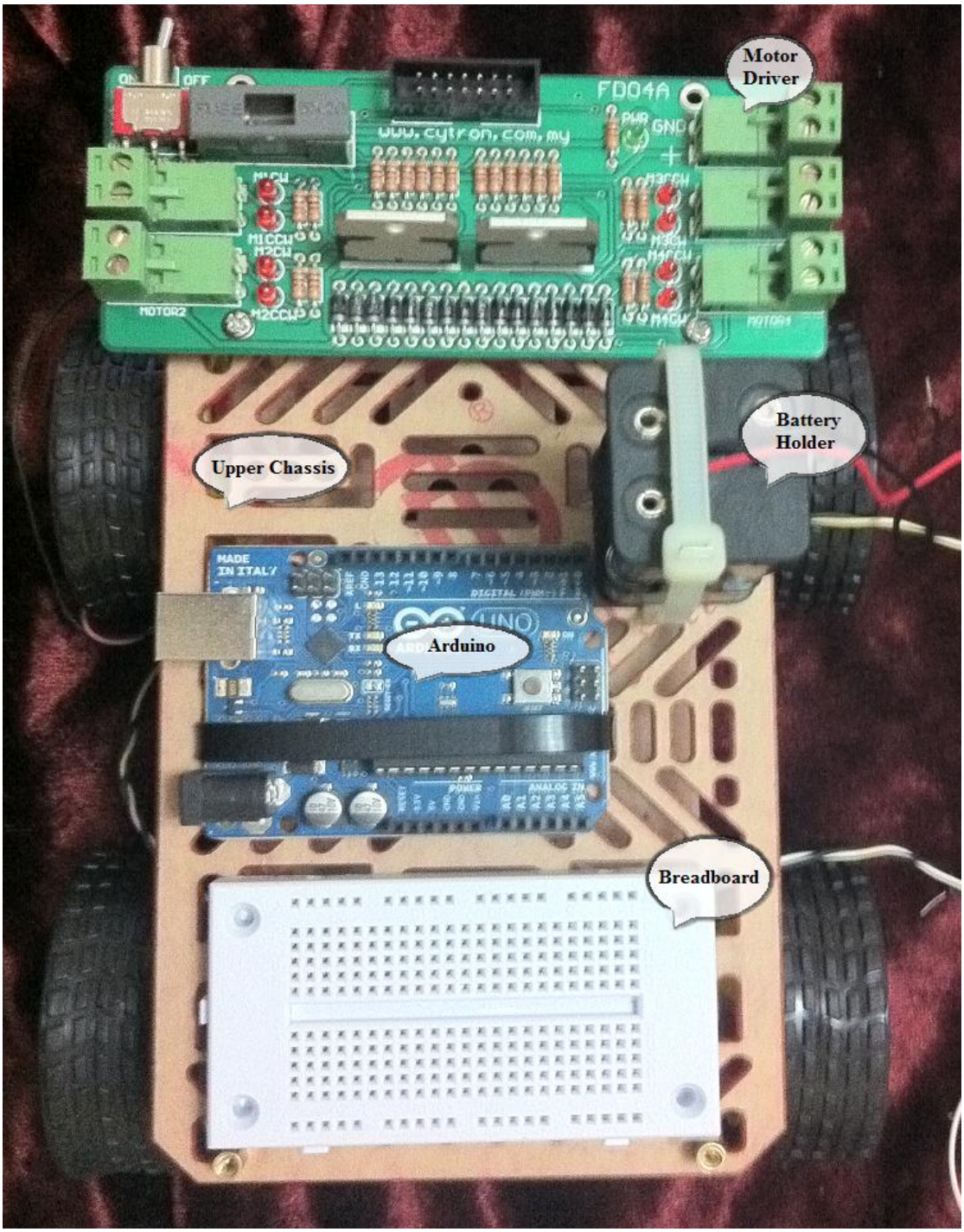


Figure 3-13 the robot after put the hardware parts

The main question now is:

Q: How I can connect all these parts to make the robot move?

A: We must read the data sheet for every kit and study it well to decide how I can connect the kits with each other.

Let us start with the motor driver to decide how we will connect it with arduino the arduino will send the pulses to the motor driver and the driver will control the dc motor. Now we will see the specification of the pin connection of the motor driver by the figure below and the table.

Figure 3-14 show the pin orientation

The function of each pin is described as below:

No.	Name	Function
1	Pin 1	Motor 1 pin 1
2	Pin 2	Motor 1 speed control
3	Pin 3	Motor 1 pin 2
4	Pin 4	Motor 2 pin 1
5	Pin 5	Motor 2 speed control
6	Pin 6	Motor 2 pin 2
7	Pin 7	Motor 3 pin 1
8	Pin 8	Motor 3 speed control
9	Pin 9	Motor 3 pin 2
10	Pin 10	Motor 4 pin 1
11	Pin 11	Motor 4 speed control
12	Pin 12	Motor 4 pin 2
13	Pin 13	5V
14	Pin 14	GND

Figure 3- 15 the function of pin

Referring to **Figure 3- 15,** there are 4 channels of FD04A and each channel has 3 control pins, which are pin 1, pin 2 and speed control. Pin 1 and pin 2 of each channel control the motor activation and direction, while speed control should be given the PWM if speed is needed. If speed is not necessary, the speed control pin is connecting to 5V. The motor will run with full speed.

Pin 1	Pin 2	Motor state
0	0	Brake to Ground
0	1	CW (relative)
1	0	CCW (relative)
1	1	Brake to V motor

Figure 3-16 True tables for pin 1 and pin 2

Lets connect the powe to the motor driver by using the Energizer 9v battery because the max power in the driver = 26 v and the $I_{max} = 3$ A.

Note: This Imax is limited by on board 3A fuse, changing the value of fuse will increase the Imax. However, it will not protect individual H-bridge.

Now you can connect the positive wire from the battery holder in the + pin in the channel and the negative wire in the GND pin in the same channel.

The final step to connect the motor with the channel in the driver let us make convention the driver will be in the front of the robotic car and the left side will be the motor no.1 and motor no.2 , so the right side will be the motor 3 and motor 4.

But wich one will take the Clock Wise move and the other will take the inverse because it is the basics of mechanic we haven't an axis for each 2 motor every motor is individual so we will chose the left side will move clock wise so we make the negative wire from motor 1 and 2 in the pin CW and the right hand side is the inverse as shown in figure below.

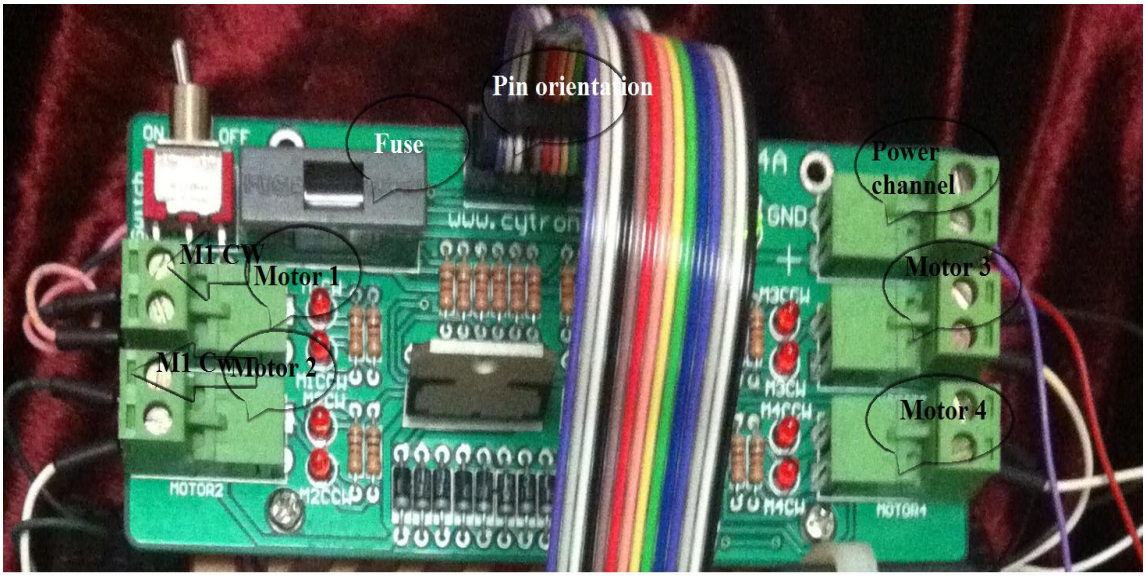
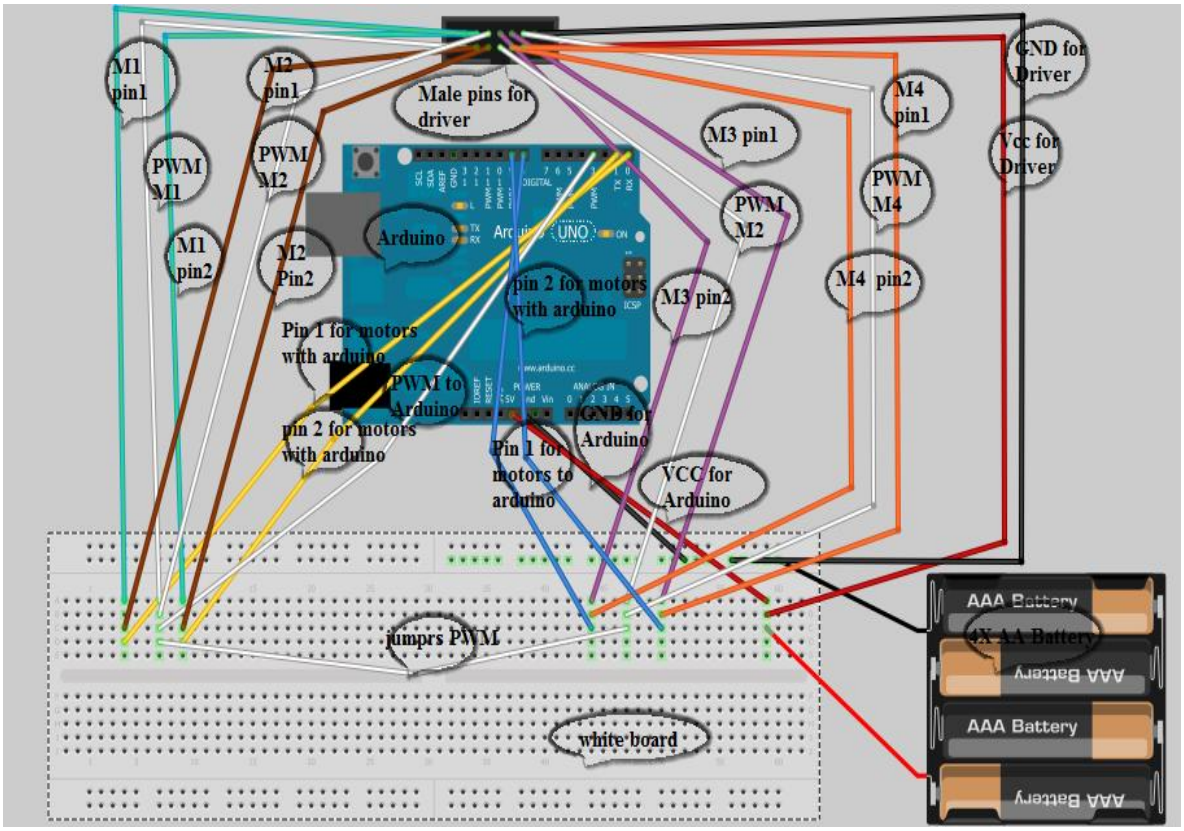


Figure 3-17 the connection of motors and the power

The next step now is to connect the arduino with the motor driver after we revise the last information about the pin orientation and the true table of the pin of each motor we decide to make the motor control speed of all motors is common in the arduino pin we will use the white board to connect the motors pin and the motor speed control.

The figure below shows the connection between arduino and the motor.



The Figure 3-18 Connect arduino with motor driver pins

To control the left side of motors (M1 and M2) by the true table to decide if the motion of the wheels of motors (M1 and M2) will take Clock wise direction or Counter-clockwise by Pin 1 and pin 2 for each motor (M1 and M 2), then you can send the pulses of the speed by PWM so we can make the PWM is common for motor (M1 and M 2) likes the pins 1 and 2.

By the same way for the Right side of the motors (M3 and M4) but take care if we chose the direction of the left side of the robot is Clock wise we chose in the right side (Counter-

clockwise) and the inverse is almost right. This is a mechanical basic of the motion of the wheels.

Bluetooth shield:

The Bluetooth shield can be implement above over the ardunio pins and the arduino pins has a position in the kit of the Bluetooth the figure below is describe this step.
And all the connection between the arduino and the motor driver is the same we don't change anything of the connection wires and jumpers.
After that you can run the Bluetooth but you must increment the data or the code to the arduino by the USB cable and then you make a configuration of the Bluetooth and enjoy the communication wireless.

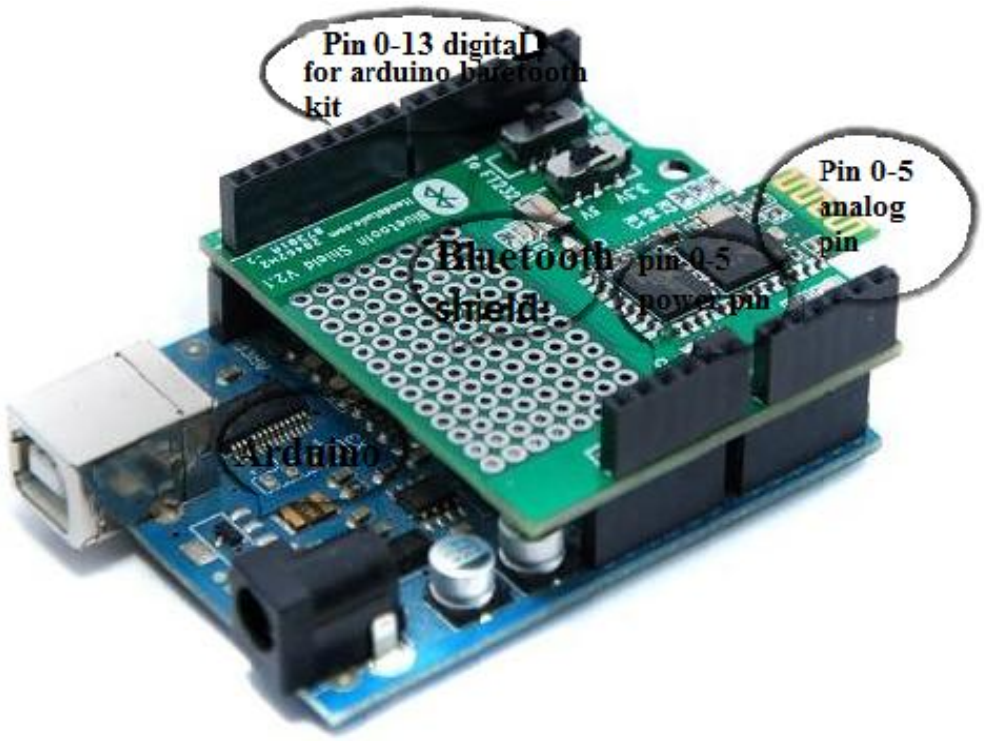


Figure 3-19 belts Bluetooth with arduino

The Arduino IDE:

```
const int m1p1 =0;
const int m1p3 =1;

const int m3p1 =8;
const int m3p3 =9;

const int pwm =3;

int middetector = 13;
int rigdetector = 12;
int lefdetector = 11;
int speed =100;
int turn_speed=255;
```

In the first 2 commands we define ((initialize)) the pin of the motor 1 and motor 2 for arduino so we chose the pin no. 0 for the pin no 1, and pin no.1 in arduino for pin 3 for the motors. ((we make the motor 1 and motor 2 common in the connection between the arduino and the driver)).

In the second 2 commands we define ((initialize)) the pin of the motor 3 and motor 4 for arduino so we chose the pin no. 8 for the pin no 1, and pin no.9 in arduino for pin 3 for the motors. ((we make the motor 1 and motor 2 common in the connection between the arduino and the driver)).

In the third command we define ((initialize)) the pin 2 for each motors((M1 and M2 and M3 and M4)) in pin 3 in the arduino.

To define the line detection pin we use three line so we define everyone in each pin as we write in the code up.

The speed here is about 2 command one for the initialize the speed as 100 and when the robot need to turn to the left or to the right we use the turn speed (max speed) .

```
#include <SoftwareSerial.h>   //Software Serial Port
 define RxD 6#
 # define TxD  7
 # define DEBUG_ENABLED  1
SoftwareSerial blueToothSerial(RxD,TxD);
```

Configuration of Bluetooth shield and the library used to define the shield this code is a formal or standard from the data sheet of the Bluetooth shield to define it in the software.

```
void setup()
}
pinMode(m1p1, OUTPUT);
pinMode(m1p3, OUTPUT);
pinMode(pwm1, OUTPUT);
pinMode(m3p1, OUTPUT);
pinMode(m3p3, OUTPUT);
pinMode(RxD, INPUT)
pinMode(TxD, OUTPUT)
pinMode(middetector, INPUT);
pinMode(rigdetector, INPUT);
pinMode(lefdetector, INPUT);
setupBlueToothConnection()
{
```

Let's take the commands and analysis it and to define every pin as input or output in our system :

```
pinMode(m1p1, OUTPUT);
pinMode(m1p3, OUTPUT);
```

In the first command we define the pin mode for motor 1and 2 pin 1 as output.

And the second is the same but for the pin 3 for each motor 1 and 2.

```
pinMode(pwm1, OUTPUT);
```

In this command we define the pin mode of the PWM as output.

```
pinMode(m3p1, OUTPUT);
```

pinMode(m3p3, OUTPUT);

the same for motor 3 and 4 we define the mode for each pin in the first command we define the pin 1 for the 2 motors as output.

And the second command we define the pin 3 for each 2 motors .

pinMode(RxD, INPUT)

pinMode(TxD, OUTPUT)

the pin mode for a Receiver for Bluetooth shield as input.

And the transmitter of the Bluetooth shield as output.

pinMode(middetector, INPUT);

pinMode(rigdetector, INPUT);

pinMode(lefdetector, INPUT);

Three line detector every on is connected in the pin in the arduino and it used as an input .

setupBlueToothConnection()

The starting of Bluetooth shield to work .

We make a function to make the robot move so we need 5 functions, Now observe with every function and how it works :

void Move_Foreword ()

}

digitalWrite(m1p1, LOW);

digitalWrite(m1p3, HIGH);

digitalWrite(m3p1, LOW);

digitalWrite(m3p3, HIGH);

{

The name of the function is move forward so we will control the 4 wheels to move forward.

In the first two commands we chose the pulse of the pin 1 and pin 3 for the motor 1 and motor 2 the pin 1 is low as zero and the pin 3 is high as one so the direction will be a clock-wise.

For the second two commands we chose the same and the movement is will be counter clock-wise because the wire of right side is connected the inverse of the wire of the left side so the pulse is the same .

```
void Move_Backword()
}
digitalWrite(m1p1, HIGH);
digitalWrite(m1p3, LOW);
digitalWrite(m3p1, HIGH);
digitalWrite(m3p3, LOW);
{
```

The second function is move backward if you read it carefully it will appear to you the inverse of the function of move forward.

```
void Move_Right()
}
digitalWrite(m1p1, LOW)
digitalWrite(m1p3, HIGH)
digitalWrite(m3p1, HIGH)
digitalWrite(m3p3, LOW)
{
```

The third function is to make the robot move to the right so the left side (motor 1 and 2) is take the pulse low for pin 1 and high for pin 3the the movement will be counter clock wise. The right side will be the inverse and the movement will be clock wise so the robot will make the left side as a centre of mass and move around it to move to the right .

```
void Move_Left()
}
digitalWrite(m1p1, HIGH);
digitalWrite(m1p3, LOW);
digitalWrite(m3p1, LOW);
digitalWrite(m3p3, HIGH);
{
```

This function is move to the left if you observe with me it is the inverse of the function of move right we make the right side as a centre of mass for the robot and the left side move inverse of it.

```
void stop()
}
digitalWrite(m1p1, LOW);
digitalWrite(m1p3, LOW);
digitalWrite(m3p1, LOW);
digitalWrite(m3p3, LOW);
{
```

The last function we make it to stop the robot and the command is clearly we make all the pin has a low pulse to make the wheel stoped.

```
void setupBlueToothConnection()
}
  blueToothSerial.begin(38400); //Set BluetoothBee BaudRate to default baud rate 38400
  blueToothSerial.print("\r\n+STWMOD=0\r\n"); //set the bluetooth work in slave mode
  blueToothSerial.print("\r\n+STNA=SeeedBTSlave\r\n"); //set the bluetooth name as
"SeeedBTSlave"
  blueToothSerial.print("\r\n+STOAUT=1\r\n"); // Permit Paired device to connect me
  blueToothSerial.print("\r\n+STAUTO=0\r\n"); // Auto-connection should be forbidden
here
  delay(2000); // This delay is required.
  blueToothSerial.print("\r\n+INQ=1\r\n"); //make the slave bluetooth inquirable
Serial.println("The slave bluetooth is inquirable !");
  delay(2000); // This delay is required.
  blueToothSerial.flush();
{
```

The code up is taken from the Bluetooth shield data sheet to complete the configure of the Bluetooth device .

The main(Loop) now is :

```
void loop()
}
```

```
 int recvChar ;
if(blueToothSerial.available() )
 { //check if there's any data sent from the remote bluetooth shield
recvChar = blueToothSerial.read()
 if(recvChar= ='s') speed =100;
   else  if(recvChar= ='m') speed =120;
   else  if(recvChar= ='h') speed =150;
  {
   analogWrite(pwm1,speed);
   while(HIGH = = digitalRead(middetector))
}
     Move_Foreword();
     analogWrite(pwm1,speed);
{
  Stop();
   while(HIGH = = digitalRead(rigdetector))
}
     Move_Right();
     analogWrite(pwm1,turn_speed);
{
   Stop();
    while(HIGH == digitalRead(lefdetector))
}
     Move_Left();
     analogWrite(pwm1,turn_speed);
{
   Stop();
{//
{
```

To understand now the main loop of the arduino

we want to make the robot read the characters of the Bluetooth and send it to the arduino then the Bluetooth receive and read and begin as a serial port.

To control the speed of the Neruosky we divide the speed into 3 parts small(S), Medium (m) and high (h) when the arduino receive the characters the wheel will take the speed that's changed by the attention ratio readied by the nerusky.

The default or the initialize speed will begin as we make it 100 and we write the command analogwrite (pwm,speed); to start the default motion of the robot.

The robot needed to turn to the left or to the right , the line detector can do this mission by reading the line sector in the terminal we put the robot on it.So we use the while loop to make the robot turn if the middle line detector read black (High) the robot will move forward and if the sensor read wait the robot will stop, then the left line detector and the right one will begin to start to check if the line is below on of them it will move to the right or to he left and if any one of the left or the right not read the black (High) it will stopby the function stop and the others will check if the line below one of them to take the function to move .

The C # Code and the Interface of the system :

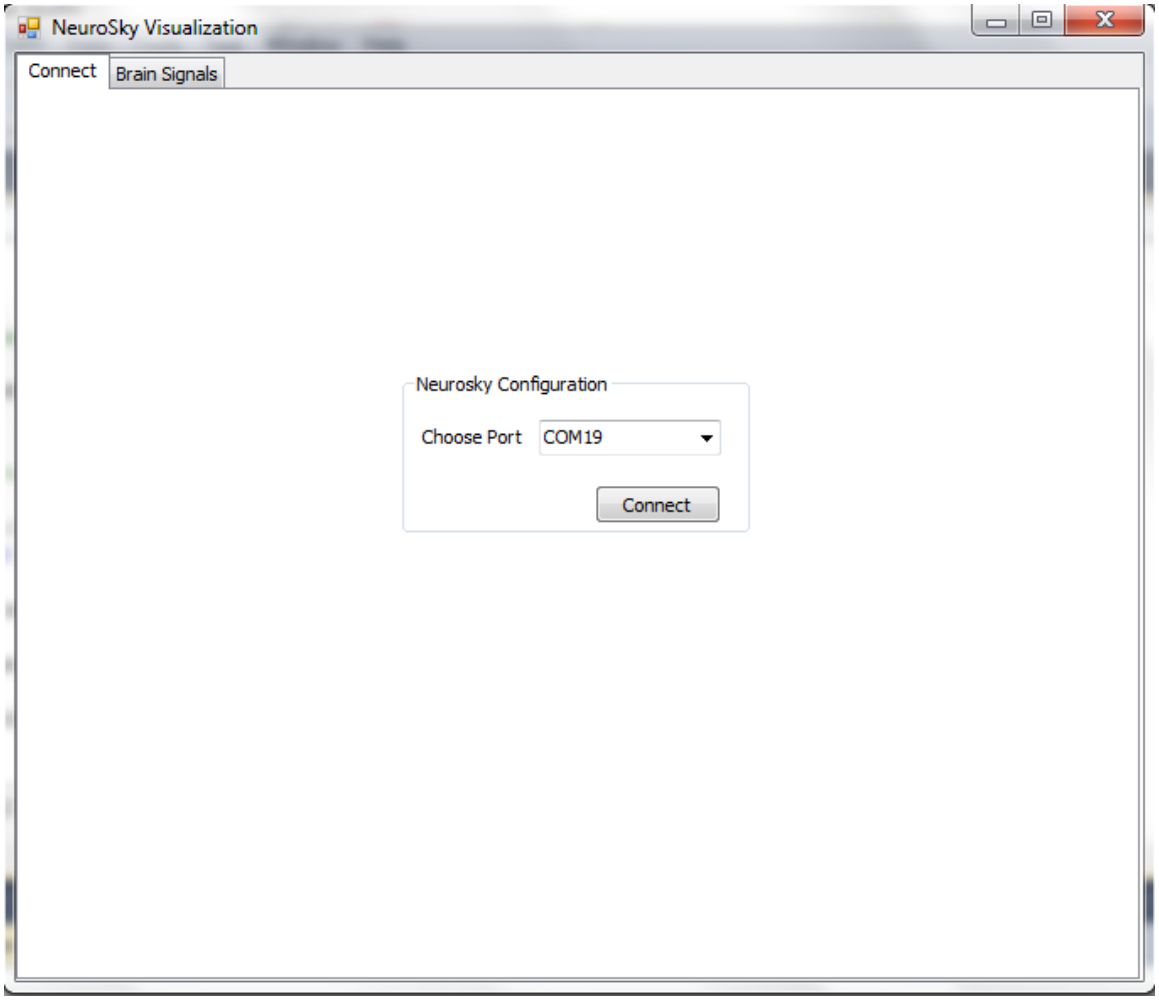


Figure 4-1 the interface of the software

When the software is start in the first we need to manually configure the Bluetooth of the arduino and the dongle of the nerousky to select the port of the Bluetooth dongle of the Neruosky then we click connect to go to the second panel the brain signals to start show the brain signals shown in the figure belwo.

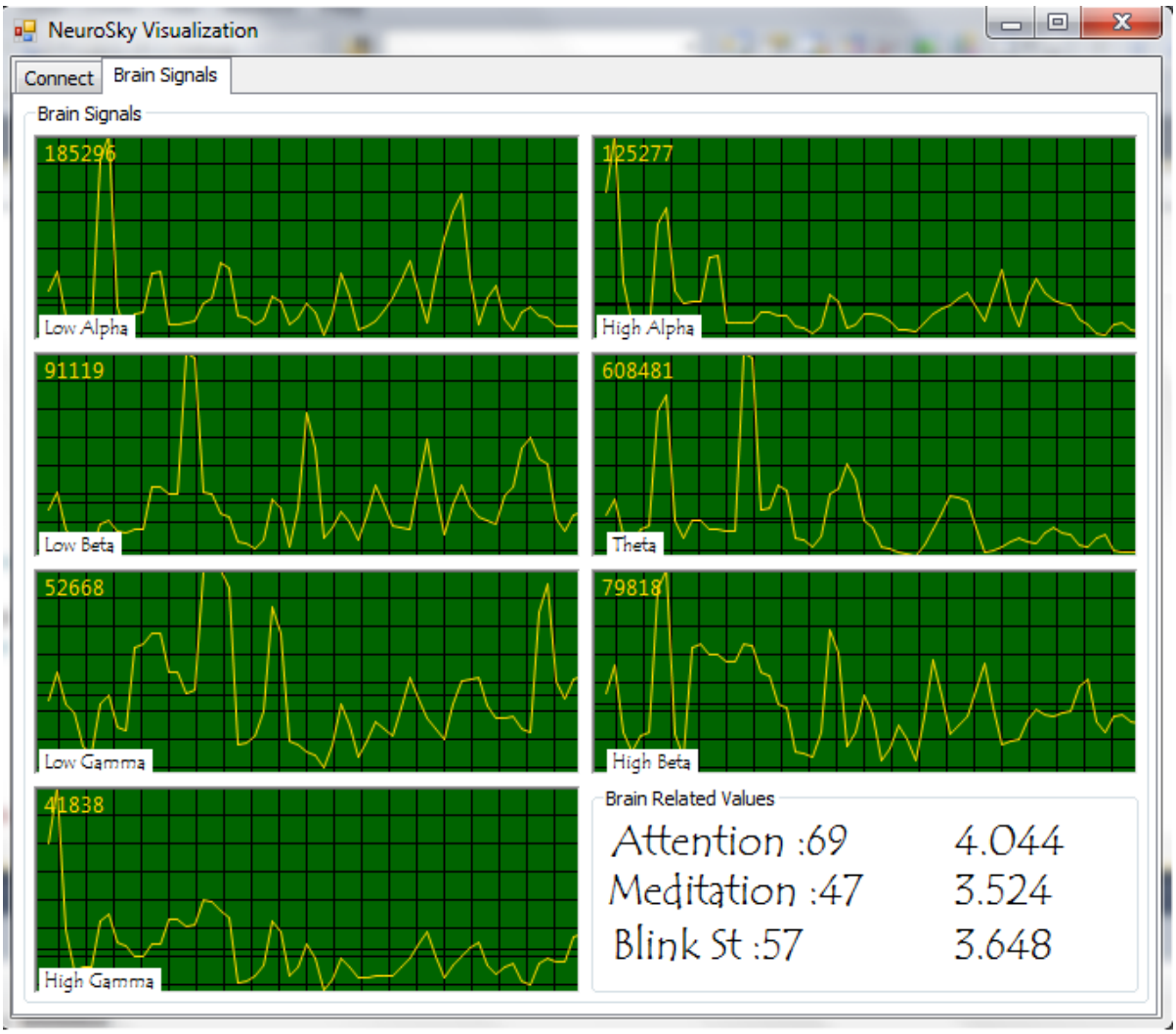


Figure 4-2 the interface of the brain signals and attention ratio

Now in this figure we have the interface of the brain signal and the attention ratio we use it as the speed in our project .

The Code of the C#:

We must copy the file of the the thinkgear.dll in the debug folder of the project to use the Neruosky packet and library.

From Nerusky Library we get the Class `ThinkGearWrapper`

And after that we can call any object in this class in our interface we use the pref chart to show the brain signals.

The library we added in the project is :

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using ThinkGearNET;
using System.IO.Ports;
using System.IO;

using System.Threading;
```

Now we must make a timer or trigger to select the delay that take the new value of the neruo sky after that we can read every value from the packet and make any interface you want to show to the user.

```
private void timer1_Tick(object sender, EventArgs e)
      {

lalpha.AddValue(Convert.ToDecimal(_thinkGearWrapper.ThinkGearState.Alpha1));
theta.AddValue(Convert.ToDecimal(_thinkGearWrapper.ThinkGearState.Theta));
halpha.AddValue(Convert.ToDecimal(_thinkGearWrapper.ThinkGearState.Alpha2));
 lbeta.AddValue(Convert.ToDecimal(_thinkGearWrapper.ThinkGearState.Beta1));
 hbeta.AddValue(Convert.ToDecimal(_thinkGearWrapper.ThinkGearState.Beta2));
lgamma.AddValue(Convert.ToDecimal(_thinkGearWrapper.ThinkGearState.Gamma1));
hgamma.AddValue(Convert.ToDecimal(_thinkGearWrapper.ThinkGearState.Gamma2));
att.Text ="Attention :"+ _thinkGearWrapper.ThinkGearState.Attention.ToString();
med.Text = "Meditation :" + _thinkGearWrapper.ThinkGearState.Meditation.ToString();
blink.Text = "Blink St :" + _thinkGearWrapper.ThinkGearState.BlinkStrength.ToString();
```

```
atten += _thinkGearWrapper.ThinkGearState.Attention;
mede += _thinkGearWrapper.ThinkGearState.Meditation;
bl += _thinkGearWrapper.ThinkGearState.BlinkStrength;

 }
```

The next step now is to select the ratio of the attention and divide it int 3 parts small(S), Medium (m) and high (h) to decide the speed of the wheel and .

```
if (_thinkGearWrapper.ThinkGearState.Attention > 0 &&
_thinkGearWrapper.ThinkGearState.Attention < 30)
        {
           arduino.Write("s");

        }
 else if (_thinkGearWrapper.ThinkGearState.Attention > 30 &&
          _thinkGearWrapper.ThinkGearState.Attention < 70)
        {
           arduino.Write("m");

        }
else if (_thinkGearWrapper.ThinkGearState.Attention > 70 &&
           _thinkGearWrapper.ThinkGearState.Attention < 100)

        {
           arduino.Write("h");

        }
```

Conclusion and future work:

In the educational uses:

- **In this project we can use a voice recognition to send the function of movement by speech and use Neruosky to control the speed.**
- **Use the Emotive EBook to control the directions of the robot by using malty sensors to make all the directions and the speed by the brain signals .**
- **Use the brain computer interface to control the robotic Arm by Neruosky or Emotive EBook.**

In the entertainment uses:

- **Make an intelligence video games**
- **By using malty sensors you can make a brain browser.**

References:

Books:

- M. Benzi, "Getting started with Arduino", O'Reilly, 2009.
- T.Karvinen,K.Karvinen, "Make a Mind-Controlled Arduino Robot", O'Reilly, 2012
- A.Stellman, J.Greene, " Head First C# ", O'Reilly,2010.
- M.Kurz, W.Almer, F.Landolt, "Brain Computer Interface" , 2006.
- Flexibot Driver User's Manual Prepared by Cytron Technologies Sdn. Bhd. 19, Jalan Kebudayaan 1A,
- "Arduino basic C review", Nuno Alves college of Engineering, Western New England University.
- K.Crowley, A.Sliney, I.Pitt, D.Murphy," Evaluating a Brain-Computer Interface to Categorise Human Emotional Response", IEEE, 2010
- Kristín Guðmundsdóttir,"Improving Players' Control Over The NeuroSky Brain Computer-Interface",School of Computer Science,2011

Websites:

- www.arduino.cc
- http://www.neurosky.com/
- http://www.oreilly.com.
- http://my.safaribooksonline.com./
- http://en.wikipedia.org/wiki/Brain%E2%80%93computer_interface#EEG
- http://www.seeedstudio.com/wiki/Grove_-_Line_Finder
- http://www.seeedstudio.com/wiki/Grove_-_Serial_Bluetooth
- http://www.electrical-neuroimaging.ch/publications/grave_esann2009.pdf