Università degli Studi di Torino

Dipartimento di Informatica

Using Arduino for Tangible Human Computer Interaction

Fabio Varesano

Advisor: Prof. Luca Console
Co-Advisor: Prof. Marco Grangetto

Laurea Magistrale in Metodologie e Sistemi informatici

April 2011

Abstract

This thesis presents the results of a nine months internal stage at the Department of Computer Science, Università degli Studi di Torino.

During my stage, supervised by Prof. Luca Console, I experienced with electronics, Arduino, micro-electromechanical sensors (accelerometers, gyroscopes and magnetometers), orientation sensing algorithms and 3D computer graphics to develop prototypes of Human Computer Interaction devices, with a a particular interest on Tangible User Interfaces.

`Document generated on Friday 8th April, 2011.`

To my family which always supported me through the University years.

I probably wouldn't be writing this thesis without their help.

Contents

List of Figures

LIST OF FIGURES

1

Introduction

This thesis aimes to experience with electronics, Arduino, MEMS sensors and 3D graphics to prototype novel human computer interaction approaches.

As a computer scientist, I started this work with only a very limited electronics knowledge, just some memories from the high school days. In a period of 9 months, I progressively improved my electronics knowledge till the designing of quite complex printed circuit boards which have been used intensively in the human computer device prototyping.

The Arduino prototyping platform has been used during this project. During my university background, we had a very limited education on low level programming and never had the possibility to program a microcontroller. During this project, I gradually became expert in Arduino functions and programming features.

I've also experienced also with MEMS sensors, mainly accelerometers, gyroscopes and magnetometers used for orientation sensing. I also had to study and understand progressively complex orientation sensing algorithms which fuse the data coming from the sensors to implement robust attitude and heading sensing.

The knowledge gained in electronics, Arduino, MEMS sensors and orientation sensing has been crucial in the prototyping of Palla, a spherical tangible user interface capable of orientation sensing.

Finally, I developed Femtoduino, an ultra small Arduino compatible board. In this

thesis, practical usages of Femtoduino are not reported but this device, thanks to its very limited size and weight, really has amazing possibilities when used in prototyping.

This thesis is outlined as follows:

Chapter 1 a simple introduction to the thesis

Chapter 2 offers an introduction to electronics useful for the reader who never had an education on it

Chapter 3 describes the Arduino electronics prototyping platform

Chapter 4 provides a description of what have been my first steps in using Arduino, both from an electronic and programming point of view

Chapter 5 introduces MEMS accelerometers, gyroscopes and magnetometers and the sensors used

Chapter 6 threats the problem of orientation sensing from a mathematical point of view

Chapter 7 describes FreeIMU, a 9 degrees of measurement IMU developed during this thesis

Chapter 8 describes Palla, one of the prototypes of tangible user interfaces developed

Chapter 9 introduces Femtoduino, an ultra small Arduino compatible board developed during this thesis for usage in size constrained Arduino prototyping

Chapter 10 gives my personal conclusion on this thesis.

1.0.1 How everything got started

What you will read in this thesis started back in May 2010. I had just finished my last exam for the Master of Science and I was visiting professors asking what kind of projects were they involved with and if they could follow me as project supervisors.

I have to say that most of my last exams were pretty hard. I'm usually quite good in both theoretical and practical exams but between them I do prefer the practical ones as I enjoy getting my hand dirty with programming when this involves complex problems.

Unfortunately tough, almost all of my last exams were theoretical so that I finished the last one almost exhausted from how much complex computer science theories my somehow limited brain had to store.

So, when I started visiting professors I was quite terrified by the fact that most of the projects they were proposing were all on the theoretical side of computer science. I rally like math, theorems, demonstrations and so on but I wanted something different for my final project. I wanted to get my hand *dirty*.

When I first met Professor Luca Console, he explained to me what were his current research projects (that was the first time I met him, never had him as teacher in my University background): some of them were pretty cool projects but nothing really caught my interests until he said something like: "Oh.. yeah.. I forgot. We also bought this thing", showing me something resembling a little PC motherboard, "It's called Arduino: it's an electronics prototyping platform. Lot of people are using it to do cool things all around the world and we would like to start exploring its possibilities. But that would require you to get some knowledge of electronics and you will probably have to get your hands dirty with wires, buttons, led, motors and stuff like that.. You will also probably have to solder" .... Bingo!

This was **exactly** what I wanted for my final project and thesis. He then gave me an Arduino board, an Arduino starter kit and I was ready to go.

Well, I have to say that the following months have been pretty hard but really satisfactory. I had to learn lot of stuff for which I never received a formal education: I messed with electronics, Arduino, soldering irons, chemicals and lot of things more. But, writing this at the end of this path, it's been really satisfactory and in the end it worth it.

2

Electronic Circuits in DC

The following chapter introduces some basic electronics concepts and theories that will be used deeply in the next chapters. The reader who never had an electronics education or if such education happened a long time ago can use the following pages as a fast and easy *cheat sheet* useful for the next chapters.

For a more deep coverage of the concepts introduced below, (27) is a good quality and libre book on these topics.

2.1 Basic concepts of electricity

Electricity is the flow of electrons in a conductor. It can be characterized by four quantities: voltage, current, resistance and power.

The voltage refers to the level of energy electrons have relative to some reference point (often called ground in a circuit). The higher the voltage, the more energy electrons have to do work as they travel through the circuit. In general, if two points are at a different voltage relative to each other, electricity will flow from one to the other if they are connected by something that conducts electricity. Voltage is usually represented by the letter E or V. The basic unit of measure is volts (V).

The current is an expression of how much charge is traveling through the conductor

per second. The unit of measurement for current is the Ampere (amp, A), defined as

$$1A = 1\frac{C}{s} \tag{2.1}$$

meaning that for every Ampere, there is a Coulomb (6.25×10^{18} electrons) of charges moving past a point every second. Voltage and current are separate things: you can have a very small current at a very high voltage, a huge current at a very high voltage.

Resistance is an expression of the degree to which electron flow will be impeded through a conductor. The unit is the Ohm (Ω). In simple circuits resistance determines the relation between voltage and current. At the extremes, a short piece of wire will have a resistance of nearly zero Ohms, while an air gap (for example in an open switch) has very large resistance (millions of Ohms). Intuitively a couple of relationships will hold: in a conductor, a voltage difference between the two ends will cause a current to flow. How much current will be determined by how much resistance the conductor offers. If there's less resistance more current will flow. In fact, given a power source of high enough capacity, if you half the resistance, you will double the current. Conversely, if you double the resistance, you will half the current.

The final quantity is power. The unit of power is the Watt. It's an expression of the overall energy consumed by a component. It is worked out by multiplying the voltage and the current together: $P = VI$ (29, 61).

2.2 Schematic Diagrams

A schematic diagram shows how each component in a circuit connects with another. It is a simple and easy to read outline of the circuit. Each type of component has a unique symbol and a name. All relevant values and component specific information are usually included.

Figure 2.1 is an example of a schematic diagram. It has 3 components: a battery (B1 - 2 horizontal lines) an LED (D1 - the triangle in the circle) and a resistor (R1 - the wavy lines).

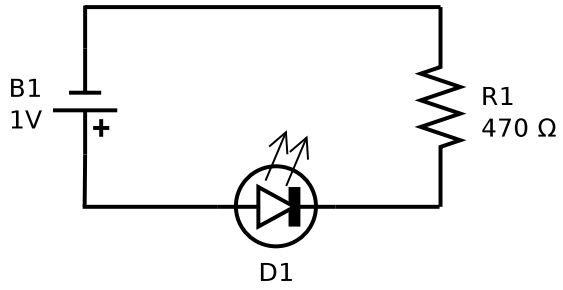

Figure 2.1: Example of a circuit schematic diagram

2.3 Ohm's law

Ohm's law is one of the most important concepts in electronics. Ohm's law states that the current through a conductor between two points is directly proportional to the potential difference or voltage across the two points, and inversely proportional to the resistance between them (74).

The mathematical equation that describes this relationship is:

$$V = IR \tag{2.2}$$

This expression can be rearranged algebraically as follows:

$$I = \frac{V}{R} \qquad R = \frac{V}{I} \tag{2.3}$$

2.4 Capacitors

A capacitor consists of two conductors separated by a non-conductive region called the dielectric medium though it may be a vacuum or a semiconductor depletion region chemically identical to the conductors. A capacitor is assumed to be self-contained and isolated, with no net electric charge and no influence from any external electric field. The conductors thus hold equal and opposite charges on their facing surfaces,[9] and the dielectric develops an electric field. In SI units, a capacitance of one farad means that one coulomb of charge on each conductor causes a voltage of one volt across the device (66).

The capacitor is a reasonably general model for electric fields within electric circuits. An ideal capacitor is wholly characterized by a constant capacitance C, defined as the ratio of charge ±Q on each conductor to the voltage V between them:

$$C = \frac{Q}{V} \tag{2.4}$$

Work must be done by an external influence to "move" charge between the conductors in a capacitor. When the external influence is removed the charge separation persists in the electric field and energy is stored to be released when the charge is allowed to return to its equilibrium position. The work done in establishing the electric field, and hence the amount of energy stored, is given by:

$$W = \int_{q=0}^{Q} V \mathrm{d}q = \int_{q=0}^{Q} \frac{q}{C} \mathrm{d}q = \frac{1}{2}\frac{Q^2}{C} = \frac{1}{2}CV^2 = \frac{1}{2}VQ \tag{2.5}$$

The current $i(t)$ through any component in an electric circuit is defined as the rate of flow of a charge $q(t)$ passing through it, but actual charges, electrons, cannot pass through the dielectric layer of a capacitor, rather an electron accumulates on the negative plate for each one that leaves the positive plate, resulting in an electron depletion and consequent positive charge on one electrode that is equal and opposite to the accumulated negative charge on the other. Thus the charge on the electrodes is equal to the integral of the current as well as proportional to the voltage as discussed above. As with any antiderivative, a constant of integration is added to represent the initial voltage $v(t_0)$. This is the integral form of the capacitor equation,

$$v(t) = \frac{q(t)}{C} = \frac{1}{C}\int_{t_0}^{t} i(\tau)\mathrm{d}\tau + v(t_0) \tag{2.6}$$

Taking the derivative of this, and multiplying by C, yields the derivative form,

$$i(t) = \frac{\mathrm{d}q(t)}{\mathrm{d}t} = C\frac{\mathrm{d}v(t)}{\mathrm{d}t} \tag{2.7}$$

2.5 Kirchhoff's circuit laws

Kirchhoff's circuit laws are two equalities that deal with the conservation of charge and energy in electrical circuits, and were first described in 1845 by Gustav Kirchhoff.

Widely used in electrical engineering, they are also called Kirchhoff's rules or simply Kirchhoff's laws (69).

Kirchhoff's current law states that at any node (junction) in an electrical circuit, the sum of currents flowing into that node is equal to the sum of currents flowing out of that node (figure 2.2).

Recalling that current is a signed (positive or negative) quantity reflecting direction towards or away from a node, this principle can be stated as:

$$\sum_{k=1}^{n} I_k = 0 \tag{2.8}$$

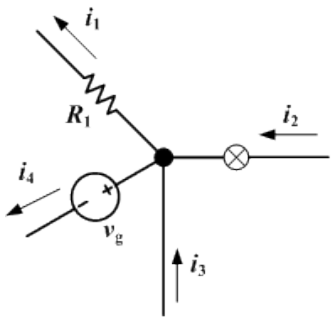

Figure 2.2: Example of Kirchhoff's current law - The current entering any junction is equal to the current leaving that junction. $i_1 + i_4 = i_2 + i_3$. Picture from (69)

Kirchhoff's voltage law states that the directed sum of the electrical potential differences (voltage) around any closed circuit is zero (figure 2.3). It can be stated as:

$$\sum_{k=1}^{n} V_k = 0 \tag{2.9}$$

2.6 Series And Parallel Circuits

Components of an electrical circuit or electronic circuit can be connected in many different ways. The two simplest of these are called series and parallel and occur very frequently. Components connected in series are connected along a single path, so the

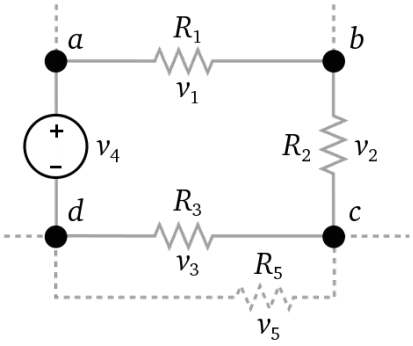

Figure 2.3: Example of Kirchhoff's voltage law - The sum of all the voltages around the loop is equal to zero. $v_1 + v_2 + v_3 - v_4 = 0$. Picture from (69)

same current flows through all of the components. Components connected in parallel are connected so the same voltage is applied to each component.

A circuit composed solely of components connected in series is known as a series circuit; likewise, one connected completely in parallel is known as a parallel circuit.

In a series circuit, the current through each of the components is the same, and the voltage across the components is the sum of the voltages across each component. In a parallel circuit, the voltage across each of the components is the same, and the total current is the sum of the currents through each component (76).

2.6.1 Series circuits

The total resistance of **resistors** in series is equal to the sum of their individual resistances:

$$R_{\text{total}} = R_1 + R_2 + \cdots + R_n \tag{2.10}$$

Figure 2.4: Resistors and Capacitors in series - Picture from (76)

The total capacitance of **capacitors** in series is equal to the reciprocal of the sum of the reciprocals of their individual capacitances:

$$\frac{1}{C_{\text{total}}} = \frac{1}{C_1} + \frac{1}{C_2} + \cdots + \frac{1}{C_n} \tag{2.11}$$

2.6.2 Parallel circuits

The current in each individual **resistor** is found by Ohm's law. Factoring out the voltage gives :

$$I_{\text{total}} = V\left(\frac{1}{R_1} + \frac{1}{R_2} + \cdots + \frac{1}{R_n}\right) \tag{2.12}$$

To find the total resistance of all components, add the reciprocals of the resistances R_i of each component and take the reciprocal of the sum. Total resistance will always be less than the value of the smallest resistance:

$$\frac{1}{R_{\text{total}}} = \frac{1}{R_1} + \frac{1}{R_2} + \cdots + \frac{1}{R_n} \tag{2.13}$$

For only two resistors, the unreciprocated expression is reasonably simple:

$$R_{\text{total}} = \frac{R_1 R_2}{R_1 + R_2} \tag{2.14}$$

For N equal resistors in parallel, the reciprocal sum expression simplifies to:

$$\frac{1}{R_{\text{total}}} = \frac{1}{R} \times N \quad \Longleftrightarrow \quad R_{\text{total}} = \frac{R}{N} \tag{2.15}$$

To find the current in a component with resistance R_i, use Ohm's law again:

$$I_i = \frac{V}{R_i} \tag{2.16}$$

The components divide the current according to their reciprocal resistances, so, in the case of two resistors,

$$\frac{I_1}{I_2} = \frac{R_2}{R_1} \tag{2.17}$$

The total capacitance of **capacitors** in parallel is equal to the sum of their individual capacitances:

$$C_{\text{total}} = C_1 + C_2 + \cdots + C_n \tag{2.18}$$

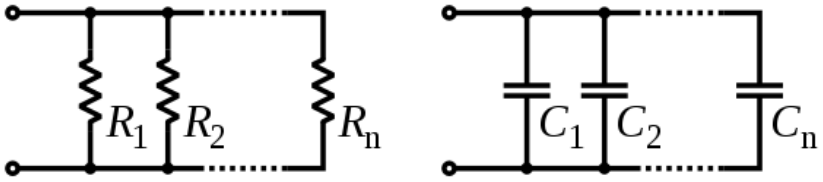

Figure 2.5: Resistors and Capacitors in parallel - Picture from (76)

3

Arduino

This chapter introduce the reader to Arduino, the electronics prototyping platform which has been used during the developments described in this thesis.

3.1 What is Arduino?

Arduino, according to Massimo Banzi, one of its creators, 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* (2, Chapter 1).

Personally, I do embrace Richard Stallman's and Free Software Foundation position (54) on how to name software which respects the user's freedoms to run, study, change and distribute the original and user modified software program (58). So I usually prefer to refer to such software, instead of using the misleading *Open Source Software* naming, with the term *Libre Software* or *Free Software* so that the reader can clearly understand that the importance is given to the freedoms, not only to the access of the source code.

Given the considerations made for the software and porting them into the hardware world, I'd rather prefer using the term *Libre Hardware* rather than *Open Source Hardware* so that it's clear that we are more concerned about the freedoms given to the user by using Libre Hardware rather than the open access to the hardware designs.

I'd also would like to note the fact that most of the success of Arduino is due to a

thrilling community of developers, hackers, hobbyists which contribute code, documentations, guides on the arduino.cc and other websites.

So, in my opinion, a better definition of Arduino would be: *a libre hardware physical computing platform based on a simple input/output (I/O) board, a development environment that implements the Processing language and a community of users which share their efforts and knowledge in their Arduino based projects.*

3.1.1 Why Arduino?

There are many hardware prototyping platforms available but Arduino is a good choice as:

- It is a libre hardware and software project, so both software and hardware are extremely accessible and very flexible and they can easily be customized and extended
- It is flexible, offers various digital and analog inputs, SPI, I^2C, a serial interface and digital and PWM outputs
- It is easy to use, it connects to a computer via USB and communicates using the standard serial protocol, runs in standalone mode and as an interface connected to PC/Macintosh computers
- It is inexpensive, less than 30 euro per board and comes with free development environment
- It is backed up by a growing on-line community, lots of source code is already available and ready to be used (77).

I should also note that most of the developers of Arduino are based in Ivrea, just 40 minutes from Torino where we are located: contacting, networking and collaborate with them in the future should be pretty easy.

3.1.2 What can we do with Arduino?

Arduino is a great tool for developing interactive objects, taking inputs from a variety of switches or sensors and controlling a variety of lights, motors and other outputs. Arduino projects can be stand-alone or they can be connected to a computer using USB. The Arduino will be seen by the computer as a standard serial interface (do you remember the COM1 on Windows?). There are serial communication APIs on most programming languages so interfacing Arduino with a software program running on the computer is pretty straightforward.

3.2 Arduino Hardware

The Arduino board is a microcontroller board, which is a small circuit (the board) that contains a whole computer on a small chip (the microcontroller). There are different versions of the Arduino board: they are different in components, aim and size, etc. Some examples of Arduino boards are: Arduino Duemilanove/UNO, Arduino Mega, Arduino Nano, Arduino Mini. Arduino schematics are distribute using an open license so anyone is free to build his own Arduino compatible board. The Arduino name is a registered trademark so it's not possible to call a cloned board Arduino: that's why it's very common to find references on *duino boards like Seeeduino, FreeDuino, Japanino, Zigduino, iDuino, etc.

Figure 3.1: Some Arduino boards - From top left to bottom right: Lylipad, Mini, Nano (two), Pro, Duemilanove, Mega

3.2.1 Arduino Shields

Arduino boards functionalities can be extended by using shields, ad hoc designed PCBs having the same pin layout of Arduino, which can be stacked above of it adding additional functionalities. Figure 3.2 shows a quite extreme example of Arduino shield usage.

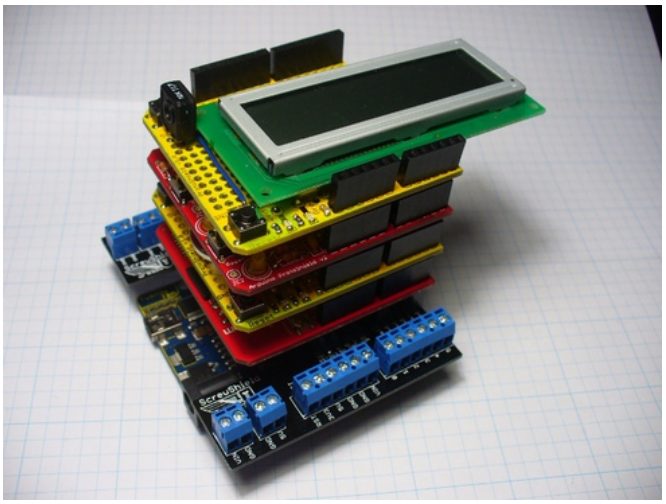

Figure 3.2: A somehow exaggerated example of Arduino shielding - Arduino it's placed on the bottom and the different shields are stacked above of it. Picture by John Boxall CC-BY-NC-SA 3.0

There is a huge amount of shields available, each one of them especially designed for one application. Some are being developed by the Arduino team while most of them have been developed by third party companies or individuals. There are shields for Motor controlling, Ethernet communication, MP3 playing, Analog video output, LCD displays, etc.. The idea is that using a shield is possible to add a specific feature to Arduino without the hassle of developing an ad hoc circuit or PCB trying to implement such feature. Moreover, some shields comes with easy to use libraries which allows fast and straightforward application development.

3.2.2 Arduino Duemilanove

My university provided me with an Arduino Duemilanove board which is, according to the Arduino developers, "the simplest one to use and the best one for learning on" (2, page 20).

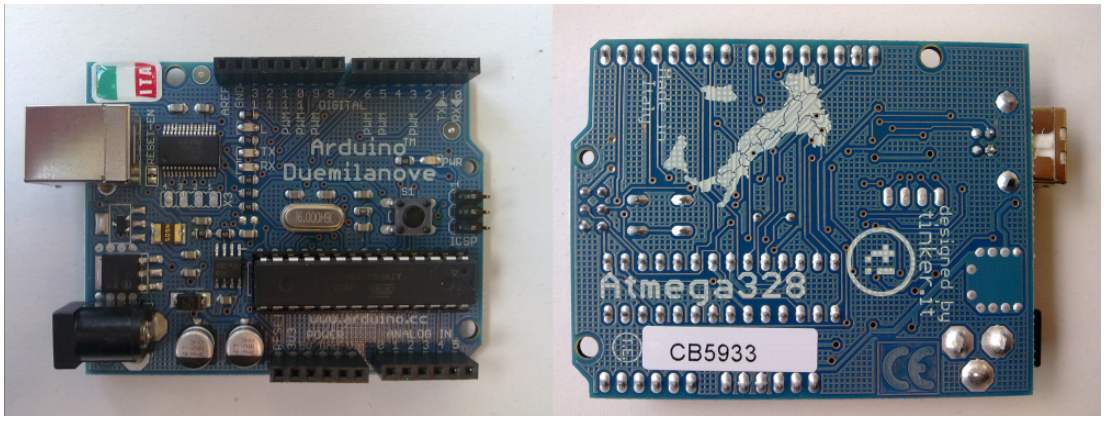

Figure 3.3: Arduino Duemilanove - Front and back view of the Arduino Duemilanove

3.2.2.1 Arduino Duemilanove internal components

Let's have a look at what's inside an Arduino Duemilanove. In figure 3.4 the most important internal components of the board are annotated and they will be described below.

1. FTDI chip. This is the component which enable the Arduino to communicate with the computer through USB. Arduino microcontroller is capable only of Serial communication. The FTDI chip converts the Serial signals to USB and vice versa. It also has an internal voltage regulator which converts the 5 V power coming from the USB to 3.3 V
2. Status LED. It is connected to pin 13 with an 1K Ω resistor. Every time a voltage is applied by the microcontroller to pin 13 the LED will light.
3. Serial TX and RX LEDs. They serves as indicators of a communication with the PC or another serial device (though digital pins 0 and 1).

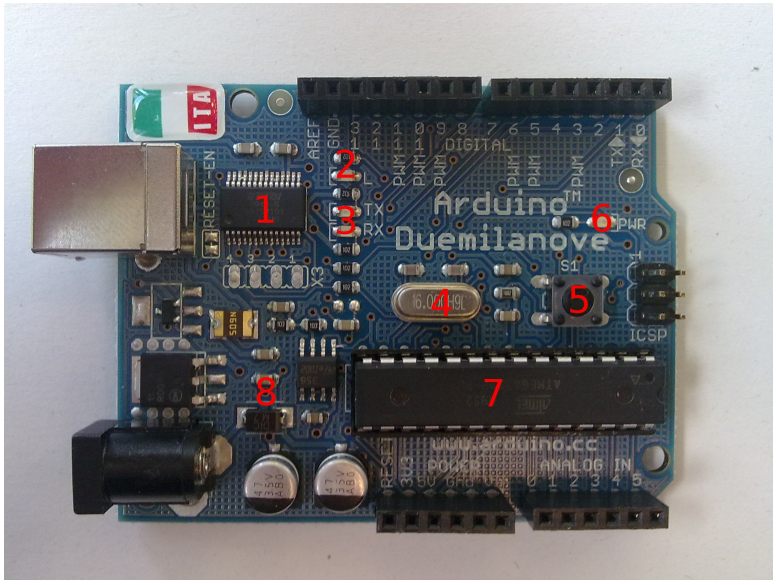

Figure 3.4: Arduino Duemilanove Front - Front view of the Duemilanove with the main components annotated

4. 16 MHz crystal. This is the component which acts as clock source to the microcontroller. Basically it generates an On-Off signal which the microcontroller uses to change its state.

5. Reset button. Once pressed, the microcontroller will reset.

6. Power LED (PWR) which will be on when the Arduino is connected to any power source indicating that the microcontroller is running.

7. This is the microcontroller, the heart of Arduino. The Duemilanove use the Atmel ATMEGA 328p, an 8-bit AVR RISC-based microcontroller which combines 32 KB ISP flash memory with read-while-write capabilities, 1 KB EEPROM, 2 KB SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible timer/counters with compare modes, internal and external interrupts, serial programmable USART, a byte-oriented 2-wire serial interface, SPI serial port, a 6-channel 10-bit A/D converter capable of running up to 200 KHz, programmable watchdog timer with internal oscillator, and five software selectable power saving modes. The ATMEGA 328p operates between 2.7-5.5 volts.

8. Various components: capacitors, diodes and voltage regulators. They are used to stabilize the power source, convert it to the correct voltage needed by the Arduino and prevent damages from shorts.

3.2.2.2 Arduino Duemilanove connectors

A key aspect of the Arduino board is the amount of connectors available. These are the components which permit wiring the Arduino boards to other components (sensors, resistors, buttons, etc..) so that it can interact with them: reading, writing, moving, etc.

As you can see from figure 3.4 above, an Arduino 2009 board has the following connectors (listed clockwise starting from the top left):

AREF: Analog Reference Pin The voltage at this pin determines the voltage at which the analog to digital converters (ADC's) will report the decimal value 1023, which is their highest level output. This means that using this pin you'll be able to change the maximum value readable by the Analog In pins: this is a way to change the scale of the analog in pins.

The AREF pin is, by default, connected to the AVCC voltage of around 5 volts (unless you are running your Arduino at a lower voltage).

GND: Digital Ground Used as Ground for Digital inputs/outputs.

DIGITAL 0-13: Digital Pins Used for digital I/O.

TX/RX Pins 0-1: Serial In/Out This pins can be used for digital I/O just like DIGITAL pins 2-13 but they can't be used if Serial communication is used. If your project use Serial communication you might want to use those for Serial communication instead of using the USB to serial interface. This can came handy while using the serial interface to interact with a non PC device (eg another Arduino or a Robot Controller)

External Interrupts Pins: 2-3 This pins can be configured to trigger an interrupt on different input conditions.

PWM: 3, 5, 6, 9, 10, 11 Provide 8-bit PWM output with the *analogWrite()* function (46).

LED: 13 There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

ICSP: In-circuit Serial Programmer Arduino comes with a bootloader which enable program uploading through the USB to serial interface. Advanced users can also directly upload programs to the Arduino board using an external programmer. This is done using the ICSP header. This way, it's possible to program Arduino without the need of the bootloader thus saving about 2 KB of program memory.

ANALOG IN 0-5: Analog input pins Used to read from an analog source (eg potentiometer, photo resistor or temperature sensor).

POWER Pins Used to get or provide power to the Arduino board

Vin when using an External Power Supply (see External Power Supply In), this provide the same voltage which is arriving from the power supply. It's also possible to provide voltage to the board trough this pin.

Gnd (2 Pins) Used as ground pins. Actually, while searching for the differences between digital ground and the other 2 Ground pins (See Power below), I found on the Arduino board schematics that all 3 ground pins on the Arduino board are actually connected together thus the digital ground pin and the 2 ground pins under the power section are actually just the same.

5V This is used to get 5V power from the board. This is the same voltage that powers the microcontroller. This can came either from Vin (External Power Supply In) or from the USB.

3V3 A 3.3 V power supply which is generated from the FTDI chip. The maximum current draw is 50mA. General consensus is to avoid using this pin power source or using it in controlled situations as shorts or a too high current drain may cause problems to the FTDI chip.

RESET By bringing this line LOW it's possible to reset the board: there is also a button for doing so on the board but, as additional shields might make the button unreachable, this can be used for resetting the board.

External Power Supply In Used to connect an external power supply to Arduino. A 2.1 mm center-positive plug connected to a battery or an AC-to-DC adapter. The current range can be 6 to 20 volts but, in order to prevent overheating and stability problems, the recommended range is 7 to 12 volts.

USB Used for uploading sketches (Arduino binary programs) to the board and for serial communication between the board and the computer. Arduino can be powered from the USB port.

3.2.3 Arduino Base Workshop KIT

The Arduino Board itself is pretty useless unless we plug it to other electrical components. Usually, coupled with an Arduino board, shops also sell Arduino kits which contain lot of useful components for developing Arduino based circuits.

My University provided me with an Arduino Base Workshop KIT which is displayed in figure 3.5.

The KIT is composed by the following components:

1. 1 x Arduino Duemilanove Board
2. 1 x USB cable
3. 1 x Straight single line pinhead connectors 2,54 40x1
4. 5 x 10K Ω Resistors 1/4W (brown, black, orange, gold)
5. 5 2.2K Ω Resistor 1/4 W (red, red, red, gold)
6. 10 x 220 Ω Resistors 1/4W (red, red, brown, gold)
7. 5 x 330K Ω Resistors 1/4W (orange, orange, yellow, gold)
8. 5 x 100nF capacitor polyester

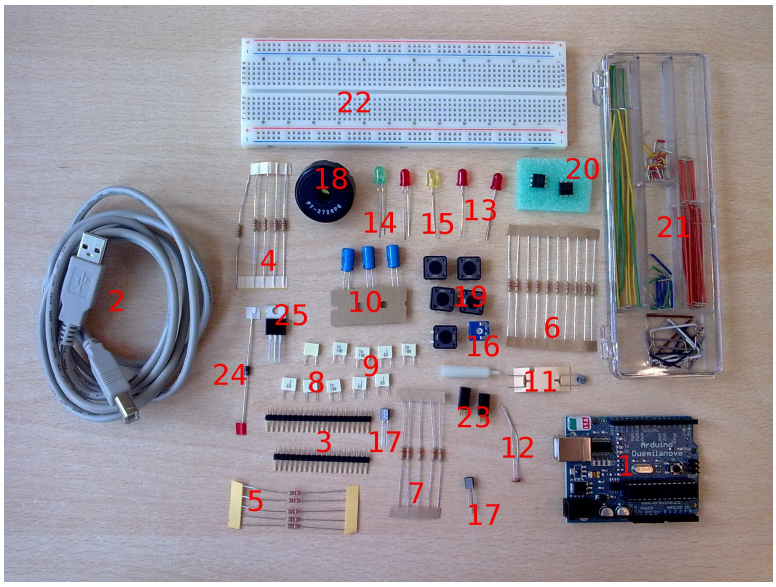

Figure 3.5: Arduino Base Workshop KIT - Components of the kit are annotated with the numbers from the list below.

9. 5 x 10nF capacitor polyester
10. 3 x 100uF electrolytic capacitor 25Vdc
11. 1 x 4,7K Ω Thermistor
12. 1 x 10..40K Ω LDR VT90N2
13. 3 x 5mm RED LED
14. 1 x 5mm GREEN LED
15. 1 x 5mm YELLOW LED
16. 1 x 10K Ω linear potentiometer, PCB terminals
17. 2 x BC547 Transistor in TO92 Package
18. 1 x Piezo buzzer
19. 5 x PCB Pushbutton, 12x12mm size
20. 2 x 4N35 Optocoupler DIL-6 package

21. 1 x Set of 70 breadboard jumper wires
22. 1 x Breadboard, 840 tie points
23. 2 x Tilt sensor
24. 1 x Diode 1n4007
25. 1 x MOS Irf540

A more detailed description of these components with examples of usages will be done in the next chapter.

3.3 Arduino Software

The other component of the Arduino platform is the Arduino IDE. This contains all the software which will run a computer in order to program and communicate with an Arduino board.

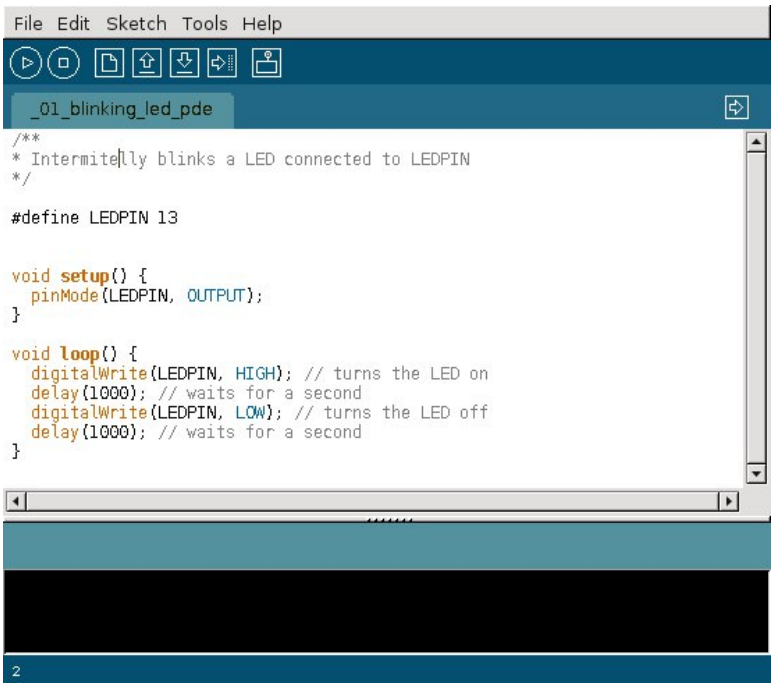

Figure 3.6: Arduino Programming IDE

The Arduino IDE contains an editor which we can use to write sketches (that's the name of Arduino programs) in a simple programming language modeled after the Processing language (45).

Using the IDE, the program we wrote is converted to a C program and then compiled using avr-gcc, a free, libre and open source compiler based on the Gnu C Compiler (gcc) especially designed for AVR microcontrollers. This process produce binary code which the microcontroller on the Arduino board will be able to understand and execute.

The binary code it's then uploaded to the Arduino microcontroller through the USB connection. This is done using the program avrdude which implements the communication protocol used to store programs into the Arduino program memory.

3.4 Arduino Community

Like many other free software and hardware projects, what makes Arduino great is the community around it. The number of users which everyday collaborate and share through the arduino.cc main website (46) is huge.

The arduino website contains a publicly editable Wiki, called the Playground, and a forum where people can ask for help on their projects or discuss about anything related to Arduino and electronics prototyping.

Arduino users are mostly hobbyists but Arduino it's also popular among students and researchers. It's not uncommon to see high quality contents on the forum and in the Wiki.

The fact that there are so many people working on Arduino has multiple advantages:

- access to ready to use Arduino based libraries for using many hardware and devices (eg: motors, steppers, sensors, network interfaces etc..)
- huge knowledge shared by other people
- possibility to easily ask for help.

3.5 Critics to Arduino

The Arduino platform has been criticized on some aspects and I think it worth noting what those critics are for the sake of transparency.

One of the most common critics is about the Arduino PCB design. As there is an additional 0.06“ spacing between the digital pin connectors, it is not possible to connect the Arduino directly on a breadboard which has 0.1” spaced connectors. For the same reason it's not possible to use standard prototyping perfboards with Arduino.

Arduino developers justified that as a simple design flaw which affected the first versions of Arduino. However, as there were already shields available for it, they decided to kept the design error for backward compatibility.

Another critics often made to Arduino is that it hides too much the inner details of the microcontroller or the program building details. People doing this critic are usually experienced developers or engineers which feels somehow limited by the over simplified programming APIs. Those people miss the fact that it's actually possible to program Arduino without using the API and directly interact to the microcontroller.

Other people think that the microcontroller used for Arduino has just too low computation power. Someone asked for a more powerful computing architecture such as an ARM based microcontroller.

However, the huge success of Arduino and the great projects people are doing with it, demonstrate that, even with its limitations, Arduino can be a very good prototyping platform. Moreover, I do think that the advantages of simplicity to use and openness fair exceed any criticism that could be made to it.

4

First steps with Arduino and electronic prototyping

In this chapter I present my first experiments with Arduino and electronic circuits. These are very simple examples but very good learning exercises.

Each example will be presented coupled with the circuit schematics, a picture of the circuit prototyped with Arduino on a breadboard and, if necessary, an explanation of the various theories and components involved.

4.1 Hello World!

It is common practice, while learning a new programming language or environment, to code a very simple program which prints the text *Hello World* to the screen.

Unfortunately, in microcontroller programming printing text is not that easy, so it's usual to blink an LED instead. That's how microcontroller programmers are used to say *Hello World*!

Before starting working on the Hello World program and circuit it's worth introducing two components which will be used a lot in the following examples: LEDs and the breadboard.

4.1.1 LED: Light-emitting diode

A light-emitting diode (LED) is a semiconductor light source. When a light-emitting diode is forward biased (switched on), electrons are able to recombine with electron holes within the device, releasing energy in the form of photons. This effect is called electroluminescence and the color of the light (corresponding to the energy of the photon) is determined by the energy gap of the semiconductor. LEDs, as any diode, only allows current flowing from the anode (+) to the cathode (-) but not in the reverse direction.

Thanks to their reliability, long lifetime, efficiency and low power consumption, LEDs are currently used in multiple applications: infrared remote controllers, state indicators, LCD displays back-lights, semaphores and car lights, etc. In electronic prototyping, LEDs can be really useful as they can be used as visual feedback for the user of the prototype.

An LED can come in various forms and packages, however in electronic prototyping the 5 mm packages is the most common one. This package is characterized by a transparent or colored round glassy case which has two metal legs coming out of the glass. The longer led is the anode (+) and the shorter one is the cathode (-).

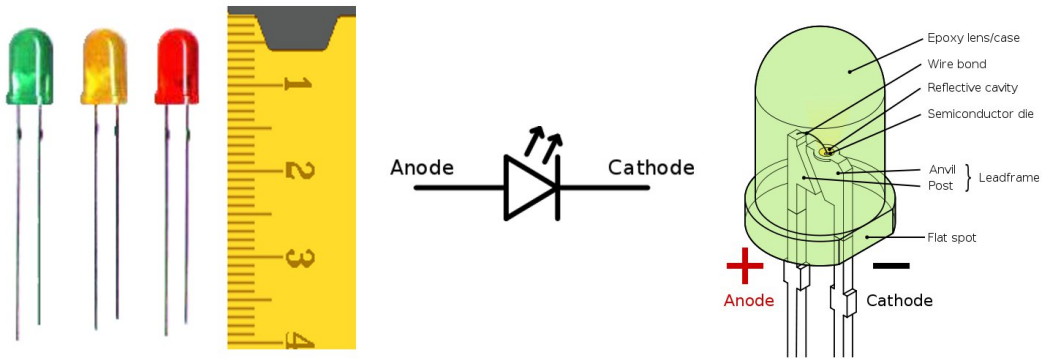

Figure 4.1: LED - Some standard 5mm LEDs (note the different length of the legs), LED electronic symbol and the parts of an LED. Pictures from (70)

Usually, an LED can be characterized by the following parameters which can be read from the LED datasheet:

I_f **typ** the typical forward (flowing from the anode to the cathode) current that should be used with this LED.

I_f **max** the maximum forward current that the LED is able to tolerate.

V_f **typ** the typical forward voltage which the LED should be connected to.

V_f **max** maximum forward voltage

There are additional parameters such as the luminous intensity, the viewing angle or the light wavelength but they can be considered of secondary importance compared to the current and voltage characterization.

In order to meet the LED characterization and limit the current flowing into the LED we have to add an in series resistor to the LED. The value of the resistor can be computed easily using the following formula, derived from the Ohm law.

$$R = \frac{(V_S - V_L)}{I} \tag{4.1}$$

Where V_S is the voltage of the power source, V_L is the voltage that will be applied to the LED and I is the current which will flow through the LED. By simply substituting V_L and I values with I_f typ and V_f typ from the LED datasheet into the R formula above we obtain the correct value of the resistance needed by our LED. It's important to note that, as the R formula above is based on Ohm law, the numbers should be expressed in Volts (V_S and V_L) and Ampere (I), when usually in the datasheets current ratings are expressed in mA.

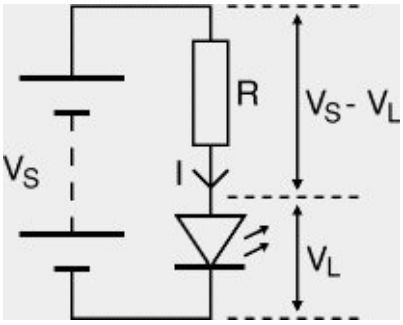

Figure 4.2: In series resistor with an LED - Visualization of the in series resistor formula.

4.1.2 Breadboard

A solder-less breadboard is a key element in electronics prototyping. It's used to rapidly build circuits without using solder just by inserting wires in its various holes. Inside, the breadboard holes are connected as per figure 4.3: lines marked with letters are connected vertically, while lines marked with + and - are connected horizontally. The breadboard in figure 4.3 has the lines A-E and F-J connected but there is no connection between the two groups of lines. Same happens for the top and bottom power lines: there is no connection between them.

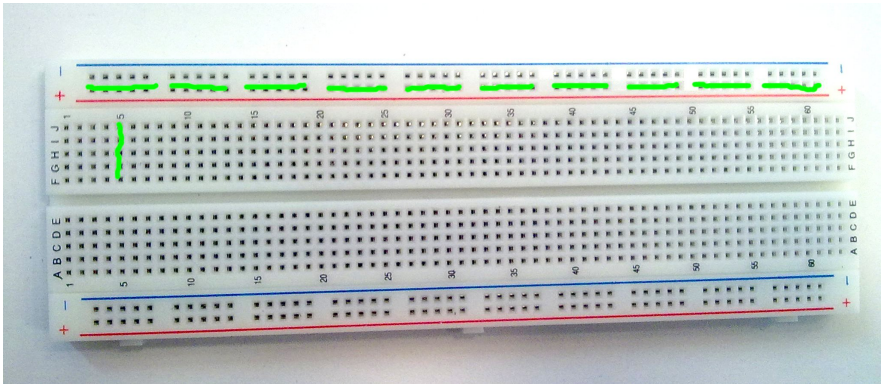

Figure 4.3: Breadboard - The green lines shows how the holes are connected.

4.1.3 Circuit Schematics and Prototype

In order to prototype the Hello World circuit, we simply connected the LED to Arduino digital pin 13 (+) and GND (-). We can connect the LED directly without using a resistor on pin 13 as there are already an LED and resistors connected to it on the Arduino printed circuit board.

Figure 4.4 A shows the simple Hello World circuit prototyped on the Arduino Duemilanove. The same circuit is then prototyped on a breadboard in figure 4.4 B.

4.1.4 Code

This is the program we will use to blink the LED.

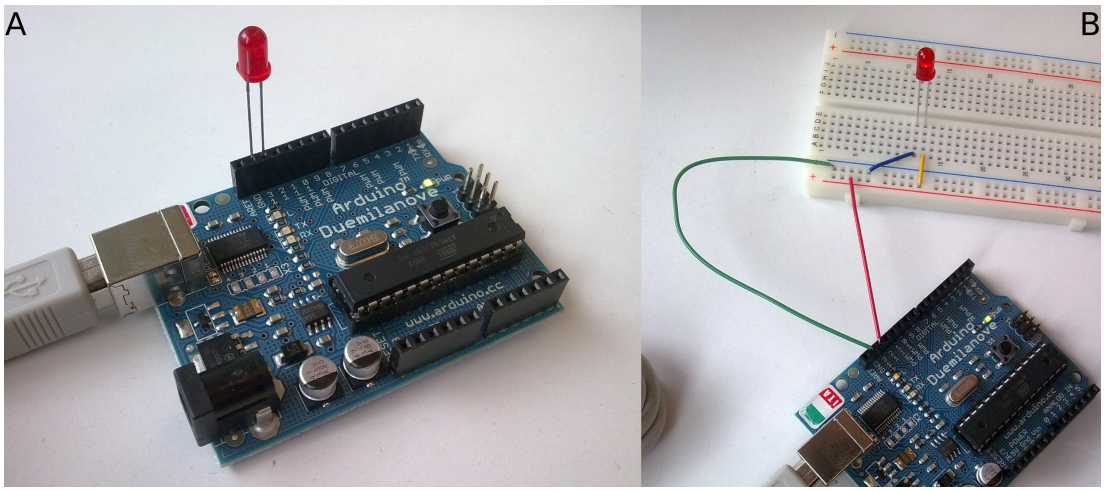

Figure 4.4: Hello World circuit prototyped - A: LED is inserted into GND and pin Digital 13 being careful to inset the long leg (+) into pin 13 and the short one (-) into the GND. **B:** This is the same circuit as in A but the breadboard has been used instead of directly plugging into the Arduino

```
1  /**
2  * Intermitelly blinks a LED connected to LEDPIN
3  */
4
5  #define LEDPIN 13
6
7  void setup() {
8    pinMode(LEDPIN, OUTPUT);
9  }
10
11 void loop() {
12   digitalWrite(LEDPIN, HIGH); // turns the LED on
13   delay(1000); // waits for a second
14   digitalWrite(LEDPIN, LOW); // turns the LED off
15   delay(1000); // waits for a second
16 }
```

As you can see, this is a pretty simple program but it is perfect to move the first steps into the Arduino programming APIs.

We first define a constant called *LEDPIN* and we set its value to 13. This is a good programming habit which helps avoiding magic numbers (71).

setup() is executed at the beginning of the program execution and it's used to setup how the Arduino board will work. In this program we set the board to use *LEDPIN* (defined as pin 13 in the *#define* statement) as an output, this means that we will be able to drive the pin on or off from our program.

loop() is executed continually by the Arduino microcontroller. So, It does the following in an infinite loop:

1. set *LEDPIN* to *HIGH*: this means that we are driving 5 V on pin *LEDPIN*. The LED will turn on.
2. wait for a second
3. set *LEDPIN* to *LOW* (this means that we are not delivering voltage on pin 13). The LED will turn off.
4. wait for another second

Simply by inserting the program above into the Arduino IDE editor and by uploading it, the Arduino board will store it in its microcontroller program memory, it will execute it and the LED will start blinking.

4.1.4.1 Blinking without using delay()

In the code above we used *delay()* to delay the execution of the following lines of code. Unfortunately there is a limitation using this solution: *delay()* is blocking, the microcontroller will actually stop execution and the whole board will be unusable while waiting for a *delay()* call. So you won't be able to program your board to do anything while it's blocked into a *delay()* call.

The code below, taken from the Arduino Tutorial, does exactly the same of the above one, but without using the *delay()* function.

```
1  // constants won't change. Used here to
2  // set pin numbers:
3  const int ledPin =  13;      // the number of the LED pin
4
5  // Variables will change:
6  int ledState = LOW;             // ledState used to set the LED
7  long previousMillis = 0;        // will store last time LED was updated
8
9  // the follow variables is a long because the time, measured in
       miliseconds,
10 // will quickly become a bigger number than can be stored in an int.
11 long interval = 1000;           // interval at which to blink
       (milliseconds)
12
13 void setup() {
14   // set the digital pin as output:
15   pinMode(ledPin, OUTPUT);
16 }
17
18 void loop()
19 {
20   // here is where you'd put code that needs to be running all the time.
21
22   // check to see if it's time to blink the LED; that is, if the
23   // difference between the current time and last time you blinked
24   // the LED is bigger than the interval at which you want to
25   // blink the LED.
26   unsigned long currentMillis = millis();
27
28   if(currentMillis - previousMillis > interval) {
29     // save the last time you blinked the LED
30     previousMillis = currentMillis;
31
32     // if the LED is off turn it on and vice-versa:
33     if (ledState == LOW)
34       ledState = HIGH;
35     else
36       ledState = LOW;
37
38     // set the LED with the ledState of the variable:
39     digitalWrite(ledPin, ledState);
40   }
41 }
```

The code above is still pretty simple. With the call to *millis()* we get the number of milliseconds passed from when the last time the microcontroller has been reset (eg: when switched on, pressed the reset button or when a new program has been uploaded). So, the code switches the state of the LED whenever the difference between the last timer reset and the current time is greater than the configured interval.

As you can see, there are no calls to *delay()* here so we could have added more instructions to be executed between one switching of the LED and another. This wouldn't have happened with *delay()*.

4.1.5 Extension

It's pretty simple to extend this example so that, instead of blinking one LED, two LEDs will blink alternatively. Of course, we will need two LEDs now. As I said before, pin 13 has an in series resistor connected to it, so we didn't need to use a resistor when we connected the first LED. This applies only to LED 13: any other digital pin doesn't have any resistor connected: an in series resistor will be needed when connecting the second LED. A picture of the prototyped circuit is shown in figure 4.5.

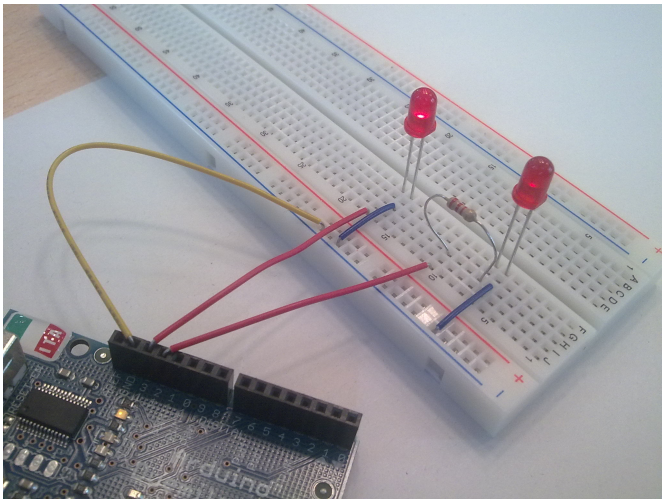

Figure 4.5: Hello World circuit Extended - This is the same circuit as in figure 4.4 but another LED has been added. Note the presence of an additional 2.2 K Ω resistor in series with the new LED.

Programming two LEDs instead of one is just a matter of duplicating the same instructions of the first Hello World program. This has been done in the program below.

```
1  /**
2  * Intermitelly blinks two LEDs connected to LEDPIN1 and LEDPIN2
3  */
4
5  #define LEDPIN1 13
6  #define LEDPIN2 12
7
8
9  void setup() {
10   pinMode(LEDPIN1, OUTPUT);
11   pinMode(LEDPIN2, OUTPUT);
12 }
13
14 void loop() {
15   digitalWrite(LEDPIN1, HIGH); // turns the LED on
16   delay(500); // waits for a second
17   digitalWrite(LEDPIN2, HIGH); // turns the LED on
18   delay(500); // waits for a second
19   digitalWrite(LEDPIN1, LOW); // turns the LED off
20   delay(500); // waits for a second
21   digitalWrite(LEDPIN2, LOW); // turns the LED off
22   delay(500); // waits for a second
23 }
```

4.2 digitalRead(): using pushbuttons and tilt sensors

In the previous section we used *digitalWrite()* to drive a voltage on an Arduino pin to lights one or more LEDs. In this section instead we will use *digitalRead()*, the function used to read a logic level from the arduino pins.

digitalRead() is pretty simple: it will return HIGH (a constant equal to 1) if the voltage on its pin is high or LOW (a constant equal to 0) if low. Note that an ATMEGA microcontroller running at 5 V will read any voltage major than 2.5 V as logic high. With this function we can then easily read the status of a switch or button. Before doing so, let's have a closer look at the components we will use: pushbuttons and tilt sensors.

4.2.1 Pushbuttons

A pushbutton is a simple switch mechanism which allows for user generated changes in the state of a circuit. Pushbutton usually comes with four legs but, as you can see from the picture below, legs are always connected in groups of two. When the pushbutton is pressed all the 4 legs get connected.

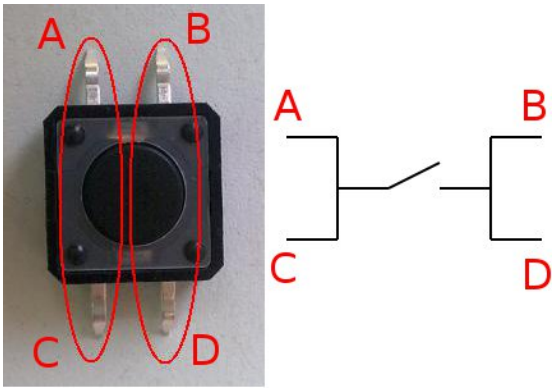

Figure 4.6: A pushbutton and its schematic representation - This is the pushbutton available in the Arduino Base Workshop KIT. You can note how its legs are connected two by two when the button is not pressed. When it's pressed, all the legs get connected.

4.2.2 Tilt Sensors

Tilt sensors or tilt switches are a pretty simple electronic component which consists of a small plastic case which contains a metal ball. At the bottom of the sensor there are four legs which are disconnected. The ball inside the case is able to move: when you move the sensor with the legs at the bottom, the ball will move to the bottom connecting the four legs. When the sensor is placed with legs up, the ball moves to the top disconnecting the legs. You can understand the behavior by having a look at figure 4.7. Basically, a tilt sensor is a switch, just like a pushbutton. The difference between a tilt sensor and a pushbutton is how they mechanically change their open/close state: the tilt sensor by tilting, the pushbutton by pushing.

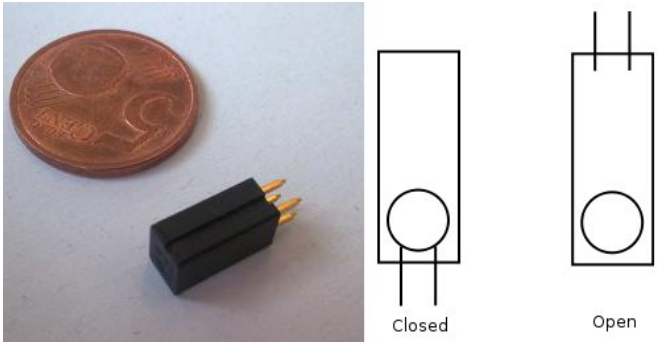

Figure 4.7: A tilt sensor or tilt switch - The drawing on the left clearly explain what happens inside a tilt switch.

4.2.3 Simple example with a Pushbutton and a Tilt sensor

It's useful to prototype a very simple circuit which shows us how Pushbuttons and Tilt sensors actually work. So, we will use them as switches to light an LED. This circuit is shown in figure 4.8. The result is that, when we push on the pushbutton, the circuit will close and the LED will lights on. Same result when the tilt switch has the legs pointing the bottom, when it's legs up the LED will be off.

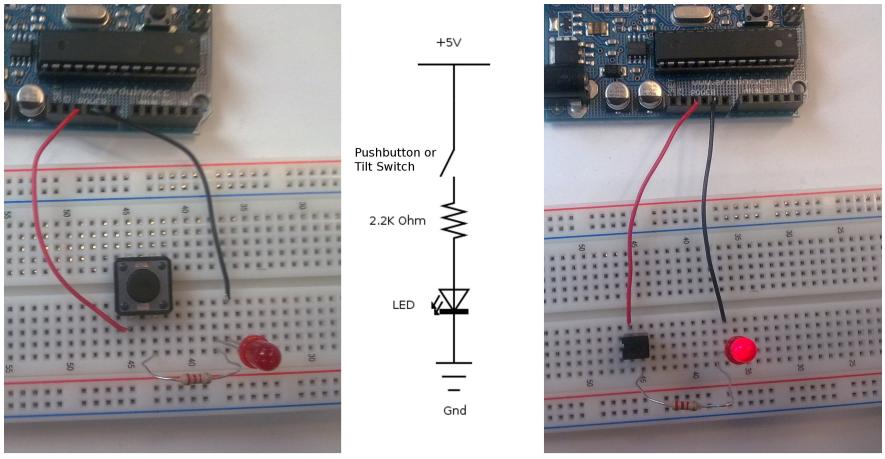

Figure 4.8: Simple example circuit for pushbuttons and tilt sensors - When we close the circuit, by pushing the pushbutton or by orientate the tilt switch with the legs bottom, the LED will light. Note the 2.2K Ω resistor connected in series to the LED to limit the current flowing through it.

4.2.4 Reading the switch status from Arduino

The circuits we created above are really simple. We can't do much with them. Instead pushbuttons and tilt sensors might be really useful if we could get their open/closed state as an input value to the Arduino board. We could use this value to trigger actions in an Arduino program generating all different kind of output from the board.

From what we have seen in the sections above, we may be tempted to use something like the circuits in figure 4.9 to read the state of a switch from the Arduino board. Unfortunately this won't work.

When we want to read a logical value in Arduino we have to set the pin as input. This can be done in the *setup()* routine of our Arduino program by using the function *pinMode()* (47). Using *pinMode()* we can set the status of the pin as input. This means that the state of pin will now be as *high impedance*: the pin is not being driven actively by the circuit and it will float (acting as an antenna) reading randomly HIGH or LOW. That's why the circuits in figure 4.9 won't work as expected as when the switch is open and digital pin 2 is in high impedance state the status read by *digitalRead()* will be floating. Of course, when we close the switch we will get a stable *digitalRead()* result but this is still not useful.

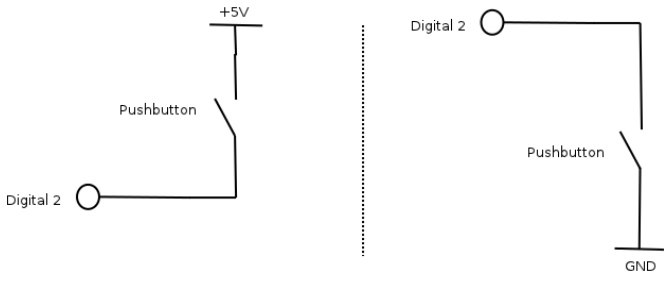

Figure 4.9: Wrong circuits for connecting a switch to a digital input - This circuits wont work as expected. As pin 2 has been set as INPUT during setup, it's now in high impedance state so, if not connected to a logic value it will float between logical values.

4.2.4.1 Pull-up and Pull-down resistors

In order to fix the floating circuit described above we have to introduce the usage of pull-up or pull-down resistors. Example circuits are displayed in figure 4.10.

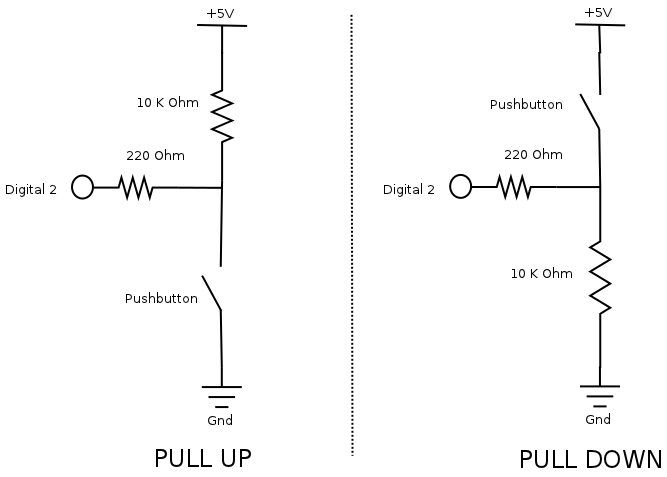

Figure 4.10: Example circuit for Pull-up and Pull-down usage - Using these circuits the input read on pin 2 isn't floating anymore but it's reliable.

The intuitive idea behind the pull-up and pull-down technique is that we need to somehow give a default connection to the switch even when it is open so that we can fix the floating behavior of the high impedance microcontroller input pin.

Let's explain how this technique works by focusing on the pullup circuit in figure 4.10. When the switch is open, there is a weak connection from digital pin 2 to the 5 V source through the 10K Ω resistor: our digital pin will read HIGH. Instead, when we close the switch there will be a strong pull of the digital pin 2 to ground which will override the very weak pull to HIGH: our pin will read LOW.

Similar, but inverted behavior is achieved using the pull-down technique. When the switch is open, there will be a weak pull of digital pin 2 to ground so it will read LOW. Instead, when the switch is closed there will a strong pull of digital pin 2 to the 5 Volt source, and pin 2 will read HIGH.

You may have noted the presence of a 220 Ω resistor placed just before the input pin. This resistor acts as buffer protecting from shorts caused by incorrectly use the pin 2

as output instead of input. If by error, we set the pin as output and set the output HIGH we may cause a short which could damage the pin or the whole microcontroller.

It's important to note how the read values on the digital pin change if we use a pull-up or pull down scheme with the same switch state. This is summarized in the table below:

Pull technique	Switch open	Switch closed
Pull-up	reads HIGH	reads LOW
Pull-down	reads LOW	reads HIGH

Table 4.1: Read values for pull-up and pull-down techniques

4.2.4.2 Debouncing a button

Before reading the state of a pushbutton or tilt switch on Arduino we still have something to cover. In the section above we assumed that as soon as we press our pushbutton we will immediately get the opposite value. Unfortunately, it's not so simple.

Due to the mechanical characteristics of a switch, the read value in the first instants before the change of state will somehow oscillate. This is displayed in figure 4.11. The state of the button somehow bounces in the first moments the button is pressed. This can cause some problems as our microcontroller will actually detect most of those state changes potentially giving us problems.

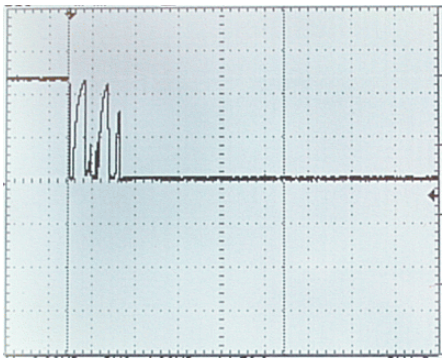

Figure 4.11: A bouncing button on an oscilloscope - Note how the change of state isn't immediate but it oscillate in the first moments.

There are different ways of fixing this issue. Using a pure electronic approach, it can be fixed using a capacitor which will compensate the bounces. An example of such solution is explained in (19).

However, it's also possible to fix this undesired issue directly on the microcontroller by adding a small delay each time a change of the switch status is detected. If the delay is long enough to cover the bouncing time then we won't detect any of the bouncing change of state. This solution is implemented in the code in the following pages.

4.2.4.3 Controlling an LED in Arduino according to the status of a switch in input

We can now wire up the first example of reading the status of a switch in Arduino. We will use the input coming from a switch to light an LED connected to another pin. The idea is to mimic the behavior of the last example but this time using the Arduino microcontroller to drive the output to the LED.

The circuit schematics and a picture of it prototyped on a breadboard with Arduino are shown in figure 4.12. You can see how we used a pull-down approach so we expect the pin 2 to read LOW when the button is not pressed while we expect an HIGH when the button is pressed.

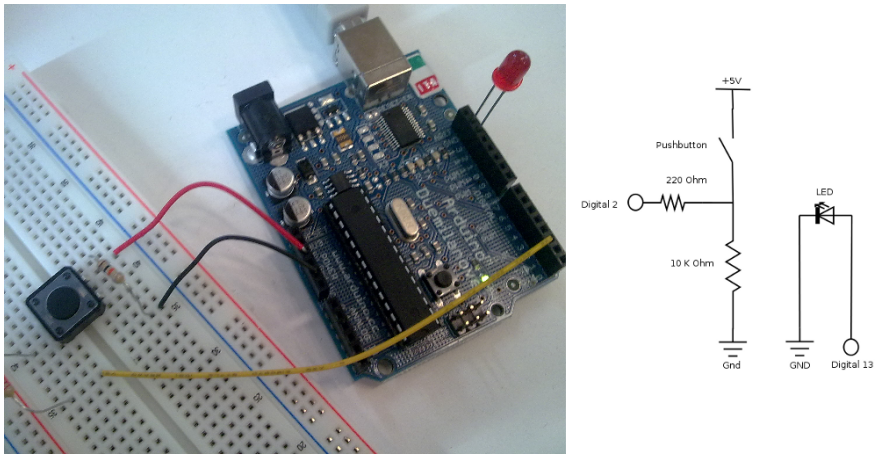

Figure 4.12: Controlling an LED in Arduino according to the status of a switch in input - Note the pull-down approach.

Once we have the circuit prototyped, we can use the following code to obtain the desired behavior.

```
/**
* Turn on an LED connected to LEDPIN only when
* the value readed on INPIN is HIGH
*/

#define LEDPIN 13
#define INPIN 2


void setup() {
  pinMode(LEDPIN, OUTPUT);
  pinMode(INPIN, INPUT);
}

void loop() {
  if(digitalRead(INPIN) == HIGH){
    digitalWrite(LEDPIN, HIGH); // turns the LED on
  }
  else {
    digitalWrite(LEDPIN, LOW); // turns the LED off
  }
  delay(10); // debounces switch
}
```

In *setup()* we configure the pin 2 as high impedance setting it as INPUT. We also configure pin 13 to be used as output so that we can drive the LED.

The actual code in *loop()* is still quite simple: we simply write HIGH or LOW to the LED pin according to the value read on the button pin. Note the call to *delay()* which debounces our switch.

4.2.4.4 Interrupts in Arduino from a switch

By using exactly the same circuit prototyped in figure 4.12 above we can introduce another very useful programming feature of the Arduino microcontroller: interrupts.

Let's see how the code became:

```
/**
```

```
2  * Turn on an LED connected to LEDPIN only when the input
3  * readed on INTERRUPTPIN (0 is Pin 2) changes its value
4  */
5
6  #define LEDPIN 13
7  #define INTERRUPTPIN 0
8
9  volatile boolean state = LOW;
10
11 void setup() {
12   pinMode(LEDPIN, OUTPUT);
13   attachInterrupt(INTERRUPTPIN, buttonChange, CHANGE);
14 }
15
16 void loop() {
17   digitalWrite(LEDPIN, state);
18 }
19
20 void buttonChange()
21 {
22   state = !state;
23 }
```

The idea is that in *setup()* we setup *buttonChange()* to be the interrupt handler using *attachInterrupt()*. So, each time there is a change in the state of INTERRUPTPIN the function *buttonChange()* will be executed toggling the value of the volatile variable *state*. As in *loop()* we continually write the value of *state* to the LEDPIN this code actually produce the desired behavior.

Unfortunately, we can't use the function *delay()* inside interrupt handlers (due to limitations in the internal functioning of the microcontroller) so we can't use the software debouncing approach here. An hardware debouncing solution will be needed.

4.3 analogRead(): Reading analog values with Arduino

Until now, we only worked with digital signals whose values could change only between HIGH and LOW. However there are components which doesn't output a digital signal but instead output a continuous signal ranging from 0 to a known voltage. This kind of signal is called analog signal and the value of such signal is called analog value.

A microcontroller is however a discrete entity which is only capable of working on discrete variables (actually only on 0 and 1) so we need some way of converting an analog value into a digital one.

This analog to digital conversion happens inside the analog-to-digital converter (ADC). The Arduino microcontroller has an integrated ADC with a 10-bit precision. The ADC is capable of converting any analog signal ranging from 0 V to the voltage applied on the analog reference pin (AREF) to a discrete value ranging from 0 to 1023.

The Arduino function for sampling values from the ADC is *analogRead()*. Before doing a real example we still need to introduce a couple more concepts.

4.3.1 Voltage divider circuits and Potentiometers

A voltage divider circuit is capable of divide the voltage applied on one side resulting in the divided output voltage on the other side. A voltage divider circuit is displayed in figure 4.13.

If we recall the Ohm law as we introduced it in section 2.3, we know that the voltage drop across any resistor is given by:

$$E_n = I_n R_n \tag{4.2}$$

where I_n is the current flowing into the resistor n and R_n is the resistor value.

We also know that the total current flowing into a series circuit is given by:

$$I_{tot} = \frac{E_{tot}}{R_{tot}} \tag{4.3}$$

By combining those two formulas substituting I_n with I_{tot} we obtain the voltage divider formula:

$$E_n = E_{tot}\frac{R_n}{R_{tot}} \quad \Longleftrightarrow \quad \frac{E_n}{E_{tot}} = \frac{R_n}{R_{tot}} \tag{4.4}$$

This formula states that the ratio of individual resistance to total resistance is the same as the ratio of individual voltage drop to total supply voltage in a voltage divider circuit.

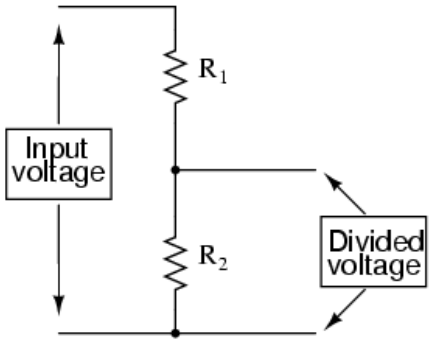

Figure 4.13: Voltage divider circuit - The voltage across R_2 follows the voltage divider formula: $E_{R_2} = V_{in}\frac{R_2}{R_{tot}}$. Picture from (27)

One device frequently used as a voltage-dividing component is the potentiometer, which is a resistor with a movable element positioned by a manual knob or lever. The movable element, typically called a wiper, makes contact with a resistive strip of material (commonly called the slidewire if made of resistive metal wire) at any point selected by the manual control as shown in figure 4.14.

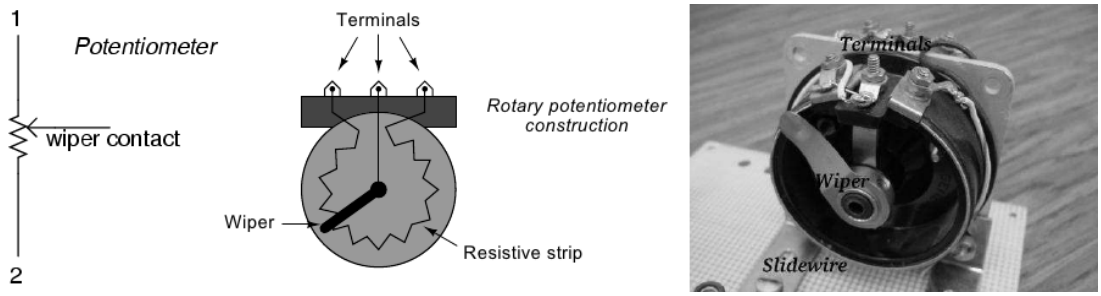

Figure 4.14: Anatomy of a potentiometer - Its circuit behavior, components inside and a picture of a real potentiometer. Pictures from (27).

When the knob is rotated, we are able of changing the values of the R_1 and R_2 resistors in associated voltage divider circuit of the potentiometer. As result the divided voltage changes when we rotate the knob.

We can experience the voltage dividing capability of the potentiometer by building a very simple circuit using Arduino as shown in figure 4.15. We take the 5 V source of Arduino and connect a potentiometer in the middle between the ground and an LED.

This way, the potentiometer acts as a voltage divider circuit and the voltage which reaches the LED, thus the intensity of emitted light, depends on how we rotate the knob.

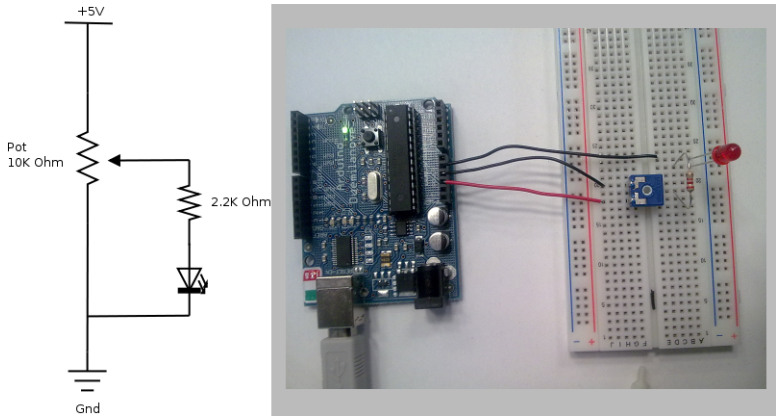

Figure 4.15: Simple circuit for experience with a potentiometer - As we rotate the knob, the amount of light emitted by the LED will change.

4.3.2 Reading a potentiometer with Arduino

The obvious evolution of the example above is trying to get the analog value coming out from the potentiometer into Arduino using the integrated ADC of the ATMEGA 328p with *analogRead()*. The goal will be using the analog value read from the potentiometer and use it as interval between the blinks of an LED.

We can prototype a simple circuit as in picture 4.16. We connect the potentiometer to 5 V, ground and to Arduino Analog pin 2. We insert an LED into Digital pin 13.

Now, if we use the following Arduino code we get the expected behavior from the circuit.

```
1 /* Reads the value from a potentiometer and use it as delay */
2
3 #define POTPIN 0
4 #define LEDPIN 13
5
6 int val = 0;
7
8 void setup() {
```

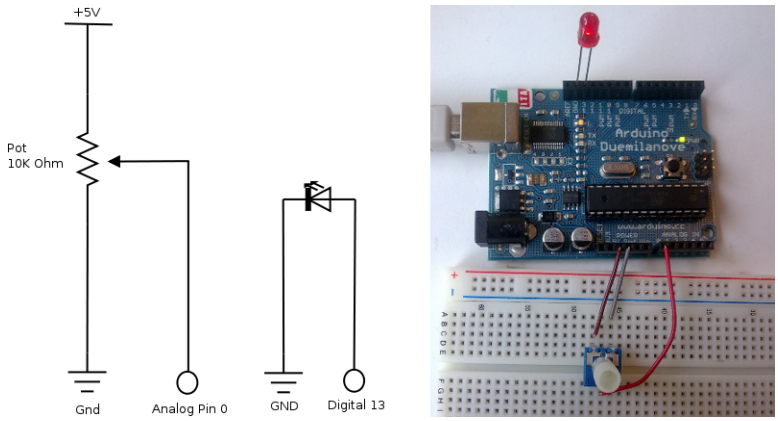

Figure 4.16: Circuit for reading a potentiometer with Arduino - The potentiometer acts as voltage divider modifying the voltage read on Analog pin 0. The read value is then used as delay between the LED blinking.

```
9    pinMode(LEDPIN, OUTPUT);
10 }
11
12 void loop() {
13   val = analogRead(POTPIN);     // read the value from the sensor
14   digitalWrite(LEDPIN, HIGH);   // turn the ledPin on
15   delay(val);                   // stop the program for some time
16   digitalWrite(LEDPIN, LOW);    // turn the ledPin off
17   delay(val);                   // stop the program for some time
18 }
```

This code is still very simple. We simply read the voltage coming out from the potentiometer to analog pin 2 using *analogRead()* and use that value as milliseconds delay for the calls to *delay()*.

4.3.3 Thermistors and Light dependent resistors with Arduino

In the past sections we introduced voltage divider circuits and gave the potentiometer as an example of voltage divider circuit. However there are many different components which can be used in a voltage divider circuit as they somehow provide different resistor values depending on external factors. Examples of such components are Thermistors and Light dependent resistors (LDRs).

4.3.3.1 Thermistors

A thermistor is a type of resistor whose resistance varies with temperature. Thermistors are widely used as inrush current limiters, temperature sensors, self-resetting overcurrent protectors, and self-regulating heating elements.

Thermistors follows the following rule:

$$\Delta R = k * \Delta T \tag{4.5}$$

where ΔR is the change in resistance, ΔT is the change in temperature and k is the first-order temperature coefficient of resistance.

As example, given k positive and k=0.7, if we increase the temperature of 5 degrees, the component resistance will also increase by $\Delta R = k * \Delta T = 0.7 * 5 = 3.5\ \Omega$.

Note that some thermistors have a temperature coefficient of resistance which is negative. This will make ΔT and ΔR inversely proportional.

In figure 4.17 there is the thermistor available in the Arduino Base Workshop KIT. You can also see an example circuit for reading it and a picture of it prototyped on the Arduino. As you can see, this is still a voltage divider circuit whose output value this time depends on the temperature detected by the thermistor.

Testing this circuit is simple. We can run exactly the same code used for reading the potentiometer: if we touch the thermistor we will produce an increase of its internal temperature which will also increase its associated resistance. The LED blinking frequency should change once we touch the thermistor.

4.3.3.2 Light dependent resistors (LDRs)

A photoresistor or light dependent resistor is a resistor whose resistance decreases with increasing incident light intensity. In the Arduino Base Workshop KIT there is a 10..40K Ω LDR (VT90N2) which is depicted in figure 4.18. By using exactly the same circuit used for the thermistor but replacing it with the LDR, we can now have the voltage on Analog pin 0 depends on the amount of incident light on the LDR.

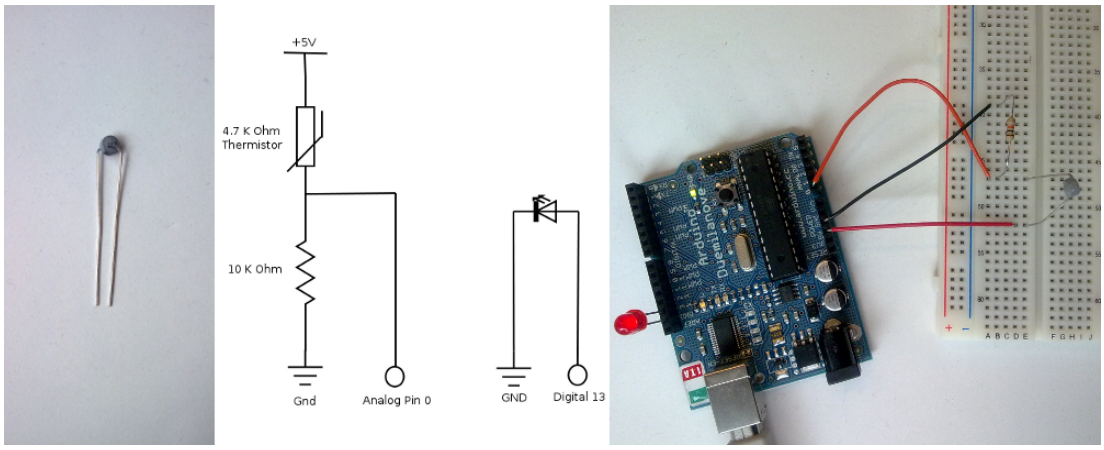

Figure 4.17: A thermistor and an example circuit with Arduino - The resistance of the thermistor will variate with temperature so that the voltage across analog pin 0 will also variate with temperature.

Testing the circuit is simple. We still can use the same code used for the potentiometer on Arduino and, once we run it, the blinking interval of the LED should change if we modify the quantity of light getting to the LDR (we can cover it with our hand).

4.4 Driving bigger loads: Transistors and Optocouplers

In the past sections we only used the output functionalities of Arduino to light LEDs. However, Arduino can be used to drive many other kind of devices. For example, we may be interested in using Arduino to light a big lamp or to activate an electrical motor.

This kind of devices usually require a big amount of current and a higher voltage than the 5 V that the Arduino is capable of deliver. Moreover an ATMEGA 328p can only deliver 50mA of current: if you connect a bigger load, the pin or the whole microcontroller could get damaged.

Fortunately, there are components which helps in driving a circuit from another one. Those components are transistors and optocouplers.

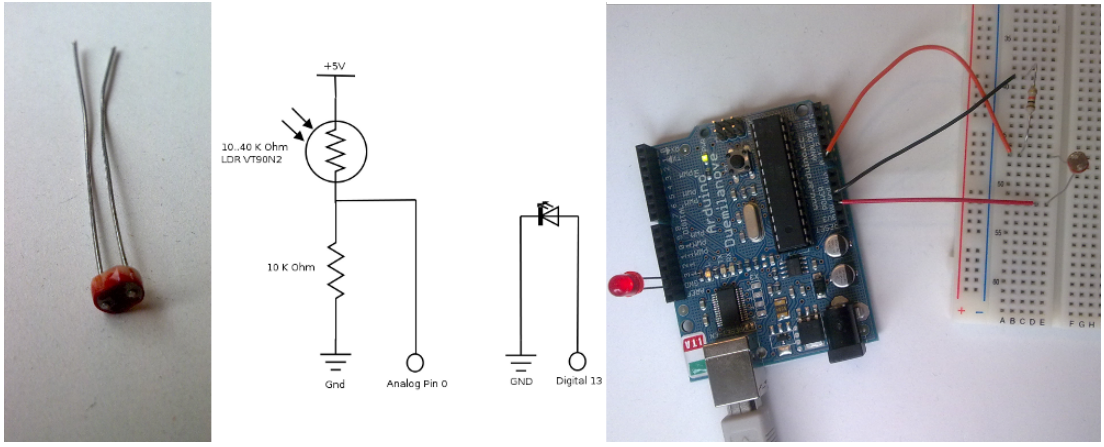

Figure 4.18: A Light dependent resistor and an example circuit with Arduino - The resistance of the LDR will variate with the quantity of incident light on it so that the voltage across analog pin 0 will also variate with the quantity of incident light.

4.4.1 Transistors

A Transistor is a semiconductor device extensively used in analog and digital electronics. Transistors usually have 3 connectors called collector, base and emitter. In normal state the collector and emitter are disconnected but, when a current is applied to the base connector, the transistor change its state and the collector and emitter get connected allowing current to flow between them.

The Arduino Base Workshop KIT comes with two types of transistors, displayed in figure 4.19: a MOS Irf540 (left) and a BC547 (right). They differs from the building technique used which results in different specifics. For big currents (eg powering motors) the MOS Irf540 will be perfect. The BC547 is not capable of delivering lot of current so use it with care. For all the details on these two components the respective datasheets should be consulted.

4.4.1.1 Using transistors with Arduino

We can prototype a very simple circuit to check how a transistor works. The circuit is displayed in figure 4.20: we connected three LEDs to the 9 V source of Arduino (which

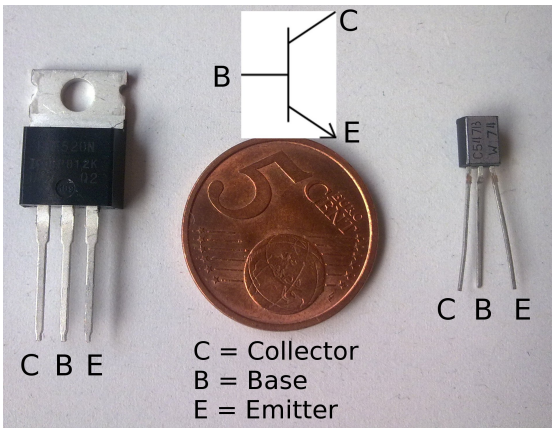

Figure 4.19: Transistors - MOS Irf540 (left) and BC547 (right). In the center the schematic diagram of a transistor.

has to be connected to an external battery to work) and placed a transistor between them and the ground. We connected the Digital pin 2 to the base connector on the transistor. This way, when we apply a voltage on pin 2 from the Arduino we can close the 9V circuit allowing current to flow in it.

Note that three LEDs is not really a very big load but this just serves as example as at the time of this tests I didn't have access to a DC motor.

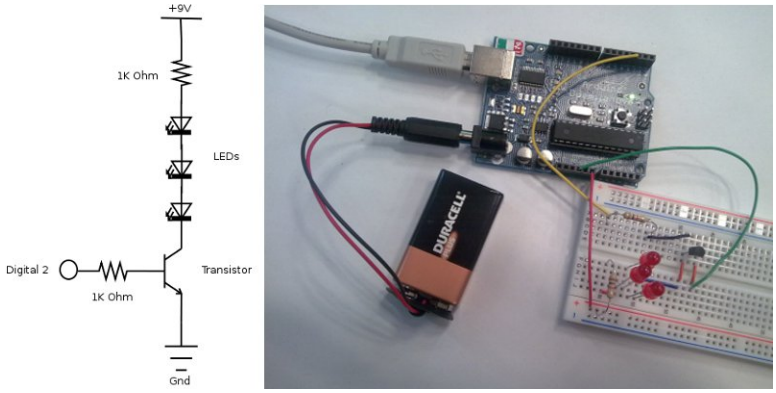

Figure 4.20: Transistor circuit - MOS Irf540 (left) and BC547 (right). In the center, the schematic diagram of a transistor.

We can test this circuit using the Hello World program (see code in section 4.1.4 page 30) with a simple modification: we will use pin 2 as output (in the Hello World program

we used pin 13).

By running that code and using digital pin 2 as output, each time the output on pin 2 is HIGH our transistor will get a voltage on its base connector resulting in the collector and the emitter getting connected. Current coming from the +9V source can the flow down through the resistor and the three series LEDs lighting them on.

Again, it's important to note how the 5 V circuit of the Arduino is completely isolated from the 9 V circuit driving the LEDs.

4.4.2 Optocouplers

An optocoupler, also called opto-isolator, optical isolator, optical coupling device, photocoupler, or photoMOS, is an electronic device that usually contains an infrared light-emitting diode (LED) and a photodetector and use them to transfer an electronic signal between element of circuits maintaining them electrically isolated (figure 4.21.

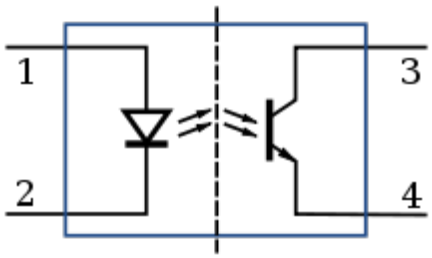

Figure 4.21: Inside an optocoupler - The two circuits are isolated. When the LED is light current can flows in the circuit on the right.

When a voltage is applied to the LED, the LED lights and illuminate the photodetector which produces an output current on the photodetector: basically this means that now the photodetector circuit is connected and current can flow in it.

4.4.2.1 Using optocouplers with Arduino

The Arduino Base Workshop kit comes with two 4N35 Optocouplers packaged as a DIL-6 package. This little component has 6 legs each of them having a different usage.

It can be easily understood by looking at figure 4.22 from the 4N35 datasheet which shows us the inside schematics of the 4N35.

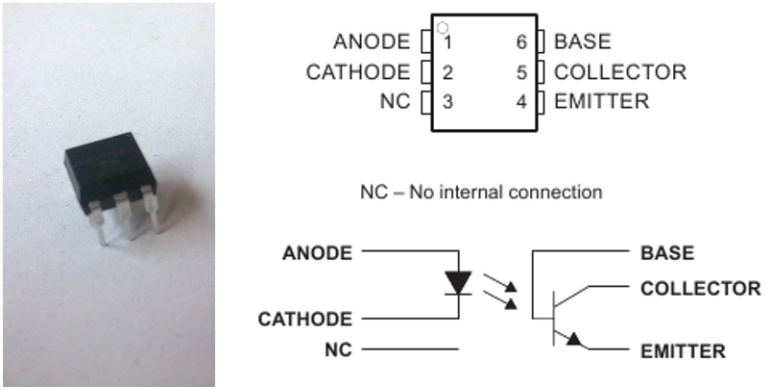

Figure 4.22: 4N35 Optocoupler - Picture from the 4N35 Datasheet.

We have leg 1 and 2 near the printed dot on the chip (that's visible on it if we look carefully) that acts respectively as anode and cathode. Leg 3 isn't connected to anything: it's just useless. We then have leg 4, 5, 6 respecly emitter, collector and base.

We already know these terms from the transistor introduction above. They do exactly the same of the legs of a transistor. The difference here is that we can leave the base unconnected and just use the LED (legs 1 and 2) to connect the collector and the base.

In order to test how an optocoupler works we can use the circuit depicted in figure 4.23: as you can see it is really similar to the circuit we used for testing the transistor.

We can test this circuit by using exactly the same code used on the transistor example (Hello World with Digital pin 2 as output). As expected, the LEDs will light when digital pin 2 is HIGH and will switch off when digital pin 2 is LOW.

4.5 Pulse Width Modulation (PWM): analog outputs with digital means

Digital boards and processors, like the Arduino board and its ATMEGA 328 microcontroller, usually have some problems providing an Analog Output, a variable signal

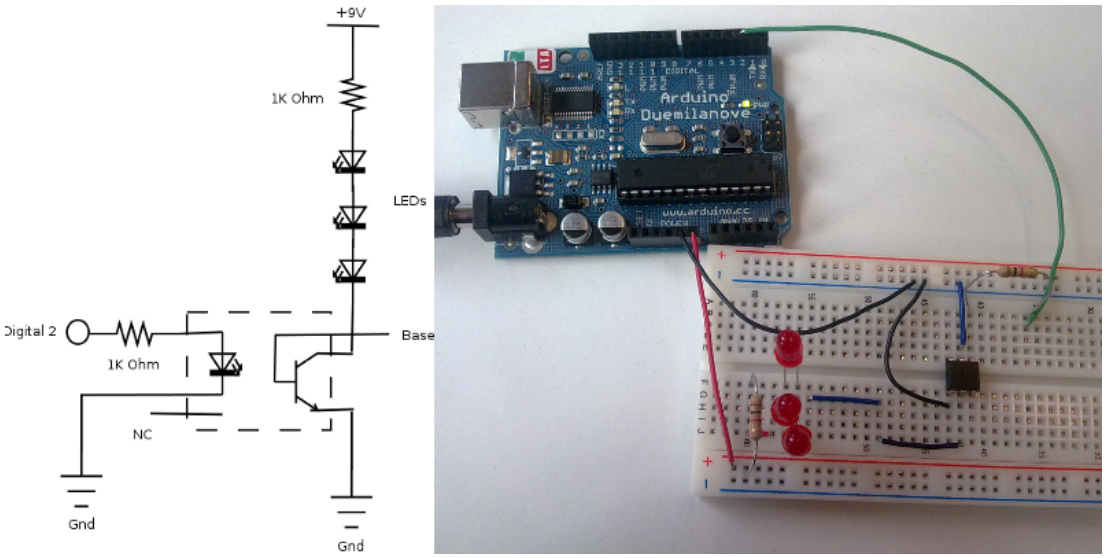

Figure 4.23: Optocoupler circuit and Arduino - Picture from the 4N35 Datasheet.

which can range from eg 0 to 5 V. This is a consequence of the fact that they are digital components, so they work using 0 and 1, they are not capable of such a variable output.

Fortunately, there is the Pulse Width Modulation (PWM) technique which makes possible to get an analog output using digital means.

Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 V) and off (0 V) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5 V controlling the brightness of the LED.

In Arduino, we have the function *analogWrite()* which implements PWM. It gets a parameter on a value between 0 and 255. You can see some examples of square waves generated with analogWrite() in figure 4.24.

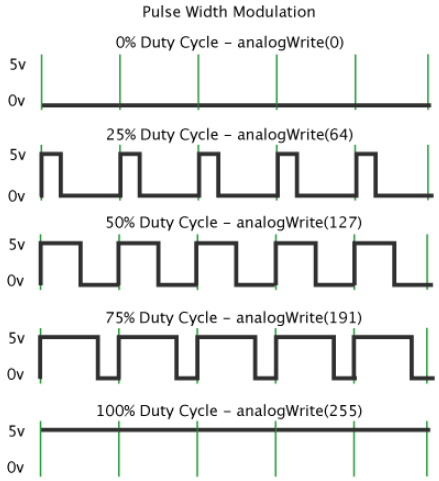

Figure 4.24: Example of Pulse Width Modulation (PWM)

4.5.1 Fading an LED using PWM with Arduino analogWrite()

Let's try the PWM analogWrite() implementation of Arduino with a simple example. Let's try fading on and off an LED. We just need a 1K Ω resistor in series with an LED connected to PWM capable digital pin on Arduino. We'll use pin 9.

The following code, coming from the Arduino analogWrite() tutorial implements exactly the desired effect.

```
1  /*
2   Fading
3
4   This example shows how to fade an LED using the analogWrite() function.
5
6   The circuit:
7   * LED attached from digital pin 9 to ground.
8
9   Created 1 Nov 2008
10  By David A. Mellis
11  Modified 17 June 2009
12  By Tom Igoe
13
14  http://arduino.cc/en/Tutorial/Fading
15
16  */
17
```

```
18
19 int ledPin = 9;     // LED connected to digital pin 9
20
21 void setup()  {
22   // nothing happens in setup
23 }
24
25 void loop()  {
26   // fade in from min to max in increments of 5 points:
27   for(int fadeValue = 0 ; fadeValue <= 255; fadeValue +=5) {
28     // sets the value (range from 0 to 255):
29     analogWrite(ledPin, fadeValue);
30     // wait for 30 milliseconds to see the dimming effect
31     delay(30);
32   }
33
34   // fade out from max to min in increments of 5 points:
35   for(int fadeValue = 255 ; fadeValue >= 0; fadeValue -=5) {
36     // sets the value (range from 0 to 255):
37     analogWrite(ledPin, fadeValue);
38     // wait for 30 milliseconds to see the dimming effect
39     delay(30);
40   }
41 }
```

4.6 Serial communication with Arduino

Arduino, as we have seen in the past sections, offers a lot of possibilities to interact with sensors, actuators, motors, etc.. But this is somehow limited by the simple capabilities of Arduino itself.

Fortunately, it's pretty simple to interface Arduino with more complex devices like a PC or another Arduino board. This can be achieved using the integrated Serial interface of Arduino.

As we saw on section 3.2.2.2, Arduino connects to the PC using an USB port. Anyway, that USB connection is actually used like a Serial (RS232) connection. The Arduino IDE uses it to upload our programs to the board but the serial connection can also be used for any other kind of communication.

Arduino also has digital pins 0 (RX) and 1 (TX) which can be used to directly connect Serial interfaced wires into Arduino. They deliver the same signals sent on the USB Serial interface.

It's important to note that, when the serial communication is in use, it's impossible to use digital pins 0 and 1 for anything else then serial communication. They are delivering serial signals (even if you don't plug them and you are using USB) so you can't use them for anything else.

4.6.1 Arduino Serial programming

The Arduino programming language comes with a really simple Serial API. It's all contained into the Serial library which contains the following functions:

begin() sets the datarate in bits per second (baud) for serial data transmission.

end() disable serial communication

available() gets the number of bytes (characters) available for reading over the serial port.

read() reads the first byte of incoming serial data available.

flush() flushes the buffer of incoming serial data.

print() prints data to the serial port.

println() prints data to the serial port, followed by a carriage return character(ASCII 13, or `\r`) and a newline character (ASCII 10, or `\n`)

write() writes binary data to the serial port.

A complete description of these function is out of scope here. For more information and all the details of the Arduino Serial APIs you should refer to the official Arduino Serial API documentation (48).

4.6.2 Writing data to the Serial interface with Arduino: reading the state of one button

Let's start practicing with Arduino serial API by writing a simple program which reads the state of one button and, if pressed, lights on an LED and send the button state over the serial interface.

For doing so we'll start from the *digitalRead()* example we saw in section 4.2.4.3. The circuit will be almost the same: the only difference will be that we'll use a pull-up resistor rather than a pull-down one. This decision will make sense in the next examples. Note that a pull-up connected pin will read HIGH when the button is not pressed while it will read LOW when the button is pressed. The circuit is depicted in figure 4.25.

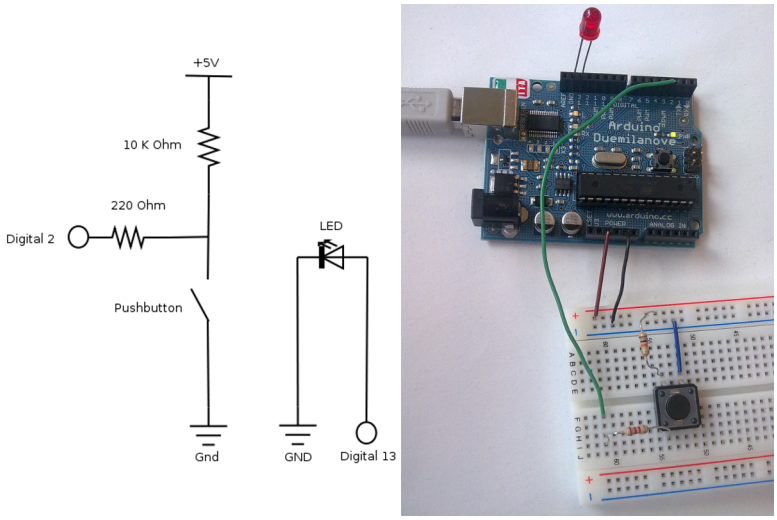

Figure 4.25: Circuit for reading the state of one button - Note the usage of a pullup resistor.

We can use the following program to read the state of the button from the Digital In pin and then switch off the LED when the button is pressed and communicate the button state over the Serial interface.

```
1 /**
2 * Read the state of the button from the Digital In pin and
3 * then switch off the LED when the button is pressed and communicate
4 * the button state over the Serial interface
5 */
6
```

```
7  #define LEDPIN 13
8  #define INPIN 2
9
10 int state = LOW;
11
12 void setup() {
13   Serial.begin(9600); // setup serial connection speed
14   pinMode(LEDPIN, OUTPUT);
15   pinMode(INPIN, INPUT);
16 }
17
18 void loop() {
19   delay(10); // debounces switch
20   int sensorValue = digitalRead(INPIN);
21   if(state != sensorValue) {
22     state = sensorValue;
23     digitalWrite(LEDPIN, sensorValue); // turns the LED on or off
24     Serial.println(sensorValue, DEC);
25   }
26 }
```

In the *setup()* routine we initialize the Serial interface and set its speed to 9600 bauds. In *loop()* we debounce the switch then we read the button status with *digitalRead()*. We keep track of the current state of the button. By doing this we are able to only communicate state changes in the button.

I think that it's pretty important to keep the amount of information flowing trough the Serial interface low. It seems that most of the how-tos and examples I've found online each loop they print to the serial interface. I think this is pretty useless as we are only interested in state changes, we do know that between two state changes the button hasn't changed its status.

So, only if we detect a state change, we use *digitalWrite()* to turn on or off our LED and then print the value as decimal using *Serial.println()*.

Once we create the circuit and we upload the program above to Arduino, we are able to use the Arduino IDE Serial Monitor to show everything that passes on the serial connection and thus test our program.

4.6.3 Reading the state of two buttons with Arduino and communicate their state via Serial interface

We now try to make the above example a little bit more complex by reading two buttons instead of one. We'll now need two pull-up resistors and another LED. Remember that pin 13 has a 1K Ω resistor connected in series, so it's safe to directly connect the LED to it. If we want to add another LED, for example on Digital pin 12, we'll have to use a series 1K Ω to avoid damages to the LED. The resulting circuit can be seen in figures 4.26 and 4.27.

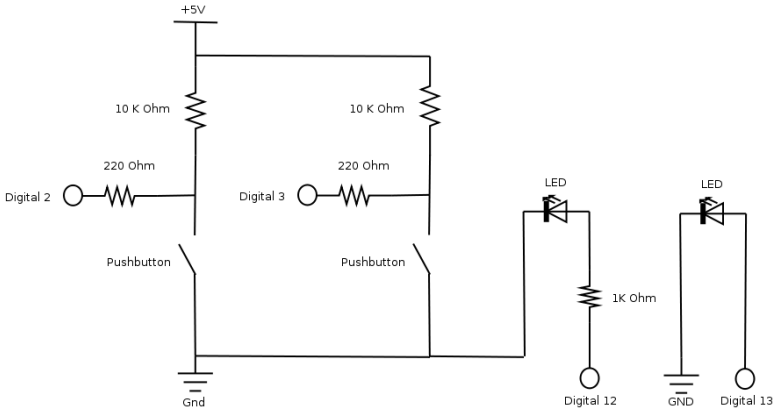

Figure 4.26: Circuit for reading the state of two buttons - Note the usage of a pullup resistors.

Extending the program presented in section 4.6.2 for using two buttons instead of one is trivial and won't be reported here for brevity. You can note however that the presence of external pullups somehow adds complexity to the whole circuit and you can easily understand how this can became a problem as the number of buttons increase. A solution to this issue is presented in the following section.

4.6.4 Using internal pull-up resistors

The ATMEGA chip mounted on the Arduino provides internal 20K pull-up resistors on any digital input pin. This feature can be enabled in the software with this two calls, usually implemented in *setup()*:

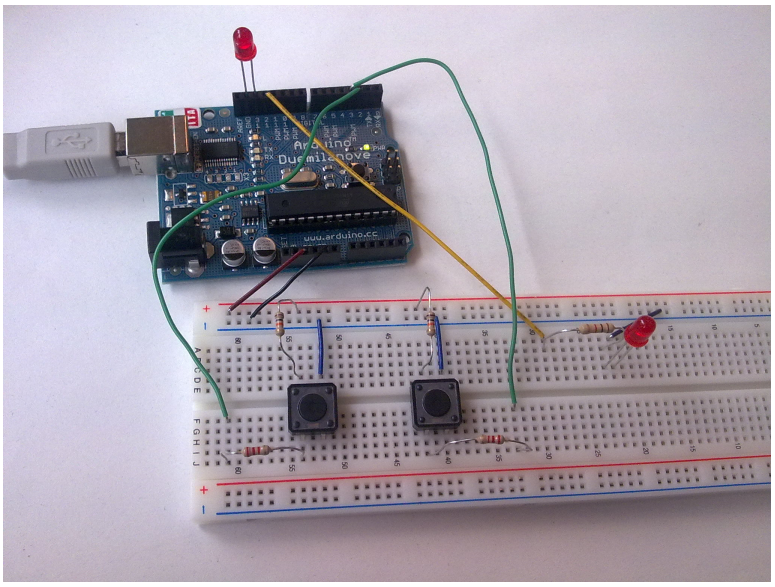

Figure 4.27: Picture of the circuit for reading the state of one button - Note the usage of a pullup resistor.

```
pinMode(pin, INPUT);            // set pin to input
digitalWrite(pin, HIGH);        // turn on pullup resistors
```

By writing HIGH on a digital input pin previously set as INPUT we enable the 20K internal pull-up resistors. Doing so, we no more need to connect external pull-up resistors. The circuit became as in figure 4.28.

The complete code for reading two buttons, blink the associated LEDs and print the status of the buttons over serial interface is reported below.

```
1  /**
2  * Uses internal pullups to read 2 pushbutton states,
3  * Communicate the state of the button using serial interface and
4  * lights on/off 2 LEDs associated with the buttons
5  */
6
7  #define LEDPIN1 13
8  #define LEDPIN2 12
9  #define INPIN1 2
10 #define INPIN2 3
11
12 int state1 = HIGH;
```

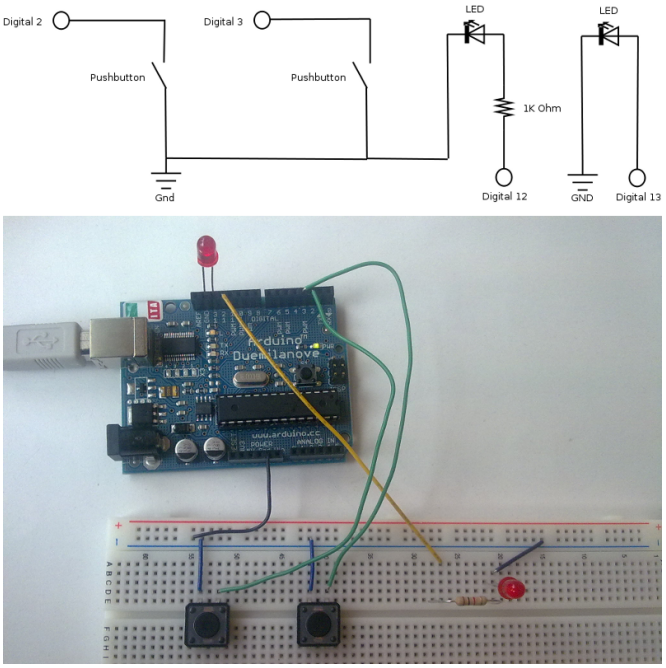

Figure 4.28: Reading two buttons using the internal pullups resistors - Note how the circuit looks really more simple.

```
13 int state2 = HIGH;
14
15 void setup() {
16   Serial.begin(9600);
17   pinMode(LEDPIN1, OUTPUT);
18   pinMode(LEDPIN2, OUTPUT);
19
20   pinMode(INPIN1, INPUT);
21   digitalWrite(INPIN1, HIGH); // enable pullup resitor
22
23   pinMode(INPIN2, INPUT);
24   digitalWrite(INPIN2, HIGH); // enable pullup resistor
25 }
26
27 void loop() {
28   delay(10); // debounces switches
29   int val1 = digitalRead(INPIN1);
30   int val2 = digitalRead(INPIN2);
31   if(state1 != val1 || state2 != val2) {
32     state1 = val1;
33     state2 = val2;
```

```
34      digitalWrite(LEDPIN1, val1); // turns the LED on or off
35      digitalWrite(LEDPIN2, val2); // turns the LED on or off
36      Serial.print(val1, DEC);
37      Serial.println(val2, DEC);
38    }
39  }
```

4.6.5 Two-way Serial communication with Arduino

In the examples above we used serial communication only to print something to the Serial interface from the Arduino board. However, Arduino can also read from the serial interface.

Now, we will try to implement a little program which could leverage two ways communication capabilities of the Arduino board: we'll plug only one LED to pin 13 and we'll try to light it on or off using a command coming from the serial interface.

In the programs above we used *Serial.print()* and *Serial.println()* to print data to the Arduino serial interface. In order to implement a simple two-way communication we also have to be able to read from the serial interface. The *Serial.read()* function does exactly that.

The following program reads from the Serial connection, if it read a 1 it turns on the LED, if it read 0 it turn off the LED.

```
1  /**
2  * Reads commands coming from serial interface to drive an LED on/off
3  * Also prints led status back
4  */
5
6  #define LEDPIN 13
7
8  int state = LOW;
9  char incomingByte = 0;
10
11 void setup() {
12   Serial.begin(9600);
13   pinMode(LEDPIN, OUTPUT);
14 }
15
```

```
16 void loop() {
17
18   // send data only when you receive data:
19   if (Serial.available() > 0) {
20      // we receive a char representing an integer. let's converto to int
21     int incomingState = (Serial.read() == '1');
22
23     // say what you got:
24     Serial.print("I␣received:␣");
25     Serial.println(incomingState, DEC);
26
27     if(incomingState != state) {
28       state = incomingState;
29       digitalWrite(LEDPIN, state); // turns the LED on or off
30       Serial.print("Setting␣LED␣as:␣");
31       Serial.println(state);
32     }
33     else {
34       Serial.print("Doing␣nothing.␣LED␣already:␣");
35       Serial.println(state);
36     }
37   }
38 }
```

4.7 A multisensors game controller with Arduino and Processing

In the previous sections we introduced most of Arduino input output capabilities. We now have enough information to create a very simple but functional game controller which interacts with a program running on the PC. The idea is to read the status of some switches, a tilt sensor, a potentiometer and an LDR and send those to the computer via Serial interface and use them to modify the state of the program running on the computer.

4.7.1 Multisensors controller circuit

The circuit used is displayed in figure 4.29. We use the internal pullups on digital inputs 2 to 7 so we can connect 5 buttons (up, down, left, right, fire) and the tilt switch

directly without external pullups. We add 3 LEDs to digital 11, 12 and 13 so that we can use them as visual feedback for the user. On the right we have the analog inputs: the LDR and the potentiometer.

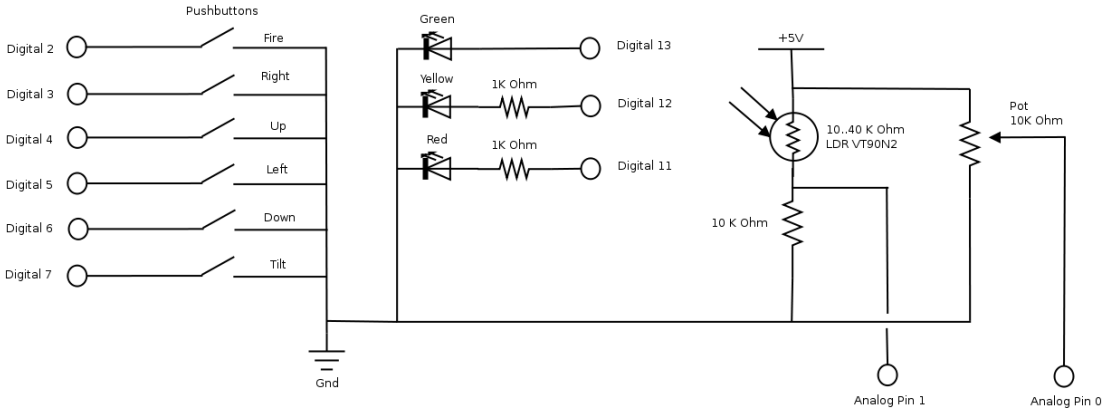

Figure 4.29: Multisensors controller circuit - Note how the circuit looks really more simple.

4.7.2 Processing

Processing is an libre programming language and integrated development environment (IDE) built for the electronic arts and visual design communities with the purpose of teaching the basics of computer programming in a visual context, and to serve as the foundation for electronic sketchbooks.

The Arduino programming language is modeled after Processing, so these two languages share lot of things. A complete introduction to Processing is out of scope here. For a complete documentation on Processing you should have a look at the official documentation.

4.7.3 The final "video game"

By using the input information read from the Arduino and sending them to the Processing application through the Serial communication, I implemented a very simple *video*

game which let you move a square into the graphics by pushing the various buttons on the controller (figure 4.30).

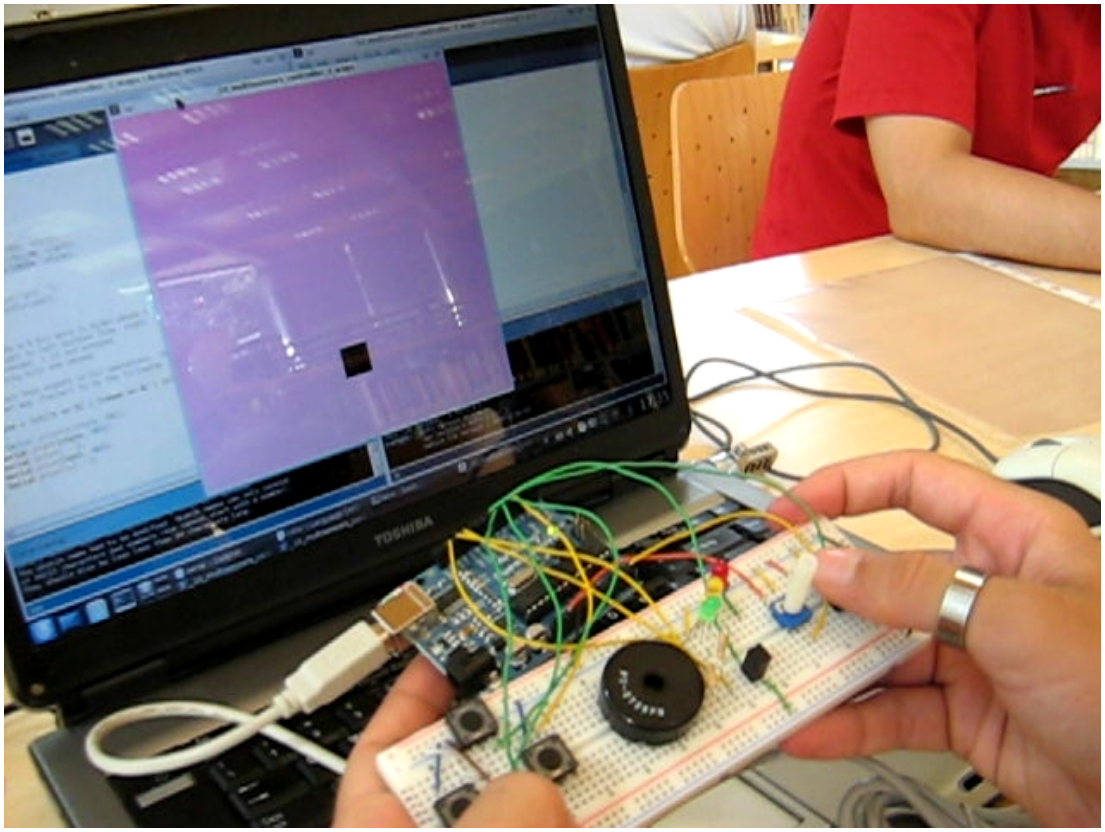

Figure 4.30: Multisensors controller demo

5

MEMS Sensors: accelerometers, gyroscopes and magnetometers

Microelectromechanical systems (MEMS) are small integrated devices or systems that combine electrical and mechanical components. They range in size from the sub micrometer (or sub micron) level to the millimeter level, and there can be any number, from a few to millions, in a particular system. MEMS extend the fabrication techniques developed for the integrated circuit industry to add mechanical elements such as beams, gears, diaphragms, and springs to devices.

Examples of MEMS device applications include inkjet-printer cartridges, accelerometers, miniature robots, microengines, locks, inertial sensors, microtransmissions, micromirrors, micro actuators, optical scanners, fluid pumps, transducers, and chemical, pressure and flow sensors. New applications are emerging as the existing technology is applied to the miniaturization and integration of conventional devices.

These systems can sense, control, and activate mechanical processes on the micro scale, and function individually or in arrays to generate effects on the macro scale. The micro fabrication technology enables fabrication of large arrays of devices, which individually perform simple tasks, but in combination can accomplish complicated functions (63).

In the past decades advances in MEMS technologies made the fabrication of MEMS technology based sensors possible and economically feasible so that nowadays many consumer products now includes MEMS based sensors. Currently the most widespread

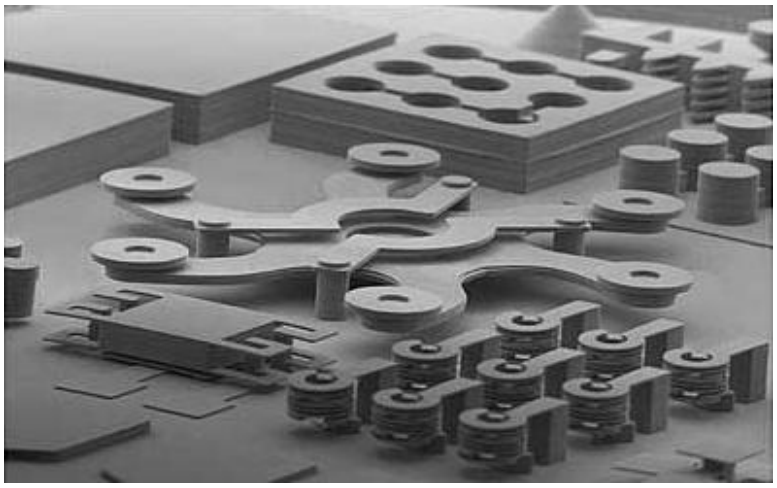

Figure 5.1: Magnified picture of a MEMS device - Micro mechanical elements are clearly visible.

kind of sensors are MEMS based accelerometers but the past five years have also seen the introduction of MEMS based gyroscopes and magnetometers.

In the following sections we will introduce what MEMS accelerometer, gyroscopes and magnetometers measure and how they internally works.

5.1 The accelerometer

An accelerometer is a device that measures its proper acceleration, that is the physical acceleration experienced by it relative to a free-fall, or inertial, observer who is momentarily at rest relative to the object being measured ([64], [75]).

This definition will be elaborated in the following sections.

5.1.1 Modelization of an accelerometer: a mass on a spring

The functioning of an accelerometer can be understood by thinking of it as a mass on a spring system (figure 5.2). When the system is steady and no accelerations act on the system, the mass will lie in the O point. When an acceleration a is applied to the system, the mass will displace of x from the origin O.

As we know from the Newton's second law of motion, a mass m which is accelerated by an acceleration a will be subject to a force $F = ma$. As the mass is also connected to the spring, the spring itself, following Hooke's law, will generate an opposite force proportional to x so that $F = kx$ where k is a constant dependent of the spring characteristics called spring constant.

As the movements of the mass will always be constrained by the spring, we will have that $F = ma = kx$ and from this equation, by knowing k and m and by measuring on our accelerometer the displacement x we can compute the external acceleration using simply $a = \frac{kx}{m}$.

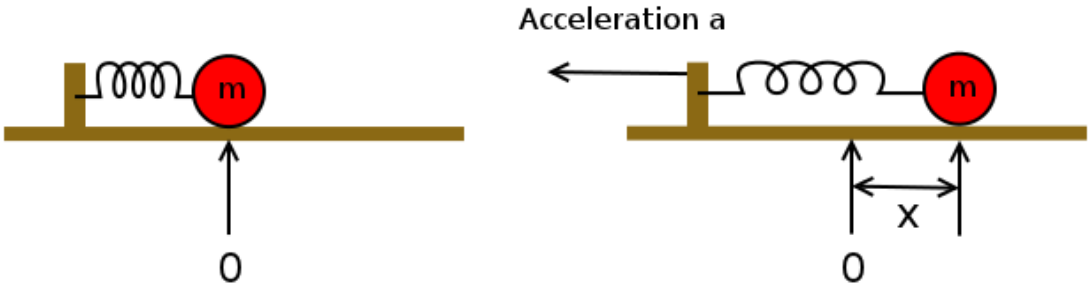

Figure 5.2: Mass on a spring model of a single axis accelerometer - On the left the accelerometer is at rest while on the right an acceleration a is applied to it resulting in a displacement x of the mass m.

By doing so we have transformed the problem of measuring the external acceleration which acts on an accelerometer to simply measure the displacement of its internal mass, something which is possible even on a very small MEMS accelerometer (50).

5.1.2 The accelerometer and gravity

Sometimes an accelerometer is wrongly described as a device which measures accelerations. This is not entirely true: in the definition we gave in the section introduction we said that it measures proper accelerations and we will show now what that means by doing some simple examples.

Let's suppose to place our accelerometer in the outer space (figure 5.3 A). If the accelerometer is at rest, there are no forces nor accelerations acting on the accelerometer

and on its internal mass. We expect the accelerometer to read zero.

When we place the accelerometer on a space ship (figure 5.3 B) accelerated by its engines by a we will have a force $F = ma$ acting on the mass contrasted by the elastic force of the springs $F = kx$. We expect our accelerometer to read a.

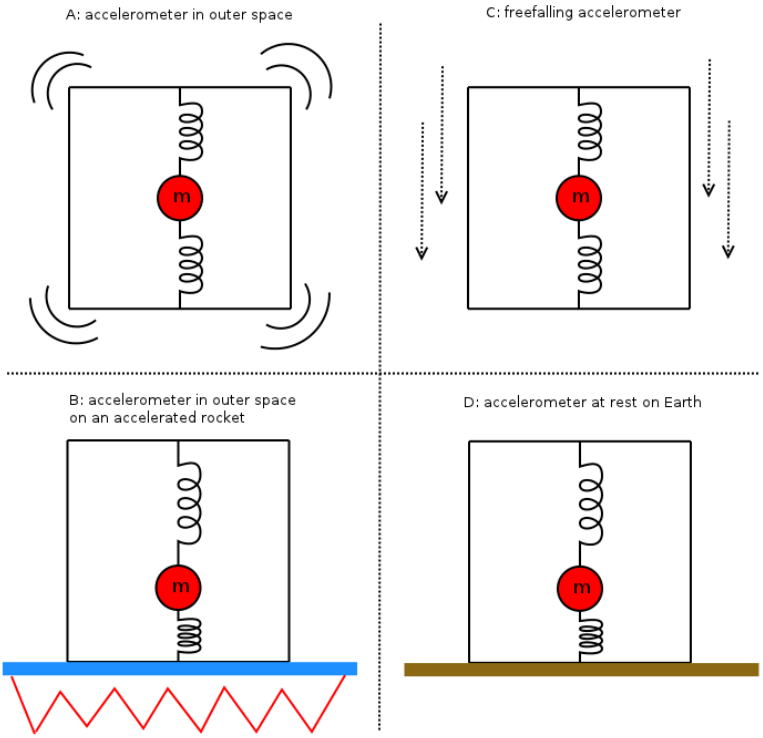

Figure 5.3: Effects of gravity and external accelerations to an accelerometer

Instead, if we let the accelerometer fall down under the effect of gravity and in absence of frictions caused by the air (figure 5.3 C) , the whole accelerometer will be accelerated by an acceleration g (with $g = 9.80665 \frac{m}{s^2}$). In such situation, called freefall, objects appear to have no weight, so the mass inside the accelerometer won't cause any displacement in the springs resulting in our accelerometer reading 0. This is a consequence of the fact that Newton's laws show that a body in free-fall follows is an inertial system such that the sum of the gravitational and inertial forces equals zero.

Finally, if we place our accelerometer at rest on the Earth (figure 5.3 D), our accelerometer will read a value of $a = g$. This is because the weight of the mass m will be subjected to gravity resulting in a displacement of the mass towards the bottom of the

accelerometer (just as in figure 5.3 B). So, an accelerometer placed at rest on the Earth will actually measure the normal force F_n acted by the ground on the accelerometer case. The reason of this is that the accelerometer case, with respect to a free-falling reference frame, is accelerating upwards.

With the above examples we demonstrated that an accelerometer is subject to the effect of gravity thus:

- when placed at rest on Earth it will read an acceleration $a = g$.
- during freefall it will read an acceleration $a = 0$.
- if we are interested in coordinate acceleration (change of velocity of the device in space) we have to remove gravity from the accelerometer output, doing what is called gravity compensation.
- if we rotate the accelerometer, the effect of gravity on its internal mass will variate with the rotation angle, giving a different output with different rotation angle. We can use this to implement tilt sensing with the accelerometer.

Tilt sensing with the accelerometer will be explored deeply in the next sections and chapters.

5.1.3 MEMS accelerometers

A MEMS accelerometer consist of a silicon chip, into which the sensor and the sensing structure are fashioned (see figure 5.4). It is made entirely of silicon and is in two parts: the first is a lump (often called the proof mass or seismic mass) suspended by means of a spring formed at each end; and the second is a pair of fixed sensing electrodes that enable the electronics to detect the movement of the lump relative to the surrounding platform of silicon.

When the chip is subjected to an acceleration, the lump moves a little relative to the chip and the fixed structures on it. The amount of movement depends on the size of the acceleration, the stiffness of the springs, and the mass of the lump. When the lump is deflected, the electrical capacitance between it and the sensing structures on the chip

changes, and this change is detected by the electronics, which converts it to a value for acceleration ([28]).

In a two or three axis accelerometer, this kind of structure is replicated, with the opportune change of orientation, for each of the accelerometer axis so that is possible to detect accelerations on each one of them.

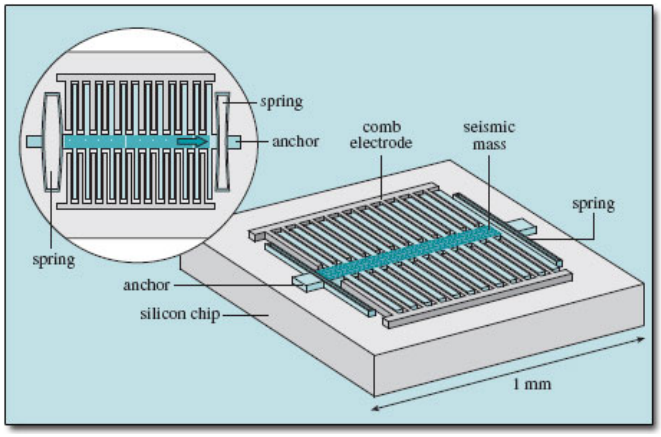

Figure 5.4: Detail of a typical MEMS accelerometer - Picture from ([28])

5.2 The gyroscope

A gyroscope is a device used to measure angular motion (eg. angular velocity). There are many kind of gyroscopes which operates following different principles but in general they can be grouped in two main categories: mechanical gyroscopes and optical gyroscopes.

The simplest mechanical gyroscope (figure [5.5]), invented by Foucault in 1852, is a spinning wheel mounted in a gimbaled structure capable of assuming any orientation. As the wheel is spinning, it has an high angular momentum which let the wheel maintain its orientation nearly fixed when an external torque is applied to the structure.

Unfortunately, gimbaled and optical gyroscope are quite large in size and quite expensive so, they are quite impractical for usage in small devices like mobile phones or mice. Over the last few years, vibrating structure gyroscopes have been introduced, which

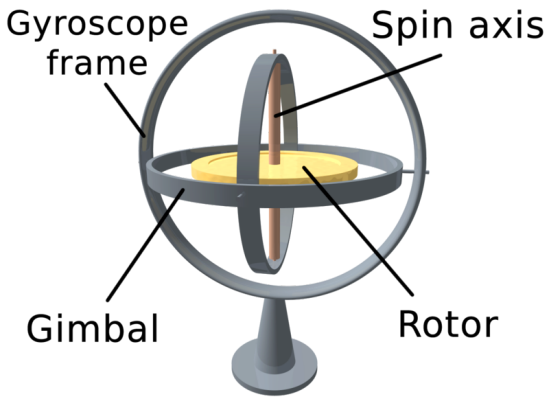

Figure 5.5: A mechanical gyroscope - Picture from (67).

can be produced using MEMS techniques resulting in small, inexpensive but precise devices.

5.2.1 Vibrating structure gyroscope

A vibrating structure gyroscope can me modeled as in figure 5.6. A mass m vibrates through the dotted trajectory at a speed V. When the gyroscope is rotated, the mass m is subjected to the Coriolis effect that causes a secondary vibration orthogonal to the original vibrating direction.

The Coriolis force is given by $\vec{F_C} = -2m(\vec{\Omega} * \vec{V})$ where $\vec{\Omega}$ is the angular rate of rotation and $\vec{V}$ is the velocity the mass m is moving.

In a similar way to the accelerometer model, also the Coriolis force is opposed by an elastic force produced by the springs surrounding the mass. Following Hooke's law, this force will be defined as $F_e = kx$ where x is the displacement and k is the system's elastic constant. F_e will always be opposite to F_C.

So, by measuring the displacement x, knowing the system elastic constant k and noting that we will always have $F_e = F_C$ we can then calculate the angular rate Ω.

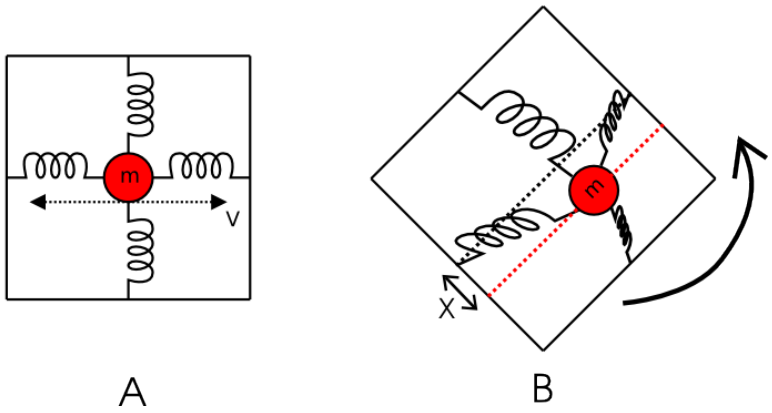

Figure 5.6: Model of a vibrating structure gyroscope - Note how the trajectory of the mass displaces by x from its original position when subjected to a rotation. Gravity does not affect this model.

5.2.2 MEMS gyroscope

The gyroscope model introduced in the previous section can be directly mapped into a MEMS gyroscope. In figure 5.7 a surface-micromachined vibratory rate gyroscope is depicted.

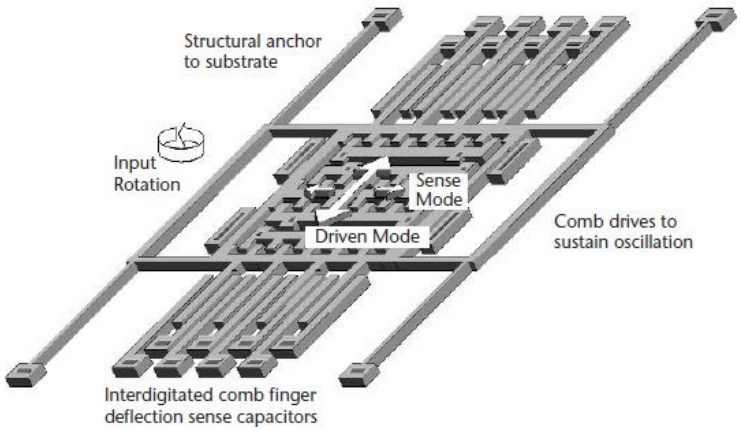

Figure 5.7: Detail of a Surface-micromachined vibratory rate gyroscope - Picture from (59).

Standard comb drive actuators are used to excite the structure to oscillate along one in-plane axis (x-axis), which allows relatively large drive amplitudes. Any angular rate

signal about the out-of-plane axis (z-axis) excites a secondary motion along the other in-plane axis (y-axis) (59).

This secondary motion causes the comb fingers sensors to deflect and this deflections is detected by the electronics, which convert it to the output of the gyroscope.

In a two or three axis gyroscope, this kind of structure is replicated, with the opportune change of orientation, for each of the gyroscope axis so that is possible to detect angular velocities on each one of them.

5.3 The Magnetometer

A magnetometer is a device used to measure the strength and/or direction of the magnetic field in the vicinity of the instrument (72). Magnetometers, as any magnetic sensitive device, are also subject to the influence of Earth's magnetic field so that it's possible to use them for calculating the device heading.

Magnetometers can be used in different kind of applications ranging from geophysical surveys to handheld GPS navigation systems and thus are available using different technologies in different sizes and costs depending on the specific application the magnetometer is used.

In size constrained applications, the Anisotropic Magnetoresistive (AMR) technology offers good precision and small device size for affordable costs so it's currently one of the most used technologies in handheld devices (6).

5.3.1 Anisotropic Magnetoresistive Sensor

Anisotropic Magnetoresistence is the property of a material in which a dependence of electrical resistance on the angle between the direction of electric current and orientation of an applied magnetic field is observed (73).

This behavior is shown in figure 5.8. A Permalloy thin film (NiFe) during fabrication has been deposited in a strong magnetic field (6) producing a magnetic field M on the film. When the film is at rest and there aren't external applied fields, the magnetic

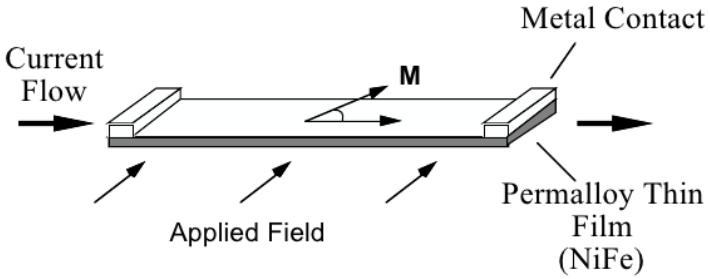

Figure 5.8: Principle of operation for Magnetoresistive Sensors - Picture from (22).

field M is parallel to the film and it's resistance is R. But, when an applied field H is present, the magnetic field M gets deviated resulting in a difference of resistance ΔR proportional to the angle between H and M.

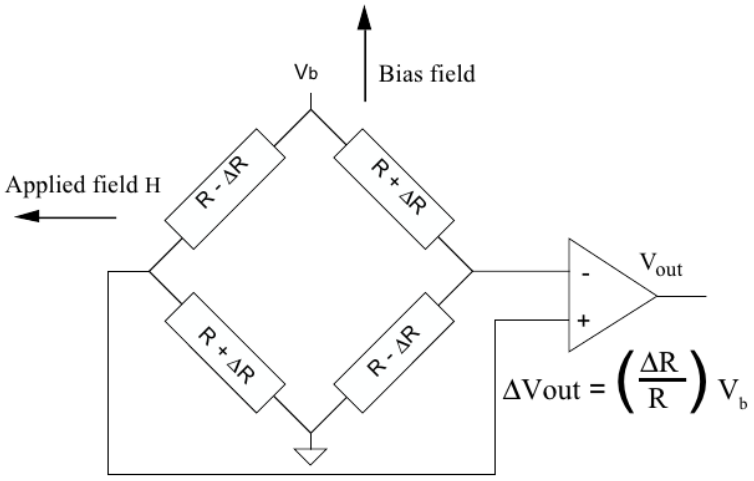

Figure 5.9: Magnetoresistive transducers - Picture from (22).

The transducer is in the form of a Wheatstone bridge (Figure 5.9). The resistance, R, of all four magnetoresistors is the same. The bridge supply, V_b, causes current to flow through the resistors. A crossed applied field, H, causes the magnetization in two of the oppositely placed resistors to rotate towards the current, resulting in an increase in the resistance, R. In the remaining two oppositely-placed resistors magnetization rotates away from the current resulting in a decrease in the resistance, R. In the linear range the

output becomes proportional to the applied field $\Delta V = SHV_b$. The range of linearity of the transfer function is inversely proportional to the sensitivity(3, 20, 22, 35, 40).

Is possible to produce this kind of structure using micromachined techniques resulting in very small sensors (figure 5.10).

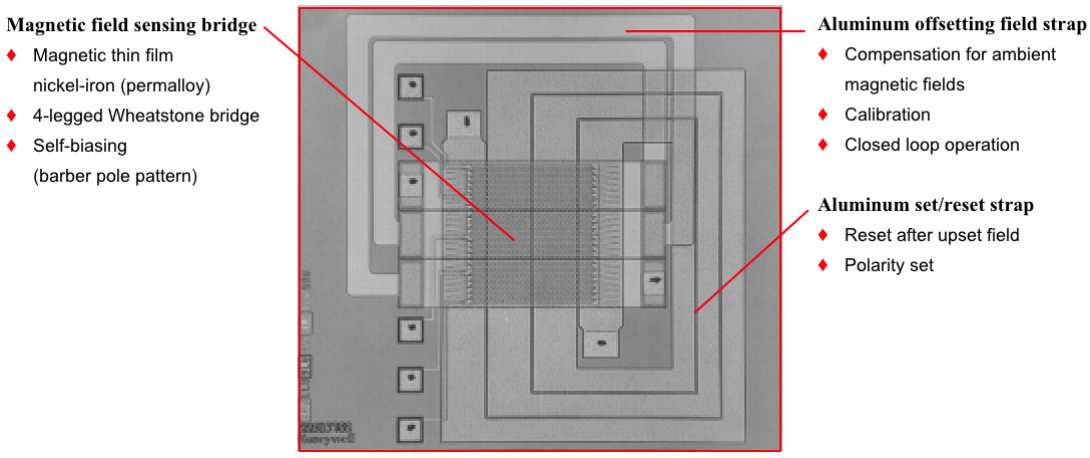

Figure 5.10: Magnetoresistive sensing element - Picture from (22).

5.4 ADXL330: an analog 3-axis accelerometer

The first sensor I used has been the ADXL330, an analog 3 axis accelerometer. It contains a polysilicon surface micromachined sensor and signal conditioning circuitry to implement an open-loop acceleration measurement architecture. The output signals are analog voltages that are proportional to acceleration. The accelerometer can measure the static acceleration of gravity in tilt sensing applications as well as dynamic acceleration resulting from motion, shock, or vibration (9).

The ADXL330 main features are:

- XYZ 3-axis acceleration sensing
- ± 3 g acceleration scale range
- 1.8 to 3.6 Volts single supply operation

- low power (180 uA at $V_s = 1.8$ V)
- selectable output bandwidth from 0.5 Hz to 1600 Hz (X and Y axes) and 0.5 Hz to 550 Hz for the Z axis
- ratiometric sensitivity of around 300 $\frac{mV}{g}$
- very small 4mm x 4mm x 1.45mm package (9).

The ADXL330 has been chosen as it was a very well known accelerometer whose documentation and code examples were widely available. Moreover the fact that it has been used with the first generation of the Nintendo Wii controller has been seen as a complete proof of quality.

5.4.1 Wrong Buying: learning by making mistakes

Unfortunately, we didn't have any ADXL330 available at the University so I had to go online and search for sellers of them. At that moment I hadn't actually an idea of what exactly I was looking for, I barely knew that I needed to buy an ADXL330.

Usually, when there is the need of using a surface mounted device (SMD), a device which comes in a package suitable only for soldering over a printed circuit board (PCB), a very small PCB, called breakout board, is used. The breakout board simply breaks out the pins of the SMD device into standard 0.1 inches pins so that its possible to use them on a breadboard.

However, this whole breakout board thing wasn't actually known to me at the time of buying the ADXL330. So, instead of buying a breakout board, for example the 30$ ADXL330 breakout board from Sparkfun (12) (figure 5.11 A), I actually bought a raw ADXL330 chip because it was cheaper, only 6$, and I simply assumed I was getting a breakout board.

Of course, I was really a beginner at that time and I had no idea that instead of getting a PCB with easy to use 0.1 inches pins, I had actually bought a 4 mm x 4 mm IC with 0.3 mm pins (figure 5.11 B). The reader can imagine my face when I opened that package and saw that tiny chip instead of an handy breakout board.

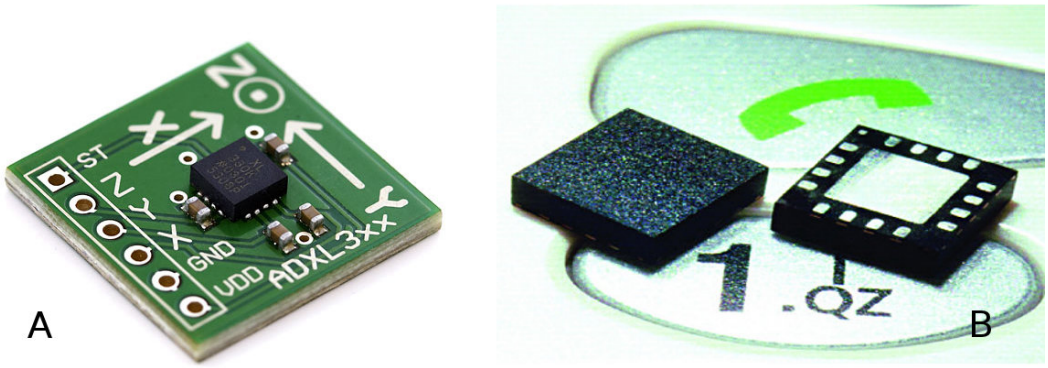

Figure 5.11: ADXL330 - A: Sparfun Electronics breakout board for the ADXL330. B: A raw ADXL330 chip. Pictures from (12, 41).

At that point I was a bit discomforted about my mistake but, instead of dumping the chip, I went to the Arduino forum and asked if there was the possibility to use the chip anyway. People there suggested that, by using a good magnifying glass and with some practice, it was actually possible to solder on those very tiny pads some wires doing some kind of very primitive wire based breakout board. An user on the forum even sent me some inspiring pictures of one of his attempts trying to solder on such tiny chips. The whole procedure looked hard but possible.

So, I asked for help to my uncle which is a long time electronic hobbyists who provided me with a professional soldering station and a magnifying glass with integrated lamp (really similar to those used by dentists). With these tools I had everything needed to solder directly on the ADXL330.

The result looked quite similar to figure 5.12. With some patience and by doing some tests I've been able to solder all the needed wires on the pad of the ADXL330.

The real trick for somebody without a proper education on electronic engineering like myself was actually understanding the ADXL330 datasheet which I have to admit looked quite terrifying the first time I saw it. But, documenting myself on books and online, I've been able to understand it and use it as a base for building the circuit to use it with Arduino.

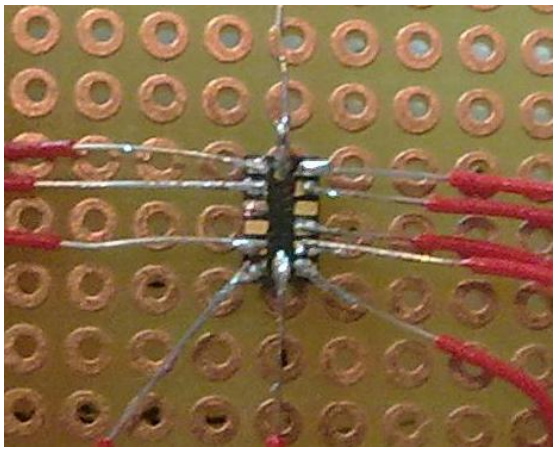

Figure 5.12: Tiny wires hand soldered to an SMD chip - This is actually an ADXL345 but the process followed for the ADXL330 was the same.

Making this mistake and buying the wrong chip has been a great learning experience which let me experience with SMD soldering and prototyping for the first time. This is something which has been crucial in the developments of the following months, as we will see in the following chapters and sections.

5.4.2 Electronic schematics for using the ADXL330 with Arduino

Once the soldering of the tiny wires on the accelerometer was complete, I soldered slightly bigger and more prototyping friendly wires to the smaller ones so that I could use them on a breadboard. The electronic schematics used to connect the ADXL330 to Arduino is displayed in figure 5.13.

The voltage supply pins (V_s, pins 14 and 15) have been connected to the 3.3 Volts pin on the Arduino. The 0.1 μF capacitor C1 is used to decouple the accelerometer from noise on the power supply. The various common pins (COM, pins 3, 5, 6 and 7) have been connected directly into Arduino GND pin. The AREF pin on the Arduino has been connected to the 3.3 Volts power source in order to use that voltage as reference to the ADC.

The accelerometer pins Xout, Yout and Zout (12, 10 and 8), whose serve as analogical signal pins, have been connected respectively to Analog 0, 1 and 2 on the Arduino.

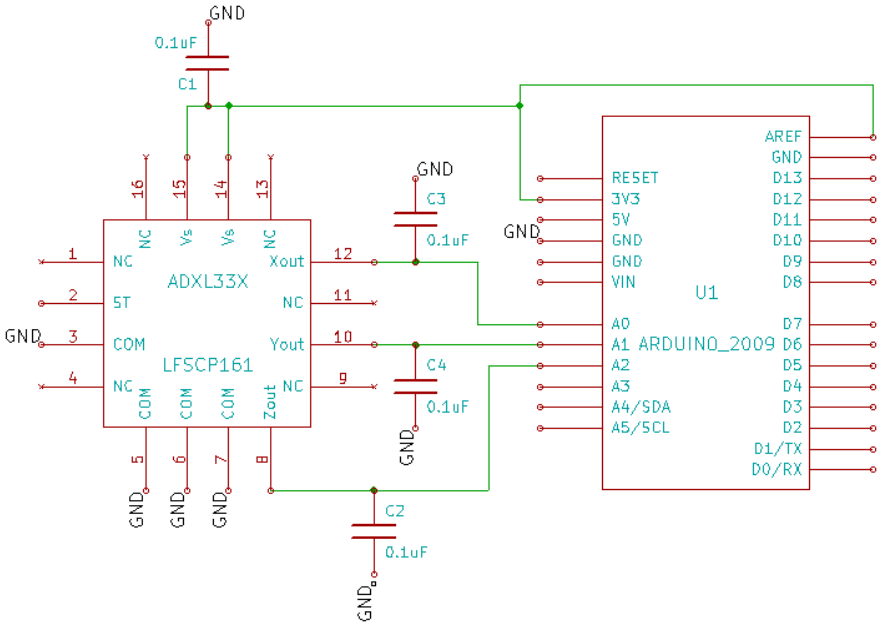

Figure 5.13: ADXL330 and Arduino Schematics - Pins labeled with GND are connected together

The bandwidth on these pins can be selected by adding capacitors to those pins. As per the ADXL330 datasheet, the bandwidth follows this simple formula:

$$F = \frac{5\mu F}{C_{X,Y,Z}}$$

In order to reach the output bandwidth of 50 Hz and to clean the outputs from noise, 0.1 μF capacitors (C2, C3, C4) have been added to the outputs.

5.4.3 Reading data from the ADXL330

With the above connections, the analog-digital converter of the ATMEGA 328p can be used to read the values from the accelerometer. The handy `analogRead()` function, which we introduced in section 4.3, can be used to read the analog value provided by the accelerometer.

By executing `analogRead()` on the accelerometer output pins we will obtain a value from 0 to 1023 proportional to the voltage read on the pin.

In order to convert the read value to something usable on our application, two things must be done:

- zero g offset calibration
- express the read value in g units

The zero g offset calibration is used to understand which read value we obtain from the accelerometer when the read acceleration is zero. Once the calibration is done, an offset is computed, which will be subtracted from the read value. This way we can differentiate from positive and negative acceleration values.

The calibration offsets can be obtained simply by rotating the accelerometer putting each axis in the zero g position. This is a position in which the expected value is zero which means placing the axis to calibrate to be perpendicular to gravity. The read value in this position can then be used as zero g offset. A more sophisticated calibration procedure is explained in (37).

Finally, we can express the read value in g units by noting from the datasheet that the accelerometer has a sensitivity of about 300 $\frac{mV}{g}$ when running at 3 Volts. As we are running it at 3.3 Volts and the sensitivity is ratiometric, it is safe to assume a sensitivity of 330 $\frac{mV}{g}$.

As the read value from the ADC is an integer from 0 to 1023, the AREF pin is connected to 3.3 Volts and our sensitivity is 330 $\frac{mV}{g}$, the following holds true:

$$\frac{0.33\,V}{3.3\,V} = \frac{1g\,val}{1023} \Longleftrightarrow 1g\,val = \frac{0.33\,V}{3.3\,V} * 1023 = 102.3$$

This means that the read value from the accelerometer on the Arduino can be converted into g units by dividing it by 102.3.

5.5 Digital sensors

After experiencing with the ADXL330, we started looking for additional sensors in order to implement some orientation sensing capable device (more in the next chapters). We needed to get a gyroscope and a magnetometer.

I started documenting about what were the most adequate sensors available at that time and it seemed that newly available sensors were more powerful and precise moreover they were also digital and somehow intelligent.

I decided that the ADXL345 accelerometer, the ITG3200 gyroscope and the HMC5843 were good candidates that were widely used in other project thus well known while still being quite new sensors. All these three sensors are completely digital, which means that they do embed digital logic capable of converting the analog signal coming from the mechanical components into digital values accessible using a digital communication protocol. Moreover, they all provided different configurable parameters which change how the sensor internally work (eg: configurable sampling rate) as well as configurable interrupt conditions.

5.5.1 I^2C

The three sensors chosen communicates to an host microcontroller using I^2C, a multi-master serial single-ended computer bus invented by Philips semiconductor division (now NXP) (68).

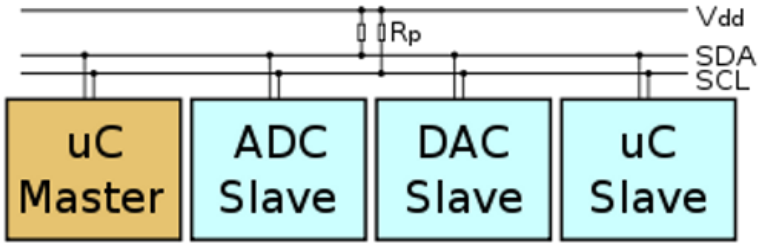

Figure 5.14: I^2C - example of connections

I^2C uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock (SCL), pulled up with resistors. Usually signal voltages used are +5 Volts or +3.3 Volts although systems with other voltages are permitted. Devices are connected in parallel to both the SDA and SCL lines (figure 5.14) and, as the wires are open-drain, they can pull the lines low. Common speeds for the I^2C bus are 100 kbit/s (standard mode) and 400 kbit/s (fast mode) but other speeds (10 kbit/s, 1 Mbit/s and

3.4 Mbit/s) are usable. Each device define its own I^2C speed capabilities which are usually present in the device's datasheet.

Devices on the bus have two roles:

master issues the clock and addresses slaves.

slave receive the clock signal and the addresses sent by the master. Can communicate replying to requests made by the master.

A detailed description of the I^2C communication protocol is out of scope here. For a detailed but informal and simple to understand description on it, the reader should consult (24).

We can make however a communication example (figure 5.15), just to give the reader an insight on the protocol functioning.

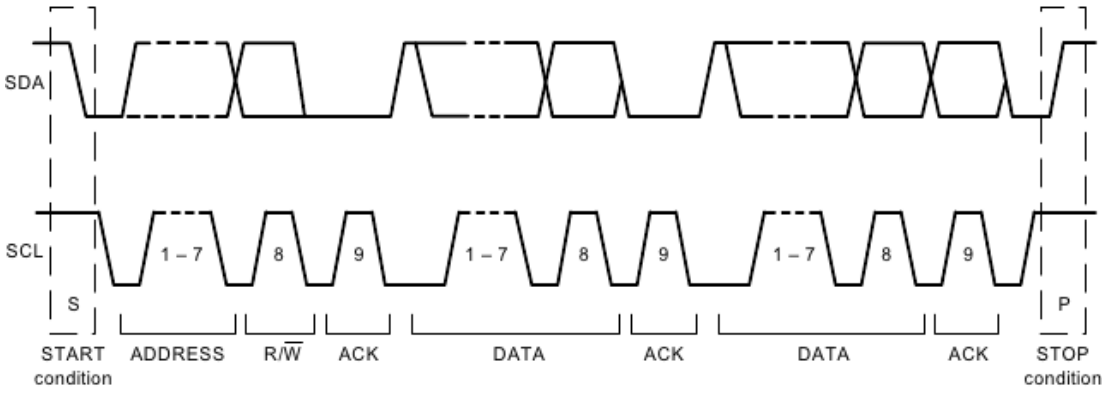

Figure 5.15: Complete I^2C Data Transfer - Picture from (23)

After beginning communications with the START condition (S), the master sends a 7-bit slave address followed by an 8th bit, the read/write bit. The read/write bit indicates whether the master is receiving data from or is writing to the slave device. Then, the master releases the SDA line and waits for the acknowledge signal (ACK) from the slave device. Each byte transferred must be followed by an acknowledge bit. To acknowledge, the slave device pulls the SDA line LOW and keeps it LOW for the high period of the SCL line. Data transmission is always terminated by the master with

a STOP condition (P), thus freeing the communications line. However, the master can generate a repeated START condition (Sr), and address another slave without first generating a STOP condition (P). A LOW to HIGH transition on the SDA line while SCL is HIGH defines the stop condition. All SDA changes should take place when SCL is low, with the exception of start and stop conditions (23).

5.5.2 Arduino and I²C

The I²C communication protocol can easily be used in Arduino using the Wire library (49) which provide relatively easy to use APIs for the protocol.

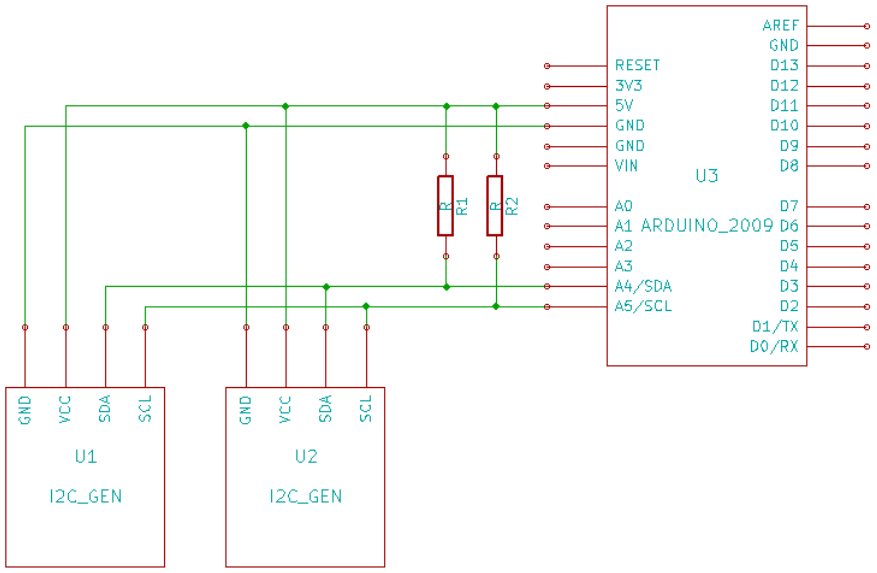

Figure 5.16: Arduino connected to two 5V I²C devices - R1 and R2 pullup values can range from 2K to 10K Ω.

On the Arduino Duemilanove, the SDA and SCL pins are respectively A4 and A5, so that those pins have to be connected to the correspondent pins on the slave device in order to use the I²C protocol (figure 5.16). Pullups resistors have to be added to the SDA and SCL lines: when connecting to 5 Volts devices, the pullups should be connected to 5 Volts; when connecting to 3.3 Volts devices the pullup should be connected to 3.3 Volts.

The Wire library by default enables the internal pullups of the microcontroller, which should be disabled when external pullups are used. It's very important that the internal pullups are disabled when connecting to a 3.3 Volts device as the internal pullups are connected to the voltage the microcontroller is using (5 Volts on 16Mhz Arduino like the Duemilanove), so that a 3.3 Volts device could get damaged when used with 5 Volts signals.

5.6 Low cost, do-it-yourself method for making printed circuit boards

As soon as we choose the sensors we were interested in using, I started documenting about the costs involved in buying breakout boards for those sensors. Prices were ranging from 30$ to 50$. Unfortunately, at that time my university was under some economical problems due to the cuts made by the Berlusconi government to the public university. It seemed that getting founds for buying those sensors was almost impossible.

So, it was clear that if I wanted to continue working on this thesis I had to buy those sensors from my own pockets. Fortunately, as I had worked during all the university years, I had some money saved which I could spend for buying the sensors. My budget was however quite limited and I surely couldn't afford buying ready to use breakout boards for the sensors.

The fact that I was able to buy the bare sensors for only a fraction of the costs of a commercial breakout board made me thinking about following the same soldering procedure used for the ADXL330. However, as the pins under the sensors chosen were very small, soldering below them would have been quite impossible: a different procedure was needed.

I started documenting on different procedures to use surface mounting devices (SMD) without professional tools and equipment and I found that there were some ways of building a printed circuit board (PCB) at home using the same procedure used by hobbyists.

The procedure consists in three main stages:

- designing the PCB using an electronic design automation (EDA) tool
- physically build the design starting from a copper clad board
- solder SMD components on the PCB.

The above stages will be briefly introduced in the following sections.

5.6.1 Designing a PCB with Kicad

The first step in building a PCB, is designing it. This is done by using a electronic design automation (EDA) tool which helps the designer drawing the circuit schematics and actual PCB using software aided tools.

There are many commercial EDA suites available, ranging from a hundred dollars to thousands depending of how many advanced features available in the package. EAGLE (18), an EDA suite by CadSoft Computer GmbH, is currently the most used PCB design tool among hobbyists. In this thesis, I wanted to use only libre software, so using EAGLE was not an option.

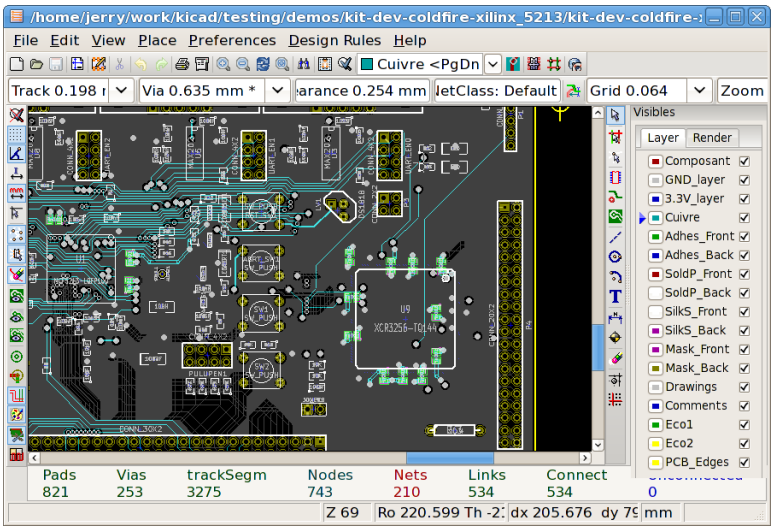

Figure 5.17: Screenshot of KiCad - In this picture from (60) the pcbnew component of KiCad is being used.

I started documenting on libre software EDA suites and I found out that KiCad (figure 5.17) (60), an electronic design automation released under the GNU GPL, was perfect for my simple needs of PCB design.

KiCad, is organized in five main parts:

kicad the project manager.

eeschema the schematic editor.

cvpcb the footprint selector for components used in the circuit design.

pcbnew the PCB layout program which also has a 3D View.

gerbview the Gerber (photoplotter documents) viewer.

The process of designing a PCB using KiCad can be summarized in the following steps:

1. design the circuit schematics using eeschema
2. associate to each component of the circuit a footprint using cvpcb
3. use the schematics and the footprint association to design the actual PCB using pcbnew
4. export the designs into a format suitable for printing or manufacturing the PCB.

A detailed description of how to design a PCB with KiCad is out of scope here. For all the details the reader can consult KiCad website (60) and the documentation provided in the software package.

5.6.2 Etching a PCB

After the PCB design is completed, the actual PCB has to be constructed. There are different procedures, but the press and peel sheets way seemed to be the most simple and inexpensive to follow. The procedure involves using an etchant to etch out the printed circuit from a copper clad board.

Once the PCB design is ready, a printer friendly output is generated (eg PDF). Such design is then printed mirrored into a press and peel (PNP) sheet using a laser printer (figure 5.18 1). The toner used by laser printers isn't actually ink, is rather a powder which is settled on the paper by heating it.

Once the toner is on the PNP sheet it can be settled on any other surface by simply heating it enough. If we place the PNP sheet above the copper clad board and then heat the PNP sheet enough by using an iron (figure 5.18 2), then the toner will settle on the copper clad board (figure 5.18 3).

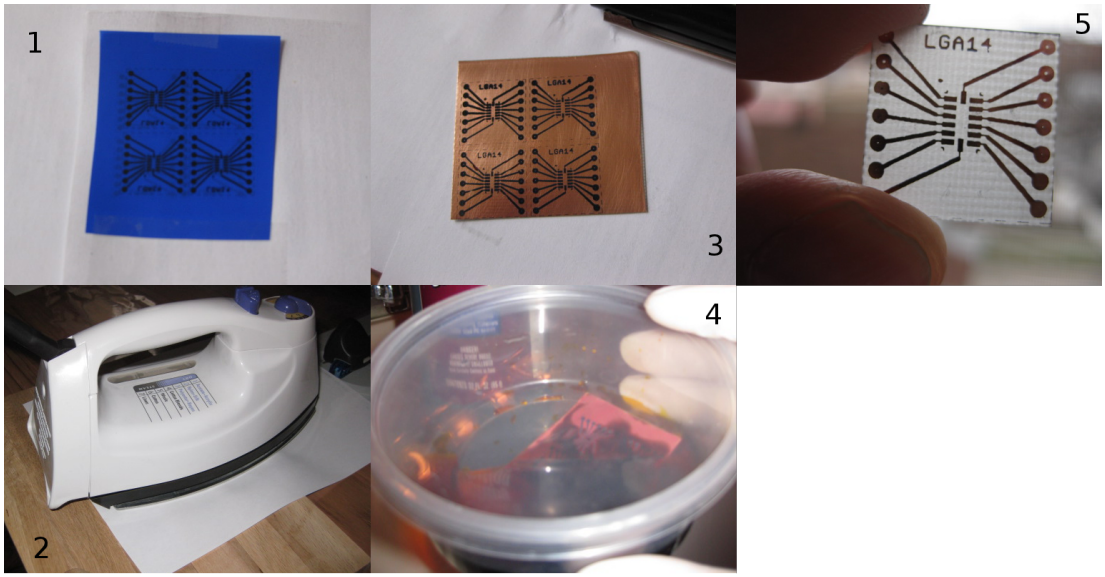

Figure 5.18: Etching a PCB using the PNP procedure - 1: press and peel sheets printed. 2: ironing the PNP sheet on the copper clad board. 3: toner settled on the copper clad board. 4: etching the PCB using ferric chloride 5: completed PCB. Pictures from (56)

The next step involves using ferric chloride ($FeCl_3$) to etch the PCB. The board is inserted into a container full of etchant (figure 5.18 4). The etchant will etch only the copper which isn't covered by any toner thus leaving nicely formatted tracks on the PCB. After about half an our the PCB etching is complete (figure 5.18 5).

IMPORTANT: Ferric chloride is a potentially dangerous substance: follow adequate safety measures when using it.

5.6.3 Soldering surface mounted devices on a PCB

Once the PCB has been produced the last step is soldering SMD devices on the PCB. This can be achieved using different procedures.

The simplest is placing solder on the contact pads of both the PCB and the SMD device using a soldering iron with the help of flux. Components can be positioned on top of their respective pads. The PCB can then be heated till the solder reflow temperature using a skillet or an hot air gun (figure 5.19 top).

Another way is using solder paste, a gray sticky material, which can be applied on the PCB with the help of a stencil. Once the solder paste is applied the components can be positioned. The whole PCB can then be heated using an hot air gun or a pizza oven till it reaches the reflow temperature. This heating process should follow the temperatures explained by the solder paste manufacturer reflow profile (figure 5.19 bottom).

Figure 5.19: Reflow soldering SMD devices on a PCB - Top images: reflow soldering using standard solder and skillet or hot air gun. Bottom images: solder paste SMD reflow using a pizza oven. Pictures from (39, 56)

5.7 ADXL345: a digital 3-axis accelerometer

The ADXL345 is a 3-axis digital-output MEMS accelerometer produced by Analog Devices. Its main features include:

- low power consumption: 40 μA in measurement mode and 0.1 μA in standby mode (with V_s = 2.5 Volts)
- high resolution (13-bit) at up to $\pm$16 g
- selectable bandwidth
- embedded FIFO queue to minimize communication overhead
- SPI and I^2C digital interfaces
- single and double tap detection
- activity and inactivity monitoring
- freefall detection
- configurable interrupts to two different interrupt pins

The ADXL345 has been chosen as it was widely known, for the quality of the documentation provided by Analog devices, for the presence of tap and double tap detection and of course for its pretty good resolution.

5.7.1 Schematics and PCB designs for a breakout board for the ADXL345

A simple breakout board has been designed for the ADXL345 using KiCad. The schematics are depicted in figure 5.20: the breakout board simply breaks out the SDA, SCL and INT1 pins from the ADXL345. The breakout board only supports I^2C connections. Both the V_s and V_{dd} pins have been connected to the same power source pin with associated 0.1 μF decoupling capacitors (C1 and C2). The various GND pins of the ADXL345 have been connected to the GND of the breakout board.

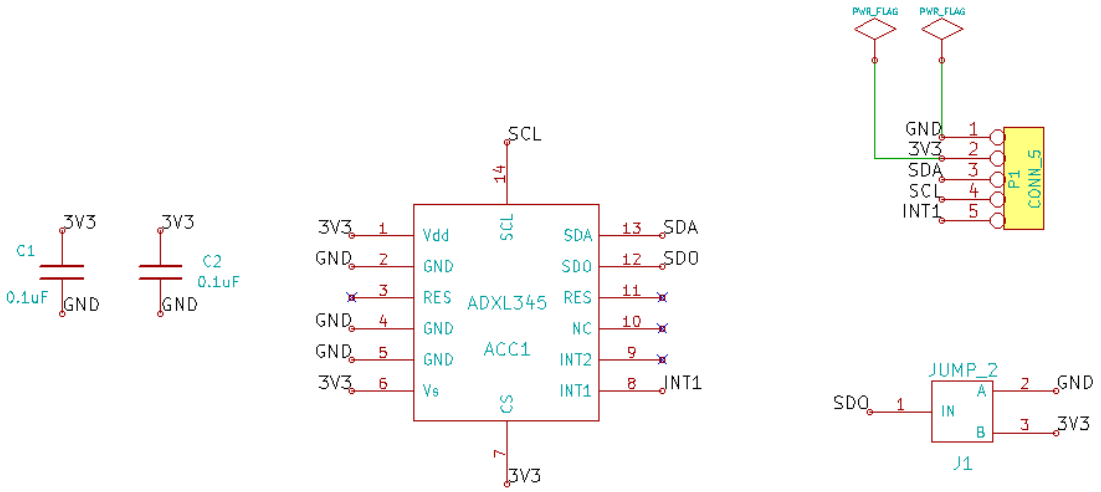

Figure 5.20: ADXL345 breakout board schematics - The PWR_FLAG components are KiCad specifics for using ERC validation.

The SDO pin, if connected to logic HIGH or LOW changes the device address on the I^2C bus. Such pin has been connected to a solder jumper (J1) so that the user can change between HIGH or LOW simply by connecting it to the needed logic level using a soldering iron.

The actual PCB of the breakout board of the ADXL345 is depicted in figure 5.21. A two side PCB has been designed using KiCad and etched using ferric chloride. 0805 size capacitors have been chosen for C1 and C2. Note the presence of 4 vias which connects the top layer with the bottom layer of the PCB. The copper tracks break out into 0.1 inches spaced connectors.

5.7.2 Using the ADXL345

The ADXL345 breakout board can be connected to an Arduino as any other I^2C device (see figure 5.16). The ADXL345 is a 3 Volts device so it should be connected to the 3.3 Volts pin on Arduino.

The accelerometer responds to the 7-bits I^2C addresses 0x1D (SDO connected to HIGH) and 0x53 (SDO connected to LOW). Output data and various internal configurations

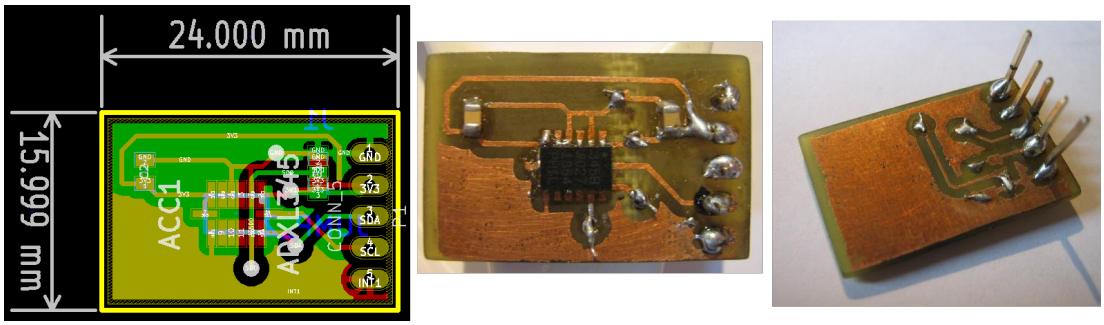

Figure 5.21: ADXL345 breakout board PCB - PCB design, Actual PCB top and bottom views.

are available by reading or writing to its internal registers.

The most interesting registers are DATAX0 to DATAZ1 (0x31 to 0x37) whose provide the 3-axis output acceleration, POWER_CTL (0x2D) which controls the power settings of the accelerometer and INT_ENABLE, INT_MAP and INT_SOURCE (0x2E to 0x30) whose controls the various interrupt settings. A detailed description of the various registers is available on the device datasheet (10).

5.8 ITG3200: a digital 3-axis gyroscope

The ITG3200 is a digital-output 3-axis MEMS gyroscope produced by Invensense. Its main features include:

- XYZ axis angular rate sensors with a 14.375 LSBs per deg/sec sensitivity and a full scale range of ±2000 deg/sec.
- integrated temperature sensor
- 6.5 mA operating current consumption
- 4x4x0.9 mm QFN package
- digitally programmable low-pass filter
- I^2C digital interface up to 400 KHz (23)

The ITG3200 has been chosen as it was the only I^2C compatible 3-axis gyroscope available at that time and it provided all the features needed.

5.8.1 Schematics and PCB designs for a breakout board for the ITG3200

A breakout board for the ITG3200 has been designed. Its schematics are reported in figure 5.22 and are based on the suggested connections from the datasheet (23). V_{dd} and V_{logic} pins have been connected together and broke out to the 3.3V pin of the breakout board connector. SDA, SCL, INT and AD0 pins have been simply broke out. The various capacitors used follow the value suggested in the datasheet. No external clock is used thus the CLKIN pin has been connected to GND. The AD0 pins allows user configurable I^2C address.

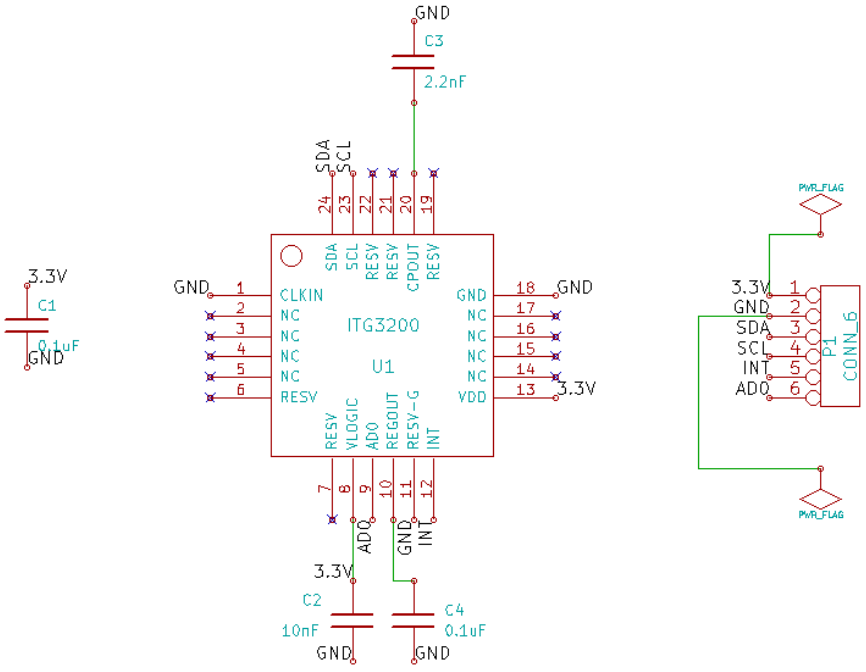

Figure 5.22: ITG3200 breakout board schematics

From the schematics in figure 5.22 a simple breakout board PCB has been designed (figure 5.23). The PCB has a two side design with 0805 capacitors and a 0.1 inches spaced connector and 5 vias which connects the top layer to the bottom layer of the PCB.

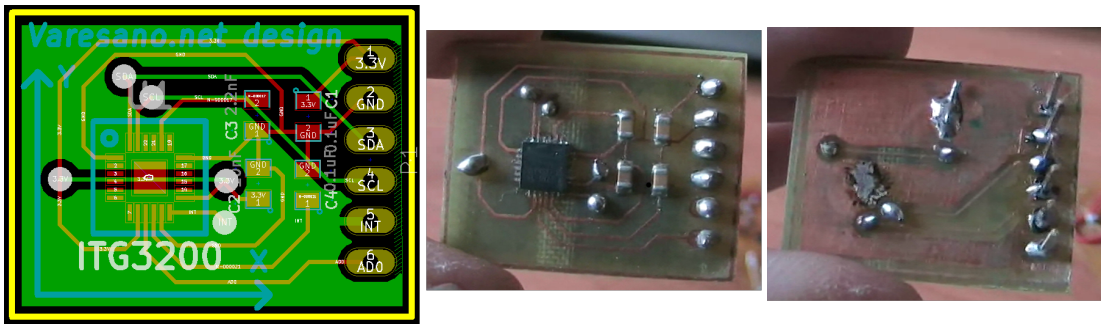

Figure 5.23: ITG3200 breakout board PCB - PCB design, Actual PCB top and bottom views.

5.8.2 Using the ITG3200

The ITG3200 can be accessed on the I^2C bus using the 0x69 (AD0 connected to HIGH) and 0x68 (AD0 connected to LOW) 7-bit addresses. After power up, the gyroscope needs 70 ms to start functioning (50ms from gyro startup + 20ms register r/w startup).

After this delay the output has to be configured by setting the sample rate and the lowpass filter bandwidth (registers SMPLRT_DIV 0x15 and DLPF 0x16). Output data from the temperature sensor and the XYZ angular rate outputs are available from registers 0x1B to 0x22 stored as 16-bit 2's complement data.

A detailed descriptions of all the registers is available on the ITG3200 datasheet (23).

5.9 HMC5843: a digital 3-axis magnetometer

The HMC5843 is a digital 3-axis magnetometer produced by Honeywell. Its main features are:

- 3-axis Anisotropic Magnetoresistive sensor with a 7 milli-gauss resolution and full scale range of ±4 gauss
- 4 x 4 x 1.3 mm surface mount package
- current draw of 0.8 mA

- maximum output rate of 50Hz
- I^2C digital interface (21)

The HMC5843 has been chosen as it was the only 3-axis magnetometer widely available at the time of writing this thesis.

5.9.1 Schematics and PCB designs for a breakout board for the HMC5843

A breakout board for the HMC5843 has been designed using KiCad. The schematics (figure 5.24) are based upon the suggestions made in the datasheet (21, single supply reference design) for a single supply connection.SDA, SCL and power lines have been broken out into a 0.1 inches connector. The value of the C1, C2, C3 capacitors follows the recommendations in the datasheet.

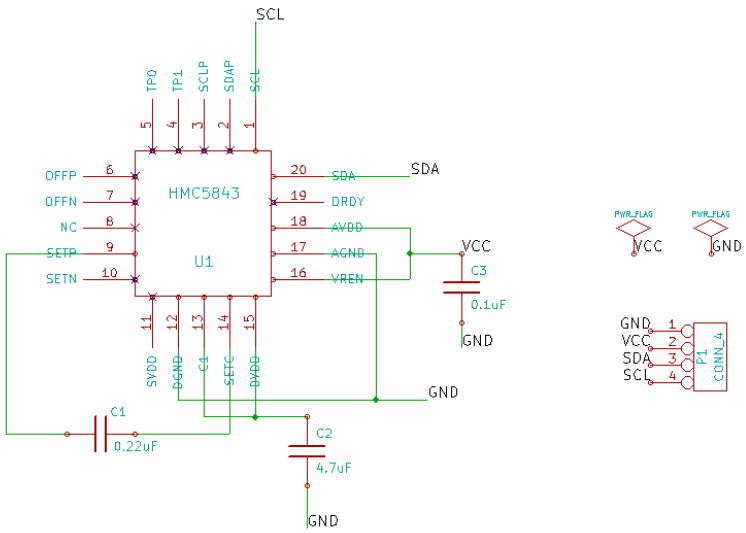

Figure 5.24: HMC5843 breakout board schematics

From the above schematics a breakout board PCB has been designed using KiCad (figure 5.25). As usual, 0805 package capacitors and a 0.1 inches spaced capacitor have been used.

It's important to note that the 4.7 μF C2 and 0.22 μF C1 have been chosen to have low equivalent series resistance (ESR) as suggested by the HMC5843 datasheet.

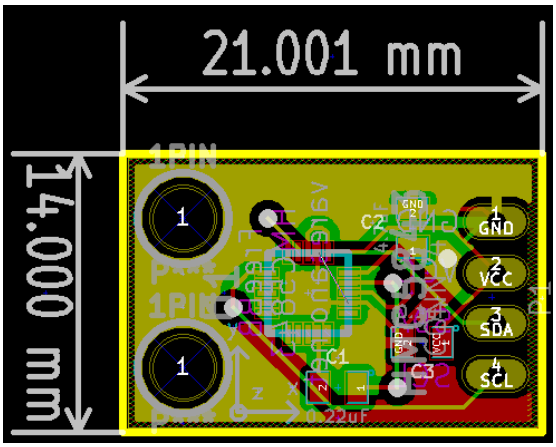

Figure 5.25: HMC5843 breakout board PCB

5.9.2 Using the HMC5843

The HMC5843 communicates on the I^2C bus using the 0x1E 7-bit address. The sensor provides a self test routine which can be used for a very simplistic calibration procedure. The HMC5843 provides two configuration registers (A and B - 0x00 and 0x01) whose allows the user to set various relevant settings such as the output bandwidth (from 0.5 Hz to 50 Hz) and the gain settings.

Relevant data registers are 0x03 to 0x08: they provide 2's complement 16 bit values (split into two 8-bit registers) for each axis. It's important to note that the HMC5843 will automatically wrap its internal pointer after reading register 0x08 again to 0x03 to ease the access to the data decreasing the I^2C communication overhead.

5.10 9 degrees of measurement MARG sensor array on a breadboard

By connecting the three breakout boards for the ADXL345, ITG3200 and the HMC5843 presented in the previous sections on the same I^2C bus, it's possible to create a 9 degrees of measurement (DOM - sometime incorrectly indicated as degrees of freedom - DOF) magnetic, angular rate, gravity (MARG) sensor. The three breakout boards can be connected in parallel on the I^2C bus as depicted in figure 5.26 and the three sensors

can be accessed as usual. It's convenient to place the 3-axis sensors so that their axis are aligned. The resulting circuit, prototyped on a breadboard is depicted in figure 5.27.

With this setup, is possible to start implementing various orientation sensing algorithms which will be presented in the next chapter.

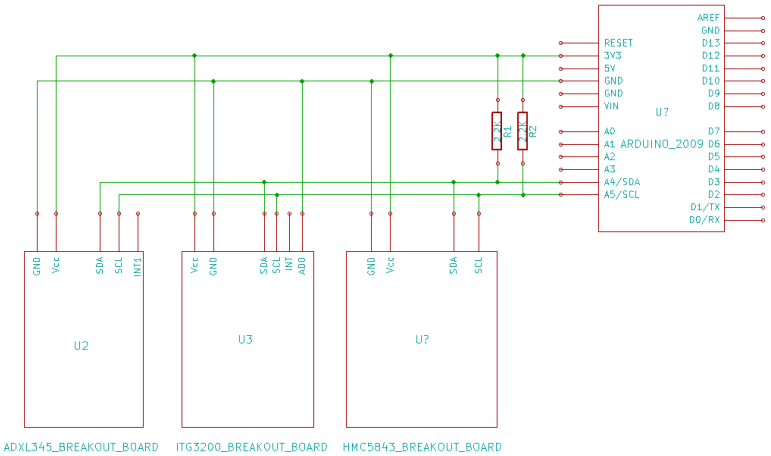

Figure 5.26: Schematics of a 9 DOM MARG sensor array using the ADXL345, ITG3200 and HMC5843 breakout boards. - A logic level converter could have been added to the schematics to avoid problems in case of a noisy connection.

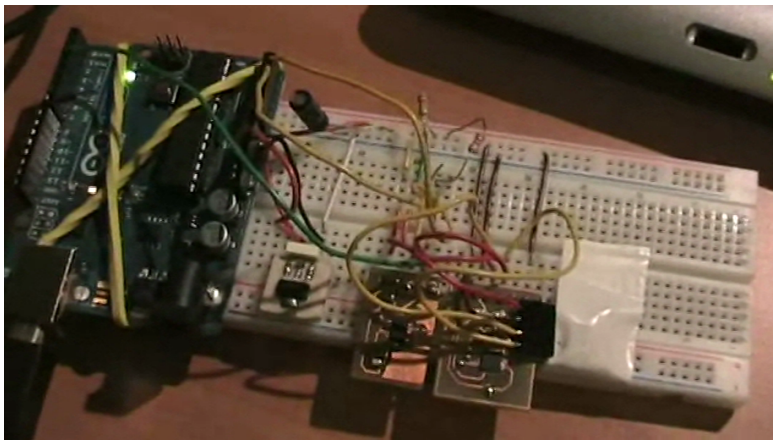

Figure 5.27: A 9 DOM MARG sensor array using the ADXL345, ITG3200 and HMC5843 breakout boards prototyped with Arduino.

6

Orientation Sensing

In the previous chapter I introduced the three sensors I used during this thesis, the respective breakout boards designed and the most important programming aspects for their usage.

In this chapter, I present the mathematical concepts and algorithms needed to fuse the sensors output into progressive levels of orientation sensing. I'll start by using only the accelerometer for a simple tilt sensing application and then I'll gradually add complexity till the implementation of a MARG sensor fusion algorithm for orientation sensing.

6.1 Tilt sensing using an accelerometer

The output of a three axis accelerometer, as seen in section 5.1.2 is subject to gravity, thus can be used for tilt sensing: we can compute the pitch, defined as the angle between the X_s axis and the horizontal plane, and the roll, defined as the angle between the Y_s axis and the horizontal plane (37).

6.1.1 Single axis tilt sensing

Let's start analyzing a simpler problem: the measurement of tilt using a single axis of the accelerometer. As depicted in figure 6.1, when an user holds an accelerometer (or

the device which contains it), the gravity force g is contrasted by the normal force F_n so that $g = F_n$. Our accelerometer measures proper accelerations so, as seen in section 5.1.2, it will actually measure the normal force acted by the user who holds the device.

When the user tilts the accelerometer by an angle α, the output of the accelerometer A_x will be the projection of the normal force F_n on the X axis of the accelerometer.

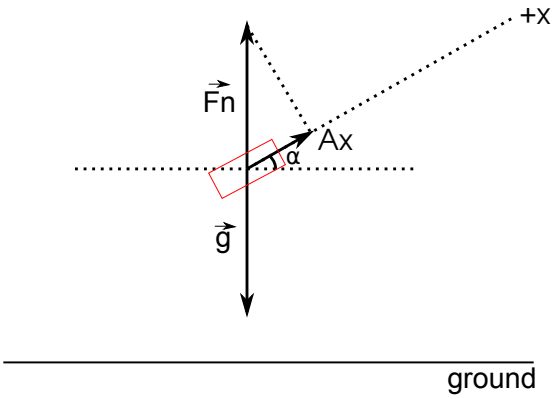

Figure 6.1: Tilt measurement using a single axis accelerometer

Referring to basic trigonometry it's possible to demonstrate that, for an ideal value of 1g for gravity, the output acceleration is

$$A_x[g] = 1g \times \sin(\alpha) \tag{6.1}$$

Using (6.1) it's possible to estimate the tilt angle:

$$\alpha = \arcsin\left(\frac{A_x[g]}{1g}\right) \tag{6.2}$$

Because this approach uses only a single axis and requires the gravity vector, the calculated angle of inclination is accurate only when the device is oriented such that the x-axis is always in the plane of gravity. Any rotation about the other axes reduces the magnitude of the acceleration on the x-axis and results in error in the calculated angle of inclination (15). In order to remove this constraint an accelerometer with more sensing axis is needed.

6.1.2 Tri-axis tilt sensing

With a three axis accelerometer, when the user tilts the accelerometer, the normal force F_n will be projected on all the three sensing axis. The problem of determining the pitch (θ), the angle between the X_s axis and the horizontal plane, and roll (ϕ), the angle between the Y_s axis and the horizontal plane (37) can be solved geometrically by analyzing picture 6.2.

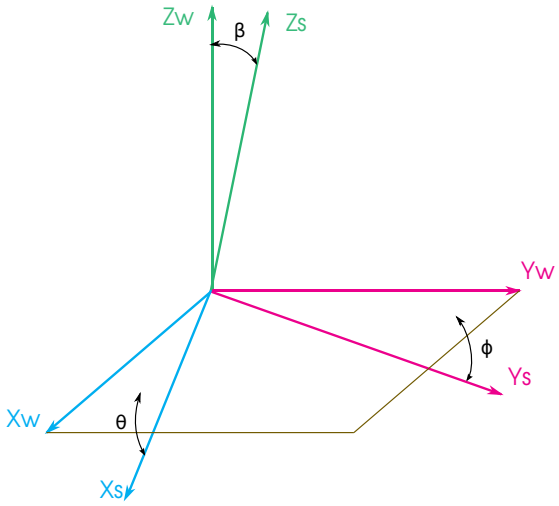

Figure 6.2: Tilt measurement using a three axis accelerometer - θ is the angle between X_s axis and the horizontal plane, ϕ is the angle between the Y_s axis and the horizontal plane and β is the angle between Z_s and Z_w.

With a simple trigonometric analysis we can obtain that:

$$\text{pitch} = \theta = \arctan\left(\frac{A_x}{\sqrt{{A_y}^2 + {A_z}^2}}\right) \tag{6.3}$$

$$\text{roll} = \phi = \arctan\left(\frac{A_y}{\sqrt{{A_x}^2 + {A_z}^2}}\right) \tag{6.4}$$

Similarly we can also compute β as the angle between the normal force and the Z_s axis of the sensor:

$$\beta = \arctan\left(\frac{\sqrt{{A_x}^2 + {A_y}^2}}{A_z}\right) \tag{6.5}$$

6.1.3 Limitations of using only an accelerometer for tilt sensing

As seen, using an accelerometer for tilt sensing is simple and straightforward. However, calculating tilt using only an accelerometer has some limitations. As the accelerometer will also varies its output due to external accelerations, if the user moves rapidly the device or if the device is being used in a vibrating environment (eg. in a car or a plane), the pitch and roll angles that we'll compute with the formulas presented above will be completely wrong and unreliable. Averaging various reading of the accelerometer can helps filtering out some external accelerations, but in general the formulas presented are only reliable with the assumption of an almost steady device.

Another limitation is that, with only an accelerometer, it's not possible to have any information on the yaw angle, defined as the angle between a fixed heading point (eg. Earth North) and the X_s axis of the device. This is a consequence of the fact that, using only gravity as reference vector, any rotation of the device around the gravity vector won't produce any difference in the output of the accelerometer.

6.2 Fusing accelerometer and gyroscope data for reliable tilt sensing

As seen, the method of determining tilt using only an accelerometer suffers from errors caused by external accelerations which sums to gravity and makes accelerometer based tilt sensing unreliable in presence of external accelerations and vibrations. Gyroscopes are much less subject to external accelerations and their angular rate output can be trusted even in presence of external accelerations. In this section, I present a simple algorithm adapted from (55, 57) which combines accelerometer and gyroscope data for tilt sensing so that its gravity vector output can be trusted even under the influence of external accelerations.

In figure 6.3, the normal vector R is displayed with respect to the sensors frame. A_{xz} is defined as the angle between R_{xz}, the projection of R on the xz plane, and Z. Similarly, A_{yz} is defined as the angle between R_{yz}, the projection of R on the yz plane, and Z.

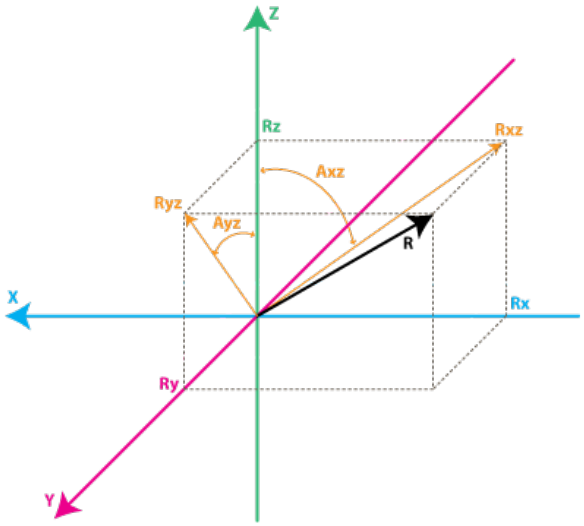

Figure 6.3: Normal vector R and projections angles - Picture from (57)

Let's suppose to sample data from the sensors each Δt seconds. With the notation $a(n)$ we will indicate the quantity a at the n-th sample which should occur $n\Delta t$ seconds from the beginning of the algorithm. The output of the algorithm will be $\vec{R}_e(n)$ which is the estimate of the normal force vector $\vec{R}$ at the n-th sample.

A 3 axis gyroscope will measure the angular rate ω around the X, Y, Z sensor axis and we will indicate them as ω_x, ω_y, ω_z, (this algorithm will only use ω_x and ω_y). If we know the angle $\mathrm{A}_{xz}(n-1)$, we can use the gyroscope output to compute $\mathrm{A}_{xz}(n)$ as

$$A_{xz}(n) = A_{xz}(n-1) + \omega_y \cdot \Delta t \tag{6.6}$$

The accelerometer will output it's measurement of $\vec{R}$, indicated as $\vec{R}_a$. As we are also making samples from the accelerometer, we indicate them as $\vec{R}_a(n)$. As initial conditions we can simply assume that $\vec{R}_e(0) = \vec{R}_a(0)$.

The key aspect of this algorithm is the derivation of $\vec{R}_g(n)$ which will be the estimation of $\vec{R}$ computed from the gyroscope data which will use only $\vec{R}_e(n-1)$ and $\vec{\omega}(n)$ as read from the device.

Supposing to be at the n-1-th step of the algorithm, we would know $\vec{R}_e(n-1)$. By

looking at 6.3, we can compute $A_{xz}(n-1)$ from $\vec{R_e}(n-1)$ with:

$$A_{xz}(n-1) = \text{atan2}(R_{e,x}(n-1), R_{e,y}(n-1)) \tag{6.7}$$

where atan2 is a variation of the arctangent function defined as in (65).

As we already seen in equation 6.6, we can update the angle to its n-th value using

$$A_{xz}(n) = A_{xz}(n-1) + \omega_y \cdot \Delta t \tag{6.8}$$

In the same way we can compute $A_{yz}(n)$

$$A_{yz}(n) = A_{yz}(n-1) + \omega_x \cdot \Delta t \tag{6.9}$$

Let's indicate $R_{g,x}(n)$ with x, $R_{g,y}(n)$ with y and $\vec{R_{g,z}}(n)$ with z in the following calculations. Assuming $\vec{R}$ normalized we can write:

$$x = \frac{x}{1} = \frac{x}{\sqrt{x^2+y^2+z^2}} \tag{6.10}$$

By dividing both the numerator and denominator by $\sqrt{x^2+z^2}$, we obtain:

$$x = \frac{\frac{x}{\sqrt{x^2+z^2}}}{\sqrt{\frac{x^2+y^2+z^2}{x^2+z^2}}} \tag{6.11}$$

Note that $\frac{x}{\sqrt{x^2+z^2}} = \sin(A_{xz})$, so:

$$x = \frac{\sin(A_{xz})}{\sqrt{1+\frac{y^2}{x^2+z^2}}} \tag{6.12}$$

Multiplying numerator and denominator of fraction inside the square root by z^2, we obtain:

$$x = \frac{\sin(A_{xz})}{\sqrt{1+\frac{y^2z^2}{(x^2+z^2)z^2}}} \tag{6.13}$$

But, $\frac{z}{\sqrt{x^2+z^2}} = \cos(A_{xz})$ and $\frac{y}{z} = \tan(A_{yz})$, so:

$$x = \frac{\sin(A_{xz})}{\sqrt{1+\cos^2(A_{xz})\tan^2(A_{yz})}} \tag{6.14}$$

Going back to the initial notation, we obtain:

$$R_{g,x}(n) = \frac{\sin(A_{xz}(n))}{\sqrt{1 + \cos^2(A_{xz}(n)) \tan^2(A_{yz}(n))}} \tag{6.15}$$

Similarly, we can obtain:

$$R_{g,y}(n) = \frac{\sin(A_{yz}(n))}{\sqrt{1 + \cos^2(A_{yz}(n)) \tan^2(A_{xz}(n))}} \tag{6.16}$$

The value of $R_{g,z}(n)$ can be obtained as:

$$R_{g,z}(n) = \sqrt{1 - R_{g,x}(n)^2 - R_{g,y}(n)^2} \tag{6.17}$$

and the same sign of $R_{e,z}(n-1)$ can be used for $R_{g,z}(n)$.

Now, we have $\vec{R_a}(n)$ from the accelerometer and $\vec{R_g}(n)$ from the computation above. We can fuse them into $\vec{R_e}(n)$ using a weighted average as:

$$\begin{aligned} \vec{R_e}(n) &= \frac{\vec{R_a}(n) \cdot w_1 + \vec{R_g}(n) \cdot w_2}{w_1 + w_2} \\ &= \frac{\vec{R_a}(n) \cdot \frac{w_1}{w_1} + \vec{R_g}(n) \cdot \frac{w_2}{w_1}}{\frac{w_1 + w_2}{w_1}} \\ &= \frac{\vec{R_a}(n) + \vec{R_g}(n) \cdot w_g}{1 + w_g} \end{aligned} \tag{6.18}$$

where $w_g = \frac{w_2}{w_1}$ can be sized empirically to the practical applications. Usually values from 5 to 20 produce good results.

By normalizing $\vec{R_e}(n)$ we obtain the output of the algorithm.

6.3 Tilt compensated digital compass

As we seen in section 5.3, a magnetometer, as any magnetic sensitive device, is also subject to the influence of Earth's magnetic field so that it's possible to use it to calculate the device heading, intended as the angle between Earth's magnetic north and the X sensing axis of the device.

Suppose to have a three axis magnetometer sitting on the local horizontal plane (the plane normal to the Earth's gravity vector). With this assumption, the effect of the

Earth's magnetic field would impact only the X and Y axis of the magnetometer (we are not taking into account the inclination of the magnetic field). With these assumptions, we can compute the device heading (ψ) simply using:

$$\text{heading} = \psi = \text{atan2}(Y_h, X_h) \tag{6.19}$$

where with Y_h and X_h we indicate the device's X and Y axis when it's sitting on the horizontal plane.

When the three axis compass is tilted, pitch and roll angles are not zero (figure 6.4) and a more complex approach than equation 6.19 is needed.

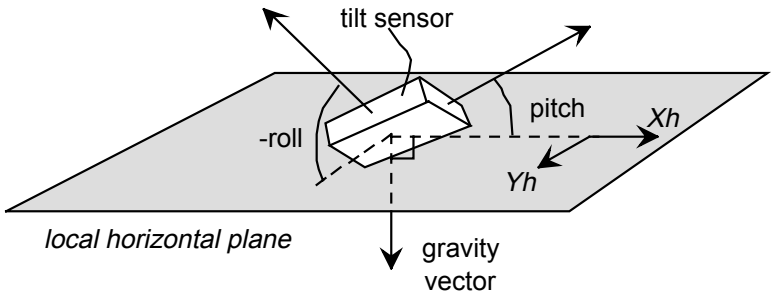

Figure 6.4: A tilted compass - Pitch and roll angles have to be taken into account when calculating heading. Picture from (7)

When the compass tilted with roll (ϕ) and pitch (θ) tilt angles referenced respectively as the angles between the Y and X axis and the horizontal plane. The X, Y, and Z magnetic readings can be transformed to the horizontal plane (X_h and Y_h) by applying the rotation equations 6.20 and 6.21. If these equations are not used, then appreciable errors will result in the heading calculations (7).

$$X_h = X \cdot \cos(\theta) + Y \cdot \sin(\phi) \cdot \sin(\theta) - Z \cdot \cos(\phi) \cdot \sin(\theta) \tag{6.20}$$

$$Y_h = Y \cdot \cos(\phi) + Z \cdot \sin(\phi) \tag{6.21}$$

Once we have X_h and Y_h we can simply compute the heading (ψ) using equation 6.19.

Of course, as this simple sensor fusion of magnetometer and accelerometer data doesn't make use of a gyroscope, this approach is also subject to the same problem of the tilt

sensing using only an accelerometer. As the accelerometer will sense accelerations plus gravity, this approach is only accurate assuming an almost steady device. In presence of external accelerations, this approach will fail giving incorrect heading information.

The reader may think about using the gyroscope+accelerometer gravity sensing algorithm presented above as source of pitch and roll to feed into equation 6.20. This approach, even if possible, wouldn't be optimal, both in term of performance and accuracy. In the next section, a better algorithm for sensor fusion is discussed.

6.4 Accelerometer, gyroscope and magnetometer fusion for orientation sensing

As seen in the previous sections, by fusing the data from different kind of sensors (accelerometers, gyroscopes and magnetometers), it's possible to sensibly increase the reliability of the orientation sensing capabilities of the algorithms. I already presented algorithms to fuse accelerometer and gyroscopes as well as magnetometer and accelerometer. In this section I present an algorithm which, by fusing the data coming from the whole MARG sensor, is capable of 3 degrees of freedom orientation sensing, thus can compute yaw, pitch and roll of the sensor frame with respect to the world frame.

The algorithm is based on the work by Robert Mahony et al (1, 13, 33, 34), especially (33), originally developed for usage on unmanned aerial vehicles, whose first publicly available implementation has been done by William Premerlani et al (44), known as the *DCM filter*. The algorithm has been then extended by Sebastian Madgwick to incorporate the magnetic distortion compensation algorithms from his filter (31).

The actual code I used is based on a reference quaternion implementation made by Madgwick which has been adapted to our sensors and Arduino APIs by myself. To my knowledge, this orientation filter represent the current state of the art in orientation sensing, being a very accurate but fast algorithm which can be implemented even on low cost microcontrollers like the ATMEGA 328p in the Arduino. A good introduction to the basic concepts used in the algorithm is available in (44). The algorithm uses quaternions to represents rotations: a good introduction to using quaternions to represent rotation is available in (43) and (32).

6.4.1 Orientation from angular rate

As seen in the previous chapter, a gyroscope will measure the angular rates about the x, y and z axes of the sensor frame, termed ω_x, ω_y and ω_z. If these rates are arranged in a vector ${}^S\boldsymbol{\omega}$ defined as:

$$ {}^S\boldsymbol{\omega} = [0 \quad \omega_x \quad \omega_x \quad \omega_z] \tag{6.22} $$

then it's possible to describe the rate of change of orientation of the earth frame relative to the sensor frame as the quaternion derivative

$$ {}^S_E\dot{\boldsymbol{q}} = \frac{1}{2} {}^S_E\hat{\boldsymbol{q}} \otimes {}^S\boldsymbol{\omega} \tag{6.23} $$

where $\otimes$ is the quaternion product determined using the Hamilton rule (32).

Let's suppose to sample gyroscope readings with a sample period Δt, ${}^S\boldsymbol{\omega}_n$ will be the n-th sample which will occur at $n\Delta t$. Supposing to know the initial orientation ${}^S_E\hat{\boldsymbol{q}}_{e,0}$, we can compute the estimated orientation of the earth frame relative to the sensor frame at the n-th sample, ${}^S_E\hat{\boldsymbol{q}}_{e,n}$, by numerically integrating ${}^S_E\dot{\boldsymbol{q}}_n$ as (31):

$$ {}^S_E\dot{\boldsymbol{q}}_n = \frac{1}{2} {}^S_E\hat{\boldsymbol{q}}_{e,n-1} \otimes {}^S\boldsymbol{\omega}_n \tag{6.24} $$

$$ {}^S_E\hat{\boldsymbol{q}}_{e,n} = {}^S_E\hat{\boldsymbol{q}}_{e,n-1} + {}^S_E\dot{\boldsymbol{q}}_n \cdot \Delta t \tag{6.25} $$

This approach, even if correct in theory, won't be accurate practically. Gyroscope drifting and numerical errors in the integration will progressively add drifting to the computed orientation estimate making this approach inadequate when used with low cost MEMS gyroscopes. The idea behind the sensor fusion algorithm is to use observations of gravity and earth's flux vectors obtained respectively from the accelerometer and magnetometer to compute an adjusted measurement of ${}^S\boldsymbol{\omega}$, which we'll call ${}^S\boldsymbol{\omega}^a$, to limit the effects of drifting in the orientation estimate.

6.4.2 Algorithm inputs and outputs

From a 9 DOM MARG sensor array, we can obtain the following measurements:

- ${}^S\boldsymbol{\omega}$: the angular rate about the x, y and z axes of the sensor frame,

- ${}^S\boldsymbol{a}$: the projection of the gravity vector and external accelerations on the axes of the accelerometer,
- ${}^S\boldsymbol{m}$: the projection of earth's flux vector on the axes of the magnetometer.

With ${}^S\hat{\boldsymbol{a}}$ and ${}^S\hat{\boldsymbol{m}}$ we will indicate the extension of the three elements vectors ${}^S\boldsymbol{a}$ and ${}^S\boldsymbol{m}$ with a fourth element 0 placed on the head of the three element vector. This extension will simplify operation with quaternions.

We can consider ${}^S\boldsymbol{\omega}$, ${}^S\boldsymbol{a}$, ${}^S\boldsymbol{m}$ as the inputs to our algorithm. The output will be ${}^S_E\hat{\boldsymbol{q}}_{e,n}$

6.4.3 Algorithm step

Let's suppose to have just completed the $n-1$-th step of the algorithm. Then we would know ${}^S_E\hat{\boldsymbol{q}}_{e,n-1}$. As we know that the gravity vector is always normal to the ground plane, we can compute the estimated direction of the gravity vector in the sensor frame from ${}^S_E\hat{\boldsymbol{q}}_{e,n-1}$ using:

$$\begin{aligned} {}^S\hat{\boldsymbol{v}}_{n-1} &= {}^E_S\hat{\boldsymbol{q}}_{e,n-1} \otimes {}^E\hat{\boldsymbol{v}} \otimes {}^E_S\hat{\boldsymbol{q}}^*_{e,n-1} \\ &= {}^S_E\hat{\boldsymbol{q}}^*_{e,n-1} \otimes {}^E\hat{\boldsymbol{v}} \otimes {}^S_E\hat{\boldsymbol{q}}_{e,n-1} \end{aligned} \tag{6.26}$$

Let's suppose to also know the earth's magnetic field vector ${}^E\hat{\boldsymbol{b}}$. With the same approach used above we can express the estimated direction of the magnetic field vector in the sensor frame from ${}^S_E\hat{\boldsymbol{q}}_{e,n-1}$ using:

$$\begin{aligned} {}^S\hat{\boldsymbol{b}}_{n-1} &= {}^E_S\hat{\boldsymbol{q}}_{e,n-1} \otimes {}^E\hat{\boldsymbol{b}} \otimes {}^E_S\hat{\boldsymbol{q}}^*_{e,n-1} \\ &= {}^S_E\hat{\boldsymbol{q}}^*_{e,n-1} \otimes {}^E\hat{\boldsymbol{b}} \otimes {}^S_E\hat{\boldsymbol{q}}_{e,n-1} \end{aligned} \tag{6.27}$$

We can express the error made between the estimated orientation and the correct one as the cross product between reference direction of fields and direction measured by sensors:

$$\boldsymbol{e}_n = {}^S\boldsymbol{a}_n \times {}^S\boldsymbol{v}_{n-1} + {}^S\boldsymbol{m}_n \times {}^S\boldsymbol{b}_{n-1} \tag{6.28}$$

We can track the errors done in the various steps by integrating $\boldsymbol{e}_n$:

$$S(\boldsymbol{e}_n) = S(\boldsymbol{e}_{n-1}) + \boldsymbol{e}_n \cdot k_i \tag{6.29}$$

where k_i is the integral gain which governs rate of convergence of gyroscope biases.

We can now compute ${}^{S}\boldsymbol{\omega}_n^a$, the adjusted gyroscope readings, as:

$${}^{S}\boldsymbol{\omega}_n^a = {}^{S}\boldsymbol{\omega}_n + \boldsymbol{e}_n \cdot k_p + S(\boldsymbol{e}_n) \tag{6.30}$$

where k_p is the proportional gain which governs rate of convergence to accelerometer and magnetometer.

Finally, we can compute the algorithm output as:

$${}^{S}_{E}\hat{\boldsymbol{q}}_{e,n} = {}^{S}_{E}\hat{\boldsymbol{q}}_{e,n-1} + \left(\frac{1}{2}{}^{S}_{E}\hat{\boldsymbol{q}}_{e,n-1} \otimes {}^{S}\boldsymbol{\omega}_n^a\right) \cdot \Delta t \tag{6.31}$$

6.4.4 Magnetic distortion compensation

In the previous section we assumed to know the earth's magnetic field vector ${}^{E}\hat{\boldsymbol{b}}$ which we used as reference to compute $\boldsymbol{e}_n$. But this is not the case as the magnetic field can be distorted due to inferences and magnetic field inclination.

The measured direction of the earth's magnetic field in the earth frame at the n-th sample, ${}^{E}\hat{\boldsymbol{h}}_n$, can be computed as the normalized magnetometer measurement, ${}^{S}\hat{\boldsymbol{m}}_t$, rotated by the estimated orientation of the sensor provided by the filter:

$${}^{E}\hat{\boldsymbol{h}}_n = [0 \quad h_x \quad h_x \quad h_z] = {}^{S}_{E}\hat{\boldsymbol{q}}_{e,n-1} \otimes {}^{S}\hat{\boldsymbol{m}}_n \otimes {}^{S}_{E}\hat{\boldsymbol{q}}^{*}_{e,n-1} \tag{6.32}$$

The effect of an erroneous inclination of the measured direction earth's magnetic field, ${}^{E}\hat{\boldsymbol{h}}_n$, can be corrected if the filter's reference direction of the earth's magnetic field, ${}^{E}\hat{\boldsymbol{b}}_n$, is of the same inclination. This is achieved by computing ${}^{E}\hat{\boldsymbol{b}}_n$ as ${}^{E}\hat{\boldsymbol{h}}_n$ normalized to have only components in the earth frame x and z axes:

$${}^{E}\hat{\boldsymbol{b}}_n = [0 \quad \sqrt{h_x^2 + h_y^2} \quad h_x \quad h_z] \tag{6.33}$$

Compensating for magnetic distortions in this way ensures that magnetic disturbances are limited to only affect the estimated heading component of orientation (31).

7

FreeIMU

In section 5.10, I presented a 9 degrees of measurement solution based on three separate breakout boards for the ADXL345 accelerometer, the ITG3200 gyroscope and the HMC5843 magnetometer. A breadboard has been used to prototype a magnetic, angular rate, gravity (MARG) sensor array which can be used to implement orientation sensing algorithms, as seen in chapter 6.

In figure 5.27, we can see that the size and connection complexity of this solution is quite relevant. It can works for prototyping the various algorithms and software but it surely poses quite some limitations when the same sensors have to be used inside a small device like a mouse or a remote controller. When I started trying using the same setup inside a size constrained device, it became clear that a smaller and integrated solution was needed.

At that time, there were no commercial boards incorporating a 9 degrees of measurement board having new generation I^2C based sensors. The available products used all analog sensors and were quite limited in precision and features. From the experience obtained from the design of the various breakout boards, I had all the knowledge to develop an integrated solution for a 9 degrees of measurement MARG sensor board.

In this chapter, I present FreeIMU, a 9 degrees of measurement board I designed and built. FreeIMU incorporates the ADXL345 accelerometer, the ITG3200 gyroscope and the HMC5843 magnetometer on a single and small printed circuit board.

7.1 Dorkbot PDX group PCB buying service

When I designed the various breakout boards described in the previous chapters, I used the do-it-yourself method of using ferric chloride to etch the PCB out from a copper clad board (section 5.6). As seen, this simplistic approach works well on simple PCB designs but it surely has quite some big limitations when used in more complex designs. Etching a PCB for a three sensors board like FreeIMU wasn't a viable solution.

There are many companies which can fabricate professionally constructed PCBs from an EDA tool like KiCad. Professionally constructed PCBs can use small tracks and vias size, allowing to squeeze even complex designs into small PCBs. Usually, the PCB design software has an export feature which outputs Gerbers files, the files used by many photo-plotters used by these companies in the manufacturing process of the actual PCB. Producing PCB in small quantities for prototyping is however quite expensive. The manufacturer has various fixed costs in setting up the PCB for manufacturing which adds up a lot to the final price of the PCB prototypes.

As we still were short on budget, basically no budget at all from the University, I had to find a different solution for making small quantities of professionally made PCBs. The solution arrived when I found Dorkbot PDX, a group of electronics hobbyists located in Portland, Oregon USA. As they were a big number of people making their own PCB designs, they were organizing a monthly group PCB order from a local USA PCB manufacturer. By merging all the various designs of the different PCBs in the order, they where able to produce a big panel (figure 7.1) containing all the various designs. Building a big panel instead of small PCBs singularly considerably lowers the per PCB costs lowering the entry price for professionally built PCBs.

The Dorkbot PDX service has been crucial for having FreeIMU PCBs fabricated at an affordable prices. I'm sure that this kind of services will open many possibilities in developing community driven hardware projects.

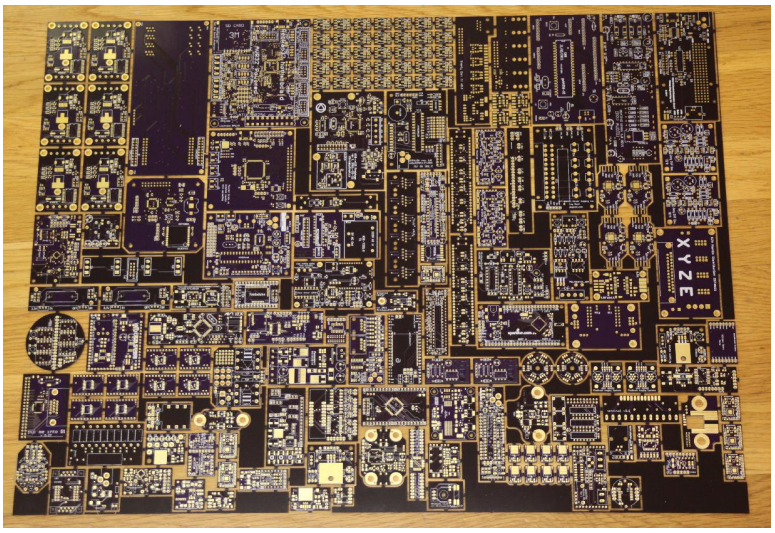

Figure 7.1: A PCB panel from the Dorkbot PDX group order - Various PCB designs have been merged into a big panel. Picture courtesy James Neal.

7.2 FreeIMU version 0.1

FreeIMU version 0.1, the first FreeIMU board developed, uses the ADXL345 accelerometer, the ITG3200 gyroscope and the HMC5843 magnetometer trying to keep the PCB schematics and designs as simple as possible.

The hierarchic schematics in figure 7.2 describe all the connections of FreeIMU version 0.1. The three sensors have been connected in parallel on the I^2C bus and on the power and ground connections. An attentive reader will note how the per chip schematics are very similar to the one used on the various breakout boards. The new elements are the C10 10 μF capacitor, used to stabilize the power to the PCB, and the two solder jumpers J1 and J2, used to set the alternative addresses of the accelerometer and gyroscope.

From the schematics, the actual PCB has been designed, trying to minimize the size of the whole design. As usual, 0805 capacitors and a 0.1 inches connector have been used. A package A tantalum capacitor has been chosen for the C10 10 μF capacitor. Two handy mounting holes have been added to the design. The sensors axis have been aligned to ease data processing and sensor fusion. Soldering has been done with solder paste, a plastic stencil and a pizza oven.

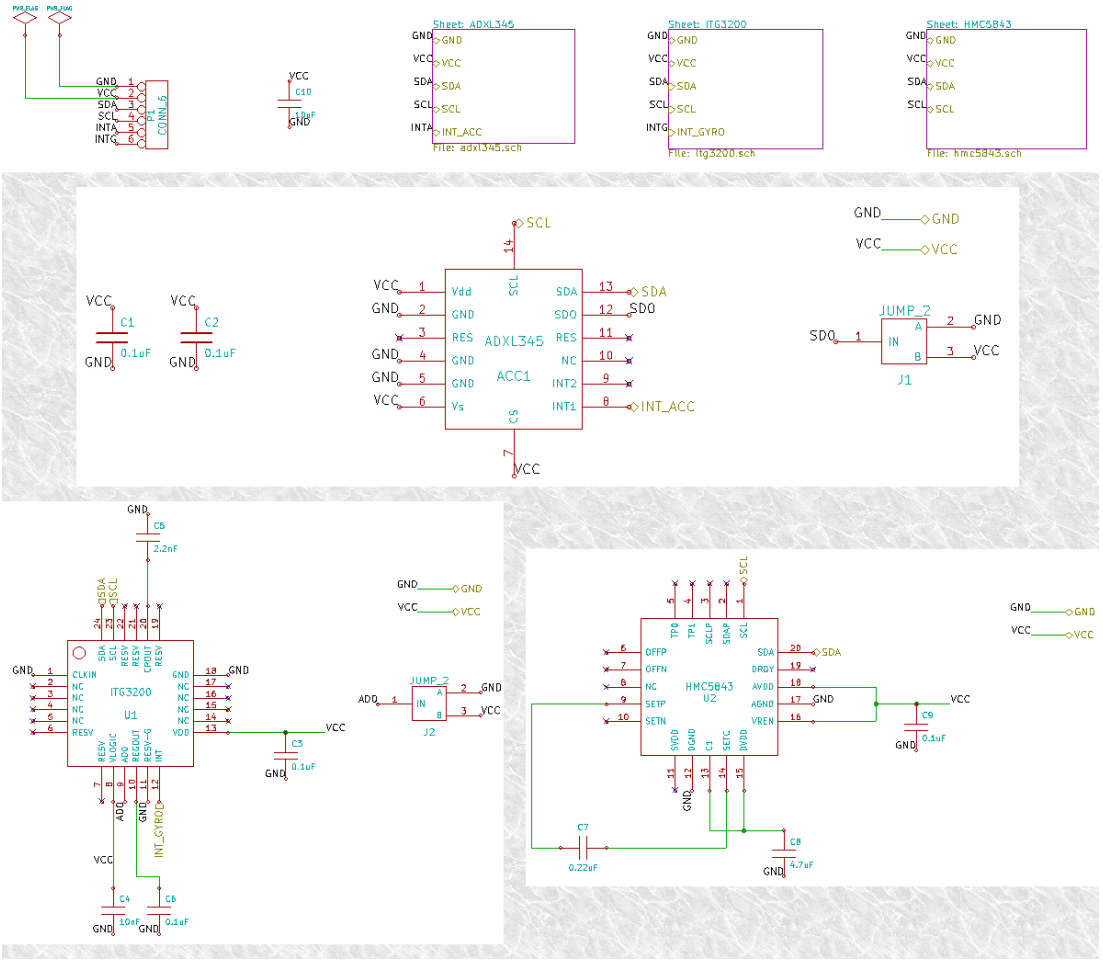

Figure 7.2: FreeIMU v0.1 Schematics - This is the hierarchic schematics FreeIMU v0.1.

The result of the PCB design is a 38.5 x 23 mm PCB which, once assembled with the respective components, weights about 10 grams (figure 7.3). In order to use FreeIMU version 0.1, it has to be connected to a 3.3 V power source and to the I^2C bus: as there are no pullups integrated on FreeIMU v0.1, external pullups have to be added.

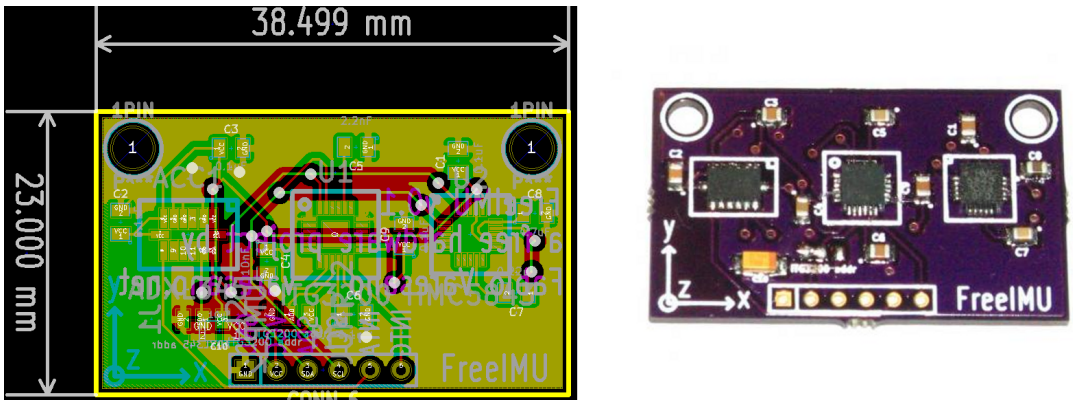

Figure 7.3: FreeIMU v0.1 PCB - KiCad designs and a picture of a built FreeIMU v0.1

7.3 FreeIMU version 0.2

In FreeIMU version 0.1 there isn't any voltage level regulator and no pullup resistors for the I^2C bus. When used on a 5 V microcontroller it has to be connected to a 3.3 V power source and external pullups have to be added.

This can cause some problems because the logic level voltages used by the 3.3 V sensors are 3.3 V too and the sensors don't tolerate higher voltages. This can pose problems if FreeIMU is connected to a 5 V Arduino and internal pullups of the ATMEGA 328p are enabled. This means that the sensors will receive 5 V signals which can cause irreparable damages to them.

The ATMEGA 328p running at 5 V will take any voltage higher than 2.5 V as a logic HIGH, so an I^2C bus pulled up to 3.3 V should work without problems. However, when there is noise on the bus, for example caused by a motor running nearby, the delta between 2.5 V and 3.3 V is too small and can cause communications problems. I

personally experienced occasional hangs of the I2C bus due to noise in the communication line.

This potential issues have motivated me in designing FreeIMU version 0.2 whose schematics are depicted in figure 7.4. A MIC5205 (36), a 3.3 V voltage regulator, has been added with the associated capacitors to provide a stable and regulated power source to the sensors. A logic level converter, the PCA9306 from NXP (52), has also been added to translate 5 V signals coming from the ATMEGA 328p into 3.3 V signals used by the sensors. The addition of the LLC also added the possibility of embedding pullups resistors in the PCB itself which can be enabled or disabled using solder jumpers JP1 and JP2.

With these additions there are now two logic levels and power sources on the board. This make possible to use two different connectors, one running at 3.3 V (P1) and one running at 5 V (P2). With this connectors, it's possible to connect a 5 V Arduino directly into the 5 V connector without any additional pullup resistor or logic level converter. The presence of a 3.3 V connector make it possible to also use FreeIMU with 3.3 V microcontrollers. Another possibility is using it to add other 3.3 V devices to the same I^2C bus.

Figure 7.5 shows the PCB design as well as some pictures of FreeIMU version 0.2. As usual, 0805 capacitors and resistors have been used a part from C11 and C12, the capacitors needed by the voltage regulator, which are tantalum capacitors in package A. The PCA9306 has been chosen in package SO-8 and the MIC5205 in package SOT23-5.

7.4 Making FreeIMU a libre hardware project

During the development of FreeIMU, I made constant posts on my personal website, updating on the status of the project and constantly releasing the various revisions of the schematics and designs of FreeIMU. This generated quite a big interest on FreeIMU as well as some suggestions from people working on the same kind of sensors.

When the designs have been completed, I published them under a creative commons license CC-BY-SA which allows users to study and modify FreeIMU designs freely.

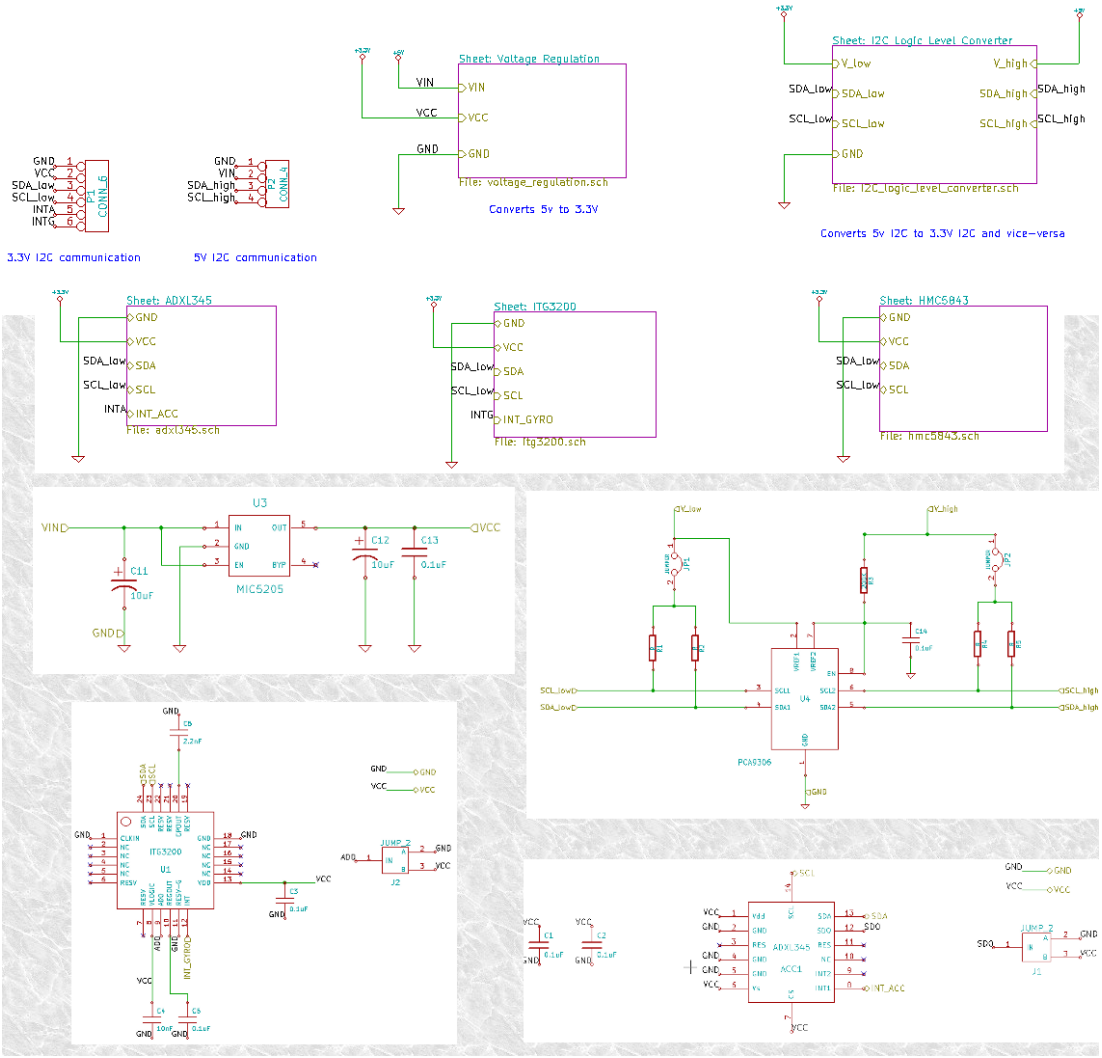

Figure 7.4: FreeIMU v0.2 Schematics - This is the hierarchic schematics FreeIMU v0.2.

Figure 7.5: FreeIMU v0.2 PCB - KiCad designs and top and bottom pictures of a built FreeIMU v0.2

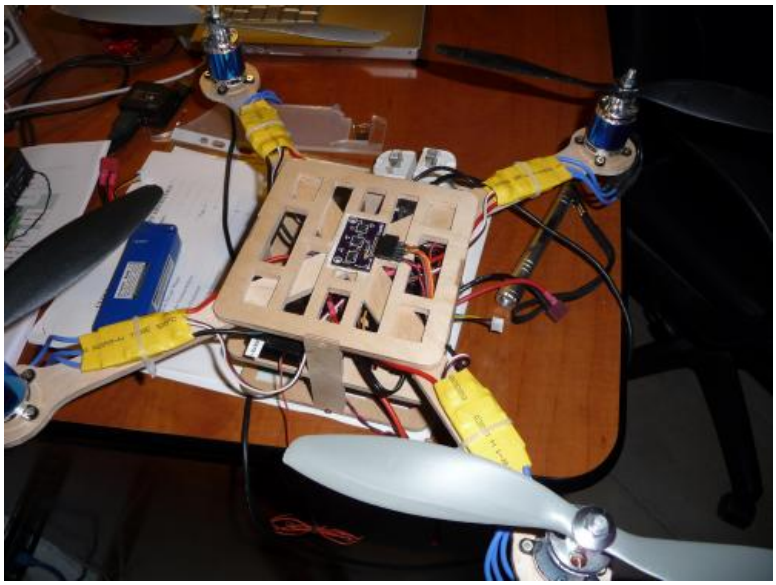

Figure 7.6: FreeIMU v0.1 mounted on a quadcopter - FreeIMU is used as an AHRS for stabilization of the multi rotor.

FreeIMU is currently being used as base for many various projects. I'm aware of it being used for freebie tracking, multicopter stabilization 7.6, human motion tracking as well as human computer device prototyping.

I'm currently aware of two children projects based on FreeIMU. One involves adding an ATMEGA 328p to produce an intelligent 9 DOM board capable of being chained with other peer boards for using in human body movement tracking. Another project involves adding an ATMEGA 328p and a Bluetooth module building a wireless orientation sensing device.

7.5 Competing commercial products

Just a couple of weeks after I published on my personal website the schematics and PCB designs of FreeIMU, Sparkfun Electronics, a Boulder, Colorado based company released a very similar board based on exactly the same sensors used in FreeIMU.

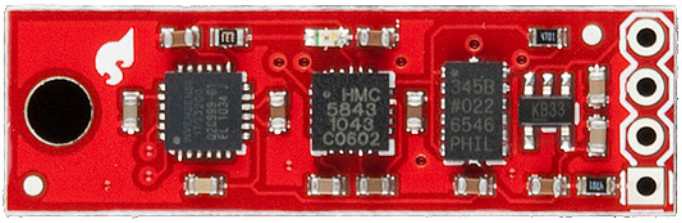

Figure 7.7: 9 Degrees of Freedom - Sensor Stick - The board made by Sparkfun Electronics with the same sensors used in FreeIMU.

The product is called *9 Degrees of Freedom - Sensor Stick* and is available for about 70 euro (without customs and shipping costs). It features the same sensors used in FreeIMU, it has an integrated voltage regulator and pullup resistors and it's slightly smaller than FreeIMU.

This board however presents some design issues. The accelerometer has been placed far from the mounting hole and the PCB is 0.8 mm thick: this can cause problems as when mounted on a case it's probable that the accelerometer will oscillate due to external

vibrations surely decreasing its accuracy. This issue may give very poor results when used in a motorized object such as a quad-rotor.

Another issue could be the complete absence of interrupt pins for the sensors which haven't been broke out to the PCB connectors. This surely limits the possibilities of the board which can't take advantages of the interrupt based features of the ADXL345 accelerometer (eg. single and double tap detection). The absence of interrupt pins also constrains the programmer in using only I^2C polling based sensor reading while an interrupt based approach could be useful especially at higher sampling rates.

The last issue is that the magnetometer axis haven't been aligned to the other two sensors axis. By doing so they have been able to make the board slightly smaller. This misalignment is fixable in software but adds avoidable complexity for the user without a clear advantage.

8

Palla

In the previous chapters I introduced Arduino prototyping, orientation sensing algorithms for magnetic, angular rate and gravity (MARG) sensors and presented FreeIMU, an integrated MARG sensor PCB capable of orientation sensing.

In this chapter, all these topics will be glued together into Palla, a prototype of a spherical tangible user interface capable of orientation sensing, single and double tap detection, user hand proximity measurement, vibration feedback and wireless communication to the PC.

8.1 Previous works

During my bibliographical research among scientific publications, proceedings, books and websites I wasn't able to find any previous work on MARG sensors powered orientation sensing capable tangible user interfaces.

Traditionally, orientation sensing in tangible prototypes has been mostly powered by some kind of computer vision techniques, eg (14, 17, 26, 42). The usage of MARG sensors, probably due to the relative recent introduction of such sensors, seems to be quite limited or non existent.

There are however tangible user interface prototypes based on accelerometers only orientation sensing. The most notable works are the Tangerine SMCube (4, 5), which

also have vibration feedback for the user, the Display Cube (25) which uses the rotation data to visualize information to the user using an embedded LCD, and Gizmo (16) a tangible cube capable of orientation sensing for browsing architectural designs. Perhaps the most similar work is the Cubic Mouse (17), a 6 DOF sensing capable cubical tangible user interface developed for visualizing car designs.

The lack of MARG sensors approaches in orientation sensing for tangible user interfaces makes Palla a relevant contribution in the field.

8.2 Palla's schematics

Palla's schematics are reported in figure 8.1. Palla contains a regular Arduino Duemilanove (U2) which is powered by a 9 Volts battery (BT1) activated by a switch (SW1). Arduino's internal voltage regulator will provide a 5 Volts source on its 5V pin.

A FreeIMU version 0.2 is connected using its 5 Volts connector to Arduino 5V and GND as well as on the I^2C bus on pin A4 and A5. Voltage level conversion from Arduino's 5 Volts signals and FreeIMU 3.3 Volts internal signals will happen in the PCA9306 (52) logic level converter embedded into FreeIMU v0.2.

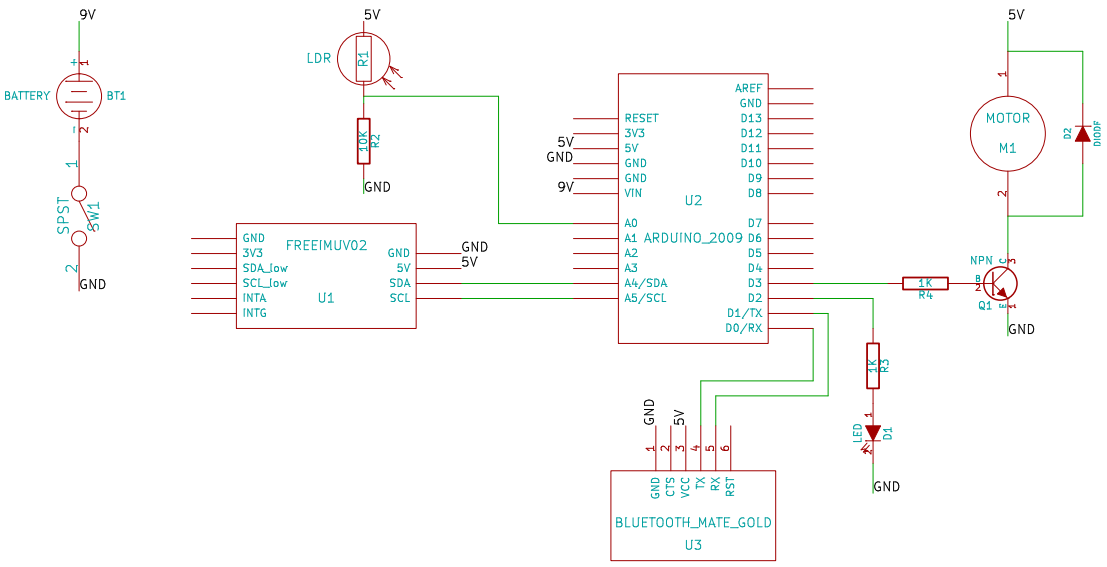

Figure 8.1: Palla's schematics

A Bluetooth Mate Gold from Sparkfun Electronics (11) is used as Bluetooth module which enable wireless serial communication to a Bluetooth capable computer. The Bluetooth Mate Gold already has its own circuitry for voltage regulation and logic level conversion so it can be simply connected to a 5 Volts source and the TX and RX pins on the Arduino.

A light dependent resistor (LDR - R1) has been connected in series with a 10K Ω resistor (R2) implementing a voltage divider circuit (as seen in 4.3.1 and 4.3.3.2) which varies its output voltage depending on the amount of light detected by the LDR. The variable voltage between R1 and R2 is available for reading on the Arduino by connecting A0 between the two resistors.

A simple DC brushed motor (M1) is used as actuator for the vibration feedback. The motor is activated from Arduino by using a transistor (Q1) through a 1K Ω resistor (R4). The motor is connected to 5 Volts and a diode (D2) which serves as protection against Back Electromagnetic Flux (BEMF) voltage harming the transistor (8).

Finally an LED (D1) has been connected to Arduino through a 1K Ω resistor (R3). This LED can be used as status or feedback indicator.

8.3 Building Palla

In order to simplify the building of the circuit described above, a perfboard has been used to create a very simple Arduino Shield (3.2.1). The shield uses 0.1 inches spaced female connectors which have long legs which can permits stacking the shield above the Arduino. The various wires have been soldered to the perfboard and the shield connectors (figure 8.2 A). Regular 1/4 W resistors have been used for the various resistors. For the LDR a VT90N2 has been used while the transistor is a BC547 in TO92 package. FreeIMU v0.2 and the Bluetooth module are connected into female connectors (figure 8.2 B).

The practical construction of Palla consisted in using a 10 cm diameter rigid plastic ball. A 9 Volts square battery and a on/off switch have been fixed on the bottom of one of the semi-spheres (figure 8.2 C). On the other semi-sphere a pager motor has

been placed. On the motor axle, a non balanced weight has been mounted: this weight produces a vibration when the motor rotates (figure 8.2 D).

On top of the battery an Arduino Duemilanove has been fixed. Above of it the shield can be mounted and then connected to the motors and the other components (figure 8.2 E). When the two semi-spheres are closed together, Palla is complete (figure 8.2 F).

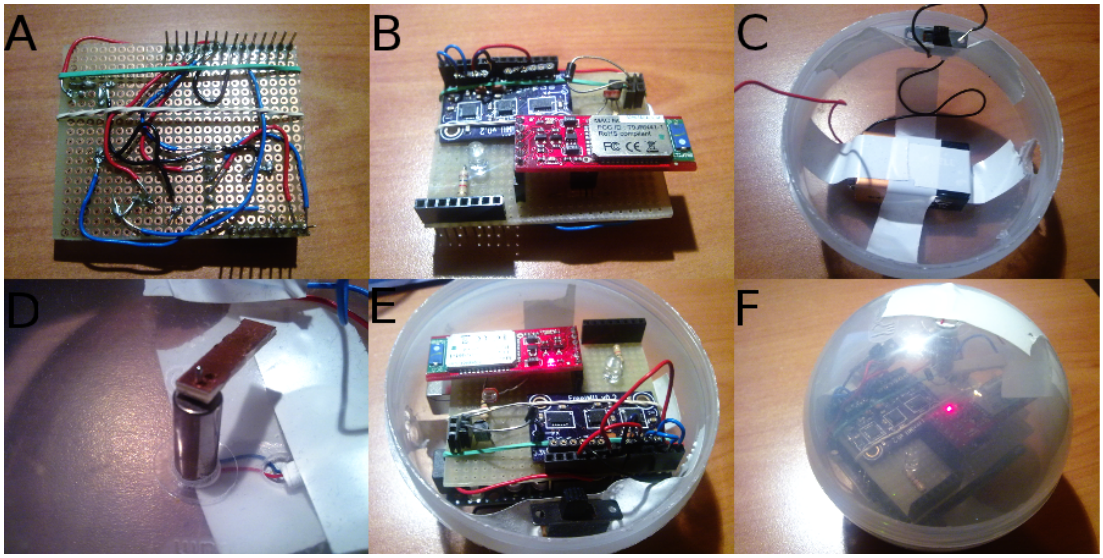

Figure 8.2: Palla prototype - A: bottom of the Arduino shield developed with all the Palla connections. B: Top of the Arduino shield: FreeIMU, the Bluetooth Mate, the diode, the LDR and the LED are visible. C: bottom semi-sphere with a 9 Volts PP3 battery and the switch. D: vibration motor with unbalanced weight. E: Arduino and the shield mounted inside of Palla. F: final prototype.

8.4 Palla capabilities and possible usages

Palla, by fusing the FreeIMU sensors outputs and fusing them using the algorithm presented in section 6.4 can sense its orientation precisely in the space. By using the interrupt based features of the ADXL345 accelerometer included in FreeIMU, Palla can also sense per axis single and double taps.

Palla's LDR will variate it's resistance when covered by shadows in a luminous environment and thus can be used to sense the proximity of the user hand. The motor and

the status LED enable physical and visual feedback for the user.

Palla can then be used in various kind of tangible user interface applications. Palla can be manipulated directly by holding it or it can be rotated or rolled on a plane surface. As it is completely wireless and self contained it can be used by multiple users in collaborative applications.

Its orientation sensing capabilities are particularly suited for using in three dimensional user interfaces. In figure 8.3, Palla is being used to control player view in a first person shooter game. Palla is particularly suited as a tangible user interface for browsing three dimensionally structured data. Examples of such data could be chemical molecules, DNA structures, geographical maps, 3D CAD designs, etc.

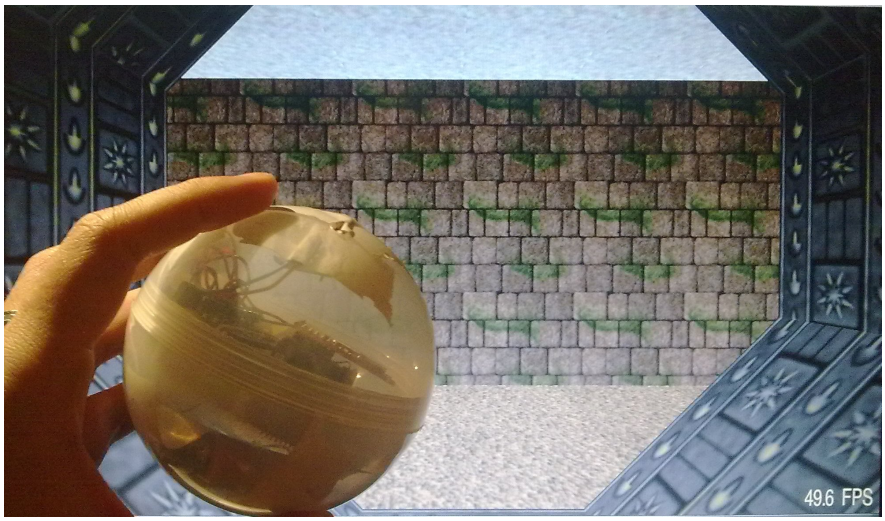

Figure 8.3: Palla in 3D environments - Palla used as controller in a FPS game.

9

Femtoduino

In chapter 7 I presented FreeIMU, an integrated solution for a 9 degrees of measurement MARG sensor array. The main reason for developing FreeIMU instead of using the already developed breakout boards was the unpractical size and complexity of the prototype. Using the three different breakout boards in the Palla prototype would have been quite impossible.

In fact, the prototype size is a limitation in many projects: there are many applications in which a prototype as big as an Arduino Duemilanove is simply too big and it's not a viable solution. There are however smaller Arduino compatible boards, such as the Arduino Nano or the Arduino Pro Mini. These boards however are still too big for many applications.

In this chapter, I present Femtoduino, a very tiny Arduino compatible board especially designed for ultra-small prototyping. Femtoduino is only 20.70 x 15.24 millimeters in size for only 2 grams of weight, making it the smaller an lighter Arduino board currently available while it can deliver exactly the same computing power of the Arduino Duemilanove or UNO.

9.1 Schematics

Femtoduino schematics have been based upon the Arduino Nano, Pro Mini and UNO schematics. As with Femtoduino the goal was to extremely reduce the size of the board,

a very minimal approach has been followed when adding components to its electronic design.

Fetmduino uses the QFN 32 version of the ATMEGA 328p, exactly the same microcontroller used in the Arduino Duemilanove or UNO but it comes in a very small 5 x 5 x 1 mm 32 pins package. The microcontroller is marked as component IC1 on the schematics. Peculiar of such package is the presence of a big conductive pad below it which has to be connected to ground: the pad has been marked as 33 in the microcontroller design.

The microcontroller has been connected to a resonator (Q1) which provides a 16 or 8 MHz square wave signal which serves as clock. When a 8 MHz clock is used, the microcontroller can be powered by 3.3 V. Instead, when used with an 16 MHz clock , the microcontroller has to be connected to a 5 V power source.

The various input/outputs of the microcontroller have been broke into the P1, P2 and J2 connectors. 0.1 μF decoupling capacitors have been added to the AREF pin and to the power connections. Following Arduino convention top have an LED on board connected to digital 13, an LED has been connected in series with a resistor to the SCK pin.

The reset switch group, implements a simple pullup for the reset pin of the microcontroller: by closing the SW1 switch pulling down the reset pin it's possible to reset the microcontroller. The same result can be achieved by bringing the DTR connector to HIGH.

The voltage regulation follows the same design used in FreeIMU v0.2. The MIC5205 voltage regulator and three capacitors (C1, C2 and C3) deliver a stable power source to the microcontroller. Depending on the needed clock frequency (8 or 16 MHz) a 3.3 or 5 V marked MIC5205 regulator has to be used. The voltage regulator can provide a current up to 150mA. In parallel to the voltage regulator output, a LED and its associated resistor has been connected (R1 and D1): this LED serves as power indicator.

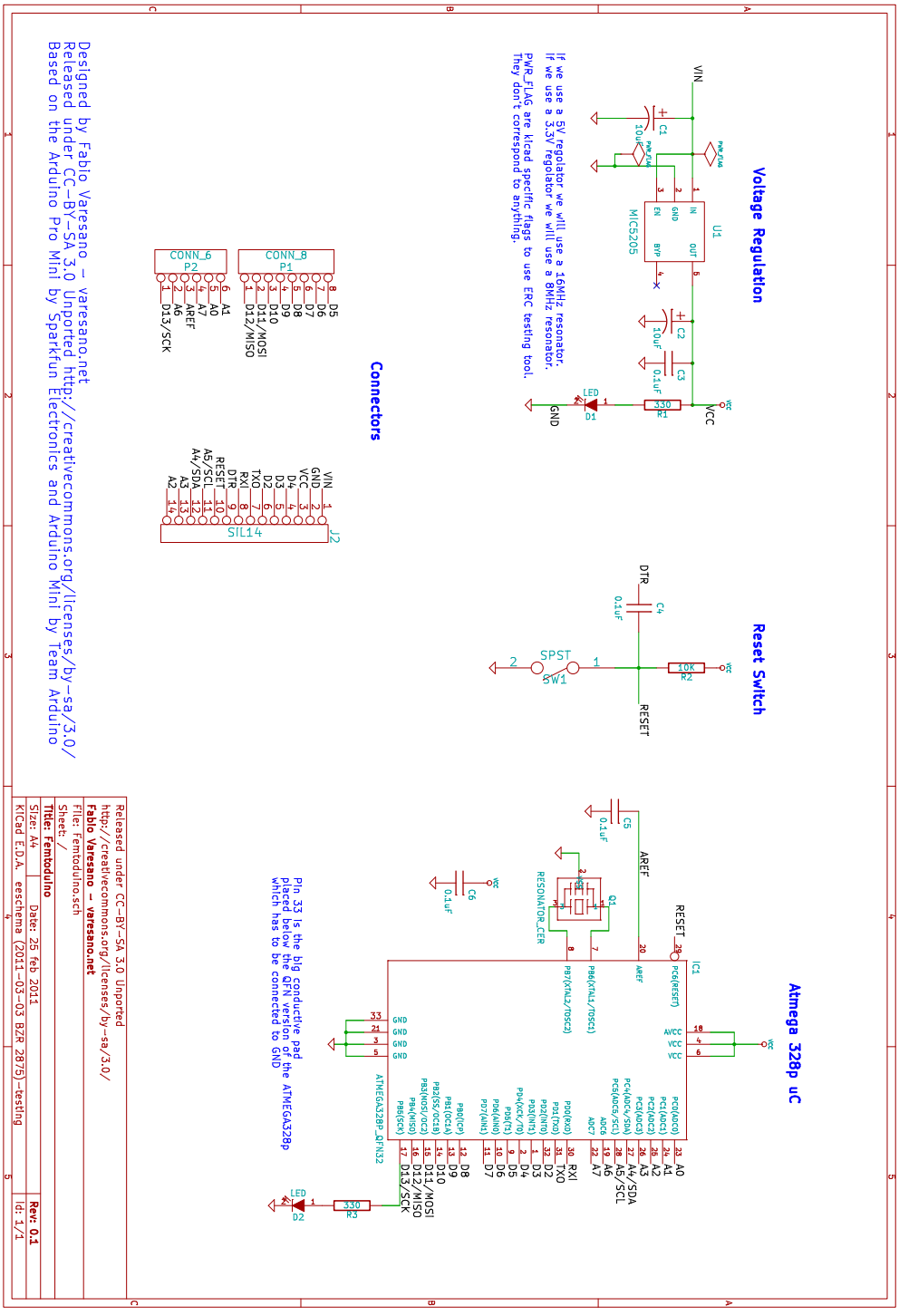

Figure 9.1: Femtoduino Schematics

9.2 PCB desing

The main goal of Femtoduino is a very small size: any decision during the design of the PCB has been towards keeping its size as small as possible.

As already said, the microcontroller used is the ATMEGA 328p in the QFN32 package which is currently the smallest microcontroller of that microcontroller series available. As the package is only 5 x 5 millimeters this dramatically reduce the whole size of the PCB.

On standard sized PCBs for prototyping, the connectors are usually 0.1 inches spaced. In Femtoduino instead, 0.05 inches spaced connectors have been used, so that the space occupied by them is an half of that of a 0.1 inches connector. It's important to note how the connectors have been arranged to minimize the number of vias in the PCB.

Passive components have been chosen with 0402 package, which is the smallest package which could be hand assembled without using industrial procedures. LEDs are instead 0805 packaged, so that they are clearly visible by the user. Capacitors C1 and C2, as usual, are tantalum capacitors in package A.

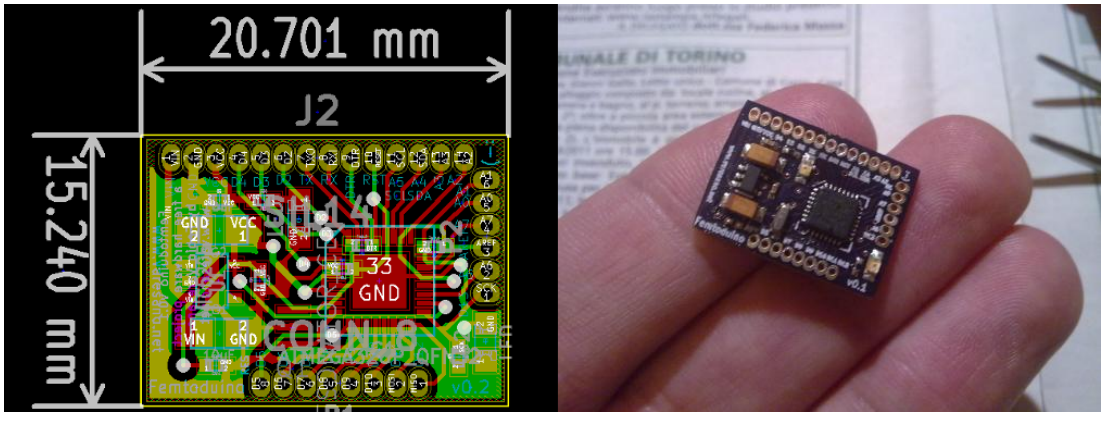

Figure 9.2: Femtoduino PCB design and picture

9.3 A libre hardware: media coverage and commercial productions

I published the designs of Femtoduino on my personal website under a libre license (CC-BY-SA), including KiCAD schematics, PCB designs, Gerbers and the bill of materials for building it. This is everything needed for building a complete and functioning Femtoduino. This generated quite some interest on the project as the need for really small prototyping seems to be considerable.

Femtoduino has been featured on many technical websites, most notably on the official Arduino blog and on Hackaday, a globally known website for electronics hobbyists. Femtoduino is currently one of the most popular board in the Dorkbot PDX group PCB order.

Femtoduino also generated a lot of interest as a commercial project and there are now at least three companies in process of building mass produced Femtoduinos.

The interest generated by Femtoduino should be a good indicator of it's quality in the design and project idea.

10

Conclusions

During this thesis I experienced with electronics, Arduino, MEMS sensors and orientation sensing algorithms to produce a prototype of a tangible user interface called Palla.

Thanks to the work done in this thesis, I learned many new things for which I never received an education. With a computer science background, I never received education on electronics and printed circuit boards design but, as proven by the projects created, the knowledge gained on such topics is quite relevant.

Arduino, its programming APIs and the practical implementation issues are now well known for me. Currently, I'm also intimate with MEMS accelerometers, gyroscopes and magnetometers and I've been able to design PCB as well as implement practical applications for them.

I discovered some of the theoretical and practical issues of designing a tangible user interface prototype with focus on orientation sensing.

10.1 Future Works

This thesis open the way to many possible future developments mainly related to improvement in the orientation sensing approach and practical usages of FreeIMU and Femtoduino.

10.1.1 Orientation Sensing

Regarding the orientation sensing algorithms presented there is still much to be done on the topic of calibration of the sensors. As the precision needed by my prototypes didn't had to be very accurate, a slightly lazy approach on calibration has been implemented: basically the calibration is done simply using the internal self test features of the sensors.

However, for a more accurate orientation estimation the sensors should be calibrated using a more detailed approach. Good examples on calibration procedures for the devices could be (30, 38, 62). (38) is extremely of interest as, behind the calibration suggestions, there are also pointers on how to compensate for sensors misalignment.

The market of MEMS sensors is extremely active. As those kind of sensors are being added to handheld devices, possibly generating sells in huge numbers, there is a constant push on competition between the various producers which makes research and improvement on the sensors extensive. There are already sensors which looks more powerful than those used on FreeIMU (eg: MPU6050, LSM303DLH). Further works and developments should evaluate the various new sensors on the market.

10.1.2 FreeIMU

FreeIMU has proven to be a great prototype tool for orientation sensing devices. However, there are many possible usages of FreeIMU, both in the commercial and research fields. Personally, I'm very interested in human tracking and I'd like to continue the work done with FreeIMU to implement some kind of full body immersion into virtual reality. There are research projects already working on these topics (eg (50)) but I'm extremely interested on working on something like this.

10.1.3 Palla and Femtoduino

In this thesis I developed a prototype of Palla which works and looks interesting in many different applications. However, there hasn't been any serious effort into developing a complete application based on Palla. Such development would be surely great for testing the quality of user experience when using Palla. Contacts with Telecom Italia

Lab, research division of Telecom Italia, proposed the usage of a controller similar to Palla into a three dimensional user interface for digital television. This is surely a field which could greatly make use of Palla.

The design of Femtoduino has been designed for size constrained applications. However, even if successfully tested Femtoduino and used it on a couple of draft prototypes, a practical usage of Femtoduino still has to be done. This little Arduino compatible board opens many very interesting research possibilities. I think that it can be extremely useful in many research fields, for example ubiquitous computing, robotics, HCI, etc. I'm sure that if the project gains enough fame it will be slowly enter into many research labs as Arduino already did.

10.2 Acknowledgments

I'd like to thank my supervisor, Prof. Luca Console, for his guidance during the various steps of this thesis and for giving me the opportunity to work with him. I also would like to thank Prof. Marco Grangetto, for his enlightening review on this thesis and the whole project.

I also would like to thank Fabiana Vernero and Rossana Simeoni for following my work closely and for their smart suggestions. Of course, I'd like to thank the Univesità degli Studi di Torino for providing me some of the tools widely used in this thesis.

I'd like to thank the Arduino community for helping me in my first steps with this thesis, and the Dorkbot PDX community, especially James Neal, for their help in the printed circuit boards design and production. I also would like to thank Sebastian O.H. Madgwick for his wonderful orientation sensing algorithm and for the personal help I received from him on a couple of problems encountered.

Finally, I'd like to thank my family for staying close to me even in the most difficult days and for always been of encouragement to me. I thank my girlfriend Arianna for her encouragement and support and for her great pasta amatriciana.

I'd like to thank my grandparents Pierino and Riccardo for teaching me how to be a good person.

References

[1] Grant Baldwin, Robert Mahony, Jochen Trumpf, Tarek Hamel, and Thibault Cheviron. **Complementary filter design on the Special Euclidean group SE(3)**. 107

[2] Massimo Banzi. *Getting Started with Arduino*. O'Reilly, 2009. 13, 17

[3] Tamara Bratland, Michael J. Caruso, Robert W. Schneider, and Carl H. Smith. **A New Perspective on Magnetic Field Sensing**. Available from: http://www.sensorsmag.com/sensors/electric-magnetic/a-new-perspective-magnetic-field-sensing-855. 77

[4] Omar Cafini, Elisabetta Farella, Luca Benini, Stefano Baraldi, Nicola Torpei, Lea Landucci, and Alberto Del Bimbo. **Tangerine SMCube: a smart device for human computer interaction**. In *Proc. of IEEE European Conference on Smart Sensing and Context*. IEEE Computer Society, 2008. Available from: http://www.micc.unifi.it/publications/2008/CFBBTLD08. 121

[5] Omar Cafini, Piero Zappi, Elisabetta Farella, Luca Benini, Stefano Baraldi, Nicola Torpei, Lea Landucci, and Alberto Del Bimbo. **Evolving TUIs with Smart Objects for Multi-context Interaction**. In *Proc. of International Conference on Computer-Human Interaction (CHI)*, Florence, Italy, April 2008. ACM, ACM Press. Available from: http://www.micc.unifi.it/publications/2008/CZFBBTLD08. 121

[6] Michael J. Caruso, Tamara Bratland, Dr. Carl H. Smith, and Robert Schneider. **A New Perspective on Magnetic Field Sensing**. Available from: http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/Magnetic__Literature_Technical_Article-documents/A_New_Perspective_on_Magnetic_Field_Sensing.pdf. 75

[7] M.J. Caruso. **Applications of magnetic sensors for low cost compass systems**. In *Position Location and Navigation Symposium, IEEE*, 2000. Available from: http://hnc.ru/lib/a%26c%20%28automatic%20%26%20controls%29/sensors/DataSheet/Magnit/Honeywell/lowcost.pdf. 106

[8] Reston Condit. **AN905: Brushed DC Motor Fundamentals**. Available from: http://ww1.microchip.com/downloads/en/AppNotes/00905a.pdf. 123

[9] Analog Devices. **ADXL330 Accelerometer Datasheet**. Available from: http://www.analog.com/static/imported-files/data_sheets/ADXL330.pdf. 77, 78

[10] Analog Devices. **ADXL345 datasheet**. Available from: http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf. 93

[11] Sparkfun Electronics. **Bluetooth Mate Gold**. Available from: http://www.sparkfun.com/products/9358. 123

[12] Sparkfun Electronics. **Triple Axis Accelerometer Breakout - ADXL330**. Available from: http://www.sparkfun.com/products/692. 78, 79

[13] Mark Euston, Paul Coote, Robert Mahony, Jonghyuk Kim, and Tarek Hamel. **A Complementary Filter for Attitude Estimation of a Fixed-Wing UAV**. 107

[14] Mark Fiala. **The SQUASH 1000 Tangible User Interface System**. *Mixed and Augmented Reality, IEEE / ACM International Symposium on*, **0**:180–181, 2005. 121

[15] Christopher J. Fisher. **AN1057: Using an Accelerometer for Inclination Sensing**. Available from: http://www.analog.com/static/imported-files/application_notes/AN-1057.pdf. 100

[16] Randolph Fritz, Chih-Pin Hsiao, and Brian R. Johnson. **Gizmo and WiiView: Tangible User Interfaces Enabling Architectural Presentations**. Available from: http://dmg.caup.washington.edu/pdfs/ACADIA09.Giz.Wii.pdf. 122

[17] Bernd Fröhlich and John Plate. **The cubic mouse: a new device for three-dimensional input**. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '00, pages 526–531, New York, NY, USA, 2000. ACM. Available from: http://doi.acm.org/10.1145/332040.332491. 121, 122

[18] CadSoft Computer GmbH. **EAGLE PCB design tool homepage**. Available from: http://www.cadsoftusa.com/index.htm. 87

[19] Dr. Andrew Greensted. **Switch Debouncing**. Available from: http://www.labbookpages.co.uk/electronics/debounce.html. 41

[20] H. Hauser, G. Stangl, W. Fallmann, R. Chabicovsky, and K. Riedling. **Magnetoresistive Sensors**. Available from: http://www.iemw.tuwien.ac.at/publication/workshop0600/Hauser.html. 77

[21] Honeywell. **HMC5843 datasheet**. Available from: http://www.sparkfun.com/datasheets/Sensors/Magneto/HMC5843.pdf. 96

[22] Honeywell. **Magnetic Sensors Overview**. Available from: http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/Magnetic__Literature_Application_notes-documents/Magnetic_Sensor_Overview.pdf. 76, 77

[23] Invensense. **ITG-3200 - Product Specification Revision 1.4 (datasheet)**. Available from: http://invensense.com/mems/gyro/documents/PS-ITG-3200-00-01.4.pdf. 84, 85, 93, 94, 95

[24] Jean-Marc Irazabal and Steve Blozis. **AN10216: I2C Manual**. *Philips Semiconductors*. Available from: http://www.nxp.com/documents/application_note/AN10216.pdf. 84

REFERENCES

[25] Matthias Kranz, Dominik Schmidt, Paul Holleis, and Albrecht Schmidt. **A Display Cube as Tangible User Interface**. In *Adjunct Proceedings of UbiComp 2005 (Poster and Demo)*, September 2005. 122

[26] Adam Kumpf. *Trackmate: Large-Scale Accessibility of Tangible User Interfaces*. Massachusetts Institute of Technology, 2009. 121

[27] Tony R. Kuphaldt. *Lessons In Electric Circuits, volume I - DC*. 2006. 5, 45

[28] OpenLearn LearningSpace. **A problem with sensors**. Available from: http://openlearn.open.ac.uk/mod/oucontent/view.php?id=397841&section=6. 72

[29] MakingThings LLC. **Introduction to Electronics**. Available from: http://www.makingthings.com/teleo/products/documentation/teleo_user_guide/electronics.html. 6

[30] Sebastian O.H. Madgwick. **Automated calibration of an accelerometers, magnetometers and gyroscopes - A feasibility study**. Available from: http://www.x-io.co.uk/res/doc/automated_calibration_of_an_accelerometers_magnetometers_and_gyroscopes_a_feasibility_study.pdf. 134

[31] Sebastian O.H. Madgwick. **An efficient orientation filter for inertial and inertial/magnetic sensor arrays**. Available from: http://www.x-io.co.uk/res/doc/an_efficient_orientation_filter_for_inertial_and_inertialmagnetic_sensor_arrays.pdf. 107, 108, 110

[32] Sebastian O.H. Madgwick. **Quaternions**. Available from: http://www.x-io.co.uk/res/doc/quaternions.pdf. 107, 108

[33] Robert Mahony, Sung-Han Cha, and Tarek Hamel. **A coupled estimation and control analysis for attitude stabilisation of mini aerial vehicles.** 107

[34] Robert Mahony, Tarek Hamel, and Jean-Michel Pflimin. **Nonlinear Complementary Filters on the Special Orthogonal Group**. In *IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 53, NO. 5, JUNE 2008*. 107

[35] Howard Mason. **Basic Introduction to the use of Magnetoresistive Sensors**. Available from: http://www.diodes.com/_files/products_appnote_pdfs/zetex/an37.pdf. 77

[36] Inc. Micrel. **MIC5205 Datasheet**. Available from: http://www.micrel.com/_PDF/mic5205.pdf. 116

[37] ST Microelectronics. **AN3182: Tilt measuring using a low-g 3-axis accelerometer**. Available from: http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/APPLICATION_NOTE/CD00268887.pdf. 82, 99, 101

[38] ST Microelectronics. **AN3192: Using LSM303DLH for a tilt compensated electronic compass**. Available from: http://www.st.com/stonline/products/literature/an/17353.pdf. 134

[39] Hack N Mod. **How to: Reflow Surface Mount (SMD) Soldering Tutorial**. Available from: http://hacknmod.com/hack/diy-reflow-surface-mount-soldering-smd-tutorial/. 90

[40] J. Neamu, W. Kappel, V. Alecu, and A. Patroi. **Design And Fabrication Of Anisotropic Magnetoresistive Microsensor On Oxidized Silicon Wafer**. *Journal of Optoelectronics and Advanced Materials Vol. 6, No. 3, September 2004, p. 983 - 986*. Available from: http://www.inoe.ro/JOAM/pdf6_3/Neamtu.pdf. 77

[41] Rob O'Reilly, Kieran Harney, and Alex Khenkin. **Sonic Nirvana: MEMS Accelerometers as Acoustic Pickups in Musical Instruments**. Available from: http://goo.gl/I5zHU. 79

[42] J. Patten, H. Ishii, J. Hines, and G. Pangaro. **Sensetable: A Wireless Object Tracking Platform for Tangible User Interfaces**. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 253–260, New York, NY, USA, 2001. ACM. Available from: http://dx.doi.org/10.1145/365024.365112. 121

[43] Edward Pervin and Jon A. Webb. **Quaternions in Computer Vision and Robotics**. 107

[44] William Premerlani and Paul Bizard. **Direction Cosine Matrix IMU: Theory**. 107

[45] Processing. **Processing programming language website**. Available from: http://processing.org/. 24

[46] Arduino API Reference. **analogWrite**. Available from: http://arduino.cc/en/Reference/AnalogWrite. 20, 24

[47] Arduino API Reference. **pinMode**. Available from: http://arduino.cc/en/Reference/pinMode. 38

[48] Arduino API Reference. **pinMode**. Available from: http://arduino.cc/en/Reference/Serial. 57

[49] Arduino API Reference. **Wire Library**. Available from: http://www.arduino.cc/en/Reference/Wire. 85

[50] Daniel Roetenberg. *Inertial and Magnetic Sensing of Human Motion*. Universiteit Twente, 2006. 69, 134

[51] David Sachs. **Sensor Fusion on Android Devices: A Revolution in Motion Processing**. Available from: http://www.youtube.com/watch?v=C7JQ7Rpwn2k.

[52] NXP Semiconductors. **PCA9306 Datasheet**. Available from: http://www.nxp.com/documents/data_sheet/PCA9306.pdf. 116, 122

[53] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. John Wiley & Sons, Inc., 2006.

[54] Richard Stallman. **Why Open Source misses the point of Free Software**. Available from: http://www.gnu.org/philosophy/open-source-misses-the-point.html. 13

[55] Starlino. **Arduino code for simplified Kalman filter. Using a 5DOF IMU (accelerometer and gyroscope combo)**. Available from: http://www.starlino.com/imu_kalman_arduino.html. 102

[56] Starlino. **DIY Surface Mount on a Budget – Complete Walkthrough from PCB etching to Reflow**. Available from: http://www.starlino.com/surface_mount_reflow.html. 89, 90

[57] Starlino. **A Guide To using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications**. Available from: http://www.starlino.com/imu_guide.html. 102, 103

[58] GNU Operating System. **The Free Software Definition**. Available from: http://www.gnu.org/philosophy/free-sw.html. 13

[59] Hani Tawfik. **A Glimpse at MEMS**. Available from: http://knol.google.com/k/a-glimpse-at-mems. 74, 75

[60] KiCad Team. **KiCad PCB design tool homepage**. Available from: http://kicad.sourceforge.net/wiki/Main_Page. 87, 88

[61] ThinkQuest Team. **Electronics: An online guide for beginners**. Available from: http://library.thinkquest.org/16497/intro/index.html. 6

[62] J. F. Vasconcelos, G. Elkaim, C. Silvestre, P. Oliveira, and B. Cardeira. **A Geometric Approach to Strapdown Magnetometer Calibration in Sensor Frame**. 2008. 134

[63] Salvatore A. Vittorio. **MicroElectroMechanical Systems (MEMS)**. Available from: http://www.csa.com/discoveryguides/mems/overview.php. 67

[64] Wikipedia. **Accelerometer**. Available from: http://en.wikipedia.org/wiki/Accelerometer. 68

[65] Wikipedia. **atan2**. Available from: http://en.wikipedia.org/wiki/Atan2. 104

[66] Wikipedia. **Capacitor**. Available from: http://en.wikipedia.org/wiki/Capacitor. 7

[67] Wikipedia. **Gyroscope**. Available from: http://en.wikipedia.org/wiki/Gyroscope. 73

[68] Wikipedia. **I2C**. Available from: http://en.wikipedia.org/wiki/I%C2%B2C. 83

[69] Wikipedia. **Kirchhoff's circuit laws**. Available from: http://en.wikipedia.org/wiki/Kirchhoff%27s_circuit_laws. 9, 10

[70] Wikipedia. **Light-emitting diode**. Available from: http://en.wikipedia.org/wiki/Light-emitting_diode. 28

[71] Wikipedia. **Magic number - programming**. Available from: http://en.wikipedia.org/wiki/Magic_number_(programming). 32

[72] Wikipedia. **Magnetometer**. Available from: http://en.wikipedia.org/wiki/Magnetometer. 75

[73] Wikipedia. **Magnetoresistance**. Available from: http://en.wikipedia.org/wiki/Magnetoresistance. 75

[74] Wikipedia. **Ohm's law**. Available from: http://en.wikipedia.org/wiki/Ohm%27s_law. 7

[75] Wikipedia. **Proper acceleration**. Available from: http://en.wikipedia.org/wiki/Proper_acceleration. 68

[76] Wikipedia. **Series and parallel circuits**. Available from: http://en.wikipedia.org/wiki/Series_and_parallel_circuits. 10, 12

[77] Prof. Fabian Winkler. **Envision Art 01: the responsive screen**. 2007. Available from: http://web.ics.purdue.edu/~fwinkler/590E/index.html. 14