Filippo Castelli
filippocastelli

Carlo Emilio Montanari
carlidel

https://github.com/filippocastelli/uolli

Filippo Maria Castelli, MSc
PhD Student @ LENS
European Laboratory for NonLinear Spectroscopy
castelli@lens.unifi.it

Carlo Emilio Montanari, MSc
PhD Student @ UNIBO
carlo.montanari5@unibo.it

Uolli

A tale of software bugs and fried hardware

Bologna, 06/12/2019

About us

- Filippo Maria Castelli, MSc
  - PHD Student @ LENS (European Laboratory for NonLinear Spectroscopy – Sesto Fiorentino, Italy)
  - Master's in Applied Physics @ UNIBO
  - Bachelor's in Physics @ UNIPI

- Carlo Emilio Montanari, MSc
  - PhD Student @ UNIBO
  - Master's in Applied Physics @ UNIBO
  - Bachelor's in Physics @UNIBO

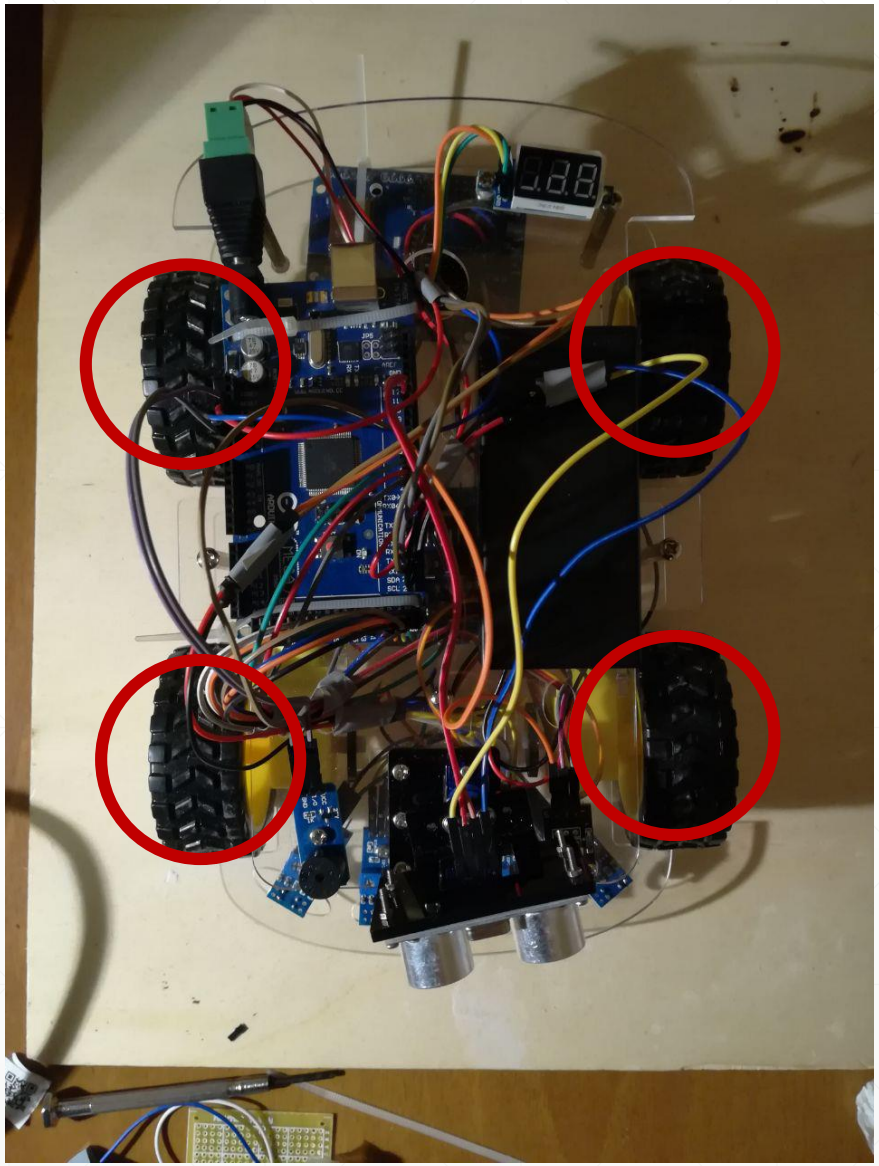
CM-01 Automatic Car

What we wanted it to be:

- Something that actually moves around without catching fire
  - (it actually catched fire at some point)
- Something that can be controlled remotely
- Something that somehow perceives its proximities

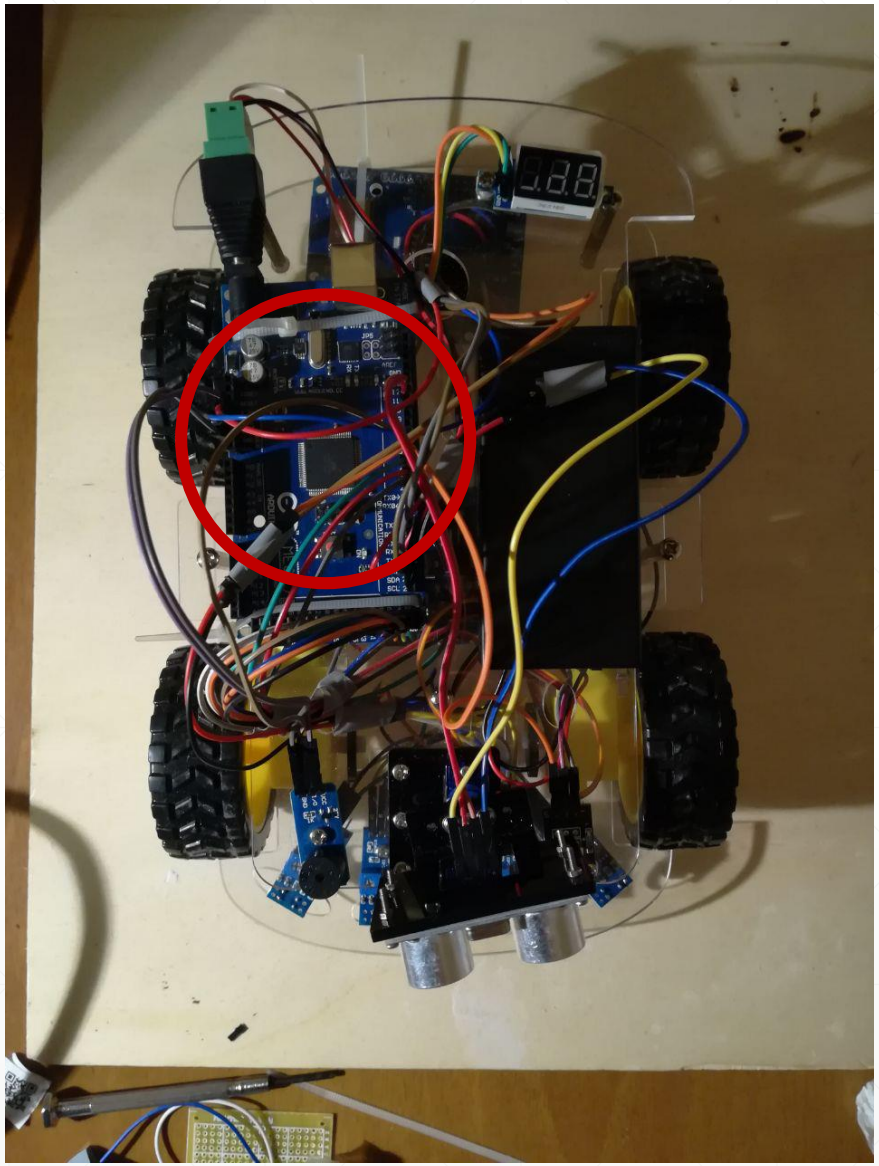
What we achieved:

- Mobility: 4 wheels, controlled in pairs
- Remote Control:
  - Wifi: using ESP8266
  - Infrared remote
- Proximity sensing:
  - HCSR04 ultrasonic range sensor + servo for scanning at 180°
  - IR range sensor for line following



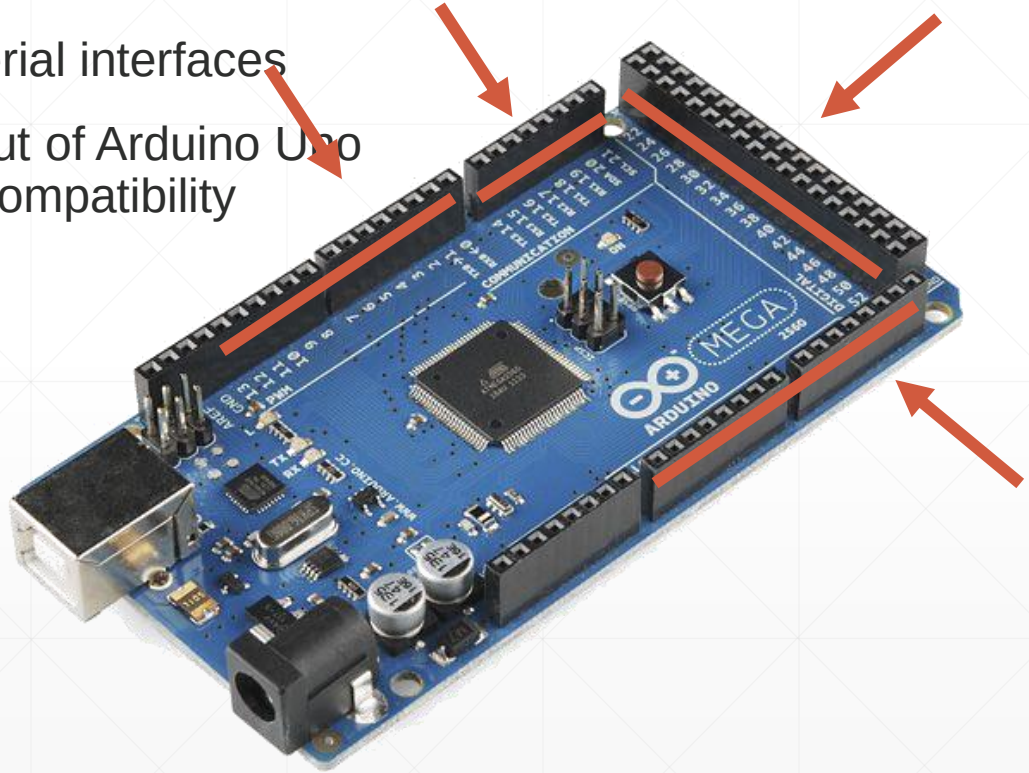
Components:

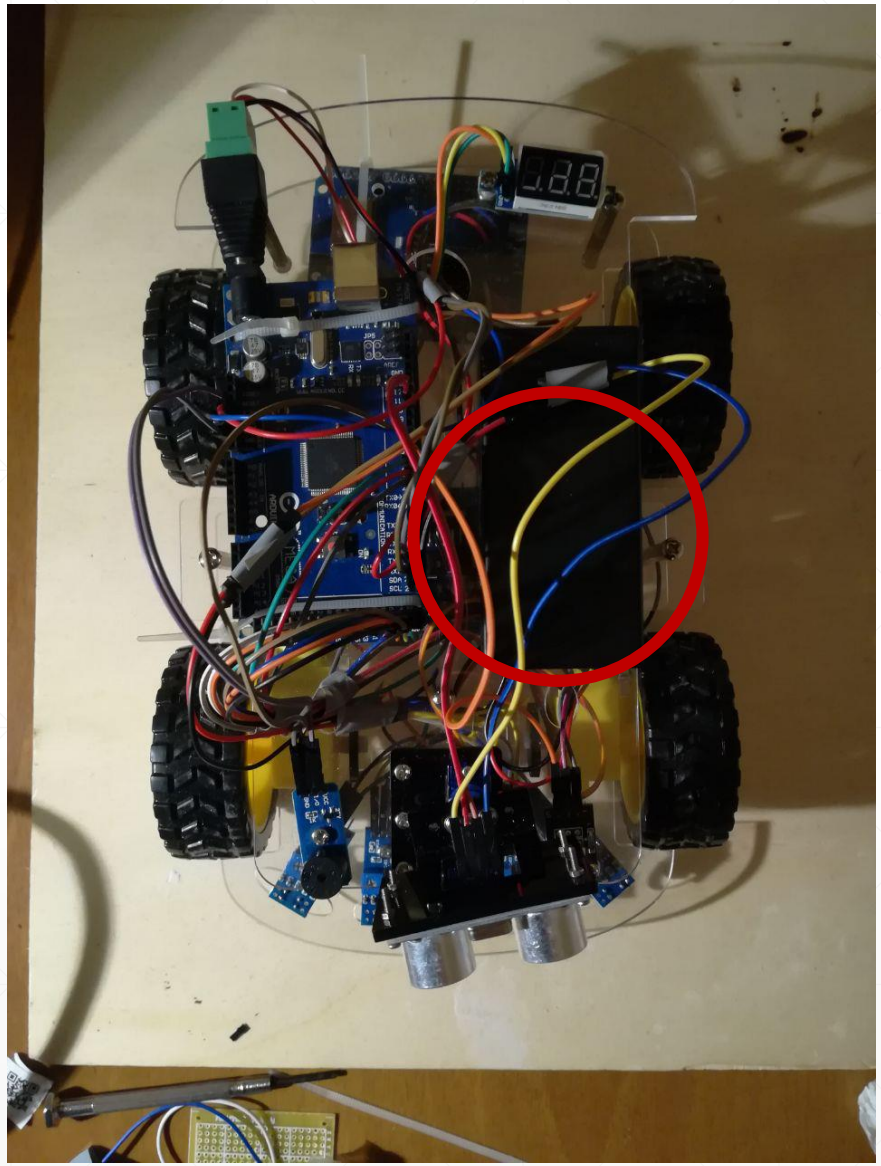
- DC motors + Wheels:
  - Sold in couples, 5 euros on Amazon, much less on Aliexpress.
  - Rated 3V – 6V



Components:

- Arduino Mega 2560 REV3
  - Has LOTS of GPIO pins, PWM-enabled pins and Analog pins
  - Multiple Serial interfaces
  - Same layout of Arduino Uno for shield compatibility

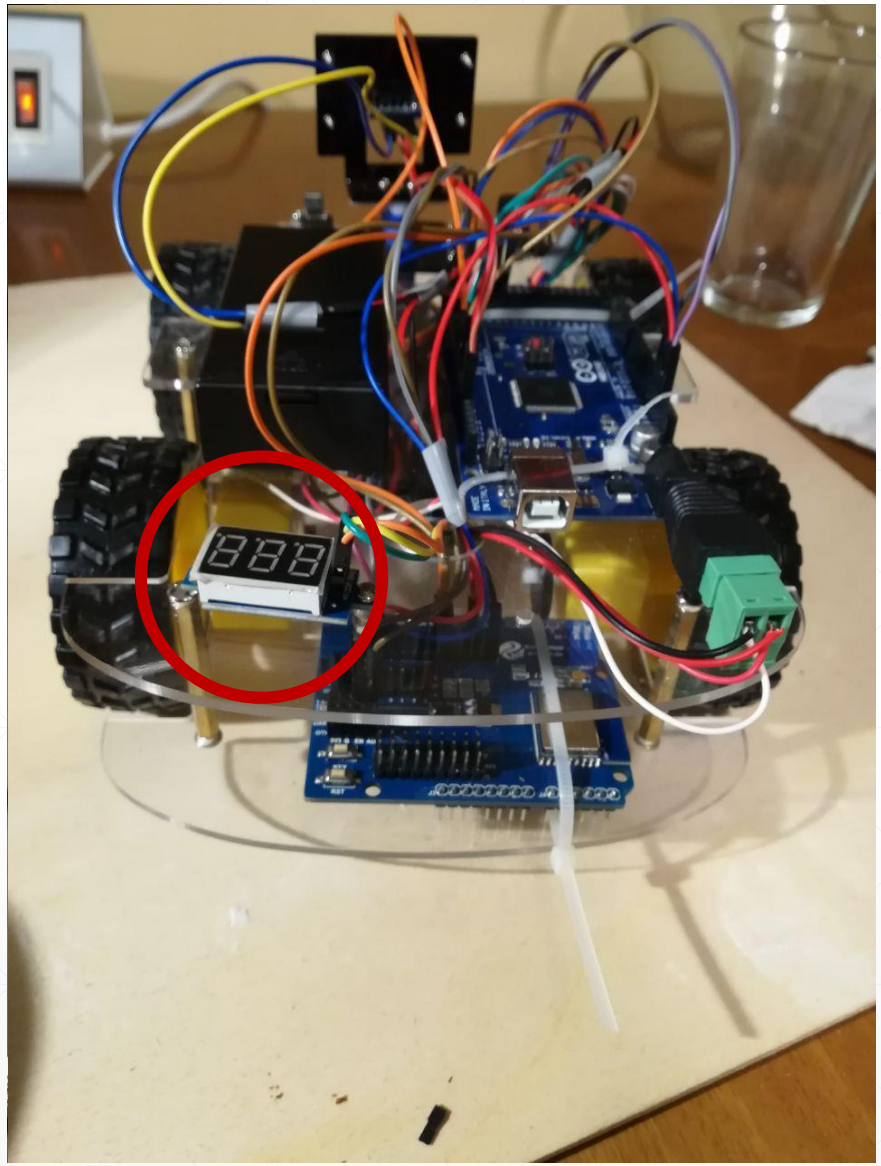




Components:

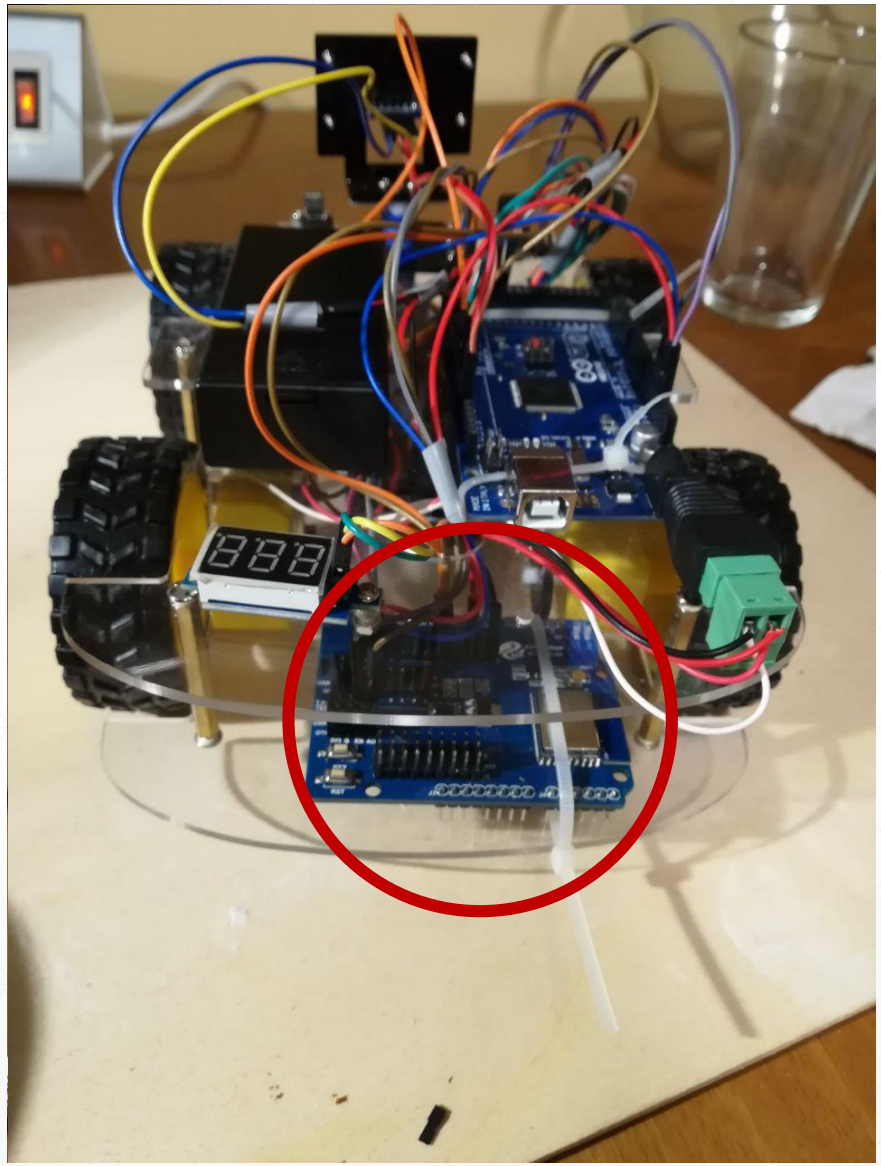
- 2x 18650 Li-ion cells
  - 3.6V each (actually almost 4V when fully charged)
  - **!!Faulty battery socket can make them catch fire!!**
    - But as long as you're careful it should be ok

18650



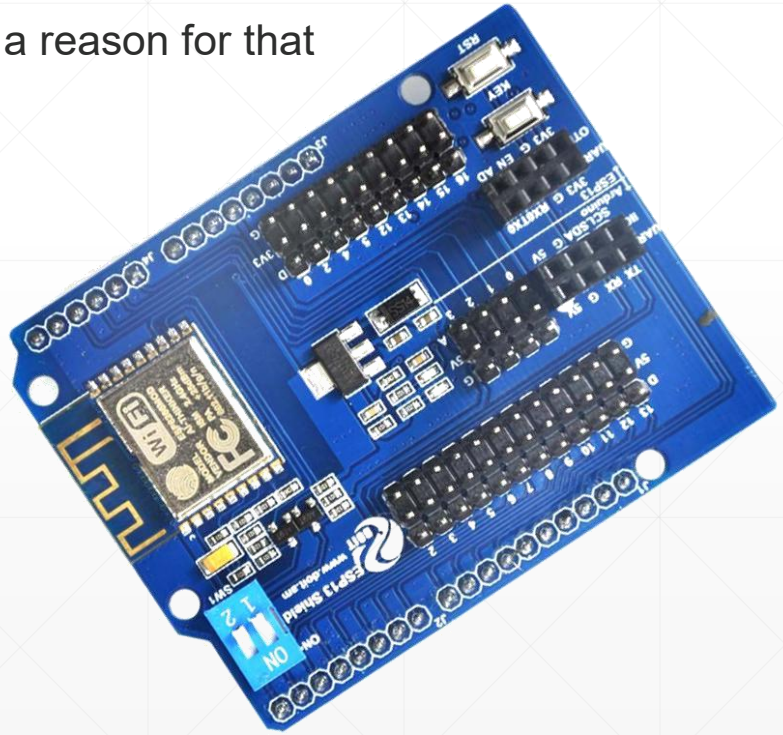
Components:

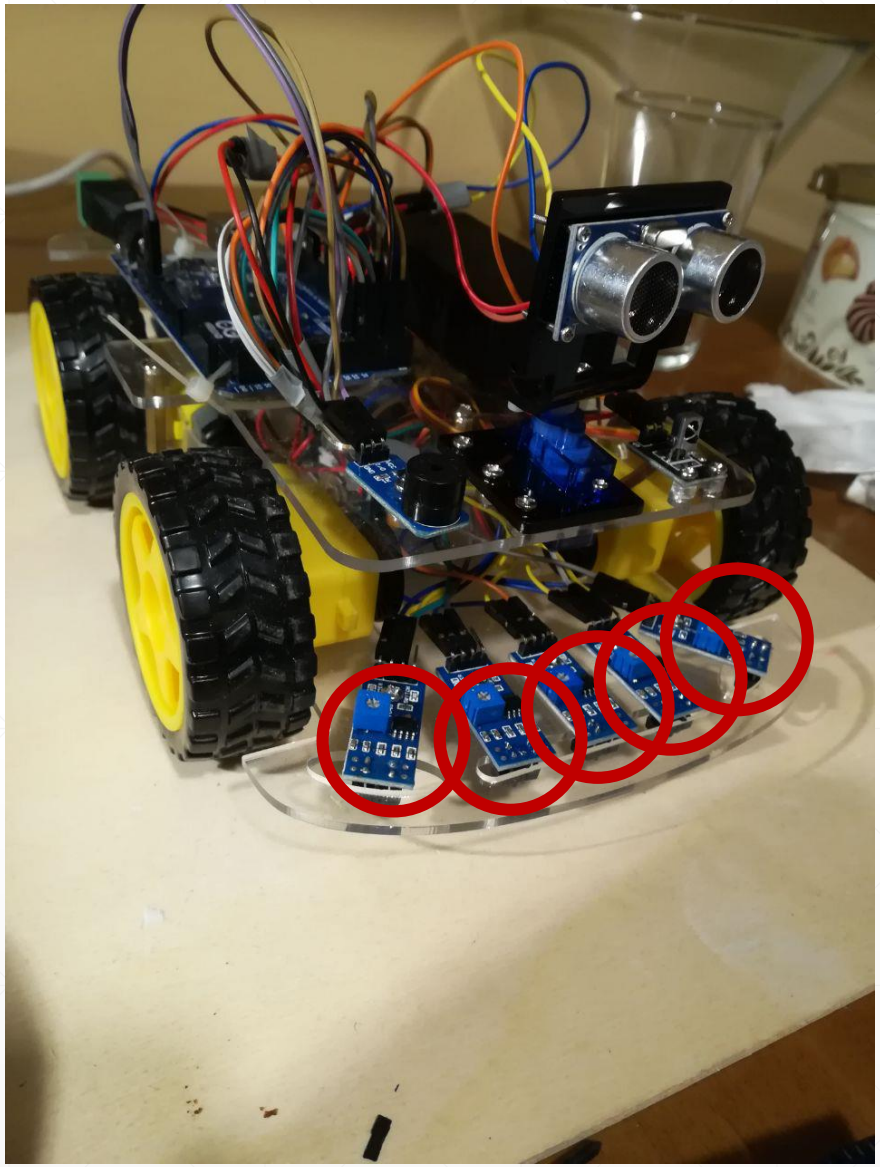
- Voltage meter:
  - Small
  - Not very accurate but it’s used only for battery charge status estimation



Components:

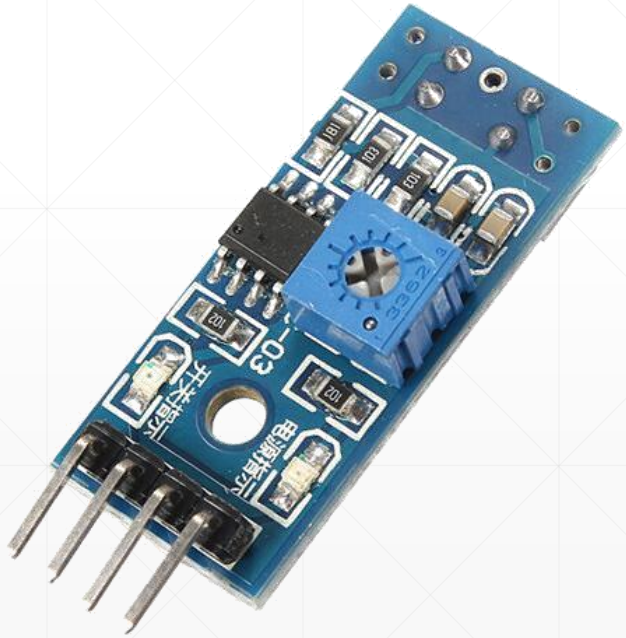
- ESP-13 shield
  - Carries an ESP-WROOM-02 module based on ESP8266
  - Not mounted as a shield (going to see why soon)
  - Cheap compared to other ESP8266 shields
    - Turns out there's a reason for that

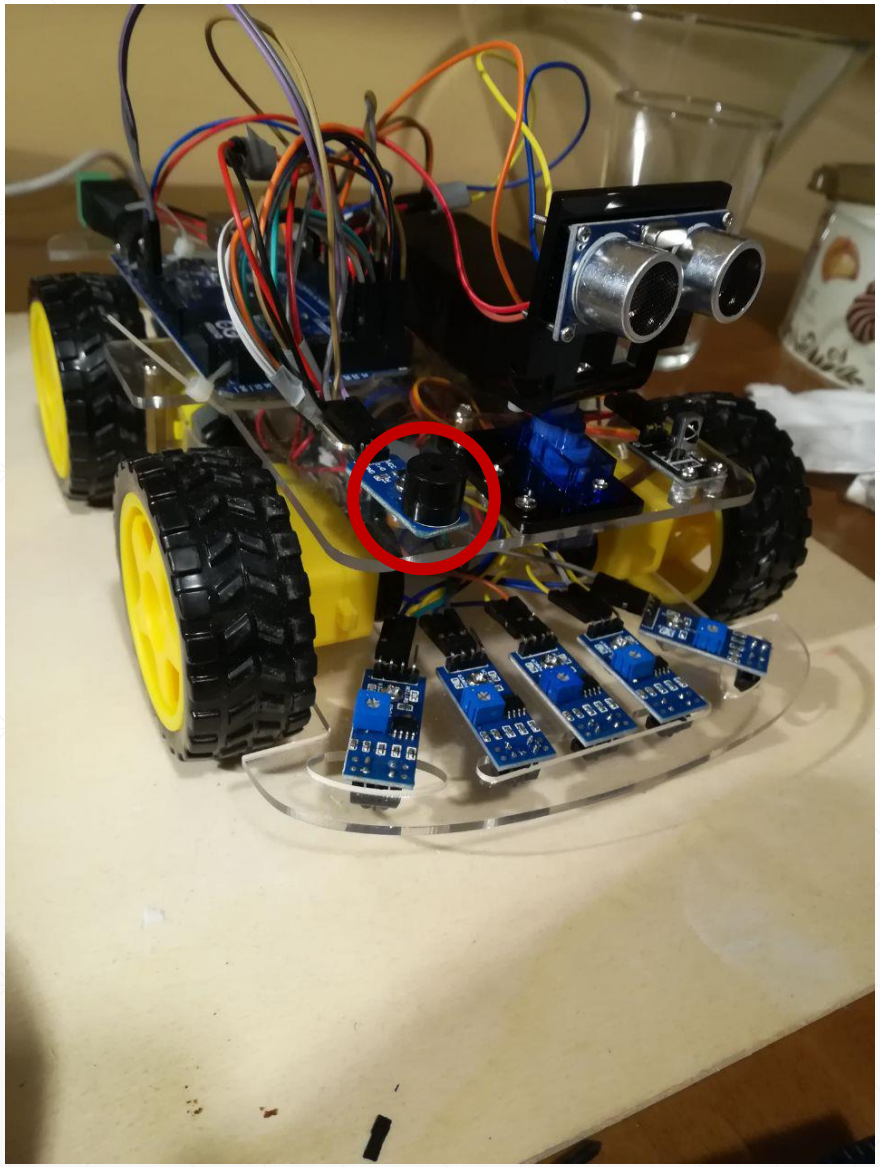




Components:

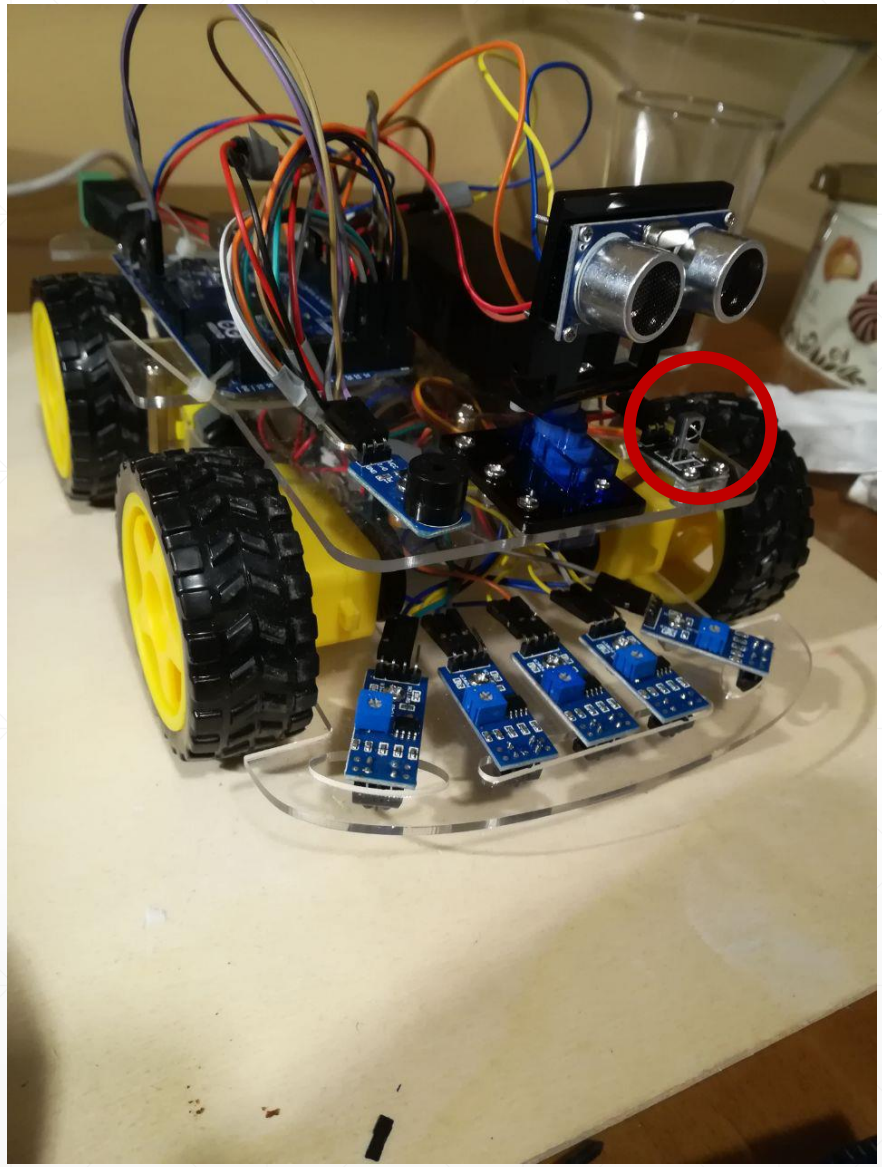
- 5X IR proximity sensing modules
  - Contains an LML393 comparator by ON Semiconductor coupled with a TRT5000 sensor by Vishay
  - Functioning explained later
  - Used to let the car follow a black line on the terrain





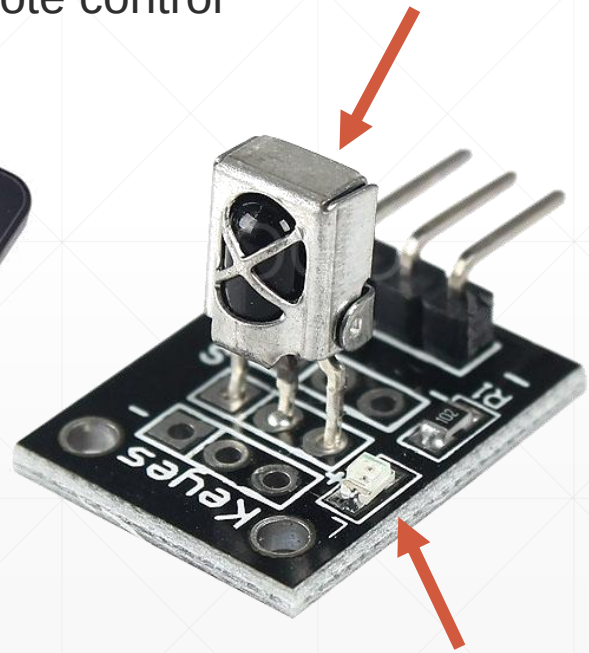
Components:

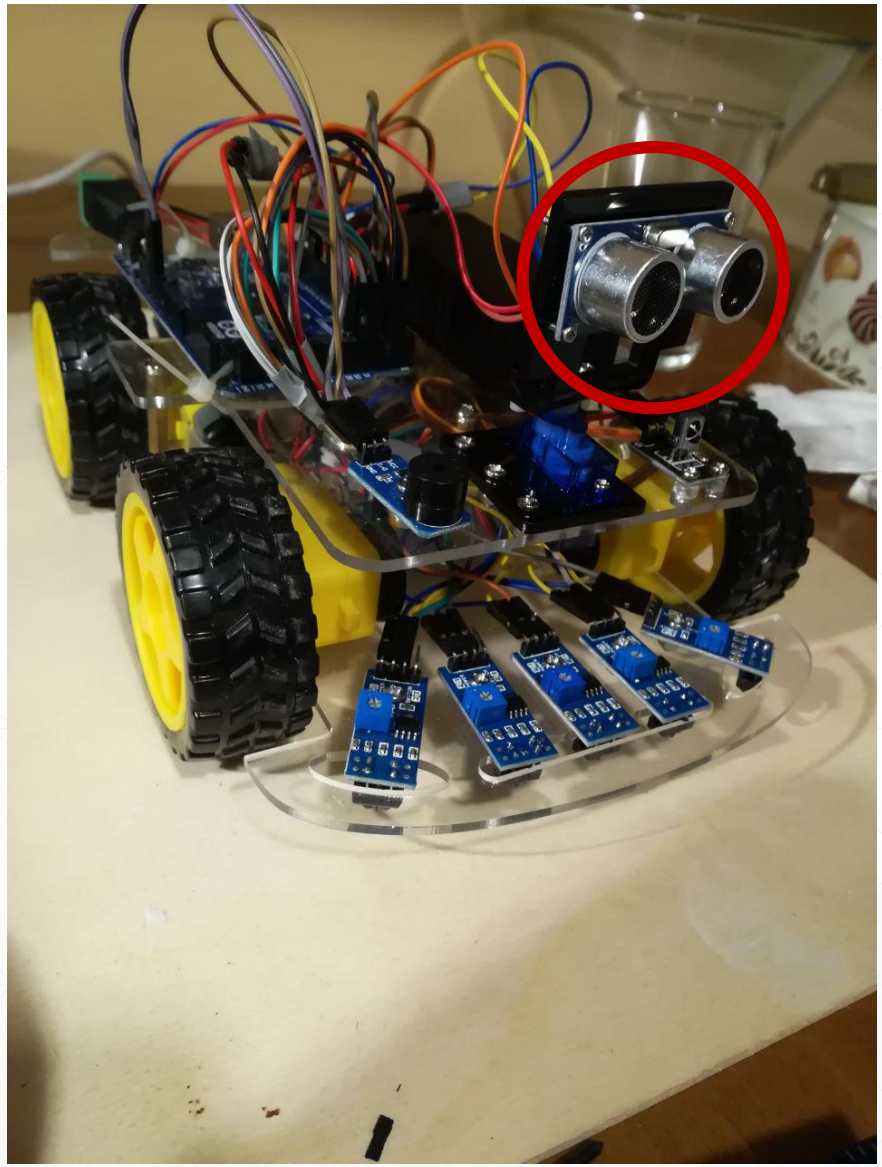
- Piezo Active Buzzer Module
  - Rated for 3.3V-5V (transistor onboard for power mode switching)
  - Different from passive buzzer:
    - Can generate a fixed tone when I/O is HIGH (3.3V)
    - Con: no frequency control
    - Pro: doesn't require a pre-generated waveform



Components:

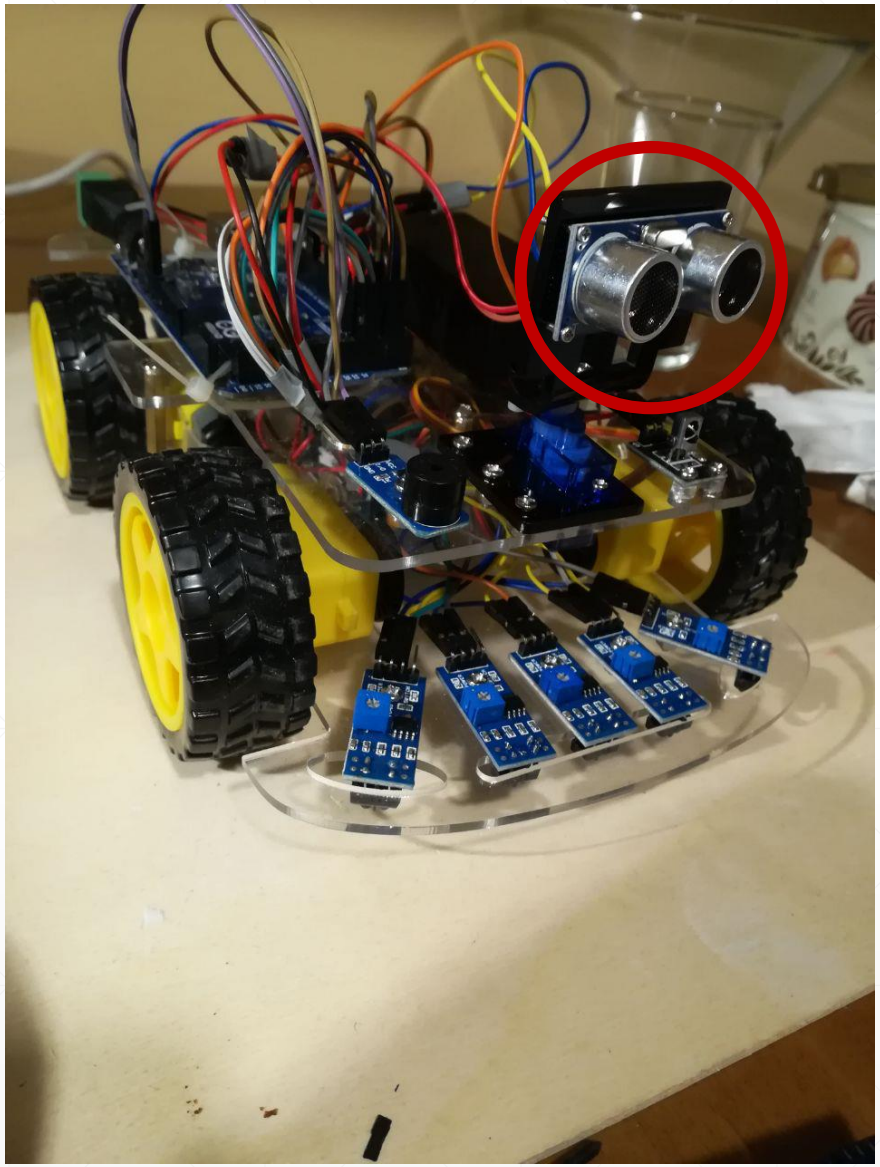
- IR receiver module
  - Basically a TL1838 Infrared Receiver mounted on a pcb + LED to signal incoming IR message
  - Allows car to receive signals from any cheap IR transmitter or remote control





Components:

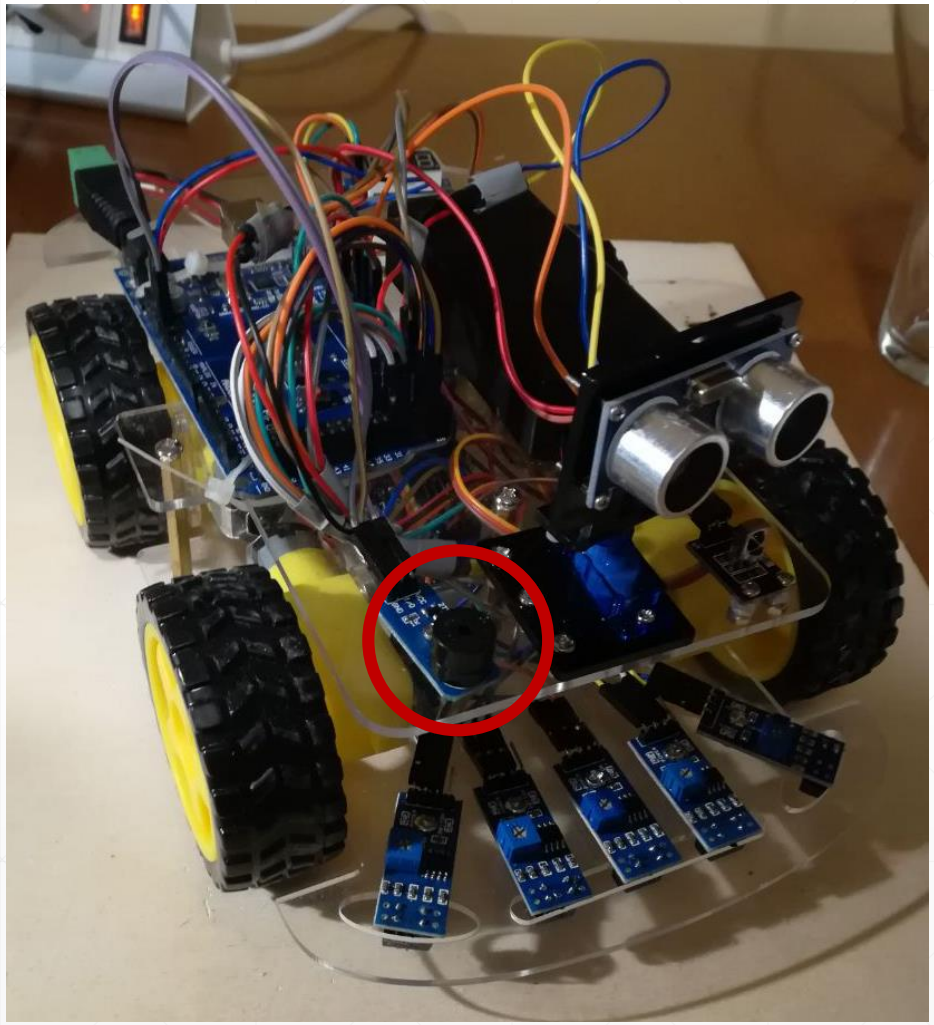
- HCSR04 Ultrasonic Ranging Module:
  - A couple of transducers, one for transmitting and one for receiving are used to measure an echo delay, thus estimating a linear distance from the sensor itself.
  - Mounted on a rotating support ot scan multiple directions
  - Kinda makes the whole thing look like the Curiosity rover



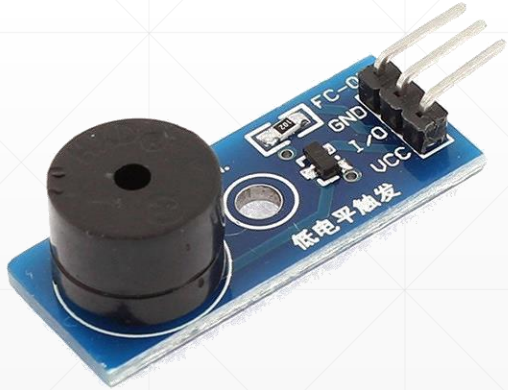
Components:

- Jumper Wires
  - Everything is connected using jumper wires:
    - MtF
    - MtM
    - FtF

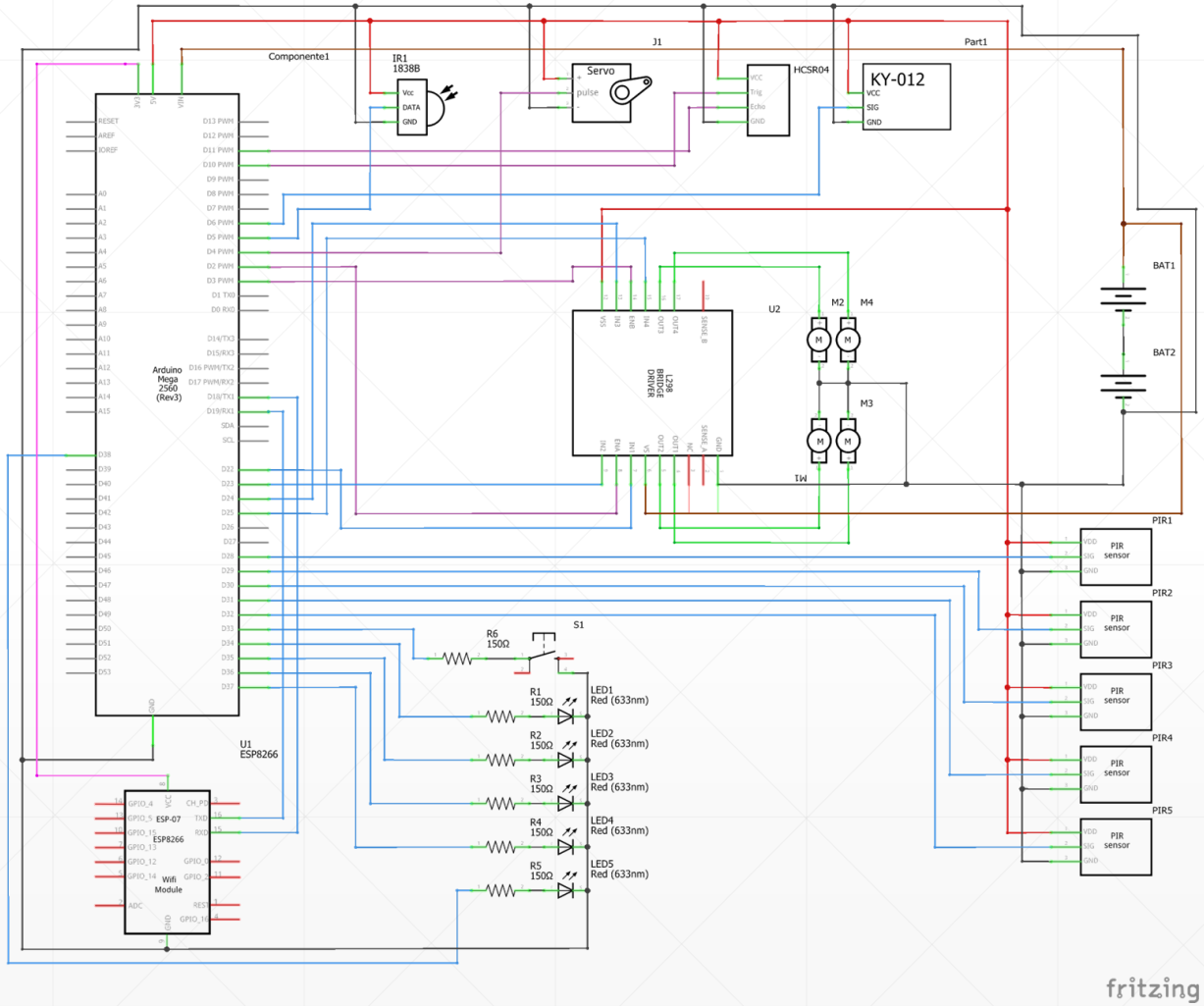
Components:

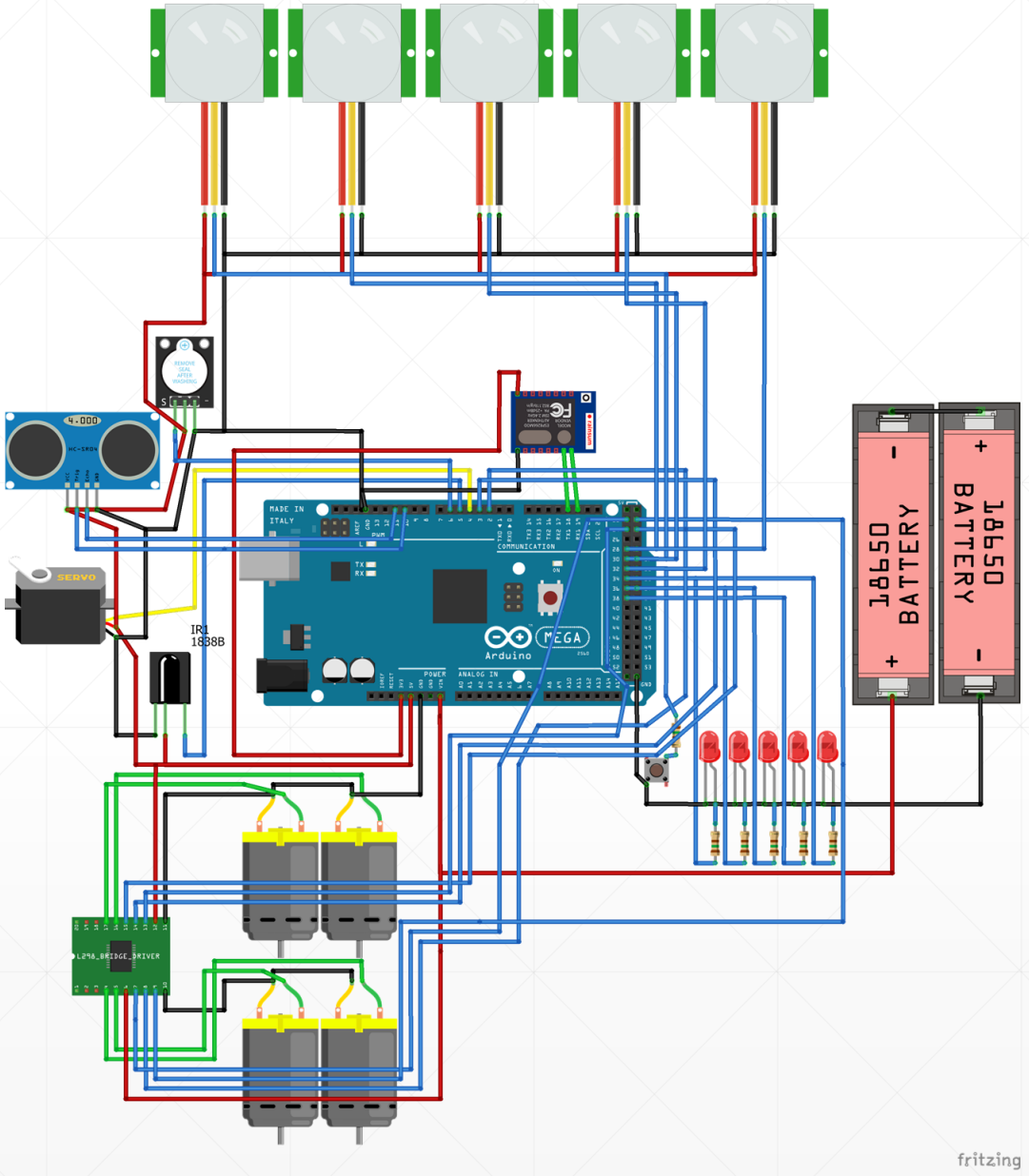


- Piezo Active Buzzer Module
  - Rated for 3.3V-5V (transistor onboard for power mode switching)
  - Different from passive buzzers

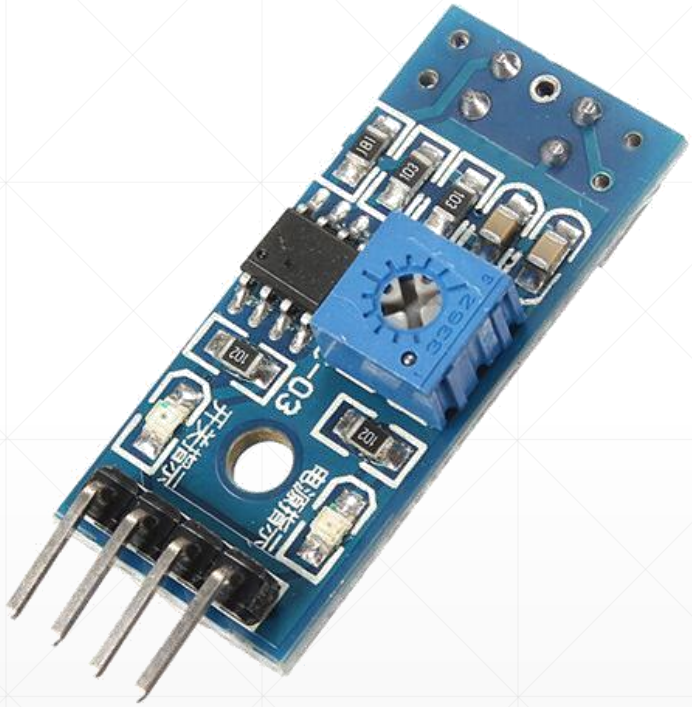


Schematics


Componente1
IR1
1838B
Servo
J1
HCSR04
KY-012
Part1
Arduino
Mega
2560
(Rev3)
U2
L298
BRIDGE
DRIVER
M2
M4
M3
BAT1
BAT2
PIR1
PIR2
PIR3
PIR4
PIR5
PIR
sensor
R6
150Ω
S1
R1
150Ω
LED1
Red (633nm)
R2
150Ω
LED2
Red (633nm)
R3
150Ω
LED3
Red (633nm)
R4
150Ω
LED4
Red (633nm)
R5
150Ω
LED5
Red (633nm)
U1
ESP8266
ESP-07
ESP8266
Wifi
Module
fritzing


SERVO
IR1
1838B
MADE IN
ITALY
MEGA
Arduino
L298_BRIDGE_DRIVER
18650
BATTERY
18650
BATTERY
fritzing

IR Tracking Sensor Module



The main idea behind it

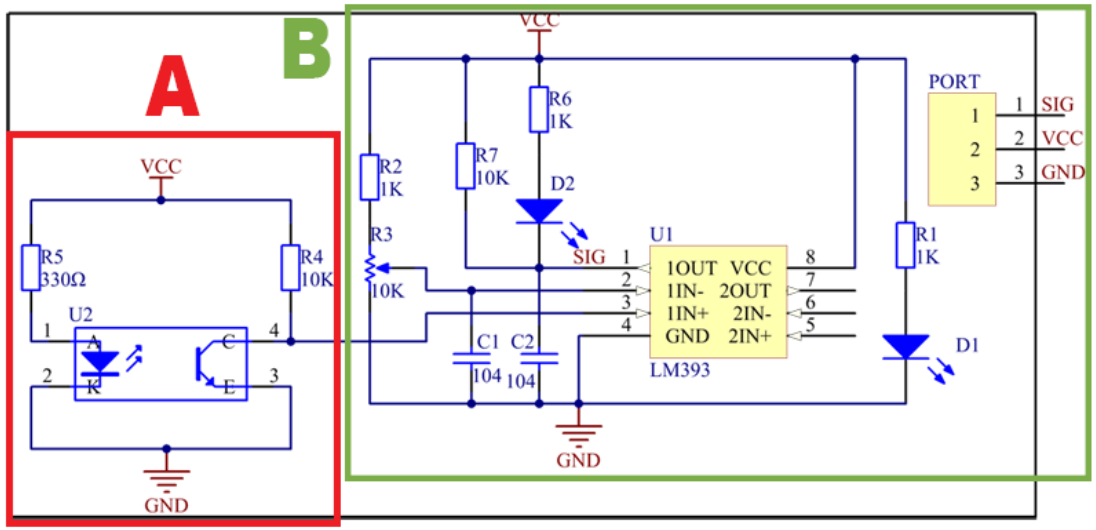
An IR led shines on a reflective/absorbing surface. A photoresistor then changes its resistance depending on the received IR radiation and the resulting voltage is then compared by an LM393 dual comparator chip.

Interface

A standard IR Tracking Sensor has a 3 Pin interface. VCC and GND for power suppling and a SIG pin for checking if the reflectance is beyond the threshold.

Our IR Tracking Sensor Module has also a fourth pin that allows the user to read the actual voltage signal of the sensor. However, we do not use that pin in our project.

The Schematics:



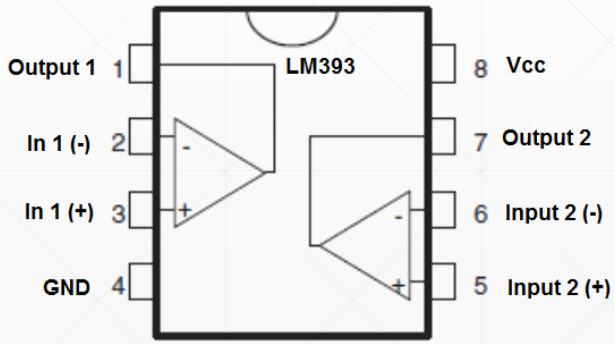
LM393 is constituted by 2 standard low offset voltage comparators (only one is used) that returns VCC as +VCC and GND as -VCC, working indeed as a sort of Boolean operator. Capacitors C1 and C2 are placed in order to maintain SIG and 1IN- stable in case of power supplying problems.

IR Tracking Sensor Module

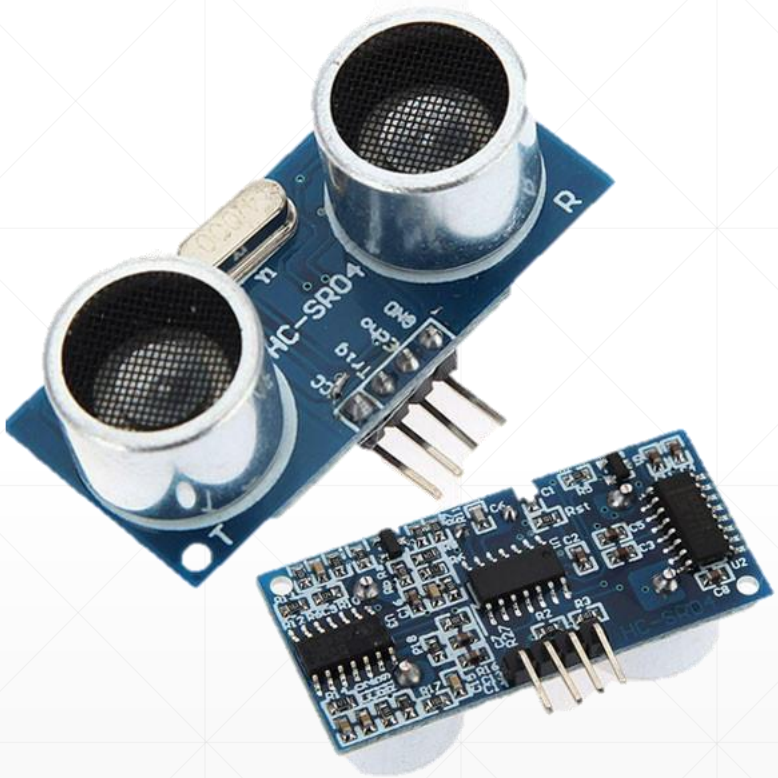
Block A contains the **U2** module that has the **IR** led and the **Photoresistor**.

Block B contains a simple **comparing circuit** with two status led (**D1** and **D2**) and one **potentiometer R3** for manual regulating the threshold of the entire sensor.

D1 indicates whether the sensor is powered up or not. D2 indicates if the signal is positive (**SIG=VCC**).



HC-SR04 Echo Locator



Main idea behind it

The sensor emits ultrasonic soundwaves to determine the distance of an object from **2cm** to **400cm** with a resolution of **0.3cm** and a measuring angle of **30 degrees.**

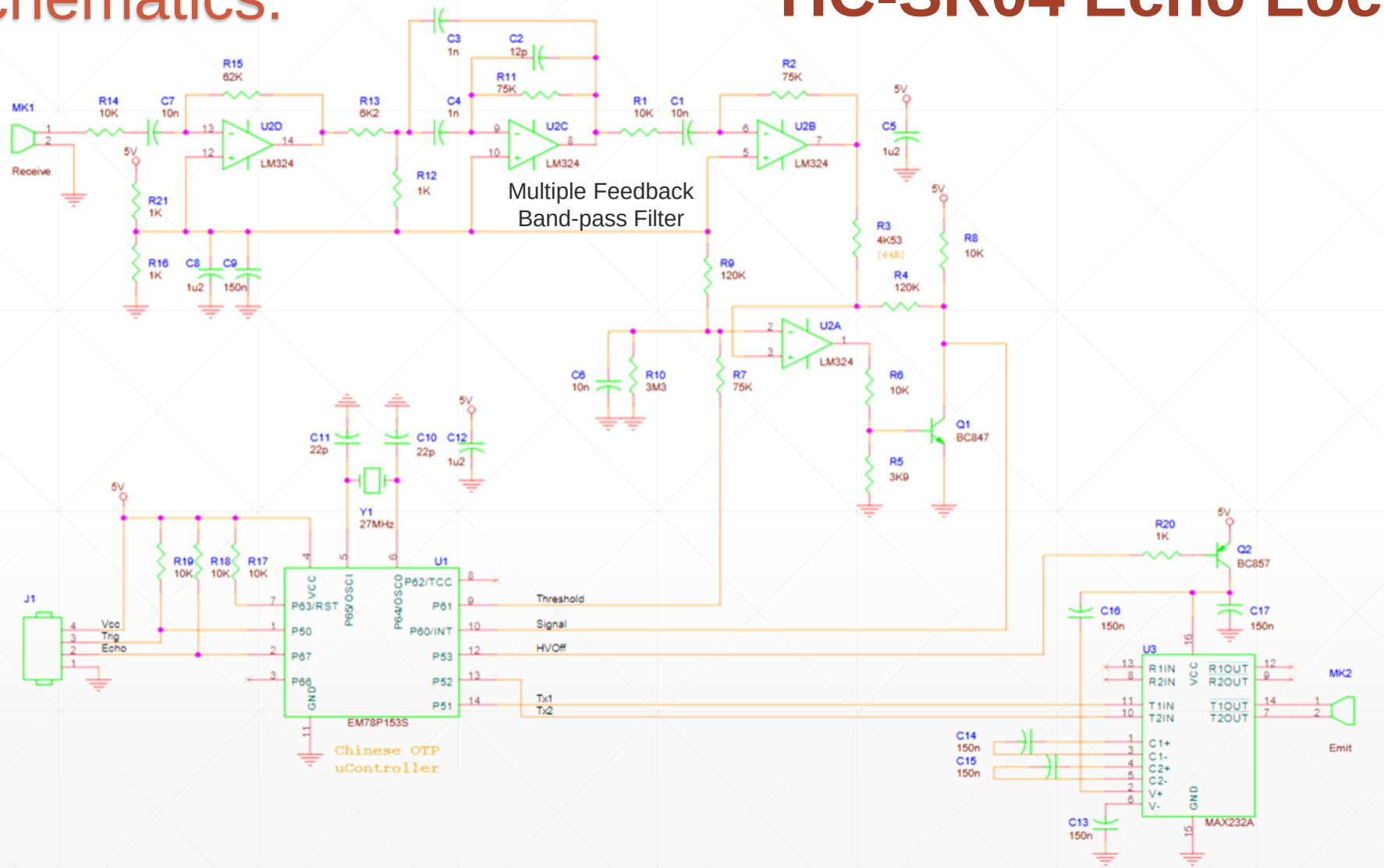
Main notes on interface and functionality

The sensor has an interface of 4 pins (**VCC, Trig, Echo, GND**). Whenever the Trig pin is set on HIGH, the **transmitter** (left piezo) sends the high-frequency soundwave. Whenever the receiver (right piezo) detects the echo of the soundwave, the Echo pin sets on HIGH.

You can send and read signals to the sensor manually setting up **triggers** and **delays of 5-10µs** with a standard Arduino Board**.**

The Schematics:

HC-SR04 Echo Locator



The Schematics:

HC-SR04 Echo Locator

The HC-SR04 unit consists of three main parts: microcontroller, Transmitter and Receiver with associated amplifiers.

Microcontroller U1 (EM78P153):

- Interface with Trig an Echo pins
- Timing and sending antiphase burst to the transmitter module
- Squelch control (i.e. cutting receiver during transmission in order to avoid bogus echoes)
- Receiving processed signal by the filtering part as a Rising edge interrupt

Transmitter U3 (MAX232):

- Voltage drive to TX transducer from the antiphase TX signals from the micro (U1)

Receiver U2 (LM324):

Quad op-amp (low grade 1MHz unity gain bandwidth device)

More notes

HC-SR04 Echo Locator

The **receiver** is a chain of **three op-amps** as small signal stages, the final stage is a variable threshold hysteresis comparator with output switch.

First stage is an inverting amplifier of the **RX transducer signal**, with gain around **5.6** (allowing for tolerances). The output is attenuated with reference to mid-rail.

Second stage is a bandpass filter hopefully with tolerances **centred around 40 kHz** the frequency of the emitted pulses.

Third stage is another amplifier with gain of approximately **10**.

The last op-amp stage is a **variable threshold hysteresis comparator** with output switch, the output witch is formed by a transistor in **U3** and pullup resistor **R17**.

Piezo Active Buzzer

Main idea behind it

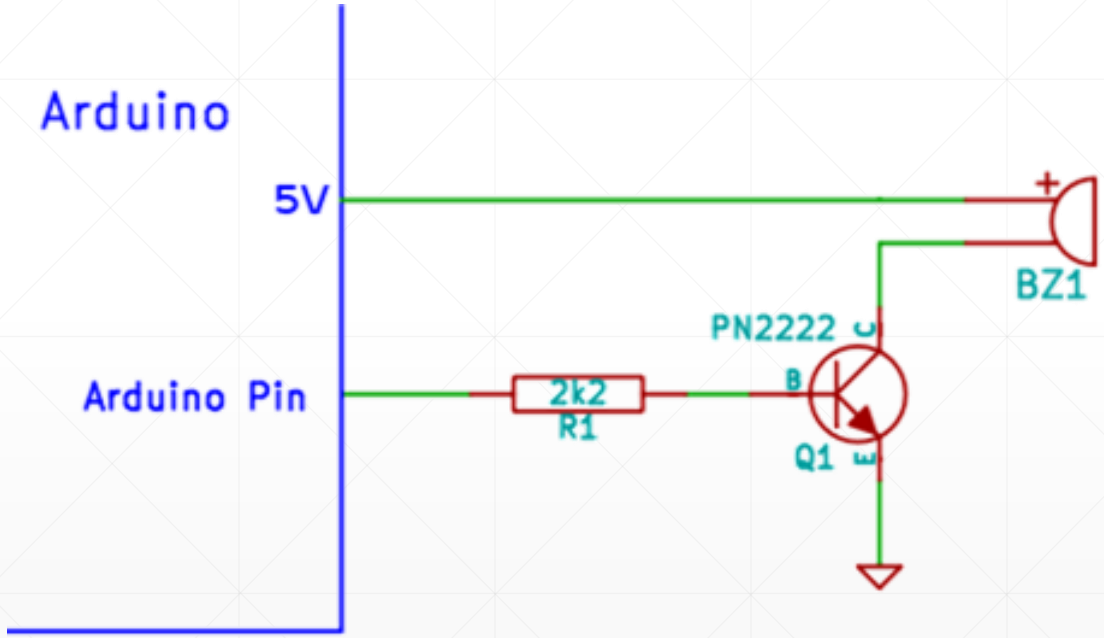
Whenever the module is powered and the trigger is set off, a membrane vibrates at a **fixed frequency** (**Active Buzzer**) generating a precise sound wave.

Interface

VCC and GND for power supply and I/O for triggering the sound.

If this were a **Passive Buzzer**, there would be no I/O pin, but we would had to give the precise sound frequency through the VCC and GND pins.

Piezo Active Buzzer – Schematic

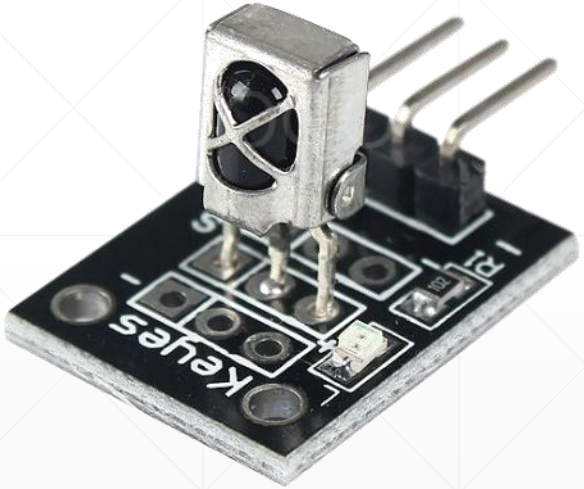


Observations on the schematics

The circuit is simply constituted by a **resistance** and a **npn transistor** that acts as a switch on the I/O pin.

The Active Buzzer has a built-in oscillating source that will make a precise sound wave when electrified.

IR Receiver Module



Main idea behind it

A photodiode with integrated demodulator electronics is sealed on a simple PCB with 2 led status and a 3 pin interface.

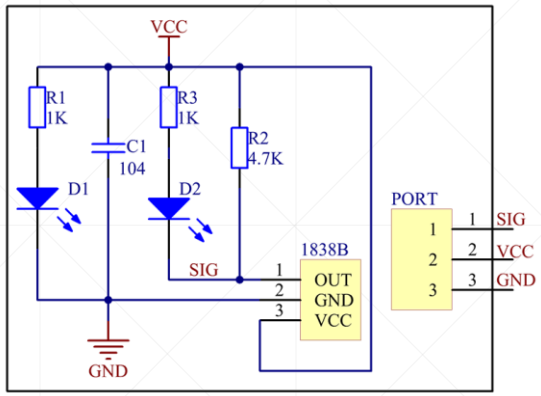
The photodiode is a TL1838 that works on carrier frequencies of 35-40KHz

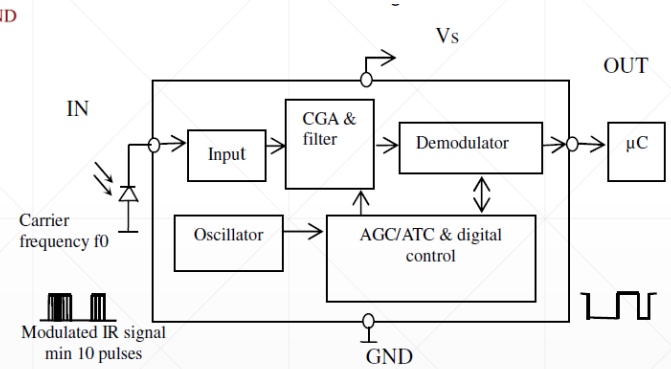
Interface

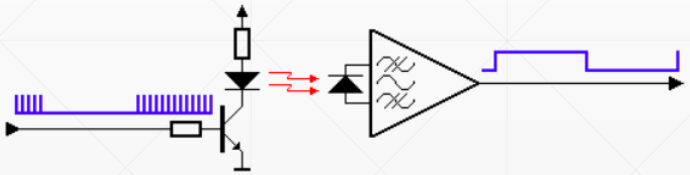
VCC and GND for power supply and SIG for measuring the IR signals recieved.

IR (~300GHz) Receiver Module

The Schematics:







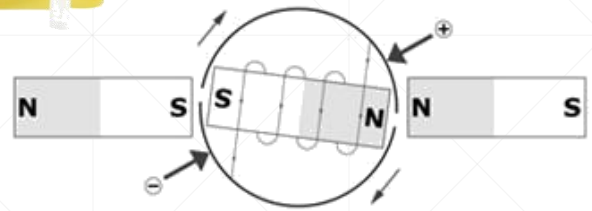
Observations on the schematics

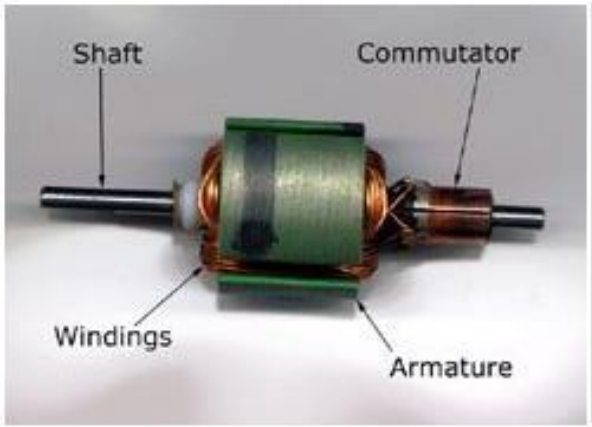
The interface on the PCB is a simple wrapping circuit with a power status led (D1), a SIG status led (D2) and a capacitor for power stability (C1).

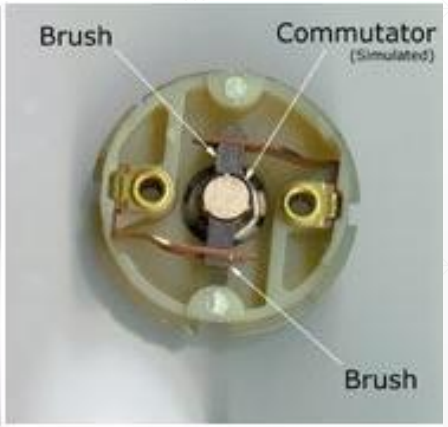
Remarks on the TL1838

The component has all the elements for receiving 35-40KHz modulated signals in intervals of 600µs. Unfortunately, we did not find the precise schematics of the component but only black-box schemes of its functioning.

The (Simple) Physics of a DC Motor





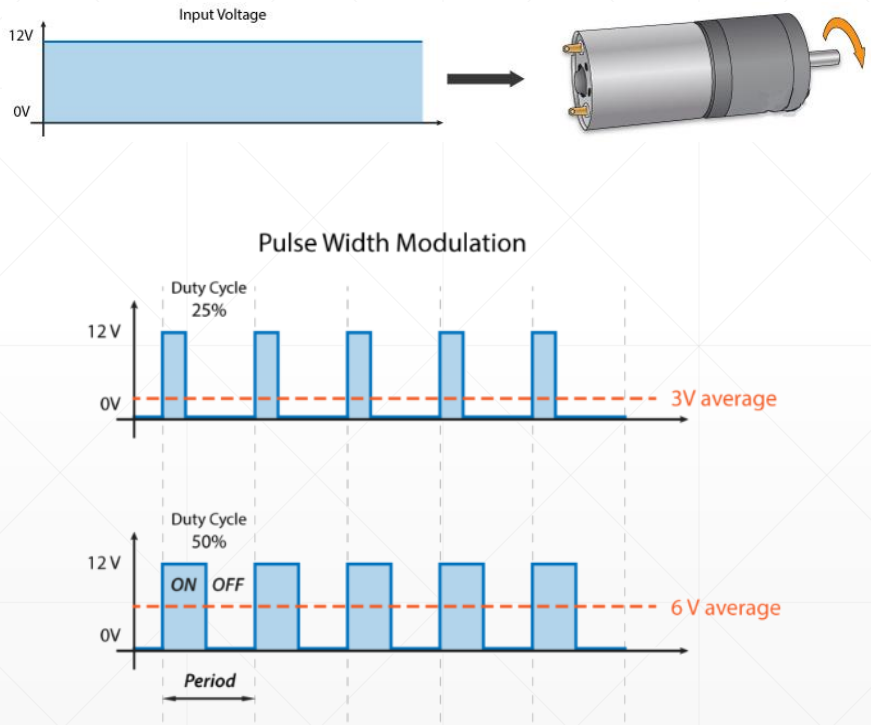


From electrical to rotational energy

Two permanent magnets are placed next to an electromagnet whose inductor connectors (a pair of brushes) are placed in contact with a split-ring commutator.

With this simple configuration shown in these pictures (real DC motors may have more than 2 sectors), as the electromagnet is moving its polarity is changing and the shaft may keep rotating.

Controlling the Speed with PWM:



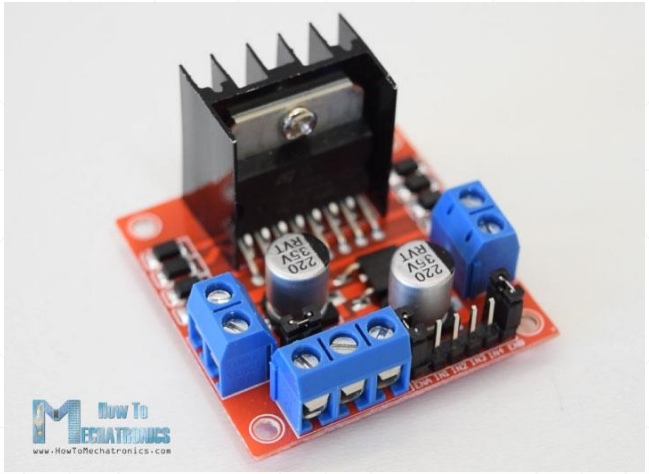
In order to have different speeds on a DC motor, it's sufficient to provide different values of voltage. But how can we provide such values when or microcontrollers only support Digital Outputs (i.e. all or nothing)?

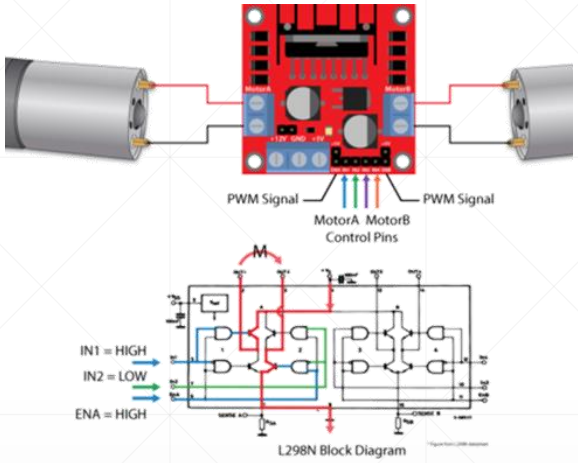
With Pulse Width Modulation (PWM)

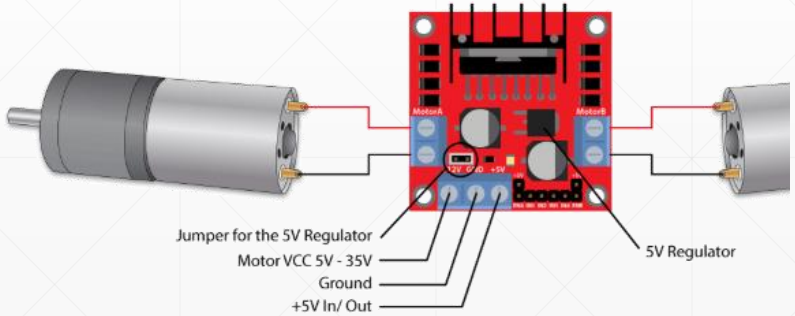
PWM is a technique which permits to adjust the average value of the voltage that's going to the electronic device by turning on and off the power at a fast rate. The average voltage depends on the duty cycle.

Almost every Arduino module and shield comes with at least one GPIO which supports PWM.

One module to control them all...





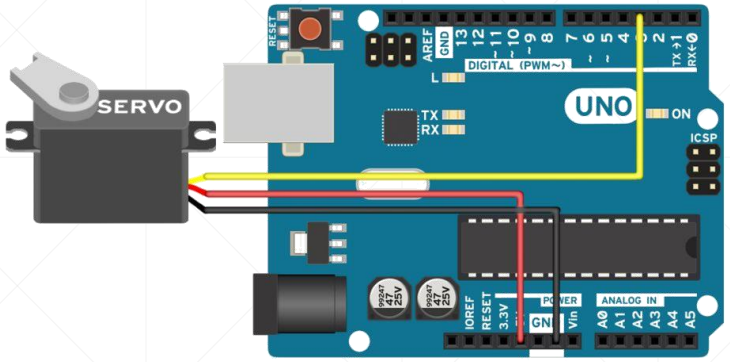


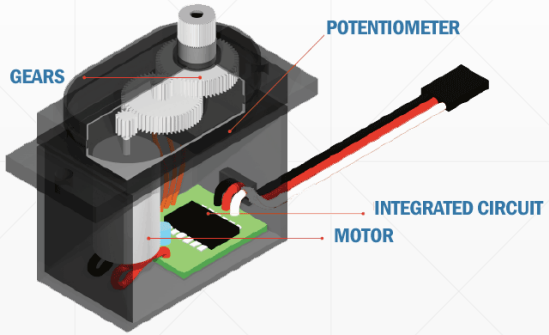
In order to control both the PWM 'voltage' and the wheels' direction (inverting the voltage means inverting the rotation), the most common module in amateur projects is the **L298N**.

This module supports voltages between 5 and 35V, with a peak current up to 2A.

Here we show a single module capable of controlling two wheels at the same time, but we actually make use of a bigger module with 2xL298N and some support circuits for controlling also Servo Motors.

Servomotors





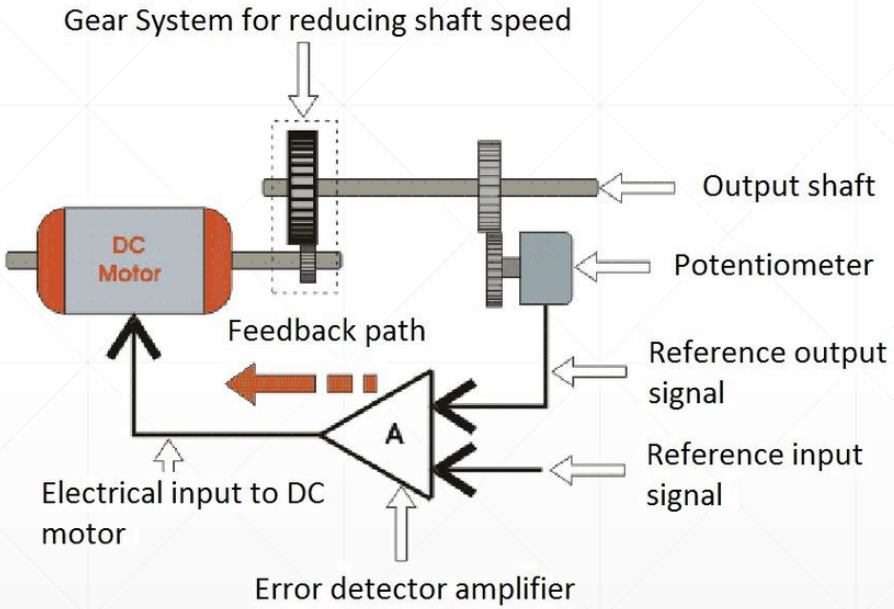
Perfect control over angular position

Servomotor is a term used to refer to motors suitable for use in a closed-loop control system (e.g. a system where the control action from the controller is dependent on feedback from the process).

In our contest, our servomotor is a rotary actuator whose orientation can be precisely controlled with a simple PWM signal.

This allows us to make the Echo Locator of the car ‘look around’ at precise degrees.

ServoMotors: how do they work?

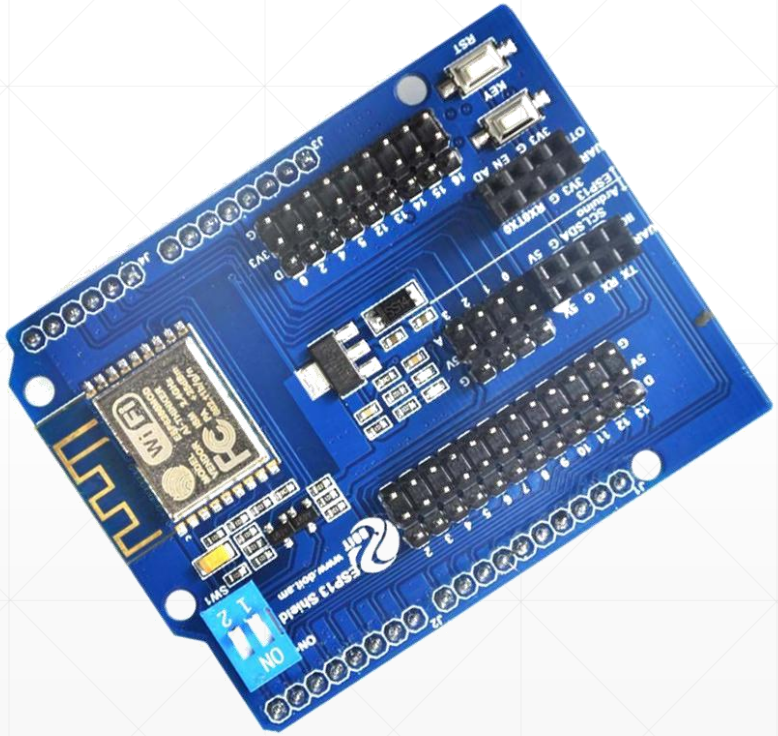


Servomotors are a simple combination of a DC motor, a potentiometer and a closed-loop control circuit.

The DC motor is attached by gears to control the output shaft orientation. When the shaft of the motor is at the desired position, the power supplied to the motor is stopped by the op-amp in the closed-loop control circuit.

The motor's speed is proportional to the difference between its actual position and desired position. This is called proportional control.

Putting everything together:



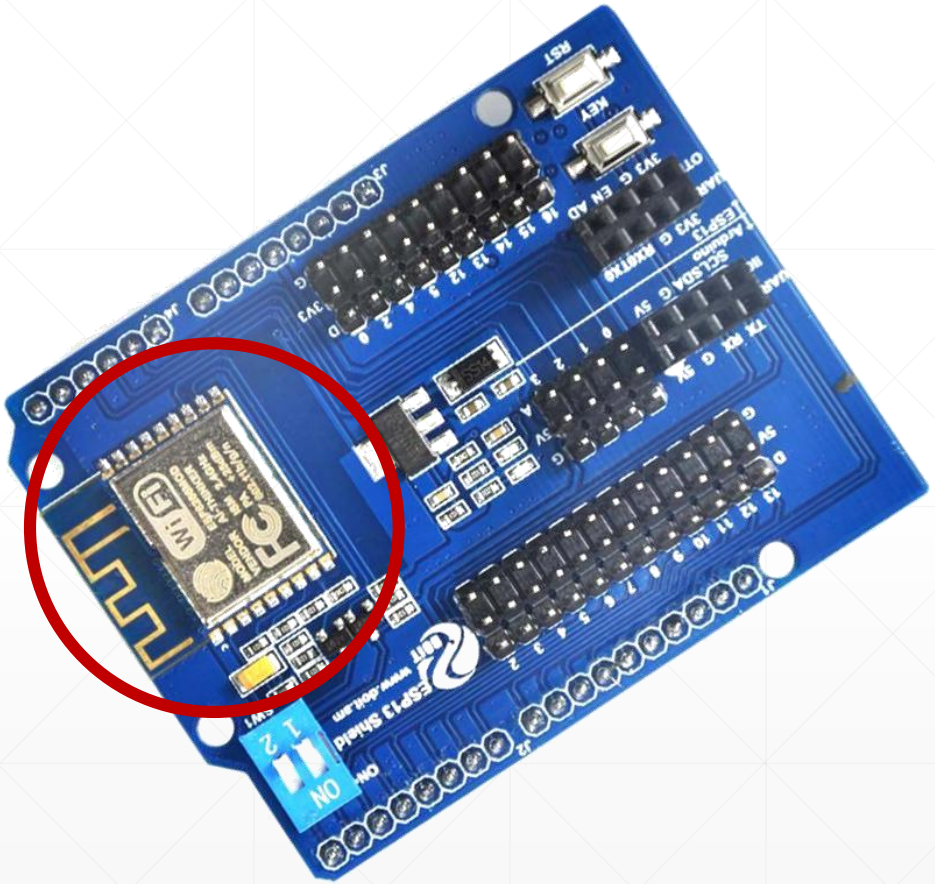
The ESP-13 shield
(odissey)

The ESP shield

The **ESP-13** is an Arduino Uno shield whith an embedded **ESP-WROOM02** module.

The **ESP-WROOM02** by **Espressif** is an integrated WiFi 802.11 b/g/n module designed for providing wireless connection to embedded systems, it's powered by an **ESP8266EX** chip, a 32 bit low-power microcontroller, with 2MB of onboard memory and can be programmed with the Arduino IDE.

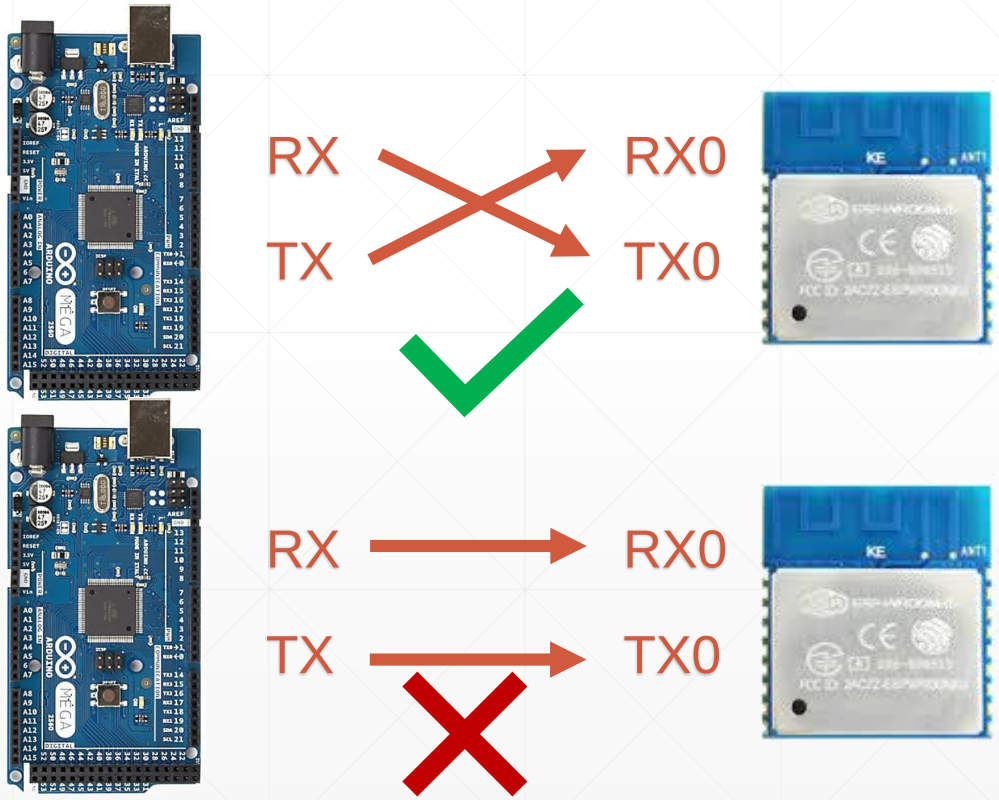
ESP-13 by **Jaycar** offers a breakout for all the pins of the ESP-WROOM02, an onboard voltage divider for powering the ESP-WROOM02 itself (Arduino offers a 5V line but ESP8266 uses 3.3V signals both for powering and GPIO communication) and a level shifter for the logic signals coming from and to the Arduino.

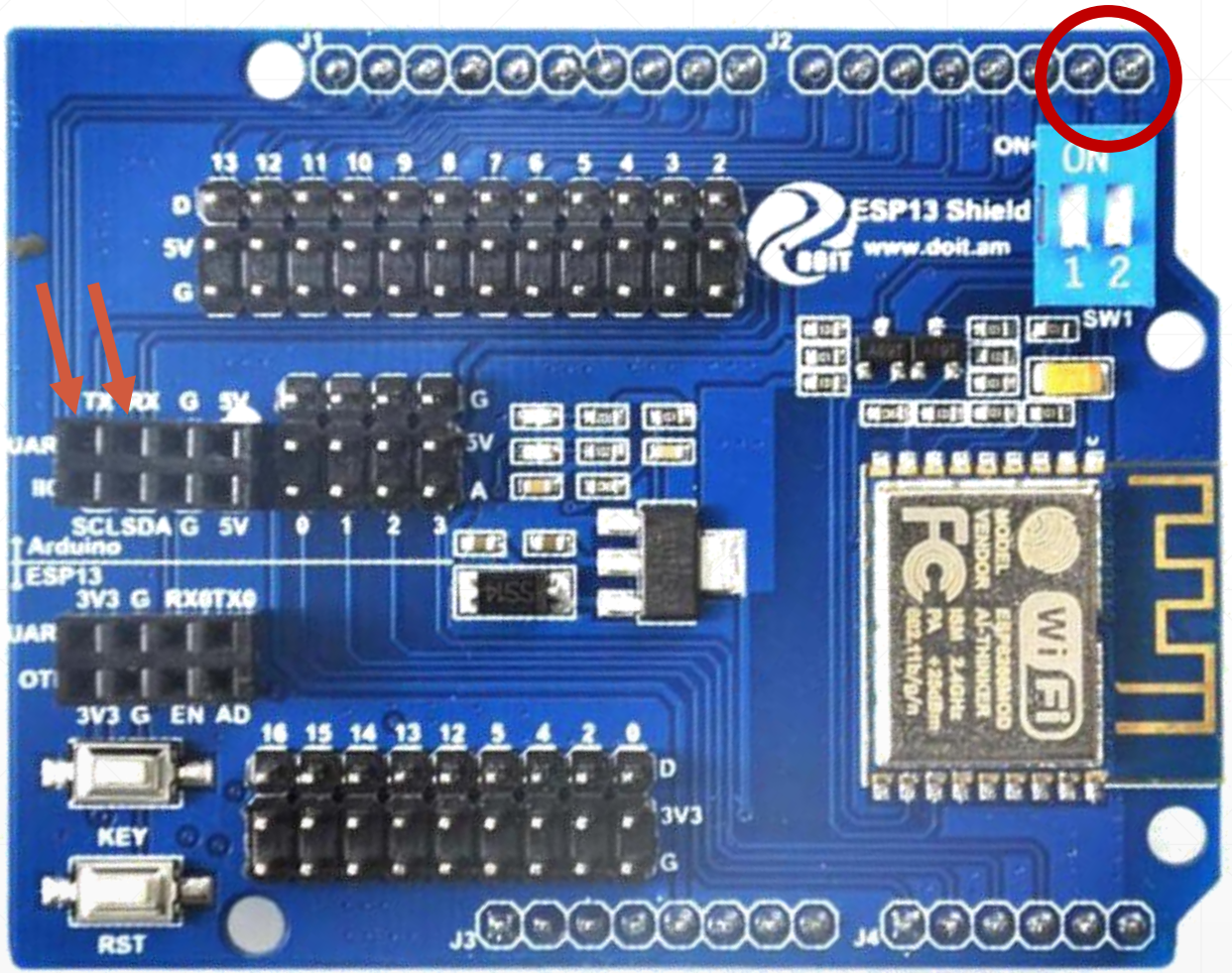


Or at least it should…

The ESP shield

The **RX** and **TX** pins on the shield, corresponding to the pins **0 (RX)** and **1 (TX)** in the Arduino layout were actually **switched**, rendering impossible to use the shield in its intended configuration.

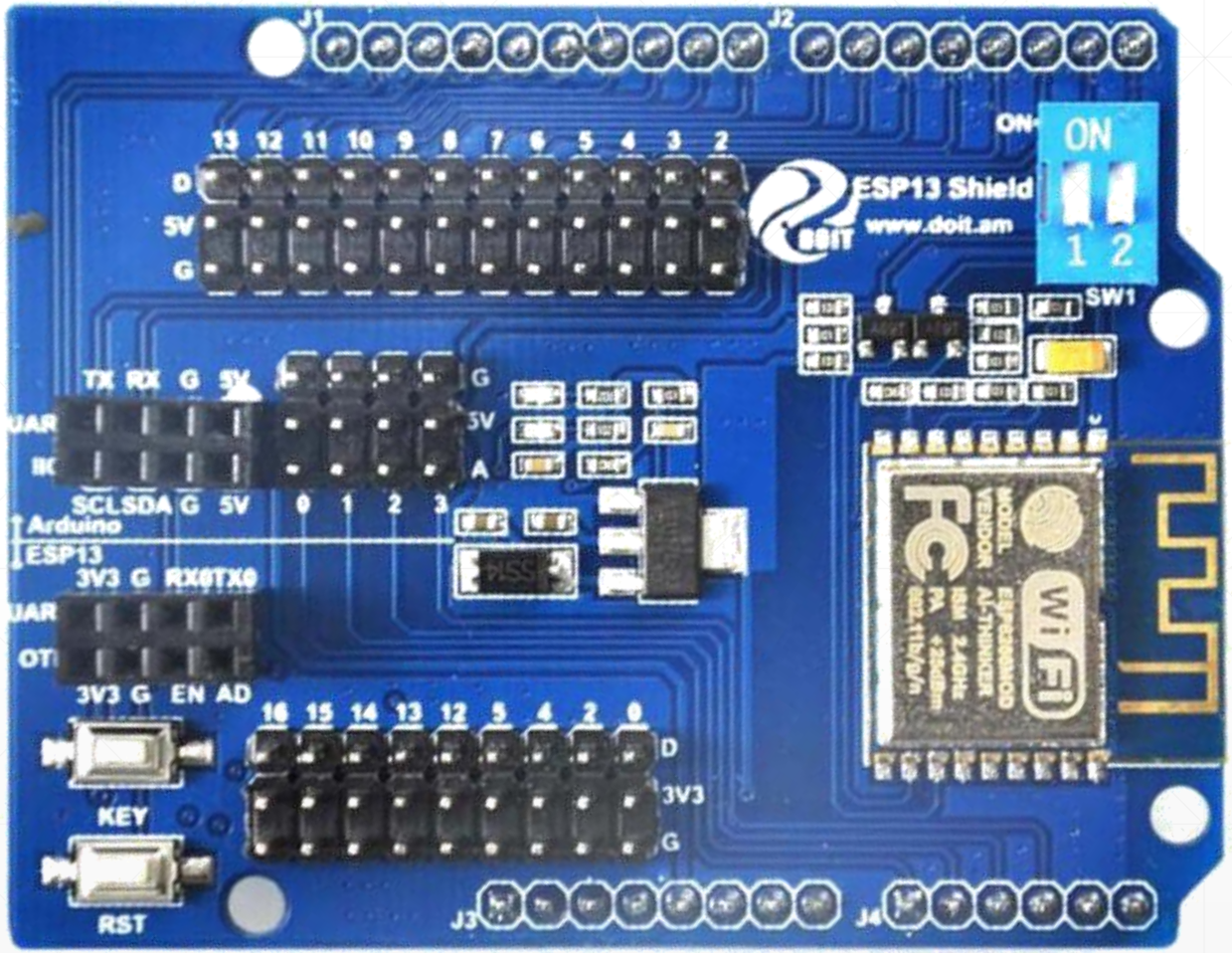




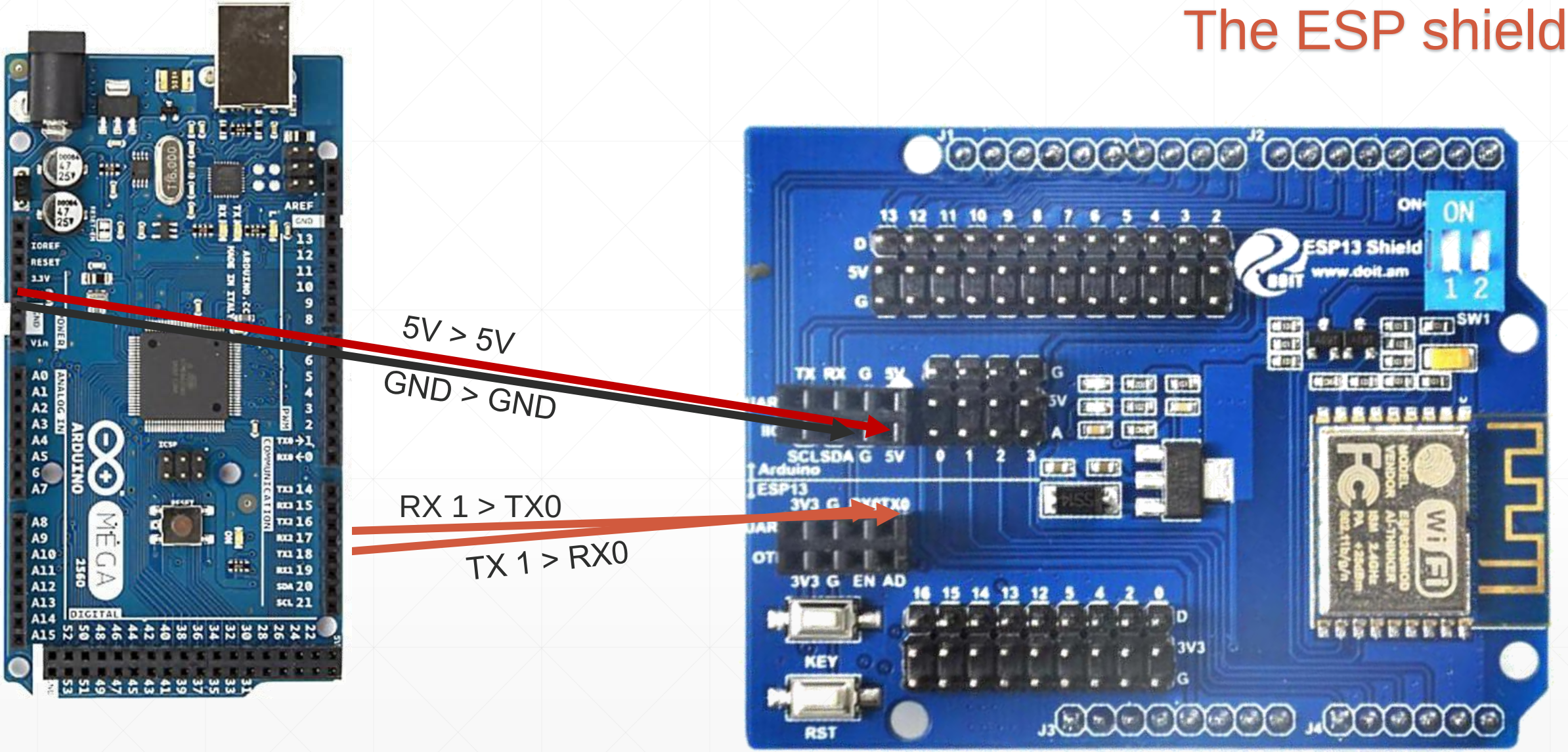
The ESP shield

Possible solutions:

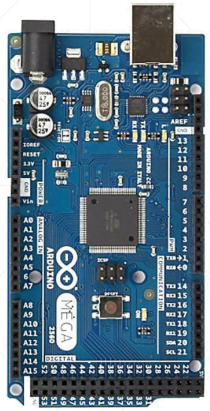
- Trash the whole thing
  - Tempting but not very cost-effective
- Fix the onboard traces
  - Scrape the soldermask to expose the traces
  - cut the traces in two different points
  - run two jumper wires
  - Time-consuming
  - We didn’t have any small-diameter wire laying around that day
- Just connect it using jumper wires
  - **Quickest solution**



The ESP shield



The ESP shield



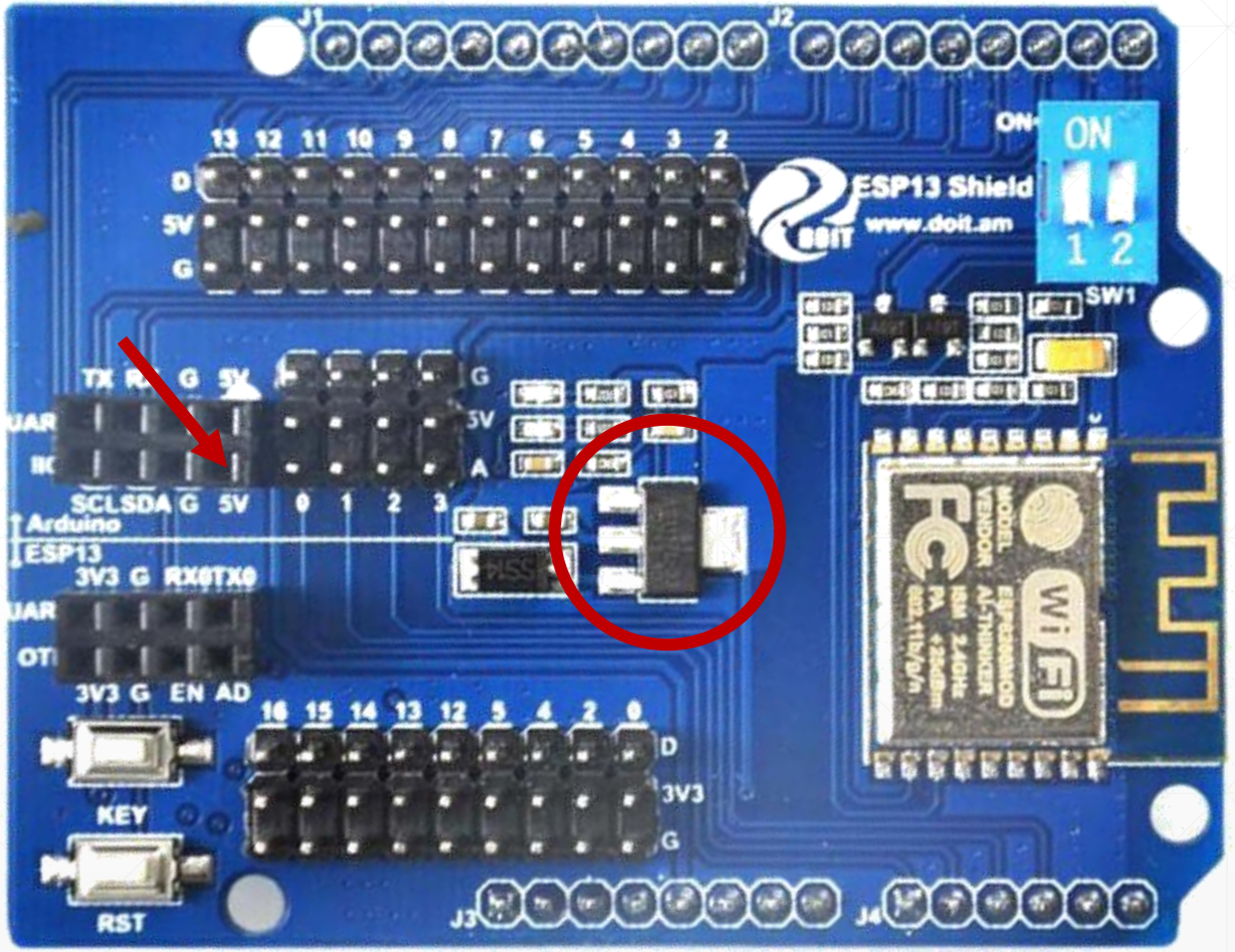
5V (ARDUINO) → **3.3V (ESP8266)**

The onboard voltage regulator should convert the 5V power provided by arduino and convert it to a 3.3V line that can be used by the ESP8266

BUT…

The serial lines on the ESP8266 should also use 3.3V logic: we need a level shifter so that 5V signals from the Arduino can be translated to 3.3V for the ESP8266 and vice versa.

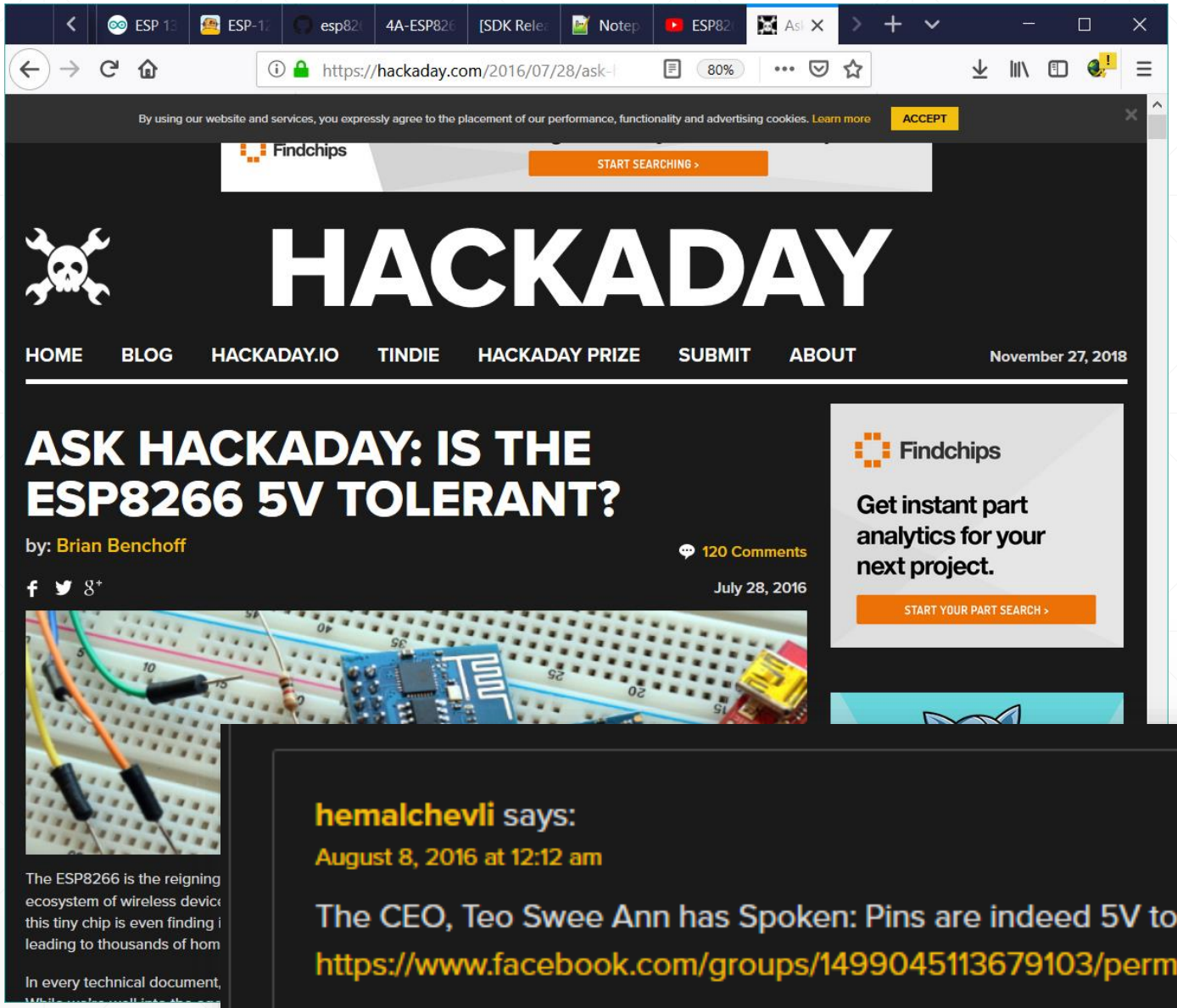
or do we?

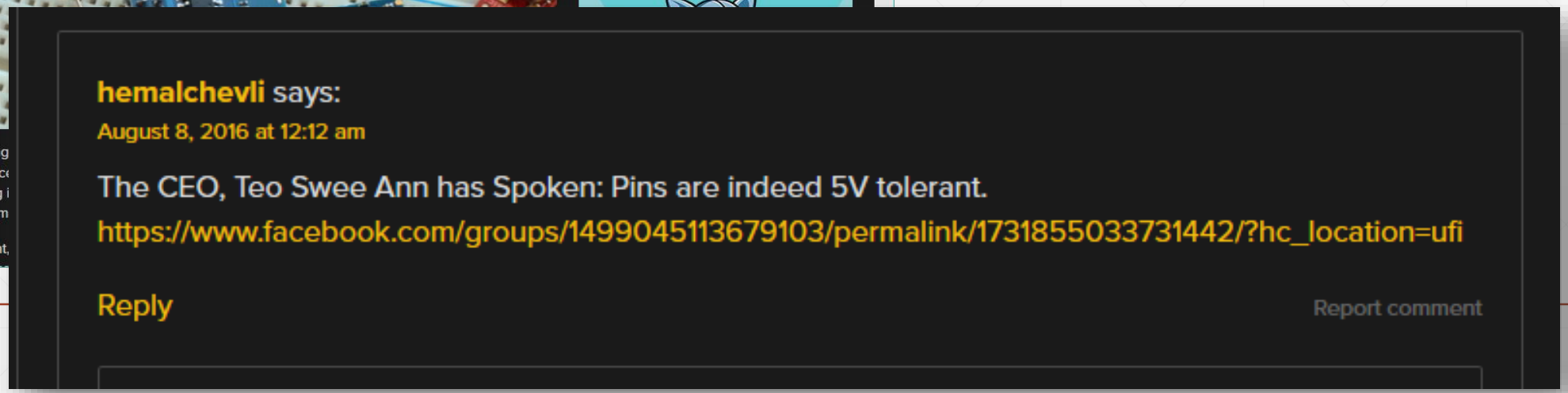


The ESP shield

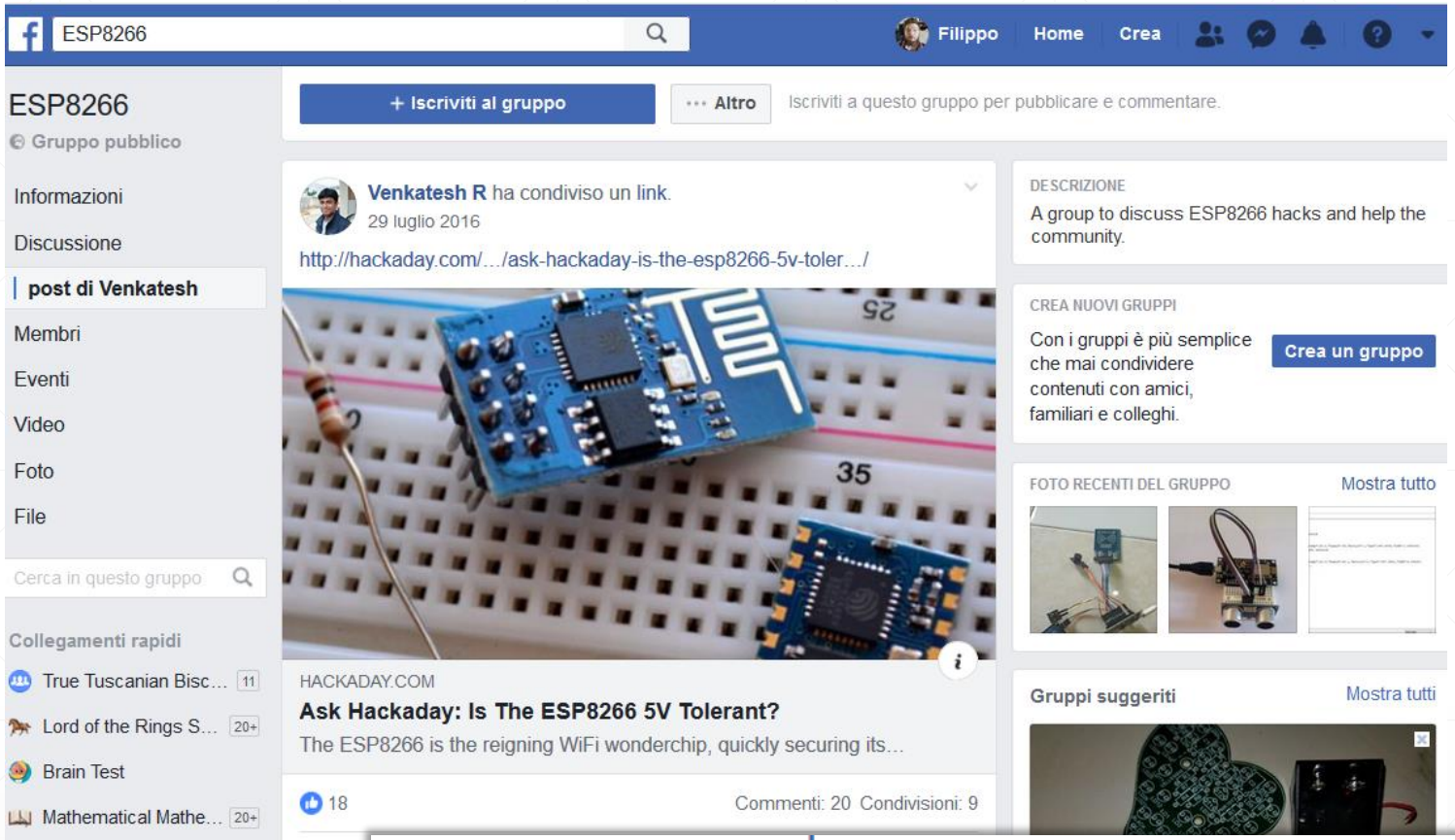
I remembered seeing some years a go a post on **hackaday.com** discussing wether the ESP8266 was actually 5V tolerant: people were using the ESP8266 serial lines with 5V logic without consequences

In order to have more valid sources to support this assertion. I decided to look more into it.

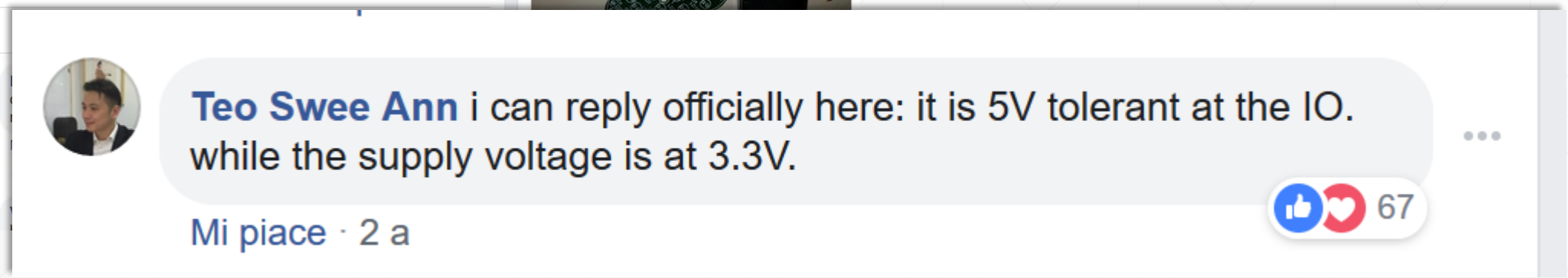




The ESP shield



With the benediction of the Espressif CEO himself we could connect the ESP8266 to 5V logic without any worries



The ESP shield

Can we finally use the ESP8266?

Nope

To be programmed using the Arduino IDE, the ESP8266 needs the **AT Firmware** by Espressif

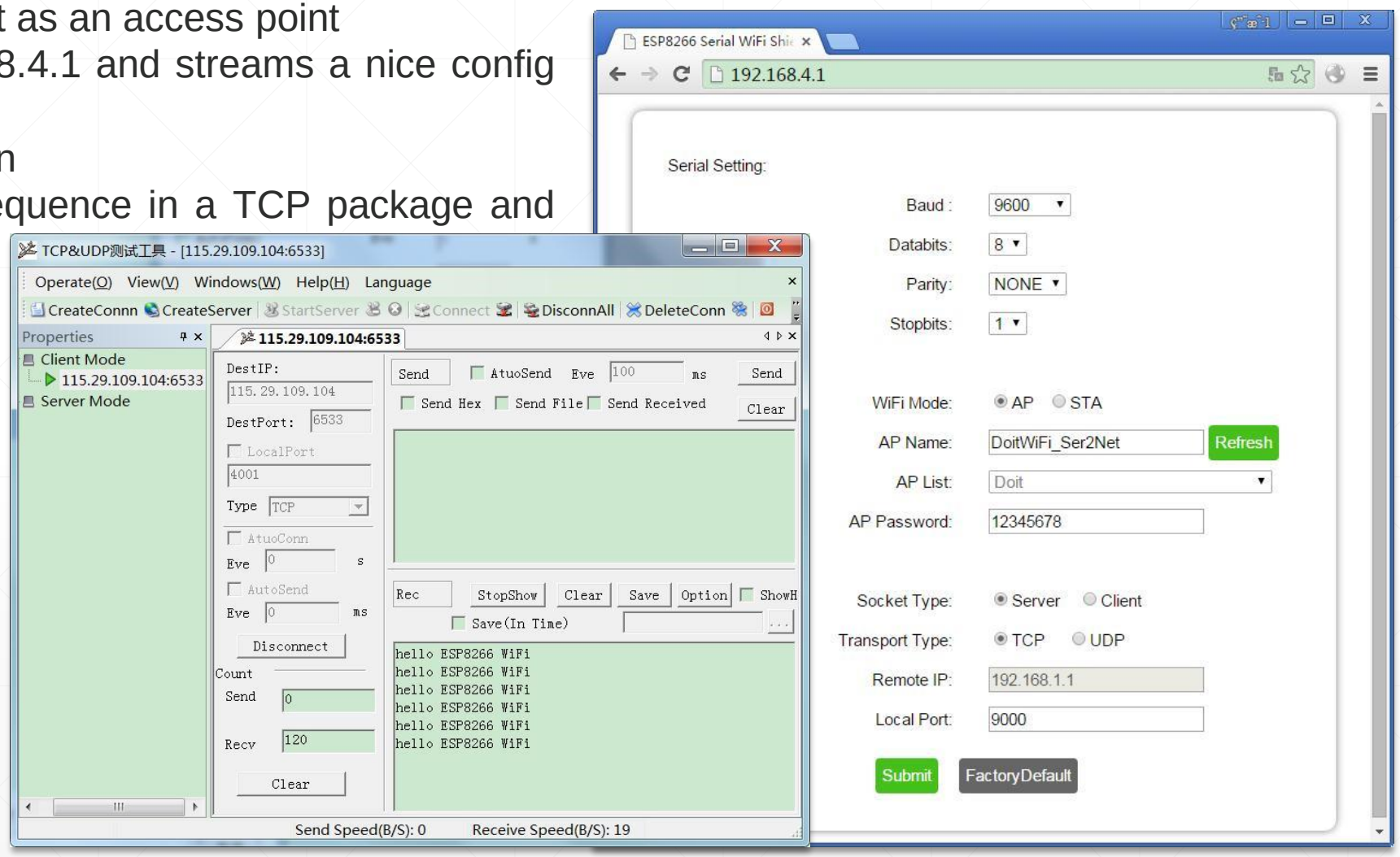
Turns out the ESP8266 came pre-flashed with the chinese **DOIT firmware**

The ESP shield

It's not a bad firmware per-se, but it does only one thing (and actually does it well):

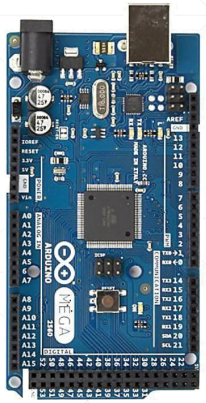
- Configures the ESP8266 to act as an access point
- Starts a webserver on 192.168.4.1 and streams a nice config page from there
- Listens on the serial connection
- Encapsules every received sequence in a TCP package and streams it on the webserver.

But that's not what we need!



The ESP shield

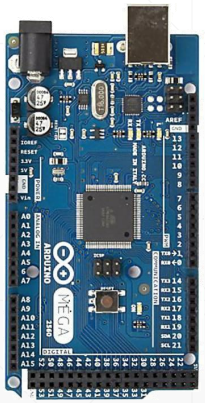
DOIT configuration



our configuration

turn right!

turn right!



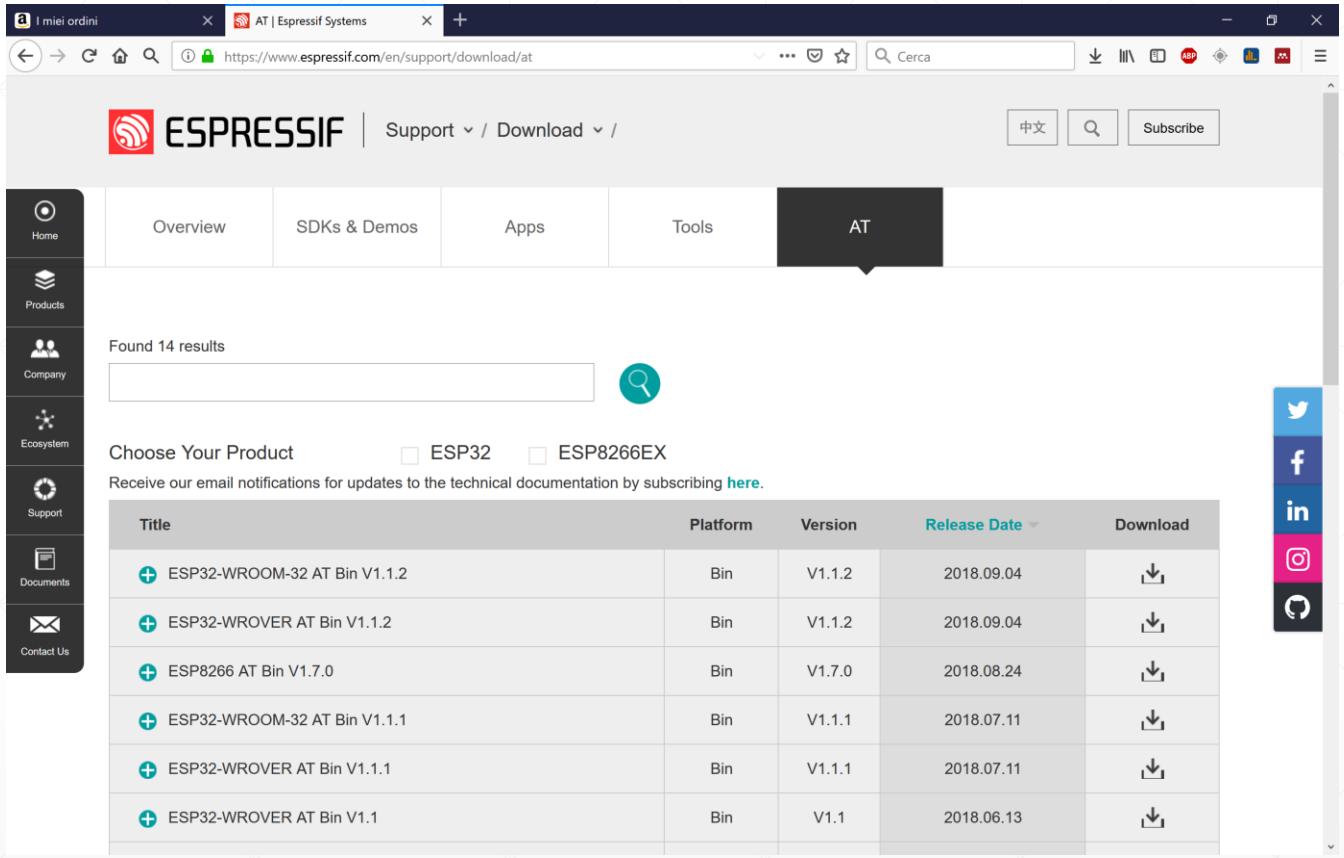
* turns right *

The ESP shield

We need to flash the AT firmware:

It can be downloaded from the Espressif website at www.espressif.com along with the **firmware flashing tool**. So far seems easy!

but...



ESP8266 AT Bin V1.7.0	Bin	V1.7.0	2018.08.24	
ESP8266 AT Bin V1.6.2	Bin	V1.6.2	2018.06.08	

How do we connect the ESP8266 to the pc?

The ESP-WROOM02 doesn't have its own **USB-UART** hardware, like the Arduino has, so it needs some kind of **USB-TTL** interface to be programmed, they're quite cheap to purchase on any online shop...

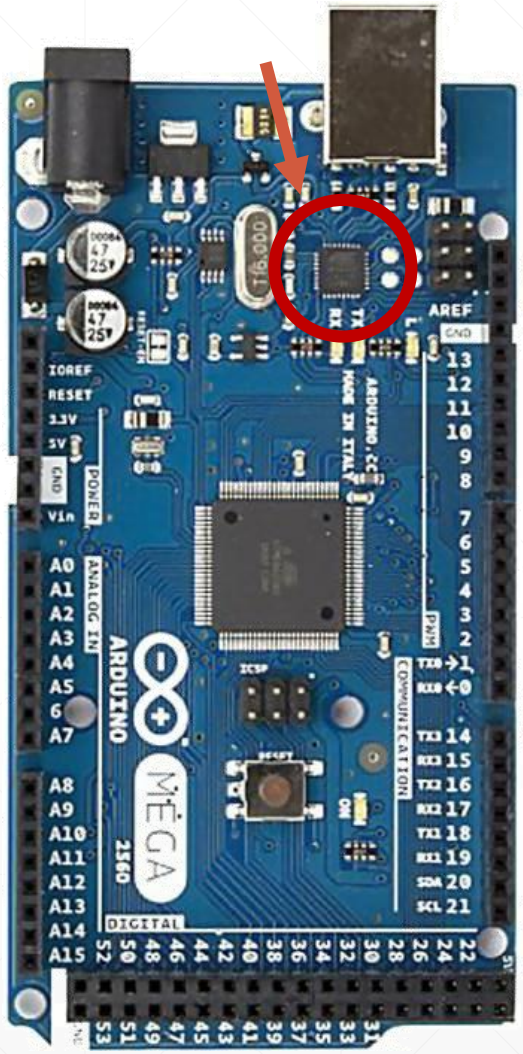
...but is there another way?

We can use what we already have: **Arduinos**!

Arduino has it's own USB-TTL interface to program the onboard **Atmel ATMEGA328P,** we can use that instead of purchasing a USB-TTL adapter.

How to use what we have?

The ESP shield



How do we connect the ESP8266 to the pc?

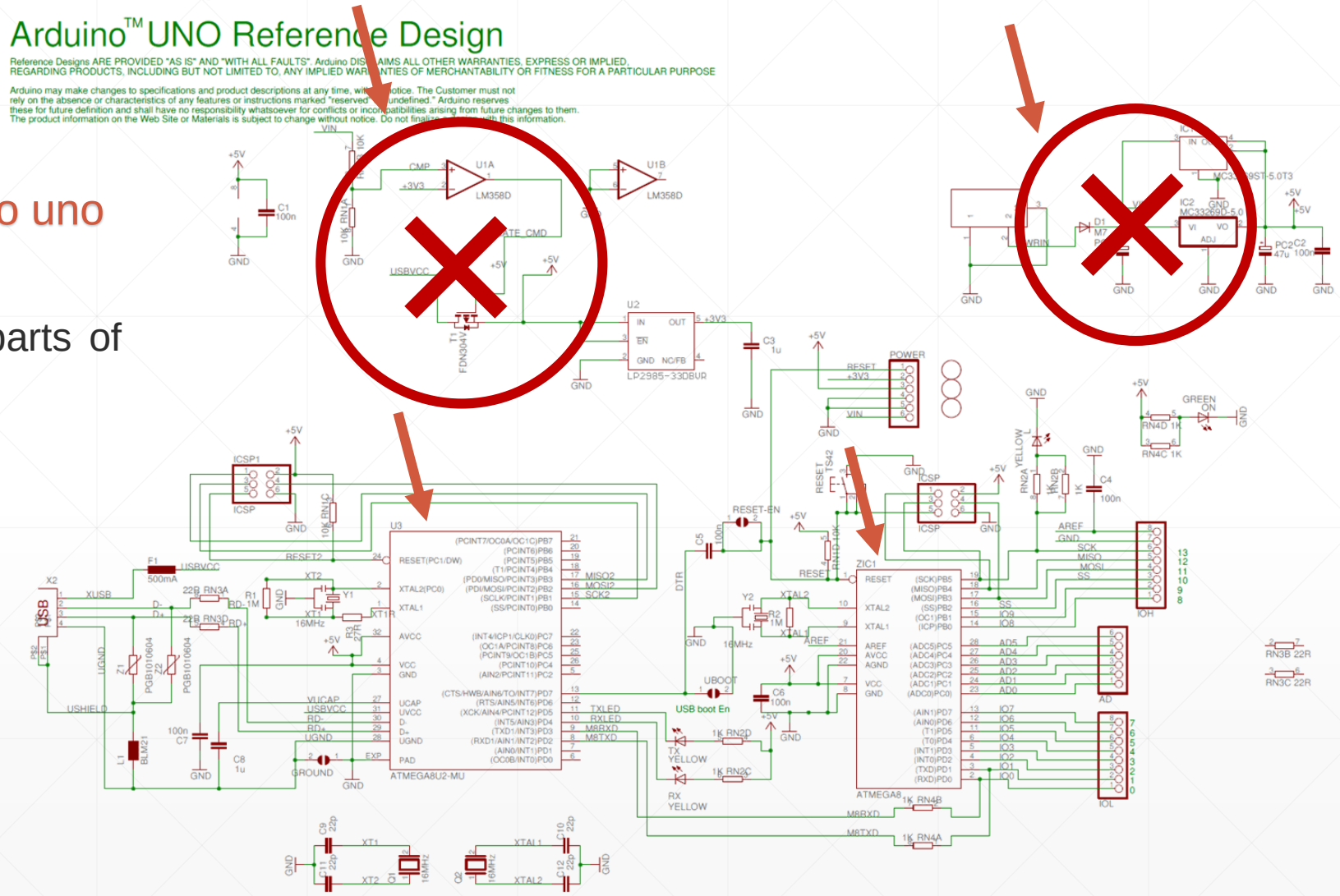
The ESP shield

Let's take a look at the Arduino uno schematics:

If we ignore the power-related parts of the interface...

We find the interesting parts:

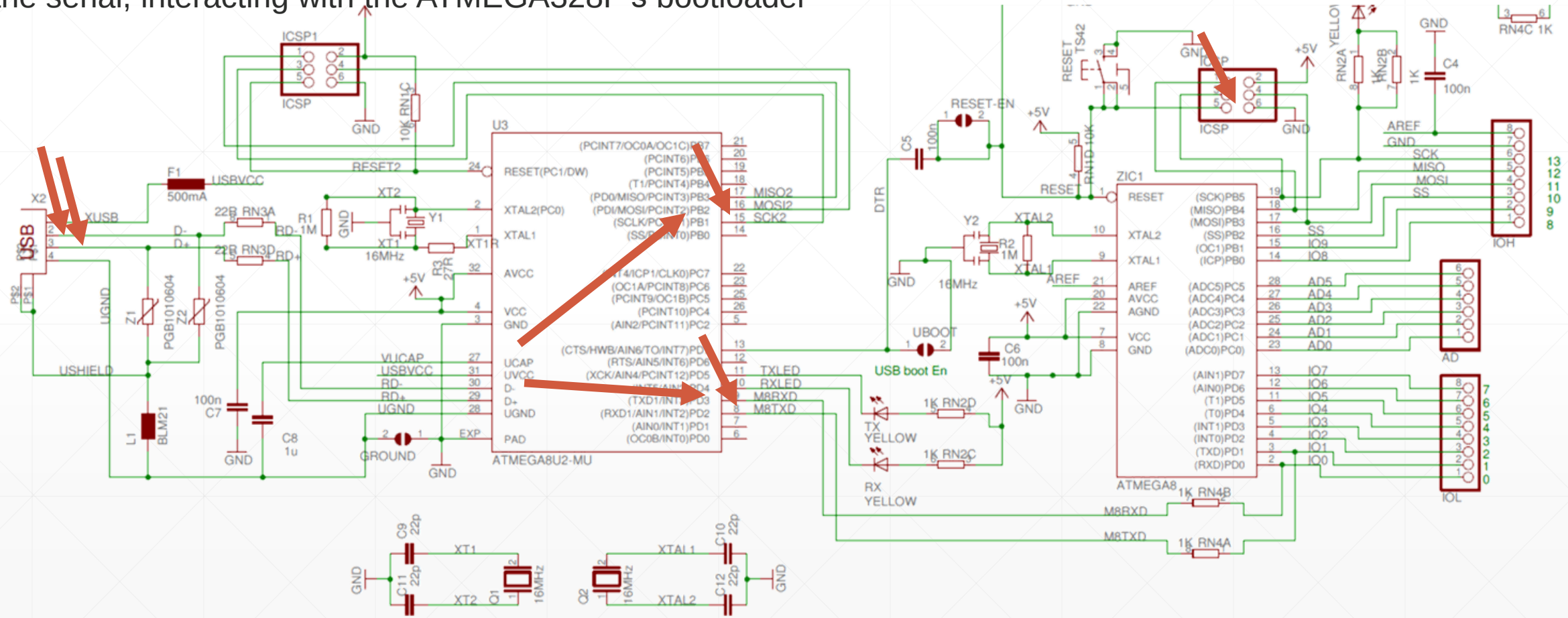
- The ATMEGA **controller**
- The USB-TTL **adapter**



The ESP shield

How do we connect the ESP8266 to the pc?

The D+ and D- differential data lines reach the USB-TTL interface:
we can then decide to program the chip talking via the **ISP** programmer or using the serial, interacting with the ATMEGA328P's bootloader



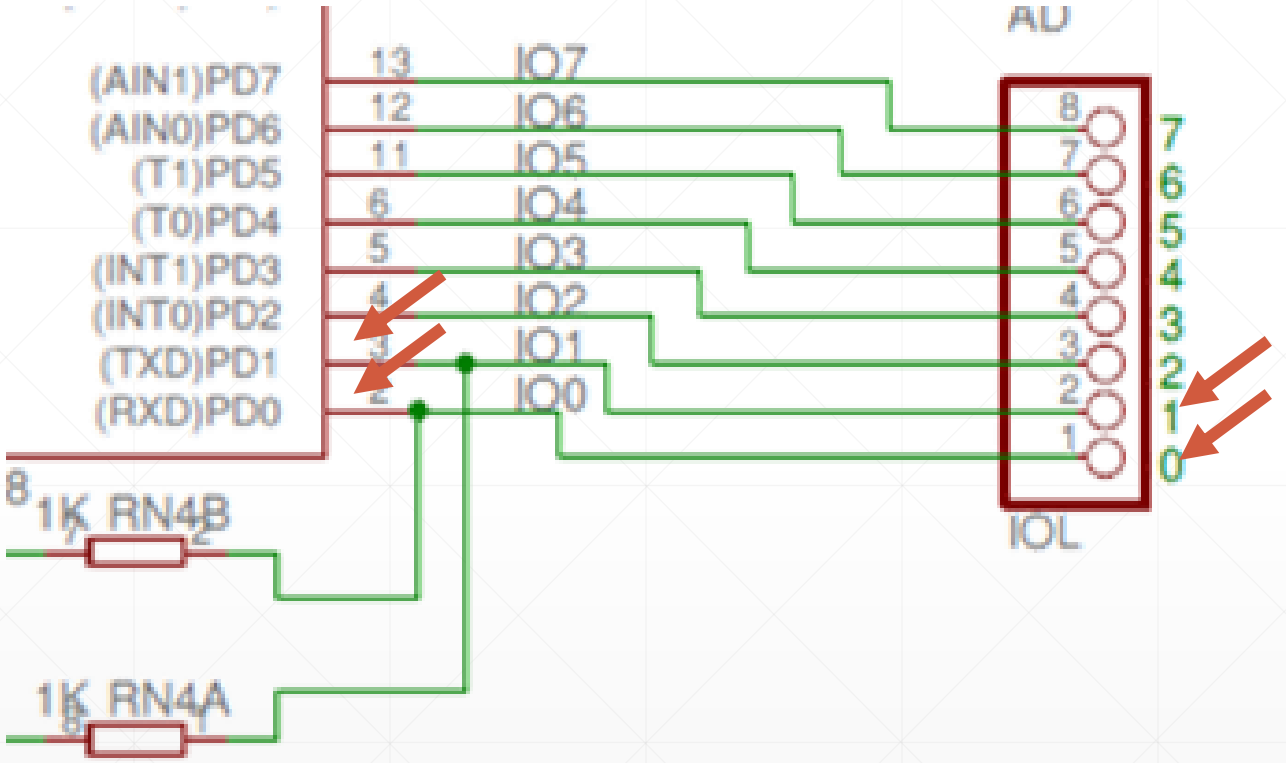
The ESP shield

How do we connect the ESP8266 to the pc?

The **PD0** and **PD1** pins of the ATMEGA328P are exposed as pin 0 and 1 on the Arduino board!

That solves part of our economical problems as we probably won't have to buy a USB-TTL adapter!

The ESP8266 can be flashed from serial:
when the **GPIO0** pin is held on **LOW(GND)** it will execute a **serial bootloader** which will receive the programs we want to flash from the **Espressif Flash Tool.**



How do we connect the ESP8266 to the pc?

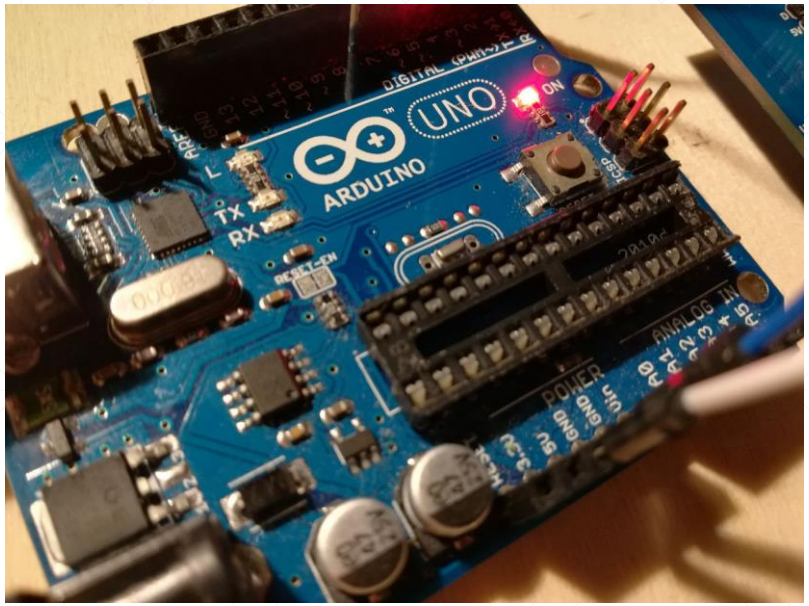
We now need a practical method to adapt the USB-TTL interface to our needs:

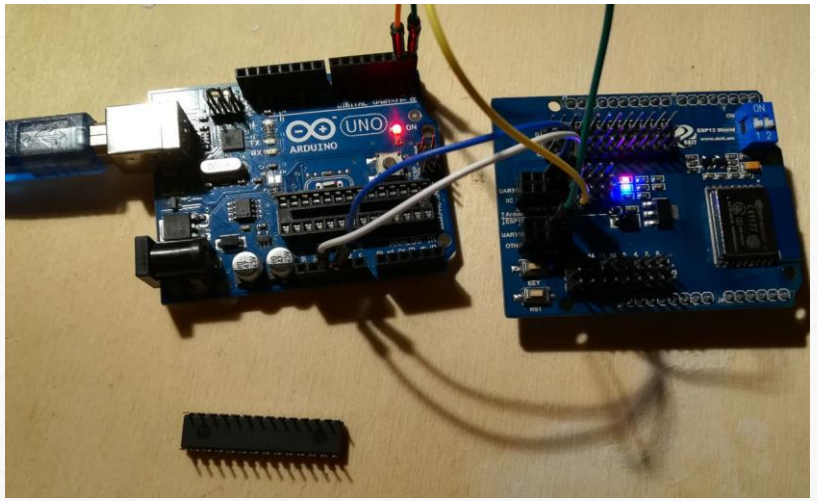
Method 1: The Literal No-brainer

- Remove the **ATMEGA328P** from its socket
- Power up the ESP8266 connecting the 3.3V line to the Arduino 5V through the voltage regulator as we did before
- Connect the RX and TX pins of the ESP8266 to the Arduino
  - We're not trying to communicate with a the arduino but are trying to talk directly to the USB-TTL chip, the right connection scheme now is **RX0 -> RX, TX0 -> TX**!

That looks <u>good</u> if we have a socketed controller like in the Uno case, but what about the Mega?

The ESP shield





The ESP shield

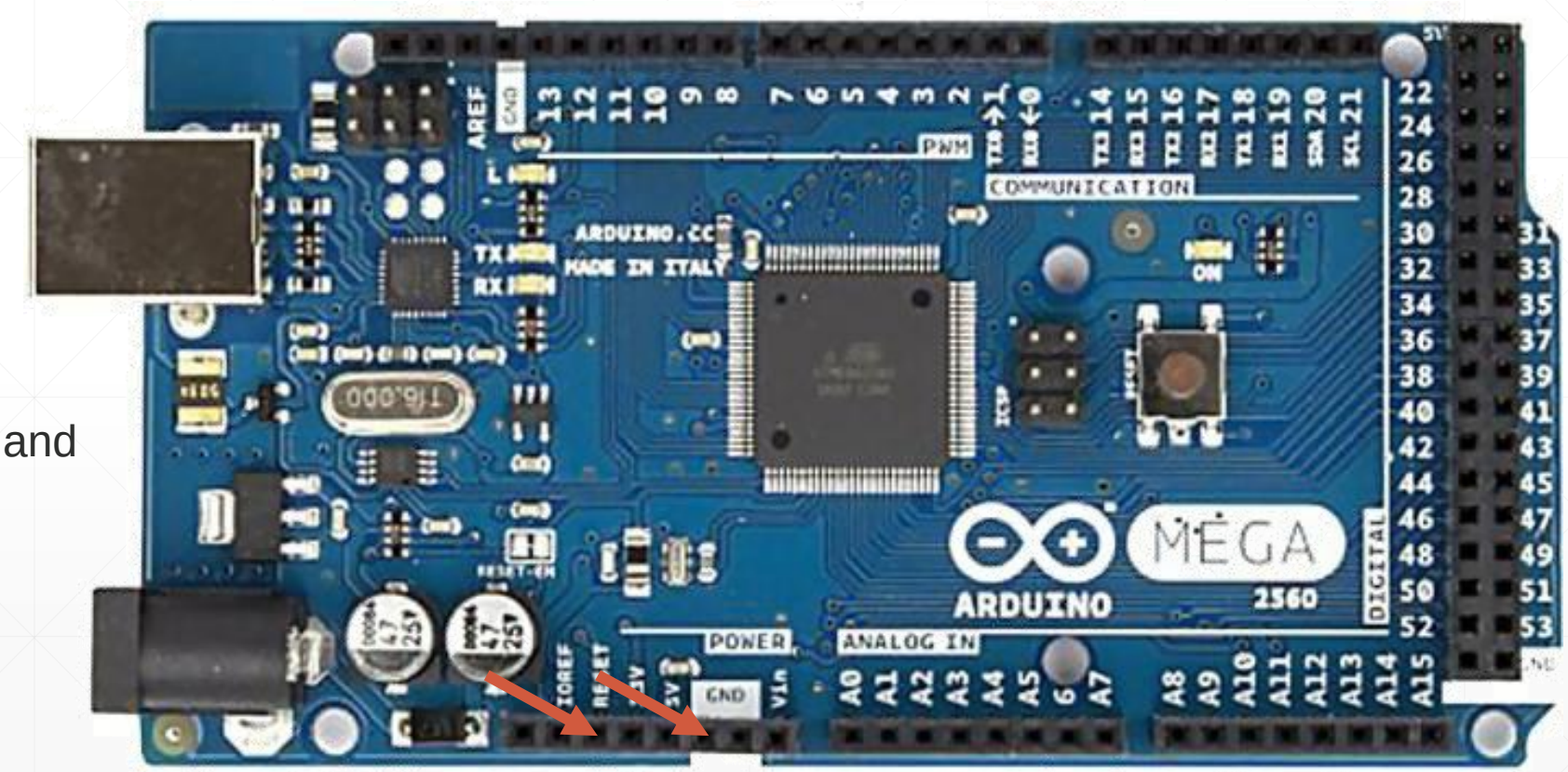
How do we connect the ESP8266 to the pc?

But the Mega has a soldered **ATMEGA2560** controller that might be a little bit impractical to remove.

Method 2: The Sleeping Beauty

If we can't remove the controller itself we can just silence it when we need to speak with the ESP8266 by grounding it's **RESET** signal.

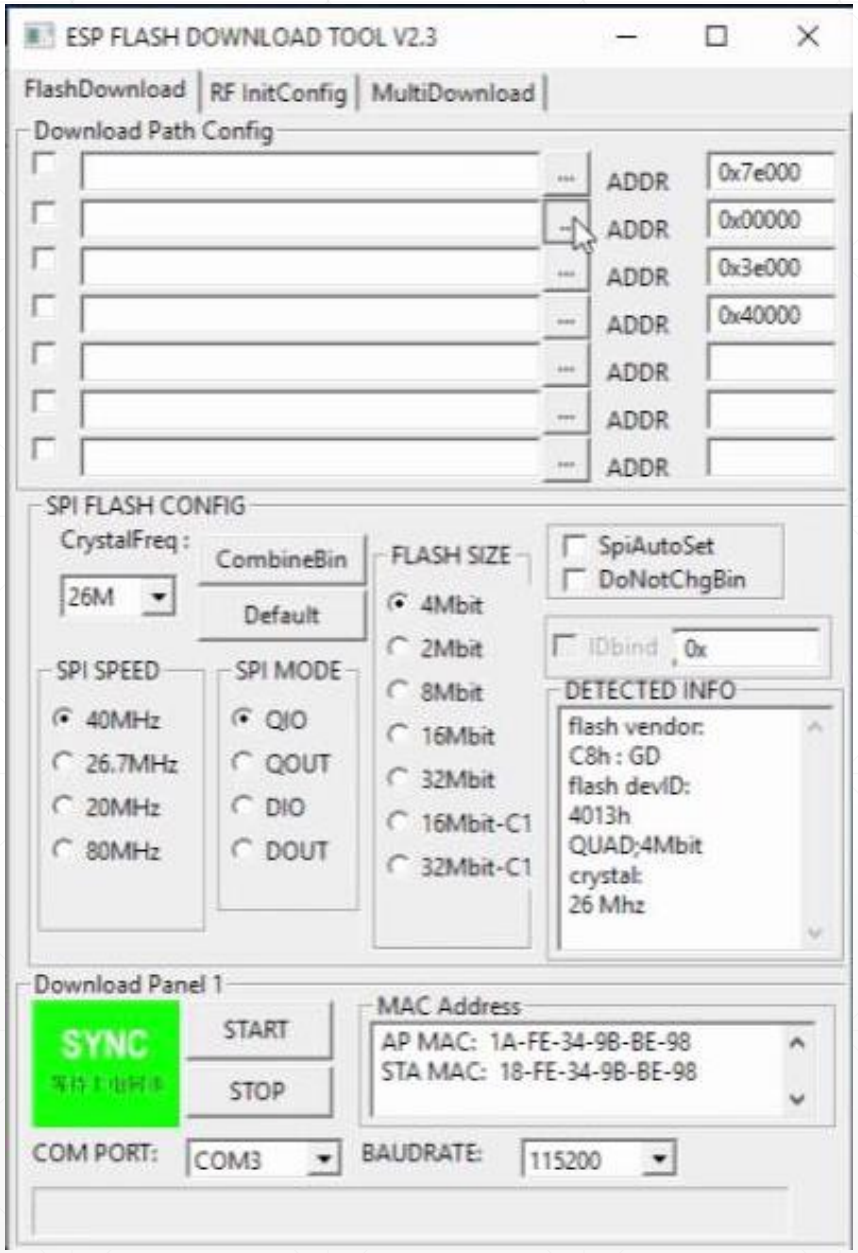
- Run a jumper wire between **RESET** and **GND**
- Proceed exactly as Method 1

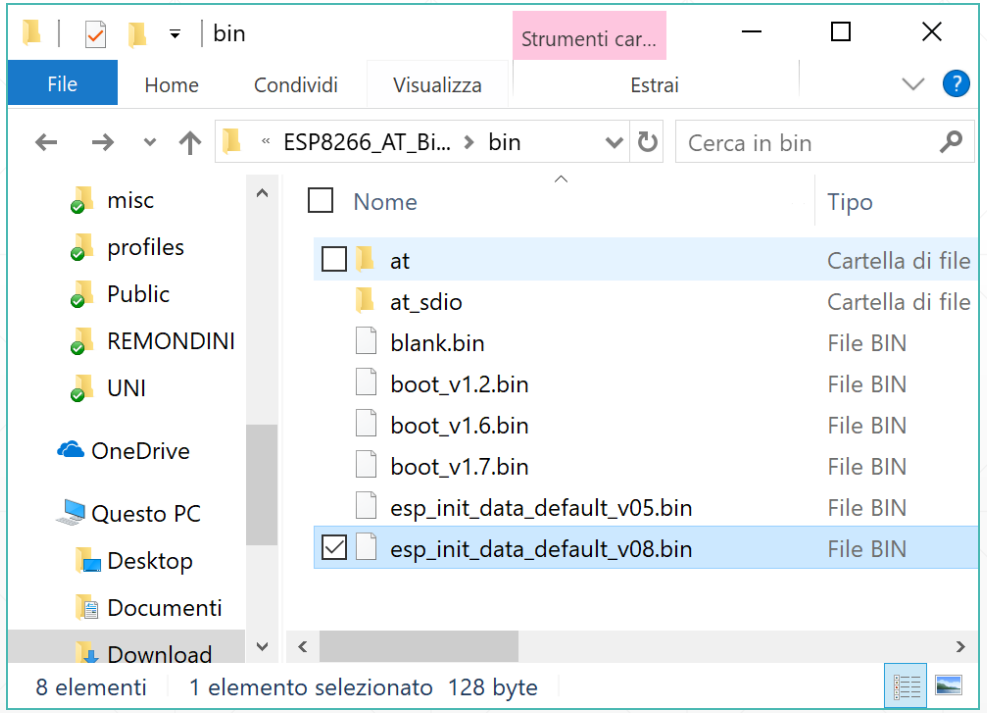


The ESP shield

Now we can finally proceed to flash the AT firmware on the **ESP-WROOM02.**

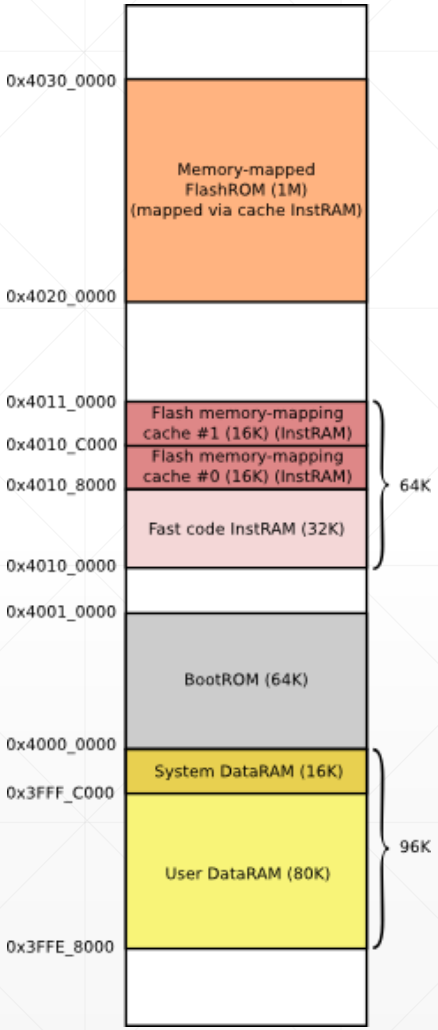
The AT firmware comes with a bunch of files to be flashed at different positions in the memory map





The ESP shield

The documentation by Espressif helps to choose which .bin is flashed to which memory address as memory can be organized differently from package to package

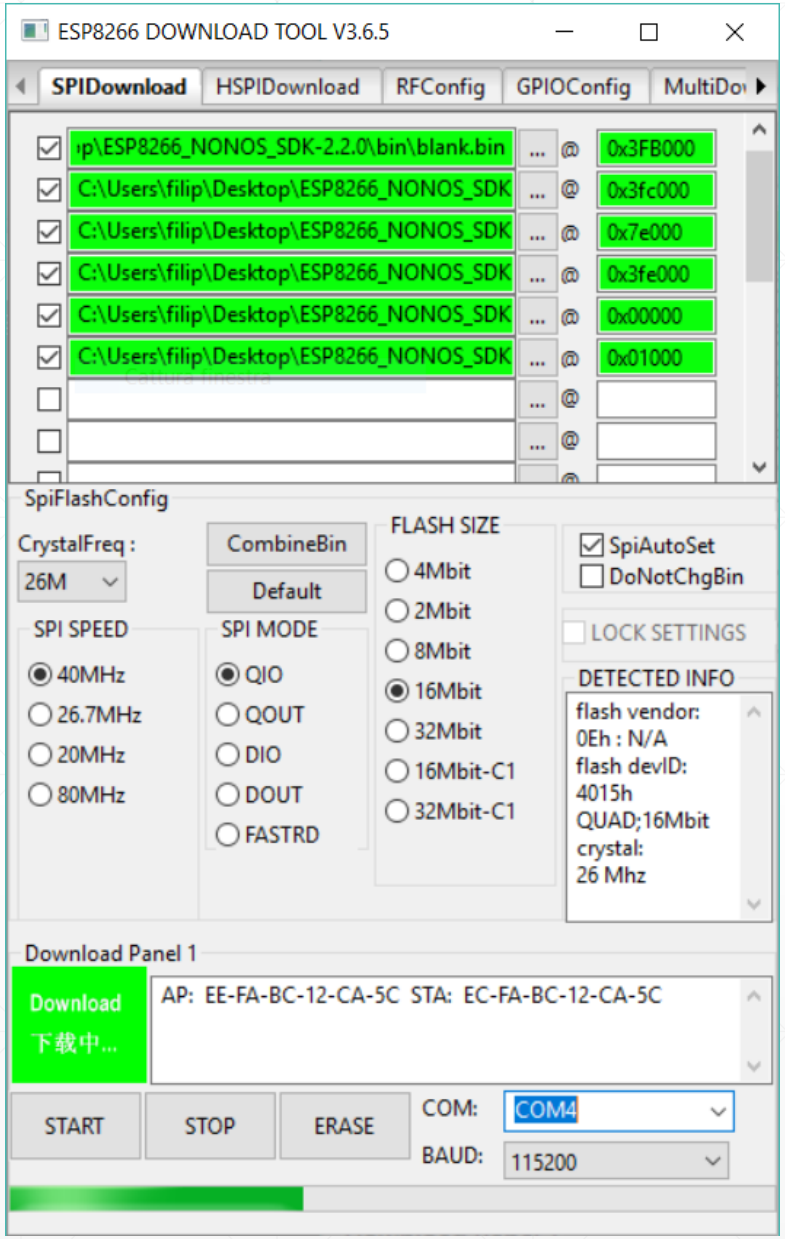


1.2.1. 16 Mbit Flash, Map: 1024 KB + 1024 KB

Use Espressif Flash download tool and select flash size: 16 Mbit-C1.

BIN	Address	Description
blank.bin	0x1FB000	Initializes RF_CAL parameter area.
esp_init_data_default.bin	0x1FC000	Stores default RF parameter values, has to be downloaded into flash at least once. If the RF_CAL parameter area is initialized, this bin has to be downloaded too.
blank.bin	0xFE000	Initializes Flash user parameter area, more details in ***Appendix***.
blank.bin	0x1FE000	Initializes Flash system parameter area, more details in ***Appendix***.
boot.bin	0x00000	In ***/bin/at***.
user1.2048.new.5.bin	0x01000	In ***/bin/at/1024+1024***.

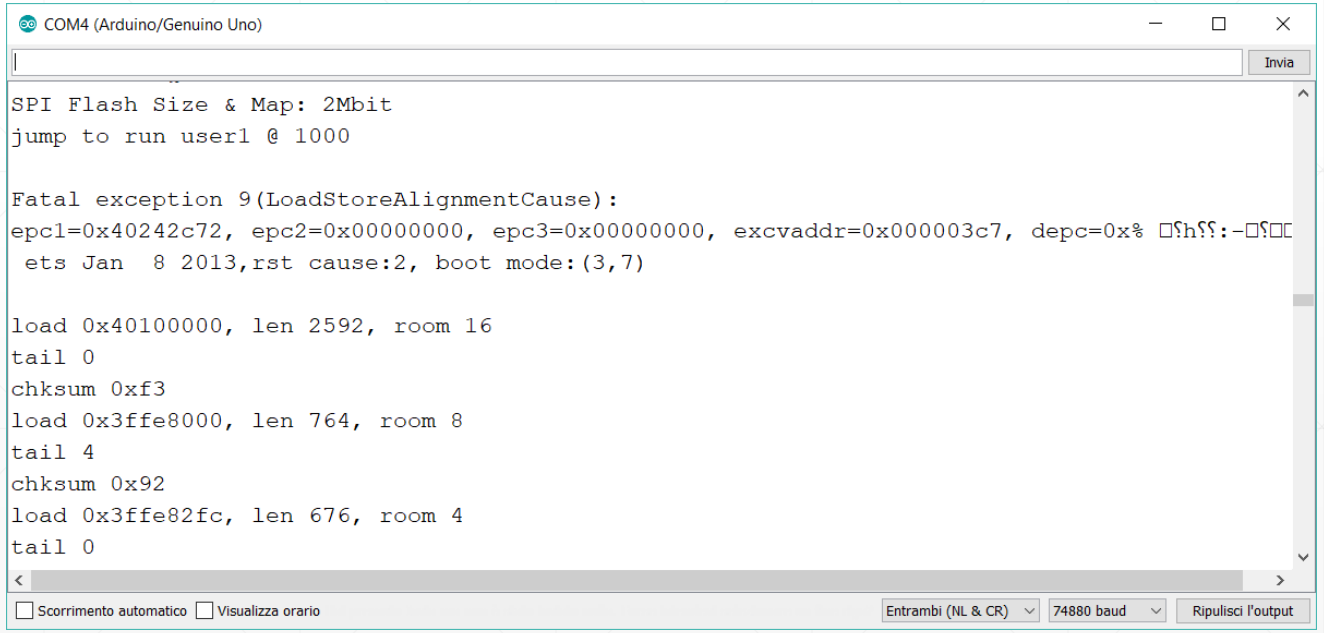
The ESP shield



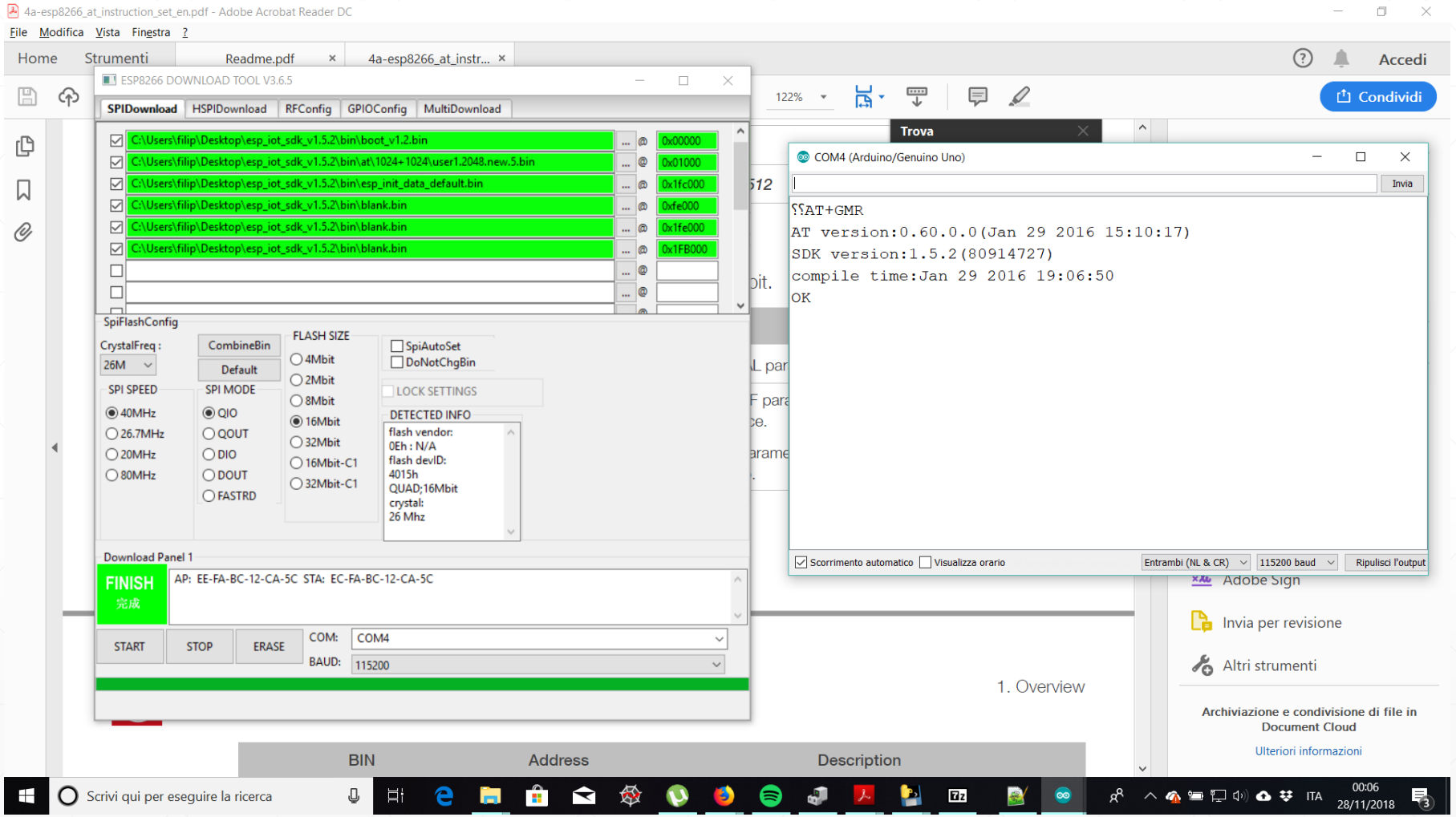
Just define your memory map flashing schedule and a few parameters like

- the communication port you're using
- the baud rate
- various characteristics of the ESP8266 package can be detected automatically by the tool

it can take a while to figure out the correct settings and the correct firmware version to use for your specific chip.



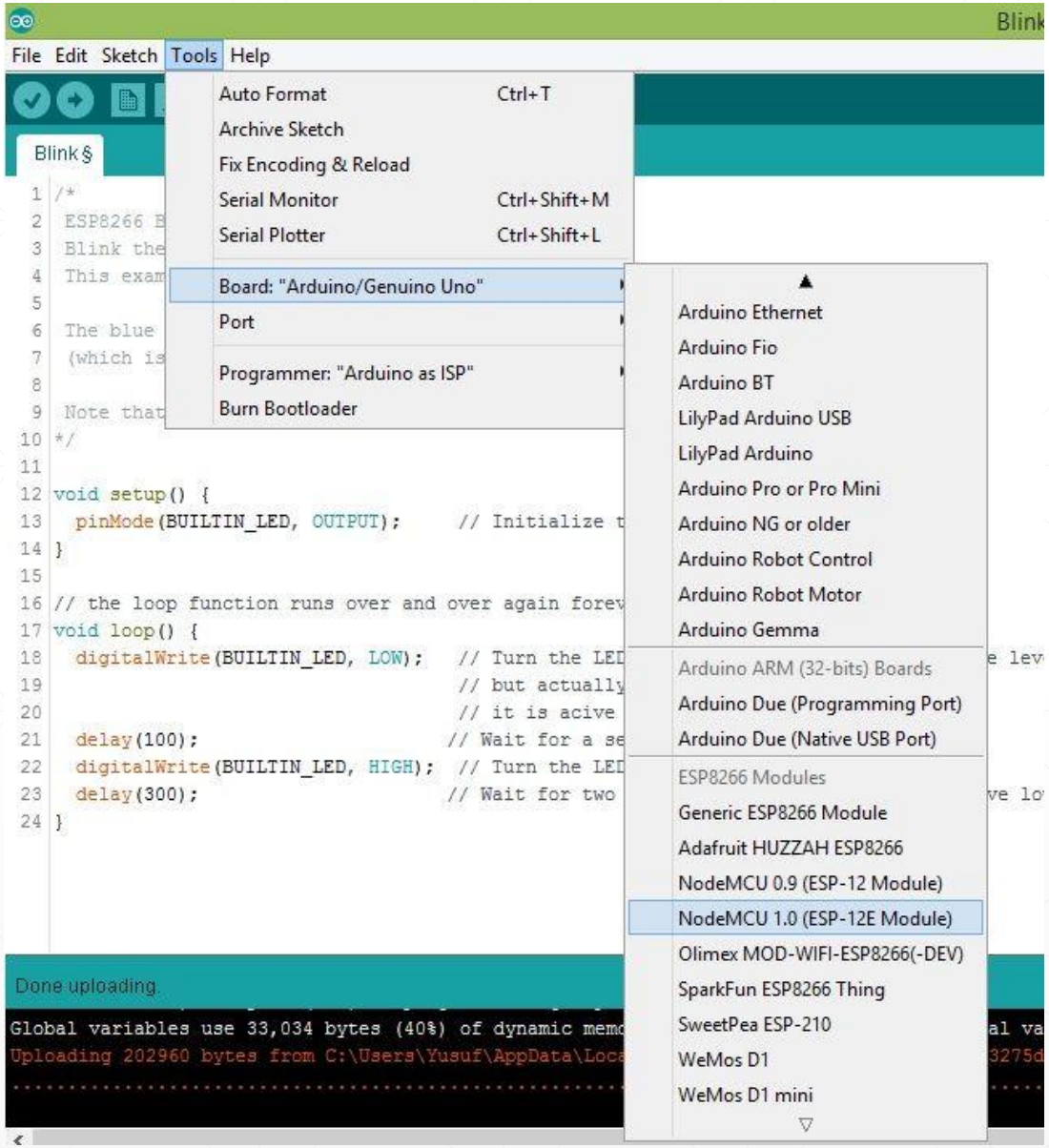
The ESP shield



If the firmware is flashed correctly, when sending the command **AT+GMR?** on serial the ESP8266 should reply with the current firmware version.

Using serial commands you can change settings without needing to reflash a program everytime.

The ESP shield



We can now develop a program for the **ESP8266** module directly in the Arduino IDE! (it requires some extensions available on the Arduino website) using the same C++ language you use for normal Arduino programming:

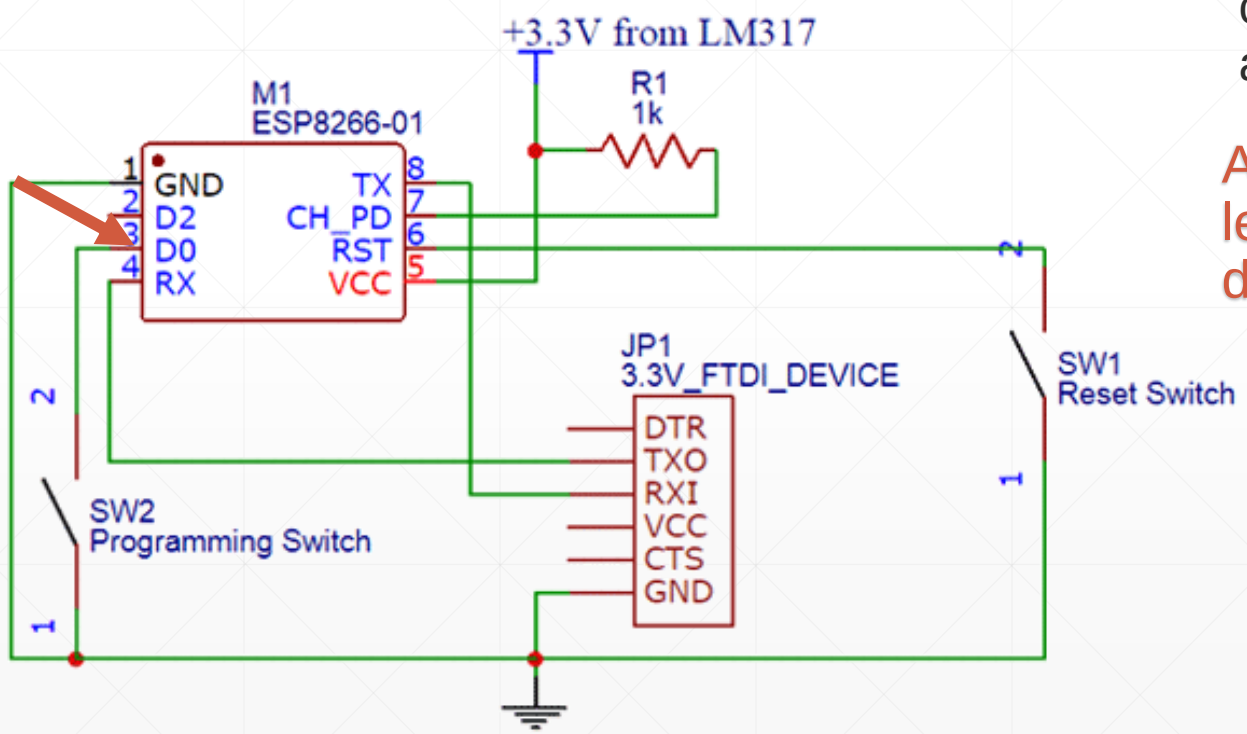
We set up a webserver wich streams a simple webpage which we use to control the car.

To flash your program you just need the same setup we used to flash the firmware (Arduino IDE only flashes a memory region allocated to program execution).

The ESP shield

Of course, when everything works, something unexpected happens.

And we fried everything



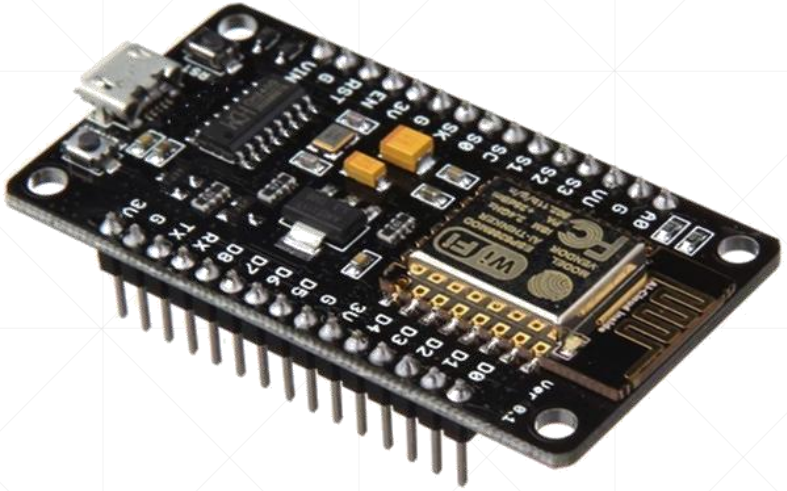
To put the ESP8266 in **programming mode** you need to short the **D0** pin to ground: if the chip is powered in this configuration it will start a **bootloader** instead of the flashed program.

After repeating the same operation at least 10 times, we accidentally shorted a different pin to ground.

the VCC.

This short circuit fried the poor controller…

The ESP shield

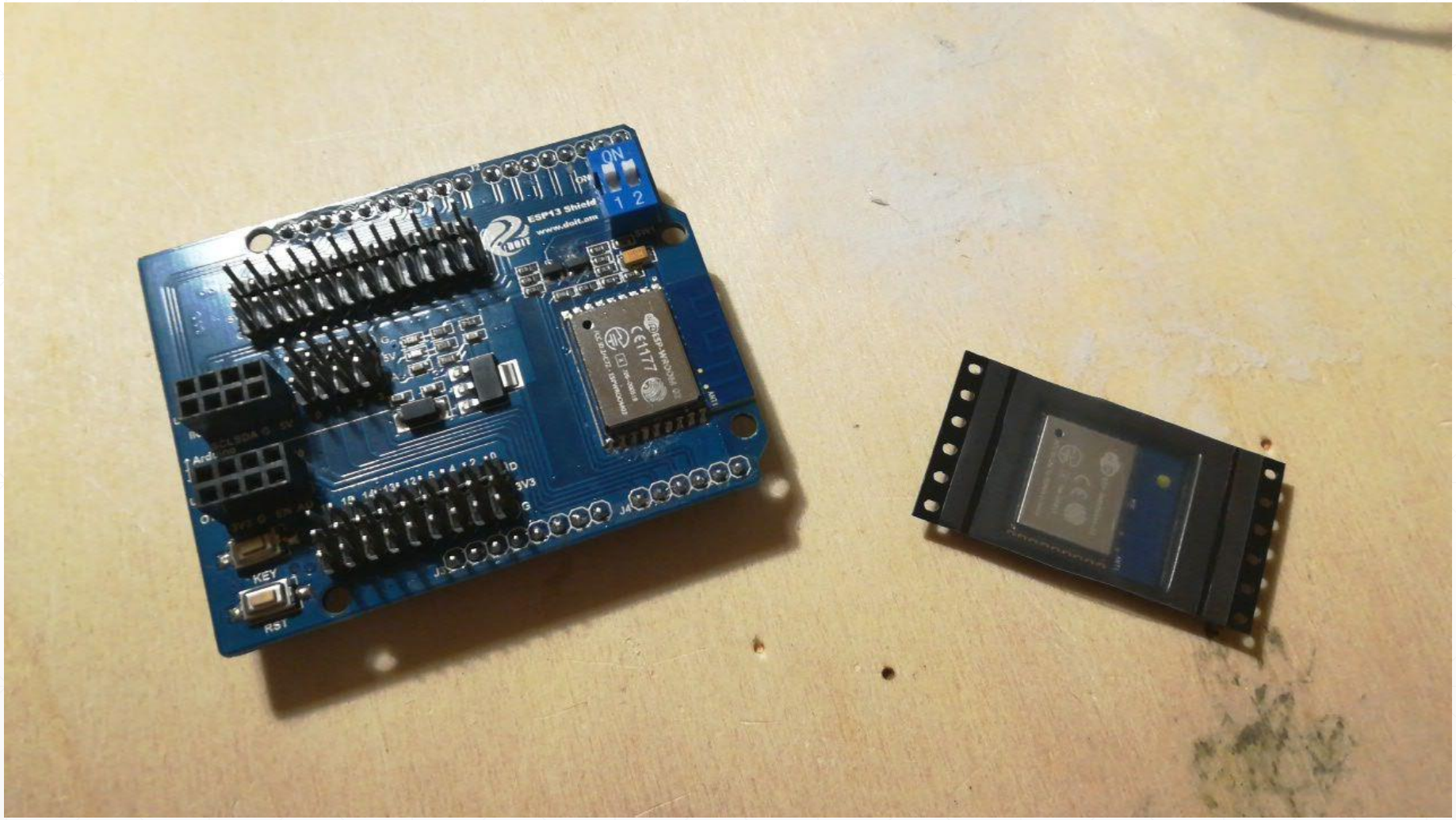


- **Option #1: trash it and replace it**
  - and change it with a more reasonable board, like the NodeMCU, that has its own USB UART hardware.
- **Option #2: the hard way**
  - Order just the ESP8266 controller, without any additional hardware, and substitute it to the fried one.

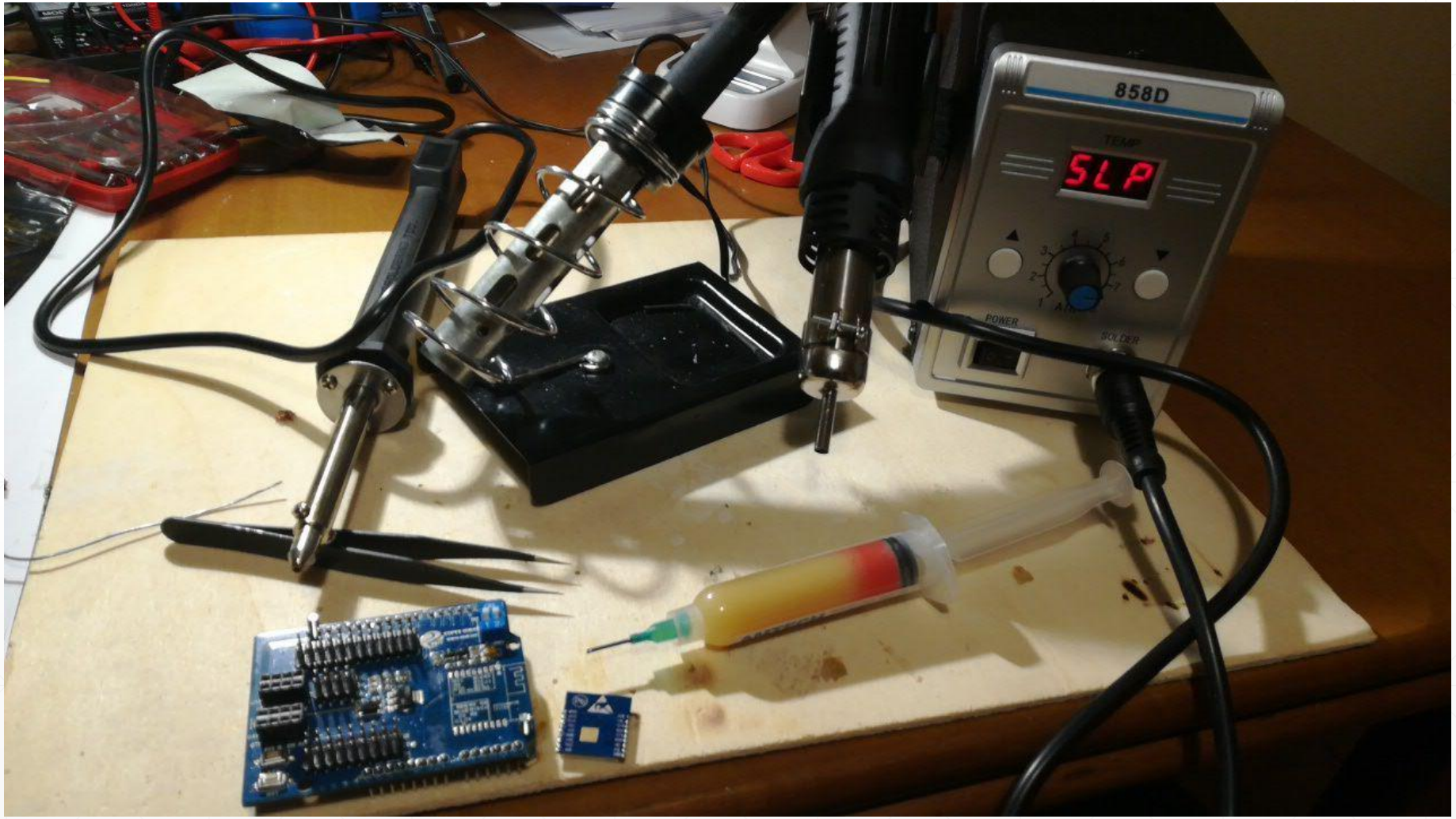
guess which way we went...

(we actually ordered both, but it was just a week before Christmas and the NodeMCU parcel was **LOST,** until a day after we took the hard way)

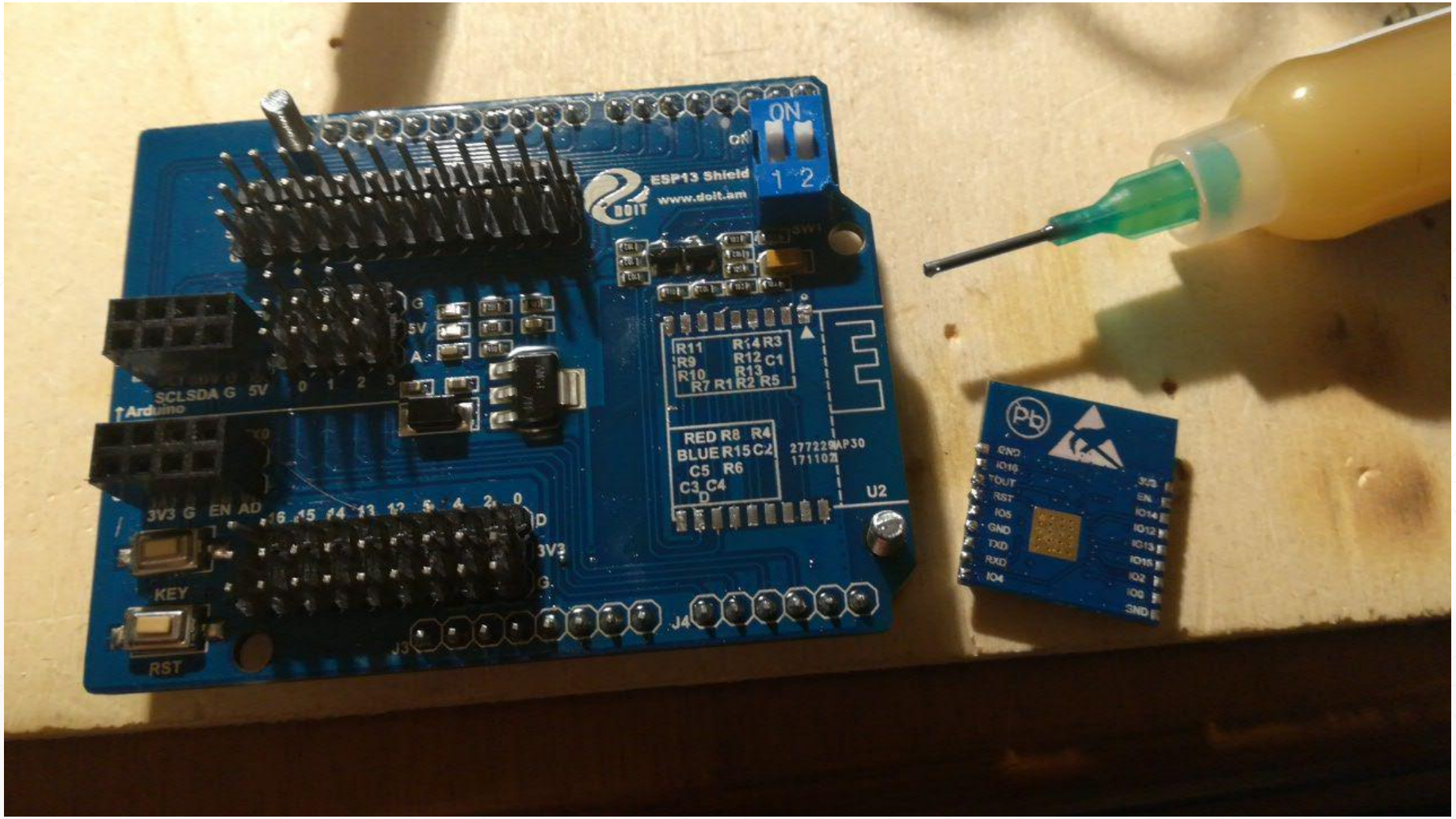
The ESP shield



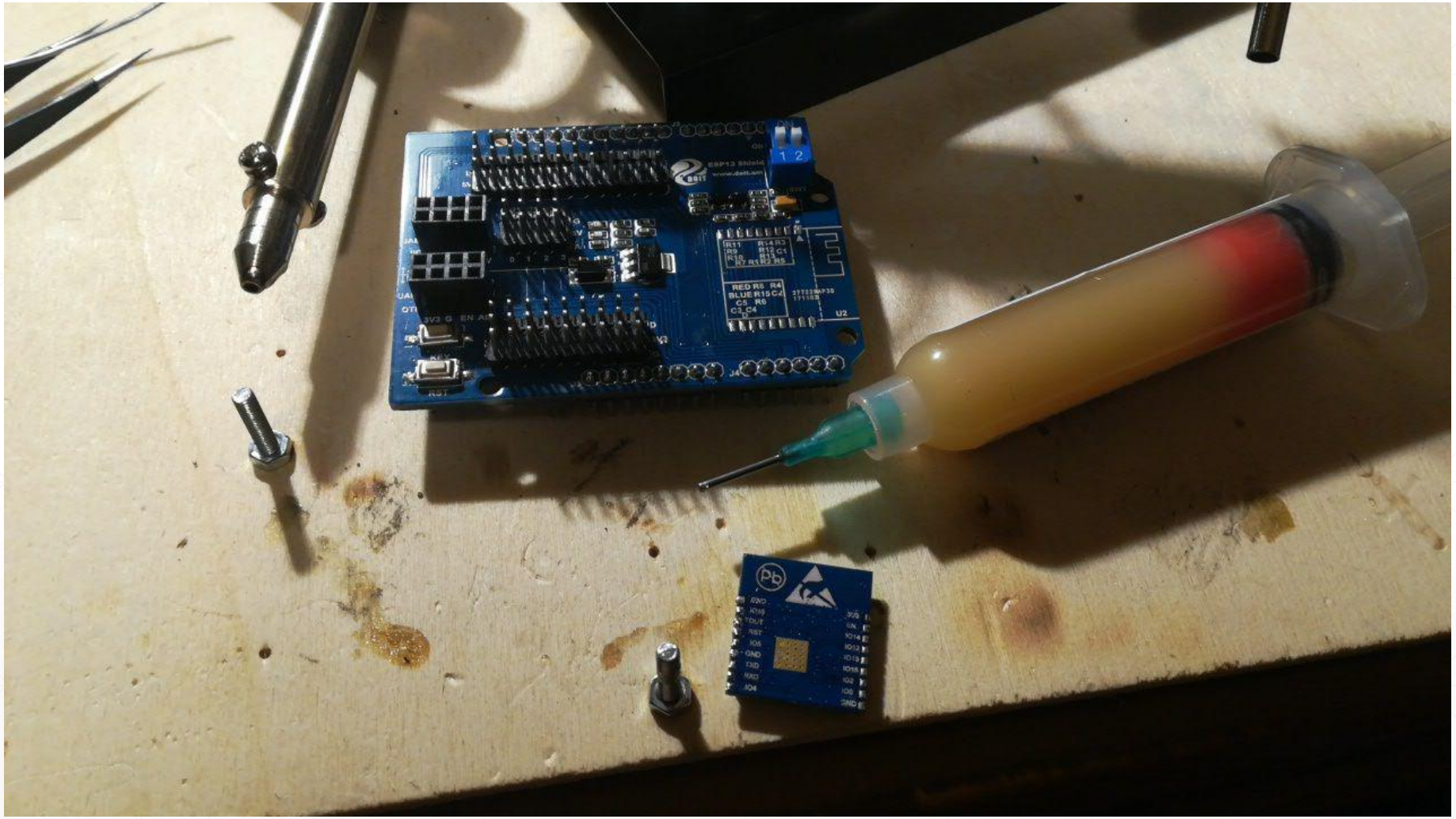
The ESP shield



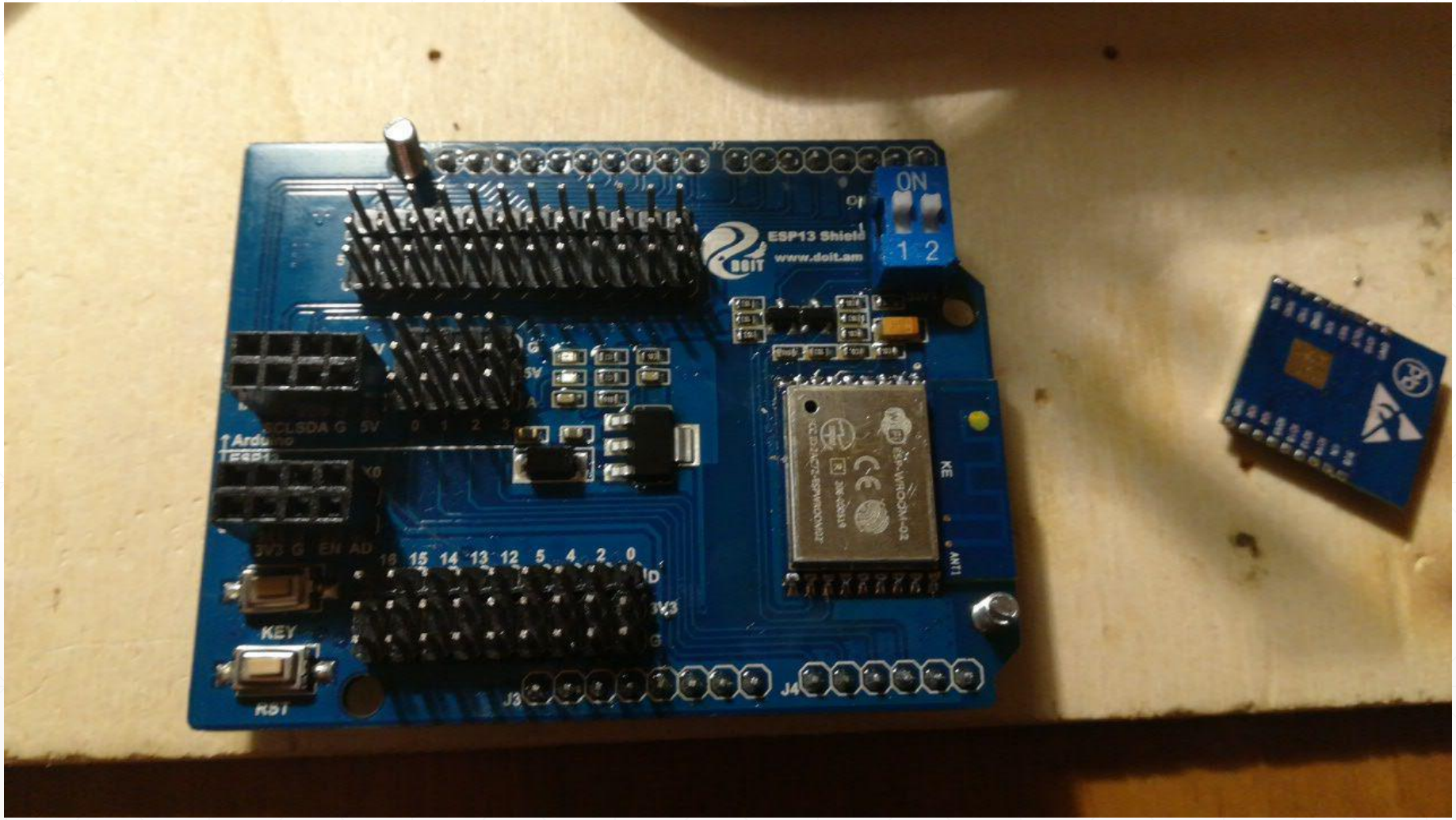
The ESP shield



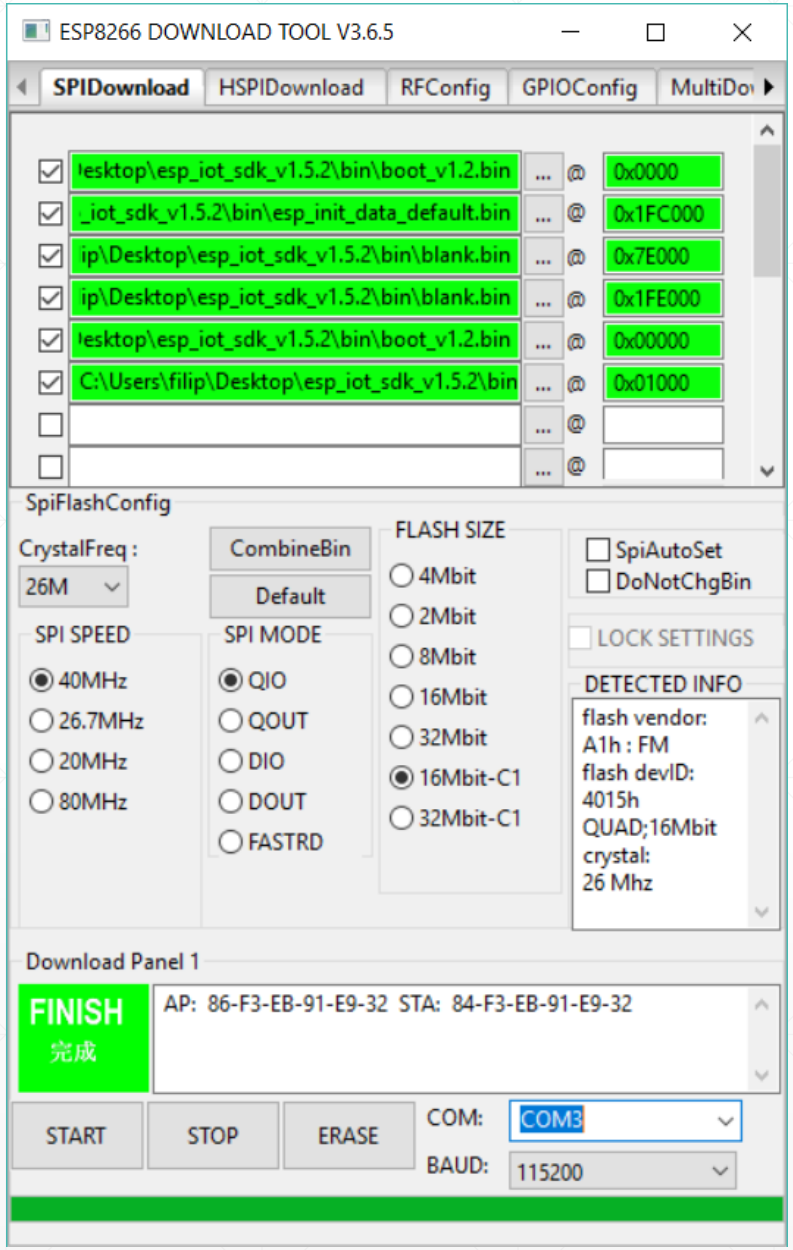
The ESP shield



The ESP shield



The ESP shield



We had to slightly change the flashing settings because the new chip was a **different revision** (took some trial and error to figure out)

But it all ended well!

Putting everything together:

The Software

Filippo Castelli
filippocastelli

Carlo Emilio Montanari
carlidel

`https://github.com/filippocastelli/uolli`

The Software

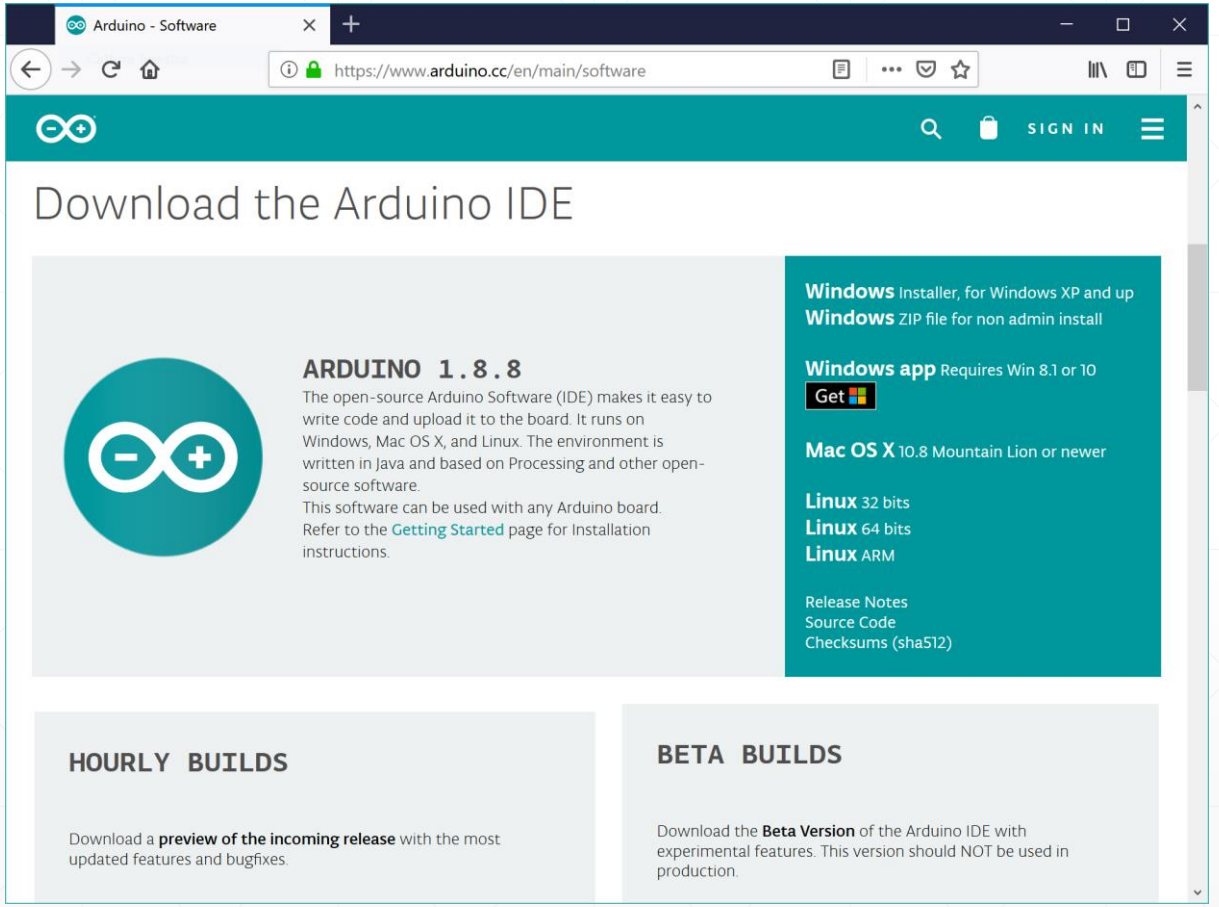
We tried to keep things **as simple as reasonably possible**, so we decided to cut down all the software paraphernalia to only a single tool:

The **`Arduino IDE`**

The `Arduino IDE` can be found on

`www.arduino.cc/en/main/software`

pre-compiled for a variety of platforms and it's the ideally the only thing you need to start programming on the Arduino platform.

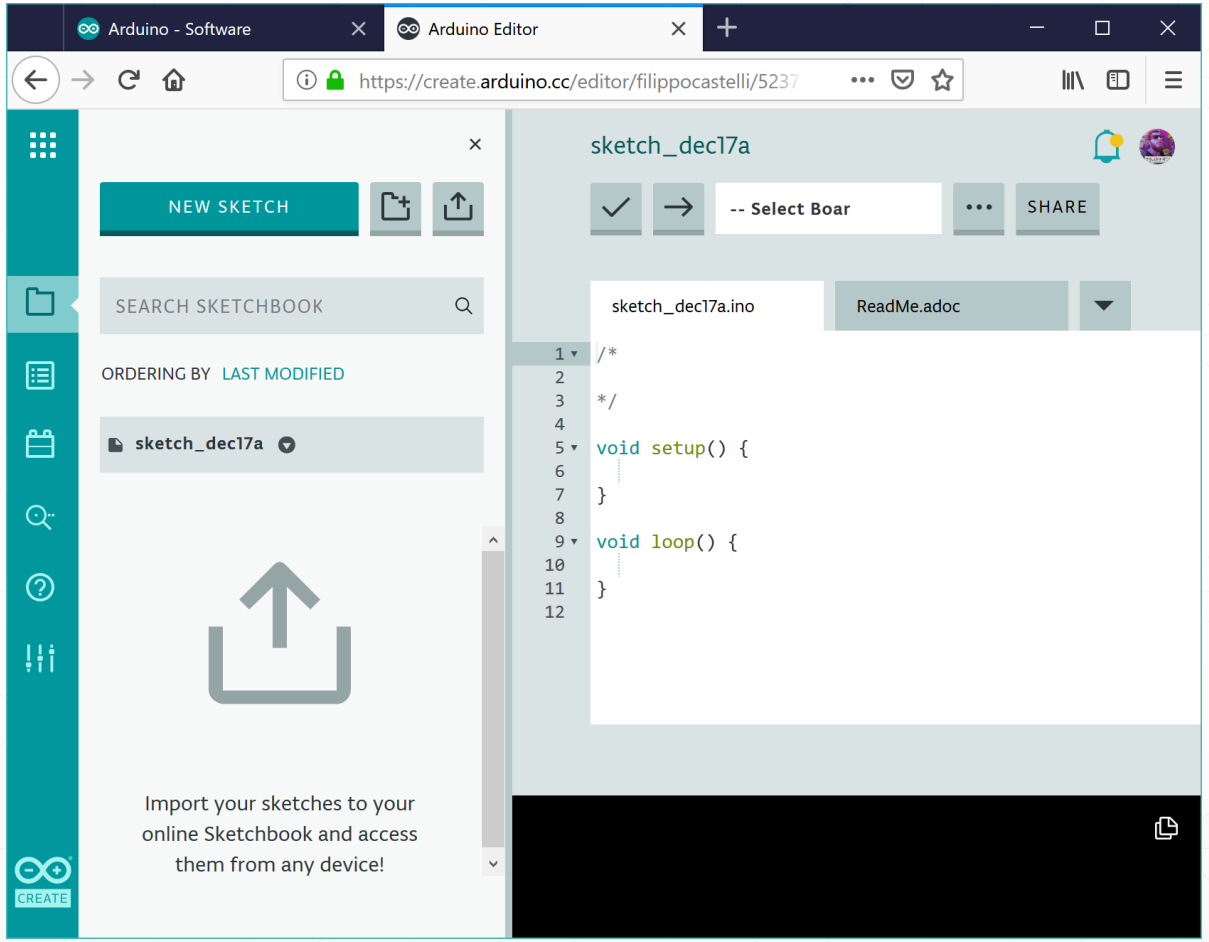


The Software

You don’t even need to install it to start coding!
You can use a web-based **Arduino Create! IDE** available on the Arduino webpage.

Arduino Create!

https://create.arduino.cc/editor

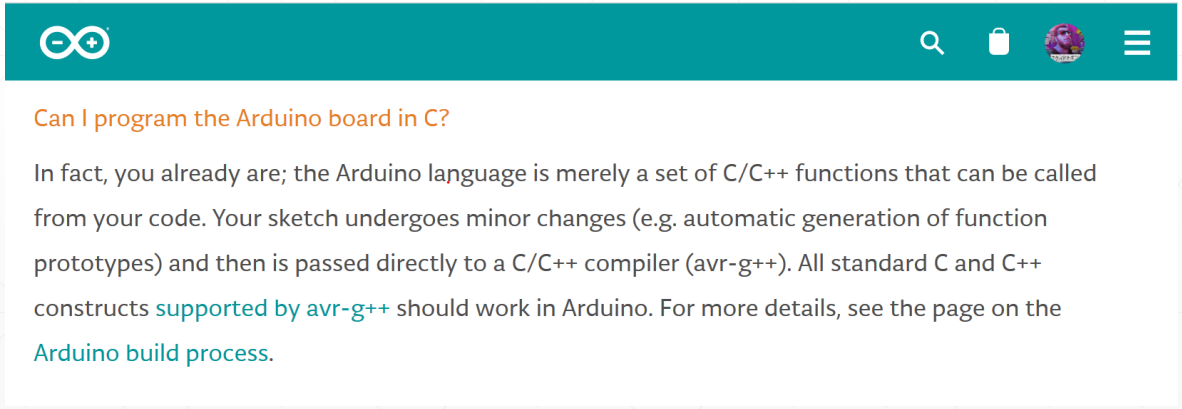


The Software

Anatomy of and Arduino Sketch:

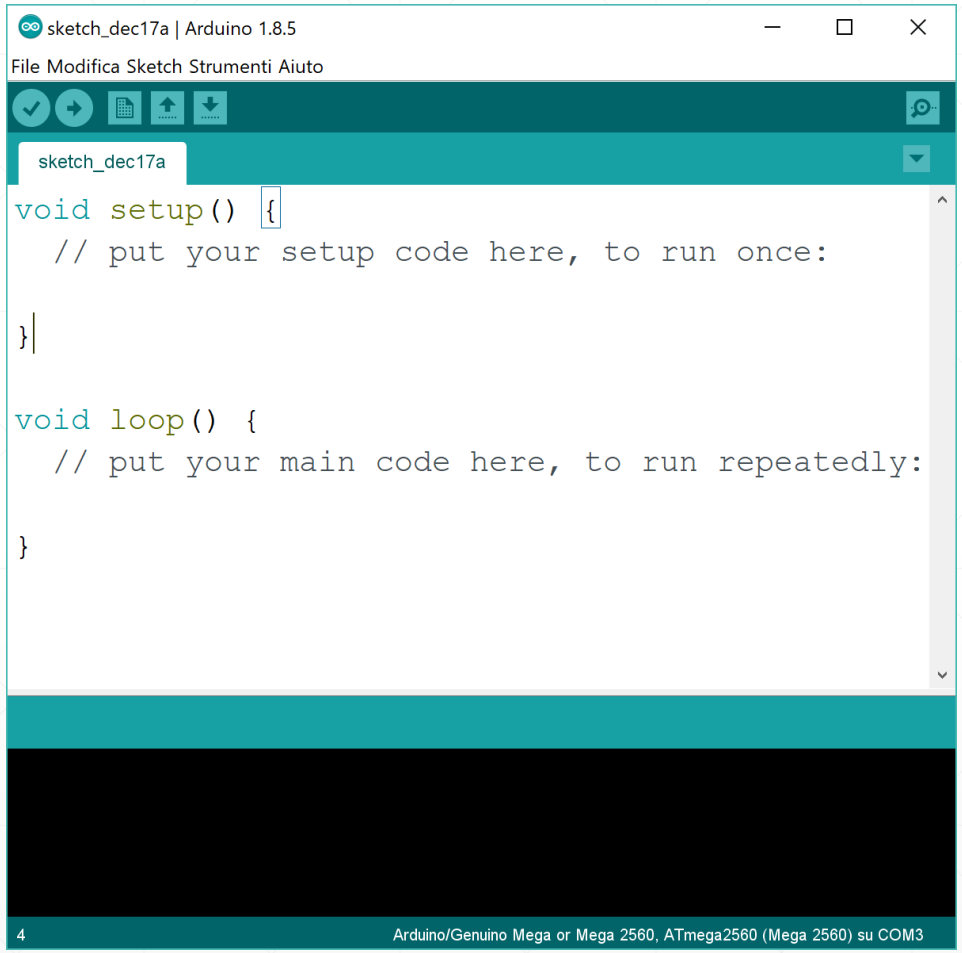
Arduino can be programmed in a special **C / C++ dialect** that requires absolutely **0 learning effort** if you've already programmed in C/C++

As a matter of fact every time that you compile an **Arduino Sketch** it's actually translated to C/C++ code and compiled using the **AVR** version of the g++ compiler with the compiler we're all familiar with, called **avr-g++**



Can I program the Arduino board in C?

In fact, you already are; the Arduino language is merely a set of C/C++ functions that can be called from your code. Your sketch undergoes minor changes (e.g. automatic generation of function prototypes) and then is passed directly to a C/C++ compiler (avr-g++). All standard C and C++ constructs supported by avr-g++ should work in Arduino. For more details, see the page on the Arduino build process.



https://www.arduino.cc/en/tutorial/sketch

The Software

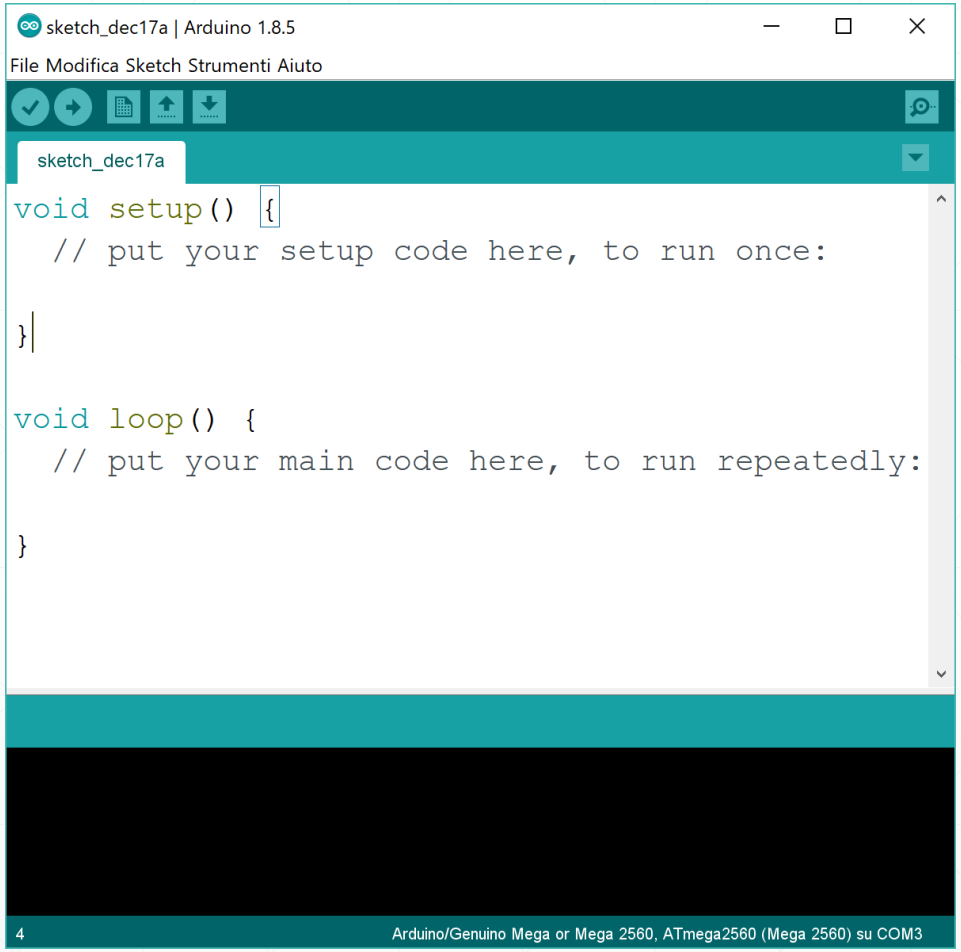
Anatomy of and Arduino Sketch:

Any sketch has to implement at least two functions:

- the `setup()` function:
  - I's the first part of the program and it's called only once when the Arduino wakes up.
- the `loop()` function:
  - as the name suggests it's looped over and over until the thermodynamic death of the universe

You may want to use `setup()` for setting up all you need in terms of IO pin settings and then run the *actual* program in `loop()`

`https://www.arduino.cc/en/tutorial/sketch`

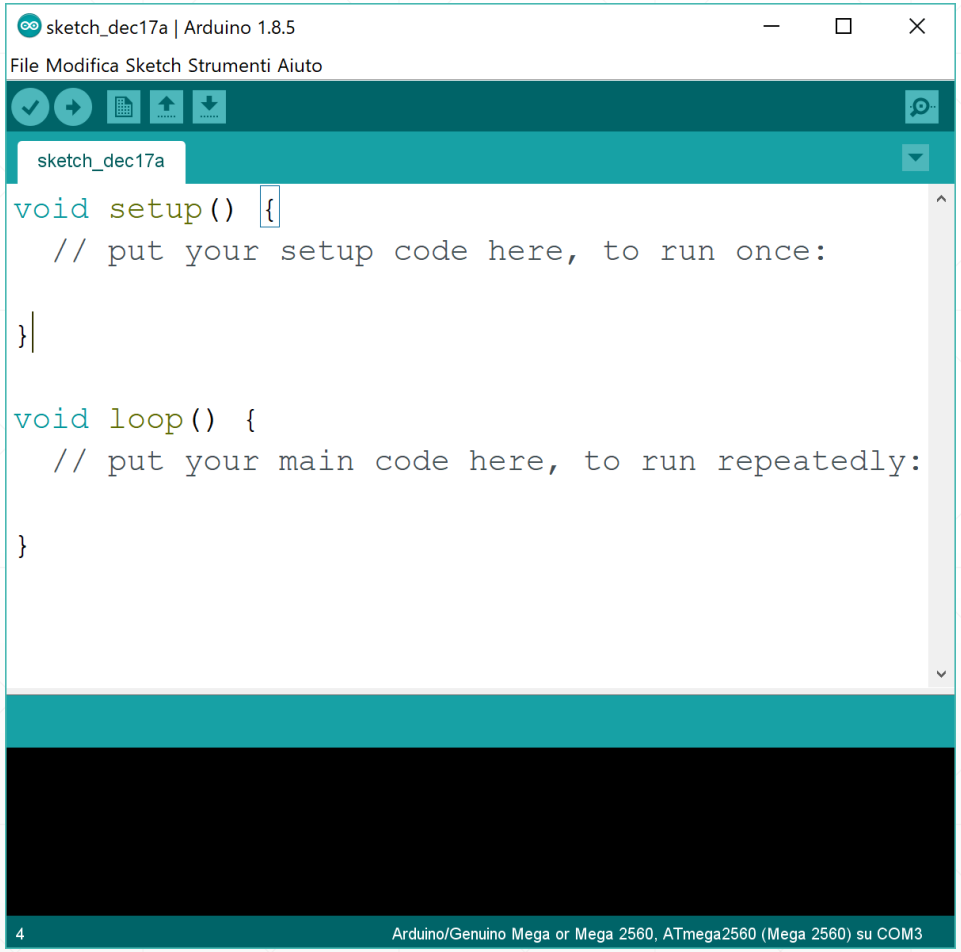


The Software

Anatomy of and Arduino Sketch:

You can also divide your code and function definitions in multiple files without caring for linking as long as:

1. The main sketch has the same name of the directory it's in
2. The other files are in the same directory



https://www.arduino.cc/en/tutorial/sketch

The Software

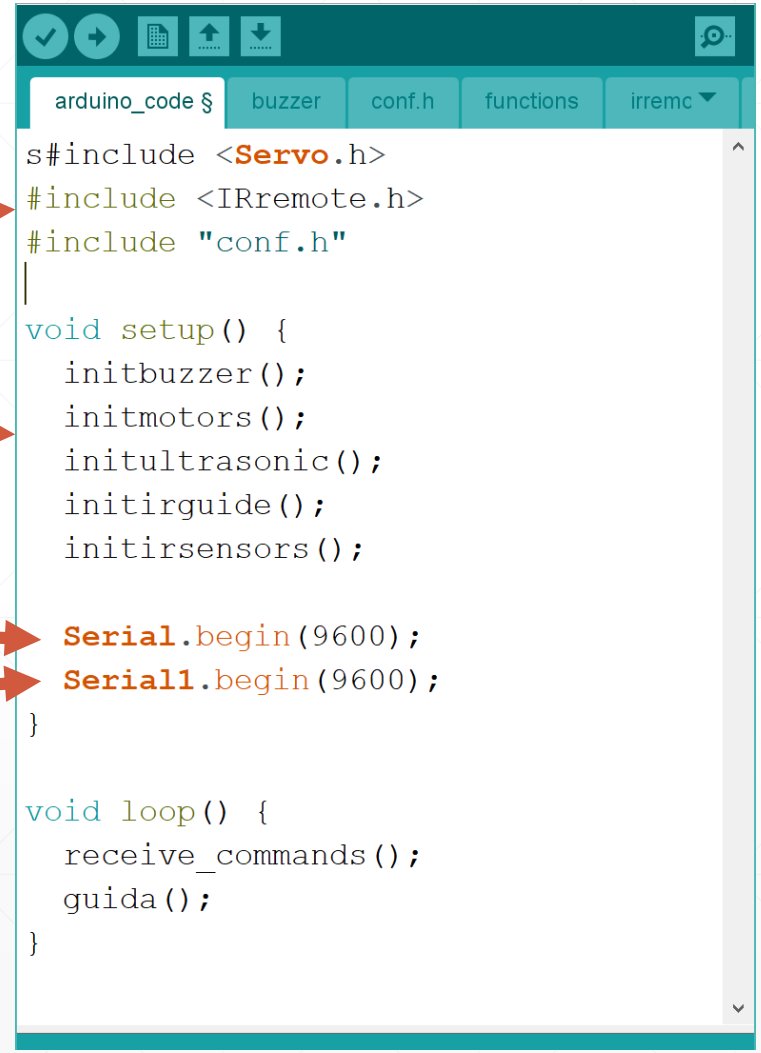
Our Arduino Sketch:

The main file is extremely simple:

Just some imports for `IRremote.h`, `Servo.h` and our configuration file `conf.h`

In the `setup()` we call specific setup routines we've written for everything we need and open two serial connections

- One on TX0,RX0 for usb debugging
- One on TX1, RX1 for communication with the ESP8266



https://github.com/filippocastelli/uolli

The Software

Our Arduino Sketch:

The `loop()` calls two different functions:

- one for receiving the commands
  - Either from Serial (from the ESP8266) or from the IR receiver.
- one for the actual driving

Let's take a look at the configuration file

`https://github.com/filippocastelli/uolli`

The Software

Our Arduino Sketch:

The `loop()` calls two different functions:

- one for receiving the commands
  - Either from Serial (from the ESP8266) or from the IR receiver.
- one for the actual driving

https://github.com/filippocastelli/uolli

The Software

Our Arduino Sketch:

Just overvieweing a couple of things in the code, we can see that the `init_xyz()` we called in the `setup()` look similar to each other: we use the **`pinMode()`** command to set some specific GPIO pins (described in the conf.h) to act either as `INPUT` or `OUTPUT`.

Other functions use the previously set pins to send digital signals (**`HIGH/LOW`**) (5V/0V) with functions like **`digitalWrite()`** that's used to define a state of a GPIO pin when set as `OUTPUT`

```
void initmotors()
{
  pinMode(L1pin, OUTPUT);
  pinMode(L2pin, OUTPUT);
  pinMode(speed_lpin, OUTPUT);
  pinMode(R1pin, OUTPUT);
  pinMode(R2pin, OUTPUT);
  pinMode(speed_rpin, OUTPUT);
  ferma();
}
```

```
void ferma()
{
  digitalWrite(L1pin, LOW);
  digitalWrite(L2pin,LOW);
  digitalWrite(R1pin,LOW);
  digitalWrite(R2pin,LOW);
}
```

https://github.com/filippocastelli/uolli

The Software

Our Arduino Sketch:

For when we need different voltages, we can theoretically implement our PWM routine, but this has a big disadvantage**: the output signal depends on what the processor is doing** and we can't focus on other operations than producing the PWM signal

To overcome this, Arduino has some PWM-enabled pins that can be controlled using the **`analogWrite()`** function without consuming precious controller cycles:

We can indicate the duty cycle as an int between 0 and 255

`https://github.com/filippocastelli/uolli`

```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delayMicroseconds(100); // Approximately 10% duty cycle @ 1KHz
  digitalWrite(13, LOW);
  delayMicroseconds(1000 - 100);
}
```

```
void setspeed(int speed_L,int speed_R)
{
  analogWrite(speed_lpin,speed_L);
  analogWrite(speed_rpin,speed_R);
}
```

The Software

Our Arduino Sketch:

The guide step is done without using too many delays because we can't afford to leave the controller idle a long period of time if we want it to be reasonally reactive: the longest routines have to be implemented using a time reference that's given by the `millis()` function

`millis()` returns the number of milliseconds that have passed from the controller's bootup.

We can use it to start timers and to control the duration of a certain operation.

```
void guida()
{
  if(stato_guida == GUIDA_MANUALE || stato_guida == IRMODE)
  {
    //Serial.println(azione);
    switch (azione)
    {
      case AVANTI:
          //Serial.println(azione);
          avanti();
          setspeed(255,255);
          RunningFlag = true;
          RunningTimeCnt = 1;
          RunningTime=millis();
          break;
      case SINISTRA:
          sinistra();
          setspeed(255,255);
          RunningFlag = true;
```

https://github.com/filippocastelli/uolli

The Software

Our Arduino Sketch:

The automatic mode implements really simple logics based on the perceived distances between the car and surrounding objects: these are measured using the **HSR04 module**.

We've written a simple routine to drive the module: we first tell the HSR04 to send a signal on the transducer with a digitalWrite() for 15 ms, then wait for an echo.

If there's an echo, the HSR04 sends an HIGH signal whose duration is the same as the time between the signal start and the echo receiving

https://github.com/filippocastelli/uolli

```
int measuredist(){
  long dist;
  digitalWrite(trigpin,LOW);
  delayMicroseconds(5);
  //accendo trasduttore
  digitalWrite(trigpin,HIGH);
  //tengo acceso per 15us
  delayMicroseconds(15);
  //spengo trasduttore
  digitalWrite(trigpin,LOW);
  //misuro tempo impulso di ritorno
  dist=pulseIn(echopin,HIGH);
  dist=dist*0.01657; //distanza oggetto in cm
  return round(dist);
}
```

The Software

Our Arduino Sketch:

We can measure the length of a pulse using the **`pulseIn()`** function: if it's set on HIGH it waits for the argument pin to switch from LOW to HIGH and, waits for it to return to LOW and then returns the HIGH time duration.

```
int measuredist(){
  long dist;
  digitalWrite(trigpin,LOW);
  delayMicroseconds(5);
  //accendo trasduttore
  digitalWrite(trigpin,HIGH);
  //tengo acceso per 15us
  delayMicroseconds(15);
  //spengo trasduttore
  digitalWrite(trigpin,LOW);
  //misuro tempo impulso di ritorno
  dist=pulseIn(echopin,HIGH);
  dist=dist*0.01657; //distanza oggetto in cm
  return round(dist);
}
```

https://github.com/filippocastelli/uolli

The Software

Our Arduino Sketch:

The rest of the Arduino code is quite self explanatory and going in detail would take too much time.

You may also want to check the ESP8266 code that basically creates an HTML page with some interactive buttons, detects the button presses and sends values to serial depending on which button has been pressed.

`https://github.com/filippocastelli/uolli`

In Conclusion:

What have we learned from this?

- Making something actually work can be rewarding.
  - It's more rewarding **when it doesn't catch fire.**
- You shouldn't buy the cheapest part **if** you want to finish you projects on time.
- **Documentation is everything**: if you don't know what you're doing you should always go for the most documented solution.
- If you buy the cheapest parts, you will (have to) learn a lot in the process
  - But, at least in the electronics world, you should also consider investing in a Chinese course.
- **Look online** when you have a problem: someone probably had it before you.
- **It's ok to be cheap/lazy**: justifying your bad habits can lead you to some new knowledge.
- **Don't short power rails a week before Christmas**: replacements for your fried controllers may take forever to arrive.

Filippo Castelli
filippocastelli

Carlo Emilio Montanari
carlidel

`https://github.com/filippocastelli/uolli`

Filippo Maria Castelli, MSc

castelli@lens.unifi.it

github.com/filippocastelli

Carlo Emilio Montanari, MSc

carlo.montanari5@unibo.it

github.com/carlidel

Thanks for your attention!