Politecnico di Torino

Department of Electronics and Telecommunications

Neuronica Lab

The Hexapod

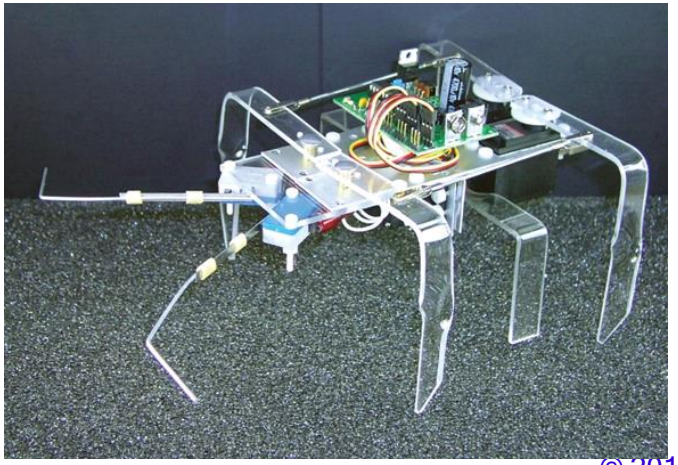
Francesco Rugiano - 2013

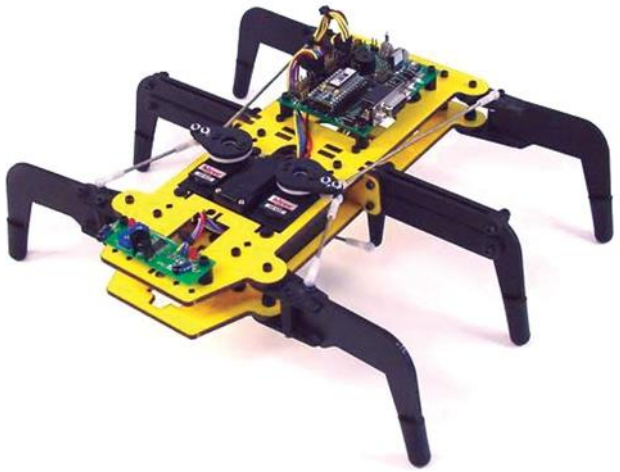
Customer requirements

- The aim of the project are the design and development of a robot. The required features are:
  - It must be as cheap as possible;
  - It must be simple enough so that even children (with limited programming knowledge) can use it;
  - It should (preferably) use open source programs, so that it's easier to modify and expand it.

Technical specifications

- The proposed architecture derives from the hexapod one; the result will be a walking robot with six legs moved by three servomotors.

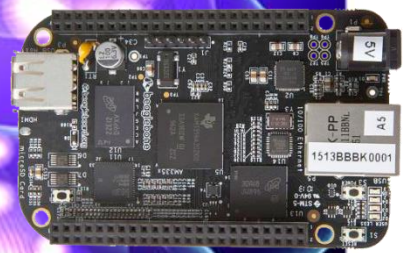




Technical specifications

- In order to keep the robot interactive, at least some sensors have to be used.
- In order to keep the total price as low as possible, two proximity sensors made by two switches are used, one to detect obstacles on the left side and one to detect the ones on the right side. Since the hexapod looks like an insect, the sensors will have the shape of two antennas.
- Another sensor will be added, in order to operate the robot by remote control. An IR decoder is chosen because of its low price.
- The power supply of the robot will be four AA batteries (4 x 1.5 = 6V).

Components choice



- Recently lots of general purpose boards (e.g. Arduino, Raspberry PI, BeagleBone) became more and more important in new designs. However, even if much cheaper than common development boards, they are too expensive for this project (around 20-40€).
- Luckily a lot of them are open source projects, so the schematics and firmwares are available on the internet; this way you can make a custom board.
- Among these, the Arduino UNO was chosen because of its lower complexity; moreover it uses through-hole components, so it's easier to build a custom board.

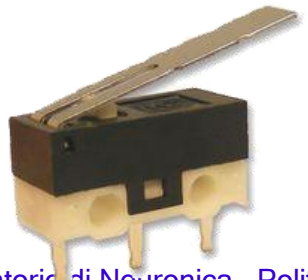
Components choice

- The robot will use three servo motors.
- In order to choose this component you need to know the maximum torque required; estimating a torque of at least 2 kg·cm, the cheapest one is the Hitech HS-311.

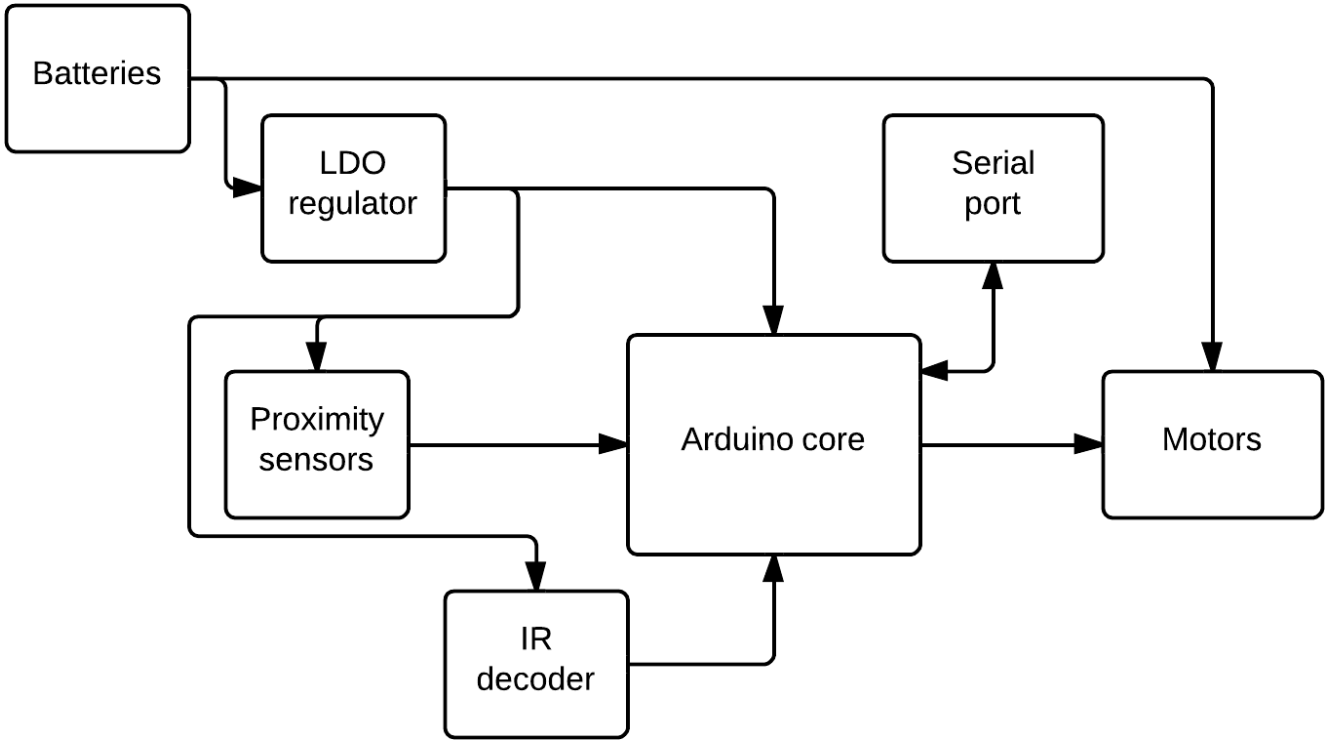
Hitech HS-311
Operating Voltage: 4.8-6.0 Volts
Stall Torque: 3.0 kg·cm @ 4.8V
3.7 kg·cm @ 6.0V

Components choice

- As for the power, the Arduino UNO needs 5V, while the batteries deliver 6V; in order to get the required voltage a small LDO integrated circuit was used; the MCP1700-500 (TO-92 package) from Microchip was chosen.
- As for the IR sensor, the TSOP2438 was chosen, because it has already got the filtering part and so it can be directly interfaced to the microcontroller.
- As for the switches, the cheapest limit switches were chosen.

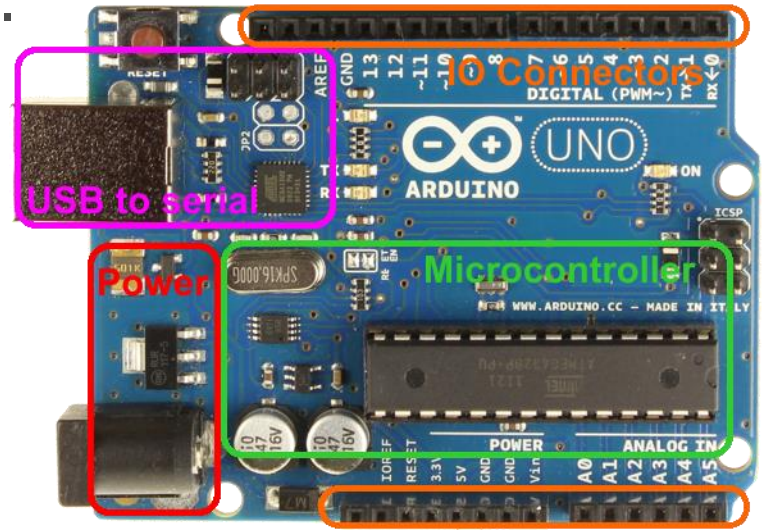


Block diagram



Schematic realisation

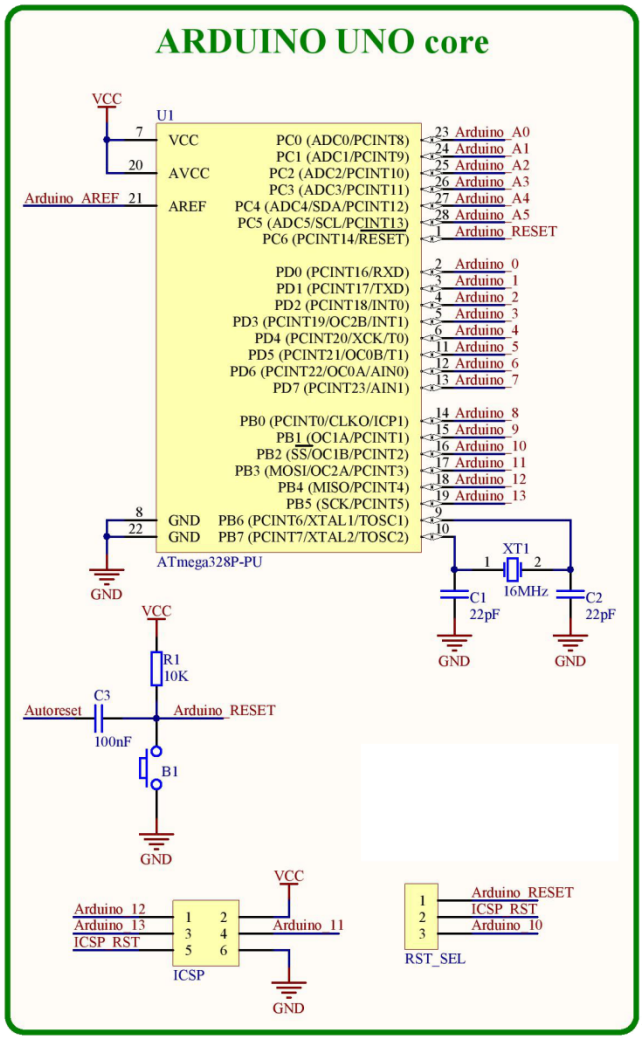
- The starting point is the Arduino UNO schematic, available from the project site.
- The circuit can be divided into several blocks
  - the microcontroller section;
  - the power section;
  - a USB to serial (UART) converter;
  - the IO connectors.

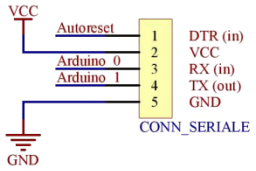


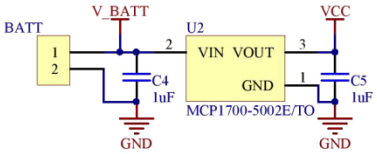
Schematic realisation

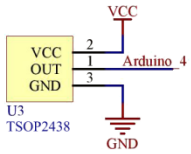
- The microcontroller section is the core of the Arduino UNO, so this has to be left almost untouched. The microcontroller is an Atmel ATmega328P.
- The power section is also useful, but the original one has higher performances than the ones needed by the project.
- The USB to UART converter has been removed, since it is the most complicated part and it is not strictly required. The conversion between USB and UART (or RS232 and UART) must be done off-board.

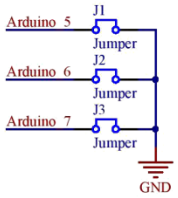
Schematic realisation

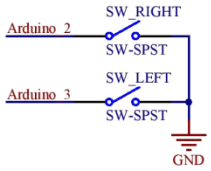


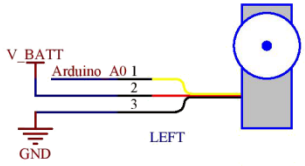


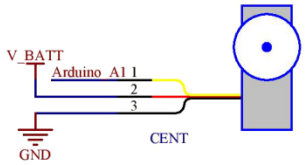


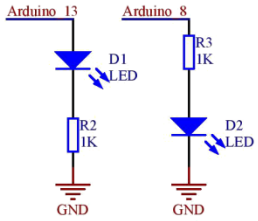


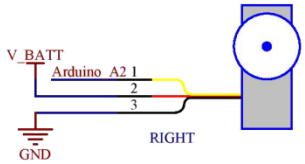








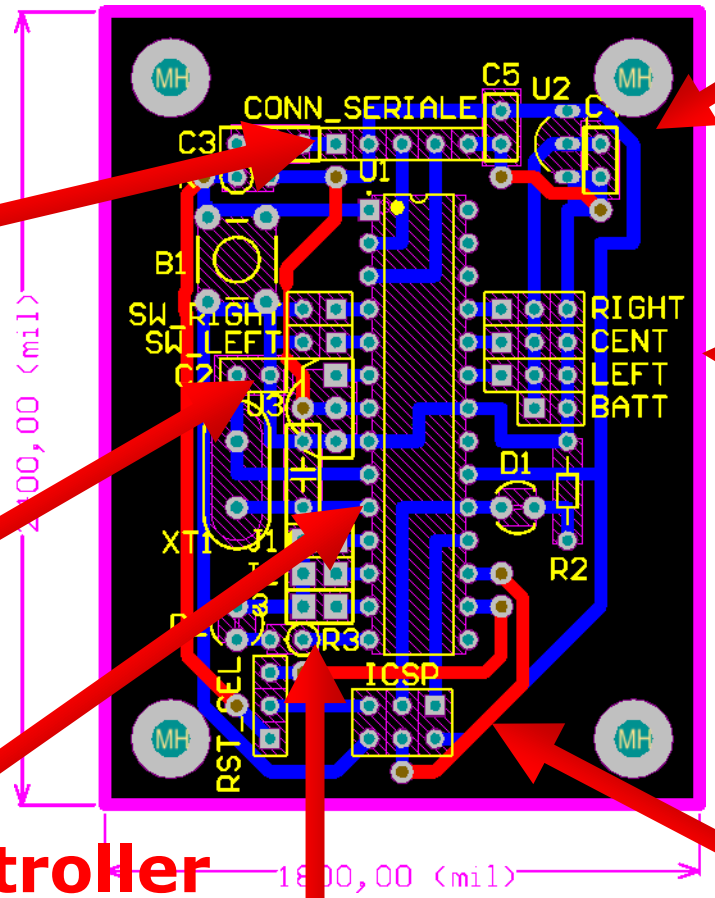




The layout

- As for the physical implementation, a PCB was considered too much expensive.
- Since it is a simple circuit and it uses only through hole components, a perfboard was used.
- However also a PCB layout was designed; this way the circuit can be replicated in an easier way and, in the prototype phase of the project, this layout can be used as a reference.
- For this reason in the layout a 100 mils grid was used (the spacing of the perfboard) and the circuit was developed mainly on the bottom side of the PCB.

The layout



Voltage regulator

Serial connector

Motors and power connectors

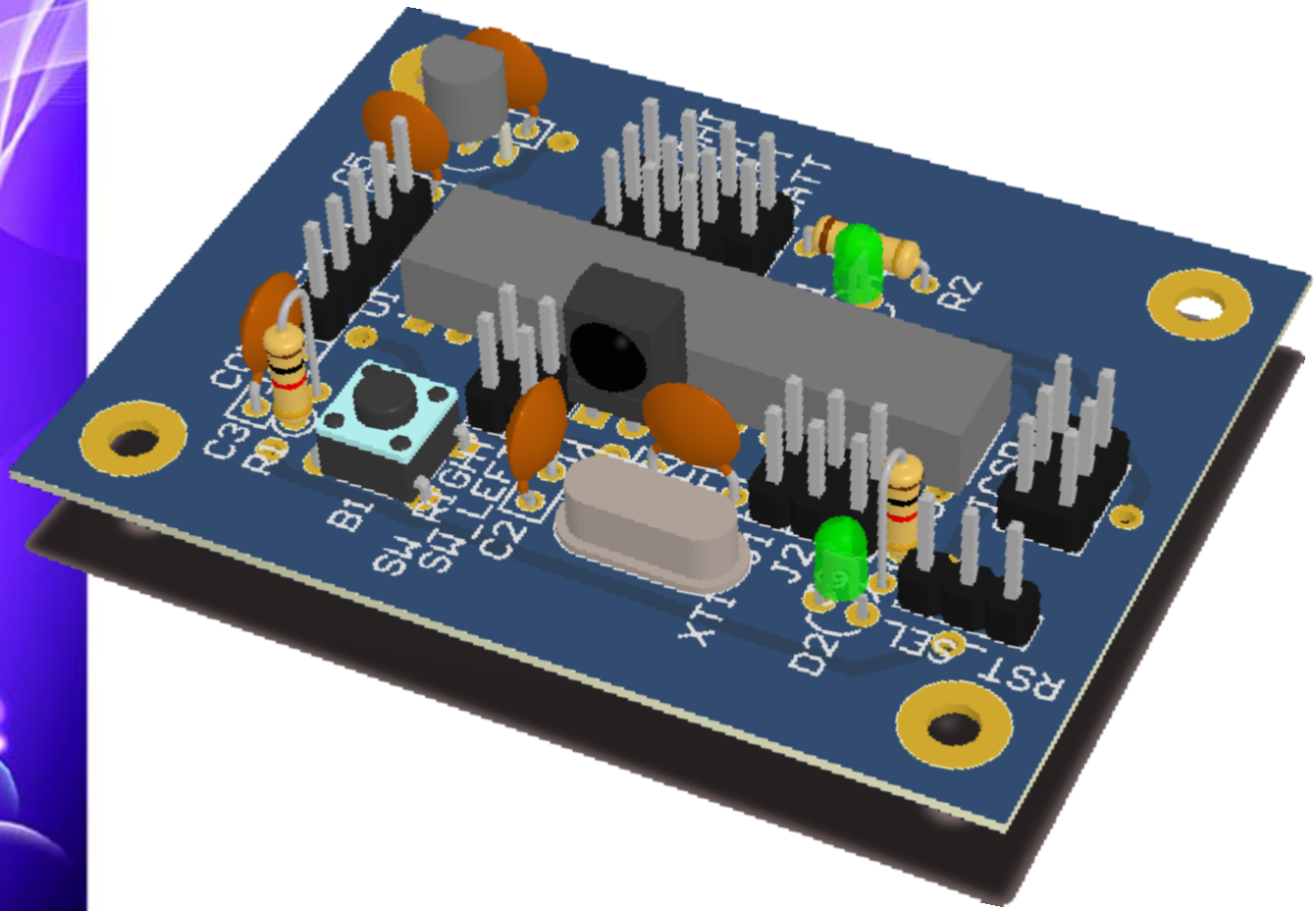
"Sensors"

Microcontroller

Mode switches

ICSP connector

The layout



The firmware

- The firmware has been divided in two parts: a library, which contains the more complex functions (such as controlling the servo motors or decoding IR codes), and a sketch, which implements different programs according to the configuration at startup.

- The sketch is divided into different files: one contains the "main" functions, while the others contain one mode each.

The firmware

- In this project, we have different programs to be executed.
- Every program has its own setup and loop functions (called respectively `*_setup` and `*_loop`, where * is the function name).
- In the `setup` function the program checks the program code (set through the "mode selector" jumpers) and executes the appropriate *_setup function.
- The `loop` function, however, can't check the code every time, both for security and performance reasons. Consequently a callback function is used (set in the `setup` and called in the `loop`).

The firmware

```
typedef void (*loopfunction)();
loopfunction currentloop;

void setup() {
    pinMode(MODE_PIN2,INPUT_PULLUP);
    pinMode(MODE_PIN1,INPUT_PULLUP);
    pinMode(MODE_PIN0,INPUT_PULLUP);
    byte mode = 7 - ((digitalRead(MODE_PIN2) << 2) |
        (digitalRead(MODE_PIN1) << 1) | digitalRead(MODE_PIN0));

    switch(mode) {
        case 0: // Shut down
            break;
        case 1: // Serial
            serial_setup();
            currentloop = serial_loop;
            break;
        [...]
    }
    if (!currentloop)
        while(1); // Halt the program if no loop function defined
}

void loop(){
    // Execute the action according to the required mode
    currentloop();
}
```

The firmware

- The library contains four different modules:
  - Heartbeat, which controls the HB led;
  - Servomotor, which contains function to control the motors;
  - IR, which has got functions to decode IR data;
  - Antenna, which controls the state of the antenna switches.

- The library exposes a class which collects objects belonging to the different classes in order to make the user able to control them.

The firmware

- For this robot, three different "user" modes and two "service" modes have been implemented.
- The "user" modes are:
  - Seriale (serial, code 1): the robot is controlled through commands received by the serial port
  - Casuale (random, code 2): the robot moves randomly, changing direction if hits something
  - ConTelecomando (with remote, code 3): the robot is controlled through a remote controller
- The "service" modes are:
  - Spento (shut down – code 0): do nothing
  - ImpostaTelecomando (set remote, code 7): through the serial port you can change the remote codes associated with commands.

Mode example: remote

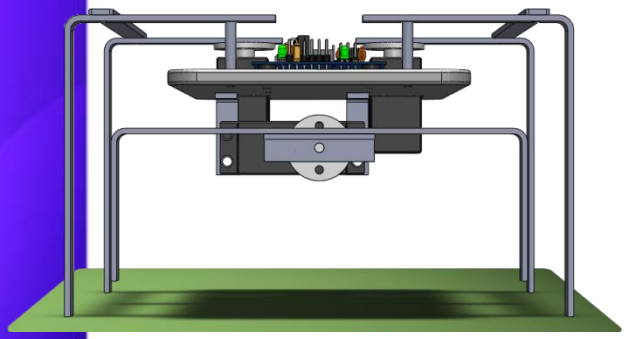
- The remote mode (ConTelecomando) is used to control the robot with a remote controller.
- You can use any IR remote controller, as long as you set the right codes through the mode ImpostaTelecomando.
- In this mode, you have to turn on every robot peripheral, then check the IR command and then, if the motors are ready, move them.

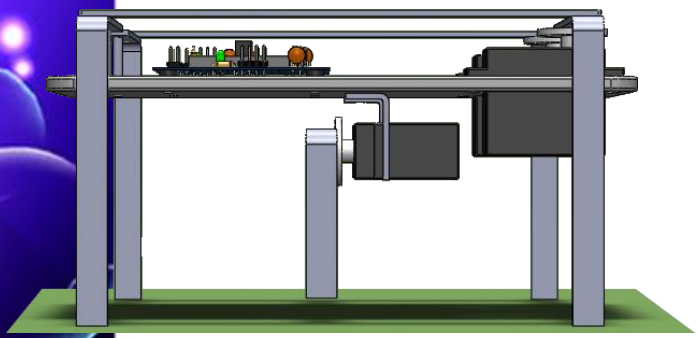
Remote mode (code)

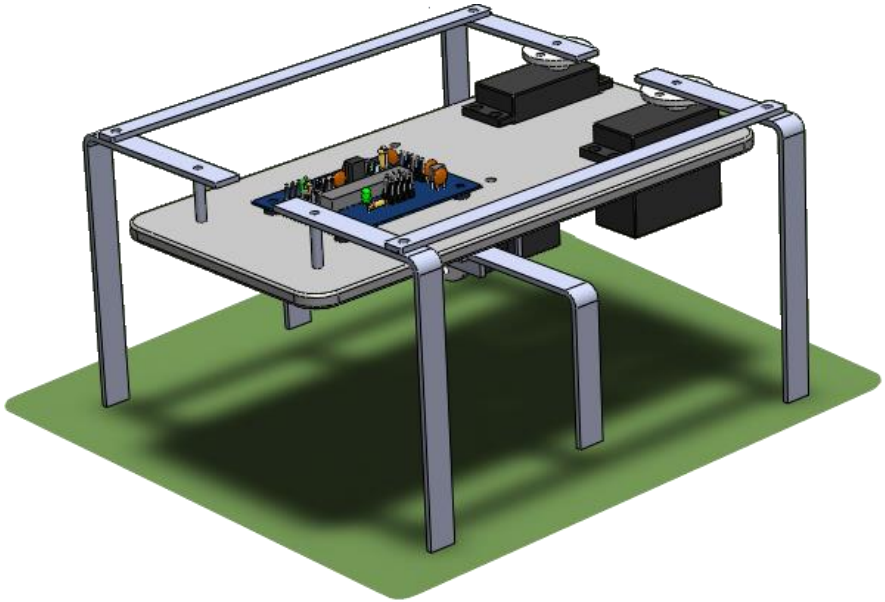
```
void conTelecomando_setup() {
   robot.enableIR();
   robot.enableMotors();
   robot.setSpeed(MOTORLEFT & MOTORCENTER & MOTORRIGHT, 13);
   robot.enableHB();
   int States[4] = {100,100,100,900};
   robot.setHBStates(States, 4);
   robot.enableAntenna(AntLeft, AntRight);
}
void conTelecomando_loop() {
   static RoboDuino_IRCommands::EnumType CurrentCommand;
   robot.loop();
   CurrentCommand = robot.readIR(true);
   if (!robot.isHittingLeftAntenna())&&(!robot.isHittingRightAntenna())
      OstacoloRilevato = false;
   if (!robot.busyMotors()) {
      switch (CurrentCommand)
      {
         case RoboDuino_IRCommands::Avanti:
                if (!OstacoloRilevato)
                   robot.avantiMotors();
                break;
      case RoboDuino_IRCommands::Indietro:
                robot.indietroMotors();
                break;
      [...]
      }
   }
}
```

The model

- As for the mechanical realization, a 3D model was produced in order to have a reference for the cutting and assembling. The metal parts are made of aluminum.

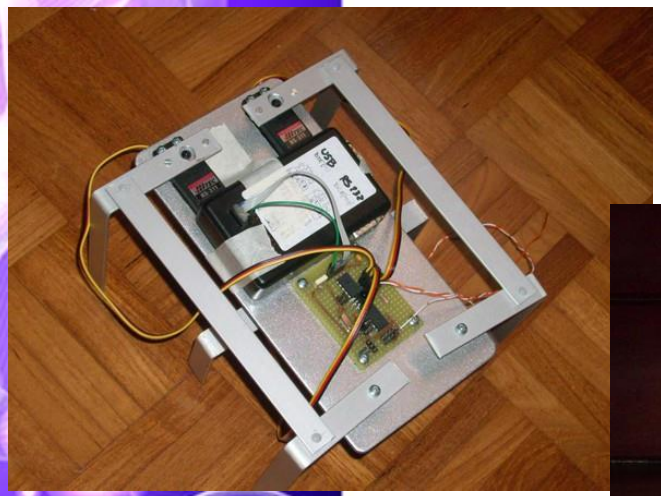


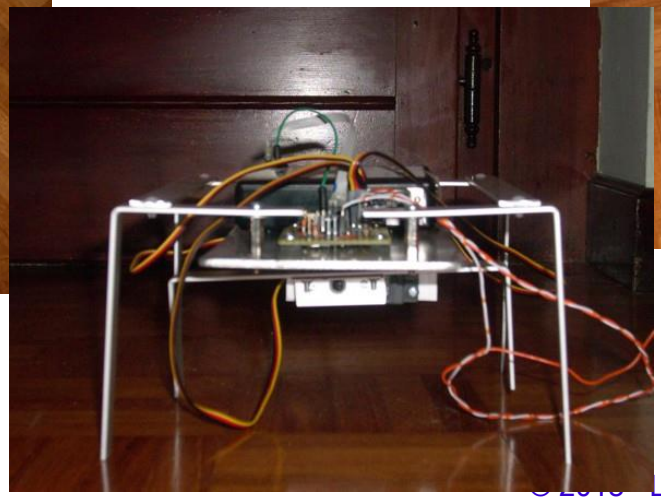


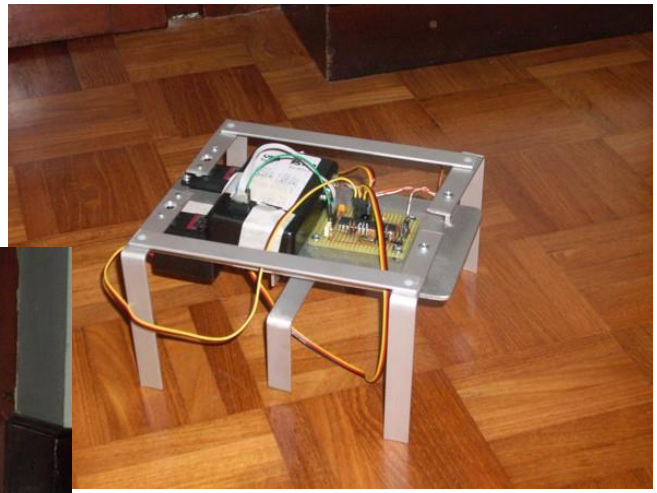


The realisation

- These are the photos of the prototype, not yet in its final version. Particularly the black box has to be removed (it is a USB->UART converter used just for the first debug phases).

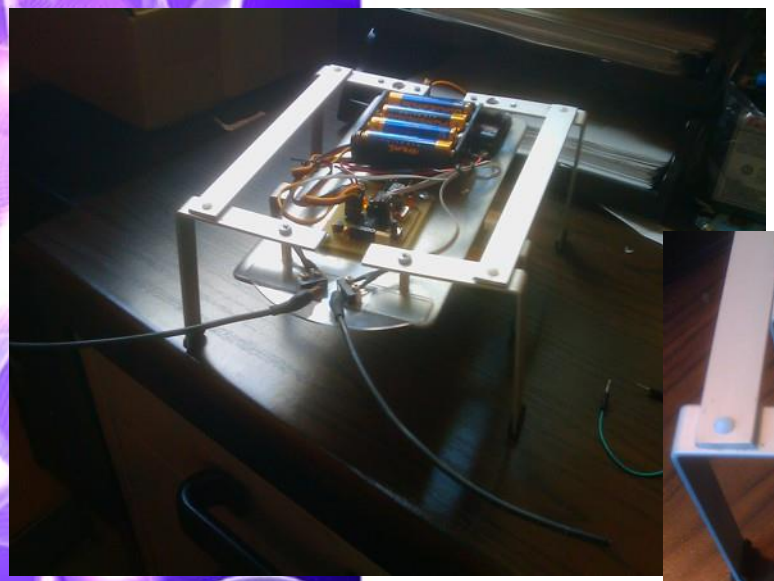


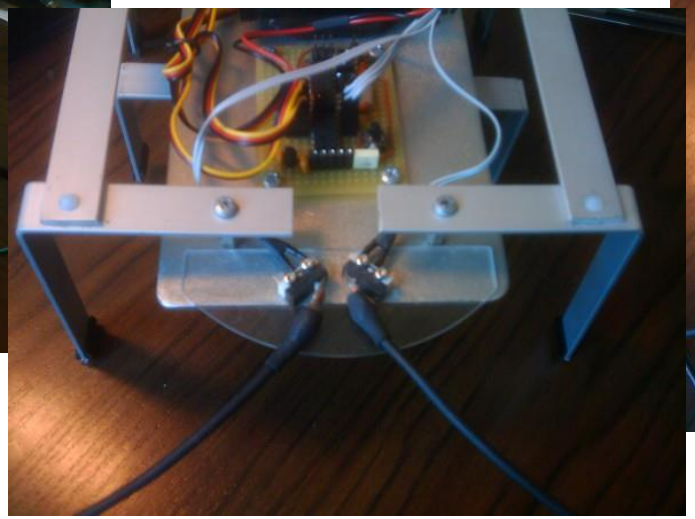


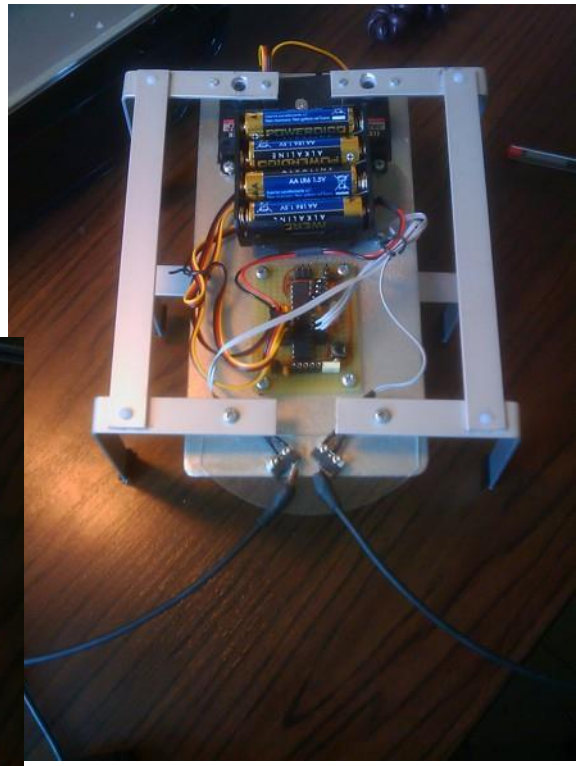


The final prototype

- And this is the final prototype. The battery holder has been added, along with the two antennas.





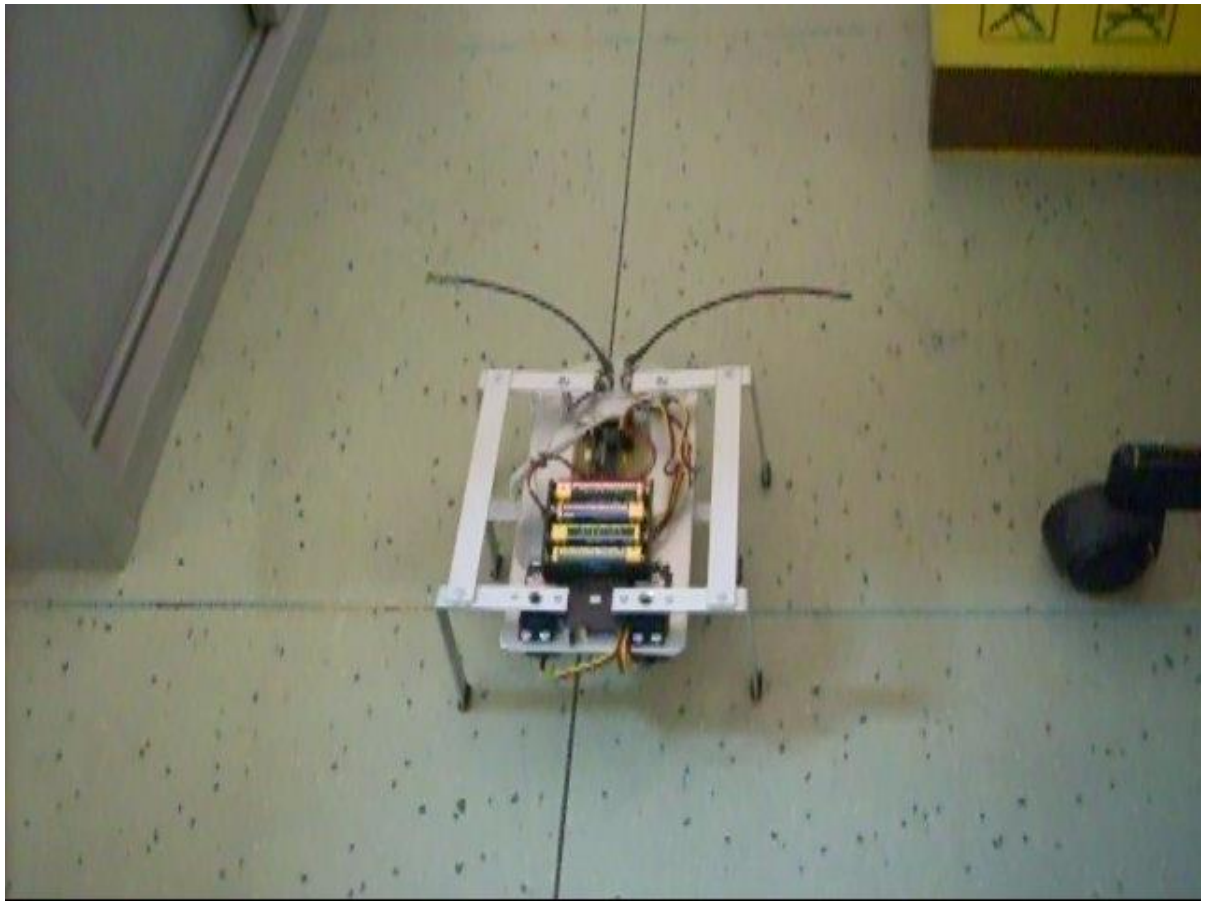


Prototype costs

Name	Unit cost	Quantity	Total
Chassis	~10€	1	10€
Atmega328P	3€	1	3€
IR Decoder	1.20€	1	1.20€
Battery holder	1.80€	1	1.80€
Other electronic	4€	1	4€
Perfboard	~3€	1	3€
Motors	8€	3	24€

Total: 47 €

Remote controlled mode



The video is also available at www.youtube.com/watch?v=GRCuVDTIO4g

Random mode

The video is also available at www.youtube.com/watch?v=03TBpros8ZI