README.md

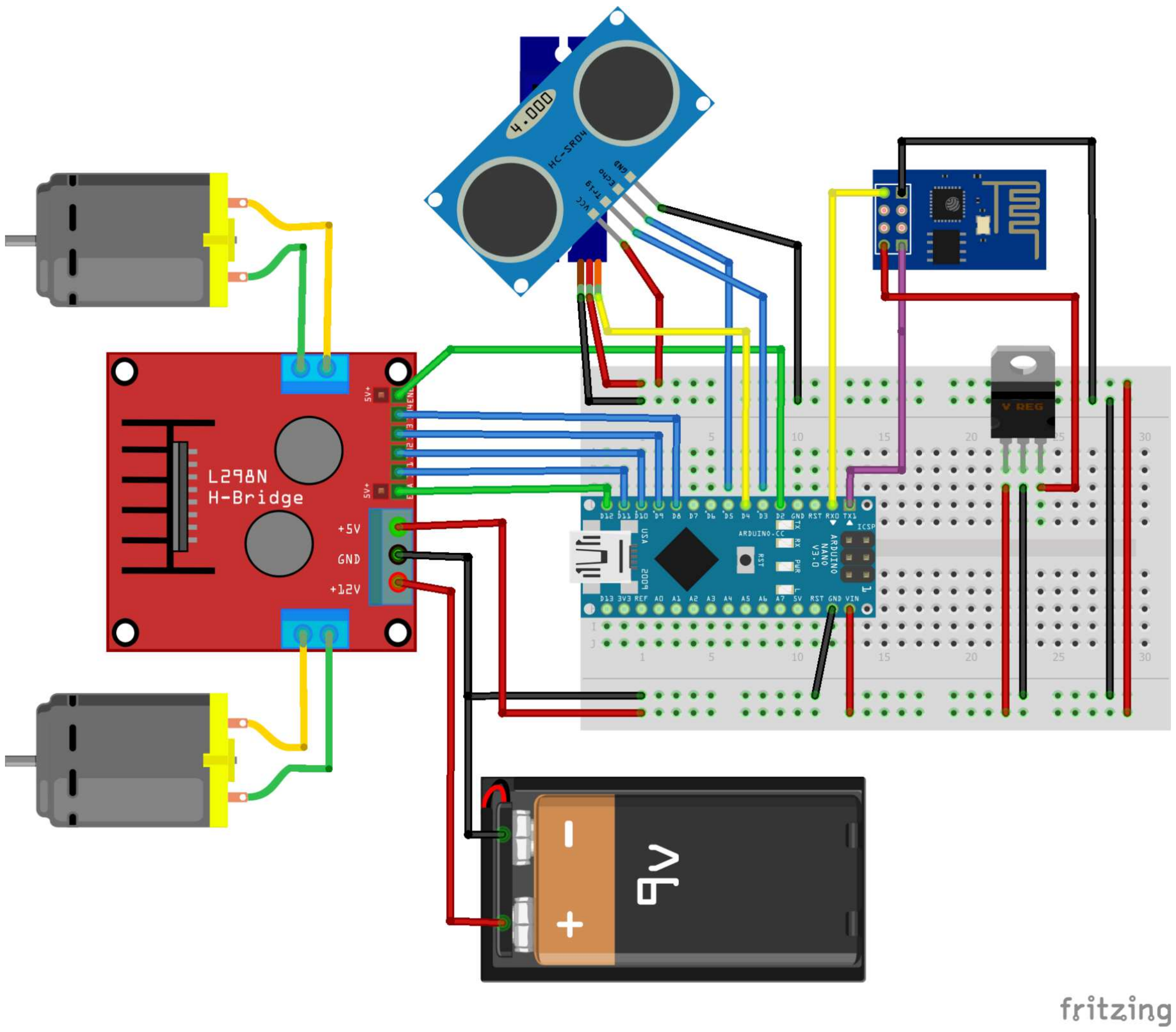
Tracker : your personal robot follower

Antoine Gauthier, Antony Lopez

This project consists in a robot able to follow a moving object on a flat surface. It can be remotely controlled by an user using Wi-Fi, but its main goal is to follow a target, using a sonar sensor.

Building the robot

Here is a quick view of the final schematic:

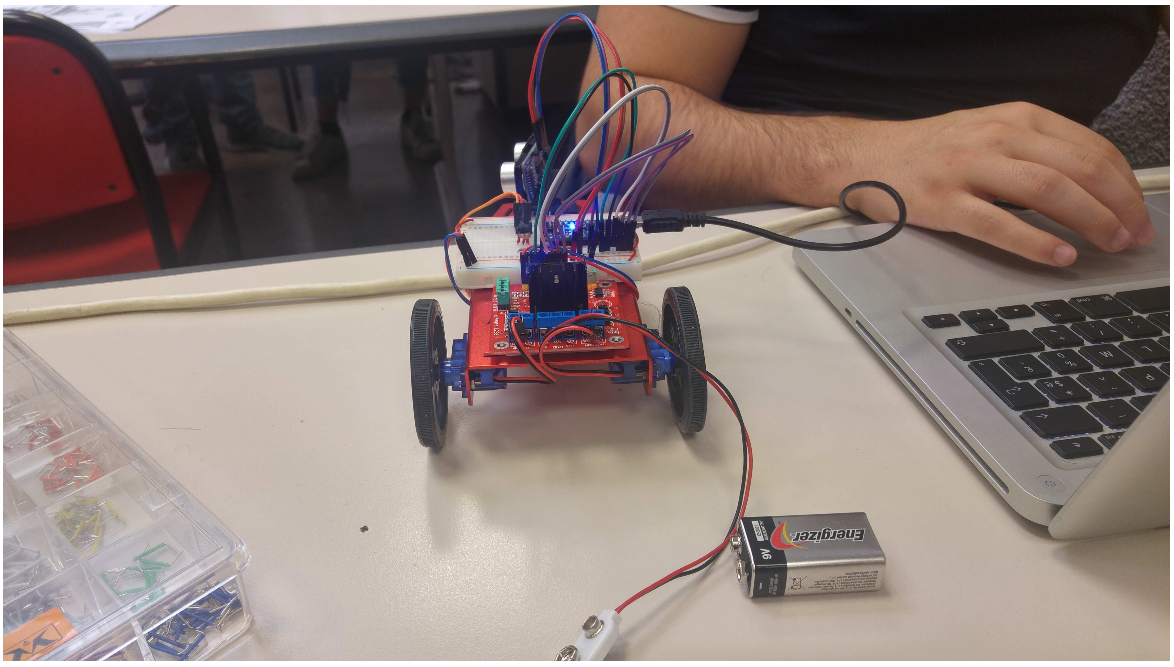


We use the KeyesL298N board (in red) to control the motors. The idea is to deliver much more power to the motors than what would normally be able to deliver the Arduino Nano board. Also the Keyes L298 gives us a +5V pin regulated from the 9V battery to power the Arduino.

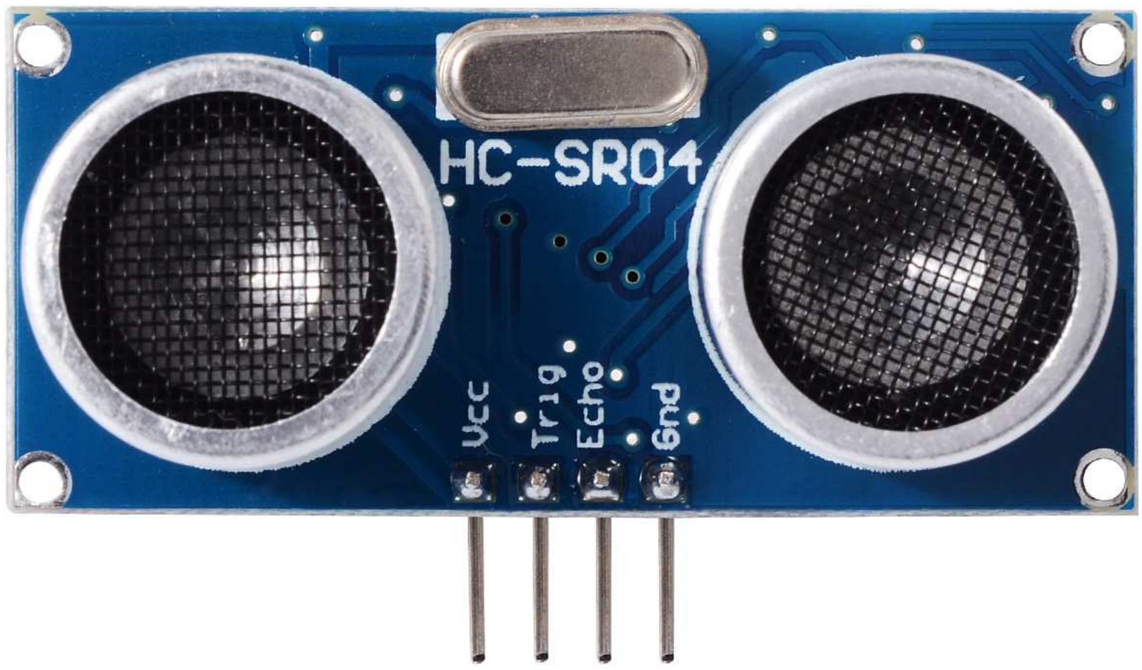
Moreover, KeyesL298 is a really easy to use board, thanks to all the libraries which already exist. For instance, using the Alonso's L298N library, in order to make the motor go forward you just use the following code:

```
/*#########################
##     MOTOR CONSTANTS   ##
#########################*/

const uint8_t ENA = 12;
const uint8_t ENB = 7;
const uint8_t IN1 = 11;
const uint8_t IN2 = 10;
const uint8_t IN3 = 9;
const uint8_t IN4 = 8;
const uint8_t MINSPEED = 128;
const uint8_t REVERSE  = false;

uint8_t speed = 130;
uint8_t duration = 130;

L298N driver(ENA, IN1, IN2, IN3, IN4, ENB, REVERSE, MINSPEED);
driver.forward(speed, duration);
```

Picture of the robot without the WiFi module



Ultrasonic Module SR04



In order to detect the environment, we use a sonar sensor at the front of the robot, mounted on a servo motor. This way, we are able to detect obstacles in any direction in front of the robot.

The distance estimation process is quite simple, you send an ultrasonic wave, waits for it to bounce on an obstacle, and then detect the reflection. The time it took for the ultrasonic wave to go back and forth let us deduce the distance of the obstacle. We were able to detect obstacles from at most 50 centimeters and the system finally could give good results when we applied a basic average filter to get rid of some random incoherent values.

Motors

The very first issue we had to face was the motors's speed. Indeed, below a PWM duty cycle of 50%, we could not make it move at all. The solution we used is kind of original, we applied a PWM behaviour to our calls to the PWM Arduino functions.

To be more specific, we chose a discrete amount of time (6ms) and divided this time in two phases, PWM at 100% and PWM at 0%. This way we were abble to maintain a pretty smooth speed on the motors. For example, a smooth "low" speed could be achieved spending 2ms at 100% and 4ms at 0%.

To implement this "software" PWM we based our code from the `drive()` function of the KeyesL298 library. Here is our implementation:

```
void L298N::cdrive(uint8_t direction, uint8_t speed, int delay_time, uint8_t ms_diff)
{
    if ( direction == FORWARD  || direction == FORWARD_R  || direction == FORWARD_L  || \
            direction == BACKWARD || direction == BACKWARD_R || direction == BACKWARD_L || \
            direction == RIGHT    || direction == LEFT       || \
            direction == STOP     || direction == BRAKE )
    {
        if (speed > 100) speed = 100;
        digitalWrite(IN1, bitRead(direction, 3));
        digitalWrite(IN2, bitRead(direction, 2));
        digitalWrite(IN3, bitRead(direction, 1));
        digitalWrite(IN4, bitRead(direction, 0));

        int onTime = DISCRETE_MV_TIME * (float(speed) / 100); //DISCRETE_MV_TIME = 6ms by default
        int offTime = DISCRETE_MV_TIME  - onTime;

        //if moving with a tilt
        if ( direction == FORWARD_R  || direction == FORWARD_L || direction == BACKWARD_R || direction == BACKWARD_L)

            // ms_diff variable represents the amount
            //  of time (in percent) spent at STOCK_SPEED by the slower wheel
            //  compared to the faster wheel. Setting this variable
            // to 100 will have the same behaviour than FORWARD/BACKWARD
            if (ms_diff > 100) ms_diff = 100;
            int slaveOnTime = onTime * (float(ms_diff) / 100);

            // Generating a PWM behavior on the PWM output
            for (int i = 0; i < delay_time; i += DISCRETE_MV_TIME) {
                analogWrite((direction == FORWARD_R || direction == BACKWARD_R) ? ENB : ENA, STOCK_SPEED);
                delay(onTime - slaveOnTime);
                analogWrite((direction == FORWARD_R || direction == BACKWARD_R) ? ENA : ENB, STOCK_SPEED);
                delay(slaveOnTime);
                analogWrite(ENB, 0);
                analogWrite(ENA, 0);
                delay(offTime);
            }
        }
        //if moving on itself
        else if (direction == RIGHT || direction == LEFT ) {
            analogWrite((direction == RIGHT) ? ENA : ENB, 0);
            for (int i = 0; i < delay_time; i += DISCRETE_MV_TIME) {
                analogWrite((direction == RIGHT) ? ENA : ENB, STOCK_SPEED);
                delay(onTime);
                analogWrite((direction == RIGHT) ? ENA : ENB, 0);
                delay(offTime);
            }
        }
        //if moving in a straight line
        else {
            for (int i = 0; i < delay_time; i += DISCRETE_MV_TIME) {
                analogWrite(ENA, STOCK_SPEED);
                analogWrite(ENB, STOCK_SPEED);
                delay(onTime);
                analogWrite(ENA, 0);
                analogWrite(ENB, 0);
                delay(offTime);
            }
        }
    }

}
```

Sensor & Following

What can you do once you have the sensors information and two fully functional motors ? The answer is : follow !

We decided to only modify the direction of the robot, which was related to the difference between the speed of the two motors.

$$\omega_2 - \omega_1 = f(I_{sensor}(t), t)$$

We decided to use a control system that use the distance of the target as the control variable over the motors's speed.

The basic idea is that when:

$$\frac{dI_{sensor}}{dt} > 0,$$

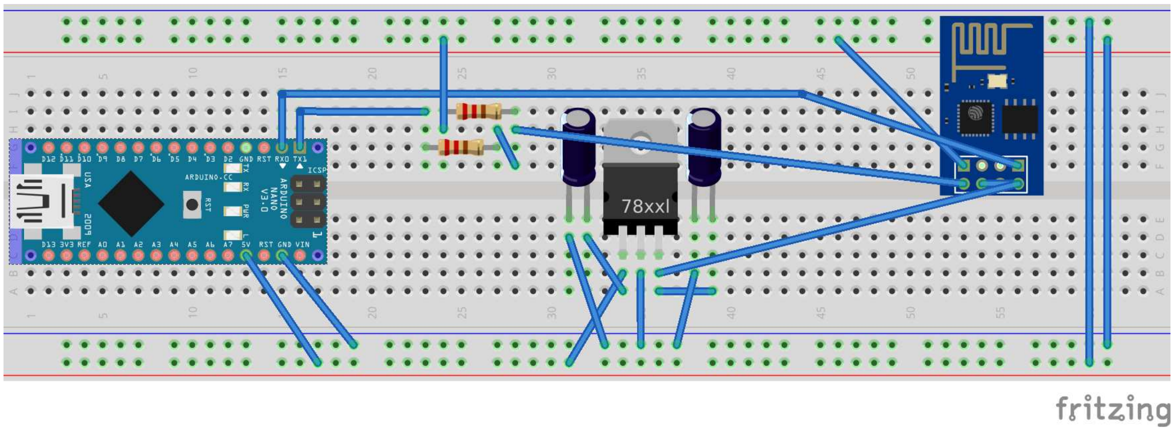
We continue to go in the same direction. However, when:

$$\frac{dI_{sensor}}{dt} < 0$$

We change the direction.

With this method, we can follow the target pretty efficiently.

Controlling through Wi-Fi



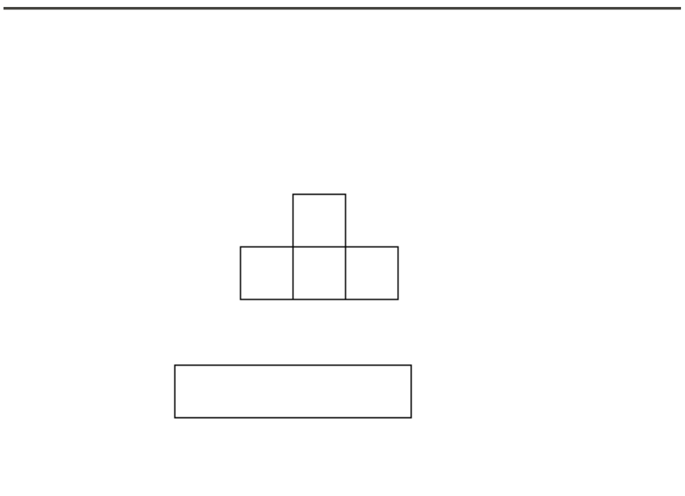
We use the ESP8266 Wi-Fi chip to interface the Arduino and the Wi-Fi. The component is controlled through a SoftwareSerial object in the Arduino code. We use a library designed to control this card and its Wi-Fi. activity.

We then create a TCP server on the Process application, the program waits for the client to connect.
Once the robot is connected, we can send orders to the robots through packets containing the input.

```
#include "ESP8266.h"

#define SSID        "Athens2016"
#define PASSWORD    "Arduino2016"
#define HOST_NAME   "192.168.0.129"
#define HOST_PORT   22522

SoftwareSerial mySerial(6,7);

ESP8266 wifi(mySerial);

void setup(void)
{
    Serial.begin(9600);
    Serial.print("setup begin\r\n");

    Serial.print("FW Version:");
    Serial.println(wifi.getVersion().c_str());

    if (wifi.setOprToStation()) {
        Serial.print("to station + softap ok\r\n");
    } else {
        Serial.print("to station + softap err\r\n");
    }

    if (wifi.joinAP(SSID, PASSWORD)) {
        Serial.print("Join AP success\r\n");
        Serial.print("IP:");
        Serial.println( wifi.getLocalIP().c_str());
    } else {
        Serial.print("Join AP failure\r\n");
    }

    if (wifi.disableMUX()) {
        Serial.print("single ok\r\n");
    } else {
        Serial.print("single err\r\n");
    }

    Serial.print("setup end\r\n");
}

void loop(void)
{
    uint8_t buffer[128] = {0};

    if (wifi.createTCP(HOST_NAME, HOST_PORT)) {
        Serial.print("Connected to server \n");
    } else {
        Serial.print("Disconnected from server \n");
    }

    uint32_t len = wifi.recv(buffer, sizeof(buffer), 10000);
    if (len > 0) {
        Serial.print("Received:[");
        for(uint32_t i = 0; i < len; i++) {
            Serial.println((int)buffer[i]);//The buffer contains the value : 1 = z / 2 = q / 3 = s /4 = d
        }
        Serial.print("]\r\n");
    }
}
```

Processing App

The interface is basic, we just use a graphical way of seeing the input we enter. Every time the user hits a key, we get it, and if it is directional keys or the spacebar, we change the packet value, from 0 to a specified value for each key.



During this time, the server send packets all the time, sending its state. It allows the user to control remotely the robot.

Problems we encountered

We spent a full day mounting the whole robot:

- Choosing and gluing/connecting the frame
- Choosing and connecting the H bridge
- Connecting the Arduino, the Servo and the SR04

Then Antony spent the second day writing the function to control the motors while Antoine was trying to connect his Processing program the the Wi-Fi chip.

For the very last day, the Wi-Fi had a behavior way too random while send/receiving data and connecting. That's why we canceled the Wi-Fi feature of our project. We then focused on the ultrasonic following algorithm which can be found in the `Follower` directory.

Here is a pic of the robot with the Wi-Fi module mounted: