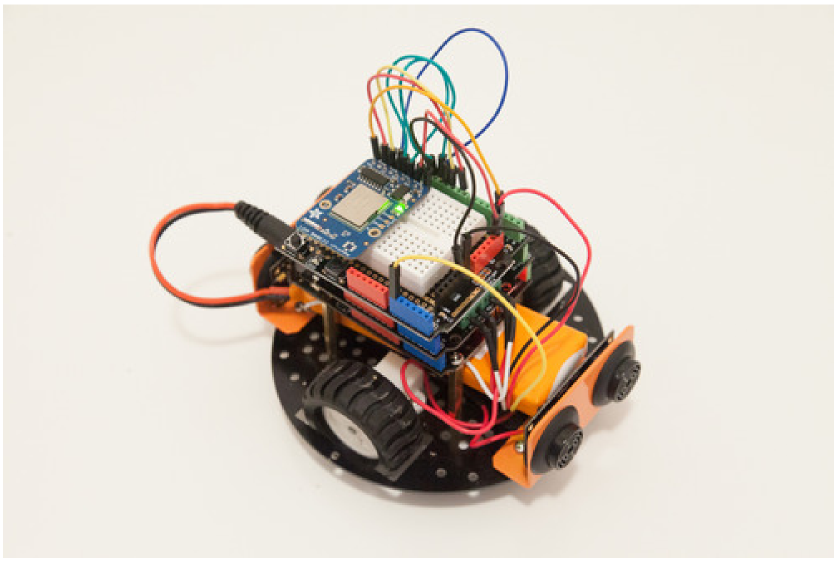
adafruit learning system

WiFi Controlled Mobile Robot

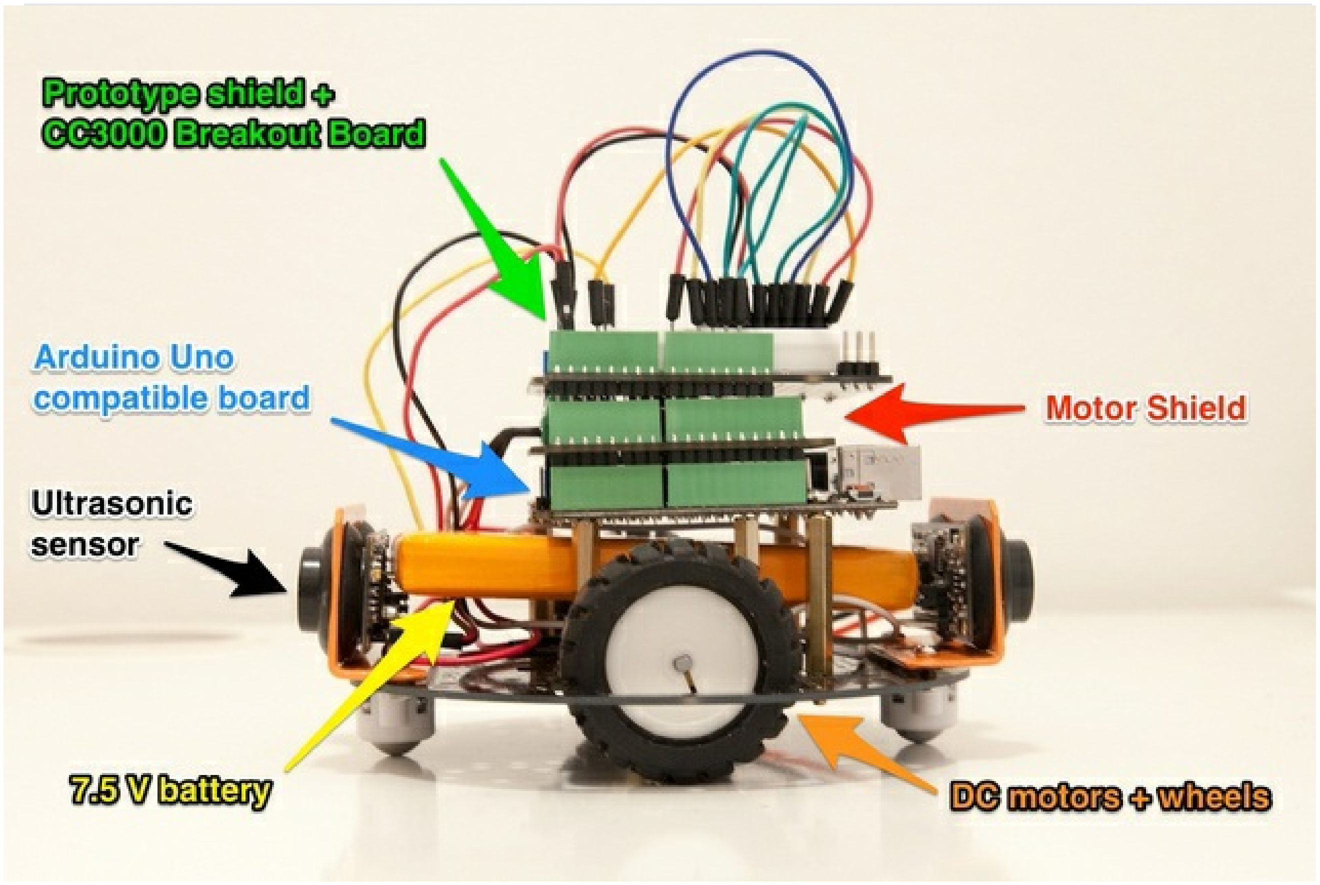
Created by Marc-Olivier Schwartz



Last updated on 2015-05-03 05:00:10 AM EDT

Guide Contents

Introduction



The CC3000 WiFi chip has a wide range of applications, and thanks to breakout boards like the Adafruit CC3000 breakout board, it is very easy to interface this chip with open-source platforms like Arduino.

In this guide, we will focus on one of these applications: robotics. We are going to see how to plug the CC3000 WiFi breakout board on a mobile robot to control it remotely from your computer, or from your smartphone or tablet.

This particular type of control uses a seperate server computer that the robot contacts to get "instructions", rather than hosting the server locally. If you'd like to check out an example of how to run a lightweight server on the CC3000+Arduino itself check out the WiFi candy bowl project (http://adafru.it/cVI)!

The tutorial will start by introducing the different components that you need to build a robot suitable for this tutorial, and how to plug the WiFi chip to your robot. Then, we'll see how to write the Arduino sketch & the server-side interface. Finally, you will find a short video of the robot in action. Let's dive in!

Connections

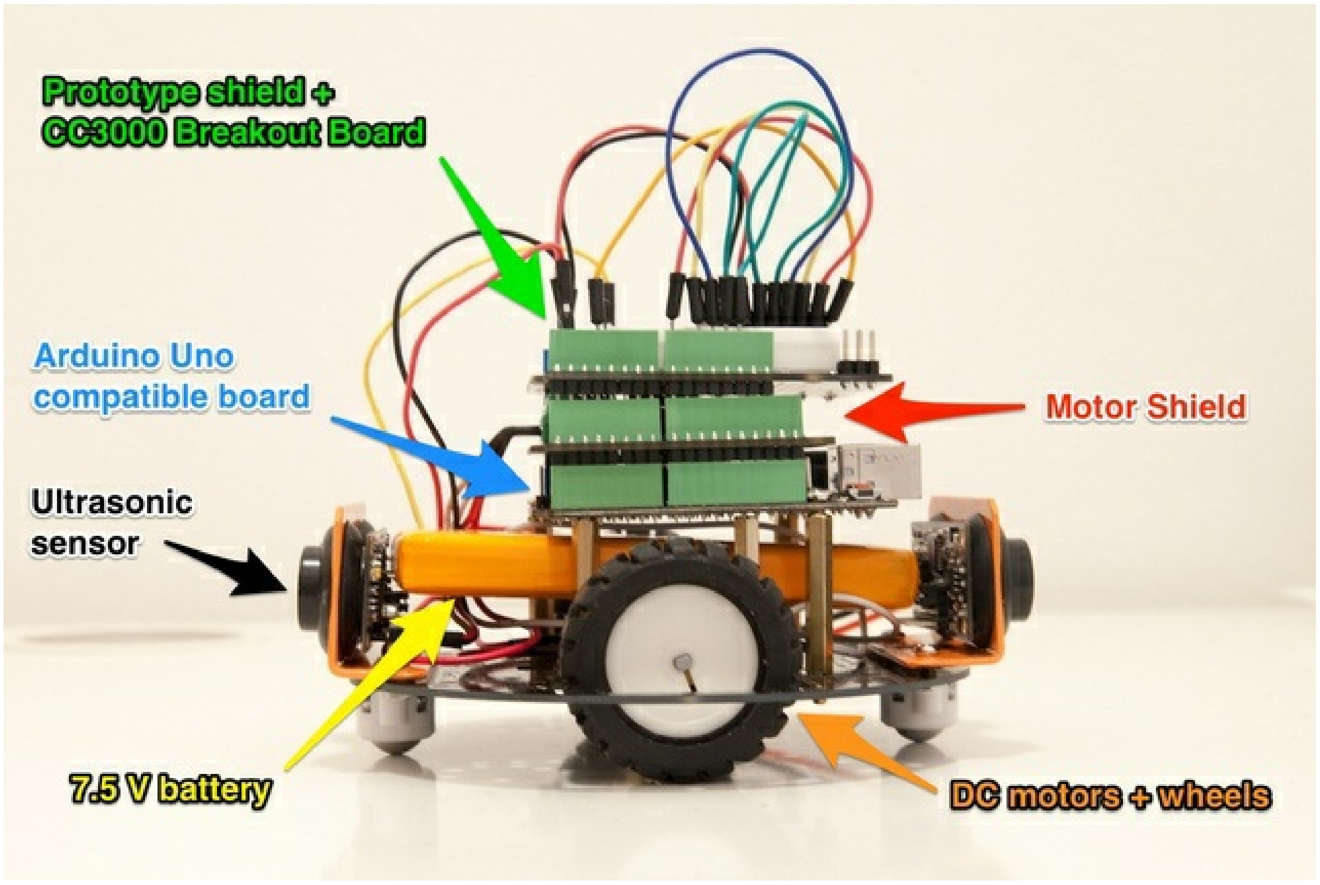
In this part, we are going to see how to completely setup the hardware part for this project. The core of this project is to connect the CC3000 breakout board to your robot. For this reason, I will not go through the details of the different parts of the robot, as your own robot can be slightly different. Instead, I will go through the main parts that your robot should have so this tutorial can work.

The robot is basically a 2-wheels robot with a round base, and two ball casters to make it stable. The wheels are connected to 6V DC motors. The whole project is powered by a 7.5 V battery. This guide will work as well for servos or stepper motors, but you will have to make some changes in the code.

Then, the command part is composed of an Arduino Uno, a motor shield, and a prototype shield to mount the CC3000 breakout board. I used boards from DFRobot for all these components, but you can use similar components from the supplier of your choice. You will find links to the equivalent Adafruit boards in the featured products on the right.

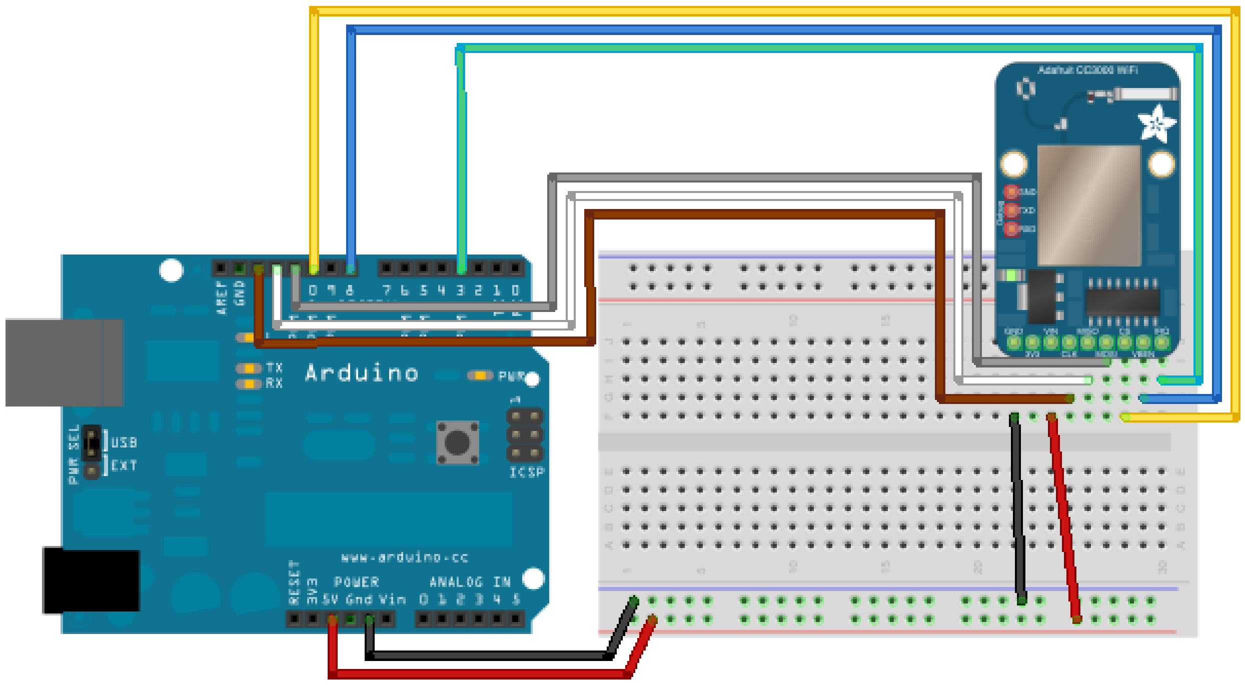
I also added two ultrasonic sensors that won't be used in this tutorial, one on the front, one on the back of the robot.

The following picture summarises the essential parts of the robot I used for this tutorial:

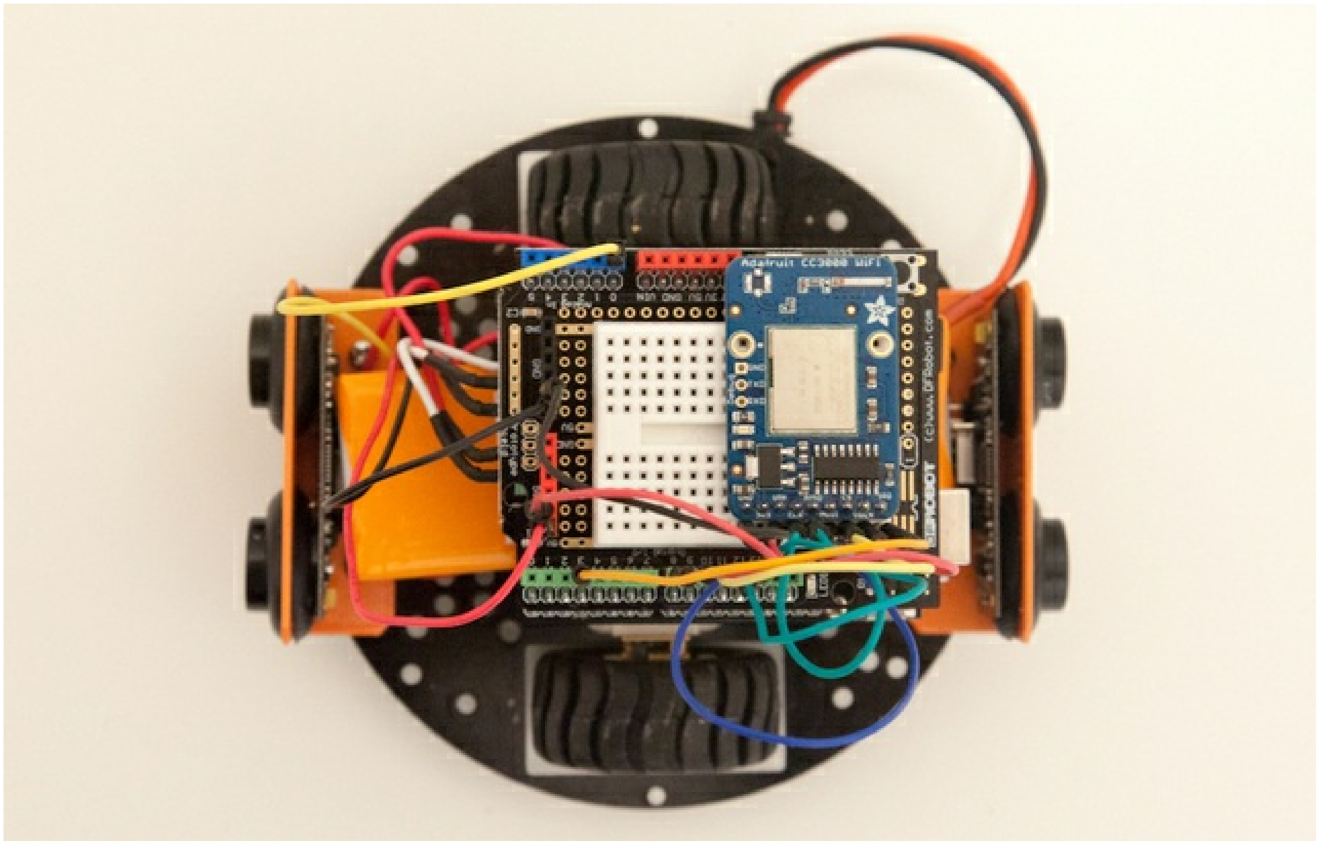


When you have your robot ready, you have to connect the CC3000 breakout board on the prototype shield on top of the robot. The important thing here is that you have to make sure that you don't interfere with the motor shield you have between the Arduino and the prototype shield. For example, on my board the motors are using pins 4,5,6 and 7, so I made sure that I didn't connect anything to these pins. Write down these pins for your motor shield, we will need them later.

The following picture describes the hardware connections for the CC3000 breakout board:



This is how it should look like at the end:



Arduino sketch

Let's now see the Arduino sketch for this project. You will need to have the Arduino IDE (http://adafru.it/f5E) installed on your computer, as well as the Adafruit CC3000 library (http://adafru.it/cFn) and the aREST (http://adafru.it/dis) library. To install the library, simply clone the Git repository or extract the files in your Arduino /libraries folder.

The sketch basically connects the CC3000 WiFi chip to your WiFi network, creates a web server on the Arduino board, and then start listening for incoming connections. When it gets the commands, it applies them directly to the two motors to move the robot accordingly. The main parts of the sketch are explained below, and you can find the complete code on the GitHub repository of the project (http://adafru.it/f5F).

The sketch starts by importing the required libraries:

```
#include <Adafruit_CC3000.h>
#include <SPI.h>
#include <aREST.h>
#include <avr/wdt.h>
```

After that, we have to declare the pins for the CC3000 breakout board:

```
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  8
#define ADAFRUIT_CC3000_CS    10
```

And the pins corresponding to your motor shield (that's where you need to enter your own values depending on the motor shield you are using):

```
int speed_motor1 = 6;
int speed_motor2 = 5;
int direction_motor1 = 7;
int direction_motor2 = 4;
```

We can now set the parameters that depends on your WiFi network. You will need to modify these lines of code with your WiFi network configuration:

```
#define WLAN_SSID       "yourNetwork"        // cannot be longer than 32 characters!
#define WLAN_PASS       "yourPassword"
#define WLAN_SECURITY   WLAN_SEC_WPA2 // This can be WLAN_SEC_UNSEC, WLAN_SEC_WEP, WL
```

We also need to create the CC3000 instance:

```
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS, ADAFRUIT_CC3000_IRQ, ADAFRUIT_
                                         SPI_CLOCK_DIV2);
```

We then create a server on our Arduino, that will accept the incoming connections coming from the network:

```
// The port to listen for incoming TCP connections
#define LISTEN_PORT           80

// Server instance
Adafruit_CC3000_Server restServer(LISTEN_PORT);
```

Now, in the setup() function of the sketch, there is a large number of functions to connect to the WiFi network that we won't detail here. We also start the WiFi server:

```
restServer.begin();
Serial.println(F("Listening for connections..."));
```

And enable the watchdog, so our sketch restarts automatically if there is any problem:

```
wdt_enable(WDTO_4S);
```

In the loop() function, we accept incoming connections and process them, and check if we are still connected to the network:

```
// Measure distance
distance = measure_distance(distance_sensor);

// Handle REST calls
Adafruit_CC3000_ClientRef client = restServer.available();
rest.handle(client);
wdt_reset();

// Check connection
if(!cc3000.checkConnected()){while(1){}}
wdt_reset();
```

Let’s now have a look at one of the functions we will use to control the robot. Let’s take the one that makes the robot goes forward:

```
int forward(String command) {

  send_motor_command(speed_motor1,direction_motor1,100,1);
  send_motor_command(speed_motor2,direction_motor2,100,1);
  return 1;
}
```

We can see that we simply call the function *send_motor_command* twice, once per motor. This function is also defined in the sketch:

```
void send_motor_command(int speed_pin, int direction_pin, int pwm, boolean dir)
{
  analogWrite(speed_pin,pwm); // Set PWM control, 0 for stop, and 255 for maximum speed
  digitalWrite(direction_pin,dir);
}
```

It’s now time to test the sketch. Make sure you modified it with your own WiFi parameters, and upload it to the robot. Open the Serial monitor, and check that the connection details are correctly printed, and note the IP address of your board.

Let’s it was 192.168.1.103, like in my case. You can now disconnect the USB cable from the robot. Then, go a web browser and type:

http://192.168.1.103/id

You should get the same answer as before, meaning the aREST API is working via WiFi. Now, just type:

http://192.168.1.103/forward

You should see the robot going forward at full speed. To stop it, just do the same with the stop command.

Building the web interface

Now, it’s great to be able to command our robot remotely, but it is not ideal: we still need to type in commands in a browser. This is why we are going to build an interface based on the powerful Node.js framework. We are going to see some insights about this interface, but of course you can just skip this whole part and go straight at the end where we use the robot interface.

For the interface, you will need to have Node.js installed on your computer. You can find all the installation instructions on the official Node website (http://adafru.it/dIP).

We first start with the main app.js file, which we will execute later. It starts by declaring that we are using the express module of Node.js:

```
var express = require('express');
var app = express();
```

We also define the port on which we will access our interface:

```
var port = 3000;
```

We also have to declare some things concerning the express application:

```
// View engine
app.set('view engine', 'jade');

// Set public folder
app.use(express.static(__dirname + '/public'));
```

What’s important now is to define the main route of the application, which is where the application will redirect us when we access it in the browser. Here, we will simply render the interface that we will define later:

```
// Serve interface
app.get('/', function(req, res){
  res.render('interface');
});
```

After that, we import the aREST node module, that will handle all the communication with our robot. We also add a device at the address of our robot:

```
var rest = require("arest")(app);

rest.addDevice('http','192.168.1.103');
```

Finally, we start the app and print a message in the console:

```
app.listen(port);
console.log("Listening on port " + port);
```

Let’s now see the interface file, that is located in the application /views subfolder. This is a file written in the Jade format, which is basically a way to simplify HTML. You don’t need to know all the details, just that we define one button per function of the robot, and also a field to print out the distance measured by the front sensor:

```
doctype
html
  head
    title Robot Control
    link(rel='stylesheet', href='/css/interface.css')
    link(rel='stylesheet', href='/css/flat-ui.css')
    script(src="/js/jquery-2.1.1.min.js")
    script(src="/js/interface.js")
  body
    .mainContainer
      .title Robot Control
      .buttonBlock
        button.btn.btn-block.btn-lg.btn-primary#1 Forward
      .buttonBlock
        button.btn.btn-block.btn-lg.btn-primary#2 Left
        button.btn.btn-block.btn-lg.btn-primary#3 Right
      .buttonBlock
        button.btn.btn-block.btn-lg.btn-primary#4 Backward
      .buttonBlock
        button.btn.btn-block.btn-lg.btn-danger#5 Stop
      .buttonBlock
        div.display#distance Distance:
      .buttonBlock
        div.status#wifiStatus Offline
```

Finally, we define a Javascript file to handle the interface, located in the /public/js folder of the application. For each button, we define an event in case the user click on it. For example, the first button is called forward, so we naturally route it to the forward function on the robot:

```
$("#1").click(function() {
  $.get('/robot/forward');
});
```

Finally, we refresh the status of the distance sensor & the connection indicator every second using this piece of code:

```
setInterval(function() {

    $.get('/robot/distance', function( json_data ) {
        if (json_data.distance){
            $("#distance").html("Distance: " + json_data.distance);
        }
        if (json_data.connected == 1){
            $("#wifiStatus").html("Online");
            $("#wifiStatus").css("color","green");
        }
        else {
            $("#wifiStatus").html("Offline");
            $("#wifiStatus").css("color","red");
        }
    })
    .fail(function() {
       $("#wifiStatus").html("Offline");
       $("#wifiStatus").css("color","red");
    });

}, 1000);
```

Note that all the code for this part can be found in the GitHub repository of the project:

https://github.com/openhardwaredrones/wifi-mobile-robot (http://adafru.it/f5F)

It's now time to test our application. Go the interface folder of the code you downloaded, and type:

```
sudo npm install express arest jade
```

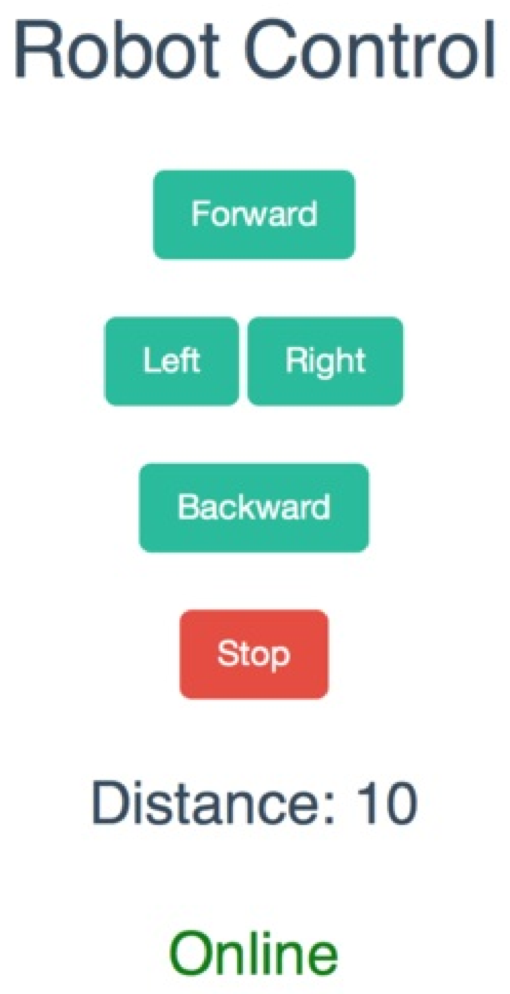
This will install the required modules for the application. You can now type:

```
node app.js
```

This will start the application, and you should get the confirmation message in the console. You will also see that the robot is added as a new device inside the application. You can now go to your favorite browser and go to:

http://locahost:3000/

You should see the interface being displayed inside the web browser:



You should see that the robot is immediately going online, and you should also see the measurement coming from the front sensor. Now, go ahead and press the buttons: you should see that the robot is reacting accordingly.

How to Go Further

Let’s summarise what we did in this project. We built an Arduino-based mobile robot, and controlled it via WiFi using the aREST API. We also built a simple interface based on Node.js to control the robot via a graphical interface.

How course, there are several ways to go further with this project. You can for example add more sensors to the robot, for example an accelerometer, and then also integrates them into the Node.js interface. Finally, you can also add more complex behaviour inside the robot code, for example automatically move the robot back when detecting an obstacle in front of it.