

IoT - Controlling a Raspberry Pi Robot over internet with HTML and shell scripts only

by **mjrovai** on May 28, 2016

Table of Contents

IoT - Controlling a Raspberry Pi Robot over internet with HTML and shell scripts only	1
Intro: IoT - Controlling a Raspberry Pi Robot over internet with HTML and shell scripts only	2
Step 1: Bill of Material	2
Step 2: Installing WiringPi library	2
Step 3: Controlling Motors with Raspberry Pi and WiringPi	3
Step 4: Controlling motor speed and turn ON/OFF a LED (optional)	7
Step 5: Installing the LIGHTTPD WebServer	8
Step 6: Creating an HTML page to control the Robot	9
Step 7: Assembly the Robot body	12
Step 8: Installing the RPi	13
Step 9: Conclusion	14
Related Instructables	15
Advertisements	15
Comments	15



Author: [mjrovai](http://mjrovai.com) MJRoBot.org

Engineer, Brazilian, Married to Ilza, Paula's father, Living in Santiago, Chile.

Intro: IoT - Controlling a Raspberry Pi Robot over internet with HTML and shell scripts only

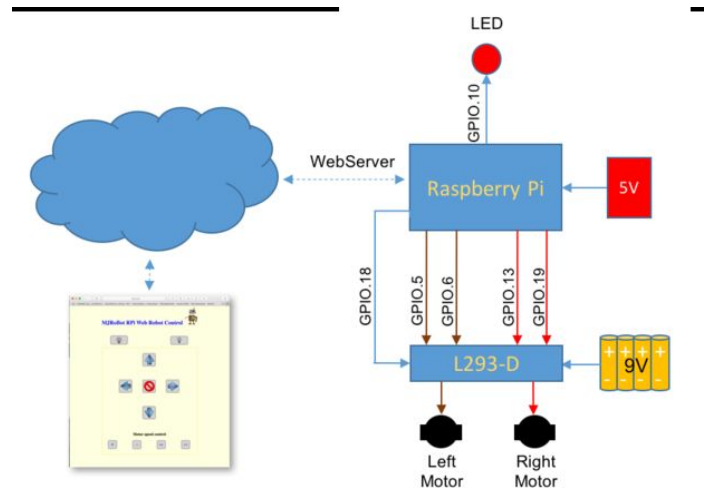
In this instructables, we will learn how to create a fully controlled IoT Raspberry Pi Robot. The idea is to control the robot (or anything), using low level commands written with shell scripts command straight from the HTML page. Not using higher level languages as Python for example, makes the robot very responsive and quick to act.

A great video tutorial about building such type of robots can be found below:

[Short Tutorial on How I Built my Raspberry PI Web Browser Controlled Robot](#), developed by Alan H.

A lot of ideas come from there. Thanks Alan for a great tutorial.

The block diagram shows the basic idea of the project. The RPi will be set as a WebServer and will receive commands from an HTML page. Those commands will change the GPIOs status, making the RPi control motors (direction and speed) and turn ON/OFF LEDs. As you can realize, the robot is in fact a particular case of an IoT project. You can control what you want. This Instructables intends to give you those ideas to take off from here.



Step 1: Bill of Material

- Raspberry Pi model 2 or 3
- L293-D H Bridge
- DC Motors (2x)
- 9V battery for motors
- 5V battery for RPi
- LEDs and a 330 ohms resistors
- Breadboard
- Wirings
- Acrylic support for the motors/electronics (can be a kit, CDs, etc.)

Step 2: Installing WiringPi library

WiringPi is a GPIO access library written in C for the Raspberry Pi. It is very ease to use and simplify a lot any project involving RPi and electronics.

The WiringPi library includes a command-line utility "**gpio**" (wiringpi.com/the-gpio-utility) that can be used to program and setup the GPIO pins. You can use this to read and write the pins and even use it to control them from **shell scripts**. It can be used in scripts to manipulate the GPIO pins – set outputs and read inputs. It's even possible to write entire programs just using the gpio command in a shell-script.

To Install WiringPi:

```
git clone git://git.drogon.net/wiringPi
```

```
cd wiringPi
```

```
./build
```

To print the version, use the command:

```
gpio -v
```

To read all the normally accessible pins and prints a table of their numbers (wiringPi, BCM_GPIO and physical pin numbers), you can use the command "**gpio read all**", that create a cross-reference chart, with their modes and current values. This command also will detect the version/model of your RPi and printout the pin diagram appropriate to your Pi.

```
gpio readall
```

The first monitor screenshot, shows the above 2 commands result.

<http://www.instructables.com/id/IoT-Controlling-a-Raspberry-Pi-Robot-Over-Internet/>

To setup pins (BCM_GPIO pin numbering scheme) as OUTPUT mode, for example the command must be used:

```
gpio -g mode 10 out
```

Once the pin has its mode defined, you can sign a logic state to the output:

```
gpio -g write 10 1
```

Install a LED from GPIO.10 to ground using the 330ohm resistor and anode and GND. Do some tests to be sure that everything is working. Check the monitor with gpio-readall command (see photo here).

Besides setting the pins as output or input, you can set some of them as an PWM output. This is the case of physical pin Pin 12 or GPIO.18.

to setup the pin:

```
gpio -g mode 18 pwm
```

or

```
gpio mode 1 pwm note: ("1") is the wPi id for GPIO.18
```

to setup a pwm value:

```
gpio pwm 1 XXX note: [XXX ia value between 0 —> 1023]
```

ex.:

```
gpio pwm 1 512 note: a motor or a LED will be at 50% duty-cycle
```

to remove the setup of this particular pin:

```
gpio unexport 1
```

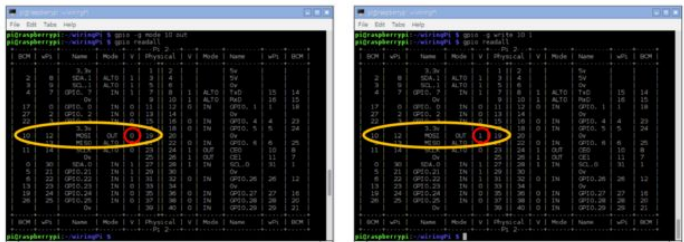
to remove all pin setup:

```
gpio unexportall
```

```
pi@raspberrypi:~/wiringPi
pi@raspberrypi:~/wiringPi $ gpio -v
gpio version: 2.32
Copyright (c) 2012-2015 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

Raspberry Pi Details:
Type: Pi 2, Revision: 01, Memory: 1024MB, Maker: Embest
* Device tree is enabled.
* This Raspberry Pi supports user-level GPIO access.
-> See the man-page for more details
-> ie. export WIRINGPI_GPIOMEM=1

pi@raspberrypi:~/wiringPi $ gpio readall
+-----+-----+-----+-----+-----+-----+
| BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
+-----+-----+-----+-----+-----+-----+
| 2 | 8 | 3.3v | ALTO | 1 | 1 | 2 | | 5v | | | |
| 3 | 9 | SDA.1 | ALTO | 1 | 3 | 4 | | 5v | | |
| 4 | 7 | SCL.1 | ALTO | 1 | 5 | 6 | | 0v | | |
| 17 | 0 | GPIO. 7 | IN | 0 | 7 | 8 | 1 | ALTO | TxD | 15 | 14 |
| 27 | 2 | GPIO. 0 | IN | 0 | 9 | 10 | 1 | ALTO | RxD | 16 | 15 |
| 22 | 3 | GPIO. 2 | IN | 0 | 11 | 12 | 0 | IN | GPIO. 1 | 1 | 18 |
| 10 | 12 | GPIO. 3 | IN | 0 | 13 | 14 | 0 | IN | 0v | | |
| 9 | 13 | 3.3v | ALTO | 0 | 15 | 16 | 0 | IN | GPIO. 4 | 4 | 23 |
| 11 | 14 | MOSI | ALTO | 0 | 17 | 18 | 0 | IN | GPIO. 5 | 5 | 24 |
| 0 | 30 | MISO | ALTO | 0 | 19 | 20 | 0 | IN | 0v | | |
| 5 | 21 | SCLK | ALTO | 0 | 21 | 22 | 0 | IN | GPIO. 6 | 6 | 25 |
| 6 | 22 | SDA.0 | IN | 1 | 23 | 24 | 1 | OUT | CEO | 10 | 8 |
| 13 | 23 | SCL.0 | IN | 1 | 25 | 26 | 1 | OUT | CE1 | 11 | 7 |
| 19 | 24 | SDA.1 | IN | 1 | 27 | 28 | 1 | IN | SCL.0 | 31 | 1 |
| 26 | 25 | GPIO.21 | IN | 1 | 29 | 30 | 0 | IN | 0v | | |
| 13 | 22 | GPIO.22 | IN | 1 | 31 | 32 | 0 | IN | GPIO.26 | 26 | 12 |
| 19 | 24 | GPIO.23 | IN | 0 | 33 | 34 | 1 | OUT | 0v | | |
| 26 | 25 | GPIO.24 | IN | 0 | 35 | 36 | 0 | IN | GPIO.27 | 27 | 16 |
| | | GPIO.25 | IN | 0 | 37 | 38 | 0 | IN | GPIO.28 | 28 | 20 |
| | | 0v | | | 39 | 40 | 0 | IN | GPIO.29 | 29 | 21 |
+-----+-----+-----+-----+-----+-----+
| BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
+-----+-----+-----+-----+-----+-----+
```



Step 3: Controlling Motors with Raspberry Pi and WiringPi

At this point, the WiringPi library is installed and you can control any GPIO directly from the command line at your RPi monitor. The next step is to create a logic to control the motors. To do that, we will use an H-Bridge, the L-293-D. This H-Bridge can control 2 motors. It has for each motors, 3 inputs:

- Motor Left: "enable"; "motor+" and "motor-"
- Motor Right: "enable"; "motor+" and "motor-"

Both "motors enables" input will be connected together and will be controlled by GPIO.18. This pin will be responsible for the speed control. If you do not want control the speed, leave those pins in HIGH, for example. we will discuss this further in this step.

Let's make a convention that if we want the motor LEFT to run forward, we must setup motor+ as HIGH and motor- as LOW. To run on a reverse direction, we must do the opposite: motor- as HIGH and motor+ as LOW.

The best way to really define the correct inputs for controlling the motor direction, is to test them when the real motor is assembly.

Let's assigned the GPIOs for H-Bridges inputs:

<http://www.instructables.com/id/loT-Controlling-a-Raspberry-Pi-Robot-Over-Internet/>

- Motor left +: GPIO.5
- Motor left - : GPIO.6
- Motor right +: GPIO.13
- Motor right - : GPIO.19

Based on the above assumptions, a logic table can be built with the possible levels to be assigned to the GPIOs (see table picture). The next step is to create shell scripts to run the motors

Every script file is essentially plain text. When a text file is attempted to be executed, shells will parse through them for clues as to whether they're scripts or not, and how to handle everything properly. Because of this, there are a few guidelines you need to know.

- Every script should begin with "#!/bin/bash" (The Hash-Bang Hack)
- Every new line is a new command
- Comment lines start with a #
- Commands are surrounded by ()

When a shell parses through a text file, the most direct way to identify the file as a script is by making your first line: #!/bin/bash (The Hash-Bang Hack). If you use another shell, substitute its path here. Comment lines start with hashes (#), but adding the bang (!) and the shell path after it is a sort of hack that will bypass this comment rule and will force the script to execute with the shell that this line points to.

For example, to create a shell script to run the motors "Forward", based on the above table, we must create the file below (use the best editor for you. I am using NANO for that):

sudo nano forward.cgi

```
#!/bin/bash

gpio -g write 5 1
gpio -g write 6 0
gpio -g write 13 1
gpio -g write 19 0
.
```

Once the script is created, we must give it permission to be executed:

sudo chmod 755 forward.cgi

Now, to execute the script:

sudo ./forward.cgi

4 LEDs were used for testing the scripts, the real motors will be added on a further step. If works, the correspondent LEDs must be ON (see photo).

Note that I use .cgi as the file extension. CGI means "Common Gateway Interface". It is a standard way for web servers to interface with executable programs installed on a server that generate web pages dynamically. Such programs are known as CGI scripts or simply CGIs; they are usually written in a scripting language, but can be written in any programming language.

Going on, the same idea must be applied for the other possibilities of the previous table:

sudo nano stop.cgi

```
#!/bin/bash

gpio -g write 5 0
gpio -g write 6 0
gpio -g write 13 0
gpio -g write 19 0
.
```

sudo nano reverse.cgi

```
#!/bin/bash

gpio -g write 5 0
gpio -g write 6 1
gpio -g write 13 0
gpio -g write 19 1
.
```

sudo nano left.cgi

```
#!/bin/bash

gpio -g write 5 0
gpio -g write 6 1
```

<http://www.instructables.com/id/loT-Controlling-a-Raspberry-Pi-Robot-Over-Internet/>

```
gpio -g write 13 1
```

```
gpio -g write 19 0
```

```
.
```

```
sudo nano right.cgi
```

```
#!/bin/bash
```

```
gpio -g write 5 1
```

```
gpio -g write 6 0
```

```
gpio -g write 13 0
```

```
gpio -g write 19 1
```

```
.
```

Once the scripts are created you must give them permission to be executed, same as was done with forward.cgi

```
sudo chmod 755 stop.cgi
```

```
sudo chmod 755 reverse.cgi
```

```
sudo chmod 755 left.cgi
```

```
sudo chmod 755 right.cgi
```

Now, execute some tests to confirm that all is working:

```
./forward.cgi
```

```
./left.cgi
```

```
./reverse.cgi
```

```
./right.cgi
```

```
./stop.cgi
```

It is a good practice that we have a specific directory for the programs used and call it "bin". So, to save the scripts that we will use in the project, we must create a directory like cgi-bin containing all executable scripts (or binary files). For example, /var/www/cgi-bin.

So, let's create the directory www under var, where our webpage will be located and under it, the cgi-bin directory with the scripts:

```
sudo mkdir /var/www
```

```
sudo mkdir /var/www/cgi-bin
```

Now, let's move all files to this new directory:

```
sudo mv /*.sgi /var/www/cgi-bin
```

```
cd /var/www/cgi-bin
```

Using the line command ls, you can see the files created (see photo).

```
ls
```

One last thing before we move to another step. If you reboot the RPi, the GPIOs will return to their default condition that is INPUT. So, we must change the script /etc/rc.local that is executed at any Raspberry's start. Just before the last command at the script ==> exit 0, we must include the GPIOs mode commands:

```
sudo nano /etc/rc.local
```

```
...
```

```
gpio -g mode 5 out
```

```
gpio -g mode 6 out
```

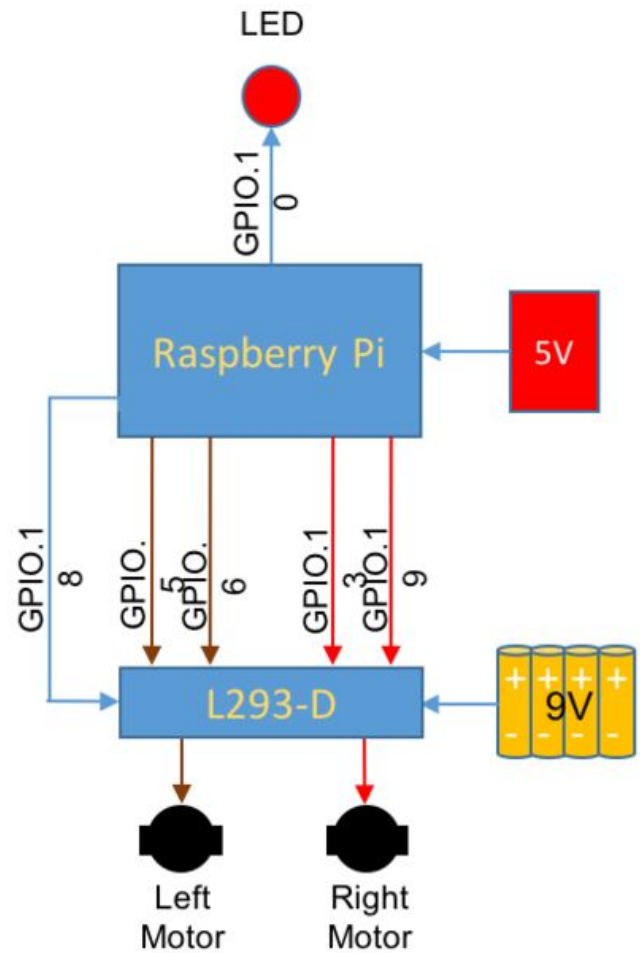
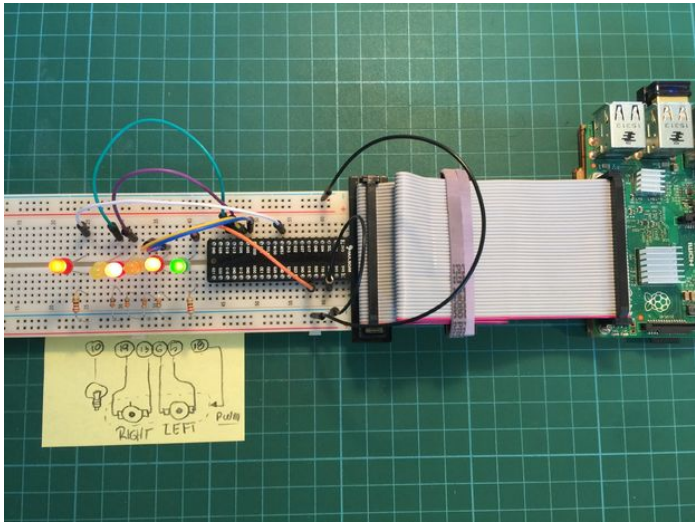
```
gpio -g mode 13 out
```

```
gpio -g mode 19 out
```

```
...
```

```
exit 0
```

Now, anytime that the RPi starts, it will be ready to control the designed outputs.



	LEFT MOTOR		RIGHT MOTOR	
Stop	L	L	L	L
Right	H	L	L	H
Left	L	H	H	L
Reverse	L	H	L	H
Forward	H	L	H	L
GPIO.	5	6	13	19

```

pi@raspberrypi: /var/www/cgi-bin
File Edit Tabs Help
pi@raspberrypi:/var/www/cgi-bin $
pi@raspberrypi:/var/www/cgi-bin $ ls
forward.cgi left.cgi reverse.cgi right.cgi stop.cgi
pi@raspberrypi:/var/www/cgi-bin $
  
```

Step 4: Controlling motor speed and turn ON/OFF a LED (optional)

Once we will use an H-Bridge, the L293-D for motor control, we must decide if besides direction, we also want control direction.

Two possibilities here:

- Fixed speed: Connection of H-Bridge Enable Pins (1 and 9) to +5V (full speed) or any other proportional value using a voltage divisor with 2 resistors
- Enable Pins 1 and 9 connected to Raspberry Pi GPIO.18

Usually you can keep the speed constant, but let's explore how to control it. To to that, a PWM signal must be used. Let's create a group of scripts, same as we did with "directions" (the values must be tested for your specific motor):

sudo nano nospeed.cgi

```
#!/bin/bash
```

```
gpio pwm 1 0
```

```
.
```

sudo nano lowspeed.cgi

```
#!/bin/bash
```

```
gpio pwm 1 250
```

```
.
```

sudo nano regularspeed.cgi

```
#!/bin/bash
```

```
gpio pwm 1 512
```

```
.
```

sudo nano highspeed.cgi

```
#!/bin/bash
```

```
gpio pwm 1 1023
```

```
.
```

Once the scripts are created you must give them permission to be executed, same as was done with forward.cgi

sudo chmod 755 nospeed.cgi

sudo chmod 755 lowspeed.cgi

sudo chmod 755 regularspeed.cgi

sudo chmod 755 highspeed.cgi

Now, it is only execute some tests to confirm that all is working:

./lowspeedcgi

./regularspeed.cgi

./highspeed.cgi

./nospeedcgi

In my case, at the stage I have a LED connected to GPIO.18, so I can see by the intensity of its bright, that the command is working.

Last, but not least, let's have an extra script to control a digital output, to turn ON or OFF a lamp, for example. We will use the GPIO.10 for that:

sudo nano llighton.cgi

```
#!/bin/bash
```

```
gpio -g write 10 1
```

```
.
```

sudo nano llightoff.cgi

```
#!/bin/bash
```

```
gpio -g write 10 0
```

```
.
```

sudo chmod 755 llighton.cgi

sudo chmod 755 llightoff.cgi

Same as before if you decide to use those additional GPIOs, you must change the script /etc/rc.local:

<http://www.instructables.com/id/loT-Controlling-a-Raspberry-Pi-Robot-Over-Internet/>


```
sudo nano /etc/rc.local
```

...

```
gpio -g mode 5 out
```

```
gpio -g mode 6 out
```

```
gpio -g mode 13 out
```

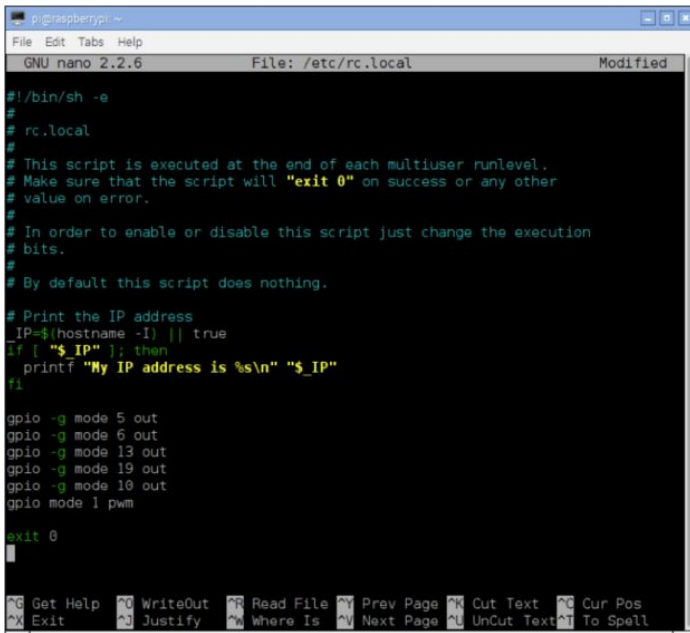
```
gpio -g mode 19 out
```

```
gpio -g mode 10 out
```

```
gpio mode 1 pwm
```

```
exit 0
```

It is good to have the gpio mode 1 pwm as the last line before exit 0

A screenshot of a terminal window on a Raspberry Pi. The window title is 'pi@raspberrypi: ~'. The terminal shows the GNU nano 2.2.6 editor editing the file /etc/rc.local. The content of the file is as follows:

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.
#
# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
    printf "My IP address is %s\n" "$_IP"
fi

gpio -g mode 5 out
gpio -g mode 6 out
gpio -g mode 13 out
gpio -g mode 19 out
gpio -g mode 10 out
gpio mode 1 pwm

exit 0
```

The bottom of the terminal shows various nano editor shortcuts like 'Get Help', 'WriteOut', 'Read File', etc.

Step 5: Installing the LIGHTTPD WebServer

We will install LIGHTTPD that is a very "light" and fast WebServer (can be used instead of Apache for example). As described on its [lighttpd wiki page](#), "Lighttpd is a secure, fast, compliant, and very flexible web-server that has been optimized for high-performance environments. It has a very low memory footprint compared to other WebServers and takes care of cpu-load.

Let's install Lighttpd web server and components:

```
sudo apt-get -y install lighttpd
```

```
sudo lighttpd-enable-mod cgi
```

```
sudo lighttpd-enable-mod fastcgi
```

By default, Lighttpd is looking for a **index.html** page at **/var/www/html**. We will change it, so the index.html will be placed under /var/www. For that, we must edit the Lightd config file:

```
sudo nano /etc/lighttpd/lighttpd.conf
```

change:

```
server.document-root = "/var/www/html"
```

by:

```
server.document-root = "/var/www/"
```

In order to aloud this change to take effect, we must stop and re-start the web server:

```
sudo /etc/init.d/lighttpd stop
```

```
sudo /etc/init.d/lighttpd start
```

At this point the web server is running and if a page index.html is located at /var/www, we can access it from any browser, typing the RPi IP address:

let's create a simple test webpage (Note that for testing, previously I moved a png file - /images/robot52.png - to image directory):

<http://www.instructables.com/id/loT-Controlling-a-Raspberry-Pi-Robot-Over-Internet/>


```
cd /var/www
```

```
sudo nano index.html
```

```
<html>
<head>
<script Language="Javascript">
var xmlhttp;
xmlhttp=new XMLHttpRequest();

</script>
</head>
<style>
body {background-color: lightyellow}
h1 {color:blue}

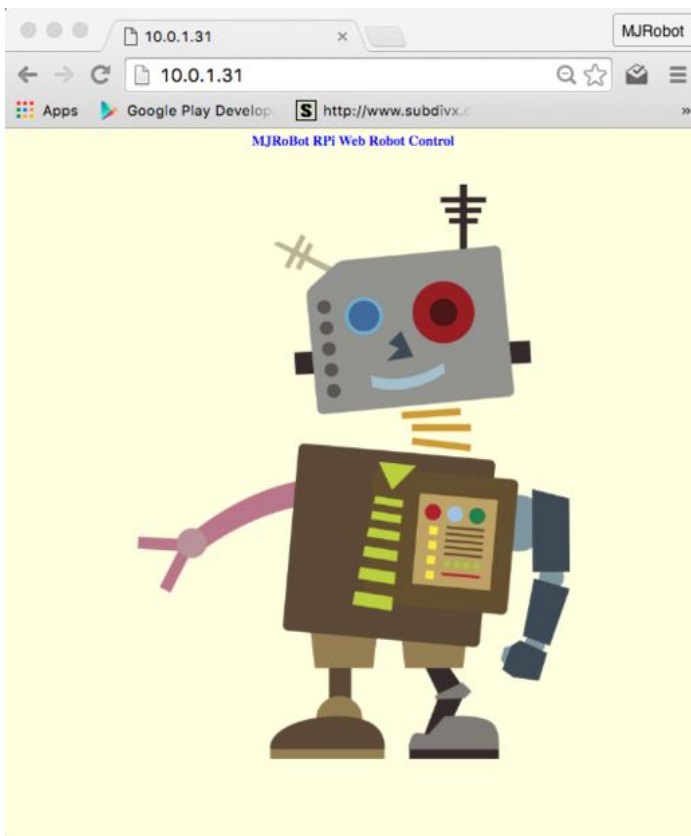
</style>
<body>
    <div style="text-align:center">
    <h1>MJRoBot RPi Web Robot Control</h1>
<br><br>

</body>
</html>
```

Once you finish the page edition, save it and change the permissions:

```
sudo chmod 755 index.html
```

The final result can be seen at screenshot above.



Step 6: Creating an HTML page to control the Robot

Let's think about a simple design for our page. What commands we may have?

1. Two buttons for turn ON and OFF the lights ==> will work with the scrips: lighton.cgi and lighoff.cgi
2. Five buttons for motor direction control ==> will work with the scrips: forward.cgi, stop.cgi, left.cgi, right.cgi and reverse.cgi
3. Three buttons for motor speed control ==> will work with the scrips: lowspeed.cgi, regularspeed.cgi and high speed.cgi

Using the index.html that we created in the last step, let's include buttons that will call functions to execute the scripts.

For example, let's create a button to light ON the LED (GPIO.10):

```
button {
    color: blue;
    background:lightgrey;
    border: 1px solid #000;
    border-radius: 8px;
}
```

<http://www.instructables.com/id/loT-Controlling-a-Raspberry-Pi-Robot-Over-Internet/>

```
<button style="height: 50px; width: 100px" onclick="lighton()"></button>
```

The above HTML would create a rounded button with a "lightened lamp" as an image (see the final page above).

When the button is pressed, due the command "onclick=lighton()", the function "lighton()" is called:

```
function lighton()
{
    xmlhttp.open("GET","cgi-bin/lighton.cgi",true);
    xmlhttp.send();
}
```

And once the function lighton() is called, the script lighton.cgi is executed and "voilà", the LED will be ON.

The same procedure must be used for all buttons. There are some HTML functions that will organize the look&fill that you can realize looking the full HTML page.

The HTML source can be seen bellow (test the page with the LEDS setup used on the previous steps):

```
# <html> <===== take the # out before execution
<head>
<script Language="Javascript">
var xmlhttp;
xmlhttp=new XMLHttpRequest();

function lighton()
{
    xmlhttp.open("GET","cgi-bin/lighton.cgi",true);
    xmlhttp.send();
}

function lightoff()
{
    xmlhttp.open("GET","cgi-bin/lightoff.cgi",true);
    xmlhttp.send();
}

function forward()
{
    xmlhttp.open("GET","cgi-bin/forward.cgi",true);
    xmlhttp.send();
}

function stop()
{
    xmlhttp.open("GET","cgi-bin/stop.cgi",true);
    xmlhttp.send();
}

function left()
{
    xmlhttp.open("GET","cgi-bin/left.cgi",true);
    xmlhttp.send();
}

function right()
{
    xmlhttp.open("GET","cgi-bin/right.cgi",true);
    xmlhttp.send();
}

function reverse()
{
    xmlhttp.open("GET","cgi-bin/reverse.cgi",true);
    xmlhttp.send();
}

function lowspeed()
{
    xmlhttp.open("GET","cgi-bin/lowspeed.cgi",true);
    xmlhttp.send();
}

function regularspeed()
{
    xmlhttp.open("GET","cgi-bin/regularspeed.cgi",true);
    xmlhttp.send();
}

function highspeed()
{
    xmlhttp.open("GET","cgi-bin/highspeed.cgi",true);
    xmlhttp.send();
}

function nospeed()
{
    xmlhttp.open("GET","cgi-bin/nospeed.cgi",true);
    xmlhttp.send();
}
</script>
</head>
<style>
body {background-color: lightyellow}
h1 {color:blue}

button {
    color: blue;
    background:lightgrey;
    border: 1px solid #000;
    border-radius: 8px;
    position: center;
}

</style>
<body>
    <div style="text-align:center">
        <h1> MJRoBot RPi Web Robot Control     </h1>
    </div>
</body>
```

<http://www.instructables.com/id/IoT-Controlling-a-Raspberry-Pi-Robot-Over-Internet/>

```

<br><br>
<button style="height: 50px; width: 100px" onclick="lighton()"></button>
<img hspace="20" style="padding-left: 200px">
<button style="height: 50px; width: 100px" onclick="lightoff()"></button>
<br><br>

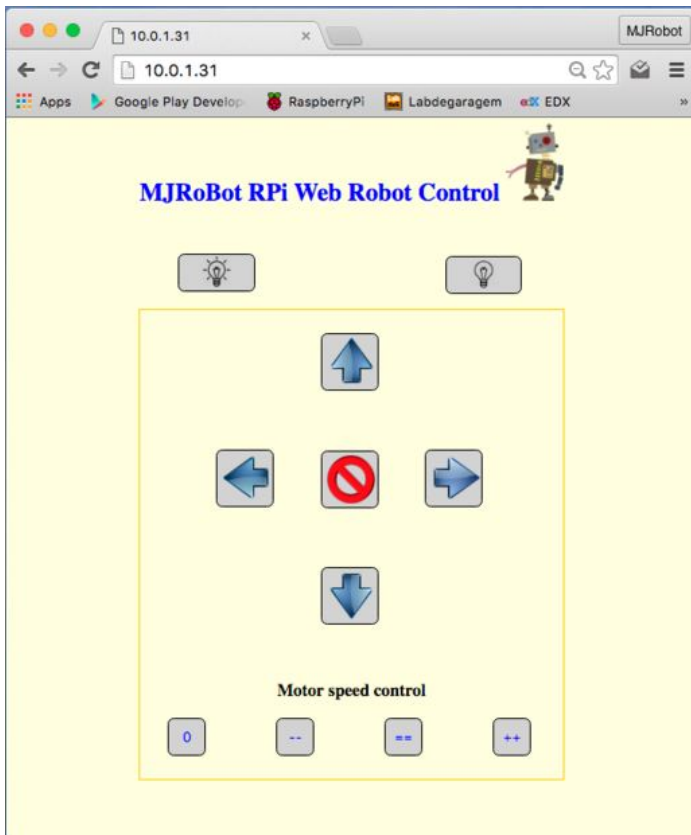
<span style="display:inline-block;padding:5px;border:1px solid #fc0; font-size: 140%;font-weight: bold;">
<br>
<button style="height: 75px; width: 75px" onclick="forward()"></button>
<br><br><br><br>
<img hspace="10" style="padding-left: 5px">
<button style="height: 75px; width: 75px" onclick="left()"></button>
<img hspace="20" style="padding-left: 10px">
<button style="height: 75px; width: 75px" onclick="stop()"></button>
<img hspace="20" style="padding-left: 10px">
<button style="height: 75px; width: 75px;" onclick="right()"></button>
<img hspace="10" style="padding-left: 5px">
<br><br><br><br>
<button style="height: 75px; width: 75px" onclick="reverse()"></button>

<br><br><br>
<p>Motor speed control</p>
<img hspace="10" style="padding-left: 5px">
<button style="height: 50px; width: 50px; font-size: 18px" onclick="nopeed()">0</button>
<img hspace="30" style="padding-left: 20px">
<button style="height: 50px; width: 50px; font-size: 18px" onclick="lowspeed()">--</button>
<img hspace="30" style="padding-left: 20px">
<button style="height: 50px; width: 50px; font-size: 18px" onclick="regularspeed()">==</button>
<img hspace="30" style="padding-left: 20px">
<button style="height: 50px; width: 50px; font-size: 18px" onclick="highspeed()">++</button>
<img hspace="10" style="padding-left: 5px">

<br><br>
</span>

</div>
</body>
</html>

```

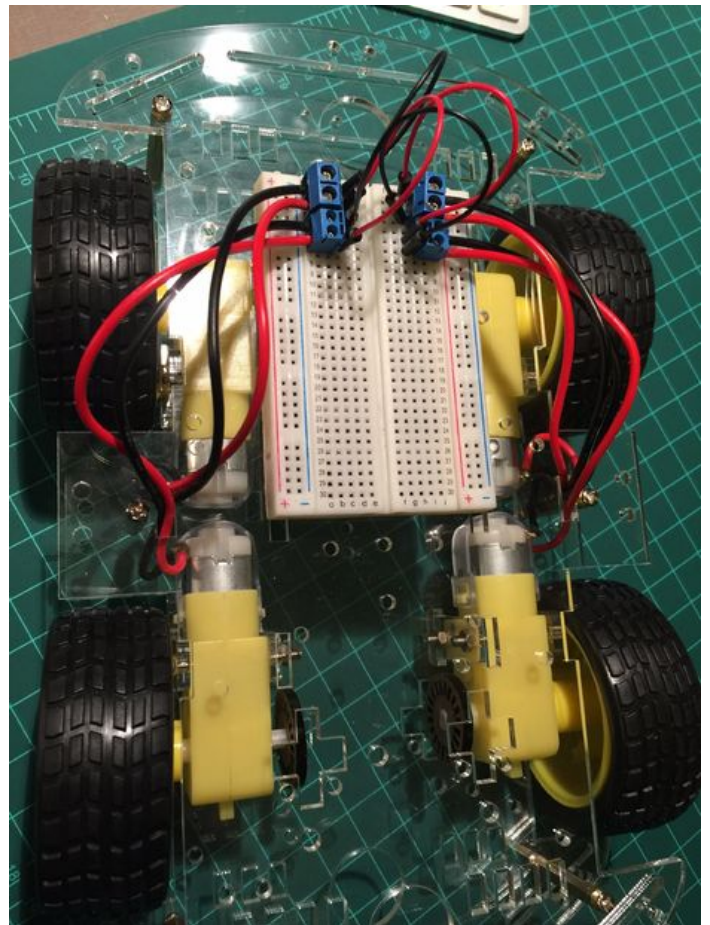
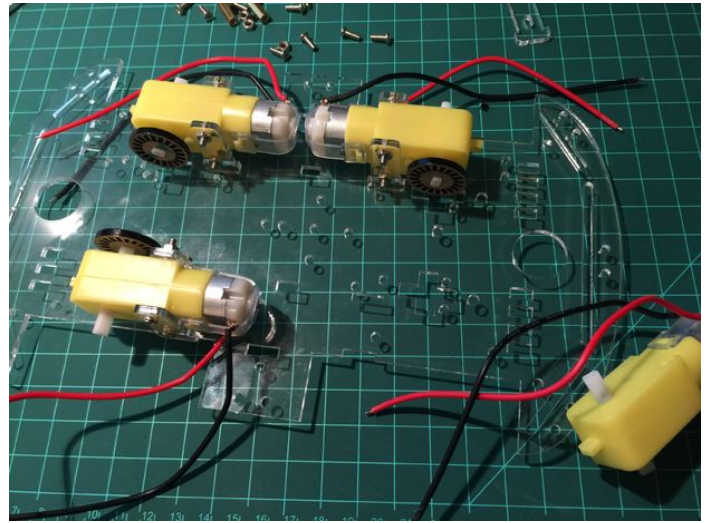
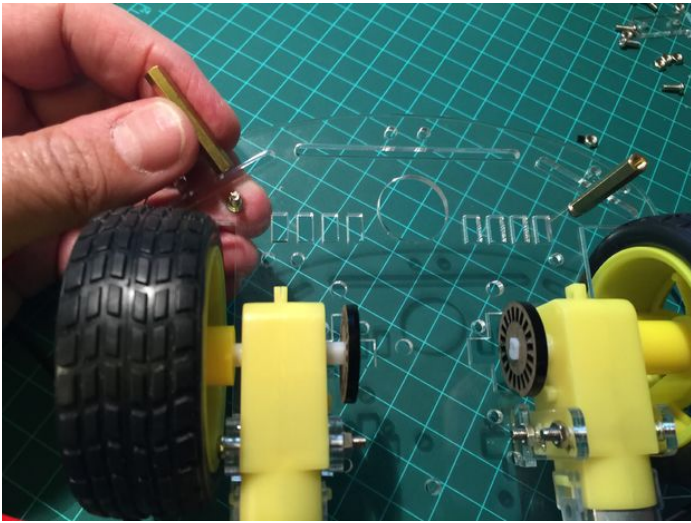


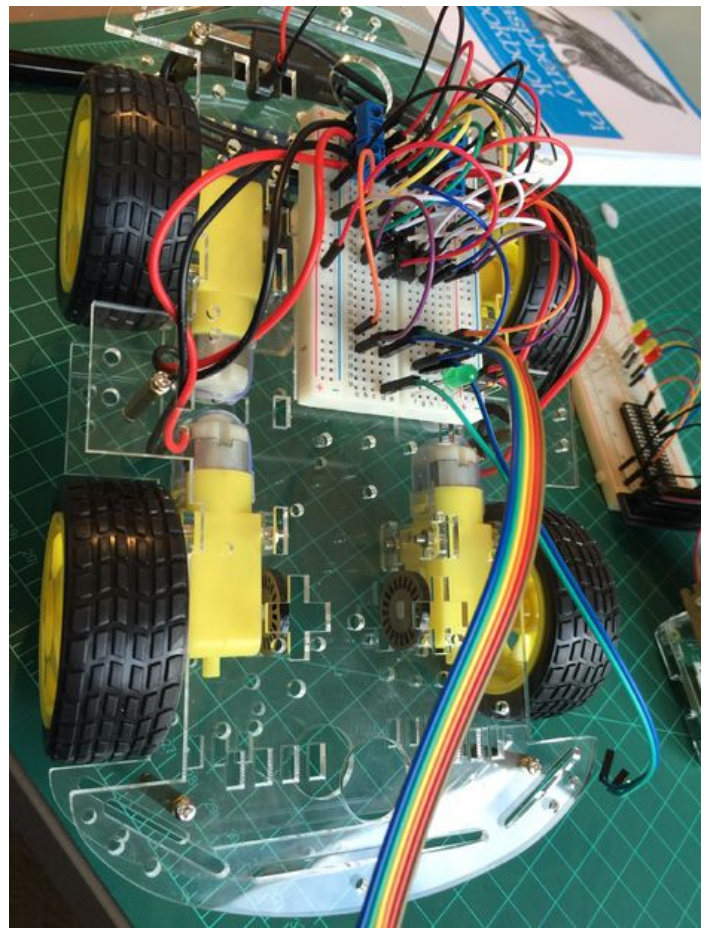
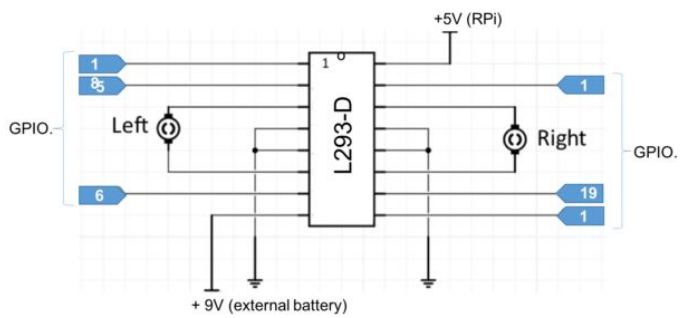
Step 7: Assembly the Robot body

First thing is to find a platform. I had a 4WD kit with 4 DC motors. I realized that it is not easy to control the 4WD and once the idea here is not a mobility analyses, I assembled the full car, but for real test I will only use 2 of the wheels and a caster (took from an used deodorant roll).

After the Body and motors are in place, It is time to include a protoboard, make the motor connections and test it. Use the external battery to test each one of the motors.

The motors will be controlled by a H-Bridge L293-D as shown at above diagram. At this point you can also make tests, controlling the motors using +5V and GND probes at H-Bridge inputs, simulating the RPI GPIOs.





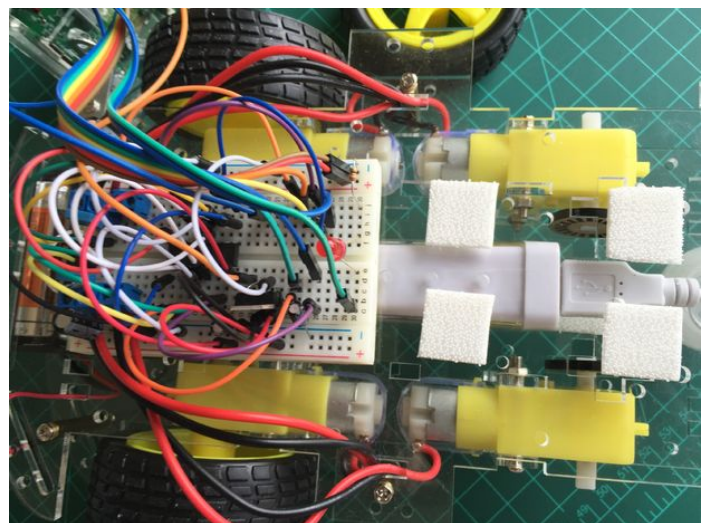
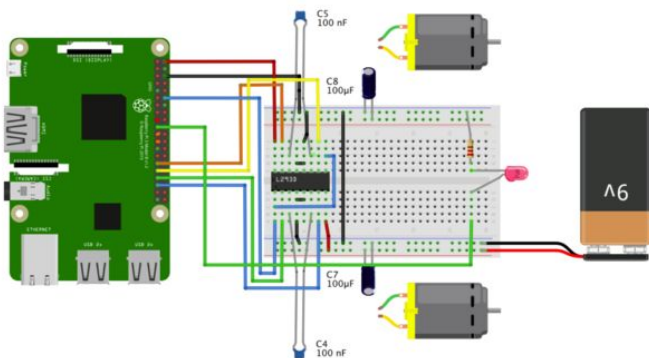
Step 8: Installing the RPi

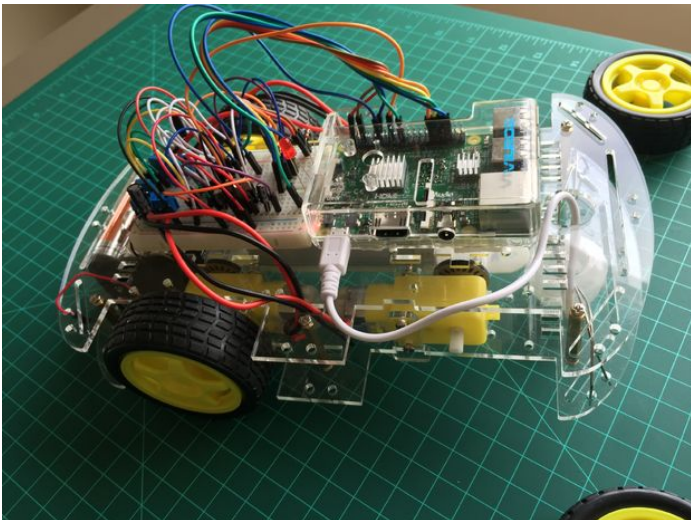
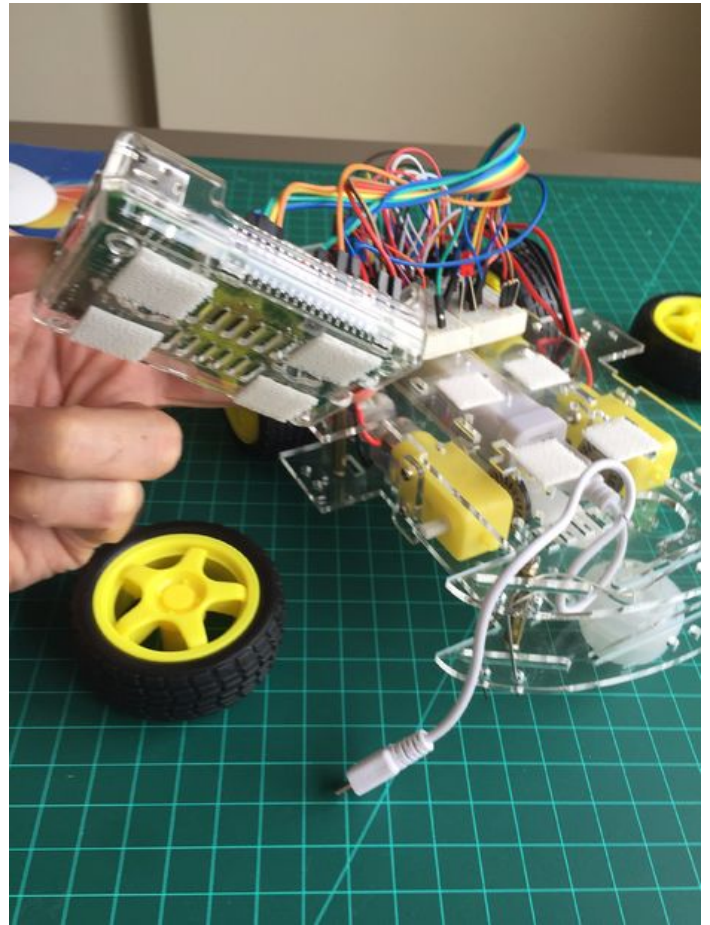
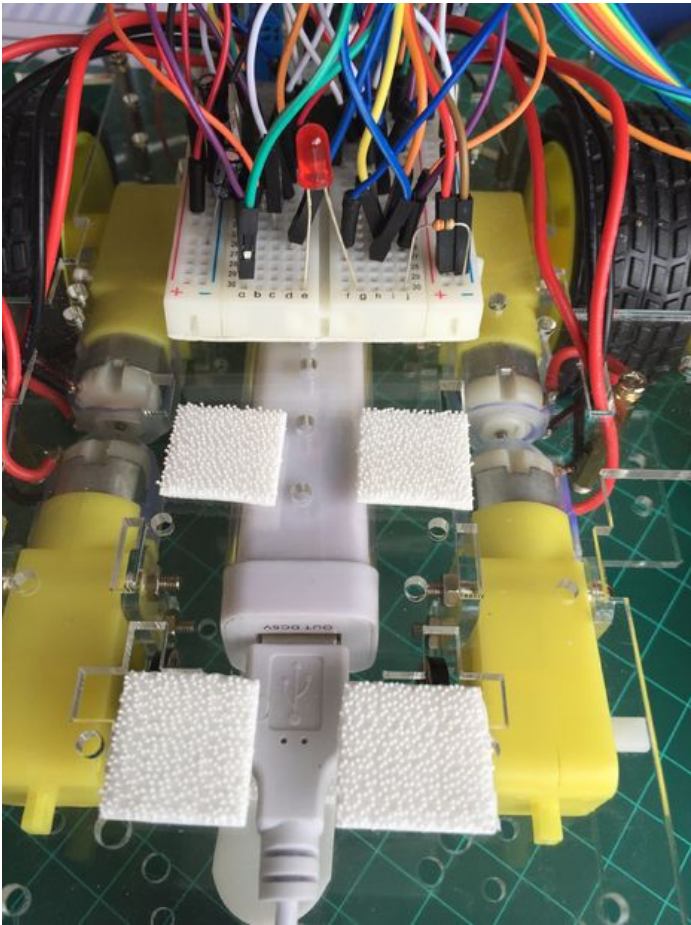
Once all is working it is time to add the RPi. The complete circuit is shown above.

I install the RPi 5V battery between the motors at lower level chassis. The RPi is on top.

Make all cable connections and turn on the RPi. If all the previous steps were OK, you can control the robot, using the RPi IP address. Open your favorite WebBrowser and safe trip!

Bellow, a video where the robot is tested using the Web Page:





Step 9: Conclusion

As discussed at introduction, the robot is almost an excuse to develop an IoT project. what we did here is control the RPi GPIOs over the internet. We can control anything starting from there!

My next Instructable will explore the PiCam. How to streaming video and also how to control the camera position using servos. The goal will be to merge both projects, having a rover that can be controlled by internet, but receiving video streaming.

The video bellow show how it will end:

As always, I hope this project can help others find their way in the exciting world of electronics, robotics, Raspberry Pi and IoT!

For more projects, please visit my blog: MJRoBot.org

Saludos from the south of the world! See you at my next instructable!

Thank you

<http://www.instructables.com/id/loT-Controlling-a-Raspberry-Pi-Robot-Over-Internet/>



Related Instructables



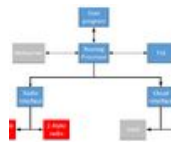
Smart Glasses to Help the Blind, With Pivthead LiveModPro and Intel Edison by PaulG166



My Physical Web Space by Dimitris Platis



Host your own blog from a \$25 Raspberry Pi computer by lightning9



Intel Edison Data Oriented Gateway Engine by bunneydude



EROBOT by kalyanbojanapu



How to send SMS from a PHP website through HTTP by using Raspberry Pi by LgnTurner

Comments