UNIVERSIDADE PRESBITERIANA MACKENZIE

ESCOLA DE ENGENHARIA

ENGENHARIA ELÉTRICA

FELIPE BONI VILALTA

EMERSON CARDOSO DE SOUSA

JOÃO GABRIEL DOS SANTOS SILVA

MATHEUS ANDRADE SANTANA

TECHTRASH

Código .ino: <https://github.com/MatheusAndrade1/TechTrash>

Apresentação: <https://youtu.be/M0NXbwhdtmw>

São Paulo

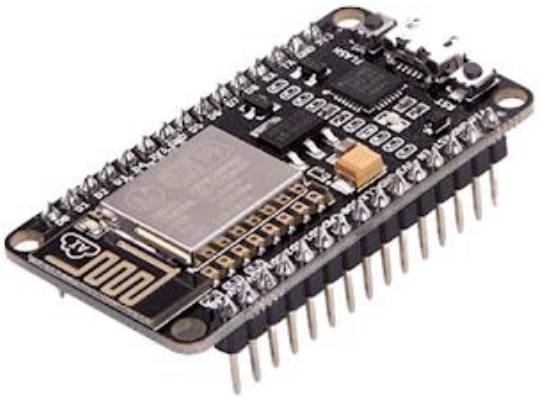
2018

Índice

1. ESP8266 NodeMCU WiFi Devkit

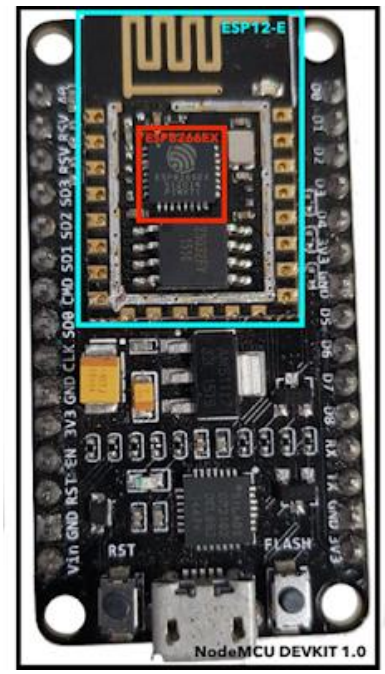
Deste longo nome, o termo NodeMCU refere-se ao firmware de todo esse hardware, enquanto a placa é conhecida como Devkit. Embarcado em todo este hardware, está o ESP-12E, placa desenvolvida pela AI-THINKER, sendo constituída por um ESP8266EX.

Figura 1 - A placa ESP8266 NodeMCU WiFi Devkit



O ESP8266EX é um microcontrolador desenvolvido pela Espressif que possui WiFi integrado e baixo consumo de energia. Sua arquitetura é baseada num sistema de 32 bits com clock máximo de 160 MHz.

Figura 2 - Componentes da placa Devkit



FONTE: KOYANAGI, Fernando.

Especificações:

- Tensão: 3.3 V
- Wi-Fi Direct (P2P), soft-AP
- Consumo de corrente: 10 µA~170 mA
- Memória FLASH: 512 KB
- Protocolo TCP/IP integrado
- Microprocessador: Tensilica L106 32-bit 80~160 MHz
- RAM: 32KB + 80KB
- GPIOs: 17
- +19,5 dBm output power em 802.11b
- 802.11 com suporte b/g/n

1.1. Detalhes técnicos - ESP8266EX:

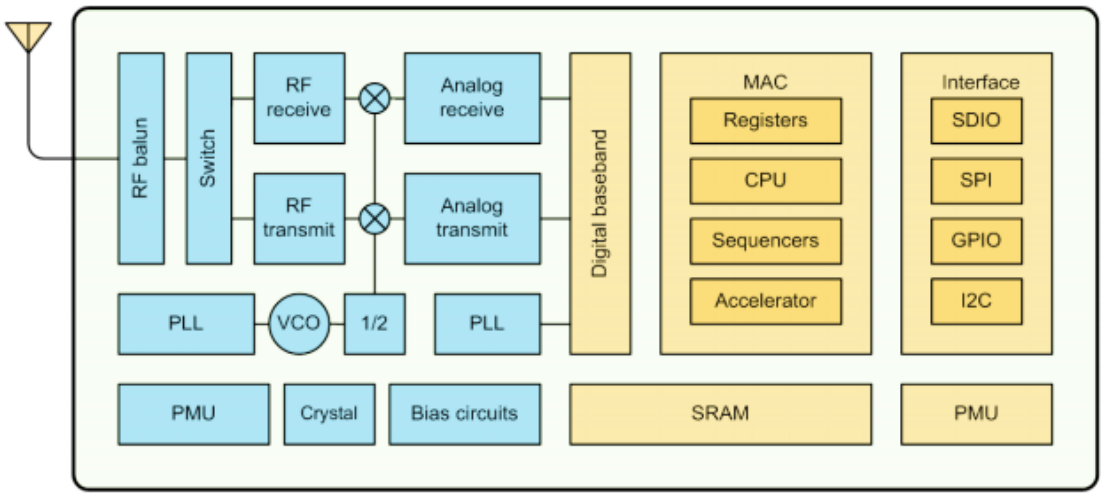
Abaixo está uma tabela com os parâmetros de operação do ESP8266EX:

Categories	Items	Values
WiFi Parameters	Certificates	FCC/CE/TELEC/SRRC
	WiFi Protocles	802.11 b/g/n
	Frequency Range	2.4G-2.5G (2400M-2483.5M)
	Tx Power	802.11 b: +20 dBm
		802.11 g: +17 dBm
		802.11 n: +14 dBm
	Rx Sensitivity	802.11 b: -91 dbm (11 Mbps)
		802.11 g: -75 dbm (54 Mbps)
		802.11 n: -72 dbm (MCS7)
	Types of Antenna	PCB Trace, External, IPEX Connector, Ceramic Chip
Hardware Parameters	Peripheral Bus	UART/SDIO/SPI/I2C/I2S/IR Remote Control
		GPIO/PWM
	Operating Voltage	3.0~3.6V
	Operating Current	Average value: 80mA
	Operating Temperature Range	-40°~125°
	Ambient Temperature Range	Normal temperature
	Package Size	5x5mm
	External Interface	N/A
Software Parameters	WiFi mode	station/softAP/SoftAP+station
	Security	WPA/WPA2
	Encryption	WEP/TKIP/AES
	Firmware Upgrade	UART Download / OTA (via network)
	Software Development	Supports Cloud Server Development / SDK for custom firmware development
	Network Protocols	IPv4, TCP/UDP/HTTP/+A1:C25FTP

FONTE: (ESPRESSIF SYSTEMS IOT TEAM, 2015)

Abaixo é apresentado o diagrama em blocos do ESP8266EX.

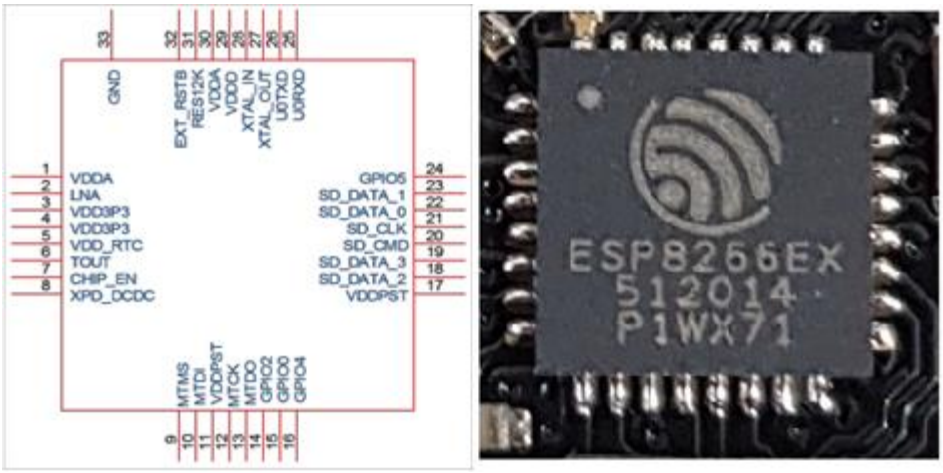
Figura 3 - Diagrama em blocos do ESP8266EX.



FONTE: (ESPRESSIF SYSTEMS IOT TEAM, 2015)

O ESP8266EX tem um encapsulamento que conta com 33 pinos, cuja imagem e descrição estão abaixo:

Figura 4 - Encapsulamento do ESP8266EX



FONTE: (ESPRESSIF SYSTEMS IOT TEAM, 2015)

Pino	Nome	Tipo	Função
1	VDDA	P	Analog Power 3.0 ~3.6V
2	LNA	I/O	RF Antenna Interface. Chip Output Impedance=50Ω No matching required but we recommend that the π-type matching network is retained.
3	VDD3P3	P	Amplifier Power 3.0~3.6V
4	VDD3P3	P	Amplifier Power 3.0~3.6V
5	VDD_RTC	P	NC (1.1V)

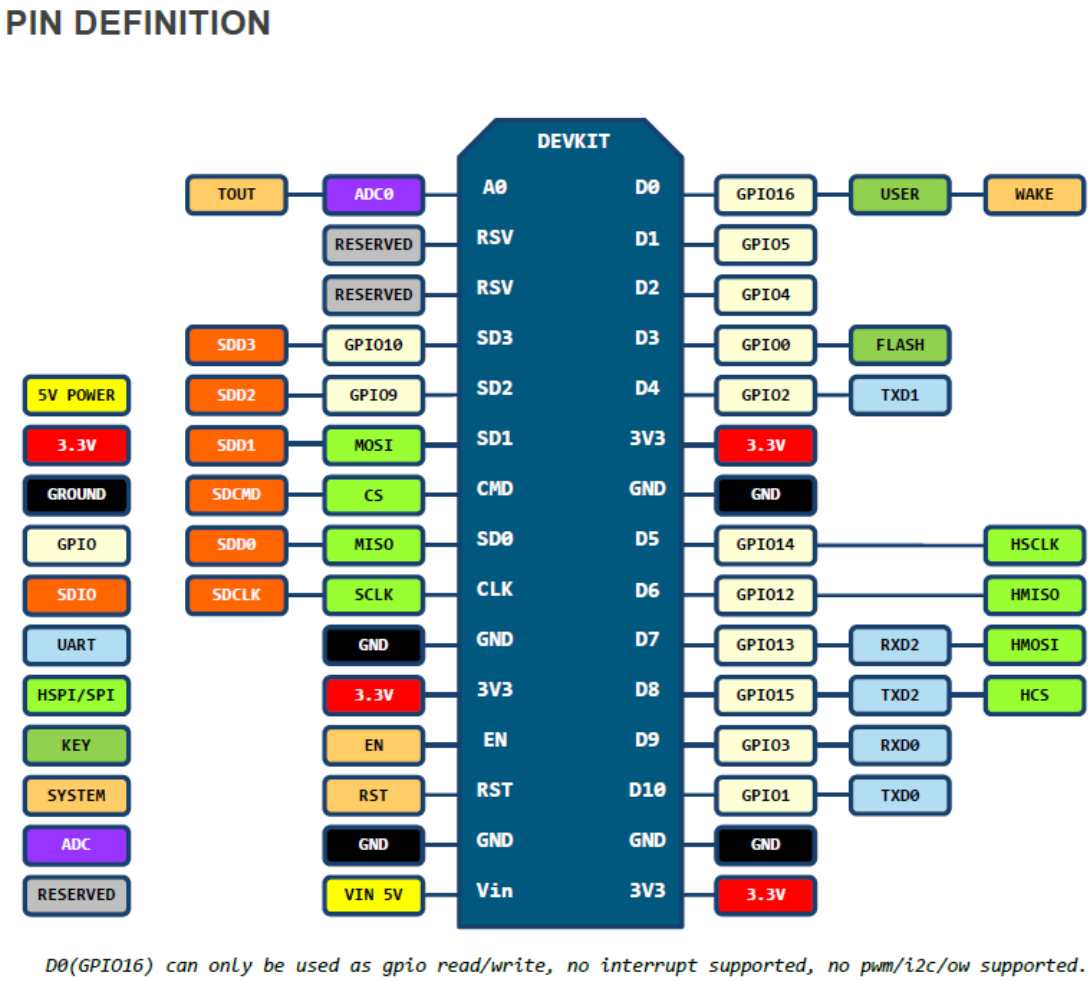
6	TOUT	I	ADC Pin (note: an internal pin of the chip) can be used to check the power voltage of VDD3P3 (Pin 3 and Pin4) or the input voltage of TOUT (Pin 6). These two functions cannot be used simultaneously.
7	CHIP_EN	I	Chip Enable. High: On, chip works properly; Low: Off, small current
8	XPD_DCDC	I/O	Deep-Sleep Wakeup ; GPIO16
9	MTMS	I/O	GPIO14; HSPI_CLK
10	MTDI	I/O	GPIO12; HSPI_MISO
11	VDDPST	P	Digital/IO Power Supply (1.8V~3.3V)
12	MTCK	I/O	GPIO13; HSPI_MOSI; UART0_CTS
13	MTDO	I/O	GPIO15; HSPI_CS; UART0_RTS
14	GPIO2	I/O	UART Tx during flash programming; GPIO2
15	GPIO0	I/O	GPIO0; SPI_CS2
16	GPIO4	I/O	GPIO4
17	VDDPST	P	Digital/IO Power Supply (1.8V~3.3V)
18	SDIO_DATA_2	I/O	Connect to SD_D2 (Series R: 200Ω); SPIHD; HSPIHD; GPIO9
19	SDIO_DATA_3	I/O	Connect to SD_D3 (Series R: 200Ω); SPIWP; HSPIWP; GPIO10
20	SDIO_CMD	I/O	Connect to SD_CMD (Series R: 200Ω); SPI_CS0; GPIO11
21	SDIO_CLK	I/O	Connect to SD_CLK (Series R: 200Ω); SPI_CLK; GPIO6
22	SDIO_DATA_0	I/O	Connect to SD_D0 (Series R: 200Ω); SPI_MSIO; GPIO7
23	SDIO_DATA_1	I/O	Connect to SD_D1 (Series R: 200Ω); SPI_MOSI; GPIO8
24	GPIO5	I/O	GPIO5
25	U0RXD	I/O	UART Rx during flash programming; GPIO3
26	U0TXD	I/O	UART Tx during flash progamming; GPIO1; SPI_CS1
27	XTAL_OUT	I/O	Connect to crystal oscillator output, can be used to provide BT clock input
28	XTAL_IN	I/O	Connect to crystal oscillator input
29	VDDD	P	Analog Power 3.0V~3.6V
30	VDDA	P	Analog Power 3.0V~3.6V
31	RES12K	I	Serial connection with a 12 kΩ resistor and connect to the ground
32	EXT_RSTB	I	External reset signal (Low voltage level: Active)

FONTE: (ESPRESSIF SYSTEMS IOT TEAM, 2015)

1.2. Pinagem do NodeMCU

O NodeMCU Devikit possui 11 portas digitais e uma porta somente analógica A0, que pode ser usada para receber os dados de algum sensor.

Figura 5 - Pinagem NodeMCU



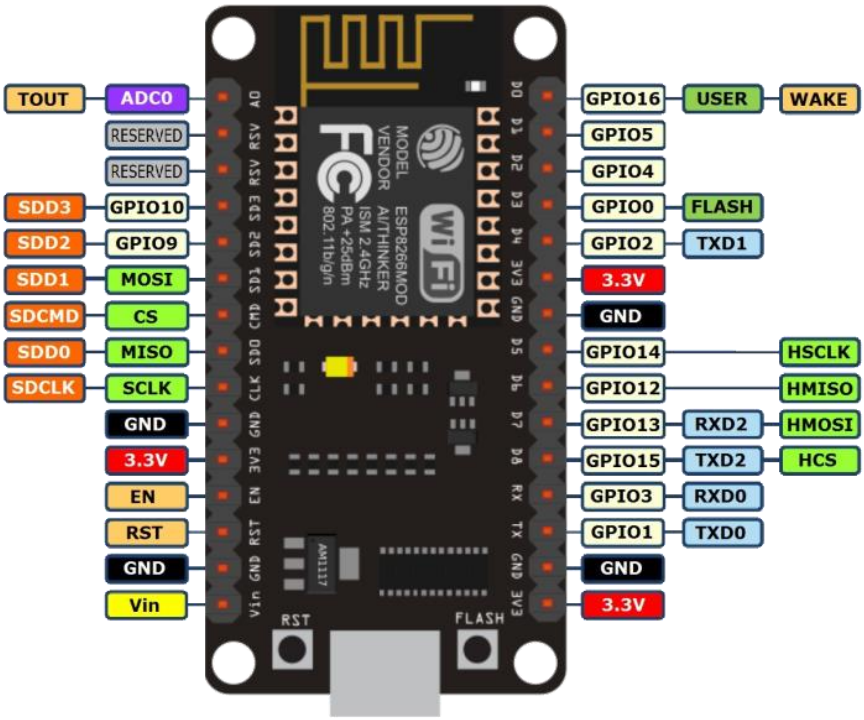
Quanto a função de cada pino, podemos dizer:

- VIN – Esse é o pino de alimentação externa. Suporta uma alimentação de até 9 V, mas é recomendável utilizar apenas 5 V.
- GND– Esse é o terra da placa.
- RST – Reset do módulo ESP-12. Nível LOW(0V) dá um reboot na placa.
- EN – (Enable) ativa o módulo ESP-12 quando o nível for HIGH(3,3V).
- 3.3V – saída do regulador interno 3,3V.
- CLK – interface SPI (clock) – pino SCLK (GPIO_6)
- SD0 – interface SPI (master in serial out) – pino MISO (GPIO_7)
- CMD – interface SPI (chip select) – pino CS (GPIO_11)

- SD1 – interface SPI (master out serial in) – pino MOSI (GPIO_8)
- SD2 – pino GPIO_9 pode ser usado também para comunicação com SD Card (SDD2)
- SD3 – pino GIPO_10 – pode ser usado também para comunicação com SD Card (SDD3)
- RSV – reservado (não usado).
- ADC0– pino de entrada do conversor analógico digital ADC de 10 bits. Tensão máxima de 1,1V (variação do valor digital – 0 a 1024).
- D0 – pino GIPO_16 pode ser usado para acordar (WAKE UP) o ESP8266 em modo sono profundo (Deep sleep mode).
- D1 – pino GPIO_5 – entrada ou saída.
- D2 – pino GIPO_4 – entrada ou saída.
- D3 – pino GPIO_0 é usado também para controlar o upload do programa na memória Flash. Está conectado no botão FLASH.
- D4 – pino GPIO_2 – UART_TXD1 quando carregando o programa na memória FLASH
- D5 – pino GPIO_14 pode ser usado em SPI de alta velocidade (HSPI-SCLK)
- D6 – pino GPIO_12 pode ser usado em SPI de alta velocidade (HSPI-MISO)
- D7 – pino GPIO_13 pode ser usado em SPI de alta velocidade (HSPI-MOSI) ou UART0_CTS.
- D8 – pino GPIO_15 pode ser usado em SPI de alta velocidade (HSPI-CS) ou UART0_RTS.
- RX – pino GPIO_3 – U0RXD quando carregando o programa na memória FLASH.
- TX – pino GIPO_1 – U0TXD quando carregando o programa na memória FLASH.

Além das portas, na placa existem 2 LEDs indicadores e dois botões (RST e FLASH):

Figura 6 - Pinagem e estrutura Física do NodeMCU



- Led indicador Azul – está conectado no pino GPIO_16. Um pulso LOW(0V) acionará o led.
- Led Indicador ESP-12 – pisca quando a memória Flash está sendo gravada
- Botão de RST – dá um pulso LOW (0V) no pino -RST (reset) reboot no módulo ESP-12.
- Botão de FLASH – dá um pulso LOW (0V) no pino GPIO_0 – permite a gravação do programa no ESP-12.

2. SENSOR ULTRASSÔNICO – HC-SR04

2.1. Princípio de funcionamento

O sensor ultrassônico faz uso de ultrassons na leitura dos dados. O princípio de operação desse sensore é exatamente o mesmo do sonar, usado pelo morcego para detectar objetos e presas em seu voo cego.

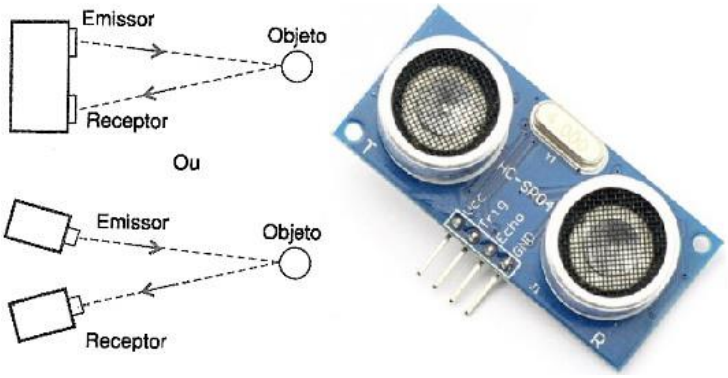
Figura 7 - As ondas ultrassônicas refletem nos objetos que estão em seu percursso

FONTE (BRAGA, 2018)

O comprimento de onda usado e portanto a frequência são muito importantes nesse tipo de sensor, pois ele determina as dimensões mínimas do objeto que pode ser detectado. Os sinais passam através de objetos cujas dimensões sejam muito menores do que o comprimento de onda. Um sinal de 1000 Hz, por exemplo, teria 34 cm de comprimento de onda, sendo teoricamente esse o tamanho do menor objeto que poderia ser detectado por essa frequência, considerando-se uma velocidade aproximada do som de 340 m/s. (BRAGA, 2018).

Na prática um sensor ultrassônico é formado por um emissor e um receptor, tanto fixados num mesmo conjunto como separados, dependendo do posicionamento relativo desejado.

Figura 8 - Esquema de um sensor ultrassônico e o HC-SR04 ao lado



2.2. Funcionamento HC-SR04

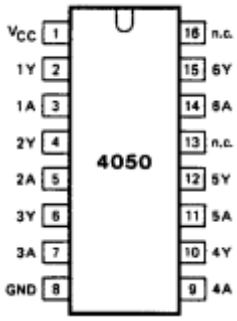
Para funcionar, o módulo requer uma alimentação de 5 V. O pino Trigger dá o disparo para o início da operação, assim, para iniciar é necessário que ele esteja em nível alto por mais de 10µs. Quando o sinal é emitido o pino ECHO fica em nível alto até receber o sinal de volta. Dessa forma a distância é calculada utilizando o tempo que o ECHO ficou em nível alto multiplicado pela velocidade do som dividido por 2 (ida e volta).

$$Distância = \frac{ECHO_{nivel\ alto} \times v_{som}}{2}$$

O pino ECHO então retornará com um nível de 5 V para o microcontrolador, mas, como dito anteriormente, a tensão de operação do ESP 8266 é de até 3,3 V. Por isso, ligaremos o pino ECHO do HC-SR04 em um driver que converterá os 5 V em 3,3 V, esse driver será o CI 4050.

O circuito integrado 4050 possui seis buffers não inversores com capacidade de saída de alta corrente, o qual será alimentado por uma tensão de 3,3 V, tensão que nós desejamos que o sinal ECHO seja convertido. Abaixo está a pinagem dele, sendo A as portas de entrada e Y as de saída.

Figura 9 - Pinagem CI 4050



3. CÓDIGOS

3.1. Primeiros passos

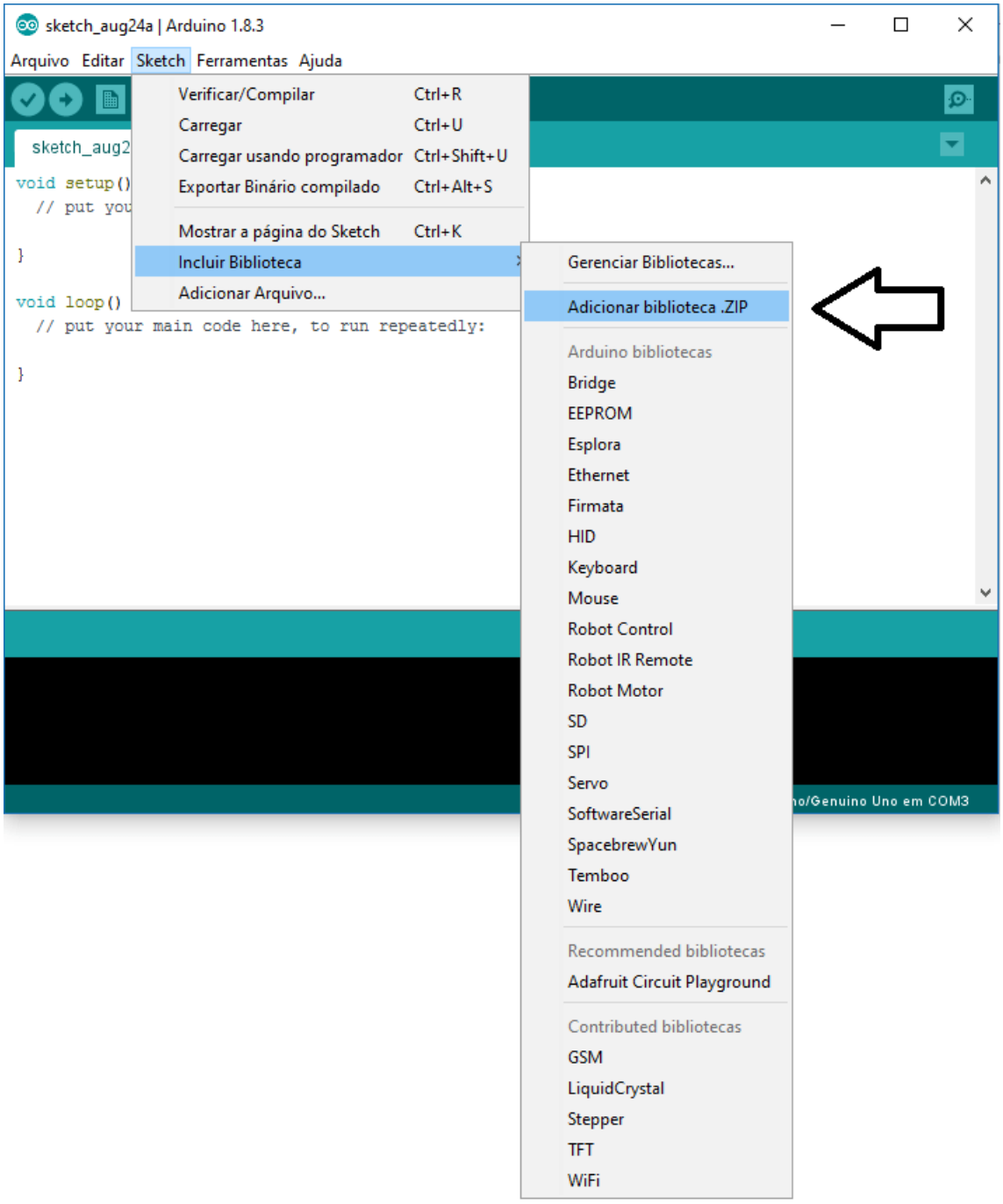
Antes de começar a programar é necessário realizar o download do driver do NodeMCU, pois sem ele o computador não reconhece o dispositivo. O driver pode ser baixado nesse link <https://s3-sa-east-1.amazonaws.com/robocore-tutoriais/163/CP210x_Windows_Drivers.zip>.

Para instalá-lo é preciso ir até o Gerenciador de dispositivos do Windows, identificar o dispositivo do nodeMCU, clicar com o botão direito e selecionar a opção atualizar driver, e selecionar a pasta do driver baixado.

Além disso é necessário fazer o download das bibliotecas utilizadas, a do sensor e do microcontrolador, os links estão abaixo.

- Sensor: <https://portal.vidadesilicio.com.br/wp-content/uploads/2017/05/Ultrasonic.zip>.
- ESP8266: <https://github.com/itead/ITEADLIB_Arduino_WeeESP8266/archive/master.zip>.

Primeiramente, faça o download dos arquivos da biblioteca compactados no formato zip. Em seguida, basta abrir o IDE e ir em “Sketch -> Incluir Biblioteca -> Adicionar biblioteca .ZIP” E selecionar o aquivo baixado:


sketch_aug24a | Arduino 1.8.3
Arquivo Editar Sketch Ferramentas Ajuda
Verificar/Compilar Ctrl+R
Carregar Ctrl+U
Carregar usando programador Ctrl+Shift+U
Exportar Binário compilado Ctrl+Alt+S
Mostrar a página do Sketch Ctrl+K
Incluir Biblioteca
Adicionar Arquivo...
Gerenciar Bibliotecas...
Adicionar biblioteca .ZIP
Arduino bibliotecas
Bridge
EEPROM
Esplora
Ethernet
Firmata
HID
Keyboard
Mouse
Robot Control
Robot IR Remote
Robot Motor
SD
SPI
Servo
SoftwareSerial
SpacebrewYun
Temboo
Wire
Recommended bibliotecas
Adafruit Circuit Playground
Contributed bibliotecas
GSM
LiquidCrystal
Stepper
TFT
WiFi
void setup()
// put you
}
void loop()
// put your main code here, to run repeatedly:
}
no/Genuino Uno em COM3

3.2. Código da IDE do arduíno

```
1. /*TECH TRASH
2. Autores: Emerson Cardoso, Felipe Boni Vilalta, João Gabriel,
   Matheus Andrade
3. Novembro 2018
4. Hardware utilizado:
5.   NodeMCU ESP 8266 Devikit
6.   Sensor ultrassonicoHC-SR04
7.   CI 4050
8. */
9. //Bibliotecas do nodemcu
10.   #include <ESP8266WiFi.h> //This library provides ESP8266
   specific Wi-Fi routines we are calling to connect to network.
11.   #include <WiFiClient.h> //Creates a client that can connect to
   to a specified internet IP address
12.   #include <ESP8266WebServer.h> //Possui as funções para hospedar
   a páginas Web
13.   //=======================
14.
15.   //Biblioteca sensor ultrassonico
16.   #include <Ultrasonic.h> //Carrega a biblioteca do sensor
   ultrassonico
17.   //=======================
18.
19.   //Definição dos pinos do sensor ultrassonico
20.   #define pino_trigger 4
21.   #define pino_echo 5 //Iremos utilizá-lo na interrupção
22.   //=======================
23.
24.
25.   const char* ssid = "TECHTRASH";
26.   const char* password = "12345678";
27.
28.   ESP8266WebServer server(80); // Declaração da página Web Server
   na porta 80, porta padrão da Web
29.
30.   Ultrasonic ultrasonic(pino_trigger, pino_echo); //Inicializa o
   sensor nos pinos definidos acima
31.
32.   const int led = 2;
```

```
33.   float cmMsec, lixo; //Variaveis de medicao
34.
35.   //====================Endereço Web server sendo acessado
36.   void handleRoot() {
37.     String textoHTML;
38.     if(cmMsec>=0 && cmMsec<=100)
39.     {
40.       textoHTML += cmMsec;
41.     }else
42.     {
43.       textoHTML += "Erro";
44.     }
45.     server.send(200, "text/html", textoHTML);
46.   }
47.   //=======================
48.
49.
50.   //====================Caso o acesso não tenha ocorrido com
   sucesso
51.   void handleNotFound(){
52.     String message = "File Not Found\n\n";
53.     message += "URI: ";
54.     message += server.uri(); //URI da Web server
55.     message += "\nMethod: ";
56.     message += (server.method() == HTTP_GET)?"GET":"POST";
57.     message += "\nArguments: ";
58.     message += server.args();
59.     message += "\n";
60.     for (uint8_t i=0; i<server.args(); i++){
61.       message += " " + server.argName(i) + ": " + server.arg(i) +
   "\n";
62.     }
63.     server.send(404, "text/plain", message);
64.   }
65.   //=======================
66.
67.
68.   void setup(void){
69.     Serial.begin(115200); //Iniciando o monitor serial na mesma
   velocidade de comunicação do ESP
70.     Serial.flush();
```

```
71.     WiFi.mode(WIFI_STA); //Station (STA) mode is used to get ESP
   module connected to a Wi-Fi network established by an access point.
72.     WiFi.begin(ssid, password); //Actual connection to Wi-Fi is
   initialized
73.     Serial.println("");
74.
75.     // Wait for connection
76.     while (WiFi.status() != WL_CONNECTED) {
77.       delay(500);
78.       Serial.print(".");
79.     }
80.
81.     Serial.println("");
82.     Serial.println("TECH TRASH");
83.     Serial.println("Emerson Cardoso, Felipe Boni Vilalta, João
   Gabriel, Matheus Andrade");
84.     Serial.println("");
85.     Serial.print("Conectado na rede: ");
86.     Serial.println(ssid);
87.     Serial.print("Endereco IP: ");
88.     Serial.println(WiFi.localIP());  //Imprimi  o  endereço  IP
   definido para o ESP
89.
90.     server.on("/", handleRoot); //Tell the server what URIs it
   needs to respond to:
91.
92.     server.on("/inline", [](
93.     {
94.       server.send(200, "text/plain", "this works as well");
95.     });
96.
97.     server.onNotFound(handleNotFound); //Caso o servidor não seja
   localizado chama a função handleNotFound
98.
99.     server.begin(); //Inicia o Web server
100.    Serial.println("HTTP server started");
101.    wdt_enable(WDTO_2S); //Definindo a rotina watchdogs para reset
   em 1 segundos
102.  }
103.
104.  void medida()
```

```
105.  {
106.    long microsec = ultrasonic.timing();  //Le os dados do sensor
   com o tempo de retorno do sinal
107.    cmMsec    =    ultrasonic.convert(microsec,    Ultrasonic::CM);
   //Calcula a distancia em cm a partir do tempo de retorno em
   microsegundos
108.    cmMsec=map(26-cmMsec,0,26,0,100);   //Regra   de   tres   para
   calcular a porcentagem de carregamento do lixo
109.    server.handleClient();
110.    delay(1000);
111.  }
112.
113.  void loop(void){
114.    medida();
115.    wdt_reset ( );
116.  }
```

No comando `WiFi.mode(WIFI_STA);`, o modo é definido com STA porque ele facilita o controle da conexão wi-fi, pois caso a conexão seja perdida, o ESP automaticamente vai tentar conectar no último ponto de acesso usado.

O loop `while (WiFi.status() != WL_CONNECTED)` continuará até que o ESP esteja conectado na rede, até isso ocorrer serão imprimidos “.” no monitor serial. Isso é necessário porque a conexão pode demorar alguns segundos até que seja completada.

A função `void handleRoot()` será executada toda vez que o endereço raiz do ESP8266 for acessado pelo navegador. Dentro dessa função temos a declaração de uma variável do tipo "String" chamada "html". Nessa string colocamos todo nosso código HTML que será exibido ao carregarmos a página inicial do ESP8266. Depois de preencher todo valor do HTML na variável, chamamos a função server.send(), passando os parâmetros de código de retorno (200 significa retorno OK), o tipo do retorno (text/html significa que retornaremos dados do tipo texto na formatação HTML), e a variável "html" que contém o conteúdo da página.

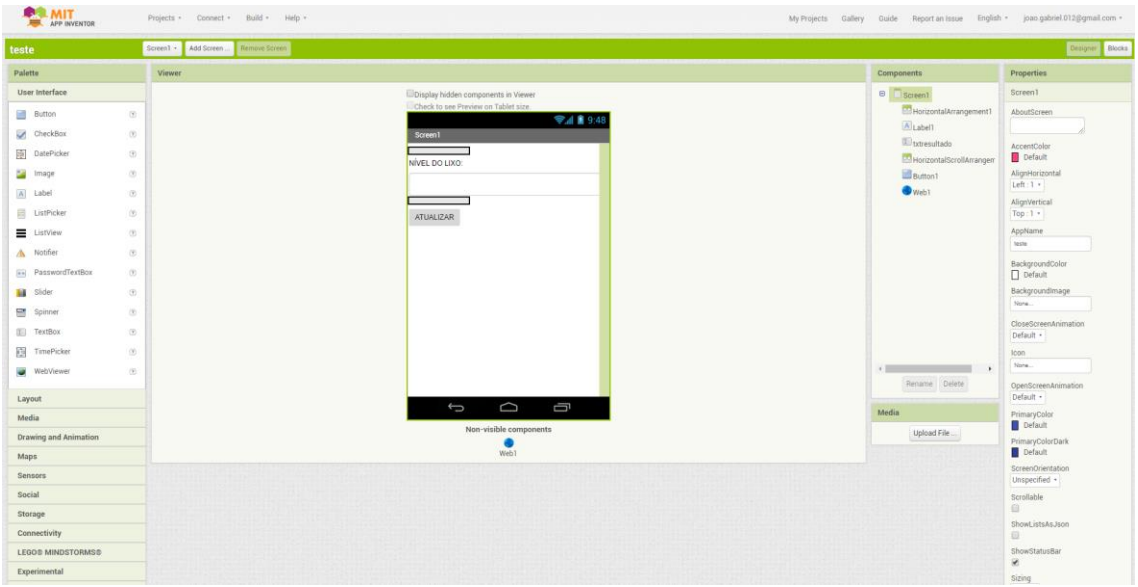
A subrotina `server.handleClient();` dentro da função `void loop(void)` fica eternamente lidando com a conexão do cliente. Nesse cenário, o servidor é o ESP8266 e o cliente é o Navegador de Internet que utilizamos para acessar as páginas HTML do ESP8266.

3.3. Aplicativo

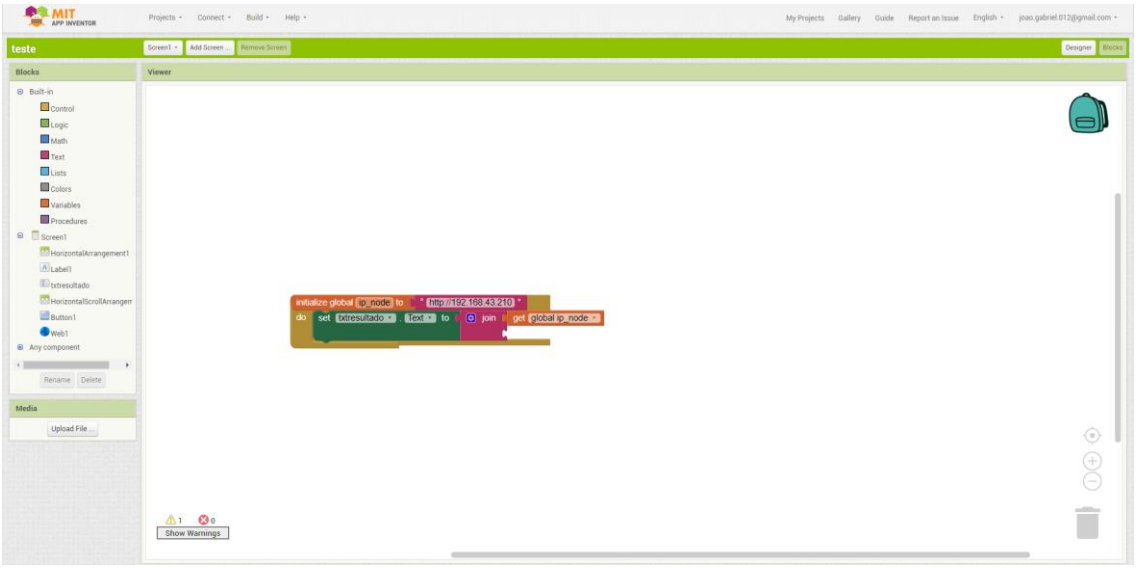
O aplicativo para Android foi feito na plataforma do AppInventor, onde nós construímos o design do aplicativo e depois o programamos através de blocos, como pode ser visto a seguir.

O AppInventor é uma ferramenta desenvolvida pelo Google e, atualmente, mantida pelo Instituto de Tecnologia de Massachusetts (MIT); que permite a criação de aplicativos para smartphones que rodam o sistema operacional Android, sem que seja necessário conhecimento em programação.

Para a construção do design, é necessário selecionar a opção Designer (lado superior direito da imagem) e arrastar os blocos que estão na imagem abaixo para o quadro central.

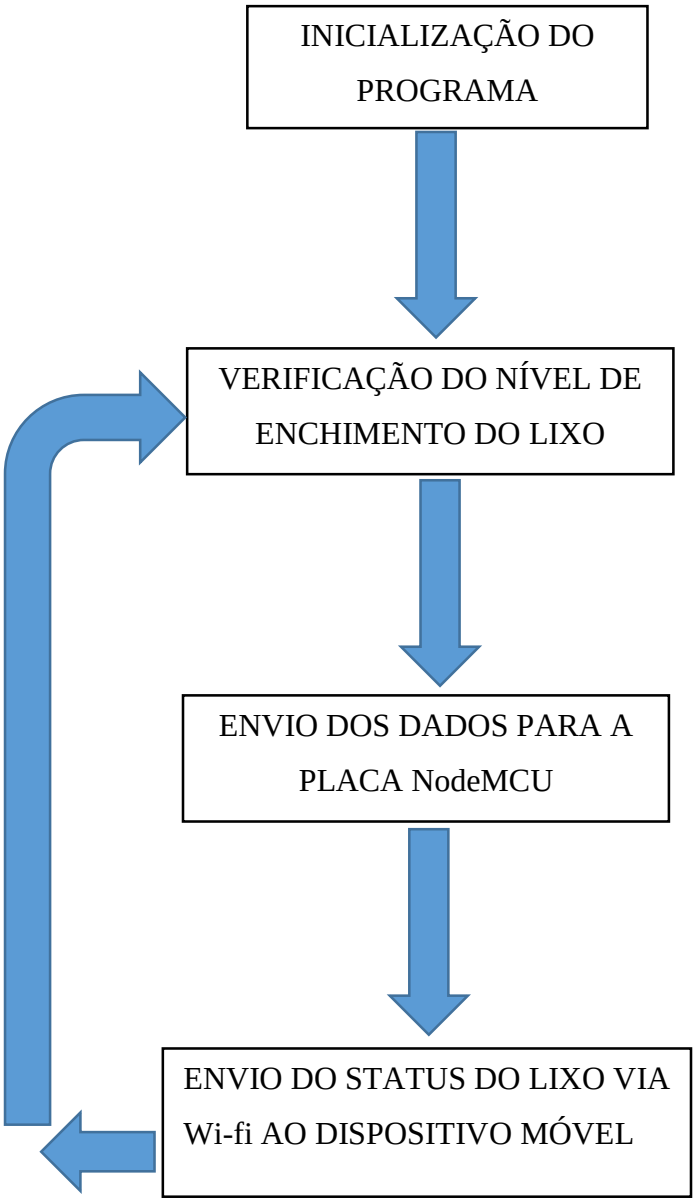


Para programar temos que mudar para a opção Blocks (lado superior direito da tela), e arrastar os blocos como na imagem abaixo. No lugar do IP é necessário mudar para o IP que aparecerá no monitor serial do nodeMCU.

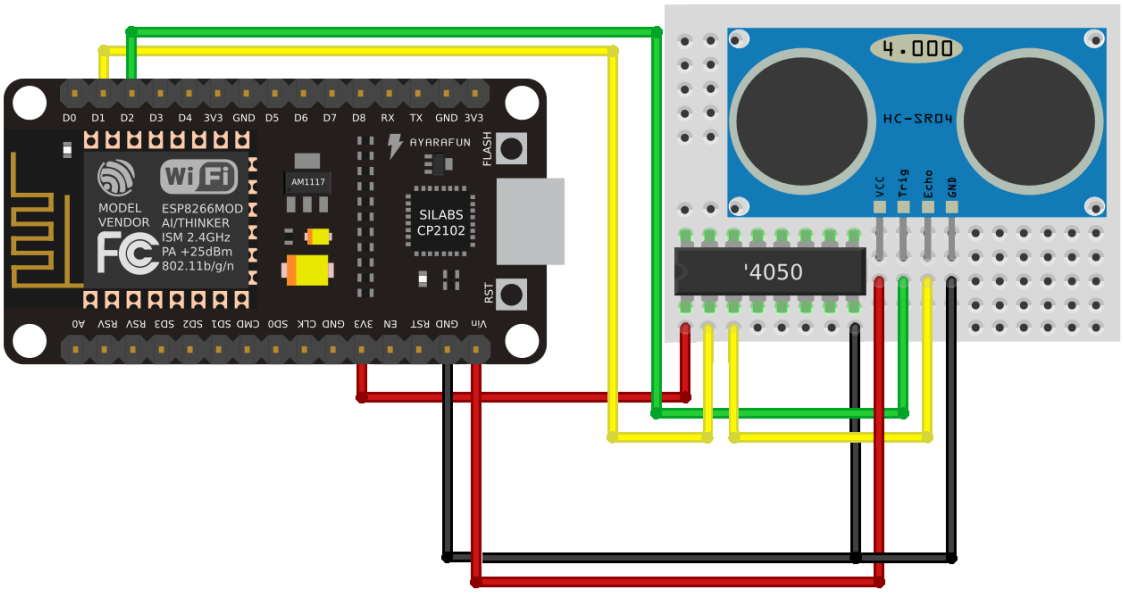

MIT APP INVENTOR
Projects
Connect
Build
Help
My Projects
Gallery
Guide
Report an Issue
English
joao.gabriel.012@gmail.com
teste
Screen1
Add Screen ...
Remove Screen
Designer
Blocks
Blocks
Built-in
Control
Logic
Math
Text
Lists
Colors
Variables
Procedures
Screen1
HorizontalArrangement1
Label1
txtresultado
HorizontalScrollArrangem
Button1
Web1
Any component
Rename
Delete
Media
Upload File ...
Viewer
initialize global ip_node to
http://192.168.43.210
do
set txtresultado . Text to
join
get global ip_node
1
0
Show Warnings

4. FLUXOGRAMA E ESQUEMA DO PROJETO

4.1. Fluxograma



4.2. Esquema do projeto



O pino Vcc do sensor HC-SR04 é ligado no pino Vin do NodeMCU, que vai fornecer a tensão de alimentação da placa (5 V) para o sensor. O pino Trigger é conectado na porta 3 do CI, e a porta do do CI é ligada na port D1 do NodeMCU.

As portas 1 e 8 do CI são ligados no 3,3 V e no GND, respectivamente.

Para alimentar o NodeMCU basta utilizar o cabo USB ligado em um transformador de 5 V e liga-lo na tomada.

5. RESULTADOS OBTIDOS

Foram realizados alguns testes de precisão com o sensor e foi constatado que ele possui uma boa precisão, mas pode apresentar alguns resultados inconsistentes. Uma solução para isso foi utilizar o lixo com tampa, pois ele diminui a dispersão do sinal emitido pelo sensor.

Outro resultado constatado foi que após conectado em uma rede, mesmo depois de desligar e ligar o nodeMCU, ele mantém o mesmo IP.

REFERENCIAS

BRAGA, Newton C.. **Como funcionam os sensores ultrassônicos (ART691)**. Disponível em: <http://www.newtoncbraga.com.br/index.php/como-funciona/5273-art691>. Acesso em: 11 nov. 2018.

ESPRESSIF SYSTEMS IOT TEAM. **ESP8266EX Datasheet.** 4.3 : Copyright © 2015, 2015. 31 p. Disponível em: <https://cdn-shop.adafruit.com/product-files/2471/0A-ESP8266__Datasheet__EN_v4.3.pdf>. Acesso em: 02 nov. 2018.

HANDSON TECHNOLOGY. ESP8266 NodeMCU WiFi Devkit: **User Manual**. 1.2 , 2015. Disponível em: <http://www.handsontec.com/pdf_learn/esp8266-V10.pdf>. Acesso em: 02 nov. 2018.

KOYANAGI, Fernando. NODEMCU ESP8266: **DETAILS AND PINOUT**. Disponível em: <https://www.instructables.com/id/NodeMCU-ESP8266-Details-and-Pinout/>. Acesso em: 02 nov. 2018.

MURTA, Gustavo. **Guia completo do NodeMCU – ESP12**: Introdução (1). Disponível em: <http://blog.eletrogate.com/nodemcu-esp12-introducao-1/>. Acesso em: 11 nov. 2018.