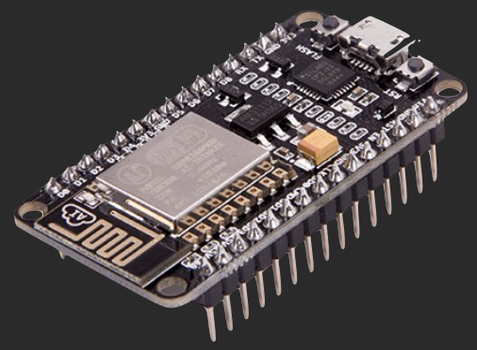
Projeto Final - Home Sensor

INF1805 - Sistemas Reativos

Matheus Cunha
Victor Meira

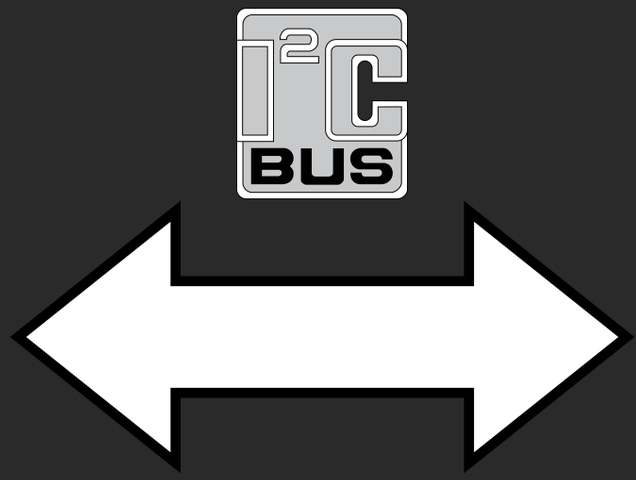
Funcionamento

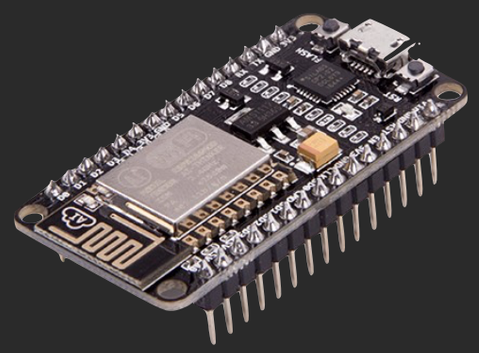
Arduino



NodeMCU

Funcionamento

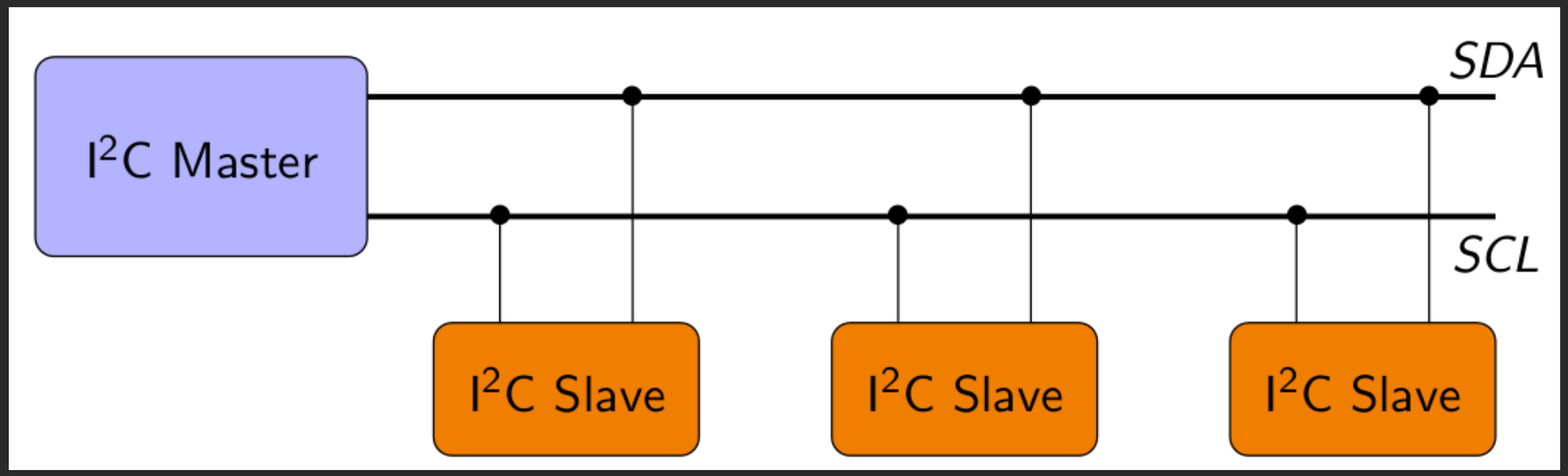




Arduino

NodeMCU

Protocolo I²C BUS



Master: NodeMCU
Slave: Arduino

Funcionamento

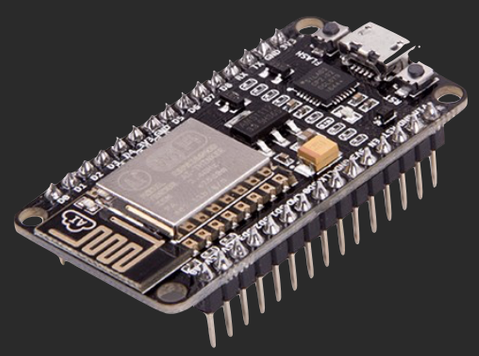
Sensor de efeito hall

LEDs

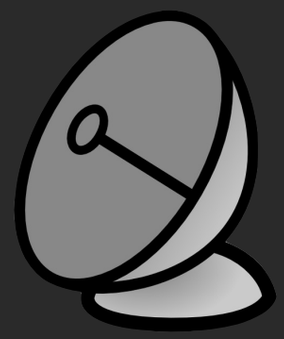
Sensor de detecção de toque de metal

Botão

I²C BUS



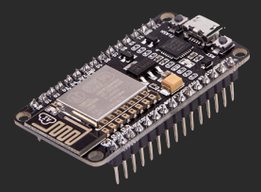
Sensor ótico reflexivo

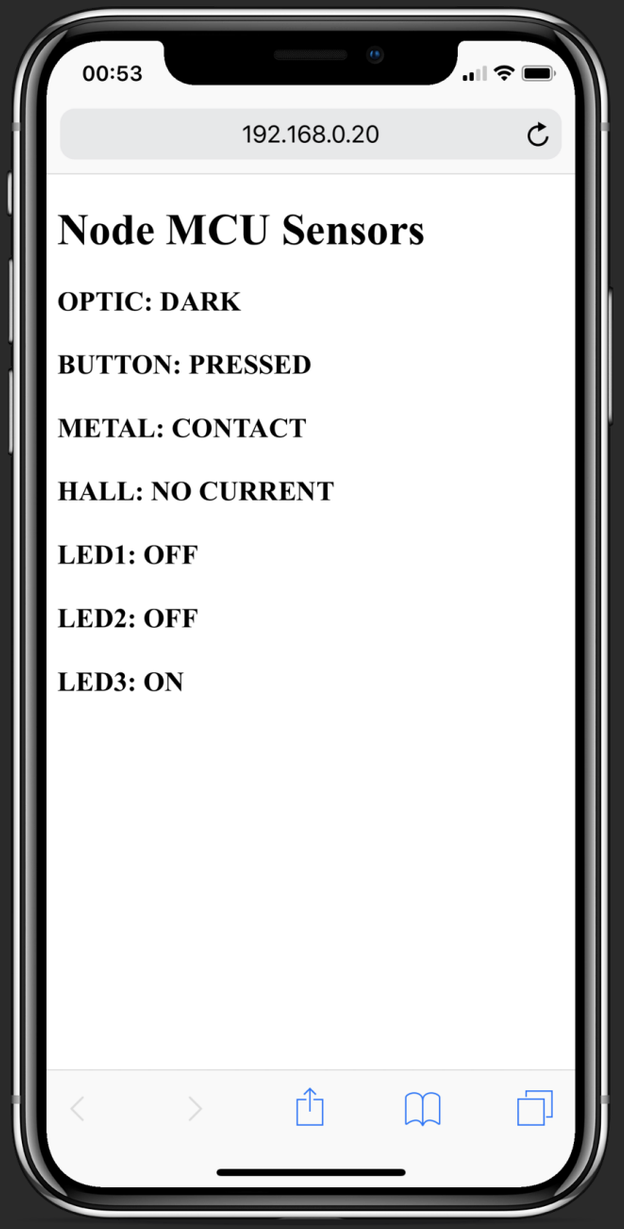


Arduino

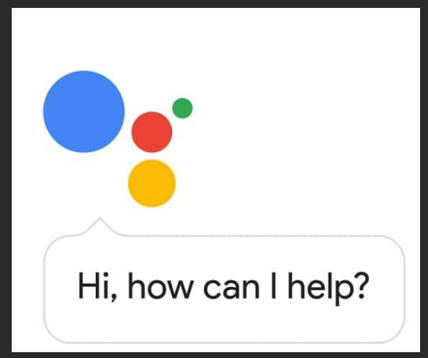
NodeMCU

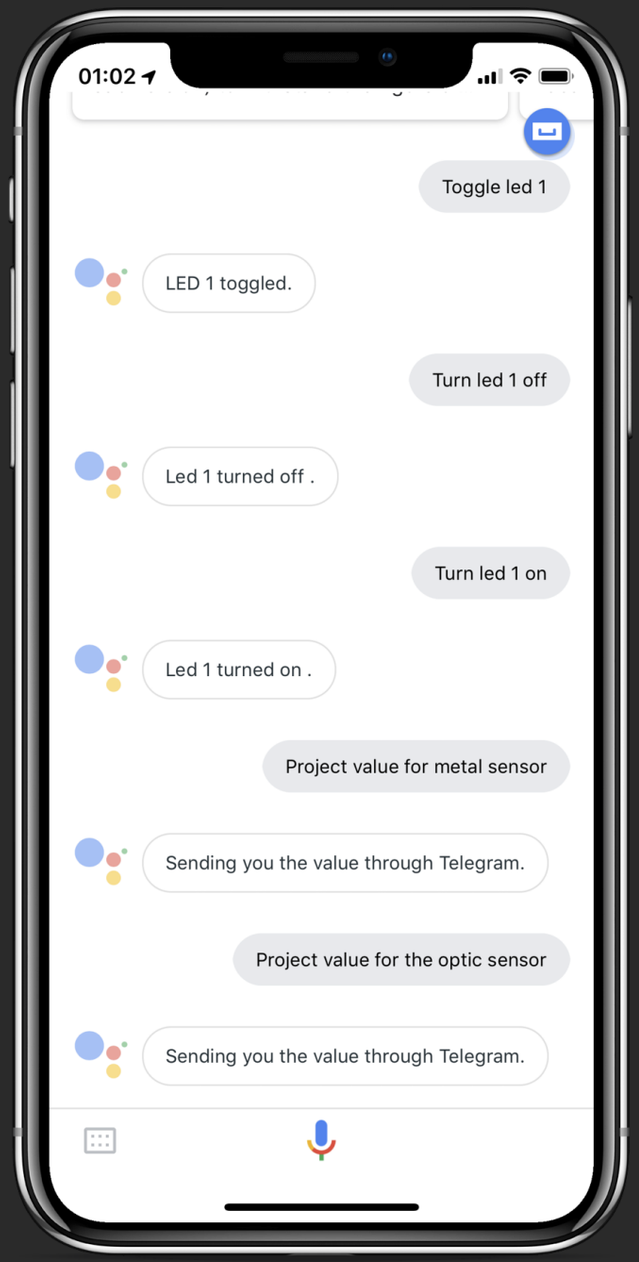
Funcionamento



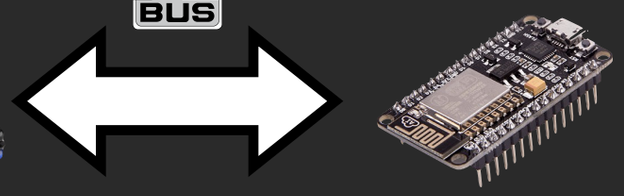


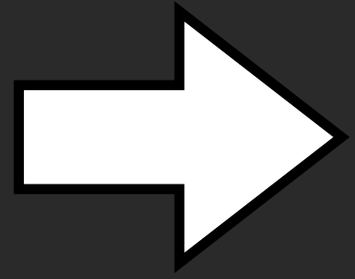
Funcionamento

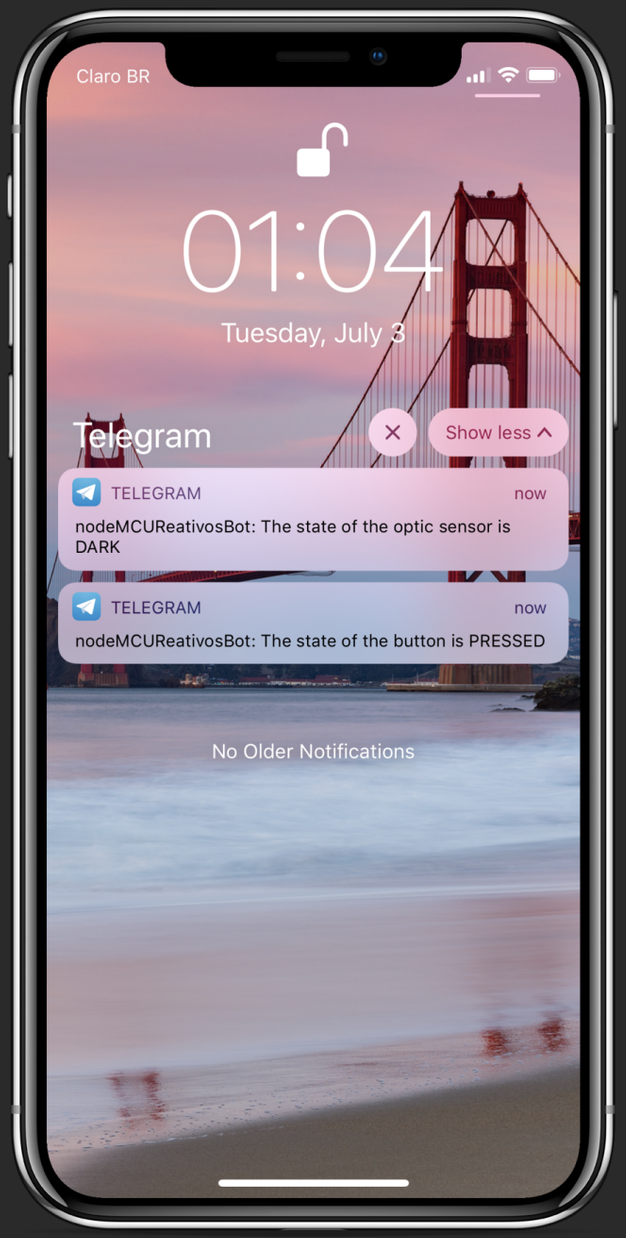




Funcionamento







Implementação - Arduino

Setup

```
void setup() {
  Serial.begin(9600);

  Wire.begin(NODE_ADDR);
  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestEvent);

  pinMode(led_control1, OUTPUT);
  pinMode(led_control2, OUTPUT);
  pinMode(led_control3, OUTPUT);

  digitalWrite(led_control1, LOW);
  digitalWrite(led_control2, LOW);
  digitalWrite(led_control3, LOW);

  optic = initialize_sensor(0, 8, 12);
  hall = initialize_sensor(1, 9, 13);
  button = initialize_sensor(2, 7, 11);
  metal = initialize_sensor(3, 6, 10);

}
```

Implementação - Arduino

Evento Recebido

```
void receiveEvent(int howMany)
{
  char j;
  char str_received[4] = "";
  int i = 0;

  while(Wire.available())
  {
    j = Wire.read();

    str_received[i] = str_received[i] + j;
    i++;
  }
  str_received[i] = '\0';

  read_string(str_received);

  Serial.println(str_received);
}
```

Implementação - Arduino

Evento requisitado

```
void requestEvent()
{
  String send_String;
  int led1_state = digitalRead(led_control1);
  int led2_state = digitalRead(led_control2);
  int led3_state = digitalRead(led_control3);

  hall.sensor_state = check_sensor_state(hall);
  optic.sensor_state = check_sensor_state(optic);
  button.sensor_state = check_sensor_state(button);
  metal.sensor_state = check_sensor_state(metal);

  send_String = "h " + String(hall.sensor_state) + ";o " + String(optic.sensor_state) + ";b " +
     String(button.sensor_state)
  + ";m " + String(metal.sensor_state) + ";" + String(led1_state) + String(led2_state) + String
    (led3_state);

  send_String.toCharArray(send_str, 61);

  Wire.write(send_str);
}
```

Implementação - Arduino

Evento requisitado

```
void requestEvent()
{
  String send_String;
  int led1_state = digitalRead(led_control1);
  int led2_state = digitalRead(led_control2);
  int led3_state = digitalRead(led_control3);

  hall.sensor_state = check_sensor_state(hall);
  optic.sensor_state = check_sensor_state(optic);
  button.sensor_state = check_sensor_state(button);
  metal.sensor_state = check_sensor_state(metal);

  send_String = "h " + String(hall.sensor_state) + ";o " + String(optic.sensor_state) + ";b " +
     String(button.sensor_state)
  + ";m " + String(metal.sensor_state) + ";" + String(led1_state) + String(led2_state) + String
    (led3_state);

  send_String.toCharArray(send_str, 61);

  Wire.write(send_str);
}
```

Implementação - NodeMCU

NodeMCU Client

```
require("TelegramBot")
require("googleAssist")
require("arduinoComm")
require("webServer")

--sendToTelegram("HELLO TEST")
makeConn()
upServer()

local myTimer = tmr.create()
myTimer:register(1000, tmr.ALARM_AUTO, reloadSensorData)
myTimer:start()
```

Implementação - NodeMCU

Comunicação com Arduino

```
-- initialize i2c, set pin1 as sda, set pin2 as scl
i2c.setup(id, sda, scl, i2c.SLOW)

function writeToArduino(data)
    print(data)
    i2c.start(id)
    i2c.address(id, dev_addr, i2c.TRANSMITTER)
    i2c.write(0, data)
    i2c.stop(id)
end

function readFromArduino(bytes)
    i2c.start(id)
    i2c.address(id, dev_addr, i2c.RECEIVER)
    data = i2c.read(id, bytes)
    i2c.stop(id)
    return data
end

function reloadSensorData()
    writeToArduino("req")
    sensorString = readFromArduino(20)
    print(sensorString)
    parseSensorString(sensorString)
end
```

Implementação - NodeMCU

Google Assistant

```
function messageReceivedCallback(client)
    local function messageTreatment(userdata, topic, message)
        print("Received message:", message)
        if message == "metal" then
            sendToTelegram("The state of the metal sensor is ".. _G.metal)
        elseif message == "button" then
            sendToTelegram("The state of the button is ".. _G.button)
        elseif message == "optic" then
            sendToTelegram("The state of the optic sensor is ".. _G.optic)
        elseif message == "hall" then
            sendToTelegram("The state of the hall sensor is ".. _G.hall)
        elseif string.find(message,"led") then
            --led<num><act> from MQTT
            lednum = string.sub(message,4,4)
            ledact = string.sub(message,6,6)
            writeToArduino("l"..lednum..ledact)
            print("l"..lednum..ledact)
        end
    end

    client:on("message",messageTreatment)
end
```

Implementação - NodeMCU

Telegram

```
function sendToTelegram(messageText)

    webHooksKey = "cB2Z3ma0b6FHUsDl0AVF_vm7mCMmg0
    event = "nodeMCURequest"
    address = "http://maker.ifttt.com/trigger/" .. event .. "/with/key/" .. webHooksKey

    postData = [[ {"value1": "]] .. messageText .. [["  }  ]]

    local numberOfTries = 5
        local function callbackPost(code,data)
            if (code < 0) then
                print("HTTP request failed :", code)
                numberOfTries = numberOfTries - 1
                print("Number of remaining tries: " .. numberOfTries)
                tmr.delay(1000000)
                if(numberOfTries > 0) then
                        http.post(address, 'Content-Type: application/json\r\n',postData, callbackPost)
                end
            else
                print(code, data)
            end
        end
   print(address, postData)
   http.post(address, 'Content-Type: application/json\r\n',postData, callbackPost)
end
```

Implementação - NodeMCU

Web Server

```
function upServer()
  sv = net.createServer(net.TCP, 30)

  if sv then
    sv:listen(80, function(conn)
      conn:on("receive", receiver)
       -- HTML Header Stuff
          conn:send('HTTP/1.1 200 OK\n\n')
          conn:send('<!DOCTYPE HTML>\n')
          conn:send('<html>\n')
          conn:send('<head><meta  content="text/html; charset=utf-8">\n')
          conn:send('<title>Servidor NodeMCU</title></head>\n')
          conn:send('<body><h1>Node MCU Sensors</h1>\n')
          conn:send('<body><h5>OPTIC: ' .. _G.optic .. '</h5>\n')
          conn:send('<body><h5>BUTTON: ' .. _G.button .. '</h5>\n')
          conn:send('<body><h5>METAL: ' .. _G.metal .. '</h5>\n')
          conn:send('<body><h5>HALL: ' .. _G.hall .. '</h5>\n')
          conn:send('<body><h5>LED1: ' .. _G.led1 .. '</h5>\n')
          conn:send('<body><h5>LED2: ' .. _G.led2 .. '</h5>\n')
          conn:send('<body><h5>LED3: ' .. _G.led3 .. '</h5>\n')
          conn:send('</body></html>\n')
          conn:on("sent", function(conn) conn:close() end)
      end)
    end
  end
```

Principais Dificuldades

- Conexão entre NodeMCU e Arduino
    - Serial não funcionou e seria difícil de implementar (problemas com compilação e debug)
    - i2c foi a forma utilizada, porém vários obstáculos encontrados (diferença de nível lógico entre os dois)
        - i2c aceitava apenas poucos bytes
- Google Assistant não podia responder com variáveis
    - Não aceitava “perguntas normais”
- Serviço do telegram e NodeMCU
    - IFTTT para mandar as mensagens