ESCOLA DE NOVAS TECNOLOGIAS DOS AÇORES

Curso - Técnico de Eletrónica e Telecomunicações

Anos letivos 2013-2016

Relatório da Prova de Aptidão Profissional - P.A.P

Eficiência prática – Alternativa económica

Arduino Placa Input

Formando:Pedro Pacheco de Sousa

Formadores e orientadores: Daniel Oliveira, Fábio Amaral e Paulo Martinho

19/2/2016

Agradecimentos

Em primeiro lugar, quero expressar aqui o meu profundo agradecimento à Escola de Novas Tecnologias - ENTA- que me acolheu, e ao Dr. João Lima pela oportunidade que me deu ao ingressar nos estudos, através do curso - Técnico de Eletrónica e Telecomunicações.

Durante a realização desta Prova de Aptidão Profissional pude contar com o apoio de diversas pessoas e com o recurso a várias fontes de informação, que contribuíram de diversas formas para o resultado obtido.

Agradeço aos meus Formadores, pelo tempo disponibilizado e pelo empenho com que me auxiliaram.

Ao professor Daniel Oliveira, por ter partilhado comigo a sua sabedoria, o que me permitiu limar determinadas arestas do meu trabalho;

Ao professor Fábio Amaral, por me ter orientado e por me ter facultado os materiais necessários para que nada faltasse;

Ao professor Paulo Martinho, por me orientar para a área da minha vocação e na qual pretendo focar toda a minha aprendizagem. Teria sido muito diferente sem a sua ajuda.

De igual modo, agradeço aos restantes formadores, Sílvia Gouveia, Anabela Ferreira, Cecília Dowling, Pedro Fonseca, Duarte Cota, Pedro Rosa, André Rodrigues.

Agradeço em especial aos meus pais, António e Délia Sousa, por todo o apoio recebido e por me proporcionarem as melhores condições de trabalho.

“If you really want to do something, you'll find a way. If you don't, you'll find an excuse.”

- Jim Rohn

Resumo:

O projeto do meu trabalho final, na ENTA, despertou em mim um interesse adormecido pela área da programação de componentes eletrónicos com microcontroladores. Iniciei este projeto com o objetivo de melhorar, em termos de transmissão de dados, uma das placas de INPUT da GlobeStar Systems, que demonstrou ser também uma alternativa open source económica, sendo bem-vinda face às situações de crise dos dias de hoje. Enquanto desenvolvi uma placa que substituísse a placa Input da GlobeStar Systems que utiliza um PIC, este projeto proporcionou-me um ponto de vista mais amplo sobre a programação em C++ com microcontroladores Atmega.

Desde princípios de Agosto, dei início ao meu trabalho, tendo como primeira etapa sentir o ambiente da GlobeStar Systems, frequentando o espaço, de modo a tirar quaisquer dúvidas sobre ligação entre servidor cliente, tanto com o pessoal da empresa, bem como com o meu orientador. Utilizei a placa Input, o meu arduino mega+ Ethernet shield, cabo RJ45 como materiais e dei início às pesquisas sobre servidores e clientes em geral, salvando os resultados todos em blocos de notas, documentados após testes. Pesquisei sobre o mesmo tema com arduinos e comparei os princípios encontrados, focando-me mais na base TELNET CHAT SERVER do arduino pela curiosidade que me despertou.

Concluída esta parte, continuei a pesquisar os componentes na placa INPUT, com os dados fornecidos pela GlobeStar Systems e também pesquisei os sensores que ia usar mais futuramente para efeitos demonstrativos. De acordo com a minha pesquisa, a ligação ao servidor com a placa arduino foi um sucesso. Em paralelo, os testes individuais dos sensores-botões correram como esperado e foram documentados para futura integração. Após comparar as diferenças entre as minhas tentativas com os trabalhos base, pude também observar vários métodos de tornar a minha placa mais autónoma (e prática), de forma a que ao ser conetada à rede ficasse constantemente a fazer “reset” a si mesma até se ligar ao servidor.

Concluindo, ficou verificado que é possível estabelecer uma ligação ao programa Connexall da GlobeStar, atravez do Standard Input Clien,t usando a interface TCP Client, utilizando um cabo RJ45 conectado do Arduino Ethernet Shield ao modem.

Em simultâneo, descobri durante as pesquisas outras maneiras de simplificar o meu projeto, de forma a tornar a placa mais “amigável” para o utilizador ao implementar a função “auto ip”, que atribui um IP automaticamente à placa e ficando o mesmo visível para o utilizador, bem como a função “Auto reset Ethernet” em que basta ligar a placa e colocar o cabo de seguida. Aprendi imenso com este projeto, que servirá de base para outros projetos, na medida em que posso fazer ligações entre servidores e enviar os dados que mais forem convenientes.

Palavras-Chave

Económico; RJ45; Open Source; Microcontroladores; Placa Input; Placa Arduino; Arduino Ethernet Shield; “Auto IP; ”Auto reset Ethernet” “Amigável ao utilizador”;

ÍNDICE Pág.

INTRODUÇÃO

A presente Prova de Aptidão Profissional - PAP - integrada no Curso de Técnico de Eletrónica e Telecomunicações na Escola de Novas Tecnologias dos Açores foi me sugerida para demonstrar as minhas capacidades a partir dos conhecimentos adquiridos ao longo dos 3 anos do curso, permitindo-me concluir com aproveitamento a formação profissional. Dediquei-me à projeção e utilização de microcontroladores para as ligações cliente-servidor, servidor-cliente, melhorando as capacidades atuais de processamento, transmissão e custo monetário.

Este projeto foi sugerido pelo formador Fábio Amaral, no sentido de me aproximar mais da programação devido ao meu interesse nesta área, o que me beneficiou como pessoa e como estudante.

Observei que a placa Input da Globestar Systems utiliza basicamente um microcontrolador das séries PIC, imensos optocopuladores para a receção de dados e por fim a parte mais interessante, a velha entrada I/O RS232. Constatei que estava ao meu alcance, como formando, o desafio de atualizar esta placa com um Arduino Mega 2560 e um adaptador Ethernet SHIELD, de modo a que as comunicações usufruissem dos benefícios da rede local via RJ45, que atualmente se encontra na maioria dos locais de trabalho.

FASE 1

1. DECRIÇÃO DO PROJETO / Funcionamento geral

Decidi dedicar-me à projeção e programação de uma placa Cliente que estabelecesse ligação a servidores, neste caso a um servidor da GlobeStar Systems, que já tem a sua placa atual de input, baseada num PIC 17C752, com uma série de Optocopuladores em circuito com a função de detectar sinais emitidos pelos aparelhos.

Apliquei os meus métodos, utilizando por base a placa Arduino mega 2560, de maneira que este funcionasse de igual modo com a receção de dados em relação à placa baseada no PIC 17C752, com o extra de os dados serem transmitidos via internet.

Este projeto foi motivado por uma conversa com o meu diretor de turma, professor Paulo Martinho, que ao notar o meu interesse pela programação não hesitou em direcionar-me para o Professor Fábio Amaral, que por sua vez orientou-me no esqueleto da ideia. A partir de então comecei a ganhar mais interesse pela minha área de estudo.

1. Programação – Ligação do arduíno ao servidor

Comecei por me organizar com rascunhos do compilador arduino em cada pasta, com um bloco de notas a acompanhar. Efetuei as minhas pesquisas sobre cada sensor adquirido, e correspondendo a cada sensor, guardei um programa que me servisse de exemplo para ver o sensor a trabalhar individualmente, permitindo-me testar o funcionamento ao alterar o código e introduzir as minhas variáveis ajustadas.

Primeiramente, concentrei-me somente em estabelecer uma ligação ao servidor, tomei por base o servidor TELNET nos exemplos do arduino. Uma vez funcionando, organizei o código, ficando cada sensor individualmente na sua função, para ser mais fácil detetar erros e facilitar o acesso e manutenção de variáveis.

Tendo por base somente o arduino mega e o Ethernet Shield, defeni as bibliotecas que ia usar, seguido do respetivo mac address, o IP do servidor a ligar e por fim para evitar introduzir o IP manualmente(pelo menos do SHIELD). Antes do setup, loop, chamei as livrarias necessárias “#include <Ethernet.h>, #include <SPI.h>” eAtribuí um mac address e também introduzi manualmente o IP do servidor de Input.

“Byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };”

```
//  IP do servidor SIC
```

char server[] = "192.168.1.69"; //Este tem tendência a mudar de rede para rede;

Criei no “void setup(){}” um conjunto de funções, entre elas o conjunto de instruções, condições e variáveis que denominei em comentário“Auto Ethernet”, que visa atribuir um IP à placa de rede, fazer a ligação automaticamente à rede, iniciando a ligação à internet por “Ethernet.begin(mac,Ethernet.localIP());”tornando a obtenção do IP mais acessível. “postData();”é chamada e indica o estado da ligação do arduino. Caso falhe a ligação em vez de “("Ligado")” obtemos “("Ligação falhou")” e automaticamente chama a função “resetEthernetShield();” que faz com que o programa fique repetidamente a tentar conectar-se ao servidor com os dados previamente introduzidos.

A partir daqui comentei o código a cores por ser conveniente destacar:

“#include <Ethernet.h>
#include <SPI.h>”

“Byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };”

char server[] = "192.168.1.69"; //IP do SIC

```
void setup()
{
  Serial.begin(115200); //Monitor de série é iniciado

//Auto Ethernet - iniciação da ligação à internet:
  if (Ethernet.begin(mac) == 0) {//Se o mac address não for estabelecido
    Serial.println("Verifique o cabo de RJ45");//notifica por m.série
// Não faz mais nada se o problema for físico. Aqui isto somente acontece quando o mac address entra em conflito, ou seja, quando não houver internet ou falha do cabo.
    resetEthernetShield();//Chama a função e repete a ligação;
       for(;;)
      ; }
// Imprime o endereço de IP local atribuido à placa arduino
```

```
  Serial.println("IP adquirido");//m.série notifica
  Serial.println(Ethernet.localIP());//m.série imprime o IP
//Fim auto Ethernet*********************************

   Ethernet.begin(mac,Ethernet.localIP());Inicia a ligação à internet.
 delay(1000);//delay para estabilidade
Serial.println("Ligando...");//Fim auto Ethernet

  postData();// postData é chamada aqui para enviar dados sobre o estado da ligação;


void postData() {// postData, para enviar os dados sobre a ligação cliente-servidor


  if (client.connect(server, 22000)) {// Se a ligação for bem sucedida
    Serial.println("Ligado");//O m.série notifica

    client.println("Arduino ligado");//O cliente notifica

  }
  else {//Caso contrário se a ligação falhar
    Serial.println("Ligação falhou");//O m.série notifica
    Serial.println("Deslligando");//O m.série notifica
    resetEthernetShield();//Chama a função e repete a ligação;

//em suma notifica e chama a função reset. Isto funciona quando o servidor está
desligado, o arduino insiste indefenidamente pela ligação IP que foi configurada pelo
utilizador.
    //é ótimo para não termos que fazer reset nós mesmos em instalação, sendo mais
prático.
  }
}
```

```
void resetEthernetShield() // Esta função no final do código é uma mera repetição dos parâmetros que se encontram no void setup(){}, sendo  somente chamada em condicionais, caso a ligação à internet falhe.
{
  delay(100);//delay para estabilidade
  Serial.println("Fazendo reset à Ethernet");//m.série notifica
  client.stop();//cliente pára.
//Auto Ethernet
  // iniciação da ligação à internet:
  if (Ethernet.begin(mac) == 0) {//Se o mac address não for estabelecido
    Serial.println("Verifique o cabo de RJ45");//m.série sugere solução
// Não faz mais nada se o problema for físico
    for(;;)
      ;
  }
// Imprime o endereço de IP local atribuido à placa arduino
  Serial.println("IP adquirido");//O m.série notifica
Serial.println(Ethernet.localIP());//m.série imprime o IP
//Fim auto Ethernet

   Ethernet.begin(mac,Ethernet.localIP()); //Inicia a ligação à internet.
   delay(1000);//delay para estabilidade
Serial.println("Ligando..."); //m.série notifica
  delay(100);//delay para estabilidade
    postData();();// postData é chamada aqui para enviar dados sobre o estado da ligação;

}
```

1.1 – Integração de sensores

Antes do “void setup(){}”, além dos parâmetros de ligação à internet para efeitos de demonstração de receção de dados ao servidor, declarei as variáveis onde seriam guardados valores como “long”, constantes numéricas, “float” para aproximar os dados analógicos em valores mais contínuos e a declaração dos pinos dos sensores bem como declarações de estado iniciais, valores de tolerância. Abaixo, iremos encontrar a azul as respetivas declarações de botões, sensores devidamente comentadas a verde, por questões de conveniência e iremos encontrá-las ao longo do programa até ao final.

```
long ct=0; //guarda 4 bytes, servirá para contagens tendo em conta a
long ct2=0; //frequência do micro controlador

// variáveis para o botão e led

//estados dos leds
const int ledvermelho = 0; //Estado lógico inicial
const int ledverde = 1; //Estado lógico inicial

const int gasPin = A1; // Input para o sensor de gás(A0)
float GAS;//variável float para valores mais precisos
//variáveis dos pinos
const int buttonPin = 12; // pino do 1º botão
const int buttonPin2 = 9;// pino do 2º botão
const int buttonPin3 = 8;// pino do 3º botão
const int buttonPin4 = 7; //pino do 4º botão
const int ledPin =  10;
int led2 = 11; //led vermelho
int buttonState = 0; //Estado lógico inicial do 1º Botão
int buttonState2 =0; //Estado lógico inicial do 2º Botão
int buttonState3 =0; //Estado lógico inicial do 3º Botão
int buttonState4= 0; //Estado lógico inicial do 4º Botão
```

```
//Variáveis do sensor de temperatura (TMP36)
float temp; //variável float para valores mais precisos
int sensorPin = A0; //O PIN ANALÓGICO DO tmp36 Vout, tem uma resolução de 10mV por grau centígrado com limite de 500mV para permitir temperaturas negativas.


int tempMin = 22;   // temperatura mínima
int tempMax = 35;   //temperatura máxima
int fan = 6;       // o pin onde o ventilador esta ligado
int fanSpeed = 0;//defenição da velocidade inicial do ventilador
int tolerancia = 6; //Tolerância em graus, para evitar o ligar e desligarconstante sempre que a temperatura máxima é atingida, sendo agora a tempMax = tempMin+tolerancia = 22+6 = 28 ºc
//Graças a isto, o ventilador só para de trabalhar quando a temperatura chega a 22 ºc
//Sem isto, o ventilador começava e parava constantemente entre valores próximos de 28 ºc para manter a temperaturaabaixo dos 28 ºc

int GASMAX =750;// defenição do valor máximo de gás permitido
int GASMIN =35;// defenição do valor mínimo de gás permitido
int TOLERANCIA=200;// para evitar ligar e desligar os ventiladores nos limites
int FAN =3;// ventilador de gás 1
int FAN2 =4;// ventilador de gás 2
int Led = 13 ; // define LED Interface
int SENSOR = 5 ; // Valor defenido para o sensor magnético
int val ; // defenição da variável val
int estado1 = 0;// Estado inicialdo sensor magnético
```

Nota: Decidi comentar(EXCLUIR) o LCD e o vetor de leds devido a problemas de interferência.

Contudo o contador Geiger faz a sua função principal, enviando as contações por minuto para o servidor.

```
//Geiger começa aqui
```

```
// livraria para o lcd
//#include <LiquidCrystal.h>

//Iniciação da livraria com os pinos
//LiquidCrystal lcd(3,4,5,6,7,8);



// Limites para a barra led
#define TH1 45 //Thresholds, são limites
#define TH2 95
#define TH3 200
#define TH4 400
#define TH5 600

// Conversão de CPM para Usv/h

#define CONV_FACTOR 0.00812

// Variáveis
//int ledArray [] = {10,11,12,13,9};
int geiger_input = 2;
long count = 0; //guarda 4 bytes, servirá para contagens tendo em conta a frequência do microcontrolador.
long countPerMinute = 0;
long timePrevious = 0;
long timePreviousMeassure = 0;
long time = 0;
long countPrevious = 0;
float radiationValue = 0.0;
// Fim Geiger
```

1.1.1 Void setup(){}

Para iniciar as variáveis, pinos, inciei-as como OUTPUT, INPUT tal como iniciei o monitor de série para ajudar a visualizara atividade enquanto tudo decorre.

```
// o led a ser ativado
  pinMode(ledPin, OUTPUT);// ler verde quando envia dados dos botões
   pinMode(led2, OUTPUT); //led vermelho quando não envia dados dos botões
  // o butão de este para premir
  pinMode(buttonPin, INPUT);

//Pinos do ventilador e sensor
 pinMode(fan, OUTPUT);//Pinos do ventilador
 pinMode(sensorPin, INPUT);// Pino declarado como input

   pinMode (Led, OUTPUT) ;    // defenição de LED como interface
pinMode (SENSOR, INPUT) ;  // defenição do sensor magnético como input

analogReference(EXTERNAL);//referência para valores analógicos estáveis

//Geiger
  pinMode(geiger_input, INPUT);
  digitalWrite(geiger_input,HIGH);
for (int i=0;i<5;i++){
//    pinMode(ledArray[i],OUTPUT); //As variáveis para o vetor em desuso
  }


  attachInterrupt(0,countPulse,FALLING); // pino interrupt 2, que facilita operações em
simutâneo sem interferências.
//Fim Geiger
}
}
```

1.2 – **Organização dos sensores**

Aqui decidi organizar no void loop(){} as funções de forma individual para ser mais fácil encontrar erros. Aqui no loop, o programa corre para verificar qual destas funções é chamada de forma repetida. Isto também impede o programa correr funções que desnecessáriamente usam processamento, quando estas não são chamadas, como o caso dos botões que só funcionam uma vez pressionados e no caso o sensor magnético.

```
void loop() {

carregarBotao1();//chama a função Botão1
  delay(1);       // delay para estabilidade
carregarBotao2();//chama a função Botão2
  delay(1);       // delay para estabilidade

carregarBotao3();//chama a função Botão3
  delay(1);       // delay para estabilidade

  carregarBotao4();//chama a função Botão4
  delay(1);       //delay para estabilidade

//os botões foram postos separadamente para melhor segurança e organização
ventilador1();//chama a função ventilador2, somente ativado pelo senso de temperatura
  delay(1); // delay para estabilidade

gas();//chama a função gas, ativa dois ventiladores ao atingir a tolerância de gás máxima
delay(1); // delay para estabilidade

doorsensor();//chama a função sensor magnético
delay(1); // delay para estabilidade

GEIGER(); //chama a função geiger
delay(1);  // delay para estabilidade

}
```

2.3 - As funções dos sensores

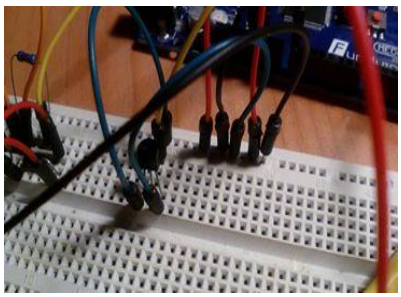
Aqui encontraremos os sensores que vão enviar os respetivos dados para o servidor a partir das funções estabelecidas, onde poderemos observar maioria das variáveis declaradas acima.

```
void carregarBotao1() {//Quando a função é chamada executa

if((debounceButton(buttonPin)==1) && (buttonState==0))//
{
  buttonState=1;//o uso de estados aqui serve para “bloquear” o botão no estado inicial, limitando o funcionamento a uma única mensagem por pressionamento.
Serial.println("Nurse Call");//m.série notifica
   client.println("Nurse Call");//cliente notifica
digitalWrite(ledPin, HIGH);//Acende led verde quando envia dados
    digitalWrite(led2, LOW);//Apaga led vermelho quando envia dados
//aqui teremos a menságem dependente do estado do botão pressionado
}

if((debounceButton(buttonPin)==0) && (buttonState==1))
{
  buttonState=0;
digitalWrite(ledPin, LOW);//Apaga led verde quando nada é enviado
digitalWrite(led2, HIGH);//Acende led vermelho quando nada é enviado
}  //

}
// Os seguintes três botões têm a mesma configuração e princípio, mudando somente a númeração para 2,3,4 respetivamente.
```

```
void carregarBotao2() {

if((debounceButton2(buttonPin2)==1) && (buttonState2==0))
{
  buttonState2=1;
   Serial.println("Next Patient");
client.println("Next Patient");
digitalWrite(ledPin, HIGH); //Acende led verde quando envia dados
    digitalWrite(led2, LOW); //Apaga led vermelho quando envia dados
//aqui teremos mais uma mensagem dependente do estado do botão pressionado
}

if((debounceButton2(buttonPin2)==0) && (buttonState2==1))
{
  buttonState2=0;
digitalWrite(ledPin, LOW); //Apaga led verde quando nada é enviado
    digitalWrite(led2, HIGH); //Acende led vermelho quando nada é enviado
}  //
}


void carregarBotao3() {
if((debounceButton3(buttonPin3)==1) && (buttonState3==0))
{
  buttonState3=1;
   Serial.println("X Ray room");
      client.println("X Ray room");
digitalWrite(ledPin, HIGH); //Acende led verde quando envia dados
    digitalWrite(led2, LOW); //Apaga led vermelho quando envia dados
 //aqui teremos mais outra mensagem dependente do estado do botão pressionado
}

if((debounceButton3(buttonPin3)==0) && (buttonState3==1))
{
```

```
  buttonState3=0;
digitalWrite(ledPin, LOW); //Apaga led verde quando nada é enviado
    digitalWrite(led2, HIGH); //Acende led vermelho quando nada é enviado
}
}

void carregarBotao4() {
if((debounceButton4(buttonPin4)==1) && (buttonState4==0))
{
  buttonState4=1;
   Serial.println("Maternity Ward");
client.println("Maternity Ward");
digitalWrite(ledPin, HIGH); //Acende led verde quando envia dados
    digitalWrite(led2, LOW); //Apaga led vermelho quando envia dados
 //e a menságem do último botão
}

if((debounceButton4(buttonPin4)==0) && (buttonState4==1))
{
  buttonState4=0;
digitalWrite(ledPin, LOW); //Apaga led verde quando nada é enviado
    digitalWrite(led2, HIGH); //Acende led vermelho quando nada é enviado
}  //
}
```

```
void ventilador1() {//função ventilador é executada

int reading = analogRead(sensorPin); //lê o sensor de temperature em volts

loat voltage = reading * 5.0;// Conversão da leitura para voltagem
voltage /= 1024.0;

// Conversão da voltage para temperatura
float temperatureC = (voltage - 0.5) * 100 ;  //cpnversão de 10 mv por grau com 500 mV de "offset"
//para graus ((voltagem - 500mV) vezes 100)

// se quisermos em fahrenheit
 float temperatureF = (temperatureC * 9.0 / 5.0) + 32.0;
// Deixei fahrenheit comentado porque é pouco usado nestes lados da europa.
 if (ct==100)// Aqui temos o micro controlador a contar 100 vezes antes de ler e emitir o valor da voltagem e temperatura.
Útil se quisermos ter várias tarefaz a serem executadas pelo mesmo micro controlador, sem interrupções.
 {
   Serial.print(voltage); //monitor de série notifica o utilizador
   Serial.println(" volts");//monitor de série notifica o utilizador
   Serial.print(temperatureC);//monitor de série notifica o utilizador
   Serial.print(" Celsius ");//monitor de série notifica o utilizador
client.println(temperatureC) && client.print(" Celsius =");
// Cliente notifica servidor com dados da temperatura em Celsius
//Serial.print(temperatureF); Serial.println(" degrees F");
//Serial.println();

 ct=0;// Aqui o contador faz reset e volta a zero
 }
```



```
 delayMicroseconds(1); //espera de microsegundo
ct++;//Adiciona +1 para iniciar a contagem, contando de novo até 100 para efetuar a leutura e envio novamente.

if(temperatureC < tempMin) {      //se a temperatura for mais baixa que o valor mínimo+tolerancia
      fanSpeed = 0;      // O ventilador fica imóvel
      digitalWrite(fan, LOW);
  }
  if((temperatureC >= tempMin+tolerancia) && (temperatureC <= tempMax))    //se a temperatura for mais alta que o valor mínimo+tolerancia
  {
      fanSpeed = map(temp, tempMin, tempMax, 240, 255); // A velocidade atual do ventilador é máxima
      analogWrite(fan, fanSpeed);  //  Rodar o ventilador
// client.println("Ventilador ativo");
       Serial.print(" Ventilador ativo ");//m.série notifica
}
//
 }

void gas(){ //função gás é executada

//fan function
//getting the voltage reading from the temperature sensor
int reading2 = analogRead(gasPin);


    if (ct2=10000000) )// Aqui temos o micro controlador a contar 10000000 vezes antes de ler e emitir o valor da voltagem e temperatura.
Útil, mais uma vez se quisermos ter várias tarefaz a serem executadas pelo mesmo micro controlador, sem interrupções entre si.

{
```

```
Serial.println(analogRead(gasPin));
   Serial.print("GAS ");
ct2=0;// Aqui o contador faz reset e volta a zero
  }
  delayMicroseconds(1);
  ct2++;//Adiciona +1 para iniciar a contagem, contando de novo até 10000000 para efetuar a leutura e envio novamente.

if(reading2 < GASMIN) {       // se o valor do gás for mais baixo que o valor mínimo+tolerancia
fanSpeed = 0;      // ventiladors ficam imóveis
      digitalWrite(FAN, LOW);
      digitalWrite(FAN2, LOW);
}
   if((reading2 >= GASMIN+TOLERANCIA) && (reading2 <= GASMAX))  //se o valor do gás for mais alto que o valor mínimo+tolerancia
   {
      fanSpeed = map(GAS, GASMIN, GASMAX, 240, 255); // A velocidade atual dos ventiladors é máxima
      analogWrite(FAN, fanSpeed);  //Rodar o ventilador
       analogWrite(FAN2, fanSpeed);  //Rodar o ventilador
       client.println("LEAK ALERT");//cliente notifica o servidor
//  Serial.print(" Ventilador ativo ");
   }
    }
```

```
void doorsensor(){//função sensor magnético é executada
      val = digitalRead (SENSOR) ; // leitura do sensor

if((val==1) && (estado1==0))
  {
   estado1=1;
     client.println("Door Open");//cliente notifica
     Serial.print(" Door Open ");// m.série notifica
}
  if((digitalRead(SENSOR)==0) && (estado1==1))
  {;//o uso de estados aqui serve para “bloquear” o botão no estado inicial, limitando o funcionamento a uma única mensagem por pressionamento. É o mesmo princípio usado nos botões, por detetar ou não um campo magnético.

estado1=0;
     client.println("Door Closed");//Cliente notifica
     Serial.print(" Door Closed");//m.série notifica
}
     }
```

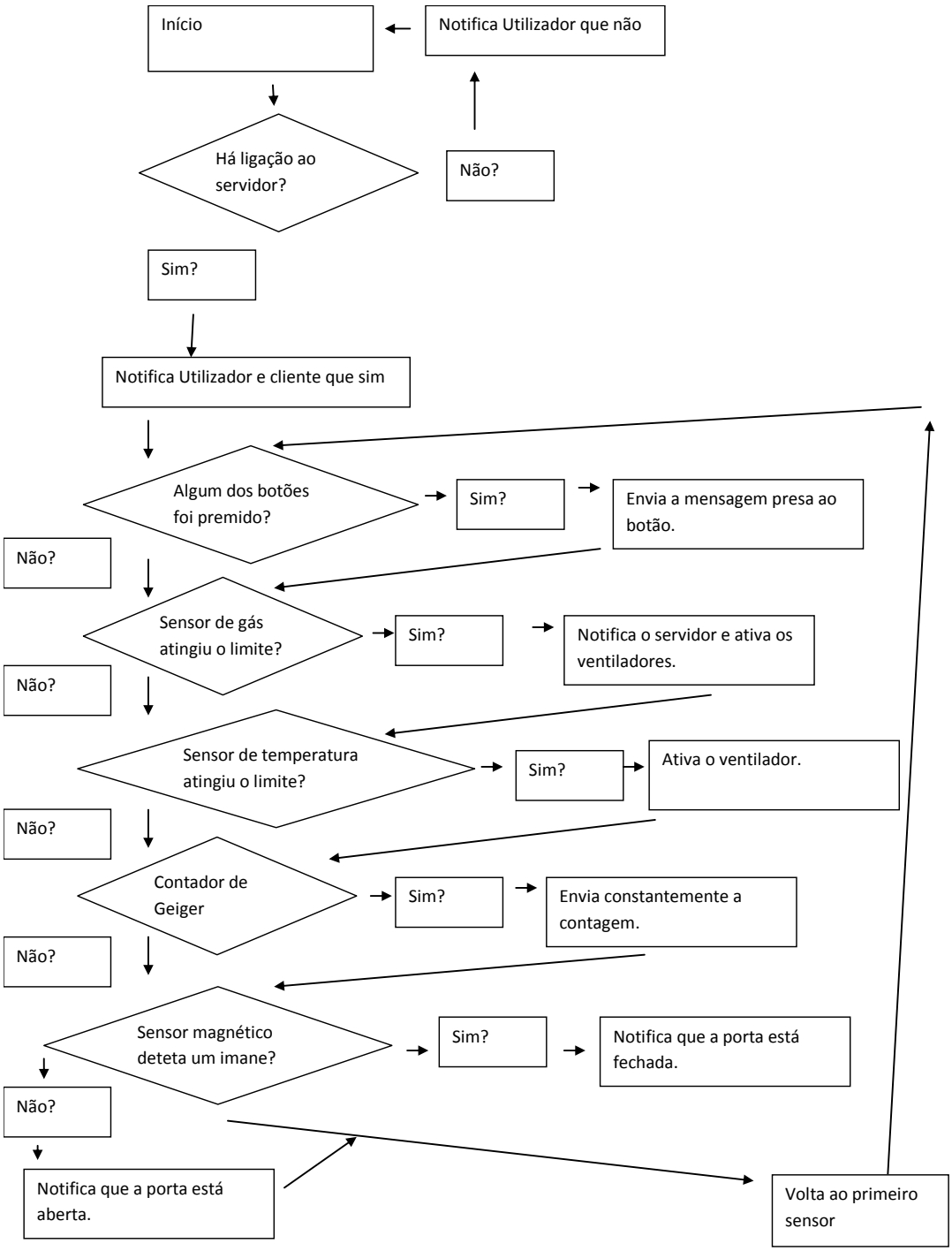
```
//aqui estão as funções debounce que serão chamadas para evitar contagens erradas devido às oscilações metálicas dos botões, pois sendo constituídos por partes que oscilam, um botão apesar de bloqueado pode enviar duas mensagens,embora do ponto de vista virtual só devesse enviar uma.
DebounceButton, limita o tempo de contagem do microprocessador, para que este apenas tenha tempo de ouvir a primeira oscilação de todas, reduzindo a probabilidade de ler mais do que uma oscilação ao premir um botão, para próximo de zero.

boolean debounceButton(boolean state)
{
  boolean stateNow = digitalRead(buttonPin);
  if(state!=stateNow)
  {
    delay(10);
    stateNow = digitalRead(buttonPin);
}
  return stateNow;//Devolve o estado

}

// As seguintes três funções debounce têm a mesma configuração e princípio, mudando somente a númeração para 2,3,4 respetivamente.

boolean debounceButton2(boolean state)
{
  boolean stateNow = digitalRead(buttonPin2);
  if(state!=stateNow)
  {
    delay(10);
    stateNow = digitalRead(buttonPin2);
  }
  return stateNow;
```

```
}

boolean debounceButton3(boolean state)
{
  boolean stateNow = digitalRead(buttonPin3);
  if(state!=stateNow)
  {
    delay(10);
    stateNow = digitalRead(buttonPin3);
}
  return stateNow;

}

boolean debounceButton4(boolean state)
{
  boolean stateNow = digitalRead(buttonPin4);
  if(state!=stateNow)
  {
    delay(10);
    stateNow = digitalRead(buttonPin4);
}
  return stateNow;

}
```

```
void GEIGER(){//função Geiger é exectada
   if (millis()-timePreviousMeassure > 10000){//Aqui com a estrutura millis() o micro controlador devolve o número de milisegundos desde que oo arduino começou a corer o programa. O número volta a zero após 48 dias.

   countPerMinute = 6*count;//contagens por minute em conversão
radiationValue = countPerMinute * CONV_FACTOR;
   timePreviousMeassure = millis();
Serial.print("cpm = "); //Notificção em m.série
   Serial.print(countPerMinute,DEC); //Notificção em m.série
   Serial.print(" - ");//Notificção em m.série
   Serial.print("uSv/h = ");//Notificção em m.série
   Serial.println(radiationValue,4); //Notificção em m.série
      client.println("         cpm = "); //Notificção pelo cliente
   client.println(countPerMinute,DEC); //Notificção pelo cliente

//como referi acima, o LCD encontra-se em desuso devido a conflitos com o resto to programa
//lcd.clear();
  //lcd.setCursor(0, 0);
  //lcd.print("CPM=");
  //lcd.setCursor(4,0);
  //lcd.print(countPerMinute);  //  lcd.setCursor(0,1);
  //lcd.print(radiationValue,4);
  //lcd.setCursor(6,1);
//lcd.print(" uSv/h");

// O vetor também se encontra em desuso devido a conflitos com o programa
//led var setting
   if(countPerMinute <= TH1) ledVar(0);
   if((countPerMinute <= TH2)&&(countPerMinute>TH1)) ledVar(1);
   if((countPerMinute <= TH3)&&(countPerMinute>TH2)) ledVar(2);
   if((countPerMinute <= TH4)&&(countPerMinute>TH3)) ledVar(3);
```

```
    if((countPerMinute <= TH5)&&(countPerMinute>TH4)) ledVar(4);
    if(countPerMinute>TH5) ledVar(5);

    count = 0;

}
    }

//esta função lida com o pino attachIterrupt, complementando a função GEIGER
void countPulse(){
  detachInterrupt(0);
  count++;
  while(digitalRead(2)==0){
}
  attachInterrupt(0,countPulse,FALLING);
}
// Esta função bem podia estar toda comentada, pois desativei os leds como mencionei
previamente, mas tenciono pô-los a funcionar em breve, por isso apenas comentei o
essencial para permitir que o programa corra sem interferências.
void ledVar(int value){
  if (value > 0){
    for(int i=0;i<=value;i++){
//      digitalWrite(ledArray[i],HIGH);
    }
    for(int i=5;i>value;i--){
   //   digitalWrite(ledArray[i],LOW);
}
  }
  else {
    for(int i=5;i>=0;i--){
//   digitalWrite(ledArray[i],LOW);
}
  }
}
```

FLUXOGRAMA – REPRESENTAÇÃO DA TOTALIDADE DO **CÓDIGO**



Decidi alimentar o projeto em questão com uma fonte de computador, excluindo o sensor de temperatura por questões técnicas.

Botões de pressão:

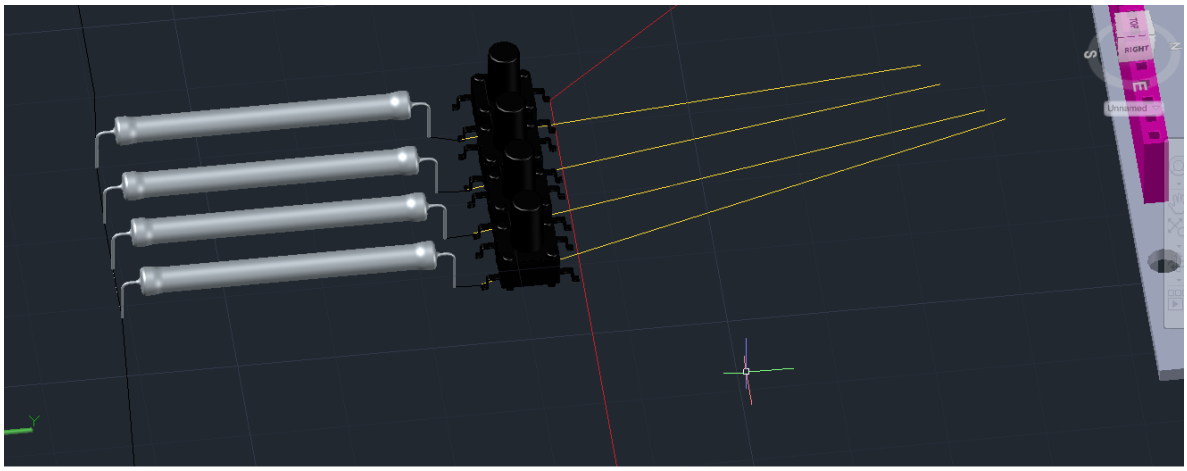
Estes em si requerem uma ligação à corrente protegida por uma resistência de 1k, outra à massa e outra a uma porta lógica, que irá detetar a passagem de corrente à massa ao premir o botão.

Os ventiladotres:

Requerem um transístor2n222 para controlar a frequência das rotações por portas que suportam PWM, protegidas por uma resistência na base e um diódo ligado ao coletor para evitar s inversões de carga das espiras existentes nos ventiladores.

O sensor de temperatura, teve de ser ligado únicamente aos 5V e massa analógica do arduíno usando a porta AREF, de modo a manter os valores de temperatura mais estáveis usando 5V como tensão de referência.

Todos os restantes sensores, gás, contador geiger,sensor magnético, puderam ser ligados diretamente à massa e corrente da fonte de alimentação, devido ao facto de já estarem embutidos nos seus próprios circuitos integrados, devidamente preparados para 5v.



FASE 2

PLANEAMENTO DO PROJETO

Planeei construir uma maquete em esferovite para representar o funcionamento do meu Projeto baseado na placa Arduino, numa situação problema que visa representar um local de trabalho hospitalar/laboratorial.

Recursos materiais:

Arduino Mega 2560;
Ethernet Shield;
Fonte de alimentação 300 W aproveitadapara alimentar o projeto;
Cabos fêmea de pc aproveitados;
Três placas de esferovite prensada(por dimensões aqui(Comprimento*Largura*Altura)) para simular a infra-estrutura 3D;
Três ventiladores de pc aproveitadas;
X fios jumper para ligações;
Cabo RJ45;
Cabo USB-RS232;
Sensor de Temperatura;
Sensor de GÁS;
Sensor de campo magético;
Contador de Geiger;
Quatro botões;
Três transistores 2n222;
Três díodos IN4007;
Quatro resistências de 1K Ohm;
Uma breadboard;
Cartolina cinzenta;

Cortei a esferovite para servir de base com 100cm x 60cm, tendo as paredes por sua vez 14cm de altura.

Dividi a secção num total de seis salas, quatro de um lado, duas do outro, separadas entre si por paredes com 4cm de espessura e divididas por um corredor ao meio, perfazendo um rectângulo. Criei pelo corredor afora entradas, de modo,a esconder os cabos de forma subterrânea, perfurei uma das paredes para embutir dois dos ventiladores numa sala onde se encontra o sensor de gás.

Na segunda sala, coloquei um sensor magnético, que utilizei para nos indicar o estado da porta de entrada entre “Aberto” e Fechado”, correspondendo estes aos valores lógico “1” e “0” respetivamente.

Na terceira sala, coloquei o contador de Geiger, passando unicamente o ”tubo Geiger” para o centro da sala.

A sala principal, onde se encontra a fonte de alimentação, Arduino, Ethernet Shield, breadboard, quatro botões de pressão, sensor de temperatura, acabou por ocupar duas salas devido ao tamanho da fonte em si, o espaço para dar folga aos cabos e prática manual. Nesta Sala, o sensor de temperatura está encaixado no microchip do EthernetShield, sendo este arrefecido pelo terceiro ventilador aos 26ºc. Os botões de pressão simulam os callpoints na vida real do programa Connexall da Globestar Systems.

Chegando aqui, tendo improvisado o espaço e usando duas salas a mais, para a sala principal, acabamos na mesma por ficar com duas salas disponíveis para a PARTE II, para uma extensão futura deste projeto igualmente com demonstração, denominada PARTE II – Output.

FASE 3

REFLEXÃO CRÍTICA

Tendo finalizada a PAP, concluí ter atingido os objetivos assim que foi estabelecida uma ligação ao servidor por parte do meu cliente arduíno, sendo a proposta inicial escrita deste projeto uma solução mais rápida e económica para placa de Input de dados.

Da minha perspetiva, fazendo baixar o custo das placas Input e tirando proveito das redes locais para transferência de dados, a GlobeStar Systems ficaria a ganhar tanto na eficácia como em questões financeiras.

Na situação problema, ao lidar com sensores e envios de dados e sinal, pude constatar que a placa tem agilidade para uma variedade de situações em que é necessário lidar com valores analógicos, além dos valores digitais. A parte mais desafiante deste projeto revelou-se na situação problem, ao introduzir o Contador Geiger, que sendo um Tubo metálico com a sua placa específica para seu funcionamento mais complexa na programação, causou algumas interferências durante os testes. Mencionando testes, durante as pesquisas encontrei o Notepad++, que facilitou a comparação de diferentes rascunhos de código para ver as diferenças previamente editadas a cores, permitindo-me trabalhar mais depressa. Entre pesquisas, adquiri o hábito de salvar constantemente o rascunho atual de trabalho, numa pasta de forma organizada, comprimir e enviar para várias drives para garantir a segurança, após um acidente.

Em suma, este trabalho acabou por se tornar numa realização para a minha pessoa, ao constatar que fui além dos objetivos atingidos após inúmeros testes e pesquisas feitas, representando uma boa experiência para a minha formação e para os meus futuros estudos\profissão.

Agora sinto-me mais perto da área a que aspiro dedicar as minhas horas laborais, podendo aplicar estes conhecimentos numa diversidade de situações.

NETGRAFIA

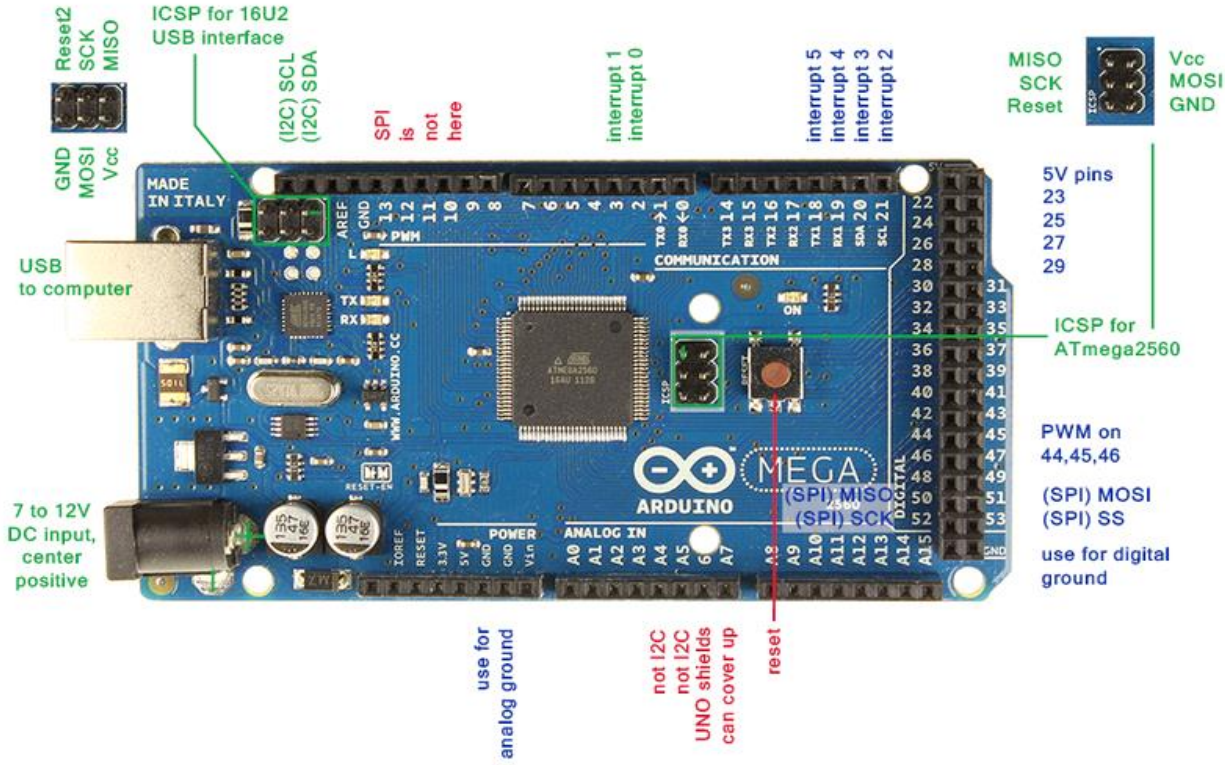
https://www.arduino.cc/en/Reference/HomePage

http://www.instructables.com/

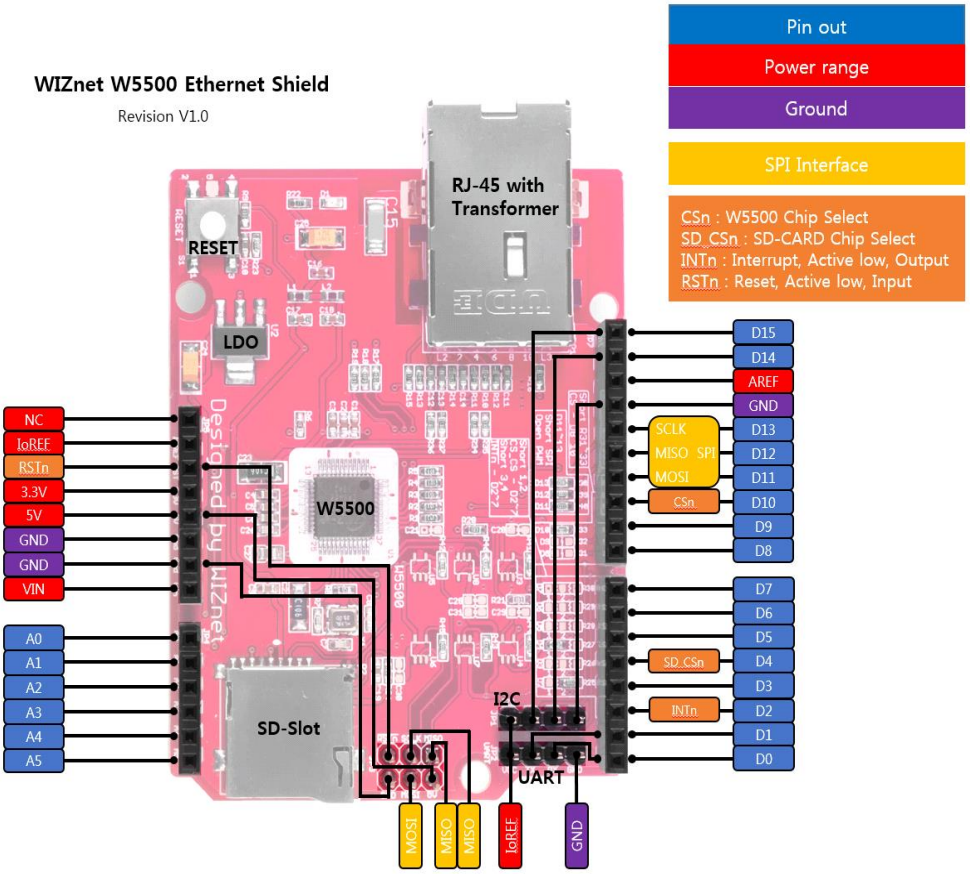
ANEXO I

Esquema das placas Arduino

Na Figura 1 podemos observar a A. Mega 2560, com todas as entradas descriminadas.



ANEXO II



https://www.arduino.cc/en/Main/ArduinoBoardEthernet

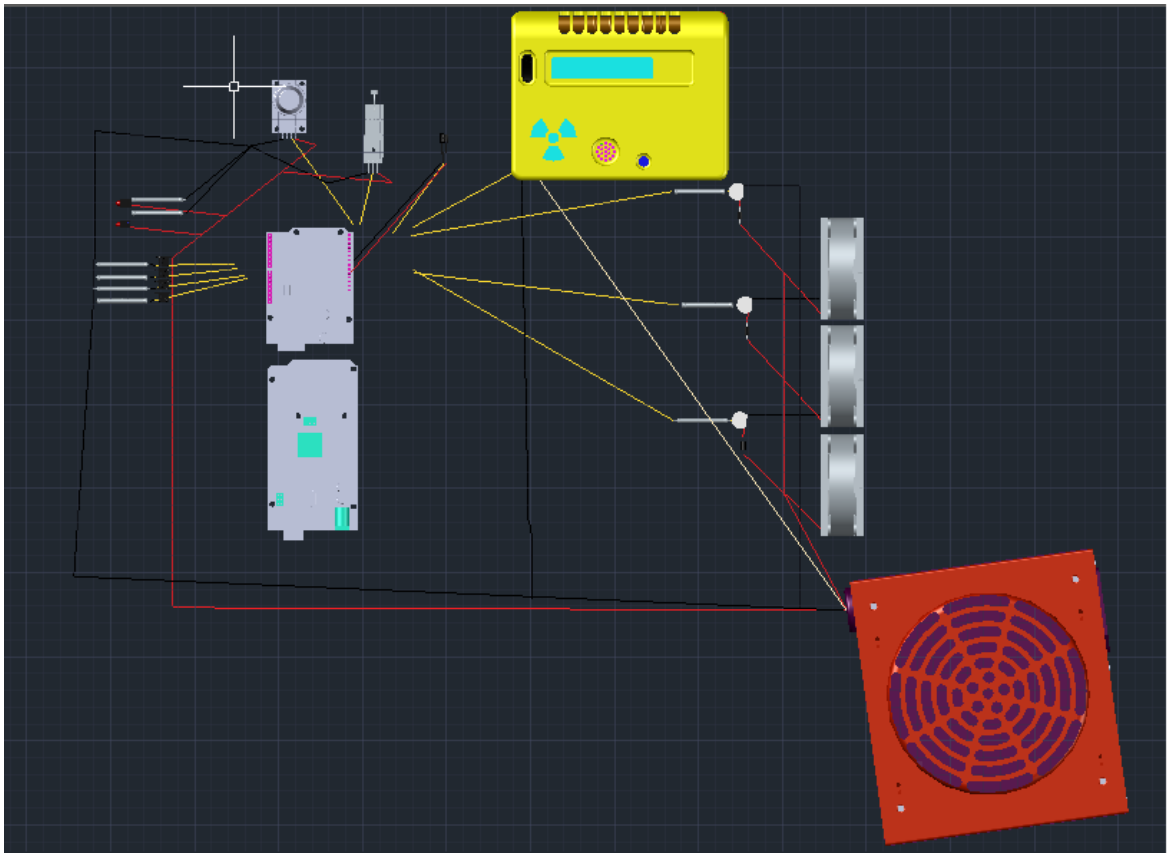
ANEXO III

Contador Geiger:

https://www.cooking-hacks.com/documentation/tutorials/geiger-counter-radiation-sensor-board-arduino-raspberry-pi-tutorial/

ANEXO IV

Projecto visto de cima



ANEXO V

Projecto visto de outra perspectiva