Sumário

[Introdução 1](#_Toc531973858)

[Objetivos 1](#_Toc531973859)

[Material e Métodos 1](#_Toc531973860)

[Resultados 1](#_Toc531973861)

[Discussão 2](#_Toc531973862)

[Matéria encaminhada para publicação 2](#_Toc531973863)

[Perspectiva de continuidade do trabalho 2](#_Toc531973864)

[Agradecimentos 2](#_Toc531973865)

[Apêndice A – Tabela de modelos de baterias 2](#_Toc531973866)

[Apêndice B – Tabela de valores de consumo completa 2](#_Toc531973867)

# 

# Introdução

O agronegócio no Brasil possui caráter de grande envergadura para toda a economia do país. Somente em maio de 2017, as exportações atingiram US$ 9,68 bilhões, valor que corresponde a aproximados 13% de aumento em referência ao mesmo período do ano anterior. Somente o valor desse superávit comercial causou um aumento de 790 milhões de dólares, demonstrando que esse setor possui alta taxa de crescimento. Dentre parte das exportações, está contido o setor de carnes, com arrecadação de 1,22 bilhão de dólares em 2017. ([SANTANDER](#referencia_santander), 2017)

Todavia, mesmo com notório crescimento, muitos fazendeiros passam por inúmeras dificuldades para acompanhar seu gado, devido a sua ausência por problemas do cotidiano que simplesmente impedem a presença diária do fazendeiro para o acompanhamento. Devido a isso, surgem ocasiões que geram transtornos e podem gerar prejuízos, tais como perder vacas por terem fugido da propriedade, por ficarem atoladas, ou mesmo perder muito tempo procurando o gado em um determinado local, sendo que o mesmo pode estar no outro extremo da região. Levando em consideração tais problemas, propõem-se formas de monitorar o gado à distância, para um melhor gerenciamento por parte dos fazendeiros.

A solução proposta por todo o projeto[[1]](#footnote-1) visa realizar o monitoramento e gerenciamento dos animais à distância, usando de tecnologia especificas, tais como o IoT - *Internet of Things*, cuja tradução direta é “Internet das Coisas”, microchips inteligentes como o ESP8266, ESP32, ATtiny13 dentre outros e protocolos de comunicações e de gerenciamento de sinais, como o Wi-Fi, Bluetooth Low Power, RSSI e MQTT.

Os principais preceitos do IOT se baseiam na ligação entre alguma “coisa” física ao meio das comunicações de rede dinâmica e global, portando, dessa maneira, a capacidade de configurar de forma inteligente ou interagir com o objeto físico em questão. Para tal interfaceamento, utilizam-se sistemas eletrônicos pré-programados conectados em alguma rede, bem como na rede global. Esta comunicação pode se dar pelo uso do Wi-Fi ou do Bluetooth.

Os chips inteligentes utilizados detêm a função de controlarem a comunicação entre si, utilizando protocolos de comunicações específicos, e gerenciar os dados colhidos. Para isso, serão pré-programados afim de cumprirem com suas respectivas funções.

Para o amplo emprego dessas tecnologias, é preciso viabilizar algumas características fundamentais no sistema, sendo estes o tamanho o projeto final, o custo e a autonomia, que é definida como o período máximo que o circuito poderá ser mantido em constante funcionamento sem apresentar falhas. Para se alcançar um bom valor, foram empregadas as mais recentes formas de tecnologia de baixo consumo disponíveis no mercado, que consistiu no emprego de Microcontroladores específicos e de modos de comunicação aplicados ao baixo consumo, além do aprimoramento de técnicas justapostas que relacionam dois ou mais modos de operação para uma combinação satisfatória de baixo consumo.

Um desafio no que diz respeito ao emprego desse sistema está justamente em cumprir com uma boa viabilidade o tamanho e o custo final do sistema eletrônico, sem que se perca a autonomia necessária para o funcionamento do código fonte[[2]](#footnote-2), que será desenvolvido paralelamente pela UFSJ – Universidade Federal de São João Del-Rei. Será considerado um tamanho que caiba em uma etiqueta utilizada pelos fazendeiros para a identificação de cada animal, como mostrado na [Figura 1](#figura_1), a qual é presa em suas respectivas orelhas. Seguidamente, será analisado o custo do protótipo, a qual recorre da compra de basicamente dois principais componentes: o processador utilizado e a bateria escolhida para alimentar o circuito.

**Figura 1 Modelo de etiqueta que será utilizada**



Fonte: <http://portuguese.animal-microchip.com> (2018).

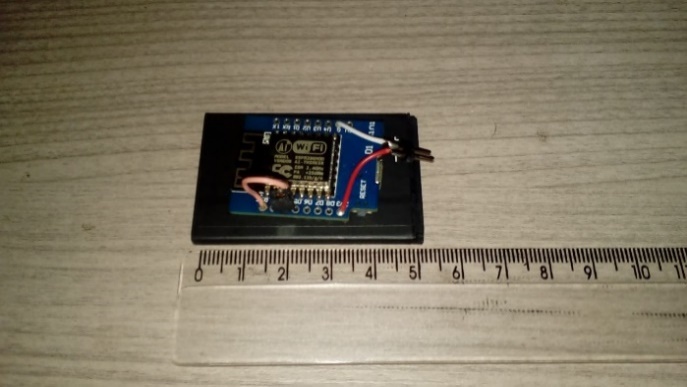
Levando-se esses problemas em consideração, os Microcontroladores escolhidos para a realização dos estudos foram os referentes à família ESP e a família ATtiny, sendo seus principais polos o ESP8266, o ESP32 e o ATtiny13A-PU. Todos foram escolhidos por apresentarem alto desempenho em eficiência energética. Ademais, esses Microcontroladores apresentam a portabilidade de modelos de comunicação distintos, de modo que o ESP8266 possa utilizar o Wi-Fi, o ESP32 possa utilizar o BLE - *Bluetooth Low Power*, e, por fim, o ATtiny13 possui compatibilidade com módulos de rádio frequência, como o NRF24l01.

Com tais Microcontroladores, pode-se contornar o problema de custo e autonomia, visto que se apresentam de fácil acesso e apresentam tecnologias já inclusas de baixo consumo. Além disso, apresentam boa compatibilidade com o tamanho total do projeto, facilitando a instalação na etiqueta.

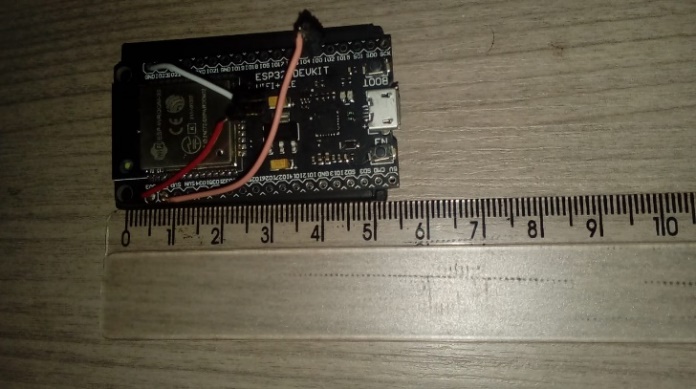
Após a escolha dos referidos Microcontroladores, foram realizadas inúmeras análises de diversos modelos de baterias, com a finalidade de se obter o modelo que melhor atende às necessidades de tamanho, custo e capacidade energética, para acréscimo da autonomia.

Mais especificamente, nesse projeto será montado um protótipo operacional de um dos nós da rede de integração de monitoramento do gado, utilizando um dos dois chips ESP, com o intuito de chegar a possíveis soluções para o projeto. Nas Figuras 2 e [3](#figura_3), apresentam-se as versões que servirão como base para os futuros protótipos, levando em consideração os modelos de bateria da mesma proporção, além dos pequenos componentes externos necessários para o funcionamento do circuito.

**Figura 2 Modelo de protótipo utilizando o ESP8266**



**Figura 3 Modelo de protótipo utilizando o ESP32**



# Objetivos

A solução proposta por todo o projeto visa realizar o monitoramento e gerenciamento dos animais à distância, usando de tecnologia especificas, tais como o IoT - *Internet of Things*, cuja tradução direta é “Internet das Coisas”, microchips inteligentes como o ESP8266, ESP32, ATtiny13 dentre outros e protocolos de comunicações e de gerenciamento de sinais, como o Wi-Fi, Bluetooth Low Energy, RSSI e MQTT.

Para apresentar uma nova forma de solução dos problemas enfrentados pelos fazendeiros, o objetivo é integrar múltiplas tecnologias, se apropriando do uso do IoT em conjunto com outros sistemas, tais como RSSI, Wi-Fi, Bluetooth Low Energy, criando um conjunto que possibilite ao fazendeiro gerenciar e localizar seu gado.

Esse sistema seria composto por antenas fixas espalhadas na área a ser monitorada e pela placa de transmissão acoplada na etiqueta de cada animal. No momento em que a placa acoplada transmite sinal para as antenas, o sinal seria processado e então serial calculada a sua localização aproximada por meio da trilateração, e esses dados ficariam dispostos ao agricultor.

A figura X esboça de forma simplificada o sistema de transmissão de dados entre uma vaca com a placa desenvolvida acoplada em sua etiqueta de identificação entre duas antenas fixas. A partir dos cálculos realizados o fazendeiro poderá saber a localização aproximada da vaca em sua fazenda.



# Material e Métodos

Inicialmente, foi realizada uma pesquisa dos diversos modelos e tipos de bateria para análise, com a finalidade de se optar por aquela que tenha maior eficiência nos critérios já mencionados (tamanho, custo e carga energética). Após a aferição de mais de 200 modelos diferentes, foram realizados vários filtros para a seleção destes, resultando na Tabela 1 que possui vinte e oito modelos.

Logo após a separação desses vinte e oito modelos, foram realizadas novas pesquisas para delimitar possíveis características que filtrassem novamente os mesmos. Foram consultados os vários Datasheets referentes aos modelos de ESP e dos modos de funcionamento, chegando a especificamente cinco modelos que se apresentam como possíveis escolhas finais. Essas baterias foram adquiridas para a realização de experimentos práticos, para assim se chegar a uma conclusão definitiva.

De modo concomitante, foram desenvolvidas placas para realizar testes na placa ESP8266, com o intuito de aferir o consumo elétrico em diferentes situações. A primeira placa, figura X, foi projetada para drenar uma corrente fixa em cada digital I/O do ESP, a partir de um conjunto de vários leds ligados em current sourcing em série com um resistor limitador de corrente em cada porta de saída. O seu esquemático eletrônico pode ser observado na figura X.

De modo concomitante, para se chegar a um valor confiável do melhor modelo de bateria, foram realizados diversos testes experimentais, todavia, para efetuar tais testes foi feita uma placa *shield* para a realização desses experimentos. Em relação ao ESP8266, foi projetado um circuito*,* cujo esquemático feito no *software Proteus* segue logo em seguida (ver figura x). Este, possui 8 leds que estão ligados em *current source* com 8 pinos digitais da placa WeMos, que serão controlados por um código fonte2 anteriormente programado. A placa foi projetada para drenar uma corrente fixa em cada digital I/O do ESP, para assim, aferir o consumo de corrente do chip.

Após os primeiros testes de consumo do ESP, foram desenvolvidos diferentes códigos para que se possa aferir o consumo de energia por cada chip em cada modo de operação, modo de transmissão de dados e em cada modo de ‘Sleep ’. Cada um desses códigos foi desenvolvido utilizando o Arduino, software IDE, sendo que cada um foi programado utilizando a linguagem C++. Após esta etapa, os códigos foram armazenados, junto aos demais arquivos do projeto, na plataforma Git Hub, com o nome “W8jonas”.

Ao final da etapa de programação e de aferição do consumo de corrente pela shield 1. Foi feita uma segunda *shield*, (ver figura X), capaz de alterar entre os diferentes modos de funcionamento por meio de um botão. O código fonte inserido na placa ESP permite a troca de funções exercidas pelo ESP, desse modo, a cada pressionar do botão, uma nova forma de funcionamento é estabelecida.

O código tem como objetivo estabelecer 6 funções diferentes para o Arduino executar cada uma delas de modo separado. Esse programa apresenta dois principais núcleos de funcionamento, a leitura do pressionar do botão, cuja a lógica é compreendida entre as chaves do comando **void setup(){ }**, e a parte da alteração das funções, em cada função declarada demonstra um modo de funcionamento diferente.

O código na íntegra se encontra abaixo:

#define Output\_1 D8

#define Output\_2 D7

#define Output\_3 D6

#define Output\_4 D5

#define Output\_5 D4

#define Output\_6 D3

#define Output\_7 D2

#define Output\_8 D1

#define entrada\_botao D0

void funcao\_\_();

void funcao\_0();

void funcao\_1();

void funcao\_2();

void funcao\_3();

void funcao\_4();

int operacao = 0;

boolean leitura = true;

void setup() {

pinMode (Output\_1, OUTPUT);

pinMode (Output\_2, OUTPUT);

pinMode (Output\_3, OUTPUT);

pinMode (Output\_4, OUTPUT);

pinMode (Output\_5, OUTPUT);

pinMode (Output\_6, OUTPUT);

pinMode (Output\_7, OUTPUT);

pinMode (Output\_8, OUTPUT);

pinMode (entrada\_botao, INPUT);

Serial.begin(115200);

}

void loop() {

leitura = digitalRead(entrada\_botao);

Serial.println(leitura);

if (leitura == LOW ){

operacao++;

delay(500);

}

funcao\_2();

}

void funcao\_\_ (){

Serial.println("funcao 00 ");

digitalWrite(Output\_1, LOW);

digitalWrite(Output\_2, LOW);

digitalWrite(Output\_3, LOW);

digitalWrite(Output\_4, LOW);

digitalWrite(Output\_5, LOW);

digitalWrite(Output\_6, LOW);

digitalWrite(Output\_7, LOW);

digitalWrite(Output\_8, LOW);

}

void funcao\_0() {

Serial.println("funcao 0 ");

digitalWrite(Output\_1, HIGH);

}

void funcao\_1() {

Serial.println("funcao 1 ");

digitalWrite(Output\_1, HIGH);

digitalWrite(Output\_2, HIGH);

digitalWrite(Output\_3, HIGH);

digitalWrite(Output\_4, HIGH);

digitalWrite(Output\_5, HIGH);

digitalWrite(Output\_6, HIGH);

digitalWrite(Output\_7, HIGH);

digitalWrite(Output\_8, HIGH);

}

void funcao\_2() {

Serial.println("funcao 2 ");

digitalWrite(Output\_1, LOW);

digitalWrite(Output\_2, LOW);

digitalWrite(Output\_3, LOW);

digitalWrite(Output\_4, LOW);

digitalWrite(Output\_5, LOW);

digitalWrite(Output\_6, LOW);

digitalWrite(Output\_7, LOW);

digitalWrite(Output\_8, LOW);

int cont = 1;

float resp = 1;

float resp2 = 1;

for(int AA = 0; AA < 500; AA++){

resp = 3 + sin(resp)/cos(resp\*resp/2) \* sqrt(sqrt(resp\*resp));

resp = resp \* 0.5;

resp2 = sqrt(AA);

yield();

}

}

void funcao\_3() {

Serial.println("funcao 3 ");

digitalWrite(Output\_1, HIGH);

int cont = 1;

float resp = 1;

float resp2 = 1;

for(int AA = 0; AA < 500; AA++){

resp = 3 + sin(resp)/cos(resp\*resp/2) \* sqrt(sqrt(resp\*resp));

resp = resp \* 0.5;

resp2 = sqrt(AA);

yield();

}

}

void funcao\_4() {

Serial.println("funcao 4 ");

digitalWrite(Output\_1, HIGH);

digitalWrite(Output\_2, HIGH);

digitalWrite(Output\_3, HIGH);

digitalWrite(Output\_4, HIGH);

digitalWrite(Output\_5, HIGH);

digitalWrite(Output\_6, HIGH);

digitalWrite(Output\_7, HIGH);

digitalWrite(Output\_8, HIGH);

int cont = 1;

float resp = 1;

float resp2 = 1;

for(int AA = 0; AA < 500; AA++){

resp = 3 + sin(resp)/cos(resp\*resp/2) \* sqrt(sqrt(resp\*resp));

resp = resp \* 0.5;

resp2 = sqrt(AA);

yield();

}

}

Além deste código, foram desenvolvidos outros códigos para o funcionamento de outros modos tratamento de transmissão de sinais, incluindo, também, diferentes formas de economia de energia. Novamente, utilizou-se do pressionar do botão para a alteração entre as funções relativas a cada estado de funcionamento do ESP.

Nesse momento, foram realizadas medidas de consumo de corrente, para isso foram utilizados três multímetros de marcas e modelos diferentes, sendo suas aferições relatadas em maior valor lido e menor valor lido. A tensão foi medida em um resistor shunt de 1 Ω em série com o protótipo de teste de consumo. Após a coleta das seis medidas, foi calculada a média aritmética para se chegar ao resultado final de consumo. A tabela X apresenta, sucintamente, os valores obtidos, todos os valores medidos, incluindo todas as características dos modos de funcionamento estão dispostos na tabela no apêndice A.

Após os valores aferidos, foi feita uma análise de autonomia em relação ao modelo de bateria e aos modos de funcionamento do ESP8266. O padrão de comportamento do *chip* se apresenta de maneira análoga em todos os casos de análise, pois o funcionamento se baseia em um tempo com o processador ligado, um período com o transmissor de sinais ligado, e um período de *Sleep*. Esse padrão foi definido dessa forma porque apresenta melhor rendimento.

Inicialmente, após a coleta dos dados de consumo, foram criadas equações matemáticas que estimam, de modo aproximado, o gasto energético da placa em diferentes modos de funcionamento com a carga energética das baterias selecionadas, calculando, dessa forma, os valores de autonomia do protótipo. O Gráfico 1 mostra a relação dos modos de operação selecionados e sua autonomia, tomando como exemplo uma bateria de carga energética igual a 650 mAh.

Cada modo de operação do Chip possui vantagens e desvantagens, que serão aprimoradas posteriormente junto à codificação final do projeto. Vale salientar que a principal diferença entre os modos está na velocidade dos rastreamentos realizados, sendo que quanto menor o tempo de repetição, mais precisa será a localização. Caso o tempo fosse zero, o monitoramento seria considerado em tempo real, o que se torna possível, mas não viável, pelo alto custo de se manter este sistema funcionando por um longo período.

A partir das análises preliminares do gráfico, é possível destacar a diferença entre a melhor curva e a pior curva de consumo, destacando, dessa forma, como a pouca diferença entre algumas variáveis altera o resultado final. De certo modo, é impossível que se tenha uma precisão de 100% no valor de autonomia, isso se deve a dois principais motivos.

O primeiro fator se deve a curva de descarga da bateria, a qual não é inteiramente linear, apresentando em seu início e fim curvas exponenciais, dificultando, dessa maneira, os cálculos precisos com relação a descarga da bateria. Essas curvas estão diretamente ligadas aos tipos de cada bateria e os diferentes modelos produzidos por cada empresa da área.

O segundo fator direciona-se ao fato que os próprios chips possuem variantes internos, estes que por sua vez variam naturalmente, além de serem susceptíveis a variações externas como temperatura. Logo, com variações externas, soma-se as variações resultantes do código fonte executado pelo ESP8266, de suas contas e de suas variações com o decorrer do tempo. Ademais, soma-se as variações de corrente consumida pelo chip em consequência das variações de tensão fornecida pela bateria, visto que esta, ao passar do tempo, tende a diminuir.

Após os primeiros testes experimentais, foi projetado e montado um circuito capaz de analisar em tempo real os valores de tensão da bateria em decorrer do funcionamento dos *chips* ESPs em diferentes modelos de bateria. O experimento foi realizado para que se possa aferir divergências entre os valores obtidos teoricamente e os obtidos no funcionamento dos circuitos.

O funcionamento deste circuito, rotulado de “Circuito datalogger de tensão”, tem como função ler, a cada minuto, o valor de tensão de cada uma das baterias dispostas no circuito que alimentam determinado ESP e armazenar os valores em um arquivo do tipo ‘.svc’ para a análise em algum programa.

Inicialmente foram gravados os mesmos códigos com suas adaptações tanto para o ESP32 quando para o ESP8266, no qual eram responsáveis por operar o protótipo de modo a seguir o padrão de consumo/funcionamento referente a tabela X.

O circuito teve como centro o Arduino Mega, a qual serviu de ponte entre a leitura de tensão de cada bateria ligada à um diferente ESP e o armazenamento dessas leituras em um cartão de memória. Para a leitura e o registro serem feitos com sucesso e uma coerente exatidão, foram utilizados dois módulos ao Arduino Mega, o módulo ‘Cartão SD’ e o módulo ‘RTC – *Real Time Clock’*.

Cada um dos módulos teve suas funções bem definidas. O ‘Cartão SD’ foi utilizado para armazenar todos os dados colhidos das leituras, contendo a data e hora da medida, que foram extraídos do RTC. As gravações foram feitas em um arquivo de nome “TENSOEX.SVC”, onde ‘X’ representa o número do teste. Como por exemplo, o teste número 3 teve nome igual a: “TENSOE3.SVC”.

Dentro desse arquivo, foram armazenados os dados separados por um caractere indicador, nos primeiros testes, foi escolhido o caractere ‘**,** ’, porém, após algumas análises, o caractere escolhido foi substituído para ‘**;** ’. Isso foi feito para que, após o termino das gravações, o arquivo possa ser aberto utilizando algum software para tabulações, o mais conhecido, e que foi utilizado é o Microsoft Excel. A partir dele, é possível abrir esse arquivo e colocar cada valor lido separado em uma célula para posteriormente trabalhar com alguma aplicação no Excel, como a construção de gráficos comparativos.

Os dados foram mantidos em uma hierarquização de ordem, em outras palavras, todos os dados foram sempre armazenados na mesma ordem, com a finalidade de se manter um padrão e facilitar a aplicabilidade em diversos programas de tabulação. A ordem escolhida para a escrita no cartão de memória foi: data, hora, valor do sensor 1, valor do sensor 2, erro 1, erro 2, ||; Dessa forma, os dados foram gravados como demonstrado abaixo:

|  |
| --- |
| 11.09.2018; 16:56:00; 3.51; 3.57; 3; 2; ||; |
| 11.09.2018; 16:57:00; 3.51; 3.56; 3; 2; ||; |
| 11.09.2018; 16:58:00; 3.50; 3.55; 4; 2; ||; |
| 11.09.2018; 16:59:00; 3.50; 3.54; 4; 2; ||; |
| 11.09.2018; 17:00:00; 3.50; 3.53; 4; 2; ||; |

Logo, transcrevendo as informações, temos:

Data: 11.09.2018

Horas: 16:56:00

Valor do sensor 1: 3.51

Valor do sensor 2: 3.57

Erro 1: 4

Erro 2: 2

|| : Caractere sinalizador de fim de linha.

A cada nova leitura de tensão nas baterias, é escrito uma nova linha, logo abaixo do anterior, com a exata mesma sintaxe. Todavia, com seus dados atualizados com os novos valores referentes a leitura.

Todo o código fonte pode ser observado abaixo:

|  |
| --- |
| #include <DS3231.h>  #include <SPI.h>  #include <SD.h>  #define chip\_select 4  int pino\_sensor\_de\_tensao\_ESP32 = A0;  float valor\_do\_sensor\_de\_tensao\_ESP32 = 0;  int pino\_sensor\_de\_tensao\_ESP8266 = A8;  float valor\_do\_sensor\_de\_tensao\_ESP8266 = 0;  int pino\_sensor\_controle\_ESP32 = A1;  float valor\_do\_sensor\_de\_controle\_ESP32 = 0;  int pino\_sensor\_controle\_ESP8266 = A9;  float valor\_do\_sensor\_de\_controle\_ESP8266 = 0;  unsigned int minuto\_antigo = 0;  unsigned int minuto\_atual = 0;  unsigned int marcador\_tempo\_1 = 0;  unsigned int marcador\_tempo\_2 = 0;  bool flag = false;  bool flag2 = false;  bool flag\_controle\_marcador\_ON = false;  bool flag\_controle\_marcador\_OFF = false;  bool flag\_ligado = false;  String texto\_marcador = "string marcador";  String dados = "string dados";  String condicao\_ = "condicao";  unsigned char erro\_marcador\_1 = 0;  unsigned char erro\_marcador\_2 = 0;  void leitura();  void marcador(String marcador);  void gravar\_dados\_cartao();  DS3231 rtc(SDA, SCL);  Time t;  File datalogger;  void setup() {    Serial.begin(115200);  rtc.begin();  while (!Serial) {;}    if (!SD.begin(chip\_select)) {  Serial.println("Erro ao ler cartao de memoria");  return;  }    }  void loop() {  t = rtc.getTime();  minuto\_atual = t.min;    if ( digitalRead(10) == HIGH) {  condicao\_ = "LIGADO";  marcador(condicao\_);  } else {  condicao\_ = "DESLIGADO";  marcador(condicao\_);  }    if ( minuto\_antigo != minuto\_atual ){  minuto\_antigo = minuto\_atual;  leitura();  dados = texto\_marcador + rtc.getDateStr() + ";" + rtc.getTimeStr() + ";" + valor\_do\_sensor\_de\_tensao\_ESP8266 + ";" + valor\_do\_sensor\_de\_tensao\_ESP8266 + ";" + erro\_marcador\_1 + ";" + erro\_marcador\_2 + ";\*";  gravar\_dados\_cartao();  }  valor\_do\_sensor\_de\_controle\_ESP32 = analogRead(pino\_sensor\_controle\_ESP32);  valor\_do\_sensor\_de\_controle\_ESP32 =(valor\_do\_sensor\_de\_controle\_ESP32 \* 3.75 ) / 843;  if(valor\_do\_sensor\_de\_controle\_ESP32 > 2){  flag = true;  }  if( (valor\_do\_sensor\_de\_controle\_ESP32 < 1) && (flag == true) ){  erro\_marcador\_1++;  Serial.println("Erro no marcador 2 (ESP32)");  Serial.println(String (erro\_marcador\_1));  Serial.println(" ");  flag = false;  }  valor\_do\_sensor\_de\_controle\_ESP8266 = analogRead(pino\_sensor\_controle\_ESP8266);  valor\_do\_sensor\_de\_controle\_ESP8266 =(valor\_do\_sensor\_de\_controle\_ESP8266 \* 3.75 ) / 843;  if(valor\_do\_sensor\_de\_controle\_ESP8266 > 2){  flag2 = true;  }  if( (flag2 == true) && (valor\_do\_sensor\_de\_controle\_ESP8266 < 1) ) {  erro\_marcador\_2++;  Serial.println("Erro no marcador 1 (ESP8266)");  Serial.println(String (erro\_marcador\_2));  Serial.println(" ");  flag2 = false;  }    }  void leitura() {  int x = 0;  float valor\_do\_sensor\_de\_tensao\_ESP32\_soma = 0;  float valor\_do\_sensor\_de\_tensao\_ESP8266\_soma2 = 0;    while ( x < 10 ) {  valor\_do\_sensor\_de\_tensao\_ESP32 = analogRead(pino\_sensor\_de\_tensao\_ESP32);  valor\_do\_sensor\_de\_tensao\_ESP32 = (valor\_do\_sensor\_de\_tensao\_ESP32 \* 3.75 ) / 843;  valor\_do\_sensor\_de\_tensao\_ESP32\_soma = valor\_do\_sensor\_de\_tensao\_ESP32 + valor\_do\_sensor\_de\_tensao\_ESP32\_soma;    valor\_do\_sensor\_de\_tensao\_ESP8266 = analogRead(pino\_sensor\_de\_tensao\_ESP8266);  valor\_do\_sensor\_de\_tensao\_ESP8266 = (valor\_do\_sensor\_de\_tensao\_ESP8266 \* 3.75 ) / 843;  valor\_do\_sensor\_de\_tensao\_ESP8266\_soma2 = valor\_do\_sensor\_de\_tensao\_ESP8266 + valor\_do\_sensor\_de\_tensao\_ESP8266\_soma2;    x++;  }  valor\_do\_sensor\_de\_tensao\_ESP32 = valor\_do\_sensor\_de\_tensao\_ESP32\_soma / 10;  valor\_do\_sensor\_de\_tensao\_ESP8266 = valor\_do\_sensor\_de\_tensao\_ESP8266\_soma2 / 10;  }  void marcador(String controle){  if ( (controle == "LIGADO") && (flag\_controle\_marcador\_ON == false) && (flag\_ligado == false) ) {  flag\_controle\_marcador\_ON = true;  texto\_marcador = "\n \n \n \n \n \n \n \n \n \n Data;Hora;valor\_do\_sensor\_de\_tensao\_ESP32;valor\_do\_sensor\_de\_tensao\_ESP8266;erro marcador 1 (ESP8266);erro marcador 2 (ESP32); ||; \n";  flag\_ligado = true;  Serial.println(texto\_marcador);  delay(1000);  } else {  if ( (controle == "DESLIGADO") && (flag\_controle\_marcador\_OFF == false) && (flag\_ligado == true) ) {  flag\_controle\_marcador\_OFF = true;  texto\_marcador = "\n \n \n \n \n \n \n \n \n \n ; ; ; ; ; || \n";  Serial.println(texto\_marcador);  delay(1000);  }  }  }  void gravar\_dados\_cartao() {    datalogger = SD.open("2x6.svc", FILE\_WRITE);  if ( datalogger ) {  Serial.println("Atualizando datalogger");  Serial.print(dados);  Serial.println("\n");  unsigned int tamanho\_char = 100;  char vetor\_dados [tamanho\_char] = {""};  dados.toCharArray(vetor\_dados, tamanho\_char);  for ( int i = 0; i <= tamanho\_char; i++){  if( vetor\_dados[i] == '\n' ){  datalogger.println(" ");  }  if ( vetor\_dados[i] == '\*' ) {  break;  }  datalogger.write(vetor\_dados[i]);  }  datalogger.println(" ||;");  datalogger.close();  } else {  Serial.println("Erro ao abrir datalogger");  }  Serial.println("Atualizado");  texto\_marcador = "";  dados = "";  } |

Após a leitura dos valores de tensão, foram montados gráficos afim de se analisar o comportamento prático dos primeiros protótipos.

**Como se espera que o gado mantenha uma movimentação suave e lenta, foi considerado o período de duas horas para cada monitoramento, pois é o que demonstra a maior autonomia, com aproximadas 4.000 horas, equivalentes a cinco meses e dez dias. Todavia, esse valor pode ser aumentado caso a bateria escolhida pelo fazendeiro seja de maior porte. Exemplificando: caso a opção seja por uma bateria de 1000 mAh, o valor de autonomia será de oito meses, aproximadamente.**

# Resultados

A partir da plotagem dos gráficos referentes aos valores de tensão armazenados no cartão SD, foram extraídas as conclusões iniciais. No gráfico presente na figura X, percebe-se uma relativa aproximação com os valores teóricos, cuja a divergência foi o fator da descarga da bateria acontecer de forma exponencial, de maneira a impossibilitar os cálculos teóricos exatos.

Com base na análise do gráfico 2 de consumo – figura X, percebe-se a presença de várias variações bruscas de tensão, esse fato se deve ao modo de funcionamento em que as placas ESPs foram pré-programadas, pois com a troca do modo de operação, ocorre a troca de intensidade do consumo de corrente elétrica.

Isso é provado a partir da comparação entre o gráfico 3 de consumo – figura X, no qual os ESPs não estão programados para realizar a troca de rotina de operações. Desse modo permanecem constantes sem alterar o modo de funcionamento. Por esse motivo não é perceptível nenhuma mudança abrupta no gráfico.

Devido a tal fato, a vida útil da bateria é fortemente danificada, pois não são preparadas para sofrerem esse tipo de variação de corrente elétrica. Tal fato é percebível em todos os demais testes.

# Discussão

A partir dos resultados previamente encontrados, é visto que a característica autonomia e fortemente atingida pelas características de comportamento dos *chips* ESPs, dessa forma, faz-se necessário o estudo de soluções alternativas para eventualmente corrigir o problema e contornar a baixa vida útil da bateria.

Em primeiro momento, está sendo visado a utilização do Microcontrolador ATtiny13, como operador central dos comandos, em substituição do ESP, pelo fato de não consumir grandes valores de corrente elétrica. Como transmissor de sinais, está sendo utilizado o módulo NRF24L01, pelos mesmos motivos anunciados ao ATtiny.

Com essas novas tecnologias, é prevista uma forma de recuperar os altos valores de autonomia demonstrados nos resultados dos testes teóricos, visto que seu comportamento será mais semelhante aos teóricos do que propriamente os dos ESPs.

|  |
| --- |
|  |

# Matéria encaminhada para publicação

Asdsddd

# Perspectiva de continuidade do trabalho

Asdasdddfg

Gfgghjfdgfddd

# Agradecimentos

Asdfsffdgf

Gfgfgddd

# Apêndice A – Tabela de modelos de baterias

Sdsdff

Fggfgfd

Ddfgdgdfg

# Apêndice B – Tabela de valores de consumo completa

Asdffggdf

Gfgfgsdfg

1. O projeto completo constitui-se de duas partes separadas, que serão desenvolvidas em paralelo. A primeira parte é a que constitui o código fonte de todo o sistema de monitoramento e gerenciamento. A segunda parte constitui-se do *Hardware* envolvido nos nós da rede, nos estudos referentes aos modelos de bateria e nos diferentes modos de operação envolvidos no estudo do consumo do circuito. [↑](#footnote-ref-1)
2. O código fonte resume-se nas operações pré-programadas que o circuito eletrônico fará atuando no meio físico. [↑](#footnote-ref-2)