**Carta de apresentação**

Santo André,  28 de fevereiro de 2018.

À Ilustríssima Pró-Reitora de Pesquisa,

Profª. Dra. Marcela Sorelli Carneiro Ramos

Encaminho o relatório do aluno Miguel dos Reis referente ao projeto de pesquisa junto ao programa de Iniciação Científica na modalidade PDPD no edital ***(número do edital).***

***(descrever o desempenho do aluno durante a pesquisa)***.

Nome e assinatura do Orientador:

## 1. Resumo das Atividades

As atividades realizadas no período se dividem em: estudo dos hardwares inclusos no projeto (sensores e módulo de radiofrequência), breve estudo sobre radiofrequência, familiarização com a plataforma de prototipagem Arduino, prática e estudo de programação e de algoritmos, estudo de projetos que abordassem situações similares envolvendo arduino e transmissão de dados por radiofrequência, estudo e aplicação de métodos de tratamento de dados e de protocolos de comunicação direcionados a projetos microcontrolados, e elaboração do código para o Arduino para o envio e o recebimento de dados.

## 2. Descrição dos Objetivos e Metas

O objetivo final do projeto é definir os componentes, os pré-requisitos e principalmente o algoritmo envolvido, para um sistema que, baseado na plataforma de prototipagem arduino, receba, por radiofrequência, os dados gerados pelos sensores durante a instrumentação de teste estático dos motores dos foguetes da UFABC Rocket Design a uma distância de pelo menos 500 metros.

A transmissão dos dados do teste estático por radiofrequência elimina a necessidade de se ter um computador conectado por comunicação serial diretamente ao Arduino da plataforma de testes, de forma a diminuir os danos causados por uma possível explosão do motor, tornando o teste mais seguro.

Para atingir este objetivo, é necessário obter um entendimento prático do funcionamento do módulo de radiofrequência junto ao Arudino e a forma como ele transmite informações, para estabelecer a comunicação entre os módulos e o processamento de dados da melhor maneira possível.

O projeto também teve como meta secundária aprimorar a capacidade de desenvolvimento em arduino e obter familiaridade com a plataforma e sua linguagem de programação, além de compreender a transmissão por radiofrequência e protocolos de comunicação.

## 3. Metodologia

Os métodos utilizados para a realização do projeto foram: análise dos componentes envolvidos e dos dados por eles gerados, estudo do funcionamento prático do módulo de radiofrequência, revisão bibliográfica buscando projetos com abordagens que possam ser utilizadas, análise dos dados gerados pelo sistema a fim de definir os parâmetros para o protocolo de comunicação entre os módulos transmissor e receptor e também os métodos de tratamento de dados necessário que podem ser aplicados, e por fim programação do algoritmo de transmissão e recepção para a plataforma arduino.

### 3.1. Análise dos Componentes

A análise de cada componente foi feita a partir de informações obtidas de diversas fontes, em especial as especificações disponibilizadas pelos fabricantes [7, 8, 9], quando encontradas, e os relatórios internos da UFABC Rocket Design, referentes aos componentes eletrônicos usados pela equipe e ao teste estático. Também foram consultados diversos fóruns online especializados em assuntos relacionados a plataforma Arduino [6, 10].

### 3.2. Estudo da Modulação

O estudo tanto do módulo de radiofrequência LoRa SX1278 como a modulação utilizada por ele foram feitas principalmente a partir das informações disponibilizadas pela fabricante Semtech [7] e por artigos acadêmicos que tratam sobre a modulação [1, 2, 3, 4, 5], apesar dos artigos terem maior foco em aspectos de projetos para a “Internet das coisas” que foge do escopo do projeto. Aqui também foram consultados fóruns online especializados na plataforma Arduino com a intenção de obter informações práticas e direcionadas ao uso do módulo em projetos com o Arduino.

### 3.3. Revisão Bibliográfica

Além da revisão bibliográfica direcionada a compreensão do funcionamento de cada componente, também foi feita uma busca por material sobre projetos que lidassem com situações similares e projetos que utilizassem o módulo LoRa. Essa busca foi realizadas em fóruns especializados em assuntos relacionados a plataforma Arduino [15, 16, 17, 19, 20,

21, 22], em projetos disponibilizados no Github [11, 13, 14 , 18], e em sites dedicados expor tais projetos [23].

### 3.4. Análise dos Dados Gerados

A partir do entendimento do funcionamento dos componentes do sistema foi necessário entender qual é a informação gerada pelos sensores e qual o seu formato, a fim de estabelecer que tipo de processamento se faz necessário.

A compreensão e definição de quais são os dados envolvidos na comunicação entre transmissor e receptor é importante para a configuração dos comandos necessários para a inicialização do módulo RF, que estão descritos na documentação da biblioteca de comandos utilizada.

Além disso, a compreensão dos dados e da informação envolvida é necessária para a parametrização do protocolo de comunicação.

### 3.5. Definição de Parâmetros do Protocolo de Comunicação

A partir da análise dos dados gerados, foram definidos quais parâmetros deveriam ser estabelecidos para que os dados enviados fossem compreendidos e reprocessados pelo receptor, isto é, qual o número de informações (dados) transmitidas, o tamanho em bits de cada uma delas e a ordem em que a transmissão é feita. A partir desses parametros, é programado diretamente no arduino o protocolo, ou seja, os parâmetros que cada transmissão deve seguir para que a comunicação aconteça.

### 3.6. Método de Tratamento de Dados

Antes da programação do arduino, foram estudados possiveis métodos de tratameno de dados que poderiam ser aplicados no projeto. Foram pesquisados métodos já existentes que pudessem tornar a comunicação mais rápida, mais estruturada ou mais precisa. Esses métodos foram encontrados em fóruns [19, 20, 21, 22] e em documentações de projetos no GitHub[25] e outros sites [24, 26].

### 3.7. Elaboração do Algoritmo

A programação do algoritmo foi realizada no ambiente de desenvolvimento e na linguagem de programação nativos do Arduino. Ela foi feita utilizando a biblioteca de comandos do módulo LoRa, criada e disponibilizada na internet para projetos sem fins lucrativos por Stuart Robinson e já previamente utilizada pela UFABC Rocket Design. Também foi utilizada a biblioteca que engloba funções de controle do tempo [27].

O objetivo do algoritmo foi envolver a configuração necessária para a inicialização do módulo e da comunicação entre transmissor e receptor, o protocolo de comunicação e o processamento dos dados.

## 4. Resultados e Discussões

### 4.1. Sobre os Componentes

Os hardwares inclusos no projeto são a plataforma de prototipagem Arduino Uuno, que contém o microcontrolador ATmega328, e o módulo de radiofrequência LoRa SX1278, fabricado pela Semtech, que são os componetes que realizam o processamento, a transmissão e a recepção, além de uma célula de carga, que contém um transdutor de força, e um termopar, que são os sensores que gerarão a informação a ser transmitida.

Cada sensor deve ser conectado a uma das portas de comunicação analógica do Arduino e ao pino GND (referência ou comum). O módulo SX1278 também deve ser conectado ao Arduino, a tabela abaixo indica uma forma de fazê-lo.

| Pino LoRa | Nome do Pino | Direção do Pino | Descrição |
|---|---|---|---|
| 1 | M0 | Entrada | Utilizado para configurar modo de operação |
| 2 | M1 | Entrada | Utilizado para configurar modo de operação |
| 3 | RXD | Entrada | Receptor UART de dados, conectado ao TX do microcontrolador |
| 4 | TXD | Saída | Transmissor UART de dados, conectado ao RX do microcontrolador |
| 5 | AUX | Saída | Indicação do estado do módulo |
| 6 | VCC | Entrada | Fonte de energia 3,3V ou 5V |
| 7 | GND | Entrada | Comum, ground |

Figura 4.1.1: configuração da pinagem do módulo SX1278 em Arduino.

Durante o teste estático, o transdutor presente na célula de carga é acionado e produzirá uma tensão de 0 a 5V proporcional a força exercida sobre ele pelo motor. Esse

sinal é amplificado e convertido pelo arduino para um valor numérico de 10 bits , ou seja, um valor entre 0 e 1023. O mesmo ocorre para o termopar ou para qualquer outro sensor que seja adicionado.

### 4.2. Sobre o Módulo RF e Sua Modulação

O módulo LoRa SX1278 é um dispositivo de radiofrequência criado para transmitir e receber informações através de distâncias que vão, segundo o fabricante, até 15Km, com alta imunidade a interferências e com baixíssimo consumo de energia. O bitrate do módulo é de no máximo 37,5Kbps e depende de algumas das configurações escolhidas. A fabricante Semtech disponibiliza um software chamado *LoRa Modem Calculator Tool*, que calcula a perfomance de acordo com o fator de espalhamento (tradução livre de espreading factor) , a largura de banda e o fator de correção de erro (número de redundâncias) configurados.

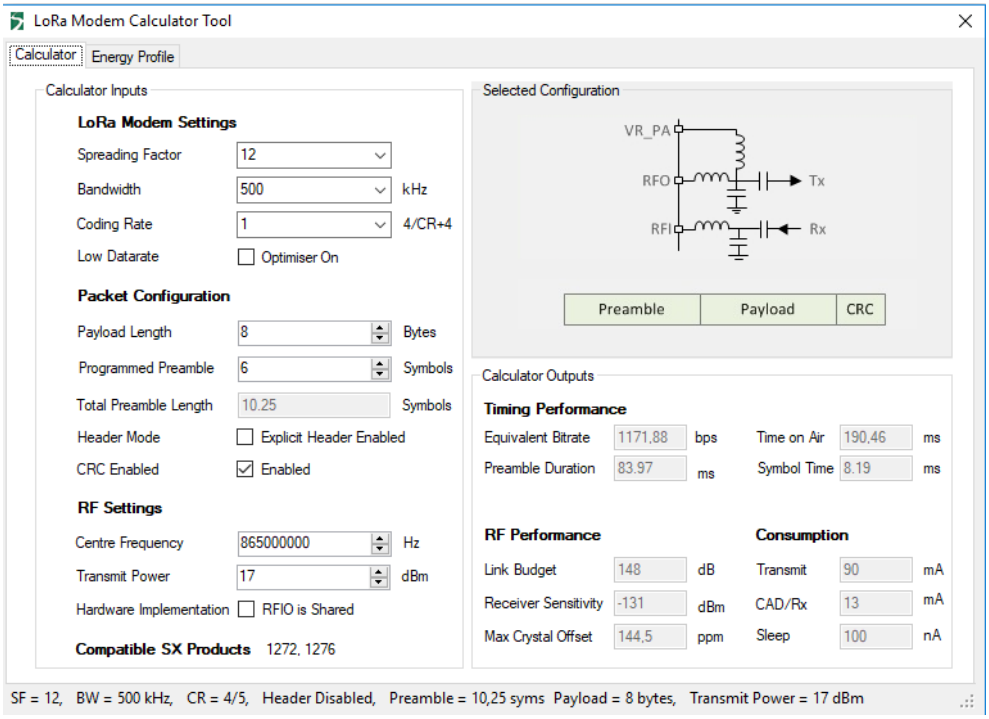


Figura 4.2.1: software da fabricante Semtech que calcula parâmetros do módulo LoRa.

A modulação realizada pelo SX1278 é chamada de LoRa e também é criada pela Semtech. Ela é do tipo FSK, mas atinge um alcance muito maior do que uma modulação FSK pura se utilizando uma técnica chamada de Chirp Spread Speactrum (CSS).

Chirp Spread Speactrum, que pode ser traduzido livremente para espectro de espalhamento de sinal, é uma forma de codificação em que o sinal é espalhado através de toda a banda disponível enquanto sua frequência aumenta ou diminui através do tempo, obedecendo alguma expressão matemática. O uso de todo o espaço disponível na banda confere ao sinal alta resistência à interferência e à propagação do sinal por mais de um caminho.

Para a programação do módulo na plataforma Arduino existem diversas bibliotecas de comandos disponíveis criadas por usuários do módulo. Para o projeto, será utilizada a versão criada e disponibilizada por Stuart Robinson, por ser simples se comparada às outras opções, mas também por já ser a escolhida pela UFABC Rocket Desgin.

Esse módulo e método de modulação de sinal são ótimos para o objetivo do projeto pois, das opções viáveis, é a única que satisfaz a necessidade de longo alcance e baixo consumo de energia, além de ser simples de programar. Entretanto, para apresentar tais vantagens, o LoRa apresenta certa limitação com relação ao bitrate, que fica limitado a 37,5Kbps.

### 4.3. Revisão Bibliográfica

Como o Arduino é a plataforma de prototipagem mais popular do mundo, sua comunidade de usuários é muito vasta e portanto o conteúdo disponível online sobre projetos de escopo diversos é muito vasto. Foram encontrados muitos tópicos abertos em fóruns com questões levantadas por usuários sobre o uso do LoRa com o Arduino e também do Arduino com outras opções de transmissão por radiofrequência.

Apesar de nem todos os projetos encontrados apresentarem conceitos que pudessem ser diretamente aplicados, eles são úteis a medida que dão ideias e direcionamentos para o projeto.

Foram encontrados projetos que também precisaram lidar com a limitação do bitrate, e projetos que lidaram com a questão de sincronização, dos quais alguns

apresentaram soluções adaptáveis ou levaram a elas. As soluções que foram incluídas no projeto serão abordadas nos próximos tópicos.

Também foram encontrados conteúdos sobre projetos mais robustos que utilizam o LoRa de um modo mais complexos, desenvolvendo novas bibliotecas e placas com o módulo embarcado. Esses projetos são pouco adaptáveis ao escopo deste, porém, o conteúdo dele serve como aprendizado e demonstra o alcance do que é possível se atingir utilizando o LoRa.

### 4.4. Sobre os Dados Gerados

Mesmo as informações geradas sendo simples, é necessário entendê-las com cuidado. Tanto o sinal gerado pelo transdutor da célula de carga como pelo trandutor são tensões de 0 a 5V que serão enviadas ao Arduino. O comando *analogRead()* do Arduino irá então converter ambos os sinais em dados digitais de 10 bits, ou seja, um número de 0 a 1023, proporcional à intensidade da tensão recebida na porta de entrada.

No Arduino, apesar de cada leitura dos sensores ser um dado de 10 bits, não é definido na linguagem de programação nenhum tipo de dado cujo tamanho dentro da memória seja de 10 bits, existem apenas de oito, 16 ou 32, se fazendo necessário algum processamento ou causando um possível desperdício de espaço no pacote a ser transmitido.

É importante ressaltar que, para o objetivo de um teste estático, é necessário saber quanto tempo se passou entre uma medição feita pelo sensor e a medição seguinte, de forma que esta é outra informação que deverá ser transmitida.

### 4.5. Sobre o Protocolo de Comunicação

No escopo desse projeto, o protocolo de comunicação que se faz necessário é simples. É necessário apenas estabelecer na trasmissão e na recepção a mesma ordem de informações. Os tamanhos dos pacotes transmitidos são definidos no comando fornecido pela biblioteca de funções do LoRa, sendo necessário na programação apenas usar variáveis do tamanho correto, para que não hajam erros ou perda de informação.

### 4.6. Sobre Métodos de Tratamento de Dados

Alguns métodos de processamento e tratamento de dados, principalmente algoritmos e bibliotecas de compressão, foram considerados como forma de buscar otimizar o projeto, porém a maioria não foi considerada apropriada.

O primeiro tipo a ser considerado foram os métodos clássicos de compressão, que usam algoritmos pra codificar e decodificar dados, de forma que o pacote a ser armazenado e trasmitido tenha um tamanho menor e portanto ocupe menos espaço na banda, possibilitando por exemplo que sejam utilizadas redundâncias. Os métodos estudados foram Huffman coding, RLE (Run Length Encoding), LZO (Lempel-Ziv-Oberhumer), LZSS (Lempel-Ziv-Storer-Szymanski), LZW (Lempel-Ziv-Welch) e a biblioteca de compressão Zlib. Entretanto, esses métodos não foram considerados apropriados pois demandam muita memória RAM para serem aplicados e não são rápidos o suficiente para processar os dados durante a transmissão, tal como é necessário no projeto. Além disso, esses métodos são mais apropriados para comprimir dados mais estruturados, como uma imagem ou um arquivo de texto, nos quais em geral há repetições de padrões, diferentemente do simples fluxo de bits que são os dados gerados no projeto, no qual não há como prever a presença ou não de padrões a serem identificados.

Também foi considerada o uso da biblioteca de compressão própria para o uso em Arduino chamada *Arduino Hscompression* e outras derivadas desta. Entretanto, tal como os primeiros, esses métodos foram considerados complexos demais para os propósitos do projeto e também são métodos inapropriados para compressão durante a transmissão.

Outro método cuja utilização foi considerada e descartada foi a utilização da versão do Google Protocol Buffer para microcontroladores chamada NanoPB. Google Protocol Buffers é um método de serialização e reestruturação de dados que independe da plataforma e da linguagem utilizadas. Ele possibilita a geração de um código que define como os dados serão estruturados, permitindo que dois dispositivos se comuniquem se baseando nessa estrutura. O NanoPB serviria, portanto, como processamento e também atuaria como protocolo de comunicação, mas foi descartado pois exigiria espaço demais na memória do Arduino e aumentaria desnecessariamente a complexidade do projeto, já que, nesse caso, é possível cumprir o mesmo propósito através de programação simples.

Por fim, o método de processamento que foi adotado foi o *Delta Encoding*, no qual programa-se o transmissor para enviar apenas o valor da variação (delta) entre uma leitura e outra, possibilitando o uso de pacotes menores no envio, no caso, utilizando apenas oito bits por *delta.* A ideia de usar Delta Encoding veio de um dos tópicos de fóruns mencionados anteriormente, nele, um dos usuários reportava uma situação similar de limitação no tamanho do pacote a ser enviado e um outro usuário observou que a varição por a cada medição era mínima e sugeriu esse método.

## 4.7. Algoritmo

### 4.7.1. Bibliotecas Utilizadas

Foram três as bibliotecas adicionais utilizadas. A já mencionada bibliotecas de funções do LoRa *LoRa_library*, que contém os comandos para programar o módulo, a *Time Library*, que adiciona os comandos para manter a contagem do tempo, e a SPI, que deve ser incluída para que o Arduino consiga se comunicar com dispositivos periféricos, como é o caso do módulo.

### 4.7.2. Código

No código, o primeiro passo tanto no programa de transmissão (TX) como o de recepção (RX) é incluir as bibliotecas e declarar as variáveis que serão utilizadas no corpo do programa. No programa de transmissão, as variáveis são utilizadas para armazenar duas leituras, a nova e a anterior, e o delta entre as leituras. Além disso, “minuto” e “segundo” são usadas para armazenar as leituras de tempo.

Em seguida é necessário definir os parâmetros para a configuração e inicialização do LoRa. São configurados um *spreading factor* de seis, o menor possível para evitar que haja efeitos no bitrate, não são utilizadas redundâncias, nem otimização de dados, e a largura de banda é definida como 500 KHz. Segundo o LoRa calculator, isso confere ao LoRa um bitrate de 37,5Kbps, o máximo possível. Essas configurações podem facilmente ser alteradas conforme o uso e a necessidade.

No código de transmissão, o módulo é configurado para transmitir a informação contida no buffer, definido como lora_TXBuff, sem um destino específico, realizando uma transmissão *broadcast*. O tamanho do pacote é definido como um ou dois bytes, conforme

a necessidade durante o código. As demais configurações são definidas conforme as recomendações na documentação da biblioteca.

O código de transmissão é programado para transmitir o minuto e segundo atuais, seguidos das leituras iniciais. Por fim, em um laço, são enviados o minuto e o segundo atuais, seguidos das variações nas leituras até que o programa seja encerrado.

No laço em que são enviadas as variações, o bit mais significativo do byte é utilizado para informar se a variação é negativa ou positiva.

O código do programa de recepção é configurado para receber na sequência em que a transmissão realizada. Portanto, as primeiras informações a serem recebidas são o tempo e as primerias leituras, seguidos por nova leitura de tempo e a variações das medições. No laço em que são recebidas as variações, é realizado o processamento que define se é uma variação negativa ou positiva.

TX

```
#include <SPI.h> // Inclui a biblioteca SPI
#include <Lora_Library.h> // Inclui as variáveis, constantes e funções da bilbioteca
#include <TimeLib.h> //Inclui os comando  que mantém a contagem de tempo

int CelCarga = A0;  //Entrada sinal da célula de carga
int Termo = A1;
int auxC = 0; //Leitura anterior da célula de carga
int auxT = 0; //Leitura anterior do termopar
int leitura = 0; //Nova leitura
byte Delta = 0; //Variação entre leituras
byte minuto = 0;
byte segundo = 0;

void setup()
{
  Serial.begin(9600);
  pinMode(CelCarga, INPUT);
  pintMode(Termo, INPUT);
  Program_Setup();
}

void loop()
{
  lora_ResetDev();      //Reinicia o dispositivo
  lora_Setup();       //Do the initial LoRa Setup
  lora_SetFreq(434.400);    //Define a frequÊncia
  lora_Tone(1000, 1000, 5);
```

Figura 4.7.2.1: Código de transmissão

TX

```
void loop()
{
  lora_ResetDev();      //Reinicia o dispositivo
  lora_Setup();       //Do the initial LoRa Setup
  lora_SetFreq(434.400);    //Define a frequÊncia
  lora_Tone(1000, 1000, 5);
  lora_SetModem(lora_BW500, lora_SF6, lora_CR4_5, lora_Explicit, lora_LowDoptOFF);
  lora_PrintModem();
  Serial.println();

  minuto = minute();
  *lora_TXBuff = minuto;
  lora_Send(lora_TXStart, lora_TXEnd, 2, 255, 1, 10, 2);
  segundo = second();
  *lora_TXBuff = segundo;
  lora_Send(lora_TXStart, lora_TXEnd, 2, 255, 1, 10, 2);

  leitura = analogRead(CelCarga);
  *lora_TXBuff = leitura; //Tentar acrescentar ponteiros
  auxC = leitura;
  lora_Send(lora_TXStart, lora_TXEnd, 2, 255, 1, 10, 2); //Checar se der


  leitura = analogRead(Termo);
  *lora_TXBuff = leitura;
  auxT = leitura;
  lora_Send(lora_TXStart, lora_TXEnd, 32, 255, 1, 10, 2);
```

Figura 4.7.2.2: Código de transmissão

TX

```
leitura = analogRead(Termo);
*lora_TXBuff = leitura;
auxT = leitura;
lora_Send(lora_TXStart, lora_TXEnd, 32, 255, 1, 10, 2);

while(1)
{
  minuto = minute();
  *lora_TXBuff = minuto;
  lora_Send(lora_TXStart, lora_TXEnd, 2, 255, 1, 10, 2);

  segundo = second();
  *lora_TXBuff = segundo;
  lora_Send(lora_TXStart, lora_TXEnd, 2, 255, 1, 10, 2);

  leitura = analogRead(CelCarga);
  if((leitura - auxC)>=0){
    Delta = leitura - auxC;
    *lora_TXBuff = Delta;
  }
  else{
    Delta = leitura - auxC + 128;
    *lora_TXBuff = Delta;
  }
  auxC = leitura;
  lora_Send(lora_TXStart, lora_TXEnd, 2, 255, 1, 10, 2);
```

Figura 4.7.2.3: Código de transmissão

TX

```
      Delta = leitura - auxC;
      *lora_TXBuff = Delta;
    }
    else{
      Delta = leitura - auxC + 128;
      *lora_TXBuff = Delta;
    }
    auxC = leitura;
    lora_Send(lora_TXStart, lora_TXEnd, 2, 255, 1, 10, 2);

    leitura = analogRead(Termo);
    if((leitura - auxT)>=0){
      Delta = leitura - auxT;
      *lora_TXBuff = Delta;
    }
    else{
      Delta = leitura - auxT + 128;
      *lora_TXBuff = Delta;
    }
    auxT = leitura;
    lora_Send(lora_TXStart, lora_TXEnd, 2, 255, 1, 10, 2);
  }
}
```

Figura 4.7.2.4: Código de transmissão

RX §

```
#include <SPI.h>                          //include the SPI library:
#include <Lora_Library.h>                 //include the LoRa Library file of constants

int CelCarga;
int auxC;
int Termo;
int auxT;
byte Delta;
byte minuto;
byte segundo;

void setup()
{
  Serial.begin(9600);
  Program_Setup();                        //do initial program setup
}

void loop()
{
  lora_ResetDev();      //Reinicia o dispositivo
  lora_Setup();       //Faz a configuração inicial
  lora_SetFreq(434.400);    //Define a frequência
  lora_Tone(1000, 1000, 5);             //Transmit an FM tone
  lora_SetModem(lora_BW500, lora_SF6, lora_CR4_5, lora_Explicit, lora_LowDoptOFF);
  lora_PrintModem();
  Serial.println();
```

Figura 4.7.2.5: código de recepção

RX§

```
while (1)
{
  lora_Listen(5);                    //listen for a packet, no timeout

  minuto = *lora_RXBUFF;
  Serial.print(minuto)
  segundo = *lora_RXBUFF;
  Serial.print(":");
  Serial.println(segundo);

  CelCarga  = *lora_RXBUFF;
  Serial.print("Célula de carga: ");
  Serial.println(CelCarga);
  auxC = CelCarga;

  Termo  = *lora_RXBUFF;
  Serial.print("Termopar: ");
  Serial.println(Termo);
  auxT = Termo;

  while(1){
    minuto = *lora_RXBUFF;
    Serial.print(minuto)
    segundo = *lora_RXBUFF;
    Serial.print(":");
    Serial.println(segundo);
```

Figura 4.7.2.6: código de recepção

RX §

```
Delta = *lora_RXBUFF;
if(Delta >= 128){
  Delta -= 128;
  auxC -= Delta;
  Serial.print("Célula de Carga: ");
  Serial.println(auxC);
}
else{
  auxC += Delta;
  Serial.print("Célula de Carga: ");
  Serial.println(auxC);

}

Delta = *lora_RXBUFF;
if(Delta >= 128){
  Delta -= 128;
  auxT -= Delta;
  Serial.print("Termopar: ");
  Serial.println(auxT);
}
else{
  auxT += Delta;
  Serial.print("Termopar: ");
  Serial.println(auxT);
}
```

Figura 4.7.2.7: código de recepção

## 5. Referências Bibliográficas

1. A Study of LoRa: Long Range & Low Power Networks for the Internet of Things. [S.l.: s.n.], 2016. Disponível em: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5038744/>. Acesso em: 15 nov. 2017.

2. UNDERSTANDING the Limits of LoRaWAN. [S.l.]: IEEE Comunication Magazine, 2017. Disponível em: <https://arxiv.org/pdf/1607.08011.pdf>. Acesso em: 15 nov. 2017.

3. LoRa Alliance . White Paper: A Technical Overview of Lora and Lorawan. The LoRa Alliance; San Ramon, CA, USA: 2015.

4. LORA Modulation Basics | CSS Modulation | Advantages, Properties. Disponível em: <http://www.rfwireless-world.com/Terminology/LoRa-modulation-vs-CSS-modulation.html>. Acesso em: 10 nov. 2017.

5. Spread spectrum communications using chirp signals . Disponível em: <https://www.researchgate.net/publication/3867801_Spread_spectrum_communications_using_chirp_signals>. Acesso em: 20 nov. 2017.

6. BUILDING your LoRa devices. Disponível em: <http://cpham.perso.univ-pau.fr/LORA/LoRaDevices.html>. Acesso em: 12 out. 2017.

7. SEMTECH CORPORATION. **LoRa Modulation Basics**. Disponível em: <https://www.semtech.com/uploads/documents/an1200.22.pdf>. Acesso em: 26 mar. 2019.

8. ANALOGREAD(). Disponível em: <https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/>. Acesso em: 16 out. 2017.

9. BYTE. Disponível em: <https://www.arduino.cc/reference/en/language/variables/data-types/byte/>. Acesso em: 16 abr. 2018.

10. GROMES, Jan. **Arduino Long Range Communication Tutorial ? LoRaLib Library** . Disponível em: <http://www.deviceplus.com/how-tos/arduino-guide/arduino-long-range-communication-tutorial-loralib-library/>. Acesso em: 10 dez. 2018.

11. LORACOMMON Library. 2016. Disponível em: <https://github.com/sppnk/LoRa-srnet-files/blob/master/LoRaCommon.h>. Acesso em: 12 dez. 2018.

12. MCCAULEY, Mike . **RadioHead Library** . [S.l.: s.n.], 2014. Disponível em: <http://www.airspayce.com/mikem/arduino/RadioHead/>. Acesso em: 12 dez. 2018.

13. LOW-COST LoRa IoT framework developed in the EU H2020 WAZIUP project. [S.l.: s.n.], 2016. Disponível em: <https://github.com/CongducPham/LowCostLoRaGw>. Acesso em: 23 jan. 2018

14. STM32 HAL driver for LoRa SX1278 module. [S.l.: s.n.], 2017. Disponível em: <https://github.com/wdomski/SX1278>. Acesso em: 24 jan. 2018.

15. TOPIC: USE LoRa Shield to listen RF 433MH. Disponível em: <https://forum.arduino.cc/index.php?topic=540674.0>. Acesso em: 03 mar. 2018.

16. TOPIC: SENDING GPS readings using RadioHead for LoRa Dragino Module. Disponível em: <https://forum.arduino.cc/index.php?topic=518181.0>. Acesso em: 10 mar. 2018.

17. TOPIC: LORA mesh network. Disponível em: <https://forum.arduino.cc/index.php?topic=515565.0>. Acesso em: 03 mar. 2018.

18. LOW-COST LoRa IoT framework developed in the EU H2020 WAZIUP project. Disponível em: <https://github.com/CongducPham/LowCostLoRaGw>. Acesso em: 14 dez. 2017.

19. TOPIC: Compression Libraries. Disponível em: <http://forum.arduino.cc/index.php?topic=39436.0>. Acesso em: 14 mar. 2018.

20. TOPIC: Data compression in my code. Disponível em: <http://forum.arduino.cc/index.php?topic=182810.0>. Acesso em: 20 mar. 2018.

21. ARDUINO: Lightweight compression algorithm to store data in EEPROM. Disponível em: <https://stackoverflow.com/questions/1606102/arduino-lightweight-compression-algorithm-to-store-data-in-eeprom>. Acesso em: 16 abr. 2018.

22. LOOKING for an arduino friendly LZ-String comp/decomp implementation. Disponível em: <https://www.reddit.com/r/arduino/comments/63c8w6/looking_for_an_arduino_friendly_lzstring/>. Acesso em: 17 abr. 2018.

23. VOKES, SCOTT. Uzlib - Deflate/Zlib-compatible LZ77 compression/decompression library . Disponível em: <https://spin.atomicobject.com/2013/03/14/heatshrink-embedded-data-compression/>. Acesso em: 17 abr. 2018.

24. PROTOCOL Buffers. Disponível em: <https://developers.google.com/protocol-buffers/>. Acesso em: 08 fev. 2018.

25. NANOPB - Protocol Buffers for Embedded Systems. 2014. Disponível em: <https://github.com/nanopb/nanopb>. Acesso em: 08 fev. 2018.

26. A PRACTICAL guide to protocol buffers. Disponível em: <http://www.minaandrawos.com/2014/05/27/practical-guide-protocol-buffers-protobuf-go-golang/#WhyProtocolBuffers>. Acesso em: 10 fev. 2018.

27. ARDUINO Time library. Disponível em: <https://playground.arduino.cc/code/time>. Acesso em: 10 jun. 2018.

Figura 3.1.1: <https://www.filipeflop.com/blog/primeiros-passos-lora-com-arduino/>