# Comunicação Sem Fio Entre Microcontroladores na Automação Residencial.

**Nathan Partel Balduino Oliveira**

São Carlos

2015

# Comunicação Sem Fio Entre Microcontroladores na Automação Residencial.

**Nathan Partel Balduino Oliveira**

Trabalho de conclusão de curso de Engenharia Mecatrônica apresentado à Escola de Engenharia de São Carlos como parte dos requisitos necessários à aprovação na disciplina Trabalho de Conclusão de Curso.

Orientador: Eduardo do Valle Simões

São Carlos

2015

# DEDICATÓRIA

Dedico este trabalho à minha avó, Berenice Tereza de Rossi Partel. Em meus pensamentos você estará presente, na colação de grau e sempre.

# AGRADECIMENTOS

Agradeço a todos que fizeram parte de minha jornada até aqui. Me considero extremamente afortunado por ter contado com o apoio de pessoas tão generosas, que se dedicaram intensamente para me auxiliar sem nunca pedir algo em troca. Em especial, agradeço a meus pais, meus avós e minha namorada por terem participado de minha vida durante todos estes anos, sempre fornecendo suporte e incentivo. Por vocês sou uma pessoa melhor.

**“**I am iron man, I am iron man, I am iron man, I am iron…**”**

***HM-10 module datasheet***

# RESUMO

Neste trabalho se desenvolveu um conjunto de funções de software para possibilitar a comunicação sem fio de microcontroladores por meio do padrão *Bluetooth* 4.0. As funções foram implementadas em linguagem C para a utilização com as plataformas Arduino e Raspberry Pi em conjunto com o módulo de comunicação sem fio HM-10. A implementação do conjunto de hardware e software foi realizada em um cenário de teste representativo de um ambiente doméstico para avaliação de métricas relevantes como consumo energético, alcance e simplicidade de utilização.



Palavras-chave: Comunicação Sem Fio, *Bluetooth*, Redes de Sensores Sem Fio, Domótica.



# ABSTRACT

In this project a group of software functions was developed with the intention of allowing wireless communications between microcontrolers by use of the 4.0 Bluetooth standard. The functions were implemented with the C programming language for use with the Arduino and Raspberry Pi platforms along with the HM-10 Bluetooth module. Both the hardware and software components were tested in a domestic representative scenario in order to allow for measuring of relevant data, such as power consumption, range and ease of use.



Keywords: Wireless communications, Bluetooth, Wireless sensor networks, Domotics.

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| IDR | Identificador do receptor |
| IDS | Identificador do emissor |
| CS | *Checksum* |
| ISM Bands | *industrial, scientific and medical (ISM) radio bands* |
| ITU | *International Telecommunication Union* |
| SRAM | *Static Random Access Memory* |
| EEPROM | *Electrically-Erasable Programmable Read-Only Memory* |
| GFSK | *Gaussian frequency-shift keying* |
| MAC Address | *Media Access Control Address* |

# SUMÁRIO

# 1. INTRODUÇÃO

A automação residencial, ou domótica, tem como objetivo possibilitar e beneficiar a comunicação entre dispositivos domiciliares, assim como sua interação com o usuário. Ao atingir tal objetivo a domótica permite que o usuário possa usufruir de maior conforto e segurança, ou ainda possibilitar economia de recursos por meio da redução do consumo em processos que possam ser otimizados. Exemplos de processos que são beneficiados pela automação incluem sistemas de controle de temperatura, iluminação e irrigação (KASTNER *et al*., 2005).

Com a crescente popularização de dispositivos de computação pessoal a população tem se tornado cada vez mais aberta e receptiva a novas tecnologias. Nesse sentido, a implementação de sistemas de computação que se integrem ao dia-a-dia dos consumidores se torna mais viável, aumentando a demanda por sistemas de automação residencial (MATERNAGHAN, 2010). Apesar de tal popularização, a integração simples de sensores em uma rede ainda apresenta significativo desafio. O acesso a dados de sensoriamento remoto atualmente apresenta dificuldades como a necessidade de se utilizar múltiplas linguagens de programação, plataformas e formatos de dados. Ademais, o desenvolvimento de tais redes de sensores usualmente envolve programação de baixo nível e interação com plataformas proprietárias, de modo que aplicações são desenvolvidas para sensores específicos e apresentam poucas possibilidades de adaptação e integração (ZHENG, PERRY, JULIEN, 2014).

Neste contexto, a implementação de uma rede sem fio para comunicação entre os sensores possui vantagens, tanto para instalações novas quanto para adaptação de instalações antigas. Tais vantagens incluem a versatilidade física de instalação, dado que não são necessárias adaptações para cabeamento, e a flexibilidade de alteração na rede, dado que o funcionamento da rede independe da mudança de posição de seus componentes. Dessa forma, aplicações que exijam alterações ou expansões da rede ou que possuam limitações físicas de instalação podem se beneficiar da utilização de comunicação sem fio entre os componentes do sistema (REINISCH *et al*., 2007).

Ainda segundo Reinisch *et al* (2007), redes sem fio para automação residencial possuem duas características necessárias para que um sistema seja aceito pelo mercado, sendo elas custo e autonomia. O custo das unidades que compõe a rede se torna particularmente importante devido ao grande volume em que são utilizadas, fazendo com que pequenas diferenças de custo unitário tenham um impacto considerável no custo final. A autonomia, por outro lado, tem implicações na manutenção do sistema, de modo que uma rede em que sejam necessárias trocas constantes de bateria se torna pouco viável.

Tendo em vista as demandas apresentadas, o presente projeto propõe a definição e implementação de um sistema de comunicação sem fio para automação residencial que apresente baixo custo, baixo consumo energético e seja de simples utilização. Para tal, a rede será baseada nas plataformas abertas Arduino e Raspberry Pi, assegurando baixo custo e ampla disponibilidade. A transmissão sem fio será feita através do módulo Bluetooth HM-10, de baixo consumo. Por fim, serão desenvolvidas bibliotecas que se encarreguem das rotinas de baixo nível relacionadas à comunicação Bluetooth, permitindo que o usuário tenha acesso às suas funcionalidades de modo simples.

# 2. REVISÃO BIBLIOGRÁFICA

## 2.1. Domótica

A palavra “domótica” deriva do latim “*domus”*, que significa “casa” ou “lugar que se habita”. Em termos gerais, se refere ao estudo e implementação de sistemas de sensoriamento e controle dentro do âmbito doméstico. Dessa forma, uma residência comum se torna inteligente como resultado da implantação de sistemas domóticos e métodos modernos de construção (BOLZANI, 2004). De fato, tem-se que uma casa inteligente:

> - Utiliza dispositivos que desenvolvem funções extras contribuindo para a sua própria gestão e para a gestão da residência, substituindo ou complementando os tradicionalmente usados.
> - Utiliza conceitos modernos de arquitetura e de construção, possibilitando o uso mais apropriado de fontes naturais de energia, reduzindo a taxa de utilização de equipamentos de iluminação, ventilação, aquecimento e esfriamento, reduzindo, por consequência, o consumo de energia elétrica. (BOLZANI, 2004)

Esta área tem se tornado mais interessante conforme tecnologias baseadas em microcontroladores de baixo custo se tornam mais amplamente disponíveis, viabilizando a implantação de sistemas distribuídos. Sistemas domóticos podem se valer da maior capacidade de processamento local, permitindo que não só a coleta de dados, mas também o acionamento, se deem de forma distribuída, reduzindo a carga no controlador central (MICHEL *et al,* 2008).

Ainda segundo Michel *et al* (2008), instalações de automação residencial devem ser projetadas de modo a não causarem grande interferência no ambiente, em especial não criando a necessidade de modificações estruturais no prédio. Dessa forma, percebe-se que há significativa demanda para o desenvolvimento de redes de automação sem fio, que atuem de maneira pouco intrusiva. Como exemplo de desenvolvimentos nesta área tem-se o módulo de

sensoriamento sem fio *Telos,* feito na Universidade de Berkeley, e os módulos *XBee*, desenvolvidos pela Digi.

## 2.2. Redes de Sensores Sem Fio

As redes de sensores sem fio fornecem uma infraestrutura que permite ao administrador a possibilidade de observar e reagir a eventos que ocorram no ambiente em questão. Exemplos de redes de grande escala e alto custo incluem radares para controle de tráfego aéreo, a rede nacional de distribuição de energia e estações meteorológicas. Em se tratando de menor escala e custo observam-se aplicações na área de segurança predial, cuidados com indivíduos debilitados e automação residencial (SOHRABY, MINOLI, ZNATI, 2007).

Redes de sensores sem fio são formadas por quatro componentes básicos: um conjunto de módulos de sensoriamento remoto, uma rede de conexão sem fio, um módulo central e um conjunto de recursos para processamento das informações no módulo central (SOHRABY, MINOLI, ZNATI, 2007). Os módulos de sensoriamento, também conhecidos como *motes*, possuem como características pequenas dimensões e baixo consumo energético, de modo que possam ser instalados sem modificações estruturais no ambiente (MICHEL *et al*, 2008). Tais módulos podem ou não ser equipados com a capacidade de processar os dados coletados, dependendo dos requisitos específicos de cada projeto.

## 2.3. Bluetooth

O nome *Bluetooth* teve origem com Harald Bluetooth, segundo rei da Dinamarca, famoso por unir facções inimigas na região da Escandinávia sob seu comando. De forma similar, a tecnologia *Bluetooth* foi criada com o objetivo de permitir que diferentes produtos de diferentes empresas possam se comunicar através de um mesmo padrão, aberto e acessível. O padrão é mantido por um “grupo de interesse especial” (ou SIG -

"*special interest group"* em inglês) que conta com mais de 25 mil empresas, dentre elas Apple, Intel, Microsoft e Toshiba (Bluetooth SIG, [200-?]).

A tecnologia de transmissão sem fio *Bluetooth* se baseia em ondas de baixo comprimento de onda com frequência na banda de 2,4 GHz. A frequência de 2,4 GHz faz parte das bandas de rádio para uso industrial, científico e médico (*ISM bands)*, para as quais não há necessidade de licenciamento prévio para uso (ITU, 2012). Por não se tratar de uma banda licenciada é comum que sinais de dispositivos diferentes interfiram no funcionamento um do outro. Para evitar que tal interferência prejudique a transmissão de dados, o *Bluetooth* utiliza uma técnica chamada "*frequency hopping spread spectrum"*, na qual a frequência de banda utilizada para transmissão muda constantemente, evitando que um sinal fixo interfira significativamente na transmissão; no caso específico do padrão *Bluetooth* essa troca de frequências ocorre 1600 vezes por segundo (Bluetooth SIG, 2008). A transmissão dos dados ocorre por meio de modulação por chaveamento de frequência gaussiana (GFSK, da sigla em inglês), de modo que mudanças na frequência de transmissão do sinal indicam o valor de cada bit.

Quando dois ou mais dispositivos equipados com *Bluetooth* se conectam, forma-se uma "piconet". O termo *piconet* se refere a uma rede formada por dois ou mais dispositivos, sendo que um deles é designado como *master* e os demais são designados como *slaves*. Cada *piconet* possui um *clock* e uma sequência de transição únicos, definidos com base no *clock* e no endereço do dispositivo atuando como *master*. Como uma *piconet* é definida pelos dispositivos conectados entre si é possível que várias redes compartilhem o mesmo espaço físico, com a limitação de que cada dispositivo só pode se conectar a uma das redes de cada vez (Bluetooth SIG, [200-?]).

## **2.4.** Topologia de Rede

### 2.4.1. Topologia Daisy Chain

Na topologia *daisy chain* os dispositivos se conectam em série ou em anel, formando uma espécie de corrente, de modo que para chegar ao seu destino as mensagens devem passar por cada dispositivo entre o ponto de partida e o de destino.

Na versão linear cada dispositivo se conecta com outros dois, com exceção de dois deles, que são definidos como extremos e representam o início e o fim da corrente. A versão da topologia chamada “topologia em anel” pode ser obtida conectando os dispositivos dos extremos da corrente entre si. A imagem 1 abaixo ilustra os dois tipos de topologia.

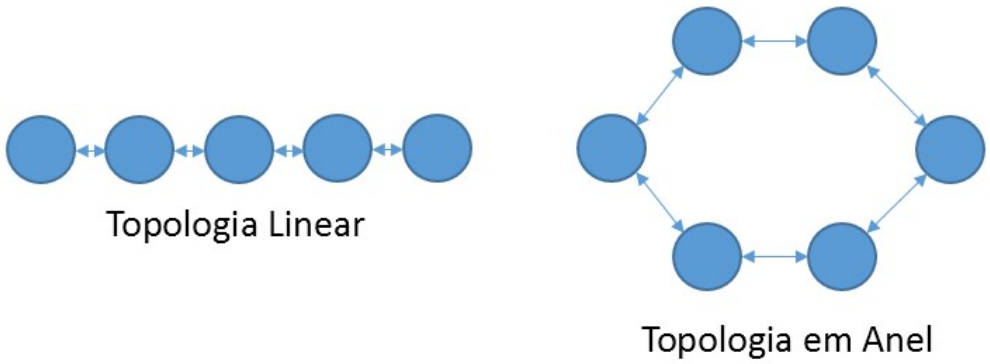


*Figura 1: Topologia linear e em anel*
*Fonte: Elaborada pelo autor*

É importante notar que a conexão entre os dispositivos é feita de modo que as mensagens podem ser transmitidas em ambas as direções. Devido a essa característica a topologia em anel possui as vantagens de reduzir o trajeto de uma mensagem em até 50% e de poder continuar operando em caso de falha de um dos nós (TANENBAUM, 2010).

### 2.4.2. Topologia em estrela

Na topologia em estrela um dos nós é definido como coordenador da rede, tornando-se o ponto central da malha de dispositivos. Esse coordenador tem a responsabilidade de inicializar e comandar os nós periféricos, recebendo dados e enviando comando a todos eles, possuindo tantas conexões quanto for o número de nós na rede. Os nós periféricos não se conectam entre si, de modo que cada um deles só apresenta uma conexão, com o nó central (SOHRABY, MINOLI, ZNATI, 2007). Devido a estas características de conexão a representação de uma rede em estrela lembra uma estrela, como pode ser visto na Figura 2. Esta disposição de rede é resistente a falhas nos nós periféricos, já que uma falha em um deles não afeta os outros, porém é

dependente do funcionamento correto do nó central, sem o qual a rede deixa de funcionar.

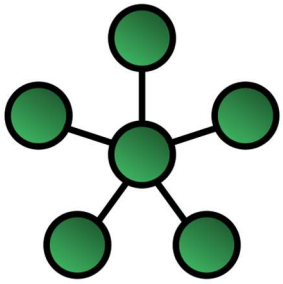

*Figura 2: Topologia em estrela*
*Fonte: wikipedia.org*

### 2.4.3. Topologia em malha

Em redes estruturadas pela topologia em malha qualquer nó da rede pode atuar como dispositivo fonte e se conectar a outro nó, atuando como dispositivo destino. Redes estruturadas dessa forma são auto organizáveis e possuem grande resistência falhas e capacidade de auto reparação. Para que os dispositivos da rede possam se conectar é necessário o uso de um algoritmo de roteamento, sendo ele baseado em árvore ou matriz (SOHRABY, MINOLI, ZNATI, 2007). A Figura 3 mostra a representação gráfica de uma rede em malha.

*Figura 3: Topologia em malha*
*Fonte:wikipedia.org*

# 3. IMPLEMENTAÇÃO

## 3.1. Hardware

### 3.1.1. Arduino Pro Mini

A plataforma Arduino integra componentes de hardware, na forma de diversos modelos de microcontroladores, e software, na forma de uma IDE de desenvolvimento baseada na linguagem C e do software que é embarcado nos microcontroladores. Por ter como um dos pilares de seu desenvolvimento a facilidade de uso e a acessibilidade, a plataforma apresenta grande sinergia com a proposta deste projeto. Tal facilidade de uso provém de dois pontos, sendo eles a simplicidade na coleta de dados e envio de sinais e a grande quantidade de tutoriais e informações disponíveis (Arduino, [200-?]).

Em termos de hardware, a plataforma Arduino apresenta mais de uma dezena de modelos diferentes, cada um com suas características e aplicações recomendadas. Tais modelos apresentam como características comuns a presença de pinos de entrada e saída e um microcontrolador, permitindo que possam receber informações de sensores e transmitir informação para atuadores. As diferenças entre os modelos estão no número de pinos, modelo de microcontrolador, tamanho da placa, interface de comunicação serial, entre outras.

Levando em conta as características desejadas neste projeto, como tamanho reduzido e baixo consumo, o modelo selecionado foi o Arduino Mini Pro (Figura 4). Este modelo é indicado pelo fabricante para uso semi-permanente, sendo assim compatível com a aplicação desejada. As características do modelo podem ser observadas na Tabela 1.

*Tabela 1: Especificações do Arduino Mini Pro*

| | |
|---|---|
| Microcontolador | ATmega328p |
| Voltagem de operação | 5V |
| Voltagem de entrada | 5 – 12V |
| Pinos de entrada/saída digital | 14 (sendo que 6 podem ter saída em PWM) |
| Pinos de entrada analógica | 8 |
| Corrente DC por pino de entrada/saída | 40mA |
| Memória Flash | 32kB |
| SRAM | 2kB |
| EEPROM | 1kB |
| Velocidade de Clock | 16MHz |
| Dimensões | 17,8mm x 33mm x 1.6mm |

Além das características listadas acima, um fator importante na decisão pelo uso deste modelo foi o custo. Atualmente o Arduino Pro Mini é vendido por $9,95 no fabricante oficial e por menos de $2 em fabricantes alternativos. Comparado ao valor de $24,95 do modelo Arduino UNO tem-se uma redução significativa nos custos.

A maior limitação do modelo se encontra no fato de que ele não possui interface USB integrada, sendo necessário o uso de um adaptador USB-serial externo para comunicação entre um computador e o módulo. Como a intenção do projeto é de instalar os módulos em pontos dispersos de uma residência e realizar comunicação via Bluetooth, tal limitação é significativamente mitigada. De fato, a falta de uma interface USB acaba por beneficiar o projeto devido à redução inerente no consumo de energia.

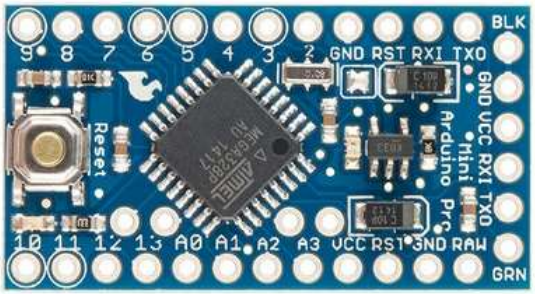


*Figura 4: Arduino Mini Pro*
*Fonte: Arduino.cc*

### 3.1.2. Raspberry Pi

Desenvolvido pela *Raspberry Pi Foundation*, uma empresa educacional sem fins lucrativos sediada no Reino Unido, o Raspberry Pi é um computador de baixo custo com a finalidade de permitir que pessoas de todas as idades e níveis de conhecimento possam ter acesso a uma plataforma de aprendizado em programação e eletrônica (RASPBERRY PI FOUNDATION, [200-?]). Apesar de seu tamanho reduzido e baixo custo, o Raspberry Pi possui todas as funções que se espera de um computador usual, tendo capacidade de navegar na internet, reproduzir arquivos de vídeo ou áudio e editar documentos.

A decisão de utilizar este computador como unidade central na rede de automação desenvolvida neste projeto foi baseada na facilidade e flexibilidade que o mesmo apresenta em termos de interação com o usuário. A interação com o Raspberry Pi pode ser feita através de mouse e teclado conectados diretamente a ele por USB, juntamente com um monitor de vídeo HDMI, ou através de um terminal remoto por meio da rede Wi-Fi local. Além disso, o computador poderia ser usado como parte de um sistema de entretenimento, assumindo mais de uma função no ambiente doméstico, como *player* de músicas e vídeos, por exemplo.

O modelo utilizado neste projeto foi o *Raspberry Pi 1 Model B* (Figura 5). Trata-se de uma versão antiga da plataforma, tendo sido substituído inicialmente pelo modelo B+ e atualmente pelo modelo 2B. O modelo apresenta as seguintes especificações:

*Tabela 2: Especificações da Raspberry Pi 1 modelo B*

| | |
|---|---|
| Processador | Broadcom BCM2835 |
| Clock do Processador | 700 MHz |
| RAM disponível | 256 MB |
| Conector de Rede | Ethernet 10/100 |
| Conector de Vídeo | HDMI, RCA |
| Conectores USB | 2.0 (2x) |
| Pinos de I/O | 26 |
| Consumo | 5V; 700mA; 3,5W |
| Dimensões | 85 x 56 x 17 mm |

O modelo utilizado no desenvolvimento do projeto não está mais à venda, mas suas versões mais novas podem ser encontradas por $25, para o modelo B+, e $35, para o modelo 2B. Os modelos mais recentes possuem vantagens como processamento mais rápido, maior capacidade de memória e mais pinos de I/O. Apesar das diferenças, o que foi desenvolvido neste projeto é inteiramente compatível com os novos modelos, não havendo prejuízo na substituição entre eles.

*Figura 5: Raspberry Pi 1 modelo B*
*Fonte: iot-playground.com*

### 3.1.3. Módulo Bluetooth HM-10

O módulo Bluetooth HM-10 é o que permite a comunicação sem fio na rede doméstica desenvolvida neste projeto. Desenvolvido pela *JNHuaMao Technology Company* e baseado no microcontrolador CC2541 da *Texas Instruments*, o módulo é capaz de transmitir informações sem fio utilizando o padrão Bluetooth 4.0, ou

*"Bluetooth Low Energy"* (JNHUAMAO, 2014). Como o próprio nome diz, a versão 4.0 do padrão Bluetooth tem como principal atrativo o baixo consumo de energia, sendo que o módulo utilizado pode diminuir sua corrente para até 400µA em modo *sleep*.

De modo geral, o módulo atua como uma ponte entre dois dispositivos da rede. Após efetuada a conexão, tudo que é disponibilizado na porta de comunicação serial do módulo é enviado para seu par, que por sua vez transmite os dados para a porta serial conectada a ele. A Figura 6 representa de forma simplificada essa estrutura.

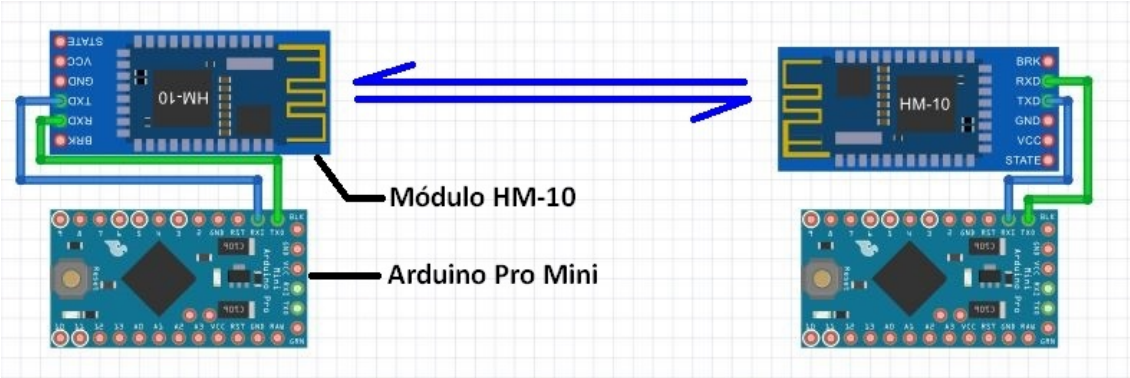


*Figura 6: Comunicação sem fio entre dois Arduinos*
*Fonte: Elaborada pelo autor*

As especificações do módulo podem ser observadas na Tabela 3, a seguir. Adicionalmente, a Figura 7 apresenta o desenho esquemático do mesmo. Durante o desenvolvimento deste trabalho os principais pinos utilizados foram os pinos 1 e 2, responsáveis pela comunicação serial com o Arduino ou com o Raspberry Pi, os pinos 12 e 13, para alimentação do módulo, e o pino 24, que quando conectado a um LED apresenta um indicador visual do estado de conexão do módulo. É possível notar que existem 11 pinos de I/O disponíveis para uso no módulo, devido ao fato de que o mesmo possui a capacidade de realizar leituras de sensores de forma independente. Essa capacidade não foi utilizada no projeto pois implicaria em maior dificuldade de implementação pelo usuário e redução da flexibilidade de aplicações.

*Tabela 3: Especificações do módulo Bluetooth HM-10*

| | |
|---|---|
| Microcontrolador | TI CC2541 |
| Versão Bluetooth | 4.0 |
| Frequência de Operação | 2,4GHz |
| Modulação | GFSK (Modulação Gaussiana) |
| Voltagem de Operação | 3,3V |
| Corrente de Operação | 50mA max; 8,5mA em operação |
| Corrente em Modo Sleep | 400µA |
| Dimensões | 27 x 13 x 2,2 mm |

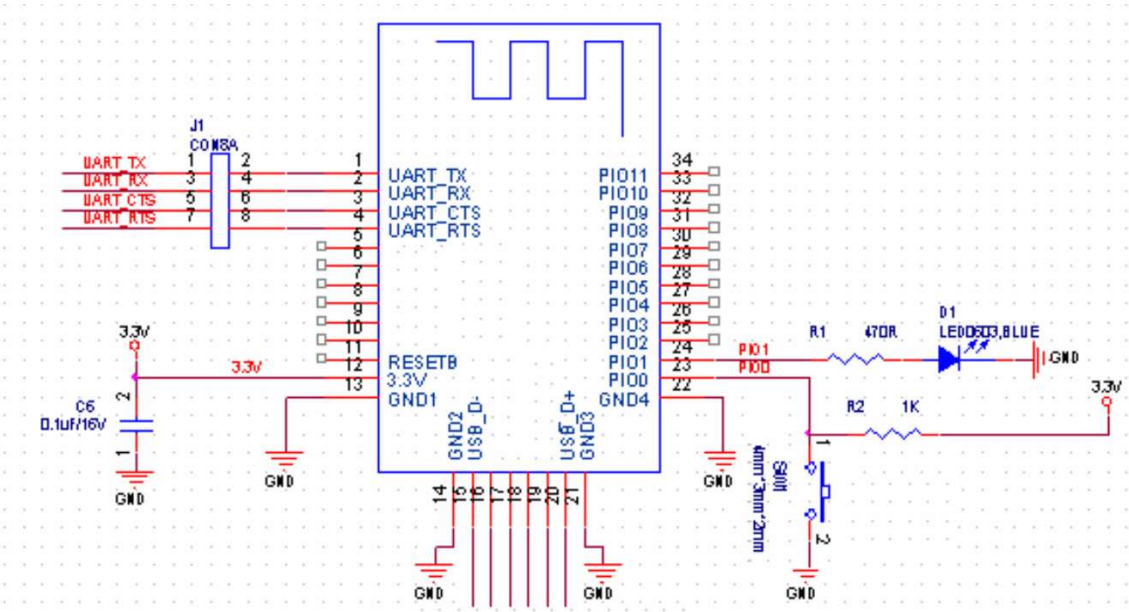


*Figura 7: Desenho esquemático do módulo HM-10*
*Fonte: HM-10 Module Datasheet*

## 3.2. Software

O objetivo da parcela de software do projeto é de proporcionar ao usuário uma forma simples e robusta de utilizar as capacidades de comunicação sem fio do módulo *Bluetooth* para transferência de dados entre os microcontroladores. Para permitir que indivíduos com pouco conhecimento em programação e eletrônica utilizem as tecnologias propostas neste projeto, desenvolveu-se um conjunto de funções de simples utilização que executam as principais tarefas necessárias para a comunicação.

Espera-se que, quando munido de tais funções, um usuário leigo possa desenvolver facilmente aplicações utilizando o módulo *Bluetooth*.

### 3.2.1. Pacote de dados

Para garantir que a comunicação entre as unidades da rede se dê de forma confiável foi definida uma estrutura para transmissão de dados. Tal estrutura define que cada mensagem enviada carregue não só apenas os dados desejados, mas também informações sobre remetente, destinatário e um valor para conferência de integridade da mensagem. A Figura 8 representa graficamente a organização dos dados dentro de um pacote de transmissão.


*Figura 8: Estrutura do pacote de dados*
*Fonte: Elaborada pelo autor*

Pode-se observar que o pacote é composto por quatro partes, sendo elas:

- IDS (ID *Sender*): É composta por 12 caracteres e carrega o valor do "endereço MAC" do módulo que enviou as informações.
- IDR (ID *Receiver*): É composta por 12 caracteres e carrega o valor do "endereço MAC" do módulo ao qual as informações foram destinadas.
- Dados: Possui tamanho variável e é composta pelos dados que foram selecionados para transmissão. É essa a parte do pacote à qual o usuário tem acesso.

- CS (*Checksum*): É composta por apenas um *byte* de verificação. Permite que a integridade da mensagem seja verificada pelo receptor, reduzindo a chance de erros na transmissão da informação.

A construção e a interpretação do pacote acima são feitas de modo interno às funções desenvolvidas, de modo que não é necessário que o usuário tenha a estrutura em mente durante a implementação de seus projetos. Detalhes sobre a utilização do pacote podem ser encontrados nas descrições a seguir, em especial dentro das funções ***send*** e ***getpacket***.

### 3.2.2. Conjunto de Funções desenvolvidas

#### 3.2.2.1. Função Connect

```
int connect(String);
```

A função ***connect*** tem o objetivo de permitir a conexão entre duas unidades de forma confiável a partir de um único comando. Para atingir tal objetivo a função não apenas envia os comandos adequados para o módulo *Bluetooth* como também recebe e analisa a resposta de modo a determinar se a conexão foi efetuada ou não.

Quando chamada, a função deve receber como argumento o "endereço MAC" do dispositivo ao qual se deseja conectar. Este endereço é uma sequência de 12 caracteres alfanuméricos que identifica de forma única cada módulo HM10 e tem como objetivo definir a qual módulo a unidade deve se conectar.

Após a execução da função existem três possíveis cenários, listados a seguir:

1. Módulos estão conectados. Função retorna inteiro "0"
2. Módulos não estão conectados pois o módulo alvo não foi encontrado. Função retorna inteiro "1"
3. Módulos não estão conectados pois o módulo local não foi capaz de interpretar o comando de conexão. Função retorna inteiro "-1"

Um fluxograma representando o funcionamento da função pode ser observado na Figura 9. Além disso, os próximos parágrafos descrevem detalhadamente a execução da função.

Inicialmente a função verifica se está executando a primeira tentativa de conexão e em caso afirmativo envia o comando necessário para configurar o módulo como "*master*", permitindo que o mesmo se conecte a outros módulos. A seguir, o comando de conexão é enviado para o módulo, juntamente com o MAC do módulo ao qual se deseja conectar. Imediatamente a função recebe e analisa a resposta do módulo, de modo a identificar se o comando foi aceito e uma tentativa de conexão será efetuada (resposta do módulo: "OK+CONNA"). Em caso negativo a função retorna o valor "-1", indicando o erro. Em caso afirmativo a função prossegue para o recebimento da segunda parte da mensagem, que indica se a conexão foi de fato efetuada.

O tempo de execução dessa etapa varia conforme a realização da conexão, sendo que o módulo permanece buscando por até 10s. A função está preparada para seguir com sua execução imediatamente após a chegada da segunda parte da resposta, independentemente do tempo transcorrido até sua chegada.

Após receber a segunda parte da resposta a função avalia se a conexão foi efetuada (resposta do módulo: "OK+CONN") ou não. Se os módulos estiverem conectados a função retorna o valor "0" e encerra sua execução. Caso a conexão tenha falhado a função verifica quantas tentativas já foram feitas e decide entre tentar novamente, no caso de menos de 3 tentativas, ou retornar "1", indicando que a conexão não foi possível.

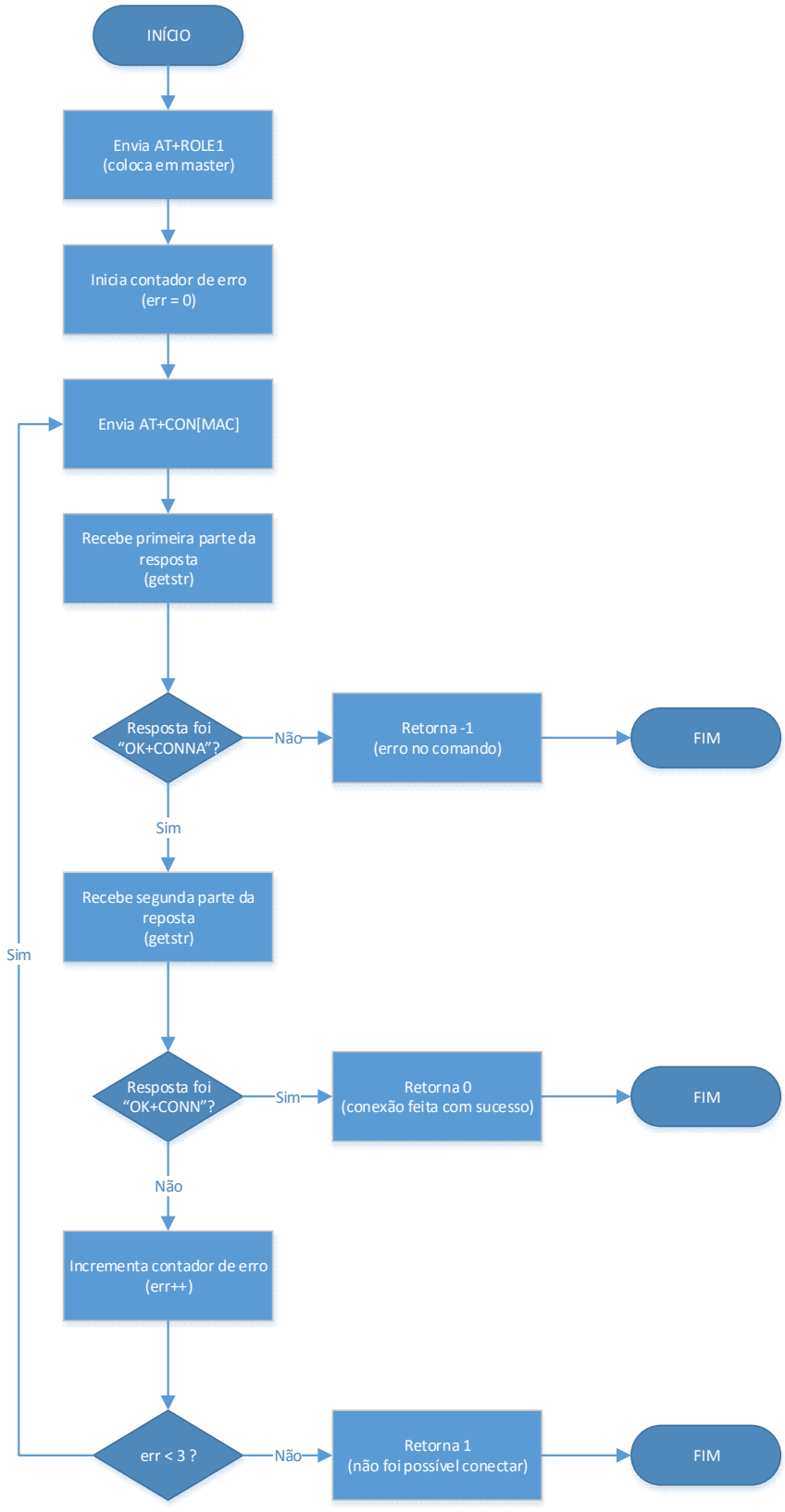


*Figura 9: Fluxograma da função **connect***
*Fonte: Elaborada pelo autor*

### 3.2.2.2. Função Disconnect

```
int disconnect(void);
```

A função ***disconnect*** atua de forma oposta à função ***connect***, permitindo que duas unidades sejam desconectadas com um simples comando. A desconexão é feita de modo mais simples, porém ainda de forma robusta, analisando a resposta do módulo para determinar se a desconexão foi efetuada como desejado.

Por não necessitar de parâmetros específicos a função ***disconnect*** não recebe valores como entrada. A execução da função independe de qual unidade esteja conectada, ou até mesmo de o módulo estar ou não conectado.

Após a execução da função existem três possíveis cenários, listados a seguir:

1. Conexão terminada com sucesso. Função retorna inteiro 0
2. Módulo já não apresentava conexão antes da execução. Função retorna inteiro -1
3. Módulo responde de forma inesperada. Função retorna inteiro -2

Um fluxograma representando o funcionamento da função pode ser observado na Figura 10, na página seguinte. Quando chamada, a função envia ao módulo o comando “AT”, de modo a efetuar a desconexão. Em seguida, a função recebe a resposta do módulo e a analisa para verificar em qual das duas possibilidades ela se encontra. A primeira resposta, “OK+LOST”, indica que uma conexão estava presente e que a mesma foi devidamente terminada. Se verificada, a resposta leva o módulo a retornar o valor 0, indicando sucesso. A segunda resposta possível, “OK”, indica que o módulo não estava conectado mesmo antes da execução da função, o que pode indicar que algum erro tenha ocorrido na operação da unidade. Dessa forma, a função retorna o valor -1, indicando erro. Por fim, a função envia o comando “AT+ROLE0” para o módulo, a fim de deixá-lo em modo *slave* e disponível para solicitações de conexão por outras unidades.

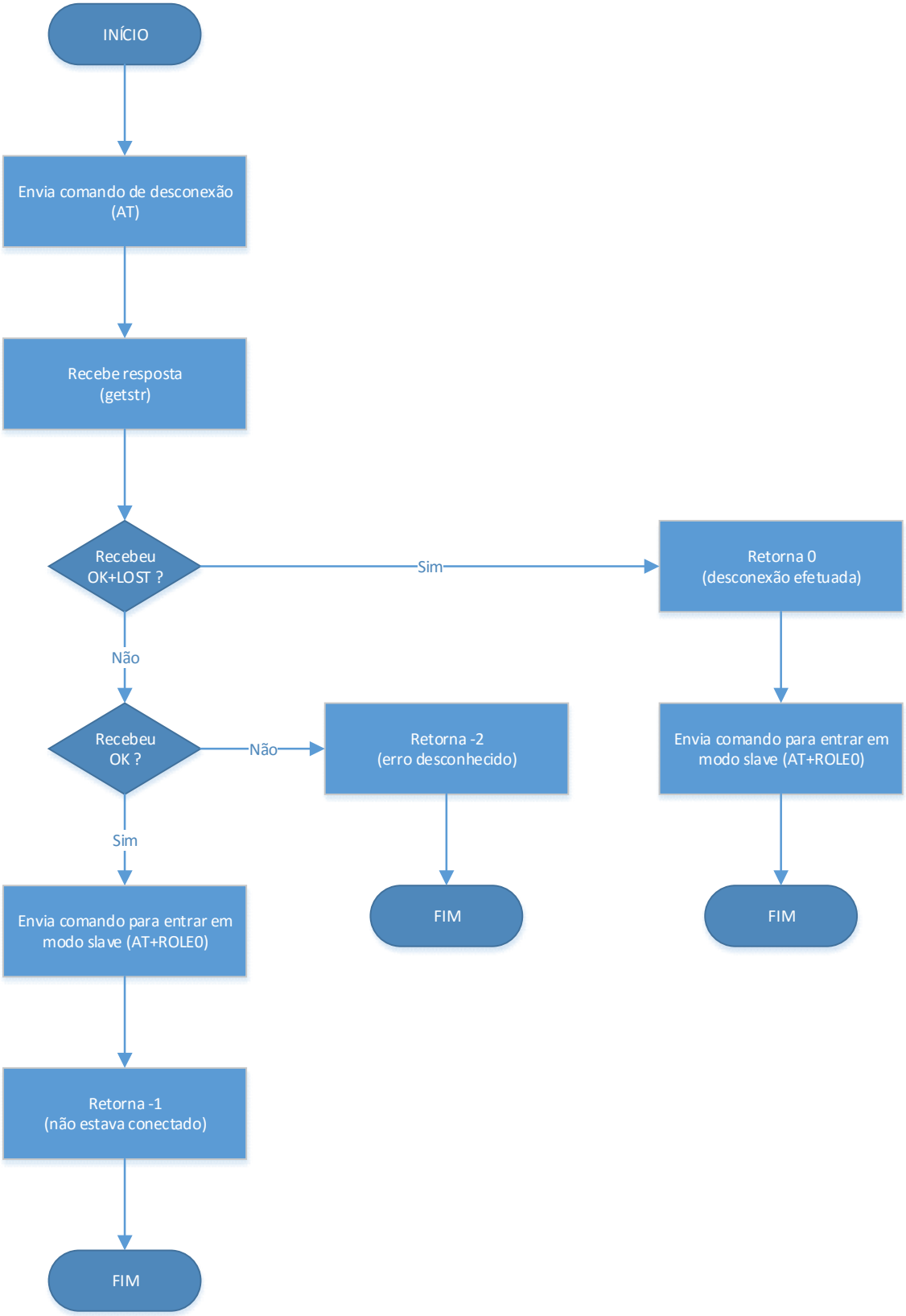


*Figura 10: Fluxograma da função **disconnect***
*Fonte: Elaborada pelo autor*

### 3.2.2.3. Função GetString

```
String getstr(void);
```

A função ***getstring*** tem como objetivo permitir que a unidade receba mensagens através da porta serial de forma simples. O uso da função é adequado quando se sabe que a mensagem provém do módulo local, e não de outra unidade, tendo em vista que a mensagem é simplesmente interpretada como uma *string*, sem a formatação proposta.

A função independe de parâmetros de entrada, tendo como retorno uma *string* de caracteres. Devido a limitações da comunicação serial tanto o Arduino quanto a Raspberry Pi somente podem acessar um caractere por vez, de modo que a organização de todos os caracteres de uma mensagem permite que a mesma seja interpretada como um todo, facilitando a comunicação. Esse processo é usado múltiplas vezes dentro de outras funções, como ***connect*** e ***disconnect***, sendo necessário para o funcionamento das mesmas.

Após a execução da função existem dois possíveis cenários, listados a seguir:

1. Informação recebida com sucesso. Função retorna a informação em forma de *string*.
2. Tempo limite de 10s excedido sem que informação seja recebida. Função retorna *string* com um único caractere 0.

Um fluxograma representando o funcionamento da função pode ser observado na Figura 11, na página seguinte.

Ao ser chamada, a função ***getstring*** inicia um contador e aguarda dois possíveis cenários: A porta serial acusa que existem *bytes* disponíveis no *buffer*; ou o *timer* atinge 10 segundos sem que nenhuma mensagem seja recebida. No primeiro caso, a função copia a mensagem *byte* a *byte* para uma *string* temporária, aplicando um pequeno atraso entre as cópias para garantir que nenhum *byte* se perca, retornando finalmente a *string* como resultado. No segundo caso, a função entende que nenhuma mensagem

foi recebida e retorna uma *string* com um único caractere, 0, de modo a indicar que nada foi recebido.

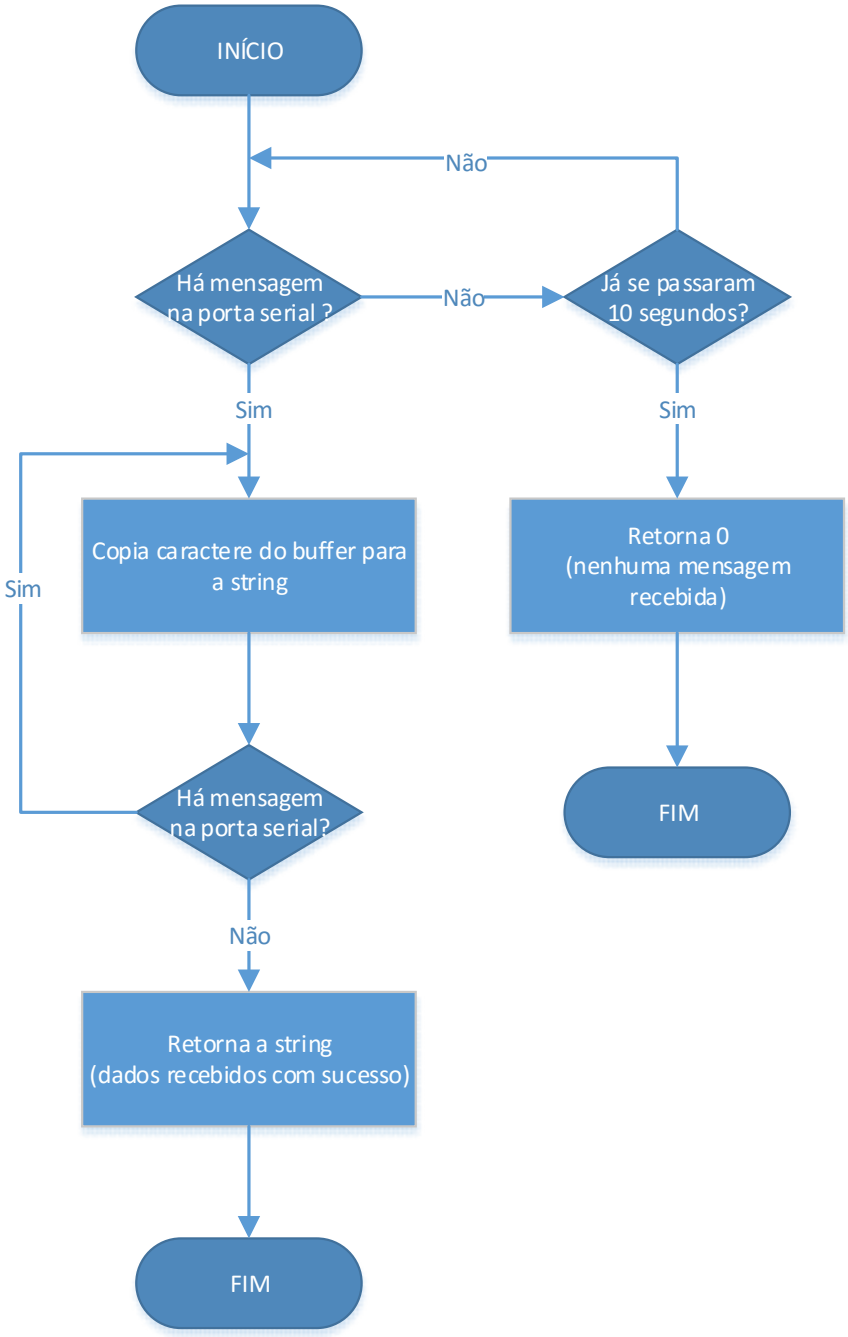


*Figura 11: Fluxograma da função* ***getstring***
*Fonte: Elaborada pelo autor*

### 3.2.2.4. Função GetPacket

```
packet getpkt(void);
```

A função ***getpacket*** funciona de modo similar à função ***getstring***, porém em vez de apenas receber os dados da porta serial e transferi-los para uma *string*, ela organiza essa informação em um pacote de comunicação padronizado. A organização leva em conta o tamanho da *string* recebida e o tamanho padrão das partes do pacote. Desse modo, a *string* é quebrada em quatro partes conforme o seguinte critério:

Caracteres 0 a 11 – ids (identificador do remetente)

Caracteres 12 a 23 – idr (identificador do receptor)

Caracteres 24 a *length*-1 – *data* (dados da mensagem)

Caracteres *length*-1 a *length* – *checksum*

No critério acima "*length*" indica o tamanho da *string* recebida, de modo que a parte de dados da *string* é identificada como os caracteres do 24 ao penúltimo e o *checksum* é identificado como o último caractere da mensagem. Caso nenhuma informação seja recebida durante os 10 segundos estabelecidos como limite, a função retorna um pacote de erro, com ids e idr nulos e mensagem de erro na parte de dados, além de *checksum* nulo.

Além de organizar a informação recebida, a função verifica se os dados recebidos mantiveram sua integridade durante a transmissão. Isso é feito através do cálculo do *checksum* dos dados e subsequente comparação com o *checksum* recebido na mensagem. Caso os valores sejam idênticos, uma mensagem de confirmação é enviada ao remetente ("CSOK"). Caso os valores sejam diferentes, uma mensagem de falha é enviada ("CSFAIL") e a função passa a aguardar o reenvio das informações corrompidas. Após três tentativas de recebimento falharem a função encerra o recebimento e retorna um pacote de erro para o programa. A Figura 12 apresenta uma representação gráfica do comportamento da função.

Com o objetivo de aumentar a robustez da função foi criada uma exceção para os casos em que a mensagem recebida não possui caracteres suficientes para constituir um pacote de dados. Nesses casos, a função retorna um pacote com IDS, IDR e CS nulos, mas com a mensagem recebida dentro da área de dados. Caso a função a mensagem recebida seja “OK+CONN”, por exemplo, a função retornará um pacote com IDS, IDR e CS nulos e com “OK+CONN” na região de dados.

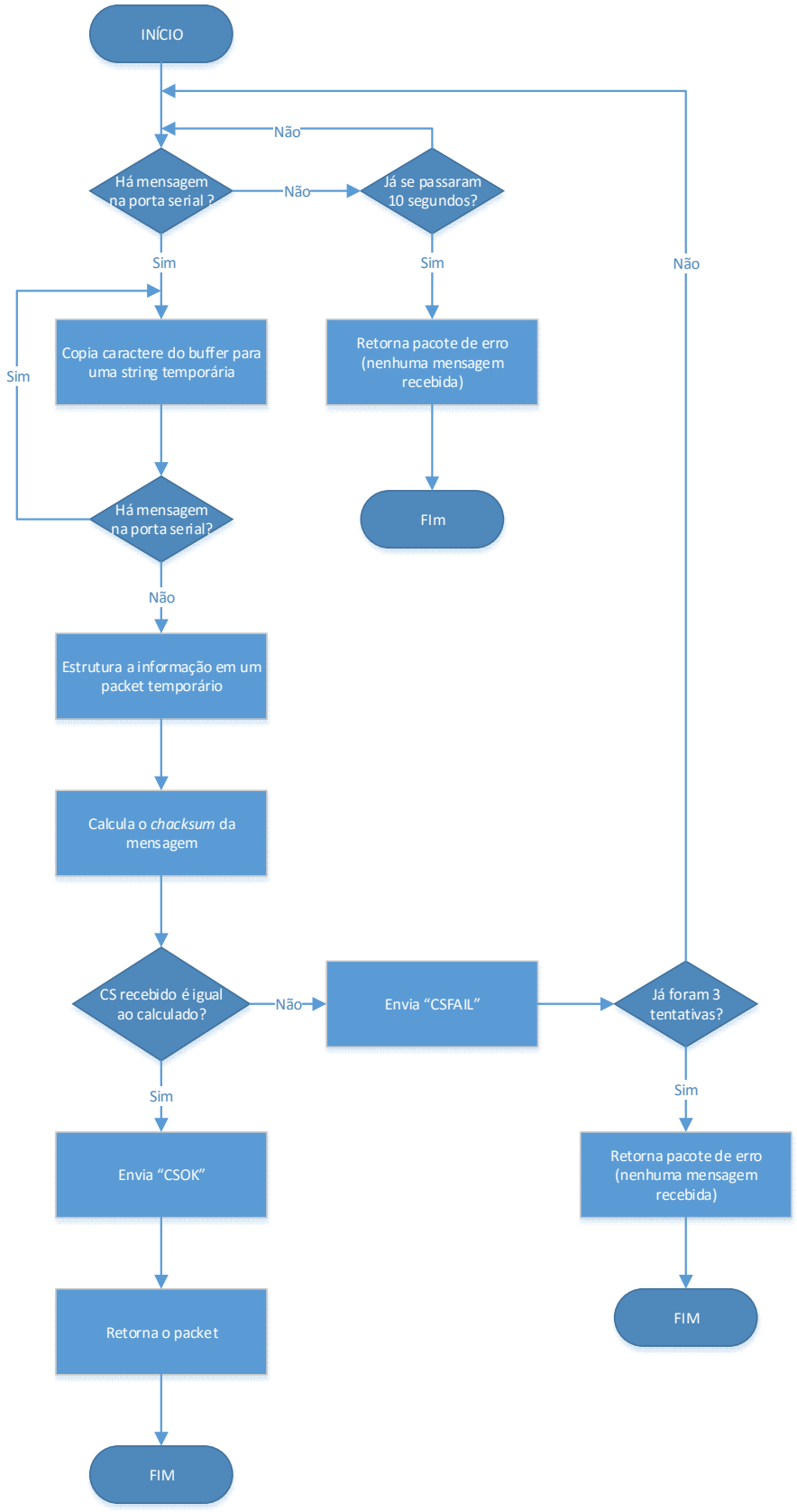


*Figura 12: Fluxograma da função **getpacket***
*Fonte: Elaborada pelo autor*

#### 3.2.2.5. Função Send

```
int send(String, String);
```

O objetivo da função ***send*** é enviar dados especificados pelo usuário de forma organizada e robusta. Para tal, a função se vale da estrutura apresentada anteriormente, combinando os dados com os identificadores IDS e IDR, além de calcular o *checksum* da mensagem.

Ao chamar a função o usuário deve especificar dois argumentos, sendo eles o identificador do destinatário (IDR) e a informação que deve ser transmitida (dados). A função então organiza o identificador do módulo local (IDS), o IDR, os dados e o *checksum* em uma única *string*, para que seja enviada.

Após o envio da mensagem o código aguarda uma resposta da unidade com a qual está se comunicando, de modo a garantir que a mensagem foi transmitida com sucesso. Caso a unidade remota responda indicando que a mensagem foi corrompida durante a transmissão inicia-se uma nova tentativa de envio. Após três falhas no envio da mensagem a função encerra suas atividades e retorna código de erro (1) ao programa.

Na Figura 13 pode-se observar um fluxograma que representa o funcionamento da função. Por motivos de claridade se omitiu uma etapa de decisão onde a função verifica se está enviando uma mensagem de confirmação (“CSOK” ou “CSFAIL”) e em caso positivo não aguarda resposta. Esse comportamento permite que as funções **getstring** e **send** não resultem em um loop infinito de confirmação.

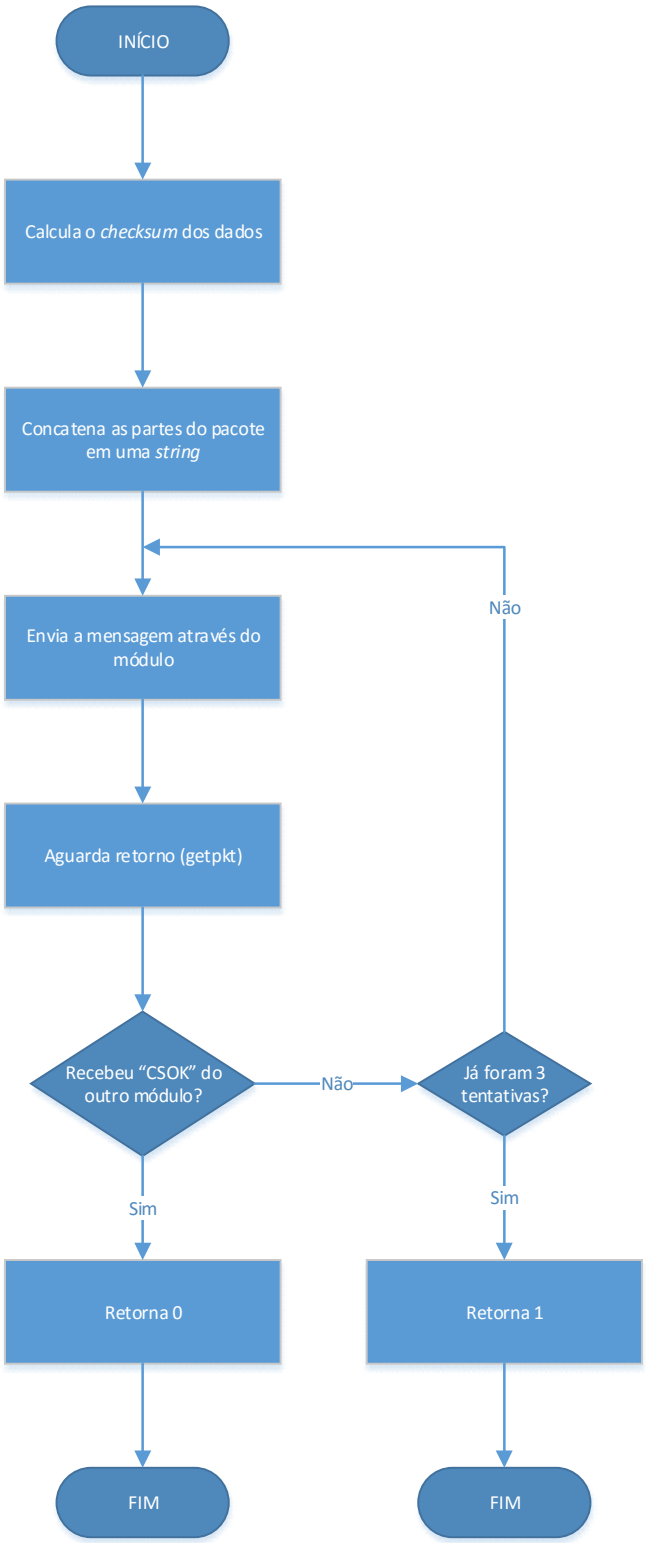


*Figura 13: Fluxograma da função* ***send***
*Fonte: Elaborada pelo autor*

## 3.3. Atrasos

Durante o desenvolvimento do projeto notou-se que a aplicação de atrasos no código é necessária para que o módulo *Bluetooth* funcione adequadamente. Especificamente, atrasos devem ser aplicados quando mais de um comando é enviado ao módulo de forma consecutiva, para evitar que uma instrução atrapalhe no funcionamento da outra, ou quando são enviados dados e comandos em sequência. No algoritmo da Figura 14, abaixo, é possível observar um exemplo dessa situação:

```
serialPuts (fd,"AT+ROLE1");
getstr(check);
delay(800);

buff[0] = '\0';
strcat(buff,"AT+CON");
strcat(buff,MAC);
puts(buff);

for(err=0; err<3; err++)
{
    serialPuts (fd,buff);
    getstr(check);
```

*Figura 14: Exemplo de código com atraso*
*Fonte: Elaborada pelo autor*

O trecho de código exibido faz parte da função ***connect***. A primeira linha em destaque envia o comando “AT+ROLE1”, colocando o módulo em modo *master*, para que ele possa iniciar uma conexão com outro módulo. A segunda linha em destaque aplica um atraso de 800ms, permitindo que o módulo processe adequadamente a instrução enviada antes de prosseguir com a execução. A terceira linha em destaque envia um segundo comando, “AT+CON[MAC]”, ordenando que o módulo se conecte a outro. Foi estabelecido por meio de testes variando-se o atraso entre 10ms e 1000ms que, nesse caso, um atraso mínimo de 800ms entre os comandos é necessário para garantir que as instruções sejam executadas de forma correta.

O uso de atrasos também é necessário em situações em que a transmissão de dados é acompanhada do uso de comandos. A razão para isso é que quando uma mensagem e um comando são enviados em sucessão o módulo não é capaz de distinguir os dois, de modo que acaba por enviar o comando como se fosse parte dos dados. Um trecho do código no qual tal situação acontece está representado na Figura 15. Nessa situação a função ***getpkt*** envia uma mensagem de confirmação, indiciando que a mensagem chegou e não está corrompida, e a função ***disconnect*** envia o comando "AT" em seguida, para desconectar o módulo. Sem a aplicação do atraso o comando acaba por ser enviado como parte da mensagem, anexado ao final, de modo que o módulo que está recebendo os dados receberia "[mensagem]AT", e o comando não seria executado. Novamente, por meio de testes variando-se o atraso entre 10ms e 1000ms, determinou-se que o atraso mínimo necessário neste tipo de situação é de 500 milisegundos.

```
send(foward.idr, foward.data);
reply = getpkt();
delay(500);
disconnect();
```

*Figura 15: Exemplo de código com atraso 2*
*Fonte: Elaborada pelo autor*

É importante notar que mesmo que o módulo envie uma resposta para o comando não há garantia de que a operação tenha sido executada adequadamente. A Figura 16 apresenta o terminal da Raspberry Pi durante a execução da função ***connect*** sem a aplicação de atraso. Pode-se observar que após receber o comando de conexão o módulo responde com "OK+CONN", indicando que a conexão foi estabelecida com sucesso. Porém, ao enviar o comando de desconexão para o módulo a resposta recebida é "OK", o que indica que o módulo não estava conectado. Além disso, não foi possível enviar dados entre os módulos supostamente conectados e a conexão era terminada espontaneamente após alguns momentos.

É importante também destacar que a documentação do fabricante não menciona a necessidade de incluir esses atrasos, causando vários problemas durante a prototipação do sistema. Esses problemas tiveram de ser solucionados experimentalmente para viabilizar a utilização dos módulos HM-10 no estabelecimento da rede de comunicação pretendida.

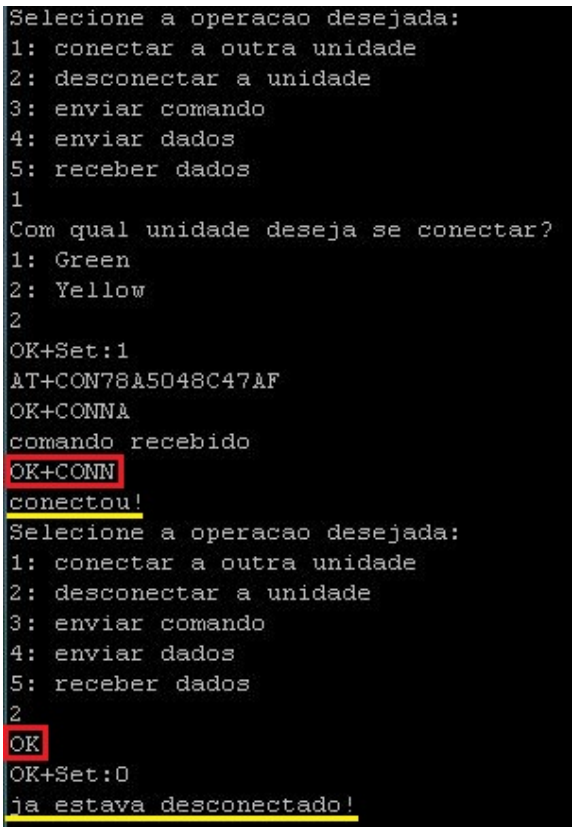


*Figura 16: Função connect sem aplicação de atraso*
*Fonte: Elaborada pelo autor*

## 3.4. Programação

A seção anterior trata do software de forma abstrata, apresentando a lógica de operação do sistema e descrições de alto nível, de modo que o conteúdo é valido para as duas plataformas utilizadas. Apesar de compartilharem a mesma lógica de funcionamento, as duas plataformas utilizadas não podem ser programadas de forma idêntica, fazendo necessária a criação de duas versões do software. Esta seção apresenta os trechos do código onde a implementação divergiu e explica as razões para tal.

## 3.5. Comunicação Serial

No Arduino, a comunicação serial pode ser implementada nas portas específicas para comunicação serial ou por emulação em portas de I/O digitais. A emulação é feita por meio da biblioteca "*SoftwareSerial*", que faz parte do conjunto padrão de bibliotecas da plataforma. A utilização da emulação no desenvolvimento deste projeto foi baseada no fato de que ela permite que o computador se comunique com o microcontrolador mesmo quando o módulo *Bluetooth* estando conectado. As funções disponíveis na biblioteca mais utilizadas no desenvolvimento do projeto estão descritas na Tabela 4, abaixo. A documentação completa da biblioteca pode ser encontrada em: *arduino.cc/en/Reference/SoftwareSerial*

*Tabela 4: Funções da biblioteca SoftwareSerial*

| | |
|---|---|
| void SoftwareSerial mySerial(rx, tx) | Indica as portas a serem utilizadas para emulação da comunicação (*setup*) |
| void mySerial.begin(int baud) | Inicia a comunicação e determina a taxa de transmissão de bits a ser utilizada *(setup*) |
| int mySerial.available() | Indica a quantidade de bytes disponíveis para leitura no buffer |
| char mySerial.read() | Retorna o valor do primeiro byte disponível e o apaga do buffer |
| void mySerial.write("data") | Envia uma sequência de caracteres determinada no argumento da função |
| void mySerial.print(string data) | Envia os dados contidos na variável especificada no argumento |

Na Raspberry Pi o acesso à comunicação serial não é tão simples, dado que o código é executado dentro de um sistema operacional linux, o que aumenta a complexidade de acessar componentes do hardware. Para mitigar tal complexidade foi utilizada uma biblioteca desenvolvida para facilitar a utilização de comunicação UART na placa, chamada "*Wiring Pi Serial Library*". Esta biblioteca foi elaborada de modo a oferecer comandos e funcionalidades similares às encontradas no Arduino, possibilitando que o código desenvolvido para o mesmo pudesse ser facilmente adaptado. As funções disponíveis na biblioteca mais utilizadas no desenvolvimento do

projeto estão descritas na Tabela 5. A documentação completa da biblioteca pode ser encontrada em: *wiringpi.com/reference/serial-library/*

*Tabela 5: Funções da biblioteca Wiring Pi Serial*

| | |
|---|---|
| int serialOpen (char *device, int baud) ; | Abre e inicializa a comunicação serial no dispositivo, com a taxa de bits determinada |
| void serialPutchar (int fd, unsigned char c) | Envia um caractere para o dispositivo especificado |
| void serialPuts (int fd, char *s) | Envia a *string* de caracteres para o dispositivo especificado |
| int serialDataAvail (int fd) | Indica a quantidade de bytes disponíveis para leitura no buffer |
| char serialGetchar (int fd) | Retorna o valor do primeiro byte disponível e o apaga do buffer |
| void serialFlush (int fd) | Apaga todos os bytes presentes no buffer |

Para que a biblioteca possa ser utilizada é necessário que seus arquivos sejam copiados do servidor do *GitHub* para o dispositivo. Isso pode ser feito através do seguinte conjunto de comandos, que devem ser inseridos no terminal:

| | |
|---|---|
| sudo apt-get update | (Necessário se o sistema estiver desatualizado) |
| sudo apt-get upgrade | (Necessário se o sistema estiver desatualizado) |
| sudo apt-get install git-core | (Necessário se o *GitHub* não estiver instalado) |
| gitclone git://git.drogon.net/wiringPi | Copia os arquivos do servidor para o armazenamento local |

Tendo os arquivos no armazenamento local, é necessário que durante a compilação se indique o uso da biblioteca. Isso pode ser feito pela adição do trecho *"-lwiringPi*" ao final do comando de compilação, resultando em "*gcc -o nomeprograma nomeprograma.c –lwiringPi*".

## 3.6. Strings

O modo como os dois sistemas lidam com *strings* apresenta diferenças significativas. A Raspberry Pi, tendo sido programada em C, não reconhece um tipo de variável diferenciado para este tipo de dado, de modo que uma palavra precisa ser tratada como um vetor contendo um caractere em cada compartimento. Outro elemento importante de uma *string* em C é o byte nulo (‘/0’) que deve ser inserido na última posição do vetor, sem o qual há o risco de o programa invadir espaços de memória próximos quando interagir com o vetor.

```
while(serialDataAvail(fd)>0)
{
    delay(10);
    buff[j] = serialGetchar(fd);
    j++;
}
buff[j] = '\0';
```

*Figura 17: Recebimento de mensagens na Raspberry Pi*
*Fonte: Elaborada pelo autor*

A Figura 17 apresenta o modo como os bytes recebidos na porta serial são copiados para um vetor, para que a mensagem possa ser interpretada adequadamente. O atraso aplicado entre a leitura de cada caractere garante que a mensagem será recebia em sua totalidade, evitando que a função encerre seu funcionamento antes da chegada de todos os bytes. Note que é necessário utilizar um contador para determinar a posição do vetor a ser preenchida, além da necessidade de acrescentar o caractere nulo ao final.

A biblioteca “string.h”, que faz parte do conjunto padrão de bibliotecas da linguagem, possui funções que possibilitam a manipulação e comparação de *strings* de forma simplificada. A Tabela 6 apresenta as funções mais utilizadas neste projeto. Tais

funções são usadas principalmente nas funções ***send*** e ***getpkt***, nas quais é necessário construir uma *string* única a partir de um pacote de dados e vice-versa.

*Tabela 6: Funções da biblioteca String.h*

| | |
|---|---|
| char *strcat(char *dest, const char *src) | Concatena a *string* src com a *string* dest |
| int strcmp(const char *str1, const char *str2) | Compara as *strings* str1 e str2, retornando "0" caso sejam iguais |
| char *strncpy(char *dest, const char *src, int n) | Copia os n primeiros caractreres da *string* src para a *string* dest |
| int strlen(const char *str) | Indica a quantidade de caracteres na *string* str, com exceção do caractere nulo |

O Arduino é capaz de trabalhar com *strings* da mesma forma que a descrita acima. Essa não é, porém, a única forma de fazê-lo, já que o Arduino apresenta uma classe de dados chamada "String" (com S maiúsculo), que permite maior flexibilidade na utilização das mesmas. A Figura 18 apresenta um trecho de código análogo ao código mostrado na Figura 17.

```
while (mySerial.available() > 0)
{
  c = mySerial.read();
  buff += c;
  delay(10);
}
```

*Figura 18: Recebimento de mensagens no Arduino*
*Fonte: Elaborada pelo autor*

É possível notar que a utilização de um contador para monitorar a posição do vetor a ser preenchida não é mais necessária, bastando um simples comando de adição entre a *String* ("buff") e o caractere desejado ("c"). Outros benefícios da biblioteca incluem a possibilidade de utilizar um simples comparador "==" entre duas *Strings,* sem que haja necessidade do uso de uma função especial, e a possibilidade de acessar partes da *String* por meio das funções "*substring*" e "*charAt*".

# 4. Resultados e discussões

## 4.1. Analise de Custo

Com o objetivo de mensurar o custo de implementação do sistema foram averiguados os custos de cada componente em três fontes distintas, sendo elas o mercado brasileiro, o mercado americano e o mercado chinês. Os valores apresentados na Tabela X são referentes ao preço médio encontrado no dia 29/11/2015, com a cotação do dólar vigente na data de R$3,823. É importante notar que apenas um fornecedor do módulo HM-10 foi encontrado no mercado brasileiro, de modo que a disponibilidade do componente pode vir a ser um problema caso a compra no mercado chinês não seja desejável.

*Tabela 7: Custo dos componentes da rede*

| **Componente** | **Preço Brasil (R$)** | **Preço EUA** | | **Preço China** | |
|---|---|---|---|---|---|
| | | **(R$)** | **(US$)** | **(R$)** | **(US$)** |
| Arduino Pro Mini | 65,00 | 38,04 | 9,95 | 6,12 | 1,60 |
| Raspberry Pi 1 B | 229,00 | 95,57 | 25,00 | 172,04 | 45 |
| HM-10 module | 49,90 | 36,70 | 9,60 | 28,67 | 7,50 |

Considerando que os componentes utilizados no desenvolvimento deste trabalho foram obtidos no mercado chinês o custo da unidade central é de cerca de R$200, enquanto que o custo de cada unidade periférica é de cerca de R$35.

## 4.2. Autonomia

Uma característica desejada para o sistema é que as unidades de sensoriamento e acionamento possam ser instaladas sem a necessidade de alimentação externa. Para que isso seja possível, é necessário que as unidades sejam capazes de operar por períodos de tempo prolongados a partir de uma pequena fonte de energia, como por exemplo uma bateria de 9V ou pilhas AA. Para averiguar o desempenho do sistema nesse quesito foram realizados testes de consumo do módulo *Bluetooth* e do Arduino, mensurando a corrente de alimentação em vários cenários. O consumo da Raspberry Pi

é pouco relevante, dado que a mesma será instalada em um ponto central da casa, próximo a fontes de alimentação externa, mas varia entre 460mA e 750mA dependendo da necessidade de processamento. A Tabela 8e a Tabela 9 abaixo apresentam o resultado das medições para o módulo *Bluetooth* e o Arduino, respectivamente.

*Tabela 8: Consumo de energia para o módulo HM-10*

| **Cenário** | **Consumo (mA)** |
|---|---|
| Desconectado, em repouso | 8,4 – 8,5 |
| Conectado, em repouso | 8,3 – 8,4 |
| Conectado, transmitindo | 8,7 – 8,8 |
| Modo *Sleep* | 0,45 – 0,47 |

*Tabela 9: Consumo de energia para o Arduino Mini Pro*

| **Cenário** | **Consumo (mA)** |
|---|---|
| Aguardando comunicação | 19,8 – 20,0 |
| Transmitindo dados | 19,6 – 19,7 |
| Modo *Sleep ADC* | 10,8 – 10,9 |
| Modo *Sleep Power Down* | 8,7 – 8,8 |

É possível observar que tanto para o módulo HM-10 quanto para o Arduino o consumo apresenta pouca variação entre os cenários de repouso e transmissão, assim como entre os cenários em que o módulo está conectado e desconectado. Tal variação aumenta significativamente quando se observa o consumo de ambos em modo *Sleep*, com o módulo hm10 reduzindo seu consumo para uma fração de miliampere e o Arduino reduzindo seu consumo em mais da metade.

Com base nos números obtidos é possível calcular a autonomia de uma unidade para alguns casos distintos. Tomando como base a capacidade de uma bateria AA comum (1500mAh) tem-se os seguintes cenários:

*Tabela 10: Autonomia de uma unidade remota*

| **Cenário** | **Autonomia (h)** | **Autonomia (dias)** |
|---|---|---|
| 100% transmissão | 52,6 | 2,2 |
| 20% transmissão / 80% *Sleep* | 114,4 | 4,8 |
| 10% transmissão / 90% *Sleep* | 134,0 | 5,6 |
| 100% *Sleep* | 161,8 | 6,7 |

Como se pode observar, mesmo no cenário mais propício, com ambos o módulo e o Arduino em hibernação durante todo o período, o sistema não atinge uma semana de autonomia de várias semanas. Além disso, é importante lembrar que os dados acima consideram apenas o Arduino e o módulo HM-10, de modo que a adição de sensores e aturadores ao conjunto tornariam a autonomia ainda menor. Dessa forma, fica evidente a necessidade de implementar melhorias no consumo da unidade, particularmente no Arduino.

Com o objetivo de reduzir o consumo de energia no Arduino removeram-se dois elementos desnecessários ao funcionamento do mesmo: o LED que indica que o microprocessador está ligado e o regulador de tensão. Após a remoção de ambos, novas medidas foram feitas para quantificar a redução no consumo. Constatou-se que sem os elementos desnecessários e em modo de hibernação o microprocessador apresentou consumo de 3,5mA, representando uma redução da ordem de 50% em relação às medidas anteriores. Apesar da redução significativa, fontes diversas indicam que a remoção dos componentes deveria levar a um consumo da ordem de 6 microamperes em modo de hibernação, o que não pôde ser replicado nos testes realizados.

## **4.3.** Alcance

Para que a implementação de uma rede doméstica seja factível é necessário que os módulos sejam capazes de se conectar quando instalados em cômodos distintos. Nessa situação as distâncias são baixas, usualmente abaixo dos 30m, porém a quantidade de obstáculos é grande, de modo que o sinal precisa atravessar paredes e mobilha para atingir seu destino. Com o objetivo de verificar se os módulos são capazes de atender tal requisito foram realizados testes para determinação de alcance da conexão. Nos testes verificou-se que o alcance dos módulos com visada (sem obstáculos) é de cerca de 30 metros, enquanto que o alcance dentro de um ambiente doméstico varia entre 3 e 6 metros, dependendo dos obstáculos. Tendo em vista o resultado pouco satisfatório julgou-se necessária a implementação de alguma forma de expandir o alcance, resultando em uma nova função.

### 4.3.1. Função Bounce

A função ***bounce*** tem como objetivo permitir que dois módulos distantes se comuniquem através de um módulo intermediário, expandindo o alcance efetivo de comunicação. Para atingir tal objetivo a função ***bounce*** utiliza as outras funções desenvolvidas neste projeto. De fato, a função simplesmente executa as funções ***connect, disconnect, send*** e ***getpkt*** em uma sequência pré-determinada, com a adição de atrasos.

Para que a retransmissão dos dados seja transparente ao usuário a função é executada automaticamente quando é verificado que o endereço do receptor (IDR) presente em uma mensagem recebida difere do ID do módulo. Quando chamada, a função recebe como argumento o pacote de dados que deve ser retransmitido. Tal pacote contém todas as informações necessárias para a retransmissão, como o endereço do emissor da mensagem, o endereço de quem deve receber a mensagem e a mensagem em si.

Assim que a função é executada a unidade intermediária encerra a conexão original e se conecta ao módulo identificado como receptor pela mensagem. Em seguida, a mensagem é enviada para o receptor e sua resposta é armazenada. Novamente, a unidade se desconecta, para então voltar a se conectar ao emissor da mensagem original. Por fim, a resposta da unidade de destino é transmitida ao emissor, encerrando a função. Essa sequência de ações pode ser observada no fluxograma abaixo, Figura 20. No fluxograma estão descritas as ações e as funções utilizadas para executá-las, entre parênteses. O código da função pode ser observado na Figura 19.

```
int bounce(packet foward)
{
  packet reply;

  disconnect();
  connect(foward.idr);
  send(foward.idr, foward.data);
  reply = getpkt();
  delay(500);
  disconnect();
  connect(foward.ids);
  delay(500);
  send(foward.ids, reply.data);
  return (0);
}
```

*Figura 19: Código da função **bounce***
*Fonte: Elaborada pelo autor*

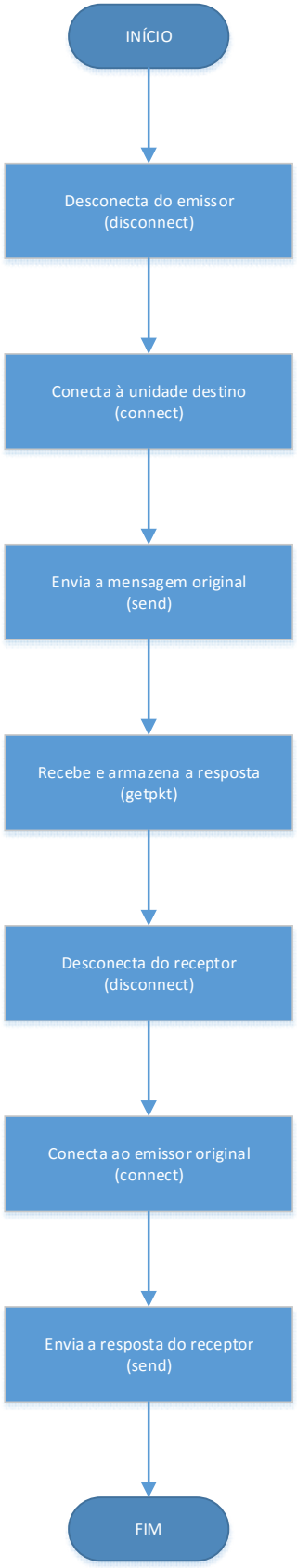


*Figura 20: Fluxograma da função **bounce***
*Fonte: Elaborada pelo autor*

## 4.4. Resultados Experimentais

Para verificar se o projeto cumpriu o objetivo de simplicidade de implementação ao qual se propôs, foi desenvolvido um cenário de teste representativo de uma situação típica de implementação na automação residencial. O cenário consiste de uma unidade atuando como sensor de humidade (Figura 21), para verificar se há chuva, uma unidade atuando como controlador de abertura de uma janela (representada por um LED, Figura 22) e a unidade central atuando como interface entre o usuário e ambas as unidades remotas (Figura 23).

A unidade com o sensor de humidade verifica regularmente a leitura do sensor para determinar se está chovendo ou não. Caso o sensor indique que está começando a chover a unidade se conecta ao controlador da janela e envia um comando para que ele a feche. Similarmente, caso o sensor indique que a chuva terminou um comando é enviado para que a janela se abra. Além disso o usuário pode, através da unidade central, se conectar ao sensor e requisitar a leitura atual, ou se conectar ao controlador e requisitar informação sobre o estado da janela, podendo inclusive comandar que a janela seja aberta ou fechada, independente da leitura do sensor. Neste caso, os critérios utilizados para se avaliar a simplicidade de implementação foram a quantidade de linhas de código e a quantidade de adaptações das funções necessárias para realizar as ações desejadas. Abaixo, nas Figuras 24 e 25, pode-se observar o código utilizado nas unidades de sensoriamento e de controle.

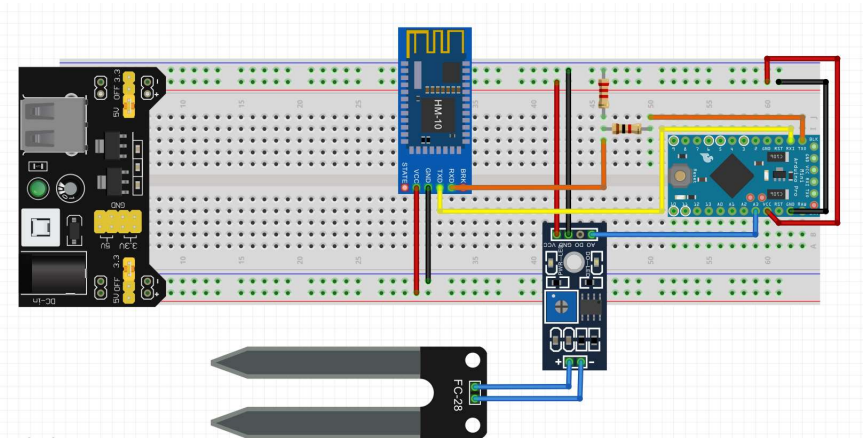


*Figura 21: Unidade de sensoriamento*
*Fonte: Elaborada pelo autor*

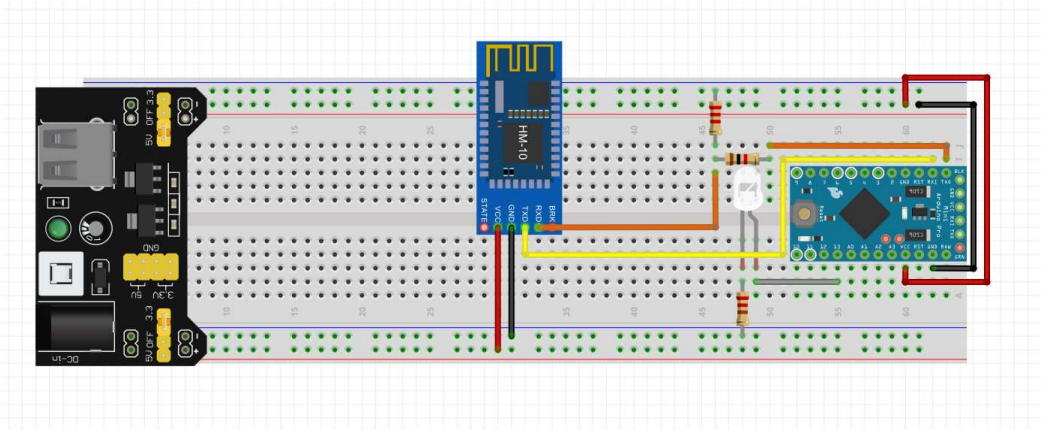


*Figura 22: Unidade de acionamento*
*Fonte: Elaborada pelo autor*

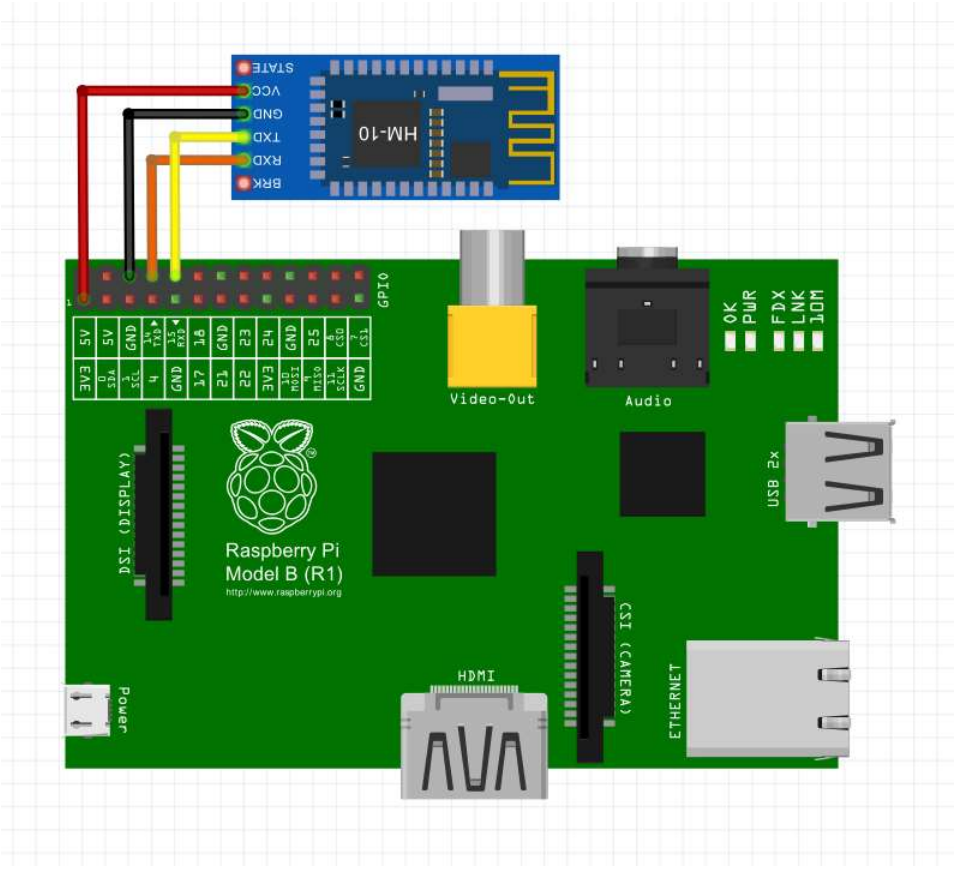


*Figura 23: Unidade central*
*Fonte: Elaborada pelo autor*

```
//leitura do sensor de agua
agua = analogRead(A3);
if (agua > 400)
{
  if (trigger == 0)
  {
    if (connect(yellow) == 0)
    {
      trigger = 1;
      delay(500);
      send(yellow, "fechajanela");
      delay(500);
      disconnect();
    }
  }
}
if (agua < 400)
{
  if (trigger == 1)
  {
    if (connect(yellow) == 0)
    {
      trigger = 0;
      delay(500);
      send(yellow, "abrejanela");
      delay(500);
      disconnect();
    }
  }
}

//loop para esperar solicitacao do master
if (mySerial.available())
{
  pkin = getpkt();
  if (pkin.data == "OK+CONN")
    conn = 1;
  if (pkin.data == "OK+LOST")
    disconnect();
  conn = 0;
  if (pkin.data == "agua")
  {
    delay(500);
    agua = analogRead(A3);
    stout = "";
    stout += agua;
    send(pkin.ids, stout);
  }
}
}
```

*Figura 24: Código implementado na unidade de sensoriamento*
*Fonte: Elaborada pelo autor*

```
//loop para esperar solicitacao do master
if (mySerial.available())
{
  pkin = getpkt();
  if (pkin.data == "OK+CONN")
    conn = 1;
  if (pkin.data == "OK+LOST")
    disconnect();
  conn = 0;
  if (pkin.data == "abrejanela")
  {
    digitalWrite(13, HIGH);
    estado = 1;
  }
  if (pkin.data == "fechajanela")
  {
    digitalWrite(13, LOW);
    estado = 0;
  }
  if (pkin.data == "estadojanela")
  {
    delay(500);
    if (estado == 1) send (pkin.ids, "Janela aberta");
    if (estado == 0) send (pkin.ids, "Janela fechada");
  }
 }
}
```

*Figura 25: Código implementado na unidade de controle*
*Fonte: Elaborada pelo autor*

Quanto ao número de linhas de código, pode ser observado na Figura 24 que a ação de se conectar a outra unidade, enviar o sinal de comando e se desconectar foi programada com oito linhas de código. Ainda na mesma Figura é possível observar que o ato de receber uma mensagem e tomar uma decisão baseada em seu conteúdo pode ser feita em duas linhas, por meio da função ***getpkt*** e de um condicional *"if"*. A Figura 25 apresenta mais um exemplo de recepção e análise de mensagem, novamente utilizando a função ***getpkt*** e condicionais "*if*".

Em termos de adaptação das funções, observa-se que ocorreram em duas formas no código apresentado. A primeira ocorre na forma de aplicação de atrasos entre a execução de mais de uma função em sequência. Isso é necessário para evitar que

comandos ou mensagens sejam enviados juntos, impedindo que sejam interpretados corretamente, como descrito na seção 3.3. O segundo caso foi a necessidade de converter um dado do tipo inteiro para um dado do tipo *String*, como pode-se observar na Figura 24. Isso foi necessário pois a função ***send*** requer que os dados a serem enviados estejam na forma de *String*, dada a forma como o pacote de dados foi definido e estruturado.

Também é interessante notar que ambos os programas tem a capacidade de atuar como intermediários na comunicação entre dois módulos que estejam em seu alcance. Como descrito na seção 4.3.1, a função ***bounce*** é executada dentro da função ***getpkt*** sempre que o ID de destino da mensagem difere do ID local. Dessa forma, caso a unidade central envie o comando “agua” para a unidade de controle com o ID do módulo de sensoriamento, a unidade de controle automaticamente irá se conectar ao módulo de sensoriamento, requisitar e receber a leitura do sensor e retransmitir essa leitura à unidade central.

# 5. CONCLUSÕES

Após a realização e avaliação do presente trabalho considera-se que o projeto cumpriu parcialmente com os objetivos propostos. As funções desenvolvidas se mostraram capazes de permitir a utilização da comunicação sem fio por meio do módulo *Bluetooth* HM-10 mesmo para usuários leigos no funcionamento do módulo. Os microcontroladores escolhidos foram capazes de executar suas funções de sensoriamento, atuação e comunicação com o usuário, além de apresentarem baixo custo de aquisição. Ademais, ambos são amplamente utilizados no segmento de eletrônica casual, apresentando sinergias com a proposta do projeto. Apesar disso, o consumo energético dos microcontroladores Arduino se mostrou demasiadamente alto, inviabilizando sua instalação sem fonte de alimentação externa para operar por várias semanas com baterias comuns de baixo custo. O módulo HM-10, utilizando o padrão *Bluetooth* 4.0, apresentou grande eficiência energética, especialmente quando o modo de hibernação é levado em conta, o que justifica seu custo relativamente elevado. Por outro lado, constatou-se que o alcance do módulo não é suficiente para utilização em ambientes domésticos, tornando necessária a utilização de adaptações, como a função proposta em 4.3.1.

Durante a realização do projeto encontrou-se grande dificuldade em trabalhar com o módulo *Bluetooth* HM-10. Tal dificuldade foi gerada pela falta de documentação adequada, dado que o manual do fabricante apresenta inúmeros erros de tradução e não aborda elementos críticos para a utilização do mesmo. Em diversos momentos foi necessário realizar testes para compreender o funcionamento de funções ou do módulo em si. Em especial, a relação do módulo com atrasos durante a troca de mensagens não é tratada na documentação, mas se mostrou absolutamente essencial para a implementação do módulo. Dessa forma, considera-se que os conhecimentos documentados neste texto servem de referência para futuros desenvolvimentos utilizando o módulo HM-10.

Com base nos resultados obtidos, dois pontos são recomendados para análise em trabalhos futuros. Primeiramente, recomenda-se que sejam avaliadas formas de

reduzir o consumo dos microcontroladores Arduino, de modo reduzir o consumo para a ordem de microamperes e proporcionar maior autonomia às unidades remotas. Também se sugere a análise de possíveis substitutos ao módulo HM-10, levando em conta a necessidade de maior alcance para implementação efetiva em uma rede distribuída, além dos benefícios que um componente com documentação adequada e completa proporcionaria a projetos futuros.

Todo o código desenvolvido na execução deste projeto pode ser encontrado em:

https://github.com/simoesusp/HomeControl/

# 6. REFERÊNCIAS

ARDUINO. WHAT is Arduino?. Disponível em: https://www.arduino.cc/en/Guide/Introduction>. Acesso em: 15 nov. 2015.

RASPBERRY PI FOUNDATION. WHAT is a Raspberry Pi?. Disponível em: <https://www.raspberrypi.org/help/what-is-a-raspberry-pi/>. Acesso em: 15 nov. 2015.

JNHUAMAO CORPORATION. Bluetooth 4.0 BLE module Datasheet. Disponível em: <https://www.seeedstudio.com/wiki/images/c/cd/Bluetooth4_en.pdf>. Acesso em: 15 nov. 2015.

BLUETOOTH SIG. COMMUNICATIONS Topology. Disponível em: <https://developer.bluetooth.org/TechnologyOverview/Pages/Topology.aspx>. Acesso em: 16 nov. 2015

BOLZANI, Caio Augustus Morais. **Desenvolvimento de um simulador de controle de dispositivos residenciais inteligentes: uma introdução aos sistemas domóticos.** 2004. 115 p. Tese (Mestrado) – Curso de Engenharia de Sistemas Eletrônicos, Departamento da Escola Politécnica da Usp, Usp, São Paulo, 2004.

SOHRABY, Kazem; MINOLI, Daniel; ZNATI, Taeib. **Wireless sensor networks: technology, protocols, and applications.** Hoboken, New Jersey: John Wiley & Sons, 2007. 307 p.

MICHEL, Hugo C. C; BRAGA, Anisio R; BRAGA, Carmela M. P; BRAGA, Laura C. **Caracterização do perfil de sistemas domóticos via redes de sensores sem-fio.** Programa de Pós-Graduação em Engenharia Elétrica, LVAS-EE/UFMG, Belo Horizonte, 2008.

REINISCH, Christian; KASTNER, Wolfgang; NEUGSCHWANDTNER, Georg; GRANZER, Wolfgang. **Wireless technologies in home and building automation.** In proc. 5th IEEE International conference on industrial informatics, 2007.

KASTNER, Wolfgang; NEUGSCHWANDTNER, Georg; SOUCEK, Stefan; NEWMAN, Michael H. **Communication systems for building automation and control.** Proceedings of the IEEE invited paper, 2005.

ZHENG, Xi; PERRY, Dewayne E.; JULIEN Christine. **BraceForce: A middleware to enable sensing integration in mobile applications for novice programmers.** The Center for Advanced Research in Software Engineering, University of Texas at Austin, Austin, 2014.

MATERNAGHAN, Claire. **The homer automation system.** Technical report, Department of Computing Science and Mathematics, University of Stirling, Stirling, 2010.

TANENBAUM, Andrew S.; WETHERALL, David J. **Computer Networks**. 5ª edição. Seattle: Pearson, 2010

# ANEXO A – Código das funções – Arduino

```
int connect(String MAC)
{
  mySerial.write("AT+ROLE1");
  delay(800);
  pkt = getpkt();
  int err;
  MAC = "AT+CON" + MAC;
  for (err = 0; err < 3; err++)
  {
    mySerial.print(MAC);
    pkt = getpkt();
    if (pkt.data == "OK+CONNA")
    {
      pkt.data.remove(0);
    }
    else return (-1);
    pkt = getpkt();
    if (pkt.data == "OK+CONN")
    {
      pkt.data.remove(0);
      return (1);
    }
  }
  pkt.data.remove(0);
  return (0);
}

int disconnect()
{
  mySerial.write("AT");
  check = getstr();
  if (check == "OK+LOST")
  {
    delay(500);
    mySerial.write("AT+ROLE0");
    check = getstr();
    return (0);
  }
  else if (check == "OK")
  {
    delay(500);
    check.remove(0);
    mySerial.write("AT+ROLE0");
    check = getstr();
    return (-1);
  }
  return(-2);
}
```

```
int send(String idr, String data)
{
  packet temp;
  String buff = "";
  char cs = checksum(data);
  buff = idlocal + idr + data + cs;
  for (int i = 0; i < 3; i++)
  {
    mySerial.print(buff);
    if (data == "CSOK") return (0);
    if (data == "CSFAIL") return (0);
    temp = getpkt();
    if (temp.data == "CSOK") return (0);
  }
  return (1);
}

String getstr()
{
  char c;
  String buff = "";
  int i = 0;
  while (!mySerial.available() && i < 1000 )
  {
    i++;
    delay(10);
  }
  if (i == 1000)
  {
    Serial.write("sem dados no serial\n");
    return ("0");
  }
  while (mySerial.available() > 0)
  {
    c = mySerial.read();
    buff += c;
    delay(10);
  }
  return (buff);
}
```

```
packet getpkt()
{
  char c;
  String buff = "";
  packet ret;
  int l;
  int i = 0;
  for (int z = 0; z < 3; z++)
  {
    while (!mySerial.available() && i < 1000 )
    {
      i++;
      delay(10);
    }
    if (i == 1000)
    {
      Serial.write("sem dados no serial\n");
      return (errorpkt);
    }
    while (mySerial.available() > 0)
    {
      c = mySerial.read();
      buff += c;
      delay(10);
    }
    Serial.print(buff);
    l = buff.length();
    if (l<24)
    {
      ret = errorpkt;
      ret.data = buff;
      return(ret);
    }
    ret.ids = buff.substring(0, 12);
    ret.idr = buff.substring(12, 24);
    ret.data = buff.substring(24, l - 1);
    ret.cs = buff.charAt(l - 1);
    if (ret.data == "CSOK") return (ret);
    if (ret.data == "CSFAIL") return (ret);
    if (ret.idr != idlocal)
    {
       l = bounce(ret);
       if (l == 0) return(errorpkt);
    }
    if (ret.cs == checksum(ret.data))
    {
      send(ret.ids, "CSOK");
      Serial.write("OK!\n"); //debug
      return (ret);
    }
    else send(ret.ids, "CSFAIL");
  }
  return (errorpkt);
}
```

```
int bounce(packet foward)
{
  packet reply;
  disconnect();
  connect(foward.idr);
  send(foward.idr, foward.data);
  reply = getpkt();
  delay(500);
  disconnect();
  connect(foward.ids);
  delay(500);
  send(foward.ids,reply.data);
  return(0);
}
```

# ANEXO B – Código das funções – Raspberry Pi

```
int connect(char* MAC)
{
    char check[30];
    char buff[20];
    int err;
    serialPuts (fd,"AT+ROLE1");
    getstr(check);
    delay(800);
    buff[0] = '\0';
    strcat(buff,"AT+CON");
    strcat(buff,MAC);
    puts(buff);

    for(err=0; err<3; err++)
    {
        serialPuts (fd,buff);
        getstr(check);

        if (strcmp(check,"OK+CONNA") == 0)
        {
        check[0] = 0;
        }
        else return(2);
        getstr(check);

        if ((strcmp(check,"OK+CONN") == 0))
        {
            check[0] = 0;
            return(0);
        }
    }
    check[0] = 0;
    return (1);

}
```

```
int disconnect()
{
    char check[30];
    serialPuts (fd,"AT");
    getstr(check);
    if ((strcmp(check,"OK+LOST") == 0))
    {
        check[0] = '\0';
        delay(500);
        serialPuts (fd,"AT+ROLE0");
        getstr(check);
        return(0);
    }
    else if ((strcmp(check,"OK") == 0))
    {
        check[0] = '\0';
        delay(500);
        serialPuts (fd,"AT+ROLE0");
        getstr(check);
        return(3);
    }
}
int send(char* idr, char* data)
{
    char* ids = "78A5048C378C";
    char buff[191];
         buff[0] = '\0';
    char cs[2];
        cs[0] = checksum(data);
        cs[1] = '\0';
    struct packet temp;
    int z;
    strcat(buff,ids);
    strcat(buff,idr);
    strcat(buff,data);
    strcat(buff,cs);
    for (z=0; z<3; z++)
    {
        serialPuts (fd,buff);
        if (strcmp(data,"CSOK") == 0) return(0);
        if (strcmp(data,"CSFAIL") == 0) return(0);
        temp = getpkt();
        if (strcmp(temp.data,"CSOK") == 0) return (0);
        if (strcmp(temp.data,"OK+LOST") == 0) return (0);
    }
    return(1);
}
```

```
char* getstr(char* buff)
{
    int i = 0;
    int j = 0;
    while(!serialDataAvail(fd) && i<1000)
    {
        i++;
        delay(10);
    }
    if (i== 1000)
    {
        buff = "erro";
        return(buff);
    }
    while(serialDataAvail(fd)>0)
    {
        delay(10);
        buff[j] = serialGetchar(fd);
        j++;
    }
    buff[j] = 0;
    return(buff);
}
```

```
struct packet getpkt( void )
{
    int i = 0;
    int j = 0;
    int z;
    char buff[193];
    struct packet ret;
    for (z=0; z<3; z++)
    {
        while(!serialDataAvail(fd) && i<1000)
        {
            i++;
            delay(10);
        }
        if (i== 1000)
        {
            return(errorpkt);
        }
        while(serialDataAvail(fd)>0)
        {
            delay(10);
            buff[j] = serialGetchar(fd);
            j++;
        }
        buff[j] = '\0';
        if (j<23)
        {
            ret = errorpkt;
            strcpy(ret.data,buff);
            return(ret);
        }
        strncpy(ret.ids,buff,12);
        ret.ids[12] = '\0';
        strncpy(ret.idr,buff+12,12);
        ret.idr[12] = '\0';
        strncpy(ret.data,buff+24,(j-25));
        ret.data[j-25] = '\0';
        ret.cs = buff[j-1];

        if (strcmp(ret.data,"CSOK") == 0) return(ret);
        if (strcmp(ret.data,"CSFAIL") == 0) return(ret);
        if (ret.cs == checksum(ret.data))
        {
            send(ret.ids,"CSOK");
            return(ret);
        }
    }
    return(errorpkt);
}
```