UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

ESCOLA DE ENGENHARIA

TRABALHO DE CONCLUSÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Integração de Cadeira de Rodas com Sistema de Automação Predial-Residencial

Autor: Douglas Dal'Bello

Orientador: Prof. Dr. Carlos Eduardo Pereira

Porto Alegre, novembro de 2014

Sumário

Sumário ii
Agradecimentos iv
Resumo v
Abstract vi
Lista de Figuras vii
Lista de Equações viii
Lista de Símbolos ix
Lista de Abreviaturas e Siglas x
1 Introdução 1
1.1 Motivação 1
1.2 Objetivo 2
1.3 Organização do Trabalho 2
2 Revisão Bibliográfica 3
2.1 Automação Residencial 3
2.2 Protocolo TCP/IP 4
2.3 Protocolo HTTP 5
3 Materiais e Métodos 6
3.1 Sistema de Automação Predial-Residencial da Empresa Homesystems 6
3.2 Opções de Arquitetura do Sistema 7
3.3 Análise das Opções de Implementação 7
3.3.1 Namimote 7
3.3.2 Arduino 8
3.3.3 Shield Ethernet 9
3.3.4 Shield Wifi 10
3.3.5 Opção de Hardware Adotada 10
3.4 Sensores 11
3.4.1 LM35 11
3.4.2 LDR 12
3.4.3 Conexão dos Sensores com o Arduino 13
3.5 Roteador 13
3.6 Instalação dos Equipamentos 14
4 Teste de verificação do sistema e criação de cenários de controle 15
4.1 Utilização do *Software ScapeWorks* 16
4.1.1 Conexão do Software ScapeWorks com o Systembox 18
4.1.2 Criação de Eventos 18

4.1.3 Criação de Variáveis 19
4.1.4 Upload das Configurações para o Systembox 20

5 Comunicação do Arduino com o *Systembox* 21

5.1 Comunicação Utilizando o *Shield Ethernet* 21

5.1.1 Inclusão de Bibliotecas 21
5.1.2 Leitura dos Sensores 22
5.1.3 Envio das Informações 22
5.1.4 Parametrização do Envio de Informações 24

5.2 Comunicação Utilizando o *Shield Wifi* 25

5.2.1 Inicialização do Shield Wifi 25
5.2.2 Parametrização e Envio das Informações 26

5.3 Resultado das Comunicações 26

5.3.1 Resultado da Comunicação via Shield Ethernet 26
5.3.2 Resultado da Comunicação via Shield Wifi 27

6 Conclusões e Trabalho Futuros 30

7 Referências 31

8 Apêndices 32

Agradecimentos

Aos meus pais, Gelson e Rosane, e à minha irmã, Bruna, pelo amor, carinho e incentivo em todos os momentos.

À Maísa, pelo amor e companheirismo.

Aos meus amigos, pela ajuda e pelo compartilhamento de conhecimentos ao longo do curso.

À empresa Homesystems, por fornecer o equipamento e o apoio necessário à realização deste trabalho.

Ao Professor Carlos Eduardo Pereira, ao Engenheiro Marcos Vizzotto e ao Mestrando Carlos Solon pelo apoio no desenvolvimento deste trabalho.

Resumo

Com o desenvolvimento da automação predial-residencial, conceitos como automação predial-residencial assistiva, voltada a prover acessibilidade a pessoas idosas ou com dificuldades motoras, ganham importância. Este trabalho tem como objetivo desenvolver a integração entre uma cadeira de rodas automatizada e um sistema de automação predial-residencial, através de sensores de temperatura e luminosidade localizados na cadeira e do desenvolvimento de uma comunicação capaz de enviar os dados coletados pelos sensores ao sistema de controle central. A comunicação entre os sistemas foi validada através do acionamento de relés correspondentes aos dispositivos atuadores em um sistema de automação residencial, como ar-condicionados e lâmpadas, usando os dados coletados no local do cadeirante, enviados por uma plataforma eletrônica prototipada.

Palavras-chave: automação predial-residencial assistiva – acessibilidade – sensores – controle central – comunicação entre sistemas

Abstract

With the development of home and building automation, concepts such as home automation assistive technology, designed to assist elderly and disabled people, become more important. This work intends to develop the integration between an automatic wheelchair and a home automation system, by using sensors located on the wheelchair and by developing a communication able to send data from the wheelchair to the control center. The communication between the systems was verified by turning on the relays related to the actuator devices in a home automation system, such as air conditioning and lights, using the data collected on the wheelchair, sent from an electronic prototyping platform.

Keywords: home automation assistive technology – accessibility – sensors – central control – communication between systems

Lista de Figuras

Figura 1: Diagrama de dispositivos de uma residência....3
Figura 2: Sistema de automação predial disponível na UFRGS....4
Figura 3: Arduino UNO 9
Figura 4: *Shield Ethernet* para Arduino....9
Figura 5: *Shield Wifi*....10
Figura 6: Esquemático do sensor LM35....11
Figura 7: Esquemático de montagem do LDR....12
Figura 8: Ilustrações do sensor LDR....12
Figura 9: Ilustração da montagem dos sensores com o Arduino....13
Figura 10: Roteador TP-LINK....13
Figura 11: Equipamentos (da esquerda para a direita): *Smarthub*, *Relay16* e *Systembox*....14
Figura 12: Esquemático de montagem dos equipamentos....15
Figura 13: Request em HTTP realizado via browser....17
Figura 14: Tela principal do *ScapeWorks*....18
Figura 15: Configuração da conexão....19
Figura 16: Criação de eventos....20
Figura 17: Exemplo de criação de variáveis....21
Figura 18: Inclusão de bibliotecas no *sketch*....22
Figura 19: Leitura dos sensores....23
Figura 20: Trecho de código da criação e inicialização da instância "*Client*"....23
Figura 21: Trecho de código da função "EnviaDados()"....24
Figura 22: Trecho de código correspondente à inicialização do *timer*....25
Figura 23: Trecho de código correpondente à atualização do *timer*....25
Figura 24: Trecho de código correspondente à inicialização do *shield Wifi*....26
Figura 25: Comunicação via *Shield Ethernet* vista pelo *Serial Monitor*....27
Figura 26: Comunicação via *Ethernet Shield* vista pelo *ScapeWorks*....27
Figura 27: Resultado da comunicação *wifi* visto pelo *Serial Monitor*....28
Figura 28: Resultado da comunicação *wireless* visto pelo *ScapeWorks* 29

Lista de Equações

Equação 1: Equação de transformação de unidades para temperatura.....12

Lista de Símbolos

Hz – *Hertz*

V – *Volt*

Mb/s – *Megabits per Second*

°C – *Graus Celsius*

mV – *Mili Volts*

µA – *Micro Ampère*

mA – *Mili Ampère*

Ω – *Ohm*

Mbps – *Megabits per Second*

VDC – *Volt Direct Current*

A – *Ampère*

Lista de Abreviaturas e Siglas

AVRGCC – *Average C*

DHCP – *Dynamic Host Configuration Protocol*

EEPROM – *Electrically Erasable Programmable Read-Only Memory*

GCAR – Grupo de Controle, Automação e Robótica

HSNET – *Homesystems Network*

HS-RS485 – *Homesystems Recommended Standard* - 485

HTTP – *Hypertext Transfer Protocol*

ID – Identificador

IDE – *Integraded Development Environment*

IEEE – *Institute of Electrical and Electronics Engineers*

INCT NAMITEC – Instituto Nacional de Ciência e Tecnologia de Sistemas Micro e Nanoeletrônicos

IP – *Internet Protocol*

LAN – *Local Area Network*

LDR – *Light Dependant Resistor*

LM35 – *Linear Monolithic* 35

MAC – *Media Access Control*

PPGEE – Programa de Pós-Graduação em Engenharia Elétrica

RFID – *Radio Frequency Identificator*

RSSF – Rede de Sensores Sem Fio

SD – *Secure Digital*

SPI – *Serial Peripheral Interface*

SRAM – *Static Random-Access Memory*

TCP – *Transmition Control Protocol*

UDP – *User Datagram Protocol*

USB – *Universal Serial Bus*

UFRGS – Universidade Federal do Rio Grande do Sul

WAN – *Wide Area Network*

WAP2 – *Wireless Application Protocol*

WEP – *Wired Equivalency Privacy*

1 Introdução

1.1 Motivação

As técnicas em automação residencial assistiva (PORTAL NACIONAL DE TECNOLOGIAS ASSISTIVAS, 2014) estão cada vez mais contextualizadas com os avanços sociais e tecnológicos. Um exemplo disso é a aplicação de sistemas de automação predial para o auxílio no cotidiano de pessoas com diversos níveis de deficiência física. Com a tecnologia atual, é possível fazer a automação de diversos mecanismos nas residências. É possível, por exemplo, controlar o horário em que as lâmpadas estarão ligadas, visando a um menor gasto de energia elétrica, controlar a temperatura, o fechamento de janelas e portas e pode-se fazer a implementação de um sistema biométrico de acesso à residência para promover maior segurança a quem reside no ambiente. Visando à segurança, é possível fazer o controle do uso de gás e de suas válvulas de contenção, assim como sensores de incêndio. Dessa forma, o nível de integração dos dispositivos automatizados possibilita um projeto destinado a prover acessibilidade nas residências de pessoas idosas ou com diversos tipos de deficiências motoras, através da abertura e fechamento automático de portas com a aproximação de um cadeirante, do controle automático de temperatura, segundo os valores de preferência do usuário, ou do ajuste de iluminação do ambiente voltado ao conforto do cadeirante.

Na Universidade Federal do Rio Grande do Sul, existem alguns projetos em desenvolvimento nesta área. Um dos trabalhos é voltado à elaboração e validação de uma arquitetura de sistema inteligente controlado passivamente, através de uma interface cérebro-computador (MENEZES, 2014). Com esta arquitetura, será feita a adaptação de um sistema inteligente de acordo com a emoção do usuário, aumentando seu rendimento numa dada tarefa ou no ambiente de trabalho em geral.

Outro trabalho realizado dentro do âmbito deste projeto, visa à localização de uma cadeira de rodas no ambiente doméstico, através da utilização de redes de sensores sem fio (RSSF) e de identificação por rádio frequência (RFID). O trabalho objetiva combinar as duas tecnologias, onde as etiquetas RFID são utilizadas como marcos de referência para a calibração automática das redes de sensores sem fio (MARQUES, 2014).

1.2 Objetivo

Com a intenção de aplicar o conceito de automação predial-residencial assistiva, o projeto consiste em desenvolver e implementar uma proposta de integração entre uma cadeira de rodas automatizada com um sistema de automação predial, através de sensores localizados na cadeira e de um controlador que forneça os dados para que o sistema possa acionar os dipositivos necessários para prover conforto ao usuário.

1.3 Organização do Trabalho

O presente trabalho foi organizado da seguinte forma: o capítulo 2 apresenta uma revisão bibliográfica, explicando conceitos de automação residencial assim como a lógica que comanda esse sistema, além da explicação do sistema de automação presente na Universidade Federal do Rio Grande do Sul e do protocolo TCP/IP. No capítulo 3, são expostos os materiais e métodos utilizados para o desenvolvimento do projeto, assim como as opções de implementação do trabalho. No capítulo 4, são apresentados os testes de verificação de montagem do equipamento e como é feita a utilização do *software* utilizado no trabalho. No capítulo 5, é demonstrado como foi feita a comunicação do *hardware* Arduino com o sistema de controle. O capítulo 6 é composto pelas conclusões e sugestões para a continuação do assunto trabalhado neste projeto.

2 Revisão Bibliográfica

Neste capítulo serão apresentados alguns conceitos importantes para a compreensão do desenvolvimento deste trabalho.

2.1 Automação Residencial

A automação residencial ou domótica pode ser definida no seu conceito mais amplo como a integração da tecnologia e dos serviços para uma melhor qualidade de vida no ambiente residencial e comercial. (SMART HOMES, 2014).

A domótica é geralmente baseada em um controle central, que gerencia as informações passadas pelos sensores distribuídos pela residência. Através dos sinais dos sensores, o controle faz o acionamento dos dispositivos a fim de customizar o ambiente de acordo com as preferências dos usuários.

A Figura 1 ilustra uma residência apta a ser controlada.

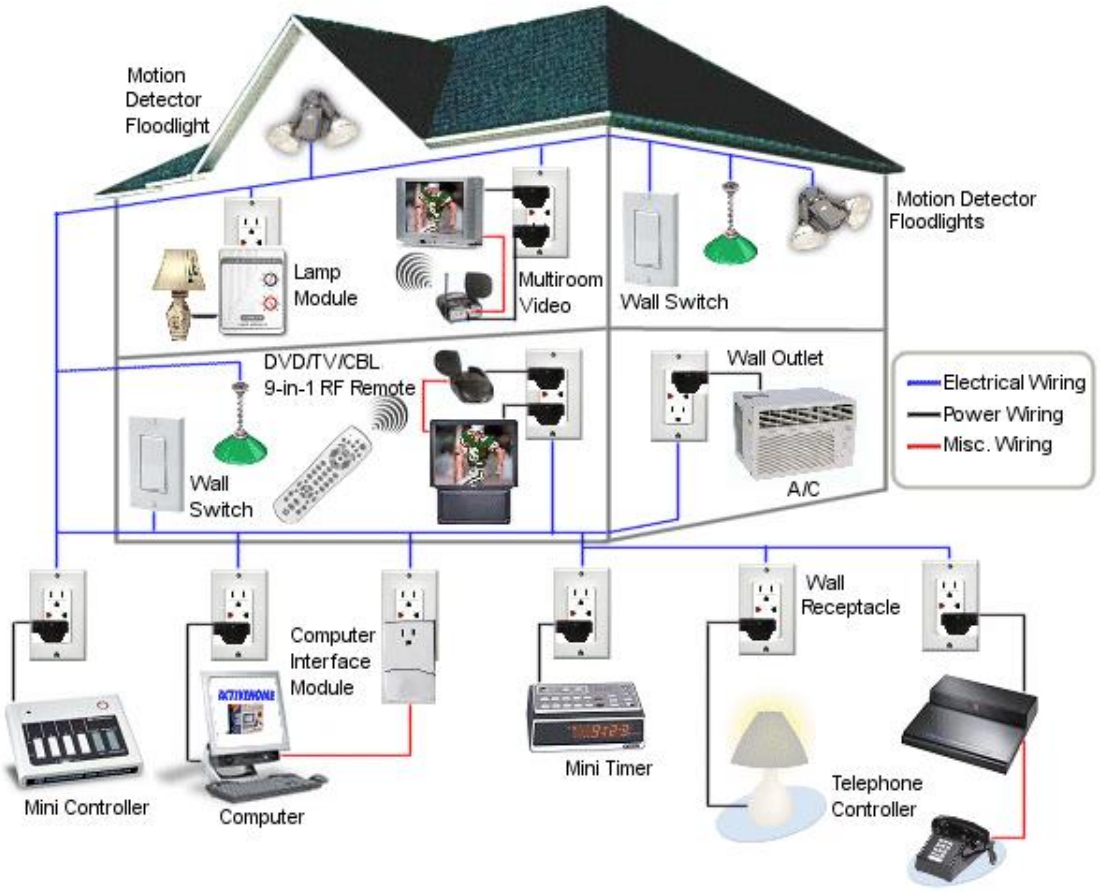

Figura 1: Diagrama de dispositivos de uma residência. Fonte: HACKNMOD (2014).

Como é possível observar na Figura 1, uma residência possui inúmeros dispositivos aptos a serem controlados. A automação residencial pode adaptar-se a cada usuário, podendo implementar-se um sistema tão simples ou tão complexo quanto se queira.

A instalação de sensores como de temperatura e luminosidade, por exemplo, são o ponto de partida para fazer o controle de temperatura e de gastos energéticos. Uma vez determinados os *set points* pelo controlador, os sinais de acionamento são enviados para os aparelhos, acionando ou não os dispositivos necessários para a correta configuração do ambiente.

Dessa forma, pode-se fazer a automação residencial de inúmeras maneiras. No caso deste trabalho, pode-se considerar que a preparação do ambiente será feita visando ao maior conforto do cadeirante, possibilitando uma maior mobilidade e acesso aos dispositivos da sua residência.

2.2 Protocolo TCP/IP

O procotolo TCP/IP foi inicialmente desenvolvido com a finalidade de atender algumas necessidades básicas de comunicação, como envio de arquivos, correio eletrônico e acesso remoto (GILBERT, 1995).

O protocolo é composto pelas seguintes camadas básicas:

a) IP – é um protocolo da camada de rede, sendo responsável por movimentar um pacote de dados de um nó para outro, realizando roteamento de mensagens e definindo as rotas a serem seguidas;
b) TCP – é um protocolo da camada de transporte, sendo orientado à conexão e responsável por verificar a correção do envio dos dados do cliente para o servidor;
c) *Sockets* – é um ponto final de um fluxo de comunicação entre processos, sendo formado pelo endereço de IP e por um número de porta vinculada ao protocolo TCP ou UDP.

O protocolo associa um número único para cada dispositivo da rede. O “número de IP” é um número de 4 *bytes* que, por convenção, é expresso pela conversão de cada *byte* em um número decimal (de 0 a 255) e separando os *bytes* com um ponto. Por exemplo, o IP do *Systembox*, neste trabalho, é 192.168.0.100.

2.3 Protocolo HTTP

Neste trabalho, foram utilizados comandos na linguagem HTTP (*Hypertext Transfer Protocol*). Esta linguagem é baseada em requisições e respostas entre um cliente e um servidor. Este protocolo é utilizado em navegadores de *internet* onde a comunicação é aberta entre o computador (cliente) e o servidor (*site* que se deseja acessar).

A linguagem HTTP utiliza caracteres de textos para formar o corpo da mensagem a ser enviada (Código ASCII) e define alguns métodos que são utilizados para fazer as requisições.

Dentre esses métodos pode-se citar:

a) GET – Faz a requisição de uma recurso específico. Esse método é utilizado para receber dados do servidor. No caso deste trabalho, o sistema de controle utilizado processa essa requisição como comandos a serem executados dentro do sistema de automação predial-residencial.
b) HEAD – Faz a requisição de uma resposta igual a realizada na função GET, mas sem o corpo da resposta.
c) POST – Faz a requisição ao servidor para acrescentar informações.

3 Materiais e Métodos

3.1 Sistema de Automação Predial-Residencial da Empresa Homesystems

A empresa Homesystems possui uma parceria com a Universidade Federal do Rio Grando do Sul (UFRGS) e disponibilizou um equipamento de automação residencial que pode ser ilustrado pela Figura 2.

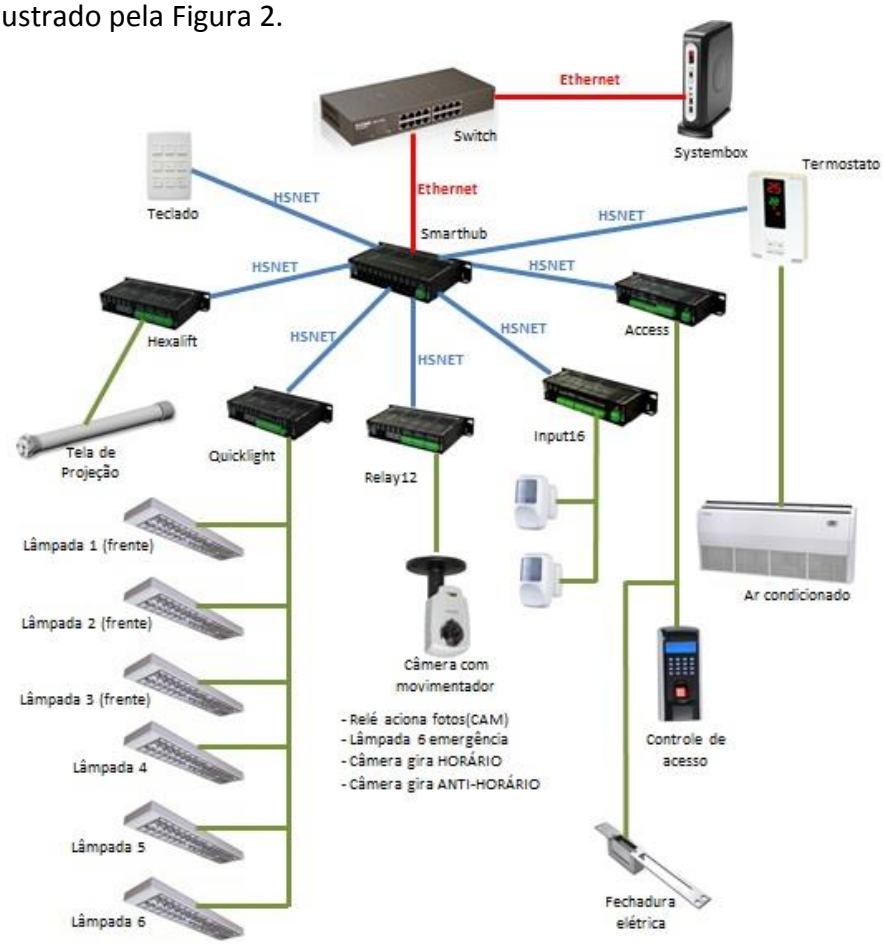

Figura 2: Sistema de automação predial disponível na UFRGS. Fonte: VIZZOTTO (2014).

A seguir, são descritos os componentes que constituem o *hardware* do sistema Homesystems:

a) *Systembox* – principal dispositivo de controle, é um computador embarcado com sistema operacional baseado em Linux, que armazena as instruções (cenários e eventos), recebe e envia ordens para os outros componentes a partir de comandos pré-definidos pelo usuário;

b) *Smarthub* – faz a conexão dos componentes do sistema. Converte o protocolo *Ethernet* em HS-RS485 (protocolo proprietário da empresa);

c) *Relay12* ou 16 – conjunto de relés que acionam os elementos atuadores;

d) *Input16* – conecta-se com os sensores de temperatura, movimento, entre outros, e possui 16 canais de entrada digital e 2 canais de entrada analógica.

Os componentes que constituem o *software* do sistema são descritos a seguir:

a) *Smarthub* – pode controlar os dispositivos, entretanto sua interface é limitada e não permite a criação de eventos, somente cenários. É responsável, também, por mostrar os endereços dos dispositivos na rede HSNET;

b) *ScapeWorks* – cria eventos e cenários, que são ações de controle executadas de acordo com uma lógica programável e permite o monitoramento das variáveis e do estado dos dispositivos em tempo real.

3.2 Opções de Arquitetura do Sistema

Para a realização do projeto proposto, algumas arquiteturas foram estudadas para a implementação na cadeira de rodas. Duas opções se destacaram: utilizar um sistema microcontrolado como nó da rede HSNET ou utilizar um dispositivo capaz de fazer a comunicação direta com o *Systembox* ou com o *Smarthub* através de *Sockets*. Para a primeira opção, é necessária a utilização de um dispositivo capaz de implementar o protocolo HSNET, forçando aquele a ser um sensor da rede. A segunda opção requer um *hardware* capaz de se comunicar via protocolo TCP/IP, o que remove a necessidade de transformar o dispositivo em um sensor da rede.

3.3 Análise das Opções de Implementação

Analisando as opções viáveis para a implementação dos objetivos, encontraram-se algumas soluções plausíveis, a seguir descritas.

3.3.1 Namimote

A primeira alternativa analisada foi o *hardware* Namimote, desenvolvido no âmbito do Instituto Nacional de Ciência e Tecnologia (INCT) NAMITEC, do qual pesquisadores do Grupo de Controle, Automação e Robótica (GCAR) da UFRGS participam. Esse equipamento é composto por uma placa de circuito impresso com sensores de temperatura, luminosidade e um acelerômetro, assim como um dispositivo de rádio que possibilita a comunicação *wireless*. A partir deste equipamento, seria feita a integração com o sistema de automação predial da Homesystems, disponível na UFRGS.

Esse sistema de automação, conforme descrito anteriormente, possui um módulo de controle central, que faz comunicação com os sensores do sistema já existentes e tem a capacidade de se comunicar com sensores externos, como o *hardware* Namimote. Dessa forma, essa escolha exige a implementação de uma comunicação do *hardware* com o sistema da Homesystems.

Essa comunicação pode ser feita, por exemplo, através da integração do Namimote diretamente com o sistema da Homesystems, ou seja, transformar o *hardware* em um dos sensores do sistema de automação. Para isso, é necessário que o controle central se comunique com o Namimote da mesma forma que ele se comunica com seus outros sensores. Isso ocorre através do protocolo de comunicação HS-NET, que é baseado no protocolo RS-485. Dessa forma, é necessário implementar o protocolo no *firmware* do Namimote.

Outra maneira cogitada para a implementação do projeto é a utilização de um *hardware* capaz de fazer uma comunicação via cabo ou *wireless* com o dispositivo da Homesystems através de *sockets* com o protocolo TCP/IP. Dessa forma, não é necessário transformar o *hardware* em um sensor da rede, bastando apenas a sua comunicação diretamente com o sistema de controle.

3.3.2 Arduino

Baseado na comunicação TCP/IP, considerou-se o *hardware* Arduino, muito utilizado atualmente. Este *hardware* propicia facilidade e agilidade na sua utilização, visto que tanto a sua parte de *software* quanto a parte de *hardware* são *open source*, ou seja, os códigos de programação, os esquemáticos de montagem e de *design* são abertos para todos os usuários.

O Arduino é compatível com diversos *shields*, que são responsáveis por funções como comunicação *wired* e *wireless*, acionamento de motores, comuicação via tecnologia GSM, entre outros.

A descrição do Arduino UNO é feita a seguir:

a) Microcontrolador – é construído de forma a integrar diversos componentes em um único circuito integrado, evitando, assim, a necessidade de adicionar componentes externos ao microcontrolador. No Arduino UNO é utilizado o processador ATMega328 que possui 32 *Kbytes* de memória *flash*, 2048 *Bytes* de SRAM e 1024 *Bytes* de EEPROM;

b) Periféricos – o restante do Arduino UNO é composto pelas entradas de alimentação e USB e pelos componentes e circuitos que permitem a gravação e utilização do microcontrolador;

c) Possui 14 entradas ou saídas digitais, das quais 6 podem ser usadas como saídas de PWM, 6 entradas analógicas e um cristal de 16MHz;

d) O *software* para desenvolvimento, chamado de IDE (*Integrated Development Environment)*, permite a programação em diversas linguagens como *Assembly*, linguagem C/AVRGCC e linguagem *Wiring* C/C++.

Figura 3: Arduino UNO. Fonte: ARDUINO (2014b).

3.3.3 Shield Ethernet

Este *shield* é responsável por fazer a comunicação do Arduino com a *internet*. Ele é baseado no *chip* Wiznet W5100, que fornece um IP capaz de realizar comunicação via TCP ou UDP. Ele suporta até 4 conexões de *socket* simultâneas.

A seguir seguem dados relevantes:

a) Comunicação IEEE 802.3af;

b) Tensão de entrada com *range* de 36V a 57V;

c) Tensão de saída de 9V;

d) 1500V de isolação da entrada para a saída.

Os LEDs presentes na placa possuem o seguinte significado:

a) ON (*Power*): indica que a placa está ligada;

b) LINK: indica a presença de uma rede e pisca quando transmite ou recebe dados;

c) 100M: indica a presença de uma conexão de 100Mb/s;

d) RX: pisca quando o *shield* recebe dados;

e) TX: pisca quando o *shield* envia dados.

Figura 4: *Shield Ethernet* para Arduino. Fonte: ARDUINO (2014a).

3.3.4 Shield Wifi

Este *shield* permite a conexão do Arduino com a *internet* através da comunicação *wireless*. Ele é baseado no *chip* AT32UC3, fabricado pela Atmel, que fornece um IP capaz de se comunicar através do protocolo TCP e UDP.

A seguir são expostas algumas características:

a) Conexão via 802.11b/g;

b) Tensão operacional de 5V (fornecida pelo Arduino);

c) Aceita tipos de encriptação WEP e WPA2 *Personal*;

d) Se conecta com o Arduino através da porta SPI;

e) Possui um *micro* SD-*Slot* integrado;

f) Conexão *mini*-USB para fazer a atualização do *firmware*.

A Figura 5 representa as partes frontral e traseira do *shield wifi*.

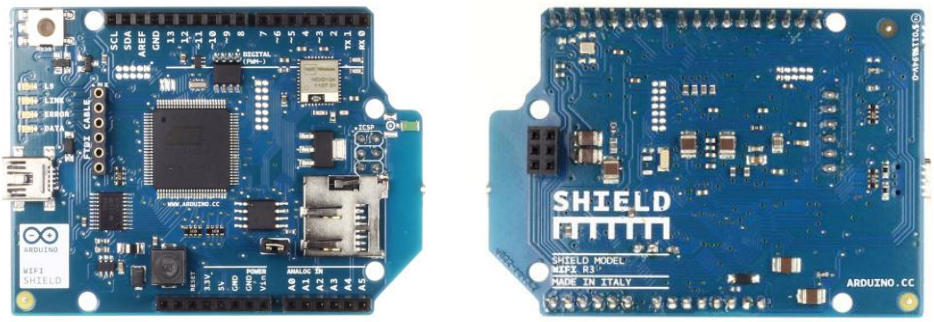

Figura 5: *Shield Wifi.* Fonte: ARDUINO (2014c).

3.3.5 Opção de Hardware *Adotada*

Devido à facilidade de uso e de bibliotecas disponíveis para programação, optou-se pela escolha do Arduino UNO como o *hardware* para a implementação do sistema. A ideia será, portanto, fazer a medição de temperatura e luminosidade pelas portas analógicas do Arduino e mandar os dados medidos e os *setpoints* necessários para o controle via TCP/IP para o sistema da Homesystems. Por fim, o *Systembox* é responsável por executar os eventos de controle pré-estabelecidos e acionar os relés correspondentes.

3.4 Sensores

Para fazer a medição de temperatura e luminosidade no local do cadeirante, escolheu-se, respectivamente, os sensores LM35 e LDR. A escolha foi feita devido à facilidade de compra e ao baixo custo dos componentes, além do fato de possuírem uma precisão adequada para este tipo de aplicação.

3.4.1 LM35

O sensor LM35 é um circuito integrado de precisão para medição de temperatura, com tensão de saída linearmente proporcional à temperatura em graus Celsius. A seguir, são descritas algumas características importantes do sensor:

a) Calibrado diretamente em ˚C (centígrados);
b) Operação linear com fator de escala de + 10mV/˚C;
c) 0.5˚C de precisão assegurada (quando utilizado na temperatura de 25˚C);
d) Apto para temperaturas de -55˚C até 150˚C;
e) Opera de 4 a 30V;
f) Drena menos de 60µA de corrente;
g) Baixo autoaquecimento: 0.08˚C ao ar livre;
h) Baixa impedância de saída: 0.1Ω para carga de 1mA.

A Figura 6 mostra opções de *package* para a utilização do sensor LM35, assim como a opção adotada que, neste projeto, foi a *Plastic Package TO-02 (LP)*.

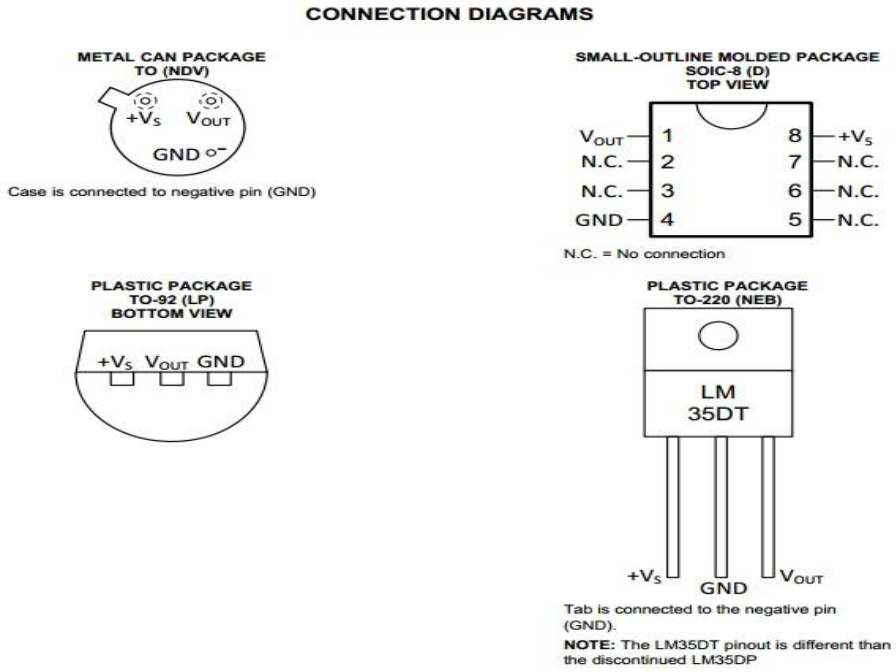

Figura 6: Esquemático do sensor LM35. Fonte: TEXAS INSTRUMENTS INCORPORATED (1999).

Para a utilização do LM35 no Arduino, é necessário um fator de correção de 0.488. A leitura feita na porta analógica do Arduino correspondente ao LM35 retorna um valor inteiro entre 0 e 1023. Como o sensor possui sensibilidade de 10mV/˚C, e a tensão que o Arduino fornece para alimentação é de 5V, a expressão do valor de temperatura em ˚C é dada por:

$$temperatura = \left((valor\ lido) * \left(\frac{5V}{1023bits}\right) * \frac{1}{\frac{10\text{mV}}{^{\circ}\text{C}}} \right) = valor\ lido * 0.48875 \qquad (1)$$

3.4.2 LDR

O sensor LDR (*Light Dependant Resistor*) varia sua resistência de acordo com a incidência de luz sobre ele. Este sensor é indicado em aplicações que medem variações de luz, como a mudança do dia para a noite. Dessa forma, é possível observar variações significativas na iluminação da cadeira de rodas, acionando as lâmpadas necessárias para iluminar adequadamente o ambiente para o usuário.

As Figuras 7 e 8 ilustram o sensor utilizado.

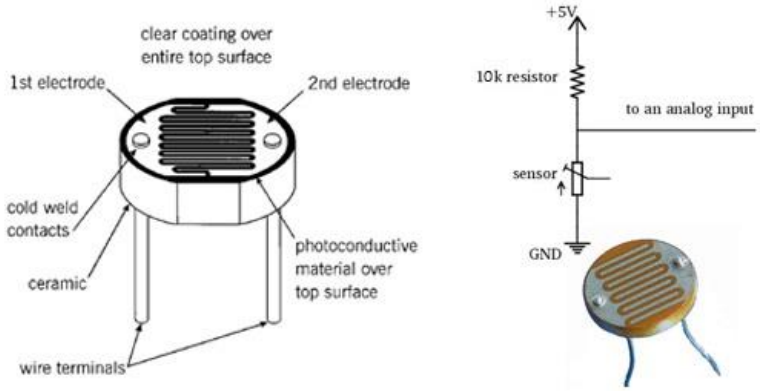

Figura 7: Esquemático de montagem do LDR com um circuito de linearização. Fonte: GISPERT (2012).

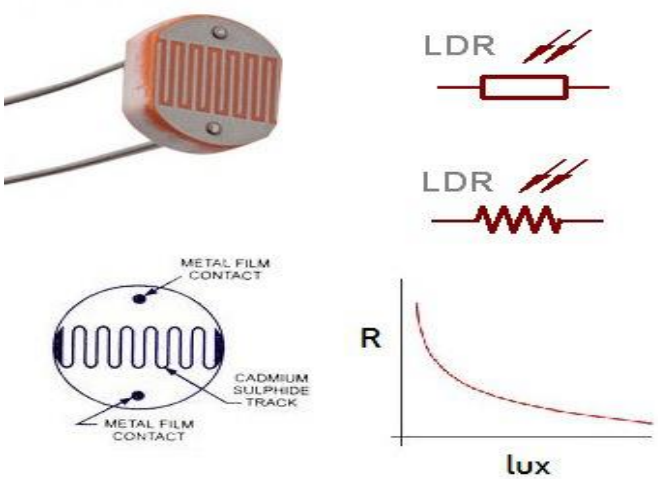

Figura 8: Ilustrações do sensor LDR com a curva de operação. Fonte: EDUSOFT (2014).

Visto que o LDR varia sua resistência de acordo com a luminosidade incidente sobre ele, a medida fornecida pelo Arduino correspondente à leitura deste sensor foi feita de forma a aplicar um divisor de tensão com um resistor de 10kΩ, conforme a Figura 7. O valor medido, por tanto, é o resultado desta divisão de tensão.

3.4.3 Conexão dos Sensores com o Arduino

Para a ligação dos sensores com o Arduino, utilizou-se uma *protoboard*. A conexão dos sensores com o Arduino está ilustrada na Figura 9.

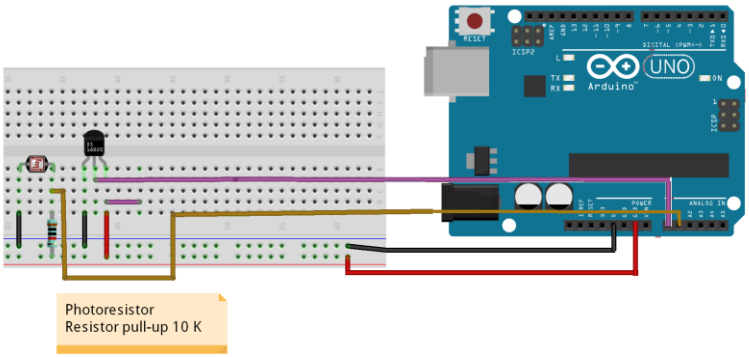

Figura 9: Ilustração da montagem dos sensores com o Arduino. Fonte: Autor.

3.5 Roteador

Para fazer a comunicação entre o sistema e o Arduino, adquiriu-se um roteador TP-LINK TL-WR740. A seguir, seguem alguns dados importantes do roteador:

a) Possui 4 portas LAN e uma porta WAN de 10/100 Mbps;

b) Fonte de alimentação externa de 9VDC com 0.6A;

c) Padrão de comunicação *wireless* IEEE 802.11n, 802.11g e 802.11b.

Figura 10: Roteador TP-LINK. Fonte: TP-LINK (2014).

Para configurar o roteador, fizeram-se necessárias algumas restrições: a distribuição de IPs através do método DHCP (*Dynamic Host Configuration Protocol*), precisa garantir que a faixa de valores abrangerá o IP fixo do *Systembox* e do *Smarthub*: 192.168.0.100 e 192.168.0.200, respectivamente. Para isso, na hora da instalação do roteador, configurou-se a distribuição de IPs de forma que o primeiro valor começasse em 192.168.0.55, terminando em 192.168.0.200.

3.6 Instalação dos Equipamentos

Conforme citado anteriormente, a UFRGS possui um *kit* com um sistema de automação fornecido pela empresa Homesystems. Esse sistema, porém, está instalado em uma sala de aula do PPGEE-UFRGS, onde aulas de mestrado e doutorado são ministradas diariamente, o que inviabiliza a execução dos testes necessários para este trabalho. Dessa forma, foi solicitado junto à empresa um novo *kit*, para que os testes fossem feitos sem a necessidade de utilizar a sala de aula.

A Homesystems prontamente forneceu um novo equipamento, conforme Figura 11.

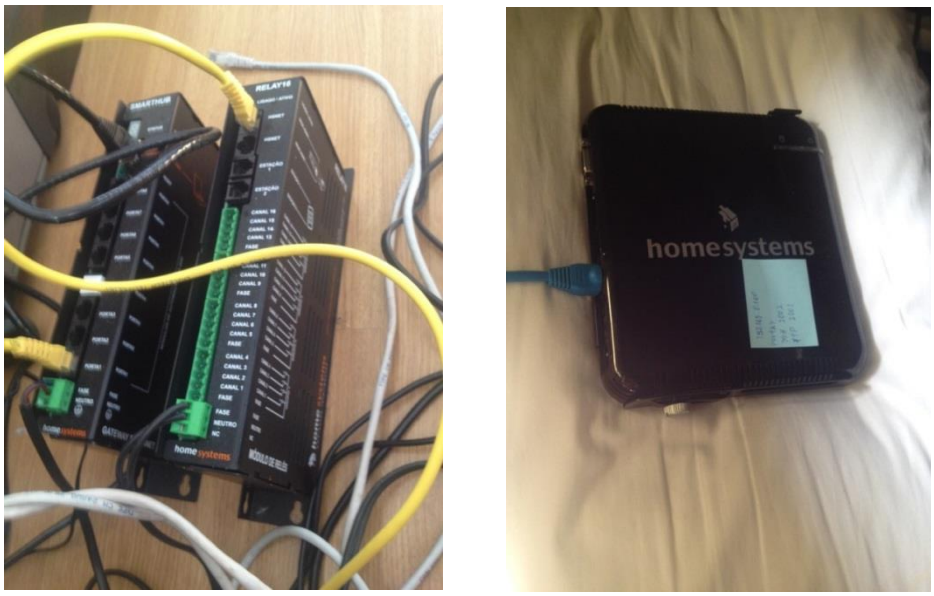

Figura 11: Equipamentos (da esquerda para a direita): *Smarthub*, *Relay16* e *Systembox*. Fonte: Autor.

A instalação dos equipamentos é relativamente simples: primeiro, conecta-se o *Smarthub* no *Relay16*. Em seguida, liga-se o *Smarthub* no roteador. Por fim, conecta-se o *Systembox* no roteador. Todas as ligações são feitas utilizando cabo de rede e cada equipamento é alimentado com uma tensão de 127V. O esquemático da Figura 12 ilustra o procedimento de montagem.

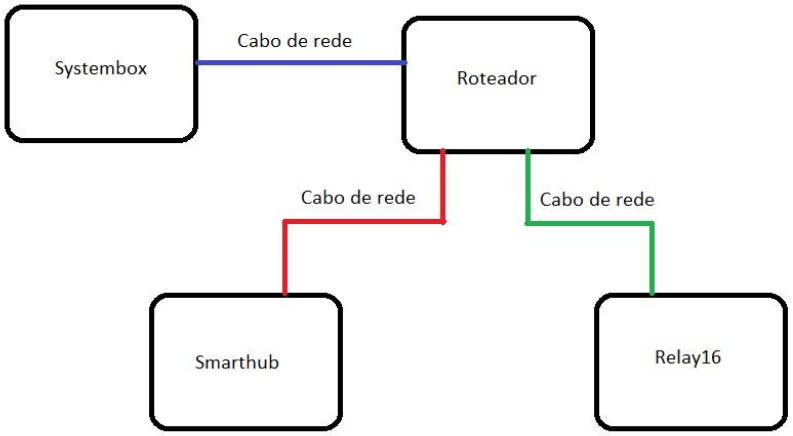

Figura 12: Esquemático de montagem dos equipamentos. Fonte: Autor.

4 Teste de verificação do sistema e criação de cenários de controle

Antes de programar e utilizar o Arduino, fez-se relevante testar o sistema para verificar se a instalação foi feita corretamente e se o *Systembox* está apto a receber as informações enviadas.

O *Systembox* aceita alguns tipos de comunicações, entre elas, *requests* em HTTP (*Hypertext Transfer Protocol*). Alguns comandos pré-estabelecidos são utilizados para enviar e receber dados, os quais são descritos a seguir:

1) *Set unit* → Atribui valor para uma *unit*:
   http://IPSYSTEMBOX/monitor/monitor.cgi?ref_page=set&unit=UNIT&newvalue=VALOR
   - Exemplo: http://192.168.0.208/monitor/monitor.cgi?ref_page=set&unit=182&newvalue=0
   - Possível resposta: 182-1=0* → *Unit* estava com valor 1 e agora está com o valor zero.
2) *Get unit* → Obtém o valor de uma *unit*:
   http://IP_SYSTEMBOX/monitor/monitor.cgi?ref_page=get&unit=UNIT
   - Exemplo: http://192.168.0.208/monitor/monitor.cgi?ref_page=get&unit=182
   - Possível resposta: 182-1(0)* → A *unit* 182 está com o valor 1 e temporização igual a zero.
3) *Get Variable* → Obtém o valor de uma variável:
   http://IP_SYSTEMBOX/monitor/monitor.cgi?ref_page=getvar&unit=ID_VARIAVEL
   - Exemplo: http://192.168.0.208/monitor/monitor.cgi?ref_page=getvar&unit=2
   - Possível resposta: 34 → A variável com ID 2 está com o valor 34.
4) *Set variable* → Define o valor de uma variável:
   http://IP_SYSTEMBOX/monitor/monitor.cgi?ref_page=setvar&unit=ID_VARIAVEL&newvalue=VALOR&newflags=VALOR_FLAG
   - Exemplo: http://192.168.0.208/monitor/monitor.cgi?ref_page=setvar&unit=2&newvalue=30&newflags=0

- Possível resposta: 40 = 30 → A variável identificada com o ID 2, que estava com o valor 40, agora está com o valor 30. *Newflags* é um atributo para variáveis de hora.

As *units* passadas como atributos nos comandos são valores atribuídos internamente pelo sistema para identificar para qual equipamento a mensagem será enviada. No caso deste trabalho, por exemplo, o *Relay16* é a *unit* 104. Ele possui 16 saídas digitais, cada uma correspondendo a uma *unit*.

A primeira saída digital do *Relay16* possui o valor de *unit* 182; a segunda, *unit* 183, e assim sucessivamente. Ou seja, se utilizarmos o comando para setar uma *unit* e passarmos como valor o número 1, estamos ligando o relé daquela saída. Pode-se conectar uma lâmpada ou fazer um acionamento de um ventilador ou ar-condicionado, por exemplo. No caso deste trabalho, não será conectado nenhum dispositivo na parte de testes, pois o foco será enviar valores para configurar as variáveis internas do sistema.

Após conhecer os possíveis comandos de comunicação com o *Systembox*, fez-se um teste para verificar o funcionamento do sistema. Foi utilizado um *notebook* Dell Vostro 5740, conectado à rede *wifi* fornecida pelo roteador TP-LINK descrito anteriormente. Utilizou-se, também, o *browser* Google Chrome para mandar o *request*. A Figura 13 ilustra o teste.

Figura 13: *Request* em HTTP realizado via *browser*. Fonte: Autor.

Conforme esperado, o comando desligou o relé correspondente a *unit* 182. A resposta 182-1=0* comprova que a comunicação está funcionando. Além disso, é possível verificar em tempo real o *status* das saídas digitais e das variáveis através do *software ScapeWorks*, o que será demonstrado a seguir.

4.1 Utilização do *Software ScapeWorks*

O sistema de automação utilizado neste trabalho possui um *software* de gerenciamento capaz de verificar e definir variáveis e acionar saídas digitais. O *ScapeWorks* foi fornecido, também, pela empresa Homesystems e está demonstrado na Figura 14.

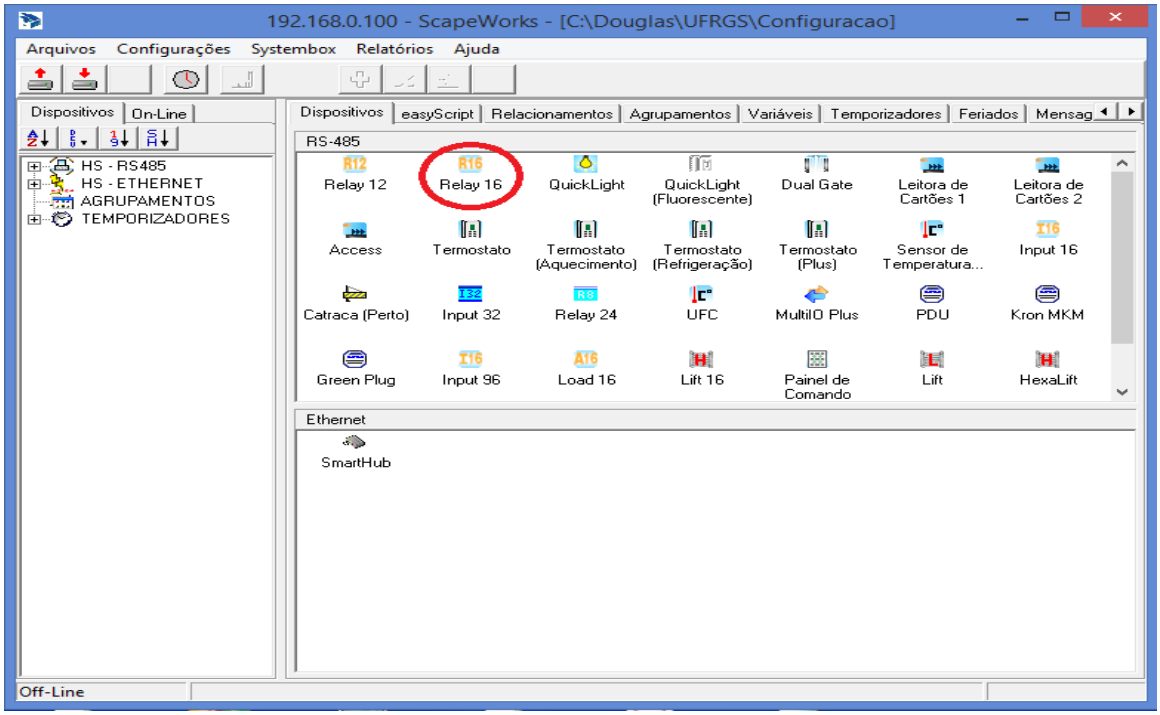

Figura 14: Tela principal do *ScapeWorks*. Fonte: Autor.

Nesta tela, é possível observar algumas funções importantes:

a) Na parte esquerda da tela, há duas abas correspondentes a "dispositivos" e "*on-line*". "Dispositivos" é onde se encontram todos os equipamentos selecionados pelo usuário. Conforme se observa na parte central da tela, há inúmeros equipamentos selecionáveis, como o *Relay16*;

b) Ainda na parte esquerda, há 2 botões responsáveis por fazer *download* das configurações presentes no *Systembox* e também pelo *upload* das modificações feitas pelo usuário;

c) O *software* organiza os dispositivos da seguinte forma: "HSNET" corresponde aos equipamentos que se comunicam através do protocolo HSNET como o *Relay16*. "*ETHERNET*" corresponde aos dispositivos que se comunicam com o *Systembox* através de um cabo de rede (*ethernet*), como o *Smarthub*. É possível ligar mais de um *Smarthub* na rede, assim como é possível ligar mais de um *Relay16* em cada *Smarthub*. "AGRUPAMENTOS" corresponde ao agrupamento de dispositivos para facilitar o uso do *software*. "TEMPORIZADORES" mostra todos os *timers* que estão sendo utilizados no *ScapeWorks*;

d) Na aba "*On-line*", é feito o monitoramento em tempo real das variáveis e dispositivos conectados à rede, podendo-se modificar os seus valores a qualquer momento;

e) Na parte direita da tela, é possível selecionar abas como: “dispositivos”, “*easyScript”*, “variáveis”, entre outros. A aba “*easyScript*“ corresponde à criação de eventos, que utilizam variáveis para acionar determinados dispositivos.

4.1.1 Conexão do Software ScapeWorks *com o* Systembox

Para conectar o *ScapeWorks* com o *Systembox,* é necessário que o computador esteja conectado na mesma rede que o *Systembox*, ou seja, através do roteador TP-LINK.

Para isso, configura-se o *software* da maneira demonstrada na Figura 15.

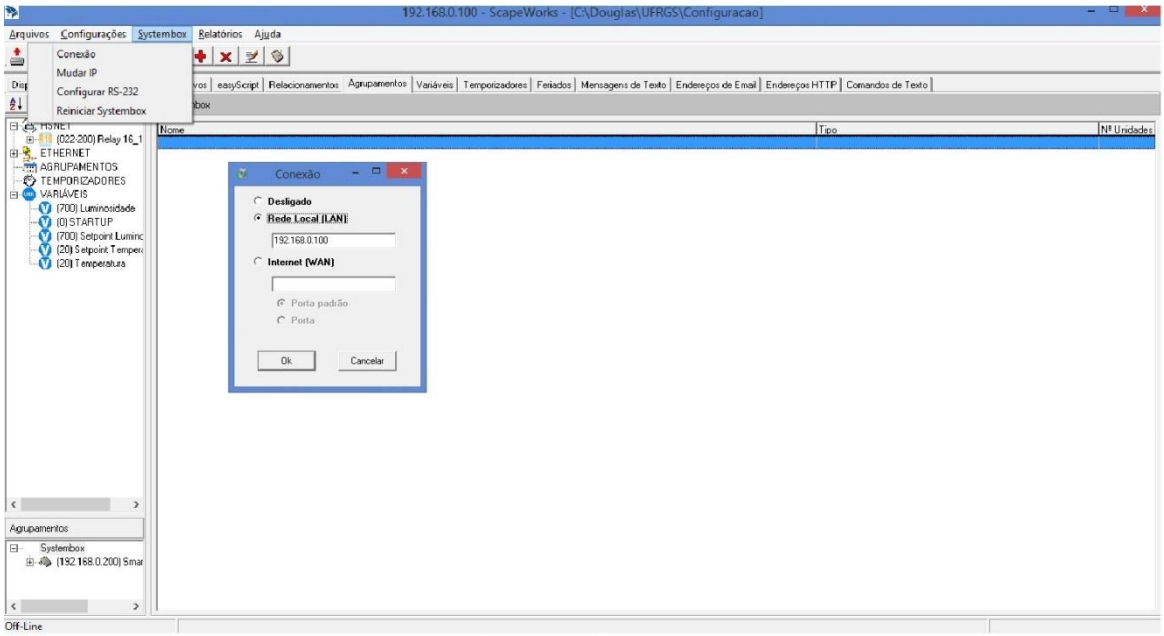

Figura 15: Configuração da conexão. Fonte: Autor.

Na opção conexão, é possível escolher entre rede local e *internet*. Neste trabalho, a comunicação se dá através da rede local e o IP do *Systembox* é 192.168.0.100.

Uma vez conectado ao *Systembox*, a aba “On-Line” acende uma luz verde correspondente à conexão bem sucedida, tornando-se possível monitorar e controlar o sistema em tempo real.

4.1.2 Criação de Eventos

A criação de eventos é feita através da aba *easyScript*. A Figura 16 ilustra o procedimento.

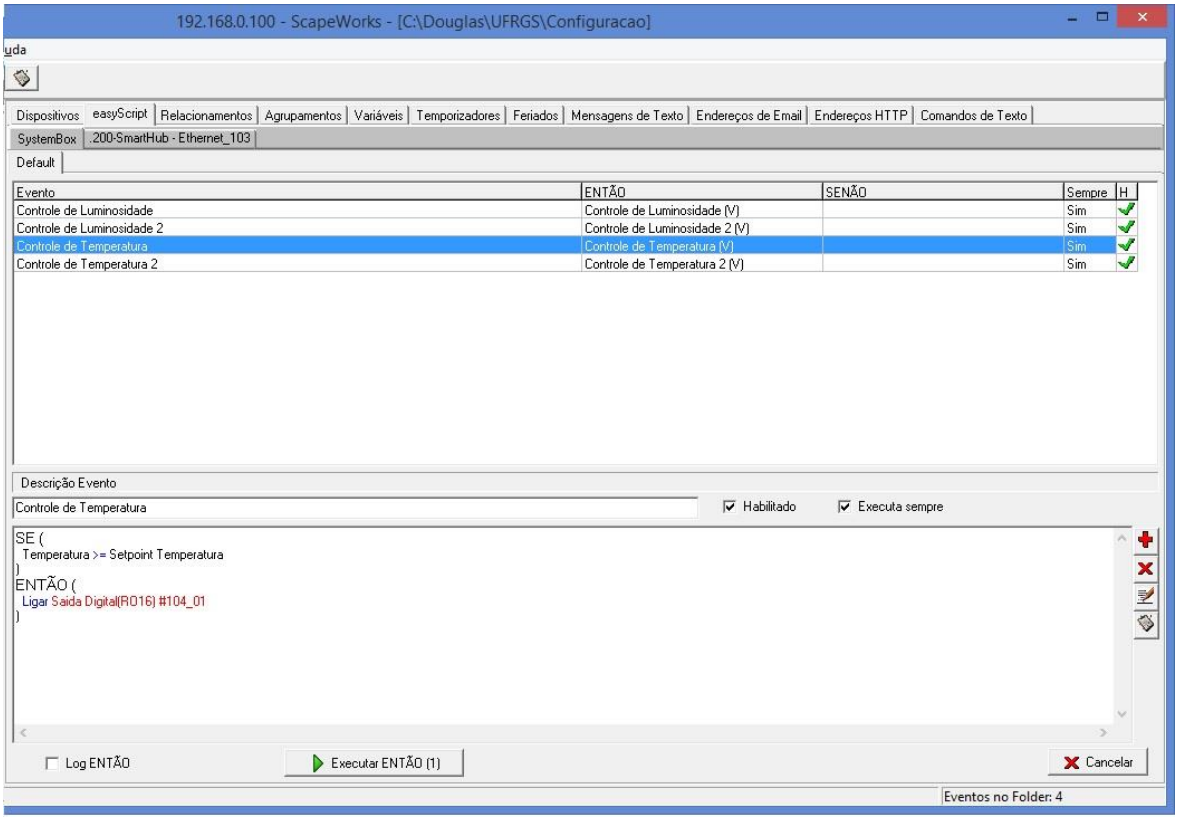

Figura 16: Criação de eventos. Fonte: Autor.

É possível escolher variáveis ou saídas digitais para controlar outras saídas do sistema. A programação é feita através de um *script* onde são definidas duas condições: A condição "Se", onde é escolhida a variável que irá condicionar uma ação; e a condição "Então" escolhe-se o resultado desta ação.

Na Figura 16, observa-se que as variáveis Temperatura e *Setpoint* Temperatura fornecem a condição necessária para o acionamento ou não da saída digital do *Relay16*. Dessa forma, pode-se criar diversos tipos de eventos a partir de variáveis do sistema.

4.1.3 Criação de Variáveis

Para criar variáveis, é necessário selecionar a aba "variáveis" e clicar no botão "+" vermelho, no topo da tela. A Figura 18 ilustra o processo.

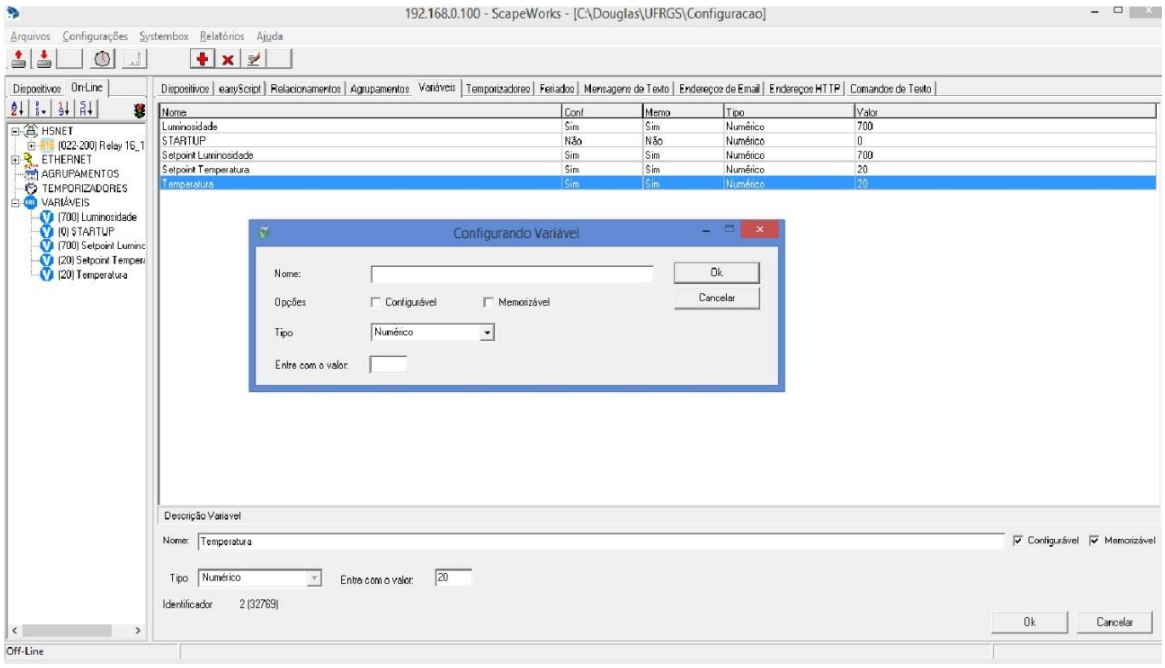

Figura 17: Exemplo de criação de variáveis. Fonte: Autor.

Nota-se que, após clicar no botão “+”, abre-se uma janela para configuração da variável a ser criada, sendo possível definir se ela será configurável e memorizável, assim como o seu tipo (numérica, *string*, numérica longa e horário) e o valor inicial que se deseja para a variável.

Ao fundo da Figura 17, é possível observar a variável Temperatura. Na parte debaixo da tela, é mostrado o identificador de cada variável. Este valor é o mesmo ID que será passado pelos comandos *set variable* no *request* em HTTP.

Na parte esquerda da Figura 17, observa-se a aba “*On-Line*” selecionada, onde pode-se monitorar os valores das variáveis. Os números entre parênteses à esquerda dos nomes das variáveis correspondem ao seu valor naquele momento.

4.1.4 Upload das Configurações para o Systembox

Após criar as variáveis e os cenários de controles, é necessário fazer o *upload* das configurações para o *Systembox*. Depois de conectar o *ScapeWorks* com o *Systembox*, basta clicar no botão “*upload*” e as configurações são enviadas.

5 Comunicação do Arduino com o *Systembox*

5.1 Comunicação Utilizando o *Shield Ethernet*

5.1.1 Inclusão de Bibliotecas

No início do desenvolvimento deste trabalho, se dispunha somente do *shield Ethernet*. Como será explicado, os comandos de envio de mensagem com o *shield Ethernet* são os mesmo utilizados com o *shield Wifi.* A principal diferença entre a utilização dos dois *shields* é o método de conexão com o servidor. O *shield Wifi* se conecta à rede *wireless* através do SSID e do *Password* da rede ao passo que o *shield Ethernet* estabelece a conexão enviando como parâmetro o seu número de indentificação (MAC). Dessa forma, a comunicação foi feita, primeiramente, com o *shield Ethernet* e, posteriormente, se fez a aquisição e a programação da comunicação com o *shield Wifi.*

O ambiente de programação do Arduino é chamado de IDE (*Integrated Development Environment*) e foi programado em *Java*, derivado de projetos nas linguagens *Processing* e *Wiring.* Ele possui uma biblioteca chamada *Wiring*, que possibilita a programação em linguagem C e C++. Os programas desenvolvidos no ambiente IDE são chamados de *Sketchs*.

No *software* de desenvolvimento do Arduino, duas funções são definidas para compor a função principal do programa:

a) *Setup()* – é a função onde se inicializam as configurações do programa;
b) *Loop()* – é a função que repete um bloco de comandos. Pode ser interrompida com a utilização de um *timer* com *overflow*.

Para utilizar o *Ethernet shield*, é necessário inserir a biblioteca "*Ethernet*" no *sketch*. As inclusões de bibliotecas são feitas no início do código, através da função "#include". Cada biblioteca é composta de um arquivo com extensão ".h", que é chamada quando é necessário utilizar alguma função nela contida.

A inclusão de bibliotecas está ilustrada na Figura 18.

```
#include <Ethernet.h>
#include <String.h>
#include <Timer.h>
#include <SPI.h>
```

Figura 18: Inclusão de bibliotecas no *sketch*. Fonte: Autor.

5.1.2 Leitura dos Sensores

Para se fazer a leitura dos sensores, foram utilizadas as entradas analógicas 0 e 1 do Arduino. Para isso, utilizou-se o comando "analogRead()", onde é passado como parâmetro o pino onde irá se fazer a leitura.

Quando a primeira medição de temperatura e de luminosidade foi feita, observou-se que a leitura do LDR estava influenciando na leitura do LM35, pois as medidas foram realizadas simultaneamente, sem um intervalo entre as aquisições de dados o que resultava numa medição instável da port analógica. Se corrigiu este problema realizando a leitura do sensor de temperatura mais de uma vez, utilizando somente o último valor lido, gerando um intervalo de tempo entre as medições e eliminando a influência da outra entrada na leitura do sensor. A Figura 19 ilustra esse processo.

```
luminosidade = analogRead(1);  //faz a leitura do sensor de luminosidade na porta analógica 1 do arduino UNO

for (int i = 0; i<8 ; i++)   //esse for se fez necessário para evitar conflitos de leitura entre as portas analógicas
{
temperatura= analogRead(0);  //faz a leitura do sensor de temperatura na porta analógica 0 do arduino UNO
}
```

Figura 19: Leitura dos sensores. Fonte: Autor.

5.1.3 Envio das Informações

O envio das informações foi feito através de um *request* em HTTP. Cada *shield Ethernet* possui um endereço de camada física diferente. Este valor é chamado de "MAC" e é declarado no código para se poder fazer a inicialização da comunicação. Uma vez configurado o "MAC", é necessário criar uma instância "*Client*" da classe *EthernetClient.* Esta instância será inicializada através do método "Ethernet.begin()", que aceita dois tipos de inicialização:

a) DHCP – Se for passado somente o valor de MAC como parâmetro de entrada, o *shield Ethernet* tentará buscar IP automaticamente;
b) IP fixo – Se forem passados os valores de MAC e de IP como parâmetros de entrada, o *shield Ethernet* não tentará buscar um IP automaticamente, através do método DHCP. Ao invés disso, ele fixará como IP o valor passado como parâmetro de entrada da função.

```
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0F, 0x98, 0xA0 };  //definição do MAC do shield Ethernet: cada shield possui o seu valor de MAC
IPAddress ip(192,168,0,177);  //definição de um ip fixo caso o método de obtenção de ip automático (DHCP) falhar
EthernetClient client;  //cria uma instância da classe EthernetClient

 void setup()

 {

if (Ethernet.begin(mac) == 0)  //Caso a rede não conseguiu configurar automaticamente o ip do shield ethernet
{
   Serial.println("Failed to configure Ethernet using DHCP");

   Ethernet.begin(mac, ip);// Configura o ip do shield ethernet com o valor definido anteriormente (192.168.0.177)
 }

   Serial.print("My IP address: ");  //Disponibiliza na porta serial o ip adquirido pelo shield ethernet
 for (byte thisByte = 0; thisByte < 4; thisByte++)
 {
   // imprime o valor de cada tipo de IP:
   Serial.print(Ethernet.localIP()[thisByte], DEC);
   Serial.print(".");
 }
```

Figura 20: Trecho de código da criação e inicialização da instância "*Client*". Fonte: Autor.

Após a inicialização do *shield Ethernet* como um cliente, é necessário fazer a sua conexão com o servidor. O IP do servidor (*Systembox*) é definido no início do código, através da declaração de variável "IPAddress server(192,168,0,100)".

Para conectar o cliente no servidor, utiliza-se a função "client.connect()", onde são passados como parâmetros de entrada o IP do servidor e a porta onde será feita a comunicação (no caso do *Systembox*, pode-se utilizar a porta 80 e 2004). Para este trabalho, utilizou-se a porta 80, que é a porta padrão para *requests* em HTTP.

As informações a serem enviadas são: o valor da temperatura medida pelo sensor LM35, o *setpoint* de temperatura, o valor da luminosidade medida pelo sensor LDR e o *setpoint* de luminosidade.

Para enviá-las, após conectar-se o Arduino com o servidor, é necessário fazer o *request*. Para isso, utilizou-se o comando "client.println()", que recebe como parâmetro de entrada a *string* correspondente ao que será enviado ao servidor. Para formar a *string* de envio de mensagem, foi necessário utilizar uma biblioteca específica chamada de "String.h". Ela permite a quebra de um *string* em várias partes, além da concatenação de um valor inteiro quando se faz a reconstrução das partes fracionadas.

A função "EnviaDados()" é composta por 4 *requests*. Exemplifica-se o envio da temperatura para o servidor na Figura 21.

```
/* Envio de temperatura */

if (client.connect(server, 80))  //caso consiga conectar com o systembox na porta 80, porta padrão de comunicação via HTTP e uma das portas aceitas pelo systembox (a outra porta é 2004)

 {
   Serial.println("connected");


stringOne = "GET /monitor/monitor.cgi?ref_page=setvar&unit=2&newvalue=";
stringTwo = "&newflags=0";
stringThree = stringOne + temperatura + stringTwo; //faz a concatenação dos strings a fim de inserir um valor variável na string
Serial.println(stringThree);
client.println(stringThree);
Serial.println("Disconnecting");
client.stop();  //quando termina o envio da mensagem encerra o client
  }
 else
  {
   Serial.println("Não conseguiu enviar temperatura");   //caso não tenha conseguido se conectar com o systembox envia mensagem de erro
  }

 delay(1000);
```

Figura 21: Trecho de código da função “EnviaDados()”. Fonte: Autor.

Observa-se que, via Arduino, o *request* em HTTP utiliza a requisição GET. O servidor, conforme explicado anteriormente, processa os *requests* de acordo com o que for enviado após a função GET. No exemplo ilustrado acima, o comando configura a variável com ID 2 (correspondente, no *software ScapeWorks*, à variável temperatura) com o valor da variável temperatura medida pelo sensor LM35 e passa o parâmetro *newflags* (correspondente à variável de tempo, que para variáveis numéricas é irrelevante) com o valor 0.

Após o envio da temperatura, a comunicação com o servidor é fechada, utilizando-se a função “client.stop()”. Em seguida, ela é aberta novamente, da mesma maneira mostrada para o envio de temperatura, para se enviar o valor do *setpoint* de temperatura. O processo é repetido para os envios da luminosidade e do *setpoint* de luminosidade.

Todo o processo pode ser monitorado pelo *Serial Monitor* do IDE, que replica as mensagens enviadas pela porta serial.

5.1.4 Parametrização do Envio de Informações

A fim de controlar a frequência com que são enviadas as informações do Arduino para o *Systembox*, resolveu-se fazer a parametrização do envio das informações. Para isso, utilizou-se a biblioteca “timer.h”. Ela permite a criação de uma instância da classe *Timer*, assim como a utilização das funções descritas a seguir:

a) t.every(t, função) – recebe como parâmetros de entrada um valor de tempo, em milisegundos, e uma função. A função especificada é executada cada vez que a interrupção ocasionada pelo *overflow* do *timer* acontecer, o que ocorre a cada “t” milisegundos;

b) t.update() – esta função é chamada dentro da função *loop()*, e é responsável por atualizar o valor do *timer* e por indicar onde o programa será interrompido quando ocorrer a interrupção.

Dessa forma, dentro da função *setup()*, o *timer* é declarado conforme Figura 22.

```
void setup()
{
t.every(10000,EnviaDados);  //Executa, a cada 10000 milisegundos (10 segundos) a função EnviaDados
```

Figura 22: Trecho de código correspondente à inicialização do *timer.* Fonte: Autor.

O *timer* é atualizado na função *loop()*, onde acontecerá a interrupção quando ocorrer o *overflow*, conforme demonstrado na Figura 23.

```
void loop()
{

  t.update(); //atualiza o timer
```

Figura 23: Trecho de código correspondente à atualização do *timer*. Fonte: Autor.

Dessa forma, a função "EnviaDados()", que executa os comando de *request* ao servidor, é chamada a cada 10 segundos. Quando acontecer o *overflow*, há uma interrupção do processo de leitura dos sensores na função *loop*(), e o programa é desviado para a execução do envio das informações ao *Systembox.*

Com esta parametrização, é possível escolher o melhor intervalo de tempo para se fazer a comunicação.

5.2 Comunicação Utilizando o *Shield Wifi*

Quando se iniciou o desenvolvimento deste trabalho, o único *shield* disponível para realizar a comunicação era o *Ethernet*. Porém, considerando-se que o objetivo deste trabalho é enviar os dados de temperatura e luminosidade medidos em uma cadeira de rodas, é de grande importância que a comunicação seja feita sem a necessidade de fios.

Desta forma, fez-se imprescindível a aquisição do *shield Wifi*, que possui funções semelhantes ao *shield Ethernet*. A diferença de utilização entre os dois *shields* se dá somente na inicialização da conexão com o roteador. Para conectar o Arduino via *wifi*, é necessário declarar no *sketch* o nome e o *password* da rede.

5.2.1 Inicialização do Shield Wifi

Algumas funções são importantes para se entender a inicialização do *shield*:

a) WiFi.firmwareVersion() – retorna o valor correspondente à versão do *firmware* presente no *shield.* Este valor é necessário para identificar a necessidade de atualização do *firmware*;

b) WiFi.begin(ssid, pass) – recebe como parâmetros de entrada o nome e o *password* da rede onde se quer conectar o Arduino.

```
char ssid[] = "DouglasDB"; //  your network SSID (name)
char pass[] = "douglasdalbello";    // your network password (use for WPA, or use as key for WEP)
WiFiClient client;  //cria uma instância da classe WiFiClient

   void setup()

   {

       String fv = WiFi.firmwareVersion();
        if ( fv != "1.1.0" )
          Serial.println("Please upgrade the firmware");

        // tenta conectar na rede Wifi:
        while (status != WL_CONNECTED) {
          Serial.print("Attempting to connect to SSID: ");
          Serial.println(ssid);
          // Conecta na rede WPA/WPA2:
          status = WiFi.begin(ssid, pass);

          delay(10000);
        }
```

Figura 24: Trecho de código correspondente à inicialização do *shield Wifi*. Fonte: Autor.

5.2.2 Parametrização e Envio das Informações

O processo de parametrização e envio das informações é rigorosamente idêntico ao processo descrito na utilização do *shield Ethernet*. A mesma função “EnviaDados()” é utilizada para o envio das informações.

5.3 Resultado das Comunicações

A comunicação, conforme explicado anteriormente, pode ser monitorada através da janela *Serial Montor*, no IDE do Arduino, e através do *software ScapeWorks*.

5.3.1 Resultado da Comunicação via Shield Ethernet

A Figura 25 representa a tela do *Serial Monitor*, onde são monitoradas as informações enviadas ao *Systembox*. O processo de comunicação visto pelo *ScapeWorks* está representado na Figura 26.

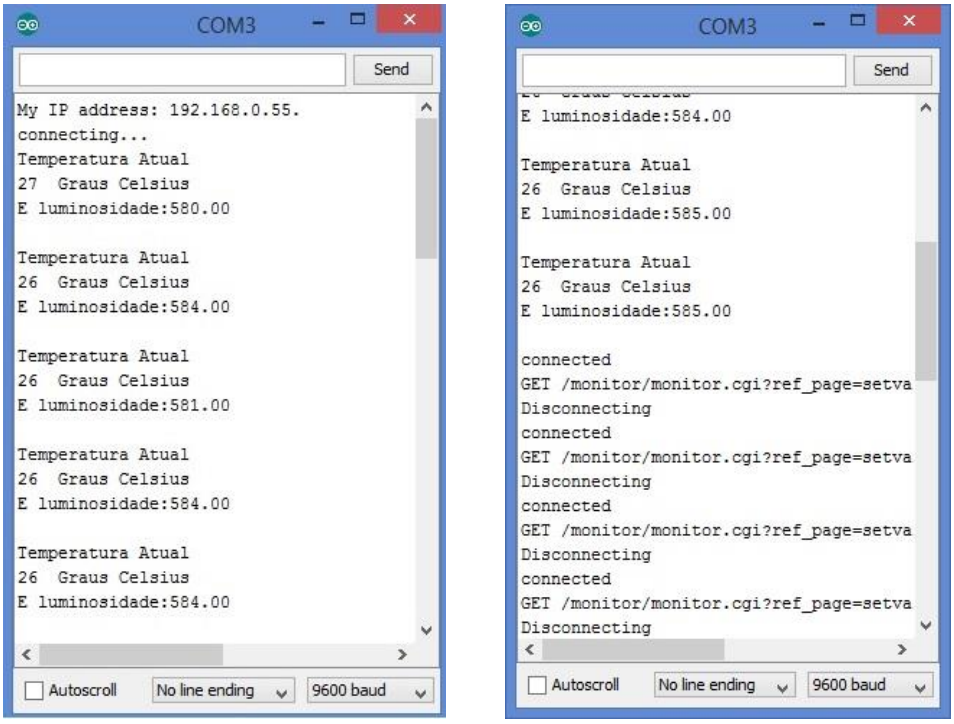

Figura 25: Comunicação via *Shield Ethernet* vista pelo *Serial Monitor*. Fonte: Autor.

Figura 26: Comunicação via *Ethernet Shield* vista pelo *ScapeWorks.* Fonte: Autor.

Como é possível observar, os valores entre parênteses na Figura 26 representam os dados passados pelo Arduino, os quais estão de acordo com os valores mostrados no *Serial Monitor*, na Figura 25. A variável STARTUP é uma variável interna do sistema do *Systembox* e não tem relevância para este projeto.

5.3.2 Resultado da Comunicação via Shield Wifi

O resultado da comunicação *wireless* é semelhante ao obtido via *ethernet*. Observa-se, contudo, na Figura 27, a conexão do *shield* com a rede *wireless* fornecida pelo roteador.

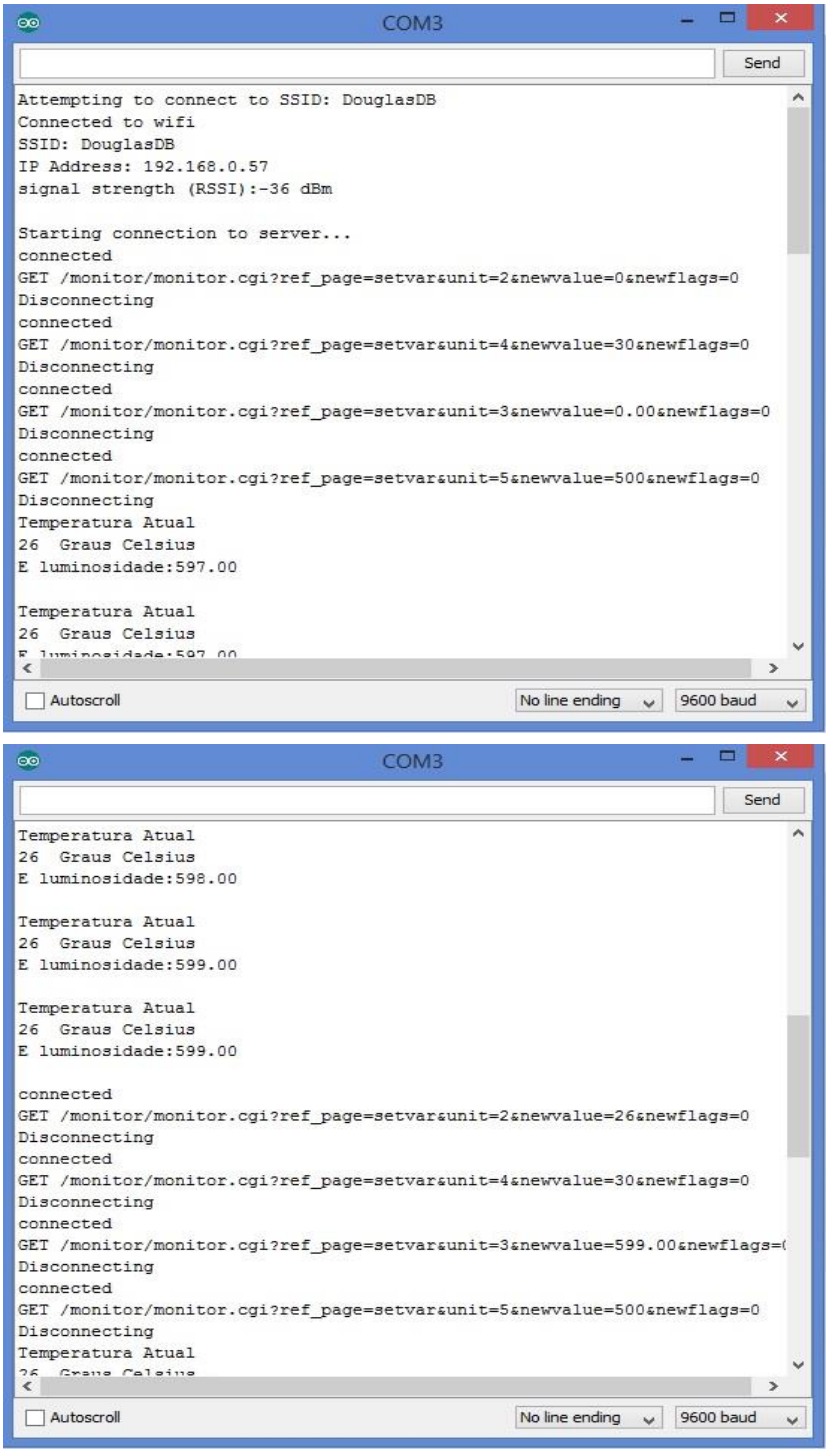

Figura 27: Resultado da comunicação *wifi* visto pelo *Serial Monitor*. Fonte: Autor.

É possível observar que, primeiramente, o Arduino se conecta na rede *wifi* do roteador. Uma vez conectado, é mostrado o valor de IP recebido da rede e o sinal da conexão *wifi*. Em seguida, é feito o envio das variáveis com os valores iniciais de temperatura e luminosidade definidos como 0. Em seguida, tem início a medição de temperatura e luminosidade e, após o tempo estabelecido pelo *timer*, são enviados os dados. No caso deste exemplo, observa-se que a temperatura está em 26 ˚C e a luminosidade possui o valor de 599.

O processo de comunicação via *wifi* visto pelo *ScapeWorks* está representado na Figura 28.

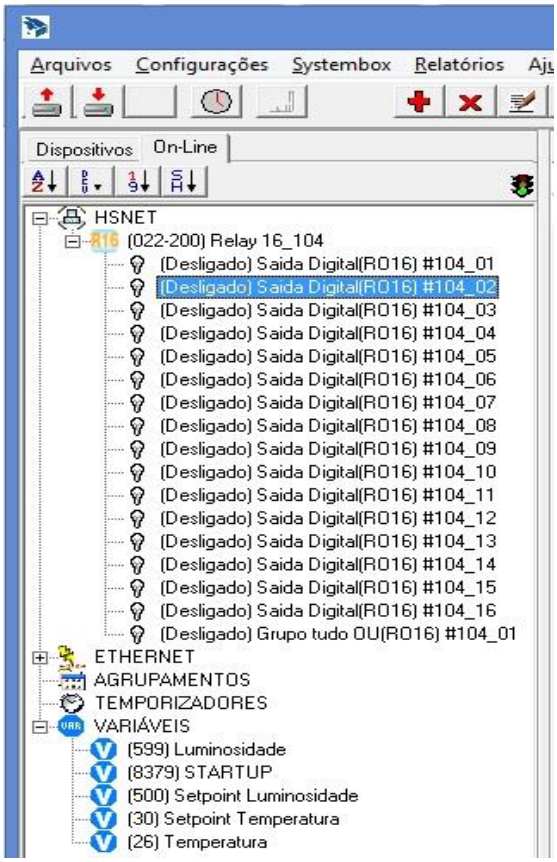

Figura 28: Resultado da comunicação *wireless* visto pelo *ScapeWorks*. Fonte: Autor.

Observa-se que os valores entre parênteses, correspondentes aos valores das variáveis, são os mesmos medidos e enviados pelo Arduino. É possível observar, também, que, como nem o valor de temperatura nem o valor de luminosidade ultrapassam seus *setpoints*, nenhuma saída digital está acionada, respeitando um dos eventos de controle criado anteriormente.

6 Conclusões e Trabalho Futuros

A evolução da automação predial-residencial proporciona a sua utilização em prol da geração de conforto para pessoas idosas e portadoras de deficiência, caracterizando a automação residencial assistiva.

Neste trabalho, utilizaram-se conceitos e técnicas de diversas disciplinas ministradas no Curso de Engenharia de Controle e Automação da UFRGS, tais como Eletrônica, Programação, Circuitos Elétricos e Protocolos de Comunicação, para realizar a implementação de uma comunicação entre uma plataforma eletrônica prototipada, chamada de Arduino, e um controle central de um sistema de automação predial-residencial, visando ao envio dos dados de sensores de temperatura e luminosidade.

Conforme explicado anteriormente, com base nos conceitos de automação predial assistiva, é possível integrar esse sistema com uma cadeira de rodas, fazendo as medições no local do cadeirante ou até mesmo utilizar uma cinta com monitores cardíacos, para fazer o controle dos sinais vitais de uma pessoa doente, por exemplo.

Os resultados obtidos nos testes práticos mostraram que a comunicação entre Arduino e *Systembox* funciona corretamente, sendo possível enviar comandos e definir variáveis através de uma comunicação baseada em *requests* HTTP. Além disso, é possível parametrizar a comunicação, definindo o tempo em que será feito o envio dos dados para o sistema de controle.

Este trabalho tratou apenas do envio e da simulação do controle da temperatura e luminosidade, através do acionamento de dois relés correspondentes a cada um dos dados enviados. A expansão do projeto através do envio de outras informações, como a utilização de um acelerômetro instalado na cadeira de rodas para detectar a presença ou não de uma pessoa, é viável.

Outros tópicos são relevantes para dar continuidade a este projeto:

a) Desenvolver um método de localização da cadeira de rodas no ambiente residencial (por exemplo, através da integração deste projeto com o trabalho desenvolvido por Patrick J. Marques);
b) Fazer a conexão dos relés com dispositivos como ar-condicionado e lâmpadas;
c) Utilizar a comunicação realizada neste projeto para enviar dados fisiológicos de usuários com algum tipo de doença visando ao seu monitoramento por um médico responsável em uma central de controle.

7 Referências

ARDUINO. **Arduino Ethernet Shield**. Disponível em: <http://arduino.cc/en/Main/ArduinoEthernetShield>. Acesso em: 7 out. 2014.

_____. **Arduino UNO**. Disponível em: <http://arduino.cc/en/Main/ArduinoBoardUno>. Acesso em: 7 out. 2014.

_____. **Arduino Wifi Shield.** Disponível em: <http://arduino.cc/en/Main/ArduinoWiFiShield>. Acesso em: 7 out. 2014.

EDUSOFT. **Light Dependence Resistance.** Disponível em: <http://edusofteg.com/edusoft/pdfs/18.pdf>. Acesso em: 28 out. 2014.

GILBERT, Howard. **Introduction to TCP/IP.** New Haven: Yale University, 1995. Disponível em: <http://www.yale.edu/pclt/COMM/TCPIP.HTM>. Acesso em: 3 nov. 2014.

GISPERT, Marc. **Final Project Presentation.** Barcelona**:** Fab Academy, 2012. Disponível em: <http://academy.cba.mit.edu/2012/students/gispert.marc/academy_18.html>. Acesso em: 19 out. 2014.

HACKNMOD. **How to: DIY Home Automation Tutorial.** Disponível em: <http://hacknmod.com/hack/diy-home-automation-tutorial/>. Acesso em: 25 set. 2014.

MARQUES, Patrick J. **Proposta de Sistema de Localização para Cadeira de Rodas Automatizada em Ambiente Inteligente.** Congresso Brasileiro de Automática de Belo Horizonte, 2014.

MENEZES, Maria Luiza Recena. **Sistema de Controle Baseado em Emoções Através de Interface Cérebro Computador**. Proposta de Tese de Doutorado. UFRGS, 2014.

PORTAL NACIONAL DE TECNOLOGIAS ASSISTIVAS. **Automação Residencial Assistiva.** Disponível em: <http://assistivaitsbrasil.wordpress.com/2014/07/01/automacao-residencial-assistiva/>. Acesso em: 14 out. 2014.

SMART HOMES. **Domotics**. Disponível em: <http://www.smart-homes.nl/Domotica.aspx?lang=en-US>. Acesso em: 10 nov. 2014.

TEXAS INSTRUMENTS INCORPORATED. **LM35 Precision Centigrade Temperature Sensors**. Dallas: Texas Instruments Incorporated, 1999. Disponível em: <http://www.ti.com/lit/ds/symlink/lm35.pdf>. Acesso em: 20 out. 2014.

TP-LINK. **TL-WR740N Wireless Router**. Disponível em: <http://www.tp-link.com/en/products/details/?model=tl-wr740n>. Acesso em: 10 nov. 2014.

VIZZOTTO, Marcos. **Imagem Representativa do Sistema de Automação Disponível na UFRGS**. UFRGS, 2014.

8 Apêndices

Apêndice 1: Código utilizado para comunicação com o *shield Ethernet:*

```
    /*   Este código foi criado por Douglas Dal'Bello, aluno de Engenharia de Controle e
Automação da UFRGS. Porto Alegre, novembro de 2014 */
    #include <String.h>
    #include <SPI.h>
    #include <Ethernet.h>
    #include <Timer.h>
    //Declaração de variáveis importantes do sistema
    int setluminosidade = 500;  //setpoint de luminosidade que será enviado ao systembox
    int settemperatura = 30;    //setpoint de temperatura que será enviado ao systembox
    int temperatura = 0;        //temperatura que será enviada ao systembox
    double luminosidade;       //luminosidade que será enviada ao systembox
    byte mac[] = { 0x90, 0xA2, 0xDA, 0x0F, 0x98, 0xA0 };  //definição do MAC do shield
Ethernet: cada shield possui o seu valor de MAC
    IPAddress ip(192,168,0,177);  //definição de um ip fixo caso o método de obtenção de ip
automático (DHCP) falhar
    EthernetClient client;  //cria uma instância da classe EthernetClient
    IPAddress server(192,168,0,100);  //definição do ip do servidor (nesse caso, do
systembox)
    String stringOne, stringTwo, stringThree;  //as 3 strings que serão utilizadas para
enviar a mensagem ao systembox
    Timer t;  //timer que será utilizado para parametrizar de quanto em quanto tempo as
mensagens serão enviadas ao systembox
      void setup()
    { t.every(10000,EnviaDados);  //Executa, a cada 10000 milisegundos (10 segundos) a
função EnviaDados
     Serial.begin(9600);  //Inicia a porta serial com BaudRate de 9600
     if (Ethernet.begin(mac) == 0)  //Caso a rede não conseguiu configurar automaticamente
o ip do shield ethernet
     {    Serial.println("Failed to configure Ethernet using DHCP");
        Ethernet.begin(mac, ip);// Configura o ip do shield ethernet com o valor definido
anteriormente (192.168.0.177)  }
        Serial.print("My IP address: ");  //Disponibiliza na porta serial o ip adquirido
pelo shield ethernet
      for (byte thisByte = 0; thisByte < 4; thisByte++)
      {    // imprime o valor de cada tipo de IP:
        Serial.print(Ethernet.localIP()[thisByte], DEC);
        Serial.print(".");}
      Serial.println();
      delay(1000);
      Serial.println("connecting...");
      delay(100);}
     void loop()
    {  t.update(); //atualiza o timer
    luminosidade = analogRead(1);  //faz a leitura do sensor de luminosidade na porta
analógica 1 do arduino UNO
    for (int i = 0; i<8 ; i++)   //esse for se fez necessário para evitar conflitos de
leitura entre as portas analógicas
    {temperatura= analogRead(0);  //faz a leitura do sensor de temperatura na porta
analógica 0 do arduino UNO}
    temperatura = temperatura * 0.488;  //aplica o fator de correção para a temperatura ser
mostrada em °C
    Serial.print("Temperatura Atual");
    Serial.println();
    Serial.print(temperatura);
    Serial.print("  Graus Celsius");
    Serial.println();
    Serial.print("E luminosidade:");
    Serial.println(luminosidade);
    Serial.println();
    delay(1000);}
    void EnviaDados()
    //esta é a função que faz o envio de dados para o systembox
    {
      /* Envio de temperatura */
     if (client.connect(server, 80))  //caso consiga conectar com o systembox na porta 80,
porta padrão de comunicação via HTTP e uma das portas aceitas pelo systembox (a outra porta
é 2004)
      {   Serial.println("connected");
    stringOne = "GET /monitor/monitor.cgi?ref_page=setvar&unit=2&newvalue=";
    stringTwo = "&newflags=0";
    stringThree = stringOne + temperatura + stringTwo; //faz a concatenação dos strings a
fim de inserir um valor variável na string
    Serial.println(stringThree);
    client.println(stringThree);
```

```
    Serial.println("Disconnecting");
    client.stop();  //quando termina o envio da mensagem encerra o client
      }
     else
      {   Serial.println("Não conseguiu enviar temperatura");   //caso não tenha conseguido
se conectar com o systembox envia mensagem de erro
      }
      delay(1000);
       /* Envio do setpoint de temperatura  */
      if (client.connect(server, 80))
     {    Serial.println("connected");

    stringOne = "GET /monitor/monitor.cgi?ref_page=setvar&unit=4&newvalue=";
    stringTwo = "&newflags=0";
    stringThree = stringOne + settemperatura + stringTwo;
    Serial.println(stringThree);
    client.println(stringThree);
    Serial.println("Disconnecting");
    client.stop();
     }
     else
     {   Serial.println("Não conseguiu enviar setpoint de temperatura");
     }
      delay(1000);
      /* Envio da luminosidade */
      if (client.connect(server, 80))
     {    Serial.println("connected");
    stringOne = "GET /monitor/monitor.cgi?ref_page=setvar&unit=3&newvalue=";
    stringTwo = "&newflags=0";
    stringThree = stringOne + luminosidade + stringTwo;
    Serial.println(stringThree);
    client.println(stringThree);
    Serial.println("Disconnecting");
    client.stop();
     }
     else
     {   Serial.println("Não conseguiu enviar luminosidade");
     }
      delay(1000);
       /* Envio do setpoint de luminosidade */
      if (client.connect(server, 80))
     {    Serial.println("connected");

    stringOne = "GET /monitor/monitor.cgi?ref_page=setvar&unit=5&newvalue=";
    stringTwo = "&newflags=0";
    stringThree = stringOne + setluminosidade + stringTwo;
    Serial.println(stringThree);
    client.println(stringThree);
      Serial.println("Disconnecting");
    client.stop();
     }
     else
     {   Serial.println("Não conseguiu enviar setpoint de luminosidade");
     }
      delay(1000);
     }
```

Apêndice 2: Código utilizado para comunicação com o *shield Wifi:*

```
    /*    Este código foi criado por Douglas Dal'Bello, aluno de Engenharia de Controle e
Automação da UFRGS. Porto Alegre, novembro de 2014*/
    #include <String.h>
    #include <SPI.h>
    #include <WiFi.h>
    #include <Timer.h>
    char ssid[] = "DouglasDB"; //  your network SSID (name)
    char pass[] = "douglasdalbello";     // your network password (use for WPA, or use as
key for WEP)
    //Declaração de variáveis importantes do sistema
    int setluminosidade = 500;  //setpoint de luminosidade que será enviado ao systembox
    int settemperatura = 30;    //setpoint de temperatura que será enviado ao systembox
    int temperatura = 0;        //temperatura que será enviada ao systembox
    double luminosidade;       //luminosidade que será enviada ao systembox
    Address  server(192,168,0,100);    //definição  do  ip  do  servidor  (nesse  caso,  do
systembox)
    IPAddress ip(192,168,0,177);  //definição de um ip fixo caso o método de obtenção de ip
automático (DHCP) falhar
    WiFiClient client;  //cria uma estância da classe EthernetClient
```

```
    String stringOne, stringTwo, stringThree;  //as 3 strings que serão utilizadas para
enviar a mensagem ao systembox
    Timer t;  //timer que será utilizado para parametrizar de quanto em quanto tempo as
mensagens serão enviadas ao systembox
    int status = WL_IDLE_STATUS;
    void setup()
    { t.every(10000,EnviaDados);   //Executa, a cada 10000 milisegundos (10 segundos) a
função EnviaDados
     Serial.begin(9600);  //Inicia a porta serial com BaudRate de 9600
    String fv = WiFi.firmwareVersion();
      if ( fv != "1.1.0" )
        Serial.println("Please upgrade the firmware");
      // attempt to connect to Wifi network:
      while (status != WL_CONNECTED) {
        Serial.print("Attempting to connect to SSID: ");
        Serial.println(ssid);
        // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
        status = WiFi.begin(ssid, pass);
        // wait 10 seconds for connection:
        delay(10000);
      }
      Serial.println("Connected to wifi");
      printWifiStatus();
      Serial.println("\nStarting connection to server...");
      delay(1000);
     }
    void loop()
    {  t.update(); //atualiza o timer
    luminosidade = analogRead(1);   //faz a leitura do sensor de luminosidade na porta
analógica 1 do arduino UNO
    for (int i = 0; i<8 ; i++)    //esse for se fez necessário para evitar conflitos de
leitura entre as portas analógicas
    {temperatura= analogRead(0);   //faz a leitura do sensor de temperatura na porta
analógica 0 do arduino UNO
    }
    temperatura = temperatura * 0.488;  //aplica o fator de correção para a temperatura ser
mostrada em °C
    Serial.print("Temperatura Atual");
    Serial.println();
    Serial.print(temperatura);
    Serial.print("  Graus Celsius");
    Serial.println();
    Serial.print("E luminosidade:");
    Serial.println(luminosidade);
    Serial.println();
    delay(1000);}
    void EnviaDados() //esta é a função que faz o envio de dados para o systembox
    {  /* Envio de temperatura */
        if (client.connect(server, 80))  //caso consiga conectar com o systembox na porta
80, porta padrão de comunicação via HTTP e uma das portas aceitas pelo systembox (a outra
porta é 2004)
      {    Serial.println("connected");
    stringOne = "GET /monitor/monitor.cgi?ref_page=setvar&unit=2&newvalue=";
    stringTwo = "&newflags=0";
    stringThree = stringOne + temperatura + stringTwo; //faz a concatenação dos strings a
fim de inserir um valor variável na string
    Serial.println(stringThree);
    client.println(stringThree);
    Serial.println("Disconnecting");
    client.stop();  //quando termina o envio da mensagem encerra o client
      }
     else   {    Serial.println("Não conseguiu enviar temperatura");    //caso não tenha
conseguido se conectar com o systembox envia mensagem de erro  }
      delay(1000);
      /* Envio do setpoint de temperatura  */
      if (client.connect(server, 80))
     {    Serial.println("connected");
    stringOne = "GET /monitor/monitor.cgi?ref_page=setvar&unit=4&newvalue=";
    stringTwo = "&newflags=0";
    stringThree = stringOne + settemperatura + stringTwo;
    Serial.println(stringThree);
    client.println(stringThree);
    Serial.println("Disconnecting");
    client.stop();
     }
     else {   Serial.println("Não conseguiu enviar setpoint de temperatura");
     }
      delay(1000);
       /* Envio da luminosidade */
      if (client.connect(server, 80))
     {    Serial.println("connected");
```

```
stringOne = "GET /monitor/monitor.cgi?ref_page=setvar&unit=3&newvalue=";
stringTwo = "&newflags=0";
stringThree = stringOne + luminosidade + stringTwo;
Serial.println(stringThree);
client.println(stringThree);
Serial.println("Disconnecting");
client.stop(); }
 else {   Serial.println("Não conseguiu enviar luminosidade");
 }
 delay(1000);
 /* Envio do setpoint de luminosidade */
 if (client.connect(server, 80))
 {    Serial.println("connected");
stringOne = "GET /monitor/monitor.cgi?ref_page=setvar&unit=5&newvalue=";
stringTwo = "&newflags=0";
stringThree = stringOne + setluminosidade + stringTwo;
Serial.println(stringThree);
client.println(stringThree);
  Serial.println("Disconnecting");
client.stop(); }
 else {   Serial.println("Não conseguiu enviar setpoint de luminosidade");
 }
  delay(1000); }
void printWifiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());
  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);
  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");}
```