Luimar Donini

Projeto de um Posicionador para Caracterização de Diagramas de Irradiação de Antenas

Alegrete - RS

30 de junho de 2016

Luimar Donini

Projeto de um Posicionador para Caracterização de Diagramas de Irradiação de Antenas

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Telecomunicações. Área de Concentração em eletrônica, da Universidade Federal do Pampa (Unipampa - RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Telecomunicações**.

Universidade Federal do Pampa – Unipampa

Curso de Engenharia de Telecomunicações

Orientador: Prof. Me. Edson Rodrigo Schlosser

Alegrete - RS

30 de junho de 2016

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

D952p Donini, Luimar
Projeto de um Posicionador para Caracterização de Diagramas de Irradiação de Antenas / Luimar Donini.
174 p.

Trabalho de Conclusão de Curso(Graduação)-- Universidade Federal do Pampa, ENGENHARIA DE TELECOMUNICAÇÕES, 2016.
"Orientação: Edson Rodrigo Schlosser".

1. Antenas. 2. Posicionadores. 3. Diagramas de irradiação. I. Título.

Luimar Donini

Projeto de um Posicionador para Caracterização de Diagramas de Irradiação de Antenas

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Telecomunicações. Área de Concentração em eletrônica e automação, da Universidade Federal do Pampa (Unipampa - RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Telecomunicações**.

Trabalho aprovado. Alegrete - RS, 30 de junho de 2016:

Prof. Me. Edson Rodrigo Schlosser
Orientador

Prof. Dr. Jumar Luís Russi
Unipampa

Prof. Dr. Marcos V. T. Heckler
Unipampa

Alegrete - RS
30 de junho de 2016

"Para quebrar as regras,
primeiro você deve dominá-las com maestria."
(Audemars Piguet)

Resumo

Sistemas para medidas de diagrama de irradiação de antenas são fundamentais para a caracterização e validação de protótipos projetados através de ferramentas de análise eletromagnética. Como parte deste sistema, um posicionador comercial possui custo bastante elevado, fato esse, que resultou na elaboração do projeto aqui descrito. Tal projeto implementa um posicionador de um eixo, automatizado, eficiente, de baixo custo e que permite o interfaceamento com outros dispositivos que englobam este sistema, além do usuário. O desenvolvimento do posicionador consiste na implementação de uma interface em ambiente *Matlab*, utilizada para o controle do posicionador e para ler e armazenar os dados medidos a partir de um analisador de espectro. A comunicação entre a interface e o *hardware* de controle é realizada através de *Ethernet* por protocolo TCP/IP (*Transfer Control Protocol/Internet Protocol*), diferente de outros modos de comunicação tais como USB, serial e paralela, que apresentam limitações muito restritas quanto à distância entre dois pontos de conexão. O controle é feito através de um Arduino, que é responsável por acionar os transistores de um *driver* de potência construído para acionamento do motor de passo. Para cada passo do motor, é realizada a medida do sinal recebido pela antena em teste, através do analisador de espectro, o qual é conectado com a interface e transmite os dados medidos para armazenamento e plotagem. A comunicação com o Arduino é realizada através de uma *Shield Ethernet*, a qual recebe comandos por uma página HTML (*HyperText Markup Language*) hospedada na própria *Shield* e disponibilizada para acesso externo. Após receber um comando de algum cliente HTML ou TCP/IP, a *Shield* repassa este comando ao Arduino, que, por sua vez, se encarrega de executar a função desejada. O *driver* de potência é necessário, uma vez que os pinos de I/O (*Input/Output*) de um Arduino não fornecem corrente suficiente para acionar as bobinas de um motor. O projeto aqui descrito poderá ser utilizado para sistemas de medição em câmaras anecoicas e em espaço livre.

Palavras-chave: antenas, posicionadores, diagramas de irradiação.

Abstract

Systems for antenna radiation pattern measurement are fundamental for the characterization and validation of prototypes designed by electromagnetic simulators. As part of this system, a commercial positioner has very high cost, a fact that resulted in the elaboration of the project described here. Such a design implements a positioner with one axis, automated, efficient, low cost and that allows interfacing with other devices that include this system, besides the user. The development of the positioner includes an interface in Matlab, used for the control of the positioner and to read and store the measured data from a spectrum analyzer. The communication between the interface and control of hardware is performed via Ethernet TCP/IP (Transfer Control Protocol/Internet Protocol), unlike other modes of communication, such as USB, serial and parallel, which have distance limitations between two connection points. The control is made through an Arduino, which is responsible for activating the transistors of the stepper motor driver. For each motor step, the signal as received by the antenna under test is performed by the spectrum analyzer which is connected to the interface and transmits the measured data storage and plotting. Communication with the Arduino is performed through a Shield Ethernet, which receives commands from an HTML page (HyperText Markup Language) hosted on the own Shield and made available for external access. After receiving a command from any HTML or TCP/IP client, Shield passes this command to the Arduino, which in turn is responsible to perform the desired function. The Driver power is needed, since the pin I/O (Input/Output) a Arduino does not provide enough current to drive the coils of a motor. The project described here can be used for measurement systems in anechoic chambers and in free space.

Key-words: antennas, positioners, radiation pattern.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Um sistema para medida de diagramas de irradiação de antenas em campo distante é basicamente composto por um gerador de sinais, antena transmissora, antena receptora (protótipo), posicionador e analisador de espectro. Atualmente, o posicionador deste sistema apresenta elevado custo de aquisição, atingindo um valor de milhares de reais. A finalidade de cada instrumento citado pode ser descrita de forma simplificada como: o gerador de sinais é utilizado para gerar uma onda senoidal em determinada frequência e amplitude; a antena transmissora é responsável por transmitir esse sinal através da irradiação de ondas eletromagnéticas. Posteriormente, um protótipo localizado a uma distância de campo distante (BALANIS, 2009) é utilizado para receber o sinal transmitido, e a amplitude desse sinal é lida através de um analisador de espectro conectado na antena receptora. A amplitude do sinal está intimamente ligada ao alinhamento entre as antenas transmissora e receptora e à polarização. O ganho da antena receptora (protótipo) depende das características construtivas e varia conforme o seu posicionamento no espaço tridimensional. Em projetos práticos, faz-se necessária a caracterização da antena em termos de ganho em determinada direção, sendo assim, indispensável o uso de um posicionador, que é o equipamento responsável por rotacionar a antena receptora. Este projeto implementa um posicionador de 1 eixo, automatizado, de baixo custo e com interfaceamento do controle do posicionador e da medida realizada pelo analisador de espectro. No campo de análise e síntese de antenas, existem diversos métodos numéricos que levam à obtenção das características de irradiação da antena, tais como a ferramenta da *CST* e da *Ansys*. Assim sendo, se faz necessária a validação do protótipo após o projeto e construção, de forma a identificar possíveis erros decorrentes do processo de fabricação. Em (BROWN; GOORA; ROUSE, 2011) encontra-se um projeto completo de um sistema para medidas de ganho e padrão de irradiação de antenas, construído com motor de passo, microcontrolador, amplificador de baixo ruído (LNA), filtros e outros componentes. Tal projeto assemelha-se em muito com o descrito neste trabalho, uma vez que foi construído para ser de baixo custo e atender a requisitos de desempenho satisfatórios. O sistema descrito em (BROWN; GOORA; ROUSE, 2011), também utiliza interface com software de controle do sistema e leitura de dados.

Geralmente, as medidas com antenas são feitas com a antena em modo de recepção, e o procedimento de medição, em campo distante, consiste em iluminar a antena em teste

por ondas de amplitude e fase uniformes (ondas planas). Para se chegar a uma precisão confiável de onda plana, a antena em teste deve ser posicionada a uma grande distância da antena transmissora (campo distante). Deve-se levar em conta, entretanto, as possíveis fontes de degradação da iluminação da antena em teste, tais como: reflexões no solo e em objetos próximos e curvatura da frente de onda irradiada devido às finitas distâncias de separação. Embora necessárias, técnicas experimentais de medição possuem muitas deficiências, tais como alto custo, condições climáticas fora de controle em medidas ao ar livre, limitação de tamanho para medidas internas, entre outras. Com o progresso acelerado de sistemas de defesa aeroespaciais, métodos de medidas mais precisos são necessários. Para tanto, foram desenvolvidos instrumentos e técnicas aprimoradas, como câmaras anecóicas piramidais, campos de medida compacto e de extrapolação, técnicas de amostragem de campo próximo, aprimoramento de técnicas de polarização e de medidas com varredura de frequência, técnicas de amostragem de campo próximo, medidas indiretas das características de antenas e sistemas automatizados de medidas. Em geral, os parâmetros que se quer obter para a caracterização do desempenho de um sistema de antenas são: diagrama (amplitude e fase), ganho, diretividade, eficiência, impedância, distribuição de corrente e polarização (BALANIS, 2009).

O sistema de medida e posicionamento de antenas aqui descrito, é composto de uma estrutura giratória confeccionada em aço, *hardware* de controle e alimentação, gerador de sinais e analisador de espectro. O circuito de controle é composto por um Arduino, *shield Ethernet* e por um circuito desenvolvido para ser acoplado aos dois primeiros (Arduino e *shield*). O Arduino é quem fornece toda a lógica de controle do sistema através de seus pinos de I/O (*Input/Output*), e também é responsável por processar dados lidos de sensores. A *shield Ethernet* possibilita o controle remoto do Arduino através de uma rede LAN (*Local Area Network*) ou WAN (*Wide Area Network*). O circuito desenvolvido e detalhado na seção 3.1, é composto por um circuito de alimentação para o Arduino, transistores de potência (*drivers*) para acionamento de um motor de passo, sensor de temperatura e sensores de campo magnético (sensores *hall*) para posicionamento da estrutura (calibragem). Todo o sistema, inclusive o gerador de sinais e o analisador de espectro, é interligado por rede através de um roteador, podendo assim, controlá-lo (o sistema) remotamente através de um notebook, tablet ou até mesmo de um celular. O controle também pode ser feito de qualquer parte do mundo, bastando, para isso, integrar um modem devidamente configurado ao sistema.

O gerador de sinais e o analisador de espectro são controlados remotamente através de comandos SCPI (*Standard Commands for Programable Instruments*), que podem ser

embarcados em outras linguagens de programação como *Matlab*, por exemplo. Assim sendo, todo o sistema pode ser controlado e configurado com apenas um computador ou dispositivo capaz de acessar uma rede *Ethernet*.

1.1 Organização deste trabalho

Este trabalho está organizado da seguinte maneira:

- Referencial teórico: Este capítulo abrange o funcionamento e a teoria por trás dos periféricos do sistema, tais como antenas, motores de passo, Arduino, geradores de sinais, analisadores de espectro, entre outros;
- Desenvolvimento: Neste capítulo são descritos todos os passos de projeto que foram utilizados para a criação do sistema de medidas. Trata-se de temas como a criação de um *driver* de potência para acionamento de motor de passo, e interfaces de controle;
- Montagem do sistema e resultados: Capítulo destinado à descrição da montagem do sistema, testes e análise de resultados. Isto é, interconexão em rede, de todos os dispositivos que compõem o sistema, configuração remota e controle da medida de diagramas de irradiação de antenas em teste;
- Conclusão: Com base nos dados apresentados, este capítulo faz uma breve síntese do que foi discutido no trabalho, fazendo-se conclusões sobre diversos fatores do sistema;
- Bibliografia: Capítulo destinado à referências bibliográficas do texto;
- Apêndices: Aqui são apresentados alguns códigos descritos e referenciados durante o texto, bem como, são detalhadas algumas funcionalidades de equipamentos que fazem parte do sistema de medição.

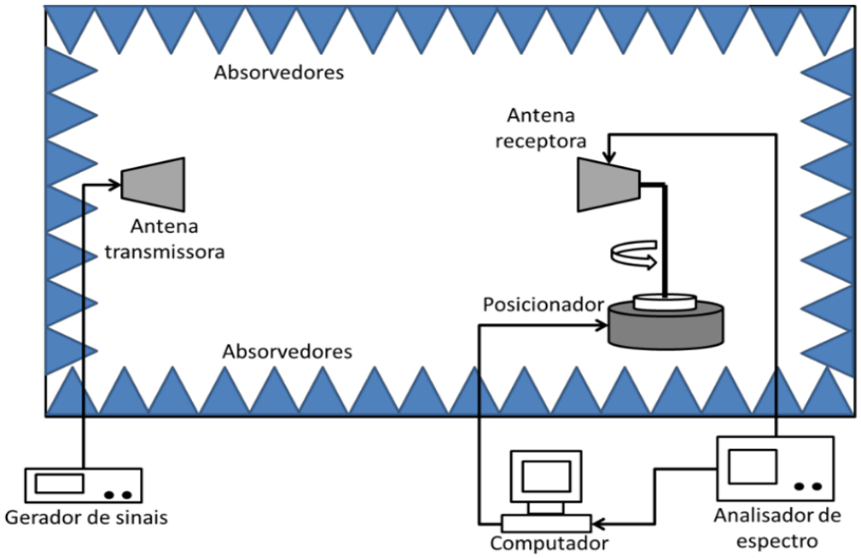
2 Revisão Bibliográfica

Todos os dispositivos utilizados na comunicação sem fio requerem antenas para transmitir e receber sinais através de ondas eletromagnéticas. Para esta aplicação, é necessário dispor de sistemas capazes de caracterizar todos os tipos de dispositivos irradiantes de potência que venham a ser utilizados em qualquer sistema de comunicação, denominados sistemas de medida de antenas. Alguns sistemas encontrados na literatura são: sistemas de medição no espaço livre e em câmaras anecóicas, ambos em campo distante, e sistemas de medição em campo próximo. No projeto apresentado em (BROWN; GOORA; ROUSE, 2011) o sistema foi validado fazendo-se medições em uma câmara anecoica e os resultados foram comparados com os obtidos através de um analisador de espectro comercial.

Existem, ainda, vários subsistemas que fazem parte das medições de antenas, tais como absorvedores, posicionadores e controladores, gerador de sinais, analisador de espectro, antenas e *softwares*. Os posicionadores citados podem apresentar diversos graus de liberdade (eixos), permitindo medir diagramas no plano azimutal (ϕ), elevação (θ) e também por polarização (BALANIS, 2009).

Um sistema completo para caracterização de antenas apresenta, em sua maioria, elevado custo de implementação, podendo-se investir na faixa de milhões de reais para implantar um sistema de medição em campo distante dentro de câmaras anecóicas (KRAUS; MARHEFKA, 2001). Um esquema simplificado de medição de antenas em uma câmara anecoica pode ser visualizado na Fig. 1, onde a antena transmissora irradia a onda proveniente do gerador de sinais. Na recepção (ainda em referência à Fig. 1), a antena receptora capta o sinal transmitido, cuja amplitude é detectada por um analisador de espectro. A amplitude detectada para cada posição da antena receptora, é, então, enviada a um computador para armazenamento e tratamento. Após o armazenamento da amplitude, o computador envia comandos para que a estrutura giratória (posicionador) rotacione para a próxima posição. Todo o processo é realizado novamente, até que todas as amplitudes (em 360°) sejam recebidas. Com os valores das amplitudes de cada posição armazenados, a plotagem do diagrama de irradiação para a caracterização da antena pode ser feito através de algum *software* específico, alguma linguagem de programação, ou ainda, desenhado manualmente em uma carta polar (KUMMER, 1992).

Figura 1: Esquema simplificado de um sistema de medição de antenas.



2.1 Campos de antenas

Campos de antenas são onde os testes e avaliações de antenas são realizados. São classificados em campos exteriores e interiores, tendo cada um suas próprias limitações. Em campos exteriores, existe a interferência das condições climáticas; já em campos interiores, as limitações são devidas às restrições de espaço.

Existem basicamente dois tipos de campos de antenas: os campos de reflexão e os campos de espaço livre. Os campos de reflexão, se projetados cuidadosamente, podem criar interferência construtiva na região da antena em teste, ou seja, reflexões no solo se combinam construtivamente com os raios de visada direta. Já campos de espaço livre são projetados para minimizar ou até mesmo extinguir as interferências do ambiente que o circunda, e incluem campos elevados, campos inclinados, câmaras anecoicas, campos compactos e os chamados campo eletromagnético próximo e distante (BALANIS, 2009).

Os campos de irradiação definidos como campo próximo e campo distante são de especial relevância no projeto e análise de antenas, pois definem os tipos de distribuição de linhas de campo nestes dispositivos irradiadores. A região de campo próximo também é conhecida como região de indução ou de Fresnel, e diz respeito às linhas de campo mais próximas da antena, as quais deixam de existir imediatamente ao cessar a causa, isto é, quando cessa a corrente na antena ocorre a anulação por um semiciclo, e as linhas não

chegam a se fechar, portanto, não se propagam. O efeito de campo próximo é utilizado em projetos com um ou mais elementos "parasitas", de forma a induzir nestes elementos a energia que seria desperdiçada. As linhas de campo que se fecham, propagam-se pelo espaço e continuam carregando consigo a energia irradiada. A este efeito se dá o nome de "campo distante" ou de Fraunhofer. O campo elétrico na região distante varia com o inverso do quadrado da distância, enquanto que na região próxima isto não acontece. As equações que delimitam as regiões de campo próximo e de campo distante são

$$R = 10\lambda \tag{2.1}$$

$$R = \frac{2L^2}{\lambda}, \tag{2.2}$$

onde L é o maior comprimento físico da antena, λ é o comprimento de onda e R é a distância de separação entre antena transmissora e receptora.

2.2 Diagramas de irradiação

Os diagramas de parâmetros de antenas, tais como diagramas de irradiação, polarização e ganho, são medidos na superfície de uma esfera de raio constante. Para tanto, é utilizado o sistema de coordenadas esféricas, como mostra a Fig. 2. Como o raio é constante, apenas as coordenadas angulares (θ e ϕ) são necessárias para a identificação da posição. Logo, o diagrama de uma antena nada mais é do que a representação de suas características de irradiação em função de θ e ϕ para uma distância radial constante e em uma determinada frequência de operação, ou seja, é o mapeamento da distribuição de energia irradiada, levando em conta o campo tridimensional (POZAR, 2009). A obtenção do diagrama de irradiação de antenas se faz de duas maneiras: testes em campo aberto ou simulação computacional, e são geralmente medidos em dBi (ganho), expresso em relação a uma antena isotrópica.

Uma vez que medidas de diagramas tridimensionais são impraticáveis, estes são realizados por meio de diagramas bidimensionais, e podem ser medidos no modo de transmissão ou recepção. Para antenas recíprocas, a medição em modo de recepção é geralmente escolhida.

Para realizar uma medida confiável de diagrama de irradiação de uma antena em teste, deve-se escolher um local onde não haja interferências externas, tais como árvores, calhas, rufos, arames, linhas de transmissão de energia ou telefônicas, carros, entre outros.

Geralmente, levanta-se o diagrama com separações entre antenas transmissora e receptora, maiores do que 10 vezes o comprimento de onda relativo à frequência de teste. De um modo genérico, três passos devem ser realizados para a obtenção de um diagrama de irradiação

1. Gira-se a antena em teste de forma a descrever um círculo;

2. A intervalos regulares, toma-se a medida do campo irradiado de forma a obter-se um gráfico;

3. Os valores devem ser anotados ou em valores absolutos, ou em valores relativos ao seu máximo (normalizados).

O resultado da plotagem dos valores das amplitudes anotadas em função dos ângulos de rotação θ e/ou ϕ é chamado de diagrama de irradiação. A Fig. 3 mostra um diagrama obtido após a síntese de uma de uma rede linear de antenas de microfita (TOLFO, 2016). Na Fig. 3, é possível notar que a direção de máximo apontamento (lóbulo principal) está em aproximadamente 80°. Os demais lóbulos são conhecidos como lóbulos secundários, e geralmente, é o que se deseja minimizar, através de técnicas de síntese de diagramas.

2.3 Instrumentos de medição

Para identificar os instrumentos necessários para efetuar medidas de antenas, devem ser levados em consideração seus requisitos funcionais. Em geral, tais instrumentos podem ser classificados em cinco categorias:

1. Antena-fonte e sistema transmissor;

2. Sistema receptor;

3. Sistema de posicionamento;

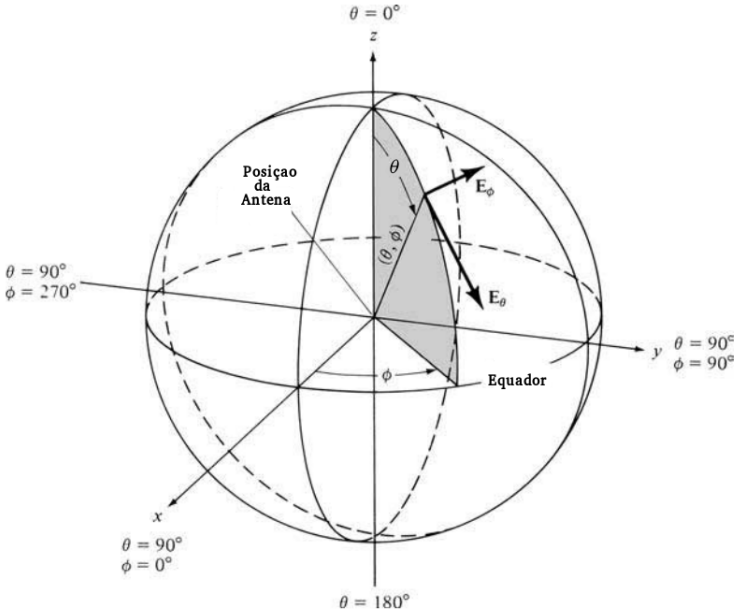
4. Sistema de plotagem;

5. Sistema de processamento de dados.

A Fig. 4 mostra um diagrama de blocos de um sistema que possui as características citadas anteriormente.

As antenas fonte devem ser cuidadosamente escolhidas, a fim de se obter um melhor resultado na medição. Em geral, o que se procura para os transmissores é: estabilidade de frequência, pureza espectral, nível de potência e modulação. Para conseguir um diagrama

Figura 2: Sistema de coordenadas esféricas.

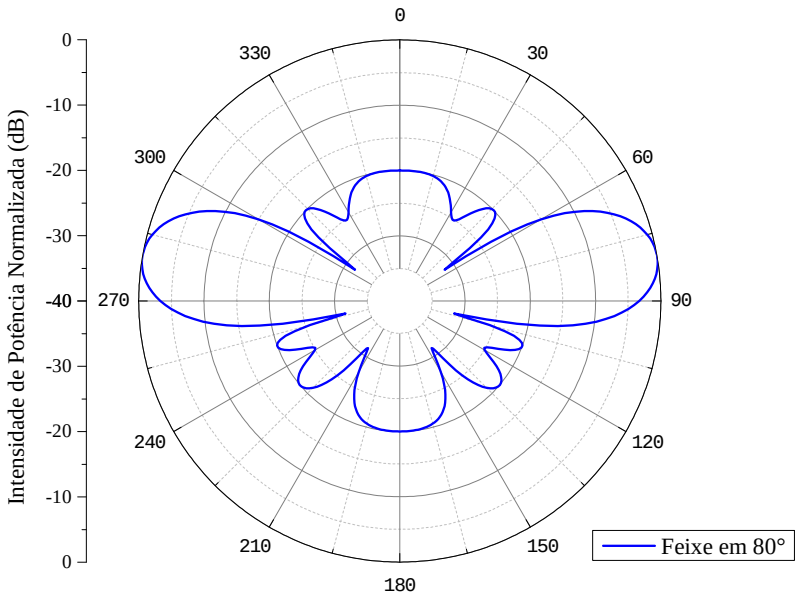


Fonte: Modificado de (BALANIS, 2009).

completo, a estrutura do sistema de medição deve ser capaz de girar em vários planos. Isso se consegue através de pedestais giratórios devidamente instalados, que podem girar tanto no plano azimutal quanto no plano de elevação. Duas juntas rotativas são mostradas na Fig. 5. Quanto aos tipos de plotagem, existem basicamente dois: os retangulares e os polares. Os mais utilizados são os polares, por permitirem uma melhor visualização da distribuição da irradiação da antena no espaço. Em geral, o sistema de plotagem é projetado para traçar o diagrama relativo, e é calibrado para traçar os diagramas de campo ou potência. Os diagramas de potência são calibrados em decibéis, e, para a maioria das aplicações, uma faixa dinâmica de 0 a -40 dB oferece resolução suficiente para examinar a estrutura dos lóbulos principal e secundários do diagrama.

Um sistema moderno para medida de diagramas de antenas e de seção radar (RCS), que utiliza um analisador de redes e é automatizado por computador, é mostrado na Fig. 6. À medida que a antena em teste é girada, a intensidade do campo recebido é medida por um analisador de espectro ou medidor de potência. Também é possível implementar um analisador de redes para medir a impedância de entrada associada ao parâmetro S_{11} da antena em teste e do meio de transmissão (parâmetro S_{21}) entre a

Figura 3: Diagrama de irradiação de uma rede linear de antenas de microfita.



Fonte: Modificado de (TOLFO, 2016).

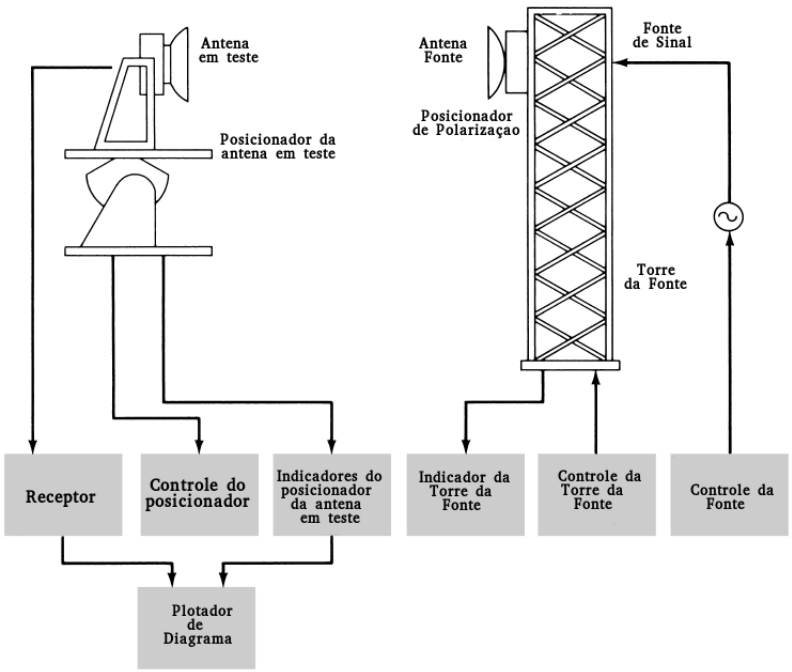
fonte e a antena em teste.

Em geral, em sistemas de comunicação ponto-a-ponto, deve-se combinar o tipo de polarização entre as antenas transmissora e receptora, bem como alinhar o ângulo de inclinação de ambas para um acoplamento máximo, e, consequentemente, uma máxima transferência de potência (FUSCO, 2009).

2.4 Motores de passo

Os motores de passo são dispositivos que convertem pulsos elétricos em movimentos mecânicos sequenciais, que geram variações angulares discretas. O rotor (ou eixo) de um motor de passo pode ser rotacionado em pequenos incrementos angulares, denominados "passos", quando pulsos elétricos são aplicados em uma determinada sequência nas bobinas do motor. A sequência na qual esses pulsos são aplicados reflete diretamente na direção em que o motor irá girar. A velocidade com que o rotor gira é dada pela frequência de pulsos recebidos, e o tamanho do ângulo rotacionado é diretamente relacionado com o número de pulsos aplicados (BRITES; SANTOS, 2008).

Figura 4: Sistema completo para medidas de antenas.



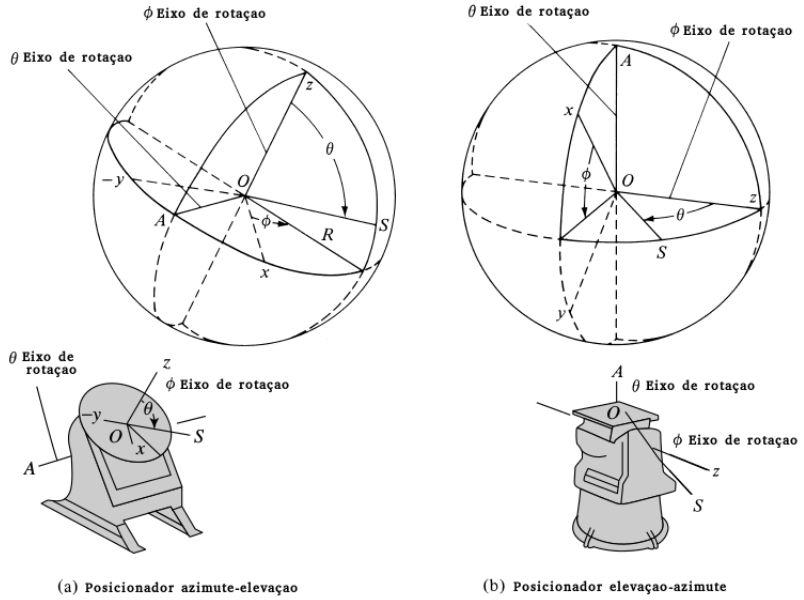
Fonte: Modificado de (BALANIS, 2009).

O funcionamento do motor de passo se dá pela interação entre campos magnéticos. Quando uma bobina é percorrida por uma corrente constante, tem-se como resultado a produção de um campo magnético uniforme no seu interior. Invertendo o sentido da corrente elétrica, inverte-se o sentido do campo magnético. Expandindo a ideia de uma bobina para uma combinação dessas, pode-se controlar a posição de um ímã permanente situado no meio das bobinas, como mostra a Fig. 7.

Existem três tipos de motores de passo

1. Motores de relutância variável (VR – *Variable Reluctance*): Este tipo de motor não possui um ímã permanente no rotor, de forma que este possa girar livremente sem nenhum torque na ausência de energia. Motores VR são pouco utilizados em aplicações industriais. Este tipo não é sensível à polaridade da corrente e necessita de um arranjo de *driver* diferente dos outros modelos. O termo relutância, utilizado para caracterizar este tipo de motor, é devido à propriedade que o material constituinte do dispositivo oferece ao fluxo magnético. Os motores VR se aproveitam do fato de uma peça de material

Figura 5: Juntas rotativas (a)azimute-elevação e (b)elevação-azimute.



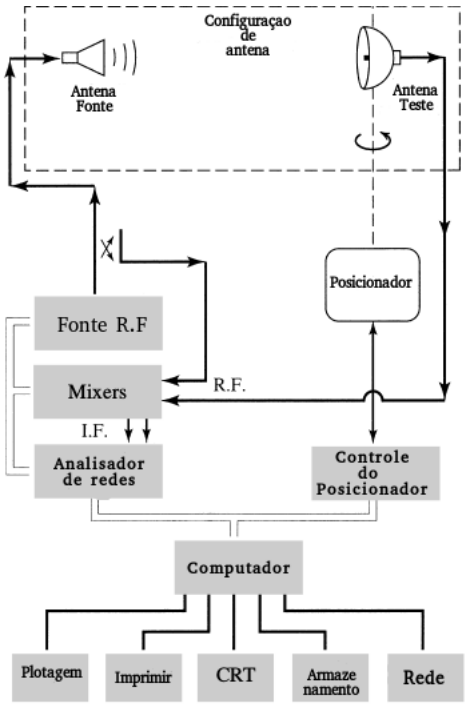
Fonte: Modificado de (BALANIS, 2009).

ferroso sempre se alinhar com a direção na qual a relutância é mínima, quando submetido a um campo magnético (SOUZA, 2012).

2. Motores de ímã permanente (PM – *Permanent Magnet*): É talvez o tipo de motor mais amplamente utilizado para aplicações não industriais. Ele é essencialmente um dispositivo de baixo custo, baixo torque e baixa velocidade, ideal para aplicações em campos como periféricos de informática. A construção do motor resulta em ângulos de passo relativamente grandes, porém, a simplicidade geral permite a produção em larga escala a custo muito baixo. O motor de vão axial, ou disco, é uma variação do projeto de ímã permanente que apresenta um melhor desempenho, em grande parte devido à inércia muito baixa do motor. No entanto, isto restringe as aplicações do motor às que envolvem baixa inércia caso seja exigido todo o desempenho do motor.

3. Motores híbridos (Hb - *Hybrid*): combinam as características dos dois primeiros tipos de motores, e são os mais utilizados na indústria e em aplicações profissionais. A maioria dos motores híbridos é de 2 fases, embora sejam utilizadas versões de 3 e 5 fases. Este tipo de motor apresenta as seguintes características principais: rotor e estator multidentados; o rotor, que é de ímã permanente, é constituído por duas partes

Figura 6: Sistema automatizado por computador.



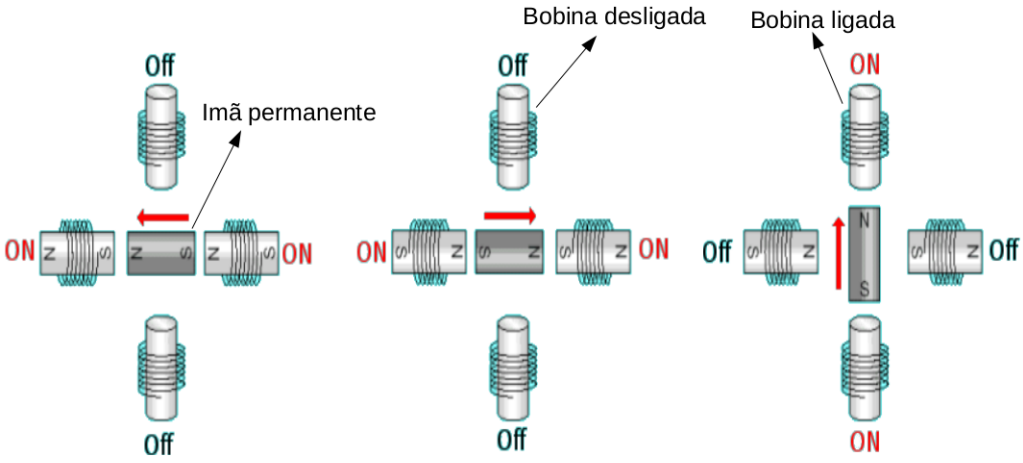
Fonte: Modificado de (BALANIS, 2009).

multidentadas, tem vários polos, e é magnetizado de maneira axial. Para exemplificar o funcionamento de um motor de passo híbrido, é apresentada e detalhada a Fig. 8.

Na Fig. 8(a), a bobina do topo (2b) está ativada, atraindo o dente superior do eixo. Em um segundo passo, a Fig. 8(b) mostra o acionamento da bobina da direita (1a), o que faz com que o quarto dente à direita da referência seja atraído pela bobina 1a. Quando a bobina 2a é ativada, o terceiro passo é dado, como mostra a Fig. 8(c). Por último, a Fig. 8(d) mostra a bobina à esquerda (1b) sendo ativada, rodando novamente o eixo. Quando a bobina do topo (2b) for ativada novamente, o eixo terá rodado em um dente de posição. Como existem 25 dentes no caso exemplificado, serão necessários 100 passos para uma rotação completa, e cada passo corresponde à $3,6°$. A Fig. 9 mostra como é a parte externa (9(a)) e o interior (9(b)) de um motor híbrido real, respectivamente.

Existe ainda uma subclasse que classifica os motores em relação à existência ou não de uma derivação central. Estes são conhecidos como motores bipolares e unipolares. Basicamente, a diferença entre um e outro é que motores bipolares precisam de uma

Figura 7: Controle do posicionamento de um ímã permanente.



Fonte: Modificado de (GONÇALVES; PINTO, 2012).

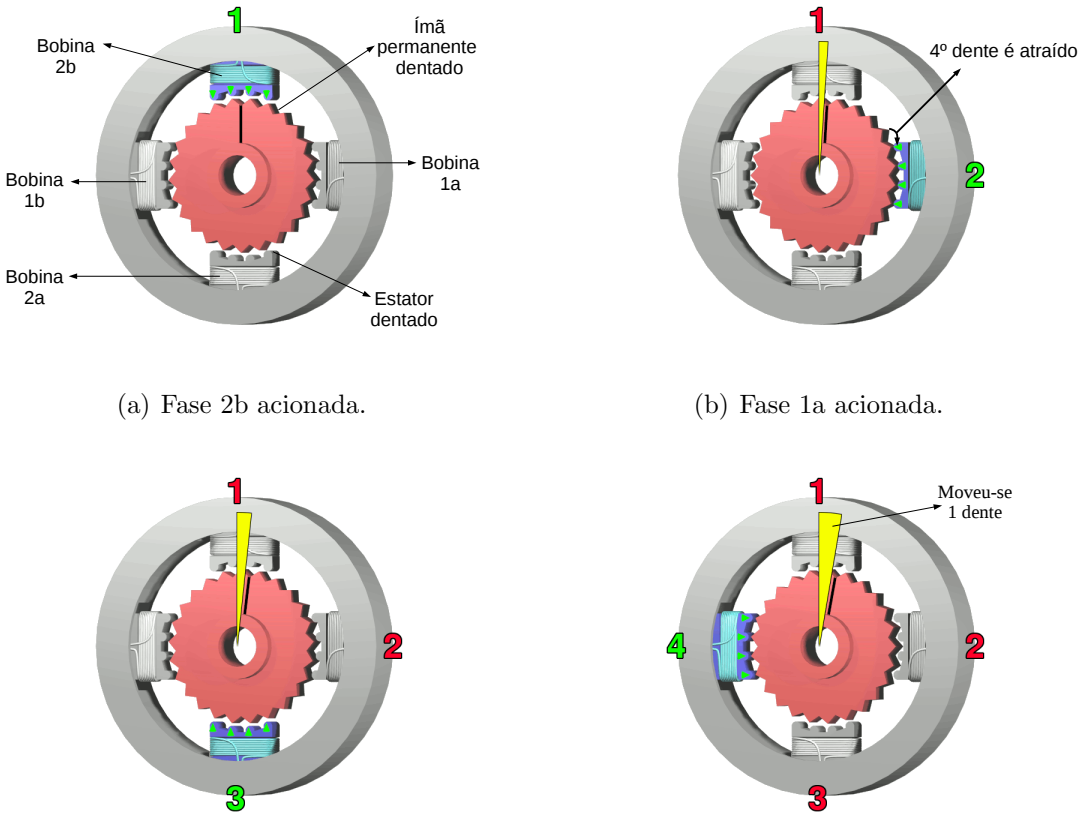
inversão de tensão nas bobinas para controlar o sentido de rotação, ou seja, trabalha com dois pólos de tensão - daí o nome *bipolar*. Isto não acontece nos motores unipolares, em que a presença de uma derivação central permite a divisão da tensão entre as bobinas sem precisar inverter a polaridade (SOUZA, 2012).

2.4.1 Motores unipolares

Os motores de passo unipolares são facilmente reconhecidos pela derivação central em cada uma das bobinas. O número de fases é duas vezes o número de bobinas, uma vez que cada bobina se encontra dividida em duas. Na Fig. 10 tem-se a representação de um motor de passo unipolar de 4 fases (1a, 2a, 1b e 2b). A fase 1a vai da derivação central até a extremidade "a" na bobina 1, e a fase 1b vai da derivação central à extremidade "b", nessa mesma bobina. As fases na bobina 2 se dão de forma análoga à bobina 1.

Normalmente, a derivação central das bobinas é ligada ao positivo da fonte de alimentação, e os extremos de cada bobina são ligados sequencialmente ao terra por um circuito apropriado (controlador e *driver*), conforme o modo de acionamento adotado, para assim produzir o movimento de rotação contínuo numa direção (SOUZA, 2012).

Figura 8: Controle do posicionamento de um motor de passo híbrido.



(a) Fase 2b acionada.

(b) Fase 1a acionada.

(c) Fase 2a acionada.

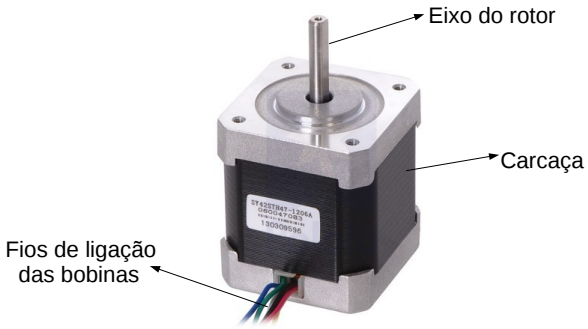
(d) Fase 1b acionada.

2.4.2 Motores bipolares

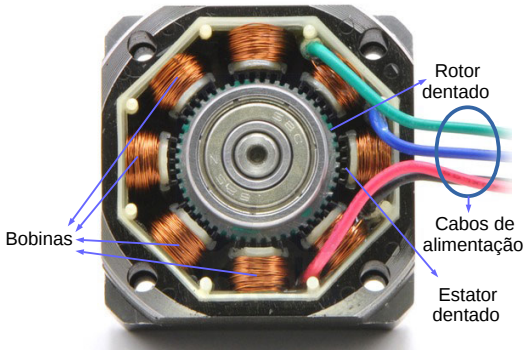
Os motores bipolares são constituídos por bobinas sem derivação central. Por esse fato, essas bobinas devem ser energizadas de tal forma que a corrente elétrica flua na direção inversa a cada dois passos, permitindo assim, o movimento contínuo do rotor, entre outras palavras, a polaridade deve ser invertida durante o funcionamento do motor. Conforme pode ser visto na Fig. 11, existem duas bobinas, 1 e 2. No caso do motor bipolar, agora o número de fases é igual ao número de bobinas que compõem o enrolamento do motor. Então, tem-se as fases 1a e 1b, 2a e 2b.

Os motores de passo bipolares são conhecidos por sua excelente relação tamanho/-torque: eles proporcionam um maior torque, cerca de 40% a mais, quando comparado a um motor unipolar do mesmo tamanho. Isso se deve ao fato de que quando se energiza uma fase, magnetizam-se ambos os polos em que a fase (ou bobina) está instalada. Assim,

Figura 9: Motor de passo híbrido real.



(a) Parte externa.



(b) Parte interna.

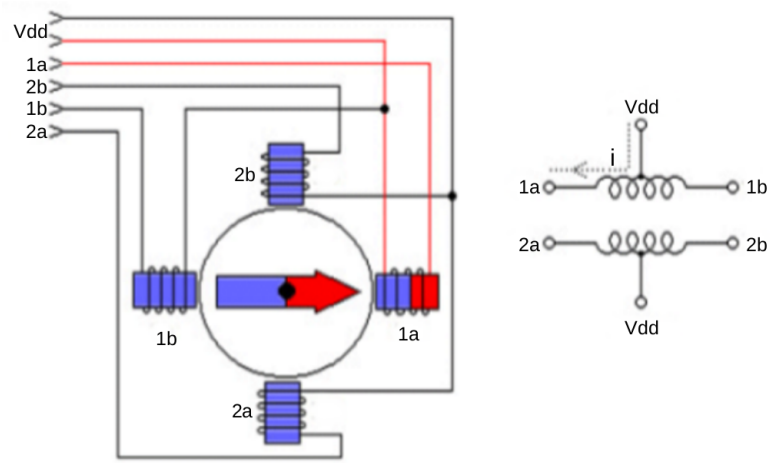
o rotor sofre a ação de forças magnéticas de ambos os polos, ao invés de apenas um, como acontece no motor unipolar (SOUZA, 2012).

2.4.3 Modos de acionamento de motores de passo

Um motor de passo pode ser acionado de quatro formas diferentes de acordo com o comprimento do passo, são elas:

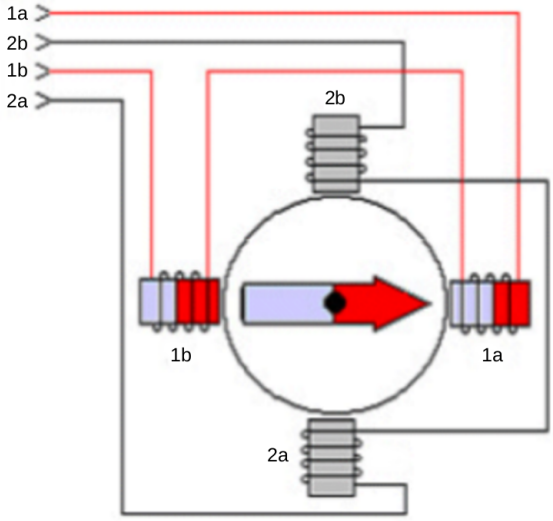
1. Passo Completo (*"Full Step"*): Este modo caracteriza-se pelo fato de que o motor desloca seu rotor em passo completo a cada pulso de acionamento que recebe em suas bobinas, a partir do circuito de potência. Nesse modo, existem ainda dois submodos: modo normal e modo *wave*. No modo completo normal, duas bobinas sequenciais do motor são alimentadas ao mesmo tempo, assim, esse modo possui maior torque e consumo

Figura 10: Esquema elétrico de um motor de passo unipolar



Fonte: Modificado de (GONÇALVES; PINTO, 2012).

Figura 11: Esquema elétrico de um motor de passo bipolar



Fonte: Modificado de (GONÇALVES; PINTO, 2012).

de energia do que o modo *wave*. No modo *wave*, apenas uma bobina é alimentada por vez. As Tabelas 1 e 2, respectivamente, mostram como é feito o acionamento nos modos normal e *wave*, e fazem referência ao motor unipolar da Fig. 10. No exemplo, assume-se

uma lógica positiva para o acionamento das bobinas, ou seja, o nível lógico "1" representa a presença de corrente, enquanto o nível lógico "0" representa a ausência desta.

Tabela 1: Modo normal.

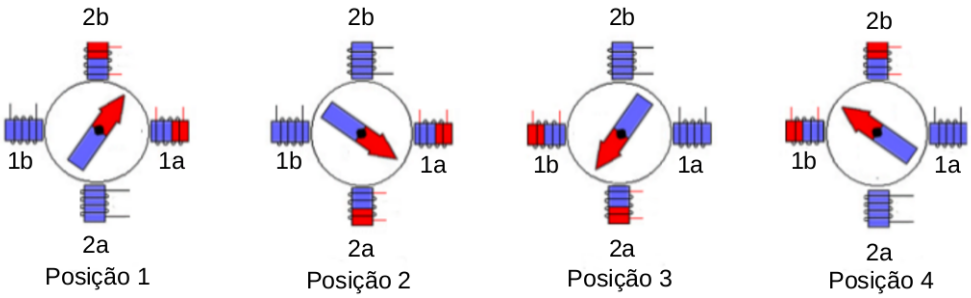
Passo	1a	2a	1b	2b
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1
5	1	0	0	1
6	1	1	0	0
7	0	1	1	0
8	0	0	1	1

Tabela 2: Modo *wave*.

Passo	1a	2a	1b	2b
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	1	0	0	0
6	0	1	0	0
7	0	0	1	0
8	0	0	0	1

As Fig. 12 e 13 ilustram uma volta inteira (360°), de um motor unipolar em passo completo, para os submodos normal e *wave*, respectivamente. A Fig. 14 mostra uma volta inteira (360°) no modo passo completo para motores bipolares, onde percebe-se claramente a troca de polaridade entre as bobinas para gerar a sequência de giro do motor (SOUZA, 2012).

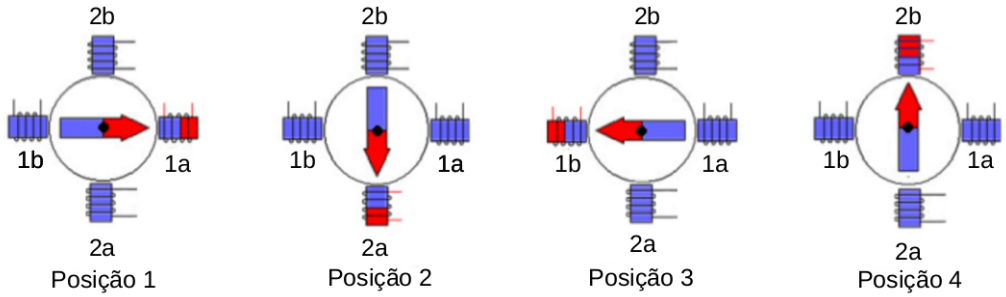
Figura 12: Giro de 360° em um motor unipolar em passo completo normal.



Fonte: Modificado de (GONÇALVES; PINTO, 2012).

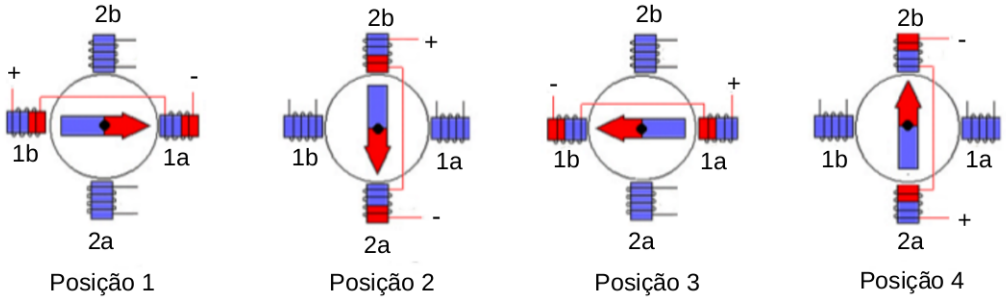
2. Meio-Passo (*"Half-Step"*): Outro tipo de acionamento possível consiste em energizar, alternadamente, uma e duas bobinas. Este processo permite deslocar o rotor em meio passo de cada vez, denominado Meio-Passo (*"Half-Step"*). Neste modo de acionamento o número de "passos" é duplicado, ou seja, o motor possui maior resolução

Figura 13: Giro de 360° em um motor unipolar em passo completo *wave*.



Fonte: Modificado de (GONÇALVES; PINTO, 2012).

Figura 14: Giro de 360° em um motor bipolar em passo completo.



Fonte: Modificado de (GONÇALVES; PINTO, 2012).

em termos de ângulo de rotação. Na Tabela 3 é apresentada a sequência de acionamento no modo *half-step* para o motor unipolar da Fig. 10.

Levando em conta o que já foi abordado anteriormente para os modos de acionamento, no modo *"Half-Step"* os passos serão alternadamente fortes e fracos (em termos de torque). Isso não significa uma limitação importante no desempenho do motor - o torque disponível é obviamente limitado pelo passo mais fraco.

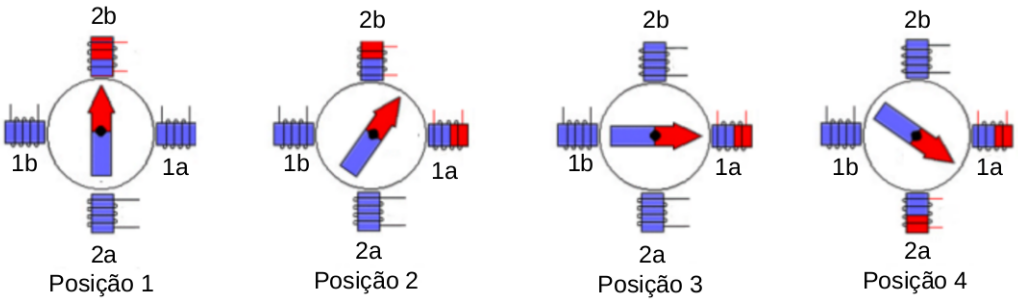
Certamente, o que se deseja é produzir um torque aproximadamente igual em todos os passos, e esse torque deveria estar ao nível do passo mais forte. Pode-se obter isso empregando um nível mais alto de corrente quando houver apenas uma fase energizada. Isso não provoca dissipação excessiva de potência do motor, implicando em aumento excessivo da temperatura da carcaça desse motor, pois a classificação de corrente do

Tabela 3: Sequência de acionamento das bobinas de um motor de passo em modo de meio passo *"Half-Step"*.

Passo	1a	2a	1b	2b
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1
9	1	0	0	0
10	1	1	0	0
11	0	1	0	0
12	0	1	1	0
13	0	0	1	0
14	0	0	1	1
15	0	0	0	1
16	1	0	0	1

fabricante supõe que duas fases estejam sendo energizadas (a classificação de corrente se baseia na temperatura permissível na carcaça). Com apenas uma fase energizada, dissipar-se-á o mesmo total caso a corrente seja elevada em 40%. Empregando essa corrente mais alta no estado de apenas uma fase ligada, produz-se um torque aproximadamente igual nos passos alternados. A Fig. 15 mostra quatro passos de um motor unipolar em modo *Half-Step*, onde é possível notar um menor ângulo de precisão no giro do motor.

Figura 15: 4 passos em um motor unipolar em modo de meio passo.



Fonte: Modificado de (GONÇALVES; PINTO, 2012).

3. Micro-Passo (*Microstepping*): Verificou-se anteriormente que se duas fases forem

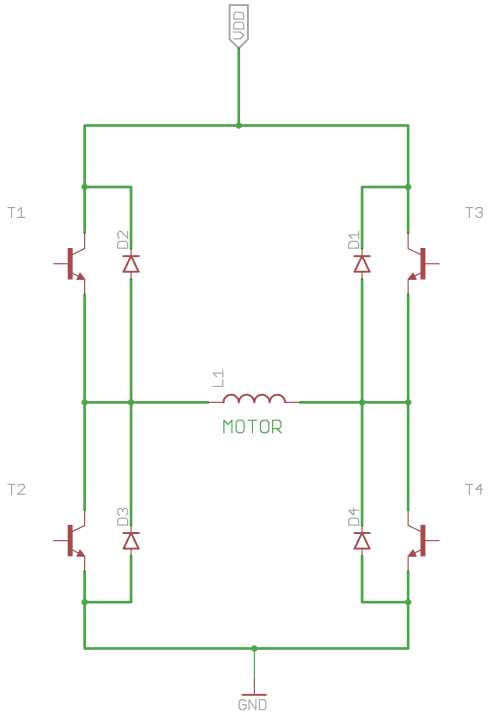
energizadas com correntes iguais, produz uma posição de passo intermediária a meio caminho entre as posições em que há uma fase única ligada. Caso as correntes nas duas fases sejam desiguais, a posição do rotor será deslocada em direção ao polo mais forte. Este efeito é empregado no *driver* de micro-passo, que subdivide o passo básico do motor estabelecendo uma escala proporcional da corrente nas duas fases. Dessa forma, o tamanho do passo é diminuído e a suavidade do movimento em baixas velocidades é sensivelmente melhorada. Os *drivers* de micro-passo de alta resolução dividem o passo do motor em até 500 micro-passos, propiciando 100.000 passos por revolução no caso de um motor de passo com 200 passos por volta (tipo mais comum no mercado). Nessa situação, o padrão de corrente nas fases é muito semelhante a ondas senoidais com um deslocamento de fase de 90° entre elas.

2.4.4 Acionamento de motor de passo

Geralmente, deseja-se acionar um motor de passo através de algum dispositivo de controle, como um microcontrolador ou um Arduino. No entanto, esses dispositivos fornecem baixas correntes, e, se um motor de passo for acionado diretamente através de uma das saídas de um Arduino, por exemplo, a chance de queimar a saída é bastante grande. Para solucionar o problema, existem os chamados *drivers* de acionamento de motor de passo. Esses *drivers* são constituídos de transistores ligados às bobinas do motor, e podem ser construídos manualmente ou adquiridos em circuitos integrados. O mais conhecido dos *drivers* de acionamento é o circuito *Ponte H*, que tem a forma da letra H como mostra a Fig. 16.

Para acionar a parte esquerda da bobina, na Fig. 16, os transistores $T1$ e $T4$ devem ser acionados simultaneamente, e os transistores $T3$ e $T2$ devem permanecer "abertos". Para acionar a parte direita da bobina, os transistores $T3$ e $T2$ são ligados simultaneamente, enquanto $T1$ e $T4$ permanecem desligados. Deste modo, é possível controlar a alimentação das bobinas de um motor de passo, além de garantir que uma corrente alta possa alimentar o motor com a escolha apropriada dos transistores. Os diodos no circuito da Fig. 16 são conhecidos como diodos "roda livre", e são utilizados para a proteção dos transistores e do circuito. Quando uma bobina é desenergizada, a corrente não para de fluir instantaneamente, uma vez que uma bobina nada mais é do que um indutor, cujo efeito é armazenar campo magnético. Esta corrente pode ter picos muito elevados, e tende a descarregar no circuito. Com o diodo *roda livre*, a corrente restante, após uma bobina ser desenergizada, fica em um *loop* constante entre o transistor e o diodo, até ser dissipada completamente.

Figura 16: Esquema elétrico de um circuito de acionamento ponte *H*.



Fonte: Modificado de (GONÇALVES; PINTO, 2012).

Os transistores do circuito ponte H da Fig. 16, podem ser acionados através de circuitos lógicos, microcontroladores, Arduino ou qualquer outro circuito ou dispositivo que forneça um nível lógico alto em sua saída. A lógica de acionamento dos transistores pode ser controlada por *software*. Vale salientar, que o circuito ponte H não é o único circuito de acionamento de motores de passo, e, ele está aqui descrito, meramente para fins explicativos. O próprio método utilizado neste trabalho, difere do circuito ponte H. Neste projeto, é utilizado um circuito de potência contendo um transistor para cada fase das bobinas do motor de passo, e, a construção deste *driver* está descrita na seção 3.2.

2.5 Arduino

Arduino é uma plataforma de prototipagem eletrônica, criado por Massimo Banzi e David Cuartielles em 2005 com objetivo de permitir o desenvolvimento de controle de sistemas interativos, de baixo custo e acessível a todos.

O projeto foi criado pensando em pessoas leigas no assunto, mas que gostariam de desenvolver suas ideias, ou seja, não é necessário ter conhecimentos prévios em eletrônica ou programação para iniciar-se no mundo Arduino.

Com essa plataforma é possível também enviar e receber informações de praticamente qualquer outro sistema eletrônico. Assim, é possível construir, por exemplo, um sistema de captação de dados de sensores, como temperatura e iluminação, e posteriormente processar e enviar esses dados para um sistema remoto.

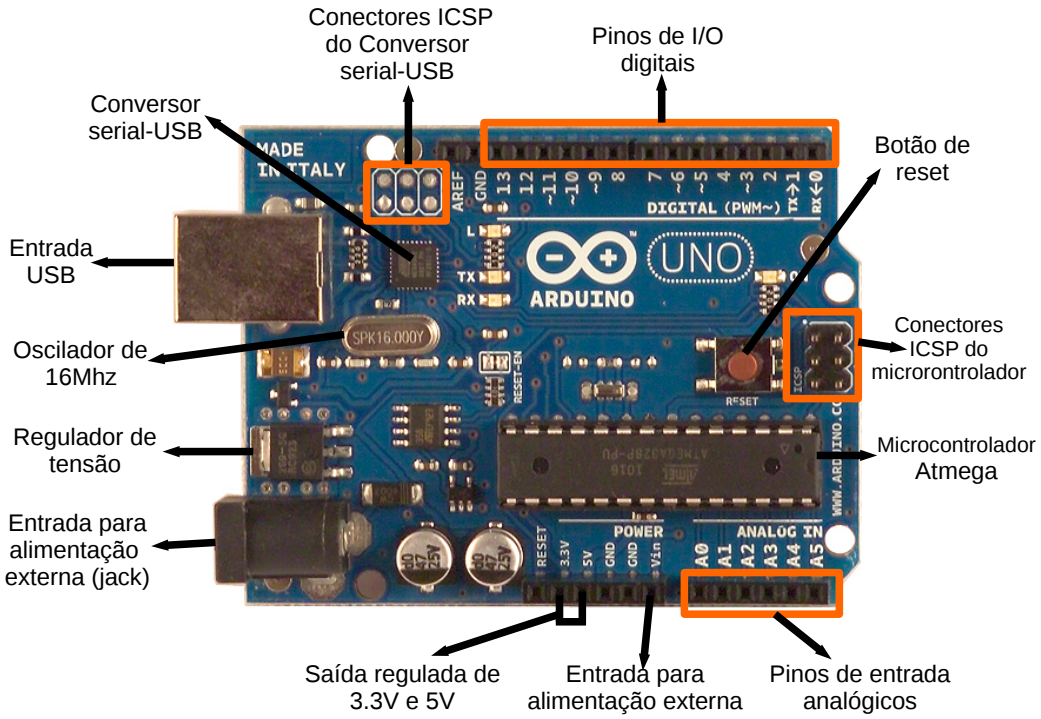
Outra característica importante, é que todo material (*software*, bibliotecas, *hardware*) é *open-source*, ou seja, pode ser reproduzido e usado por todos sem a necessidade de pagamento de *royalties* (direitos autorais) (FILHO, 2012).

A plataforma é composta essencialmente de duas partes: O *Hardware* e o *Software*. O *hardware*, com destaque em seus principais componentes, pode ser visualizado na Fig. 17, que trás como exemplo uma placa Arduino UNO.

O Arduino UNO mostrado na Fig. 17 possui uma placa de microcontrolador baseado no *ATmega328P*. Dispõe de 14 pinos digitais de entrada/saída (dos quais 6 podem ser usados como saídas PWM (*Pulse Width Modulation*), 6 entradas analógicas, um cristal oscilador de quartzo de 16 MHz, uma conexão USB, uma tomada (jack) de energia, dois conectores ICSP (*In Circuit Serial Programmer*) (CAMPOS, 2015a) e um botão de *reset*. Ele contém todo o necessário para apoiar o microcontrolador, basta conectá-lo a um computador com um cabo USB ou ligá-lo com um adaptador AC-CC ou bateria. A placa Arduino em si não possui qualquer recurso de rede (exceto na versão Arduino *ethernet*), porém é comum combinar o Arduino com extensões apropriadas chamadas de *shields*.

É importante salientar, que Arduino não é um microcontrolador, e sim uma plataforma de desenvolvimento que utiliza os microcontroladores da *ATMEL* (*atmega*). Para fins de diferenciação entre ambos, pode-se dizer que para um microcontrolador executar uma tarefa como, por exemplo, fazer uma leitura analógica, ativar um transistor que ativa um relé e acende uma lâmpada, acionar um motor ou se comunicar com algum módulo ou dispositivo, é preciso programá-lo. Para tanto, pode-se utilizar linguagens de programação como *Assembly*, C, C++ e em alguns casos até mesmo *Visual Basic*; a linguagem mais completa e recomendável é a linguagem C. Depois de programar o microcontrolador, é preciso montar o seu circuito: só para alimentar o microcontrolador será preciso um regulador de tensão, cristal oscilador, capacitor para desacoplamento, e uma fonte de alimentação. Para pessoas com pouca ou nenhuma experiência em eletrônica

Figura 17: Hardware de um Arduino UNO.

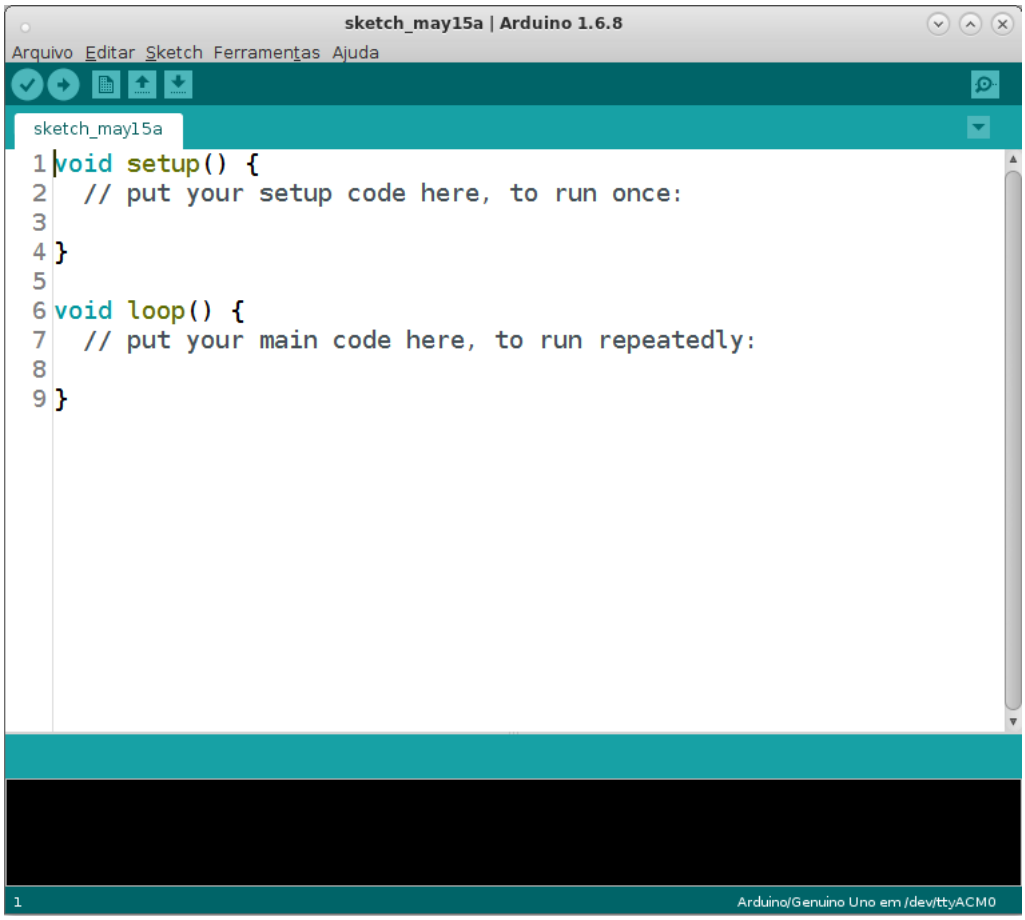


Fonte: Modificado de Soares (2013).

e programação, isso se torna um problema, pois além de ter que montar todo o circuito do microcontrolador em uma placa ou *protoboard*, ainda é preciso montar o circuito do projeto. Para resolver essa questão, o Arduino foi desenvolvido: para facilitar a prototipagem, não só de técnicos em eletrônica e programadores, mas também de pessoas comuns, que querem automatizar seus projetos mas não possuem muito conhecimento na área. O Arduino é uma plataforma de desenvolvimento composta pelo *Hardware* (Arduino UNO, Mega UNO, Duemilanove, Leonardo, Nano, etc) e uma IDE (*Integrated Development Environment* - ambiente de desenvolvimento integrado) onde é possível escrever o código em linguagem C/C++ e a própria IDE fará todo o processo de compilação e transferência do código do computador para o microcontrolador *Atmega*, o que torna todo o processo de programação e prototipagem muito mais rápido e simples (TACIO, 2013). Outra característica interessante do Arduino é a sua vasta documentação encontrada em fóruns, *blogs*, e *sites* em geral.

Outra grande vantagem do Arduino é que sua IDE é multi-plataforma (programada em Java), ou seja, pode ser instalada nos sistemas operacionais *Windows*, *Linux* e *MAC OS X*. Por ser uma IDE dedicada ao Arduino, ela possui nativamente diversos exemplos de código e opções, como por exemplo, passar o *bootloader* (ARDUINO, 2016b)-(ELETRONICO, 2014) de um Arduino para um microcontrolador *Atmega* novo e vários outros exemplos de utilização dos recursos do *Hardware*. A Fig. 18 mostra como é a IDE do Arduino, que está na sua versão 1.6.9 até a data da escrita deste trabalho.

Figura 18: IDE do Arduino.



Como já mencionado, a linguagem de programação adotada pela plataforma Arduino, é a linguagem C/C++, que é estruturada e possui uma sintaxe limpa, clara e objetiva. Para facilitar mais ainda, o Arduino já tem diversas funções prontas, como, por exemplo, para leitura e escrita de dados analógicos e digitais, além de comunicação serial (PPM, I2C, TTL, etc) e paralela. Sendo assim, não é preciso se preocupar com o *clock*

do microcontrolador e nem implementar protocolos de comunicação.

As duas funções na Fig. 18, **void setup()** e **void loop()**, são obrigatórias no Arduino. A função **void setup()** não retorna qualquer tipo de dado (*null*) e nem recebe qualquer argumento, essa função é utilizada para configuração dos pinos como I/O (*Input/Output* - Entrada/Saída), declaração de variáveis e outras configurações. A função **void loop()** também não retorna e nem recebe qualquer tipo de argumento; quando se roda um programa (*firmware*) no microcontrolador, é de interesse que este programa fique rodando para sempre, em um *loop* infinito. Para tanto, deve-se escrever as rotinas do *firmware* dentro da função *loop*, pois esta será repetida infinitamente.

O baixo custo da placa Arduino é também outra grande vantagem em relação a outras plataformas de desenvolvimento. A placa Arduino Uno, por exemplo, custa de R$ 55,00 a R$ 100,00, e a IDE é gratuita e *open source*. A Tabela 4 traz algumas especificações da placa Arduino UNO Rev. 3.

Tabela 4: Especificações de uma placa Arduino UNO Rev. 3.

Microcontrolador	ATmega328P
Tensão operacional	5V
Tensão de entrada (recomendado)	7-12V
Tensão de entrada (limite)	6-20V
Pinos Digitais de I/O	14 (dos quais 6 fornecem saída PWM)
Pinos de entrada analógica	6
Corrente DC por Pino de I/O	20mA (recomendado) - 40mA (máx.)
Corrente DC por pino para 3.3V	50mA
Memória Flash	32 KB (ATmega328P) - 0,5 KB para o bootloader
SRAM	2KB (ATmega328P)
EEPROM	1KB (ATmega328P)
Velocidade do *clock*	16 MHz
Comprimento	68,6 mm
Largura	53,4 mm
Peso	25 g

2.6 Analisadores de espectro

O analisador de espectro é um dispositivo eletrônico utilizado para a análise de sinais no domínio da frequência. Possui certa semelhança com um osciloscópio, uma vez que o resultado da medida é apresentado em uma tela, mostrando os dados de amplitude na vertical e os dados de frequência na horizontal. A grande diferença entre ambos, é que o osciloscópio mede amplitudes no âmbito temporal, enquanto o analisador de espectro trabalha no domínio da frequência.

Um analisador de espectro é essencialmente um receptor de rádio passivo, com uma interface gráfica (*display*) para a análise e medida do sinal no domínio da frequência. Esses dispositivos indicam, geralmente, a informação contida no sinal de forma direta, tais como a tensão, a potência, o período e a frequência. Analisadores de espectro são capazes de fazer uma grande variedade de medições, e isso significa que eles são uma ferramenta inestimável para os laboratórios de desenvolvimento de *design* de RF (*Radio Frequency*), bem como têm muitas aplicações para o serviço de campo especializado.

A maneira mais natural de olhar para formas de onda é no domínio do tempo - olhando como um sinal varia em amplitude enquanto o tempo avança. No entanto, essa não é a única maneira pela qual os sinais podem ser exibidos. A análise espectral de um sinal fornece informação adicional difícil de ser obtida numa análise temporal. Por exemplo, ao analisar-se um sinal senoidal levemente distorcido, em função do tempo dificilmente se percebe essa imperfeição. Na análise no domínio da frequência, pequenas distorções e imperfeições (que implicam em componentes de frequência diferentes) são facilmente identificadas, pois cada componente de frequência é visualizada separadamente.

As escalas de um analisador de espectro são, em geral, logarítmicas, o que facilita a leitura de sinais de baixa amplitude. Assim, a amplitude pode ser diretamente lida em dB (decibéis - unidade mais usual em sistemas de comunicação), e na escala horizontal, um amplo espectro de frequências pode ser visualizado simultaneamente.

Os analisadores de espectro também podem medir parâmetros de modulação, distorção e ruído: em sistemas de comunicação via rádio é fundamental a análise dos níveis de potência relativos à cada frequência, do grau e da qualidade de modulação, da largura de banda ocupada no espectro, entre outros parâmetros; sistemas supostamente lineares (amplificadores, transmissores e receptores de rádio, filtros, etc) apresentam sempre um certo grau de não linearidade, gerando consequentemente, distorções no sinal (harmônicas, intermodulação, emissões espúrias); todo circuito ou elemento ativo gera ruído, tipicamente em uma faixa larga de frequências. Medidas como figura de ruído e relação sinal/ruído são importantes na caracterização de sistemas eletrônicos ou dispositivos (BONFIM, 2003).

Basicamente, existem três tipos de analisadores de espectro, são eles: banco de filtros, FFT (*Fast Fourier Transform* - transformada rápida de fourier) e analisador por varredura ou heteródino. O detalhamento dos tipos de analisadores de espectro, bem como o modelo utilizado para este projeto, podem ser consultados no APÊNDICE A.

2.7 Geradores de sinais

Geradores de sinais, também conhecidos como geradores de funções, são aparelhos eletrônicos utilizados para gerar sinais elétricos de formas de onda, frequências (de alguns Hz ou KHz a alguns GHz) e amplitudes (tensões) diversas. São muito utilizados em laboratórios de eletrônica como fonte de sinal para teste de diversos aparelhos e equipamentos eletrônicos, como antenas, por exemplo.

Um gerador de sinais pode ser capaz de gerar sinais senoidais, triangulares, quadrados, e dependendo do modelo, até sinais dente-de-serra. Normalmente, geradores de sinais possuem um frequencímetro acoplado e diversos botões de ajuste e seleção, além de conectores para saída do sinal. Seu uso é muito ligado à utilização do osciloscópio, com o qual se pode verificar as suas formas de onda. Também é bastante utilizado em conjunto com analisadores de espectro, como em um sistema de medição e teste de antenas, por exemplo. Seu funcionamento é baseado em circuitos eletrônicos integrados, osciladores, filtros e amplificadores. As características fundamentais dos geradores de sinais são:

- **Tipos de sinais fornecidos**: Os sinais variam de modelo para modelo. Dentre os tipos de sinais mais comuns fornecidos por um gerador, os mais utilizados são os que apresentam formas de onda: senoidal, quadrada e triangular.

- **Faixa de frequência**: O gerador de sinais fornece, geralmente, sinais em uma frequência que vai de alguns KHz a vários MHz - geradores atuais podem chegar a mais 6 GHz. Os manuais dos fabricantes informam a faixa de frequência que o equipamento pode fornecer. Por exemplo, de 250 KHz a 3 GHz.

- **Tensão máxima de pico-a-pico na saída**: A tensão máxima de pico-a-pico é o valor máximo de amplitude do sinal que o gerador pode fornecer.

- **Impedância de saída (ou Resistência de saída)**: A impedância de saída é a impedância que o gerador apresenta entre os terminais de saída. Os geradores podem ser de: alta impedância de saída, para circuitos a válvula; média impedância de saída, para circuitos transistorizados (geralmente, sua impedância é de 600 Ω); baixa impedância de saída, para trabalhos em circuitos digitais (em geral, sua impedância de saída fica em torno de 50 Ω). É importante conhecer as características do gerador de funções, porque isso permite obter a máxima transferência de potência entre gerador e carga.

O gerador de sinais utilizado para o presente trabalho, é o modelo E4438C - *ESG Vector Signal Generator* - da antiga Agilent Technologies, hoje conhecida como Keysight Technologies (consultar o APÊNDICE B para o detalhamento das funcionalidades do dispositivo mencionado). Esse gerador possui, entre muitas outras, as seguintes características:

- Frequência de saída de 250 kHz a 6 GHz;
- Controle de nivelamento automático (ALC);
- Calibração de energia no modo ALC-off;
- Oscilador de referência de 10 MHz com saída externa;
- Interfaces de comunicação GPIB, RS-232, e 10Base-T LAN;
- Modulação em fase;
- Modulação por pulso;
- Entradas de modulação externa para AM, FM e ΦM;
- Configuração de modulações simultâneas;
- Baixa impedância de saída (50Ω), 0 a 3 V_p;
- Formas de onda selecionáveis: seno, onda quadrada, rampa, onda triangular, ruído, swept-seno, dual-seno e pulso;
- Taxas de modulação de frequência variável.

2.8 SCPI (*Standard Commands for Programmable Instruments*)

Na década de 1960, surgiram os primeiros instrumentos de medida comerciais controlados por computador. Entretanto, cada fabricante possuía seu próprio protocolo de comunicação para o instrumento e, é claro, uma interface proprietária. Em 1975, o *IEEE - Institute of Electrical and Electronic Engineers* - aprovou o padrão IEEE 488-1975, que define uma interface elétrica e mecânica padrão para conectores e cabos. Também padronizou o estabelecimento de conexão, endereçamento e protocolo geral para a transmissão de bytes individuais de dados entre os instrumentos e computadores. A norma foi atualizada para *IEEE 488.1-1987*, e embora tivesse resolvido o problema de

transmissão de dados (*bytes*) entre instrumento e computador, o IEEE 488 não especificou como esses dados deveriam ser escritos. Deste modo, cada fabricante inventava comandos diferentes.

No início da década de 1980, criaram-se padrões adicionais para especificar como interpretar os dados enviados via IEEE 488. Em 1987, o *IEEE* lança o IEEE 488.2-1987: códigos e formatos, protocolos e comandos mais comuns para uso com IEEE 488.1-1987. Esta norma define os papéis dos instrumentos e controladores de um sistema de medição e um esquema estruturado de comunicação. Em particular, IEEE 488.2 descreve como enviar comandos para instrumentos e como enviar as respostas aos controladores (computadores). Definiram-se alguns comandos usados com mais frequência, porém cada fabricante de instrumento poderia nomear quaisquer outros tipos de comandos e definir a sua função.

O IEEE 488.2 especificou como certos tipos de recursos devem ser implementados, se eles fossem ser incluídos em um instrumento. Ele geralmente não especifica quais recursos ou comandos devem ser implementados para um instrumento em particular. Assim, foi possível que dois instrumentos similares pudessem estar em conformidade com o IEEE 488.2 e, ao mesmo tempo, poderiam ter um conjunto de comandos totalmente diferentes.

Comandos Padrão para Instrumentos Programáveis (SCPI) é uma linguagem de comando para controlar instrumentos, que vai além do IEEE 488.2 e pode tratar de uma grande variedade de funções do instrumento de uma forma padrão. SCPI promove a consistência, do ponto de vista de programação remota, entre os instrumentos da mesma classe e entre instrumentos com a mesma capacidade funcional. Para um dado parâmetro como a frequência ou a tensão, SCPI define o conjunto de comandos específicos que estão disponíveis para essas funções. Assim, dois osciloscópios construídos por fabricantes diferentes podem ser usados para fazer medições de frequência da mesma maneira. Comandos SCPI são fáceis de aprender, auto-explicativos e podem ser usados por qualquer tipo de usuário. Uma vez familiarizado com a organização e a estrutura de SCPI, consideráveis ganhos de eficiência podem ser alcançados durante o desenvolvimento do programa de controle, independente da linguagem de programação utilizada.

O link de comunicação físico não é definido pelo padrão SCPI, embora originalmente criado para o protocolo GPIB (IEEE-488), ele também pode ser usado com RS-232 , *Ethernet* , USB , VXIbus , HiSLIP , etc (ZHU; LI; LI, 2008).

Comandos SCPI são textos *ASCII*, que são enviados para o instrumento através da

camada física (por exemplo, **IEEE-488**). São uma série de uma ou mais palavras-chave, muitas das quais recebem parâmetros. Na especificação, palavras-chave são escritas como em "CONFigure": toda a palavra-chave pode ser usada, ou pode ser abreviada apenas pela parte em maiúsculo. As respostas a comandos de consulta são tipicamente cadeias *ASCII*, no entanto, para grandes quantidades de dados, formatos binários podem ser utilizados.

Um instrumento pode realizar uma série de operações através de comandos SCPI ou responder a uma simples consulta, como a leitura de uma tensão, por exemplo. Consultas são enviadas a um instrumento acrescentando-se um ponto de interrogação ao fim de um comando. Alguns comandos podem ser utilizados tanto para operações, quanto para consultar um instrumento. Por exemplo, o modo de aquisição de dados de um instrumento pode ser definido usando o comando "ACQuire:MODe", ou pode ser consultado usando o comando "ACQuire:MODe?". Alguns comandos podem realizar funções e consultar um instrumento ao mesmo tempo. Por exemplo, o comando "*CAL?" executa uma rotina de auto-calibração em alguns equipamentos e, em seguida, devolve os resultados da calibração.

Comandos semelhantes estão agrupados em uma hierarquia ou estrutura ("árvore"). Por exemplo, qualquer instrução para ler uma medição de um instrumento começará com "MEASure". Sub-comandos específicos dentro da hierarquia são aninhados com dois pontos (:). Por exemplo, o comando para "medir uma tensão DC" seria da forma "MEASure:VOLTage:DC?", e o comando para "medir uma corrente AC" seria igual a "MEASure:CURRent:AC?".

Alguns comandos requerem argumentos adicionais. Argumentos são escritos após o comando, e são separados por um espaço. Por exemplo, o comando para definir o modo de *Trigger* de um instrumento para "normal", pode ser escrito como "TRIGger:MODe NORMal", onde a palavra "NORMal" é usada como argumento para o comando "TRIGger:MODe".

Múltiplos comandos podem ser enviados para um instrumento em uma única cadeia de caracteres, basta separar os comandos com um caractere ponto e vírgula (;). Por exemplo, o comando para "medir uma tensão DC", em seguida, "medir uma tensão AC", seria enviado como "MEASure:VOLTage:DC?;:MEASure:CURRent:AC?". Comandos simples que começam com dois pontos (:) são interpretados em relação à raiz da árvore de comando. Caso contrário, eles referem-se implicitamente ao último nó do comando anterior (a menos que eles comecem com um asterisco). Por exemplo,

":SOURce:FREQuency:STARt 100;STOP 200" é uma forma abreviada do comando ":SOURce:FREQuency:STARt 100;:SOURce:FREQuency:STOP 200".

Como já foi dito, a sintaxe dos comandos SCPI mostra alguns caracteres maiúsculos e outros minúsculos. Para abreviar os comandos, basta o envio de apenas os caracteres maiúsculos. Por exemplo, o comando "SYSTem:COMMunicate:SERial:BAUD 2400", que estabelece uma interface de comunicação RS-232 com *baudrate* de 2400 bits/s, poderia ser substituído pela sua forma abreviada "SYST:COMM:SER:BAUD 2400". O comando de consulta "SYSTem:COMMunicate:SERial:BAUD?", instrui o instrumento a relatar a sua taxa de transmissão atual.

Os comandos e parâmetros SCPI são enviados a partir de um controlador para um instrumento utilizando interfaces IEEE 488.1, VXIbus, RS-232,*Ethernet*, etc. Instrumentos que aceitam SCPI são muito flexíveis, e podem receber uma variedade de formatos de comandos e parâmetros, o que torna o aparelho mais fácil de programar. As respostas dos instrumentos, enviadas de volta para o controlador, podem ser dados ou informações de *status*. O formato SCPI de resposta do instrumento a uma consulta particular é bem definido, e reduz o esforço de programação. Informações de dados podem ser formatadas de modo que seja independente do dispositivo e da medição (CONSORTIUM et al., 1999).

Na Tabela 5 são apresentados os comandos obrigatórios, os quais todos os instrumentos que implementam SCPI devem possuir.

Tabela 5: Comandos obrigatórios em instrumentos com SCPI.

Comando	**Função**
∗CLS	Limpar o estado atual do comando
∗ESE	Habilita o *status* de evento padrão
∗ESE?	Consulta o *status* de evento padrão
∗ESR?	Consulta o registrador de *status* de evento padrão
∗IDN?	Consulta a identificação do instrumento
∗OPC	Comando de operação completa
∗OPC?	Consulta se a operação foi completada
∗RST	Comando de reset (reiniciar)
∗SRE	Ativa a solicitação de serviço
∗SRE?	Consulta se a solicitação de serviço está ativa
∗STB?	Ler e retornar o byte status
∗TST?	Realiza um teste interno e retorna o resultado
∗WAI	Aguarda pela continuação do comando

2.9 Redes *Ethernet* e protocolo TCP/IP

O propósito geral dos primeiros sistemas computacionais era agilizar o processamento de informações a fim de se obter maior produtividade em atividades repetitivas, mantendo sempre a qualidade e a baixa probabilidade de erros. Ao longo dos anos, percebeu-se que a computação estava sendo sub-utilizada, e que se poderia usufruir da mesma de maneira colaborativa. Assim, haveria redução de custos para as empresas, uma vez que as máquinas das corporações compartilhariam recursos e *softwares*, e todo o gerenciamento poderia ser feito por um único computador mais poderoso, o **servidor**.

Foi considerando as vantagens citadas anteriormente, que surgiu a ideia de rede de computadores, ou rede de comunicação de dados. Nos primórdios das redes de computadores, tanto o *hardware* quanto o *software* eram proprietários, ou seja, dispositivos de fabricantes diferentes não conseguiam se comunicar entre si. Nos dias atuais, a tecnologia de redes é aberta, o que possibilitou a criação da **Internet**.

Uma rede de computadores é definida como sendo um conjunto de linhas e nós, em que cada nó pode ser representado por um dispositivo pertencente à rede, e linha é o meio físico (canal) que interliga os nós.

No começo, as redes públicas foram criadas para realizar apenas um tipo de serviço (monosserviço), a saber, o tráfego de voz para telefonia. Entretanto, com o desenvolvimento tecnológico e a demanda do mercado, as redes acabaram tendo que se adaptar e implementar o tráfego de dados, tornando-se redes de acesso a dados e redes de *backbone* (multisserviço).

Com o surgimento das redes de computadores, também surgiram os processos compartilhados, ou seja, um computador de maior poder de *hardware* (servidor) é responsável pelo gerenciamento dos demais computadores da rede. Para isso, apenas é preciso instalar um *software* de controle nesta máquina, e os outros dispositivos precisam ter permissões de acesso à ela. Além disso, os computadores interligados em uma rede, compartilham os periféricos instalados no computador de controle, o que é conhecido como computação colaborativa.

No que se refere à distância de abrangência, as redes de computadores podem ser classificadas como:

- **Rede LAN (*Local Area Network*)**: Interligam equipamentos à pequenas distâncias - cerca de dezenas de metros, no máximo - são utilizadas em redes internas

de empresas, estabelecimentos e redes caseiras de pequeno porte;

- **Rede MAN (*Metropolitan Area Network*)**: Interligam equipamentos à distâncias de centenas de metros ou até algumas unidades de quilômetros;
- **Rede WAN (*Wide Area Network*)**: Interligam equipamentos à distâncias de dezenas ou até centenas de quilômetros. Esse é o tipo de rede da internet - a rede mundial de computadores.

O comitê 802 do *IEEE* mantém três padrões para a implementação de redes LAN, que são:

- *Ethernet*: Rede aberta para qualquer tipo de dispositivo;
- *Token Ring*: Tipo de rede para dispositivos IBM;
- *Arc Net*: Padrão antigo e obsoleto.

O padrão que mais se difundiu foi o *Ethernet*, e este é o padrão utilizado na atualidade. Os motivos que levaram à sua grande aceitação são: baixo custo; boas velocidades de tráfego de dados; tecnologia conhecida e de fácil implementação; capacidade de trabalhar com um grande número de equipamentos.

Uma rede de comunicação é formada por no mínimo dois dispositivos interconectados. À medida que os sinais são tratados nestas máquinas, eles ficam disponíveis nos terminais de saída das mesmas, garantindo assim, que possa haver troca de informações entre elas. A informação trafega pela rede efetivamente através de dados binários (*bits*), que nada mais são do que níveis de tensão alto (para representar o *bit* 1) e baixo (para representar o *bit* 0), considerando uma lógica positiva.

Os meios de transmissão de dados em uma rede, também chamados de canais de comunicação, são os meios físicos por onde esses dados (ou sinais elétricos) trafegam. Basicamente, existem dois tipos de meios de transmissão: transmissão por cabos (*wireline*) e transmissão no espaço livre (*wireless)*. Em geral, a escolha do meio físico para um projeto de rede adequado deve levar em consideração os seguintes aspectos: tipo de rede (local ou de grandes distâncias), serviço que será oferecido, distâncias a serem percorridas pelo sinal de informação, relevo do terreno e o número de multiplexações a serem transmitidas.

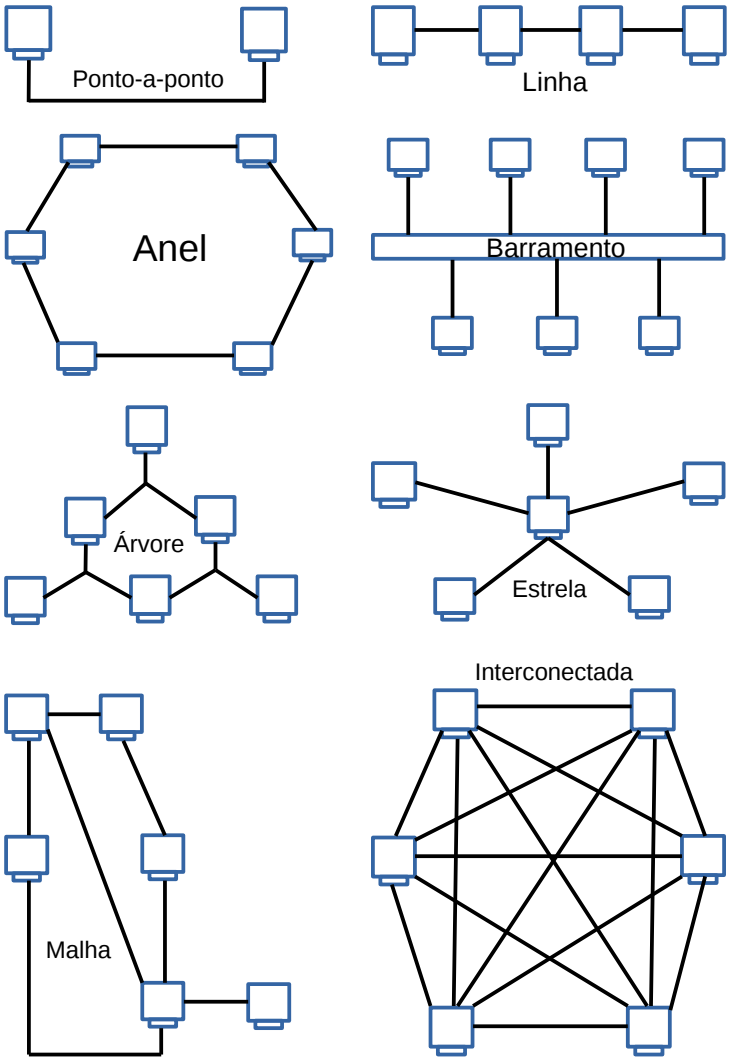
Na comunicação via cabos exitem os cabos metálicos, capazes de conduzir sinais elétricos, que podem ser cabos coaxiais ou de par trançado; cabos ópticos, capazes de conduzir sinais luminosos, são divididos em fibras monomodo e multimodo de índice de refração degrau ou gradual (KEISER, 2000). Os cabos de par trançado possuem baixo custo, fácil instalação e manutenção, porém baixa capacidade de transmissão em bits por segundo (*bps*), o que se torna um problema para altas taxas de dados. Cabos coaxiais possuem taxas de transmissão maiores do que cabos de par trançado, além de possuírem excelente blindagem eletromagnética. Os cabos ópticos, por sua vez, apresentam grandes vantagens em comparação aos demais, como dimensões, peso, flexibilidade, imunidade a ruídos e altas taxas de transmissão de dados; porém, apresentam alto custo de implementação.

Para estabelecer uma rede, deve-se adotar uma ou mais topologias de rede. Topologia de rede é, essencialmente, a maneira como os nós (dispositivos) de uma rede estão interligados. Neste sentido, existem basicamente duas topologias de rede: topologia **física** e topologia **lógica**. A topologia física é a aparência visível da rede, ou seja, como os dispositivos e cabos estão distribuídos fisicamente no espaço. Já a topologia lógica define como os dados (sinais) irão trafegar na rede, sem necessariamente ter conhecimento da topologia física utilizada. No contexto de topologia lógica estão contidos os protocolos e métodos de comunicação, que podem ser reconfigurados dinamicamente por tipos especiais de equipamentos, como roteadores e *switches*. Os dois tipos de topologias lógicas mais comuns são o *Broadcast* e a passagem *Token*. Na primeira, um nó que queira se comunicar envia seus dados a todos os nós espalhados pela rede, este é o caso das redes *Ethernet*. Já na passagem de *Token*, um sinal de *Token* controla o envio de dados pela rede. Essa topologia é utilizada em redes *Token Ring* (MARTINEZ, 2016). A escolha da topologia adequada para cada caso, pode influenciar significativamente no desempenho da comunicação da rede. A Fig. 19 ilustra as oito topologias (físicas) básicas de rede.

A topologia em estrela é a mais utilizada na atualidade, e usa cabos de par trançado e um dispositivo concentrador (roteador) como ponto central da rede. O concentrador se encarrega de retransmitir todos os dados para todas as estações, com a vantagem de tornar mais fácil a localização de falhas, já que se um dos cabos, uma das portas do concentrador ou uma das placas de rede estiver com defeito, apenas o nó ligado ao componente defeituoso ficará fora da rede. As velocidades de transmissão em redes estrela podem chegar a 1000 Mbps dependendo da compatibilidade entre cabos, placas e dispositivo concentrador. Nessa topologia, os dados de informação não percorrem todos os nós. O concentrador se encarrega de entregar o pacote ao destinatário através do

caminho mais curto.

Figura 19: Topologias físicas de rede.



Alguns equipamentos encontrados na maioria das redes de comunicação são: placa de rede, *tranceiver*, *hub*, multiplexador, *switch*, roteador e *bridge* (ponte).

2.9.1 Protocolo TCP/IP

Um protocolo de comunicação é um conjunto de regras que devem ser seguidas para que os dispositivos que implementam este protocolo possam se comunicar, indepen-

dentemente do fabricante do dispositivo. Essa ideia surgiu em 1983, quando a Organização Internacional de Normalização (ISO - *International Standards Organization*) aprovou um modelo de arquitetura para sistemas abertos que definia diretivas genéricas para a comunicação entre dispositivos, independente da tecnologia de implementação. Esse modelo foi denominado como OSI (*Open Systems Interconnection*) e é referência mundial para a criação de novos padrões de comunicação (PINTO, 2010).

Com a criação do modelo OSI para desenvolvimento de protocolos de comunicação entre dispositivos, vários novos protocolos foram criados desde então, sendo uns dos principais, os protocolos TCP/IP, mundialmente conhecidos e utilizados para comunicação entre redes de computadores e dispositivos de monitoramento.

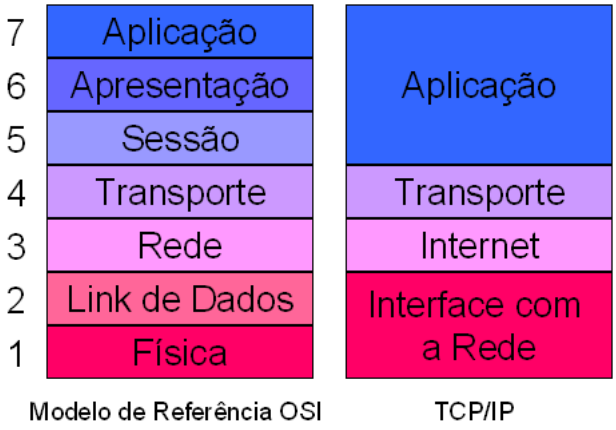
A partir do modelo OSI, as arquiteturas de redes são formadas por camadas (ou níveis), interfaces e protocolos. As camadas são processos implementados por *hardware* ou *software*, que se comunicam com o processo correspondente na outra máquina. Cada camada oferece um conjunto de serviços ao nível superior, usando funções realizadas no próprio nível e serviços disponíveis nos níveis inferiores. Em uma estrutura de rede em camadas, os dados trafegados em uma comunicação de um nível específico não são enviados diretamente ao processo do mesmo nível em outra estação. Os dados seguem o fluxo através de cada camada adjacente da máquina transmissora até o nível inicial, onde é transmitido, para então seguir o caminho contrário através de cada nível adjacente da máquina receptora.

Dentro dessa ideia, o modelo OSI define uma arquitetura genérica de sete camadas para o sistema computacional. Com exceção da camada mais alta, cada camada é usuária dos serviços prestados pela camada imediatamente inferior, e presta serviços para a camada imediatamente superior. Cada camada tem como função adicionar um cabeçalho aos dados do usuário a serem transmitidos para outro sistema. Assim, a função de cada camada do outro sistema é exatamente a inversa, ou seja, retirar os cabeçalhos dos dados que chegam e entregá-los ao usuário em sua forma original. As camadas do modelo de referência OSI, do nível mais alto ao mais baixo, são: aplicação, apresentação, sessão, transporte, rede, enlace e física.

O protocolo TCP/IP é o protocolo de rede mais utilizado atualmente e, na verdade, não é apenas um protocolo, mas sim um conjunto destes. Inclusive, o próprio nome (TCP/IP) faz referência a dois protocolos distintos, o TCP (*Transmission Control Protocol*) e o IP (*Internet Protocol*). Existem ainda, outros protocolos que compõem a pilha TCP/IP, tais como o FTP, o SMTP, o HTTP, o UDP, etc. O protocolo TCP/IP

possui quatro camadas, diferentemente do modelo OSI com suas sete camadas. Na Fig. 20 é possível ver um comparativo entre o modelo OSI e o protocolo TCP/IP.

Figura 20: Comparativo entre as camadas do modelo OSI e TCP/IP.



O funcionamento do protocolo TCP/IP é bastante complexo. Basicamente, a camada de aplicação após processar as requisições dos programas, comunica-se com outro protocolo da camada de transporte, normalmente o protocolo TCP. Na camada de transporte, os dados vindos da camada de aplicação são divididos em pacotes e enviados à camada imediatamente inferior, ou seja, à camada de internet. Outra responsabilidade da camada de transporte é a ordenação ou organização dos pacotes recebidos da rede, uma vez que estes podem chegar fora de ordem, e, também, verificar a integridade dos dados. Na camada de internet funciona o protocolo IP, o qual tem por finalidade pegar os pacotes recebidos da camada de transporte e adicionar informações de endereçamento virtual, ou seja, adiciona o endereço do dispositivo que está enviando os dados e do que irá receber esses dados. Tais endereços virtuais são conhecidos como endereços IP. A seguir, os pacotes são enviados para a camada imediatamente inferior - a camada de interface com a rede. Neste ponto, os pacotes são conhecidos como **datagramas**. A camada de interface com a rede recebe os pacotes enviados pela camada de internet, e então os envia para a rede, ou caso esteja recebendo dados, receberá os dados da rede. A composição desta camada depende do tipo de rede na qual o dispositivo está conectado. Atualmente, praticamente todos os dispositivos utilizam o tipo de rede *Ethernet*, no qual estão incluídas as redes sem fio (TORRES, 2007).

O padrão *Ethernet* é um conjunto de tecnologias que foi desenvolvido, inicialmente, para especificar o funcionamento de uma rede local, onde é possível encontrar diversos

tipos de cabeamentos e transmissão de dados. Este padrão é definido pelo órgão IEEE como 802.3 e é baseado no seguinte princípio: Todas as máquinas da rede *Ethernet* estão conectadas a uma mesma linha de comunicação. As redes *Ethernet* funcionam em um meio *broadcast*, ou seja, todos os dispositivos da rede recebem e enviam dados de e/ou para todos os demais dispositivos. Assim sendo, os dispositivos devem possuir endereços próprios, conhecidos como endereços de *hardware*, físico, *Ethernet* ou endereço MAC. Tal endereçamento é feito com base em uma sequência de 6 bytes (48 bits) e cada dispositivo é único na rede, ou seja, o endereço MAC é a "impressão digital" do equipamento. O padrão *Ethernet* funciona nas duas primeiras camadas do modelo OSI: camada física e camada de enlace. Na camada física é onde os dados são tratados em sua forma mais bruta, ou seja, pulsos elétricos. Já na camada de enlace, os dados são tratados em forma de pacotes para agilizar o processo de transmissão. Cada pacote *Ethernet* tem um cabeçalho de 14 octetos, onde estão contidos o endereço da fonte de transmissão e o endereço de destino. Dessa forma, cada dispositivo visualiza apenas os pacotes endereçados a si, mesmo tendo recebido pacotes destinados a outros dispositivos. A velocidade de transmissão dos padrões *Ethernet* é medida em bits por segundo, e existem atualmente três padrões de velocidades, são eles: *Standard Ethernet* (10Mbps), *Fast ethernet* (100Mbps) e *Gigabit Ethernet* (1000Mbps ou 1Gbps – 10Gbps para fibras ópticas). Como o padrão *Ethernet* se tornou o mais amplamente utilizado para LAN's, ele foi rapidamente estendido para redes MAN's e WAN's, o que resultou na utilização de componentes altamente confiáveis e de baixo custo e que estão disponíveis em todo o mundo, ou seja, padronizados.

3 Desenvolvimento

O primeiro passo de projeto foi o levantamento e detalhamento das funcionalidades dos dispositivos e *softwares* envolvidos no sistema, os quais estão todos descritos no cap. 2. Este passo é de extrema importância, pois embora existam muitas coisas "prontas" na literatura que podem ser reaproveitadas, muitas vezes é preciso adaptar, ou mesmo criar novas soluções para poder prosseguir. Desse modo, é de crucial importância, em um projeto, que se conheça muito bem o funcionamento dos dispositivos que compõem o sistema. As demais etapas do projeto são descritas neste capítulo, em suas seções e subseções subsequentes.

Com as ferramentas necessárias em mãos, passou-se a realizar testes de acionamento de cargas com Arduino, com o intuito de desenvolver, posteriormente, uma placa de circuito impresso (PCB - *Printed Circuit Board*) contendo o *driver* (ver cap. 2 seção 2.4.4) de potência para acionamento do motor, um circuito de alimentação para o Arduino, um circuito para acomodar dois sensores *hall* (BESSE et al., 2002)-(BRAGA, 2014)-(CASSIOLATO, 2006) e um sensor de temperatura.

3.1 Circuito de alimentação

Um dos passos cruciais do projeto foi a escolha da alimentação elétrica do sistema do posicionador. Levando em conta que seria preciso uma tensão de 12V para alimentar o motor de passo e uma tensão entre 7-12V para alimentar o Arduino, optou-se por instalar no sistema, uma fonte ATX (*Advanced Technology Extended*) de computador (BROWN, 2014), a qual apresenta baixo custo. A fonte ATX disponibiliza para o circuito de controle tensões de 12V, 5V e GND (*Ground*). Os 12V vão para o motor de passo e para um regulador de tensão LM-317, que rebaixa o nível de tensão para 9V, de forma a alimentar o Arduino. De acordo com a Tabela 4, o Arduino pode ser alimentado com 12V, porém não é recomendável utilizar essa tensão, pois o regulador de tensão interno do dispositivo pode aquecer demais. O esquemático do circuito rebaixador de tensão que utiliza o LM-317 é mostrado na Fig. 21. Este esquemático foi elaborado de acordo com o *datasheet* do regulador LM-317, que fornece todas as especificações necessárias para se trabalhar com segurança com o dispositivo. Os diodos D1 e D2, protegem o regulador contra curto-circuito, caso a saída seja conectada ao terra por acidente; os capacitores C3

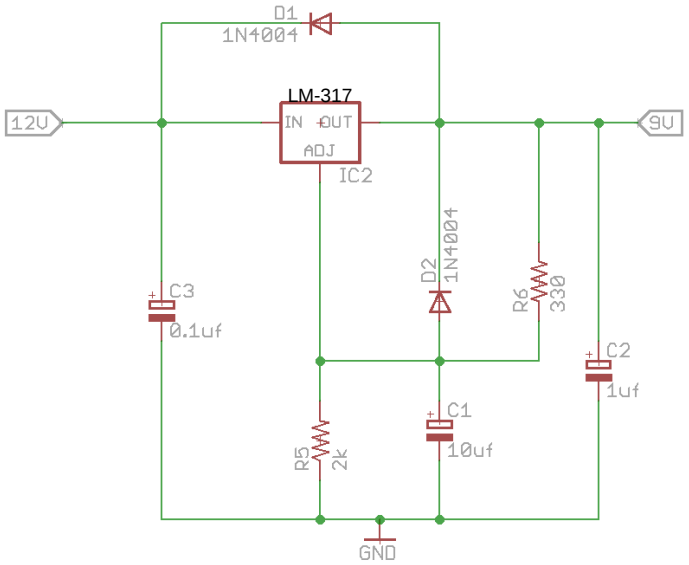
(entrada) e C2 (saída) agem como filtros de frequências para deixar a tensão CC o mais "limpa" possível. Esses capacitores são conhecidos como *bypass* (desvio), pois desviam as frequências indesejadas. O capacitor C1 é calculado para rejeição de *Ripple* de cerca de 15dB, e também é utilizado para proteção, sendo descarregado por D2 em caso de curto-circuito na saída. Os resistores R5 e R6 são responsáveis por ajustar a tensão de saída, de acordo com

$$V_o = V_{REF}(1 + \frac{R5}{R6}) + I_{ADJ}R5, \tag{3.1}$$

onde V_{REF} é uma tensão de referência interna entre a saída e o terminal de ajuste, e equivale, segundo o *datasheet*, a 1,25V; I_{ADJ} é a corrente de ajuste, e está especificada para ser de no máximo $100\mu A$. A tensão de saída para alimentar o Arduino foi especificada para ser 9V, então, restam apenas R5 e R6 para serem calculados. O valor de R6 deve ser fixo, enquanto o valor de R5 é variado para obter a tensão de saída desejada. Fixando R6 em 330Ω e rearranjando (3.1) em termos de R5, tem-se

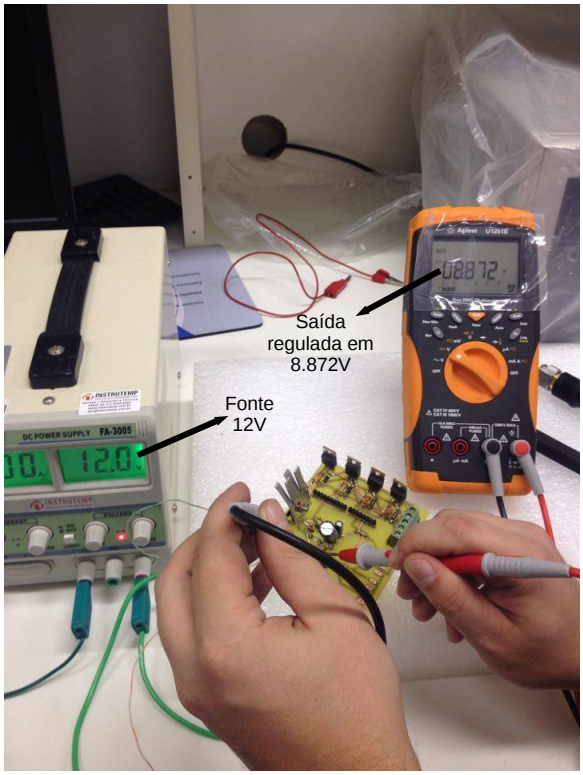
$$R5 = \frac{(V_o - V_{REF})}{(\frac{V_{REF}}{R6} + I_{ADJ})} = \frac{9 - 1.25}{(\frac{1.25}{330}) + 100^{-6}} = 2k\Omega. \tag{3.2}$$

Figura 21: Esquemático do circuito rebaixador de tensão para o Arduino.



Uma resistência de $2k\Omega$ não existe comercialmente, porém, é bastante trivial sua obtenção por meio de dois resistores de $1k\Omega$ em série. O circuito de alimentação, após a montagem, apresentou excelente estabilidade e condições de operação, entregando na saída 8,872V, como pode ser visto na Fig. 22.

Figura 22: Medida da tensão de saída do circuito regulador de tensão.



É importante salientar que o modo de alimentação para o Arduino e o sistema, aqui descrito, foi escolhido por razões de simplicidade, organização e robustez. A fonte ATX fornece, com sobra de recursos, as tensões e correntes necessárias ao sistema. Outra característica é que a fonte pode ser embutida dentro da estrutura giratória, junto com o restante do circuito (Arduino, *Shields*, sensor *hall*, etc). Assim, basta apenas ligar a fonte à energia, e todo o sistema poderá ser alimentado. Os sensores *hall* e o sensor de temperatura (LM35) poderiam ser alimentados pela saída de 5V do próprio Arduino (ver Fig. 17), porém, de acordo com as especificações do fabricante do dispositivo, cada pino de I/O pode fornecer, individualmente, uma corrente de no máximo $40mA$, e uma

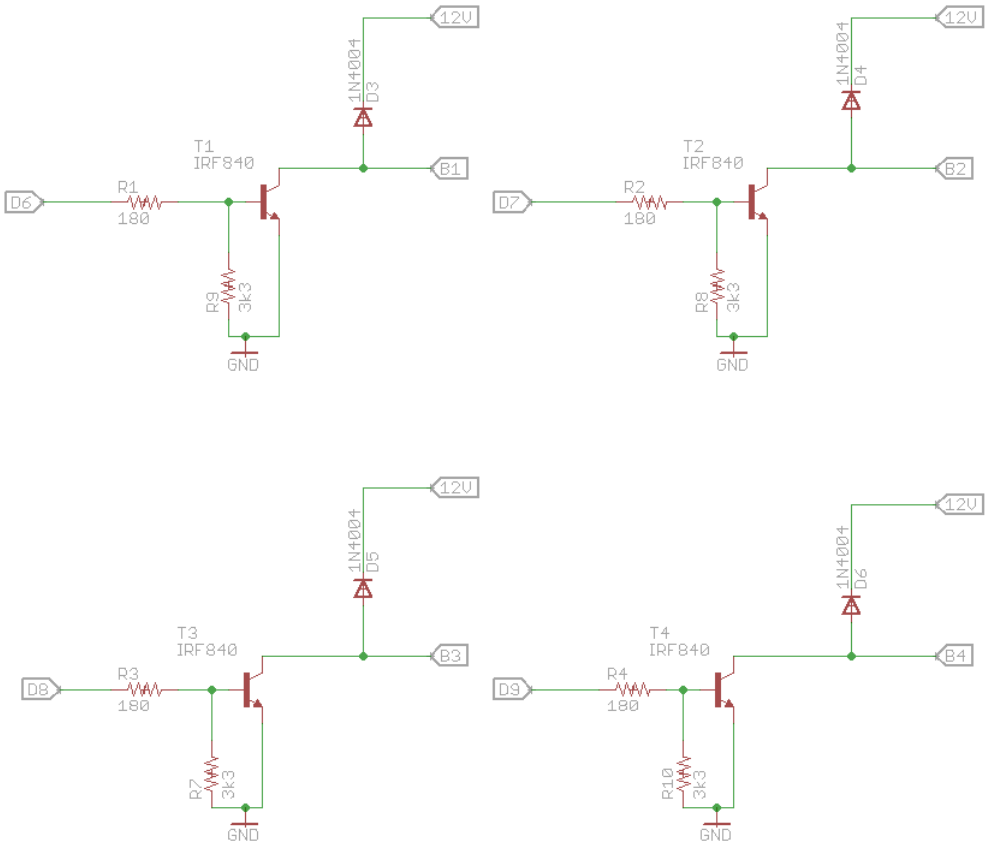
corrente de no máximo $200mA$ no caso de um conjunto de pinos ligados. Como o sistema utiliza dois sensores *hall* e um sensor de temperatura, torna-se arriscado trabalhar com as baixas correntes fornecidas pelo Arduino. Além disso, ficam sobrando os pinos 3.3V e 5V, que podem ser utilizados posteriormente para alimentar algum outro módulo que se queira adicionar ao sistema.

3.2 Circuito de potência

Como já mencionado em outras seções, a estrutura giratória do sistema de medição utiliza um motor de passo para seu funcionamento. Também, foi visto na subseção 2.4.4 que, para acionar um motor através de um microcontrolador ou de um Arduino, é preciso utilizar um circuito de potência formado por transistores. No caso da subseção 2.4.4, foi exemplificado o circuito ponte H. Entretanto, para o projeto aqui desenvolvido, são utilizados transistores individuais para cada fase das bobinas. Optou-se por utilizar transistores FET (*Field Effect Transistor*) de potência modelo IRF840, os quais suportam consideráveis correntes (ver *datasheet* do dispositivo) e possuem diodos internos de proteção (diodos roda livre). Assim sendo, o sistema pode ser modificado para acomodar motores mais potentes e, consequentemente, antenas mais pesadas. A Fig. 23 ilustra o esquemático do circuito de potência para acionamento das quatro fases do motor.

Na Fig. 23, os diodos D3, D4, D5 e D6 são os referidos diodos roda livre. Embora os transistores possuam diodos de proteção internos, a colocação dos diodos externos faz aumentar ainda mais a segurança do circuito. Outra característica de segurança é a utilização dos resistores de *Gate* R1, R2, R3 e R4, que limitam a corrente drenada do Arduino em caso de curto-circuito interno nos transistores. Os comandos lógicos, provenientes do Arduino, são representados pelas *labels* (rótulos) D6, D7, D8 e D9, que representam os pinos de saída digitais 6, 7, 8 e 9, respectivamente. As *labels* B1, B2, B3 e B4, representam as fases das bobinas do motor e são ligadas no *Drain* (Dreno) dos transistores. O motor utilizado é unipolar (possui derivação central e características operacionais descritas na seção 2.4.1), onde a derivação é alimentada diretamente com 12V. Quando o *Gate* de algum transistor é acionado pelo Arduino, o chaveamento do transistor ocorre, tornando possível a passagem de corrente de *Dreno* para *Source* (fonte), e assim, alimentando uma fase de uma das bobinas do motor. Os resistores R7, R8, R9 e R10 são conhecidos como resistores de *pull-down* (ELECTRONICS, 2013) - (BACK, 2014), e servem para desativar o transistor após um acionamento, caso o mesmo não desarme sozinho.

Figura 23: Circuito de potência para acionamento de motor de passo.



O cálculo para os resistores de *Gate* é obtido pela lei de *Ohm*, descrita em (3.3). A tensão "V" equivale a 5 volts, proveniente do Arduino; a corrente "I" foi especificada para ser de no máximo $30mA$, $10mA$ abaixo da especificação de corrente máxima por pino de I/O do Arduino. Isolando R em (3.3) e substituindo-se os valores de V e I, obtém-se a resistência de $166,6\Omega$.

$$V = RI \tag{3.3}$$

A corrente I é inversamente proporcional à resistêcia R. Isso quer dizer que se R aumenta, I diminui. Então, qualquer valor maior ou igual ao encontrado irá garantir uma corrente menor ou igual a $30mA$. Assim sendo, uma resistência de *Gate* de 180Ω, como mostrado na Fig. 23, é suficiente para atender à especificação.

Na literatura encontram-se diversos valores para a utilização dos resistores de *pull-down*, sendo os valores mais utilizados o de $3{,}3k\Omega$ e o de $10k\Omega$. A utilização dos

resistores de 3,3$K\Omega$ para o circuito de potência, apresentou ótimos resultados.

3.3 Circuito dos sensores

O circuito de alimentação e acionamento do posicionador também possui sensores, como já foi mencionado. Dois sensores *hall* são utilizados para o posicionamento correto da estrutura giratória, e um sensor LM35 é utilizado para verificar a temperatura do sistema. A Fig. 24 ilustra as conexões feitas para os dois sensores *hall*, e a Fig. 25 mostra as ligações para o sensor de temperatura LM35.

Figura 24: Circuito dos sensores *hall*.

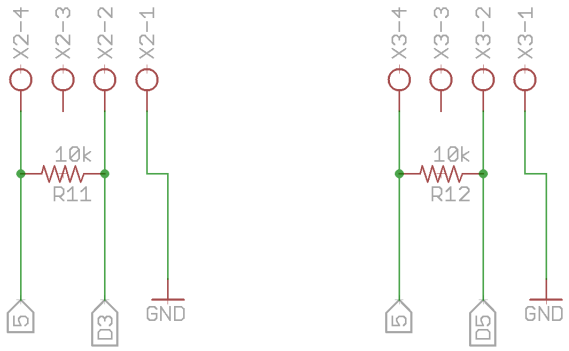
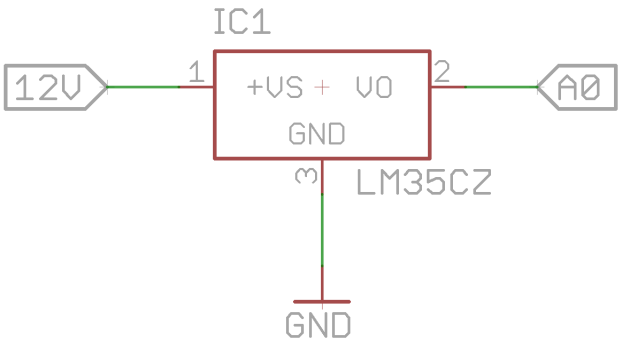


Figura 25: Ligações do sensor de temperatura LM35.



Na Fig. 24, X2-4, X2-3, X2-2, X2-1,X3-4, X3-3, X3-2 e X3-1 são conexões para os pinos de dois conectores de quatro terminais cada um, para que os sensores *hall* possam ser fixados em qualquer lugar na estrutura e conectados com o circuito controlador

através de fios. Nos pinos X2-4 - X3-4, é entregue uma tensão de 5V, proveniente da fonte ATX. Essa tensão não foi escolhida ao acaso. Os sensores *hall* podem ser alimentados com tensões bem mais altas, porém, a tensão de alimentação é a mesma que surge no terminal de saída do sensor sob a presença de um campo magnético. A tensão de saída do sensor é enviada ao Arduino para ser processada e para tomada de decisões. Como o Arduino trabalha com tensões internas de no máximo 5V, a alimentação dos sensores foi feita com essa tensão. Os resistores de $10k\Omega$, R11 e R12, servem para limitar a corrente entre entrada e saída e são especificados pelo fabricante do sensor. Os pinos X2-2 - X3-2 são as saídas dos sensores e são conectados aos pinos digitais do Arduino, D3 e D5, respectivamente. Os pinos X2-1 - X3-1 são conectados ao terra do circuito.

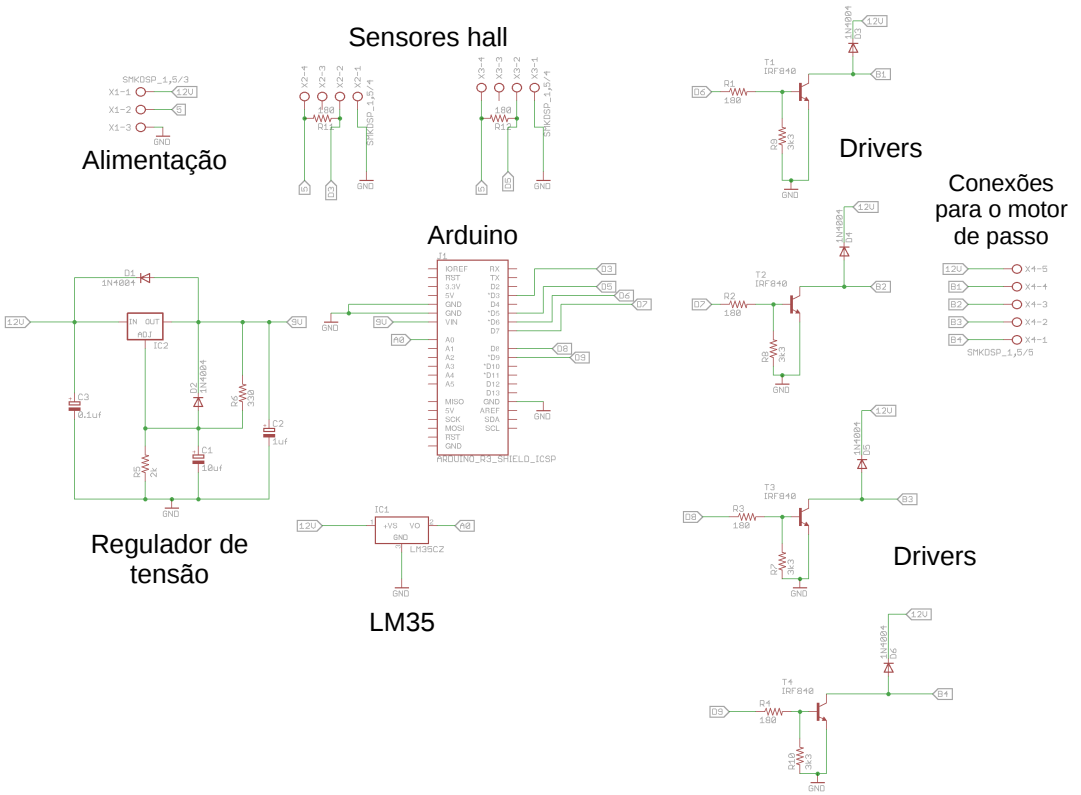
3.4 Circuito completo em PCB

Após testados em *protoboard*, os circuitos de alimentação, potência e sensores, foram integrados e montados em uma placa de circuito impresso de dupla face. O esquemático completo do circuito é mostrado na Fig. 26, e o *layout* para confecção da PCB, gerado a partir do esquemático, é apresentada na Fig. 27. Como pode ser observado, existem trilhas azuis e vermelhas. As trilhas azuis localizam-se no *top-layer* (camada de cima) da placa, e as trilhas vermelhas localizam-se no *bottom-layer* (camada de baixo). A escolha de dupla face para o projeto foi devida à necessidade de acomodar muitos componentes em uma área pequena de circuito, o que se tornaria muito difícil com uma única face. A desvantagem de se utilizar dupla face é em relação a dificuldade de soldar componentes grandes, como os conectores, por exemplo, que cobrem os buracos de seus pinos no *top-layer*. Este problema foi resolvido com a colocação de vias, para que nos componentes grandes a solda fosse feita apenas no *bottom-layer*. A placa de circuito impresso, já com todos os componentes soldados e já acoplada ao Arduino e *Ethernet Shield*, é mostrada em dois ângulos diferentes para melhor visualização, na Fig. 28.

3.5 Programação do Arduino

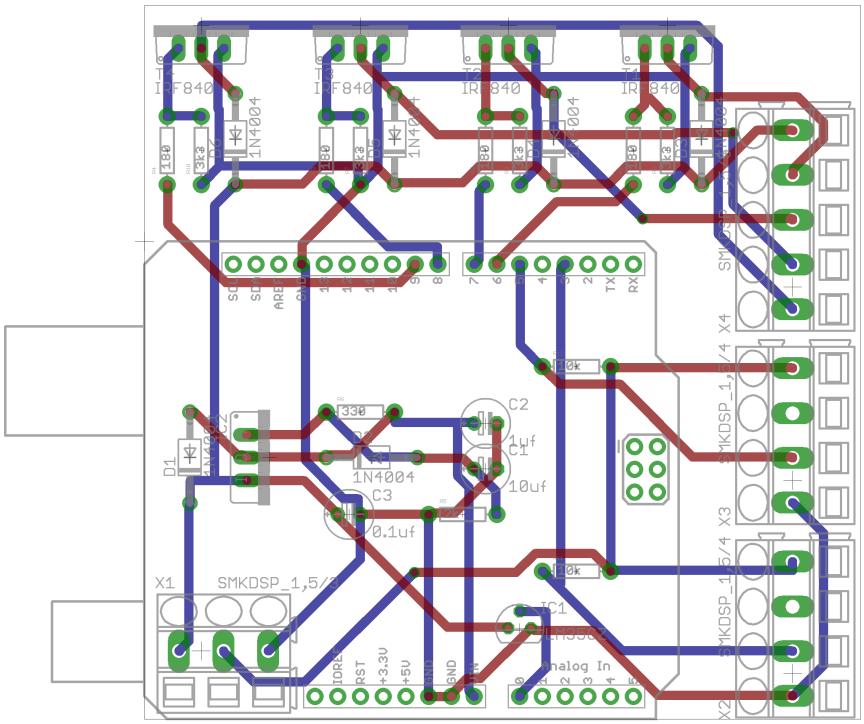
A plataforma Arduino possui uma linguagem própria de programação baseada em C/C++, e uma IDE de desenvolvimento (ver seção 2.5). O primeiro programa, desenvolvido para testes, apenas acionava sequencialmente as bobinas de um motor de passo no sentido horário, ou seja, apenas ativava sequencialmente os pinos de saída do Arduino. Em seguida, uma página *Web* programada em HTML (*Hyper-Text Markup*

Figura 26: Esquemático completo do circuito desenvolvido.



Language) foi desenvolvida e embarcada no código do Arduino. A página *Web* apresentou ótimos resultados, sendo capaz de acionar diversos dispositivos controlados pelo Arduino, tais como *leds*, *coolers* e motores de passo, além de apresentar a temperatura do sistema em tempo real. Um recurso interessante que foi adicionado ao HTML da página, é a transmissão, em tempo real, de um *streaming* de vídeo proveniente de uma *webcam*. Para isso ser possível, é necessário que um servidor de *streaming* esteja disponibilizando o fluxo da *webcam* na rede, através de algum *codec* de compressão como ogg, mpeg-2, mpeg-4, entre outros. Com este recurso, é possível, por exemplo, visualizar o sistema de medição remotamente através da rede ou da internet, e pode ser bastante útil para medições em câmaras anecoicas. O código completo utilizado para o Arduino, já com a programação da página *Web* embarcada, é apresentado no APÊNDICE C deste documento. De um modo geral, o código segue o algoritmo descrito pelo fluxograma apresentado na Fig. 29. Como pode ser visto, após o programa (*sketch*) ser iniciado, são feitas as importações

Figura 27: Layout para a confecção da placa de circuito impresso.

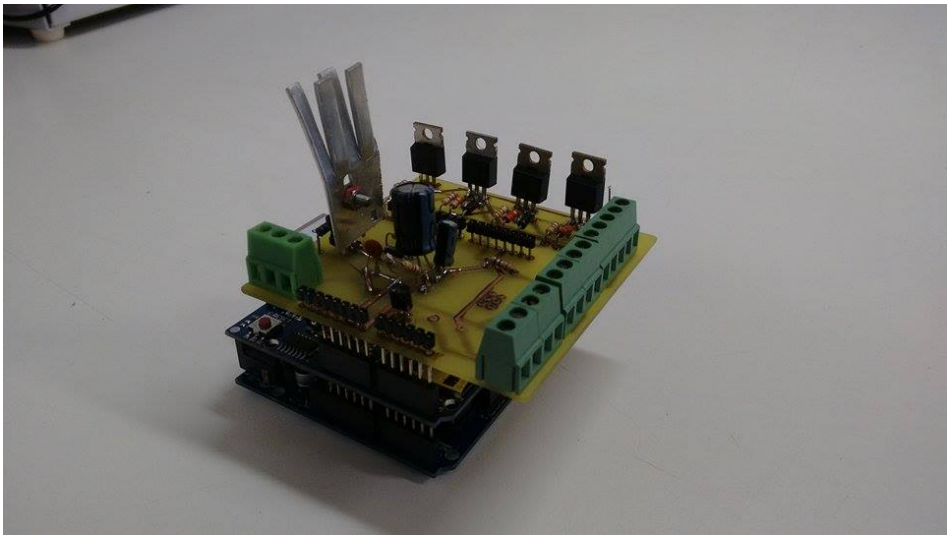


das bibliotecas necessárias para as funções que se deseja utilizar. Nessa parte, também são feitas as declarações de variáveis globais (podem ser acessadas por qualquer função dentro do código), pode-se, inclusive, criar variáveis para numerar os pinos do Arduino. Abaixo estão listadas essas configurações, em um trecho de código.

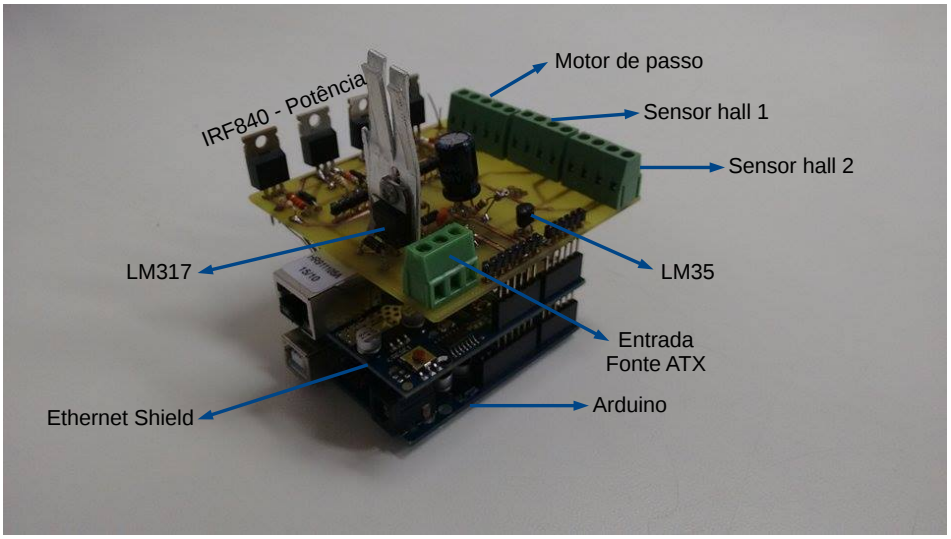
```
#include <SPI.h>
#include <Ethernet.h>
#include <EEPROM.h>

byte mac[] = {
0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(10,2,6,201);
byte gateway[] = { 10 , 2, 6, 1 };
byte subnet[] = { 255, 255, 254, 0 };
IPAddress myserver(0,0,0,0);
EthernetServer server(80);
EthernetClient client;
String readString;
```

Figura 28: Placa do circuito finalizada e acoplada ao Arduino.



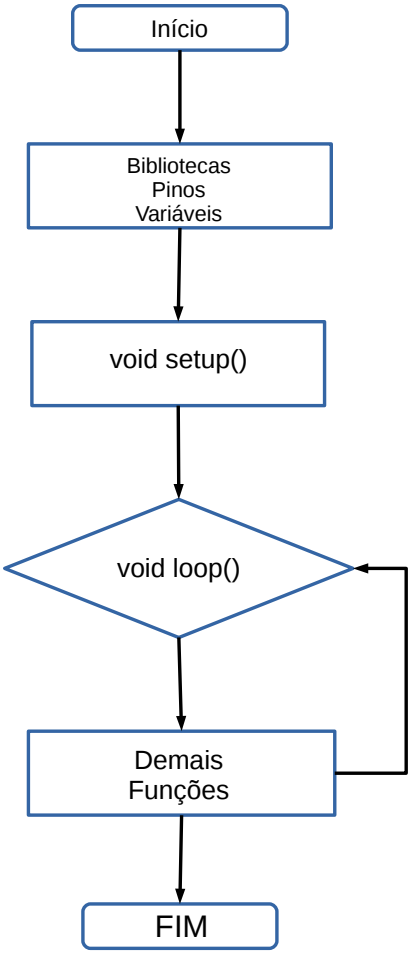
(a) Placa do circuito finalizada e acoplada ao Arduino.



(b) Identificação dos componentes do circuito

```
const int tempo = 4000;
const int tempo2 = 700;
const int LM35 = A0;
float temperatura = 0;
float valorLido = 0;
int statePin3 = LOW;
int statePin5 = LOW;
int addr = 0;
int value = 0;
```

Figura 29: Fluxograma geral do algoritmo do Arduino.



A biblioteca “SPI.h”, incluída na linha 1 do código anterior, é responsável pela comunicação entre o *Ethernet Shield* e o Arduino, por meio do protocolo SPI (*Serial Peripheral Interface*). A biblioteca “Ethernet.h” provê todo o necessário para criar um servidor *Web* capaz de receber comandos remotos por TCP/IP, bem como enviar dados de resposta à requisições de clientes. Esse servidor *Web* fica rodando no *Ethernet Shield*, e disponibiliza a página HTML embarcada no código do Arduino. A biblioteca “EEPROM.h” provê os recursos necessários para a gravação e leitura de dados na EEPROM (*Electrically-Erasable Programmable Read-Only Memory*) do microcontrolador Atmega328p. Este recurso é utilizado para guardar a última posição do motor de passo, de acordo com a última fase acionada. Como o acionamento de motores de passo é feito pela alimentação sequencial correta das fases das bobinas, é essencial que se tenha controle de qual bobina está ativa em cada acionamento. A memória EEPROM permite a leitura de dados mesmo após o dispositivo ser desenergizado.

O *Ethernet Shield* não vem com um endereço MAC de fábrica, de modo que é preciso configurar esse parâmetro no código. Isso é feito nas linhas 5 e 6 do código anterior, através do vetor “mac[ ]” do tipo *byte*, e pode ser utilizado qualquer endereço MAC que não esteja em uso na rede. As linhas de 7 a 11 configuram os parâmetros de rede tais como IP, *gateway*, máscara de sub-rede e porta de conexão. Esses parâmetros são de extrema importância, uma vez que são os responsáveis pelo envio e recebimento de dados para o(s) dispositivo(s) correto(s), e devem ser configurados de acordo com a rede que se deseja trabalhar. No presente caso, o servidor *Web* responde no IP “10.2.6.201”, porta “80” (padrão de servidores *Web*), em uma rede cuja máscara é “255.255.254.0” e cujo *gateway* responde no IP “10.2.6.1” (ver seção 2.9 para mais detalhes sobre redes). A linha 12 declara um cliente *Ethernet* chamado “client”. As demais linhas, até a linha 22, são declarações de variáveis que serão utilizadas ao longo do código.

Em seguida, o código executa sua primeira função - *void setup()* - que é responsável pela configuração geral do Arduino. Aqui, são definidos os pinos de I/O e inicializado o servidor *Web*, bem como são ativadas as comunicações serial (com *baudrate de 9600*) e TCP/IP. O trecho de código abaixo ilustra esses procedimentos.

```
1 void setup(){

3     analogReference(INTERNAL);
      pinMode(LM35, INPUT);
5     pinMode(3, INPUT);
      pinMode(5, INPUT);
7     pinMode(6, OUTPUT);
```

```
    pinMode(7, OUTPUT);
9   pinMode(8, OUTPUT);
    pinMode(9, OUTPUT);
11  Ethernet.begin(mac, ip);
    server.begin();
13  Serial.begin(9600);

15 }
```

Na linha 3 do código apresentado, a referência para os pinos analógicos do Arduino é configurada para "INTERNAL". Este artifício é utilizado para conseguir melhor precisão na medida da temperatura com o LM35. O pino A0 (LM35) é definido como entrada (INPUT), e receberá as tensões provenientes do sensor de temperatura. Os pinos 3 e 5 são configurados como entrada (INPUT), e receberão as tensões dos sensores *hall*, quando esses sensores forem ativados. Os pinos de 6 a 9 são definidos como saídas, e enviarão os pulsos para o *driver* de acionamento, que, por sua vez, move o motor.

A função principal do código vem a seguir: *void loop()*. É aqui que o código realmente funciona: fazendo cálculos, armazenando valores, lendo sensores, enviando pulsos e requisitando outras funções. Abaixo, um trecho de código mostra o início da função.

```
1  void loop(){

3      valorLido = analogRead(LM35);
       temperatura = valorLido/9.31;
5      Serial.print("Temperatura: ");
       Serial.println(temperatura);
7      delay(400);

9      pagina();

11     if(readString.indexOf("bobina1on") >0){

13       digitalWrite(6, HIGH);
         digitalWrite(7, LOW);
15       digitalWrite(8, LOW);
         digitalWrite(9, LOW);
17       EEPROM.write(addr, 1);
         delay(tempo);
```

```
19
    }
```

A primeira coisa que a função *void loop()* faz é ler o pino analógico do Arduino, que recebe informações (tensões) do sensor de temperatura LM35. Assim, na linha 4, é feito o cálculo da temperatura de acordo com o método descrito em (ARDUINO, 2016a) para se obter melhor resolução para os valores medidos. Para um teste de comunicação, as linhas 5 e 6 imprimem na porta serial, a *String* "Temperatura: " seguida do valor da temperatura calculado na linha 4. Então, a palavra "Temperatura: 25", por exemplo, pode ser visualizada no monitor serial da própria IDE do Arduino, ou por algum outro dispositivo que se comunique pelo protocolo serial. As leituras de temperatura são calculadas e enviadas à porta serial a cada nova iteração da função *void loop()*, assim como todas as outras funções e condições (if/else).

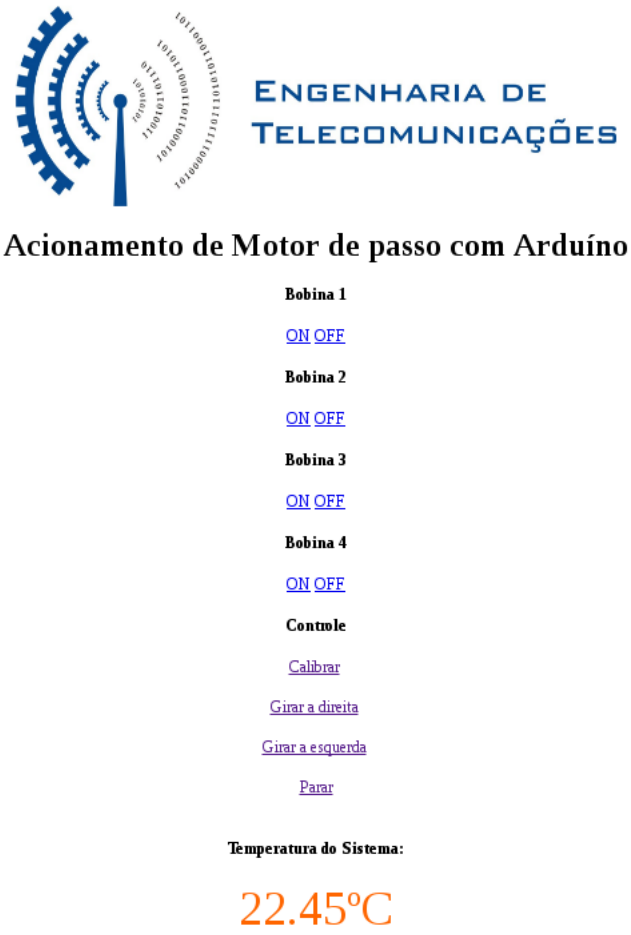
Após esperar um tempo de 400 ms (linha 7), a função "pagina()", na linha 9, é chamada. Essa função tem três objetivos principais: ficar aguardando novas requisições de um cliente (navegador de internet, por exemplo); se uma requisição de cliente chegar, estabelecer a conexão e enviar ao cliente o HTML da página Web; e por último, ler e armazenar, na variável *readString*, os comandos enviados por um usuário, para tomada de decisão. A Fig. 30 mostra o *layout* da página, visto por um usuário através de um navegador cliente.

Quando a função "pagina()" termina sua execução, o código retorna à função *void loop()*, e retoma a execução de onde parou. Caso um usuário tenha clicado na opção de acionar a bobina 1, o trecho de código das linhas 11 até 20 é executado. Esse trecho de código aciona a primeira fase da sequência de acionamento das bobinas do motor, e mantém as outras fases desligadas. Logo em seguida, na linha 17, o inteiro "1" é armazenado na EEPROM do microcontrolador do Arduino para registrar que a bobina foi acionada. O trecho de código das linhas 11 a 20 é repetido para as outras 3 fases de acionamento. Note-se que as funções de controle estão todas fora da função *void loop()*, elas apenas são chamadas quando um retorno da função *void pagina()* as invoca, e após sua execução, retornam à função *void loop()*.

As funções de controle para a estrutura giratória são:

- ***void calibrar()***: Função que realiza o posicionamento automático da estrutura para que essa fique exatamente na origem (0°). Para tanto, a estrutura é girada

Figura 30: *Layout* da página HTML de controle do sistema giratório.



no sentido horário, até que um ímã fixado na estrutura encontre um dos sensores *hall* e o acione. Caso a estrutura esteja posicionada a um ângulo $0° < \theta < 180°$, ela irá girar no sentido horário até 180°, encontrar o segundo sensor *hall*, retornar no sentido anti-horário até encontrar o primeiro sensor *hall* (posicionado na origem), e então parar. Por outro lado, se a estrutura estiver posicionada a um ângulo $180° < \theta < 360°$, ela irá girar no sentido horário e encontrará o primeiro sensor *hall* diretamente, sem o auxílio do segundo sensor. Este artifício de calibragem foi escolhido para não haver enrolamentos dos fios da antena em teste - perceba-se que o sistema nunca dá uma volta completa de 360°, e sim duas voltas de 180° cada, uma no sentido horário e outra no sentido anti-horário;

- ***void automaticoh()***: Essa função é chamada quando algum usuário clica em

"Girar a direita", na página *Web* da Fig. 30. Quando isso acontece, a *String* "*readString*" guarda o valor "*authon*", que é o responsável por acionar a função "*automaticoh()*", que por sua vez, tem a tarefa de acionar as fases das bobinas do motor. A função "*automaticoh()*" é executada até que a posição da estrutura esteja em 180° ou até que o usuário clique em parar. Quando a estrutura chega em 180°, ela retorna automaticamente para 0°, e então para. Abaixo é descrito um trecho de código que caracteriza a função '*automaticoh()*'.

```
void automaticoh(){

  statePin3=digitalRead(3);
  statePin5=digitalRead(5);

  while(readString.indexOf("authon") > 0 && statePin3==LOW){

    value = EEPROM.read(addr);

    if (value == 1) {

    digitalWrite(6, LOW);
    digitalWrite(7, HIGH);
    digitalWrite(8, LOW);
    digitalWrite(9, LOW);
    EEPROM.write(addr, 2);
    delay(tempo);

    pagina();

    statePin3=digitalRead(3);
    statePin5=digitalRead(5);

    if(readString.indexOf("parar") >0 || statePin5==HIGH){

      desligar();

    }

    if(statePin3==HIGH){

      voltar();

    }
```

```
36      else {

38      digitalWrite(6, LOW);
        digitalWrite(7, LOW);
40      digitalWrite(8, HIGH);
        digitalWrite(9, LOW);
42      EEPROM.write(addr, 3);
        delay(tempo);
44

        }
```

Note-se que a função "pagina()" é sempre chamada entre as execuções dos códigos. Isso é necessário para verificar, em tempo real, se o usuário clicou em algum novo comando, como "parar", por exemplo. As linhas dentro do "*while*", no código acima, são repetidas para todas a fases das bobinas, bem como para todos os valores possíveis que podem estar na EEPROM do microcontrolador. Assim sendo, a função "*automaticoh()*" toma decisões de acordo com a última fase acionada do motor, para que a sequência de acionamento não saia da ordem correta;

- ***void automaticoah()***: Essa função é idêntica à função "*void automaticoh()*", diferindo apenas na sequência de acionamento das fases. Aqui, a sequência faz com que a estrutura gire no sentido anti-horário. O comando que invoca a função "*void automaticoah()*", quando um usuário clica em "Girar à esquerda", é "autahon". Novamente, aqui o comando é passado para a função correspondente através da *String* "*readString*";

- ***void voltar()***: A função "*voltar()*" é utilizada pela função "*calibrar()*" e pela função "*automaticoh()*" para retornar à origem (0°), fazendo o acionamento das fases no sentido anti-horário. Assim sendo, quando a estrutura estiver em 180° tendo partido de 0°, ela retorna automaticamente para a origem, e então, para;

- ***void voltar2()***: É utilizada pela função "*void automaticoah()*" para retornar à origem, e aciona as fases no sentido horário. Quando a estrutura encontra a origem (sensor 1 em 360°), a função "*voltar2()*" termina sua execução e desliga as fases das bobinas, fazendo com que a estrutura pare. O código então, volta para a função "*void loop()*" e começa tudo novamente, até que algum cliente faça nova requisição;

- ***void desligar()***: Como o próprio nome sugere, essa função é responsável por desligar as fases das bobinas do motor, ou seja, interromper qualquer tipo de movimento da estrutura. A função "*desligar()*" também limpa o conteúdo da variável "*readString*", para assegurar que nenhum comando seja executado. Abaixo é descrita a função "*desligar()*", em um trecho de código.

```
void desligar(){

    digitalWrite(6, LOW);
    digitalWrite(7, LOW);
    digitalWrite(8, LOW);
    digitalWrite(9, LOW);
    readString="";
    readString="";

}
```

A função "*pagina()*" merece uma atenção especial, e está descrita no trecho de código abaixo.

```
void pagina(){

  EthernetClient client = server.available();

  if (client) {
   while (client.connected()) {
   if (client.available()>0) {
    char c = client.read();
    if (readString.length() < 100) {
    readString += c;
    }

    if (c == '\n') {
     client.println(F("HTTP/1.1 200 OK"));
     client.println(F("Content-Type: text/html"));
     client.println();
     client.println(F("<HTML>"));
     client.println(F("<HEAD>"));
     client.println(F("<TITLE>Arduino - Acionamento de Motor de passo</
   TITLE>"));
     client.println(F("</HEAD>"));
```

```
21      client.println(F("<BODY><center><img src=
        colocar_aqui_o_caminho_para_a_img height=200 width=600 align=bottom></
        center>"));

23      client.println(F("<center><H1>Acionamento de Motor de passo com Ardu&
        iacute;no</H1></center>"));
        client.println(F("<center>"));
25
        client.println(F("<h4>Bobina 1</h4>"));
27      client.println(F("<a href=\"/?bobina1on\" target=\"inlineframe\">ON</
        a>"));
        client.println(F("<a href=\"/?bobina1off\" target=\"inlineframe\">OFF
        </a>"));
29
        client.println(F("<h4>Bobina 2</h4>"));
31      client.println(F("<a href=\"/?bobina2on\" target=\"inlineframe\">ON</
        a>"));
        client.println(F("<a href=\"/?bobina2off\" target=\"inlineframe\">OFF
        </a>"));
33
        client.println("<h4>Bobina 3</h4>");
35      client.println("<a href=\"/?bobina3on\" target=\"inlineframe\">ON</a>
        ");
        client.println("<a href=\"/?bobina3off\" target=\"inlineframe\">OFF</
        a>");
37
        client.println("<h4>Bobina 4</h4>");
39      client.println("<a href=\"/?bobina4on\" target=\"inlineframe\">ON</a>
        ");
        client.println("<a href=\"/?bobina4off\" target=\"inlineframe\">OFF</
        a>");
41      client.println("<h4>Controle</h4>");
        client.println(F("<a href=\"/?calibrar\" target=\"inlineframe\">
        Calibrar</a>"));
43      client.print(F("<BR>"));
        client.print(F("<BR>"));
45      client.println(F("<a href=\"/?authon\" target=\"inlineframe\">Girar a
         direita</a>"));
        client.print(F("<BR>"));
47      client.print(F("<BR>"));
        client.println(F("<a href=\"/?autahon\" target=\"inlineframe\">Girar
        a esquerda</a>"));
49      client.print(F("<BR>"));
```

```
      client.print(F("<BR>"));
51    client.println(F("<a href=\"/?parar\" target=\"inlineframe\">Parar</a
    >"));
      client.println(F("</center>"));
53
      client.print(F("<BR>"));
55
      client.print(F("<center><h4>Temperatura do Sistema: </h4> <font size
    =7> <font color=\"#ff6600\"> "));
57    client.print(temperatura);
      client.print(F("&#186;C </font></font></center>"));
59    client.print(F("<BR>"));
      client.println("<video id=video src=http://10.2.6.176/desktop.ogg
    type=video/ogg; codecs=theora autoplay=autoplay>");
61    client.println(F("<IFRAME name=inlineframe style=\"display:none\" >")
    );
      client.println(F("</IFRAME>"));
63    client.println(F("</BODY>"));
      client.println(F("</HTML>"));
65    delay(1);
     client.stop();
67
     }
69
     }
71
     }
73   }
  }
```

No código acima, as linhas de 3 a 11 são responsáveis por monitorar requisições de clientes, bem como estabelecer a conexão entre ambos (cliente e servidor). Após a "conversa" de estabelecimento de conexão, uma nova linha em branco é enviada pelo cliente, no caso de este ser um cliente HTTP (*Hyper-Text Transfer Protocol*), como um navegador de internet, por exemplo. Esta é justamente a condição para executar o "if" da linha 13 (\n = nova linha), a qual enviará para o cliente todo o código HTML da página *Web* de controle. Percebe-se, na linha 21, que é possível adicionar *links* para imagens no código HTML.

A função "*client.println()*" é quem envia o texto do HTML para o cliente. A função "*F()*" que aparece dentro da função "*client.println()*", serve para poupar a memória dinâmica do Arduino. Esta função indica ao compilador que os trechos de texto fixo, a serem enviados para um cliente através da função "*println()*" e seus derivados, não devem ser copiados para a RAM. De acordo com a Tabela 4, o Arduino possui apenas 2kB de memória RAM, logo, seu limite teria excedido em muito sem a ajuda da função "*F()*" (CAMPOS, 2015b).

Na linha 60, tem-se a *tag "video"* do HTML. Ela serve para reproduzir um "*streaming*" de vídeo na página *Web*, proveniente de algum servidor na rede ou na internet. No caso da linha 60, existe um servidor de "*streaming*" na rede local no endereço de IP "http://10.2.6.176", que está disponibilizando um fluxo de vídeo compactado em *ogg* utilizando o *codec theora*. Assim, é possível instalar um módulo de monitoramento via *webcam* no sistema do posicionador, de modo a visualizar, em tempo real, a medição caso esta seja feita dentro de uma câmara anecoica. Para disponibilizar o fluxo de imagem de uma *webcam* na rede, foi utilizado o *player* de vídeo "VLC", atuando como servidor de "*streaming*" de vídeo.

A linha 65 executa um intervalo de tempo (*delay*) de 1ms, e a linha 66 encerra a comunicação cliente-servidor.

3.6 Interface Matlab

Matlab é uma linguagem de programação voltada ao desenvolvimento de algoritmos de natureza técnica. O nome *Matlab* vem de *Matrix Laboratory*, e foi originalmente desenvolvido para prover um acesso amigável ao tratamento de vetores e matrizes, aliás, os elementos básicos da linguagem são exatamente vetores e matrizes.

Atualmente, o *Matlab* dispõe de uma poderosa biblioteca de funções matemáticas, geração de gráficos e manipulação de dados que auxiliam muito o trabalho do programador. E ainda possui uma vasta coleção de bibliotecas denominadas *toolboxes*, utilizadas para áreas específicas como: equações diferencias ordinárias, estatística, processamento de imagens, processamento de sinais, finanças, aeroespacial, controle de instrumentos, e várias outras.

A linguagem e o ambiente de programação permitem ainda que o usuário possa escrever suas próprias bibliotecas em *Matlab*. Assim, o usuário pode contribuir com a linguagem, incorporando a ela novas funções.

O fato de o *Matlab* possuir diversas funções prontas para se trabalhar com vetores e plotar gráficos foi que levou à escolha da utilização do *software*. Além disso, a comunicação com dispositivos externos tais como analisadores de espectro e Arduino, pode ser feita através de vários protocolos, como o protocolo TCP/IP utilizado para este trabalho. Este fato permite que o controle de todos os dispositivos do sistema de medidas possa ser feito via *Matlab*. A interface desenvolvida é apresentada na Fig. 31.

Figura 31: Interface *Matlab* de controle do sistema de medição.

O painel à esquerda na Fig. 31, possui os botões de controle da estrutura giratória, e realiza as mesmas funções que a página *Web*. O comando do *Matlab*, responsável por enviar sinais de controle ao Arduino, é o comando "*urlread()*", que leva como parâmetro o endereço IP que se quer acessar, seguido do comando que executa a função desejada. O comando "*urlread()*" retorna todo o texto da URL que está acessando, no caso da página *Web* do *Ethernet Shield*, todo o código HTML da página é retornado ao *Matlab*. Estes dados de retorno não são úteis neste caso. Entretanto, ao enviar o endereço IP seguido do comando que executa uma função, como parâmetro da função "*urlread()*", ela fará com que o comando seja executado pelo HTML da página, identicamente a um navegador. O comando "urlread('http://10.2.6.201/?calibrar');", por exemplo, envia o comando para a calibragem do posicionador, supondo um *Ethernet Shield* com um servidor *Web* rodando no IP "10.2.6.201". O mesmo comando é utilizado pelos outros botões de controle, mudando apenas o parâmetro que aciona a função correspondente (após o ponto de interrogação).

Nos dois painéis do meio, na Fig. 31, encontram-se os botões de configuração do gerador de sinais e do analisador de espectro. Para o gerador de sinais, apenas a amplitude e a frequência do sinal a ser transmitido devem ser configurados, pois o comando que ativa a saída de RF está implícito no comando que configura a amplitude do sinal. A opção de obter dados, tanto do gerador quanto do analisador, serve para testar a comunicação entre o *Matlab* e estes dispositivos. Em caso de se estabelecer corretamente a conexão TCP/IP, os dados dos dispositivos tais como fabricante, nome e modelo, são imprimidos na caixa de texto abaixo do botão "Limpar dados". O botão "Limpar dados", como o próprio nome sugere, exclui o conteúdo da caixa de texto, para que esta fique disponível para uma nova consulta.

Para enviar os dados digitados pelo usuário, para ambos gerador e analisador, são utilizados comandos SCPI (ver seção 2.8), além do protocolo de comunicação TCP/IP. O código *Matlab* que implementa a função para ajustar a frequência inicial do analisador de espectro, é apresentado a seguir:

```
1  function pushbutton18_Callback(hObject, eventdata, handles)

3  freq_start = get(handles.edit7,'String');
   mxa_ip = '10.2.6.202';
5  mxa_port = 5025;
   mxa = tcpip(mxa_ip,mxa_port);
7  fopen(mxa);
   scpi_freq_start = ':FREQ:STAR %s GHz\n';
9  fprintf(mxa,scpi_freq_start,freq_start);

11 fclose(mxa);
   delete(mxa);
13 clear mxa;
```

Quando um usuário digita um valor de frequência (por exemplo: 1.5) na caixa de texto e clica em "OK", a função "pushbutton18_ Callback()" acima, é chamada. Na linha 3, a função "*get()*" captura uma *String* digitada no campo de texto nomeado como "edit7", que nesse caso, é a frequência digitada pelo usuário. As linhas 4 e 5, definem o endereço IP para onde os comandos serão enviados e a porta de conexão, neste caso, o IP e porta do analisador de espectro. Na linha 6, é criado um objeto TCP/IP e armazenado na variável "*mxa*". Na linha 7, a conexão é estabelecida. A linha 8 cria o texto com o comando SCPI apropriado para ser enviado ao analisador. Percebe-se que o comando

SCPI está em sua forma compacta e que possui um caractere de dois pontos (:) no começo, ou seja, refere-se à raiz da árvore de comandos. O comando ":FREQ:STAR <VALOR> GHz", onde "<VALOR>" é definido pelo usuário, ajusta a frequência inicial do analisador de espectro para "VALOR GHz". Na linha 9, o comando SCPI é construído por meio da concatenação da *String* da linha 8 com a *String* que guarda o valor digitado pelo usuário (*freq_ start*). Esse comando é imprimido no instrumento por meio do objeto TCP/IP criado (*mxa*), que é parâmetro da função "fprintf()". As linhas 11 a 13 encerram a comunicação com o instrumento.

É importante salientar que o analisador de espectro deve estar devidamente configurado e com os parâmetros de rede apropriados para que a conexão seja estabelecida, podendo-se assim, controlar remotamente o instrumento. As funções que implementam os ajustes de frequências central e final possuem a mesma sintaxe da função utilizada para a frequência inicial descrita anteriormente, mudando apenas o comando SCPI e o valor da frequência a ser enviado para o instrumento. Por exemplo, para ajustar uma frequência central de 2 GHz, o comando SCPI seria ":FREQ:CENT 2.0 GHz", e para uma frequência final de 2.5 GHz, o comando utilizado é ":FREQ:STOP 2.5 GHz".

As sintaxes das funções descritas para o analisador de espectro valem para o gerador de sinais. As únicas variáveis que diferem do código anterior são o IP do instrumento, porta de conexão,comando SCPI e o campo de texto onde o usuário digita o valor desejado.

Para a obtenção dos dados dos instrumentos, quando um usuário clica em "Obter dados do instrumento", é implementada a função abaixo:

```
function pushbutton19_Callback(hObject, eventdata, handles)

mxa_ip = '10.2.6.202';
mxa_port = 5025;
mxa = tcpip(mxa_ip,mxa_port);
fopen(mxa);
idn = query(mxa,'*IDN?');
set(findobj(gcf,'Tag','edit8'),'String',idn);
fclose(mxa);
delete(mxa);
clear mxa;
```

Note-se, na linha 7 do código acima, o comando SCPI descrito na Tabela 5,

responsável pela identificação dos instrumentos. A linha 8 apenas escreve, na caixa de texto, os caracteres recebidos referentes à identificação do instrumento (variável "idn"). Como pode ser visto nos códigos apresentados, o método de comunicação entre computador e instrumento é sempre o mesmo (TCP/IP).

O código que implementa a função de obter a tela do analisador é apresentado a seguir:

```
1  function pushbutton22_Callback(hObject, eventdata, handles)
2
3  mxa_ip = '10.2.6.202';
4  mxa_port = 5025;
5  mxa=tcpip(mxa_ip, 5025);
6  set(mxa,'InputBufferSize',100000);
7  fopen(mxa);
8  fprintf(mxa,':FORM:DATA REAL,32');
9  nr_points = str2double(query(mxa,':SWE:POIN?'));
10 ref_lev = str2num(query(mxa,'DISP:WIND:TRAC:Y:RLEV?'));
11 fprintf(mxa,':TRAC? TRACE1');
12 data = binblockread(mxa,'float32');
13 fscanf(mxa);
14
15 figure(1)
16 plot(1:nr_points,data)
17 xlim([1 nr_points])
18 ylim([ref_lev-100 ref_lev])
19 grid on
20 title('Swept SA trace')
21 xlabel('Point index')
22 ylabel('Amplitude (dBm)')
23
24 fclose(mxa);
25 delete(mxa);
26 clear mxa;
```

A linha 6 do código ajusta o tamanho do *buffer* de entrada para receber os dados dos traços da tela. Esse parâmetro deve ser de pelo menos 4 vezes o número de pontos de traços para um formato real de 32 bits. A linha 8 define o formato de dados da tela para Real de 32 bits. A operação da linha 9 retorna e armazena o número de pontos da tela do instrumento. A linha 10 obtém o nível de referência do instrumento. As linhas 11 a 13

capturam e armazenam os dados dos traços da tela. As linhas 15 a 22 encarregam-se da plotagem dos dados obtidos, na tela do computador. Deste modo, é possível visualizar a tela do instrumento, remotamente e em tempo real. Perceba-se a contínua utilização de comandos SCPI embarcados no código *Matlab*.

O painel mais à direita na Fig. 31 - Visualizar sistema - é capaz de controlar uma *webcam* instalada no sistema de medidas, para visualização remota. Como nos outros painéis de controle da interface, o botão "Detectar *Webcam*" serve para verificar a existência de um dispositivo de captura de vídeo e retornar seus dados de configuração (da *webcam*) na caixa de texto. Em caso de comunicação bem sucedida, o botão "Play" é capaz de acionar o fluxo de vídeo da *webcam*, como descrito em (MATHWORKS, 2016).

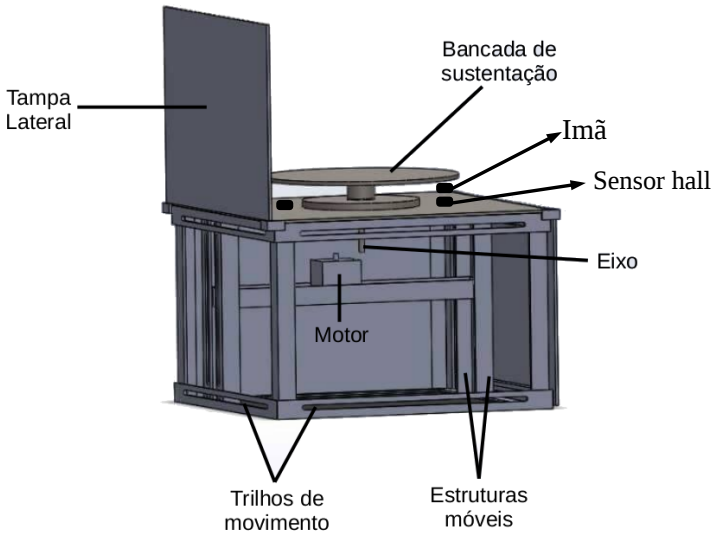
3.7 Estrutura giratória

A estrutura giratória foi desenhada de maneira a ser adaptável, isto é, estruturas móveis permitem que se façam ajustes no posicionador, a fim de acomodar um motor maior ou algum tipo de circuito, caso seja necessário. A Fig. 32 ilustra a estrutura projetada. Como pode ser observado, a estrutura possui trilhos de movimento que permitem que peças móveis possam ser realocadas. Isto permite que a estrutura seja adaptável nas três dimensões do sistema de coordenadas espaciais (x, y e z). Nas tampas laterais da estrutura existem dobradiças, para que se tenha acesso à parte interna do posicionador. A bancada de sustentação é movida pelo eixo da estrutura, que é conectado ao eixo do motor de passo. Além disso, um rolamento é colocado no eixo da estrutura para suavizar e diminuir o atrito do movimento. Este passo de projeto foi feito em parceria com um aluno do curso de Engenharia Mecânica.

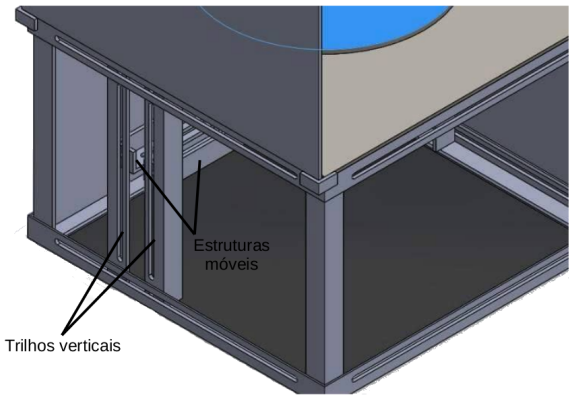
A estrutura foi feita em aço, de modo que foi necessário colocar absorvedores planares na mesma para evitar a reflexão das ondas eletromagnéticas. Os sensores *hall* foram colados nos absorvedores e ligados ao Arduino por meio de fios, que passam por baixo dos absorvedores.

A montagem final resultou em uma estrutura giratória extremamente leve (por causa do rolamento) e resistente (feita em aço). Um pequeno problema foi que a bancada de sustentação ficou ligeiramente inclinada. Este problema foi resolvido, em partes, com a compensação da inclinação no nivelamento do cano de sustentação das antenas, o qual vai em cima da bancada. No capítulo 4 são apresentadas as imagens do sistema montado na prática, onde é possível ver os componentes mencionados anteriormente. Entretanto,

Figura 32: Estrutura giratória.



(a) Detalhes de projeto.



(b) Partes móveis verticais

na Fig. 33, é possível visualizar imagens da estrutura isolada, ou seja, não conectada ao restante do sistema. Percebe-se, a presença de canos de PVC (*Polyvinyl Chloride*), que servem para a fixação e sustentação das antenas em teste. Também, nota-se a presença dos já mencionados absorvedores planares colados à estrutura, bem como dos sensores *hall*, abaixo da bancada de sustentação.

Figura 33: Estrutura giratória após montagem.

(a) Vista completa.

(b) Vista lateral.

(c) Sensor *hall*.

(d) Vista frontal.

4 Montagem do Sistema e Resultados

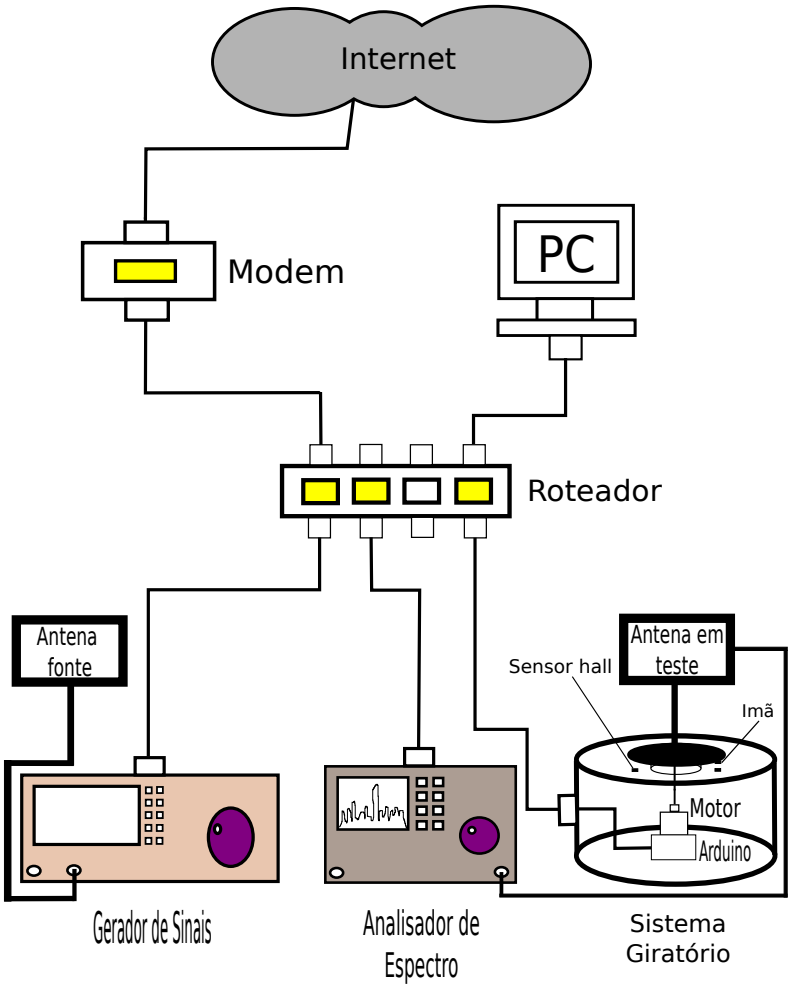
Após ter testado todas as partes do sistema individualmente, passou-se à sua montagem completa. O resultado ficou como ilustrado na Fig. 34. Como pode ser visto, o sistema fica disponível para acesso externo (internet). Entretanto, os testes realizados foram todos feitos em uma estrutura de rede local (LAN) utilizando a topologia em estrela (ver seção 2.9). Após conectados os equipamentos na rede (para gerar e obter a amplitude do sinal), ambos (gerador e analisador) foram configurados para operar em uma frequência central de 2,595 GHz. A configuração dos equipamentos foi toda realizada através da interface descrita na seção 3.6.

Para validar o sistema desnvolvido, foi realizada a caracterização do diagrama de irradiação de uma antena de microfita recebendo sinais em 2,595 GHz. Esta antena foi validada em campo distante (câmara anecoica) no IFI (Instituto de Fomento e Coordenação Industrial do Departamento de Ciência e Tecnologia Espacial) e construída na Universidade Federal do Pampa - campus Alegrete - conforme descrito em (SCHLOSSER, 2014). A Fig. 35 mostra a antena sendo validada em campo distante. Para a obtenção do diagrama de irradiação em campo próximo, foi utilizado um *Near Field Scanning* (NFS), na UNIPAMPA, conforme pode ser visto na Fig. 36.

O cenário de testes para o sistema de medição desenvolvido pode ser descrito da seguinte maneira: uma antena transmissora, operando em 2,595 GHz, foi conectada ao gerador de sinais e posicionada a uma distância de campo distante - mais do que 2 metros, neste caso - da antena receptora; no lado receptor, a antena em teste foi fixada ao sistema posicionador e conectada ao analisador de espectro. Todos os dispositivos - analisador de espectro, gerador de sinais, posicionador e controlador (computador) - foram interligados por meio de um roteador. Com as devidas ligações feitas, todo o controle da medição foi realizado através de um *notebook* contendo a interface em *Matlab*. Os parâmetros ajustados na interface, em relação a configuração do *setup* de medição para a antena de 2,595 GHz, podem ser visualizados na Fig. 37, enquanto que o cenário de medição, nas Fig. 38 e 39.

A Fig. 38 mostra uma foto do cenário onde foram feitos os testes do sistema de medição. A antena transmissora foi fixada em um pedestal, para que a mesma não balançasse durante a medida, e para que se pudesse ajustar sua altura. A Fig. 39 mostra a parte de recepção do sinal, onde encontram-se a antena em teste, o analisador de espectro

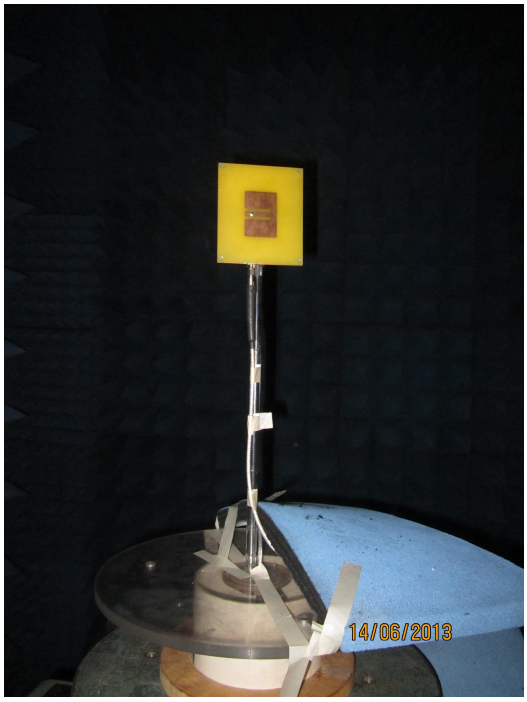
Figura 34: Ilustração do sistema de medição completo, após montagem.



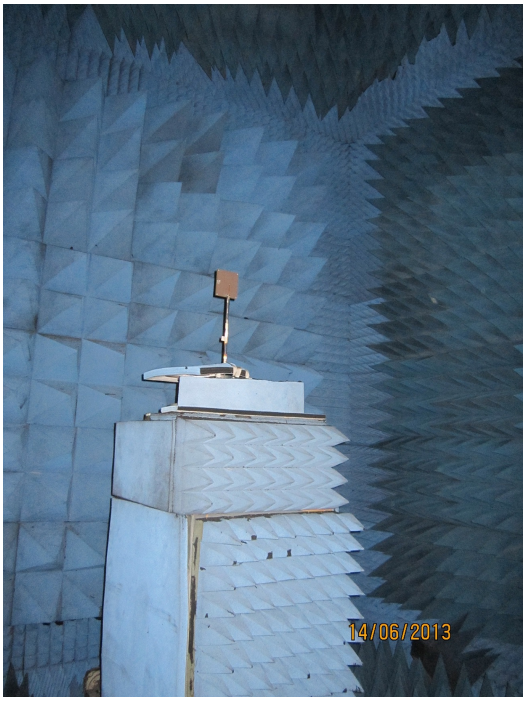
e o sistema posicionador.

Para proceder com a medida, primeiramente calibrou-se o sistema para que o mesmo, após a colocação da antena em teste, ficasse posicionado exatamente na origem, isto é, em visada direta com a antena transmissora. Após a calibração, foi realizada a medida no sentido horário, ou seja, $0° \leq \theta \leq 180°$. As amplitudes recebidas pela antena em teste, para cada ângulo de rotação, foram guardadas em um vetor no *Matlab*. Em seguida, realizou-se a medida no sentido anti-horário ($180° \leq \theta \leq 360°$), procedendo-se da mesma forma. O vetor resultante foi plotado juntamente com um vetor "*theta*", que

Figura 35: Antena sendo validada na câmara anecoica do IFI.



(a) Visualização frontal da antena.

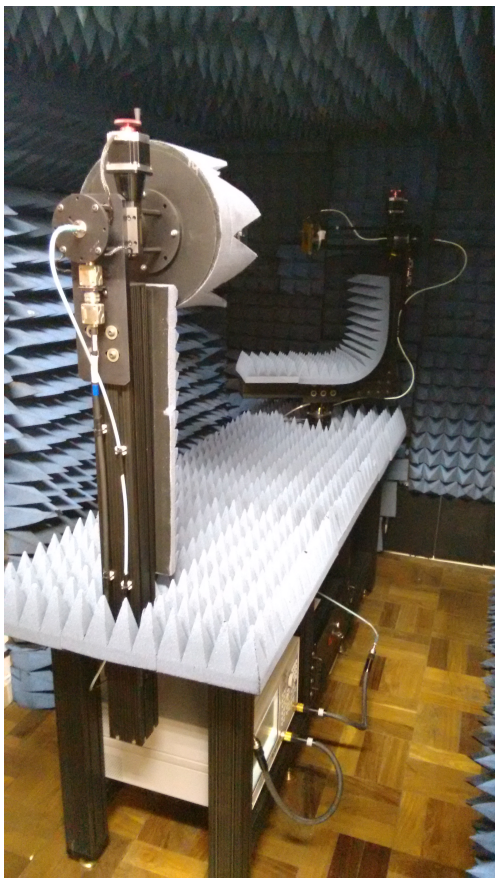


(b) Câmara anecoica e visualização traseira da antena.

representa os ângulos de passo da estrutura giratória, neste caso, passos de $1,8°$. Após a medição e plotagem do diagrama, a curva obtida foi comparada às curvas que validaram a antena em teste. A Fig. 40 mostra as curvas mencionadas, plotadas em um mesmo gráfico para o plano H. É possível ver que todas as curvas apresentam comportamentos parecidos, sendo a curva em azul obtida através do sistema de medição aqui descrito. As imperfeições nos lóbulos traseiros (curva em azul), devem-se às reflexões da onda eletromagnética em objetos metálicos e paredes de concreto que circundavam o ambiente em que foi feita a medida. Mesmo assim, o diagrama obtido com o sistema de medição em espaço livre apresenta comportamento quase idêntico, na direção de apontamento principal, aos diagramas obtidos em simulação, câmara anecoica e NFS. Percebe-se que a curva em preto, na Fig. 40, é a que mais destoa dentre todas. Esta curva foi a obtida em câmara anecoica, utilizando-se o *Near Field Scanning*. A curva em vermelho foi obtida através de um *software* de simulação eletromagnética chamado HFSS, e a curva em verde foi gerada na câmara anecoica do IFI.

Outra curva de interesse é o diagrama no plano E da antena em teste, o qual é mostrado na Fig. 41 juntamente com as curvas obtidas por simulação, câmara anecoica e

Figura 36: Antena sendo validada em campo próximo em um NFS.

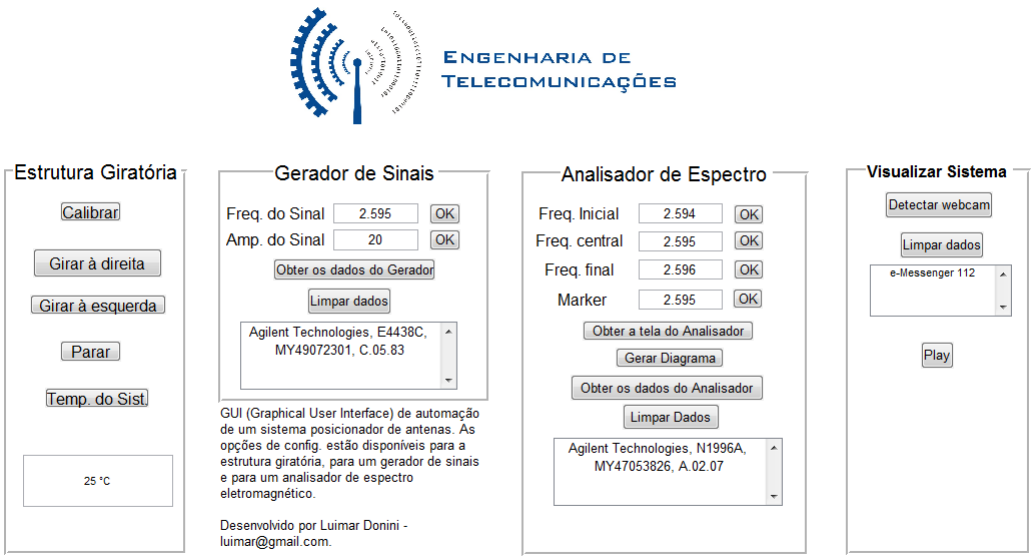


NFS.

O sistema projetado também foi utilizado para gerar o diagrama de irradiação, em campo distante e em espaço livre, de uma rede de antenas filamentares cujo propósito é ser embarcada em um modelo real de aeronave radiocontrolada (YOSHIMOTO, 2016). Para esta medição, utilizou-se como antena transmissora um diedro refletor transmitindo sinais em uma frequência de 5,8 GHz. O diedro utilizado como transmissor pode ser visualizado na Fig. 42(a), e a rede de antenas filamentares em teste é mostrada na Fig. 42(b).

A rede de antenas filamentares em teste foi medida em três posições diferentes, e os diagramas obtidos em espaço livre foram plotados juntamente com os obtidos através de simulação eletromagnética (HFSS) e NFS. As Fig. 43, 44 e 45 ilustram a vista superior, lateral e frontal, respectivamente, do diagrama de irradiação que caracteriza a rede de antenas em teste. Note-se que os diagramas são apresentados de forma espelhada, uma

Figura 37: Interface com os parâmetros de configuração.



vez que a rede filamentar é simétrica.

As medidas realizadas para a antena em formato E (Fig. 39) e para a rede de antenas filamentares apresentaram algumas discrepâncias em relação às curvas obtidas pelo NFS e HFSS, mostrando a necessidade de ambientes adequados para se instalar os equipamentos de medição. Neste sentido, não deveriam haver obstáculos no local, os quais provocam a reflexão da onda eletromagnética irradiada pela antena transmissora. No cenário em que foram realizadas as medições, o sistema ficou sujeito à interferências de prédios localizados entre 10 a 20 metros do posicionador. Como solução, a aplicação de uma antena transmissora mais diretiva poderia fornecer melhores resultados para as medidas, além da escolha de um ambiente mais adequado para medição. Outro fator que contribuiu para algumas imperfeições nas curvas obtidas foram as deficiências mecânicas do sistema, tais como o baixo torque do motor de passo. A solução mais conveniente, para este caso, seria a substituição do motor atual por um com maior torque. Reflexões da onda eletromagnética no solo, em árvores próximas e no próprio posicionador, também afetaram, em partes, o desempenho do sistema. Também, o alinhamento das antenas transmissoras e receptoras não foi realizado a partir de equipamento adequado para nivelar a estrutura, desta forma, a bancada de sustentação e as orientações das antenas podem ter ficado desnivelada e desalinhada, respectivamente.

Figura 38: Antena transmissora utilizada no sistema de medição - 2,595GHz.

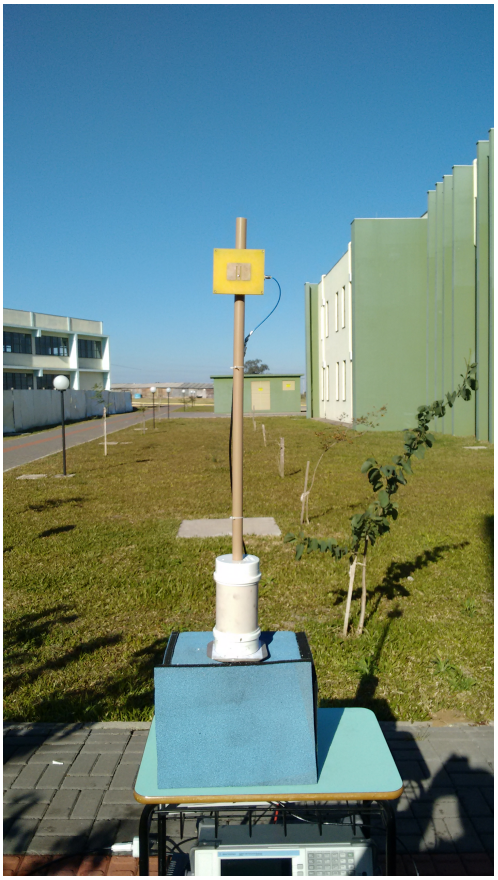
A rede de antenas filamentares da Fig. 42(b) foi medida em um dia que estava ventando, fazendo com que a estrutura balançasse, e, consequentemente, produzisse *ripples* na curva dos diagramas de irradiação.

As medições, sem levar em consideração a montagem do sistema, foram feitas em um tempo médio de dez minutos cada uma, visto que o tempo de medição depende das configurações de aquisição de dados. A cada passo do motor, o valor da amplitude do sinal recebido pela antena em teste era lido do analisador de espectro através do *Matlab*. Esta leitura foi programada para acontecer a cada três segundos, e, este tempo foi sincronizado com o tempo de passo do motor, através da programação do Arduino.

Apesar das pequenas diferenças entre as curvas medidas, o sistema pode ser usado para caracterização de antenas. A principal vantagem está relacionada ao custo de implementação, que é apresentado na Tabela 6. O posicionador desenvolvido, juntamente

Figura 39: Sistema de recepção.

(a)



(b)

com a interface, apresenta um custo estimado entre R$ 1.000,00 a R$ 1.200,00, enquanto que produtos comerciais custam de dezenas a centenas de milhares de reais, tornando vantajosa a utilização do sistema projetado para a validação de protótipos em nível acadêmico.

Tabela 6: Estimativas de custo para o sistema desenvolvido.

Componente	Preço (em R$)
Arduino + *Shield Ethernet* + componentes eletrônicos	150,00
Fonte de alimentação ATX	60,00
Motor de passo de alto torque	300,00 a 400,00
Estrutura em aço	300,00 a 400,00
Roteador	100,00
Cabos de par trançado	50,00
Canos de PCV	20,00
Total	1.000,00 a 1.200,00

Figura 40: Comparativo entre diagramas de irradiação obtidos por diferentes modos de medida no plano azimutal.

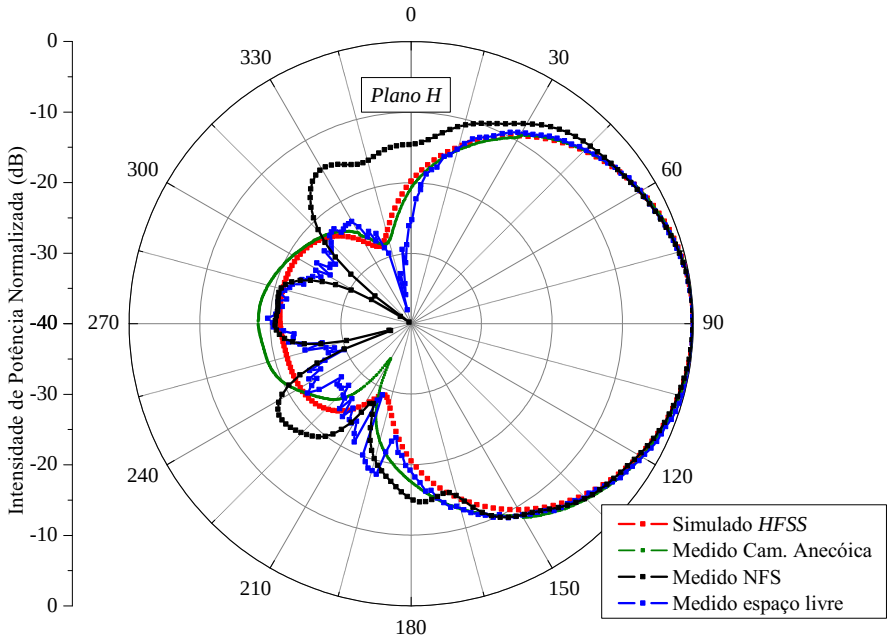


Figura 41: Comparativo entre diagramas de irradiação obtidos por diferentes métodos - plano de elevação.

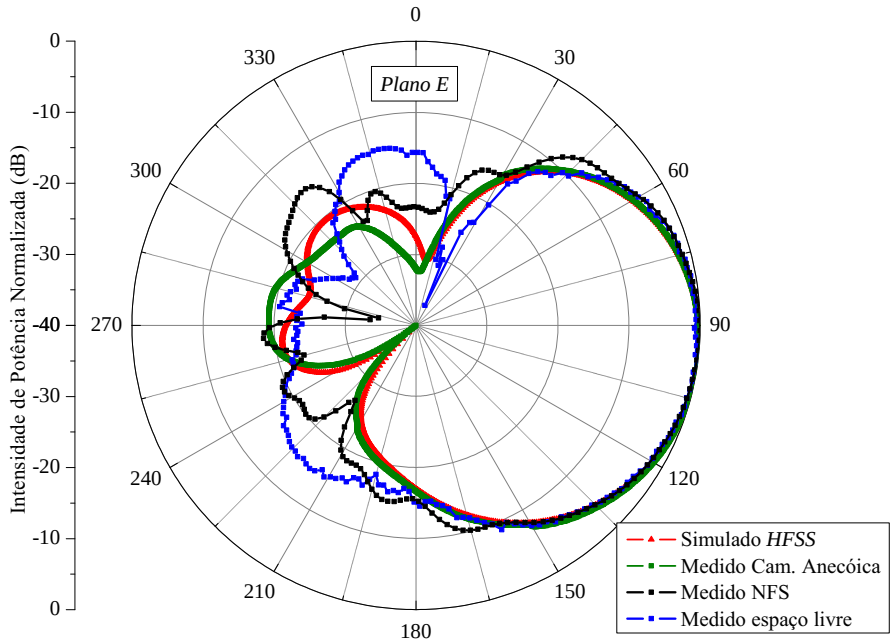
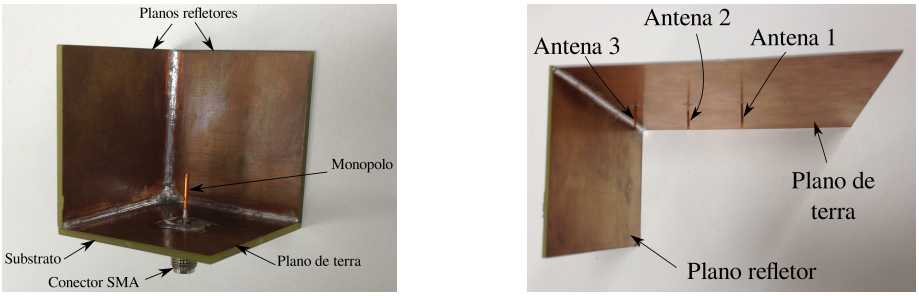


Figura 42: Antenas transmissora e receptora para testes à 5,8 GHz.

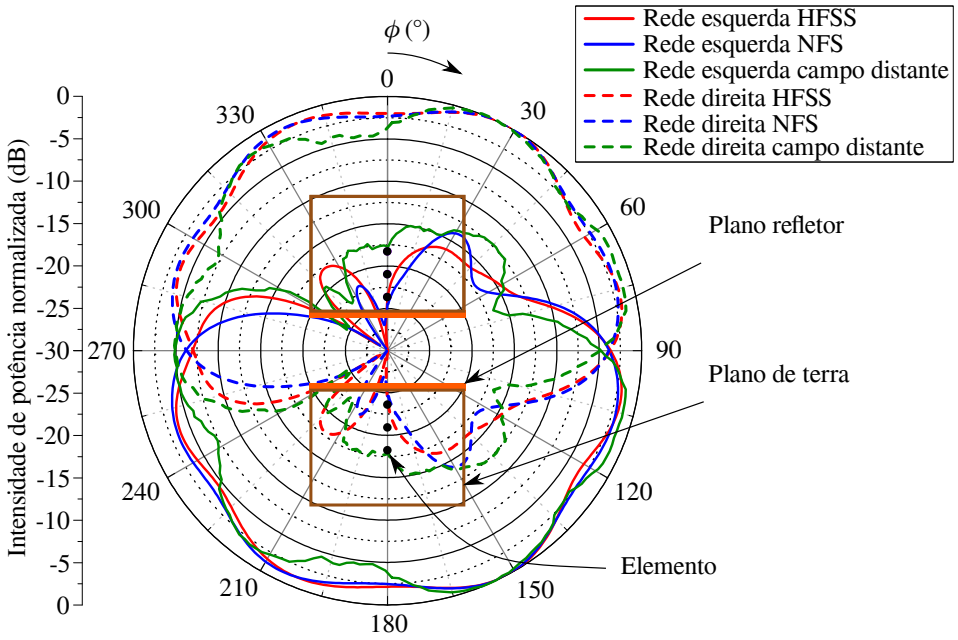


(a) Diedro refletor utilizado como transmissor.

(b) Rede de antenas filamentares utilizada para recepção.

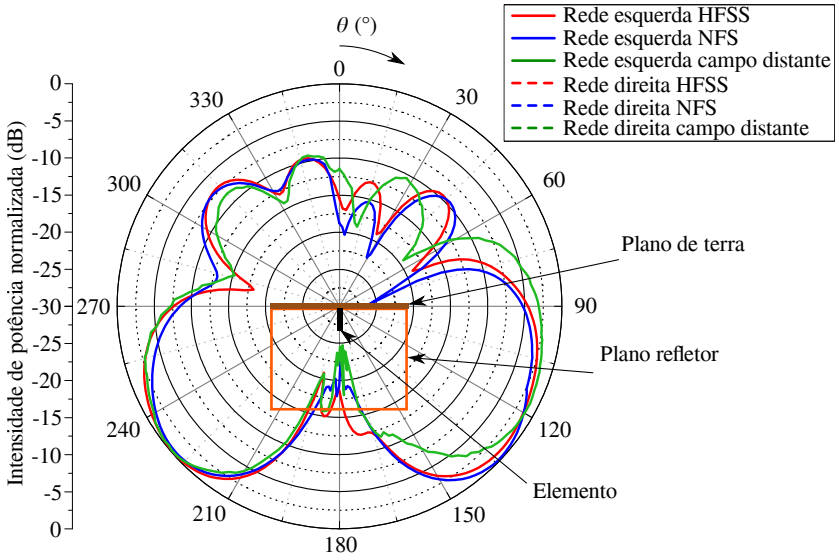
Fonte: (YOSHIMOTO, 2016).

Figura 43: Diagrama da rede de antenas filamentares em teste - vista superior.



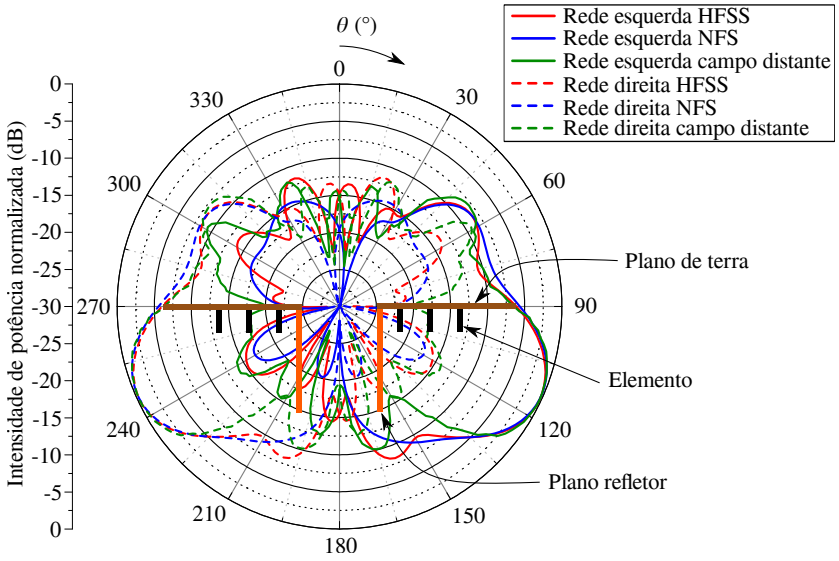
Fonte: (YOSHIMOTO, 2016).

Figura 44: Diagrama da rede de antenas filamentares em teste - vista lateral.



Fonte: (YOSHIMOTO, 2016).

Figura 45: Diagrama da rede de antenas filamentares em teste - vista frontal.



Fonte: (YOSHIMOTO, 2016).

5 Conclusão

No presente trabalho, foi apresentado o projeto de um posicionador automático de antenas, o qual foi acoplado a outros equipamentos para realizar medidas de diagramas de irradiação. Foram discutidos e detalhados todos os passos seguidos para a implementação do posicionador. Foi visto que todos os componentes do sistema precisam ser bem entendidos; caso contrário, a intercomunicação entre esses componentes seria inviável. Assim sendo, um estudo detalhado das principais funcionalidades dos equipamentos foi levantado, a fim de se conhecer seu comportamento e quais seriam as melhores opções e comandos de acesso remoto.

A ideia de se projetar um posicionador automático surgiu da necessidade de validar protótipos construídos em componentes curriculares do curso de Engenharia de Telecomunicações, como no componente de antenas. Um posicionador comercial apresenta um elevado custo de aquisição e, neste sentido, o projeto descrito é de baixo custo de implementação e bastante eficiente.

Foi construída uma estrutura giratória com movimento em um eixo, cuja rotação é dada por um motor de passo de $1,8°$ de precisão. Para o controle do motor e, consequentemente, da estrutura, foi desenvolvido um *driver* de potência à base de transistores, para chaveamento das fases das bobinas do motor. A placa do *driver* foi construída de maneira a acomodar um circuito de alimentação para um Arduino e três circuitos de sensores, utilizados para posicionamento e leitura de temperatura. O conjunto de acionamento e controle da estrutura é composto por um Arduino, uma *Shield Ethernet* e o circuito de potência mencionado. A escolha de utilizar Arduino para o projeto deveu-se à simplicidade de se trabalhar com a plataforma, alinhada ao baixo custo da placa.

Todo o sistema de medida é controlado remotamente através de um único dispositivo controlador, um *notebook*. A *Shield Ethernet* é capaz de armazenar um servidor *Web*, o qual pode ser acessado de um cliente qualquer, como um navegador de internet ou através do *Matlab* por TCP/IP. Este servidor *Web* é capaz de receber comandos e retransmiti-los ao Arduino, para que este execute a função desejada, como iniciar uma medição. O sistema conta com sensores para posicionamento correto da estrutura (calibração), para que as medições sejam mais precisas.

O projeto foi validado levantando-se o diagrama de irradiação de uma antena de

microfita e de uma rede de antenas filamentares. As curvas obtidas com o sistema foram comparadas aos diagramas obtidos por outros métodos, tais como simulações em *software* e medidas em câmaras anecoicas (campo próximo ou distante). Os resultados obtidos foram bastante satisfatórios.

Como continuação deste trabalho, pretende-se realizar algumas melhorias no sistema, tais como a aquisição de um motor com maior torque; otimização de códigos e da comunicação de dados entre dispositivos; organização da estrutura interna do posicionador para acomodar os circuitos e a fonte de alimentação; posicionamento mais eficiente dos sensores para uma calibragem e medida mais precisas; aprimorar o método de fixação das antenas em teste, no posicionador; aprimorar o sistema de monitoramento, através de técnicas mais eficientes de transmissão de vídeo digital.

Referências

ARDUINO. *LM35HigherResolution*. 2016. Disponível em: <http://playground.arduino.cc/Main/LM35HigherResolution>. Citado na página 70.

ARDUINO. *Using an Arduino as an AVR ISP (In-System Programmer)*. 2016. Disponível em: <https://www.arduino.cc/en/Tutorial/ArduinoISP>. Citado na página 41.

BACK, M. *Resistor PULL UP e PULL DOWN*. 2014. Disponível em: <http://arduinomais.blogspot.com.br/2014/05/resistor-pull-up-e-pull-down.html>. Citado na página 60.

BALANIS, C. A. *Teoria de Antenas*. Rio de Janeiro: LTC - tradução e revisão técnica J. R. Souza, 2009. Citado 8 vezes nas páginas 17, 18, 21, 22, 25, 27, 28 e 29.

BESSE, P.-A. et al. Detection of a single magnetic microbead using a miniaturized silicon hall sensor. *Applied Physics Letters*, AIP Publishing, v. 80, n. 22, p. 4199–4201, 2002. Citado na página 57.

BONFIM, M. *Analisador de Espectros*. 2003. Disponível em: <http://www.eletr.ufpr.br/marlio/medidashf/apostila/apostila2a.pdf>. Citado 2 vezes nas páginas 43 e 108.

BRAGA, I. N. C. *Como funcionam os sensores de Efeito Hall (ART1050)*. 2014. Disponível em: <http://www.newtoncbraga.com.br/index.php/como-funciona/6640-como-funcionam-os-sensores-de-efeito-hall-art1050>. Citado na página 57.

BRITES, F. G.; SANTOS, V. P. d. A. Motor de passo. 2008. Citado na página 26.

BROWN, B. C.; GOORA, F. G.; ROUSE, C. D. The design of an economical antenna-gain and radiation-pattern measurement system. *IEEE Antennas and Propagation Magazine*, v. 53, n. 4, p. 1–13, 2011. Citado 2 vezes nas páginas 17 e 21.

BROWN, G. *Como funciona a fonte de alimentação de um computador*. 2014. Disponível em: <http://tecnologia.hsw.uol.com.br/fonte-computador.htm>. Citado na página 57.

CAMPOS, A. *Entendendo os 6 pinos de ICSP dos Arduinos*. 2015. Disponível em: <http://br-arduino.org/2015/05/arduino-icsp-attiny-atmega.html>. Citado na página 39.

CAMPOS, A. *Mais memória no Arduino: indo além dos 2KB de RAM com a PROGMEM*. 2015. Disponível em: <http://br-arduino.org/2015/06/arduino-progmem-sram.html>. Citado na página 77.

CASSIOLATO, C. *Sensor Hall – A tecnologia dos Posicionadores Inteligentes de última geração*. 2006. Disponível em: <http://www.smar.com/brasil/noticias/conteudo?id_not=sensor-hall-a-tecnologia-dos-posicionadores-inteligentes-de-ultima-geracao>. Citado na página 57.

CONSORTIUM, S. et al. Standard commands for programmable instruments (scpi) volume 1: Syntax and style. *USA, May*, 1999. Citado na página 48.

ELECTRONICS, A. *Resistor Pull Up e Pull Down - Circuitos Digitais - aula*. 2013. Disponível em: <https://www.youtube.com/watch?v=AQ3PeiEw1gg>. Citado na página 60.

ELETRÔNICA, S. *Analisadores de Espectro: Entenda a importância desse instrumento na Automação Industrial*. 2001. Disponível em: <http://www.sabereletronica.com.br/artigos/2797-analisadores-de-espectro-entenda-a-importncia-desse-instrumento-na-automao-industrial>. Citado na página 107.

ELETRONICO, R. *Arduino – Como gravar/regravar o bootloader (Com UNO no MEGA)*. 2014. Disponível em: <https://robsoneletronico.wordpress.com/2014/08/22/arduino-como-gravarregravar-o-bootloader-com-uno-no-mega/>. Citado na página 41.

FILHO, D. O. B. *Curso de Arduino - Aula 1 - O que é o Arduino*. 2012. Disponível em: <http://www.robotizando.com.br/curso_arduino_o_que_e_arduino_pg1.php>. Citado na página 39.

FUSCO, V. F. *Teoria e técnicas de antenas: princípios e prática*. [S.l.]: Bookman Editora, 2009. Citado na página 26.

GONÇALVES, J. A. B.; PINTO, A. L. C. Motor de passo - arduino. 2012. Citado 6 vezes nas páginas 30, 33, 34, 35, 36 e 38.

KEISER, G. Optical fiber communication. *NY: McGraw-Hill*, 2000. Citado na página 51.

KRAUS, J. D.; MARHEFKA, R. J. *Antennas for all Applications; 960 pages; ; ISBN 0072321032*. [S.l.]: McGraw-Hill Science/Engineering/Math (not provided), 2001. Citado na página 21.

KUMMER, W. H. Basic array theory. *Proceedings of the IEEE*, v. 80, n. 1, p. 127–140, 1992. Citado na página 21.

MARTINEZ, M. *Topologias de Redes*. 2016. Disponível em: <http://www.infoescola.com/informatica/topologias-de-redes/>. Citado na página 51.

MATHWORKS. *Acquire Images from Webcams*. 2016. Disponível em: <http://www.mathworks.com/help/supportpkg/usbwebcams/ug/acquire-images-from-webcams.html>. Citado na página 82.

PINTO, P. *Redes – Sabe o que é o modelo OSI?* 2010. Disponível em: <http://pplware.sapo.pt/tutoriais/networking/redes-sabe-o-que-e-o-modelo-osi/>. Citado na página 53.

POZAR, D. M. *Microwave engineering*. [S.l.]: John Wiley & Sons, 2009. Citado na página 23.

SCHLOSSER, E. R. *Síntese de Redes Lineares de Antenas de Microfita com Diagramas de Irradiação Conformados para Sistemas de Comunicação 4G*. 2014. Citado na página 85.

SOARES, K. *O que é um Arduino e o que pode ser feito com ele?* 2013. Disponível em: <http://www.techtudo.com.br/noticias/noticia/2013/10/o-que-e-um-arduino-e-o-que-pode-ser-feito-com-ele.html>. Citado na página 40.

SOUZA, P. J. A. de. *Tutorial Motor de Passo - Parte 1: Introdução, Tipos, Modos de Acionamento*. 2012. Disponível em: <http://labdegaragem.com/profiles/blogs/tutorial-sobre-motor-de-passo>. Citado 4 vezes nas páginas 28, 30, 32 e 34.

TACIO, P. *O QUE É E PARA QUE SERVE O ARDUINO*. 2013. Disponível em: <http://www.mundodoshackers.com.br/o-que-e-e-para-que-serve-o-arduino>. Citado na página 40.

TECHNOLOGIES, K. *E4438C ESG Vector Signal Generator, 250 kHz to 6 GHz*. 2016. Disponível em: <http://www.keysight.com/pt/pd-1000004297%3Aepsg%3Apro/esg-vector-signal-generator?nid=-32463.536880956&cc=BR&lc=por>. Citado 2 vezes nas páginas 111 e 116.

TOLFO, S. M. Desenvolvimento de uma ferramenta computacional para síntese de redes de antenas. 2016. Citado 2 vezes nas páginas 24 e 26.

TORRES, G. *Como o Protocolo TCP/IP funciona - Parte 1*. 2007. Disponível em: <http://www.clubedohardware.com.br/artigos/como-o-protocolo-tcp-ip-funciona-parte-1/1351>. Citado na página 54.

YOSHIMOTO, E. *Projeto de Rede de Antenas Filamentares Embarcada em um Modelo Real de Aeronave Radiocontrolada*. 2016. Citado 3 vezes nas páginas 88, 93 e 94.

ZHU, B.; LI, H.-f.; LI, H. Labview driver program to intelligence instrument based on scpi language [j]. *Instrument Technique and Sensor*, v. 9, p. 020, 2008. Citado na página 46.

Apêndices

APÊNDICE A – Tipos de analisadores de espectro

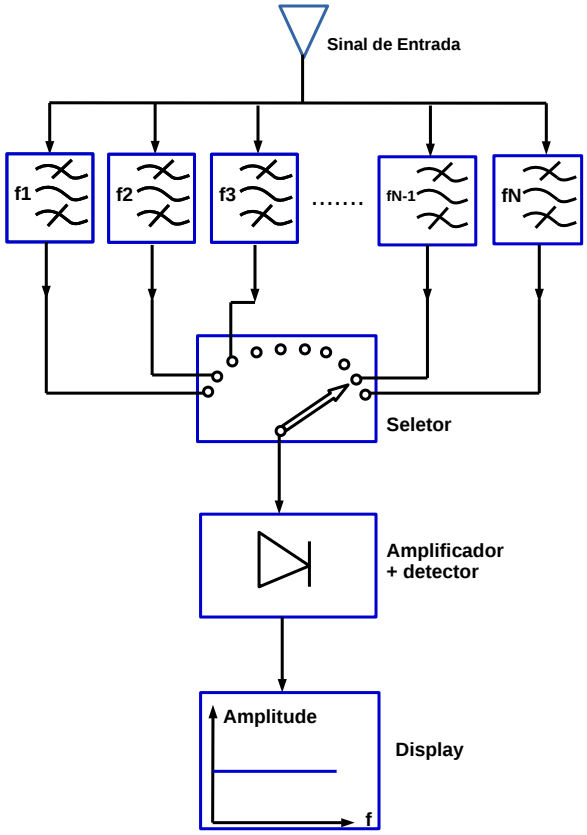
O analisador de espectro por banco de filtros possui filtros seletivos em frequência, cuja entrada é o sinal a ser analisado. Cada filtro possui uma frequência central e uma largura de banda, de modo a cobrir uma determinada faixa do espectro de frequências. A saída de cada filtro é retificada e então filtrada, onde o nível DC resultante é aplicado a um indicador visual como um display de LED's ou LCD, por exemplo. A medida é feita em paralelo.

Uma vez projetados os filtros, a frequência central e a largura de banda permanecem fixos, o que limita a faixa de frequências a ser analisada. Seu custo e complexidade de implementação tornam seu uso inviável quando uma alta resolução de frequência faz-se necessária, pelo grande número de filtros com uma largura de banda estreita. Essa topologia de analisador de espectro é largamente utilizada em indicadores de potência de áudio, onde a faixa de frequências é fixa (tipicamente 20Hz – 20kHz) e o número de bandas é relativamente pequeno (baixa resolução em frequência). A Fig. 46 mostra um diagrama em blocos com os componentes de um analisador de espectro por banco de filtros.

Uma forma de otimizar e minimizar o número de filtros usados na topologia banco de filtros, seria a utilização de um único filtro sintonizável em frequência através de um sinal de controle (rampa de tensão, controle digital), de modo a variar a frequência central ao longo da faixa espectral a ser analisada, fazendo-se uma varredura temporal. Dessa forma, o mesmo sinal de controle seria utilizado para indicar a frequência, podendo ser usado como eixo horizontal. Tal filtro sintonizável é realizável porém de difícil implementação, principalmente quando se trata de altas frequências como em sinais de comunicação via rádio, por exemplo.

Uma forma alternativa e mais simples de se fazer essa varredura, consiste na utilização de um filtro de frequência fixa associado a um processamento do sinal de entrada, de modo a deslocá-lo no espectro de frequência (varredura) de forma controlada. Uma forma simples de processamento é pela multiplicação analógica do sinal de entrada por um sinal senoidal (ou cossenoidal), cuja frequência pode ser facilmente controlada

Figura 46: Diagrama em blocos de um analisador de espectro por banco de filtros.



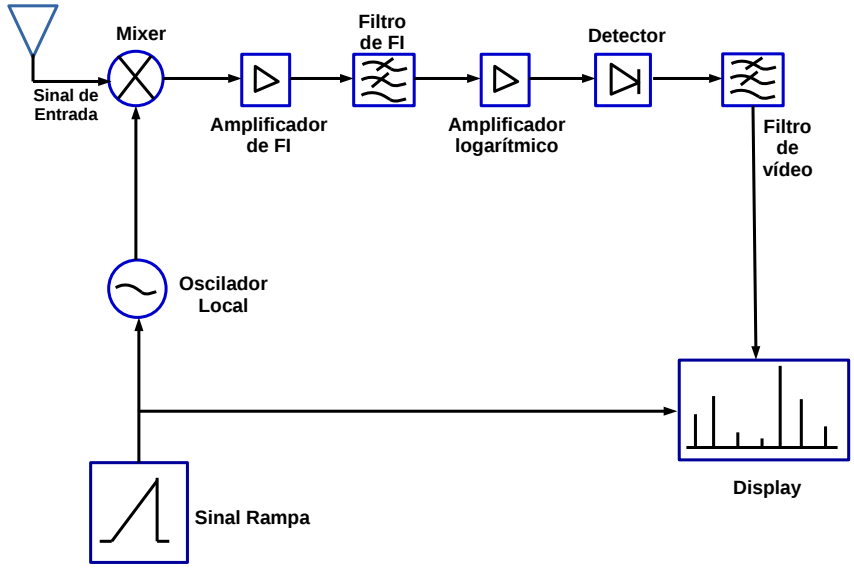
eletricamente. Este princípio é o mesmo utilizado em receptores de rádio AM, e a esse tipo de analisador dá-se o nome de analisador de espectro heteródino ou de varredura. Supondo que o sinal de entrada é composto por uma única frequência (ω_i), pode-se representar o processo de multiplicação por um outro sinal de frequência (ω_l) como descrito na eq. (A.1):

$$COS(\omega_l t)COS(\omega_i t) = \frac{COS(\omega_l + \omega_i)t + COS(\omega_l - \omega_i)t}{2} \tag{A.1}$$

Assim sendo, dado um filtro de frequência fixa ω_c, pode-se obter o resultado equivalente à um filtro variável pela varredura da frequência ω_l multiplicada pelo sinal de entrada. O processo de multiplicação é efetuado por um dispositivo não linear denominado misturador, o qual é composto essencialmente de diodos. A varredura de frequência é obtida pela utilização de um oscilador local (LO) controlado por tensão. Na Fig. 47 é

esquematizado, em diagrama de blocos, o funcionamento de um analisador de espectro por varredura (ou heteródino).

Figura 47: Diagrama em blocos de um analisador de espectro heteródino ou por varredura.



O funcionamento do circuito da Fig. 47 é essencialmente equivalente ao de um receptor de rádio AM (*Amplitude Modulation*) super-heteródino, e seus principais componentes são:

- **Seletor de escalas de entrada:** funciona de modo análogo ao de um osciloscópio, permitindo a adequação da amplitude do sinal de entrada ao instrumento.

- **Misturador:** é o principal elemento do circuito. Ele realiza, eletronicamente, a operação de multiplicação do sinal de entrada por um sinal senoidal de frequência f_{LO} (oscilador local), gerando em sua saída dois sinais principais correspondentes à soma e à diferença entre f_{LO} e a(s) frequência(s) presente(s) no sinal de entrada ($f_{LO}+f_i$ e f_{LO}-f_i). A parcela que será utilizada efetivamente na análise espectral, é o sinal diferença f_{LO}-f_i. Os sinais individuais f_{LO} e f_i também estão presentes na saída do misturador, devido à não idealidade da operação de multiplicação efetuada por esse circuito. O sinal na saída do misturador, também denominado sinal de frequência intermediária (IF), contém uma "cópia" do sinal de entrada transladado

em frequência por um valor f_{LO}. Variando-se a frequência do oscilador local f_{LO}, desloca-se proporcionalmente o sinal de entrada no domínio da frequência.

- **Oscilador Local:** gera um sinal senoidal de frequência f_{LO} que é aplicado à uma das entradas do misturador. Consiste em um oscilador controlado por tensão (VCO - *Voltage Control Oscilator*), cuja frequência pode ser variada continuamente dentro de uma faixa espectral selecionada pelo usuário. Possui uma alta linearidade entre tensão de controle e a frequência de saída, pois a mesma tensão é utilizada na varredura horizontal do feixe do TRC (Tubo de Raios Catódicos).

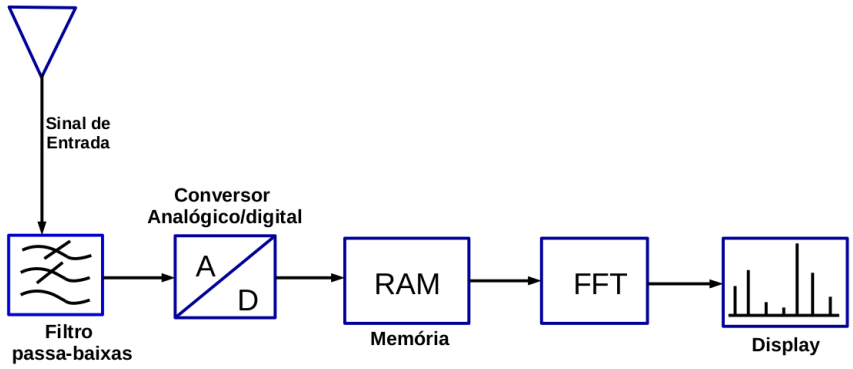
- **Gerador de rampa:** gera uma rampa de tensão em função do tempo (função rampa), que é utilizada no controle da frequência do oscilador local e também na varredura horizontal do TRC. É gerada de forma idêntica ao sinal de varredura horizontal de um osciloscópio analógico.

- **Filtro de IF:** é um filtro passa faixa de frequência central fixa f_{IF}, usado para selecionar a parcela do sinal de IF que contém o sinal a ser analisado a partir do sinal diferença f_{LO}-f_i. A largura de banda desse filtro determina a resolução em frequência do instrumento (RWB – *Resolution BandWidth*), podendo ser ajustada de acordo com o tipo de medida e sinal de entrada. Para que se possa distinguir entre dois sinais de frequências próximas (*e.g.*: 10kHz e 11kHz), é necessário que o RBW seja inferior à diferença entre essas frequências.

Realizando-se uma varredura na frequência f_{LO}, a parcela do sinal de entrada presente em f_{LO}-f_i é transladada em frequência. À medida que frequências presentes no sinal de entrada coincidem com a frequência f_{IF}, um sinal correspondente pode ser detectado na saída do filtro. Este processo é equivalente à uma varredura na frequência f_{IF}, porém bem mais fácil de ser implementado na prática. A frequência f_{IF} coincide com o menor valor de f_{LO}. Para $f_{LO} = f_{IF} \Rightarrow 0$Hz, equivalente à um sinal DC. O sinal de saída é retificado e filtrado, sendo em seguida aplicado ao circuito de deflexão vertical do TRC, de maneira análoga ao osciloscópio analógico.

Quanto aos analisadores de espectro por FFT, a Fig. 48 ilustra seu diagrama em blocos. Pode-se dizer, a grosso modo, que a diferença entre o analisador tipo FFT e o heteródino, é a faixa de frequências em que cada um pode operar. O FFT é destinado para baixas frequências (na ordem de 1000 kHz) enquanto o heteródino é para altas (e extra-altas) frequências (vários GHz).

Figura 48: Diagrama em blocos de um analisador de espectro por FFT.



A primeira etapa no diagrama da Fig. 48, é um filtro passa-baixas, que limita a frequência do sinal de entrada. Após a filtragem, o sinal é enviado a um conversor analógico/digital e, por ser de natureza transitória, é então armazenado temporariamente no bloco de memória RAM (*Random Access Memory*). O quarto bloco do instrumento é composto pelos circuitos de processamento, cujo *software* possui um algoritmo de cálculo, que implementa a determinação da série de Fourier do sinal (FFT). Este bloco, segundo as taxas de amostragem, resgata os dados armazenados na RAM e, após os cálculos da FFT, mostra através de um diagrama de barras, as respectivas amplitudes das frequências harmônicas de um sinal em uma tela (ELETRÔNICA, 2001).

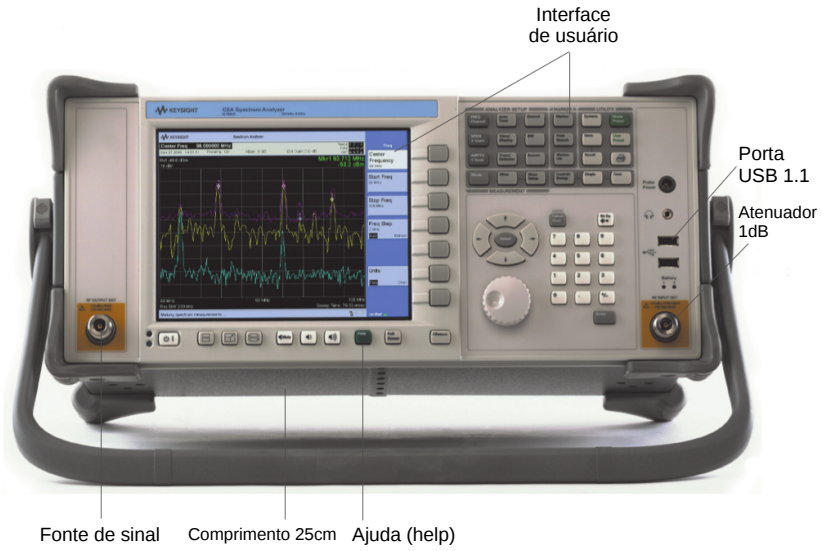
Analisadores de espectro modernos possuem inúmeras funções e controles, porém, as quatro principais são: faixa de frequência a ser exibida na tela, que determina o "tamanho" da figura a ser exibida na tela do analisador; faixa de nível, que determina os limites do sinal exibido; resolução da frequência, cujo ajuste é uma função do circuito de filtragem da frequência intermediária (IF), e é análogo ao controle "tempo/div" nos osciloscópios; sweep time, que é específico para os analisadores de espectro operando em modo heteródino, e determina o tempo necessário para a gravação do espectro de frequências a ser estudado.

É fato que a análise de espectro no domínio das frequências é mais comum no campo das telecomunicações, onde o estudo (e posterior ajuste) da frequência dos sinais transmitidos é fundamental para a boa performance do sistema. Entretanto, recentemente um novo modo de aplicação para o analisador de espectro ganhou muita importância: a automação industrial.

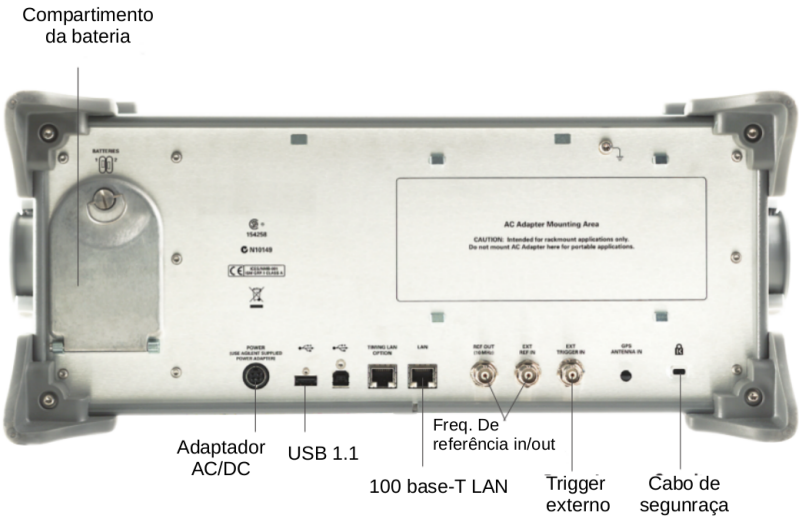
Muitos fabricantes de equipamentos de automação têm seu faturamento acres-

cido, em grande parte, pela exportação de seus produtos. Uma exigência comum dos consumidores internacionais é a "compatibilidade eletromagnética". A compatibilidade eletromagnética (EMC) é um conjunto de características que garantem que determinado equipamento não emite interferências eletromagnéticas (EMI) acima dos níveis permitidos pelos órgãos internacionais competentes. A EMC passou a ser um fator de qualidade do produto, aí é que entra a utilidade do analisador de espectro nesse cenário. Este instrumento é capaz de avaliar o nível de emissão eletromagnética e, o mais importante, determinar qual (ou quais) sua(s) faixa(s) de frequência(s) (BONFIM, 2003). A Fig. 49 traz imagens da parte frontal (49(a)) e da parte traseira (49(b)), respectivamente, de um analisador de espectro comercial do tipo heteródino, modelo N1996A, fabricado pela empresa Keysight Technologies e que é usado para o projeto descrito no cap. 3.

Figura 49: Analisador de espectro heteródino.



(a) Parte frontal.



(b) Parte traseira.

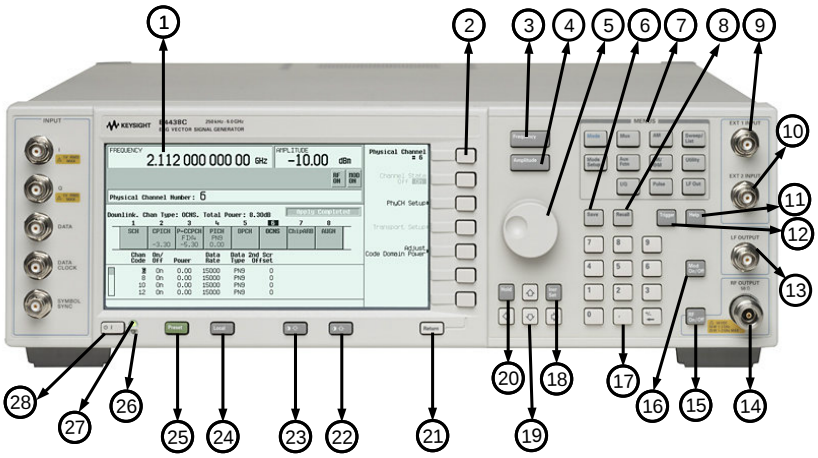
APÊNDICE B – Detalhes do gerador de sinais E4438C

O gerador de sinais ESG E4438C possui três formas de operação: onda contínua (CW - *continuous wave*); sinal de varredura (*Swept Signal*) e modulação analógica.

No modo de operação de onda contínua, o gerador de sinal está definido para uma única frequência e um único nível de potência. No modo *Swept Signal*, o gerador de sinal varre uma gama de frequências e/ou níveis de energia. No modo de modulação analógica, o gerador de sinal modula o sinal CW utilizando um dos quatro tipos de modulação analógica: AM, FM, ΦM ou modulação por pulso.

A Fig. 50 mostra a parte frontal do gerador de sinais da Keysight ESG E4438C com a numeração dos itens contidos no gerador, que serão discutidos a seguir.

Figura 50: Parte frontal do gerador de sinais E4438C da Keysight.



Fonte: Modificado de Technologies (2016).

Os itens numerados a seguir, fazem referência a Fig. 50.

1. ***Display* - visor**: O visor de LCD fornece informações sobre a função atual. A informação pode incluir indicadores de *status*, frequência, configurações de amplitude e mensagens de erro. As etiquetas para as teclas de função estão localizados no lado direito da tela.

2. **Teclas de função - *softkeys***: ativam a função indicada pela etiqueta mostrada à esquerda de cada tecla.

3. **Botão *Frequency***: Pressionando esta tecla, a função "frequência" fica ativa. É possível então, alterar a frequência de saída RF ou usar os menus para configurar atributos de frequência, como multiplicador, *offset* e referência.

4. ***Botão Amplitude***: Ativa a função "Amplitude", onde é possível alterar a amplitude de saída de RF ou usar os menus para configurar atributos de amplitude, como a busca de potência, nivelamento de usuário e ALC BW.

5. ***Botão Giratório (Knob)***: Girando-se o botão, aumenta-se ou diminui-se um valor numérico ou muda-se um dígito selecionado ou função. Também é possível usar o botão para percorrer as listas ou selecionar itens em uma fileira. O botão usa o valor ***Incr Set*** em conjunto com a razão de giro do botão (definido com a tecla ***Step/Knob Ratio***) para determinar quanto cada volta do botão muda o valor da função ativa. Por exemplo, se o valor ***Incr Set*** para a função ativa é de 10 dB e a razão de giro do botão é de 50 para 1, então cada volta do botão muda a função ativa de 0,2 dB (1/50º de 10 dB). Ao modificar qualquer um dos valores (ou ambos), altera-se a quantidade de voltas do botão.

6. ***Botão Save (salvar)***: Esta função permite gravar dados de registro de estado de memória do instrumento gerador de sinal. O registro de estado do instrumento é uma seção de memória dividida em 10 sequências numeradas de 0 a 9. Cada sequência contém 100 registros numeradas de 00 a 99. O botão ***Save*** fornece uma alternativa rápida para reconfigurar o gerador de sinais através do painel frontal ou com os comandos SCPI quando se muda entre diferentes configurações.

7. ***Teclas de Menu***: Estas teclas permitem acessar menus das teclas programáveis, que permitem a configuração de lista e passo de varredura. Permitem ainda, a configuração das funções mais úteis, configuração da saída de LF (*Low Frequency*), e vários tipos de modulações analógicas.

8. ***Teclas de Recall***: Esta tecla restaura qualquer estado do instrumento salvo anteriormente num registo de memória de estado do instrumento.

9. ***Conector de entrada externa 1 (EXT 1 Input Connector)***: Este conector de entrada BNC aceita um sinal de entrada para uso com AM, FM, ΦM, e modulação por pulso. Os níveis que podem danificar o instrumento são 5 V *rms* e 10 V_p.

10. ***Conector de entrada externa 2 (EXT 2 Input Connector)***: Este conector de entrada BNC aceita um sinal de entrada para uso com AM, FM, ΦM, e modulação por pulso. Os níveis que podem danificar o instrumento são 5 V *rms* e 10 V_p.

11. ***Tecla Help (Ajuda)***: exibe uma breve descrição de qualquer tecla de função. Existem dois modos de ajuda disponíveis: simples e contínua. O modo simples vem predefinido de fábrica. Para alternar entre o modo simples e contínuo, deve-se pressionar as teclas **Utility** > **Instrument Info/Help Mode** > **Help Mode Single Cont**.

12. ***Tecla Trigger (Disparo)***: Esta tecla inicia um evento de disparo imediato para uma função como uma lista ou passo de varredura. O modo de disparo deve ser definido como fator chave antes de iniciar um evento de gatilho com esta tecla.

13. ***Conector de saída de baixa frequência (LF OUTPUT Connector)***: Este conector BNC é a saída para sinais de modulação gerados pelo gerador de função de baixa frequência (LF). Esta saída é capaz de entregar 3 V_p (nominal) para uma carga de 50Ω. Em geradores de sinais com a opção "1EM", esta saída é transferida para um conector BNC fêmea, no painel traseiro.

14. ***Conector de saída de Rádio Frequência (RF OUTPUT Connector)***: Este conector fêmea tipo-N, é a saída de sinais de RF. A impedância da fonte é 50Ω. Para Opções 501, 502, 503 e 504 os níveis de perigo são 50 V DC, 50 W para f $\leq$ 2 GHz e 25 W para f > 2 GHz. Para opções 501, 502, 503 e 504, o circuito de proteção de potência reversa desarmará, no entanto, a potência nominal é de 1 W.

15. ***Tecla RF ON/OFF***: Essa tecla alterna o estado de funcionamento do sinal de RF presente no conector de saída de RF. O indicador RF On/Off é sempre visível no visor para indicar se o RF está ligado ou desligado.

16. ***Tecla Mod ON/OFF***: Esta tecla habilita ou desabilita todos os formatos de modulação ativos (AM, FM, ΦM, ou pulso) que são aplicados ao sinal da portadora

de saída. Este botão não configura ou ativa uma modulação AM, FM, ΦM, ou o formato de pulso; cada formato de modulação individual deve ser configurado e ativado (por exemplo, AM => AM On), ou nada será aplicado à saída para o sinal de portadora quando a tecla ***Mod On/Off*** está habilitada. O indicador ***MOD ON/OFF***, que está sempre presente no visor, indica se os formatos de modulação ativos foram ativados ou desativados com a tecla ***Mod On/Off***.

17. ***Teclado Numérico (Numeric Keypad)***: Consiste de teclas numeradas de 0 a 9, uma tecla de ponto decimal (.) e uma tecla de *backspace*. A tecla de *backspace* permite apagar um valor ou especificar um valor negativo. Ao especificar um valor numérico negativo, o sinal negativo deve ser inserido antes de entrar o valor numérico.

18. ***Tecla Incr Set Key***: Esta tecla permite definir o valor do incremento da função ativa atual. Quando a tecla é pressionada, o valor do incremento da função ativa atual aparece como a entrada ativa para a exibição. É possível utilizar o teclado numérico, botões de seta, ou o botão para ajustar o valor de incremento. Alterar o valor da função ***Incr Set*** também afeta o quanto cada volta do botão giratório muda o valor de uma função ativa, de acordo com a definição da razão atual do botão giratório.

19. ***Teclas de setas (Arrow Keys)***: As teclas de seta para cima e para baixo são usadas para aumentar ou diminuir um valor numérico, percorrer listas exibidas, ou selecionar itens em uma linha de uma lista exibida. Dígitos individuais ou caracteres podem ser selecionados com as teclas de seta esquerda e direita. Uma vez que um dígito ou caractere individual é selecionado, o seu valor pode ser alterado usando as teclas de seta para cima e/ou para baixo.

20. ***Tecla Hold (reter)***: Limpa a área de rótulo das teclas, a área de função ativa, e as áreas de texto do visor. As teclas de funções, as setas, os botões, o teclado numérico e a tecla ***Incr Set***, não têm qualquer efeito uma vez que esta tecla for pressionada. Para terminar o modo de espera, pressiona-se qualquer outra tecla que não seja as mencionadas anteriormente.

21. ***Tecla de retorno (Return)***: Esta tecla permite retornar ao menu anterior. Quando se está em um menu com mais de um nível, a tecla ***Return*** sempre retornará ao primeiro nível do menu.

22. ***Tecla de Aumento do Contraste do Visor***: Esta tecla, quando pressionada ou mantida, faz com que a imagem do visor escureça.

23. ***Tecla de decremento do Contraste do Visor***: Esta tecla, quando pressionada ou mantida, faz com que a imagem do visor fique mais clara.

24. ***Tecla Local***: Esta tecla é usada para desativar a operação remota e devolver o gerador de sinais para o controle do painel frontal.

25. ***Tecla Preset***: É utilizada para configurar o gerador de sinal para um estado conhecido (configuração de fábrica ou alguma outra configuração definida pelo usuário).

26. ***Standby LED***: Este LED amarelo indica quando o interruptor de alimentação do gerador de sinal está em modo de espera.

27. ***Line Power LED***: Este LED verde indica quando o interruptor de alimentação do gerador de sinal está definido como ligado.

28. ***Power Switch***: Este interruptor liga o gerador de sinal quando definido para a posição **Ligado**, e desativa todas as funções do gerador de sinal quando em modo de espera. Em modo de espera, o gerador de sinal permanece conectado à fonte de alimentação, que ainda alimenta alguns circuitos internos.

A Fig. 51 mostra a parte traseira do analisador ESG E4438C com seus principais componentes enumerados.

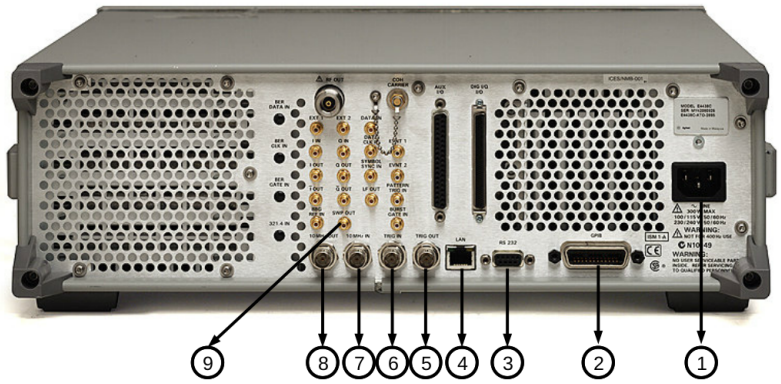
O detalhamento dos itens a seguir, dizem respeito à Fig. 51.

1. ***Entrada para alimentação AC***: A entrada do cabo de alimentação aceita um cabo de três pinos. A tensão da rede é ligada aqui.

2. ***Conector GPIB***: O conector GPIB permite a comunicação com dispositivos compatíveis, tais como controladores externos (computador). É funcionalmente equivalente aos conectores LAN e RS-232.

3. ***Conector RS-232***: Este conector é do tipo fêmea DB-9 e consiste em uma porta serial RS-232 que pode ser usada para controlar o gerador de sinal remotamente. É funcionalmente equivalente aos conectores GPIB e LAN.

Figura 51: Parte traseira do gerador de sinais E4438C da Keysight.



Fonte: Modificado de Technologies (2016).

4. ***Conector LAN***: Comunicação via LAN é suportada pelo gerador de sinal por meio do conector ***LAN*** (área de rede local). O conector de rede local permite que o gerador de sinal possa ser programado remotamente por um computador ou dispositivo controlador conectado à *LAN*. A distância entre um computador e o gerador de sinais, é limitada a 100 metros (10Base-T), supondo a utilização de um único cabo de rede.

5. ***Conector TRIG OUT***: Este conector BNC fêmea emite um sinal TTL que é colocado em nível alto no início de uma sequência existente, ou no início da espera do ponto de gatilho em modo de varrimento manual. O sinal TTL vai para nível baixo quando o tempo de espera é longo, quando o ponto de gatilho é recebido, ou uma vez por varredura durante uma varredura LF. A polaridade lógica pode ser invertida.

6. ***Conector TRIG IN***: Este conector BNC fêmea aceita um sinal CMOS para operações, como ponto-a-ponto em modo de varredura manual ou uma varredura LF no modo de oscilação externo. O disparo pode ocorrer tanto do lado positivo quanto do lado negativo. Os níveis de perigo são V > +5,5 volts e V < -0,5 volts.

7. ***Conector de entrada de 10MHz***: Este conector BNC fêmea aceita um sinal de -3,5 a 20 dBm proveniente de uma referência externa de 1, 2, 5, ou 10 MHz ± 0,2 ppm. A impedância de entrada nominal é de 50Ω. O gerador de sinal detecta quando

um sinal de referência válida está presente neste conector, e muda automaticamente o modo de operação de "interno" para o modo de "referência externa". O gerador de sinal irá mudar automaticamente apenas de interno para referência externa, quando o instrumento estiver no modo padrão de fábrica, ou se a tecla de função ***Ref Oscillator Source Auto Off On*** estiver ativada.

8. ***Conector de Saída de 10MHz***: Este conector BNC fêmea fornece um nível de sinal nominal de 3,9 dBm ± 2 dB e uma impedância de 50Ω de saída. A precisão é determinada pela base de tempo utilizada.

9. ***Conector Sweep OUT***: Este conector BNC fêmea fornece uma gama de tensões de 0 a 10 V. Quando o gerador de sinal está no modo de varredura, a varredura do sinal varia de 0 V no início do varrimento, até +10 V no final do varrimento. Isto é feito independentemente da largura de varrimento. No modo CW este conector não disponibiliza nenhum sinal em sua saída. A impedância de saída é inferior a 1Ω.

APÊNDICE C – Código do servidor *Web*

```
#include <SPI.h>
#include <Ethernet.h>
#include <EEPROM.h>

byte mac[] = {
0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(10,2,6,201);
byte gateway[] = {10,2,6,1};
byte subnet[] = { 255, 255, 254, 0 };
IPAddress myserver(0,0,0,0);
EthernetServer server(80);
EthernetClient client;
String readString;
const int tempo = 3000;
const int tempo2 = 700;
const int LM35 = A0;
float temperatura = 0;
float valorLido = 0;
int statePin3=LOW;
int statePin5=LOW;
int addr = 0;
int value = 0;
int aux = 0;
boolean desliga = false;

void setup(){

  analogReference(INTERNAL);
  pinMode(LM35, INPUT);
  pinMode(3, INPUT);
  pinMode(5, INPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.begin(9600);
```

```
39
   }
41

43 void loop(){

45   valorLido = analogRead(LM35);
     temperatura = valorLido/9.31;
47   Serial.print("Temperatura: ");
     Serial.println(temperatura);
49   delay(400);

51   pagina();

53   if(readString.indexOf("bobina1on") >0){

55     digitalWrite(6, HIGH);
       digitalWrite(7, HIGH);
57     digitalWrite(8, LOW);
       digitalWrite(9, LOW);
59    EEPROM.write(addr, 1);
       delay(tempo);
61
     }
63
     if(readString.indexOf("bobina2on") >0){
65
       digitalWrite(6, LOW);
67     digitalWrite(7, HIGH);
       digitalWrite(8, HIGH);
69     digitalWrite(9, LOW);
      EEPROM.write(addr, 2);
71     delay(tempo);

73   }

75   if(readString.indexOf("bobina3on") >0){

77     digitalWrite(6, LOW);
       digitalWrite(7, LOW);
79     digitalWrite(8, HIGH);
       digitalWrite(9, HIGH);
81    EEPROM.write(addr, 3);
```

```
      delay(tempo);

    }

    if(readString.indexOf("bobina4on") >0){

      digitalWrite(6, HIGH);
      digitalWrite(7, LOW);
      digitalWrite(8, LOW);
      digitalWrite(9, HIGH);
      EEPROM.write(addr, 4);
      delay(tempo);

    }

    if(readString.indexOf("direita") >0){

      direita();

    }

    if(readString.indexOf("esquerda") >0){

      esquerda();

    }


    if(readString.indexOf("autahon") >0){

      automaticoah();

    }

    if(readString.indexOf("authon") >0){

      automaticoh();

    }

    if(readString.indexOf("calibrar") >0){

      calibrar();
```

```
125
    }
127
    if(readString.indexOf("voltar") >0){
129
      voltar();
131
    }
133
    if(readString.indexOf("back") >0){
135
      voltar2();
137
    }
139
    if(readString.indexOf("bobina1off") >0 || readString.indexOf("
    bobina2off") >0 || readString.indexOf("bobina3off") >0 || readString.
    indexOf("bobina4off") >0 || readString.indexOf("parar") >0)
141
    {
143
      desligar();
145
    }
147
    readString="";
149 readString="";
    digitalWrite(6, LOW);
151 digitalWrite(7, LOW);
    digitalWrite(8, LOW);
153 digitalWrite(9, LOW);

155 }

157
void calibrar(){
159

161  statePin5=digitalRead(5);

163  while(readString.indexOf("calibrar") >0 && statePin5==LOW){

165    value = EEPROM.read(addr);
```

```
167     if (value == 4) {

169       digitalWrite(6, HIGH);
          digitalWrite(7, HIGH);
171       digitalWrite(8, LOW);
          digitalWrite(9, LOW);
173       EEPROM.write(addr, 1);
          delay(tempo2);
175
          pagina();
177
          statePin3=digitalRead(3);
179       statePin5=digitalRead(5);

181       if(statePin3==HIGH){
            voltar();
183       }

185       pagina();

187       if(readString.indexOf("parar") >0 || statePin5==HIGH){

189         desligar();

191       }

193       else {

195       digitalWrite(6, LOW);
          digitalWrite(7, HIGH);
197       digitalWrite(8, HIGH);
          digitalWrite(9, LOW);
199       EEPROM.write(addr, 2);
          delay(tempo2);
201
          }
203
          pagina();
205
          statePin3=digitalRead(3);
207       statePin5=digitalRead(5);
```

```
209         if(statePin3==HIGH){
              voltar();
211         }

213         pagina();

215         if(readString.indexOf("parar") >0 || statePin5==HIGH){

217           desligar();

219         }

221         else {

223         digitalWrite(6, LOW);
            digitalWrite(7, LOW);
225         digitalWrite(8, HIGH);
            digitalWrite(9, HIGH);
227         EEPROM.write(addr, 3);
            delay(tempo2);
229
            }
231
            pagina();
233
            statePin3=digitalRead(3);
235         statePin5=digitalRead(5);

237         if(statePin3==HIGH){
              voltar();
239         }

241         pagina();

243         if(readString.indexOf("parar") >0 || statePin5==HIGH){

245           desligar();

247         }

249         else {

251         digitalWrite(6, HIGH);
```

```
      digitalWrite(7, LOW);
253   digitalWrite(8, LOW);
      digitalWrite(9, HIGH);
255   EEPROM.write(addr, 4);
      delay(tempo2);
257
      }
259
      statePin3=digitalRead(3);
261   statePin5=digitalRead(5);
      pagina();
263
  }
265
  if (value == 3) {
267
      digitalWrite(6, HIGH);
269   digitalWrite(7, LOW);
      digitalWrite(8, LOW);
271   digitalWrite(9, HIGH);
      EEPROM.write(addr, 4);
273   delay(tempo2);

275   pagina();

277   statePin3=digitalRead(3);
      statePin5=digitalRead(5);
279
      if(statePin3==HIGH){
281     voltar();
      }
283
      pagina();
285
      if(readString.indexOf("parar") >0 || statePin5==HIGH){
287
        desligar();
289
      }
291
      else {
293
      digitalWrite(6, HIGH);
```

```
295         digitalWrite(7, HIGH);
            digitalWrite(8, LOW);
297         digitalWrite(9, LOW);
            EEPROM.write(addr, 1);
299         delay(tempo2);

301         }

303         pagina();

305         statePin3=digitalRead(3);
            statePin5=digitalRead(5);
307
            if(statePin3==HIGH){
309           voltar();
            }
311
            pagina();
313
            if(readString.indexOf("parar") >0 || statePin5==HIGH){
315
              desligar();
317
            }
319
            else {
321
            digitalWrite(6, LOW);
323         digitalWrite(7, HIGH);
            digitalWrite(8, HIGH);
325         digitalWrite(9, LOW);
            EEPROM.write(addr, 2);
327         delay(tempo2);

329         }

331         pagina();

333         statePin3=digitalRead(3);
            statePin5=digitalRead(5);
335
            if(statePin3==HIGH){
337           voltar();
```

```
      }
339
      pagina();
341
      if(readString.indexOf("parar") >0 || statePin5==HIGH){
343
        desligar();
345
      }
347
      else {
349
      digitalWrite(6, LOW);
351   digitalWrite(7, LOW);
      digitalWrite(8, HIGH);
353   digitalWrite(9, HIGH);
      EEPROM.write(addr, 3);
355   delay(tempo2);

357   }

359   statePin3=digitalRead(3);
      statePin5=digitalRead(5);
361   pagina();

363 }

365 if (value == 2) {

367   digitalWrite(6, LOW);
      digitalWrite(7, LOW);
369   digitalWrite(8, HIGH);
      digitalWrite(9, HIGH);
371   EEPROM.write(addr, 3);
      delay(tempo2);
373
      pagina();
375
      statePin3=digitalRead(3);
377   statePin5=digitalRead(5);

379   if(statePin3==HIGH){
        voltar();
```

```
381         }

383         pagina();

385         if(readString.indexOf("parar") >0 || statePin5==HIGH){

387           desligar();

389         }

391         else {

393         digitalWrite(6, HIGH);
            digitalWrite(7, LOW);
395         digitalWrite(8, LOW);
            digitalWrite(9, HIGH);
397         EEPROM.write(addr, 4);
            delay(tempo2);
399
            }
401
            pagina();
403
            statePin3=digitalRead(3);
405         statePin5=digitalRead(5);

407         if(statePin3==HIGH){
              voltar();
409         }

411         pagina();

413         if(readString.indexOf("parar") >0 || statePin5==HIGH){

415           desligar();

417         }

419         else {

421         digitalWrite(6, HIGH);
            digitalWrite(7, HIGH);
423         digitalWrite(8, LOW);
```

```
      digitalWrite(9, LOW);
      EEPROM.write(addr, 1);
      delay(tempo2);

      }

      pagina();

      statePin3=digitalRead(3);
      statePin5=digitalRead(5);

      if(statePin3==HIGH){
        voltar();
      }

      pagina();

      if(readString.indexOf("parar") >0 || statePin5==HIGH){

        desligar();

      }

      else {

      digitalWrite(6, LOW);
      digitalWrite(7, HIGH);
      digitalWrite(8, HIGH);
      digitalWrite(9, LOW);
      EEPROM.write(addr, 2);
      delay(tempo2);

      }

      statePin3=digitalRead(3);
      statePin5=digitalRead(5);
      pagina();

  }

  if (value == 1) {

      digitalWrite(6, LOW);
```

```
467         digitalWrite(7, HIGH);
            digitalWrite(8, HIGH);
469         digitalWrite(9, LOW);
            EEPROM.write(addr, 2);
471         delay(tempo2);

473         pagina();

475         statePin3=digitalRead(3);
            statePin5=digitalRead(5);
477
            if(statePin3==HIGH){
479           voltar();
            }
481
            pagina();
483
            if(readString.indexOf("parar") >0 || statePin5==HIGH){
485
              desligar();
487
            }
489
            else {
491
            digitalWrite(6, LOW);
493         digitalWrite(7, LOW);
            digitalWrite(8, HIGH);
495         digitalWrite(9, HIGH);
            EEPROM.write(addr, 3);
497         delay(tempo2);

499         }

501         pagina();

503         statePin3=digitalRead(3);
            statePin5=digitalRead(5);
505
            if(statePin3==HIGH){
507           voltar();
            }
509
```

```
      pagina();

      if(readString.indexOf("parar") >0 || statePin5==HIGH){

        desligar();

      }

      else {

      digitalWrite(6, HIGH);
      digitalWrite(7, LOW);
      digitalWrite(8, LOW);
      digitalWrite(9, HIGH);
      EEPROM.write(addr, 4);
      delay(tempo2);

      }

      pagina();

      statePin3=digitalRead(3);
      statePin5=digitalRead(5);

      if(statePin3==HIGH){
        voltar();
      }

      pagina();

      if(readString.indexOf("parar") >0 || statePin5==HIGH){

        desligar();

      }

      else {

      digitalWrite(6, HIGH);
      digitalWrite(7, HIGH);
      digitalWrite(8, LOW);
      digitalWrite(9, LOW);
      EEPROM.write(addr, 1);
```

```
553         delay(tempo2);

555         }

557         statePin3=digitalRead(3);
            statePin5=digitalRead(5);
559         pagina();

561       }

563     }

565 }

567 void automaticoh(){

569     while(readString.indexOf("authon") > 0 && aux<101){

571       value = EEPROM.read(addr);

573       if (value == 1) {

575       digitalWrite(6, LOW);
          digitalWrite(7, HIGH);
577       digitalWrite(8, HIGH);
          digitalWrite(9, LOW);
579       EEPROM.write(addr, 2);
          aux = aux+1;
581       delay(tempo);

583       pagina();

585       if(aux==100){

587         voltar();
            break;
589

          }
591

          pagina();
593

          if(readString.indexOf("parar") >0){
595
```

```
      desligar();

    }

    else {

    digitalWrite(6, LOW);
    digitalWrite(7, LOW);
    digitalWrite(8, HIGH);
    digitalWrite(9, HIGH);
    EEPROM.write(addr, 3);
    aux = aux+1;
    delay(tempo);

    }

    pagina();

    if(aux==100){

      voltar();
      break;

    }

    pagina();

    if(readString.indexOf("parar") >0){

      desligar();

    }

    else {

    digitalWrite(6, HIGH);
    digitalWrite(7, LOW);
    digitalWrite(8, LOW);
    digitalWrite(9, HIGH);
    EEPROM.write(addr, 4);
    aux = aux+1;
    delay(tempo);
```

```
639       }

641       pagina();

643       if(aux==100){

645         voltar();
          break;
647
       }
649
       pagina();
651
       if(readString.indexOf("parar") >0){
653
         desligar();
655
       }
657
       else {
659
       digitalWrite(6, HIGH);
661    digitalWrite(7, HIGH);
       digitalWrite(8, LOW);
663    digitalWrite(9, LOW);
      EEPROM.write(addr, 1);
665    aux = aux+1;
       delay(tempo);
667
       }
669
       pagina();
671
  }
673
  if (value == 2) {
675
       digitalWrite(6, LOW);
677    digitalWrite(7, LOW);
       digitalWrite(8, HIGH);
679    digitalWrite(9, HIGH);
      EEPROM.write(addr, 3);
681    aux = aux+1;
```

```
    delay(tempo);
683
    pagina();
685
    if(aux==100){
687
      voltar();
689   break;

691 }

693 pagina();

695 if(readString.indexOf("parar") >0){

697   desligar();

699 }

701 else {

703 digitalWrite(6, HIGH);
    digitalWrite(7, LOW);
705 digitalWrite(8, LOW);
    digitalWrite(9, HIGH);
707 EEPROM.write(addr, 4);
    aux = aux+1;
709 delay(tempo);

711 }

713 pagina();

715 if(aux==100){

717   voltar();
      break;
719
    }
721
    pagina();
723
    if(readString.indexOf("parar") >0){
```

```
725
        desligar();
727
      }
729
      else {
731
      digitalWrite(6, HIGH);
733   digitalWrite(7, HIGH);
      digitalWrite(8, LOW);
735   digitalWrite(9, LOW);
      EEPROM.write(addr, 1);
737   aux = aux+1;
      delay(tempo);
739
      }
741
      pagina();
743
      if(aux==100){
745
        voltar();
747     break;

749   }

751   pagina();

753   if(readString.indexOf("parar") >0 ||  statePin5==HIGH){

755     desligar();

757   }

759   else {

761   digitalWrite(6, LOW);
      digitalWrite(7, HIGH);
763   digitalWrite(8, HIGH);
      digitalWrite(9, LOW);
765   EEPROM.write(addr, 2);
      aux = aux+1;
767   delay(tempo);
```

```
769     }

771     pagina();

773   }

775     if (value == 3) {

777     digitalWrite(6, HIGH);
        digitalWrite(7, LOW);
779     digitalWrite(8, LOW);
        digitalWrite(9, HIGH);
781     EEPROM.write(addr, 4);
        aux = aux+1;
783     delay(tempo);

785     pagina();

787     if(aux==100){

789       voltar();
          break;
791
        }
793
        pagina();
795
        if(readString.indexOf("parar") >0){
797
          desligar();
799
        }
801
        else {
803
        digitalWrite(6, HIGH);
805     digitalWrite(7, HIGH);
        digitalWrite(8, LOW);
807     digitalWrite(9, LOW);
        EEPROM.write(addr, 1);
809     aux = aux+1;
        delay(tempo);
```

```
811
    }
813
    pagina();
815
    if(aux==100){
817
      voltar();
819   break;

821 }

823 pagina();

825 if(readString.indexOf("parar") >0){

827   desligar();

829 }

831 else {

833 digitalWrite(6, LOW);
    digitalWrite(7, HIGH);
835 digitalWrite(8, HIGH);
    digitalWrite(9, LOW);
837 EEPROM.write(addr, 2);
    aux = aux+1;
839 delay(tempo);

841 }

843 pagina();

845 if(aux==100){

847   voltar();
      break;
849
    }
851
    pagina();
853
```

```
    if(readString.indexOf("parar") >0){
855
      desligar();
857
    }
859
    else {
861
    digitalWrite(6, LOW);
863 digitalWrite(7, LOW);
    digitalWrite(8, HIGH);
865 digitalWrite(9, HIGH);
    EEPROM.write(addr, 3);
867 aux = aux+1;
    delay(tempo);
869
    }
871
    pagina();
873
  }
875
    if (value == 4) {
877
    digitalWrite(6, HIGH);
879 digitalWrite(7, HIGH);
    digitalWrite(8, LOW);
881 digitalWrite(9, LOW);
    EEPROM.write(addr, 1);
883 aux = aux+1;
    delay(tempo);
885
    pagina();
887
    if(aux==100){
889
      voltar();
891   break;

893 }

895 pagina();
```

```
897     if(readString.indexOf("parar") >0){

899       desligar();

901     }

903     else {

905     digitalWrite(6, LOW);
        digitalWrite(7, HIGH);
907     digitalWrite(8, HIGH);
        digitalWrite(9, LOW);
909     EEPROM.write(addr, 2);
        aux = aux+1;
911     delay(tempo);

913     }

915     pagina();

917     if(aux==100){

919       voltar();
          break;
921
        }
923
        pagina();
925
        if(readString.indexOf("parar") >0){
927
          desligar();
929
        }
931
        else {
933
        digitalWrite(6, LOW);
935     digitalWrite(7, LOW);
        digitalWrite(8, HIGH);
937     digitalWrite(9, HIGH);
        EEPROM.write(addr, 3);
939     aux = aux+1;
```

```
      delay(tempo);
941
      }
943
      pagina();
945
      if(aux==100){
947
        voltar();
949     break;

951   }

953   pagina();

955   if(readString.indexOf("parar") >0){

957     desligar();

959   }

961   else {

963   digitalWrite(6, HIGH);
      digitalWrite(7, LOW);
965   digitalWrite(8, LOW);
      digitalWrite(9, HIGH);
967   EEPROM.write(addr, 4);
      aux = aux+1;
969   delay(tempo);

971   }

973   pagina();

975 }


977
 }
979


981 }
```

```
983 void automaticoah(){

985   while(readString.indexOf("autahon") >0 && aux<101){

987     value = EEPROM.read(addr);

989     if (value == 1) {

991     digitalWrite(6, HIGH);
        digitalWrite(7, LOW);
993     digitalWrite(8, LOW);
        digitalWrite(9, HIGH);
995     EEPROM.write(addr, 4);
        aux = aux+1;
997     delay(tempo);

999     if(aux==100){

1001      voltar2();
          break;
1003
        }
1005
        pagina();
1007
        if(readString.indexOf("parar") >0){
1009
          desligar();
1011
        }
1013
        else {
1015
        digitalWrite(6, LOW);
1017    digitalWrite(7, LOW);
        digitalWrite(8, HIGH);
1019    digitalWrite(9, HIGH);
        EEPROM.write(addr, 3);
1021    aux = aux+1;
        delay(tempo);
1023
        }
1025
```

```
    if(aux==100){
1027
      voltar2();
1029  break;

1031  }

1033  pagina();

1035  if(readString.indexOf("parar") >0){

1037    desligar();

1039  }

1041  else {

1043  digitalWrite(6, LOW);
      digitalWrite(7, HIGH);
1045  digitalWrite(8, HIGH);
      digitalWrite(9, LOW);
1047  EEPROM.write(addr, 2);
      aux = aux+1;
1049  delay(tempo);

1051  }

1053 if(aux==100){

1055    voltar2();
        break;
1057
      }
1059
      pagina();
1061
      if(readString.indexOf("parar") >0){
1063
        desligar();
1065
      }
1067
      else {
```

```
1069
      digitalWrite(6, HIGH);
1071  digitalWrite(7, HIGH);
      digitalWrite(8, LOW);
1073  digitalWrite(9, LOW);
     EEPROM.write(addr, 1);
1075  aux = aux+1;
      delay(tempo);
1077
      }
1079
      pagina();
1081
    }
1083
    if (value == 2) {
1085
      digitalWrite(6, HIGH);
1087  digitalWrite(7, HIGH);
      digitalWrite(8, LOW);
1089  digitalWrite(9, LOW);
     EEPROM.write(addr, 1);
1091  aux = aux+1;
      delay(tempo);
1093
     if(aux==100){
1095
        voltar2();
1097    break;

1099  }

1101  pagina();

1103  if(readString.indexOf("parar") >0){

1105    desligar();

1107  }

1109  else {

1111  digitalWrite(6, HIGH);
```

```
    digitalWrite(7, LOW);
1113    digitalWrite(8, LOW);
    digitalWrite(9, HIGH);
1115    EEPROM.write(addr, 4);
    aux = aux+1;
1117    delay(tempo);

1119    }

1121    if(aux==100){

1123      voltar2();
      break;
1125
    }
1127
    pagina();
1129
    if(readString.indexOf("parar") >0){
1131
      desligar();
1133
    }
1135
    else {
1137
    digitalWrite(6, LOW);
1139    digitalWrite(7, LOW);
    digitalWrite(8, HIGH);
1141    digitalWrite(9, HIGH);
    EEPROM.write(addr, 3);
1143    aux = aux+1;
    delay(tempo);
1145
    }
1147
    if(aux==100){
1149
      voltar2();
1151      break;

1153    }
```

```
1155      pagina();

1157      if(readString.indexOf("parar") >0){

1159        desligar();

1161      }

1163      else {

1165      digitalWrite(6, LOW);
          digitalWrite(7, HIGH);
1167      digitalWrite(8, HIGH);
          digitalWrite(9, LOW);
1169      EEPROM.write(addr, 2);
          aux = aux+1;
1171      delay(tempo);

1173      }

1175      pagina();

1177    }

1179      if (value == 3) {

1181      digitalWrite(6, LOW);
          digitalWrite(7, HIGH);
1183      digitalWrite(8, HIGH);
          digitalWrite(9, LOW);
1185      EEPROM.write(addr, 2);
          aux = aux+1;
1187      delay(tempo);

1189     if(aux==100){

1191        voltar2();
          break;
1193
          }
1195
          pagina();
1197
```

```
    if(readString.indexOf("parar") >0){
1199
      desligar();
1201
    }
1203
    else {
1205
    digitalWrite(6, HIGH);
1207 digitalWrite(7, HIGH);
    digitalWrite(8, LOW);
1209 digitalWrite(9, LOW);
    EEPROM.write(addr, 1);
1211 aux = aux+1;
    delay(tempo);
1213
    }
1215
    if(aux==100){
1217
      voltar2();
1219  break;

1221 }

1223 pagina();

1225 if(readString.indexOf("parar") >0){

1227   desligar();

1229 }

1231 else {

1233 digitalWrite(6, HIGH);
    digitalWrite(7, LOW);
1235 digitalWrite(8, LOW);
    digitalWrite(9, HIGH);
1237 EEPROM.write(addr, 4);
    aux = aux+1;
1239 delay(tempo);
```

```
1241      }

1243      if(aux==100){

1245        voltar2();
          break;
1247

      }
1249

      pagina();
1251

      if(readString.indexOf("parar") >0){
1253

        desligar();
1255

      }
1257

      else {
1259

      digitalWrite(6, LOW);
1261  digitalWrite(7, LOW);
      digitalWrite(8, HIGH);
1263  digitalWrite(9, HIGH);
     EEPROM.write(addr, 3);
1265  aux = aux+1;
      delay(tempo);
1267

      }
1269

      pagina();
1271

    }
1273

    if (value == 4) {
1275

      digitalWrite(6, LOW);
1277  digitalWrite(7, LOW);
      digitalWrite(8, HIGH);
1279  digitalWrite(9, HIGH);
     EEPROM.write(addr, 3);
1281  aux = aux+1;
      delay(tempo);
1283
```

```
    if(aux==100){
1285
      voltar2();
1287  break;

1289  }

1291  pagina();

1293  if(readString.indexOf("parar") >0){

1295    desligar();

1297  }

1299  else {

1301  digitalWrite(6, LOW);
      digitalWrite(7, HIGH);
1303  digitalWrite(8, HIGH);
      digitalWrite(9, LOW);
1305  EEPROM.write(addr, 2);
      aux = aux+1;
1307  delay(tempo);

1309  }

1311  if(aux==100){

1313    voltar2();
      break;
1315
    }
1317
    pagina();
1319
    if(readString.indexOf("parar") >0){
1321
      desligar();
1323
    }
1325
    else {
```

```
1327
      digitalWrite(6, HIGH);
1329  digitalWrite(7, HIGH);
      digitalWrite(8, LOW);
1331  digitalWrite(9, LOW);
      EEPROM.write(addr, 1);
1333  aux = aux+1;
      delay(tempo);
1335
      }
1337
      if(aux==100){
1339
        voltar2();
1341    break;

1343  }

1345  pagina();

1347  if(readString.indexOf("parar") >0){

1349    desligar();

1351  }

1353  else {

1355  digitalWrite(6, HIGH);
      digitalWrite(7, LOW);
1357  digitalWrite(8, LOW);
      digitalWrite(9, HIGH);
1359  EEPROM.write(addr, 4);
      aux = aux+1;
1361  delay(tempo);

1363  }

1365  pagina();

1367 }

1369
```

```
  }
1371
1373 }
1375 void pagina(){
1377   if (Serial.available() > 0){
1379    byte inChar;
       inChar = Serial.read();
1381    if(inChar == 'g'){
1383    }
1385   }
1387   EthernetClient client = server.available();
       if (client) {
1389    while (client.connected()) {
       if (client.available()>0) {
1391     char c = client.read();
        if (readString.length() < 100) {
1393     readString += c;
        }
1395
        if (c == '\n') {
1397      client.println(F("HTTP/1.1 200 OK"));
         client.println(F("Content-Type: text/html"));
1399      client.println();
         /*****************************Pagina em HTML
       ****************************/
1401      client.println(F("<HTML>"));
         client.println(F("<HEAD>"));
1403      client.println(F("<TITLE>Arduino - Acionamento de Motor de passo</
       TITLE>"));
         client.println(F("</HEAD>"));
1405      client.println(F("<BODY><center><img src=http://w3.ufsm.br/telecom/
       images/banner.png height=200 width=600 align=bottom></center>"));
         client.println(F("<BODY>"));
1407      client.println(F("<center><H1>Acionamento de Motor de passo com
       Arduino</H1></center>"));
         client.println(F("<center>"));
```

```
1409
      client.println(F("<h4>Bobina 1</h4>"));
1411  client.println(F("<a href=\"/?bobina1on\" target=\"inlineframe\">ON</
      a>"));
      client.println(F("<a href=\"/?bobina1off\" target=\"inlineframe\">OFF
      </a>"));
1413
      client.println(F("<h4>Bobina 2</h4>"));
1415  client.println(F("<a href=\"/?bobina2on\" target=\"inlineframe\">ON</
      a>"));
      client.println(F("<a href=\"/?bobina2off\" target=\"inlineframe\">OFF
      </a>"));
1417
      client.println("<h4>Bobina 3</h4>");
1419  client.println("<a href=\"/?bobina3on\" target=\"inlineframe\">ON</a>
      ");
      client.println("<a href=\"/?bobina3off\" target=\"inlineframe\">OFF</
      a>");
1421
      client.println("<h4>Bobina 4</h4>");
1423  client.println("<a href=\"/?bobina4on\" target=\"inlineframe\">ON</a>
      ");
      client.println("<a href=\"/?bobina4off\" target=\"inlineframe\">OFF</
      a>");
1425
      client.print(F("<BR>"));
1427
      client.println("<h4>Girar Passo-a-Passo</h4>");
1429  client.println("<a href=\"/?direita\" target=\"inlineframe\">Direita
      </a>");
      client.println("<a href=\"/?esquerda\" target=\"inlineframe\">
      Esquerda</a>");
1431
      client.println("<h4>Controle</h4>");
1433  client.println(F("<a href=\"/?calibrar\" target=\"inlineframe\">
      Calibrar</a>"));
      client.print(F("<BR>"));
1435  client.print(F("<BR>"));
      client.println(F("<a href=\"/?authon\" target=\"inlineframe\">Girar a
       direita</a>"));
1437  client.print(F("<BR>"));
      client.print(F("<BR>"));
```

```
1439      client.println(F("<a href=\"/?autahon\" target=\"inlineframe\">Girar
    a esquerda</a>"));
          client.print(F("<BR>"));
1441      client.print(F("<BR>"));
          client.println(F("<a href=\"/?voltar\" target=\"inlineframe\">Voltar
    AH</a>"));
1443      client.print(F("<BR>"));
          client.print(F("<BR>"));
1445      client.println(F("<a href=\"/?back\" target=\"inlineframe\">Voltar H
    </a>"));
          client.print(F("<BR>"));
1447      client.print(F("<BR>"));
          client.println(F("<a href=\"/?parar\" target=\"inlineframe\">Parar</a
    >"));
1449      client.println(F("</center>"));

1451      client.print(F("<BR>"));

1453      client.print(F("<center><h4>Temperatura do Sistema: </h4> <font size
    =7>  <font color=\"#ff6600\">  "));
          client.print(temperatura);
1455      client.print(F("&#186;C </font></font></center>"));
          client.print(F("<BR>"));
1457      client.println("<video id=video src=http://10.2.6.176/desktop.ogg
    type=video/ogg; codecs=theora autoplay=autoplay>");
          client.println(F("<IFRAME name=inlineframe style=\"display:none\" >")
    );
1459      client.println(F("</IFRAME>"));
          client.println(F("</BODY>"));
1461      client.println(F("</HTML>"));

1463     delay(1);
         client.stop();
1465

         }
1467

         }
1469

         }
1471    }
       }
1473
```

```
1475
void desligar(){
1477
    digitalWrite(6, LOW);
1479    digitalWrite(7, LOW);
    digitalWrite(8, LOW);
1481    digitalWrite(9, LOW);
    desliga = true;
1483    readString="";
    readString="";
1485
}
1487
void voltar(){
1489
   desliga = false;
1491   aux = 0;
   statePin5=digitalRead(5);
1493
   while(desliga==false && statePin5==LOW){
1495
    value = EEPROM.read(addr);
1497
    if (value == 1){
1499
    digitalWrite(6, HIGH);
1501    digitalWrite(7, LOW);
    digitalWrite(8, LOW);
1503    digitalWrite(9, HIGH);
    EEPROM.write(addr, 4);
1505    delay(tempo2);

1507    pagina();

1509    statePin5=digitalRead(5);

1511    if(readString.indexOf("parar") >0 || statePin5==HIGH){

1513      desligar();
      break;
1515
    }
1517
```

```
    else {
1519
    digitalWrite(6, LOW);
1521 digitalWrite(7, LOW);
    digitalWrite(8, HIGH);
1523 digitalWrite(9, HIGH);
    EEPROM.write(addr, 3);
1525 delay(tempo2);

1527 }

1529 pagina();

1531 statePin5=digitalRead(5);

1533 if(readString.indexOf("parar") >0 || statePin5==HIGH){

1535   desligar();
       break;
1537
    }
1539
    else {
1541
    digitalWrite(6, LOW);
1543 digitalWrite(7, HIGH);
    digitalWrite(8, HIGH);
1545 digitalWrite(9, LOW);
    EEPROM.write(addr, 2);
1547 delay(tempo2);

1549 }

1551 pagina();

1553 statePin5=digitalRead(5);

1555 if(readString.indexOf("parar") >0 || statePin5==HIGH){

1557   desligar();
       break;
1559
    }
```

```
1561
      else {
1563
      digitalWrite(6, HIGH);
1565  digitalWrite(7, HIGH);
      digitalWrite(8, LOW);
1567  digitalWrite(9, LOW);
      EEPROM.write(addr, 1);
1569  delay(tempo2);

1571  }

1573  pagina();
      statePin5=digitalRead(5);
1575
    }
1577
    if (value == 2){
1579
      digitalWrite(6, HIGH);
1581  digitalWrite(7, HIGH);
      digitalWrite(8, LOW);
1583  digitalWrite(9, LOW);
      EEPROM.write(addr, 1);
1585  delay(tempo2);

1587  pagina();

1589  statePin5=digitalRead(5);

1591  if(readString.indexOf("parar") >0 || statePin5==HIGH){

1593    desligar();
      break;
1595
      }
1597
      else {
1599
      digitalWrite(6, HIGH);
1601  digitalWrite(7, LOW);
      digitalWrite(8, LOW);
1603  digitalWrite(9, HIGH);
```

```
    EEPROM.write(addr, 4);
1605    delay(tempo2);

1607    }

1609    pagina();

1611    statePin5=digitalRead(5);

1613    if(readString.indexOf("parar") >0 || statePin5==HIGH){

1615      desligar();
        break;
1617
    }
1619
    else {
1621
    digitalWrite(6, LOW);
1623    digitalWrite(7, LOW);
    digitalWrite(8, HIGH);
1625    digitalWrite(9, HIGH);
    EEPROM.write(addr, 3);
1627    delay(tempo2);

1629    }

1631    pagina();

1633    statePin5=digitalRead(5);

1635    if(readString.indexOf("parar") >0 || statePin5==HIGH){

1637      desligar();
        break;
1639
    }
1641
    else {
1643
    digitalWrite(6, LOW);
1645    digitalWrite(7, HIGH);
    digitalWrite(8, HIGH);
```

```
1647      digitalWrite(9, LOW);
          EEPROM.write(addr, 2);
1649      delay(tempo2);

1651      }

1653      pagina();
          statePin5=digitalRead(5);
1655
        }
1657
        if (value == 3){
1659
          digitalWrite(6, LOW);
1661      digitalWrite(7, HIGH);
          digitalWrite(8, HIGH);
1663      digitalWrite(9, LOW);
          EEPROM.write(addr, 2);
1665      delay(tempo2);

1667      pagina();

1669      statePin5=digitalRead(5);

1671      if(readString.indexOf("parar") >0 || statePin5==HIGH){

1673        desligar();
          break;
1675
        }
1677
        else {
1679
          digitalWrite(6, HIGH);
1681      digitalWrite(7, HIGH);
          digitalWrite(8, LOW);
1683      digitalWrite(9, LOW);
          EEPROM.write(addr, 1);
1685      delay(tempo2);

1687      }

1689      pagina();
```

```
1691      statePin5=digitalRead(5);

1693      if(readString.indexOf("parar") >0 || statePin5==HIGH){

1695        desligar();
          break;
1697
        }
1699
        else {
1701
        digitalWrite(6, HIGH);
1703    digitalWrite(7, LOW);
        digitalWrite(8, LOW);
1705    digitalWrite(9, HIGH);
       EEPROM.write(addr, 4);
1707    delay(tempo2);

1709    }

1711    pagina();

1713    statePin5=digitalRead(5);

1715    if(readString.indexOf("parar") >0 || statePin5==HIGH){

1717      desligar();
          break;
1719
        }
1721
        else {
1723
        digitalWrite(6, LOW);
1725    digitalWrite(7, LOW);
        digitalWrite(8, HIGH);
1727    digitalWrite(9, HIGH);
       EEPROM.write(addr, 3);
1729    delay(tempo2);

1731    }
```

```
1733      pagina();
          statePin5=digitalRead(5);
1735
        }
1737
        if (value == 4){
1739
          digitalWrite(6, LOW);
1741      digitalWrite(7, LOW);
          digitalWrite(8, HIGH);
1743      digitalWrite(9, LOW);
          EEPROM.write(addr, 3);
1745      delay(tempo2);

1747      pagina();

1749      statePin5=digitalRead(5);

1751      if(readString.indexOf("parar") >0 || statePin5==HIGH){

1753        desligar();
          break;
1755
        }
1757
        else {
1759
          digitalWrite(6, LOW);
1761      digitalWrite(7, HIGH);
          digitalWrite(8, HIGH);
1763      digitalWrite(9, LOW);
          EEPROM.write(addr, 2);
1765      delay(tempo2);

1767      }

1769      pagina();

1771      statePin5=digitalRead(5);

1773      if(readString.indexOf("parar") >0 || statePin5==HIGH){

1775        desligar();
```

```
      break;
1777
    }
1779
    else {
1781
    digitalWrite(6, HIGH);
1783 digitalWrite(7, HIGH);
    digitalWrite(8, LOW);
1785 digitalWrite(9, LOW);
    EEPROM.write(addr, 1);
1787 delay(tempo2);

1789 }

1791 pagina();

1793 statePin5=digitalRead(5);

1795 if(readString.indexOf("parar") >0 || statePin5==HIGH){

1797   desligar();
      break;
1799
    }
1801
    else {
1803
    digitalWrite(6, HIGH);
1805 digitalWrite(7, LOW);
    digitalWrite(8, LOW);
1807 digitalWrite(9, HIGH);
    EEPROM.write(addr, 4);
1809 delay(tempo2);

1811 }

1813 pagina();
    statePin5=digitalRead(5);
1815
  }
1817
    pagina();
```

```
1819      statePin5=digitalRead(5);

1821    }

1823    desligar();

1825 }

1827 void voltar2(){

1829    desliga = false;
        aux = 0;
1831    statePin5=digitalRead(5);

1833    while(statePin5==LOW){

1835     value = EEPROM.read(addr);

1837     if (value == 1){

1839     digitalWrite(6, LOW);
         digitalWrite(7, HIGH);
1841     digitalWrite(8, HIGH);
         digitalWrite(9, LOW);
1843     EEPROM.write(addr, 2);
         delay(tempo);
1845
         pagina();
1847
         statePin5=digitalRead(5);
1849
         if(readString.indexOf("parar") >0 || statePin5==HIGH){
1851
           desligar();
1853        break;

1855     }

1857     else {

1859     digitalWrite(6, LOW);
         digitalWrite(7, LOW);
1861     digitalWrite(8, HIGH);
```

```
    digitalWrite(9, HIGH);
1863 EEPROM.write(addr, 3);
    delay(tempo2);
1865
    }
1867
    pagina();
1869
    statePin5=digitalRead(5);
1871
    if(readString.indexOf("parar") >0 || statePin5==HIGH){
1873
      desligar();
1875  break;

1877 }

1879 else {

1881 digitalWrite(6, HIGH);
    digitalWrite(7, LOW);
1883 digitalWrite(8, LOW);
    digitalWrite(9, HIGH);
1885 EEPROM.write(addr, 4);
    delay(tempo2);
1887
    }
1889
    pagina();
1891 statePin5=digitalRead(5);

1893 if(readString.indexOf("parar") >0 || statePin5==HIGH){

1895   desligar();
      break;
1897
    }
1899
    else {
1901
    digitalWrite(6, HIGH);
1903 digitalWrite(7, HIGH);
    digitalWrite(8, LOW);
```

```
1905      digitalWrite(9, LOW);
          EEPROM.write(addr, 1);
1907      delay(tempo2);

1909      }

1911      pagina();
          statePin5=digitalRead(5);
1913
        }
1915

1917    if (value == 2){

1919      digitalWrite(6, LOW);
          digitalWrite(7, LOW);
1921      digitalWrite(8, HIGH);
          digitalWrite(9, HIGH);
1923      EEPROM.write(addr, 3);
          delay(tempo2);
1925
          pagina();
1927
          statePin5=digitalRead(5);
1929
          if(readString.indexOf("parar") >0 || statePin5==HIGH){
1931
            desligar();
1933        break;

1935      }

1937      else {

1939      digitalWrite(6, HIGH);
          digitalWrite(7, LOW);
1941      digitalWrite(8, LOW);
          digitalWrite(9, HIGH);
1943      EEPROM.write(addr, 4);
          delay(tempo2);
1945
          }
1947
```

```
    pagina();

    statePin5=digitalRead(5);

    if(readString.indexOf("parar") >0 || statePin5==HIGH){

      desligar();
      break;

    }

    else {

    digitalWrite(6, HIGH);
    digitalWrite(7, HIGH);
    digitalWrite(8, LOW);
    digitalWrite(9, LOW);
    EEPROM.write(addr, 1);
    delay(tempo2);

    }

    pagina();

    statePin5=digitalRead(5);

    if(readString.indexOf("parar") >0 || statePin5==HIGH){

      desligar();
      break;

    }

    else {

    digitalWrite(6, LOW);
    digitalWrite(7, HIGH);
    digitalWrite(8, HIGH);
    digitalWrite(9, LOW);
    EEPROM.write(addr, 2);
    delay(tempo2);

    }
```

```
1991
      pagina();
1993  statePin5=digitalRead(5);

1995  }

1997  if (value == 3){

1999  digitalWrite(6, HIGH);
      digitalWrite(7, LOW);
2001  digitalWrite(8, LOW);
      digitalWrite(9, HIGH);
2003  EEPROM.write(addr, 4);
      delay(tempo2);
2005
      pagina();
2007
      statePin5=digitalRead(5);
2009
      if(readString.indexOf("parar") >0 || statePin5==HIGH){
2011
        desligar();
2013    break;

2015  }

2017  else {

2019  digitalWrite(6, HIGH);
      digitalWrite(7, HIGH);
2021  digitalWrite(8, LOW);
      digitalWrite(9, LOW);
2023  EEPROM.write(addr, 1);
      delay(tempo2);
2025
      }
2027
      pagina();
2029
      statePin5=digitalRead(5);
2031
      if(readString.indexOf("parar") >0 || statePin5==HIGH){
2033
```

```
        desligar();
2035    break;

2037  }

2039  else {

2041  digitalWrite(6, LOW);
      digitalWrite(7, HIGH);
2043  digitalWrite(8, HIGH);
      digitalWrite(9, LOW);
2045  EEPROM.write(addr, 2);
      delay(tempo2);
2047

      }
2049

      pagina();
2051

      statePin5=digitalRead(5);
2053

      if(readString.indexOf("parar") >0 || statePin5==HIGH){
2055

        desligar();
2057    break;

2059  }

2061  else {

2063  digitalWrite(6, LOW);
      digitalWrite(7, LOW);
2065  digitalWrite(8, HIGH);
      digitalWrite(9, HIGH);
2067  EEPROM.write(addr, 3);
      delay(tempo2);
2069

      }
2071

      pagina();
2073  statePin5=digitalRead(5);

2075  }
```

```
2077  if (value == 4){

2079    digitalWrite(6, HIGH);
        digitalWrite(7, HIGH);
2081    digitalWrite(8, LOW);
        digitalWrite(9, LOW);
2083    EEPROM.write(addr, 1);
        delay(tempo2);
2085
        pagina();
2087
        statePin5=digitalRead(5);
2089
        if(readString.indexOf("parar") >0 || statePin5==HIGH){
2091
          desligar();
2093      break;

2095    }

2097    else {

2099    digitalWrite(6, LOW);
        digitalWrite(7, HIGH);
2101    digitalWrite(8, HIGH);
        digitalWrite(9, LOW);
2103    EEPROM.write(addr, 2);
        delay(tempo2);
2105
        }
2107
        pagina();
2109
        statePin5=digitalRead(5);
2111
        if(readString.indexOf("parar") >0 || statePin5==HIGH){
2113
          desligar();
2115      break;

2117    }

2119    else {
```

```
2121      digitalWrite(6, LOW);
          digitalWrite(7, LOW);
2123      digitalWrite(8, HIGH);
          digitalWrite(9, HIGH);
2125      EEPROM.write(addr, 3);
          delay(tempo2);
2127
          }
2129
          pagina();
2131
          statePin5=digitalRead(5);
2133
          if(readString.indexOf("parar") >0 || statePin5==HIGH){
2135
            desligar();
2137        break;

2139      }

2141      else {

2143      digitalWrite(6, HIGH);
          digitalWrite(7, LOW);
2145      digitalWrite(8, LOW);
          digitalWrite(9, HIGH);
2147      EEPROM.write(addr, 4);
          delay(tempo2);
2149
          }
2151
          pagina();
2153      statePin5=digitalRead(5);

2155    }

2157      pagina();
          statePin5=digitalRead(5);
2159


2161     }
```

```
2163     desligar();

2165 }

2167 void direita(){

2169      value = EEPROM.read(addr);

2171      if (value == 1) {

2173      digitalWrite(6, LOW);
          digitalWrite(7, HIGH);
2175      digitalWrite(8, HIGH);
          digitalWrite(9, LOW);
2177      EEPROM.write(addr, 2);
          delay(tempo);
2179
          pagina();
2181
          statePin3=digitalRead(3);
2183      statePin5=digitalRead(5);

2185      pagina();

2187      if(readString.indexOf("parar") >0){

2189        desligar();

2191      }

2193      }

2195      if (value == 2){

2197      digitalWrite(6, LOW);
          digitalWrite(7, LOW);
2199      digitalWrite(8, HIGH);
          digitalWrite(9, HIGH);
2201      EEPROM.write(addr, 3);
          delay(tempo);
2203
          pagina();
2205
```

```
    statePin3=digitalRead(3);
2207    statePin5=digitalRead(5);

2209    pagina();

2211    if(readString.indexOf("parar") >0){

2213      desligar();

2215    }

2217    }

2219    if (value == 3){

2221    digitalWrite(6, HIGH);
        digitalWrite(7, LOW);
2223    digitalWrite(8, LOW);
        digitalWrite(9, HIGH);
2225    EEPROM.write(addr, 4);
        delay(tempo);
2227
        pagina();
2229
        statePin3=digitalRead(3);
2231    statePin5=digitalRead(5);

2233    pagina();

2235    if(readString.indexOf("parar") >0){

2237      desligar();

2239    }

2241    }

2243    if (value == 4){

2245    digitalWrite(6, HIGH);
        digitalWrite(7, HIGH);
2247    digitalWrite(8, LOW);
        digitalWrite(9, LOW);
```

```
2249      EEPROM.write(addr, 1);
          delay(tempo);
2251
          pagina();
2253
          statePin3=digitalRead(3);
2255      statePin5=digitalRead(5);

2257      pagina();

2259      if(readString.indexOf("parar") >0){

2261        desligar();

2263      }

2265      }

2267  }

2269  void esquerda(){

2271      value = EEPROM.read(addr);

2273      if (value == 1) {

2275      digitalWrite(6, HIGH);
          digitalWrite(7, LOW);
2277      digitalWrite(8, LOW);
          digitalWrite(9, HIGH);
2279      EEPROM.write(addr, 4);
          delay(tempo);
2281
          pagina();
2283
          statePin3=digitalRead(3);
2285      statePin5=digitalRead(5);

2287      pagina();

2289      if(readString.indexOf("parar") >0){

2291        desligar();
```

```
2293      }

2295      }

2297      if (value == 2){

2299      digitalWrite(6, HIGH);
          digitalWrite(7, HIGH);
2301      digitalWrite(8, LOW);
          digitalWrite(9, LOW);
2303      EEPROM.write(addr, 1);
          delay(tempo);
2305
          pagina();
2307
          statePin3=digitalRead(3);
2309      statePin5=digitalRead(5);

2311      pagina();

2313      if(readString.indexOf("parar") >0){

2315        desligar();

2317      }

2319      }

2321      if (value == 3){

2323      digitalWrite(6, LOW);
          digitalWrite(7, HIGH);
2325      digitalWrite(8, HIGH);
          digitalWrite(9, LOW);
2327      EEPROM.write(addr, 2);
          delay(tempo);
2329
          pagina();
2331
          statePin3=digitalRead(3);
2333      statePin5=digitalRead(5);
```

```
2335      pagina();

2337      if(readString.indexOf("parar") >0){

2339        desligar();

2341      }

2343      }

2345      if (value == 4){

2347      digitalWrite(6, LOW);
          digitalWrite(7, LOW);
2349      digitalWrite(8, HIGH);
          digitalWrite(9, HIGH);
2351      EEPROM.write(addr, 3);
          delay(tempo);
2353
          pagina();
2355
          statePin3=digitalRead(3);
2357      statePin5=digitalRead(5);

2359      pagina();

2361      if(readString.indexOf("parar") >0){

2363        desligar();

2365      }

2367      }

2369  }
```

postextuais/codigo_arduino.ino