

FACULDADE ADVENTISTA DE HORTOLÂNDIA
CAMPUS HORTOLÂNDIA
SISTEMAS DE INFORMAÇÃO

ALCIOMAR HOLLANDA
SAMUEL SEIJI MIYANO

ACESSIBILIDADE COM ARDUINO

HORTOLÂNDIA
2015

ALCIOMAR HOLLANDA
SAMUEL SEIJI MIYANO

ACESSIBILIDADE COM ARDUINO

**Trabalho de Conclusão de Curso da
Faculdade Adventista de Hortolândia do
curso de Sistemas de Informação, sob
orientação do prof. Ms. Ackley Dias Will.**

HORTOLÂNDIA
2015

Trabalho de Conclusão de Curso da Faculdade Adventista de Hortolândia do curso de Sistemas de Informação apresentado e aprovado em 03 de junho de 2015.

Orientador: Prof. Ms. Ackley Dias Will

Segundo leitor: Prof. Claudio Xavier Gonçalves

Segundo leitor: Prof. Ms. Julio Cesar de Lemos

A Deus que nos deu vida para poder desenvolver este trabalho. Também as nossas famílias, pelo apoio dado ao longo dos anos.

AGRADECIMENTOS

- A Deus por tudo que nos tem dado;
- A Faculdade Adventista de Hortolândia por oferecer as condições ideais para um bom desenvolvimento;
- A todos os professores que dividirem seus conhecimentos conosco;
- As nossas famílias que nos apoiam sempre;

“Quanto mais aumenta nosso conhecimento, mais evidente fica nossa ignorância”.

John F. Kennedy

RESUMO

Este trabalho apresenta por meio de uma pesquisa bibliográfica a utilização de uma tecnologia assistiva, Arduino, na inclusão de deficientes visuais, pois possuem dificuldades na identificação de objetos que estão próximo dele. O contexto histórico da deficiência em especial da visual, apontando as dificuldades que enfrentam e o desenho universal como proposta de padrão para as tecnologias assistivas. Uma introdução a eletrônica como base para utilização das ferramentas requeridas para o projeto. Apresenta o Arduino, sua história, modelos, ambiente de desenvolvimento e linguagem de programação. Foi usado a placa Arduino, sensores ultrassônicos e motores no projeto NavBlind, para que o usuário conseguisse identificar objetos próximos. Por fim mostra os resultados positivos e negativos do projeto Navblind e uma proposta de desenvolvimento futuro.

Palavras-chave: Arduino; NavBlind; Tecnologia assistiva; Deficientes visuais; Sensores ultrassônicos.

ABSTRACT

This work presents a literature search through the utilization of an assistive technology, Arduino in the inclusion of visually impaired because they have difficulties in identifying objects that are close to him. The historical context of disability especially the visual, pointing out the difficulties they face and universal design as standard proposal for assistive technologies. An introduction to electronics as a basis for using the required tools for the project. Presents the Arduino, its history, models, development environment and programming language. It was used the Arduino board, ultrasonic sensors and motors in NavBlind project, for the user could identify nearby objects. Finally shows the positive and negative results of Navblind project and a proposal for future development.

Keywords: Arduino; NavBlind; Assistive technology; Visually impaired; Ultrasonic sensors.

LISTA DE ILUSTRAÇÕES

Figura 1 - Componentes de um microcontrolador	25
Figura 2 - Montagens mais usuais de transdutores piezoelétricos: (a) 40 kHz; (b) 220 kHz.	29
Figura 3 – Exemplo de emissão e recepção de ondas acústicas, produzindo um sinal u.	30
Figura 4 – a) Placa de disco metálico; b) Placa do capacitor	30
Figura 5 – Descrição do HC-RS04	32
Figura 6 – Diagrama de pinos do L293D.....	33
Figura 7 – Diagrama de circuito para controlar motor com L293D IC	34
Figura 8 – Imagem do primeiro protótipo feito em 2005.....	37
Figura 9 – Pinos do Arduino Uno e Layout da placa.	41
Figura 10 – Especificações de um Arduino Uno.....	42
Figura 11 – Informações de alguns pinos do Arduino.	42
Figura 12 – Arduino Ethernet Rev.3	44
Figura 13 – Layout do Arduino Mega	46
Figura 14 – Tela inicial do Arduino IDE	47
Figura 15 - Menu File	48
Figura 16 – Menu Edit	48
Figura 17 – Menu Sketch	49
Figura 18 – Menu Tools	49
Figura 19 – Menu Help	50
Figura 20 - Projeto HALO	66
Figura 21 - Hardware do Tacit.....	68
Figura 22 - Protoboard	69
Figura 23 - HC-SR04.....	69
Figura 24 - L293D	69
Figura 25 - Arduino Uno R3.....	70
Figura 26 - Led 10 mm	70
Figura 27 - Motor dc	70
Figura 28 - Bateria 9v	71
Figura 29 - Resistor 10 k	71
Figura 30 - Momentary Pushbutton	71

Figura 31 - Buzzer DC 5v	72
Figura 32 - Fio Jumper	72
Figura 33 - Declaração de Variaveis	74
Figura 34 - Função setup()	75
Figura 35 - Função loop()	75
Figura 36 - Função ultrasson()	77
Figura 37 - Modelo do protótipo NavBland	78

LISTA DE ABREVIATURAS, SIGLAS OU TERMOS OPERACIONAIS

OMS – Organização Mundial da Saúde

IBGE – Instituto Brasileiro de Geografia e Estatística

ddp – diferença de potencial

A – Ampere

P – Potencia

V – Volts

Hz – Hertz

LED – Light Emitting Diode (Diodo emissor de Luz)

CPU – Central Processing Unit (Unidade Central de Processamento)

I/O – Input and Output (Entrada e Saída)

IC – Integrated Circuit (Circuito Integrado)

DC – Direct Current (Corrente Continua)

VCC – Voltagem da Corrente Continua

IDE – Integrated Development Environment (Ambiente Integrado de Desenvolvimento)

USB – Universal Serial Bus (Serial Universal)

PWM – Pulse Width Modulation (Modulação por largura de pulso)

SUMÁRIO

1	INTRODUÇÃO	14
2	ACESSIBILIDADE	16
2.1	Contexto histórico	16
2.2	Tipos de deficiência e definições	17
2.3	A deficiência visual.....	18
2.3.1	Conceituação	18
2.3.2	Classificação.....	19
2.3.3	Causas.....	19
2.3.4	Solução	20
2.4	Tecnologias Assistivas	20
2.4.1	Desenho Universal	21
2.4.2	Arduino como recurso	22
3	ELETRÔNICA.....	23
3.1	Elétron e os componentes eletrônicos.....	23
3.2	Microcontroladores	25
3.2.1	História	27
3.3	Sensor Ultra-Sônico.....	28
3.4	Deteção de objetos com HC-SR04.....	31
3.5	O circuito integrado L293D	33
4	ARDUINO	36
4.1	Breve Histórico	36
4.2	<i>Hardware</i>	40
4.2.1	Arduino Uno.....	40
4.2.2	Arduino <i>Ethernet</i>	44
4.2.3	Arduino Mega.....	45
4.3	Arduino IDE	46
4.4	Arduino C	50
4.4.1	Estruturas.....	51
4.4.2	Variáveis.....	54
4.4.3	Operadores	57
4.4.4	Estruturas de controle.....	60
4.4.5	<i>Arrays</i>	62
4.4.6	Funções.....	63
4.5	Vantagens e desvantagens	65

5 PROJETO NAVBLIND.....	66
5.1 Componentes usados.....	69
5.2 Analise dos componentes.....	72
5.3 Analise do código	73
5.4 Modelo	77
6 CONSIDERAÇÕES FINAIS	79
7 REFERÊNCIAS	81

1 INTRODUÇÃO

Ao longo dos séculos, o homem sempre desenvolveu ferramentas para atender suas necessidades, desde a roda nos primórdios da raça humana até os computadores inventados nos últimos anos. No entanto tais tecnologias, na maioria das vezes são voltadas para a maioria da população que não possui deficiências físicas. Embora por muito tempo os deficientes físicos tenham sido esquecidos e até mesmo considerados como escória da sociedade, como diz Sonza et al. (2013), recentemente este paradigma tem sido repensado e alterado, de forma que toda pessoa possui o direito de ser beneficiada pelas inovações, inclusive, aquelas portadoras de uma deficiência qualquer.

De modo especial, os deficientes visuais têm enfrentado muitas dificuldades quanto a isso, porquanto exista ferramentas que auxiliam em determinadas áreas da vida como a linguagem em braile na escrita e leitura, os livros em áudio para o estudo e a bengala para a locomoção, ainda é muito pouco, tornando-se necessário mais ferramentas que permitam ao deficiente total independência. Mesmo com a bengala ou um cão guia de certa forma as dificuldades na locomoção permanecem e não há total independência, pois há lugares que não é permitido a entrada de animais. Como as tecnologias atuais poderiam auxiliar na acessibilidade e independência de tais pessoas? Qual a melhor tecnologia para esse fim?

Com a evolução tecnológica na área da computação, em especial da arquitetura de *hardware*, cada vez mais os aparelhos eletrônicos foram sendo compactados, do tamanho de uma sala à um *chip* que cabe na palma da mão. Então surgiram os microcontroladores a partir da segunda metade do século XX, com eles é possível manipular sensores, motores, entre outros dispositivos eletrônicos.

Dentre as diversas opções tecnológicas, existe o Arduino, que é um *hardware open source* para desenvolvimento de projetos complexos de automação e robóticos, mas ao mesmo tempo, simples de serem desenvolvidos, a ponto de até pessoas leigas em eletrônica construir os projetos. Seria possível utilizar as poderosas e versáteis ferramentas do Arduino adaptando-as aos deficientes visuais?

Para este trabalho será implementado um projeto de tecnologia assistiva que utiliza sensores ultrassônicos que identificarão e alertarão a distância entre os objetos e a proximidade destes com os usuários.

2 ACESSIBILIDADE

2.1 Contexto histórico

A medida que os anos passam, questões culturais, sociais e econômicas alteram o modo que a sociedade trata as pessoas com necessidades especiais. De acordo com Sonza et al. (2013), na antiguidade pessoas com limitações eram consideradas inúteis, enquanto, aqueles que estavam envolvidos em trabalhos, tais quais a agropecuária, o comércio ou artesanato eram vistos com bons olhos pela sociedade. Tanto que em muitas sociedades, bebês deficientes eram mortos, pois não seriam úteis à comunidade. Por outro lado, os hindus criam que os cegos eram abençoados com outros sentidos mais aguçados.

A influência do Cristianismo é sentida na Idade Média, quando é adotado o pensamento de que todos são criaturas de Deus, portanto não podem ser exterminadas pelo simples fato de serem diferentes, por serem deficientes físicos. Todavia, tais pessoas ainda eram ignoradas e deixadas de lado pela sociedade a mercê da caridade de outrem. No ápice do domínio da Igreja Católica, a Inquisição perseguiu os deficientes com o pretexto de estarem assim, por serem “possuídos pelo demônio”. (SONZA et al., 2013)

O advento da Idade Moderna, Renascimento e Humanismo trouxe uma nova forma de pensamento, a razão acima das crendices. Deste modo a deficiência não era mais vista como algo sobrenatural, mas uma consequência de causas naturais. Sendo inventados equipamentos para facilitar a vida de pessoas com deficiência:

Com as Revoluções Francesa e Industrial, a partir do século XVIII, iniciaram-se esforços para que as pessoas com deficiência pudessem trabalhar e, com isso, surgiram vários inventos, como a cadeira de rodas, bengalas, muletas, próteses, etc. (SONZA et al., 2013, p. 24)

Por séculos o Brasil seguiu o modelo de institucionalização, que é quando se reuniam as pessoas com deficiência em comunidades separadas da sociedade em escolas especiais. Assim permaneceu até meados da década de 50, neste período houve um forte contraponto à institucionalização por meio da Declaração Universal dos Direitos Humanos com seus ideais de igualdade, fraternidade e liberdade.

Na década de 60, surgiu em todo ocidente a ideia de normalização, isto é, “normalizar tinha como pressuposto modificar a pessoa com deficiência e ajustá-la para que pudesse ser introduzida na sociedade”. (SONZA et al., 2013, p.25) Por meio das áreas biológicas, tentava-se fazer das pessoas com deficiência, “normais”. No entanto a discussão do conceito de normal é complexa e envolve aspectos físicos, sociais e ideológicos. A partir da normalização surgiu a ideia de integração, onde o indivíduo com deficiência deve se adaptar ao resto da sociedade considerada normal.

Esta ideologia também foi criticada e por volta dos anos 80 surgiram documentos, como por exemplo, Programa Mundial de Ação Relativo às Pessoas com Deficiência, que defende o conceito de inclusão que é contrário à integração conforme explica Sassaki (2005, p.20):

[...] a inclusão consiste em adequar os sistemas sociais gerais da sociedade de tal modo que sejam eliminados os fatores que excluíaam certas pessoas do seu seio e mantinham afastadas aquelas que foram excluídas [...]

Portanto é a sociedade que deve se adaptar, de tal forma a facilitar o convívio das pessoas com deficiência. Por fim, Sonza et al. (2013, p.27), diz que:

Atualmente, antigos e novos paradigmas estão presentes na sociedade, já que a evolução não se deu de forma linear em todas as sociedades e culturas. No entanto, é visível que a sociedade está passando por um processo irreversível, tornando-se cada vez mais inclusiva.

2.2 Tipos de deficiência e definições

Existem duas categorias de pessoas no âmbito da inclusão, as pessoas com deficiência e as pessoas com mobilidade reduzida. Uma pessoa com deficiência é aquela que possui algum tipo de empecilho de longo prazo que o pode impedir de realizar qualquer atividade na sociedade, de maneira plena e igualitária comparada as outras pessoas. (CAPACITAÇÃO EM ACESSIBILIDADE, [s.d])

Já a pessoa com mobilidade reduzida, é aquela que não pode ser taxado como pessoa com deficiência, porém possui, temporariamente ou não, uma diminuição em sua mobilidade física como por exemplo, idosos ou gestantes.

A acessibilidade define que tais pessoas devem ter tratamento igual as demais pessoas em qualquer circunstância. Para tanto, medidas devem ser tomadas a fim de que haja condições para as mesmas desfrutarem dos serviços de transporte, meios de comunicação e sistemas variados com segurança e de preferência de forma autônoma. (BRASIL, 2004)

Existem diversas deficiências que podem atingir o homem, o artigo 5º do Decreto nº 5.296/04 (BRASIL, 2004) assim as classifica:

Deficiência Física: quando uma ou mais partes do corpo humano sofre alteração, de modo que a mesma fica incapacitada de realizar sua função física. Por exemplo amputação do membro, paralisia cerebral, nanismo, paraparesia e paraplegia.

Deficiência Auditiva: perda, parcial ou total, da audição de 41 decibéis (dB) ou mais em ambos os ouvidos, medido em audiogramas nas frequências 500Hz, 1.000Hz, 2.000Hz e 3.000Hz. Importante destacar que se for perda em apenas um dos ouvidos não é caracterizado como deficiência.

Deficiência Intelectual: geralmente aparece antes dos 18 anos, é a capacidade inferior à média em duas ou mais áreas das 8 habilidades adaptativas, que são: autonomia, comunicação, cuidado pessoal, habilidades sociais, habilidades acadêmicas, lazer, saúde e segurança, trabalho.

Deficiência Visual: cegueira ou quando o campo de visão, somados ambos os olhos, é menor que 60º.

Deficiências Múltiplas: duas ou mais deficiências.

2.3 A deficiência visual

2.3.1 Conceituação

O artigo 5º do Decreto nº 5.296/04, define deficiência visual como:

Cegueira, na qual a acuidade visual é igual ou menor que 0,05 no melhor olho, com a melhor correção óptica; a baixa visão, que significa acuidade visual entre 0,3 e 0,05 no melhor olho, com a melhor correção óptica; os casos nos quais a somatória da medida do campo visual em ambos os olhos for igual ou menor que 60º; ou a ocorrência simultânea de quaisquer das condições anteriores. (BRASIL, 2004)

A deficiência visual é uma condição de diminuição da visão, hereditária ou não, e que não possui reversão mesmo que realizado tratamentos, cirurgias ou uso de óculos. Esta diminuição pode ser baixa ou total, isto é, a cegueira. (ENTREAMIGOS, [s.d.])

Sonza et al. (2013), exemplifica a baixa visão dizendo que o indivíduo que a possui vê algo a 6 metros de distância da mesma forma que alguém com visão normal veria algo a 60 metros.

2.3.2 Classificação

De acordo com dados da OMS, cerca de 1% da população do mundo possui deficiência visual em qualquer um dos seus níveis.

Sendo que o último censo do IBGE (2010) realizado no Brasil apresenta como resultado a existência de 45.606.048 milhões de pessoas que possuem pelo menos um tipo de deficiência, sendo que 18% declara ter deficiência visual.

2.3.3 Causas

Dividem-se as causas em congênicas e adquiridas. As congênicas são: “amaurose congênita de Leber, malformações oculares, glaucoma congênito, catarata congênita.” (ENTREAMIGOS, [s.d.])

Já as adquiridas estão relacionadas a traumas oculares, alterações na retina devido a diabetes ou hipertensão arterial, entre outros. Há também alguns fatores de risco, como histórico familiar, o não uso de óculos especiais em atividades que o requerem, catarata e histórico pessoal de doenças como diabetes e esclerose múltipla. (ENTREAMIGOS, [s.d.])

2.3.4 Solução

A respeito das dificuldades encontradas pelas pessoas com deficiência visual, Sonza et al. (2013, p.80) dá o seu parecer:

Acreditava-se até pouco tempo que pessoas deficientes visuais não conseguiriam ter uma vida plena e autônoma, mas há alguns anos vêm-se descobrindo tecnologias que auxiliam essas pessoas a desempenharem as funções diárias.

Para tanto é preciso começar um processo de orientação espacial o quanto antes para que a vida social da pessoa com deficiência não seja atrapalhada. Nisso a família tem papel essencial em incentivar o deficiente de modo que ele sinta que é uma pessoa tão capaz quanto as outras. (SONZA et al., 2013)

2.4 Tecnologias Assistivas

Tecnologia assistiva é um termo novo, no entanto, a prática do mesmo já existe a muito tempo, pois qualquer recurso improvisado de forma a auxiliar alguma deficiência é considerado como tal, por exemplo um pedaço de pau usado como bengala. (FILHO, 2009)

Cook (1995) explica que o termo tecnologia assistiva é usado para se referir a uma larga gama de recursos, serviços e estratégias usadas para melhorar os problemas enfrentados por pessoas com deficiência.

Os recursos envolvem qualquer item, ferramenta ou sistema, sendo eles comerciais, customizados ou modificados que tenham como função exatamente auxiliar os deficientes. Os serviços são os que auxiliam estes indivíduos na aquisição e utilização de um recurso.

O conceito de desenho universal é importante na área de inclusão e tecnologia assistiva, pois define princípios que norteiam a produção destes recursos.

2.4.1 Desenho Universal

O desenho universal vem da necessidade de proporcionar as pessoas com deficiência, meios que facilitem a independência das mesmas. Este conceito surgiu na Universidade Estadual da Carolina do Norte nos Estados Unidos. Onde os arquitetos têm como objetivo beneficiar o maior número de pessoas através de medidas que regulamentem a maneira como ambientes, serviços, produtos e tecnologias são desenvolvidas.

Sete princípios regem o desenho universal apresentado pelo Centro de Desenho Universal da Universidade da Carolina do Norte:

1 – Uso equiparável: ser útil para qualquer pessoa independente de capacidade e habilidade de forma a evitar sempre excluir qualquer grupo, fornecendo a todos segurança e privacidade sem diferenças. Um exemplo são as portas com sensores que captam a aproximação de qualquer indivíduo, não importando altura ou peso e sem preconceitos.

2 – Uso flexível: o objeto em questão precisa ter opções diferentes de utilização, deve proporcionar alternativas, como por exemplo, considerar tanto destros como canhotos. Um caso disso são computadores que tenham um leitor de tela, ou mouses para destros e canhotos.

3 – Simples e intuitivo: preza por ser algo simples e de fácil entendimento, de modo que todos possam usar sem a necessidade de um nível aprofundado de conhecimento específico. As informações devem estar de tal forma a levar em consideração todos os níveis de escolaridade. Por exemplo, placas que identifiquem sanitários masculinos e femininos, informando que pessoas com deficiência física também podem usa-los.

4 – Informação perceptível: apresentar as informações da forma mais legível possível, para todas as pessoas por meio de sons, braile e placas.

5 – Tolerância ao erro: diminuir riscos de acidentes involuntários, deixando aquilo que pode representar um risco à segurança escondido ou bem protegido. Alertando para o perigo iminente.

6 – Pouca exigência de esforço físico: deve ser confortável na utilização de modo que possa ser usado por um longo tempo sem muita fadiga, sem muitas repetições mantendo o corpo relaxado. Por exemplo, uma maçaneta que possa ser acionada usando o cotovelo no caso de a pessoa não ter mão.

7 – Tamanho e espaço para acesso e uso: tudo deve estar ao alcance da pessoa, se sentada ou em pé. Levando em conta o espaçamento e distância. Por exemplo, os banheiros especiais para quem usa cadeira de rodas. (SONZA et al., 2013, p.37-42)

2.4.2 Arduino como recurso

Portanto, um exemplo de recurso que pode ser utilizado como tecnologia assistiva é o Arduino, um *hardware* livre que pode ser modificado por qualquer pessoa a um custo relativamente baixo. A proposta deste trabalho é desenvolver um projeto utilizando a plataforma Arduino, que dentro dos princípios do desenho universal, auxilie a locomoção das pessoas com deficiências visuais.

3 ELETRÔNICA

Muito do desenvolvimento das tecnologias de computação estão diretamente ligadas à evolução da eletrônica. Ela permite que através da movimentação da corrente elétrica, componentes consigam realizar cálculos matemáticos e até gerar sons e imagens. (FILHO, 2012)

Antes de conhecer o Arduino, é necessário entender os conceitos básicos por trás de seu funcionamento, como corrente elétrica, microcontroladores, entre outros componentes eletrônicos que serão usados em conjunto ao Arduino no desenvolvimento do projeto.

3.1 Elétron e os componentes eletrônicos

Cada átomo é composto por: prótons e nêutrons em seu núcleo, e elétrons orbitando em sua última camada. Porém, alguns materiais possuem elétrons que não ficam firmemente ligados a esta última camada e podem se desprender e se unir a outro átomo, estes materiais são conhecidos como condutores elétricos, por exemplo, os metais. (FILHO, 2012)

Quando o número de elétrons (carga negativa) e prótons (carga positiva) se equivalem, o átomo está eletricamente equilibrado. Fazendo com que os elétrons não saiam de sua orbita. Filho (2012) define corrente elétrica da seguinte maneira:

Num corpo condutor, os elétrons ficam num movimento desordenado, um átomo perde um elétron e em seguida ganha um elétron de outro átomo sem ordem alguma. Mas quando isso é ordenado de forma que os elétrons tenham um único caminho a percorrer e de forma ordenada isso é denominado corrente elétrica. (FILHO, 2012)

Portanto só existe corrente elétrica quando há um desequilíbrio entre dois pontos, onde em dois átomos, um possui elétrons em excesso e outro tem em falta, gerando uma tentativa de equilíbrio. Essas tentativas continuarão enquanto houver condições até que o equilíbrio seja alcançado. Este processo denomina-se diferença de potencial, e a unidade que é utilizada no cálculo do ddp é o Volt.

O Volt mede a ddp entre dois pontos com potenciais elétricos, ou seja, a força com que os elétrons se pressionam buscando o equilíbrio entre os pontos. Outras unidades de medidas importantes para se entender como funcionam os equipamentos eletrônicos são o Ampere, Watt e Hertz. (FILHO, 2012)

A formula matemática para calcular a corrente (I) é: $I = V / R$. Divide-se a tensão pela resistência, e o resultado se dá em amperes. O ampere está relacionado a corrente elétrica, de forma que calcula a quantidade de elétrons que se movimentam formando a mesma.

Já o watt mede a potência elétrica e térmica. Sua formula matemática é: $P = V * I$. multiplica-se o volt pelo ampere. O hertz mede a oscilação das ondas eletromagnéticas, calcula-se em ciclos por segundo. Um processador de 1GHz, por exemplo, executa um bilhão de ciclos por segundos. (FILHO, 2012)

Determinados materiais se comportam de maneiras variadas em relação a corrente elétrica, a partir deles foram desenvolvidos diversos componentes eletrônicos com finalidades diversas, que são usados na computação:

Resistor: sua principal característica é resistir a passagem da corrente elétrica a fim de gerar calor, pois os elétrons terão dificuldades em passar entre os átomos e precisaram de mais força para atravessar os mesmos, um exemplo é o ferro de passar roupa. Outra finalidade é reduzir a corrente elétrica em algum ponto do circuito, causando uma queda na tensão.

Capacitor: é um componente que armazena a carga elétrica, é formado por placas separadas por um material isolante. Encontra-se em circuitos eletrônicos, memória RAM dinâmica e em fonte de alimentação de computadores. Na memória RAM por exemplo os capacitores recebem a carga elétrica de modo a representar os números binários.

Diodo: faz com que a corrente elétrica passe por apenas uma direção podendo assim converter uma corrente alternada em continua. O LED é um tipo de diodo, que emite luz quando recebe a corrente elétrica.

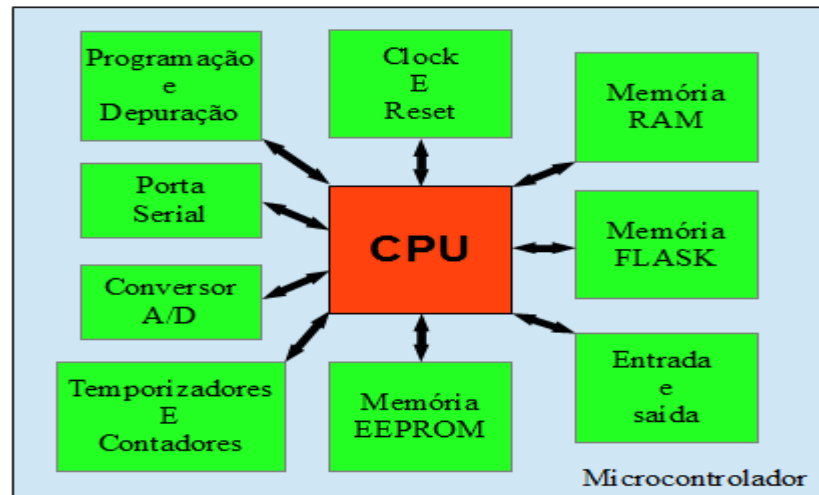
Transistor: é o principal componente eletrônico, foi ele que substituiu a válvula termiônica. Grande parte do avanço tecnológico dos últimos anos estão intimamente ligados à sua invenção. O transistor é praticamente um diodo, com a diferença que pode controlar a passagem da corrente através dos de três terminais: base, emissor e coletor. Se houver alguma tensão na base, ele permite que a corrente passe do emissor ao coletor, se não houver, ele não conduz.

Fusível: é o mais simples de todos, tem como função proteger o equipamento contra um dano permanente, caso haja uma sobrecarga elétrica. O fusível impede que o componente alcance temperaturas mais elevadas do que as que ele suporta, impedindo a passagem da corrente. Quando queimados nunca devem ser trocados por fusíveis de outra capacidade, especialmente se a capacidade for maior. Neste caso o fusível demoraria demais para agir, resultando na queima dos componentes que buscava proteger. (FILHO, 2012)

3.2 Microcontroladores

Kerschbaumer (2013), explica que os microcontroladores são comutadores de circuito integrado, de um único *chip*, que possuem todos os componentes requeridos para seu funcionamento, necessitando apenas de uma fonte de alimentação para funcionar. São diferentes de outros sistemas, por possuírem seus periféricos dentro da própria placa, sendo formada por componentes como a CPU (Unidade Central de Processamento), memória de dados, memória de programa e circuito do *clock*. A Figura 1 expõe um pouco mais os pontos característicos de um microcontrolador.

Figura 1 - Componentes de um microcontrolador



Fonte: KERSCHBAUMER, 2013.

Zanco (2005) observou que o microcontrolador tem uma inteligência programável sendo também um pequeno eletrônico, capaz de estar em quase todos os locais pelo seu tamanho e de fácil manutenção ou troca. Ele é capaz de processar diversas funções que outros equipamentos iriam necessitar de mais componentes, em uma única placa existem todos os acessórios para seu funcionamento. Com isso aprender a programar microcontroladores possibilita utilizar uma ferramenta que tem poucos circuitos em um único componente. Atualmente existem diversos microcontroladores no mercado.

Os microcontroladores estão escondidos dentro de um número surpreendente de produtos nos dias de hoje. Se o seu forno de micro-ondas tem uma tela de LED ou de LCD e um teclado, nele contém um microcontrolador. [...] Qualquer dispositivo que tenha um controle remoto quase certamente contém um microcontrolador [...]. Basicamente, qualquer produto ou dispositivo que interage com o usuário tem um microcontrolador escondido dentro. (BRAIN, [s.d.], tradução livre).¹

Os microcontroladores que encontramos no mercado vem com diferentes embalagens, velocidades de processamento e memória conforme Kerschbaumer (2013). Zanco (2005) completa dizendo que possuem arquiteturas e modelos diferentes por ter muitos fabricantes e para atender as necessidades de projetos, mudando alguns periféricos, e linhas de entrada e saída (*I/O - Input/Output*).

¹ Microcontrollers are hidden inside a surprising number of products these days. If your microwave oven has an LED or LCD screen and a keypad, it contains a microcontroller. [...] Any device that has a remote control almost certainly contains a microcontroller [...]. Basically, any product or device that interacts with its user has a microcontroller buried inside.

Kerschbaumer (2013) continua dizendo que uma das facilidades de se usar microcontroladores é a possibilidade de se alterar o *software* ou seja atualizar, já em outro circuitos como digitais tradicionais e circuitos analógicos não é praticável essa alteração. Kerschbaumer (2013) enfatiza que:

Os microcontroladores são muitos utilizados pela sua versatilidade, pois seu comportamento depende principalmente do *software* que nele é gravado. Assim um mesmo microcontrolador pode ser utilizado para uma infinidade de aplicações bastando apenas mudar o seu *software*. (KERSCHBAUMER, 2013, p.11)

3.2.1 História

A facilidade que temos hoje com os microcontroladores, só foi possível, de acordo com Moraes (2010), a partir da criação de circuitos integrados. Com isso foi possível guardar diversos transistores em um único chip colaborando na construção.

Mas em 1969 os funcionários da INTEL que estavam sendo dirigidos por Marcian Hoff, foram informados que precisavam desenvolver uma calculadora que iria utilizar poucos circuitos integrados. Como possuía experiência em projetos de computação, Hoff explicou de um jeito diferente e fundamental que a máquina teria um programa armazenado na memória que possibilita mudar sua funcionalidade, e deveria ser diferente na criação de calculadoras. Hoff e o engenheiro Federico Faggin desenvolveram o primeiro microprocessador para a Intel. (CORTELLETTI, 2006)

Ex-aluno de engenharia elétrica da Stanford, Marcian "Ted" Hoff, se tornou funcionário número 12 da Intel em 1969 e em dois anos, junto com Federico Faggin e Stan Mazor, ele já tinha inventado o principal produto da Intel: o microprocessador. O sucesso não foi por acaso. Por mais de quatro décadas, a relação Stanford-Intel rendeu não só importantes avanços em semicondutores, mas também centenas de bem-sucedidas, e às vezes lendárias, carreiras de engenharia. (ENGINEERING, 2009, tradução livre).²

Desde então os microprocessadores sempre estão passando por evoluções, mas existiam necessidades no processamento dos sistemas embarcados que não

² Stanford electrical engineering alumnus Marcian "Ted" Hoff, became Intel employee number 12 back in 1969 and within two years, along with Federico Faggin and Stan Mazor, he had invented Intel's flagship product: the microprocessor. The success was no fluke. For more than four decades, the Stanford-Intel relationship has yielded not only important advances in semiconductors but also hundreds of successful, and sometimes legendary, engineering careers.

eram supridas, alguns exemplos que não enfrentam este problema podem ser os televisores, celulares e aparelhos de som. (BRAGA, 2010)

Segundo Zanco (2005), foi lançado em 1974 no Texas o TMS 1000 de 4 bits que foi o primeiro microcontrolador capaz de suprir as necessidades de entrada e saída num único chip com memória RAM e ROM. Com o passar do tempo surge outro que possui uma memória de informações internas (RAM) e uma outra memória (ROM) de programa externa, isso aconteceu quando a Intel lançou em 1977 o microcontrolador.

Neto et al. (2012) explica que em 1980 surge a tecnologia HMOS que possui 40 pinos, adquirindo vários periféricos e uma memória de 128 bytes para dados e outra de 4k para programa podendo ser utilizada sem o chip externo. Com o passar do tempo diversas empresas entraram no mercado de desenvolvimento de microcontroladores sendo elas, Motorola®, Microchip® e a ATMEL®.

3.3 Sensor Ultra-Sônico

Sensores são ferramentas que possibilitam a identificação de objetos com ajuda de tradutores, Bastos (2007) continua dizendo que existe vários tipos de sensores que podem ser apalpadores, interruptores e provadores. Esses citados são conhecidos como de contato. Outros modelos que podemos citar como não-contato são os de ondas sonoras, luz infravermelho e raio-x, podendo ser divididos em duas categorias.

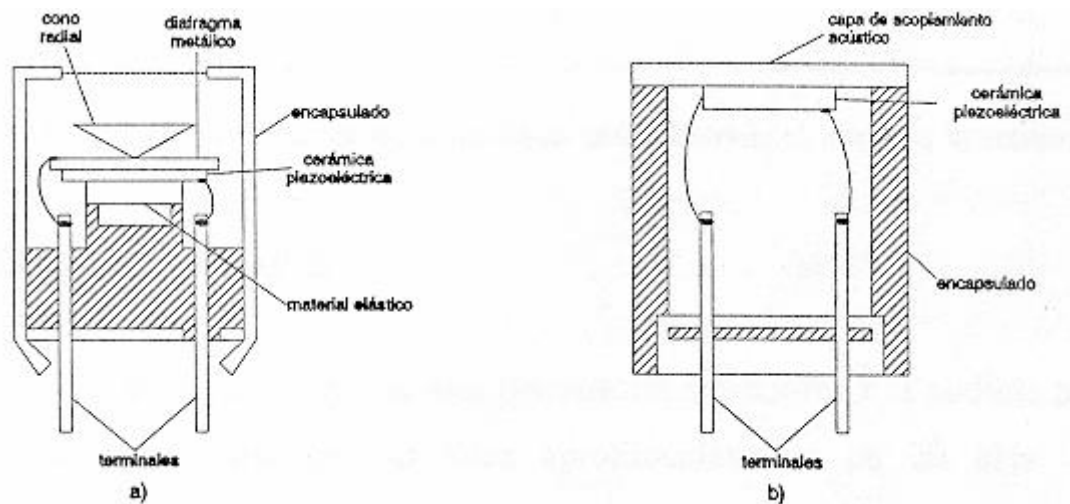
Utilizado para medir velocidade, medir posição sendo possível também a aceleração de um robô pelas rodas ou juntas que ele tem, esses sensores são conhecidos como sensores internos. Os sensores externos são usados para proteger, podendo ser visuais ou não visuais para identificar objetos e até mesmo o ambiente (BASTOS, 2007).

Marcatto ([s.d.]) define que o sensor é conhecido na eletrônica como um circuito eletrônico capaz de fazer uma análise e identificar o ambiente podendo fazer um reconhecimento da temperatura, que é algo simples, até algo que seja mais complexo como identificar a distância de um carro a um objeto ou uma rotação de um

motor. O sensor é conhecido também como um objeto que é capaz de transformar energia em outra sendo possível classificar como um tradutor.

Bastos (2007) afirma que o sensor ultrassônico não é totalmente independente para identificar objetos, que possuem duas armações básicas. Os tradutores piezelétricos podem utilizar o material conhecido como cerâmica piezelétrica como um componente que o propicia ser um tradutor, sendo também possível eletrostáticos e os piezelétricos serem utilizados como sensores. A Figura 2 mostra um exemplo de como funciona os sensores mencionados, sendo uma das mais utilizadas por piezoelétricos: (a) 40 kHz; (b) 220 kHz [3].

Figura 2 - Montagens mais usuais de transdutores piezoelétricos: (a) 40 kHz; (b) 220 kHz.



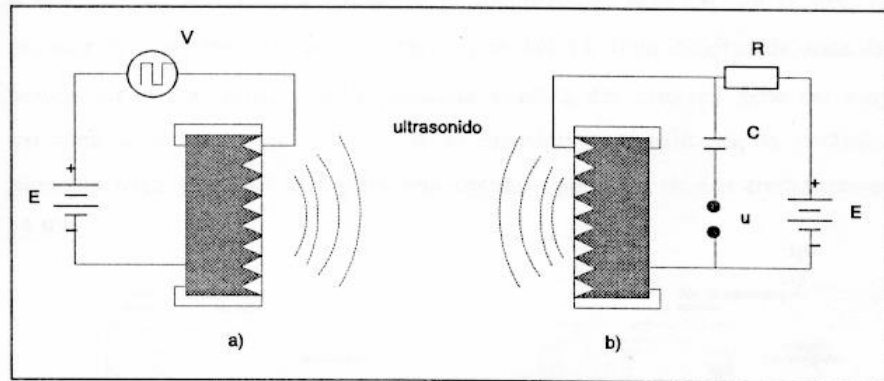
Fonte: BASTOS, 2014.

Os tradutores ultrassônicos piezelétricos apresentam uma armadura característica que é apresentada de 40KHz e a outra de 220KHz. Para os tradutores que se tem o mesmo diâmetro, uma frequência mais alta, requer uma menor passagem acústica. Para ter-se uma ampliação da sensibilidade (40KHz) normalmente é utilizado um cone radial. Já a cama de resina se emprega para aperfeiçoamento acústico do ar (220KHz) auxiliar e defende o equipamento contra sujeiras e outros riscos que podem danificar o tradutor.

Para emissão de ondas acústicas, aplica-se uma tensão alternada às duas faces opostas da cerâmica. Se a frequência da tensão aplicada coincidir com a frequência de vibração da cerâmica, ela entra em ressonância e as vibrações alcançam um máximo. As vibrações transmitem-se ao meio, produzindo uma onda acústica. O fenômeno

é recíproco de tal forma que na recepção, a pressão acústica faz aparecer cargas elétricas na cerâmica, produzindo-se assim um sinal elétrico u . (BASTOS, 2007, p. 29)

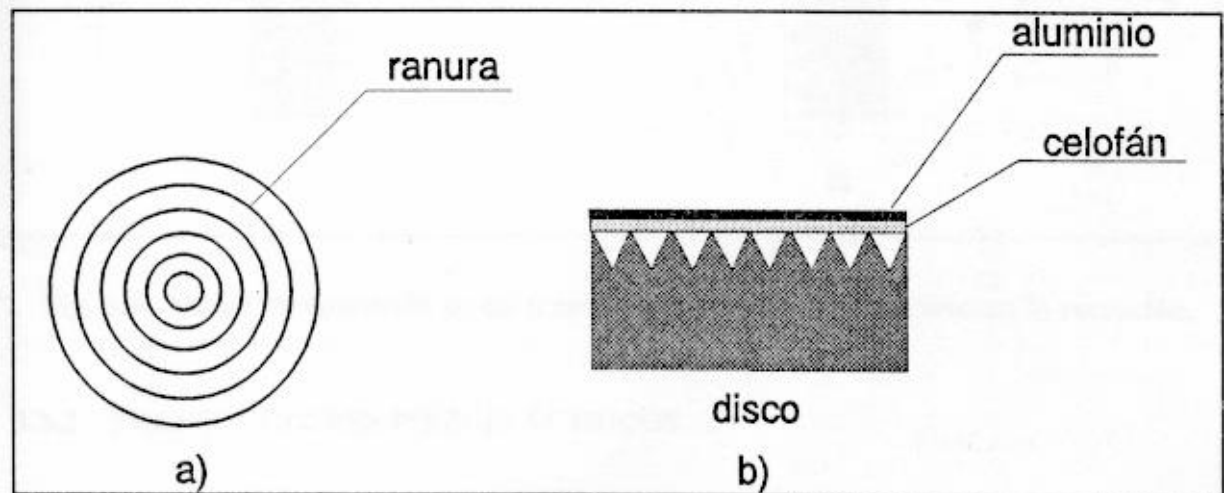
Figura 3 – Exemplo de emissão e recepção de ondas acústicas, produzindo um sinal u .



Fonte: BASTOS, 2014.

Para identificar objetos que estão próximos é utilizado um material que auxilia conter a acústica e suavizar oscilações sendo posto na frente do tradutor. O eletrostático é outro tipo de tradutor ultrassônico que tem os seguintes componentes segundo Bastos (2007), uma das faces são placas metalizadas possuindo também um capacitor de placas paralelas, outra face é constituída de uma matéria isolante. Já outras placas possuem uma forma de anéis concêntricos graças as cavidades feitas na superfície dessa placa um disco metálico. Há um material isolante que separa as placas dos capacitores, um exemplo é “Celofane” podendo também ser separadas pelo ar que estar preso dentro das rachaduras. Na figura 4 está um exemplo do que foi dito.

Figura 4 – a) Placa de disco metálico; b) Placa do capacitor



Fonte: BASTOS, 2014.

3.4 Detecção de objetos com HC-SR04

Uma ferramenta muito eficiente para identificar objetos que estão aos arredores. Ultrassom tem um princípio básico: transmitir um som e esperar por um retorno, definindo o nome como eco, com o tempo de resposta que é iniciado quando emitido um som e termina quando recebe o eco, com esse tempo se consegue identificar a distância que esse objeto se encontra do ponto de emissão dessa frequência sonora. Esse processo de identificação que o ultrassom utiliza é conhecido com ecolocalização, se parece com os mesmos métodos utilizados pelos golfinhos para identificação no fundo do mar e morcegos na escuridão, mesmo sendo diferente o sistema utilizado por esses animais, porque eles usam uma frequência muito mais baixa do que o ultrassom emprega (EVANS et al., 2013).

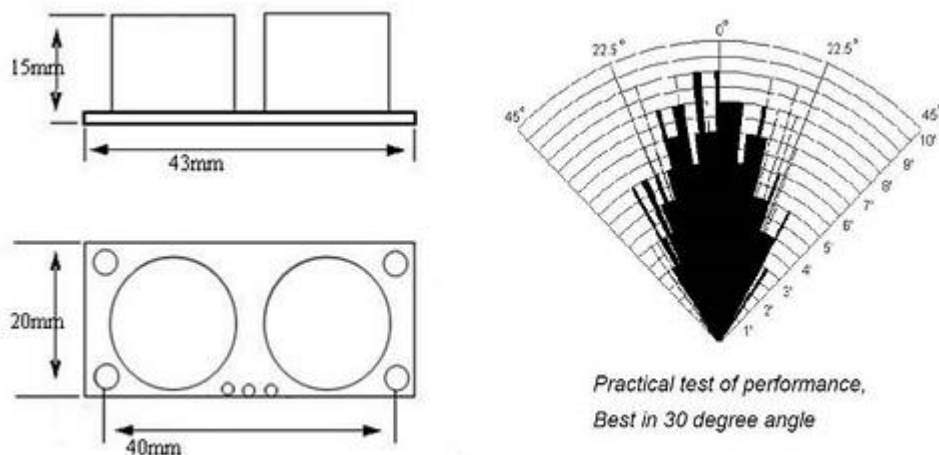
Ultrassônico baseia-se em dois elementos separados, um que proporciona um sinal e outro que recebe essa frequência sonar, sendo também incorporado um pequeno microcontrolador que determina o tempo. Os sensores se comunicam no mínimo com 0V e no máximo com 5V, o valor do tempo que é adquirido entra nos dois elementos que são transformados em uma tensão, se o atraso for maior, a tensão deverá ser maior (EVANS et al., 2013).

O ponto mais importante é o tempo de espera que ocorre quando é emitido o sinal e o seu retorno, a informação desse tempo irá depender do componente que

está sendo usado, podendo ser o Devantech SRF05 ou o Prallax Ping, entre outros. As informações sobre o equipamento são encontradas nas embalagens em diferentes formatos, um exemplo que temos é o Parallax Ping que informa 29.033uS que é o retorno da largura de pulso por centímetro. Isso é importante porque mostra a direção correta do tempo da distância percorrido, mas se você precisa de uma informação de centímetros para o objeto é necessário dividir o resultado por dois. (EVANS et al., 2013)

Sapkota ([s.d.]) nos mostra que é muito simples utilizar o HC-RS04 porque ele utiliza sinais sonares e oferece uma precisão na identificação, com leituras estáveis e uma biblioteca que é fácil de usar com Arduino. As leituras efetuadas pela ferramenta não é prejudicada por luz solar ou por outros materiais, mas existe um problema para identificar materiais macios como panos. Na figura 5 nos mostra as medidas e uma demonstração de teste HC-RS04.

Figura 5 – Descrição do HC-RS04



Fonte: FLICKR, 2015.

Em um momento zero, é enviado um pulso de ultrassom que o objeto recebe e reflete esse sinal, ao receber esse sinal o sensor transforma essa informação em um sinal elétrico. Quando o eco não estiver sendo identificado pode ser transmitido o impulso novamente, esse tempo de espera é conhecido com período de ciclo onde é

aconselhado que não seja menor que 50ms. Um impulso é transmitido para um terminal com oito disparos de 40KHz e sua largura de 10 μ s e identifica o eco de volta. Quando nenhum objeto é identificado, um sinal de alto nível de 38ms é destinado ao pino de saída.

3.5 O circuito integrado L293D

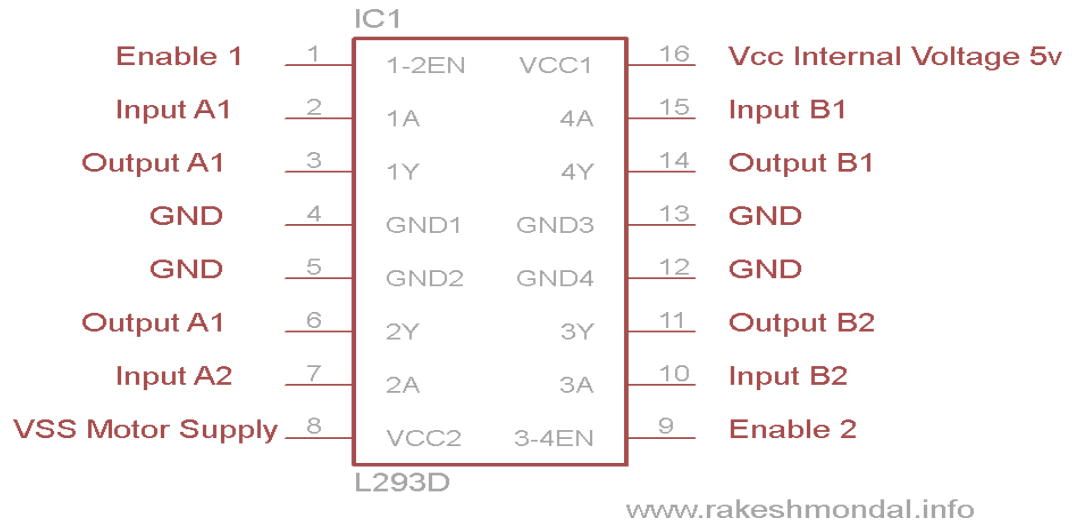
O L293D é um IC. Raskesron (2013) afirma ter 16 pinos sendo possível utilizar até dois motores DC nesse IC simultaneamente, em qualquer direção e com corrente contínua. O conceito de ponte-H é utilizado por esse circuito que permite a tensão ser mudada para qualquer direção. Quando a tensão é modificada ela pode ser capaz de alterar a direção da rotação do motor para horário ou anti-horário, com isso a ponte-H é um importante meio de condução de um motor DC.

O método mais comum para dirigir motores de corrente contínua em duas direções, sob o controle de um computador é, com um controlador de motor ponte-H. Ponte-H pode ser construído a partir do zero com transistores bipolares de junção (BJT) ou com transistores de efeito de campo (FET), ou pode ser comprado como uma unidade integrada em um único pacote de circuito integrado, como a L293. O L293 é mais simples e barato para motores de baixa corrente, para motores de alta corrente, é mais barato construir seu próprio Ponte-H a partir do zero. (DURFEE, [s.d], tradução livre).³

Raskesron (2013) continua dizendo que chip L293D possui dois circuitos ponte-H no interior do IC, sendo muito utilizado em robótica para controlar motores DC. Tem dois pinos que podem ser ativados para acionar o motor L293D, são eles os pinos 1 e 9, onde é necessário acionar para alterar, se qualquer um dos dois pinos ficar para baixo o motor poderá suspender a sua atividade. Para que esses dois pinos fiquem acionados na alta, simplesmente podem ser ligadas ao VCC pin16(5v). A figura 6 mostra os pinos disponíveis no L2293D.

Figura 6 – Diagrama de pinos do L293D

³ The most common method to drive DC motors in two directions under control of a computer is with an H-bridge motor driver. H-bridges can be built from scratch with bi-polar junction transistors (BJT) or with field effect transistors (FET), or can be purchased as an integrated unit in a single integrated circuit package such as the L293. The L293 is simplest and inexpensive for low current motors, for high current motors, it is less expensive to build your own H-bridge from scratch.



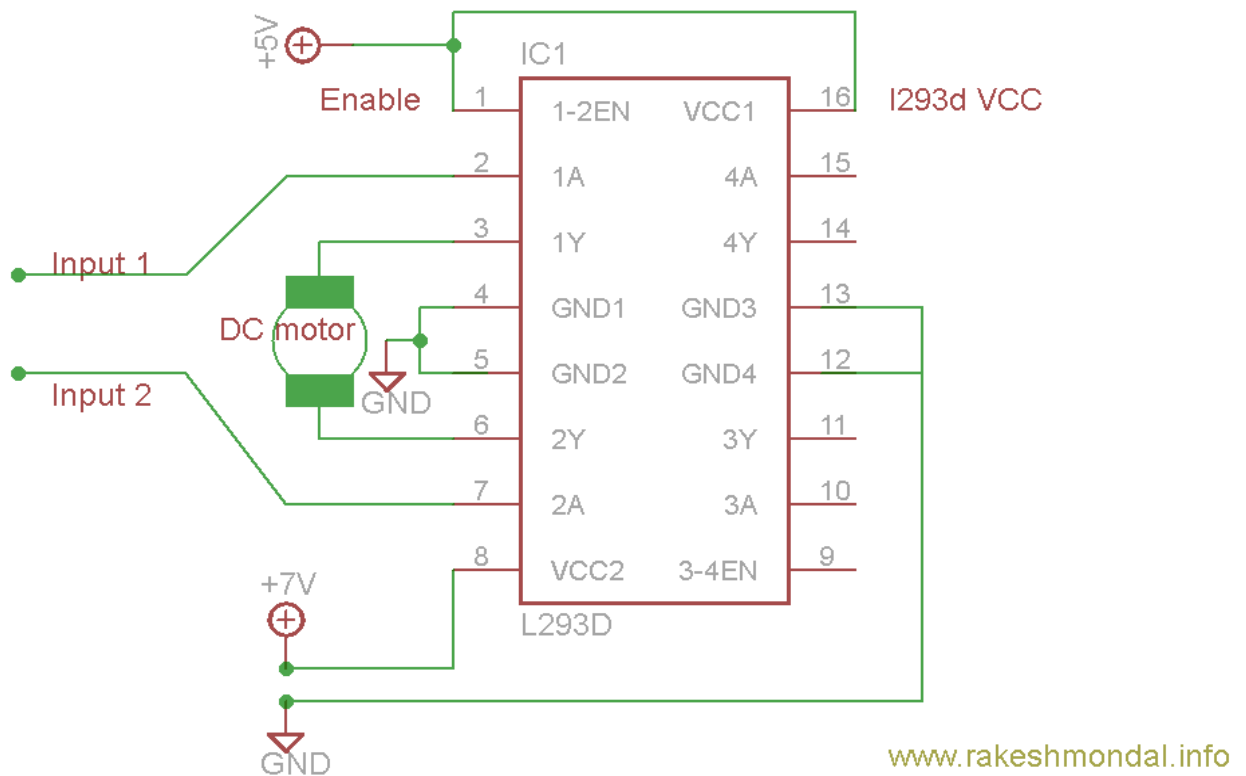
Fonte: RASKESRON, 2013.

Existem 4 pinos de acordo com Raskesron (2013), que regulam a tensão do motor, sendo eles o 2 e 7 na esquerda, 15 e 10 na direita. Os motores rodam com informações que são passadas para os pinos de entradas lógicas (0 ou 1), sendo simplesmente fornecer 1 ou 0 para girar o motor. Para que um motor rode no sentido horário ligado aos pinos do lado esquerdo (3,6), os pinos de entrada devem estar definidos 1 ou 0. Raskesron (2013) mostra diferentes formas lógicas nos pinos e qual a reação do motor:

Pino 2 = Lógica 1 e pino 7 = Lógica 0 | sentido horário. Pino 2 = Lógica 0 e Pino 7 = Lógica 1 | anti-horário Direção. Pino 2 = Lógica 0 e Pino 7 = Lógica 0 | ocioso [Sem rotação]. Pino 2 = Lógica 1 e pino 7 = Lógica 1 | ocioso [Sem rotação]. (RASKESRON, 2013).

Esse exemplo demonstrado funciona da mesma forma no lado direito nos pinos 14 e 10. A figura 7 é a que Raskesron (2013) apresenta como um diagrama de circuito motor driver controlado pelo IC L293D.

Figura 7 – Diagrama de circuito para controlar motor com L293D IC



Fonte: RASKESRON, 2013.

Para que o funcionamento interno 5V ocorra bem Rasquesron (2013) diz que é necessário o VCC que é uma tensão, o L293D não aciona o motor com essa voltagem e sim uma outra fonte de energia VSS (VCC). Quando precisamos acionar um motor com uma voltagem maior sendo um exemplo 9V é necessário fornecer uma fonte de 9V para alimentar VSS Motor. O limite do VSS do motor é de 36V, sendo possível dirigir até grandes motores como L293D, não sendo recomendado ultrapassar os 36V indicados, podendo causar danos. O VCC tem uma tensão que varia entre 5V e 36V sendo o pino 16 com essa tensão para o funcionamento interno do IC.

4 ARDUINO

4.1 Breve Histórico

Um dos fundadores da placa Arduino conhecido como Massimo Banzi, afirma John (2014), deu o nome a placa de baixo custo, “Arduino”, em homenagem ao bar que ele frequentava. A cidade de Ivrea que fica na Itália é conhecida pelos seus reis, e no ano de 1002 d.C., o rei Arduin governava o país até que o rei da Alemanha, Henrique II o tirou de seu trono. O local onde uma nova era em eletrônica surgiu, recebeu seu nome em memória ao rei que o frequentava, 'Bar Di Re Arduino '.

Na cidade de Ivrea na Itália, em 2005 afirma John (2014), surgiu a placa Arduino num laboratório do Instituto de Design Interativo. A placa Arduino é um microcontrolador de desenvolvimento em *open source* que ajudou bastante os engenheiros criativos e uma serie de designers. Um colombiano conhecido como Hernando Barragan contribuiu para o desenvolvimento dessa placa fantástica, ele se encontrava no instituto de Design Interativo pesquisando a sua tese de *hardware*. A tese de Hernando tem o nome de “Arduino–La rivoluzione dell'open *hardware*” que tem o significado de “A revolução do *hardware* aberto”. Depois mais cinco pessoas ajudaram o aluno Hernando na sua tese, foram desenvolvedores que trabalharam para torna-lo mais barato, leve, e disponível para a comunidade de código aberto, sendo possível a qualquer pessoa desenvolver a placa e contribuir no desenvolvimento fazendo melhorias.

John (2014) continua dizendo que no ano de 2002 foi contratado um arquiteto de *softwares* que se chama Benzi com a intenção de criar novas formas de fazer designer interativo, ou seja, computação física. Ele teve boas ideias, mas não conseguiu fazer muito por causa do pouco tempo de aula e orçamento baixo. Assim como seus amigos, Banzi utilizou um microcontrolador que foi desenvolvido pela empresa Parallax que tem sua sede na Califórnia, o BASIC Stamp foi utilizado por 10 anos pelos engenheiros. Esse microcontrolador foi desenvolvido utilizando a linguagem Basic e era desorganizado com muitos elementos como por exemplo uma fonte de alimentação, portas de entrada e saída que possibilitavam o *hardware* ser anexado e memória. Banzi tinha duas observações sobre a placa BASIC Stamp, ela

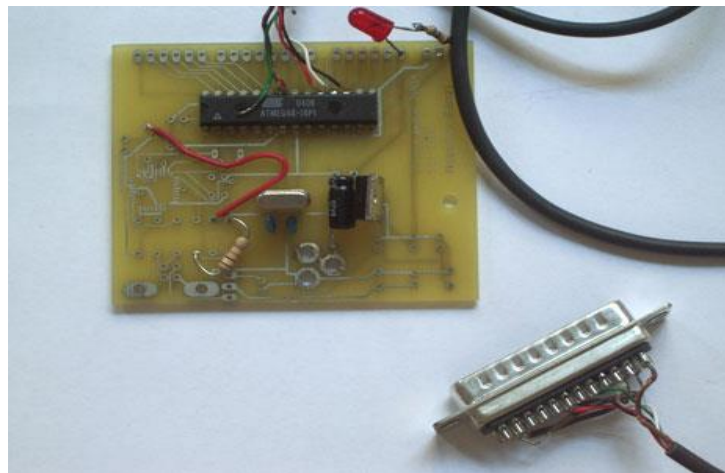
não suportava os projetos que eram feitos pelos alunos porque possuía pouco poder de processamento computacional e era uma placa que possuía um preço elevado, US\$ 100 era o valor da placa com as partes básicas.

Um amigo de Banzi desenvolveu uma linguagem de programação que tem o nome Processing sendo bem amigável, John (2014) informa que:

Processing foi rapidamente ganhando popularidade a medida que permitia até mesmo programadores amadores a criar visualizações de dados complexos e belos! Este foi um Ambiente de Desenvolvimento Integrado ou IDE de extrema fácil utilização. Banzi realmente gostou deste conceito e imaginou se ele e sua equipe poderiam criar um software semelhante ao código de um microcontrolador em vez de gráficos em uma tela. (JOHN, 2014)⁴

O estudante Hernando Barragán deu os primeiros passos no processo do desenvolvimento, ele criou a primeira placa chamada Wiring que continha tanto uma placa de circuito pronta para uso e um IDE amigável. Um projeto que continua até hoje sendo prospero, mas Banzi já tinha um pensado em algo melhor, um dispositivo mais simples de se usar e bem mais barato.

Figura 8 – Imagem do primeiro protótipo feito em 2005



Fonte: KUSHNER, 2011.

O primeiro projeto desenvolvido que podemos ver na figura 8, não recebeu o nome de Arduino mas Massimo Banzi, de acordo com Kushner (2011), iria naquele

⁴ Processing was quickly gaining popularity as it enabled even amateur programmers to create complex and beautiful data visualizations! It was an extremely easy-to-use Integrated Development Environment or IDE. Banzi really liked this concept and wondered if he and his team could create similar software programs to code a microcontroller instead of graphics on a screen.

ano assinalar como Arduino o projeto simples que tinham acabado de desenvolver. Banzi e seus colegas acreditaram no desenvolvimento *open source*, escolheram esse tipo de desenvolvimento porque o IDII estava operando já a cinco anos e estava ficando sem verbas para continuar e tudo indicava que iria fechar as portas. Os membros docentes tinham medo que os projetos fossem desviados ou não sobrevivessem. Os colaboradores acreditavam que desenvolvendo em *open source* eles criariam uma plataforma rápida e de fácil acesso, abrindo o projeto para um número maior de pessoas seria melhor ao invés de mantê-lo fechado.

Kushner (2011) diz que não tinham utilizado modelo de fonte aberta para *hardware* e sim para *software*. Para que isso desse certo eles começaram a investigar soluções de licenças que se encaixassem no projeto deles. Com o passar do tempo encontraram algo diferente, mas que iria ajudar eles, a licença *Creative Commons*, a qual normalmente é utilizada para obras culturais, como música e escrita. Banzi disse: “Você poderia pensar em *hardware* como parte da cultura que você deseja compartilhar com outras pessoas”.

O grupo tinha um objetivo que era disponibilizar a placa por \$30, “Tinha que ser o equivalente a sair para jantar em uma pizzaria”, informa Banzi. Eles tinham também como meta a alcançar, fazer algo diferente, como as placas fabricadas por outras empresas eram muito verdes eles desenvolveram placas azuis, enquanto outras economizavam na entrada e saída de pinos, seria adicionado mais em suas placas. Para finalizar eles acrescentaram um mapa da Itália na parte de trás da placa (KUSHNER, 2011).

Os usuários poderiam utilizar os produtos que as equipes desenvolviam com peças que são fáceis de se encontrar e baratas, se os usuários seguissem as instruções proposta por eles, John (2014) continua dizendo que:

No entanto, uma decisão importante foi verificar que seria essencialmente plug and play: algo que alguém poderia tirar de uma caixa, ligar em um sistema e usá-lo imediatamente. Por outro lado, placas conhecidas como o BASIC Stamp exigiu que os usuários desembolsassem um monte de outros itens que, em última análise adicionava ao custo total. No entanto, para o Arduino, um usuário precisa apenas retirar um cabo USB da placa e apenas conectá-lo a

um computador para programar o dispositivo. (JOHN, 2014, tradução livre)⁵

Para se aprender a eletrônica, dizia um engenheiro de telecomunicações conhecido como David Cuartielles Louros, a pessoa não precisava aprender álgebra ou outras coisas, ela já poderia começar a aprendê-la no primeiro dia. O Arduino é a ferramenta fundamental para esse aprendizado possibilitando o aprendizado desde o primeiro dia. A acessibilidade e o conceito fantástico que o Arduino propõe encantou um professor de física da computação que morou em Nova York, hoje ele faz parte da equipe Arduino.

A equipe começou distribuindo 300 placas de circuitos impresso para os alunos e disseram que eles deveriam buscar as instruções de montagem em linha, para produzirem suas próprias placas sendo possível utilizar para as necessidades que eles possuíam. Um dos primeiros projetos bem interessantes foi um despertador que foi prendido ao teto com um cabo sendo capaz de subir se o usuário não levantasse para desligar e apertasse o botão de soneca o relógio subia até chegar o momento que o usuário era obrigado a se levantar para desligar a ferramenta. (KUSHNER, 2011)

Sendo uma placa poderosa e com um valor acessível para todos em pouco tempo informa John (2014) que a placa Arduino se espalhou como uma onda pelo mundo sem *marketing*. O primeiro cliente da placa foi um amigo do Banzi que pediu uma unidade. Kushner (2011) diz que a utilização de microcontrolador antes do Arduino era difícil de se aprender, mas o Arduino possibilitou um ambiente de programação fácil. Um mundo que era impenetrável por pessoas que não tinham experiência agora ganhou acesso ao mundo de *hardware*, sem precisar estudar muito antes de desenvolverem ferramentas impressionantes com Arduino. Foi um momento marcante o desenvolvimento do Arduino porque as empresas que produziam esses tipos de placas eram fechadas e protegidas por patentes e o Arduino completamente aberto.

⁵ However, an important decision was to ascertain that it would essentially be plug and play: something someone could just take out of a box, plug into a system and use it right away. On the other hand, boards such as the BASIC Stamp demanded the users to shell out a lot of other items that ultimately added to the total cost. However, for the Arduino, a user needs to just pull out a USB cable from the board and merely connect it to a computer to program the device.

4.2 Hardware

O microprocessador de 8 *bits* Atmel AVR (RIS) é uma das peças fundamentais na construção do Arduino, sendo base até o momento para uma série de versões. Rodando em uma velocidade de *clock* de 16 Mhz o ATmega8 foi utilizado na criação da primeira placa com 8KB de memória *flash*, com o passar do tempo surge placas conhecidas como Diecimila (nome italiano para 10.000) e Arduino NG plus e elas utilizavam memória flash de 16KB e um microprocessador Atmega168. (EVANS et al., 2013)

Conforme Evans et al. (2013), o ATmega328 microprocessador com memória *flash* de 32KB é utilizado nas versões mais recentes do Arduino, Uno e Duemilanove e automaticamente trabalham com troca de corrente contínua (DC) e USB. Se for necessário trabalhar com mais memória e entrada/saída, há dois modelos que podem suprir essas necessidades sendo o Arduino Mega2560 com memória de 256KB um dos mais recentes disponíveis no mercado, ou o Arduino Mega1280 que tem uma memória de 128KB. Evans et al. (2013) no texto a seguir explica o que possui nas placas.

As placas têm 14 pinos digitais, e cada um pode ser definido como entrada ou saída, e seis entradas analógicas. Além disso, seis dos pinos digitais podem ser programados para fornecer uma saída de modulação por largura de pulso (PWM). Diversos protocolos de comunicação estão disponíveis, incluindo serial, bus serial de interface periférica (SPI) e I2C/TWI. Incluídos em cada placa como recurso padrão estão um conector de programação serial in-circuit (ICSP) e um botão de reset. (Evans et al., 2013, p. 26)

4.2.1 Arduino Uno

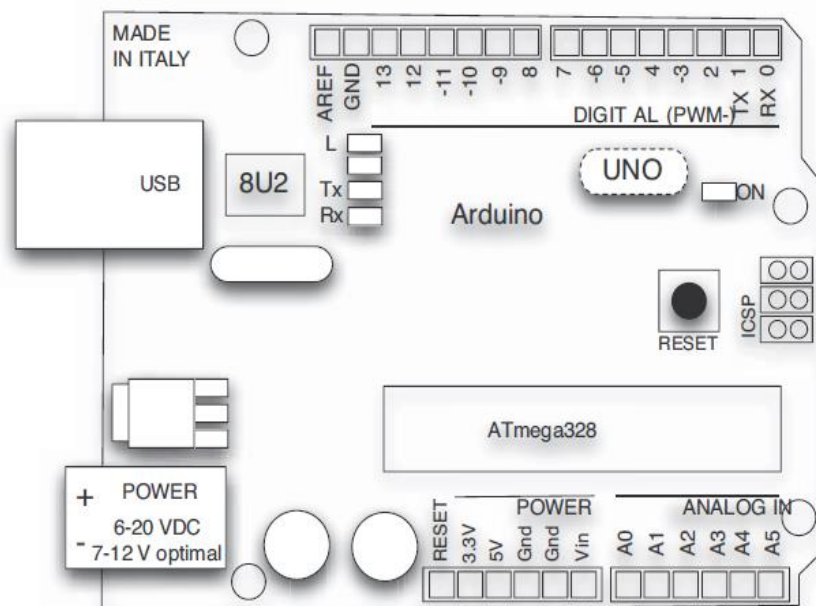
Conforme Evans et al. (2013) no dia 25 de setembro de 2010 foi anunciado um título no *blog* que dizia “O Jantar está Servido”, foi nesse tempo que marcou a chegada do Mega2560 e do seu irmão mais novo Arduino Uno (traduzindo Uno significa um). Os modelos Duemilanove e Diecimila eram modelos anteriores, mas o Arduino possuía compatibilidade com os seus pinos. A diferença que encontramos no modelo Uno com os seus antecessores é a e a substituição do *chipset* FTDI e integrando o

ATmega8U2 microcontrolador que possui um conversor USB para serial. Evans et al. continua dizendo que:

ATmega8U2 pode ser reprogramado para fazer o Arduino se parecer com outro dispositivo USB, tal como mouse, teclado ou joystick. Outra diferença é que ele possui uma tensão integrada de 3,3 V mais confiável, o que ajuda na estabilidade de algumas proteções que causavam problemas no passado. (EVANS et al., 2013, p. 27)

O Evans et al. (2013) informa que quando se necessita de uma placa de partida com tensão integrada de 3,3V regular e fonte de alimentação auto chaveada uma boa opção é utilizar o Arduino Uno sendo uma boa aposta. A figura 9, mostra com detalhes como é a placa Arduino.

Figura 9 – Pinos do Arduino Uno e Layout da placa.



Fonte: Adaptado de Arduino em Ação, p.27.

Baseado no ATmega328 o microcontrolador Arduino Uno contém os acessórios necessário para um bom funcionamento, são eles um botão de *reset*, um cabeçalho ICSP, um cerâmico ressonador de 16MHz, conexão USB, 14 pinos digitais de entrada/saída onde 6 podem ser utilizados como saídas PWM e 6 entradas analógicas. Para começar a utilizar essa placa basta conectar o cabo USB ou uma bateria podendo ser utilizado um adaptador AC para DC. O Uno tem uma transformação que o torna alterado em comparação com os outros modelos anteriores, os modelos anteriores utilizavam USB para serial FTDI já o Uno possui o ATmega8U2 que é utilizado até a versão R2 e atualmente apresenta o ATmega16U2

que funciona como um conversor USB para serial. A figura 10 apresenta um resumo do Arduino Uno. (ARDUINO, [s.d.])

Figura 10 – Especificações de um Arduino Uno.

Resumo

Microcontrolador	ATmega328
Tensão de funcionamento	5V
Tensão de entrada (recomendado)	7-12V
Tensão de entrada (limites)	6-20V
Digital pinos I / O	14 (dos quais 6 oferecem saída PWM)
Analog pinos de entrada	6
Corrente DC por I / O Pin	40 mA
Corrente DC 3.3V para Pin	50 mA
Memória Flash	32 KB (ATmega328), dos quais 0,5 KB utilizado pelo bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidade do relógio	16 MHz

Fonte: ARDUINO, 2014.

Arduino ([s.d.]) alega que a alimentação pode acontecer de duas maneiras, conexão USB sendo uma fonte de alimentação externa ou com um conector *plug* 2,1 milímetros. A voltagem para que a placa possa operar é entre 6 a 20 volts, mas o recomendado está entre 7 a 12 volts porque se for maior que 12V o regulador de voltagem pode ser danificado e deteriorar a placa. Se for menor que 7V a placa pode fornecer menos que 5V e tornar oscilante a alimentação.

A entrada e saída opera com 5 volts e podem ser utilizados os 14 pinos digitais do Uno, 40 mA é o valor que cada pino receber ou fornecer. Em conformidade com Arduino ([s.d.]) 10 bits de resolução possuem as 6 entradas analógicas que começam pelo nome A0 a A5 e por estrutura padrão da fábrica esses pinos medem 5 volts.

De acordo com Arduino ([s.d.]) existem outros pinos que possuem funções específicas como apresenta a Figura 11.

Figura 11 – Informações de alguns pinos do Arduino.

- Serial: 0 (RX) e 1 (TX). Utilizado para receber (RX) e transmitir dados seriais (TX) TTL. Estes pinos são ligados aos pinos correspondentes do ATmega8U2 USB-TTL chip de série.
- Interrupções externas: 2 e 3 Estes pinos podem ser configurados para disparar uma interrupção por um valor baixo, uma borda de subida ou queda, ou uma mudança de valor. Veja o `attachInterrupt()` função para obter detalhes.
- PWM: 3, 5, 6, 9, 10, 11 e Fornecer saída PWM de 8 bits com a `analogWrite()` função.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK) Estes pinos suporte à comunicação SPI usando a `biblioteca SPI`.
- LED: 13. Há um built-in LED conectado ao pino digital 13. Quando o pino é de alto valor, o LED está ligado, quando o pino é baixo, ele está fora.

Fonte: ARDUINO, 2014.

O ATmega328 disponibiliza ao Arduino uma facilidade para se conversar com computadores e adotar uma comunicação serial UART TTI (5V) que se encontra nos pinos digitais 1 (TX) e 0 (RX). A comunicação acontece como uma porta COM virtual através do USB onde o *firmware* 16U2 utiliza por padrão os *drivers* USB COM e não é necessário utilizar outro *drive*, somente no sistema operacional Windows é obrigatório o uso do arquivo “.inf”. Os RX e TX LEDs que se encontram na placa Arduino piscam quando o *software* Arduino é utilizado na opção monitor serial sendo acrescentado texto para serem enviados a placa ou da placa para USB. Qualquer pino digital do Arduino Uno pode ser comunicação serial graças a biblioteca *SoftwareSerial*. (ARDUINO, [s.d.])

Antes de um *upload* é necessário fazer um *reset* na placa Arduino, em lugar de criar um botão para realizar o *reset* foi desenvolvido no *software* essa operação para facilitar a no desenvolvimento. A operação de *reset* acontece segundo o Arduino ([s.d.]) da seguinte forma:

Uma das linhas de controle de fluxo de hardware (DTR) do ATmega8U2 / 16U2 está ligado à linha de reset dos ATmega328 através de um condensador 100 nanofarad capacitor. Quando esta linha é (rebaixada), a linha de reset decai por tempo suficiente para resetar o chip. O software Arduino usa esse recurso para permitir que você faça o upload de código, simplesmente pressionando o botão de upload no ambiente Arduino. Isto significa que o carregador de inicialização pode ter um tempo de espera mais curtos, como a redução de DTR pode ser bem coordenado com o início do carregamento. (ARDUINO, [s.d.])⁶

⁶ One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload

Arduino ([s.d.]) continua dizendo que existe um sistema de proteção do Arduino Uno conhecido como POLYFUSE reajustável, que projete a porta USB do computador pelos shorts e sobre corrente. Mesmo que os computadores contenham um sistema de proteção interna eles acrescentarão um fusível como uma camada extra de proteção. O fusível rompe se houver mais de 500 mA sobrecarregando a porta USB, volta ao normal até o momento que a sobre carga ou curto é amenizado.

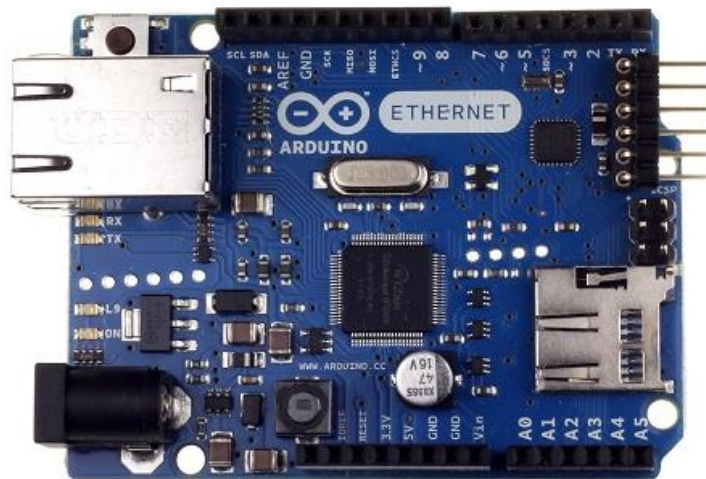
As características físicas do Arduino segundo Arduino ([s.d.]) possui um comprimento de 2,7 com uma largura de 2,1 polegadas. Para permitir que a placa seja ligada em uma superfície foi implementado 4 orifícios de passagem para parafusos, onde também temos os pinos digitais 7 e 8 com uma distância entre eles de 160mil e os outros pinos um espaçamento de 100 mil.

4.2.2 Arduino *Ethernet*

Uma versão de baixa potência o Arduino *Ethernet* foi apresentado ao mundo na mesma época que o Arduino Uno conforme EVANS et al. (2013). O Arduino *Ethernet* só possui algumas diferenças entre as versões, que são um cartão micro SD e um conector RJ45 para comunicar com conexão *Ethernet*. O *chip* controlador USB serial não foi acrescentado a essa versão do Arduino mas um cabo FTDI pode ser inserido em um conector de seis pinos ou também para que se possa programar na placa ela tem uma porta serial USB que fornece um link de comunicação externa. A alimentação pode ser desfrutada pelo módulo *Power over Ethernet*, quando o cabo *Ethernet* par trançado categoria 5 está conectado, ele proporciona ao Arduino *Ethernet* os suprimentos para se manter ligado. A figura 12, mostra como é a placa do Arduino *Ethernet*.

Figura 12 – Arduino *Ethernet* Rev.3

code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.



Fonte: ARDUINO, 2014.

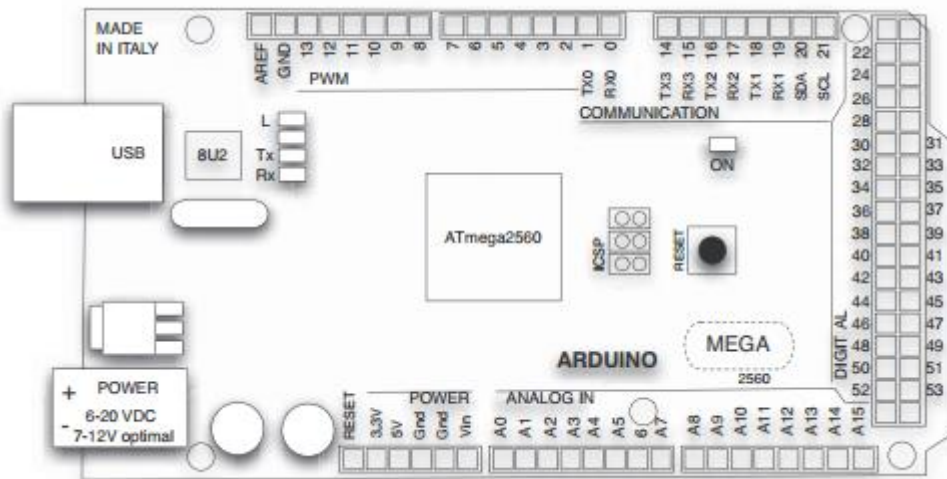
4.2.3 Arduino Mega

Com um microprocessador de superfície de montagem maior o irmão mais velho da família Arduino conhecido como Mega, de acordo com EVANS et al. (2013) foi atualizado ao mesmo tempo que o Uno, empregando o ATmega1280, sendo hoje em dia aplicado o ATmega2560. Com uma memória *flash* de 253 KB a nova versão tem mais vantagem que o Original de 128 KB. Com um aumento na funcionalidade de entrada e saída em comparação com o Arduino padrão, o Mega é ideal para projetos que requerem mais quantidades de entrada e de processamento ou porta de serial de *hardware* onde o Mega disponibiliza quatro dessas entradas. Com 54 pinos digitais de entrada, o Mega tem ainda mais 16 pinos de entrada analógica, e de saída analógica PWM são 14 pinos. Estão disponíveis suporte para dispositivos I2C/TWI com comunicação SPI, com até quatro portas seriais de *hardware* é possível fazer uma comunicação. A placa apresenta um botão de reset e integra um conector ICSP, o chipset FTDI foi substituído pelo ATmega8U2 e processa uma comunicação serial USB.

Os *Shields* disponíveis no mercado, informam EVANS et al. (2013), estão aptos para serem utilizados no Arduino Mega mas não pode se deixar de lado a consulta ao manual para verificar se é compatível com a placa na qual se deseja utilizar. É aconselhado a compra de um Mega quando há urgência em utilizar muitos pinos de

entrada/saída e adotar mais memória. A figura 13 mostra o *layout* do Arduino Mega onde é possível constatar os pinos adicionais de entrada/saída juntamente com as portas extras.

Figura 13 – Layout do Arduino Mega



Fonte: Adaptado de Arduino em Ação, p.29.

A largura da PCB Mega2560 é de 2,1 com um comprimento de 4 polegadas no máximo, junto com a tomada de conector USB pode-se estender para além do seu espaço. Observando a distância dos pinos digitais 7 e 8 são de 160 mil e dos demais pinos 100 mil. É também compatível com a maioria dos *shields* desenvolvidos para o Decimila, Uno e Duemilanove. (ARDUINO, [s.d.])

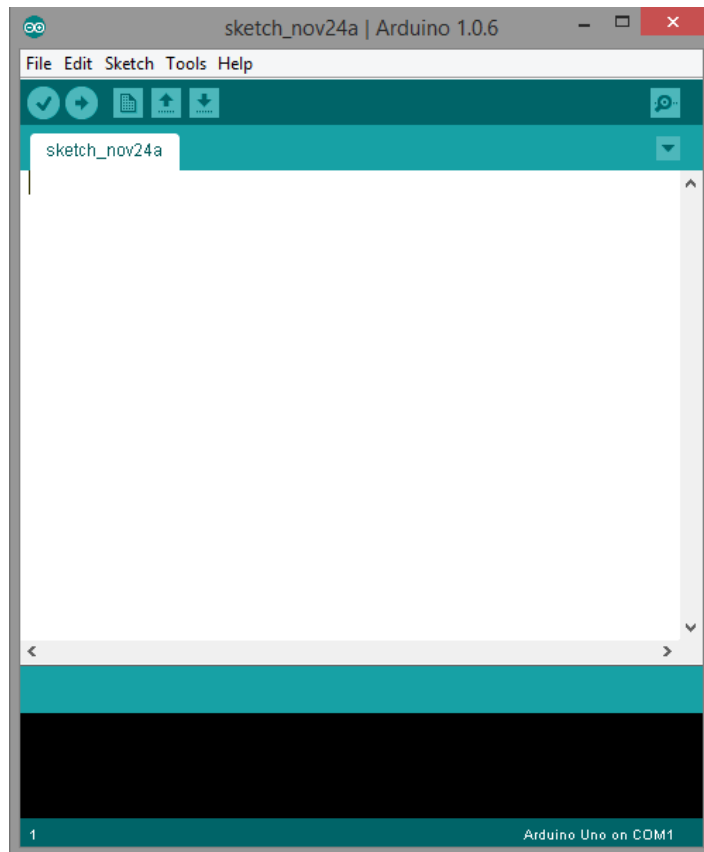
4.3 Arduino IDE

O ambiente de desenvolvimento do Arduino, é um software open-source que facilita o desenvolvimento dos *Sketches*, como são conhecidos os códigos dos programas feitos para Arduino. Ele roda em Windows, Linux e Mac OS X, e “o ambiente é escrito em Java e baseado em Processing, avr-gcc, e outros *softwares* de código aberto.” (ARDUINO, [s.d])

Atualmente está na versão 1.0.6, o software é basicamente o mesmo com apenas algumas pequenas diferenças de acordo com o Sistema Operacional, como aponta McRoberts. (2011)

McRoberts (2011) explica que a primeira tela ao ser executado é algo parecido com a figura 14:

Figura 14 – Tela inicial do Arduino IDE



Ele prossegue explicando que o IDE é dividido em três partes: a barra de ferramentas (*Toolbar*) em cima, o código ou *Sketch Window* ao centro e a janela de mensagens de erros em baixo.

Os botões da *Toolbar*, dão acesso rápido às funções contidas nos menus *File*, *Edit*, *Sketch*, *Tools* e *Help*.

São seis botões: 'Verify' onde verifica-se há erros, 'Upload' que carrega o *sketch* atual no Arduino, 'New' que cria um novo *sketch*, 'Open' que mostra um menu com todos os *sketches* além de abrir o *sketch* escolhido, 'Save' que salva o *sketch* e 'Serial Monitor' que abre o monitor de serial. (ARDUINO, [s.d])

O menu *File* (figura 15) contém as opções para criar ou abrir *sketches* do *SketchBook* ou de exemplos pré-definidos. Também para fechar, salvar, salvar como e imprimir o código, além da opção preferencias.

Figura 15 - Menu File

New	Ctrl+N
Open...	Ctrl+O
Sketchbook	▶
Examples	▶
Close	Ctrl+W
Save	Ctrl+S
Save As...	Ctrl+Shift+S
Upload	Ctrl+U
Upload Using Programmer	Ctrl+Shift+U
Page Setup	Ctrl+Shift+P
Print	Ctrl+P
Preferences	Ctrl+Comma
Quit	Ctrl+Q

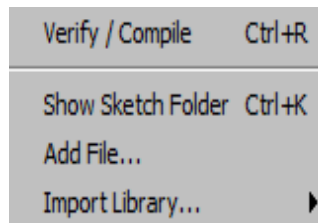
No menu *Edit* (figura 16) estão as opções de edição do código tais como copiar, recortar, colar, refazer, desfazer, comentar e procurar palavras ou frases. (MCROBERTS, 2011)

Figura 16 – Menu Edit

Undo	Ctrl+Z
Redo	Ctrl+Y
Cut	Ctrl+X
Copy	Ctrl+C
Copy for Forum	Ctrl+Shift+C
Copy as HTML	Ctrl+Alt+C
Paste	Ctrl+V
Select All	Ctrl+A
Comment/Uncomment	Ctrl+Slash
Increase Indent	Ctrl+Close Bracket
Decrease Indent	Ctrl+Open Bracket
Find...	Ctrl+F
Find Next	Ctrl+G
Find Previous	Ctrl+Shift+G
Use Selection For Find	Ctrl+E

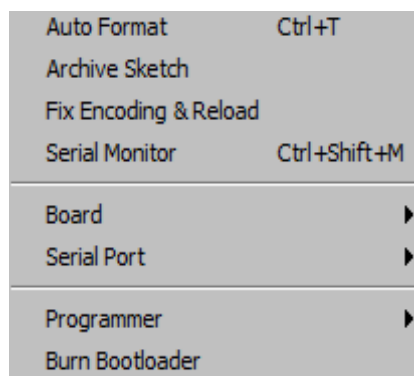
O menu *Sketch* (figura 17) permite verificar o código, abrir a pasta de *Sketch* atual, adicionar arquivos e importar bibliotecas. Segundo McRoberts (2011, p.37) adicionar arquivos, “permite que você adicione outro arquivo de fonte ao seu sketch, permitindo a divisão de sketches maiores em arquivos menores[...]”. Já a biblioteca, é um conjunto de código que pode ser adicionado ao *sketch* a fim de fornecer certas funcionalidades prontas de modo que o desenvolvedor não precise inventar uma função que já existe, podendo assim apenas reutiliza-la. É o caso da biblioteca *Stepper* que controla um motor de passo. Com essa biblioteca adicionada o desenvolvedor pode utilizar suas funções quantas vezes quiser e em quantos *sketches* precisar.

Figura 17 – Menu *Sketch*



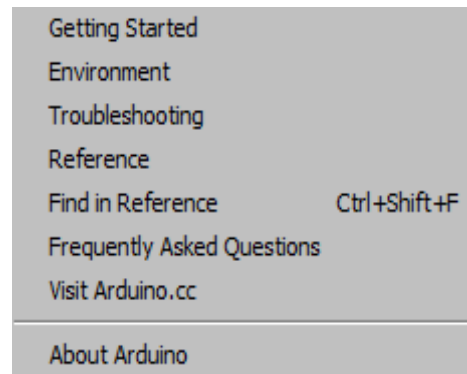
O menu *Tools* (figura 18) é onde estão as ferramentas para, formatar o código automaticamente deixando-o mais organizado, compactar o *sketch* em um arquivo zip, selecionar o modelo da placa Arduino a ser utilizada, abrir o monitor de serial e gravar o *bootloader* (gerenciador de boot) na placa Arduino. (ARDUINO, [s.d.])

Figura 18 – Menu *Tools*



Por fim o menu *Help* (figura 19) de acordo com McRoberts (2011, p. 38) “é o local em que você encontra mais informações sobre o IDE, ou links para as páginas de referência do site do Arduino e outras páginas úteis. ”

Figura 19 – Menu Help



McRoberts (2011) afirma existem outras IDEs mais sofisticadas e profissionais que podem ser usadas para programar em Arduino C, algumas delas gratuitas como Eclipse, ArduIDE, GNU/Emacs, AVR-GCC e AVR Studio.

4.4 Arduino C

A linguagem de programação usada no ambiente de desenvolvimento do Arduino tem como base a linguagem de programação computacional C, que foi fruto das pesquisas dos laboratórios Bell no início dos anos 1970 com o objetivo de integração com o sistema operacional UNIX. Os códigos escritos em C precisam passar por um compilador que converte a linguagem humana para linguagem de máquina. (EVANS, 2011)

Evans (2011) aponta que devido ao fato do C ser uma linguagem que por alguns motivos pode parecer muito complicada para iniciantes a equipe de desenvolvimento do Arduino criou uma biblioteca padrão para o Arduino, onde existem diversas funções prontas que transformam a programação do Arduino em algo mais simples.

No entanto é importante notar que essa biblioteca padrão está baseada no C, portanto a maioria do código que for escrito para Arduino, isto inclui: sintaxe, estruturas, operadores, controladores e funções; é basicamente C, o diferencial do Arduino são suas funções específicas de sua biblioteca como por exemplo `pinMode()`, `digitalWrite()` e `delay()`. Evans (20xx) refere-se a esta linguagem como Arduino C.

Segundo Evans (2011), um *software* Arduino é conhecido como um *sketch*, que em inglês significa esboço. A ideia por trás deste termo é escrever um código como se fosse o rascunho de um desenho, assim tão logo a pessoa tenha uma ideia, ela rapidamente pode transformá-la em código.

4.4.1 Estruturas

O Arduino C segue regras de sintaxe e estrutura que definem para a placa o que ela deve executar. Algumas destas regras vieram do C, outras a equipe de desenvolvimento do Arduino implementou a fim de simplificar o aprendizado para os iniciantes. Assim como C, Arduino C segue a estrutura *top-down*, isto é, as linhas do código são executadas uma a uma, desde a primeira até a última, e de cima para baixo. (EVANS, 2011)

4.4.1.1 Comentários

Evans (2011) explica que os comentários são trechos do código que são ignorados pelo compilador, porém auxiliam os desenvolvedores, pois servem como notificação e documentação do *sketch*. Assim, é possível deixar explicações das finalidades do programa, das variáveis, funções ou qualquer outra linha do código para outras pessoas que forem utilizar o *sketch*. Pode ser útil também para fazer testes, para tanto, basta comentar a linha e ver o que acontece com o programa sem ela, desse modo não se perde o código anterior, caso seja necessário voltar ao antigo estado.

Há duas maneiras de comentar, a primeira é colocar os caracteres: // no começo da linha desejada. Deste modo apenas aquela linha ficará desabilitada. Por exemplo:

```
// digitalWrite(10, HIGH);
```

Faz com que o comando “digitalWrite(10, HIGH);” não seja reconhecido pelo compilador. O // também pode ser colocado em qualquer linha em branco a fim de especificar algo. No entanto isso se torna cansativo quando o objetivo é comentar

várias linhas em sequência. A segunda maneira supre essa necessidade. Ela permite fazer comentários em várias linhas, as quais devem estar entre os caracteres: `/*` e `*/`. Eis um exemplo:

```
/* Comentários em
múltiplas
linhas! */
```

4.4.1.2 Funções Básicas

O *sketch* é dividido em blocos de códigos conhecidos como funções que formam os dois blocos principais de qualquer programa em Arduino C. Essas duas funções são `setup()` e `loop()`, elas são os requisitos mínimos de todo *sketch*, portanto o seguinte exemplo de acordo com Evans (2011) é um *sketch* completo e tecnicamente funcional, embora não execute nada:

```
void setup() {
}

void loop() {
}
```

Neste exemplo pode-se identificar o comando `'void'` que especifica o tipo data da função, o nome da função, neste caso `'setup'`, e os parênteses servem para que a função receba parâmetros que serão trabalhados dentro das chaves, onde devem ser colocados os comandos a serem executados pela função.

A função `setup()` é chamada apenas no início do *sketch* e é usada para configurar a placa Arduino, deixando-a pronta para rodar o resto do código. Só precisa ser executada uma vez. O site oficial do Arduino declara a respeito de suas possibilidades:

Use-o para inicializar variáveis, modos de pino, começar a usar as bibliotecas, etc. A função de configuração será executada apenas uma

vez, após cada energização ou reset da placa Arduino. (ARDUINO, [s.d.])

A segunda função é o loop (), que permite que o *sketch* continue a rodar infinitamente. Ela começa logo após o término da função setup (), portanto toda vez que o código chega ao final na função loop (), ele volta ao início do loop (). (EVANS, 2011)

4.4.1.3 Declarações e Sintaxe

As linhas de código inseridas entre as chaves chamam-se declarações. Algumas delas chamam outras funções, em outras é possível declarar variáveis ou realizar cálculos e comparações. Existe dois tipos de declarações, a simples e a composta. A declaração simples é a mais comum e sempre termina com ponto e vírgula (;). É preciso atentar ao ponto e vírgula pois o mais comum dos erros cometidos está em esquecer de pô-lo ao fim da declaração. Exemplos de declarações simples:

```
delay(1000);
```

```
digitalWrite(11, HIGH);
```

```
int myValue = 0;
```

```
myValue = 150 + 150;
```

A declaração composta é feita por linhas de códigos que contém outras declarações simples. Funções podem ser consideradas declarações compostas, controladores como o *if* e *else* também. Uma característica diferencial delas é que usam as chaves para envolver as declarações simples. Exemplo de declaração composta:

```
if (pin2 == HIGH) {
```

```
    digitalWrite(13, HIGH);
```

```
}
```

No entanto, as declarações compostas são mais recomendáveis quando a várias declarações simples a circundar. No exemplo acima, é possível transformar essa declaração composta em simples, ficando assim:

```
if (pin2 == HIGH) digitalWrite(13, HIGH);
```

Os parênteses sempre são usados em pares. Quando não há nada dentro, significa que determinada função não está recebendo nenhum valor ou determinado controlador não está recebendo nada para se comparar. Esse é o caso de algumas funções como `loop ()`. Por outro lado o exemplo `digitalWrite(13, HIGH);` recebe dois valores o valor do pino (13) e a constante (HIGH), porém é possível também fazer comparações entre variáveis e valores como no exemplo `if (pin2 == HIGH)`. (EVANS, 2011)

O jogo da velha (#) é usado basicamente em dois comandos, `#define` e `#include`. Segundo Arduino ([s.d.]), `#define` é um componente C que permite declarar uma variável com valor constante. É útil quando é preciso utilizar o mesmo valor em vários lugares do programa, e esses valores por algum motivo são alterados ao longo da execução. Assim ao declarar uma variável constante é só alterá-la que onde quer que ela esteja sendo usada o valor altera também. Portanto para declara-la basta escrever: `#define nomeConstante valor`.

O `#include` é usado para incluir outras bibliotecas no sketch atual. A sua sintaxe é a seguinte: `#include <biblioteca>`. É importante notar que em ambos os casos não é necessário por um ponto e vírgula no final. (ARDUINO, [s.d.])

4.4.2 Variáveis

Em programação, as variáveis são responsáveis por guardar informações. Cada uma delas deve possuir um nome que identifique o que possivelmente será guardado e também deve ter um tipo de dado, o qual vai dizer qual tipo de dado a variável poderá receber. (EVANS, 2011)

De acordo com Schildt (1996, p. 20), uma variável é definida como “uma posição nomeada de memória, que é usada para guardar um valor que pode ser modificado pelo programa.”

Antes de usar qualquer variável é preciso primeiro declará-la, segundo Schildt, (1996) a forma usual de declarar variáveis é a seguinte:

```
tipo nomeVariavel;  
  
int minhaVariavel = 0;
```

4.4.2.1 Tipos de Dados

O Arduino suporta vários tipos de dados, que são um conjunto de valores que identificam ao compilador quais valores da memória serão necessários durante a execução do programa. O site oficial do Arduino ([s.d.]) referencia os seguintes tipos: *void*, *boolean*, *char*, *unsigned char*, *byte*, *int*, *unsigned int*, *word*, *long*, *unsigned long*, *short*, *float*, *double* e *string*.

O tipo *void* é utilizado quando uma função não retorna valor, o tipo booleano (*boolean*) ocupa um *byte* na memória e tem como valor *true* (verdadeiro) ou *false* (falso).

Char também ocupa um *byte* na memória, porém guarda apenas um caractere, como por exemplo ‘a’, no entanto o que realmente fica guardado é um número que representa o caractere, para isso existe a tabela ASCII. As *strings* são *arrays* de *chars*. Quando o valor é passado para a variável há uma diferença, para *char* se passa entre aspas simples (‘A’) e para *string* se passa entre aspas duplas (“ABC”).

Um tipo primário básico de dado é o *int*, pode ocupar até dois *bytes* na memória e aceita números positivos e negativos, desde -32.767 a 32.767. Outro tipo básico é o ponto flutuante, *float*, ele ocupa quatro bytes na memória, também suporta positivos e negativos, indo de 3.4028235E+38 até -3.4028235E+38, vale notar que ele possui de seis a sete dígitos de precisão, isto é, o número total de dígitos. Sempre que possível é recomendável usar o *int* pois com ele os cálculos matemáticos são mais

rápidos do que com *float* ou *double*. Diferentemente de outras plataformas, no Arduino, o *double* não possui aumento nos dígitos de precisão. (ARDUINO, [s.d])

É possível converter valores entre tipos, basta usar a função tipo (valor). As funções de conversão são: *char()*, *byte()*, *int()*, *word()*, *long()* e *float()*. Por exemplo converter um valor *int* em *float*, fica *float(15)*. A saída será: 15.0000.

4.4.2.2 Escopo e Modificadores de Tipo de Acesso

Segundo Evans (2011), o escopo da variável é a parte do código onde ela será declarada. Schildt (1996, p. 20) argumenta que há três locais básicos onde as variáveis podem ser declaradas: “dentro das funções, na definição dos parâmetros das funções e fora de todas as funções. ”

As variáveis dentro de funções são conhecidas como variáveis locais, elas apenas funcionam dentro do bloco em que foram declaradas, sendo que a partir do momento que o bloco não está mais executando a variável é destruída podendo assim, usar uma outra de mesmo nome em blocos diferentes. A vantagem de usar variáveis locais é que a medida que o código vai sendo executado e as elas vão sendo excluídas, a memória utilizada por elas é liberada. Parâmetros das funções são os parâmetros formais. São variáveis que devem ser declaradas quando as funções contêm argumentos. Assim estas podem ser referenciadas como variáveis locais dentro do bloco da função. Variáveis globais são aquelas que estão fora de todas as outras funções, isto proporciona a elas a possibilidade de ser usada em qualquer parte do programa (SCHILDT, 1996).

Os modificadores de tipo de acesso controlam a forma com que as variáveis serão modificadas ou acessadas, um deles é ‘*const*’, variáveis deste tipo não podem ser modificadas pelo seu código, elas são constantes. Desta forma a variável estará protegida, nenhuma função poderá altera-la. Outro tipo é o ‘*volatile*’, que é usado para dizer ao compilador que esta variável só pode ser alterada quando o próprio programa não especifica a mudança, mas sim algo externo ao programa. (SCHILDT, 1996)

4.4.2.3 Constantes

Evans (2011) menciona que além das variáveis constantes que o próprio usuário declara, o ambiente de desenvolvimento Arduino possui suas próprias constantes.

Constantes booleanas, *true* ou *false*. A primeira é definida como qualquer valor inteiro que não seja zero, o qual representa a segunda. Diferentemente das outras constantes, estas são escritas em letras minúsculas. É possível definir uma constante para os pinos digitais através da função *pinMode()*, a eletricidade no pino pode ser alterada. Estas constantes são: *INPUT* (entrada), *INPUT_PULLUP* e *OUTPUT* (saída). Os pinos digitais só podem receber dois valores: *HIGH* e *LOW*, suas definições variam quando o pino está configurado em *INPUT* ou *OUTPUT*. Quando o pino estiver configurado como *INPUT* o Arduino irá especificar como *HIGH* se houver uma voltagem maior que 3V no pino, quando a placa for de 5V, ou se a voltagem for maior que 2V numa placa de 3.3V. Quando o pino está configurado como *OUTPUT*, e for configurado como *HIGH* o pino ficará com 5V em placas de 5V e com 3.3V em placas de 3.3V. Um pino *INPUT* será *LOW* quando a voltagem for menor que 3V em uma placa de 5V ou 2V em uma placa de 3.3V. O pino estando configurado como *OUTPUT* e em *LOW* fará com que o pino fique com 0V independente da placa, quer seja de 5V ou de 3.3V. (ARDUINO, [s.d.]

4.4.3 Operadores

Os operadores são caracteres especiais usados a fim de atribuir valores, realizar cálculos, entre outros. Podem ser divididos em algumas classes: operadores aritméticos, relacionais, lógicos e incremento/decremento. (EVANS, 2011)

O operador de atribuição da linguagem é o caractere “=”. Pode ser usado em qualquer lugar do código onde o objetivo seja atribuir um valor ou outra expressão qualquer à uma variável. A forma padrão é a seguinte:

```
nomeVariavel = expressao;
```

Onde expressão pode ser desde um valor qualquer até o que for preciso, como uma variável por exemplo. (SCHILDT, 1996)

4.4.3.1 Operadores Aritméticos

São utilizados a fim de realizar operações aritméticas básicas como adição (+), subtração (-), multiplicação (*), divisão (/) e módulo (%) entre dois valores do mesmo tipo de dado. (ARDUINO, [s.d.])

Evans (2011) menciona que os operadores são usualmente expressos da seguinte forma:

```
nomeVariavel = x + y;
```

Onde x e y são valores quaisquer. Os cálculos começam pela esquerda, ou seja, x + y, caso houvesse outro valor após o y seria feito primeiro x + y e então o resultado deste mais o próximo valor. Variáveis podem ser incluídas nas operações desde que estejam carregando um valor, como neste exemplo:

```
novaVariavel = nomeVariavel * 3;
```

A respeito da ordem das operações Schildt (1996, p. 44) declara: “Parênteses forcem uma operação, ou um conjunto de operações, a ter um nível de precedência maior. ” Portanto os níveis de precedência são: (); *, /, %; + e -.

4.4.3.2 Incremento e Decremento

Estes operadores são para adicionar (++) e subtrair (--) um. Podem ser usados antes ou depois do operando. Por exemplo, x++ ou ++x, significam que está se adicionando mais um ao x, portanto se x equivale a dois, x++ é igual a três. Porém há uma diferença, quando o incremento ou decremento é colocado antes do operando, o Arduino C irá executar a operação antes do valor ser usado, o contrário acontece também quando colocado após o operando, neste caso a operação acontecerá depois do valor ser usado, por exemplo:

```
x = 2;
```

```
y = ++x;
```

Aqui o y recebe o valor 3. Colocando após o operando ficaria:

```
x = 2;
```

```
y = x++;
```

Neste caso y recebe o valor 2, mas irá ter valor 3 quando for usado. Nos dois casos o y recebe 3, a diferença está no momento onde isso acontece. (SCHILDT, 1996)

4.4.3.3 Operadores Relacionais

Schildt (1996, p. 44) define operadores relacionais da seguinte maneira: “[...] refere-se às relações que os valores podem ter uns com os outros.”

Estes operadores são menores em precedência do que os operadores aritméticos, e definem se a expressão é verdadeira ou falsa, sendo que 1 representa uma verdade e 0, falso. Os operadores são: > (maior que), >= (maior ou igual que), < (menor que), <= (menor ou igual que), == (igual) e != (diferente).

Normalmente é usado, segundo Arduino ([s.d.]), para testar certas condições a serem alcançadas como por exemplo em um *if*:

```
if (x > 10)
```

```
{
```

```
  x = 15;
```

```
}
```

4.4.3.4 Operadores lógicos

Os operadores lógicos trabalham em busca de como ligar os operadores relacionais, de forma a combinar expressões que os usam. Existem três operadores lógicos básicos: `&&` (e), `||` (ou) e `!` (não). (SCHILDT, 1996)

O resultado da expressão também se dá em verdadeiro ou falso, o operador `&&` significa que somente será uma expressão verdadeira se todas as comparações forem verdadeiras:

```
10 > 5 && 2*3 >5;
```

Aqui o valor retornado é *true* pois ambas as comparações são verdadeiras, se uma delas fosse falsa o retorno seria *false*. `||` é usado quando pelo menos uma das comparações precisa ser verdadeira e `!` será verdadeiro em uma comparação falsa. (ARDUINO, [s.d.])

4.4.4 Estruturas de controle

Há dois tipos de expressões que controlam a estrutura de um programa em Arduino C: estruturas condicionais, onde é verificado se a condição seja verdadeira ou não, e estruturas de iteração que são rotinas realizadas um número qualquer de vezes até a condição ser falsa. (EVANS, 2011)

4.4.4.1 Estruturas Condicionais

Schildt (1996) declara que existe dois tipos de comandos para controle a partir de uma condição: o *if/else* e *switch*. O *if/else* é declarado assim:

```
if (condição) comando;
```

```
else comando2 ;
```

If significa 'se', neste caso lê-se: se condição for verdadeira, então execute o comando. Senão (*else*), execute comando dois. O *else* não é obrigatório. Mas quando

usado, apenas um dos dois comandos ou blocos de código será executado. É possível colocar um *if* dentro de outro e ainda usar a escada *if-else-if*, onde o programa irá verificar as condições uma após a outra até encontrar uma que seja verdadeira. (SCHILDT, 1996)

O outro tipo, declara Evans (2011), é o *switch*, que compara uma expressão com uma lista de casos. Aqui ele passa por todos independente do correspondente ser a primeira ou a última opção. A declaração básica é a seguinte:

```
switch (expressão) {  
  
    case constante1:  
  
        comandos;  
  
        break;  
  
    case constante2:  
  
        comandos;  
  
        break;  
  
    case constante3:  
  
        comandos;  
  
        break;  
  
}
```

A expressão é comparada e cada uma das constantes declaradas no comando *case*. Quando é encontrada uma equivalente, os comandos são executados até o comando *break*, saindo assim do *case* atual. Há três princípios básicos a respeito do *switch*: ele só opera testes de igualdade, não pode haver duas constantes *case* iguais e se a constante for um caractere é automaticamente convertida para um valor inteiro. Assim como *if* é possível aninhar blocos *switch*. (SCHILDT, 1996)

4.4.4.2 Estruturas de Iteração

São também conhecidos como laços. Sua função é fazer com que o código se repita até que determinada condição seja satisfeita. São três comandos que se dividem em dois tipos, o pré-definido ou o em aberto.

O laço *for* é do tipo pré-definido, ele é declarado no topo do laço, portanto existe a possibilidade desse laço não ser executado caso as condições não sejam satisfeitas. Evans (2011) argumenta que o diferencial do *for* para os outros laços é a possibilidade de escolher um número específico de vezes que o código será executado. Ele é declarado da seguinte maneira:

```
for (inicialização; condição; incremento) { comandos; }
```

A inicialização, normalmente é onde a variável que irá controlar o laço é preenchida com um valor. Na condição esse valor é comparado por meio dos operadores relacionais definindo-se o seu termino e o incremento muda a variável de controle a cada vez que o código é repetido. (SCHILDT, 1996)

While é um laço em aberto, funciona basicamente como o *for* com a diferença de ser apenas uma condição invés de três expressões. Repetindo-se enquanto a condição *for* verdadeira. Também é inicializada no começo do laço. (SCHILDT, 1996)

```
while (condição) { comandos; }
```

Evans (2011) menciona que quase não há diferença entre o *for* e o *while*, porém o *while* é preferível em situações especiais quando uma rotina deve ser executada. Para fazer o bloco ser executado pelo menos uma vez, existe o *do-while*, que executa a comparação ao fim do bloco. A declaração usual é:

```
do { comando} while(condição);
```

4.4.5 Arrays

Arrays, de acordo com Evans (2011), são listas preenchidas por diversas variáveis de um mesmo tipo de dado. Cada elemento possui um índice que pode ser

acessado pelo programa, o mais baixo representa a primeira variável e o mais alto, a última. (SCHILDT, 1996)

Utilizam grande parte da memória do microcontrolador, em Arduino C, os *arrays* são usados em listas de valores de sensores, números de pinos, strings, etc. A forma mais simples de declarar-se um *array*, é assim:

```
int array[3];
```

O valor, ou variável, dentro dos colchetes representa o índice e o tamanho do array, ou seja, de quantas variáveis a lista será formada. Neste exemplo, foram criadas três variáveis. A contagem do índice começa do zero, logo, as variáveis são: array[0], array[1] e array[2]. É possível preencher o *array* logo em sua declaração, deste modo:

```
int array[] = { 5 , 5 , 3 }
```

Neste caso não é necessário passar o tamanho do *array* pois o Arduino C o reconhece pela quantidade de valores instanciados. (EVANS, 2011)

Evans (2011) declara, que apesar de não ser necessário na maioria dos projetos, o Arduino C suporta *arrays* multidimensionais, isto é, *arrays* com mais de um índice.

4.4.6 Funções

As funções são blocos de código onde os comandos são executados. A forma geral de declaração é a seguinte:

```
tipo_do_retorno nome_da_função(parametros) { comandos }
```

O tipo do retorno é um tipo de dado qualquer que a função irá retornar. Caso não seja especificado o Arduino C reconhece tipo *int*. os parâmetros são variáveis que uma função pode receber, a fim de trabalhar com eles, devem ser separados por vírgula e declarados normalmente. Não é obrigatório possuir parâmetros, no entanto é ter parênteses. (SCHILDT, 1996)

Arduino C tem diversas funções prontas, que serão analisadas a seguir.

4.4.6.1 Funções de Entrada/Saída Digital

A função `pinMode()`, especifica se o pino será de entrada ou saída. Ao declarar, os parametros são: o valor do pino e o modo (*INPUT*, *OUTPUT* ou *INPUT_PULLUP*), logo temos, `pinMode(5, INPUT)` por exemplo.

O `digitalWrite()`, escreve os valores *HIGH* ou *LOW* para um pino digital, que de preferência tenha sido especificado pelo `pinMode()`. Sua declaração se dá semelhantemente acima, o valor do pino e o valor *HIGH/LOW*: `digitalWrite(5, LOW)`.

Por sua vez, `digitalRead()`, lê o valor de um pino digital, especificado pelo `digitalWrite()` e retorna-o em *HIGH* ou *LOW*. Basta pôr como parâmetro o valor do pino: `digitalRead(5)`. (ARDUINO, [s.d])

4.4.6.2 Funções de Entrada/Saída Analógica

Duas funções: `analogWrite()` e `analogRead()`. A primeira escreve um valor analógico no pino, pode ser usado para acionar um motor com várias velocidades, quando chamada, ela envia uma onda constante até a próxima chamada (no mesmo pino) desta função ou da `analogRead()`. Os parametros são: o pino e o ciclo de trabalho, 0 para desligado e 255 para sempre ligado. Portanto, `analogWrite(5, 255)`, significa que o pino 5 fica sempre ligado. A segunda função, `analogRead()`, lê o valor do pino analógico especificado.

4.4.6.3 Outras funções

Existem ainda funções matemáticas de potenciação, `pow(valor, potencia)`, que retorna a potenciação do primeiro parâmetro. De raiz quadrada, `sqrt(valor)`, retornando a raiz quadrada do valor. Funções que calculam o seno, coseno e tangente, `sin()`, `cos()` e `tan()` respectivamente. Há ainda a função `random()`, que retorna um valor randomizado.

A função de tempo `delay()`, pausa o programa por um tempo determinado em milissegundos, se for necessário microssegundos, existe a `delayMicroseconds()`. A

função `millis()`, calcula quanto tempo que se passou desde que o programa foi iniciado. (ARDUINO, [s.d])

4.5 Vantagens e desvantagens

O Arduino é indicado para a maioria das pessoas de acordo com Codeduino ([s.d.]), tendo recursos capazes de atender as necessidades dos iniciantes, o controlador utilizado pela UNO é o chip ATmega328 tendo a opção de ser alimentado por bateria, adaptador AC-to-DC ou diretamente da USB, se o uso for contínuo da bateria pode ser rápido o esgotamento da carga.

Se a alimentação for abaixo do normal os pinos da placa vão obter menos que 5 volts, a alimentação indicada é 9V ou 12V para não aquecer o regulador de tensão. O Uno possui 6 pinos analógicos que tem uma resolução de 10 bits sendo eles RX/TX e 1024 valores diferentes quando se ler, 40mA é o que cada pino pode desenhar. Sendo mais preciso que o MIDE, o Uno é 10 vezes mais exato. O código do programa que é compilado e enviado para a placa ficando disponível na memória flash, essa memória tem 32 KB já a SRAM tem 1KB sendo a memória de rascunho ou trabalho. As vantagens de acordo com Codeduino (2013), é ser muito barato, fácil de se utilizar, variedade de *shields* extras com diferentes recursos, muitas fontes de códigos e projetos na internet. Jimbo ([s.d.]) informa que praticamente todas as placas Arduino mesmo sendo de diferentes modelos, são programadas pelo Arduino IDE, esse *software* proporciona que o usuário visualize o código, faça upload e escreva. Por existir diferentes modelos os números de entradas e saídas podem mudar, tensão de operação e a velocidade. Uma bateria de 3.7V pode ser utilizada em algumas placas já outras o requisito mínimo é de 5V, outras é necessário a compra separada de uma interface de programação (*hardware*).

A desvantagem de acordo com Codeduino (2013) sobre o Arduino Uno, é que a placa possui poucos pinos e se for trabalhar em um projeto que requer muitos pinos pode-se encontrar dificuldades para realizar o trabalho. Outro recurso que pode atrapalhar no desenvolvimento dos projetos é a capacidade de armazenamento da placa, por ela não ter suporte a entrada de cartão de memória impedindo assim o armazenamento de um código um pouco mais complexo.

5 PROJETO NAVBLIND

O Arduino nos encantou por ser uma placa com muitos recursos e com um custo acessível, sendo possível qualquer pessoa adquirir essa maravilhosa ferramenta. No entanto, o interesse em ajudar pessoas com deficiência física surgiu e procuramos desenvolver algo que pudesse ajudar, e no processo de pesquisa encontramos o projeto do Hoefer, que apresentou um projeto de acessibilidade interessante, sendo este um aperfeiçoamento de um outro projeto realizado por Polymythic.

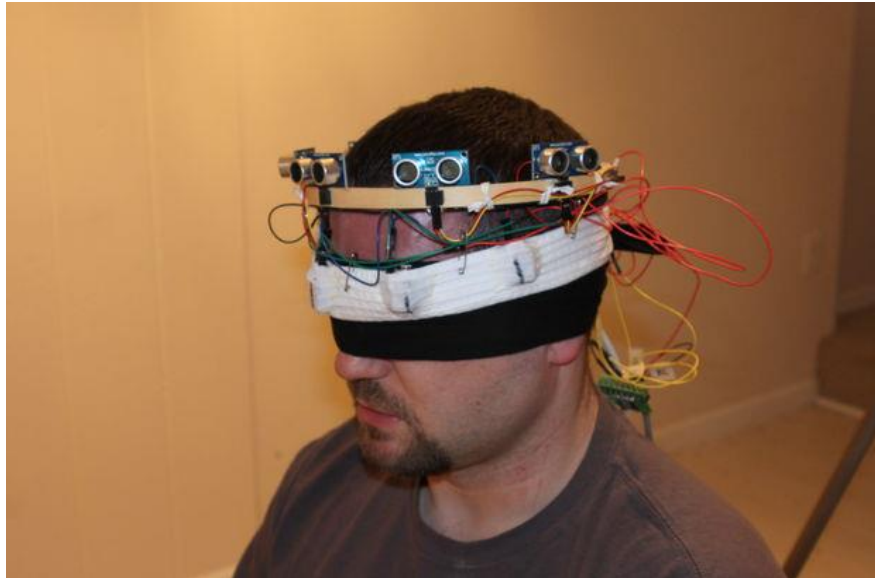
O projeto realizado por Polymythic foi inspirado em um episódio de Stan Lee's Superhumans, onde um cego usou vários clips para transmitir som como um morcego para se localizar. Com isso surgiu a ideia de que os cegos podiam se movimentar sem a ajuda de cão guia e bengala, então decidiu utilizar sensores com resultado de resposta motores que vibram colocados sobre a cabeça do usuário, como pode ser visto na figura 20. Polymythic ([s.d.]) continua dizendo:

A medida que uma pessoa se aproxima de um objeto, a intensidade e frequência da vibração aumentaria - é diretamente proporcional à distância de um objeto. Se uma região não dá retorno, então seria seguro continuar nessa direção. Eu chamo minha submissão a HALO - o Localização Assistida de Obstáculos Haptic. Eu acredito que isso pode servir muito útil para Deficientes Visuais, a fim de ter a liberdade para mover-se livremente com as mãos livres, sem o auxílio de uma bengala ou cão guia. (POLYMYTHIC, [s.d.], tradução livre)⁷

O projeto desenvolvido que utiliza componentes e sensores acessíveis a qualquer um, possibilitou a um cego evitar colisões com objeto aos seus arredores. (POLYMYTHIC, [s.d.])

Figura 20 - Projeto HALO

⁷ As a person gets closer to an object the intensity and frequency of the vibration would increase – it's directly proportional to the distance of an object. If a region was lacking feedback, then it would be safe to proceed in that direction. I call my submission the H.A.L.O. - the Haptic Assisted Locating of Obstacles. I believe this can serve very useful for the visually impaired to have the freedom to possibly move about hands-free without the assistance of a cane or seeing eye dog.



Fonte: POLYMYTHIC, 2015.

O HALO de Polymythic, foi um grande avanço por estar utilizando ferramentas com capacidade de auxiliar pessoas com deficiência. Mas por ser um projeto em fase de teste, houve algumas falhas que Hoefer (2011) identifica:

Os obstáculos mais perigosos não estão à altura da cabeça. Móveis e a maioria das outras coisas que podem ser tropeçados estão no nível da cintura ou abaixo. Motores vibrando presos em seu crânio irão incomodar rapidamente (HOEFER, 2011)⁸.

Hoefer (2011) diz que procurou soluções para o protótipo realizado por Polymythic adquirindo sensores que são mais precisos e que não necessariamente eles deveriam ficar localizados na cabeça, sendo que isso pudesse levar a um preconceito. O preço tem que ser acessível à população ainda mais se está relacionado com acessibilidade, com isso o custo no total do protótipo realizado por Hoefer ficou por US \$ 65 USD.

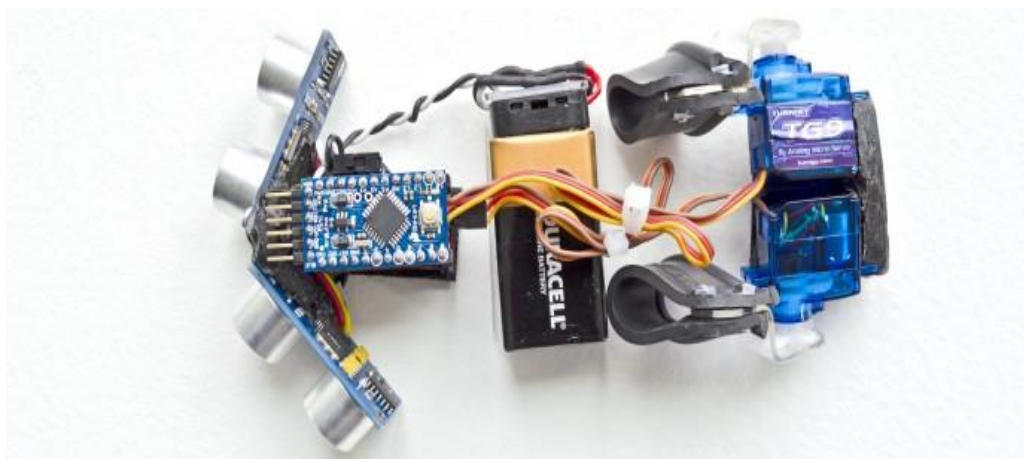
O aperfeiçoamento feito por Hoefer possibilitou a ferramenta ficar na melhor solução encontrada que foi a mão, propiciando ao usuário aponta-lo a qualquer local que esteja ao seu redor e assim não interferindo em seus outros membros.

⁸ The most dangerous obstacles are not at head level. Furniture and most of the other things that can be tripped over and stubbed on are waist level or lower. Vibrating motors stuck on your skull will drive you insane quickly.

É montado no pulso e detecta objetos a cerca de 1 polegada (2 cm) a 10 pés (3,5 m). Ele tem tempo de resposta geralmente rápido (frações de segundo) para navegar rapidamente em ambientes complexos. Ele foi projetado para ajudar pessoas com a visão prejudicada a navegar em ambientes complexos⁹.

Hoefer tentou utilizar outros sensores mas percebeu que o Ultrassom atendia as necessidades exigidas, já os motores de vibração são um problema e foi preferível não utilizar por fazerem barulhos e dificultar a audição dos deficientes visuais que é essencial para eles. Depois de pesquisar por várias ferramentas que pudesse auxiliá-lo, encontrou os motores onde ele realizou alguns ajustes com almofadas na extremidade, além de serem baratos e pequenos, possibilitou precisão.

Figura 21 - *Hardware* do Tacit



Fonte: HOEFER, 2015.

De acordo com Hoefer (2011) esse não é um projeto difícil de se estar fazendo, mas é necessário realizar a leitura com cuidado antes de começar, o diagrama e circuitos estão disponíveis no site e estão sob a licença *Creative Commons Attribution 3.0 Unported* juntamente com o código fonte sendo um projeto *open source* disponível para todos.

Baseado nestes projetos, foi desenvolvido um protótipo com algumas adaptações, chamado de Navblind, o qual iremos apresentar a seguir. O desenvolvimento do projeto utilizou-se componentes do HALO, sendo adaptado na cabeça por não termos componentes necessário para utilizar na mão, igual ao do

⁹ It's wrist mounted and senses objects from about 1 inch (2 cm) to 10 feet (3.5m). It has generally fast response time (fractions of a second) to quickly navigate complex environments. It's designed to help a vision impaired person to navigate complex environments.

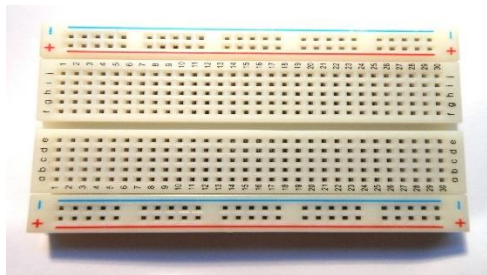
Tacit. O nosso objetivo foi criar funções que pudessem melhorar o projeto já desenvolvido. Utilizou-se uma placa Arduino Uno R3 e um sensor ultrassônico programado em três níveis de pressão.

5.1 Componentes usados

Para este projeto usamos os componentes a baixo:

- Protoboard (Figura 22)

Figura 22 - Protoboard



Fonte: MERCADO LIVRE, 2015.

- HC-SR04 (Figura 23)

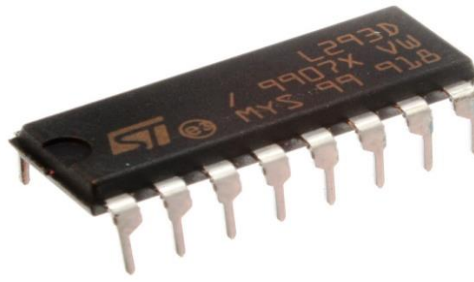
Figura 23 - HC-SR04



Fonte: UPGRADE INDUSTRIES, 2015.

- L293D (Figura 24)

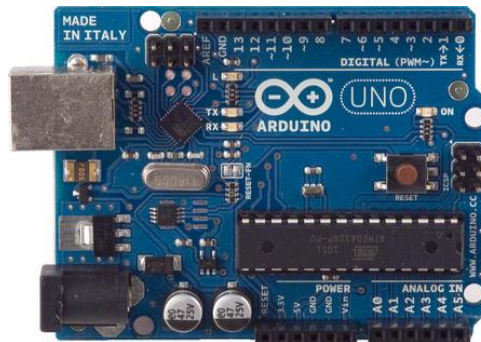
Figura 24 - L293D



Fonte: ELECTROSOME, 2015.

- Arduino Uno R3 (Figura 25)

Figura 25 - Arduino Uno R3



Fonte: ARDUINO, 2014.

- Led 10 mm (Figura 26)

Figura 26 - Led 10 mm



Fonte: FIND PARTS, 2015.

- Motor DC (Figura 27)

Figura 27 - Motor dc



Fonte: MERCADO LIVRE, 2015.

- Bateria 9v (Figura 28)

Figura 28 - Bateria 9v



Fonte: INGENIU, 2015.

- Resistor 10 k (Figura 29)

Figura 29 - Resistor 10 k



Fonte: SPIKENZIELABS, 2015.

- Momentary Pushbutton Switch (Figura 30)

Figura 30 - Momentary Pushbutton



Fonte: SPARKFUN, 2015.

- Buzzer DC 5V (Figura 31)

Figura 31 - Buzzer DC 5v



Fonte: WEBTRONICO, 2015.

- Fio Jumper

Figura 32 - Fio Jumper



Fonte: BANANA ROBOTICS, 2015.

5.2 Análise dos componentes

Os componentes necessários para o desenvolvimento do Navblind são: Arduino Uno R3, protoboard, HC-SR04, L293D, Motor DC, *momentary pushbutton switch*, *buzzer* DC, bateria 9V, resistor 10 k e um led 10mm.

A placa Arduino é onde acontece o processamento de todas as entradas e saídas de energia que determina quais componentes serão utilizados e em que momento isso deve acontecer. O *protoboard* possibilita ligar os componentes entre si.

O HC-SR04, é o sensor ultrassônico responsável por identificar os objetos por meio de ondas sonoras, informando ao Arduino a distância entre ele e os objetos. Assim o código executado pelo Arduino consegue calcular a intensidade da rotação

do motor DC, baseando-se na premissa de que quanto mais perto o objeto estiver do sensor maior será a intensidade e quanto mais longe, menor.

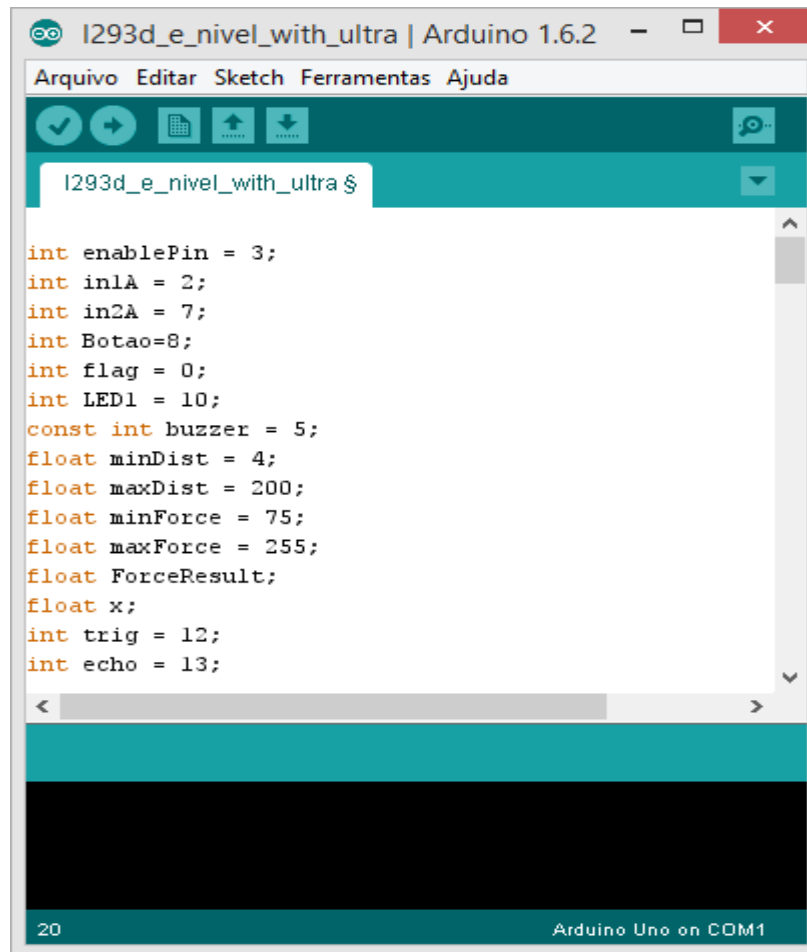
Esta intensidade que determina a rotação do motor é controlada pelo L293D, um circuito integrado que envia os pulsos de energia para o motor DC, podendo até controlar a direção da rotação, se para esquerda ou direita. Este motor é alimentado pela bateria 9V.

O *momentary pushbutton switch* serve para alterar os níveis de rotação, enquanto isso o *buzzer* DC apita a cada nível alterado, este é um modo que os deficientes visuais podem identificar a mudança, através de sons. Para conectar todas essas peças foi utilizado o fio jumper, o resistor serviu para estabilizar a energia para que algumas peças não queimassem.

5.3 Análise do código

Neste capítulo será analisado o código fonte desenvolvido para o NavBlind utilizando o IDE oficial do Arduino e escrito na linguagem Arduino C. Desde a declaração das variáveis até o cálculo necessário para identificar os objetos, entre outras funções. A figura 33 apresenta as variáveis que especificam os pinos de entrada para que ocorra a comunicação, as mesmas sendo declaradas e instanciadas.

Figura 33 - Declaração de Variáveis



```
I293d_e_nivel_with_ultra | Arduino 1.6.2
Arquivo  Editar  Sketch  Ferramentas  Ajuda

I293d_e_nivel_with_ultra $

int enablePin = 3;
int in1A = 2;
int in2A = 7;
int Botao=8;
int flag = 0;
int LED1 = 10;
const int buzzer = 5;
float minDist = 4;
float maxDist = 200;
float minForce = 75;
float maxForce = 255;
float ForceResult;
float x;
int trig = 12;
int echo = 13;
```

20 Arduino Uno on COM1

Na figura 34 é apresentada a função `setup()`, que inicializa as variáveis especificando se o pino será de I/O e também a taxa de dados necessária para haver comunicação entre o Arduino e o computador. A função `setup()` é executada apenas na inicialização da placa. O `pinMode()` define os pinos declarados anteriormente com entrada (input) e saída (output).

Figura 34 - Função setup()



A função `loop()` é inicializada após a `setup()`, e sua execução será repetida até o momento em que a placa for desligada ou reiniciada. Essa função começa com verificações de uma variável que informa o nível, cada nível tem um diferencial que é a distância de identificação, que o sensor deve captar de objetos que se encontram ao seu alcance. Para que seja possível o usuário do NavBlind identificar o nível atual é acionado o *buzzer* que emite diferentes sons, neste momento também é acionado o led e ocorre um atraso de 5 segundos. Logo após o acionamento volta para o início da função `loop()` e aguarda até que o *momentary push button* seja pressionado novamente.

Figura 35 - Função loop()

```

I293d_e_nivel_with_ultra | Arduino 1.6.2
Arquivo Editar Sketch Ferramentas Ajuda

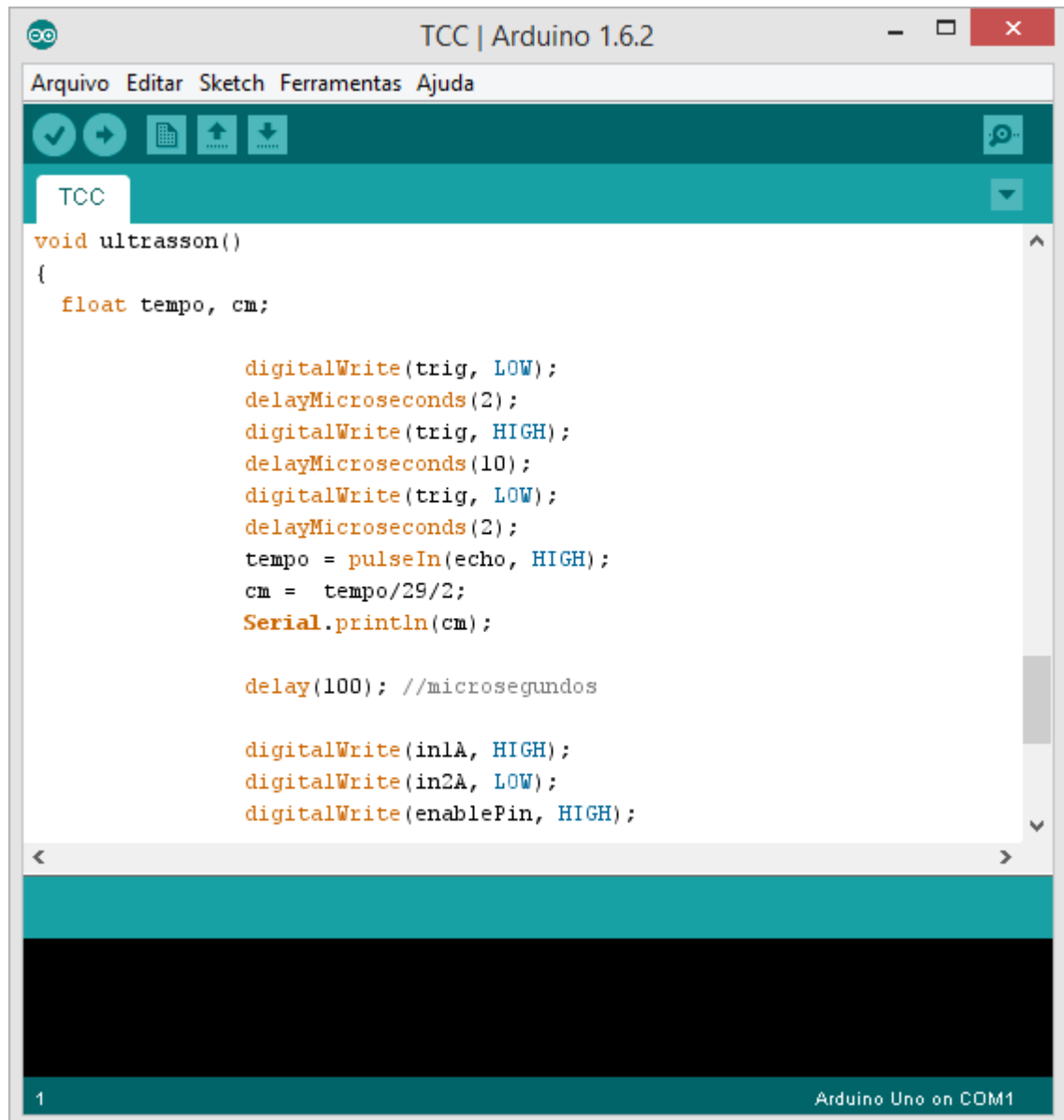
I293d_e_nivel_with_ultra $
void loop()
{
    if(flag!=0)
        ultrasson();
    if(digitalRead(Botao)==1) {
        if(flag == 0)
        {
            maxDist = 200;
            ligarBuzzer(500);
            digitalWrite(LED1,HIGH);
            flag=1;
            delay(500);
        }else if(flag==1)
        {
            maxDist = 300;
            ligarBuzzer(400);
            ligarBuzzer(400);
            digitalWrite(LED1,HIGH);
            flag=2;
            delay(500);
        }else if(flag==2)
        {
            maxDist = 400;
            digitalWrite(enablePin, LOW);
            digitalWrite(LED1,LOW);
            ligarBuzzer(200);
            ligarBuzzer(200);
            ligarBuzzer(300);
            flag=0;
        }
    }
}

```

48 Arduino Uno on COM1

A figura 35 mostra a função `ultrasson()`, que é a principal do projeto, porque ela é responsável por definir a velocidade em que o ultrassom vai funcionar e fazer o cálculo para adquirir a distância em que o objeto se encontra, além de determinar a força que o L239D deve enviar para o motor.

Figura 36 - Função ultrasson()

The image is a screenshot of the Arduino IDE interface. The title bar at the top reads 'TCC | Arduino 1.6.2'. Below the title bar is a menu bar with 'Arquivo', 'Editar', 'Sketch', 'Ferramentas', and 'Ajuda'. Underneath the menu bar is a toolbar with icons for checking, undo, redo, saving, and uploading. A tab labeled 'TCC' is active. The main text area contains the following C++ code:

```
void ultrasson()
{
    float tempo, cm;

    digitalWrite(trig, LOW);
    delayMicroseconds(2);
    digitalWrite(trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig, LOW);
    delayMicroseconds(2);
    tempo = pulseIn(echo, HIGH);
    cm = tempo/29/2;
    Serial.println(cm);

    delay(100); //microsegundos

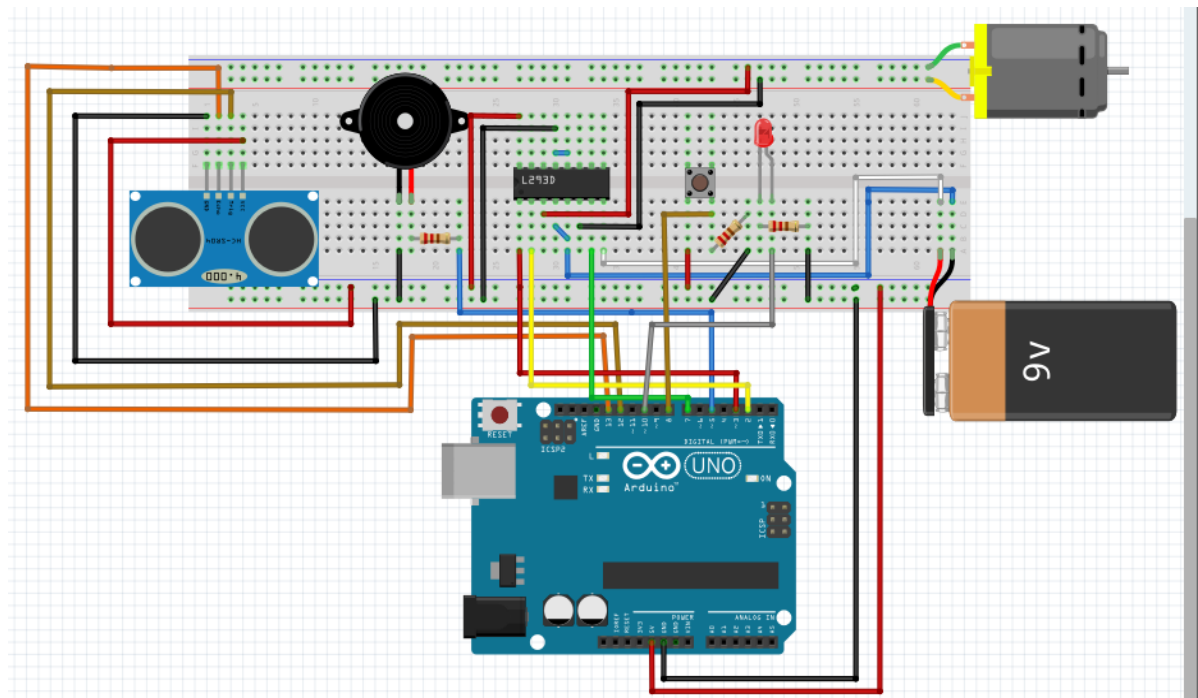
    digitalWrite(in1A, HIGH);
    digitalWrite(in2A, LOW);
    digitalWrite(enablePin, HIGH);
}
```

The bottom status bar shows '1' on the left and 'Arduino Uno on COM1' on the right.

5.4 Modelo

Desenvolvemos o modelo do protótipo do NavBlind para uma melhor identificação do projeto como visto na figura 37.

Figura 37 - Modelo do protótipo NavBlind



6 CONSIDERAÇÕES FINAIS

Concluimos que os deficientes visuais enfrentam muitas dificuldades, de locomoção e identificação de objetos. Apesar de terem sido excluídos da sociedade por muito tempo, possuem os mesmos direitos que todas as pessoas, portanto as tecnologias precisam ser desenvolvidas a fim proporcionar facilidades a eles, de acordo com o desenho universal que visa facilitar ao indivíduo sua independência.

Assim o projeto desenvolvido utilizando o Arduino, NavBlind, pode ser utilizado como uma tecnologia assistiva pois segue os princípios do desenho universal. A escolha do Arduino se deve ao fato de ser uma tecnologia barata, de fácil utilização, compacta, e com diversas ferramentas que atendem a inúmeras possibilidades.

Com análise do projeto feito por Polymythic ([s.d.]), foi possível observar as desvantagens de utilizar esse protótipo na região da cabeça, porque proporciona um desconforto por se estar utilizando motores nessa região, dificulta a audição do usuário e objetos pequenos são difíceis de serem identificados. Mas com as adaptações feitas por Hoefer (2011) houve uma melhor identificação do ambiente, sendo um tipo de bengala digital, disponibilizando mais conforto.

As adaptações realizadas no projeto que desenvolvemos foram com a intensão de atender as necessidades dos deficientes, a primeira foi criar opções de níveis de identificação do sensor, o nível 1 identifica até 2 metros, nível 2 até a 3 e o nível 3 desliga o aparelho. Essa opção possibilita quando for necessário desligar o NavBlind, ter um nível de alcance maior de identificação da área. Para que essa mudança de nível aconteça o usuário aperta um *button* e será acionado um *buzzer* alertando que o NavBlind está em outra configuração.

Observamos que toda tecnologia visa ajudar o ser humano, facilitando a execução de suas tarefas. Assim, elas podem também servir como ferramentas de acessibilidade que auxiliem os portadores de deficiência, inclusive visual. O Arduino mostrou-se facilmente adaptável aos deficientes visuais, pois possui ferramentas de acesso a informação que independem da visão como o *buzzer* que é percebido pela audição e o motor DC pelo tato.

O objetivo final é disponibilizar mais informações para o portador, já que sem a visão fica complicado a identificação a não ser por contato físico. Uma das possíveis implementações futuras é a adaptação do NavBlind com o Raspberry pi, um computador do tamanho de um cartão de crédito utilizando uma câmera e a biblioteca OpenCV. Essa adaptação será possível mapear objetos que estejam ao redor do portador, informando quais são esses, através de áudio.

7 REFERÊNCIAS

ARDUINO. Disponível em: <<http://www.arduino.cc/>>. Acesso em: 18 ago. 2014.

BANANA ROBOTICS. Disponível em: <[https://www.bananarobotics.com/shop/image/cache/data/sku/BR/0/1/0/0/4/BR010040-Male-to-Female-Jumper-Wire-\(25-pack\)/Male-to-Female-Jumper-Wire-600x600.jpg](https://www.bananarobotics.com/shop/image/cache/data/sku/BR/0/1/0/0/4/BR010040-Male-to-Female-Jumper-Wire-(25-pack)/Male-to-Female-Jumper-Wire-600x600.jpg)>. Acesso em: 01 mai. 2015.

BASTOS, T. **Oficina de Robótica**. Disponível em: <<http://www2.ele.ufes.br/~tfbastos/RobMov/ApostilaTeodiano.pdf>>. Acesso em: 10 set. 2014.

BRAIN, M. **How Microcontrollers Work**. Disponível em: <<http://electronics.howstuffworks.com/microcontroller.htm/>>. Acesso em: 19 out. 2014.

BRASIL. **Decreto nº 5.296/04. 2004**. Disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2004-2006/2004/decreto/d5296.htm>. Acesso em: 23 set. 2014.

CAPACITAÇÃO EM ACESSIBILIDADE. Disponível em: <http://www.pessoacomdeficiencia.gov.br/app/sites/default/files/arquivos/%5Bfield_enerico_imagens-filefield-description%5D_72.pdf>. Acesso em: 22 set. 2014.

CODEDUINO. **Which Arduino Should I Buy?**. Disponível em: <<http://codeluino.com/information-and-news/hardware/which-arduino-should-i-buy/>>. Acesso em: 19 març. 2015.

COOK, A. M.; POLGAR, J. M. **Cook and Hussey's assistive technologies: Principles and Practices**. EUA: Elsevier Health Sciences, 1995.

DURFEE, W. **L293 Motor Driver and H-Bridges**. Disponível em: <<http://www.me.umn.edu/courses/me2011/arduino/technotes/dcmotors/L293/L293.html>>. Acesso em 19 out. 2014.

ELECTROSOME. Disponível em: <<https://electrosome.com/wp-content/uploads/2013/05/L293D.jpg>>. Acesso em: 01 mai. 2015.

ENGINEERING, S. **Intel, Stanford Engineering share proud history of technology, talent development**. Disponível em: <<http://engineering.stanford.edu/research-profile/intel-stanford-engineering-share-proud-history-of-technology-talent-development/>>. Acesso em 19 out 2014.

ENTREAMIGOS. **Deficiência visual**. Disponível em: <<http://www.entreamigos.com.br>>. Acesso em: 29 set. 2014.

EVANS, B. **Beginning Arduino Programming**. EUA: Technology in Action, 2011.

EVANS, M.; NOBLE, J.; HOCHENBAUM, J. **Detecção de objetos: Escolhendo um sensor ultrassônico. Arduino em ação**. São Paulo: Novatec, 2013.

FILHO, O. H. B. **Componentes eletrônicos e unidades de medida, conceitos básicos**. 2012. Disponível em: <<http://www.hardware.com.br/tutoriais/componentes-eletronicos-unidades-medida-conceitos-basicos/>> Acesso em: 03 dez. 2014.

FIND PARTS. Disponível em: <<http://findparts.in/v1/app/data/images/ventor/9/96.jpg>>. Acesso em: 01 mai. 2015.

FLICKR. Disponível em: <<https://www.flickr.com/photos/buildcircuit/7886931546/>>. Acesso em: 01 mai. 2015.

GALVÃO FILHO, T. A. **Tecnologia assistiva para uma escola inclusiva: apropriação, demanda e perspectivas**. Bahia, 2009. 346f. Tese (Pós-Graduação, Doutorado). Faculdade de Educação, Universidade Federal da Bahia, Salvador, 2009.

HOEFER, Steve. **Meet The Tacit Project. It's Sonar for the Blind**. Disponível em: <<http://grathio.com/2011/08/meet-the-tacit-project-its-sonar-for-the-blind/>>. Acesso em: 07 abr. 2015.

IBGE. **Censo Demográfico 2010: Características gerais da população, religião e pessoas com deficiência**. Disponível em: <http://www.ibge.gov.br/home/estatistica/populacao/censo2010/caracteristicas_religiao_deficiencia/caracteristicas_religiao_deficiencia_tab_pdf.shtm> Acesso em: 22 set. 2014.

INGENIU. Disponível em: <[http://www.ingeniu.com/images/catalog/product.BAT9VA902.large.jpg](http://www.ingeniu.com/images/catalog/product/BAT9VA902.large.jpg)>. Acesso em: 01 mai. 2015.

JIMBO. **Arduino Comparison Guide**. Disponível em: <<https://learn.sparkfun.com/tutorials/arduino-comparison-guide/>>. Acesso em: 28 març. 2015.

JOHN. **Story and History of Development of Arduino**. 2014. Disponível em: <<http://www.circuitstoday.com/story-and-history-of-development-of-arduino>>. Acesso em: 03 dez. 2014.

KERSCHBAUMER, R. **Engenharia de controle e Automação Microcontroladores**. 2013. Disponível em: <<http://www.luzerna.ifc.edu.br/professor/ricardo/documentos/Apostila%20Microcontroladores%20ECA%202013-1.pdf>>. Acesso em 07 set. 2014.

KUSHNER, D. **The Making of Arduino**. 2011. Disponível em: <<http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino>> Acesso em: 03 dez. 2014.

MCROBERTS, M. **Arduino Básico**. São Paulo: Novatec, 2011.

MARCATTO, M.; COELHO, A. **Desenvolvimento de um sistema de navegação para veículo autônomo**. Disponível em: <<http://www.maua.br/pesquisas/desenvolvimento-de-um-sistema-de-navegacao-para-veiculo-autonomo>>. Acesso em: 07 set. 2014.

MERCADO LIVRE. Disponível em: <http://mlb-s2-p.mlstatic.com/protoboard-400-pontos-14068-MLB3606792313_122012-F.jpg>. Acesso em: 01 mai. 2015.

_____. Disponível em: <
http://img2.mlstatic.com/s_MLB_v_V_f_3896227537_022013.jpg>. Acesso em: 01
 mai. 2015.

MORAES JUNIOR, T. O.; FREITAS, N. C. de S. **Aplicação Domótica em Escala Piloto Controlado por Microcontrolador**. Paraíba, 2010. 8f. Artigo. Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, Paraíba, 2014.

NETO, B.; MONTEIRO, P.; QUEIROGA, S. **Aplicabilidade dos Microcontroladores em Inovações Tecnológicas**. Disponível em
 :<<http://propi.ifto.edu.br/ocs/index.php/connepi/vii/paper/viewFile/2433/2526>>.
 Acesso em: 25 agos. 2014.

POLYMYTHIC. **Haptic Feedback device for the Visually Impaired [Project HALO]**. Disponível em:<<http://www.instructables.com/id/Haptic-Feedback-device-for-the-Visually-Impaired/>>. Acesso em: 01 abri. 2015.

RASKESRON. **L293D Motor Driver IC**. 2013. Disponível em:
 <<http://www.rakeshmondal.info/L293D-Motor-Driver>>. Acesso em: 07 set. 2014.

SCHILDT, H. C, **Completo e Total**. São Paulo: Makron Books, 1996.

SPIKENZIELABS. Disponível em: <
[http://www.spikenzielabs.com/Catalog/images/medium/LabsImages/RES10K_MED.j](http://www.spikenzielabs.com/Catalog/images/medium/LabsImages/RES10K_MED.jpg)
[pg](http://www.spikenzielabs.com/Catalog/images/medium/LabsImages/RES10K_MED.jpg)>. Acesso em: 01 mai. 2015.

SAPKOTA, S.; **Simple ultrasonic range finder using HC-SR04**. Disponível em:
 <<http://www.buildcircuit.com/simple-ultrasonic-range-finder-using-hc-sr04/>>. Acesso
 em: 08 set. 2014.

SASSAKI, R. K. Inclusão: o paradigma do século XXI. **Revista da Educação Especial**, Brasília, v.1, n.1, p.19-23, out. 2005.

SONZA, A. P. (Org.). **Acessibilidade e tecnologia Assistiva: Pensando a inclusão sociodigital de Pessoas com necessidades especiais**. Rio Grande do Sul: s.n., 2013.

SPARKFUN. Disponível em: < <https://cdn.sparkfun.com//assets/parts/2/6/2/9/09190-03-L.jpg>>. Acesso em: 01 mai. 2015.

UPGRADE INDUSTRIES. Disponível em:
 <http://www.upgradeindustries.com/media/img/hi_res/hcsr04_hires.jpg >. Acesso em:
 01 mai. 2015.

WEBTRONICO. Disponível em: <
http://www.webtronico.com/image/cache/data/produtos/buzzer_2_1-500x500.jpg>.
 Acesso em: 01 mai. 2015.

ZANCO, W. S. **Microcontroladores PIC 16F628A/648A: Uma Abordagem Prática e Objetiva**. São Paulo: Érica, 2005.