PROTÓTIPOS EDUCACIONAIS

Utilizando o Arduino para o aprendizado de programação inicial

Kassio P. Schaider
Rafael S. Gomes
Igor C. Pulini
Renan O. Rios

Edifes

Protótipos Educacionais

Utilizando o Arduino para o Aprendizado
de Programação Inicial

Kassio Pereira Schaider
Rafael dos Santos Gomes
Igor Carlos Pulini
Renan Osório Rios
(Organizadores)

Protótipos Educacionais

Utilizando o Arduino para o Aprendizado de Programação Inicial

Edifes
Vitória, 2018

Editora do Instituto Federal de Educação,
Ciência e Tecnologia do Espírito Santo
R. Barão de Mauá, nº 30 – Jucutuquara
29040-689 – Vitória – ES
www.edifes.ifes.edu.br | editora@ifes.edu.br

Produção editorial
Revisão de texto: Roberta Patrocínio de Amorim
Projeto Gráfico: Assessoria de Comunicação Social do Ifes
Diagramação e epub: Know-How Desenvolvimento Editorial
Capa: Thalyson da Silva Cordeiro
Imagem de capa: Thalyson da Silva Cordeiro

Dados Internacionais de Catalogação na Publicação (CIP)

P967 Protótipos educacionais : utilizando o Arduino para o aprendizado de programação inicial / Kassio Pereira Schaider... [et al.]. – Vitória, ES : Edifes, 2018.
142 p. : il.

Vários autores.
ISBN: 978-85-8263-318-2 (e-book).

1. Arduino (Controlador programável). I Título.

CDD 22 – 621.381958

Bibliotecária Rossanna dos Santos Santana Rubim – CRB6- ES 403

Agradecimentos

Agradecemos ao nosso professor orientador Igor Carlos Pulini e ao coorientador Renan Osório Rios pelos inestimáveis ensinamentos que possibilitaram o desenvolvimento da pesquisa, a escrita do livro, artigos e palestras, além de nos apoiarem de todas as formas possíveis durante o trabalho.

Agradecemos ao professor Allan Francisco Forzza Amaral pela revisão que contribui bastante na finalização do livro. Também agradecemos ao Thalyson da Silva Cordeiro pela bela ilustração da capa, a Coordenadoria de Pesquisa pelo apoio ao projeto, ao IFES Campus Colatina pela bolsa de pesquisa e a construção do laboratório de pesquisa (LIA) e a Edifes pela revisão final e publicação da obra.

E somos enormemente gratos a todos os professores, alunos e servidores do IFES que nos ajudaram direta e indiretamente antes, durante e depois do projeto.

Por fim agradecemos a nossos familiares e amigos por acreditarem no potencial que tínhamos quando entramos no IFES, onde mesmo nos momentos mais difíceis não deixaram que desanimássemos. Foram fundamentais para a realização desse trabalho.

A todos, o nosso mais profundo obrigado, sem vocês esse projeto não teria sido possível!

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Abreviaturas

A	Ampère
Ah	Ampère-hora
AM	Modulação em Amplitude
CPU	Central Processing Unit
DC	Direct Current
DIL	Dual-In-Line
GND	Ground
ICSP	In-Circuit Serial Programming
IDE	Integrated Development Enviroment
mA	Miliampère
mAh	Miliampère-hora
PWM	Pulse-Width Modulation
RAM	Random Acess Memory
RF	Rádio Frequência
USB	Universal Serial Bus
V	Volts
SI	Sistema Internacional de Unidades

CAPÍTULO 1

ARDUINO

O capítulo apresenta os conceitos necessários para utilização do Arduino na aprendizagem da programação. No decorrer do livro é necessário, eventualmente, consultar este capítulo para retirar alguma dúvida.

O capítulo possui os seguintes tópicos:

- O que é o Arduino?
- Componentes Básicos do Arduino
- Circuitos Eletrônicos
- A Linguagem C do Arduino
- IDE do Arduino
- Bibliotecas

1.1 O QUE É ARDUINO?

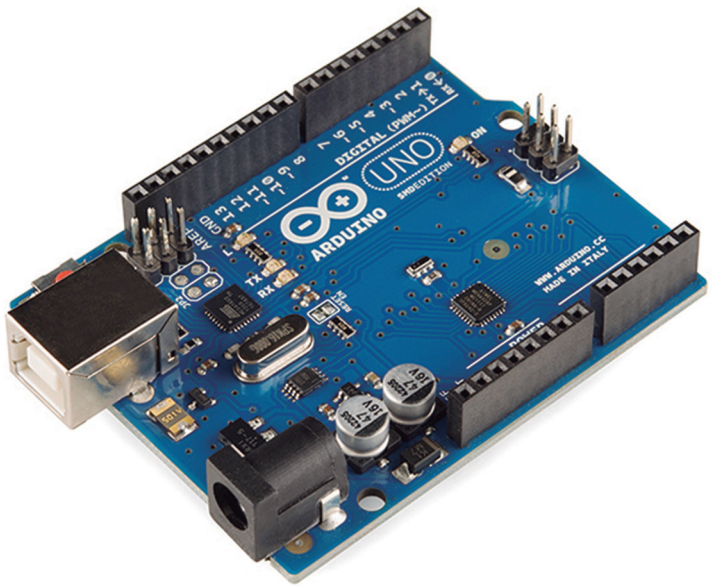
O Arduino é uma pequena placa para prototipagem eletrônica de sistemas digitais interativos e hardware livre[1] e é muito utilizado por profissionais e amadores no mundo todo. Consiste, basicamente, em um pequeno computador que possibilita controlar suas entradas, saídas e componentes externos conectados. É composto por uma placa de circuito integrado que possui um processador Atmel AVR[2], um oscilador de cristal, um regulador linear de cinco volts, um conector USB que

1 O hardware livre herda um sentido semelhante ao de software livre, o qual dispensa a necessidade de conseguir uma permissão do proprietário original para modificá-lo. Além disso, quando se trata de hardware, tem-se um produto físico que dificilmente os criadores despenderiam recursos para distribuí-los. Vale ressaçtar que um produto físico nada mais é do que a implementação de um design que pode ser compartilhado juntamente com uma licença aberta (IBM, 2010).

2 É um microprocessador de 8 bits que utiliza arquitetura RISC. (ATMEL, 2015).

possibilita ligá-lo a um computador, além de várias entradas e saídas analógicas e digitais, podendo, assim, ser conectado a vários dispositivos e sensores de baixo custo, os quais facilitam sua aquisição e, por isso, é muito utilizado para fins acadêmicos (EVANS, NOBLE e HOCHENBAUM, 2013, p. 22). A Figura 1 exibe a plataforma microcontrolada Arduino Uno.

Figura 1 – Arduino UNO



Fonte: (WIKIMEDIA, 2013)

A plataforma Arduino possui duas fileiras de pinos para a conexão de diversos circuitos eletrônicos externos, como sensores, motores, relés, diodos, microfones, autofalantes, etc.

A disposição dos pinos de conexão do Arduino permite que placas sejam encaixadas, as quais são conhecidas como *shields* e ampliam o potencial da plataforma e simplificam circuitos mais complexos. Há *shields* de vários tipos: controle de motor, displays LCD, telas de toque, WiFi, conexões com redes Ethernet, rastreio por GPS, áudio e muitos outros. Outro aspecto relevante acerca dessas placas é que, geralmente, possuem pinos de conexão que permitem agrupá-las para montar um projeto com múltiplas camadas.

A programação do Arduino é feita com o auxílio do computador, por meio de uma IDE (*Integrated Development Enviroment* – Ambiente Integrado

de Desenvolvimento) de desenvolvimento próprio e de fácil utilização, que analisa o código e envia as instruções para a placa através de uma comunicação serial (normalmente a USB – Universal Serial Bus). A IDE é disponibilizada gratuitamente para computadores com sistema operacional Windows, Mac e Linux. Mais detalhes sobre a IDE serão analisados na seção 1.5.

Uma das vantagens da plataforma Arduino é sua vasta comunidade virtual. São fóruns, blogs, vídeoblogs e apostilas disponíveis gratuitamente com códigos prontos. Isto acontece graças aos entusiastas do hardware livre, que compartilham informações pela pura vontade de expor suas experiências com a plataforma.

Esse programa foi criado na Itália no *Interaction Design Institute* na cidade de Ivrea, pelo professor Massimo Banzi, quando ele procurava um meio barato e fácil para os estudantes de design trabalharem com tecnologia. Ele discutiu seu problema com David Cuartielles, um pesquisador visitante da Suécia que estava procurando uma solução semelhante. Assim surgiu o Arduino: Banzi desenvolveu o microcontrolador juntamente com Cuartielles, que também desenhou a placa. David Mellis (aluno de Banzi) programou o software na placa e Gianluca Martino (engenheiro local) foi contratado para produzir uma tiragem inicial de duzentas unidades, as quais foram vendidas rapidamente, necessitando, por isso, de produções quantitativamente maiores para suprir a demanda. O Arduino se popularizou devido à facilidade de programar o microcontrolador e ao baixo custo. Posteriormente, placa original foi melhorada e novas versões surgiram. Atualmente, as vendas do equipamento oficial já alcançam 300 mil unidades no mundo todo (EVANS, NOBLE e HOCHENBAUM, 2013, p. 25).

1.1.1 ARDUINO UNO

Uno significa *um* em italiano, é chamado assim por trazer mudanças que iniciaram uma nova geração do Arduino. Foi lançado em 25 de setembro de 2010 no blog "*O Jantar está Servido*", juntamente com a placa Mega2560, que é uma versão mais poderosa da placa Uno (EVANS, NOBLE e HOCHENBAUM, 2013, p. 27).

O Arduino UNO possui um microcontrolador ATmega8u2 com um conversor USB para serial que torna o equipamento um dispositivo USB comum, como um mouse ou um teclado, facilitando seu manuseio. Ele dispõe de14 pinos digitais e cada um pode ser definido como dispositivo de entrada ou de saída, assim, pode-se estabelecer seis entradas analógicas, uma saída integrada de 5 V, uma saída de 3,3 V, e seis d saídas

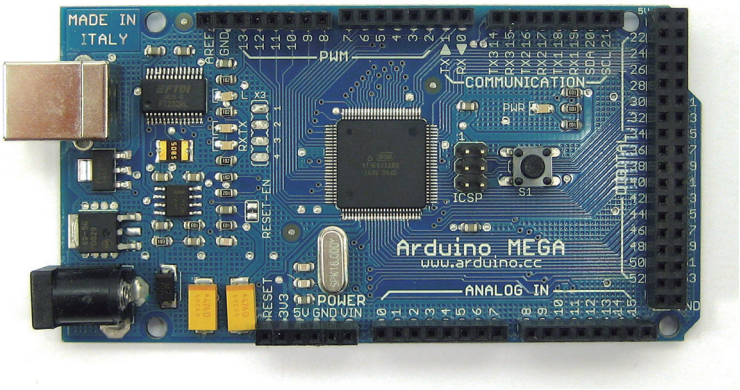
analógicas PWM[3] (*Pulse-Width Modulation* – Modulação por Largura de Pulso).

De acordo com (EVANS, NOBLE e HOCHENBAUM, 2013, p. 27), para tornar o Arduino UNO mais versátil, a placa foi construída visando a compatibilidade de pinos com os modelos anteriores, incluindo os popularíssimos Duemilanove e Diecimila.

1.1.2 ARDUINO MEGA

A placa microcontrolada Mega2560, exibida na Figura 2, é uma versão mais poderosa da placa UNO, possui memória flash de 256 KB e uma superfície maior de montagem, a qual fornece um aumento significativo nas funcionalidades de entrada, de saída e de comunicação. O Mega possui 54 pinos digitais de entrada ou de saída, no qual 14 pinos são capazes de fornecer saída analógica PWM. Além dos pinos digitais, apresenta 16 pinos de entrada analógica e 4portas seriais de hardware (EVANS, NOBLE e HOCHENBAUM, 2013, p. 29).

Figura 2 – Arduino Mega



Fonte: (WIKIMEDIA, 2010)

O Mega é ideal para projetos que controlam grandes quantidades de entradas, de saídas, de portas seriais. Ademais, necessita de mais memória.

3 É uma técnica para obtenção de resultados analógicos por meios digitais. O controle digital cria uma *square wave*, um sinal alternado entre *on* e *off*, que simula tensões entre *full on* (5 V) e *off* (0 V), alternando a parte em que o sinal passa *on* em relação ao tempo que o sinal passa *off* (ARDUINO, 2015).

1.2 COMPONENTES BÁSICOS DO ARDUINO UNO

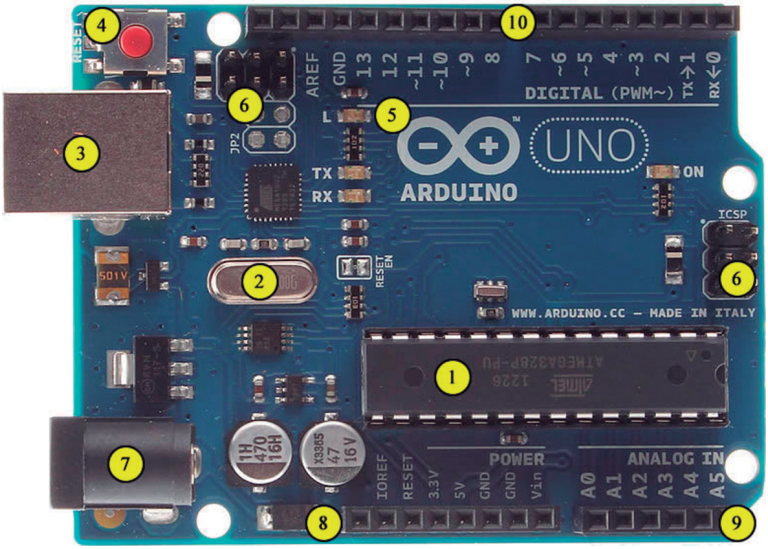
O Arduino Uno é a versão da placa utilizada neste projeto e será referenciada no decorrer do livro. Vale ressaltar que o Arduino Mega pode substituir o Arduino Uno sem qualquer alteração nos códigos ou nos circuitos.

A plataforma usada pode ser subdividida entre os componentes que fazem parte da placa, denominados *internos*, e os componentes que podem ser adicionados a placa ou ao circuito, denominados *externos*.

1.2.1 COMPONENTES INTERNOS

A Figura 3 enumera os componentes internos do Arduino Uno, sendo explicados de acordo com Monk (2014, p. 21).

Figura 3 – Componentes Internos do Arduino Mega



Fonte: (WIKIMEDIA, 2013)

1. ATmega328 – é o microcontrolador que possui 28 pinos encaixados em um soquete do tipo DIL[4] (*dual-in-line*), o que facilita a sua substituição; a UCP (Unidade Central de Processamento, do

4 Criado em 1965 por Don Forbes, Rex Rice e Bryant Rogers, o primeiro modelo era o 14-lead Ceramic Dual-in-Line Package que revolucionou a fabricação de computadores, simplificando o layout e permitindo a inserção automática em placas de circuito impresso (COMPUTER HISTORY MUSEUM, 2014).

inglês CPU – *Central Processing Unit*) busca dados do programa na memória flash e os executa. O microcontrolador tem memória flash de 32 KB, RAM (*Random Access Memory* – Memória de Acesso Randômico) de 2 KB e EEPROM[5] de 1 KB.

2. Oscilador de Quartzo – envia pulsos de tempo na frequência de 16 milhões de vezes por segundo (16 MHz) permitindo que as operações sejam executadas em um tempo correto e previsto, pois o processador só faz um cálculo por pulso recebido.
3. Conector USB – utilizado para transmitir os códigos do computador para a memória flash, além de fornecer uma alimentação de 5 V à placa que pode ser utilizada para testes durante o desenvolvimento do projeto.
4. Chave Reset – tem a função de reiniciar a execução do programa.
5. LED "L" – trata-se de um Led comum ligado ao pino 13.
6. Conectores ICSP (*In-Circuit Serial Programming*)[6] – oferecem outra forma de programar o Arduino sem usar a entrada USB. Não é utilizado nesse projeto.
7. Jack CC – recebe alimentação externa para o Arduino, entre 7,5 V e 12 V, podendo ser de um adaptador CC de tensão ou de uma bateria.
8. Conectores de Alimentação Elétrica – são conectores de funcionalidades variadas descritas a seguir. A ordem é da esquerda para a direita:

- IOREF – esse conector sempre possui a tensão que o Arduino trabalha. Por padrão, o UNO trabalha com 5 V, porém há a possibilidade de se alternar a tensão dependendo do projeto, caso uma shield esteja acoplada, por exemplo, o conector IOREF pode ser utilizado como referência da tensão atual do microcontrolador.
- Reset – quando aplicada nesse pino uma tensão momentânea de 5V é executado o mesmo comando da chave Reset. Assim, o microcontrolador será forçado a reiniciar a execução do programa.
- 3,3 V – conector que fornece 3,3 V.
- 5 V – conector que fornece 5 V.

5 EEPROM (*Electrically-Erasable Programmable Read-Only Memory*). Memória apenas para leitura de dados. Essa tecnologia permite programar circuitos sem a necessidade de removê-los, ademais, é possível apagar apenas o conteúdo de um endereço e reprogramar somente um determinado dado (TORRES, 2001, p. 361).

6 *In-Circuit Serial Programming* ou ICSP é um protocolo de comunicação com o microcrontrolador do Arduino.

- GND – conectores com 0 V. São as referências ao fim do circuito.
- Vin – quando o Arduino está sendo alimentado com uma tensão superior, através do Jack CC, esse conector também fornecerá essa tensão.

9. Entradas Analógicas – são seis no total, de A0 a A5, e a tensão aplicada sobre esse pino pode ser medida e utilizada como variável em um sketch. Para isso, possuem uma resistência elevada, o que torna insignificante a corrente que direcionam para o GND. Também podem ser utilizadas como conectores digitais de entrada ou saída, mas por default são entradas analógicas.
10. Conexões Digitais – vão de 0 à 13. Esses conectores podem ser utilizados como entrada ou saída. Quando utilizados como saída eles se comportam da mesma forma que o conector de alimentação elétrica de 5 V, além de poderem ser desligados (0 V) ou ligados (5 V) por meio do sketch. Quando utilizados como entrada eles se comportam semelhantemente a uma entrada analógica, porém, com a limitação de apenas detectar se a tensão nele está ou não acima de 2,5 V, aproximadamente. Por fim, os conectores digitais 3, 5, 6, 9, 10 e 11 (marcados com "~") são PWM, podendo fornecer saídas variáveis, ao invés de 0 V ou 5 V.

1.2.2 COMPONENTES EXTERNOS

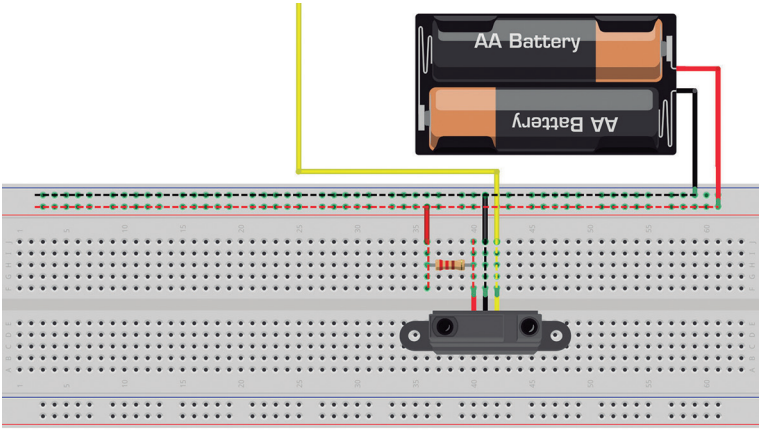
Os componentes externos são:

- Protoboard;
- Shields;
- Fontes de alimentação;
- Kits.

1.2.2.1 Protoboard

Chamada também de placa de ensaio, permite a construção de circuitos sem a necessidade de soldagem utilizando faixas horizontais e verticais conectadas por trilhas metálicas distribuídas conforme a Figura 4, nesta é possível perceber que o componente conectado à protoboard tem cada saída conectada à uma trilha. As trilhas se conectam entre si por meio de *jumpers* (fios conectores) ou outros meios, como o resistor da figura mencionada, para completar o circuito. Esse componente externo possui diversos orifícios que permitem o encaixe de componentes de circuitos integrados do tipo DIL sem que ocorra um curto-circuito.

Figura 4 – Trilhas da Protoboard



Fonte: Acervos dos autores[7].

1.2.2.2 Shields

São utilizadas para expandir a capacidade do Arduino de maneira simples e rápida. Em geral, possuem uma biblioteca[8] com funções apropriadas para sua utilização. Existe uma grande variedade de Shields, cada uma com um circuito montado para um determinado objetivo. Não é incomum a necessidade de modificar, montar ou construir uma shield totalmente nova e existem muitos fabricantes que disponibilizam os desenhos dos seus circuitos com a licença padrão de Hardware Livre, permitindo que qualquer um os copie, desenvolva e distribua a própria shield.

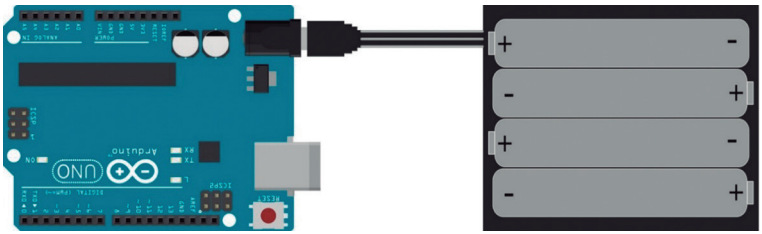
1.2.2.3 Fontes de Alimentação

O Arduino pode ser energizado por dois meios: USB e Jack CC. A entrada USB fornece 5 V para o Arduino, enquanto o Jack CC pode ser alimentado por uma variedade de modelos de fonte com tensão entre 7,5 V a 12 V (ARDUINO, 2015). A Figura 5 exibe a conexão entre o Arduino e um conjunto de pilhas com um adaptador através do Jack CC.

7 Da Figura 4 em diante, as imagens que não apresentarem fonte e referência são do acervo dos próprios autores.

8 Uma biblioteca é uma coleção de funções. Um arquivo de biblioteca armazena o nome de cada função. Quando o sketch referencia uma função contida em uma biblioteca, essa função é buscada no arquivo desta e adicionada ao código do sketch (SCHILDIT, 1996, p. 12).

Figura 5 – Fontes de Alimentação



1.2.2.4 Kits

São semelhantes as shields e possuem funcionalidades específicas para um determinado projeto. Um exemplo é o Kit Magician utilizado neste projeto e exibido na Figura 6; ele possui um manual de instruções e peças para construir um carro com duas rodas movidas por motores DC[9] e é ideal para o aprendizado de programação. É muito comum a aquisição de um kit inicial para estudar o Arduino.

Figura 6 – Kit Magician

Fonte: (SPARKFUN ELETRONICS, 2015)

9 Os motores de corrente contínua (CC) ou motores DC (Direct Current), como também são chamados, são dispositivos que operam aproveitando as forças de atração e repulsão geradas por eletroímãs e imãs permanentes. São encontrados em gravadores, brinquedos, câmaras de vídeo, aparelhos de som, etc (INSTITUTO NEWTON C. BRAGA, 2014).

1.2.2.5 Outros

Há vários componentes não citados acima, tais como, reguladores, controladores, capacitores, diodos, leds, transistores, resistores, chaves, displays, teclados, relés, etc. Além disso, com o passar dos anos, novos elementos eletrônicos são desenvolvidos e disponibilizados no mercado, por isso, é necessário estar atento e acompanhar os fóruns de discussão para ter sempre conhecimento das novas possibilidades. O tópico 4 deste tutorial contém vários sites para estudo.

1.3 CIRCUITOS ELETRÔNICOS

1.3.1 TERRA (GND) E ALIMENTAÇÃO DO CIRCUITO

Um circuito eletrônico possui, no mínimo, uma conexão em que há excesso de elétrons e,por isso, se apresenta carregada negativamente; e outra conexão em que há falta de elétrons, assim, está carregada positivamente. Quando ambos os polos são interligados por um condutor, há um fluxo de corrente elétrica que gera a alimentação momentânea do circuito devido à passagem de carga de um polo ao outro que se mantém até haver o equilíbrio das cargas. O polo negativo geralmente aparece nos circuitos com o nome de GND (*ground* ou "terra") e o polo positivo, com o nome de Vcc, ou no caso do Arduino, com o nome referente à tensão (INSTITUTO NEWTON C. BRAGA, 2014).

Em geral, uma trilha de orifícios da protoboard é conectada a um pino GND, como mostra a Figura 7. Assim, todos componentes do circuito são unidos ao GND por essa trilha. Mais informações sobre circuito elétrico e efeitos de corrente estão no tópico 4 desse tutorial.

Na fonte de alimentação, é normal conectar mais de um componente do circuito ou utilizar diversas fontes de alimentação em paralelo. A Figura 8 exibe uma protoboard com uma trilha de orifícios positivos de 9 V fornecida pela bateria externa, que o motor DC, da imagem, utiliza para funcionar. Os LEDs, por sua vez, utilizam a alimentação de 5 V fornecido pelos pinos do Arduino em que estão conectados. Ambos utilizam a trilha GND conectado à bateria.

Figura 7 – GND Conectado à Protoboard

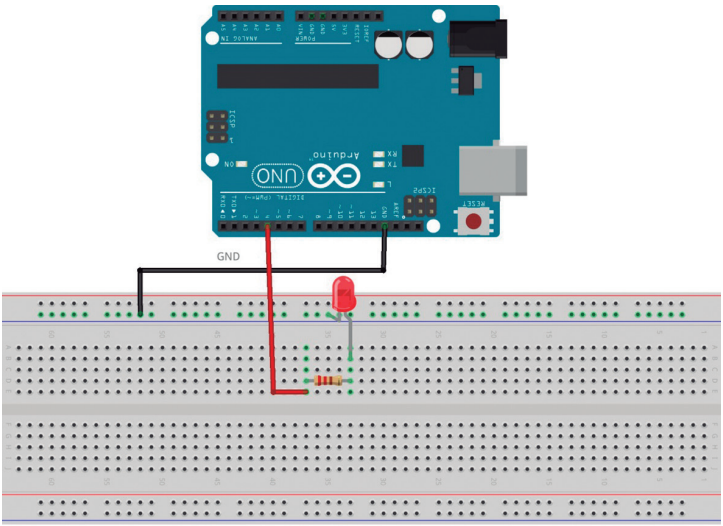
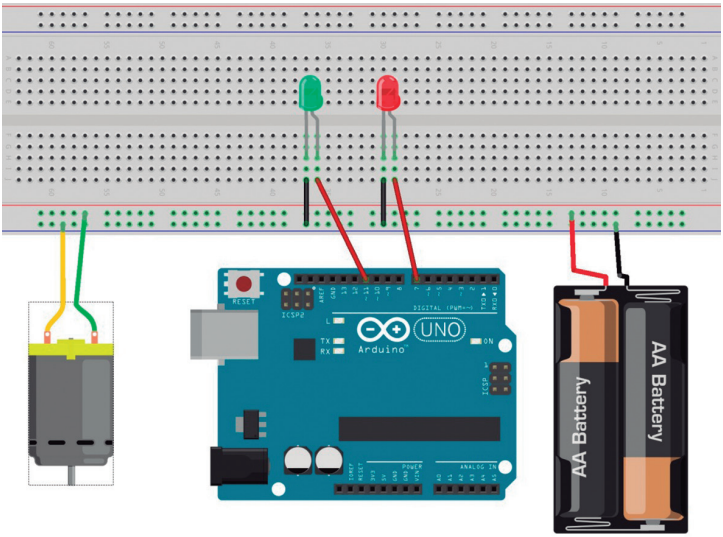


Figura 8 – Fonte de Alimentação Conectada à Protoboard



1.3.2 RESISTORES E A LEI DE OHM

A lei de Ohm faz relação entre três grandezas: a corrente (i), a tensão (V) e a resistência (R). Entende-se por resistência elétrica a dificuldade que uma corrente encontra quando passa por um condutor de eletricidade, esta é provocada pelos resistores, responsáveis por reduzir a corrente em determinado ponto do circuito (como um cano estreito em uma tubulação mais larga) A resistência é medida em ohm, sendo seu símbolo a letra grega ômega maiúscula (Ω) e seus múltiplos mais comuns são o kiloohm (kΩ = 1000 Ω) e o megaohm (mΩ = 1000 kΩ).

Todo dispositivo eletrônico possui uma tensão medida em Volts e uma intensidade, da corrente que suporta, medida em Ampères, que é informada nas especificações técnicas do dispositivo. Esses valores são importantes para escolher o resistor correto e a fonte de alimentação que será utilizada no circuito. A resistência elétrica (R) entre dois pontos de um condutor qualquer é a razão entre a diferença de potencial (V) e a intensidade de corrente elétrica (i), tal como descrito nas Equações 1, 2 e 3:

Equação 1 – Lei de Ohm

$$R = \frac{V}{i} (1)$$

Equação 2 – Lei de Ohm 2

$$V = R \times i (2)$$

Equação 3 – Lei de Ohm 3

$$i = \frac{V}{R} (3)$$

Por exemplo: 10 mA (0,01 A) de corrente elétrica é o suficiente para acender um LED. O LED possui uma resistência que diminui o equivalente a 2 V da tensão aplicada, É importante considerar que o Arduino fornece 40 mA de corrente e 5 V de tensão em cada pino digital, o que pode danificar o LED, por isso, utiliza-se resistores. Aplicando esses valores na Fórmula 1 obtém-se o valor da resistência apropriada para utilizar o LED, assim sendo, basta subtrair os 2 V de resistência do LED dos 5 V

fornecidos pelo pino digital e em seguida dividir pela corrente requisitada pelo emissor de luz, que é de 0,01 A,

$$R = \frac{(5V - 2V)}{10mA}$$

$$R = \frac{3V}{0,01}$$

$$R = 300ohms$$

Ou seja, é necessário, nesse caso, um resistor de 300 ohms para passar uma corrente de 10 mA em um LED, sem danifica-lo.

Havendo apenas um resistor de 220 ohms, utiliza-se a fórmula 3 para calcular a corrente resultante que passará pelo LED. Com os 5 V fornecidos pelo pino digital subtrai-se os 2 V da resistência do LED, a diferença é dividida por 220, obtendo 0,0014 A ou 14 mA.

$$i = \frac{(5V - 2V)}{220}$$

$$i = \frac{3V}{220}$$

$$i = 0,014A\ (aproximaçãode3casasdecimais)$$

$$i = 14mA$$

Ou seja, há um pequeno excedente de 4 mA, porém, essa amperagem é insuficiente para causar danos ao LED.

A fórmula (2) é utilizada quando se deseja obter a diferença de potencial para determinado resistor e corrente, um caso pouco comum, por isso, é a menos utilizada.

As placas da plataformaArduino são tolerantes a um pequeno excesso de corrente, contudo, alguns de seus componentes não possuem esta característica, por isso, existem diversos tipos de resistores com as mais variadas resistências. Os mais comuns são os resistores fixos de carbono e os potenciômetros, que são resistores variáveis.

1.3.3 CÁLCULO DE RESISTÊNCIA

A resistência dos resistores de carbono pode ser calculada a partir das faixas coloridas desenhadas nos resistores. A Tabela 1 apresenta os valores e os níveis de variação.

Tabela 1 – Cores dos Resistores de Carbono

	Cor	1ª faixa	2ª faixa	3ª faixa	Multiplicador	Tolerância
	Preto	0	0	0	x 10^0	±1% (F)
	Marrom	1	1	1	x 10^1	±2% (G)
	Vermelho	2	2	2	x 10^2	
	Laranja	3	3	3	x 10^3	
	Amarelo	4	4	4	x 10^4	
	Verde	5	5	5	x 10^5	±0.5% (D)
	Azul	6	6	6	x 10^6	±0.25% (C)
	Violeta	7	7	7	x 10^7	±0.1% (B)
	Cinza	8	8	8	x 10^8	±0.05% (A)
	Branco	9	9	9	x 10^9	
	Ouro				x 10^{-1}	±5% (J)
	Prata				x 10^{-2}	±10% (K)

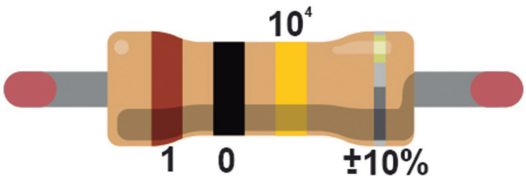
Fonte: (FOWLER, 2013)

Os passos para o cálculo do valor da resistência de um resistor de carbono são os seguintes:

1. Verificar quantas cores há no resistor. Se possuir 4 coresserão utilizadas apenas a 1ª e a 2ª faixas da Tabela 1, se possuir 5 cores, a 1ª, a 2ª e a 3ª faixas da Tabela 1; as faixas definirão o número base (aquele que iniciará o cálculo);
2. Após ter o número de cores, basta conferir na Tabela 1 o número respectivo de cada faixa de cor e agrupá-los. Exemplo: um resistor de 5 cores com as 3 primeiras: marrom, vermelho e preto, respectivamente, possui 120 de número base (“1” marrom, “2” vermelho e “0” preto). Um resistor de 4 cores com as 2 primeiras azul e marrom, possui 61 de número base (“6” azul e “1” marrom);
3. Depois de portar o número base, basta conferir a penúltima cor e multiplicar pelo múltiplo de 10 correspondente, obtendo o valor de sua resistência. Exemplo: um resistor com número base 200 e com a penúltima cor laranja (Multiplicador “10^{3}”) tem 200000 Ω (200 x 10^3) de resistência ou 2 MΩ;
4. Para obter o intervalo de variação do valor da resistência, basta conferir a última cor do resistor. Exemplo: um resistor com resistência de 150 kΩ e com a última faixa de cor prata possui uma variação de mais ou menos dez por cento (±10%), conforme definido na Tabela 1.

Na Figura 9, as cores do resistor seguem na seguinte ordem: marrom (“1”), preto (“0”), amarelo (“10^4”) e prata (“±10%”). O cálculo para encontrar o valor de resistência será: 10 x 10^4, que resulta em 100.000 **Ω** ou 100 k**Ω**, podendo haver uma variação de 10% acima ou abaixo do resultado obtido.

Figura 9 – Resistor



1.3.4 POTENCIÔMETRO

Os potenciômetros possuem uma resistência que pode variar de acordo com a posição da chave, por isso, não são utilizados apenas para fazer oposição à intensidade da corrente, mas também para que essa variação seja detectada e o circuito seja programado com base nela. Os mais utilizados são os potenciômetros lineares, exibido na Figura 10, que são utilizados como reguladores de volume em muitos aparelhos.

Figura 10 – Potenciômetro

Fonte: (WIKIMEDIA, 2006)

1.3.5 DATASHEET

É fácil encontrar as informações necessárias para construir um circuito com segurança. Geralmente, os componentes trazem suas especificações sobre a tensão (V) que utilizam e a intensidade da corrente (i) que suportam. Entretanto, nem sempre as informações estão nos componentes, assim, é necessária uma pesquisa para encontrar o fabricante e o *datasheet,* a partir do número de série. O *datasheet* traz todas as informações sobre o componente, tais como: tensão, corrente resistida, desenho do circuito, etc. A Figura 11 mostra um motor DC com informações do número de série FF-050SB.

Figura 11 – Motor DC

1.3.6 MULTÍMETRO

O multímetro é utilizado para medir a resistência, a corrente ou a tensão de um componente do circuito. Na Figura 12 é possível notar que um multímetro possui um cabo preto e um vermelho utilizados na medição, para medir (resistência, corrente ou tensão) basta colocar o cabo vermelho na entrada de energia do componente e o cabo preto na saída do componente, em seguida, basta selecionar a escala que o componente trabalha. Com uma rápida pesquisa são encontrados muitos locais na internet abordando a utilização desse componente em circuitos, alguns links indicados são descritos no capítulo 4 – Links Úteis.

1.4 A LINGUAGEM C DO ARDUINO

Ao iniciar esse livro é recomendado que o leitor já esteja familiarizado com as características da linguagem C, tais como: estruturas de decisão, estruturas de repetição, strings, matrizes, vetores, entre outros elementos básicos.

Os códigos do Arduino utilizados nesse manual serão conhecidos como sketch, termo utilizado na comunidade dessa placa para

Figura 12 – Multímetro

Fonte: (WIKIMEDIA, 2015)

prototipagem na internet. Poucas sintaxes mudam na estrutura de um código em C (Figura 13-b) para um sketch no Arduino (Figura 13-a). Cabe frisar que os operadores, os caracteres especiais, os tipos de variáveis e os structs continuam os mesmos. Enfim, apenas a função **`main()`** muda, o restante sofre apenas adições.

Figura 13 – Comparação Entre Códigos

Sketch Arduino:

```
int led = 1;
void setup()
{
  pinMode(led, OUTPUT);
}

void loop()
{
  digitalWrite(led, HIGH);
}
```

(a)

C ANSI:

```
#include<stdio.h>

int x = 2;
int main()
{
  printf("n = ", x);
  return 0;
}
```

(b)

É possível observar que na estrutura de um sketch (Figura 13-a) há duas funções principais, **`setup()`** e **`loop()`**, ao invés de apenas uma função principal **`main()`**, utilizadas no código da linguagem C padrão (Figura 13-b). Dentro dessas duas funções são organizadas as sequências de códigos que interagem com o circuito.

1.4.1 VARIÁVEIS

As variáveis em um sketch, assim como em um código em C, podem ser declaradas em qualquer local do código, ou seja, dentro ou fora das funções. No entanto, o local em que a variável é declarada influencia na zona onde podem ser acessadas. Por esse motivo, as principais variáveis são declaradas fora das funções como **globais**. Alguns tipos de variáveis utilizadas neste tutorial são descritas abaixo:

- **boolean** – Armazena valores do tipo verdadeiro (**`true`**) ou falso (**`false`**), muito utilizados para estruturas condicionais como **`if()`** ou **`while()`**;
- **byte** – Armazena números de 8 bits sem sinais. Com a combinação binária de 8 bits (1 byte) é possível obter números decimais de 0 a 255;
- **char –** Armazena apenas 1 caractere da tabela ASCII[10]. Um vetor desse tipo de variável será uma *string* (vetor de caracteres), usada para guardar cadeias de caracteres;
- **float, double** – Armazena números em ponto flutuante de até 4 bytes de informação e abrangem a faixa de 3,4028235E+38 a -3,4028235E+38. No Arduino, os tipos de variáveis **`float`** e **`double`** possuem as mesmas características;
- **int** – Armazena números inteiros com até 2 bytes de informação e compreendem a faixa de -32768 a 32767;
- **long** – Armazena números inteiros com até 4 bytes de informação e compreendem a faixa de -2.147.483,648 a 2.147.483.647;
- **unsigned int** – Armazena números inteiros não sinalizados (apenas valores positivos) com 2 bytes de informação e compreendem a faixa de 0 a 65.535;
- **unsigned long** – Armazena números inteiros não sinalizados (apenas valores positivos) com a precisão de 4 bytes e compreendem a faixa de 0 a 4.294.967.295;

10 A tabela ASCII (American Standard Code for Information Interchange, em português Código Americano Padrão para o Intercâmbio de Informações), é um conjunto de valores que representam caracteres e códigos de controle armazenados ou utilizados em computadores.

- **Outros tipos de variáveis** – Algumas bibliotecas podem utilizar tipos primitivos da linguagem C ou estruturas de dados criadas no próprio código fonte.

1.4.2 CONSTANTES

Existem na estrutura de um sketch constantes pré-definidas para facilitar a programação. Ao todo existem três duplas de constantes: `HIGH` e `LOW`; `true` e `false`; e `INPUT` e `OUTPUT`.

HIGH / LOW – Definem o estado da tensão de um pino do Arduino (alta ou baixa tensão).

- `HIGH`: Um pino com o valor `HIGH` (alto) fornece 5 V de tensão com corrente de 40 mAh[11] para o dispositivo.
- `LOW`: Um pino com o valor `LOW` (baixo) não apresenta corrente elétrica (zero volts) ou a corrente existente é desprezível para ser usado na ativação de um componente que necessite de eletricidade; pode, também, ser referenciado como "0" (zero) nas funções que utilizam essa constante como parâmetro.

INPUT / OUTPUT – Determina se um pino do Arduino é de entrada ou de saída de dados. O pino é configurado pela função `pinMode()`, a qual atribui a ele o valor de uma dessas duas constantes.

- `INPUT`: É a constante que marca um pino como entrada de dados. Pinos configurados com essa constante receberão os dados dos componentes externos.
- `OUTPUT`: É a constante que marca um pino como saída de dados. Pinos configurados por essa constante fornecem corrente elétrica para o circuito.

TRUE / FALSE – São constantes que representam um estado de verdadeiro ou falso. Ao contrário das constantes anteriores, `true` e `false` são escritas em letras minúsculas.

- `false`: Falso ou zero.
- `true`: Verdadeiro ou diferente de zero.

1.4.3 FUNÇÕES

As funções da biblioteca básica da IDE do Arduino descritas neste tópico não precisam ser incluídas no cabeçalho do código e são muito utilizadas no código sketch.

---

11 A medida de miliampère-hora (mAh) é uma unidade de carga elétrica que se refere a quantidade de carga que uma fonte pode fornecer para um dispositivo durante uma hora.

1.4.3.1 Funções Setup e Loop

setup() – Essa função é executada apenas uma vez ao ligar o Arduino e prepara o código antes de entrar na função **loop()**. Nesta função que são configurados os estados dos pinos do Arduino, a inicialização da porta serial e outras especificações iniciais para o funcionamento do sketch.

loop() – É a função chamada após a execução do **setup()**. Essa função, como o próprio nome já diz, funciona como um loop, assim, é executada repetidas vezes enquanto o Arduino estiver ligado.

1.4.3.2 Funções Para Entrada/Saída Digitais

São funções que recebem como parâmetro a localização dos pinos digitais presentes no Arduino, além de escreverem ou retornarem um valor.

pinMode(pino,modo) – Essa função configura os pinos do Arduino como saída (**OUTPUT**) ou entrada (**INPUT**). No exemplo do Código 1, a função **pinMode()** configura a variável **botao** como entrada de dados e a variável **led** como saída de dados.

Código 1 – Exemplo da Função `pinMode()`

```
1  int botao = 9; //botão conectado no pino
2  int led = 4; //LED conectado no pino 4
3  setup()
4  {
5  pinMode(botao, INPUT); //configura como entrada
6  pinMode(led, OUTPUT); //configura como
saída
7  }
```

digitalWrite(pino,valor) – Essa função se comporta de duas maneiras diferentes: uma quando o parâmetro **pino** passado para função for configurado como **OUTPUT** (saída) pela função **pinMode()** e outra quando for configurado como **INPUT** (entrada) pela função **pinMode()**.

Parâmetro **pino** configurado como **OUTPUT** pela função **pinMode()** – A função **digitalWrite()** servirá para ligar ou desligar um componente. Dessa forma, temos ainda duas alternativas que dependem do segundo parâmetro da função (**valor**) e podem ser definidas pelas palavras **HIGH** ou **LOW.** Caso o parâmetro (**valor**) seja definido como **HIGH,** o pino referenciado pela função receberá uma tensão de 5 V e caso o segundo

parâmetro (**valor**) da função seja **LOW**, o pino referenciado na função não receberá alimentação (0 V). Um exemplo dessa funcionalidade é mostrado no Código 2, no qual, dependendo do estado do botão (**HIGH** ou **LOW**, linha 13), o led é aceso (linha 15) ou apagado (linha 17).

Código 2 – Exemplo função `digitalWrite()`

```
1  int led = 10;
2  int botao = 6;
3
4  void setup()
5  {
6    pinMode(led, OUTPUT); //pino do
   led como saída
7    pinMode(botao, INPUT); //pino do
   botão como entrada
8  }
9
10 void loop()
11 {
12   int estado = botao;
13   if(estado == LOW)
14   {
15 digitalWrite(led, HIGH);//acende o
   LED (5 volts)
16   } else {
17 digitalWrite(led, LOW);//apaga o LED
   (0 Volt)
18   }
19 }
```

Parâmetro **pino** configurado como **OUTPUT** pela função **pinMode():** A função **digitalWrite()** funcionarápara ligar ou desligar um resistor interno[12] presente na placa do Arduino. Dessa forma, como no caso anterior, temos ainda duas alternativas que

12 O Arduino possui em seu interior um resistor de 20 k**Ω** que é ativado via código.

dependem do segundo parâmetro da função (**valor**), que pode ser definida pelas palavras **HIGH** ou **LOW.** Se o parâmetro passado for **HIGH,** é ativado o resistor interno do Arduino direcionado ao pino referenciado na função **digitalWrite(), opositivamente,**o valor **LOW** desativa esse resistor. Faz-se válido ressaltar que esse fato retira, em alguns casos, a necessidade da utilização de resistores externos no circuito,como exemplo temoso Código 3:o componente conectado ao pino 6 (um botão), que foi configurado como **INPUT,** pela função **pinMode()** na linha 7, não necessitará da utilização de um resistor externo no circuito, pois seráativado o resistor interno do Arduino no pino em que ele estiver conectado, na linha 8, por meioda função **digitalWrite().**

Código 3 – Função `digitalWrite()` Ativando o Resistor do Arduino

```
1  int led = 10;
2  int botao = 6;
3
4  void setup()
5  {
6    pinMode(led, OUTPUT); //pino do
   led como saída
7    pinMode(botao, INPUT); //pino do
   botão como entrada
8    digitalWrite(botao, HIGH); /*ativa
   o resistor interno   do pino 6 */
9  }
10
11 void loop()
12 {
13   if(estado == LOW)
14   {
15     digitalWrite(led, HIGH); //
   acende LED
16   } else {
17     digitalWrite(led, LOW); //apaga
   LED
18   }
19 }
```

digitalRead(pino) – Essa função lê o valor do pino digital indicado, o qual é representado por **HIGH** ou **LOW**. Caso o valor da carga presente no pino possua uma tensão maior que 3 volts, a função retornará **HIGH**; caso possua uma tensão menor que 2 V, a função retornará **LOW**. O Sketch 4 apresenta um exemplo do uso da função em que a variável **estado** irá receber a leitura digital (linha 1) do pino 4 e, dependo da resposta (**HIGH** ou **LOW**), irá realizar um trecho do código.

Código 4 – Exemplo da Função digitalRead()

```
1  estado = digitalRead(4);
2  if(estado == HIGH)
3  {
4    //fazer algo
5  }
```

1.4.3.3 Funções Para Entrada/Saída Analógicas

São funções que trabalham com os pinos **A0**, **A1**, **A2**, **A3**, **A4** e **A5** do Arduino (UNO), nos quais são conectados os sensores para a leitura de variáveis não digitais do ambiente, como a quantidade de luz, distância de um objeto, temperatura, dentre outros.

analogRead(pino) – Lê um valor analógico no pino indicado que oscila entre 0 e 1023, em números inteiros. O sensor realiza uma leitura entre 0 e 5 volts, que quando dividido por 0,0048 retorna o inteiro correspondente. Uma utilização dessa função é descrita no Código 5, no qual a variável **luz** recebe a leitura analógica de um componente ligado ao pino A1 do Arduino (linha 3), se essa leitura apresentar um valor abaixo de 100, uma função será realizada no código.

Código 5 – Exemplo da Função analogRead()

```
1  loop()
2  {
3    int luz = analogRead(A1); //
   leitura analogica
4    if(luz < 100); //se o nível for
   menor que 100
5    {
6      acenderLED(); //acende um LED
7    }
8  }
```

analogWrite(pino,valor) – Essa função envia um valor para o pino analógico, mudando seu valor atual. Um exemplo dessa função é exibido no Código 6.

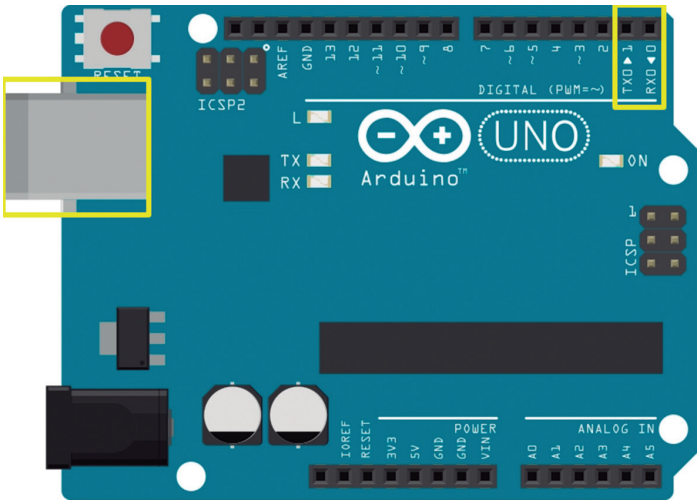
Código 6 – Exemplo da Função analogWrite()

```
1  analogWrite(A2, 50); /*envia o valor 50
para a saída analógica A2*/
```

1.4.3.4 Funções Para Comunicação Serial

Estas funções são utilizadas para comunicação serial do Arduino com o computador, a qual, normalmente, é estabelecida pela saída USB ou pelos pinos digitais 0 (RX) e 1 (TX)[13] presentes na placa. A Figura 14 mostra a localização das saídas para comunicação serial no Arduino.

Figura 14 – Saídas Para a Comunicação Serial



Serial.begin (velocidade) – Essa função abre a porta serial do Arduino e ajusta a taxa de transferência de dados entre o Arduino e o computador, normalmente declarada na função **setup()**. As seguintes taxas, em bits/segundo, podem ser definidas como parâmetro: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, e 115200. Porém, há outras

13 Os pinos RX e TX dizem respeito a um dos métodos de transmissão Serial do Arduino, o pino RX é onde se deve conectar o receptor do componente externo que leva os dados à placa do Arduino e no pino TX deve-se conectar o transmissor, oqual leva os dados da placa para o componente.

taxas que podem ser utilizadas em casos especiais, nos quais há necessidade de um valor específico. O Código 7 exemplifica o uso dessa função.

Código 7 – Exemplo da Função `Serial.begin()`

```
1  Setup()
2  {
3     Serial.begin(9600);/*inicia o
   monitor serial com uma taxa de 9600
   bit/segundo*/
4  }
```

`Serial.print()` – Essa função imprime dados no monitor serial da IDE do Arduino e pode utilizar diversos parâmetros:

- **`Serial.print(INT)`**: com um parâmetro, essa função imprime apenas um número decimal;
- **`Serial.print(INT, TIPO)`**: imprime o primeiro parâmetro da função convertido no tipo especificado no segundo parâmetro. Os tipos a serem especificados neste podem variar entre **`DEC`** (decimal), **`HEX`** (hexadecimal), **`OCT`** (octal), **`BIN`** (binário) ou **`BYTE`** (byte único).
- **`Serial.print(STRING)`**: imprime a string no monitor serial.

Para visualizar o envio e recebimento de dados no IDE do Arduino, basta acessar a ferramenta *Serial Monitor* na IDE, descrito na seção 1.5.4. O Código 8 exemplifica o uso dessas funções : na linha 3, foi iniciado o monitor serial ainda na função **`setup()`**; na linha 5 foi inserida a função **`loop()`**, sendo adicionado o número 94 na variável **`n`** (linha 7) o qual foi impresso no monitor serial em diversos sistemas numéricos; E na linha 12, foi impresso o nome "Arduino" no monitor serial.

Código 8 – Exemplo da Função `Serial.print()`

```
1  setup()
2  {
3     Serial.begin(9600);
4  }

5  loop()
6  {
7     n = 94;
```

```
8     Serial.print(n, DEC);//imprime
   "94" no monitor serial
9     Serial.print(n, HEX);//imprime
   "5E"(94 em hexadecimal)
10    Serial.print(n, OCT);//imprime
   "136"(94 em octal)
11    Serial.print(n, BIN);//imprime
   "0101 1110"
12    Serial.print("Arduino");//imprime a
   palavra Arduino
13 }
```

Serial.println() – Essa função é semelhante a execução **Serial.print()**, porém adiciona um *carriage return*[14] ('\r') e um caractere de nova linha ('\n'), ou seja, imprime os dados adicionando uma nova linha no fim. Seus parâmetros são os mesmos descritos na função **Serial.print()**.

Serial.available() – Essa função retorna a quantidade de bytes disponíveis para leitura na porta serial. O buffer serial pode armazenar até 128 bytes.

Serial.read() – Essa função retorna a quantidade de bytes dos dados que estão entrando pela porta serial,se não houver dados disponívei, função retorna (-1).

Serial.flush() – Essa função limpa o buffer serial do Arduino, desse modo, após sua chamada, não haverá dados disponíveis no buffer para leitura. O Código 9 exemplifica o uso dessas funções, na linha 10 é verificado se a quantidade de bytes retornado pela função **Serial.available()** é maior que 0, ou seja, se existe algum byte no buffer, caso exista, essa quantidade é armazenada na variável **dados_serial** (linha 12) e, no final do sketch, a função **Serial.flush()** limpa o buffer.

Código 9 – Exemplo das Funções que Trabalham com o Buffer Serial

```
1  int dados_serial;
2
3  setup()
4  {
5     Serial.begin(9600);
```

14 Caractere de controle do código ASCII, Unicode ou EBCDIC que faz com que uma impressora ou o dispositivo de saída (geralmente o monitor) mova a posição do cursor para a primeira posição da linha onde se encontra.

```
6  }
7
8  loop()
9  {
10   if(Serial.available() > 0)
   /*verifica se há dados   disponíveis
   no buffer*/
11   {
12     dados_serial = Serial.
   read();/*armazena os dados lidos na
   variável*/
13   }
14   Serial.flush(); //limpa o buffer
15 }
```

1.5 IDE DO ARDUINO

A IDE do Arduino é a interface que o usuário desenvolve um sketch para o circuito. O programa está disponibilizado gratuitamente no site oficial do Arduino (http://arduino.cc/), na seção de downloads. O site conta com versões compatíveis com os sistemas operacionais *Windows, Linux* e *Mac OS X*.

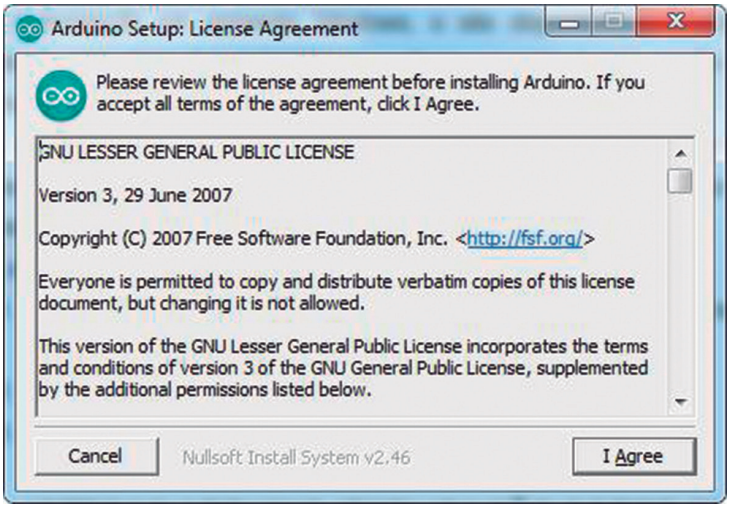
As seções seguintes explicam a instalação do programa nos sistemas operacionais *Linux* e *Windows* além de alguns recursos importantes.

1.5.1 INSTALAÇÃO NO SISTEMA WINDOWS

A instalação em ambiente Windows está disponível no site em duas opções para download: um instalador e uma versão compactada do programa. Para a instalação da versão compactada, extraia o conteúdo do arquivo em formato *zip* na pasta de sua preferência. O conteúdo extraído equivale ao programa instalado e para acessá-lo basta abrir o executável do programa presente no local em que os arquivos foram extraídos. Com o instalador siga os seguintes passos:

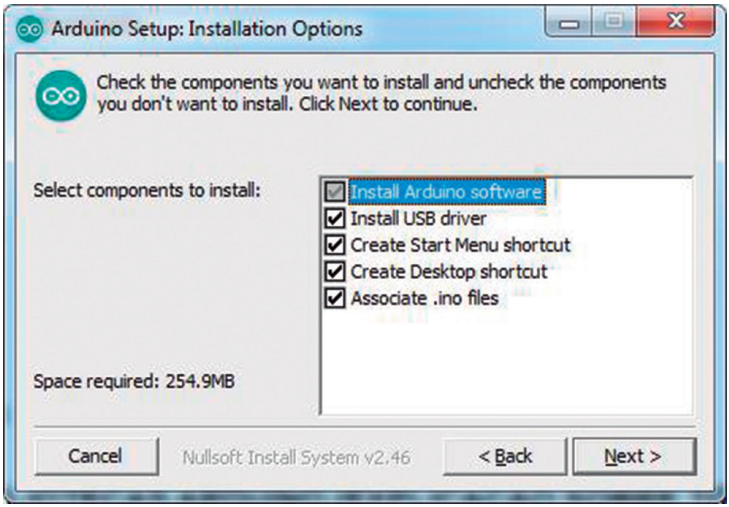
1. Baixe o instalador no site oficial do Arduino na seção de downloads.
2. Execute o instalador e aceite os termos de uso do programa da tela exibida na Figura 15.

Figura 15 – Instalação da IDE no Sistema Windows



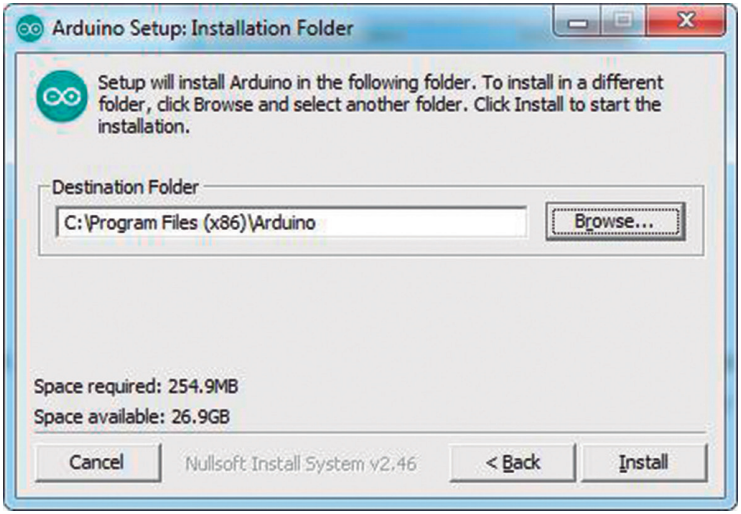
3. Clique em *Next* para avançar e deixe marcadas todas as opções, como mostrado na Figura 16 abaixo.

Figura 16 – Instalação da IDE no Sistema Windows



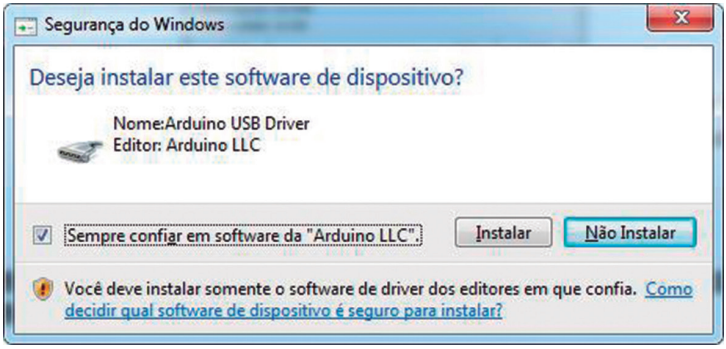
4. Selecione um local para realizar a instalação. Por padrão será a pasta "C:\Program Files\Arduino", como mostrado na Figura 17 abaixo. Após escolher o local, clique em *Install*.

Figura 17 – Instalação da IDE no Sistema Windows



5. Durante o processo é possível que apareça uma solicitação para a instalar o driver USB do Arduino, como mostrado na Figura 18 abaixo. Clique em instalar.

Figura 18 – Instalação da IDE no Sistema Windows

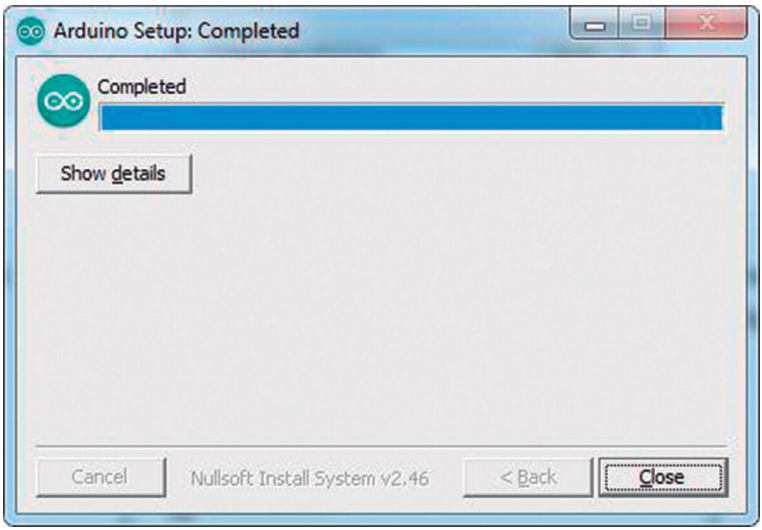


6. Para finalizar clique em close como mostrado na Figura 19. Será criado um atalho na área de trabalho.

1.5.2 INSTALAÇÃO NO SISTEMA UBUNTU

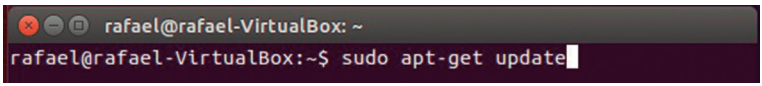
A instalação do Arduino no Linux deve ser feita seguindo os passos abaixo:

Figura 19 – Instalação da IDE no Sistema Windows



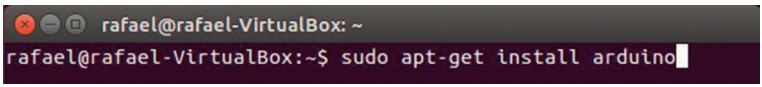
1. Abra o terminal do Linux no Ubuntu e digite o comando "**`sudo apt-get update`**", sem as aspas, e logo após aperte a tecla *ENTER*, como mostrado na Figura 20. Será solicitada a senha root do sistema, digite-a e aperte *ENTER* novamente. Assim será realizada a atualização do repositório de programas.

Figura 20 – Instalação da IDE no Sistema Ubuntu



2. Ainda no terminal digite o comando "**`sudo apt-get install arduino`**", sem as aspas, como exibido na Figura 21. Se solicitado à senha *root*, digite-a e aperte *ENTER*. Será questionado se realmente deseja instalar o programa, digite "**`y`**", sem aspas, e aperte *ENTER*.

Figura 21 – Instalação da IDE no Sistema Ubuntu



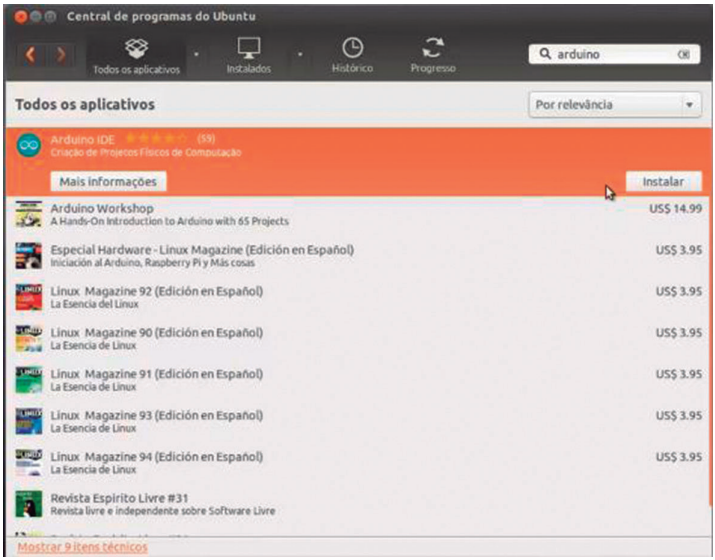
3. Após esses passos a IDE estará disponível nos aplicativos do sistema, como mostrado na Figura 22.

Figura 22 – Instalação da IDE no Sistema Ubuntu

É possível também acessar a central de aplicativos do sistema e efetuar o download e a instalação do Arduino IDE por este recurso seguindo os seguintes passos:

1. Abra a central de aplicativos Ubuntu.
2. Na aba de procura digite "Arduino" e pesquise.
3. Aparecendo o programa Arduino IDE, clique no botão "instalar" como mostrado na Figura 23 abaixo.

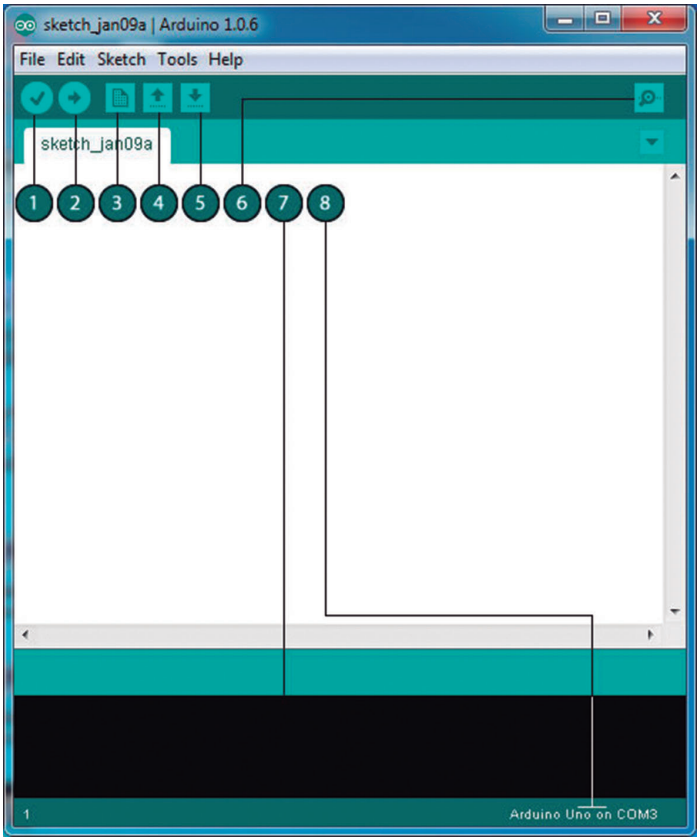
Figura 23 – Central de Aplicativos Ubuntu



1.5.3 UTILITÁRIOS

A Figura 24 mostra a interface do aplicativo visível ao usuário e destaca os pontos importantes para a utilização da IDE.

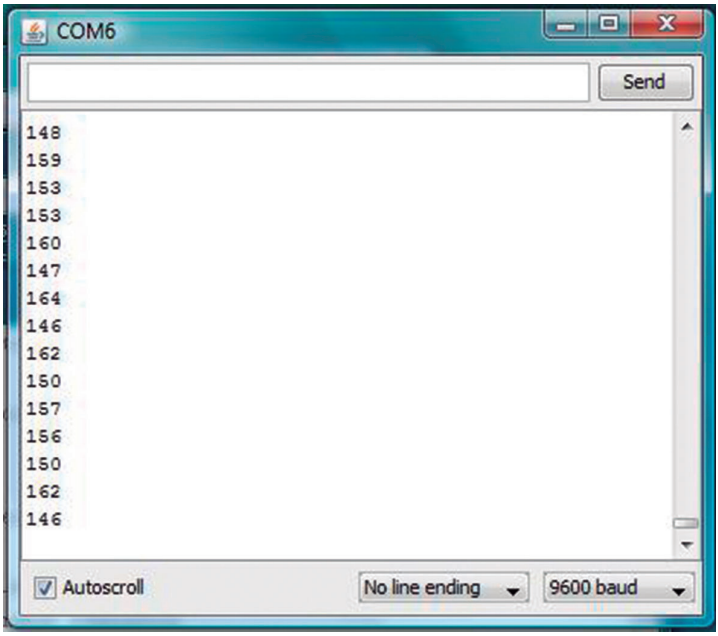
Figura 24 – Interface do Arduino



1. ***Verify***: Executa uma varredura a procura de possíveis erros de sintaxe no código. Caso encontre algum erro, este será mostrado na área negra, na parte inferior da interface, sem precisar compilá-lo, o que economiza tempo caso o programador não tenha uma placa Arduino por perto para testar seu sketch.
2. ***Upload***: Carrega o sketch para o Arduino. Caso ocorra algum erro durante o processo será mostrado na área negra na parte inferior da interface.
3. ***New***: Abre um novo documento em branco.

4. ***Open***: Expõe um sketch existente.
5. ***Save***: Salva o sketch corrente.
6. ***Serial Monitor***: Mostra uma janela como a exibida na Figura 25. Ela é responsável por exibir os dados enviados e recebidos do Arduino enquanto ele estiver em execução.

Figura 25 – Monitor Serial



7. ***Informações:*** Área negra responsável por exibir mensagens de erro e outras informações referentes ao sketch ou aos dados da conexão serial do Arduino.
8. ***Localização Serial:*** Informa qual porta serial está conectada ao Arduino.

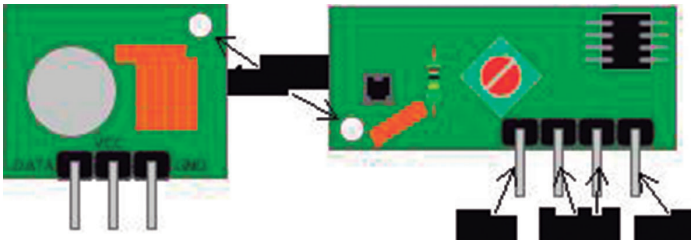
1.6 BIBLIOTECAS

Além das funções já disponíveis na linguagem do Arduino, o programador pode baixar algumas bibliotecas com estrutura de dados e funções já prontas. Ao adquirir uma expansão para o Arduino, como uma shield, é comum que ela venha com uma biblioteca para facilitar a sua utilização. As seções seguintes exploram algumas bibliotecas disponíveis.

1.6.1 VIRTUAL WIRE

A biblioteca **virtualwire** é utilizada para a comunicação via rádio. Nesse tutorial ele será usado nos módulos RF[15], mostrado na Figura 26 e presentes no projeto do carro controlado por controle remoto.

Figura 26 – Módulos Transmissor e Receptor RF



É possível observar que todas as funções da biblioteca referentes ao transmissor possuem parte da nomenclatura com **_ tx** enquanto ao receptor, **_ rx**. As funções que não apresentam nenhum desses fragmentos fazem referência a ambos. As principais funções da biblioteca **virtualwire** são:

- **vw _ setup(int bitsec):** define a velocidade em bits/segundo enviados do módulo transmissor ao módulo receptor. A variável **bitsec** é o único parâmetro da função e recebe a quantidade de bits/segundo a ser enviada na comunicação;
- **vw _ set _ tx _ pin(uint8 _ t pin):** estabelece o pino no Arduino que será utilizado para a transmissão de dados do módulo transmissor RF. Se essa função não é chamada, por padrão, os pinos utilizados serão os pinos 12 e 11. A variável **pin** recebe o número equivalente ao pino do Arduino. A mudança deve ser feita antes de chamar a função **vw _ setup()**.
- **vw _ str _ rx _ pin(uint8 _ t pin):** sua utilização vem da necessidade explicada anteriormente, porém, determina um novo pino para o módulo receptor RF. A variável **pin**, único parâmetro da função, recebe o número equivalente ao pino do Arduino que passará a receber os dados. A mudança deve ser feita antes de chamar a função **vw _ setup()**.
- **vw _ send(uint8 _ t* msg, uint8 _ t tam):** efetua o envio de uma mensagem (sequência de dados)qualificada como

15 Módulos RF ou TF modules ou *Radio Frequency Module* é um pequeno aparato eletrônico utilizado em comunicações de sinais de rádio, podendo ser transmissor ou receptor.

um *array* do tipo **uint8_t**[16], que é um ponteiro **msg** como primeiro parâmetro da função. O segundo parâmetro **tam** se trata de uma variável, também do tipo **uint8_t**, que recebe o tamanho do *array* a ser enviado.

- **vw_wait_tx():** faz o código esperar para que todos os dados sejam devidamente transmitidos e logo após dar continuidade ao sketch.

16 uint8_t é um tipo de dado da linguagem C que representa um byte sem sinal, ou seja, pode ser interpretado como um char (sem sinal) ou mesmo um tipo byte (sem sinal).

CAPÍTULO 2

CONSTRUINDO O CARRO COM CONTROLE REMOTO

O presente capítulo descreve a construção de um carro controlado por controle remoto e é dividido em 3 partes: construção do carro; construção do controle remoto; comunicação entre carro e controle. A seção 2.4 – Construção do Carro, delinea a montagem do circuito do carro, a alimentação e a programação dos movimentos executados pelo carro. A seção 2.5 – Construção do Controle, detalha a montagem do joystick, Arduino, a alimentação do circuito e a programação das configurações de comando. Por fim, a seção 2.6 – Comunicação Entre Carro e Controle, expõe a montagem do transmissor, do receptor e do código utilizado para a comunicação.

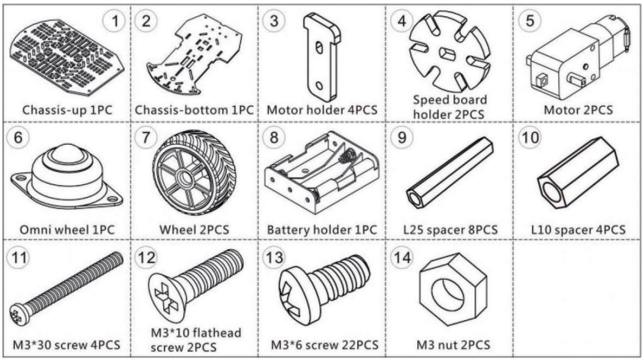
2.1 CONSTRUÇÃO DO CARRO

2.1.1 COMPONENTES NECESSÁRIOS

Um Kit plataforma robótica Magician: As peças do kit podem ser visualizadas na Figura 27 e constam as seguintes peças:

- Chassis (inferior e superior);
- Suportes de fixação para motores;
- Conjunto de porcas e parafusos M3 de 6 mm e 30 mm;
- Rodas;
- Motores DC;
- 7 Espaçadores L25 e 2 espaçadores L10;
- 1 Rodizio omni-direcional.

Figura 27 – Plataforma Magician

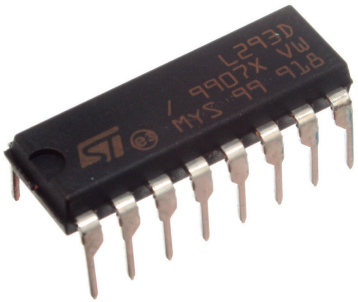


Fonte: (SPARKFUN ELETRONICS, 2013)

Essa plataforma é facilmente encontrada nas lojas especializadas em robótica disponíveis na internet.Na seção 4 – Links Úteis, contém alguns links para a aquisição do kit.

- **Arduino** – peça chave para a construção do veículo e um dos pontos principais desse livro;
- **Bateria 9V** – as duas baterias utilizadas no tutorial são de 9 V e podem ser encontradas em qualquer loja de eletrônica ou, até mesmo, em supermercados.
- **Protoboard** – protoboard (ou placa de ensaio) é o componente em que se montam os circuitos do carro e do controle. Aqui serão conectados os constituintes eletrônicos do projeto.
- **Chip L293D** – esse microcontrolador (Figura 28) será utilizado para o controle dos motores do carro.

Figura 28 – Microcontrolador L293D



Fonte: (WIKIMEDIA, 2009)

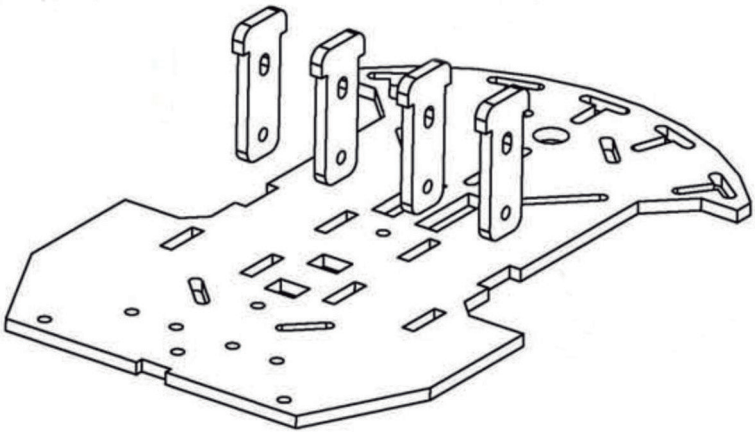
2.1.2 CONSTRUÇÃO DO CIRCUITO

2.1.2.1 Montagem do Chassi

1° Passo: Identificar o chassi inferior e o superior(na Figura 27, a imagem 1 exibe o chassi superior e a imagem 2 exibe o inferior). Este possui tantos furos quanto possível para facilitar o encaixe dos componentes com parafusos e aquele possui aberturas laterais para comportar as rodas e alguns furos maiores para redutores, suportes, encoders e outros componentes.

2° Passo: Posicionar os suportes para fixação dos motores no chassi inferior, como mostra a Figura 29. Cada par de suportes comportará um motor entre eles. O chassi inferior possui a forma simétrica, então não há um lado correto para o encaixe, porém é necessário ter em vista que os suportes devem ser dispostospara baixo.

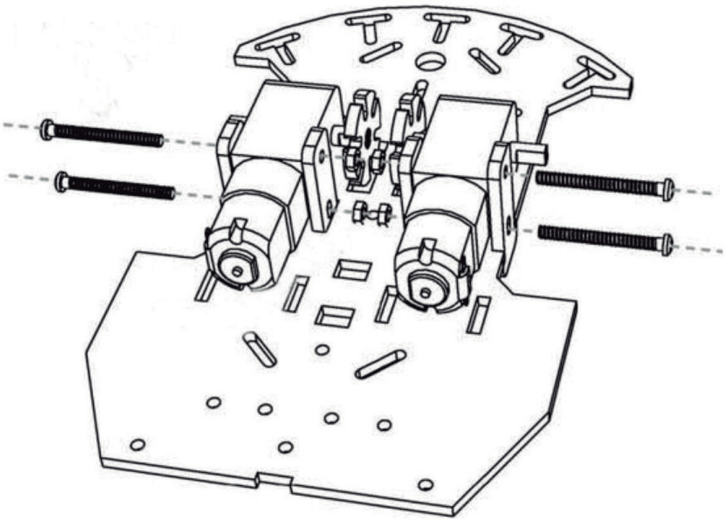
Figura 29 – Chassi Inferior com os Suportes dos Motores



Fonte: (SPARKFUN ELETRONICS, 2013)

3° Passo: Fixar os encodes rotativos nos motores., que devem ficar voltados para o espaço entre os motores. Em seguida, fixar os motores no suporte com dois parafusos M3 x 30 mm e com porcas M3. Os suportes possuem dois furos cada, deixando os motores bem fixados. É importante que os fios dos motores estejam virados para fora, facilitando, assim, o encaixe deles no circuito. As peças de encaixe das rodas devem estar posicionadas no centro do local do chassi reservado para elas, tal como na Figura 30.

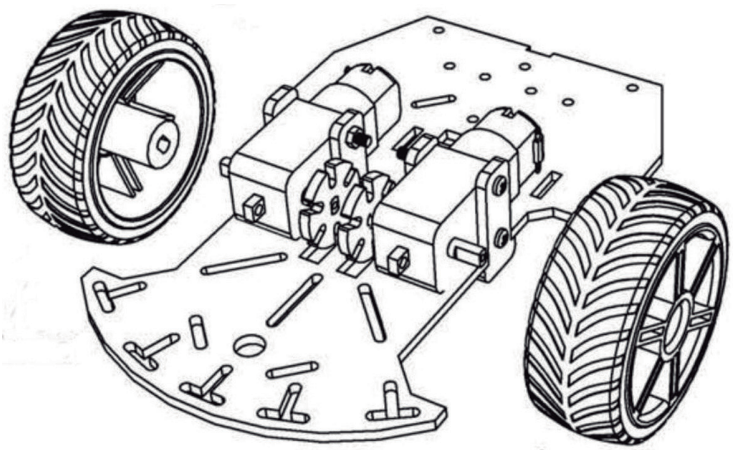
Figura 30 – Chassi Inferior com os Motores



Fonte: (SPARKFUN ELETRONICS, 2013)

4° Passo: Encaixar as rodas nos motores pressionando-as corretamente, como mostra a Figura 31. É um encaixe suave, porém muito resistente. Caso as rodas fiquem muito encostadas nos motores, afaste-as um pouco com uma ligeira folga, mas que seja idêntica nas duas para não comprometer o movimento.

Figura 31 – Chassi Inferior com as Rodas



Fonte: (SPARKFUN ELETRONICS, 2013)

5° Passo: Prender os espaçadores L25 com dois parafusos M3 x 6 mm nas posições exibidas na Figura 32, onde está o rodízio omnidirecional. Em seguida, o rodízio omnidirecional deve ser encaixado nos espaçadores L25 com dois parafusos M3 x 6 mm, essa peça ideal para dar suporte a qualquer movimento do carro.

6º Passo: Assentar o suporte para pilhas usando dois parafusos M3 x 10 mm de cabeça achatada e duas porcas M3, como mostra a Figura 33.

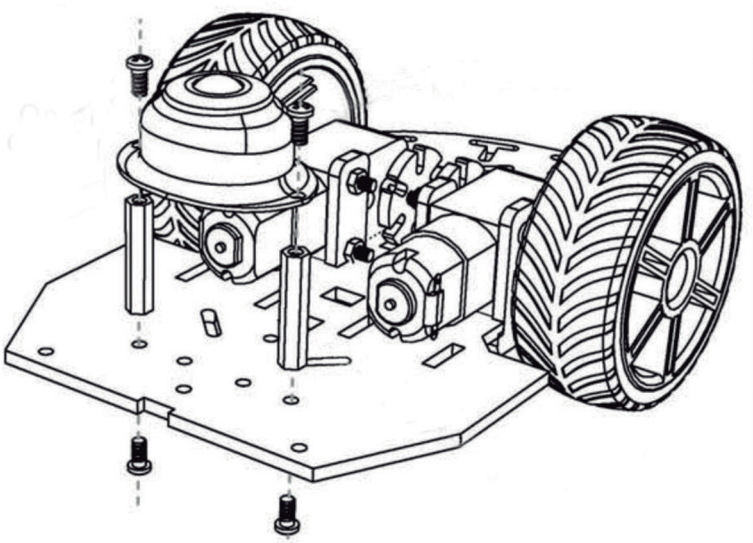
Assim, todos os componentes sob o chassi inferior já estão encaixados.

7° Passo: Inserircinco espaçadores L25 no chassi usando parafusos M3 x 6 mm nas posições exibidas na Figura 34. Para obter um espaço maior entre o chassi inferior e superior, basta utilizar espaçadores L35 ou L45 no lugar dos L25.

8° Passo: Acrescentar dois espaçadores L10 no chassi superior usando parafusos M3 x 6 mm, eles devem estar a aproximadamente 6,5 cm um do outro e próximos ao ponto marcado com um "A", na Figura 35, acima do pack de pilhas. Nesses espaçadores será fixado o Arduino.

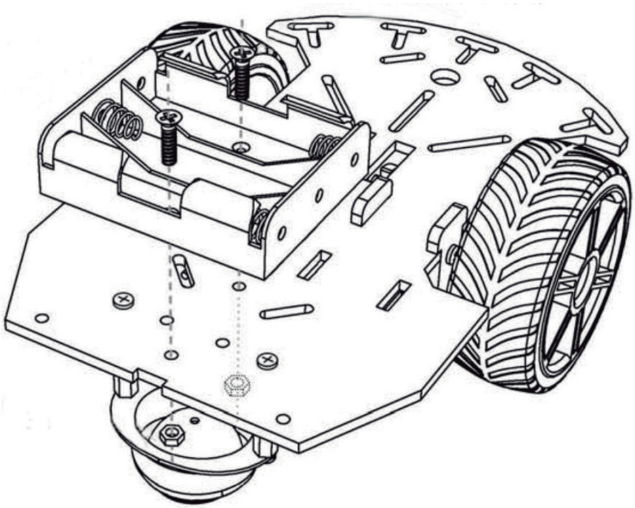
Posicionar o chassi superior sobre os espaçadores L25 usando cinco parafusos M3 x 6 mm, como mostra a Figura 35. Os parafusos foram definidos de forma a não atrapalhar o encaixe de outras peças, mas podem facilmente ter suas posições modificadas de acordo com a necessidade.

Figura 32 – Chassi Inferior com o Rodízio Omnidirecional



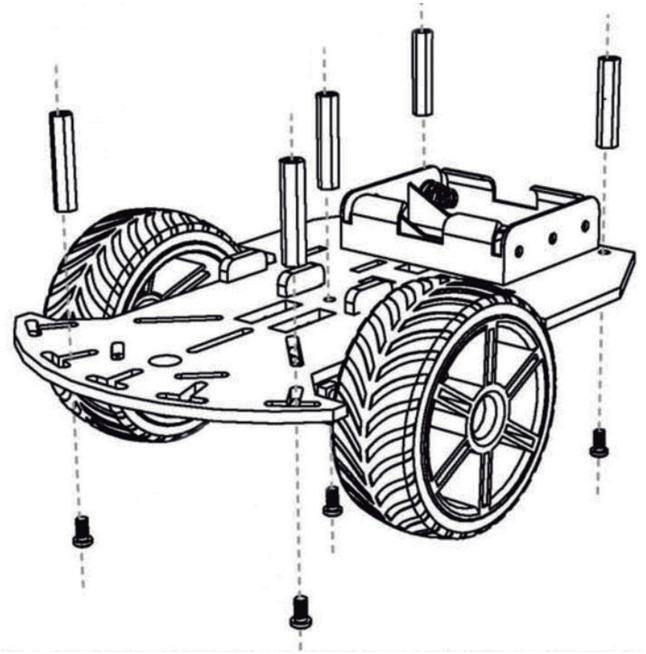
Fonte: (SPARKFUN ELETRONICS, 2013)

Figura 33 – Chassi Inferior com o Suporte de Pilhas



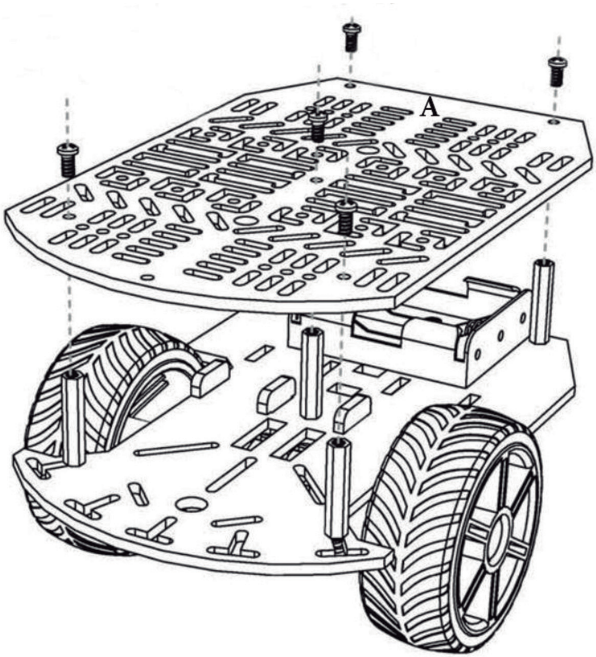
Fonte: (SPARKFUN ELETRONICS, 2013)

Figura 34 – Chassi Inferior com os Espaçadores do Chassi Superior



Fonte: (SPARKFUN ELETRONICS, 2013)

Figura 35 – Chassi Superior com Espaçadores



Fonte: (SPARKFUN ELETRONICS, 2013)

Cumprido o roteiro, o Chassi Magician está montado e pronto para se encaixar aos outros componentes. O Arduino, a protoboard e o controlador L293D serão fixados sobre o chassi superior e a alimentação do circuito será encaixada entre os chassis.

2.1.2.2 Conexão Com Arduino

Após a montagem do Chassi, o próximo passo é encaixar o Arduino e a protoboard. Essa etapa depende do posicionamento desejado dos componentes sobre o carro, sendo necessário que ambos fiquem em um local estratégico para que o veículo não possua um lado mais pesado que o outro.

Uma dica para o encaixe do Arduino é a seguinte: como mostra a Figura 35, a parte superior do chassi é formada por segmentos de fissuras que servem para encaixar alguns espaçadores que sobraram, possibilitando a montagem de uma base sólida, vale ressaltar que essa plataforma possui quatro furos para que seja encaixada nos espaçadores com parafusos.

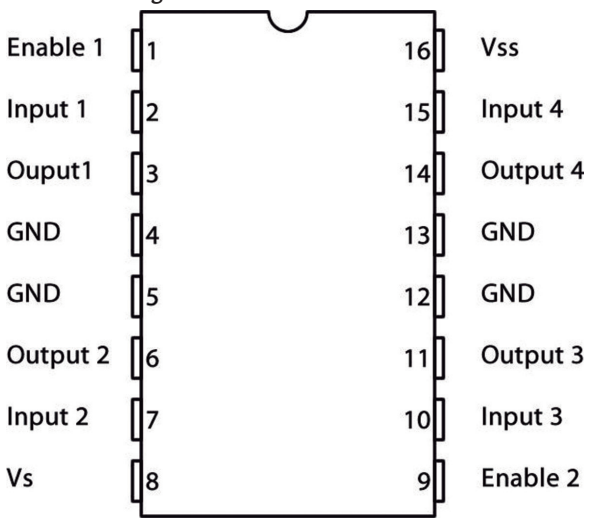
Para o encaixe da protoboard, dependendo do modelo, pode haver na parte traseira uma superfície adesiva para fixá-lo ao chassi.

Com o Arduino e a protoboard acoplados à plataforma já montada, para completar o circuito de movimento, basta apenas conectar os motores ao Arduino. Para isso é utilizado o microcontrolador L293D.

2.1.2.3 Controlador L293D

O L293D possui 2 sistemas de ponte Hque suportam até 600 mA de corrente cada, o que possibilita a controle de 2 motores com ponte H. Além disso, possui um sensor térmico que desliga o controlador caso exceda a temperatura limite, uma boa tolerância aos ruídos e suporta tensões de 4,5 V a 36 V. No projeto, são utilizados 2 motores que utilizam a referida ponte, os quais serão esmiuçados na seção 2.4.4 – Programação do Circuito. A Figura 36 exibe os pinos de conexão do L293D.

Figura 36 – Controlador L293D



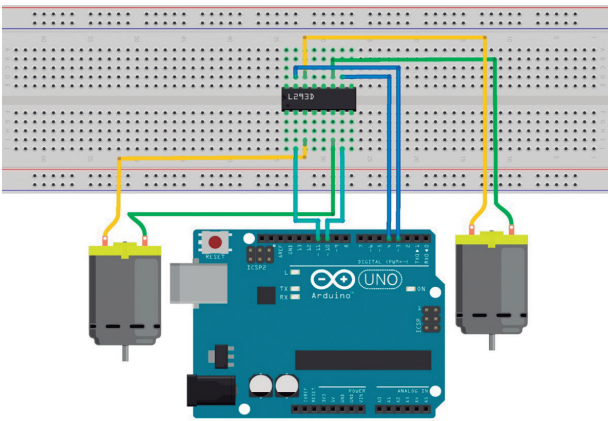
- **Enable 1** – pino que habilita a utilização do motor 1 (pinos ao lado esquerdo na Figura 36) quando é aplicada uma carga de 5 V;
- **Input 1** – pino de entrada que é conectado ao Arduino, em uma saída/entrada digital, para receber os comandos que controlam o motor 1;
- **Output 1** – pino de saída que é conectado ao motor 1, na entrada de alimentação, para transmitir o comando enviado do Arduino;
- **GND** – pino para se conectar ao GND do circuito, também utilizado pelo controlador como dissipador de calor, quando utilizado o motor 1;

- **GND** – mesma função do pino 4;
- **Output 2** – pino de saída que também é conectado ao motor 1, no fio de saída do motor, para fechar o circuito. Também faz a função do Output 1 e vice-versa;
- **Input 2** – mesma função do pino 2;
- **Supply Voltage (V_S)** – pino de alimentação do motor 1. Suporta, no mínimo, o mesmo valor do pino 16 (V_{SS}) e no máximo 36 V;
- **Enable 2** – pino que habilita a utilização do motor 2 (pinos ao lado direito na Figura 36) quando é aplicada uma carga de 5 V;
- **Input 3** – pino de entrada que é conectado ao Arduino, em uma saída/entrada digital, para receber os comandos que controlam o motor 2;
- **Output 3** – pino de saída que é conectado ao motor 2, na entrada de alimentação, para transmitir o comando enviado do Arduino;
- **GND** – pino para se conectar ao GND do circuito, também utilizado pelo controlador como dissipador de calor, quando ativado o motor 2;
- **GND** – mesma função do pino 12;
- **Output 4** – pino de saída que também é conectado ao motor 2, no fio de saída do motor, para fechar o circuito. Também faz a função do Output 3 e vice-versa;
- **Input 4** – mesma função do pino 10;
- **Logic Supply Voltage (V_{SS})** – pino de alimentação do motor 2. Suporta no mínimo 4,5 V e no máximo 36 V.

A Figura 37 exibe as conexões necessárias entre os motores e o L293D na protoboard com todos os componentes já encaixados no carro. Os fios azul-claros estão conectados aos pinos 10 e 9 da plataforma Arduino, que controlam o motor à esquerda (input 1 e input 2 do controlador L293D, respectivamente), enquanto os fios azul-escuros estão conectados aos pinos 4 e 3, que controlam o motor à direita (input 4 e input 3 do controlador L293D, respectivamente). Os fios amarelos propiciam as entradas de dados (geralmente, o fio vermelho) dos motores e os fios verdes asseguram os fechamentos dos circuitos (geralmente, o fio preto). Tanto o fio amarelo, quanto a fio verde recebem os comandos dos pinos digitais do Arduino.

É necessário cuidado ao conectar os fios, pois apesar de não haver possibilidade de danos a qualquer componente do circuito, uma inversão, por menor que seja, pode comprometer o funcionamento do software, devido à utilização da ponte H. No próximo tópico, inicia-se a construção da alimentação do circuito.

Figura 37 – Circuito com L293D



2.1.2.4 Alimentação

O conhecimento da alimentação exigida pelo circuito é importante, pois facilita a aquisição de baterias responsáveis por manter o sistema funcionando durante um período maior de tempo. Para adquirir essa informação, é preciso realizar uma soma da quantidade de corrente elétricanecessária para suprir todos os componentes do circuitoe conhecer a capacidade de carga fornecida pela fonte de alimentação utilizada. Cabe frisar que esses dados, como exposto nas baterias mostradas na Figura 38, são comunicados no rótulo dos elemento que nutrem o sistemas, a tensão é expressa em volts e a corrente fornecida em Ah – sigla correspondente à unidade ampère-hora.

Figura 38 – Baterias de 1.25 V e 2500 mAh

Fonte: (WIKIMEDIA, 2005)

Vale explicitar, ainda, que a unidade Ah refere-se à quantidade de corrente que uma bateria é capaz de fornecer durante certo período de tempo, nesse caso,1Ah é equivalente a 3600 Coulombs – Coulomb é a unidade de medida para a carga elétrica no SI (Sistema Internacional de Unidades) definida como a quantidade de carga transportada em um segundo pela corrente de 1 ampère (RODITI, 2005). Sendo assim, 1 Ah corresponde a uma corrente constante de 1 ampère durante um tempo de 1 hora, ou 3600 segundos, em um circuito. Essa informação faz referência a Equação 4.

Equação 4 – Resultante Ampère-hora

$$Ah = A * h$$

Fonte: (JUNIOR, 2013)

Em que "A" se refere à quantidade de ampères e "h" à quantidade de tempo, em horas, ou seja, uma fonte de alimentação com a capacidade 2 Ah, na prática, deverá fornecer a corrente de 2 ampères por 1 hora, com a tensão especificada no rótulo. Por exemplo, as baterias da **Figura 38** fornecem uma corrente de 2500 mAh com uma tensão de 1,25 V por um período de 1 hora.

Como os componentes utilizados no circuito criado possuem uma grande fragilidade a altas correntes, as fontes de energia utilizadas para a alimentação não apresentam em seu rótulo a unidade "Ah" (ampère-hora), mas sim a unidade "mAh" (miliampère-hora), como évisto na **Figura 38**. Trata-se de uma medida para correntes pequenas, nesse caso, 1 mAh equivale à 0,001 Ah ou 1 Ah é equivalente a 1000 mAh.

Outro conhecimento importante diz respeito à da tensão suportada pelos componentes do circuito. Esse aspecto é medido em Volts e, como já foi citado, sua quantidade está presente no rótulo da fonte de alimentação

Para chegar à informação do tempo de funcionamento do circuito, devem-se cruzar as informações da fonte de energia com a quantidade de corrente exigida pelo circuito. Contudo, os cálculos realizados não apresentarão um resultado exato do tempo em que o circuito funcionará, e sim uma aproximação. Mas, geralmente, um componente ou circuito que consome 250 mA, na prática, será executado durante 1 hora com uma bateria de 250 mAh.

O cálculo para descobrir o tempo M de funcionamento de um circuito alimentado por uma quantidade de corrente em mA é realizado a partir da quantidade de miliampere-hora (mAh) fornecida pela fonte de alimentação, como descrito na Equação 5.

Equação 5 – Cálculo de Tempo de Funcionamento do Circuito

$$M = \left(\frac{mAh}{mA} \right) * 60$$

Fonte: (JUNIOR, 2013)

Todos os componentes eletrônicos encontrados no mercado vêm com as informações referentes à capacidade de tensão e ao consumo de corrente em seus respectivos *datasheet*, normalmente disponibilizados no site do fabricante.

As informações elétricas dos componentes utilizados no circuito até o momento se encontram na **Tabela 2,** as quais foram extraídas do *datasheet* dos motores[1], do controlador L293D[2] e do próprio Arduino[3].

Tabela 2 – Informações Elétricas dos Componentes do Circuito

Componente	Corrente utilizada	Tensão recomendada
2 Motores	800 mA	+/- 4,5 V
Controlador L293D	600 mA	4.5 – 36 V
Arduino	160 mA	7 – 12 V

De acordo com o valor referente à corrente utilizada no Arduino (mostrado na tabela), e considerando que são utilizados 4 pinos de entrada/saída para a conexão dos motores com o controlador L293D, cada pino da placa necessita de uma corrente de 40 mA para funcionar. totalizando, assim, os 160 mA. Ademais, o valor total de corrente consumida pelo circuito é 1560 mA e sua alimentação deve vir de uma fonte com, no mínimo, 4,5 V e, no máximo, 12 V.

Para o circuito é recomendável utilizar pilhas AA de 1,5 V cada, facilmente encontradas no mercado, as quais serão inseridas no suporte que faz parte do Kit Magician. A quantidade de volts dessa fonte de energia é igual a soma da tensão de todos os componentes, dessa forma, 4 pilhas com 1,5 V cada resultará em uma carga de 6 V; e a corrente cedida é igual a soma das 4 pilhas em mAh. Com a fonte de alimentação já definida e o conhecimento da quantidade de mAh fornecida por ela, é possível realizar a expressão 14 e obter como resultante o tempo de funcionamento do

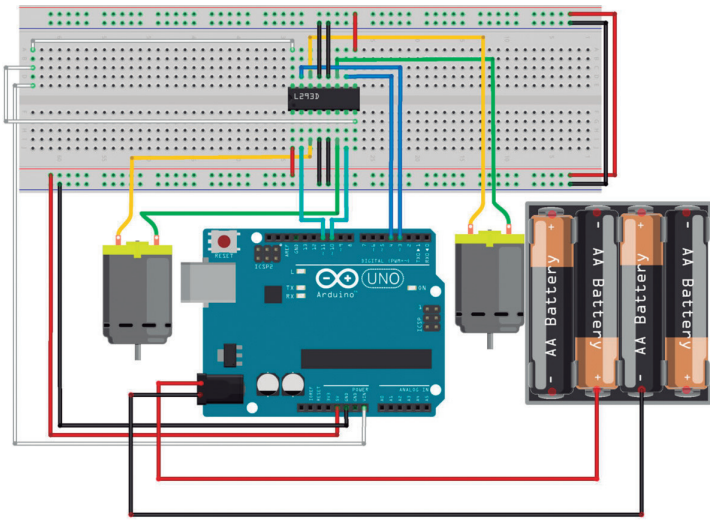
1 Disponível em: https://cdn.sparkfun.com/datasheets/Robotics/DG01D.pdf.

2 Disponível em: https://www.arduino.cc/documents/datasheets/L293D.pdf.

3 Disponível em: https://store.arduino.cc/arduino-uno-rev3.

circuito., porém, vale lembrar que, quando a alimentação chegar perto do fim, o circuito já não trabalhará com os resultados esperados. Em outras palavras, o carro parará de andar bem antes da corrente acabar, pois chegará um instante em que a força cedida pela fonte não será potente o suficiente para mover os motores sob o peso do carro. A **Figura 39** ilustra o protótipo quando as baterias são adicionadas.

Figura 39 – Circuito com Alimentação



Na Figura 39 a saída do controlador L293D VSS e VS que faz referência à alimentação dos motores estão conectados ao pino VIN do Arduino, que replica a mesma tensão fornecida pela bateria.

2.1.2.5 Análise do Circuito

O circuito resultante do roteiro revelado é mostrado na Figura 39 e suas características principais são:

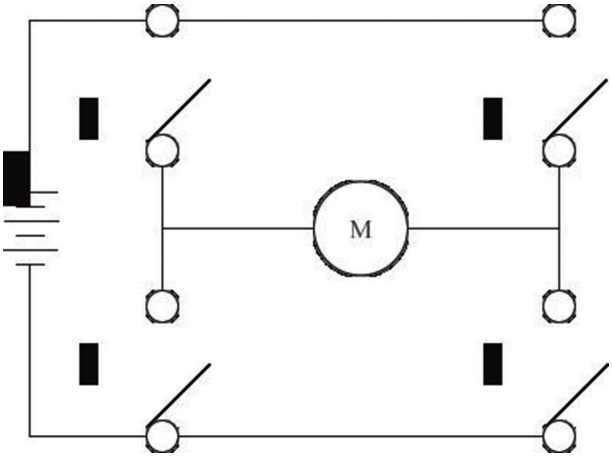
- Pinos digitais 3, 4 utilizados para conexão de um dos motores através do microcontrolador L293D e os pinos digitais 10 e 11, para a conexão com o outro motor.
- Pino VIN do Arduino cedendo alimentação direta da bateria para os pinos do controlador L293D, correspondente à alimentação dos motores.
- Alimentação cedida por uma fonte de 9 V, não importando de onde vem (baterias ou pilhas).

2.1.3 PROGRAMAÇÃO DO CIRCUITO

2.1.3.1 Funcionalidades

O controlador L293D controla até dois motores DC usando ponte H, exibida na **Figura 40**, a qual permite que cada motor possa ser induzido a rodar para os dois lados. Esse esquema é feito com as quatro entradas Input do L293D: Input 1 e 2, que estão ligados ao motor da esquerda, e os Input 3 e 4, que estão ligados ao motor da direita.

Figura 40 – Esquema de Ponte H



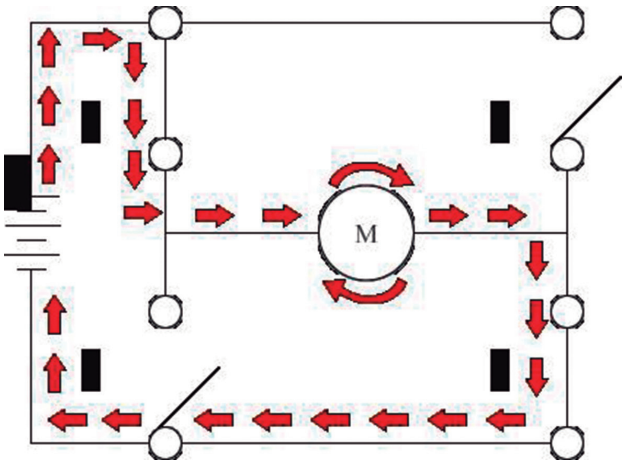
Fonte: adaptado (EVANS, NOBLE e HOCHENBAUM, 2013)

Quando um dos pinos Input recebe o valor **`HIGH`**, é feita a ligação de duas chaves da ponte H (podendo ser as chaves 1 e 4 ou as chaves 2 e 3, como ilustrado na **Figura 40**), sendo que o outro Input conectado ao motor deve receber o valor **`LOW`**. Essa ligação fará o motor girar em um determinado sentido, por exemplo, se as chaves 1 e 4 forem conectadas e o motor rodar para o lado direito, sentido horário (como mostra a **Figura 41**), então a ligação das chaves 2 e 3 rodará o motor para o lado esquerdo, sentido anti-horário (como mostra a **Figura 42**).

Cada motor utilizado porta dois pinos Input (Input 1 e 2 estão ligados ao motor esquerdo, Input 3 e 4 estão ligados ao motor direito) a partir dos quais são controlados, além disso, cada pino conecta duas chaves que permitem, também, o comando do esquema de ponte H, com base na atribuição **`HIGH`** ou **`LOW`**. Por exemplo: se atribuindo o valor **`HIGH`** ao pino Input 1 e o valor **`LOW`** ao Input 2 e o motor esquerdo rodar para o sentido horário, então pode-se deduzir que atribuindo o valor **`HIGH`** ao

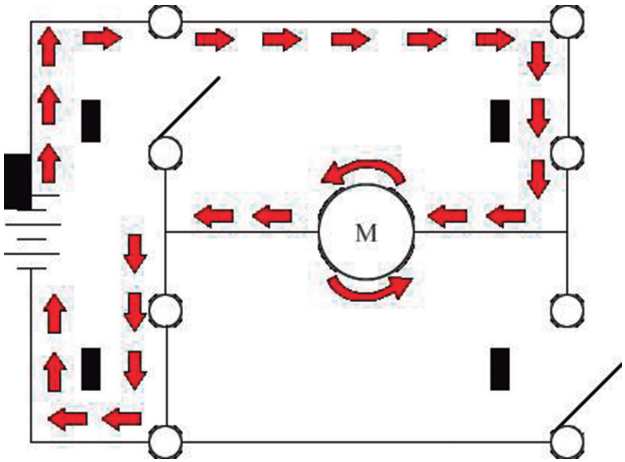
Input 2 e o valor **LOW** ao Input 1 o motor esquerdo rodará para o sentido oposto (anti-horário); o mesmo acontece com o motor direito quando manipulados os pinos Input 3 e 4. É exibido na **Figura 41** a esquemática de ponte H para o motor (representado pela letra "M") rodar no sentido horário e na **Figura 42** é exibido a esquemática de ponte H para o motor girar no sentido anti-horário.

Figura 41 – Ponte H Girando o Motor no Sentido Horário



Fonte: adaptado (EVANS, NOBLE e HOCHENBAUM, 2013)

Figura 42 – Ponte H Girando o Motor no Sentido Anti-horário



Fonte: adaptado (EVANS, NOBLE e HOCHENBAUM, 2013)

As portas lógicas Input são acionadas pelo comando `digitalWrite()` da linguagem do Arduino, passando como parâmetros o pino da plataforma conectada à porta lógica Input e o valor lógico desejado (`HIGH` ou `LOW`). A Tabela 3 exibe as combinações de valores das portas lógicas Input do L293D utilizadas no código do carro com controle remoto, juntamente com os movimentos executadas pelos motores 1 (esquerdo) e 2 (direito) do carro, que podem ser retardados (trás), acelerados (frente) ou parados. Os movimentos e suas respectivas descrições estão na cor preta na Tabela 3 abaixo.

Tabela 3 – Combinações das Portas Lógicas no Carro com Controle Remoto

Nº	Input 1	Input 2	Input 3	Input 4	O carro faz o movimento de...
	Motor 1 – Esquerdo		Motor 2 – Direito		
1	LOW	LOW	LOW	LOW	Freio, pois ambos os motores estão parados.
	Parado		Parado		
2	LOW	HIGH	LOW	LOW	Curva à direita, pois apenas o motor esquerdo acelera.
	Frente		Parado		
3	LOW	LOW	LOW	HIGH	Curva à esquerda, pois apenas o motor direito acelera.
	Parado		Frente		
4	HIGH	LOW	HIGH	LOW	Retardação, pois os dois motores retardam (trás).
	Trás		Trás		
5	LOW	HIGH	LOW	HIGH	Aceleração, pois os dois motores aceleram (frente).
	Frente		Frente		

As combinações de valores (`HIGH` e `LOW`) nº 1, 2, 3, 4 e 5 nas portas lógicas Input, na **Tabela 3,** são referentes às ações executadas pelo carro. Todas as ações possíveis são descritas no Anexo **5.10** – Tabela Completa dos Valores das Portas Lógicas INPUT do L293D e as ações realizadas no carro autônomo serão vistas no capítulo 3.4.4 – **Programação do Circuito** do carro autônomo.

2.1.3.2 Código do Circuito

O **Código 10** mostra o código do carro com comandos para o L293D e para o controle dos motores. Há uma sequência de comandos no código do carro (linhas 21 a 38), incluindo a função `movimentar()`, `delay()`, e uma atribuição do novo valor do protocolo de comunicação na variável `msg`, com o intuito de facilitar a compreensão da lógica do circuito. Cabe citar que a estrutura da função `movimentar()` (linhas 42 a 82) será

mantida para o projeto completo. Ademais, o transmissor via radiofrequência, presente no controle, enviará um vetor com quatro caracteres para o receptor localizado no circuito do carro; os motores do carro irão se mover de acordo com a mensagem recebida, interpretada pela função **`movimentar()`** (linha 42).

Código 10 – Código do Carro

```
1  #include <stdlib.h>
2  char* msg; /*String que receberá o
   protocolo de comunicação entre o
   carro e o controle ("0101", "0102",
   "0103", "0104", "0105", "0201",
   "0202", "0203", "0204", "0205").*/
3  int input1 = 11;//-------------
4  int input2 = 10;// Declara os pinos
   de conexão
5  int input3 = 4;// com os motores.
6  int input4 = 3;//-----------
7
8  void setup()
9  {
10   pinMode(input1, OUTPUT);/*Define o
   pino 11 como saída.*/
11   pinMode(input2, OUTPUT);/*Define o
   pino 10 como saída.*/
12   pinMode(input3, OUTPUT);/*Define o
   pino 4 como saída.*/
13   pinMode(input4, OUTPUT);/*Define o
   pino 3 como saída.*/
14   Serial.begin(9600);/*Inicia o
   monitor serial com uma taxa de 9600
   bits por segundos.*/
15 }
16
17 void loop()
18 {
19 /*Grava na string "msg" a mensagem
```

```
   "0105".*/
20   msg = "0105";
21 /*Imprime o conteúdo da string "msg"
   no monitor serial.*/
22   Serial.println(msg);
23 /*Usa os parâmetros gravados na
   string "msg" na função movimentar.*/
24   movimentar(msg);
25 //Cria uma pausa de meio segundo.
26   delay(500);
27   msg = "0101";
28   movimentar(msg);
29   delay(500);
30   msg = "0205";
31   movimentar(msg);
32   delay(500);
33   msg = "0201";
34   movimentar(msg);
35   delay(500);
36   msg = "0103";
37   movimentar(msg);
38   delay(500);
39 }
40
41 void movimentar(char* comando)
42 {/*Se a string recebida pela função
   for igual a "0105" ou "0104", o
   carro se move para frente.*/
43   if (((comando[1] ==
   '1')&&(comando[3] == '5')) ||
   ((comando[1] == '1')&&(comando[3] ==
   '4')))
44   {
45     digitalWrite(input1, LOW);
46     digitalWrite(input2, HIGH);
47     digitalWrite(input3, LOW);
```

```
48      digitalWrite(input4, HIGH);
49    }
50 /*Se a string recebida pela função
   for igual a "0101" ou "0102", o
   carro se move para trás.*/
51    if (((comando[1] ==
   '1')&&(comando[3] == '1')) ||
   ((comando[1] == '1')&&(comando[3] ==
   '2')))
52    {
53      digitalWrite(input1, HIGH);
54      digitalWrite(input2, LOW);
55      digitalWrite(input3, HIGH);
56      digitalWrite(input4, LOW);
57    }
58 /*Se a string recebida pela função
   for igual a "0205" ou "0204", o
   carro curva para direita.*/
59    if (((comando[1] ==
   '2')&&(comando[3] == '5')) ||
   ((comando[1] == '2')&&(comando[3] ==
   '4')))
60    {
61      digitalWrite(input1, LOW);
62      digitalWrite(input2, HIGH);
63      digitalWrite(input3, LOW);
64      digitalWrite(input4, LOW);
65    }
66 /*Se a string recebida pela função
   for igual a "0201" ou "0202", o
   carro curva para esquerda.*/
67    if (((comando[1] ==
   '2')&&(comando[3] == '1')) ||
   ((comando[1] == '2')&&(comando[3] ==
   '2')))
68    {
```

```
69      digitalWrite(input1, LOW);
70      digitalWrite(input2, LOW);
71      digitalWrite(input3, LOW);
72      digitalWrite(input4, HIGH);
73    }
74 /*Se a string recebida pela função
   for igual a "0103" ou "0204",
   desliga todos os motores.*/
75    if (((comando[1] ==
   '1')&&(comando[3] == '3')) ||
   ((comando[1] == '2')&&(comando[3] ==
   '3')))
76    {
77      digitalWrite(input1, LOW);
78      digitalWrite(input2, LOW);
79      digitalWrite(input3, LOW);
80      digitalWrite(input4, LOW);
81    }
82 }
```

No início do código, após o acréscimo da biblioteca **stdlib** (linha 1), está a declaração das variáveis: a primeira variável **msg** (declarada na linha 3) receberá um valor que será repassado à função **movimentar()** no decorrer da execução da função **loop()** (linha 18) e as outras quatro variáveis inteiras (declaradas nas linhas 4, 5, 6 e 7) receberão o número dos pinos do Arduino conectados às variáveis **input** do L293D.

Na função **setup()** (linha 9), as variáveis **input1**, **input2**, **input3** e **input4** são definidas como saída de dados (linhas 11, 12, 13 e 14), pois elas são usadas como endereço para o envio dos comandos aos motores.

Na função **loop()**, estrutura principal do sketch, a variável **msg** recebe um valor composto por 4 números em forma de *string* (linhas 21, 28, 31, 34 e 37), que, posteriormente, serão a mensagem codificada recebida do controle. Contudo, enquanto ainda não o é, configura-se como um código digitado diretamente e interpretado na função **movimentar()** para exemplificar a utilização da função. O valor de **msg** é alternado para fazer uma demonstração dos cincos movimentos possíveis do carro, no **Código**

10 (linhas 21, 28, 31, 34 e 37). Além disso, há um *delay* (atraso) de 500 milissegundos entre um comando e outro (linhas 27, 30, 33, 36 e 39) que representa o tempo que o comando ficará sendo executado no carro.

A função **`movimentar()`** recebe como parâmetro uma *string* e compara os valores presentes nas posições 1 e 3 da *string* para determinar os comandos do carro. Os comandos são os seguintes:

- **0105** ou **0104** – move o carro para frente (linha 44 a 49);
- **0101** ou **0102** – move o carro para trás (linha 52 a 57);
- **0205** ou **0204** – vira o carro para a direita (linha 60 a 65);
- **0201** ou **0202** – vira o carro para a esquerda (linha 68 a 73);
- **0103** ou **0203** – para o carro (linha 76 a 81).

São muitas as possibilidades de comandos com quatro caracteres, além disso, os comandos apresentados podem ser utilizados no loop para criar um percurso junto a função **`delay()`** para controlar o tempo de execução.

A função **`Serial.begin(9600)`** é utilizada para inicializar o monitor serial (linha 15). Com ela é possível, também, executar testes para construir um percurso observando os valores das variáveis, que podem ser exibidos no monitor serial.

2.1.3.3 Análise do Funcionamento

Após seguir os passos descritos, o sketch deve ser carregado no Arduino e o carro deverá executar a sequência: frente, ré, direita e esquerda. Completando o código, torna possível a execução de testes para a compreensão mais aprofundada do circuito.

De maneira geral, o **COD10** pode ser modificado de diversas maneiras para fins de testes. Por exemplo, é possível criar um caminho para ser percorrido pelo carro a partir de valores inseridos na função **`movimentar()`**, da adição de novas combinações dentro dessa função para realizar outros movimentos além dos existentes (mover para trás, para frente, etc.) ou das opções de ligar (**`HIGH`**) e de desligar (**`LOW`**) as variáveis **`input1`**, **`input2`**, **`input3`** e **`input4`** que fazem referência aos pinos do microcontrolador L293D. Também há a possibilidade de alterar a configuração de tempo de execução de cada comando alterando o valor passado na função **`delay(500)`** (linhas 27, 30, 33, 36 e 39). Assim, é exequível observar que o carro vai executar as funções no tempo determinado nessa função.

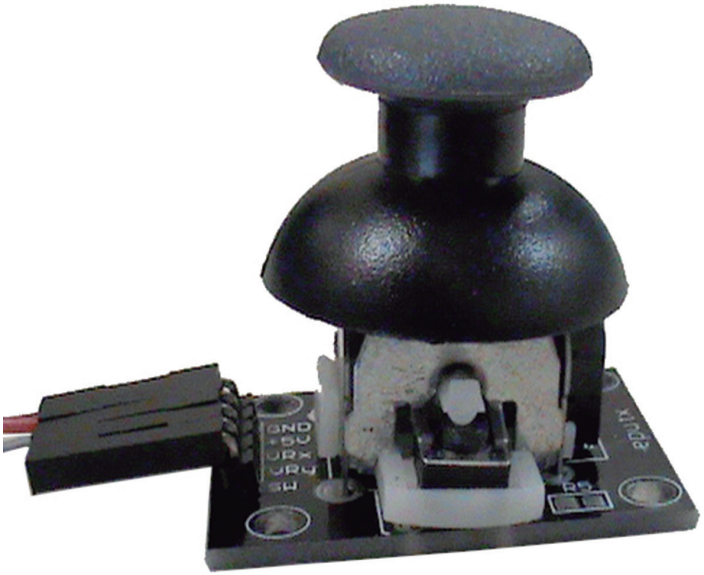
Pode-se, ainda, alterar o carro para que seja utilizado servo motores no lugar de motores DC ou, até mesmo, motores de passo. Na seção 4 – **Links Úteis** há links com informações e com tutoriais para utilizar os motores.

2.2 CONSTRUÇÃO DO CONTROLE

2.2.1 COMPONENTES NECESSÁRIOS

- **Joystick 3 Eixos (Figura 43)**: peça necessária para o controle dos movimentos do carro, trata-se de uma alavanca acrescida de um circuito;

Figura 43 – Joystick



- **Protoboard:** Sob ela são encaixados o joystick e o Arduino;
- **Arduino:** Como o controle é independente do veículo, é necessária uma placa Arduino somente para construção do controle.

2.2.2 CONSTRUÇÃO DO CIRCUITO

2.2.2.1 Montagem do Controle

A montagem do controle é simples, o foco deve estar no joystick, pois ele é o único componente presente nessa etapa que ainda não foi abordado. Esse item é composto por = um circuito e uma alavanca, a posição da segunda é transmitida ao Arduino por meio do primeiro, podendo ou não haver um botão extra. A alavacanca será conectada ao Arduino por meio do circuito para fornecer dados necessários de posicionamento.

Figura 44 – Saídas do Joystick

Na Figura 44 é possível observar 5 saídas do circuito. A função de cada uma é descrita a seguir:

- **GND:** deve ser conectada ao GND;
- **5V:** deve ser conectada a uma alimentação de 5 V;
- **X:** fornece a posição horizontal da alavanca para o Arduino através de uma entrada analógica;
- **Y:** informa a posição vertical da alavanca para o Arduino através de uma entrada analógica;
- **SW:** botão presente no circuito para funções diversas.

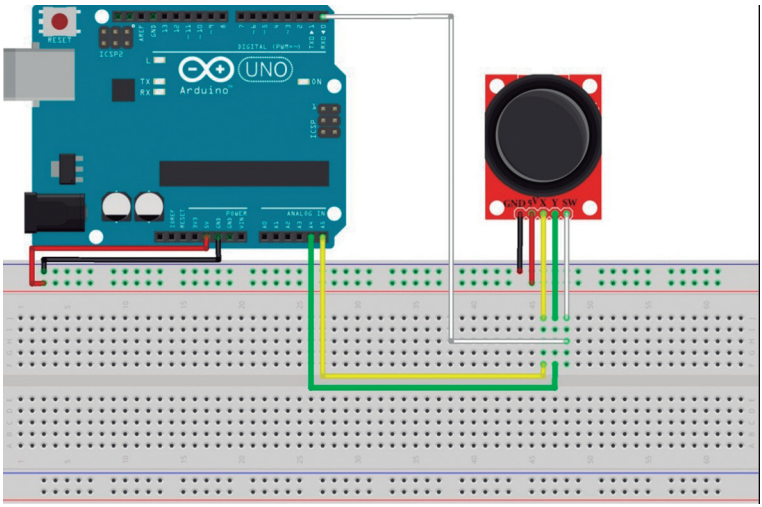
O circuito detalhado na Figura 45 será montado com o joystick:

2.2.2.2 Alimentação

A tensão suportada pelo joystick, de acordo com seu *datasheet*[4], é de 5 V. A Figura 45 mostra que a alimentação é fornecida pela saída de 5 V do Arduino que, por sua vez, é alimentado por qualquer fonte que conceda uma tensão de 7 V a 12 V.

4 Disponível em: https://www.parallax.com/sites/default/files/downloads/27800-2-Axis-Joystick-Documentation-v1.2.pdf.

Figura 45 – Circuito com Joystick



2.2.2.3 Análise do Circuito

O circuito presente na Figura 45 é suficiente para colocar em funcionamento o joystick. Nesse circuito é possível observar as seguintes características:

- A alimentação fornecida ao joystick provém da saída de 5V do Arduino;
- Os pinos que conectam as saídas X e Y do joystick são unidos às entradas analógicas A5 e A4 do Arduino, respectivamente;
- O botão do joystick é conectado a uma entrada digital do Arduino (não é utilizado no projeto).

2.2.3 PROGRAMAÇÃO DO CIRCUITO

2.2.3.1 Funcionalidades

Como explicado em tópicos anteriores, o joystick utilizado para a movimentação do carro é um circuito integrado a uma alavanca. No circuito há duas saídas “X” e “Y” que fornecem ao Arduino a posição da alavanca por meio de coordenadas em um plano.

Como a variância é muita, um simples toque na alavanca pode mudar em várias unidades as coordenadas posicionais, o que torna inviável o tratamento dessa mudança.Atécnica sugerida nesse tutorial é dividir o valor total das posições em 5 níveis, informando ao Arduino apenas quando algum desses forem alcançados. Assim, não é necessário

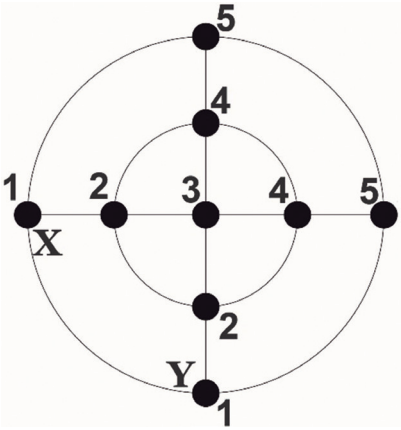
preocupar-se com manipulações insignificantes que, nos eixos "X" e "Y" – ou seja, nas posições horizontal e vertical, podem variar de 0 a 1023. Na Tabela 4, é exibida a relação entre os valores correspondentes aos intervalos de níveis da alavanca.

Tabela 4 – Intervalos de Níveis da Alavanca

Valor da Alavanca	Posição na Alavanca
0 a 204	5
205 a 409	4
410 a 614	3
615 a 819	2
820 a 1023	1

A Figura 46 representa os pontos de interesse que serão informados ao programa.

Figura 46 – Posições da Alavanca



É enviado ao programa, simultaneamente, a posição horizontal e vertical da alavanca. Para isso, é criado uma *string* de 4 caracteres, os dois primeiros são correspondentes, **01** para **X**, **02** para **Y** ou **03** para **Z,** desempenhados pelo botão presente no joystick (não utilizado no livro); os dois últimos, representam as posições que variam de 01 a 05. Por exemplo: o valor **0103** revelaque, na posição horizontal, a alavanca está no centro; o valor **0205** indica que, na posição vertical, a alavanca está em uma das extremidades.

2.2.3.2 Código do Circuito

O cd11 informa que, a cada loop, são lidas as posições X e Y da alavanca do joystick (linhas 18 e 20) e se o botão está pressionado (eixo Z, linha 22)a alavanca do joystick pode transmitir o comando de inúmeras posições, então, para não haver problemas com variações de valores pequenos, elas foram divididas em 5 níveis. CCabe esclarecer que os intervalos são de aproximadamente 204 posições(1023 / 5 = 204,6), resultando nos pontos da Figura 46, que mapeia as posições da alavanca. Essa divisão pode ser observada nas ações tomadas a partir da linha 24 do Código 11, nele, a partir de certo número, o valor a ser impresso no monitor serial muda. O Código 11 deve ser carregado no Arduino conectado ao Joystick:

Código 11 – Movimentação do Carro Usando o Joystick

```
1  /*No pino A5 é conectada a saída que
   recebe valores do eixo X do joystick
   (posição horizontal da alavanca).*/
2  int JoyStick_X = A5;
3  /*No pino A4 é conectada a saída que
   recebe valores do eixo Y do joystick
   (posição vertical da alavanca).*/
4  int JoyStick_Y = A4;
5  /*No pino 0 é conectada a saída
   que recebe valores do botão do
   joystick.*/
6  int JoyStick_Z = 0;
7
8  void setup()
9  {/*Define o pino 0 como entrada
   e ativa o resistor interno do
   Arduino*/
10   pinMode(JoyStick_Z, INPUT_PULLUP);
11 /*Inicia o monitor serial com uma
   taxa de 9600 bits por segundo.*/
12   Serial.begin(9600);
13 }
14
15 void loop()
16 {
17   int x, y, z;
18 /*Lê a posição horizontal atual da
   alavanca do joystick e armazena na
   variável "x".*/
```

```
19   x = analogRead (JoyStick_X);
20 /*Lê a posição vertical atual da
   alavanca no joystick e armazena na
   variável "y".*/
21   y = analogRead (JoyStick_Y);
22 /*Verifica o valor digital do botão
   do joystick e armazena o resultado
   na variável "z".*/
23   z = digitalRead (JoyStick_Z);
24
25 /*Caso a posição horizontal da
   alavanca for menor que 204:*/
26   if(x <= 204){
27 //É impresso o valor "0105" no
   monitor serial.
28     Serial.println("0105");
29 /*Caso a posição horizontal da
   alavanca estiver entre 204 e 410:*/
30   }else if((x > 204) && (x < 410)){
31 //É impresso o valor "0104" no
   monitor serial.
32     Serial.println("0104");
33 /*Caso a posição horizontal da
   alavanca estiver entre 410 e 615:*/
34   }else if((x >= 410) && (x < 615)){
35 //É impresso o valor "0103" no
   monitor serial.
36     Serial.println("0103");
37 /*Caso a posição horizontal da
   alavanca estiver entre 615 e 820:*/
38   }else if((x >= 615) && (x < 820)){
39 //É impresso o valor "0102" no
   monitor serial.
40     Serial.println("0102");
41 /*Caso a posição horizontal da
   alavanca for maior ou igual a 820:
   */
42   }else if(x >= 820){
43 //É impresso o valor "0101" no
   monitor serial
44     Serial.println("0101");
45   }
```

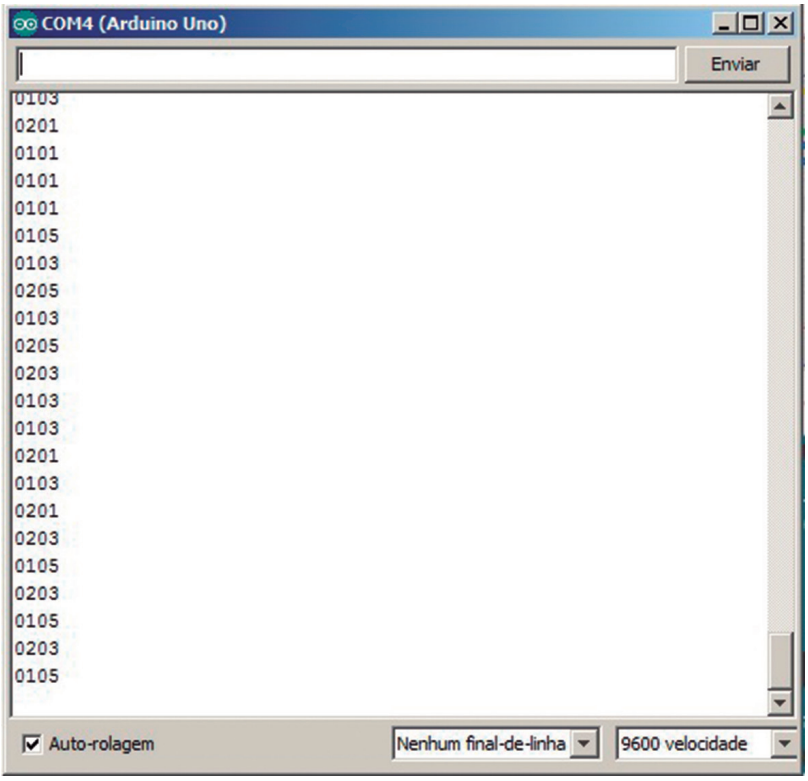
```
46
47 /*Caso a posição vertical da
   alavanca for menor que 204:*/
48   if(y <= 204){
49 //É impresso o valor "0205" no
   monitor serial.
50     Serial.println("0205");
51 /*Caso a posição vertical da
   alavanca estiver entre 204 e 410:*/
52   }else if((y > 204) && (y < 410)){
53 //É impresso o valor "0204" no
   monitor serial.
54     Serial.println("0204");
55 /*Caso a posição vertical da
   alavanca estiver entre 410 e 615:*/
56   }else if((y >= 410) && (y < 615)){
57 //É impresso o valor "0203" no
   monitor serial.
58     Serial.println("0203");
59 /*Caso a posição vertical da
   alavanca estiver entre 615 e 820:*/
60   }else if((y >= 615) && (y < 820)){
61 //É impresso o valor "0202" no
   monitor serial.
62     Serial.println("0202");
63 /*Caso a posição vertical da
   alavanca for maior que 820:*/
64   }else if(y >= 820){
65 //É impresso o valor "0201" no
   monitor serial.
66     Serial.println("0201");
67   }

68 /*O eixo "z" é o botão do joystick.
   Ele não é utilizado no projeto,
   porém a comunicação do botão com
   o carro já é estabelecida pelo
   código "0301". Pode ser programada
   alguma função para esse botão.
   Tal possibilidade é descrita no
   tópico 2.4.3.3 – Análise Geral do
   Funcionamento
```

```
69 if(z == 0){
70     Serial.println("0301");
71   }
72 }
```

2.2.3.3 *Análise do Funcionamento*

Quando carregado o código do tópico anterior no Arduino, é necessário abrir o monitor serial para observar os valores que estão sendo impressos. Se toda a montagem estiver correta, os valores exibidos no monitor serial (Figura 47) estarão de acordo com o plano descrito na Figura 46, com os dois primeiros dígitos revelando qual sentido da direção (01 para vertical e 02 para horizontal) e os dois últimos informando qual a posição da alavanca no sentido transmitido (01, 02, 03, 04 ou 05)

Figura 47 – Monitor Serial Exibindo Resultados do Receptor



2.3 COMUNICAÇÃO ENTRE CARRO E CONTROLE

2.3.1 COMPONENTES NECESSÁRIOS

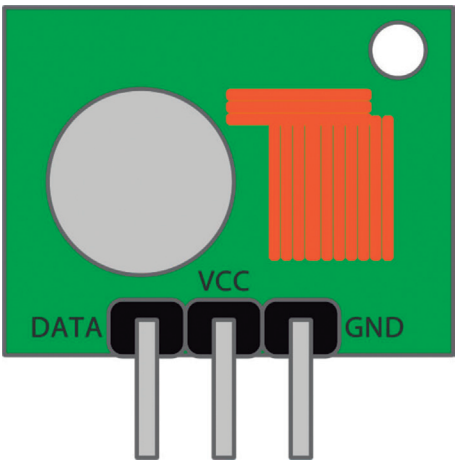
- RF Transmissor 433 Mhz AM;
- RF Receptor 433 Mhz AM.

2.3.1.1 Transmissor

O modelo do transmissor é MX-FS-03V, exibido na Figura 48. Ele possui as seguintes características de acordo com seu *datasheet*[5].

- Alcance: 20 – 200 metros (dependendo da tensão aplicada);
- Tensão: 3,5 – 12V;
- Modo de Operação: AM (Modulação em Amplitude);
- Taxa de transferência: 4KB/s;
- Potência de transmissão: 10mW;
- Frequência de transmissão: 433Mhz;
- Dimensões: 19 x 19mm.

Figura 48 – Transmissor MX-FS-03V



O transmissor possui três conectores, como exibido na Figura 48:

- Data: recebe os dados provenientes do Arduino. Esses dados são transmitidos via radiofrequência para o receptor conectado ao carro;
- GND: liga-se ao GND do circuito;

5 Disponível em https://www.simsso.de/?type=arduino/mx-05v-mx-fs-03v.

- V_{CC}: coleta a alimentação necessária ao transmissor.

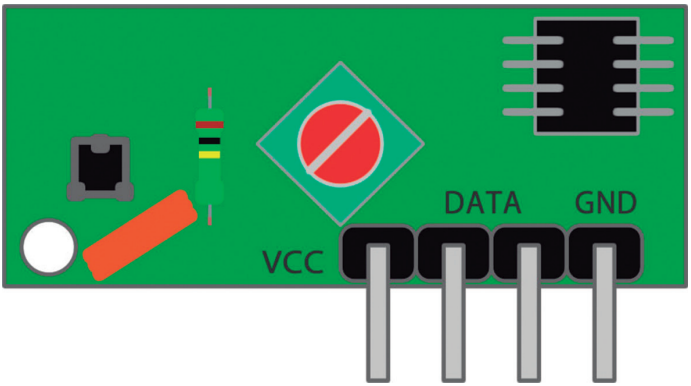
Geralmente, esse item é vendido em conjunto com o receptor MX-05V. Além disso, é usado juntamente com uma antena para melhor disseminação do sinal, a qual pode ser confeccionada com um fio de cobre e adaptada ao orifício presente na placa.

2.3.1.2 Receptor

O modelo do receptor é o MX-05V, mostrado na Figura 49. Possui as seguintes características, de acordo com o seu *datasheet*, disponível em https://www.simsso.de/?type=arduino/mx-05v-mx-fs-03v.

- Tensão: 5V DC;
- Corrente: 4mA;
- Frequência da Recepção: 433Mhz;
- Sensibilidade: -105dB;
- Dimensões: 30 x 14 x 7mm.

Figura 49 – Receptor MX-05V



O receptor possui quatro conectores, como exibido na Figura 49:

- Data: são responsáveis por transmitir ao Arduino a informação que a antena está recebendo via radiofrequência do transmissor no controle;
- VCC: é vinculado a uma fonte de alimentação de 5 V;
- GND: é ligado ao GND do circuito;

Esse objeto também é utilizado com uma antena para melhor recepção do sinal e, assim como o transmissor, sua antena é confeccionada com um fio de cobre e adaptada a um orifício presente na placa.

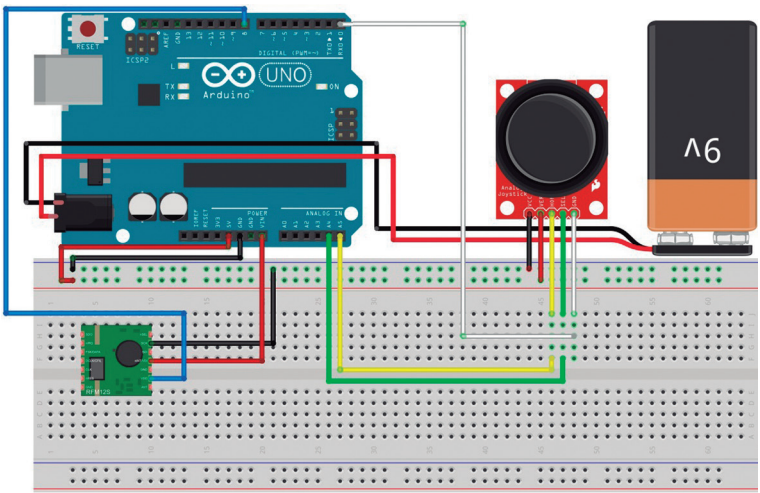
2.3.2 CONSTRUÇÃO DO CIRCUITO

2.3.2.1 Comunicação Entre Transmissor e Receptor

O método de comunicação empregado na interação existente entre o controle e o carro é a modulação por rádio frequência[6] (RF), utilizando o conjunto composto pelo transmissor MX-FS-03V e o receptor MX-05V.

O circuito do transmissor é adicionado ao do controle, como mostra a Figura 50.

Figura 50 – Circuito do Transmissor



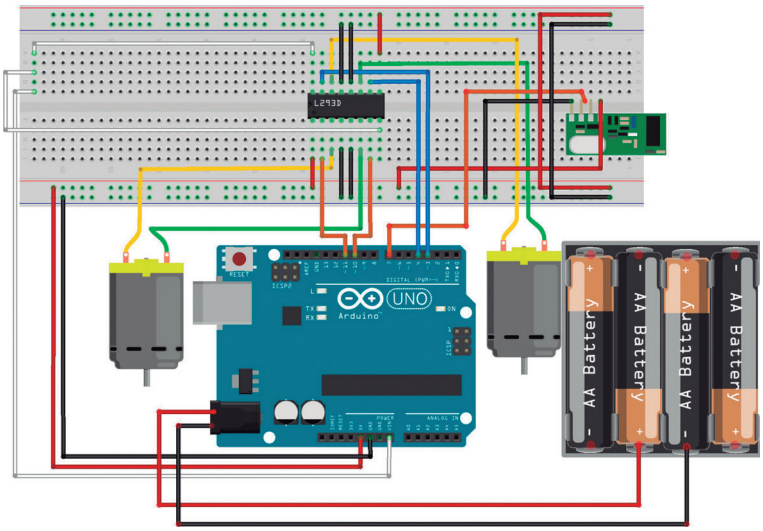
O V_{CC} do transmissor (fio vermelho) é conectado ao pino Vin para receber a carga de 9V fornecida pela bateria no Jack CC. O GND do transmissor (fio preto) está ligado à trilha GND da protoboard e a entrada de dados do transmissor (fio azul) está ligada ao pino 8 do Arduino.

O circuito do receptor é adicionado ao do carro, como mostra a Figura 51.

O V_{CC} do receptor (fio vermelho) é conectado à trilha de 5V fornecida pelo pino de 5V do Arduino; o GND (fio preto) é vinculado à trilha GND da protoboard; e a saída de dados do receptor (fio laranja) é ligada ao pino 7 do Arduino.

6 Radiofrequência (RF) é a faixa de frequência que abrange, aproximadamente, de 3 kHz a 300 GHz e que corresponde a frequência das ondas de rádio. RF geralmente se refere às oscilações eletromagnéticas ao invés de mecânicas nessa faixa de frequência, embora exista sistemas mecânicos em RF (RODITI, 2005).

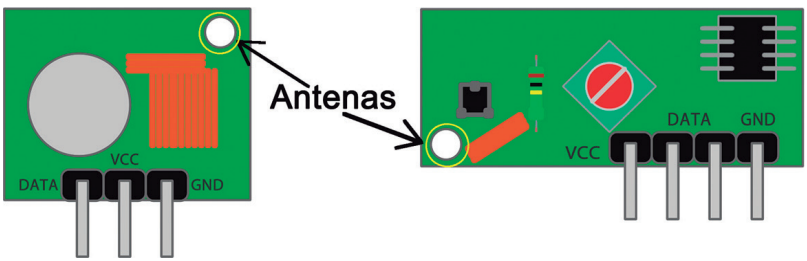
Figura 51 – Circuito do Receptor



2.3.2.2 Análise Geral do Circuito

O alcance máximo da transmissão depende da tensão utilizada pelo transmissor (até 200 metros utilizando uma tensão de 3,5 a 12 V). Para melhor distribuição e captura do sinal, pode ser adaptada uma antena para o transmissor e o receptor nos pontos indicados na Figura 52.

Figura 52 – Local para Conexão da Antena



O circuito do controle exige uma alimentação razoável, pois nele não há utilização de componentes que consomem muita energia, como um motor DC, por exemplo. Porém, o circuito do carro exige grande alimentação, visto que necessita de suprir dois motores DC e o L293D e ambos consomem muita carga. A Tabela 5 exibe as características elétricas dos circuitos construídos.

Tabela 5 – Comparação de Consumo de Energia Entre o Controle e o Carro

	Componente	Corrente utilizada	Tensão Recomendada
Carro	2 Motores	400 mA	+/- 4,5 V
	Controlador L293D	600 mA	4.5 – 36 V
	Arduino	160 mA	7 – 12 V
	Receptor	40 mA	3,5 a 12V
Controle	Transmissor	40 mA	5V
	Arduino	160 mA	7 – 12 V
	Joystick	80 mA	5 V

O consumo do circuito do carro soma o total de 1600 mA (1,2 A) e o consumo do circuito do controle soma o total de 280 mA, uma diferença de 1320 mA. Caso ambos, carro e controle, sejam alimentados por uma bateria de 280 mAh, utilizando a Equação 5, a duração da fonte de energia em cada um será equivalente a:

$$M = \left(\frac{mAh}{mA} \right) * 60$$

$$M = \left(\frac{280}{1600} \right) * 60 = 10,5$$

Duração no carro = 10,5 minutos

$$M = \left(\frac{mAh}{mA} \right) * 60$$

$$M = \left(\frac{280}{280} \right) * 60 = 60$$

Duração no controle = 60 minutos (1 hora)

2.3.3 PROGRAMAÇÃO DO CIRCUITO

2.3.3.1 Funcionalidades

Com o transmissor conectado ao controle e o receptor conectado ao carro, como mostrado nos circuitos da Figura 50 e da Figura 51, é possível adaptar em seus códigos as partes referentes ao conector e receptor. As atualizações dos códigos estão no Código 12 e no Código 13:

2.3.3.2 Código do Circuito

No Arduino presente no controle é carregado o Código 12.

Código 12 – Controle com Transmissor RF

```
1  #include <VirtualWire.h>
2  /*O pino A5 é conectado a saída
   que recebe os valores da posição
   horizontal da alavanca do
   joystick.*/
3  int JoyStick_X = A5;
4  /*O pino A4 é conectado a saída
   que recebe os valores da posição
   vertical da alavanca do joystick.*/
5  int JoyStick_Y = A4;
6  /*O pino 0 é conectado a saída que
   recebe o valor digital do botão do
   joystick.*/
7  int JoyStick_Z = 0;
8  void setup ()
9  {/*Define no pino 8 a conexão com o
   transmissor RF. Ele irá emitir os
   dados para o recepetor do carro.*/
10   vw_set_tx_pin(8);
11 /*Define a velocidade de transmissão
   de dados entre o transmissor e
   o receptor RF em 2000 bits por
   segundo.*/
12   vw_setup(2000);
13 /*Define o pino A5 como entrada de
   dados.*/
14   pinMode (JoyStick_X, INPUT);
15   pinMode (JoyStick_Y, INPUT);
16 /*Define a conexão do botão como
   entrada de dados e ativa o resistor
   interno do Arduino.*/
17   pinMode (JoyStick_Z, INPUT_
   PULLUP);
18   Serial.begin (9600);
```

```
19 }
20
21 void loop ()
22 {
23   int x, y, z;
24 /*A posição horizontal da alavanca
   do joystick é grava na variável "x",
   a posição vertical é gravada na
   variável "y" e o estado digital do
   botão é gravado na variável "z".*/
25   x = analogRead (JoyStick_X);
26   y = analogRead (JoyStick_Y);
27   z = digitalRead (JoyStick_Z);
28 /*Imprime no monitor serial os
   valores lidos das posições do
   joystick e do botão*/
29   Serial.print (x, DEC);
30   Serial.print (",");
31   Serial.print (y, DEC);
32   Serial.print (",");
33   Serial.println (z, DEC);
34
35 /*Caso a posição horizontal da
   alavanca do joystick estiver nos
   limites descritos na estrutura de
   decisão "if" é enviado uma string
   para função "enviar()".*/
36     if(x <= 204){
37   /*As strings enviadas correspondem
ao protocolo descrito no tópico 2.5.4*/
/*- Programação do Circuito*/
38         enviar("0105");
39       }else if((x < 410) && (x >
204)){
40         enviar("0104");
41       }else if((x >= 410) && (x <
615)){
42         enviar("0103");
```

```
43        }else if((x >= 615) && (x <
820)){
44          enviar("0102");
45        }else if(x >= 820){
46          enviar("0101");
47        }
48    /*Caso a posição vertical da
alavanca do joystick estiver nos
limites descritos na estrutura de
decisão "if" é enviado uma string para
função enviar().As strings enviadas
correspondem ao protocolo descrito
no tópico 2.5.4 - Programação do
Circuito*/
49        if(y <= 204){
50          enviar("0205");
51        }else if((y < 410) && (y >
204)){
52          enviar("0204");
53        }else if((y < 615) && (y >
410)){
54          enviar("0203");
55        }else if((y >= 615) && (y <
820)){
56          enviar("0202");
57        }else if(y >= 820){
58          enviar("0201");
59        }
60
61    /*Envia o valor do botão - repare
os dois primeiros caracteres formam
"03" para identificar que os dois
ultimos caracteres informa o valor
do botão (00 ou 01). Corresponde ao
protocolo descrito no tópico 2.5.4 -
Programação do Circuito*/
62      if(z == 0){
63        enviar("0301");
```

```
64        }
65    }
66
67    void enviar(char* text)
68    {
69    /*Imprime a string no monitor
serial antes de envia-la.*/
70        Serial.println(text);
71        vw_send((uint8_t *)text,
strlen(text));
72        vw_wait_tx();
73    }
```

E...Para o carro é carregado o Código 13:

Código 13 – Carro com Receptor RF

```
1  #include <VirtualWire.h>
2  #include <stdlib.h>
3  /*Define o vetor que vai receber
   a menssagem do controle com o
   tamanho máximo permitido(80 bytes)
   com uma constante da biblioteca
   "VirtualWire".*/
4  byte message[VW_MAX_MESSAGE_LEN];
5  /*Grava na variável o valor 80, que
   como já foi dito é o tamanho máximo
   permitido.*/
6  byte msgLength = VW_MAX_MESSAGE_LEN;
7  char valorAtual[4];
8  int tempo = 500;/*Define o tempo
   entre um comando e outro.*/
9
10 int input1 = 11;//---------
11 int input2 = 10;//Define os pinos
   para o
12 int input3 = 4;//controle dos
```

```
   motores.
13 int input4 = 3;//-----------
14
15 void setup()
16 {
17 /*Define no pino 7 a conexão com o
   receptor RF. Ele irá receber os
   dados transmitidos do emissor do
   controle.*/
18   vw_set_rx_pin(7);
19 /*Define a velocidade de comunicação
   em 2000 bits por segundo.*/
20   vw_setup(2000);
21 /*Inicia a transmissão via
   radiofrequência.*/
22   vw_rx_start();
23   pinMode(input1, OUTPUT);
24   pinMode(input2, OUTPUT);
25   pinMode(input3, OUTPUT);
26   pinMode(input4, OUTPUT);
27   Serial.begin(9600);
28 }
29
30 void loop()
31 {
32   char comando[4];
33 /*Define um vetor do tipo "uint8_t"
   para receber os dados do
   transmissor.*/
34   uint8_t message[VW_MAX_MESSAGE_
   LEN];
35   uint8_t msgLength = VW_MAX_
   MESSAGE_LEN;
```

```
36 /*Verifica se há alguma mensagem
   disponível, se retornar verdadeiro
   grava a mensagem no buffer de
   leitura.*/
37   if (vw_get_message(message,
   &msgLength))
38   {
39     for (int i = 0; i < msgLength;
   i++)
40     {
41 /*Passa o conteúdo do vetor de bytes
   para um vetor de caracteres.*/
42         comando[i] = message[i];
43     }
44     comando[4] = '\0';
45 /*Envia o vetor de caracteres
   recebidos para uma função que lê,
   interpreta e executa a função
   "movimentar()" passando como
   parâmetro o conteúdo do vetor.*/
46     movimentar(comando);
47     Serial.println(comando);
48   }
49 }
50 /*Essa função manda instruções para
   o carro de acordo com o conteúdo
   passado como parâmetro.*/
51 void movimentar(char* comando)
52 {
53   if (((comando[1] ==
   '1')&&(comando[3] == '5')) ||
   ((comando[1] == '1')&&(comando[3] ==
   '4')))
```

```
54    {//Move o carro pra frente.
55      digitalWrite(input1, LOW);
56      digitalWrite(input2, HIGH);
57      digitalWrite(input3, LOW);
58      digitalWrite(input4, HIGH);
59      delay(tempo);
60  //Desliga os motores.
61      digitalWrite(input1, LOW);
62      digitalWrite(input2, LOW);
63      digitalWrite(input3, LOW);
64      digitalWrite(input4, LOW);
65    }
66
67    if (((comando[1] ==
    '1')&&(comando[3] == '1')) ||
    ((comando[1] == '1')&&(comando[3] ==
    '2')))
68    {//Move os dois motores para trás.
69      digitalWrite(input1, HIGH);
70      digitalWrite(input2, LOW);
71      digitalWrite(input3, HIGH);
72      digitalWrite(input4, LOW);
73      delay(tempo);
74  //Desliga os motores.
75      digitalWrite(input1, LOW);
76      digitalWrite(input2, LOW);
77      digitalWrite(input3, LOW);
78      digitalWrite(input4, LOW);
79    }
80
81    if (((comando[1] ==
    '2')&&(comando[3] == '5')) ||
    ((comando[1] == '2')&&(comando[3] ==
```

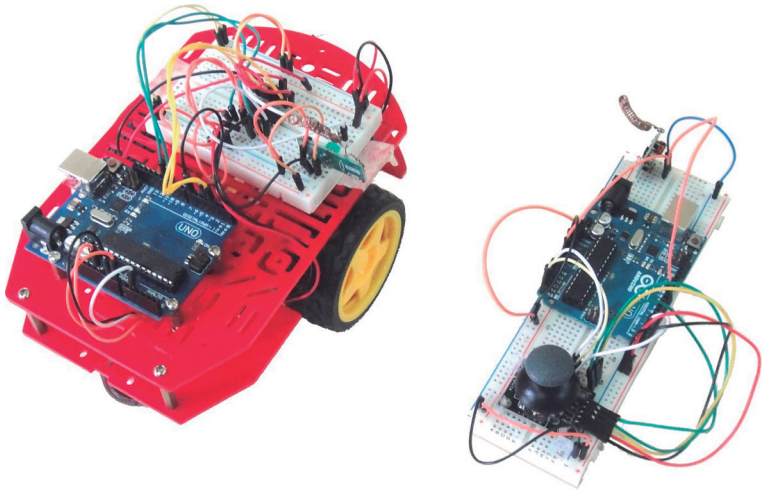
```
    '4')))
82    {//Move um motor (carro vira para
    a direita).
83      digitalWrite(input1, LOW);
84      digitalWrite(input2, HIGH);
85      digitalWrite(input3, LOW);
86      digitalWrite(input4, LOW);
87      delay(tempo);
88  //Desliga os motores.
89      digitalWrite(input1, LOW);
90      digitalWrite(input2, LOW);
91      digitalWrite(input3, LOW);
92      digitalWrite(input4, LOW);
93    }
94
95    if (((comando[1] ==
    '2')&&(comando[3] == '1')) ||
    ((comando[1] == '2')&&(comando[3] ==
    '2')))
96    {//Move um motor (carro vira para
    a esquerda).
97      digitalWrite(input1, LOW);
98      digitalWrite(input2, LOW);
99      digitalWrite(input3, LOW);
100     digitalWrite(input4, HIGH);
101     delay(tempo);
102 //Desliga os motores.
103     digitalWrite(input1, LOW);
104     digitalWrite(input2, LOW);
105     digitalWrite(input3, LOW);
106     digitalWrite(input4, LOW);
107   }
108
109    if (((comando[1] ==
```

```
    '1')&&(comando[3] == '3')) ||
    ((comando[1] == '2')&&(comando[3] ==
    '3')))
110   {//Desliga os motores.
111     digitalWrite(input1, LOW);
112     digitalWrite(input2, LOW);
113     digitalWrite(input3, LOW);
114     digitalWrite(input4, LOW);
115   }
116 }
```

2.3.3.3 *Análise Geral do Funcionamento*

Ao carregar os códigos do tópico anterior, a comunicação entre o carro e o controle estará completa e o carro deve responder aos comandos enviados pelo controle. A Figura 53 exibe a imagem do carro e do controle remoto finalizados.

Figura 53 – Carro e Controle Remoto Finalizados



CAPÍTULO 3
CONSTRUINDO O CARRO AUTÔNOMO

A construção da estrutura do carro descrita na seção **2.4.3** será a mesma utilizada para o carro autônomo, portanto, antes de iniciar o presente capítulo, a estrutura do carro deve estar pronta.

O esquema de ponte H empregado no controle dos motores com o microcontrolador descrito na seção **2.4.4.1** é o mesmo utilizado no carro autônomo, porém, não é aplicada a função **`movimentar()`** descrita naquela seção.

A adaptação foi feita para reaproveitar o circuito do carro e o esquema de ponte H do L293D, pois não haveria necessidade de modificação nestes elementos. A única parte alterada é a lista de movimentos do carro autônomo, que não serão os mesmos da **Tabela 3** visto a inclusão de movimentos circulares. Todos os movimentos do carro autônomo são descritos na **Tabela 7**.

A construção do carro autônomo foi dividida em 2 partes: 3.4 – **Construção do Circuito do Sensor** e **3.5** – **Comunicação Entre Carro e Sensor**. O tópico 3.4 – Construção do Circuito do Sensor descreve a montagem do sensor HC-SR04, botão para ligar e desligar, da alimentação do circuito integrado e da programação do circuito. O tópico **3.5** – Comunicação Entre Carro e Sensor expõe a comunicação entre os dados recebidos pelo sensor e o Arduino, explica a lógica utilizada para determinar as decisões e apresenta o código necessário para o funcionamento do Carro Autônomo.

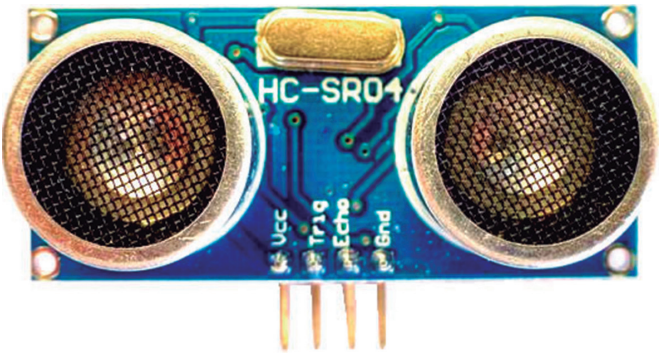
3.1 CONSTRUÇÃO DO CIRCUITO DO SENSOR

3.1.1 COMPONENTES NECESSÁRIOS

- **Sensor Ultrassônico HC-SR04** – sensor exibido na Figura 54 é responsável pelo envio de uma onda sonora em frequência muito

alta para os ouvidos humanos (ultrassônico) e, em seguida, pelo cálculo da distância do objeto a partir da velocidade com que o som do eco[1] do sinal enviado retorna.

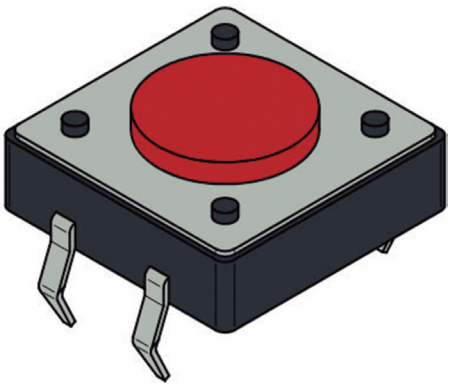
Figura 54 – Sensor Ultrassônico HC-SR04



Fonte: (WIKIMEDIA, 2014)

- **Chave Táctil (*push-bottom*)** – chave ilustrada na Figura 55 utilizada para controlar o fluxo da corrente em determinado ponto do circuito.

Figura 55 – Chave Táctil



Fonte: (WIKIMEDIA, 2009)

- **Resistor de 220 ohms** – resistor de carbono simples de aproximadamente 220 ohms.

1 Repetição de um som pela reflexão das ondas sonoras (KURY, 2001, p. 268).

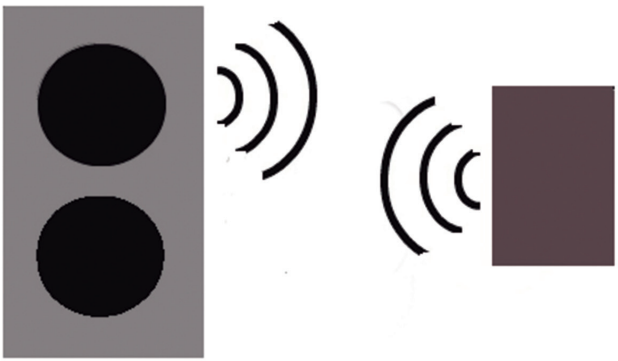
3.1.2 CONSTRUÇÃO DO CIRCUITO

3.1.2.1 Sensor HC-SR04

O componente principal desse projeto é um sensor ultrassônico (exibido na Figura 56), item amplamente utilizado em projetos que necessitam identificar distâncias ou evitar colisões.

Para obter a distância de determinado objeto, um pulso sonoro de alta frequência é lançado pelo emissor presente no circuito do sensor até o alvo, ao atingir o objeto, será refletido um sinal de eco a ser captado pelo receptor. O sinal lançado possui a velocidade do som, com essa informação é possível calcular o tempo entre sua emissão e seu retorno, desse modo, obtém-se a distância do objeto. Essa situação é exemplificada na Figura 56.

Figura 56 – Funcionamento do Sensor Ultrassônico



Para a elaboração do cálculo supracitado são organizadas algumas variáveis. A primeira é a velocidade do sinal, que é representada pela velocidade do som, ou seja, 340 m/s. A segunda é a distância percorrida pelo sinal. Como o sinal atravessa uma distância "d" até o objeto e percorre esse mesmo caminho na volta, o deslocamento total é equivalente a "2d". Conhecendo essa informação é possível resolver a Equação 6:

Equação 6 – Cálculo da Distância

$$distância = \frac{velocidade * tempo}{2} \Rightarrow d = \frac{340 * t}{2} \Rightarrow d = 170 * t$$

Fonte: (MICROPIK, 2015)

A conexão do sensor HC-SR04 no circuito é realizada a partir de quatro pinos, cada um com sua devida identificação no sensor. São eles:

- **Vcc**: pino que deve ser conectado a uma alimentação de 5 V;
- **Trig**: pino correspondente ao emissor do sinal que deve ser conectado à um pino digital e configurado como saída;
- **Echo**: pino correspondente ao receptor do sinal que deve ser conectado a um pino digital e configurado como entrada;
- **GND:** pino que deve ser conectado ao pino GND.

As características elétricas do sensor HC-SR04 seguem na Tabela 6:

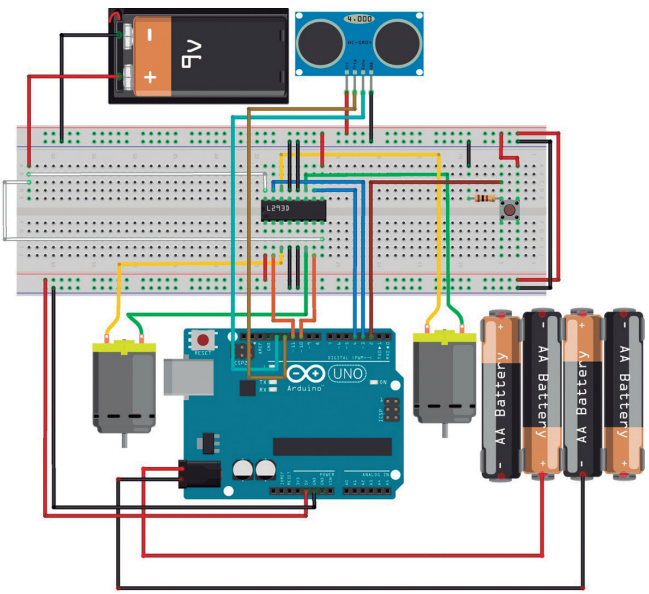
Tabela 6 – Características do Sensor HC-SR04

Característica	Valor
Alimentação	5 V
Corrente de operação	2 mA
Alcance	2 cm – 4 m
Ângulo de efeito	15 º
Precisão	3 mm

Fonte: (MICROPIK, 2015)

Para conectar o sensor HC-SR04 com o Arduino é utilizado o circuito exibido na Figura 57.

Figura 57 – Circuito do Arduino com HC-SR04

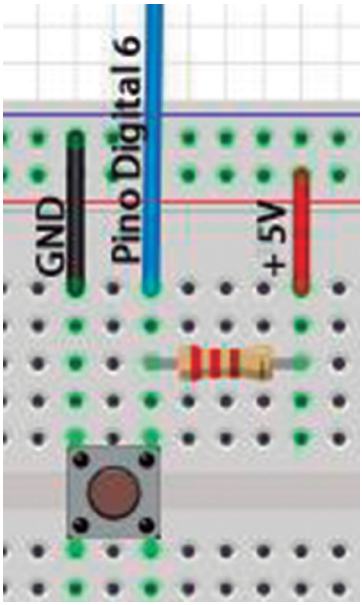


3.1.2.2 Chave Táctil

Trata-se de um pequeno botão que envia um sinal digital para o Arduino. Será utilizado como botão liga/desliga do circuito e geralmente possui quatro saídas para serem conectadas diretamente a protoboard, porém, apenas duas são utilizadas. Além disso, os orifícios devem ser usados em lados opostos, como exibida na Figura 58.

O valor enviado pode ser igual a 0 ou a 1, dependendo se o botão estiver pressionado ou não. Sempre que é utilizada uma chave táctil no circuito, ela deve ser acompanhada de um resistor localizado entre sua saída e a conexão digital com o Arduino, sendo que a outra ponta do resistor deve ser conectada a uma alimentação como exibida na Figura 58. O motivo da utilização do resistor é que pode haver variação da tensão quando o botão não está pressionado, deixando o botão em um estado indefinido, variando entre 0 e 1. Com o resistor presente no circuito essa variação desaparece, fixando um valor único para o botão.

Figura 58 – Conexão Entre uma Chave Táctil e o Arduino



Para evitar a utilização de um resistor no circuito é possível utilizar a função **`digitalWrite()`** descrito na seção 1.4.3.2 – Funções Para Entrada/Saída Digitais, desse modo, é habilitado um resistor interno presente no Arduino.

3.1.2.3 Alimentação

A alimentação do circuito será originada de duas fontes distintas. Uma delas, que é a bateria de 9 V, é destinada aos motores DC das rodas do carro, enquanto a outra suprirá o resto do circuito. Com isso, o circuito permanecerá alimentado por um tempo maior.

3.1.3 PROGRAMAÇÃO DO CIRCUITO

3.1.3.1 Código do Circuito

O Código 14 é composto de duas funcionalidades principais do carro autônomo. Esse código não apresenta nenhuma alteração nos motores, apenas imprimirá no monitor serial os estados das variáveis importantes para o projeto. Ele deve ser utilizado para testes do funcionamento do sensor e da chave táctil.

Código 14 – Código de Teste Para o Carro Autônomo

```
1  #define TRIGGER_PIN  12/*No pino 12 é
   conectada a saída do sensor HC-SR04,
   referente ao emissor que envia o
   sinal ultrassônico.*/
2  #define ECHO_PIN     13/*No pino 13 é
   conectada a saída do sensor HC-SR04,
   referente ao receptor recebe o eco
   da onda ultrassônica.*/
3
4  volatile int estado = LOW;/*O estado
   do botão começa com o valor LOW,
   representando que o carro ainda não
   foi ligado. Essa variável é volátil
   (volatile), ou seja, seu valor pode
   ser alterado fora do local em que é
   declarada.*/
5
6  void setup() {
7    attachInterrupt(0, chaveOnOff,
   RISING);/*A interrupção 0 do Arduino
   UNO é selecionada para ser utilizada
   pela função "chaveOnOff()", essa
   interrupção está ligada ao pino 2 do
   Arduino UNO. O parâmetro "RISING"
   define que quando o pino 2 do Arduino
```

```
   UNO alterar seu valor de LOW para
   HIGH, a interrupção será acionada.*/
8    pinMode(TRIGGER_PIN, OUTPUT);/*O
   TRIGGER_PIN é definido como saída,
   pois terá a função de enviar o som
   ultrassônico.*/
9    pinMode(ECHO_PIN, INPUT);/*O ECHO_
   PIN é definido como entrada, pois
   terá a função de receber o eco do
   sinal ultrassônico.*/
10   pinMode(2, INPUT);/*O pino 2 é
   definido como entrada, pois é o pino
   ligado a interrupção 0.*/
11   Serial.begin(9600);
12 }
13
14 void loop() {
15   float cm;/*Variável que receberá a
   distância do objeto detectado.*/
16   long microsec =
   detectar();/*Variável que receberá a
   distância do objeto diretamente do
   sensor, pela função "detectar()".*/
17   cm = (microsec/27.6233)/2;/*Co
   nvertendo o valor detectado para
   centímetros.*/
18
19   Serial.print("CM = ");
20   Serial.println(cm);/*Imprime o
   valor detectado do monitor serial*/
21   Serial.print("Estado da Chave= ");
22   Serial.println(estado);/*Imprime o
   valor detectado do botão no monitor
   serial.*/
23   delay(100);/*Tempo aguardado
   para fazer uma nova verificação de
   detecção.*/
24 }
25
26 long detectar() {
27   digitalWrite(TRIGGER_PIN,
   LOW);/*Desliga o TRIGGER_PIN,
   limpando a saída do som ultrassônico
   de outros envios.*/
```

```
28   delayMicroseconds(2);/*Tempo
     aguardado para a limpeza da saída do
     som ultrassônico, 2 milionésimos de
     segundo.*/
29   digitalWrite(TRIGGER_PIN,
     HIGH);/*TRIGGER_PIN enviando o som
     ultrassônico.*/
30   delayMicroseconds(10);/*Tempo
     aguardado para enviar o som
     ultrassônico, 10 milionésimos de
     segundo.*/
31   digitalWrite(TRIGGER_PIN,
     LOW);/*TRIGGER_PIN para de enviar o
     som ultrassônico.*/
32   return pulseIn(ECHO_PIN, HIGH);/*O
     retorno é o valor detectado pelo
     ECHO_PIN, este define a distância do
     objeto detecado. O retorno é uma
     variável do tipo "long".*/
33 }
34 /*A função "chaveOnOff()" altera o
     valor da variável "estado" quando o
     botão é pressionado.*/
35 void chaveOnOff() {
36   static unsigned long lastMillis =
     0; /*"lastMillis" é utilizada para
     resolver o problema do tempo de
     "bounce". Ela inicia com o valor 0,
     mas como ela é uma variável estática
     (static), mantém o seu valor entre a
     chamadas da função "chaveOnOff()".*/
37   unsigned long newMillis =
     millis();/*"newMillis" é utilizada
     para resolver o problema do tempo
     de "bounce", ela recebe o valor de
     retorno da função "millis()", que é
     a quantidade de tempo transcorrido
     desde o início da execução do
     programa.*/
38   if(newMillis - lastMillis < 50)
     {/*É feita a comparação da diferença
     entre o tempo atual (newMillis)
     e o tempo anterior de passagem
```

```
   por essa função (lastMillis), se
   essa diferença é menor do que 50,
   então o botão não foi realmente
   pressionado. O que ocorre na
   verdade é que quando o botão foi
   pressionado anteriormente causou um
   efeito de instabilidade que gerou
   a interrupção novamente, como se
   ouvesse sido pressionado 2 ou mais
   vezes seguidas em um intervalo menor
   de que 50 milissegundos.*/
39   }
40   else {
41     estado = !estado;/*Ocorre uma
   inverção da variável "estado"
   quando diferença entre o tempo
   atual (newMillis) e o tempo anterior
   de passagem por essa função
   (lastMillis) for maior do que 50
   milissegundos.*/
42     lastMillis = newMillis;/*A
   variável "lastMillis" é atualizada
   com o valor de "newMillis".*/
43   }
44 }
```

3.1.3.2 Funcionalidades

O **Código 14** possui duas funções que são responsáveis por receber estímulos externos: a função de detecção de obstáculos (linha 26) e a função de interrupção do circuito (linha 35).

A primeira, **`detectar()`**, controla os pinos conectados ao sensor ultrassônico. Um som ultrassônico é emitido quando o **`TRIGGER _ PIN`** (variável definida como pino 12 na linha 1) recebe o valor **`HIGH`** e, após aguardar 10 milionésimos de segundos, recebe o valor **`LOW`**. Em seguida, o **`ECHO _ PIN`** (variável definida como pino 13 na linha 2) recebe o valor **`HIGH`** para capturar o eco do som enviado.O sensor possui um pequeno microcontrolador que determina o tempo entre o envio e o recebimento de um som, valor de tempo detectado por ele é codificado em uma tensão que está entre 0 V e 5 V. Quanto maior o atraso entre a emissão e a recepção do som, maior a tensão. Essa tensão deve ser convertida em uma distância. A **Equação 7** converte o valor recebido para centímetros e é utilizada na linha 17 do sketch para este fim.

Equação 7 – Conversão em Centímetros

$$cm = (microsegundos / 27.6233) / 2$$

Fonte: (TETRASYS, 2015)

A faixa de distância detectada pelo sensor ultrassônico (2 cm a 4 m) é mais do que suficiente para encontrar um obstáculo próximo ao carro. Porém, esses obstáculos devem estar exatamente à frente do sensor, exigindo o posicionamento correto do carro e os movimentos milimetricamente calculados do carro.

A segunda função, **`chaveOnOff()`**, na linha 35, altera o valor da variável volátil **`estado`** (linha 4) de acordo com o estímulo recebido externamente pela chave táctil, utilizando uma interrupção. A interrupção é um recurso poderoso, pois pode parar a execução do sketch a qualquer momento. O Arduino UNO pode utilizar, no máximo, duas interrupções: a interrupção 0 é ligada ao pino digital 2 e a interrupção 1 é ligada ao pino digital 3. Uma interrupção deve ser declarada dentro da função **`setup()`** com a função **`attachInterrupt()`** (linha 7), recebendo três argumentos:

Interrupção – O primeiro argumento pode receber o valor 0 ou 1. Define qual interrupção do Arduino será utilizada.

Função – O segundo argumento recebe a função que deve ser executada quando acionada a interrupção.

Modo – O terceiro argumento pode receber quatro valores. Define de qual modo a interrupção será gerada. Os valores são:

- **LOW**: sempre que o pino digital relacionado a interrupção estiver baixo **`LOW;`**
- **CHANGE**: sempre que o pino digital relacionado a interrupção mudar de valor, seja **`LOW`** para **`HIGH`** ou **`HIGH`** para **`LOW;`**
- **RISING:** sempre que o pino digital relacionado mudar de **`LOW`** para **`HIGH;`**
- **FALLING**: sempre que o pino digital relacionado mudar de **`HIGH`** para **`LOW`**.

Geralmente as interrupções executam funções que alteram valores em variáveis voláteis, pois estas podem ser alteradas fora da parte do sketch em que aparecem. A variável volátil utilizada é a **`estado`**, que começa com valor **`LOW`** e a função **`chaveOnOff()`** declarada na linha 4, muda-se o seu valor para o inverso sempre que a chave táctil é acionada **`(estado = !estado)`**.

A função acionada pela interrupção é composta por uma estrutura capaz de solucionar um problema causado por interrupções mecânicas, conhecido como *bounce*. Segundo (EVANS, NOBLE e HOCHENBAUM, 2013, p. 58) "[..] quando um interruptor é pressionado e se move de posição aberta para fechada, o contato frequentemente não é perfeito e produz alguns sinais espúrios chamados *bounces*, fazendo com que o pino conectado se mova de **LOW** para **HIGH** muitas vezes até que se torne estável."Por isso é necessário conferir se o pino conectado à chave táctil já estabilizou o seu valor,para essa verificação é utilizada a variável estática **lastMillis** (linha 36). Por ser estática, a variável mantém o seu valor entre as chamadas de uma função específica. A função **millis()** retorna o número de milissegundos que se passaram desde o início da execução do programa, que é recebido pela variável **newMillis** na linha 37. Ao comparar as duas variáveis é possível saber se a chave táctil foi pressionada: se o resultado for menor do que 50 milissegundos, nada será feito e o código voltará ao loop; caso seja maior ou igual à 50 milissegundos, estará fora do tempo de *bounce*, indicando que a chave táctil foi realmente pressionada de novo, executando a mudança da variável **estado**.

3.1.3.3 Análise do Funcionamento

O loop do Código 14 contém um trecho de código que imprime as variáveis **cm** e **estado** no monitor serial. Assim, o funcionamento do circuito pode ser analisado.

Na linha 15 é declarada a variável **cm** que vai receber a distância. Na linha 16 a variável **microsec** chama a função **detectar()** que, através do sensor ultrassônico, detecta qual o tempo de resposta da onda sonora lançada e refletida por um objeto. Na linha 17 o tempo de resposta captado é usado para completar a equação de conversão, em centímetros, e armazenar o resultado na variável **cm**. Essa variável será exibida no monitor serial para observação, na linha 20 do sketch.

A distância exibida no monitor serial deve ser comparada à distância real do obstáculo. Pode ocorrer uma variação devido ao **delay(100)**, que é necessário para que o sensor de distância tenha o tempo necessário para não gerar problemas com ondas ultrassônicas anteriores que ainda podem estar no ambiente.

A variável **estado** também é impressa no monitor serial (linha 22 do sketch). Esse estado da chave táctil pode apresentar oscilações de valores. Caso essa oscilação seja excessiva, será necessário aumentar o valor de comparação na função **chaveOnOff(),** na linha 38.

3.2 COMUNICAÇÃO ENTRE CARRO E SENSOR

3.2.1 CÓDIGO DO CIRCUITO

O Código 15 é o código final do carro autônomo.

Código 15 – Código do Carro Autônomo

```
1  #define TRIGGER_PIN  12/*Pino 12
   conectado a saída do sensor HC-
   SR04 referente ao emissor do sinal
   ultrassônico.*/
2  #define ECHO_PIN     13/*Pino 13
   conectado à saída do sensor HC-
   SR04 referente ao receptor do sinal
   refletido.*/
3
4  volatile int estado = LOW;/*O estado
   do botão começa com o valor LOW,
   representando que o carro ainda não
   foi ligado. Essa variável é volátil
   (volatile), ou seja, seu valor pode
   ser alterado fora do local em que é
   declarada.*/
5
6  int direcao = 1;/*A variável
   "direcao" é inicializada com o valor
   1, depois é atualizada com um valor
   aleatório.*/
7  int distancia = 10;/*A variável
   "distancia" é incializada com o
   valor 10, isso significa que o carro
   deve tomar uma decisão toda vez
   que detectar um obstáculo a 10
   centímetros.*/
8  /*Variáveis "input" recebendo seus
   respectivos valores dos pinos do
   Arduino.*/
9  int input1 = 11;
10 int input2 = 10;
11 int input3 = 4;
```

```
12 int input4 = 3;
13 /*--------------------------------
   */
14 void setup() {
15   attachInterrupt(0, chaveOnOff,
   RISING);/*A interrupção 0 do Arduino
   UNO é selecionada para ser utilizada
   pela função "chaveOnOff()", essa
   interrupção está ligada ao pino 2 do
   Arduino UNO. O parâmetro "RISING"
   define que quando o pino 2 do Arduino
   UNO altera o seu valor de LOW
   para HIGH, então a interrupção é
   acionada.*/
16   pinMode(TRIGGER_PIN, OUTPUT);/*O
   "TRIGGER_PIN" é definido como saída,
   pois terá a função de enviar o som
   ultrassônico.*/
17   pinMode(ECHO_PIN, INPUT);/*O
   "ECHO_PIN" é definido como entrada,
   pois terá a função de receber o eco
   do som ultrassônico enviado.*/
18   pinMode(2, INPUT);/*O pino 2 é
   definido como entrada, pois é o pino
   ligado a interrupção 0.*/
19   pinMode(input1, OUTPUT);/*As
   variáveis "input" são todas para
   saída de dados.*/
20   pinMode(input2, OUTPUT);
21   pinMode(input3, OUTPUT);
22   pinMode(input4, OUTPUT);
23  /*--------------------------------
   -----*/
24   randomSeed(analogRead(0));/*A
   função "randomSeed()" recebe um pino
   analógico como parâmetro e retorna
   uma semente para gerar valores
   aleatórios com base nela. Aqui a
   função utiliza o pino analógico
   A0.*/
```

```
25   Serial.begin(9600);
26 }
27
28 long detectar() {
29   digitalWrite(TRIGGER_PIN,
   LOW);/*"TRIGGER_PIN" limpando a
   saída do som ultrassônico de envios
   anteriores.*/
30   delayMicroseconds(2);/*Tempo
   aguardado para a limpeza da saída do
   som ultrassônico. Equivalente a 2
   milionésimos de segundo.*/
31   digitalWrite(TRIGGER_PIN,
   HIGH);/*"TRIGGER_PIN" enviando o som
   ultrassônico.*/
32   delayMicroseconds(10);/*Tempo
   aguardado para enviar o som
   ultrassônico. Equivalente a 10
   milionésimos de segundos).*/
33   digitalWrite(TRIGGER_PIN,
   LOW);/*"TRIGGER_PIN" para de enviar
   o sinal ultrassônico.*/
34   return pulseIn(ECHO_PIN, HIGH);/*O
   retorno é o valor detectado pelo
   "ECHO_PIN". Este define a distância
   do objeto detecado. O retorno é uma
   variável do tipo "long" devido ao
   tamanho do valor recebido.*/
35 }
36
37 void loop() {
38   float cm;/*Variável que receberá a
   distância do objeto detectado.*/
39   long microsec =
   detectar();/*Variável que receberá a
   distância do objeto diretamente do
   sensor pela função "detectar()".*/
40
41   cm = (microsec/27.6233)/2;/*Co
   nvertendo o valor detectado para
```

```
   centímetros.*/
42   direcao = random(2); /*A variável
   é atualizada com um valor aleatório,
   podendo ser 0 ou 1.*/
43
44   if (estado) {/*Verifica se o botão
   foi pressionado, caso não seja a
   distância do obstáculo à frente é
   verificada.*/
45     if (cm > distancia) {/*Verifica
   se a distância detectada é maior
   do que os centímetros definidos na
   variável "cm". Enquanto for maior o
   carro se move para frente.*/
46       digitalWrite(input1,
   LOW);/*Move o carro para frente.*/
47       digitalWrite(input2, HIGH);
48       digitalWrite(input3, LOW);
49       digitalWrite(input4, HIGH);
50       delay(100);
51     } else {
52       if (direcao) {/*Verifica a
   direção escolhida aleatoriamente.*/
53         digitalWrite(input1,
   HIGH);/*Move o carro de ré.*/
54         digitalWrite(input2, LOW);
55         digitalWrite(input3, HIGH);
56         digitalWrite(input4, LOW);
57         delay(75);
58         digitalWrite(input1,
   LOW);/*Move o carro para a
   direita.*/
59         digitalWrite(input2, HIGH);
60         digitalWrite(input3, HIGH);
61         digitalWrite(input4, LOW);
62         delay(75);
63       } else {
64         digitalWrite(input1,
   HIGH);/*Move o carro de ré.*/
65         digitalWrite(input2, LOW);
```

```
66          digitalWrite(input3, HIGH);
67          digitalWrite(input4, LOW);
68          delay(75);
69          digitalWrite(input1,
HIGH);/*Move o carro para a
esquerda.*/
70          digitalWrite(input2, LOW);
71          digitalWrite(input3, LOW);
72          digitalWrite(input4, HIGH);
73          delay(75);
74        }
75      }
76    } else {/*Comandos para parar
o carro, caso o botão seja
pressionado, o carro continua o
movimento.*/
77      digitalWrite(input1, LOW);
78      digitalWrite(input2, LOW);
79      digitalWrite(input3, LOW);
80      digitalWrite(input4, LOW);
81    }
82  }
83  /*A função "chaveOnOff()" altera o
valor da variável "estado" quando o
botão é pressionado. Os valor podem
ser HIGH ou LOW.*/
84  void chaveOnOff() {
85    static unsigned long lastMillis
= 0; /*A variável "lastMillis" é
utilizada para resolver o problema
do tempo de bounce. Ela inicia com o
valor 0, mas como ela é uma variável
estática (static) pode manter o seu
valor entre a chamadas da função
"chaveOnOff()".*/
86    unsigned long newMillis =
millis();/*"newMillis" é utilizada
para resolver o problema do tempo
de bounce. Ela recebe o valor de
retorno da função "millis()", que é
```

```
   a quantidade de tempo transcorrido
   desde o início da execução do
   programa.*/
87   if(newMillis - lastMillis < 50)
   {/*É feita a comparação da diferença
   entre o tempo atual (newMillis)
   e o tempo anterior de passagem
   por essa função (lastMillis), se
   essa diferença é menor do que 50,
   então o botão não foi realmente
   pressionado. O que ocorre, na
   verdade, é que quando o botão foi
   pressionado anteriormente causou um
   efeito de instabilidade que gerou
   a interrupção novamente, como se
   ouvesse sido pressionado 2 ou mais
   vezes seguidas em um intervalo menor
   de que 50 milissegundos.*/
88   }
89   else {
90     estado = !estado; /*Ocorre uma
   inverção da variável estado quando
   diferença entre o tempo atual
   (newMillis) e o tempo anterior
   de passagem por essa função
   (lastMillis) for maior do que 50
   milissegundos.*/
91     lastMillis = newMillis;/*A
   variável "lastMillis" é atualizada
   com o valor de "newMillis".*/
92   }
93 }
```

3.2.2 FUNCIONALIDADES

A **Tabela 7** exibe as combinações de valores das portas lógicas Input do L293D utilizadas no código do carro com controle remoto, juntamente com os movimentos executadas pelos motores 1 (esquerdo) e 2 (direito) do carro, que podem ser retardados (trás), acelerados (frente) ou parados. Os movimentos e suas respectivas descrições estão na cor preta.

Tabela 7 – Combinações das Portas Lógicas no Carro Autônomo

Nº	Input 1	Input 2	Input 3	Input 4	O carro faz o movimento de...
	Motor 1 – Esquerdo		Motor 2 – Direito		
1	LOW	LOW	LOW	LOW	Freio, pois ambos os motores estão parados.
	Parado		Parado		
2	LOW	HIGH	HIGH	LOW	Roda para a direita, pois o motor esquerdo acelera e o motor direito retarda.
	Frente		Trás		
3	HIGH	LOW	LOW	HIGH	Roda para a esquerda, pois o motor direito acelera e o motor esquerdo retarda.
	Trás		Frente		
4	HIGH	LOW	HIGH	LOW	Retardação, pois os dois motores retardam (trás).
	Trás		Trás		
5	LOW	HIGH	LOW	HIGH	Aceleração, pois os dois motores aceleram (frente).
	Frente		Frente		

As combinações de valores (**HIGH** e **LOW**) nº 1, 2, 3, 4 e 5 nas portas lógicas Input, na **Tabela 7,** são referentes às ações executadas pelo carro autônomo. Todas ações possíveis são descritas no anexo **5.10** – Tabela Completa dos Valores das Portas Lógicas INPUT do L293D**.**

As ações realizadas pelo carro autônomo fazem o carro girar em seu próprio eixo, o que as diferedas ações do carro com controle remoto é que neste há um movimento de curva, isso ocorre porque apenas um dos motores roda quando o joystick envia um sinal para o carro curvar.

Após pressionar a chave táctil, o carro utiliza o sensor ultrassônico para detectar obstáculos e em seguida toma a decisão com base na distância do obstáculo, caso exista. A tomada de decisão envolve três comparações:

- **Chave Táctil:** o primeiro valor verificado é a variável **`estado`.** Caso seja **HIGH**, é testada a segunda comparação (sensor ultrassônico). Caso seja **LOW**, o carro continua parado (pinos dos motores com valor **LOW**). Trata-se de um mecanismo básico de Liga/Desliga, porém não desativa ou ativa o circuito totalmente. Para ligar o carro novamente basta apertar o botão outra vez;
- **Sensor Ultrassônico:** o segundo valor verificado é a variável **`cm`**. Caso seja maior do que a variável **`distancia`**, que possui o valor 10, significa que não foi detectado nenhum obstáculo a menos de 10 centímetrose o carro movimenta-se para frente (input 1 e 3 com valor **LOW**, input 2 e 4 com valor **HIGH**). Caso seja menor ou igual a variável **`distancia`**, significa que há um

obstáculo à frente do sensor e o carro faz a terceira comparação descrita abaixo (Direção), conferindo a distância do carro em relação a algum obstáculo à frente do sensor;

- **Direção**: o terceiro valor verificado é a variável **`direcao`**, que possui um valor aleatório igual a 0 ou a 1 atribuído (linha 43) pela função **`random()`**. A função **`random()`** recebe um número inteiro e o valor que retorna depende da variável passada como parâmetro. Para funcionar corretamente, a função necessita de uma semente, pois precisa de um valor aleatório (ou pseudoaleatório) diferente para não gerar os mesmos números. A semente é declarada na linha 24 do Código 15, na qual o pino analógico A0 será utilizado como semente. O valor retornado está entre 0 e o valor passado como parâmetro na função, subtraindo uma unidade. Por exemplo: **`random(10)`** retornará um valor x no intervalo:

$$(0 \geq x) \wedge (x < 10)$$

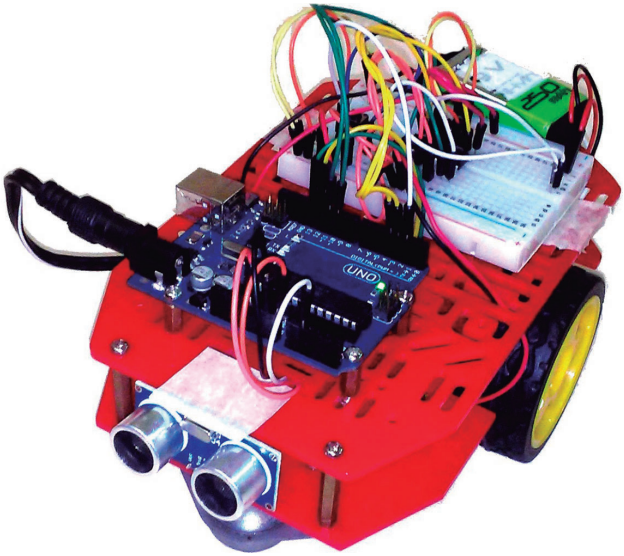
No código do carro é utilizado o **`random(2)`**, que retorna o valor 0 ou o valor 1. Caso o valor sorteado e armazenado na variável **`direcao`** seja 0, o carro tomará uma decisão de rodar para a esquerda, caso seja 1, tomará a decisão de virar para a direita.

3.2.3 ANÁLISE GERAL DO FUNCIONAMENTO

Após o código ser colocado no Arduino, o carro deve ser capaz de se mover de forma autônoma ao pressionar a chave táctil, além de parar quando for pressionada novamente e de escolher um caminho alternativo, aleatoriamente, quando encontrar um obstáculo. A **Figura 59** mostra o carro autônomo concluído:

Caso o valor sorteado e armazenado na variável **`direcao`** seja 0, então o carro recua (de acordo com os parâmetros passados nas linhas 54 a 57 do **Código 15**, **`input1`** e **`input3`** com valor **`HIGH`**, **`input2`** e **`input4`** com valor **`LOW`**) por 75 milissegundos e roda para a direita de acordo com os parâmetros passados nas linhas 59 a 62 (**`input1`** e **`input4`** com valor **`LOW`**, **`input2`** e **`input3`** com valor **`HIGH`**) por 75 milissegundos; caso seja 1, o carro recua (**`input1`** e **`input3`** com valor **`HIGH`**, **`input2`** e **`input4`** com valor **`LOW`**) por 75 milissegundo e roda para a esquerda (**`input1`** e **`input4`** com valor **`HIGH`**, **`input2`** e **`input3`** com valor **`LOW`**) por 75 milissegundos. O movimento de rotação executado é suficiente para girar o carro 90°, colocando-o na direção escolhida

Figura 59 – Carro Autônomo Completo



aleatoriamente (direita ou esquerda). Os movimentos utilizados pelo carro para virar para direita e esquerda são referentes aos valores (`HIGH` e `LOW`) das portas lógicas Input 8 e 9 da **Tabela 3**. Sendo mais específico, o 8 faz o carro rodar para a esquerda, pois a roda esquerda se move para frente (Input 3 com valor `LOW` e o Input 4 com valor `HIGH`) e a roda direita se move para trás (Input 1 com valor `HIGH` e o Input 2 com valor `LOW`) e o 7 faz o carro rodar para a direita, pois a roda direita se move para frente (Input 1 com valor `LOW` e o Input 2 com valor `HIGH`) e a roda esquerda se move para trás (Input 3 com valor `HIGH` e o Input 4 com valor `LOW`).

O algoritmo de tomada de decisão é bastante simples. O carro apenas escolhe a direção baseado em um valor aleatório com a função `random()`. É possível o estudo e a implementação doalgoritmo mais inteligente, utilizando Lógica Paraconsistente Anotada, Redes Neurais Artificiais (RNA) ou até mesmo uma estrutura de decisão mais complexa, com mais opções e mais sensores, como o Sensor Infravermelho (no anexo 4 – Links Úteis há referências para aprender mais sobre o Senhor Infravermelho).

As Lógicas Paraconsistentes surgiram da necessidade de encontrar meios de solucionar problemas que incidem em situações contraditórias. Uma Lógica Paraconsistente Anotada pode ter como reticulado finito, o de "quatro estados": verdadeiro, falso, inconsistente e paracompleto (DA SILVA FILHO, TORRES e ABE, 2006).

Uma aplicação na inteligência artificial de Lógica Paraconsistente Anotada é o Robô Móvel Autônomo Emmy, que após a finalização do projeto, os autores concluíram que:

> O robô Emmy mostrou através de diversos testes com o desvio de obstáculos de formas geométricas diferentes que a aplicação da Lógica Paraconsistente Anotada LPA2v possibilita implementações de sistemas de controle capazes de considerar as inconsistências em sua estrutura de modo não trivial. Com isso a metodologia de aplicação da LPA2v apresentada nesta pesquisa mostra-se mais propícias para controle de robôs móveis autônomos (DA SILVA FILHO, TORRES e ABE, 2006, p. 25).

O objetivo da Lógica Paraconsistente é resolver problemas com incerteza ou com inconsistência, que não podem ser tratados com lógica clássica. Quando se utiliza a inteligência artificial o pesquisador depara-se com problemas dessa natureza o tempo todo.

As redes neurais são modelos computacionais baseados no sistema nervoso central.

> Uma rede neural, como o próprio nome sugere, é uma coleção de neurônios dispostos de forma que configurem um aspecto específico. É com estes neurônios que a rede neural aprenderá as informações que serão fornecidas pelos canais de entrada dos neurônios. O aprendizado está distribuído por toda a rede, ou seja, por todos os neurônios (TAFFNER, XEREZ e RODRIGUES FILHO, 1995, p. 56).

Utilizando redes neurais é possível fazer o carro aprender sobre o ambiente a sua volta. Por exemplo, ele pode chocar-se contra uma parede e em seguida gravar essa informação como algo a ser evitado, ou seja, da próxima vez que encontrar um obstáculo ele pode tentar desviar. Se não conseguir, terá que aprender a desviar corretamente, assim por diante, até que se adapte completamente a situação proposta, tornando o algoritmo do carro cada vez mais inteligente.

CAPÍTULO 4
LINKS ÚTEIS

4.1 FÓRUNS

- Fórum do Arduino (inglês) – http://forum.arduino.cc/;
- Laboratório de Garagem – http://labdegaragem.com/;
- RoboCore–https://www.robocore.net/modules.php?name=Forums;
- Multilógica – http://multilogica-shop.com/forum/3.

4.2 LOJAS

- FilipeFlop – http://www.filipeflop.com/;
- Vida de silício – http://loja.vidadesilicio.com.br/;
- Multilógica shop – https://multilogica-shop.com/;
- Robocore – https://www.robocore.net/.

4.3 ELETRÔNICA

- Aprendendo a utilizar o multímetro – http://www.eletroalerta.com/multimetro-digital/;
- Aprendendo a utilizar o multímetro – http://www.newtoncbraga.com.br/index.php/instrumentacao/108-artigos-diversos/2713-ins131;
- Como funciona o circuito elétrico e os efeitos da corrente – http://www.newtoncbraga.com.br/index.php/como-funciona/3213-art442.

4.4 LINGUAGEM C DO ARDUINO

- Referência do site oficial – http://www.arduino.cc/en/Reference/HomePage;
- Tradução da referência do site oficial realizada pelo site Multilógica – https://multilogica-shop.com/Referencia.

4.5 COMPONENTES DO PROJETO

- Arduino UNO – http://www.arduino.cc/en/Main/ArduinoBoardUno;
- Motores – http://cdn.sparkfun.com/datasheets/Robotics/DG01D.pdf;
- Controlador L23D – http://www.arduino.cc/documents/datasheets/L293D.pdf;
- Joystick KY-023 – http://www.tinkbox.ph/sites/mytinkbox.com/files/downloads/JOYSTICK_MODULE.pdf;
- Transmissor MX-FS-03V e Receptor MX-05V – http://www.romanblack.com/RF/cheapRFmodules.htm;
- Sensor ultrassônico HC-SR04 – http://www.micropik.com/PDF/HCSR04.pdf;
- Sensor Infravermelho – http://blog.filipeflop.com/sensores/sensor-infravermelho.html.

4.6 REDES NEURAIS

- Redes Neurais (breve introdução) – http://www.icmc.usp.br/~andre/research/neural/.

4.7 LÓGICA PARACONSISTENTE

- Robô Móvel Autônomo Emmy: Uma Aplicação eficiente da Lógica Paraconsistente Anotada – http://www.paralogike.com.br/Robo%20Movel%20Autonomo%20Emmy-%20Uma%20Aplicacao%20Eficiente%20da%20Logica%20Paraconsistente%20Anotada.pdf;
- Métodos de Aplicações da Lógica Paraconsistente Anotada de Anotação com dois valores-LPA2v – http://www.paralogike.com.br/Logica%20Paraconsistente%20Anotada%20com%20anotacao%20de%20dois%20valores%20LPA2v.pdf.

CAPÍTULO 5
ANEXOS

5.1 SKETCH DO MOVIMENTO DO CARRO

```
#include <stdlib.h>
char* msg;
int input1 = 11;
int input2 = 10;
int input3 = 4;
int input4 = 3;

void setup()
{
  pinMode(input1, OUTPUT);
  pinMode(input2, OUTPUT);
  pinMode(input3, OUTPUT);
  pinMode(input4, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  msg = "0105";
  Serial.println(msg);
  movimentar(msg);
  delay(500);
```

```
23    msg = "0101";
24    movimentar(msg);
25    delay(500);
26    msg = "0205";
27    movimentar(msg);
28    delay(500);
29    msg = "0201";
30    movimentar(msg);
31    delay(500);
32    msg = "0103";
33    movimentar(msg);
34    delay(500);
35  }
36
37  void movimentar(char* comando)
38  {
39    if (((comando[1] ==
    '1')&&(comando[3] == '5')) ||
    ((comando[1] == '1')&&(comando[3] ==
    '4')))
40    {
41      digitalWrite(input1, LOW);
42      digitalWrite(input2, HIGH);
43      digitalWrite(input3, LOW);
44      digitalWrite(input4, HIGH);
45    }
46
47    if (((comando[1] ==
    '1')&&(comando[3] == '1')) ||
    ((comando[1] == '1')&&(comando[3] ==
    '2')))
48    {
49      digitalWrite(input1, HIGH);
50      digitalWrite(input2, LOW);
51      digitalWrite(input3, HIGH);
52      digitalWrite(input4, LOW);
```

```
53    }
54
55    if (((comando[1] ==
    '2')&&(comando[3] == '5')) ||
    ((comando[1] == '2')&&(comando[3] ==
    '4')))
56    {
57      digitalWrite(input1, LOW);
58      digitalWrite(input2, HIGH);
59      digitalWrite(input3, LOW);
60      digitalWrite(input4, LOW);
61    }
62
63    if (((comando[1] ==
    '2')&&(comando[3] == '1')) ||
    ((comando[1] == '2')&&(comando[3] ==
    '2')))
64    {
65      digitalWrite(input1, LOW);
66      digitalWrite(input2, LOW);
67      digitalWrite(input3, LOW);
68      digitalWrite(input4, HIGH);
69    }
70
71    if (((comando[1] ==
    '1')&&(comando[3] == '3')) ||
    ((comando[1] == '2')&&(comando[3] ==
    '3')))
72    {
73      digitalWrite(input1, LOW);
74      digitalWrite(input2, LOW);
75      digitalWrite(input3, LOW);
76      digitalWrite(input4, LOW);
77    }
78 }
```

5.2 SKETCH DA MOVIMENTAÇÃO DO CARRO USANDO O JOYSTICK

```
1  int JoyStick_X = A5;
2  int JoyStick_Y = A4;
3  int JoyStick_Z = 0;
4
5  void setup()
6  {
7    pinMode(JoyStick_Z, INPUT_PULLUP);
8    Serial.begin(9600);
9  }
10
11 void loop()
12 {
13   int x, y, z;
14
15   x = analogRead (JoyStick_X);
16   y = analogRead (JoyStick_Y);
17   z = digitalRead (JoyStick_Z);
18
19   if(x <= 204){
20     Serial.println("0105");
21   } else if((x > 204) && (x < 410)){
22     Serial.println("0104");
23   } else if((x >= 410) && (x < 615))
   {
24     Serial.println("0103");
25   } else if((x >= 615) && (x < 820))
   {
26     Serial.println("0102");
27   } else if(x >= 820){
28     Serial.println("0101");
29   }
30
```

```
31    if(y <= 204){
32      Serial.println("0205");
33    }else if((y > 204) && (y < 410)){
34      Serial.println("0204");
35    }else if((y >= 410) && (y < 615)){
36      Serial.println("0203");
37    }else if((y >= 615) && (y < 820)){
38      Serial.println("0202");
39    }else if(y >= 820){
40      Serial.println("0201");
41    }

42  if(z == 0){
43      Serial.println("0301");
44    }
45  }
```

5.3 SKETCH DO CONTROLE COM EMISSOR RF

```
1   #include <VirtualWire.h>
2   int JoyStick_X = A5;
3   int JoyStick_Y = A4;
4   int JoyStick_Z = 0;
5   void setup ()
6   {
7     vw_set_tx_pin(8);
8     vw_setup(2000);
9
10    pinMode (JoyStick_X, INPUT);
11    pinMode (JoyStick_Y, INPUT);
12    pinMode (JoyStick_Z, INPUT_
    PULLUP);
13    Serial.begin (9600);
14  }
15
16  void loop ()
17  {
```

```
18   int x, y, z;
19
20   x = analogRead (JoyStick_X);
21   y = analogRead (JoyStick_Y);
22   z = digitalRead (JoyStick_Z);
23
24   Serial.print (x, DEC);
25   Serial.print (",");
26   Serial.print (y, DEC);
27   Serial.print (",");
28   Serial.println (z, DEC);
29
30     if(x <= 204){
31       enviar("0105");
32     }else if((x < 410) && (x > 204))
{
33       enviar("0104");
34     }else if((x >= 410) && (x <
615)){
35       enviar("0103");
36     }else if((x >= 615) && (x <
820)){
37       enviar("0102");
38     }else if(x >= 820){
39       enviar("0101");
40     }
41
42     if(y <= 204){
43       enviar("0205");
44     }else if((y < 410) && (y > 204))
{
45       enviar("0204");
46     }else if((y < 615) && (y > 410))
{
47       enviar("0203");
48     }else if((y >= 615) && (y <
820)){
49       enviar("0202");
50     }else if(y >= 820){
```

```
51        enviar("0201");
52      }
53
54    if(z == 0){
55      enviar("0301");
56    }
57 }
58
59 void enviar(char* text)
60 {
61   Serial.println(text);
62   vw_send((uint8_t *)text,
   strlen(text));
63   vw_wait_tx();
64 }
```

5.4 SKETCH COMPLETO DO CARRO COM RECEPTOR RF

```
1  #include <VirtualWire.h>
2  #include <stdlib.h>
3
4  byte message[VW_MAX_MESSAGE_LEN];
5
6  byte msgLength = VW_MAX_MESSAGE_LEN;
7  char valorAtual[4];
8  int tempo = 500;
9
10 int input1 = 11;
11 int input2 = 10;
12 int input3 = 4;
13 int input4 = 3;
14
15 void setup()
16 {
17   vw_set_rx_pin(7);
18   vw_setup(2000);
```

```
  vw_rx_start();
  pinMode(input1, OUTPUT);
  pinMode(input2, OUTPUT);
  pinMode(input3, OUTPUT);
  pinMode(input4, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  char comando[4];

  uint8_t message[VW_MAX_MESSAGE_
LEN];
  uint8_t msgLength = VW_MAX_
MESSAGE_LEN;

  if (vw_get_message(message,
&msgLength))
  {
    for (int i = 0; i < msgLength;
i++)
    {
       comando[i] = message[i];
    }
    comando[4] = '\0';
    movimentar(comando);
    Serial.println(comando);
  }
}

void movimentar(char* comando)
{
```

```
48   if (((comando[1] ==
     '1')&&(comando[3] == '5')) ||
     ((comando[1] == '1')&&(comando[3] ==
     '4')))
49   {
50     digitalWrite(input1, LOW);
51     digitalWrite(input2, HIGH);
52     digitalWrite(input3, LOW);
53     digitalWrite(input4, HIGH);
54     delay(tempo);
55     digitalWrite(input1, LOW);
56     digitalWrite(input2, LOW);
57     digitalWrite(input3, LOW);
58     digitalWrite(input4, LOW);
59   }
60
61   if (((comando[1] ==
     '1')&&(comando[3] == '1')) ||
     ((comando[1] == '1')&&(comando[3] ==
     '2')))
62   {
63     digitalWrite(input1, HIGH);
64     digitalWrite(input2, LOW);
65     digitalWrite(input3, HIGH);
66     digitalWrite(input4, LOW);
67     delay(tempo);
68     digitalWrite(input1, LOW);
69     digitalWrite(input2, LOW);
70     digitalWrite(input3, LOW);
71     digitalWrite(input4, LOW);
72   }
73
74   if (((comando[1] ==
     '2')&&(comando[3] == '5')) ||
```

```
    ((comando[1] == '2')&&(comando[3] ==
    '4')))
75    {
76      digitalWrite(input1, LOW);
77      digitalWrite(input2, HIGH);
78      digitalWrite(input3, LOW);
79      digitalWrite(input4, LOW);
80      delay(tempo);
81      digitalWrite(input1, LOW);
82      digitalWrite(input2, LOW);
83      digitalWrite(input3, LOW);
84      digitalWrite(input4, LOW);
85    }
86
87    if (((comando[1] ==
    '2')&&(comando[3] == '1')) ||
    ((comando[1] == '2')&&(comando[3] ==
    '2')))
88    {
89      digitalWrite(input1, LOW);
90      digitalWrite(input2, LOW);
91      digitalWrite(input3, LOW);
92      digitalWrite(input4, HIGH);
93      delay(tempo);
94      digitalWrite(input1, LOW);
95      digitalWrite(input2, LOW);
96      digitalWrite(input3, LOW);
97      digitalWrite(input4, LOW);
98    }
99
100     if (((comando[1] ==
    '1')&&(comando[3] == '3')) ||
    ((comando[1] == '2')&&(comando[3] ==
    '3')))
```

```
101    {
102      digitalWrite(input1, LOW);
103      digitalWrite(input2, LOW);
104      digitalWrite(input3, LOW);
105      digitalWrite(input4, LOW);
106    }
107 }
```

5.5 CÓDIGO DO CARRO AUTÔNOMO PARA TESTE

```
1  #define TRIGGER_PIN  12
2  #define ECHO_PIN     13
3  volatile int estado = LOW;
4
5  void setup() {
6    attachInterrupt(0, chaveOnOff,
   RISING);
7    pinMode(TRIGGER_PIN, OUTPUT);
8    pinMode(ECHO_PIN, INPUT);
9    pinMode(2, INPUT);
10   Serial.begin(9600);
11 }
12
13 void loop() {
14   float cm;
15   long microsec = detectar();
16   cm = (microsec/27.6233)/2;
17
18   Serial.print("CM = ");
19   Serial.println(cm);
20   Serial.print("Estado da Chave= ");
21   Serial.println(estado);
22   delay(100);
23 }
24
25 long detectar() {
26   digitalWrite(TRIGGER_PIN, LOW);
27   delayMicroseconds(2);
28   digitalWrite(TRIGGER_PIN, HIGH);
```

```
29   delayMicroseconds(10);
30   digitalWrite(TRIGGER_PIN, LOW);
31   return pulseIn(ECHO_PIN, HIGH);
32 }
33
34 void chaveOnOff() {
35   static unsigned long lastMillis =
   0;
36   unsigned long newMillis =
   millis();
37   if(newMillis - lastMillis < 50) {
38   }
39   else {
40     estado = !estado;
41     lastMillis = newMillis;
42   }
43 }
```

5.6 SKETCH COMPLETO DO CARRO AUTÔNOMO

```
1  #define TRIGGER_PIN  12
2  #define ECHO_PIN     13
3  volatile int estado = LOW;
4  int direcao = 1;
5  int distancia = 10;
6  int input1 = 11;
7  int input2 = 10;
8  int input3 = 4;
9  int input4 = 3;
10
11 void setup() {
12   attachInterrupt(0, chaveOnOff,
   RISING);
13   pinMode(TRIGGER_PIN, OUTPUT);
14   pinMode(ECHO_PIN, INPUT);
15   pinMode(2, INPUT);
```

```
16    pinMode(input1, OUTPUT);
17    pinMode(input2, OUTPUT);
18    pinMode(input3, OUTPUT);
19    pinMode(input4, OUTPUT);
20    randomSeed(analogRead(0));
21    Serial.begin(9600);
22  }
23
24  long detectar() {
25    digitalWrite(TRIGGER_PIN, LOW);
26    delayMicroseconds(2);
27    digitalWrite(TRIGGER_PIN, HIGH);
28    delayMicroseconds(10);
29    digitalWrite(TRIGGER_PIN, LOW);
30    return pulseIn(ECHO_PIN, HIGH);
31  }
32
33  void loop() {
34    float cm;
35    long microsec = detectar();
36
37    cm = (microsec/27.6233)/2;
38    direcao = random(2);
39
40    if (estado) {
41      if (cm > distancia) {
42        digitalWrite(input1, LOW);
43        digitalWrite(input2, HIGH);
44        digitalWrite(input3, LOW);
45        digitalWrite(input4, HIGH);
46        delay(100);
47      } else {
48        if (direcao) {
49          digitalWrite(input1, HIGH);
```

```
50          digitalWrite(input2, LOW);
51          digitalWrite(input3, HIGH);
52          digitalWrite(input4, LOW);
53          delay(75);
54          digitalWrite(input1, LOW);
55          digitalWrite(input2, HIGH);
56          digitalWrite(input3, HIGH);
57          digitalWrite(input4, LOW);
58          delay(75);
59        } else {
60          digitalWrite(input1, HIGH);
61          digitalWrite(input2, LOW);
62          digitalWrite(input3, HIGH);
63          digitalWrite(input4, LOW);
64          delay(75);
65          digitalWrite(input1, HIGH);
66          digitalWrite(input2, LOW);
67          digitalWrite(input3, LOW);
68          digitalWrite(input4, HIGH);
69          delay(75);
70        }
71      }
72    } else {
73      digitalWrite(input1, LOW);
74      digitalWrite(input2, LOW);
75      digitalWrite(input3, LOW);
76      digitalWrite(input4, LOW);
77    }
78  }
79
80  void chaveOnOff() {
81    static unsigned long lastMillis =
    0;
82    unsigned long newMillis =
```

```
    millis();
83    if(newMillis - lastMillis < 50) {
84    }
85    else {
86      estado = !estado;
87      lastMillis = newMillis;
88    }
89  }
```

5.7 TABELA COMPLETA DOS VALORES DAS PORTAS LÓGICAS INPUT DO L293D

Tabela 8 – Todas as Combinações de Portas Lógicas Possíveis

Nº	Input 1	Input 2	Input 3	Input 4	O carro faz o movimento de...
	Motor 1 – Esquerdo		**Motor 2 – Direito**		
1	LOW	LOW	LOW	LOW	Freio, pois ambos os motores estão parados.
	Parado		**Parado**		
2	LOW	HIGH	LOW	LOW	Curva à direita, pois apenas o motor esquerdo acelera.
	Frente		**Parado**		
3	LOW	LOW	LOW	HIGH	Curva à esquerda, pois apenas o motor direito acelera.
	Parado		**Frente**		
4	HIGH	LOW	HIGH	LOW	Retardação, pois os dois motores retardam (trás).
	Trás		**Trás**		
5	LOW	HIGH	LOW	HIGH	Aceleração, pois os dois motores aceleram (frente).
	Frente		**Frente**		
6	LOW	HIGH	HIGH	LOW	Roda para a direita, pois o motor esquerdo acelera e o motor direito retarda.
	Frente		**Trás**		
7	HIGH	LOW	LOW	HIGH	Roda para a esquerda, pois o motor direito acelera e o motor esquerdo retarda.
	Trás		**Frente**		

8	HIGH	LOW	LOW	LOW	Curva à esquerda, pois apenas o motor esquerdo retarda.
	Trás		**Parado**		
9	LOW	LOW	HIGH	LOW	Curva à direita, pois apenas o motor direito retarda.
	Parado		**Trás**		
10	HIGH	HIGH	LOW	LOW	Freio, pois o motor esquerdo está em curto circuito e o motor direito está parado.
	Curto Circuito		**Parado**		
11	LOW	LOW	HIGH	HIGH	Freio, pois o motor direito está em curto circuito e o motor esquerdo está parado.
	Parado		**Curto Circuito**		
12	HIGH	HIGH	HIGH	LOW	Curva à direita, pois apenas o motor direito retarda, o motor esquerdo está em curto circuito.
	Curto Circuito		**Trás**		
13	HIGH	HIGH	LOW	HIGH	Curva à esquerda, pois apenas o motor direito acelera, o motor esquerdo está em curto circuito.
	Curto Circuito		**Frente**		
14	HIGH	LOW	HIGH	HIGH	Curva à esquerda, pois apenas o motor esquerdo retarda, o motor direito está em curto circuito.
	Trás		**Curto Circuito**		
15	LOW	HIGH	HIGH	HIGH	Curva à direita, pois apenas o motor esquerdo acelera, o motor direito está em curto circuito.
	Frente		**Curto Circuito**		
16	HIGH	HIGH	HIGH	HIGH	Freio, pois ambos os motores estão em curto circuito.
	Curto Circuito		**Curto Circuito**		

Bibliografia

ARDUINO. www.arduino.cc. **L293D**, 2015. Disponivel em: <http://www.arduino.cc/documents/datasheets/L293D.pdf>. Acesso em: 2015 Novembro 15.

ARDUINO. www.arduino.cc. **Arduino Board Uno**, 2015. Disponivel em: <https://www.arduino.cc/en/Main/ArduinoBoardUno>. Acesso em: 15 Novembro 2015.

ATMEL. www.atmel.com/products/microcontrollers/avr/. **Atmel AVR 8-bit and 32-bit Microcontrollers**, 10 out. 2015. Disponivel em: <http://www.atmel.com/products/microcontrollers/avr/>. Acesso em: 10 jun. 2015.

COMPUTER HISTORY MUSEUM. www.computerhistory.org. **1965:** Package is the First to Accommodate System Design Considerations, 2014. Disponivel em: <http://www.computerhistory.org/semiconductor/timeline/1965-Package.html>. Acesso em: 25 dez. 2014.

DA SILVA FILHO, J. I.; TORRES, C. R.; ABE, J. M. Robô Móvel Autônomo Emmy: Uma Aplicação eficiente da Lógica Paraconsistente Anotada. **Revista Seleção Documental**, Santos, v. 1, p. 19-26, mar. 2006.

EVANS, M.; NOBLE, J.; HOCHENBAUM, J. **Arduino em Ação**. 1ª. ed. São Paulo: Novatec, 2013. 424 p.

FOWLER, R. **Fundamentos da Eletricidade-corrente Contínua e Magnetismo**. 7ª. ed. New York: The Mc Graw-Hill Companies, v. I, 2013. 256 p.

IBM. www.ibm.com/developerworks/br/library/os-openhardware/. **Hardware Aberto:** Como e Quando Funciona, 06 dez. 2010. Disponivel em: <https://www.ibm.com/developerworks/br/library/os-openhardware/>. Acesso em: 10 jun. 2015.

INSTITUTO NEWTON C. BRAGA. www.newtoncbraga.com.br. **Como funciona o motor de corrente contínua (ART476)**, 2014. Disponivel em: <http://www.newtoncbraga.com.br/index.php/como-funciona/3414-art476a>. Acesso em: 16 Janeiro 2015.

JUNIOR, A. L. **Eletricidade e Eletrônica Básica**. 4ª. ed. Rio de Janeiro: Starlin Alta, 2013. 320 p.

KURY, A. D. G. **Minidiconário Gama Kury da Língua Portuguesa**. 1ª. ed. São Paulo: FTD, 2001. 860 p.

MCROBERTS, M. **Arduino Básico**. Tradução de Rafael Zanolli. 1ª. ed. São Paulo: Novatec, 2011. 456 p.

MICROPIK. www.micropik.com. **HCSR04**, 2015. Disponivel em: <http://www.micropik.com/PDF/HCSR04.pdf>. Acesso em: 10 jun. 2015.

MONK, S. **30 projetos com Arduino**. 1ª. ed. Porto Alegre: Bookman, 2014. 228 p.

RODITI, I. **Dicionário Houaiss de Física**. 1ª. ed. Rio de Janeiro: Objetiva, 2005. 264 p.

SCHILDIT, H. **C Completo e Total**. Tradução de Roberto Carlos Mayer. 3ª. ed. São Paulo: Makron Books, 1996. 810 p.

SPARKFUN ELETRONICS. http://cdn.sparkfun.com/. **Magician Chassis**, 2013. Disponivel em: <http://cdn.sparkfun.com/datasheets/Robotics/MagicianChassisInst.pdf>. Acesso em: 10 jun. 2015.

SPARKFUN ELETRONICS. www.sparkfun.com. **Magician Chassis**, 2015. Disponivel em: <https://www.sparkfun.com/products/retired/12866>. Acesso em: 10 jun. 2015.

TAFFNER, M. A.; XEREZ, M. D.; RODRIGUES FILHO, I. W. **Redes Neurais Artificiais:** Intridução e Princípios de Neurocomputação. 1ª. ed. Blumenau: Editora da FURB., 1995. 199 p.

TETRASYS. wiki.tetrasys-design.net. **HC-SR04 Ultrasonic Sensor Library**, 2015. Disponivel em: <http://wiki.tetrasys-design.net/HCSR04Ultrasonic>. Acesso em: 10 jun. 2015.

TORRES, G. **Hardware Curso Completo**. 4ª. ed. Rio de Janeiro: Axcel Books do Brasil, 2001. 1440 p.

WIKIMEDIA. commons.wikimedia.org. **Nickel-metal Hydride Batteries**, 8 Fevereiro 2005. Disponivel em: <https://commons.wikimedia.org/wiki/Category:NiMH_batteries#mediaviewer/File:NiMH_2500mAh.jpg>. Acesso em: 10 jun. 2015.

WIKIMEDIA. commons.wikimedia.org. **Potentiometer**, 27 Novembro 2006. Disponivel em: <https://commons.wikimedia.org/wiki/File:Potentiometer.jpg?uselang=pt-br>. Acesso em: 10 jun. 2015.

WIKIMEDIA. commons.wikimedia.org. **L293D Motor Driver**, 11 Agosto 2009. Disponivel em: <https://commons.wikimedia.org/wiki/File:L293D_Motor_Driver.jpg>. Acesso em: 10 out. 2015.

WIKIMEDIA. commons.wikimedia.org. **Momentary Switch, Square (shaded)**, 9 Novembro 2009. Disponivel em: <https://commons.wikimedia.org/wiki/File:Momentary_Switch,_Square_(shaded).svg?uselang=pt-br>. Acesso em: 10 jun. 2015.

WIKIMEDIA. commons.wikimedia.org. **Arduino Mega**, 12 Julho 2010. Disponivel em: <https://commons.wikimedia.org/wiki/File:Arduino_Mega.jpg>. Acesso em: 10 jun. 2015.

WIKIMEDIA. commons.wikimedia.org. **Arduino Uno 004**, 27 Março 2013. Disponivel em: <https://commons.wikimedia.org/wiki/File:Arduino_Uno_004.jpg>. Acesso em: 09 jun. 2015.

WIKIMEDIA. commons.wikimedia.org. **Arduino Uno – R3**, 21 Janeiro 2013. Disponivel em: <https://commons.wikimedia.org/wiki/File:Arduino_Uno_-_R3.jpg>. Acesso em: 10 jun. 2015.

WIKIMEDIA. commons.wikimedia.org. **HC SR04 Ultrasonic sensor 148032234 HDR Enhancer**, 8 Janeiro 2014. Disponivel em: <https://commons.wikimedia.org/wiki/File:HC_SR04_Ultrasonic_sensor_1480322_3_4_HDR_Enhancer.jpg?uselang=pt-br>. Acesso em: 06 out. 2015.

WIKIMEDIA. commons.wikimedia.org. **Digital Multimeter Aka**, 19 Fevereiro 2015. Disponivel em: <http://commons.wikimedia.org/wiki/File:Digital_Multimeter_Aka.jpg?uselang=pt-br>. Acesso em: 10 dez. 2015.