Universidade de São Paulo

Trabalho de Conclusão de Curso

Proposta de Sistema de Desvio de Obstáculos em Robótica Móvel

Autora:
Tatiani Pivem

Orientador:
Edson Gesualdo

Laboratório de Controle e Instrumentação
Departamento de Engenharia Elétrica e Computação

Novembro de 2015

.

Tatiani Pivem

Proposta de Sistema de Desvio de Obstáculos em Robótica Móvel

Trabalho de Conclusão de Curso apresentado a Escola de Engenharia de São Carlos.

Orientador: Edson Gesualdo

São Carlos
2015

Pivem, Tatiani
P579p Proposta de sistema de desvio de obstáculos em robótica móvel / Tatiani Pivem; orientador Edson Gesualdo. São Carlos, 2015.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2015.

1. Sistemas Embarcados. 2. Desvio de Obstáculos. 3. Robótica. I. Título.

FOLHA DE APROVAÇÃO

Nome: Tatiani Pivem

Título: "Sistema de desvio de obstáculos"

Trabalho de Conclusão de Curso defendido e aprovado em 30 / 11 / 2015,

com NOTA 10,0 (dez), **pela Comissão Julgadora:**

Prof. Assistente Edson Gesualdo - (Orientador - SEL/EESC/USP)

Prof. Adjunto Danilo Hernane Spatti - UTFPR

Mestre Mauricio Eiji Nakai - (Doutorando - SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Dr. José Carlos de Melo Vieira Júnior

A Léia Pivem e a Ana Cristina Pivem...

Agradecimentos

Agradeço a Deus por abrir a porta do entendimento e nos deixar trilhar e seguir nossos sonhos, a minha família, amigos, ao Professor Edson por sempre me motivar e por todo o conhecimento transmitido, pela paciência inclusive. Agradeço a TODAS as pessoas que transmitem seu conhecimento e que geram um ciclo de inovação evolutivo, agradeço a todos os criadores de código de arduino, vocês são incríveis! Filipe Flop, Criando robô com arduino, dentre tantos. Agradeço aos amigos da "baia dos estagiários - baia do amor", afinal, o carrinho está andando!!!

Obrigada!

"O que prevemos raramente ocorre; o que menos esperamos geralmente acontece."

Benjamin Disraeli

UNIVERSITY OF SÃO PAULO - USP

Abstract

São Carlos School of Engineering
Department of Electrical and Computer Engineering

Trabalho de Conclusão do Curso

System of Obstacle Avoidance

by Tatiani Pivem

This work is about the study and project of a obstacle avoidance system with an autonomous/manual control, based on ultrasonic sensor. About the commercial devices available in the market, there was the option of choice based in the benefit-cost ratio, by studying and testing the components to be used. The system works with manual control until the obstacle appears, then it changes to automatic control. The end result is a complete station of work, capable to move by itself, communicating with the user via radio and autonomy to avoid obstacles and perform the original direction route, as well as mapping the ground by a digital compass and measure the temperature and humidity, ensuring the best performance.

UNIVERSIDADE DE SÃO PAULO - USP

Resumo

Escola de Engenharia de São Carlos
Departmento de Engenharia Elétrica e Computação

Trabalho de Conclusão do Curso

Sistema de Desvio de Obstáculos

por Tatiani Pivem

Este trabalho consiste no projeto de um sistema autônomo de desvio de obstáculos com controle autônomo/manual, baseado em sensor ultrassonico. Entre os dispositivos comerciais disponíveis no mercado fez-se a opção de escolha baseada na relação custo-benefício, estudando-se e testando-se os componentes a serem utilizados. O sistema trabalha com controle manual até o aparecimento de obstáculo, então ele troca para o controle automático. O resultado final é uma estação de trabalho completa, capaz de se mover independentemente, comunicar com o usuário via radio e com autonomia para o desvio de obstáculos e correção de direção da rota original, tão bem quanto o mapeamento do solo por meio de uma bússola digital e medição de temperatura e umidade, assegurando a melhor performance.

Sumário

Lista de Figuras

Lista de Tabelas

Abreviações

IoT	**I**nternet **o**f **T**hings
RFID	**R**adio **F**requency **ID**entification
GND	**G**rou**ND**
I/O	**I**nput/**O**utput
PWM	**P**ulse **W**idth **M**odulation
SRAM	**S**tatic **R**andom **A**ccess **M**emory
EEPROM	**E**lectrically **E**rasable **P**rogrammable **R**ead **O**nly **M**emory
DC	**D**irect **C**urrent
CC	**C**orrente **C**ontínua
AC/DC	**A**lternating **C**urrent/**D**irect **C**urrent
AREF	**A**nalogue **REF**erence
Vin	**V**oltage **in**put
RF	**R**adio **F**requency
TCCRx	**T**imer **C**ounter **C**ontrol **R**egister
TCNTx	**T**imer **C**ounter **R**egister
ICRx	**I**nput **C**apture **R**egister
OCRx	**O**utput **C**ompare **R**egister
TIMSKx	**T**imer **C**ounter **I**nterrupt **Mask** register
ISR	**I**nterrupt **S**ervice **R**outine
CPU	**C**entral **P**rocessing **U**nit
LiPo	**Li**thium ion **Po**lymer battery
I2C	**I**nter **IC**
SDA	**S**ignal **DA**ta
SCL	**S**ignal **CL**ock
CMPS	**C**o**MP**a**S**s

MSB	**M**ost **S**ignificant **B**it
UR	**U**midade **R**elativa
LCD	**L**iquid **C**rystal **D**isplay
LSB	**L**east **S**ignificant **B**it
V	**V**olt
A	**A**mpère
rpm	**r**otações **p**or **m**inuto
ICSP	**I**n-**C**ircuit **S**erial **P**rogramming

Capítulo 1

Introdução

A conectividade inteligente de dispositivos físicos, conhecida mais popularmente como *"Internet of Things"* (IoT) [1], está em alto crescimento nas últimas décadas conforme mostrado pelos meios de comunicação em massa atualmente.*"Internet of Things"* não é um novo conceito, existindo antes mesmo do ano 2000, e formalizado por Kevin Ashton o qual, em suas pesquisas sobre Internet e RFID, publicou em um artigo no RFID Journal em 1999 do qual extraiu-se o seguinte trecho:

"Se tivéssemos computadores que soubessem tudo o que existe para se conhecer sobre coisas - usando dados recolhidos sem ajuda humana- nós estaríamos aptos a monitorar e contar tudo, e evitar eficientemente desperdícios, perdas e aumento dos custos. Poderíamos saber coisas que precisam ser substituídas, reparadas ou recolhidas. Se estão frescas ou passadas do ponto. Precisamos capacitar os computadores com seus próprios meios de recolher informações, para que eles possam ver, ouvir e sentir o cheiro do mundo por si mesmos, em toda sua glória aleatória. RFID e tecnologia de sensores permitem que os computadores observem, identifiquem e compreendam o mundo - sem as limitações dos dados fornecidos pelo homem [2]."

Redes de sensores além de ser um ramo de *"Internet of Things"* é também muito utilizado em Sistemas Embarcados. Um sistema embarcado (ou sistema embutido) é um sistema microprocessado no qual o computador é completamente encapsulado ou dedicado ao dispositivo ou sistema que ele controla[3] .

O crescimento dos ramos de *"Internet of Things"* e Sistemas Embarcados demonstra a importância e a necessidade do recolhimento de dados do mundo que nos circunda por meio de sensores. Este trabalho situa-se nesse contexto e fará uso e estudará alguns sensores e componentes necessários para um sistema de desvio de obstáculos.

Sistemas de Desvios de Obstáculos

A área de sistemas embarcados comporta diferentes procedimentos para se estudar um sistema de desvio de obstáculos. Pode ser direcionado para a área de *software* e elaboração de algoritmos e programas ou para a área de *hardware*, identificando os blocos fundamentais como percepção de obstáculos, sistema de processamento, controle de operação e transmissão de informação, se necessário.

Tendo em vista melhorias por *software*, pode se citar a computação reconfigurável e a análise de imagens em visão computacional. Um robô com sensor laser e câmera pode ser usado para tal estudo usando computação reconfigurável conforme descrito em [4], com o exemplo do Robô Pioneer AT mostrado na figura 1.1. Controladores podem ser desenvolvidos para o objetivo de desvio de obstáculos com sensores laser e ultrassônicos, câmera, computador embarcado, como é o caso do Robô PIONEER 2-DX descrito em [5] e mostrado na figura 1.2.

O protótipo desenvolvido neste trabalho foi batizado de Robô Julius e é mostrado na figura 1.3. Aborda soluções em *software* e *hardware* visto que seu objetivo é a integração e estudo dos blocos necessários ao sistema de desvio já descritos. O projeto do *hardware* busca soluções para a situação em que o protótipo permanece em deslocamento e o desenvolvimento do *software* busca melhor desempenho na utilização de *timers* e interrupções por mais de um componente.

FIGURA 1.1: Robô Pioneer AT do Laboratório de Robótica Móvel ICMC-USP com sensor laser e câmera[4]. Fonte: http://www.teses.usp.br/teses/disponiveis/55/55134/tde-30032010-092432/pt-br.php acessado em 04/09/2015

FIGURA 1.2: Robô móvel a rodas PIONEER 2-DX[5].
Fonte: http://portais4.ufes.br/posgrad/teses/tese_2359_DissertacaoMestradoFlavioGarciaPereira.pdf
acessado em 04/09/2015

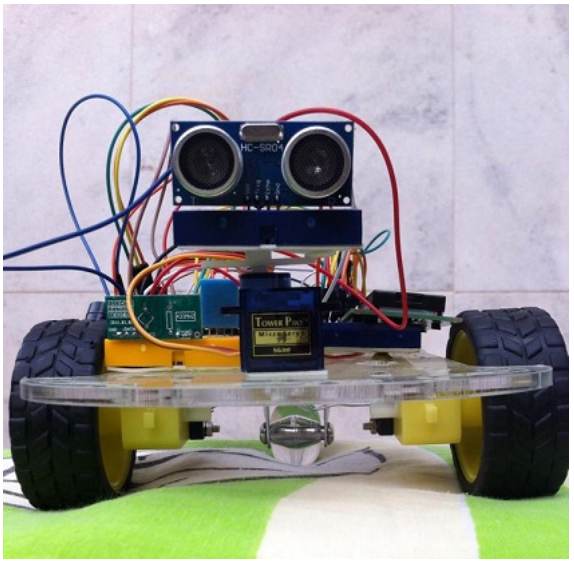

FIGURA 1.3: Robô Julius

Capítulo 2

Sistema de Desvio de Obstáculos

2.1 Protótipo

O protótipo se baseia na presença ou ausência de obstáculos, caso não exista obstáculo a frente, o usuário possui o controle do sistema (no caso, o controle remoto do protótipo), caso o obstáculo seja alcançado, o protótipo entra no modo automático, negando o controle ao usuário até que o obstáculo seja desviado e o sentido da rota retornado.

Para a percepção do obstáculo é usado o sensor ultrassônico descrito na seção 3.5. O controle manual é descrito na seção 2.2 e o controle automático na seção 2.3.
O diagrama de blocos do sistema pode ser visto na figura 2.1.

2.2 Controle Manual

O Controle Manual é realizado enquanto a mínima distância a um obstáculo a frente não seja alcançada. Se baseia em um controle remoto com um transmissor e um receptor no carrinho, descritos na seção 3.4, seu diagrama de blocos é mostrado na figura 2.2.

As direções possíveis são condicionadas às características mecânicas do protótipo e pelo sentido de giro do motor, sendo descritas na seção 3.2.

- **Frente**: O protótipo anda para frente;

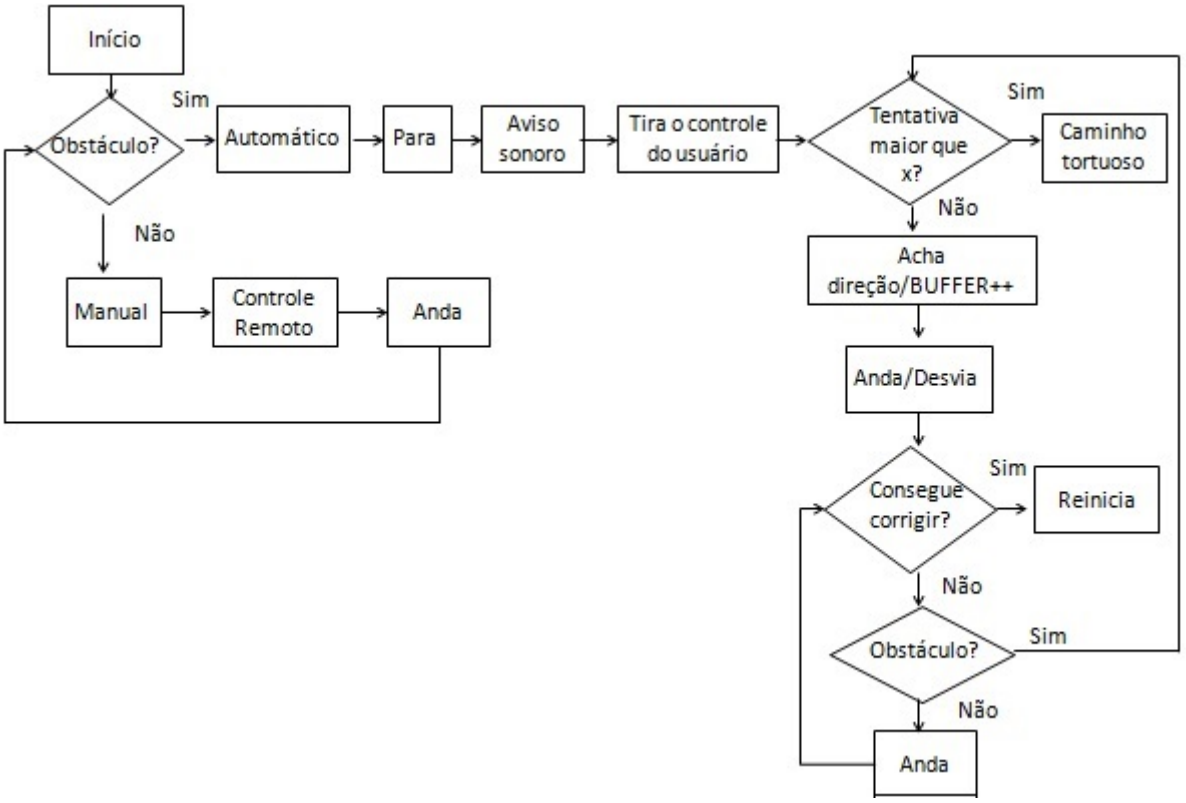

FIGURA 2.1: Sistema de desvio de obstáculos

- **Ré**: O protótipo vai para trás;
- **Esquerda**: O protótipo vira para a esquerda;
- **Direita**: O protótipo vira para a direita;
- **Parar**: O protótipo para.

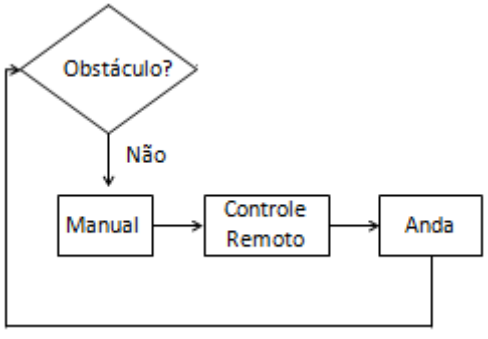

FIGURA 2.2: Diagrama de blocos controle manual

2.3 Controle Automático

O Controle Automático é realizado quando a mínima distância permitida ao obstáculo a frente é alcançada. Como mostra o diagrama de blocos da figura 2.3, o sistema efetua a parada do protótipo, um aviso sonoro é dado ao usuário, o controle do usuário é retirado e o sistema começa no seu ciclo de funcionamento até que o número máximo de tentativas é alcançado ou o obstáculo seja desviado e sua rota retornada.

O número máximo de tentativas depende dos requisitos de processamento e armazenamento, pois cada obstáculo encontrado no caminho é guardado como uma direção a ser corrigida em pilha (última direção guardada é a primeira a ser corrigida). Essa metodologia pode ser melhor entendida com a figura 2.4, na qual o círculo verde é o protótipo e os obstáculos são representados pelos quadrados a sua frente; para a parte superior da figura, existe um obstáculo a frente do sentido da trajetória do protótipo e um obstáculo a esquerda, sendo portanto sua maior distância disponível para percurso a direita. Desviando do obstáculo, é necessário uma correção de rota que mantenha o sentido inicial de percurso; caso esteja indo ao norte, deve-se manter o norte como referencial, o mesmo vale para os outros pontos cardeais. Nessa correção, caso o obstáculo encontrado o leve a desviar virando a direita, o protótipo deve posteriormente virar a esquerda e manter a rota.

Na parte inferior da figura existe um obstáculo a frente e outro a direita, sendo portanto o desvio usado a esquerda e corrigido virando-se a direita no próximo momento disponível.

Caso o protótipo não consiga corrigir a sua rota, sendo por exemplo o obstáculo muito grande em extensão, o sistema deve continuar com sua direção (caso não haja obstáculo), até que o obstáculo acabe e sua rota seja corrigida, retornando o controle ao usuário e reiniciando o sistema.

Caso o obstáculo não tenha acabado e exista outro obstáculo a frente, o sistema deve verificar se o número máximo de tentativas foi alcançado e caso não seja, o próximo desvio é guardado em um *buffer* para ser corrigido voltando sucessivamente até o primeiro. Só quando o sistema não possui obstáculos a corrigir que é retornado o controle ao usuário. Sendo o número máximo de tentativas alcançado, o sistema retorna a mensagem de terreno tortuoso e desliga.

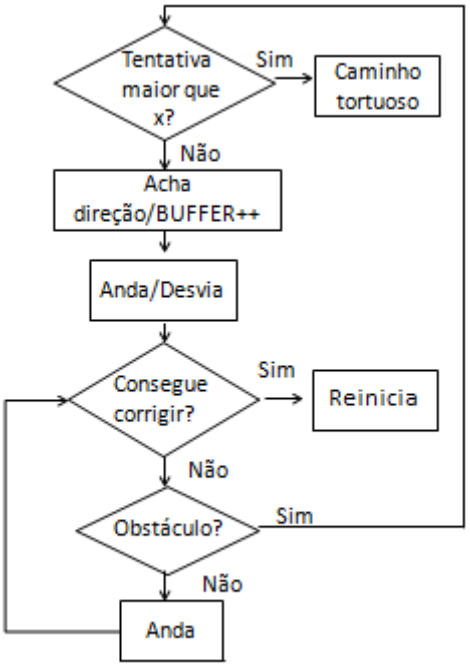

FIGURA 2.3: Diagrama de blocos correção automática

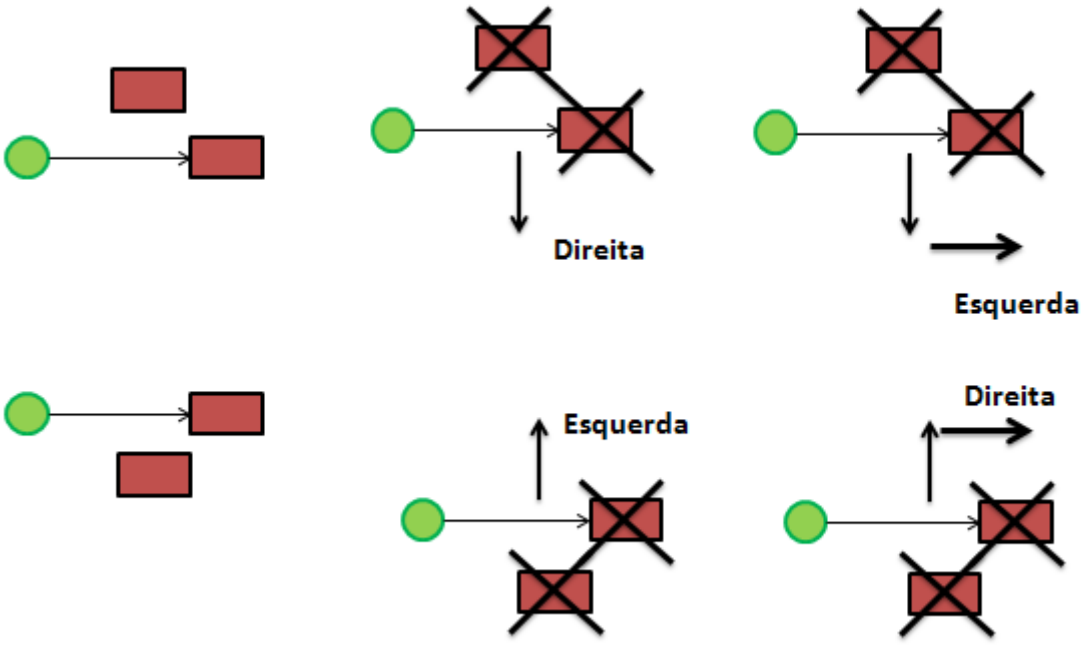

FIGURA 2.4: Desvio automático

Capítulo 3

Fundamentos e Componentes

3.1 Arduino

3.1.1 Arduino UNO

O Arduino UNO[6], mostrado na figura 3.2, é uma placa microcontroladora baseada na versão ATmega328, e trabalha com dados analógicos e digitais, com fácil interação com diversos sensores. Os dados gerais referentes ao Arduino UNO são mostrados na tabela 3.1.

Tabela 3.1: Dados gerais referentes ao Arduino Uno.

	Arduino UNO - Microcontrolador ATMega328
Tensão de Operação	+5 V
Tensão de Entrada (recomendada)	7-12 V
Tensão de Entrada (limites)	6-20 V
Pinos Digitais I/O	14 pinos (dos quais 6 podem ser usados como saída PWM)
Pinos de Entrada Analógica	6 pinos
Corrente CC por Pino I/O	40 mA
Corrente CC para o Pino de 3,3 V	50 mA
Flash Memory	32 kB
SRAM	2 kB
EEPROM	1 kB
Velocidade de *Clock*	16 MHz

3.1.1.1 Barramento de Potência

O barramento de potência, mostrado na figura 3.1, apresenta os seguintes pinos:

GND: Pino *Ground*;

5 V: Fornece uma saída regulada em 5 V (não recomendado utilizá-lo para alimentar a placa);

3,3 V: Fornece a tensão de 3,3 V e sua máxima corrente é de 50 mA;

Vin: Fornece a tensão utilizada para alimentar a placa.

FIGURA 3.1: Barramento de potência.

3.1.1.2 Barramento Digital (I/O)

Possui 14 pinos digitais de entrada/saída, dos quais 6 podem ser usados como saídas PWM, como mostra a figura 3.3; esse barramento se encontra na parte superior da placa. Apresenta nível *High* em 5 V e *Low* em 0 V.

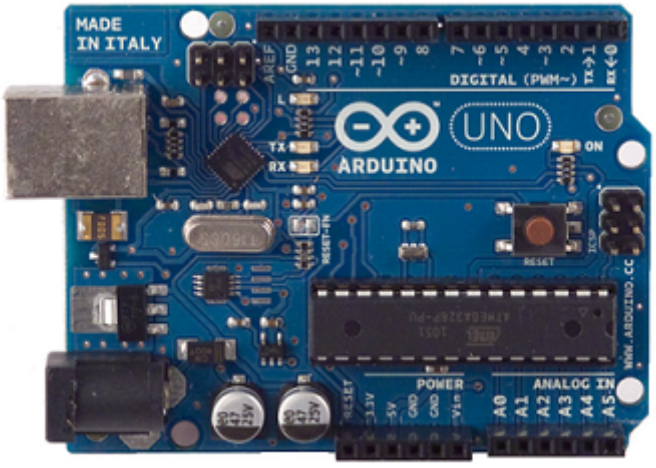

FIGURA 3.2: Arduino UNO.

FIGURA 3.3: Entradas e saídas digitais e PWM referenciado por: ∼.

3.1.1.3 Barramento Analógico

Suas seis entradas analógicas se encontram no canto inferior direito, podem assumir diversos valores de tensão, entre 0 e 5V, como mostra a figura 3.4.

FIGURA 3.4: Entradas analógicas.

3.1.1.4 Entradas e Saídas

Cada um dos 14 pinos digitais pode ser usado como entrada ou saída, com as funções `pinMode()`,`digitalWrite()` e`digitalRead()` . Operam com 5 V e cada pino pode providenciar um máximo de 40 mA.

Alguns pinos de importância para esse trabalho são:

Serial: Usado para receber (RX) e transmitir (TX);

Interrupções Externas (pinos 2 e 3): Podem ser configuradas para interrupção com valor alto ou baixo, com descida ou subida de borda, através da função `attachInterrupt()`;

PWM (pinos 3, 5, 6, 9, 10 e 11): Fornecem saída PWM com a função `analogWrite()`.

Para cada uma das seis entradas analógicas (nomeadas de A0 a A5), se possui 10 *bits* de resolução (i.e. 1024 valores distintos). Por padrão, medem do referencial *ground* a 5V, no entanto pode-se aumentar a extremidade superior da faixa usando o pino **AREF** e a função `analogReference()` .

3.1.1.5 *Timer*

Um *timer* é uma peça do *hardware* presente dentro de placas controladoras Arduino [7] e também em outros controladores, que tem como objetivo medir eventos de tempo. Pode ser programado por alguns registradores especiais.

O controlador do Arduino UNO é o Atmel AVR ATMega 168 ou ATmega328, a diferença entre ambos é o tamanho da memória interna. Ambos tem 3 *timers*, chamados: *timer*0, *timer*1 e *timer*2. *Timer*0 e *timer*2 são ambos de 8*bits* e *timer*1 é de 16 *bits*. A diferença entre um *timer* de 8*bits* e 16*bits* é a sua resolução, 8 *bits* significa que podem haver 256 possibilidades de valores e um *timer* de 16*bits* indica que podem existir 65536 valores, providenciando uma maior resolução e/ou contagem mais longa.

3.1.2 Arduino Mega

O Arduino Mega[8], mostrado na figura 3.5, é uma placa microcontroladora baseada no processador ATmega1280, tem 54 pinos digitais de entradas e/ou saídas (dos quais 14 podem ser usados como saídas PWM), 16 entradas analógicas, 4 UARTs (portas seriais para os *hardwares*), um cristal oscilador de 16 MHz, uma conexão USB, um cabeçário ICSP, e um botão de *reset*. Suas especificações estão contidas na tabela 3.2.

Tabela 3.2: Arduino Mega2560

	Dados Arduino Mega
Tensão de Operação	+5 V
Tensão de Entrada (recomendada)	7-12 V
Tensão de Entrada (limites)	6-20 V
Pinos Digitais I/O	54 pinos dos quais 15 providenciam saídas PWM
Pinos de Entrada Analógica	16 pinos
Corrente CC por Pino I/O	40 mA
Corrente CC para o Pino de 3,3 V	50 mA
Flash Memory	128 kB
SRAM	8 kB
EEPROM	4 kB
Velocidade de *Clock*	16 MHz

3.1.2.1 Alimentação

O Arduino Mega pode ser alimentado via conexão USB ou com alimentação externa.

Externa (não-USB): Pode vir de um adaptador AC/DC ou bateria. O adaptador pode ser conectado com um *plug* de 2,1 mm com centro positivo no *plug* de cor preta da placa. Para se utilizar a tensão de entrada fornecida ao arduino pela bateria, deve-se retirar sua tensão entre os pinos *Vin* e *GND*.

A placa pode operar com uma alimentação externa de 6 a 20 volts. Se alimentada com menos do que 7 V, o pino de 5 V pode alimentar o dispositivo conectado com menos do que 5 V e a placa pode ficar instável. Se usados mais de 12 V, o regulador de tensão pode danificar a placa. A faixa recomendada fica entre 7 V e 12 V.

Baterias de 9 V se encontram próximas ao centro da faixa aceitável, e são fáceis de ser encontradas e de preço popular, esse trabalho fará uso dessa bateria para alimentação do arduino quando não conectado ao computador.

FIGURA 3.5: Arduino mega.

3.1.2.2 *Timer*

O controlador da série Arduino Mega é o `Atmel AVR ATmega1280` ou `ATmega 2560`, com diferença no tamanho da memória interna. Esses controladores tem 6 *timers*. `Timer0`, `timer1`, e `timer2` são idênticos ao `ATmega168/328`. O `timer3`, `timer4` e `timer5` são todos de 16*bits*, similar ao `timer1`.

Os `timers0` e 1 [9] precisam de atenção redobrada para mudanças pois são empregados em funções internas ao arduino e bibliotecas mais conhecidas.

Timer0: 8*bits timer*.

`Timer0` é usado para funções de tempo de uso geral no arduino, como: `delay()`, `millis()` e `micros()`. Se houver mudanças nos registradores do *timer*0, isso irá influenciar tais funções, podendo comprometer seu funcionamento e desempenho caso não recebam tratamento adequado.

Timer1: 16*bits timer*.

No Arduino UNO, o `timer1` é usado pela biblioteca *"Servo library"* e pela biblioteca *"VirtualWire"*, no Arduino Mega se usa o `timer5` para ambas as bibliotecas. Quando juntas em um programa podem não funcionar adequadamente, sendo necessário a mudança dos *timers* ou a mudança de alguma das duas bibliotecas.

Existem bibliotecas para o `timer2` tanto no caso do controle de servos motores (*SoftwareServo*) como no caso de controle RF(*RCSwitch*).

Registradores do Timer

Pode-se mudar o comportamento dos *Timers* pelos registradores de *timer*. Como mostra [9], os mais utilizados são:

`TCCRx` - *Timer/Counter Control Register*. O *prescaler* pode ser configurado através desse registrador.

`TCNTx` - *Timer/Counter Register*. O valor atual do *timer* é armazenado aqui.

`OCRx` - *Output Compare Register*.

`ICRx` - *Input Capture Register* (somente para *timer* de 16 *bits*).

`TIMSKx` - *Timer/Counter Interrupt Mask Register*. Usado para habilitar/desabilitar interrupções de tempo.

`TIFRx` - *Timer/Counter Interrupt Flag Register*. Indica uma interrupção pendente.

3.1.2.3 Interrupções

O programa rodando em um controlador é normalmente sequencial, sendo executado instrução por instrução. Uma interrupção é um evento externo que desvia o fluxo

do programa e executa uma rotina de serviço de interrupção especial (*ISR-Interrupt Service Routine*). Depois da ISR terminar, o programa continua com a próxima instrução de máquina.

Todas as interrupções devem estar habilitadas para que possa ocorrer o tratamento em uma ISR. Pode-se habilitar/desabilitar uma interrupção com a função `interrupts()` e `noInterrupts()`. Por padrão o *firmware* do arduino vem com interrupções habilitadas. *Interrupt masks* também estão habilitadas, e podem ser modificadas setando/limpando *bits* no registrador da *Interrupt mask* (`TIMSKx`).

Quando uma interrupção ocorre, uma *flag* no **interrupt flag register** (`TIFRx`) é setada. Essa interrupção irá automaticamente ser limpa quando entrar na ISR ou pode ser manualmente limpa com o *bit* da **interrupt flag register**.

As funções do arduino `attachInterrupt()` e `detachInterrupt()` podem somente ser usadas para pinos de interrupção externas. A estrutura de `attachInterrupt()` recomendada é a seguinte:

`attachInterrupt(número da interrupção(pino),função de tratamento,TIPO)`, sendo o número da interrupção dado na figura 3.6 com o pino correspondente onde a mudança do TIPO vai ser observada. TIPO, como mostrado em [10] pode-se seguir as opções mostradas na tabela 3.3.

Tabela 3.3: Tipos de operação e desencadeadores da interrupção

	Tipos de interrupção
LOW	O pino do número da interrupção é colocado em nível baixo
CHANGE	O pino do número da interrupção tem seu valor lógico mudado (alto ou baixo)
HIGH	O pino do número da interrupção é colocado em nível alto
RISING	O momento da transição do pino da interrupção de alto para baixo
FALLING	O momento da transição do pino da interrupção de baixo para alto

De acordo com [11] as interrupções são chamadas "vetoradas"por conter um endereço específico, assim existirá um endereço de tratamento da interrupção para cada uma.

A ordem dos endereços determina o nível de prioridade das interrupções, quanto menor o endereço do vetor de interrupção, maior será sua prioridade. Por exemplo, a interrupção `INT0` tem prioridade sobre a `INT1`.

Uno

attachInterrupt	Name	Pin on chip (PDIP)	Pin on board
0	INT0	4	D2
1	INT1	5	D3

Mega2560

attachInterrupt	Name	Pin on chip (TQFP)	Pin on board
0	INT4	6	D2
1	INT5	7	D3
2	INT0	43	D21
3	INT1	44	D20
4	INT2	45	D19
5	INT3	46	D18

Leonardo

attachInterrupt	Name	Pin on chip (TQFP)	Pin on board
0	INT0	18	D3
1	INT1	19	D2
2	INT2	20	D0
3	INT3	21	D1
4	INT6	1	D7

FIGURA 3.6: Mapa de pinos e interrupções fonte:Interrupt names to pin mappings, Nick Gammon, disponível em: http://gammon.com.au/interrupts.

Como mostra a tabela 3.4 retirada da folha de dados do componente Atmel2560 disponível em [13], e mostrado na figura 3.7.

TABELA 3.4: Especificação da interrupção.

	Especificação da interrupção	
Endereço	Interrupção	Função
0000	`RESET`	*Reset* (Maior prioridade)
0002	`INT0`	*External Interrupt Request* 0
0004	`INT1`	*External Interrupt Request* 1
0006	`INT2`	*External Interrupt Request* 2
0008	`INT3`	*External Interrupt Request* 3
000A	`INT4`	*External Interrupt Request* 4
000C	`INT5`	*External Interrupt Request* 5(Menor prioridade)

Ainda de acordo com [11], o ATmega não suporta interrupções aninhadas, isto é, **uma interrupção não pode interromper outra em andamento**, mesmo que tenha maior prioridade. Ocorrendo uma ou mais interrupções enquanto uma está sendo tratada, é formada uma fila de espera que é atendida pela prioridade (prioridades das interrupções externas mostradas na tabela 3.4). Dessa forma, ao tratar uma interrupção a unidade central de processamento (CPU- *Central Processing Unit*) automaticamente desabilita todas as interrupções, voltando a ligá-las ao final da rotina de interrupção.

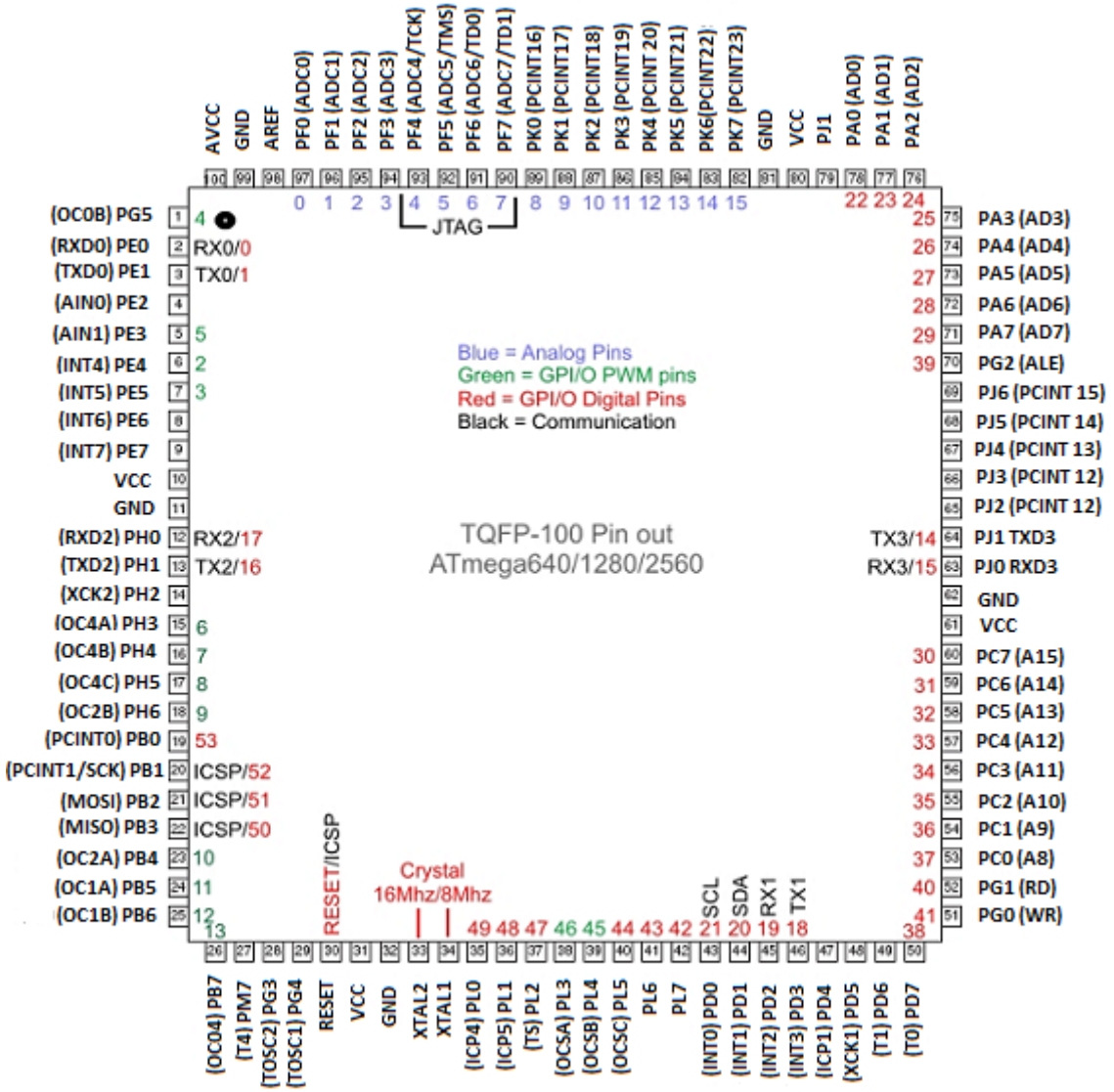

FIGURA 3.7: Diagrama de conexões- Atmel2560 e sua relação com pinos do arduino mega.fonte:Arduino MEGA 2560, Fábio Souza, disponível em:http://www.embarcados.com.br/arduino-mega-2560/ acessado 18/09/2015 [12]

3.2 Chassi

A construção do protótipo para testes em campo do projeto do sistema de desvio de obstáculos utilizou o Chassi MSGX001s, escolhido pela sua robustez mecânica pois possui placa de acrílico de 3mm e dois motores com caixa de redução (1:48). O Chassi é mostrado na figura 3.8 e suas especificações dadas pelo fabricante MAGILUX seguem na tabela 3.5.

FIGURA 3.8: Chassi.

TABELA 3.5: Especificação do chassi MSGX001 fornecidos pelo fabricante MAGILUX para operação a 25 °C

	Especificação do Chassi
Tensão de cada Motor	+3 V a +6 V
Corrente do Motor sem Carga	200 mA (6 V) e 150 mA (3 V)
Velocidade do Motor	200 rpm (6 V) e 90 rpm (3 V)
Dimensão Chassi	22 cm x 14,7 cm
Dimensão da Roda	7 cm x 7 cm x 2,6 cm
Espessura da Plataforma em Acrílico	3 mm

3.2.1 Motores

FIGURA 3.9: Motores CC.

Os motores de corrente contínua (CC ou do inglês *Direct Current* - DC), mostrados na figura 3.9, são dispositivos que operam através das relações de forças de atração e repulsão geradas por eletroímãs ou imãs permanentes.

Dada a teoria eletromagnética, quando um condutor, atravessado por corrente elétrica, é imerso em um campo magnético, surge sobre o condutor uma força mecânica, dada por:

$$F = BiL[N] \tag{3.1}$$

Juntamente com o efeito da Força Eletromagnética, ocorre a Tensão de Velocidade: quando um condutor imerso em um campo magnético é colocado em movimento, surge uma tensão induzida em seus terminais.

$$e = Blv[V] \tag{3.2}$$

Os dois processos ocorrem simultaneamente em qualquer processo de conversão eletromecânica de energia, mas o efeito da Força Eletromagnética prevalece no caso do Motor.

Estator é a parte fixa da máquina e o rotor é a parte móvel, são separadas por um entreferro.

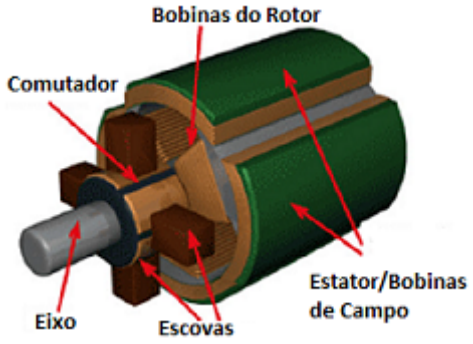

FIGURA 3.10: Esquema interno de um motor

Um elemento muito importante no motor é o comutador, mostrado nas figuras 3.10 e 3.11. Uma vez que o torque surge devido à busca de alinhamento entre os campos do rotor e do estator torna-se necessário o aparecimento de um desbalanço antes do alinhamento. Tal desbalanço é causado pelo comutador que varia continuamente a orientação do campo produzido pela armadura, não permitindo que os dois campos se alinhem e que o torque seja nulo.

3.2.2 Baterias

Para a alimentação dos motores, dada sua faixa de tensão, foram testadas algumas pilhas alcalinas, mas não foi obtida corrente suficiente, optando-se então por baterias LiPo a qual fornece tensão de 7,4V e capacidade de descarga de 2200mAh, mostrada na figura 3.12.

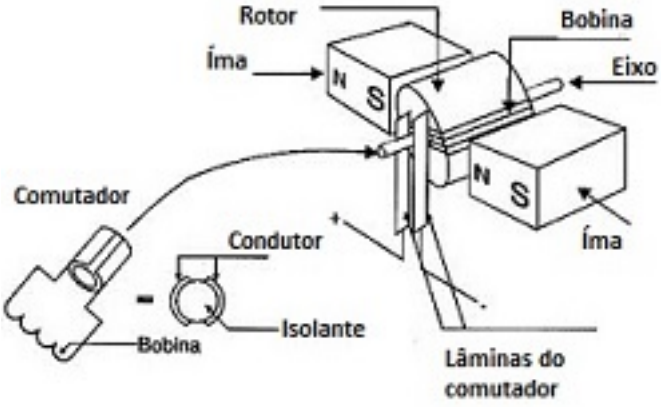

FIGURA 3.11: Esquema básico de um motor CC.

FIGURA 3.12: Bateria de LiPo utilizada

3.3 *Shield* Adafruit

O controle do sentido do protótipo pelos motores é realizado através do *Shield Motor Adafruit*, utilizado pela sua robustez, qualidade de montagem, funcionalidades obtidas e custo (ApêndiceB).

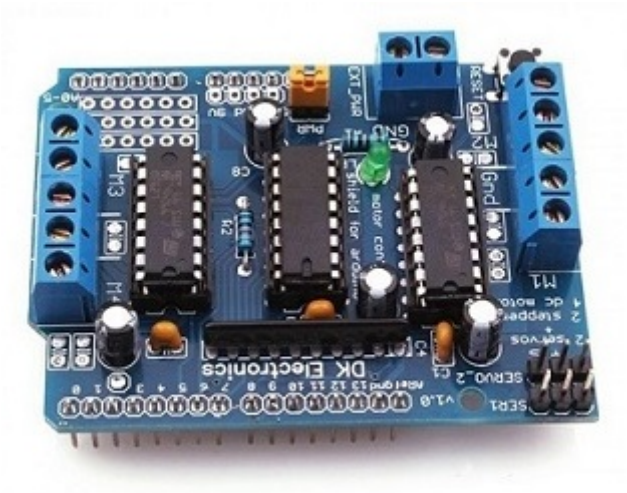

FIGURA 3.13: Motor *shield* L293D.

O *Motor Shield L293D*, como mostrado na figura 3.13 é uma placa composta por duas pontes H duplas, o que permite o controle de até 4 motores CC, 2 servos ou 2 motores de passo.

A tabela 3.6 mostra a especificação do componente de acordo com o vendedor[14]:

http://www.filipeflop.com/pd-6b643-arduino-motor-shield-l293d

Tabela 3.6: *Shield* L293D *Driver* Ponte H

	Especificação do *Shield* L293D *Driver* Ponte H
Tensão de Operação	4,5-36 V
Corrente de Saída	600 mA por canal
Corrente de Pico	1,2 A

3.3.1 Funções- *Shield* L293D

A biblioteca utilizada para controlar as funções do *shield* pode ser obtida em[15]:

https://github.com/adafruit/Adafruit-Motor-Shield-library/zipball/master

As principais funções utilizadas:

motor.setSpeed(velocidade): Define a velocidade de rotação do motor, podendo ser um valor entre 0 (motor parado) e 255 (rotação máxima);

motor.run(sentido): Aciona o motor no sentido definido sendo *FORWARD* (frente/horário), *BACKWARD* (sentido contrário/anti- horário), ou para o motor *(RELEASE).*

É necessário incluir a definição de quais portas os motores estão ligados.

3.3.2 Ponte H- L293D

A ponte H L293D é designada para prover correntes bidirecionais acima de 1 A para tensões de 4.5 V a 36 V. É uma ponte quadrupla, conforme indica seu sufixo D.

Da teoria de motores, para se controlar um motor CC, é necessário que uma corrente elétrica passe pelo mesmo no sentido em que se quer o giro, ou seja, uma inversão no sentido da corrente pode mudar o sentido do giro de um motor.

O circuito de condicionamento que promove essa aplicação é chamado de ponte H, pelo seu formato semelhante a letra H como mostra a figura 3.14. Seu diagrama de conexões pode ser visto na figura 3.15 e especificado na tabela 3.7, seu diagrama interno dado pela folha de dados presente em [16] e representado na figura 3.16

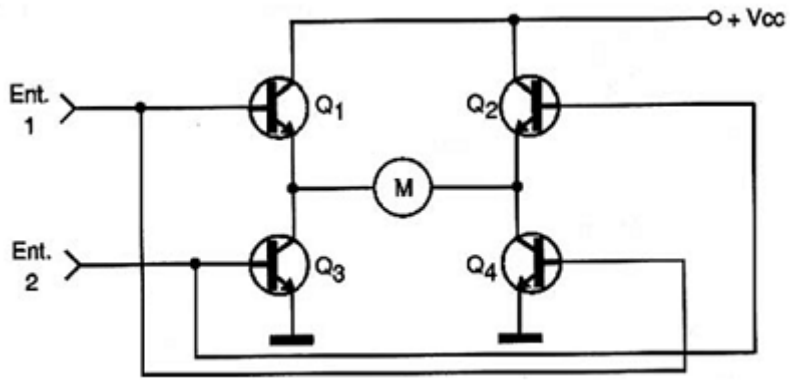

FIGURA 3.14: Ponte H.

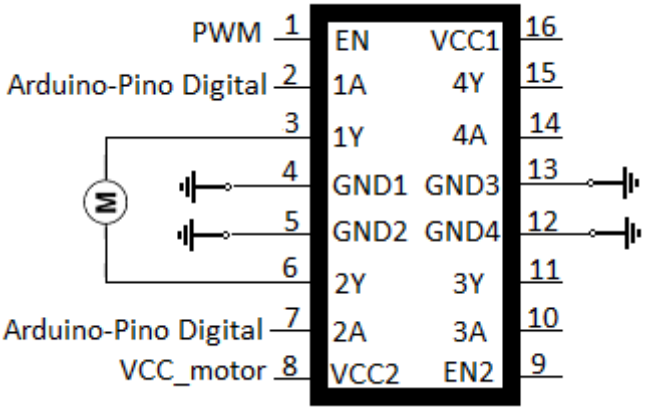

FIGURA 3.15: Diagrama de ligação- Ponte H.

TABELA 3.7: Especificação de valores máximos para operação da ponte H L293D

	Valores Máximos para Operação a 25 °C
Tensão de Entrada	7 V
VCC1	36 V
Corrente de Saída	600 mA
Corrente de Pico	1.2 A
Temperatura para Armazenamento	−65 °C a 150 °C
Dimensão	38 mm x 17 mm x 11.9 mm
Dissipação Contínua Total a 25 °C	2075 mW

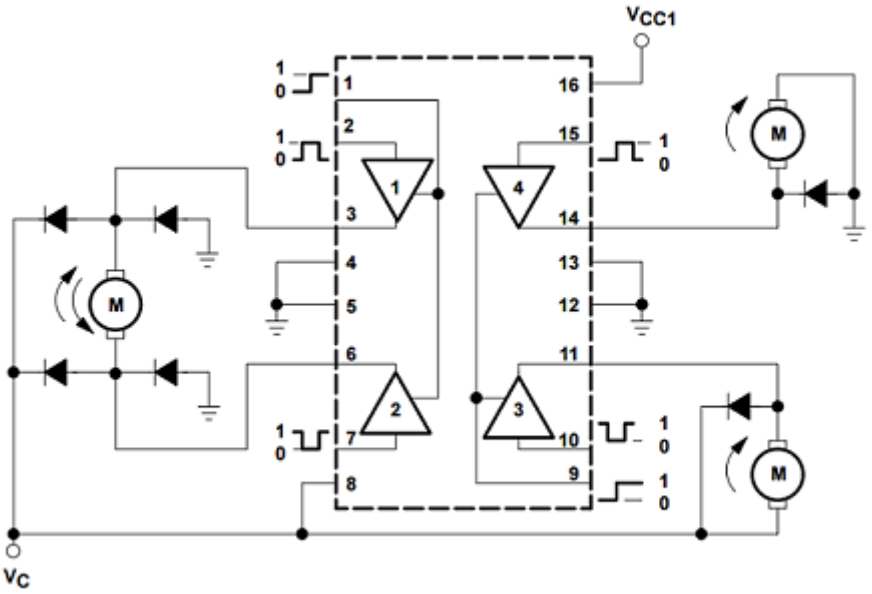

FIGURA 3.16: Diagrama interno- Ponte H L293D, retirado de sua folha de dados disponível em: http://www.ti.com/lit/ds/symlink/l293.pdf.

3.4 Módulo RF

3.4.1 Dados Gerais

O módulo RF 433 MHz, mostrado na figura 3.17, pode ser encontrado em diversos modelos. O modelo utilizado (de menor preço) não possui muitas informações disponíveis ou mesmo folhas de dados, segue com seu esquemático interno fornecido pelo vendedor [17] `http://www.electrodragon.com/`, e mostrado na figura 3.18 e tabela 3.9 como transmissor e na figura 3.19 e tabela 3.8 como receptor.

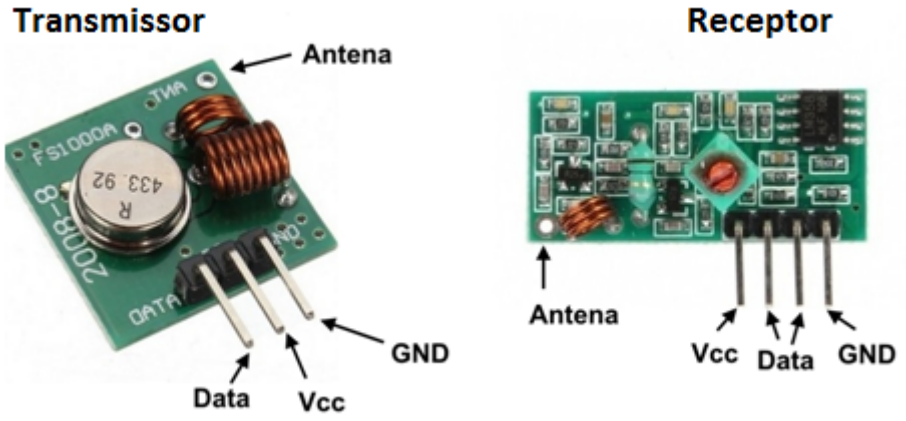

FIGURA 3.17: Módulo RF 433 MHz.

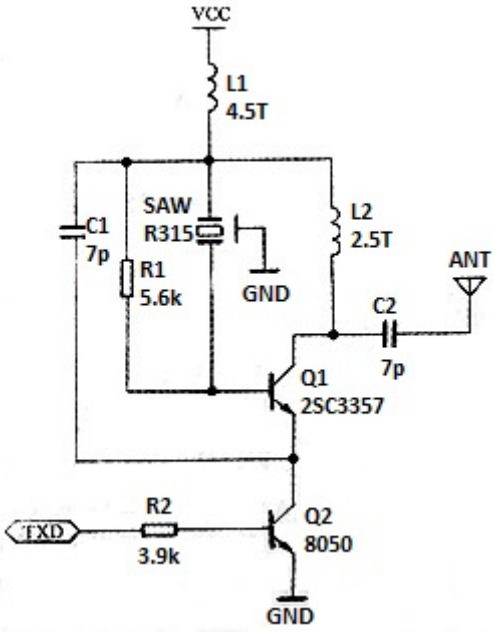

FIGURA 3.18: Transmissor- diagrama esquemático.

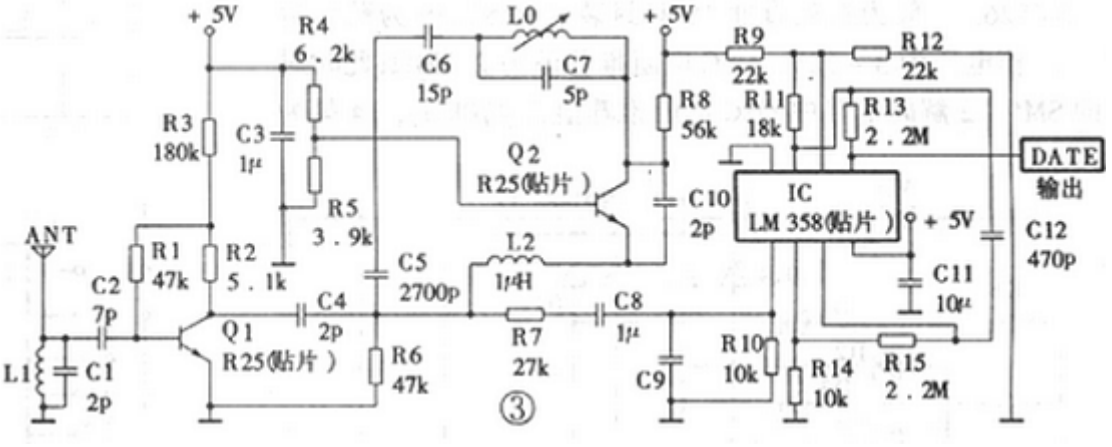

FIGURA 3.19: Receptor- diagrama esquemático.

TABELA 3.8: Especificação do receptor RF433MHz

	Especificação do receptor
Tensão de Operação	5 V
Corrente Estática	4 mA
Frequência de Recepção	433.92 MHZ
Dimensão	30 mm x 14mm x 7 mm
Antena Externa	Opcional

Tabela 3.9: Especificação do transmissor RF433MHz

	Especificação do transmissor
Tensão de Operação	3-12 V
Corrente de Operação	20-28 mA
Frequência de Operação	433.92 MHz
Temperatura de Operação	$-10\,^{\circ}\mathrm{C} \sim +70\,^{\circ}\mathrm{C}$
Distância de Transmissão	500 m
Tamanho	19mm x 8mm x 8 mm
Modo de Modulação	OOK
Potência de Saída	40 mW
Antena Externa	Opcional

3.4.2 Controle Remoto

O controle remoto baseado em [18] e mostrado na figura 3.20 é composto de um circuito transmissor e de um circuito receptor. O circuito transmissor é composto de 5 botões com 5 resistores *pull-up*.

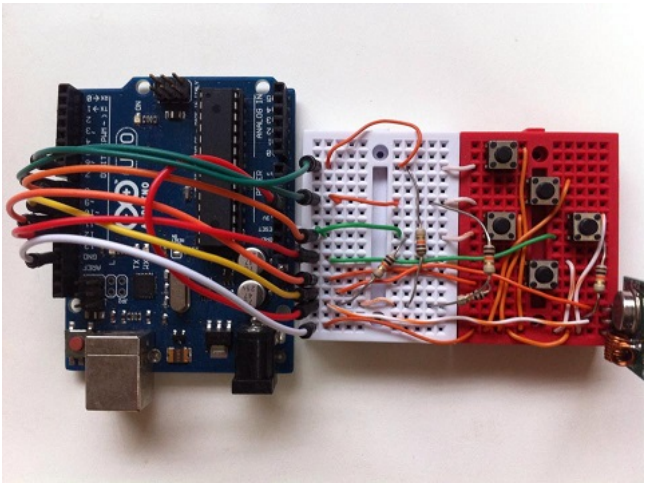

Figura 3.20: Controle remoto para o protótipo.

O pino do arduino que recebe o sinal (*Input*) sempre receberá +5 V caso o botão não seja apertado, ficando em alto. Quando o botão for pressionado, o caminho de menor impedância será em direção ao terra, por isso o pino será ligado ao terra e seu valor será igual a *GND*, ficando em estado baixo. Sem o resistor de *pull up*, mostrado na figura 3.21 com o valor de 10 kΩ entre os 5 V e o terra, haveria um curto-circuito, o qual danificaria a fonte de alimentação, o resistor impede esse processo pois funciona como um limitador de corrente.

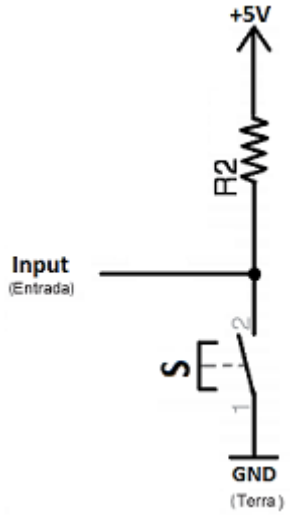

FIGURA 3.21: Conexão com resistor *pull up*.

3.4.3 Bibliotecas

Dado que a biblioteca mais popular `VirtualWire` utiliza o *Timer1* o qual também é utilizado pela biblioteca servo, dado que o servo motor, controlado por PWM, precisa de uma precisão maior do que a requerida pelo RF 433MHz, pois seu acionamento dissipa muita potência quando requerido posicionamento superior ao seu limite mecânico. Assim sendo, é necessário um estudo mais aprofundado e descritivo sobre a biblioteca utilizada nesse caso.

Comandos principais:

Transmissão:

```
RCSwitch nome = RCSwitch(); //Instância a biblioteca
```

Em *setup*:

```
nome.enableTransmit(10);
//Habilita transmissão com o pino digital 10
```

No laço *loop*:

```
nome.send(valor,24);
//Comando para enviar, caso esteja colocado ",
24"especifica número decimal, caso omitido manda-se binário.
```

Recepção:

```
RCSwitch nome = RCSwitch(); //Instância a biblioteca
```

Em *setup*:

```
nome.enableReceive(Numero da interrupção);
//Habilita a
recepção no pino correspondente ao número da interrupção externa
```

No laço *loop*:

```
if(nome.available()) //Se mensagem disponível
efetua comandos seguintes de recepção
nome.getReceivedValue(); //Número recebido,
pode se salvar em alguma variável, mostrar com print(), etc.
```

existem os comandos opcionais: `nome.getReceivedBitlenght();`

```
//o qual mostra o tamanho da informação recebida
nome.getReceivedProtocol();
//o qual mostra o protocólo utilizado na biblioteca.
```

Tais comandos não foram analisados nesse trabalho.

Para toda a finalização de leitura, se usa o comando antes do fim do *loop*:

```
nome.resetAvailable();
```

3.5 Sensor Ultrassônico

O sensor ultrassônico utilizado é constituído de duas cerâmicas piezoelétricas: uma atua como transmissora de vibração mecânica (*trigger*) e a outra como receptora dos ecos (*echo*), produzidos por obstáculos situados no caminho da onda transmitida, como mostra a figura 3.22.

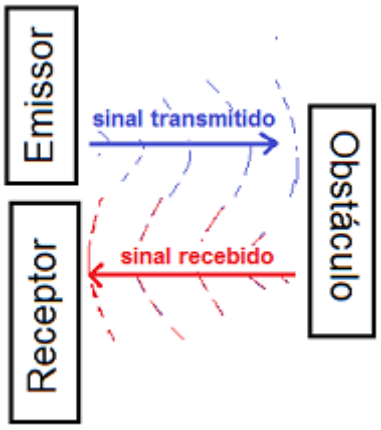

FIGURA 3.22: Sistema de detecção por ultrassom.

A distância percorrida pela onda emitida será:

$$D = V * \frac{T}{2} \tag{3.3}$$

Onde:

D é a distância do obstáculo ;

V é a velocidade do som no ar;

T é o tempo total, ou seja, o tempo de ida mais o tempo de volta (os quais são iguais).

A velocidade do som varia de acordo com a temperatura, para um ar ideal suposto com 0% de umidade, pode se definir que com a temperatura variando de 0 °C $\sim$ T °C, a velocidade é dada aproximadamente por:

$$V = (331,3 + 0,606 * T)m/s \tag{3.4}$$

Caso a umidade aumente, a velocidade também aumentará (de umidade 0% a 100% nota se uma variação de cerca de 1,5m/s a pressão e temperatura padrão, mas com o aumento da temperatura, seu efeito é aumentado como já descrito anteriormente, o dióxido de carbono causa diminuição na velocidade [19], mas não é constante dada a poluição e respiração. http://en.wikipedia.org/ acessada em 27/07 as 22horas.

Nesse trabalho será utilizado o sensor HC-S04, mostrado na figura 3.23, e os dados apresentados na tabela 3.10 baseiam se na folha de dados do fabricante (Cytron Technologies®).

FIGURA 3.23: Ultrassom utilizado.

O dispositivo possui quatro pinos, a saber: *VCC*, *Trigger*, *Echo*, *GND*, os quais são utilizados para alimentação, geração de pulsos, recepção dos pulsos e referencial terra.

Vale ressaltar que o som é uma onda longitudinal (i.e. se propaga em uma linha horizontal), caso o obstáculo não esteja perfeitamente na frente do sensor, os sons serão refletidos em outras direções que não o do receptor *echo*. Para o sensor em estudo HC SR04, o obstáculo não pode estar fora da região de 30 ° de operação, caso ocorra serão fornecidos dados incorretos. Sua faixa experimental é de 10 ° a 170 °.

Tabela 3.10: Especificação do HC SR04

	Especificação do Transmissor
Alimentação	+5 V
Corrente de Operação	15 mA
Ângulo Efetivo	15 °
Faixa de Distância	2 cm – 400 cm
Resolução	0,3 cm
Ângulo de Medição	30 °
Duração do Pulso de *Trigger*	10 µs
Dimensão	45 mm x 20 mm x 15 mm

3.6 Servo

O micro servo motor utilizado neste trabalho tem a função de movimentar o sensor ultrassônico para sua varredura de informação. Suas dimensões são mostradas na figura 3.24 e suas especificações na tabela 3.11, ambas fornecidas pelo fabricante micropik.

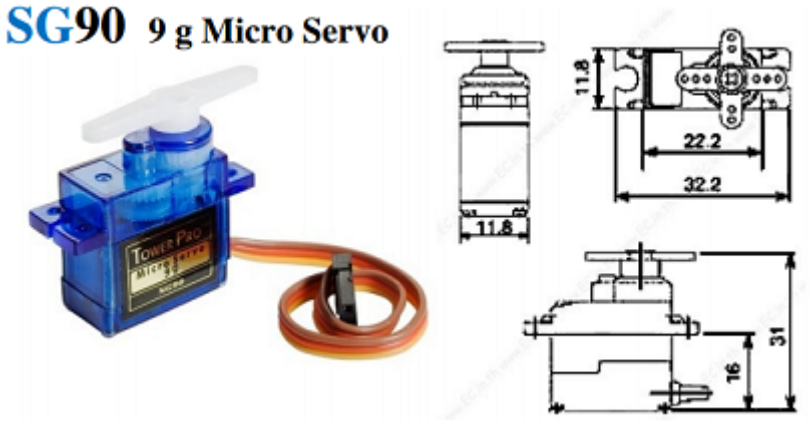

Figura 3.24: Micro servo motor.Fonte: datasheet, mikropik presente em [20], http://www.micropik.com/PDF/SG90Servo.pdf acessado em 29/09/2015

Tabela 3.11: Especificação do micro servo motor SG90

	Especificação do Micro Servo
Alimentação	+5 V
Torque	1,8 kgf·cm
Peso	9 g
Temperatura de Operação	0 °C a 55 °C
Velocidade de Operação	0,1 s/60 graus
Dimensão	22,2 mm x 11,8 mm x 31 mm

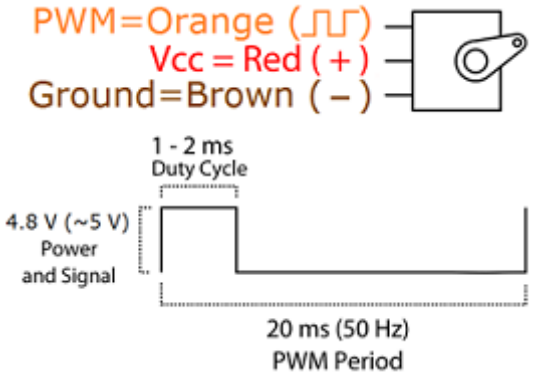

Figura 3.25: PWM - Micro servo motor.Fonte: datasheet, mikropik presente em [20], http://www.micropik.com/PDF/SG90Servo.pdf acessado em 29/09/2015

3.7 CMPS10

O módulo CMPS10, mostrado na figura 3.26 e com dados de dimensionamento mostrados na figura 3.27, é uma bússola com *tilt* compensado, possui um processador interno de 16 *bits*, o qual realiza todas as operações matemáticas necessárias para se obter saídas de 0 a 359.9 graus. Seu diagrama de conexões é mostrado na figura 3.28.

Figura 3.26: CMPS10.

Emprega um magnetômetro e um acelerômetro de 3 eixos, medindo assim com seus três sensores as componentes x, y e z do campo magnético. As medições de *pitch* e *roll*

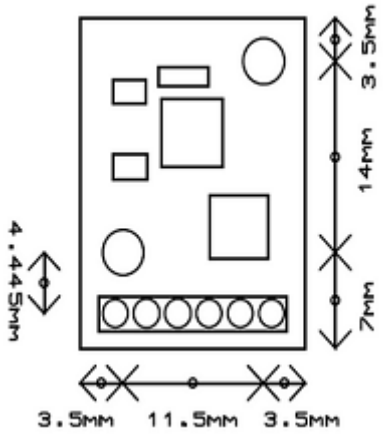

FIGURA 3.27: Dimensões do componente CMPS10 [22].Fonte:http://www.robot-electronics.co.uk/htm/cmps10doc.htm

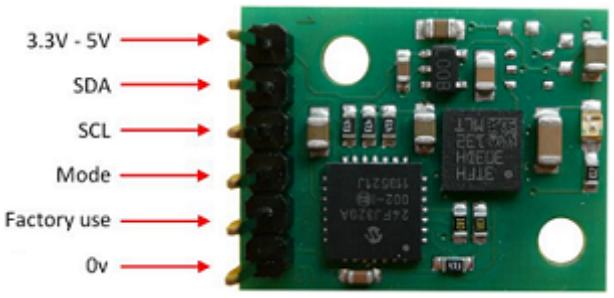

FIGURA 3.28: Diagrama de conexão- CMPS10.

são usadas para calcular o *bearing*, produzindo um resultado de 0-3599 o qual representa valores de 0 a 359.9 graus, que por sua vez indica a direção do deslocamento do protótipo.

Existem três modos de operação para se obter o *bearing* do módulo, a saber: Interface serial, interface I2C ou uma saída PWM.

Nesse Trabalho será usada a interface I2C, como mostrado na figura 3.28.

3.7.1 Modo I2C

Barramento I2C físico

São somente dois fios, chamados de SDA e SCL, o primeiro é o barramento de dados, e o segundo é a linha de *clock*, usada para sincronizar toda a transferência de

dados. Ambos são conectados a todos os dispositivos no mesmo barramento. Como mostra a figura 3.29.

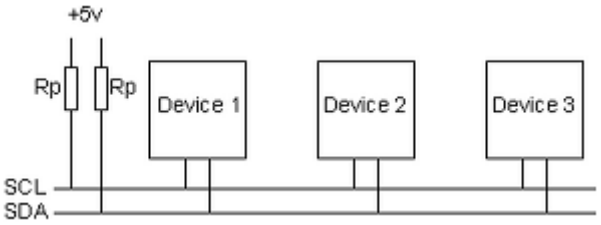

FIGURA 3.29: Barramento físico I2C [21].Fonte:http://www.robot-electronics.co.uk/i2c-tutorial

Mestres e escravos

Os dispositivos no barramento I2C são mestres e escravos, o mestre é sempre o dispositivo que deriva as linhas de *clock* (SCL), os escravos são sempre dispositivos que respondem ao mestre. Um escravo não pode inicializar uma transferência sobre o barramento I2C, somente o mestre pode, nesse trabalho, o mestre será o arduino e o escravo será o componente CMPS10. As transferências são sempre controladas pelo mestre, arduino no caso.

Protocolo físico I2C

Quando o mestre deseja conversar com o escravo, ele começa com uma sequência no barramento I2C. A sequência de início também chamada de *start sequence* é tão importante quanto a sequência de parada "*stop sequence*", são sequências especiais no sentido que são os únicos momentos que SDA pode mudar enquanto SCL está em alto. Enquanto dados estiverem sendo transferidos, SDA deve permanecer estável e não mudar enquanto SCL estiver em alto. A figura 3.30 ilustra o funcionamento dos comandos *start sequence* e *stop sequence*.

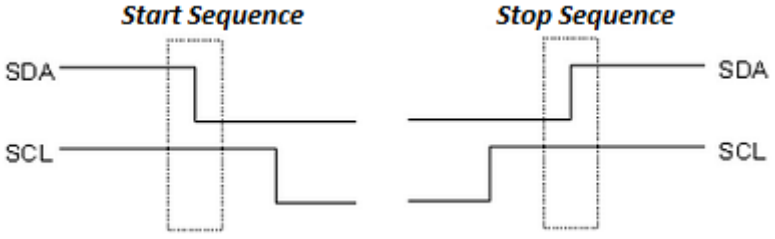

FIGURA 3.30: Ilustração: *start sequence* e *stop sequence*.Fonte:http://www.robot-electronics.co.uk/i2c-tutorial

Os dados são transferidos em sequências de 8 *bits*, que são colocados na linha SDA começando com o *bit* mais significativo (MSB - *Most Significant Bit*). O *clock* SCL é então pulsado alto e depois baixo. Para cada 8 *bits* transferidos, o receptor manda de volta um *bit* de confirmação, então existem 9 pulsos de *clock* para cada transferência de 8 *bits* de dados. Se o *bit* de confirmação for baixo, indica que o escravo recebeu os dados e está pronto para aceitar outro *byte*, se enviar um *bit* alto, isso significa que o escravo não pode aceitar outros dados e o mestre deve terminar a transferência mandando uma sequência de parada (*stop sequence*).

3.8 DHT11

De modo a formalizar os testes de operação, foi adotado o sensor DHT11 de temperatura e umidade, devido a seu baixo custo e relevante precisão, mostrado na figura 3.31, que usa um sensor capacitivo para umidade e um termistor NTC para medir a temperatura do ar circundante.

FIGURA 3.31: DHT11.

De acordo com a folha de dados do fabricante SUNROM Technologies®, é dada a tabela 3.12 com dados de resolução, faixa de operação e valores comuns de seus parâmetros.

TABELA 3.12: Especificação do DHT11

	Especificação do DHT11
Faixa de operação	20 a 90% Umidade e 0 °C a 50 °C Temperatura
Precisão para Umidade	5% UR
Precisão para Temperatura	2 °C
Numero de Pinos	4 pinos
Alimentação	5 V
Corrente em Operação	0,5 mA
Corrente em *Stand by*	100 µA
Dimensão	12 mm x 15.5 mm x 5.5 mm

3.9 *Buzzer*

O *Buzzer* Arduino, mostrado na figura 3.32, pode ser conectado à porta de I / O digital para a emissão de *beeps* sonoros. Quando a saída é baixa, *buzzer* reproduzirá um som longo. Se for ligado a uma modulação de largura de pulso analógico (PWM) de saída, o sinal sonoro pode produzir uma variedade de tonalidades.

FIGURA 3.32: *Buzzer*.

TABELA 3.13: Especificação do *Buzzer*

	Especificação do *Buzzer*
Corrente	$\leq$ 42 mA
Som de Saída	$\geq$85 DB
Frequência de Ressonância	2000 Hz a 2600 Hz
Temperatura de Operação	$-20\,^{\circ}C \sim +45\,^{\circ}C$
Temperatura de Armazenamento	$-20\,^{\circ}C \sim +60\,^{\circ}C$
Sinalizador Piezoelétrico	12 mm
Dimensão	33 mm x 14 mm

3.10 LCD

Existem no mercado diversos tipos de módulos LCD (*Liquid Crystal Display*), os quais são classificados essencialmente pelo número de colunas e linhas e tipo de luz de fundo, são utilizados como interface para observar alguma informação que acontece em um sistema eletrônico. Ressalta-se o fato de que para os módulos característicos, de acordo com a variação de colunas e linhas, também variam se o número dos seus pinos terminais, neste trabalho será utilizado um *display* característico de 16 colunas e 2 linhas (16x2), conforme mostrado na figura 3.33 e especificado na tabela 3.14.

FIGURA 3.33: *Display* de LCD (16x2).

TABELA 3.14: Pinos e descrição de funcionamento do LCD de 16x2 utilizado.

	Pinos	
1	Alimentação	GND
2	Alimentação	VCC
3	Tensão para Ajuste de Contraste	V0
4	Seleção de Dado(1) e Instrução(0)	RS
5	Seleção de Leitura (1) e Escrita(0)	R/W
6	Seleção de Habilitação, sendo Habilitado (1) Desabilitado (0)	E
7	Barramento de Dados-LSB	D0
8	Barramento de Dados	D1
9	Barramento de Dados	D2
10	Barramento de Dados	D3
11	Barramento de Dados	D4
12	Barramento de Dados	D5
13	Barramento de Dados	D6
14	Barramento de Dados-MSB	D7
15	Anodo	A
16	Catodo	K

Capítulo 4

Análise de Resultados

4.1 Componentes

4.1.1 Sensor Ultrassônico

O subsistema ultrassônico a ser testado é composto de um arduino UNO, dois sensores ultrassônicos, um *display* LCD e uma mini *protoboard*, mostrados na figura 4.1.

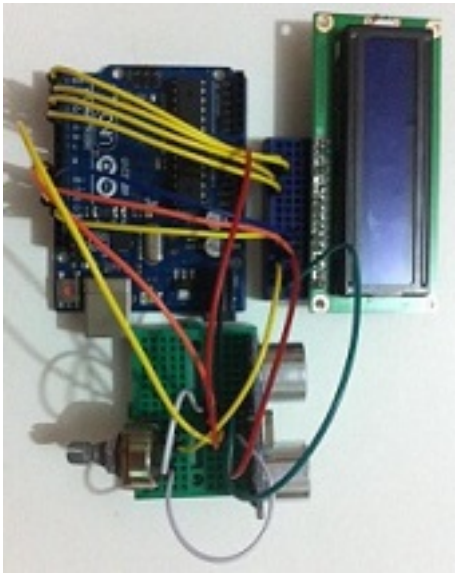

FIGURA 4.1: Sistema para a realização de testes para o sensor ultrassônico.

.

Testes para Tipos de Objetos

Para um ambiente fechado (com correntes de ar desprezíveis) e realizados com variação temporal de 3 horas, com variação de temperatura menor que 3 °C, foram realizados testes para objetos metálicos, emborrachados e não uniformes, como mostra a figura 4.2.

FIGURA 4.2: Aparatos.

Suas distâncias foram variadas com auxílio de régua e fita métrica em solo plano.

O valor de distância retornado pelo sensor é comparado com o valor medido com régua e fita métrica em todos os casos.

4.1.1.1 Objetos metálicos

Panela Grande

Testes de distância realizados com uma panela grande são indicados na tabela 4.1

Espelho

Testes de distância realizados com espelho são indicados na tabela 4.2

4.1.1.2 Objetos emborrachados e não uniforme

Chinelo

Tabela 4.1: Teste do sensor ultrassônico para operação a 29 °C

	Objeto metálico
Distância medida com régua e fita métrica	Distância retornada pelo sensor
0,1 cm	3300 cm
0,5 cm	12 cm
1 cm	9 cm
1,5 cm	4 cm
2 cm	2 cm
3 cm	3 cm
30 cm	30 cm
60 cm	60 cm
90 cm	90 cm

Tabela 4.2: Teste do sensor ultrassônico para operação a 28 °C

	Espelho
Distância medida com régua e fita métrica	Distância retornada pelo sensor
0,1 cm	15 cm
0,5 cm	5 cm
1 cm	4 cm
4 cm	4 cm
5 cm	5 cm
30 cm	30 cm
58 cm	58 cm
70 cm	70 cm
90 cm	90 cm

Testes de distância realizados com um chinelo são indicados na tabela 4.3

Tabela 4.3: Teste do sensor ultrassônico para operação a 29 °C

	Chinelo
Distância medida com régua e fita métrica	Distância retornada pelo sensor
0,1 cm	30 cm
2 cm	5 cm
4 cm	4 cm
5 cm	5 cm
30 cm	30 cm
58 cm	58 cm
70 cm	70 cm
78 cm	80 cm

4.1.1.3 Testes com Isopor

Testes de distância realizados com Isopor são indicados na tabela 4.4

Tabela 4.4: Teste do sensor ultrassônico para operação a 28 °C

	Isopor
Distância medida com régua e fita métrica	Distância retornada pelo sensor
30 cm	29 cm
31 cm	30 cm
60 cm	58 cm
70 cm	68 cm
80 cm	79 cm
90 cm	90 cm

4.1.1.4 Pedra não uniforme

Pedra

Para a pedra mostrada na figura 4.2, o desempenho foi satisfatório para operação do sistema, a face mais linear do objeto mais próxima ao sensor é medida primeiro como esperado. Sendo os valores mais precisos dentre todos os testes e obtidos a partir de dois centímetros.

4.1.1.5 Testes com corrente de ar

Sons são ondas mecânicas e utilizam o ar como meio de transporte, visando analisar a influência de variações de fluxo de ar na medida do sensor, foi repetido o teste anterior como objeto não uniforme (pedra) e com objeto metálico (panela).

Utilizando-se um ventilador da marca Mondial®, modelo NV 45 6P, em ambiente fechado a 27 °C, observou-se que a distância medida aumentava com relação a distância real, esse valor entretanto diminuiu de acordo com o tamanho dos pulsos emitidos pelo sensor ultrassônico; para precisão de 1 cm (usada pela biblioteca `Ultrassonic`) o valor a ser somado correspondia a 1 cm, para 0,5 cm tal valor caia ou mesmo 0,3 cm que é a resolução do sensor. Para o projeto em questão mostra-se que a influência de ventos para ambientes externos não é relevante. Excetuando-se ventos mais fortes do que os fornecidos pela máxima velocidade de rotação do ventilador utilizado.

4.1.2 DHT11

O sensor de temperatura e umidade DHT11 possui suas condições ideais de operação e suas limitações, como todo o componente utilizado neste trabalho. Assim, deve-se evitar que opere acima de suas especificações.

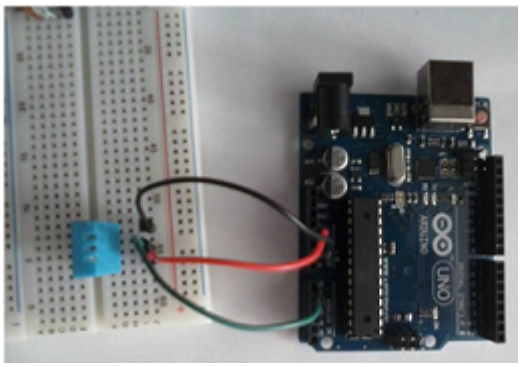

FIGURA 4.3: Sistema para a realização de testes para o sensor DHT11, vista superior.

Afim de testar seu funcionamento foram realizados testes em ambiente aberto, com seu esquemático mostrado nas figuras 4.3 e 4.4.

Seu código de teste encontra-se no Apêndice A. Os resultados obtidos são mostrados na figura 4.5.

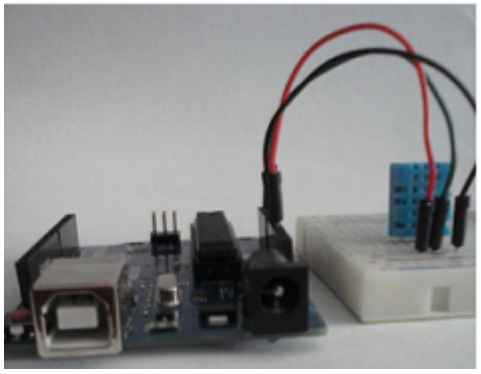

FIGURA 4.4: Sistema para a realização de testes para o sensor DHT11, vista frontal.

Apenas a nível de comparação de resultados, foram observados os valores fornecidos pelo sistema de medida dados por "climatempo"[23], para a região de Campinas, não é especificada a região exata que o aparato de medidas "climatempo"se encontra, e suas informações de temperatura e umidade são mostrados na figura 4.6. Apresentando quatro graus Celsius de diferença em temperatura e 13% em umidade, com relação ao sensor sob teste presente em Barão Geraldo (Bairro de Campinas).

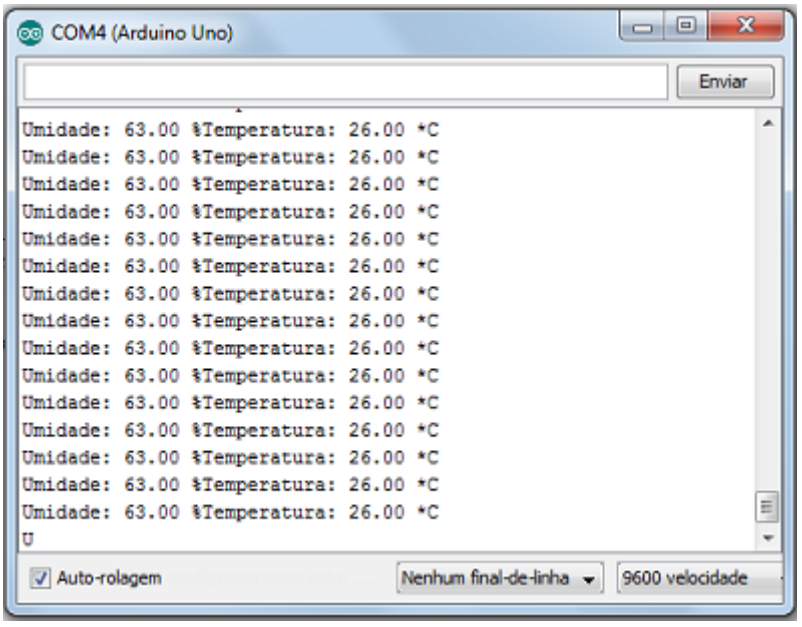

FIGURA 4.5: [Valores recebidos para teste do sensor DHT11.

FIGURA 4.6: Valores esperados para teste com sensor de temperatura DHT11.

4.1.3 CMPS10

Em primeira instância, o deslocamento relativo a um referencial previamente estabelecido será considerado para o mapeamento de rota no terreno; deste modo, será utilizada a funcionalidade de *bearing* da bússola digital CMPS10. Para seus testes, foi utilizado um aplicativo de bússola para celular chamado "Bússola"como referência, e o deslocamento angular obtido pelo CMPS10 foi medido com variação de 1 a 3 graus relativo aos valores do aplicativo.

O circuito montado pode ser visto na figura 4.7 e o aplicativo nas figuras 4.8 e 4.9.

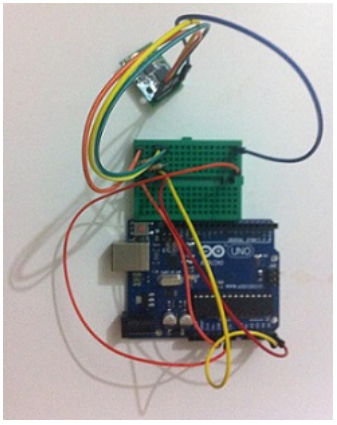

FIGURA 4.7: Sistema para a realização de testes para a bússola CMPS10.

FIGURA 4.8: Programa utilizado para comparação - utiliza GPS.

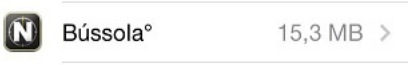

FIGURA 4.9: Símbolo e nome do programa utilizado para comparação.

4.2 Sistema

Os testes realizados para o sistema consistem da análise do controle remoto, da direção do desvio e a direção da correção. O controle remoto transmitiu as informações de comando em 433 MHz, sendo a interrupção externa do transmissor desativada no receptor

no modo automático. O sistema automático de desvio e correção foi testado integralmente, ou seja, sem a separação das atividades de desvio e correção e os testes são mostrados nas figuras 4.10 e 4.11. Ressalta-se que uma sequência foi adotada para os testes: O protótipo tem o deslocamento dado pelo controle manual até o aparecimento do obstáculo a frente, quando muda o controle para automático, seus motores são parados, calcula-se a distância disponível para trajeto a esquerda, calcula-se a distância disponível para trajeto a frente (de modo a verificar se o obstáculo ainda existe), calcula-se a distância disponível para trajeto a direita(caso exista obstáculo a direita e não exista obstáculo mais próximo a esquerda, o protótipo efetua o desvio à esquerda, caso contrário, seu desvio será a direita), verifica-se a extensão do obstáculo e caso tenha acabado inicia-se a correção e então a rota anterior ao último desvio é retornada; faz-se então uma busca no *buffer* de todas as rotas anteriores e são feitas correções das rotas anteriores até que a rota original seja alcançada e a pilha do *buffer* não apresente componentes, o sistema reinicia.

Informações gerais de desempenho estão presentes no Apêndice C.

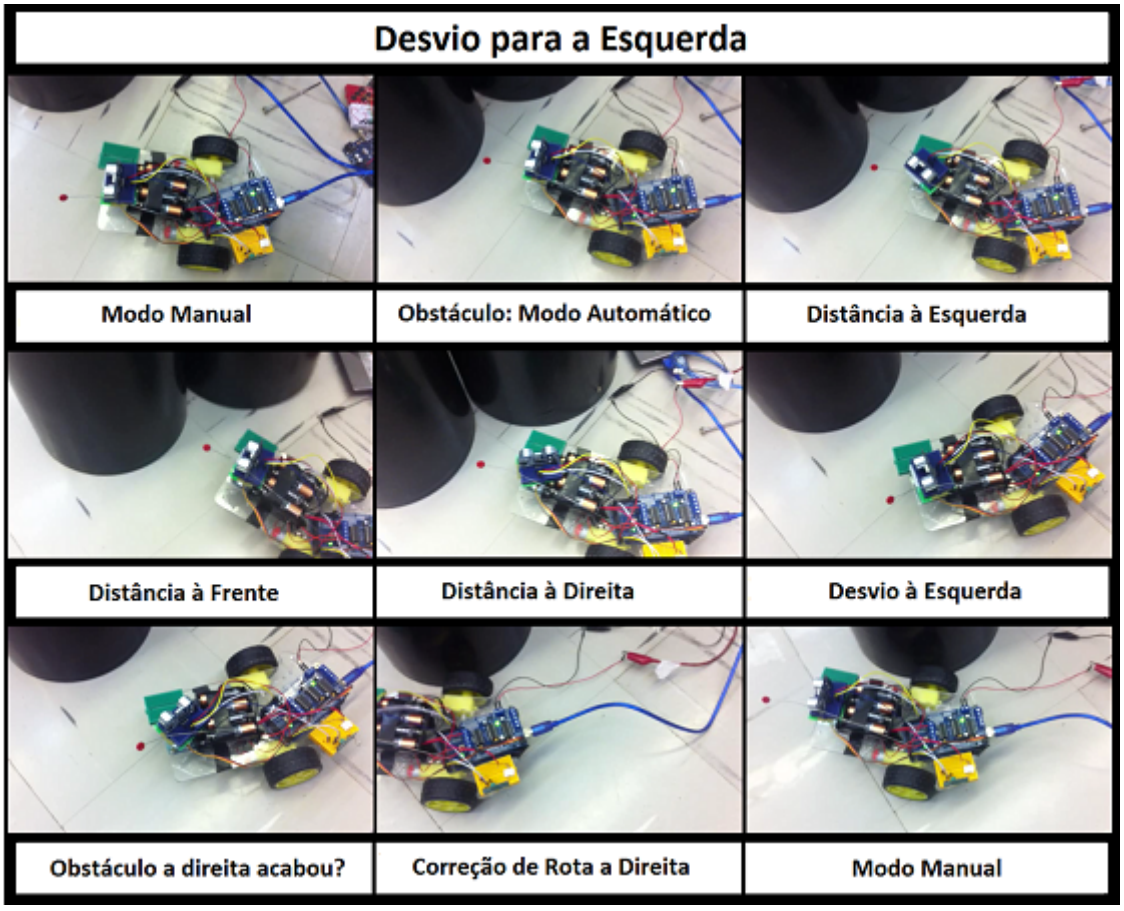

FIGURA 4.10: Teste de desvio à esquerda com correção de rota.

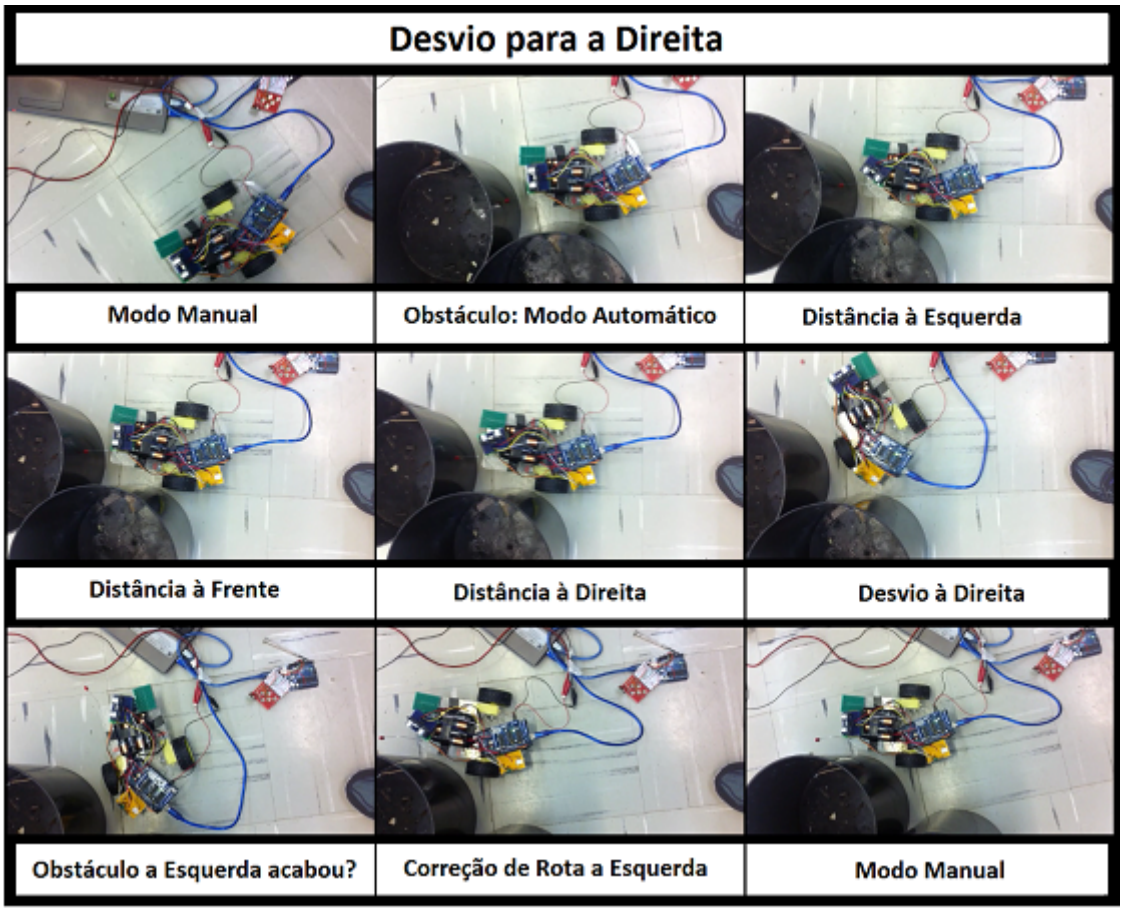

FIGURA 4.11: Teste de desvio à direita com correção de rota.

Capítulo 5

Conclusões e Trabalhos Futuros

5.1 Conclusões

O sistema de desvio de obstáculos estudado utilizou um sensor ultrassônico para percepção de obstáculos, o qual apresentou bons resultados nos testes realizados, mostrados na seção 4.1.1. O Chassi utilizado pode mover-se para a frente, para trás e girar para a direita e para a esquerda, com dois motores CC e uma roda "boba", ou seja, sem motor. A comunicação por RF foi bem sucedida, permitindo ao controle remoto operação simultânea com o servo motor, usando a biblioteca `RC Switch` com o `timer 2`. De um modo geral foram estudadas as condições de operação dos sensores, e interligação dos diversos componentes via programação arduino e a construção mecânica do protótipo. Seu custo final bem como de cada componente pode ser visto no Apêndice B e suas condições de operação no Apêndice C. Seus códigos estão presentes no Apêndice A. Algumas dificuldades surgiram como a demora de entrega das peças importadas, as quais levaram cerca de 2 meses para chegar ao Brasil, com tempo mínimo de 1 mês e máximo de 4 meses, e o mau funcionamento de algumas delas. O trabalho realizado permitiu a aplicação dos conhecimentos adquiridos durante o curso de Engenharia Elétrica em eletrônica analógica, controle, programação de microcontroladores e transdutores.

5.2 Trabalhos Futuros

Inúmeras aplicações podem ser obtidas para um sistema com a capacidade de desvio de obstáculos, visando a área de mobilidade urbana, monitoramento de plantações, competições robóticas, dentre outras.

Para aplicações em campo aberto uma das principais restrições do sistema em questão é a operação sob chuva; visando reduzir essa limitação, pode-se estudar novos componentes resistentes a chuva mantendo-se a mesma lógica adotada, e também construções de proteção mecânica para atingir, por exemplo, o objetivo de um sistema de desvio de obstáculos para monitoramento de plantações. Ainda nesse sistema, pode-se integrar o ramo já citado de IoT e através da *internet* mandar as informações monitoradas de temperatura, solo, umidade, resistividade do solo, ou quaisquer outras que possam ser obtidas com sensores apropriados, para um ponto de controle em computador.

Para melhorar o desempenho dos subsistemas que utilizam *timer* para seu funcionamento e resolução, pode-se explorar outros *timers* do microcontrolador utilizado, construindo, se necessário, novas bibliotecas para o servo motor ou controle RF.

Caso o interesse seja robótica, deve-se melhorar a robustez mecânica, controlar a velocidade de acordo com o objetivo desejado, estudar o consumo de energia do sistema, etc.

Para aplicações em mobilidade urbana pode-se, como já citado, tornar o sistema insensível a chuva e além disso usar IoT como um controle de informação do deslocamento do objeto móvel através de GPS, deve-se então aplicar uma eletrônica de potência de acordo com o motor utilizado, bem como realização de testes mais aprofundados.

Apêndice A

Códigos

A.1 Testes

A.1.1 HC SR04

```
        //Programa :  Medidor de distancia com HC-SR04
//
//O circuito:
// * LCD RS pino conectado ao pino 12
//* LCD Enable pino conectado ao pino 11
//* LCD D4 pino conectado ao pino 5
//* LCD D5 pino conectado ao pino 4
// * LCD D6 pino conectado ao pino 3
// * LCD D7 pino conectado ao pino 2
// * LCD R/W pino conectado ao pino GND
// Potenciometro de 10k conectado ao
// pino V0 para ajuste de contraste


        #include <Ultrasonic.h>  //Inclui a biblioteca Ultrasonic
#include <LiquidCrystal.h> //Inclui a biblioteca LCD
```

```
        //Define o pino do Arduino a ser utilizado com o pino Trigger do sensor
#define trigger 13


        //Define o pino do Arduino a ser utilizado com o pino Echo do sensor
#define echo 10


        //Inicializa o sensor ultrasonico
Ultrasonic ultrasonic(trigger, echo);


        //Define os pinos que serão ligados ao LCD
//e os inicializa na ordem:RS,E,D4-7
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);


        void setup()
{
Serial.begin(9600);

//Serial.begin(9600); // inicializa a comunicação serial
pinMode(trigger, OUTPUT); // define o pino trigger como saída.
pinMode(echo, INPUT); // define o pino echo como entrada.
lcd.begin(16,2); //Inicializa LCD de 16 colunas e 2 linhas
lcd.clear(); //Limpa o LCD
}

        // Loop pincipal do Arduino
void loop() {
float centimet;
//Serial.print( ultrasonic.Ranging(CM) );
//exibido no computador qual a medição do sensor.  Essa função faz todo os cálculos
e fornece as respostas
centimet=ultrasonic.Ranging(CM);
```

```
        //Apresenta os dados, em centímetros no LCD
lcd.setCursor(0,0);
lcd.print("Cent.:  ");
lcd.print();
lcd.setCursor(7,0);
lcd.print(centimet);
lcd.setCursor(0,1);
lcd.print("cm ");


        Serial.print("Cent:  ");
Serial.println(centimet);


        delay(1000);
}
```

A.1.2 DHT

O código seguinte é baseado na versão disponível em http://blog.filipeflop.com/sensores/ de filipe flop. E será utilizado para controle de temperatura.

```
# include "DHT.h"


        # define DHTPIN A1 // pino que estamos conectado
# define DHTTYPE DHT11 // DHT 11


        // Conecte pino 1 do sensor (esquerda) ao +5V
// Conecte pino 2 do sensor ao pino de dados definido em seu Arduino
// Conecte pino 4 do sensor ao GND
// Conecte o resistor de 10K entre pin 2 (dados)
// e ao pino 1 (VCC) do sensor
DHT dht(DHTPIN, DHTTYPE);
```

```
        void setup()
{
Serial.begin(9600);
Serial.println("Teste DHT");
dht.begin();
}


        void loop()
{
// A leitura da temperatura e umidade pode levar 250ms!
// O atraso do sensor pode chegar a 2 segundos.
float h = dht.readHumidity();
float t = dht.readTemperature();
// testa se retorno é valido, caso contrário algo está errado.
if (isnan(t) || isnan(h))
{
Serial.println("Falha na leitura");
}
else
{
Serial.print("Umidade:  ");
Serial.print(h);
Serial.print("% ");
Serial.print("Temperatura:  ");
Serial.print(t);
Serial.println("*C");
}
}
```

A.1.3 CMPS10

Código para mapeamento de terreno: /*

```
Autor.....:  Erik Bartmann
URL.......:  www.erik-bartmann.de
Version...:  1.0.1
Date......:  10.08.2014
Processing:  http://www.oreilly.de/catalog/processingger/
*/


 #include < Wire.h > // Biblioteca I2C
#define CMPS10Addr 0x60 // Endereço do CMPS10


 void setup()
{
Serial.begin(9600);
Wire.begin(); // Conecta I2C
}


 void loop()
{
byte byteHigh, byteLow; // byteHigh / byteLow para Bearing
char pitch, roll; // Pitch e Roll
int bearing; // Bearing


 Wire.beginTransmission(CMPS10Addr); // Comunicação com CMPS10
Wire.write(2); // Start Register (2) método de gravação envia o valor 2 para
o módulo, o que significa que queremos recuperar os dados de registo 2
Wire.endTransmission();//método endTransmission termina a transmissão iniciada.
Wire.requestFrom(CMPS10Addr , 4); // Requere 4 bytes
while (Wire.available() < 4); // Espera 4 bytes
byteHigh = Wire.read(); // High-Byte para cálculo do Bearing
```

```
byteLow = Wire.read(); // Low-Byte para cálculo do Bearing
pitch = Wire.read(); // Byte para Pitch
roll = Wire.read(); // Byte para Roll
bearing = ((byteHigh << 8) + byteLow) / 10; // Calculo do Bearing


        Serial.print(bearing);
Serial.println(";");
delay(2000);
}
```

A.2 Sistema Automático

```
        // inclusão de bibliotecas.
# include < Servo.h > // inclui biblioteca de manipulação de servos motores.
# include < Ultrasonic.h > // inclui biblioteca de manipulação do sensor ultrassônico.
# include < AFMotor.h > // inclui biblioteca de manipulação de motores DCs.



        //definindo os pinos
# define HC_SR04_TRIGGER A2 // Define o pino do Trigger do sensor ultrassônico
no pino ANALÓGICO A2
# define HC_SR04_ECHO A3 // Define o pino do Echo do sensor ultrassônico no
pino ANALÓGICO A3
# define BUZZER A0 // Define o pino do buzzer (Som) no pino ANALÓGICO A0
AF_DCMotor motor1(3); // Define o motor1 ligado ao M3
AF_DCMotor motor2(4); // Define o motor2 ligado ao M4


        Servo servoUltraSonico; // nomeando o servo motor
int distanciaCm = 0; //variável do valor da distância
int correcao = 0;
```

```
int manual=1;


        // executado na inicialização do Arduino
void setup()
{ Serial.begin(9600); // inicializa a comunicação serial para mostrar dados


        //configurações do servos motores
servoUltraSonico.attach(10); // Define o mini servo motor ligado no pino 10.
pinMode(HC_SR04_TRIGGER, OUTPUT); // Define o trigger do sensor para enviar
o sinal
pinMode(HC_SR04_ECHO, INPUT); // Define o Echo do sensor para receber o sinal
pinMode(BUZZER, OUTPUT); // Define o pino do buzzer como saída
motor1.setSpeed(190); // Define a velocidade para os motores 1.A velocidade
máxima é 255
motor2.setSpeed(190); // Define a velocidade para os motores 2.  A velocidade
máxima é 255
servoUltraSonico.write(90); // O servo do sensor se inicia a 90 graus (meio)
Parado;
}
// Função principal do Arduino
void loop()
{
if (manual == 1)
{
// recebeInputUsuario()
if (!andar()) //retorna 0 deu falha, automatico!
{
manual = 0;
}
}
else
{
desvioCorrigido();//mudar pra underline
```

```
}
}
// Função para chamar outras funções e definir o que o robô fará
int andar()  {
reposicionaServoSonar();
int distancia = lerSonar(); // Ler o sensor de distância
Serial.print("distancia:  "); // Exibe no serial
Serial.println(distancia);
if (distancia < 25)  { Parado(); return 0; //achaDirecao(); } else {
Frente();
manual=1;
return 1;
}
}
// Função para fazer o robô andar para frente
void Frente()
{
Serial.println("frente ");
motor1.run(FORWARD); // Roda vai para frente
motor2.run(FORWARD); // Roda vai para frente
delay(500);
}
// Função para ler e calcular a distância do sensor ultrassônico
int lerSonar() {
digitalWrite(HC_SR04_TRIGGER, LOW); // Desliga a emisão do som
delayMicroseconds(4); // Aguarda 4 segundos
digitalWrite(HC_SR04_TRIGGER, HIGH); // Liga a trasmisão de som
delayMicroseconds(20); // Continua emitindo o som durante 20 segundos
digitalWrite(HC_SR04_TRIGGER, LOW); // Desliga a emisão do som
delayMicroseconds(10); // Aguarda 10 segundos para poder receber o som
long pulse_us = pulseIn(HC_SR04_ECHO, HIGH); // Liga o recebedor e calcula quandos
pulsos ele recebeu
distanciaCm = pulse_us / 59; // Calcula a distaâcia em CM
delay(300);
```

```
return distanciaCm; // Retorna a distância
}
// Função para calcular a distância do centro
int calcularDistanciaCentro()
{
servoUltraSonico.write(90);
delay(200);
int leituraDoSonar = lerSonar(); // Ler sensor de distância
delay(600);
leituraDoSonar = lerSonar();
delay(600);
Serial.print("Distancia Centro:  "); // Exibe no serial
Serial.println(leituraDoSonar);
return leituraDoSonar; // Retorna a distância
}
// Função para calcular a distância da direita
int calcularDistanciaDireita()
{
servoUltraSonico.write(30);
delay(200);
int leituraDoSonar = lerSonar();
delay(600);
leituraDoSonar = lerSonar();
delay(600);
Serial.print("Distancia Direita:  ");
Serial.println(leituraDoSonar);
return leituraDoSonar;
}
// Função para calcular a distância da esquerda
int calcularDistanciaEsquerda()
{
servoUltraSonico.write(150);
delay(200);
int leituraDoSonar = lerSonar();
```

```
delay(600);
leituraDoSonar = lerSonar();
delay(600);
Serial.print("Distancia Esquerda:  ");
Serial.println(leituraDoSonar);
return leituraDoSonar;
}
// Função para pegar as distâncias lidas e calcular qual a melhor distancia
char calculaMelhorDistancia()
{
int esquerda = calcularDistanciaEsquerda();
int centro = calcularDistanciaCentro();
int direita = calcularDistanciaDireita();
reposicionaServoSonar();
int maiorDistancia = 0;
char melhorDistancia = '0';
if (centro > direita && centro > esquerda)
{
melhorDistancia = 'c';
maiorDistancia = centro;
} else if (direita > centro && direita > esquerda) {
melhorDistancia = 'd';
maiorDistancia = direita;
} else if (esquerda > centro && esquerda > direita){
melhorDistancia = 'e';
maiorDistancia = esquerda;
}
if (maiorDistancia <= 25) {
Re();
achaDirecao(); //permanece a espera caso obstaculo seja retirado
}
reposicionaServoSonar();
return melhorDistancia;
}
```

```
// Função que faz o robô virar à direita
void Direita()
{
Serial.println("Para a direita ");
motor1.run(FORWARD); // Roda vai para frente
motor2.run(BACKWARD); // Roda vai para trás
delay(100);
}
// Função que faz o robô virar à esquerda
void Esquerda()
{
Serial.println("Para a esquerda ");
motor1.run(BACKWARD); // Roda vai para trás
motor2.run(FORWARD); // Roda vai para frente
delay(100);
}
// Função para fazer o carro parar
void Parado()
{
Serial.println("Parar ");
motor1.run(RELEASE); // Motor para
motor2.run(RELEASE);
}
// Função que faz o robô andar para trás e emite som quando ele dá ré
void Re()
{
Serial.println("ré ");
tone(A0, 300, 300); // Configuração do tom do som
digitalWrite(BUZZER, HIGH); // Liga o som
delay(500); // Aguarda durante 250 milesecundos
digitalWrite(BUZZER, LOW); // Desliga o som
delay(50); // Aguarda durante 250 milesecundos
motor1.run(BACKWARD); // Roda vai para trás
motor2.run(BACKWARD); // Roda vai para trás
```

```
delay(500);
tone(A0, 300, 300);
digitalWrite(BUZZER, HIGH); // Liga o som
delay(500); // Aguarda durante 250 milesecundos
digitalWrite(BUZZER, LOW); // Desliga o som
delay(50); // Aguarda durante 250 milesecundos
Parado();
}
// Função para colocar o carrinho na melhor distância, isto é, girá-lo para
a melhor distância
int achaDirecao()
{
int dir = 0;
char melhorDist = calculaMelhorDistancia();
Serial.print("melhor Distancia:  ");
Serial.println(melhorDist);
if (melhorDist == 'c')
{
andar();
} else if (melhorDist == 'd') {
dir = dir + 1;
Direita();
} else if (melhorDist == 'e') {
dir = dir - 1;
Esquerda();
} else {
Re();
}
reposicionaServoSonar();
return dir;
}
// Função para deixar o sensor "olho"do robô no centro
void reposicionaServoSonar()
{
```

```
servoUltraSonico.write(90);
delay(200);
}
void decisao()
{
if (correcao < 0)
{
servoUltraSonico.write(30);
int distancia = lerSonar();
if (distancia > 30)
{
Direita();
correcao = correcao + 1;
}
else {
Frente();
desvioCorrigido();
}
}
if (correcao > 0) {
servoUltraSonico.write(150);
int distancia = lerSonar();
if (distancia > 30) {
Esquerda();
correcao = correcao - 1;
}
else {
Frente();
desvioCorrigido();
}
}
}
void desvioCorrigido ()
{
```

```
while (1)
{
if (correcao < -2 || correcao > 2)
{
while (1){
Parado();
}
}
if (!andar())
{
int deslocamento = achaDirecao();
correcao = correcao + deslocamento;
}
else
{
decisao();
if (correcao == 0)
{
manual=1;
break;
}
}
}
}
```

A.3 Sistema completo com controle do usuário e autonomia para desvio de obstáculo

```
        //***************************************
//inclusão das Bibliotecas
//***************************************
#include < AFMotor.h > // Inclui biblioteca de manipulação de motores DCs.
```

```
#include < RCSwitch.h > //Inclui biblioteca de manipulação de módulos RF
433 MHz.
#include < Servo.h > //Inclui biblioteca de manipulação de servos motores.
#include < Ultrasonic.h > //Inclui biblioteca de manipulação do sensor ultrassônico.


        #define HC_SR04_TRIGGER A8 // Define o pino do Trigger do sensor ultrassônico
no pino ANALÓGICO A8
#define HC_SR04_ECHO A9 // Define o pino do Echo do sensor ultrassônico no pino
ANALÓGICO A9


        //Direcoes
#define ANG_ESQUERDA 155
#define ANG_DIREITA 25
#define ANG_CENTRO 90
#define DISTANCIA_MINIMA 40


        Servo servoUltraSonico; // Nomeando o servo motor.
int distanciaCm = 0; //Variável do valor da distância.
int correcao = 0; //Variável do valor da correção (quando zero não existe correção
a ser feita).
int manual = 1; //Variável que indica o controle manual.
int ande = 10; //Variável que retorna o valor até o obstáculo.


        RCSwitch mySwitch = RCSwitch(); //Instância a biblioteca RCSwirch.
AF_DCMotor motor1(4); // Define o motor1 ligado ao M4
AF_DCMotor motor2(3); // Define o motor2 ligado ao M3


        void setup() {
Serial.begin(9600); //Inicia a serial.
mySwitch.enableReceive(3); // Recebe na interrupção 3, que corresponde ao pino
D20.
```

```
noInterrupts(); //Desabilita interrupções.


        pinMode(HC_SR04_TRIGGER, OUTPUT); // Define o trigger do sensor para
enviar o sinal.
pinMode(HC_SR04_ECHO, INPUT); // Define o Echo do sensor para receber o sinal.
//Buzzer:
pinMode(51, OUTPUT); //Pino do buzzer.


        //Velocidade dos motores
motor1.setSpeed(200); // Define a velocidade para o motor 1.A velocidade máxima
é 255.
motor2.setSpeed(210); // Define a velocidade para o motor 2.  A velocidade máxima
é 255.


        //configurações do servos motores
servoUltraSonico.attach(10); // Define o mini servo motor ligado no pino 10.
servoUltraSonico.write(90); // O servo do sensor se inicia a 90 graus (meio).
Parado; //Inicializa o protótipo parado.
}

        //**************************************
//FUNCOES MOTOR
//**************************************


        // Função para fazer o robô andar para frente
void Frente()
{
Serial.println("frente ");
motor1.run(FORWARD); // Roda vai para frente
motor2.run(FORWARD); // Roda vai para frente
delay(1000);
Parado();
```

```
}


        // Função para fazer o robô ficar parado
void Parado()
{
Serial.println("Parar ");
motor1.run(RELEASE); // Motor para
motor2.run(RELEASE);
}


        // Função que faz o robô andar para trás
void Re()
{
Serial.println("ré ");
motor1.run(BACKWARD); // Roda vai para trás
motor2.run(BACKWARD); // Roda vai para trás
delay(1000);
Parado();
}
        // Função que faz o robô virar à direita
void Direita()
{
Serial.println("VIREI A direita ");
motor1.run(FORWARD); // Roda vai para frente
motor2.run(RELEASE); // Roda vai para trás
delay(1000);
Parado();
}


        // Função que faz o robô virar à esquerda
void Esquerda()
{
```

```
Serial.println("VIREI A esquerda ");
motor1.run(RELEASE); // Roda vai para trás
motor2.run(FORWARD); // Roda vai para frente
delay(1000);
Parado();
}

        //**************************************
//Tratamento interrupcao
//**************************************

        void Tratamento() {

        if (mySwitch.available()) {

        int value = mySwitch.getReceivedValue();

        if (value == 0) {
Serial.print("Unknown encoding");
} else {
Serial.print("Received ");
Serial.print( mySwitch.getReceivedValue() );
Serial.print("/ ");

        if (value == 1) //Se o botão 1 é pressionado, i.e.  botão forward, frente
{
Frente();
Serial.println("= forward");
}
```

```
        if (value == 2) //Se o botão 2 é pressionado, i.e.  botão backward,
atrás
{
Re();
Serial.println("= backward");
}


        if (value == 3) //Se o botão 3 é pressionado, i.e.  botão para virar
a Esquerda, esquerda
{
Esquerda();
Serial.println("= left");
}


        if (value == 4) ////Se o botão 4 é pressionado, i.e.  botão para virar
a Direita, Direita
{ Direita();
Serial.println("= right");
}


        if (value == 5)//Se o botão 5 é pressionado, i.e.  botão stopped, parar
{ Parado();
Serial.println("= stopped");
}


        }


        mySwitch.resetAvailable();
}
}
```

```
        //**************************************
//Modos de operacao
//**************************************
void modoManual() { mySwitch.enableReceive(3); // Recebe na interrupção 3 que
é referente ao pino D20.
interrupts(); //Liga interrupções.
manual = 1; //Liga a flag de interrupções.
Serial.println("Modo Manual");
}
void modoAutomatico() {
manual = 0; //Desliga a flag de controle manual passando para automático.
Serial.println("Modo Automatico");
}


        //**************************************
//Função para direcionar o "olho"do robô
//**************************************


        void reposicionaServoSonar(int angulo) { if (angulo > 15 && angulo
< 160) {
servoUltraSonico.write(angulo); //Move o servo para o ângulo desejado.
delay(200);
}
}


        //**************************************
//Leitura do sonar a cada 300ms
//**************************************


        // Função para ler e calcular a distância do sensor ultrassônico
int lerSonar() {
digitalWrite(HC_SR04_TRIGGER, LOW); // Desliga a emisão do som
```

```
delayMicroseconds(4);
digitalWrite(HC_SR04_TRIGGER, HIGH); // Liga a trasmisão de som
delayMicroseconds(20);
digitalWrite(HC_SR04_TRIGGER, LOW); // Desliga a emisão do som
delayMicroseconds(10);
long pulse_us = pulseIn(HC_SR04_ECHO, HIGH); // Liga o recebedor e
calcula quandos pulsos ele recebeu
distanciaCm = pulse_us / 59; // Calcula a distancia em CM
delay(300);
return distanciaCm; // Retorna a distância
}


        //**************************************
//Função que aponta se existe obstáculos a frente
//**************************************


        // Função para chamar outras funções e definir o que o robô fará
int andar() {
reposicionaServoSonar(ANG_CENTRO);
int distancia = lerSonar(); // Ler sensor de distância
delay(600);
distancia = lerSonar();
delay(600);
Serial.print("distancia:  "); // Exibe no serial
Serial.println(distancia);
if (distancia < DISTANCIA_MINIMA)
{
Parado();
return 0;


        } else {
return 1;
```

```
}
}

        //**************************************
//Função de leitura do Sonar
//**************************************

        int calcularDistancia(int angulo) {
reposicionaServoSonar(angulo);
int leituraDoSonar = lerSonar();
delay(600);
leituraDoSonar = lerSonar();
delay(600);
Serial.print("Distancia ");
Serial.println(angulo);
Serial.println(leituraDoSonar);

        return leituraDoSonar;
}

        //**************************************
//Função que encontra a melhor distância a seguir para o desvio
//**************************************

        // Função para pegar as distâncias lidas e calcular qual a melhor distancia
char calculaMelhorDistancia() {
int esquerda = calcularDistancia(ANG_ESQUERDA);
int centro = calcularDistancia(ANG_CENTRO);
int direita = calcularDistancia(ANG_DIREITA);

        reposicionaServoSonar(ANG_CENTRO);
int maiorDistancia = 0;
```

```
char melhorDistancia = '0';


        //Compara melhor distância
if (centro > direita && centro > esquerda) {
melhorDistancia = 'c';
maiorDistancia = centro;
} else if (direita > centro && direita > esquerda) {
melhorDistancia = 'd';
maiorDistancia = direita;
} else if (esquerda > centro && esquerda > direita) {
melhorDistancia = 'e';
maiorDistancia = esquerda;
}
if (maiorDistancia <= DISTANCIA_MINIMA) {
melhorDistancia = 'r';
}
reposicionaServoSonar(ANG_CENTRO);
return melhorDistancia;
}


        //****************************************
//Acha melhor direção para desvio de obstáculo e desvia
//****************************************


        int achaDirecao() {
int dir = 0;
char melhorDist = calculaMelhorDistancia();
Serial.print("melhor Distancia:  ");
Serial.println(melhorDist); if (melhorDist == 'c') {
Frente();
delay(500);
} else if (melhorDist == 'd') {
```

```
dir = dir + 1;
Direita();
delay(500);
} else if (melhorDist == 'e') {
dir = dir - 1;
Esquerda();
delay(500);
} else if (melhorDist == 'r') {
Re();
delay(500);
}
reposicionaServoSonar(ANG_CENTRO);
return dir; }


        //**************************************
//decisao efetua correção de rota
//**************************************
void decisao()
{


        if (correcao < 0)
{
reposicionaServoSonar(ANG_DIREITA);
int distancia = lerSonar();
delay(600);
distancia = lerSonar();
delay(600);
if (distancia > DISTANCIA_MINIMA)
{
Direita();
delay(500);
correcao = correcao + 1;
```

```
Serial.println("CORRIGI PARA Direita ");


        }
else {
Serial.println("OBST GRANDE ");
Frente();
delay(200);
desvioCorrigido();
}
}


        if (correcao > 0) {
reposicionaServoSonar(ANG_ESQUERDA);
int distancia = lerSonar();
delay(600);
distancia = lerSonar();
delay(600);
if (distancia > DISTANCIA_MINIMA) {
Esquerda();
delay(500);
correcao = correcao - 1;
Serial.println("CORRIGI PARA Esquerda ");
} else { Serial.println("OBST GRANDE "); Frente();
delay(200);
desvioCorrigido();
}
}
}


        //**************************************
// desvioCorrigido:  Se tentativas de correção extrapoladas, se existe obstáculo
a frente para se guardar no buffer de correção ou se o caminho está livre e
```

```
pode-se tentar uma correção.
//**************************************


        void desvioCorrigido ()
{
while (!manual)
{
Serial.print("while not manual");


        if (correcao < -2 || correcao > 2)
{
while (1) {
Serial.println("Terreno tortuoso ");
Parado();
delay(500);
}
}
if (!andar())
{
Serial.println("MAIS UM OBST ");
int deslocamento = achaDirecao();
correcao = correcao + deslocamento;


        }
else {
decisao();
Serial.print("correcao = ");
Serial.print(correcao);
Serial.print("manual = ");
Serial.print(manual);
Serial.println("CORRIGINDO ");
if (correcao == 0)
```

```
{
Serial.println("CONSEGUIIIII ");
modoManual();
interrupts();
}
}
Serial.print("manual = ");
Serial.print(manual);
}
}

        //**************************************
//LOOP Principal

        //**************************************
//* Caso controle manual e não exista obstáculo a frente (ande=1), o controle
é dado ao usuário, caso o controle manual esteja ativado mas exista obstáculo
a frente, //o controle é retirado do usuário e o modo automático é ligado, caso
o modo automático esteja ligado, o programa irá desviar e corrigir.
//**************************************

        void loop() {

        if (manual == 1) {
ande = andar();
if (ande == 1) {

        interrupts();
Tratamento(); //ISR para receber comando.
}
else if (ande == 0) {
```

```
detachInterrupt(3); //Desliga interrupção 3.
delay(200);
tone(51,262,200); //DO
delay(200);
tone(51,262,200); //DO
delay(200);
modoAutomatico();
Serial.print("nao anda");
}
}
else if (manual == 0) {
detachInterrupt(3);
Serial.print("entra em corrigindo");
desvioCorrigido();
Serial.print("sai de corrigindo");
}


        }
```

Apêndice B

Custos

Nesse Apêndice serão mostradas duas cotações uma nacional com base no endereço eletrônico "mercadolivre"e outra internacional com base no endereço eletrônico "ebay". Para cotações nacionais a previsão de entrega é de até uma semana e para a internacional de 1 a 4 meses. O produto CMPS10 só foi encontrado fora do Brasil. Não é considerado o frete de cada produto.

Tabela B.1: Tabela de custos nacional do sistema estudado.

	Cotação Nacional
Produto	Preço
Chassi	R$ 93,88
Arduino UNO	R$ 37,89
Arduino Mega2560	R$ 72,00
Sensor Ultrassônico HC-SR04	R$ 10,80
Shield Adafruit	R$ 27,89
Bateria LiPo 7,4 V 2200 mAh	R$ 109,99
Carregador da Bateria LiPo	R$ 123,99
Bateria 9 V	R$ 7,43
Micro Servo Motor	R$ 15,57
RF 433MHz	R$ 12,00
Buzzer	R$ 4,99
CMPS10	-
DHT11	R$ 8,89
Mini *Protoboards*	R$ 29,99
Jumpers	R$ 16,90
Total	R$ 572,21

Tabela B.2: Tabela de custos internacional do sistema estudado.

	Cotação Internacional
Produto	Preço
Chassi	U$ 10,32
Arduino UNO	U$ 6,89
Arduino Mega2560	U$ 8,25
Sensor Ultrassônico HC-SR04	U$ 1,03
Shield Adafruit	U$ 2,64
Bateria LiPo 7,4 V 2200 mAh	U$ 13,85
Carregador da bateria LiPo	U$ 5,05
Bateria 9V	U$ 0,99
Micro Servo Motor	U$ 1,59
RF 433MHz	U$ 0,99
Buzzer	U$ 0,99
CMPS11	£ 23,99
DHT11	U$ 1,03
Mini *Protoboards*	5*U$ 1,91 (unidade)
Jumpers	U$ 0,99
Total	U$ 64,12 + £ 23,99

Apêndice C

Folha de dados

Tabela C.1: Tabela de especificações do sistema.

	Folha de dados do sistema
Característica	Especificação
Máxima Distância do Obstáculo	400 cm
Mínima Distância do Obstáculo	5 cm
Máxima Temperatura	40 °
Mínima Temperatura	0 °
Máxima Variação Angular para Obstáculo	30 °
Rotação Máxima do Motor	200 RPM
Máxima Umidade	80% UR
Máxima Altura de Desníveis do Solo	2,3 cm
Máximo Alcance RF	300 m (com Antena)

Referências Bibliográficas

[1] Cisco. "security: The vital element of the internet of things". *A Forrester Consulting Thought Leadership Paper Commissioned By Cisco*, pages 1–11, Março 2015. URL http://www.cisco.com/web/solutions/trends/iot/vital-element.pdf.

[2] Lopez Research. "an introduction to the internet of things (iot)". pages 1–6, Novembro 2013. URL http://www.cisco.com/web/solutions/trends/iot/introduction_to_IoT_november.pdf.

[3] Enciclopédia Livre. Sistema embarcado. *wikipedia*, Setembro 2015. URL https://pt.wikipedia.org/wiki/Sistema_embarcado.

[4] Jecel Mattos de Assupção Junior. "projeto de um sistema de desvio de obstáculos para robôs móveis baseado em computação reconfigurável". *USP-São Carlos*, Novembro 2009. URL http://www.teses.usp.br/teses/disponiveis/55/55134/tde-30032010-092432/pt-br.php.

[5] Flávio Garcia Pereira. "navegação e desvio de obstáculos usando um robô móvel dotado de sensor de varredura laser". *Universidade Federal do Espírito Santo, Centro Tecnologico*, pages 1–11, Junho 2006. URL http://portais4.ufes.br/posgrad/teses/tese_2359_DissertacaoMestradoFlavioGarciaPereira.pdf.

[6] Arduino UNO e Genuino UNO. https://www.arduino.cc/en/Main/ArduinoBoardUno, 2015. Acessado: 2014-07-28.

[7] Arduino Timers and Interrupts. https://arduino-info.wikispaces.com/Timers-Arduino, 2015. Acessado: 2015-09-15.

[8] Arduino Mega2560 e Genuino Mega2560. https://www.arduino.cc/en/Main/ArduinoBoardMega2560, 2015. Acessado: 2015-09-15.

[9] Arduino 101: Timers and Interrupts. http://letsmakerobots.com/node/28278, . Acessado: 2015-09-16.

[10] attachInterrupt(). https://www.arduino.cc/en/Reference/AttachInterrupt, 2015. Acessado: 2015-09-16.

[11] Charles Borges De Lima E Marco V. M. Villaça. *Avr E Arduino: Técnicas De Projeto*. Sistema de Bibliotecas Integradas do IFSC, 2012.

[12] Arduino MEGA 2560 fábio souza. http://www.embarcados.com.br/arduino-mega-2560/, 2015. Acessado: 2015-09-18.

[13] Atmel datasheet. http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf, 2015. Acessado: 2015-09-18.

[14] arduino motor shield - Filipe Flop. http://www.filipeflop.com/pd-6b643-arduino-motor-shield-l293d, .

[15] Adafruit Motor Shield library. https://github.com/adafruit/Adafruit-Motor-Shield-library/zipball/master, 2015. Acessado: 2015-09-18.

[16] Quadruple Half-H Drivers, L293 e L293D. pdf.datasheetcatalog.com/datasheet/texasinstruments/l293d.pdf, 2015. Acessado: 2015-09-18.

[17] RF433MHz. http://www.electrodragon.com/, 2015. Acessado: 2015-09-19.

[18] Arduino Básico, Michael McRoberts. http://www.fema.com.br/arduino/wp-content/uploads/2014/08/arduino.pdf, 2015. Acessado: 2015-09-16.

[19] Speed of Sound. http://en.wikipedia.org/wiki/Speed_of_sound, 2015. Acessado: 2015-07-27.

[20] Datasheet Servo Motor Micropik. http://www.micropik.com/PDF/SG90Servo.pdf, 2015. Acessado: 2015-09-29.

[21] I2C Tutorial. http://www.robot-electronics.co.uk/i2c-tutorial, 2015. Acessado: 2015-09-29.

[22] CMPS10. http://www.robot-electronics.co.uk/htm/cmps10doc.htm, 2015. Acessado: 2015-09-29.

[23] Campinas, Clima Tempo. http://www.climatempo.com.br/previsao-do-tempo/cidade/418/campinas-sp, 2015. Acessado: 2015-10-20.