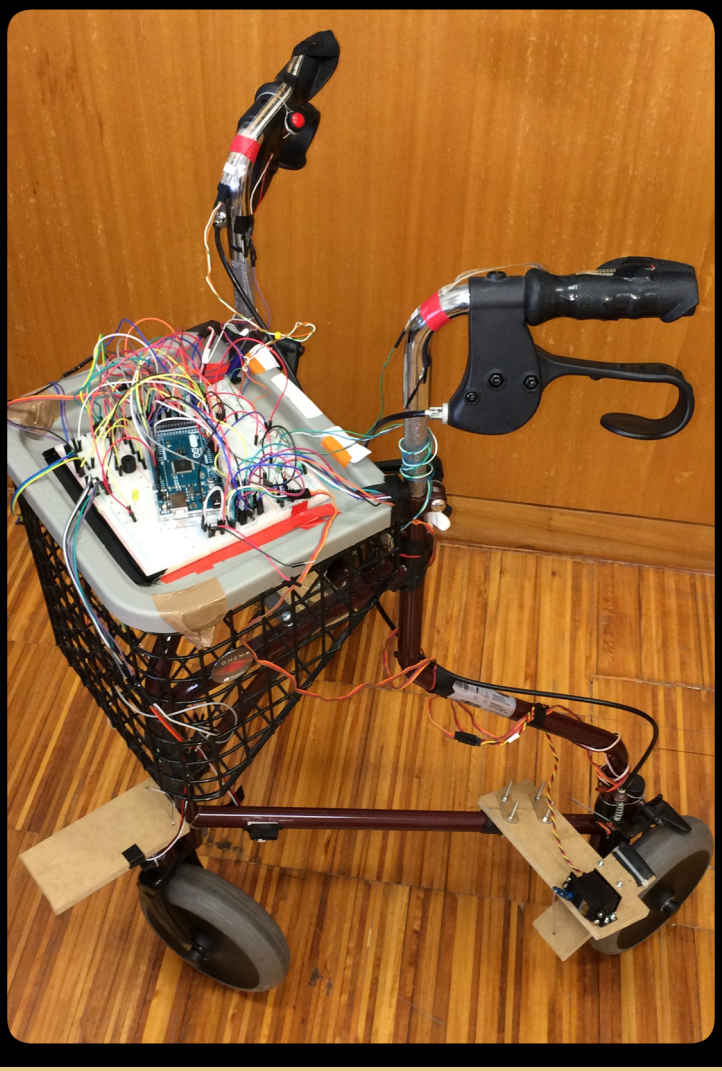
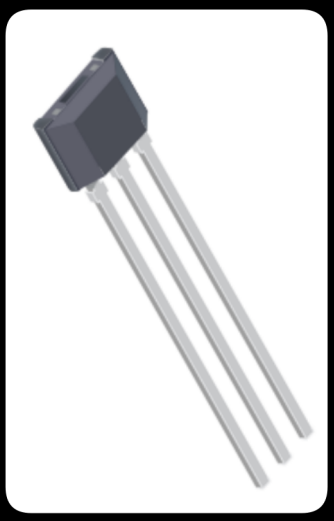
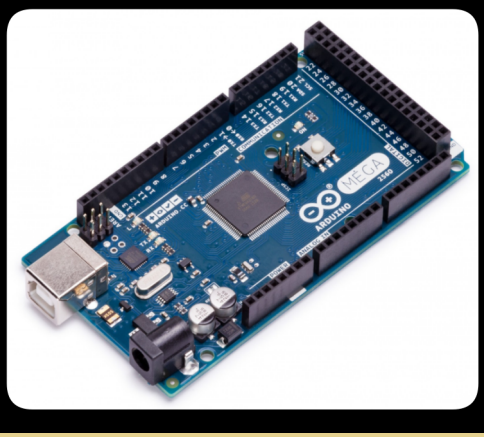
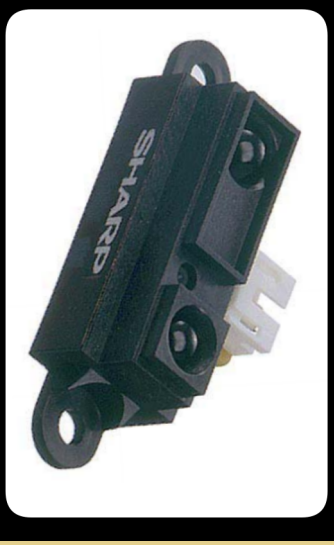








João Pedro Falhas Cavaleiro

Andarilho de reabilitação assistido eletronicamente

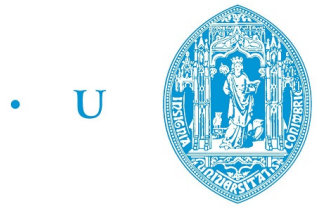
Dissertação submetida para a satisfação parcial dos requisitos do grau de Mestre em Engenharia Eletrotécnica e de Computadores

na área de especialização em Computadores

Setembro de 2017

· U C ·

Universidade de Coimbra



FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Andarilho de reabilitação assistido eletronicamente

João Pedro Falhas Cavaleiro

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador:	Doutor Manuel Marques Crisóstomo
Co-Orientador:	Doutor João Paulo Morais Ferreira

Júri

Presidente:	Doutor Mário João Simões Ferreira dos Santos
Orientador:	Doutor Manuel Marques Crisóstomo
Vogal:	Doutor Rui Alexandre de Matos Araújo

Setembro de 2017

You never fail until you stop trying.
- Albert Einstein

Success is not final, failure is not fatal: it is the courage to continue that counts.
- Winston S. Churchill

The dictionary is the only place that success comes before work. Work is the key to success, and hard work can help you accomplish anything.
- Vince Lombardi Jr.

Agradecimentos

O espaço limitado desta secção, seguramente, não me permite agradecer como devia a todas as pessoas, que ao longo do meu mestrado me apoiaram e ajudaram a cumprir os meus objetivos. Sendo assim, quero deixar-lhes algumas palavras para expressar a minha mais profunda gratidão.

Em primeiro lugar, expressar o meu profundo agradecimento à minha família, em especial aos meus pais, tia e irmão. Um enorme obrigado por estarem sempre presentes nos bons e maus momentos, e por acreditarem sempre em mim.

Quero agradecer ao Doutor António Paulo Mendes Breda Dias Coimbra (ISR, DEEC-UC), ao Doutor João Paulo Morais Ferreira (ISR, ISEC-IPC) e ao Doutor Manuel Marques Crisóstomo (ISR, DEEC-UC), por toda a orientação e apoio prestados que muito elevaram o meu grau de conhecimento e, sem dúvida, estimularam o meu desejo de querer, sempre, saber mais e a vontade constante de querer fazer melhor.

Aos meus amigos, pela vossa amizade, companheirismo e ajuda, que contribuíram, assim, para que cada dia fosse encarado com especial motivação.

À Fundação para a Ciência e Tecnologia e ao programa COMPETE 2020 pelo apoio financeiro ao projeto "Automatic Adaptation of a Humanoid Robot Gait to Different Floor-Robot Friction Coefficients" (PTDC/EEI-AUT/5141/2014).

Quero dedicar este trabalho à minha família e amigos, pois sem eles, a sua realização não seria possível.

Terminando assim esta etapa, espero poder de alguma forma retribuir e compensar todo o carinho, apoio e dedicação que, constantemente, me ofereceram.

A todos,

Muito Obrigado.

Abstract

As we observe human nature, we realize that the ability to walk and move on our own is one of the most important and practical factors of our daily life. Without it, our actions are immediately reduced and we need constant help.

Walkers are widely used devices serving as support for users, for a much safer locomotion due to their support base. If these devices are already good enough by themselves, if we equip them with several technologies for better service and support to their users, it would be even better.

Therefore, what is intended with this project is to equip a common walker with distance sensors to detect obstacles around it and the distance between the walker and the user, thereby creating an obstacle detection system; build an automatic braking system that can work together with the distance sensors, if an obstacle is detected the system automatically locks to ensure the safety of the user; equip wheels with Hall effect sensors to count rotations, set direction and measure speeds. It will also have a speed control system, if the user exceeds a certain speed limit (safety speed) the walker automatically locks up, so that the user does not run the risk of, for example, falling while walking in a inclined street. Finally, create a danger alarm system and several warning lights, to give a much greater audio-visual support to the user. All of these mechanisms specified above will be controlled by an Arduino so that the user feels more secure and comfortable when using this device.

Keywords

Electronic walker, Automatic brakes, Obstacle detector, Speed, Arduino

Resumo

Ao observarmos a natureza humana, apercebemo-nos de que a capacidade de locomoção e movimentação é um dos fatores mais importantes e práticos do nosso dia-a-dia. Sem ela, as nossas ações são imediatamente reduzidas e passamos a necessitar de ajuda constante.

Os andarilhos são aparelhos muito utilizados, servem de apoio aos utentes para uma locomoção muito mais segura devido à sua base de sustentação. Se estes aparelhos já são suficientemente bons por si só, se os equipássemos com diversas tecnologias para um melhor serviço e apoio a estes utentes, seria ainda melhor.

Sendo assim, o que se pretende com este projeto é equipar um andarilho comum com sensores de distância para detetar obstáculos à sua volta e a distância entre o andarilho e o utente, criando, deste modo, um sistema de deteção de obstáculos; construir e automatizar travões, para que desta forma se possa desenvolver um sistema de travagem automático, que funcione em conjunto com os sensores de distância, isto é, caso seja detetado um obstáculo o sistema trava automaticamente para garantir a segurança do utente; equipar as rodas com sensores de efeito de Hall para contar as rotações, definir o sentido e medir velocidades. Irá também, ter um sistema de controlo de velocidade, isto é, caso o utente ultrapasse um determinado limite de velocidade (velocidade de segurança) o andarilho trava automaticamente, para que assim o utente não corra o risco de, por exemplo, cair enquanto desce uma rua inclinada. Finalmente, criar um sistema de alarme de perigo e várias luzes de aviso, para dar um maior apoio audiovisual ao utente. Todos estes mecanismos especificados em cima, serão controlados por um Arduino para que o utente se sinta mais seguro e confortável ao utilizar este aparelho.

Palavras-Chave

Andarilho eletrónico, Travões automáticos, Detetor de obstáculos, Velocidade, Arduino

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

CSIC Consejo Superior de Investigaciones Científicas

UFES Universidade Federal de Espírito Santo

PCB Printed Circuit Board

IDE Integrated Development Environment

MHz Unidade de medida: Mega Hertz

USB Universal Serial Bus

V Unidade de medida: Volts

mA Unidade de medida: mili Ampère

DC Direct Current

I/O Input/Output

kB Unidade de medida: Quilobyte

SRAM Static Random Access Memory

EEPROM Electrically-Erasable Programmable Read-Only Memory

LED Light Emitting Diode

mm Unidade de medida: Milímetros

g Unidade de medida: gramas

kg.cm Unidade de medida: Quilogramas centímetro

ms Unidade de medida: milissegundos

BOP Operate point (Intensidade de campo magnético)

BRP Release point (Intensidade de campo magnético)

BHYS Hysteresis (Intensidade de campo magnético)

PSD Position Sensitive Detector

IRED Infrared Emitting Diode

cm Unidade de medida: Centímetros

dB Unidade de medida: decibel

Hz Unidade de medida: Hertz

ADC Analog-to-Digital Converter

rpm Unidade de medida: rotações por minuto

km/h Unidade de medida: Quilómetros por hora

lbs Unidade de medida: libras

ABS Anti-lock Braking System

GPS Global Positioning System

T Unidade de medida: Tesla

1

Introdução

No decorrer dos dias de hoje, um dos mecanismos mais importantes para que possamos ter a nossa própria autonomia é a capacidade de locomoção e mobilidade. São muitos os casos de perda desta capacidade importantíssima, seja por motivo de acidente ou na maior parte dos casos, devido aos longos anos vividos. Por isso, é extremamente importante dispor de um mecanismo de apoio e/ou reabilitação para que estas pessoas, portadoras de limitações motoras, possam viver de uma forma mais segura e confortável [1] [2].

Entre todos os mecanismos de reabilitação motora existentes, foi o andarilho o escolhido como objeto principal para a realização deste projeto. Devido à sua base de sustentação ampla, torna a marcha do utente muito mais segura e estável, permite um alívio da força que é exercida nos membros inferiores e transmite ao utente um maior equilíbrio ao movimentar-se. Existem no mercado vários tipos de andarilhos dos quais se irá falar mais afincadamente no capítulo 2, mas de uma forma geral, existem andarilhos fixos e móveis. Os andarilhos móveis possibilitam ao utente uma marcha ininterrupta, o que lhe garante uma maior independência e uma probabilidade de quedas reduzida. No caso dos andarilhos fixos, estes não possuem rodas pelo que exigem do utente força suficiente para o levantar a cada passo que efetua, contudo possui uma maior estabilidade quando se encontra apoiado no chão [2] [3] [4].

Neste projeto irá ser utilizado um andarilho com uma base móvel de três rodas, o que é mais indicado para casos de reabilitação, pois não precisa de ser levantado do solo para que o utente se possa deslocar.

Este mesmo andarilho já tinha sido adquirido e utilizado num projeto anterior, onde lhe foram aplicadas algumas soluções tecnológicas, como os sensores de força nos punhos, sensores de afastamento do corpo em relação ao andarilho e a comunicação de dados recolhidos por Bluetooth para uma aplicação de telemóvel [1] [5].

1.1 Objetivos

O objetivo principal deste projeto é modificar um andarilho convencional através da introdução de diversas tecnologias, para que este se torne mais fiável e seguro para a utilização do utente.

Em primeiro lugar, irá ser desenvolvido um sistema de travagem automático totalmente criado e construido do zero. O andarilho em si já possui um sistema de travagem manual, com alavancas nos punhos (em semelhança ao sistema usado pelas bicicletas convencionais) o qual só é acionado se o utente intervir, obrigando-o a ter força e rapidez suficiente caso queira ser bem sucedido. Portanto, para que o utente não tenha de acionar manualmente o travão, irá ser desenvolvido um sistema de travagem automático.

Em segundo lugar, irá ser desenvolvido um sistema de deteção de obstáculos, colo-

cando diversos sensores de distância à volta do andarilho, capazes de detetar obstáculos acionando imediatamente o sistema de travagem automático.

O sistema de alarme criado irá garantir que aquando da deteção de um obstáculo pelos sensores de distância, sejam ativados os alarmes sonoro e luminoso (luz LED) de forma a alertar o utilizador.

Será desenvolvido um sistema que permite saber a velocidade a que o utente se está a deslocar, assim como saber o sentido do movimento (andar para frente ou andar para trás).

Além disso, o andarilho também possui um sistema de medição da força, o qual já tinha sido implementado num projeto anterior, sendo o seu funcionamento explicado em anexo (apêndice).

1.2 Estrutura da dissertação

Esta dissertação encontra-se dividida em 6 capítulos. Após a introdução, o capítulo 2 é a revisão da literatura efetuada, isto é, refere como a locomoção e a marcha humana são importantes, dá-nos a conhecer o andarilho como equipamento de reabilitação e específica alguns dos projetos efetuados com andarilhos. No capítulo 3 encontram-se detalhados e especificados todos os materiais e dispositivos utilizados no projeto. O capítulo 4 aborda a construção e criação de todos os sistemas desenvolvidos e aplicados no andarilho, isto é, quer seja *hardware* quer seja *software*, encontra-se detalhadamente explicada a sua criação e desenvolvimento. É também, onde está explicado o modo de funcionamento, pois é onde se explica como todos os sistemas desenvolvidos interagem entre si. No capítulo 5 estão demonstrados todos os testes efetuados e respetivos resultados. Finalmente, o capítulo 6 apresenta as conclusões tiradas ao longo da dissertação e é onde estão as sugestões para trabalho futuro.

Ainda existe um apêndice no qual se pode encontrar todo o código escrito no projeto, bem como a explicação do funcionamento do sistema de medição da força.

2

Revisão da Literatura

Para que o produto seja inovador e traga benefício sob alguma forma, em todos os projetos, o autor deve ter conhecimento dos equipamentos já existentes. Para tal, será necessário realizar uma sondagem das soluções existentes.

Este capítulo sintetiza os pontos essenciais do que é um andarilho, quais são os seus objetivos principais e apresenta alguns tipos de andarilho de reabilitação tecnológicos já existentes.

Para que o Homem seja um ser autónomo necessita de uma característica essencial, a capacidade de locomoção.

Nos dias de hoje, equipamentos como cadeiras de rodas, andarilhos, próteses e até bengalas, ajudam a pessoa a contrariar disfunções da marcha e a restabelecer, de certa forma, a sua capacidade de locomoção. Dos equipamentos referidos, todos têm vantagens e desvantagens de acordo com a tipologia da disfunção do utilizador, mas falando em termos gerais, para utentes em reabilitação ou com fraca capacidade de movimentação, o dispositivo mais relevante para o seu auxílio é o andarilho [1].

2.1 Andarilho

O andarilho é um dispositivo de suporte de elevada fiabilidade que pode prevenir quedas e oferece segurança em caso de tonturas. Na maior parte dos casos, é feito em alumínio resistente e a sua altura é regulável, o que permite uma maior flexibilidade para a adaptação ao utente.

O andarilho presta apoio ao utente rodeando-o de 3 lados, fornecendo uma base de apoio mais ampla do que qualquer outro equipamento de apoio no mercado. Estes equipamentos podem também possuir rodas ou não, figura 2.1, e a sua escolha deverá ser sempre ponderada de acordo com as capacidades físicas e mentais do utente [1].

(a) Sem rodas. (b) 2 rodas frontais. (c) 3 rodas. (d) 4 rodas.

Figura 2.1: Exemplos de tipos de andarilho presentes no mercado.

A utilização destes equipamentos é feita por indivíduos que necessitam de ajuda na sua locomoção, como pessoas idosas. No entanto, a utilização do andarilho está maioritariamente associada a doenças neurológicas e doenças osteoarticulares, como osteoporose e alterações nas articulações metatarsofalângicas.

A hipótese de o utente cair em escadas/varandas e o andarilho ”escapar”por falta de aderência ao solo, são alguns exemplos de inconvenientes que os andarilhos convencionais apresentam. Os andarilhos com rodas apresentam algumas melhorias como sistemas de travagem manual, no entanto, alguns utentes podem não ter a força necessária para usar esses sistemas de forma eficiente [1] [3].

2.2 Andarilhos Inteligentes

Para combater os inconvenientes de um andarilho convencional, foi criado um novo grupo de andarilhos chamados de Andarilhos Inteligentes ou *Smart Walkers*. Na sua criação e produção são integrados sistemas e componentes avançados de tecnologias de robótica, eletrónica e mecânica. São muitas as empresas e entidades que procuram melhorar a segurança e a fiabilidade destes equipamentos, e já existem, de facto, diversos mecanismos melhorados no mercado.

2.2.1 Andarilho de Einbinder

Einbinder, em 2010 [6], apresenta um modelo de andarilho que consiste na modificação de um andarilho convencional com rodas, onde é integrado um sistema de travagem eletrónico acionado através da ação de um botão de pressão. Ou seja, o simples facto de o utente carregar no botão, faria com que as rodas do andarilho travassem, sendo o seu principal objetivo prevenir quedas [1].

2.2.2 Andarilho SIMBIOSIS

O grupo CSIC - Espanha, em 2010, desenvolveu o andarilho SIMBIOSIS [7], representado na figura 2.2, que utiliza diversos sensores para detetar e identificar o movimento do utente. Com a aplicação de sensores de ultra-sons, não só consegue detetar o movimento do pé do utente como também consegue prever as intenções do movimento que são detetadas por sensores de força colocados nos apoios do ante-braço do andarilho [1].

Figura 2.2: Andarilho SIMBIOSIS do grupo CSIC.

2.2.3 Andarilho Mobil

O andarilho multifuncional Mobil, figura 2.3, é um equipamento que foi pensado para oferecer um maior apoio ao utente, através de plataformas e rodas traseiras motorizadas. Para além destas modificações, o Mobil conta com controlo remoto e pode seguir o utente, se este usar um cinto ativo que envia sinais ultra-som para o andarilho. Consegue também, evitar obstáculos através do uso de ultra-sons [1] [3].

Figura 2.3: Andarilho Mobil.

2.2.4 UFES smart walker

Na Universidade Federal de Espírito Santo no Brasil, foi desenvolvido o protótipo UFES smart walker, figura 2.4, que tem como objetivo principal entender os movimentos feitos pelo utente, para desta maneira auxiliar as pessoas com dificuldades de locomoção a poderem andar com mais confiança e independência. Para isso, utiliza um sensor laser e sensores de força nos apoios dos ante-braços [1] [7] [8].

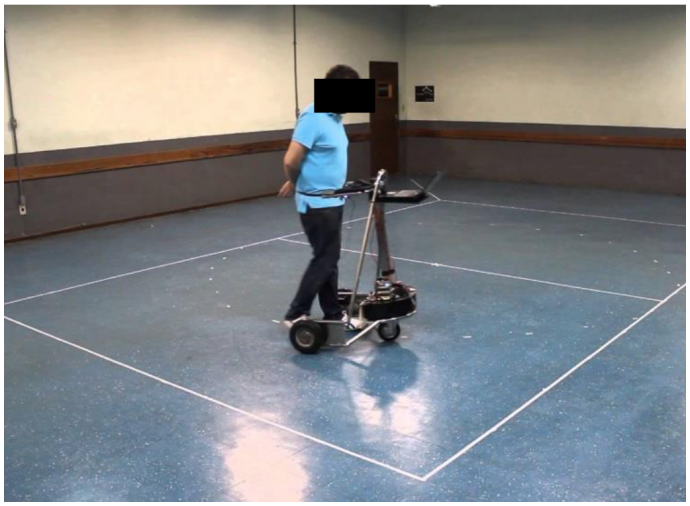


Figura 2.4: Protótipo do andarilho UFES smart walker.

3

Componentes Utilizados

Todos os materiais/componentes utilizados na conceção deste projeto, vão ser apresentados e descritos neste capítulo.

3.1 Andarilho Utilizado

O andarilho utilizado neste projeto foi um andarilho de três rodas, duas traseiras e uma frontal, como indicado na figura 3.1.

Figura 3.1: Andarilho utilizado no projeto.

Ao longo da realização deste trabalho, foram adicionados vários componentes a esse andarilho convencional de forma a torná-lo mais seguro e fiável, para que o utilizador se possa sentir mais confiante aquando da sua utilização.

A lista dos componentes incorporados é a seguinte:

- Arduino
- Servo motores para a travagem
- Sensores de efeito Hall
- Sensores de distância
- Buzzer, LEDs e Botões
- Materiais do sistema de travagem

De seguida, será apresentada uma descrição detalhada das especificações de cada componente implementado.

3.2 Arduino

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

A plataforma é composta essencialmente por duas partes: o *hardware* e o *software*. Consiste numa placa PCB de programação, baseada numa simples placa microcontroladora e um ambiente de desenvolvimento (IDE), para assim, ser possível o envio do código para placa.

Uma das vantagens do Arduino é justamente a facilidade de programação. O IDE do Arduino é gratuito e simples, contendo um editor de texto, um compilador e um monitor série.

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

Falando em termos práticos, o Arduino é um pequeno computador que se pode programar para processar entradas e saídas entre o dispositivo e os componentes externos ligados a ele, interagindo com o ambiente por meio de *hardware* e *software*. Para programar o Arduino utilizamos o seu IDE, que é um *software* onde podemos escrever um código numa linguagem semelhante a C/C++ [9].

Existem vários tipos de Arduino. De acordo com a dimensão ou a potência necessária para o projeto opta-se pelo que é mais favorável. Neste caso o Arduino utilizado começou por ser o UNO, mas como o projeto atingiu dimensões consideráveis (necessidade de mais pinos de entrada e de mais opções de ligações) no final foi o Arduino Mega 2560 o escolhido.

3.2.1 Arduino Mega 2560

O Arduino Mega 2560, figura 3.2, é uma placa microcontroladora baseada no ATmega2560. Possui 54 pinos de entrada/saída digitais, 16 entradas analógicas, um oscilador de cristal de quartzo (*clock*) de 16 MHz, uma conexão USB, um botão de reinicialização, entre outros [10].

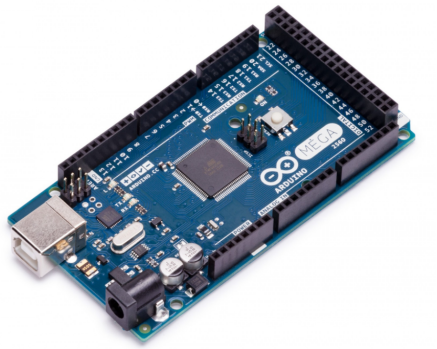


Figura 3.2: Arduino Mega 2560

Na tabela seguinte são apresentadas as especificações técnicas da placa Mega 2560:

Microcontrolador	ATmega2560
Tensão operacional	5 V
Tensão de entrada (recomendada)	7-12 V
Tensão de entrada (limite)	6-20 V
Pinos de I/O digitais	54
Pinos de entrada analógicos	16
Corrente DC por pino I/O	20 mA
Corrente DC para Pino 3.3 V	50 mA
Memória flash	256 kB dos quais 8 kB utilizados pelo bootloader
SRAM	8 kB
EEPROM	4 kB
Velocidade do relógio	16 MHz
LED BUILT IN	13
Comprimento	101,52 mm
Largura	53,3 mm
Peso	37 g

Tabela 3.1: Tabela de especificações do Arduino Mega 2560.

3.3 Servo Motores

O servo motor é um servo-mecanismo que usa o *feedback* que lhe é enviado (a partir do IDE do Arduino) com a posição, para controlar dessa forma o braço do servo motor. Internamente, um servo motor combina um motor com um circuito de realimentação, um controlador e outros circuitos complementares.

Sendo mais preciso na definição, um servo motor é um atuador rotativo ou linear que garante o controlo, velocidade e precisão em aplicações de controlo de posição em malha fechada [11].

Neste projeto, usaram-se dois servo motores Hitec HSR-5995TG - Ultra Torque, figura 3.3, nas rodas traseiras do andarilho para fazer a atuação dos travões automáticos.

Figura 3.3: Servo motor Hitec HSR-5995TG - Ultra Torque

Na tabela seguinte encontram-se as especificações [12] do servo motor Hitec HSR-5995TG - Ultra Torque:

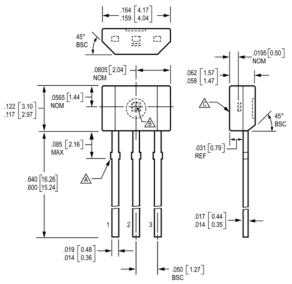
Modulação	Digital
Torque	6.0 V: 24 kg.cm 7.4 V: 30 kg.cm
Velocidade Angular	6.0 V: 0.15 seg/60° 7.4 V: 0.12 seg/60°
Peso	61,5 g
Dimensões	Comprimento: 39.9 mm Largura: 20.1 mm Altura: 38.1 mm
Tipo de motor	Coreless
Tipo de engrenagem	Titânio
Rotação/Suporte	Rolamentos duplos
Intervalo de rotação	180°
Ciclo de pulso	20 ms
Comprimento do pulso	900-2100 μs

Tabela 3.2: Tabela de especificações do servo motor Hitec HSR-5995TG - Ultra Torque.

3.4 Sensor de efeito de Hall

Foram colocados quatro sensores de efeito de Hall Allegro A1101, figura 3.4a, dois em cada roda traseira, que juntamente com doze ímanes que rodam com a respetiva roda onde estão fixados, funcionam como um encoder rotativo para detetar a rotação das rodas, sentido e velocidade.

(a) Sensor de efeito Hall Allegro A1101 vista externa.



(b) Sensor de efeito Hall Allegro A1101 dimensões.

Figura 3.4: Sensor de efeito Hall Allegro A1101.

A saída destes dispositivos muda para *low* (deteta) quando um campo magnético (polaridade sul) perpendicular ao sensor de Hall excede o limite do ponto de operação, BOP. Após a ativação, a saída é capaz ir a 25 mA e a tensão de saída é *VOUT*. Quando o campo magnético é reduzido abaixo do ponto de libertação, BRP, a saída do dispositivo fica *high* (não deteta). A diferença entre os pontos de operação e libertação magnética é a histerese, BHYS, do dispositivo [13].

Característica (Ponto)	Modelo	Min	Max	Unidades
Ponto de Operação - BOP	A1101	0.005	0.016	T
Ponto de Libertação - BRP	A1101	0.001	0.013	T
Histerese - BHYS	A1101	0.002	0.008	T

Tabela 3.3: Tabela de pontos de transição do sensor de efeito de Hall Allegro A1101.

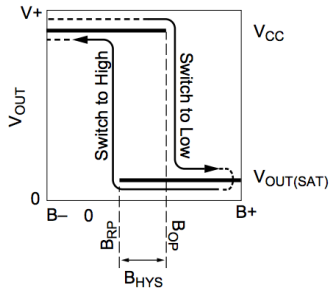


Figura 3.5: Diagrama de funcionamento do sensor de efeito Hall Allegro A1101.

Conseguimos observar no eixo horizontal da figura 3.5, a direção B+ indica o aumento da intensidade do campo magnético da polaridade sul e a direção B− indica a diminuição da intensidade do campo de polaridade sul.

3.5 Sensores de distância

Foram usadas duas gamas distintas de sensores de distância, figura 3.6, neste projeto.

Dois sensores de distância Sharp GP2Y0A41SK F 28 e quatro sensores de distância Sharp GP2Y0A21YK0F, em que apenas difere entre si a distância máxima que conseguem detetar, no caso do primeiro deteta entre 4 a 30 cm e no caso do segundo de 10 a 80 cm, como é apresentado na tabela 3.4.

No geral, o sensor de distância Sharp é um equipamento de medição de distância, composto por uma combinação integrada de PSD (detetor sensível à posição), IRED (díodo emissor infravermelho) e o circuito de processamento de sinal [14].

Figura 3.6: Sensor de distância Sharp.

	Sharp GP2Y0A41SK F 28	Sharp GP2Y0A21YK0F
Intervalo de medição de distância	4 a 30 cm	10 a 80 cm
Saída	Analógica	Analógica
Tamanho	29,5x13x13,5 mm	29,5x13x13,5 mm
Consumo de corrente típico	12 mA	30 mA
Tensão de alimentação	4,5 a 5,5 V	4,5 a 5,5 V

Tabela 3.4: Tabela de caraterísticas dos sensores de distância Sharp GP2Y0A41SK F 28 e Sharp GP2Y0A21YK0F.

Através da figura 3.7, conseguimos observar a relação entre o *output* dado pelo sensor e a distância a que se encontra o objeto.

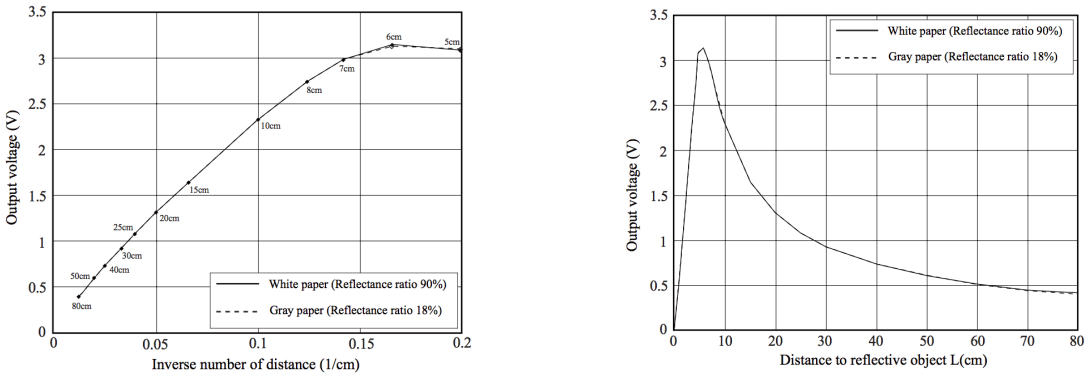


Figura 3.7: Exemplo da relação entre distância medida (cm) e o *output* obtido (V).

3.6 Buzzer, LEDs e Botões

Figura 3.8: Piezo Buzzer.

Neste projeto foram aplicados vários componentes de *feedback* para que o utente possa ter uma melhor experiência com um nível de imersão bastante elevado, e também para que possa interagir com o sistema.

Em primeiro lugar, foi dimensionado um alarme de aviso através da integração do dispositivo Piezo Buzzer, figura 3.8 [15].

Especificações	Piezo
Voltagem Operacional	3 a 30 V
Corrente	30 mA
Output a 30 cm	$\geq$ 95dB
Frequência	3,400 $\pm$500Hz
Peso	4 g

Tabela 3.5: Tabela de caraterísticas do Piezo Buzzer.

No que diz respeito à parte visual, foram colocados 6 LEDs, figura 3.9a, que dão o *feedback* do andamento das rodas, isto é, sempre que cada sensor de efeito Hall deteta um íman um LED acende, como existem 4 sensores de efeito de Hall foram disponibilizados

4 LEDs para este processo. No caso de um obstáculo ser detetado existe um LED (laranja) que acende, muito útil caso o utente não consiga ouvir o alarme sonoro por algum motivo. Na matéria de interação do próprio utente existem dois botões, figura 3.9b, que podem ser pressionados, um no guiador no punho direito e o outro é botão de *reset* do próprio Arduino. O botão que está situado no punho direito, tem como função ativar e desativar o sistema de travagem automático e o sistema de alarme, sempre que este botão é pressionado um LED (azul) é aceso. O botão de *reset* tem como finalidade reiniciar o Arduino.

(a) LED coloridos.

(b) Botão de pressão.

Figura 3.9: Figura do LED e do Botão de pressão.

3.7 Materiais do sistema de travagem

Para a criação de um sistema de travagem automático totalmente feito do zero, foram utilizados vários componentes para além dos servo motores (referidos na secção 3.3). Para construir a base de suporte e de atuação para esses servos motores, foram necessários muitos outros materiais [16] [17] [18] [19] [20], os quais vão ser mencionados e apresentados sob a forma de figuras:

(a) Madeira utilizada para construir o suporte do sistema de travagem.

(b) Abraçadeira em U de Inox utilizada na fixação do suporte.

(c) Dobradiça zinco 3/4 utilizada para produzir o movimento de alavanca.

Figura 3.10: Materiais utilizados no suporte do sistema de travagem.

(a) Arame de aço usado para construir a cabo de ação.

(b) Cerra-cabos de aço usados para fixar o arame de aço.

(c) Mola usada entre o cerra-cabos e o contato da alavanca.

Figura 3.11: Materiais utilizados no suporte do sistema de travagem.

4

Hardware e *Software* Desenvolvidos

Neste capítulo será abordado todo o processo de conceção e criação dos vários sistemas implementados no andarilho, quer a nível de *hardware* quer a nível de *software*.

4.1 Sistema de travagem automático

Depois da fase de revisão de literatura e seleção dos materiais necessários para o projeto, a criação do sistema de travagem automático foi a primeira coisa a ser pensada e desenvolvida. Este sistema tem como objetivo principal a travagem do andarilho sem que exista intervenção pela parte do utente.

4.1.1 *Hardware*

Uma vez que foram escolhidos os servo motores como o equipamento principal para criar a ação de travagem neste sistema, começou-se por criar um suporte forte e robusto para acomodar esses dispositivos, já que os servo motores conseguem exercer uma força considerável.

Para simplificar as coisas, observando a figura 4.1, podemos ver o resultado final deste sistema de travagem.

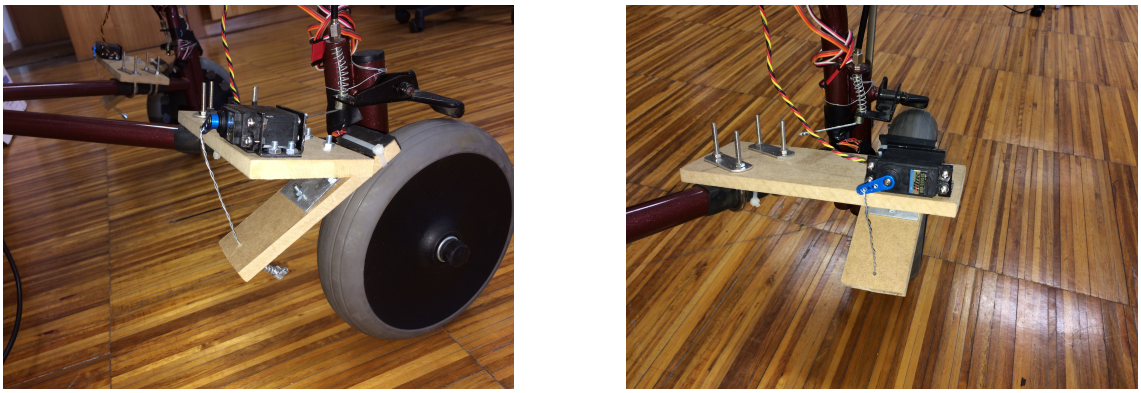


Figura 4.1: Sistema de travagem automático final.

O processo inicial passou por desenhar o suporte em madeira de forma a acomodar o servo motor sobre a roda traseira do andarilho (figura 4.2 - legenda 1). De seguida, para fixar esse suporte ao eixo do andarilho foram colocadas abraçadeiras em U (figura 4.2 - legenda 2), para uma melhor fixação e para que não exista a derrapagem do suporte, foi também colocada uma superfície de borracha entre as abraçadeiras e o eixo do andarilho. Finalizada esta parte, foi desenvolvido um mecanismo de alavanca para que fosse possível travar a roda do andarilho. Deste modo, através da ligação de uma dobradiça (figura 4.2 - legenda 4) e outra superfície de madeira (figura 4.2 - legenda 3), esta mais pequena que a do suporte, conseguiu-se criar uma alavanca que trava a roda através de contacto direto.

Para que este mecanismo tenha um funcionamento correto, através da ligação do servo motor (figura 4.2 - legenda 6), que está montado no suporte fixo, com o uso de arame de aço foi criado um cabo de ação (figura 4.2 - legenda 5), que é movimentado através da ação do braço do servo motor, isto é, sempre que o servo motor muda de posição o cabo de ação acompanha esse movimento. Portanto, foi necessário dar um comprimento e fixar este cabo de ação, para isso foi utilizado um cerra-cabos (figura 4.2 - legenda 7) com uma mola (figura 4.2 - legenda 8) para que o cabo de ação volte sempre à posição inicial e para proteger o mecanismo do servo motor caso seja exercida força em demasia. Para que o efeito de travagem seja mais eficiente foi colocada também uma camada de borracha (figura 4.2 - legenda 9) na superfície que faz o contacto com a roda do andarilho. Os materiais utilizados na conceção deste suporte já foram descritos e encontram-se na secção 3.7.

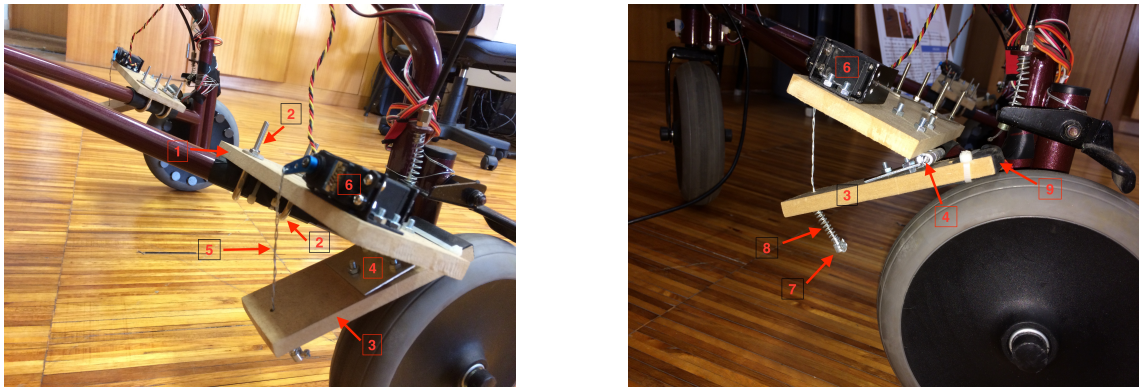


Figura 4.2: Diagrama de materiais utilizados no sistema de travagem.

Legenda da figura 4.2:

1 - Suporte em madeira construido para acomodar o servo motor sobre a roda traseira.

2 - Abraçadeira em U de INOX.

3 - Parte inferior em madeira que faz contacto com a roda do andarilho.

4 - Dobradiça de zinco de 3 furos.

5 - Arame enrolado que faz de cabo de ação para o sistema de travagem.

6 - Servo motor.

7 - Cerra-cabos que fixa o cabo de ação.

8 - Mola.

9 - Superfície de contacto direto com a roda, forrada com borracha.

Para entendermos melhor o funcionamento deste equipamento vamos recorrer à imagem da figura 4.3.

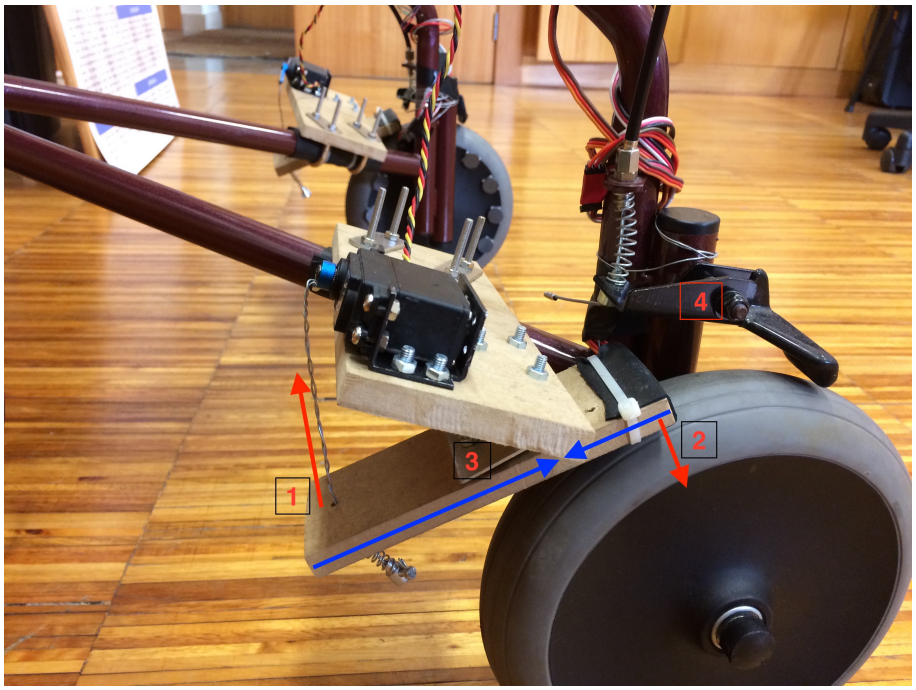


Figura 4.3: Diagrama do funcionamento do sistema de travagem.

Legenda da figura 4.3:

1 - Representação do sentido da força exercida pelo servo motor.

2 - Representação do sentido da força exercida pelo conjunto servo motor/dobradiça (efeito alavanca).

3 - Dobradiça.

4 - Sistema de travagem manual.

Observando as setas vermelhas na imagem da figura 4.3 consegue-se perceber que, se o servo motor movimentar o seu braço para cima, como demonstra a legenda 1 da figura 4.3, o efeito alavanca conseguido pela dobradiça (figura 4.3 - legenda 3) faz com que a posição oposta da superfície (que faz contacto com a roda do andarilho), aplique uma força no sentido contrário ao do servo motor, como está exemplificado na figura 4.3 - legenda 2 (o tamanho e a dimensão das setas não constituem qualquer relação com a ordem da força aplicada). E assim, é devido a essa força que a roda trava sempre que o servo motor puxa o cabo de ação. Observa-se também, através das linhas azuis na figura 4.3, que a distância da dobradiça ao cabo de ação é sensivelmente o triplo da distância da dobradiça à parte que faz contacto com a roda. Isso traduz o efeito de alavanca que foi pensado para esta aplicação, para que a força exercida no ato da travagem fosse maior, utilizando um binário menor pela parte do servo motor. No total existem quatro sistemas de travagem implementados no andarilho, dois sistemas de travagem manual como podemos observar na figura 4.3 - legenda 4 (um em cada roda traseira) que já vieram implementados no

andarilho convencional, e dois sistemas de travagem automáticos independentes (um em cada roda traseira) desenvolvidos neste projeto.

Explicada a montagem e funcionamento do suporte do sistema de travagem, passamos agora às ligações físicas deste sistema. Este sistema, como já foi referido, depende muito do seu dispositivo principal, o servo motor. Este já foi descrito e falado na secção 3.3. Neste projeto foram utilizados dois servo motores, um em cada roda traseira, e encontram-se ligados ao Arduino como está exemplificado na figura 4.4.

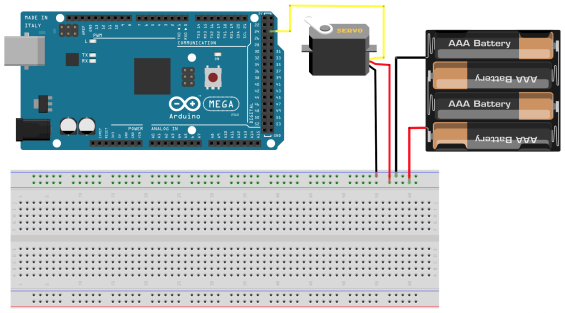


Figura 4.4: Ligação do servo motor ao arduino.

Os servo motores encontram-se ligados nos pinos digitais D24 e D25 do Arduino e possuem uma fonte externa (6 a 7 V) para um melhor funcionamento, pois com os 5 V fornecidos pelo Arduino os servo motores não funcionariam de uma forma eficiente.

4.1.2 *Software*

No que diz respeito ao *software* implementado para o funcionamento do sistema de travagem, este foi baseado na biblioteca do Arduino de controlo de servos [21]. Foi criada uma função para controlo específico da posição do braço do servo através da sua introdução em graus.

A função implementada:

```
1 /*----- Funcao SERVOS -------------------------------------------------------*/
2 // parametros de entrada, posicao do angulo de travagem (roda direita, roda ↩
      esquerda)
3 void servos(int postravagemdir, int postravagemesq) {
4 // transmite a posicao do angulo definido para o servo esquerdo
5    myservoLEFT.write(postravagemesq);
6 // transmite a posicao do angulo definido para o servo direito
7    myservoRIGHT.write(postravagemdir);
8 }
9 /*--------------------------------------------------------------------------*/
```

4.2 Sistema de deteção de obstáculos

O sistema de deteção de obstáculos, tal como o nome indica tem como objetivo principal detetar obstáculos à volta do andarilho e impedir que este embata contra eles. Os obstáculos podem ser paredes, portas, pessoas, escadas, varandas, degraus, etc. Este sistema tem que detetar esses mesmos obstáculos e prevenir que o utente que está a utilizar o andarilho sofra um acidente. Para isso, foram consideradas distâncias mínimas de segurança (30 cm na frente e nas laterais do andarilho, mais de 15 cm entre o utente e o andarilho e mais de 25 cm entre o andarilho e o chão) e sempre que os sensores detetarem distâncias inferiores, no caso das laterais e da frente, ou superiores, no caso da distância utente/andarilho e andarilho/chão, os travões automáticos e o sistema de alarme são ativados.

4.2.1 *Hardware*

Como dispositivos físicos foram utilizados seis sensores de distância à volta do andarilho (estes sensores já foram descritos e especificados na secção 3.5). Foram colocados dois sensores à frente, um a apontar para o chão e outro a apontar para a frente. Dois nas laterais do andarilho a apontar para a direita e para a esquerda, e os restantes dois são os que fazem o controlo da distância entre o utente e o andarilho, pelo que foram colocados na traseira do andarilho a apontar para as pernas do utente.

Conseguimos obter uma melhor perceção observando as imagens das figuras 4.5 e 4.6.

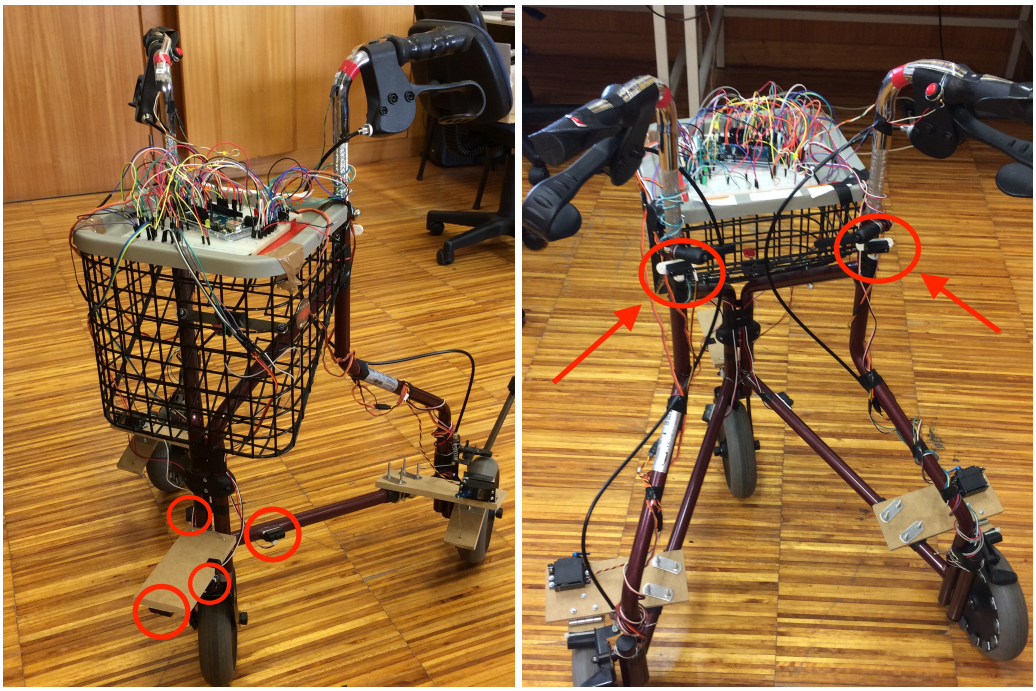


Figura 4.5: Posição dos sensores de distância - visão geral.

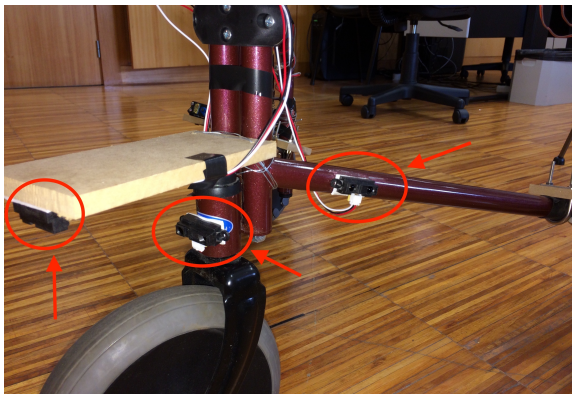


Figura 4.6: Posição dos sensores de distância frontais e laterais (o mais à esquerda aponta para o chão, o que está no meio aponta para a frente e o último é o sensor da lateral esquerda.)

O esquema de ligação dos sensores de distância encontra-se na figura 4.7, cada um dos seis sensores encontra-se ligado a uma entrada analógica do Arduino, o sensor da traseira esquerda está ligado ao pino A2, o da traseira direita ao pino A3, o da lateral direita ao pino A4, o da lateral esquerda ao pino A5, o sensor frontal está ligado ao pino A6 e o sensor que aponta para o chão está ligado ao pino A7.

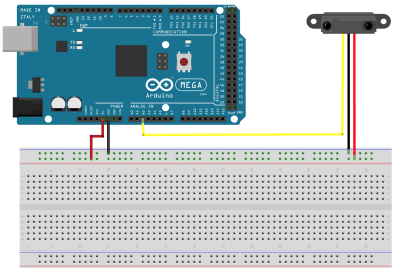


Figura 4.7: Esquema de ligação dos sensores de distância.

4.2.2 *Software*

Foi bastante complicado calibrar estes dispositivos, pois são baseados no sistema de deteção por infravermelhos pelo que variam bastante com a luminosidade do ambiente. Foram usados dois métodos de calibração e conversão (de tensão em V para distância em cm) dos valores enviados pelos sensores Sharp.

Observando então que o sensor envia um sinal analógico, é necessária a sua conversão para um valor de tensão de 0 a 5V. Para isso, foi utilizada a seguinte fórmula:

$$TensaoSensor = TensaoAnalogica \times \frac{5}{1024} \tag{4.1}$$

Onde *TensaoAnalogica* é o valor lido no pino analógico do Arduino multiplicado por 5 *Volts* e dividido por 1024 *bits* (ADC de 10 bits), para gerar assim uma variável chamada *TensaoSensor* em *Volts*.

Sendo assim, o primeiro método utilizado foi ligar cada sensor de distância ao Arduino para que se pudesse obter o valor de tensão. Foi utilizado o seguinte código:

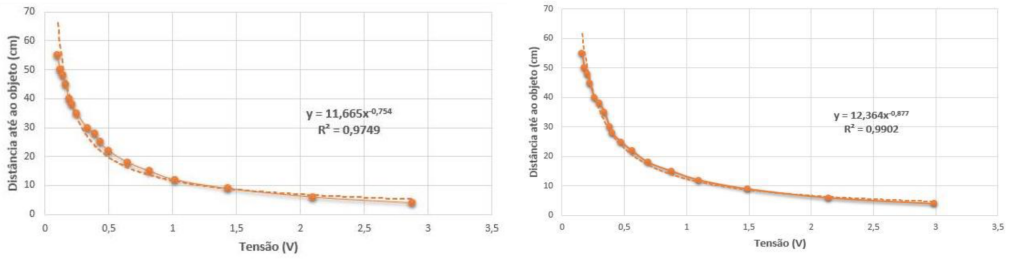
```
1 /*--------------------------------------------------------------------------*/
2 int sensorValue = analogRead(pino);  // guarda a leitura do pin analogico na ↩
      variavel sensorValue
3 float SensorOut = sensorValue * (5 / 1024);  // Converte o valor lido ADC↩
      (0-1023) para voltagem (0 - 5V)
4 Serial.println(SensorOut);  // escreve no monitor serie do Arduino o valor da ↩
      tensao
5 /*--------------------------------------------------------------------------*/
```

Sabendo o valor de tensão que o sensor está a medir, com o recurso a uma fita métrica foi colocado um obstáculo, paralelo ao sensor, a várias distâncias. Foi anotada a distância a que o obstáculo se encontrava e o valor da tensão que o sensor media. Desta forma, usando os valores anotados para fazer a conversão de tensão para distâncias em cm, utilizou-se um modelo de regressão não linear com recurso a uma função de potência. Logo a forma da função entre a distância e a tensão analógica do sensor é dada pela fórmula:

$$y = \beta \times {x_i}^a + \varepsilon_i \tag{4.2}$$

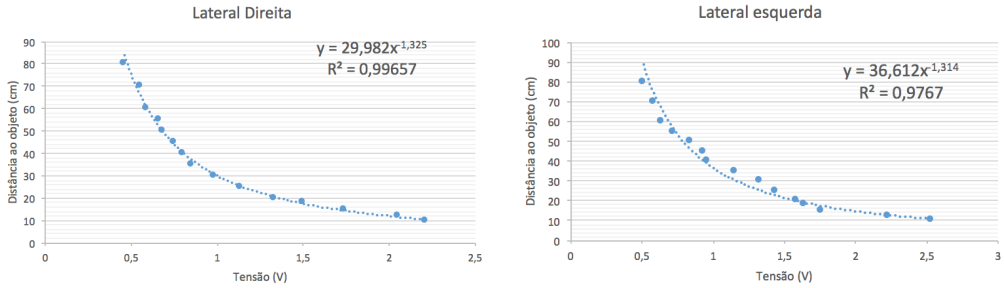
Onde y é o valor da distância obtido através do ajuste da tensão de saída do sensor a uma função de potência, x_i é o valor da tensão de saída do sensor e ε_i é o erro, ou seja, é a diferença entre o valor da amostra e o valor do modelo.

O coeficiente de correlação (R^2) é uma medida da qualidade do ajuste a um modelo estatístico. O R^2 varia entre 0 e 1, indicando, em percentagem, o quanto o modelo consegue explicar os valores observados. Quanto maior for o R^2, mais explicativo é o modelo e melhor ele se ajusta à amostra, neste caso o valor é de $97,5\%$ para o sensor do traseiro do lado esquerdo, $99,6\%$ para o sensor traseiro do lado direito, 96% para o sensor da lateral esquerda, 99% para o sensor da lateral direita, 98% para o sensor que aponta para a frente e 94% para o sensor que aponta para o chão, logo poderemos desprezar o erro na implementação da função no *software* [1] [5].



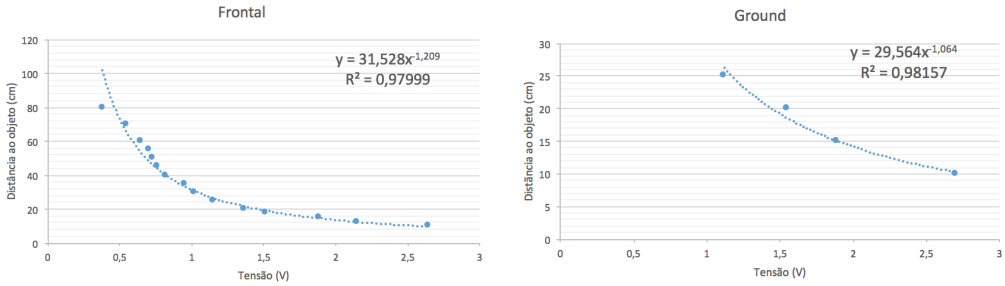
(a) Sensor traseiro esquerdo. (b) Sensor traseiro direito.

Figura 4.8: Gráficos de cálculo das funções dos sensores traseiros.



(a) Sensor lateral direito. (b) Sensor lateral esquerdo.

Figura 4.9: Gráficos de cálculo das funções dos sensores laterais.



(a) Sensor que aponta para a frente. (b) Sensor que aponta para o chão.

Figura 4.10: Gráficos de cálculo das funções dos sensores frontal e ground.

Utilizando assim o modelo de regressão não linear calculado pelos gráficos 4.8, 4.9 e 4.10, é calculada a distância entre os sensores e os obstáculos de acordo com as seguintes equações:

$$DistTraseiraEsquerda(cm) = 11,665 \times TensaoSensor^{-0,754} \tag{4.3}$$

$$DistTraseiraDireita(cm) = 12,364 \times TensaoSensor^{-0,877} \tag{4.4}$$

$$DistLateralDireita(cm) = 29,982 \times TensaoSensor^{-1,325} \tag{4.5}$$

$$DistLateralEsquerda(cm) = 36,612 \times TensaoSensor^{-1,314} \tag{4.6}$$

$$DistFrontal(cm) = 31,528 \times TensaoSensor^{-1,209} \tag{4.7}$$

$$DistGround(cm) = 29,564 \times TensaoSensor^{-1,064} \tag{4.8}$$

Onde $TensaoSensor$ é o valor calculado pela equação 4.1 e $DistTraseiraEsquerda(cm)$, $DistTraseiraDireita(cm)$, $DistLateralDireita(cm)$, $DistLateralEsquerda(cm)$, $DistFrontal(cm)$ e $DistGround(cm)$ são as distâncias em centímetros medidas nos vários sensores.

Traduzindo então estas equações para código, foram criadas funções para cada um dos sensores de distância. Temos o exemplo seguinte:

```
1  /*---- funcao do sensor de distancia perna direita -------------------------*/
2  int GetDistance_right(int pin) {                // funcao para definir a ↩
       distancia do sensor da perna direita
3  // int sensorValue = analogRead(pin);           // guarda a leitura do pin ↩
       analogico na variavel sensorValue
4  int sensorValue=MeanFilter(pin, 30);            // guarda a media calculada ↩
       com 30 amostras da leitura do pin analogico do sensor
5  float SensorOut = sensorValue * (5.0 / 1024.0); // Converte o valor lido o ADC↩
       (0-1023) para voltagem (0 - 5V)
6  float dist_cm = 12.364 * pow(SensorOut, -0.877);// regressao de potencia ↩
       distancia
7      if(dist_cm>50) dist_cm=50;                  // caso a distancia for maior ↩
           que 50cm devolve 50cm
8      if(dist_cm<0) dist_cm=0;                    // caso a distancia for menor ↩
           que 0cm devolve 0cm
9  return ((int)dist_cm);                          // retorna a distancia em ↩
       centimetros
10 }
11 /*--------------------------------------------------------------------------*/
```

É a função criada para o sensor traseiro direito do andarilho, onde podemos observar que na leitura do valor de tensão no pino de entrada do Arduino é processada por um filtro de média, o qual é definido por esta função:

```
1  /*----- Funcao do filtro da media ------------------------------------------*/
2  float MeanFilter(int pin, int numSamples) { // parametros de entrada PIN de ↩
       entrada do sensor e numero de amostras
3    int sum=0;                                  // inicializacao do soma
4    for(int i=0;i<numSamples;i++){   // ciclo para fazer a soma de todas as ↩
         amostras
5      sum+=analogRead(pin);          // faz o soma de todas as leituras da ↩
           entrada analogica pin
6    }
7   return ((int) (sum/numSamples));  // devolve a media de todas as amostras
8  }
9  /*--------------------------------------------------------------------------*/
```

Esta é usada para que o cálculo da distância seja mais preciso, pois em vez de fazer as contas com apenas 1 valor, são utilizadas 30 amostras de leitura e é calculada a média dessas amostras para obter assim o valor lido no pino de entrada. O tempo que demora a calcular as 30 amostras não prejudica a velocidade de deteção, pois demora cerca de 100 microssegundos (0.0001s) para ler e converter uma entrada analógica, portanto a taxa máxima de leitura é de cerca de 10 000 vezes por segundo.

De seguida é aplicada a equação descrita em 4.4, para gerar assim o valor da distância em centímetros.

O segundo método utilizado foi através do uso da função da biblioteca do Arduino, map(value, 0, 1023, 80, 10) [22]:

```
1 /*------------------------------------------------------------------------*/
2 float dist_cm = map(sensorValue, 0, 1023, 80, 10); // mapeia a voltagem ↩
      guardada no sensorValue e converte em cm entre 10 e 80cm
3 /*------------------------------------------------------------------------*/
```

Esta função converte o valor de tensão lido (analógico) para 0 - 5V utilizando um ADC de 10 bits (0 - 1023 bits), mapeando assim, linearmente, para valores de distância entre 10 a 80 cm.

Construindo dessa forma a função:

```
1  /*----funcao sensor de distancia lateral direito ---------------------------*/
2  int GetDistance_sright(int pin) {                  // funcao para definir a ↩
       distancia do sensor da lateral direita
3  //int sensorValue = analogRead(pin);               // guarda a leitura do pin ↩
       analogico na variavel sensorValue
4  int sensorValue=MeanFilter(pin, 30);               // guarda a media calculada ↩
       com 30 amostras da leitura do pin analogico do sensor
5  float dist_cm = map(sensorValue, 0, 1023, 80, 10);// mapeia a voltagem ↩
       guardada no sensorValue e converte em cm entre 10 e 80cm
6      if(dist_cm>=80) dist_cm=80;                    // caso a distancia for ↩
           maior que 80cm devolve 80cm
7      if(dist_cm<10) dist_cm=10;                     // caso a distancia for ↩
           menor que 10cm devolve 10cm
8  return ((int)dist_cm);                             // retorna a distancia em ↩
       centimetros
9  }
10 /*------------------------------------------------------------------------*/
```

O primeiro método explicado foi apenas utilizado nos 2 sensores de distância traseiros (4 a 50 cm), isto é, os sensores que detetam a presença do utente (distância entre o andarilho e o utente). O segundo método foi utilizado nos 4 sensores restantes (10 a 80 cm), pois obteve melhores resultados que o primeiro.

4.3 Sistema de deteção de movimento

O sistema de deteção de movimento tem como objetivo principal saber qual o sentido de rotação das rodas do andarilho (as duas rodas traseiras) e calcular a sua velocidade.

4.3.1 *Hardware*

Para construir este sistema foram estudadas várias hipóteses de sensores e encoders, de entre os quais, encoders rotativos e sensores de interrupção. Foi chegada à conclusão que os sensores de efeito de Hall seriam os mais indicados para serem aplicados neste tipo de rodas. O material constituinte das rodas não era o mais indicado para a aplicação dos restantes sensores (encoders rotativos e sensores de interrupção), tornando assim os sensores de efeito de Hall a escolha mais prática e fiável.

Existem, no total, quatro sensores de efeito de Hall inseridos no andarilho (dois em cada roda traseira). O sensores de efeito de Hall contam com um *Schmitt trigger* construído em histerese e ligado ao amplificador operacional. Quando o fluxo magnético que passa através do sensor Hall excede um valor pré-definido, a saída do dispositivo muda rapidamente da condição DESLIGADA para a uma condição LIGADA. Esta histerese embutida elimina qualquer oscilação do sinal de saída enquanto o sensor se move para dentro e fora do campo magnético. Assim, o sensor de saída digital tem apenas dois estados, LIGADO e DESLIGADO.

Existem alguns parâmetros destes sensores que são importantes para a compreensão do seu funcionamento, são eles:

BOP: Ponto de operação magnético; é o nível de campo magnético a partir do qual um dispositivo Hall liga.

BRP: Ponto de libertação magnética; é o nível de campo magnético a partir do qual um dispositivo Hall desliga.

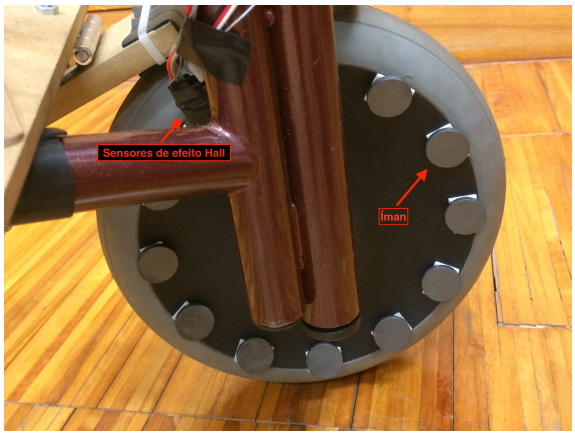
BHYS: Histerese magnética. $\mathrm{BHYS} = |\mathrm{BOP} - \mathrm{BRP}|$.

Podemos encontrar quatro tipos de sensores de efeito de Hall digitais, Unipolar, Bipolar, Omnipolar e Latch. Como neste projeto só foram usados sensores de efeito de Hall Unipolares só estes irão ser descritos.

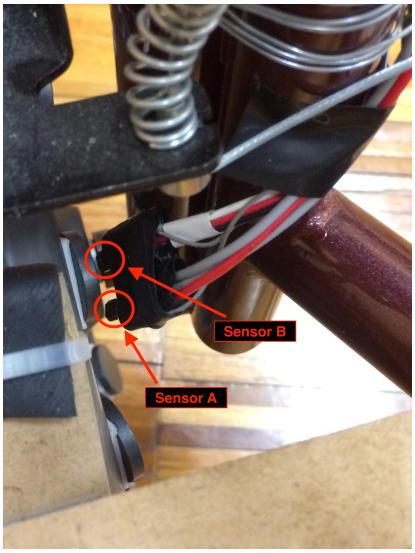
Os sensores de efeito de Hall Unipolares são o tipo de sensores que operam na presença de um campo magnético positivo, ou seja, o sensor só irá conduzir quando o pólo sul de um íman se aproximar dele. Enquanto o íman estiver próximo do sensor, ele continuará ativo (a conduzir). A condução só é cessada quando o íman é afastado.

Explicado assim o funcionamento destes sensores, observando as figuras 4.11, podemos facilmente notar que foram colocados doze ímanes (figura 4.11a), em volta da roda onde se queria medir a velocidade e o sentido de rotação, e colocaram-se dois sensores

de efeito de Hall, designados como sensor A e sensor B (figura 4.11b), de forma a que conseguissem fazer os estados de leitura de acordo com a tabela 4.1.



(a) Posição dos doze ímanes na roda do andarilho.



(b) Posição dos dois sensores de efeito de Hall junto da roda.

Figura 4.11: Posição dos ímanes e dos sensores de Hall na roda do andarilho.

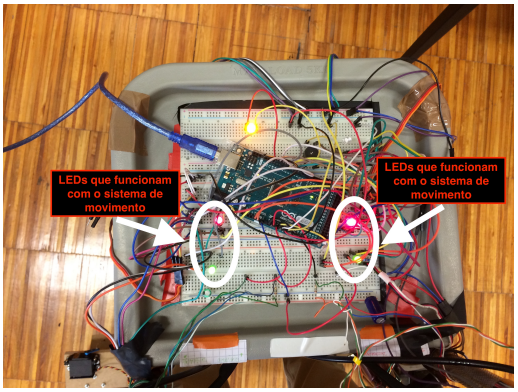
Leitura	Sensor A	Sensor B
Estado 1	0	0
Estado 2	1	0
Estado 3	1	1
Estado 4	0	1

Tabela 4.1: Tabela de estados efetuados pelos sensores de Hall.

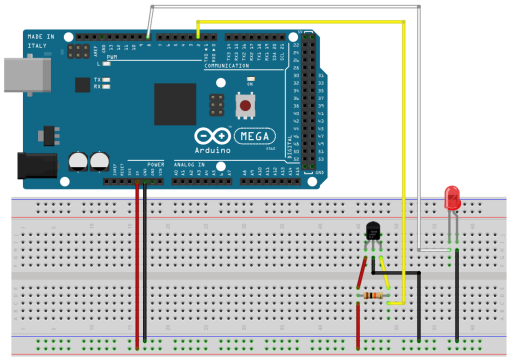
Para se ser mais exato, os sensores de efeito Hall apenas produzem a saída de 1 ou 0, isto é, quando o sensor está sem a influência de um campo magnético (não deteta nenhum dos ímanes da roda) devolve o valor de 1; caso o sensor esteja na presença de um campo magnético (deteta um dos ímanes da roda) devolve 0. Com isto, sabendo o valor lido por cada um dos sensores A e B, através da tabela de estados 4.1, conseguimos calcular o sentido e velocidade de rotação.

Cada sensor de efeito de Hall encontra-se ligado a uma entrada digital especial do Arduino, ou seja, esta entrada especial é uma entrada de interrupção externa. Para que possamos utilizar uma função de interrupção externa da biblioteca do Arduino necessitamos de utilizar a entrada destes pinos específicos.

Existem seis entradas de interrupção externa no arduino mega (D2, D3, D18, D19, D20, D21), do qual foram usadas apenas quatro, de acordo com o esquema de ligação da figura 4.12b.



(a) Quatro LEDs do sistema de movimento.



(b) Esquema de ligação do sensor de efeito Hall.

Figura 4.12: LEDs e esquema do Sistema de movimento.

Os sensores de efeito de Hall da roda esquerda encontram-se ligados nos pinos D2 (pino de interrupção externa) e D5, e os da roda direita encontram-se ligados aos pinos D3 (pino de interrupção externa) e D4. Como foi preciso utilizar mais que uma função de interrupção do Arduino, uma para fazer o algoritmo da velocidade e outra para o algoritmo do sentido de rotação necessitou-se, posteriormente, de fazer uma ligação adicional de um pino de interrupção, ou seja, o Vout (saída) de um dos sensores de efeito de hall de cada roda, encontra-se ligado a duas entradas de interrupção. Portanto, o Sensor A da roda esquerda encontra-se ligado ao pino D2 e D18, e o sensor A da roda direita encontra-se ligado ao pino D3 e D19. Através do esquema da figura 4.12b, vemos que também existe um LED para dar uma melhor perceção ao utente de como se está a movimentar, ou seja, cada vez que o sensor de efeito de Hall de cada roda deteta um íman o LED acende. Cada LED está associado a cada sensor de efeito Hall, portanto existem quatro LEDs (figura 4.12a) que acendem com o movimento das rodas, estes encontram-se ligados nos pinos D6, D7, D8 e D9 do Arduino.

4.3.2 *Software*

Relativamente ao *software* desenvolvido foram criados dois algoritmos, um de velocidade e outro para definir o sentido de rotação da roda. O algoritmo da velocidade encontra-se descrito pela função seguinte:

```
/*---------- variaveis globais --------------------------------------------*/
int rpm;                  // variavel de rotacoes por minuto
float veloc;              // variavel de rotacoes por minuto
volatile byte pulsos;     // variavel que guarda o numero de pulsos contados
unsigned long timeold;    // variavel que guarda o tempo de rotacao para actualizar as contagens

float veloc_maxima = 1;     // 1000 m/h = 1 km/h - velocidade lenta de seguranca

// Numero de pulsos por volta do sensor de Hall
unsigned int pulsos_por_volta = 12; // numero de imans nas rodas

/*---------- Funcao para calcular os rpm e a velocidade -------------------*/
int velocidade() {
  if (millis() - timeold >= 1000)
  {
    detachInterrupt(0); // Desabilita interrupcao durante o calculo
    rpm = (pulsos*60)/pulsos_por_volta; // numero de (pulsos obtidos * 60) / 12 pulsos
    veloc = 2*PI*0.00008*rpm*60; // velocidade = 2*PI*raio*rpm*60 (km/h)
    timeold = millis(); // temporizador e atualizado
    pulsos = 0;  // o contador de pulsos e reiniciado a 0

   Serial.print("RPM = ");
   Serial.print(rpm, DEC);
   Serial.print("  Velocidade (km/h) = ");
   Serial.println(veloc);

    attachInterrupt(0, contador, FALLING); // FALLING for when the pin goes from high to low.
  }
}
/*-------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------*/
void contador(){ // cada vez que uma interrupcao e ativada e chamada esta funcao
  pulsos++; // contador
}
/*-------------------------------------------------------------------------*/
```

Através da análise do código anterior, conseguimos ver que contando o número de interrupções através da função $attachInterrupt(pino, função, modo)$ [23], que devido ao modo *FALLING* que ativa a interrupção no pino de interrupção 0, sempre que este passa de HIGH para LOW, chamando a função que tem como base um contador de pulsos (conta o número de ímanes detetados).

Sabendo isto, basta apenas calcular o número de rotações por minuto através da equação $rpm = \frac{pulsos \times 60}{Nímanes}$, e posteriormente calcular a velocidade através da equação $velocidade = 2 \times \pi \times raio \times rpm \times 60$, para assim obtermos a velocidade em km/h.

Quanto ao algoritmo que define o sentido de rotação da roda, é calculado de maneira semelhante, com a pequena diferença que em vez de usar apenas um dos sensores de efeito de Hall presentes na roda, este usa os dois em simultâneo.

Vamos analisar o código para uma melhor compreensão do seu funcionamento.

```
/*---------- variaveis globais -------------------------------------------*/
volatile bool pronto;     // variavel de interrupcao para saber se o sentido esta definido
volatile bool sentido;    // variavel que define o sentido, se 1 frente se 0 tras
const byte encoderPinA = 18;  // sensor de Hall esquerda FRENTE (interrupcao 5) para o sentido
const byte encoderPinB = 5;   // sensor de Hall esquerda TRAS
static long rotacao;      // variavel para contagem de rotacoes para o sentido

/*----------- Funcao sentido (Hall Sensor) ---------------------------------*/
int sentido() {
  if (pronto == true) { // assim que o sentido estiver definido
    if (sentido == HIGH) {  // se a variavel sentido for HIGH roda esta a girar para a frente
      Serial.println (" RODA ESQUERDA FRENTE ");
    } else {   // caso contrario (se sentido = LOW) roda gira para tras
      Serial.println (" RODA ESQUERDA TRAS ");
    }
    pronto = false;
  }
}
```

```
/*-----------------------------------------------------------------------*/
void setup(){
attachInterrupt(digitalPinToInterrupt(encoderPinA), interrupt, CHANGE); //CHANGE to trigger the interrupt whenever the pin changes value
}
/*-----------------------------------------------------------------------*/
void interrupt() {
  if (digitalRead(encoderPinA) == HIGH) {  // se a leitura do sensor de hall do pinA for HIGH
    sentido = digitalRead(encoderPinB); // entao o sentido da roda vai ser dado pelo estado do sensor pinB (1 ou 0)
  } else {  // caso contrario
    sentido = !digitalRead(encoderPinB);// o sentido vai ser igual ao NAO estado pinB (1 ou 0)
  }
  pronto = true; // coloca a variavel a true para iniciar a funcao sentido()
}
/*-----------------------------------------------------------------------*/
```

Conseguimos ver que o sentido da rotação da roda é determinado na função *interrupt*(), através do método do *attachInterrupt*() [23] vemos a introdução de um novo modo, o *CHANGE* ativa esta função sempre que o valor lido no pino de interrupção se altera. Com isto conseguimos saber sempre os quatro valores de estado que já foram referido na tabela 4.1. Embora apenas consigamos determinar o sentido de rotação quando são contados dois estados, isto é, se o estado inicial detetado pela função for $sensorA = 1$ e $sensorB = 1$, só quando passa para o estado $sensorA = 0$ e $sensorB = 0$, é que o algoritmo consegue determinar em que sentido a roda está a rodar.

Vamos analisar este funcionamento mais de perto. Quando a roda do andarilho é girada, é sempre obtido um estado em cada pino como já vimos. Portanto, sempre que movimentamos o andarilho para a frente são lidos estados numa determinada ordem, como está demonstrado na figura 4.13. O mesmo acontece quando movimentamos o andarilho para trás.

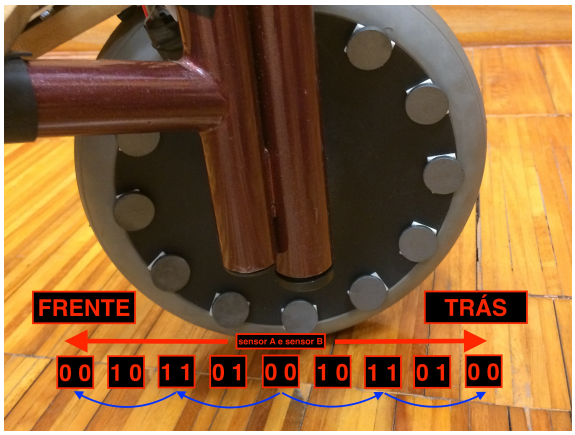


Figura 4.13: Esquema de funcionamento da deteção do sentido.

Logo, sabemos definir o sentido do movimento analisando a passagem pelos estados, ou seja, como as setas azuis na figura 4.13 indicam a análise tem que passar de dois estados iguais por dois estados diferentes e novamente por dois estados iguais para sabermos em que sentido o andarilho se movimentou.

A diferença óbvia está nos estados diferentes entre si. Se ao movimentarmos o andarilho este passar pelos estados na ordem $10 \rightarrow 11$ ou $01 \rightarrow 00$ está a andar para trás, se detetar os estados na ordem $01 \rightarrow 11$ ou $10 \rightarrow 00$ está a andar para a frente.

Relativamente ao *software* de controlo dos LEDs, para controlar os quatro LEDs associados aos quatro sensores de efeito de Hall foi criada a função seguinte:

```
/*----------- Funcao controla os LEDS do HALL SENSOR ----------------------*/
void LEDS(boolean led) {
  // LEDS para os SENSORES HALL
   if(led == 1){
    if((digitalRead(SHALL_ESQ_FRENTE)==LOW)){ // Quando deteta o iman
      digitalWrite(LEDPin_ESQ_FRENTE, HIGH);  // Acende LED
    } else {
      digitalWrite(LEDPin_ESQ_FRENTE, LOW);   // Apaga LED
    }
    if((digitalRead(SHALL_ESQ_TRAS)==LOW)){   // Quando deteta o iman
      digitalWrite(LEDPin_ESQ_TRAS, HIGH);    // Acende LED
    } else {
      digitalWrite(LEDPin_ESQ_TRAS, LOW);     // Apaga LED
    }
    if((digitalRead(SHALL_DIR_TRAS)==LOW)){   // Quando deteta o iman
      digitalWrite(LEDPin_DIR_TRAS, HIGH);    // Acende LED
    } else {
      digitalWrite(LEDPin_DIR_TRAS, LOW);     // Apaga LED
    }
    if((digitalRead(SHALL_DIR_FRENTE)==LOW)){ // Quando deteta o iman
      digitalWrite(LEDPin_DIR_FRENTE, HIGH);  // Acende LED
    } else {
      digitalWrite(LEDPin_DIR_FRENTE, LOW);   // Apaga LED
    }
   }

   if(led == 0){
    digitalWrite(LEDPin_ESQ_FRENTE, LOW); // TODOS OS LEDS OFF
    digitalWrite(LEDPin_ESQ_TRAS, LOW);
```

```
30      digitalWrite(LEDPin_DIR_TRAS, LOW);
31      digitalWrite(LEDPin_DIR_FRENTE, LOW);
32    }
33  }
34  /*-------------------------------------------------------------------------*/
```

Esta função é utilizada para acender ou apagar os LEDs de acordo com a preferência do utente, de acordo com a variável booleana de entrada que ele introduz; caso de 1 os LEDs estão acesos e piscam com o movimento do andarilho, caso de 0 todos os LEDs permanecem sempre apagados.

4.4 Sistema de medição de força

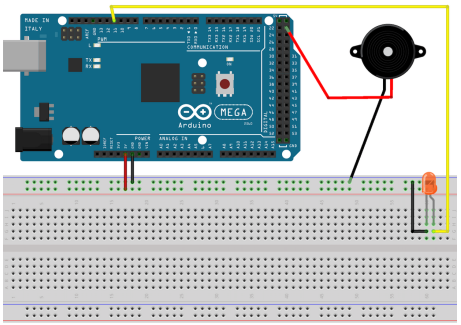
Este sistema tem como objetivo medir a força aplicada nos punhos do andarilho. Este método já se encontrava implementado e a breve explicação do seu funcionamento consta em anexo (apêndice).

4.5 Sistema de Alarme

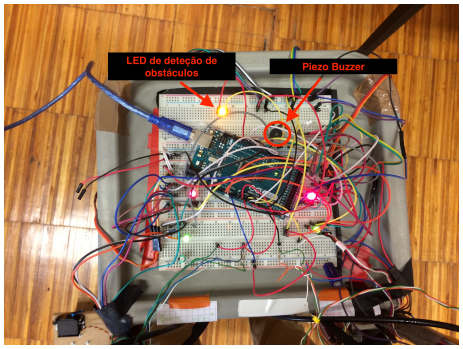
O sistema de alarme foi desenvolvido com o intuito de alertar o utente da deteção de um obstáculo, isto é, sempre que um obstáculo é detetado ou o utente se afasta demasiado do andarilho, um alarme sonoro e uma luz LED laranja são ativados.

4.5.1 *Hardware*

Os componentes que foram utilizados neste sistema são um Piezo Buzzer e um LED laranja, que se encontram especificados na secção 3.6. O esquema de ligações foi efetuado de acordo com a seguinte figura 4.14a, e o sistema tem o aspeto apresentado na figura 4.14b.



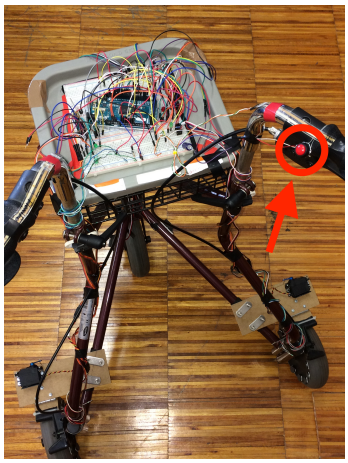
(a) Esquema de ligações do Piezo Buzzer e do LED de alarme.



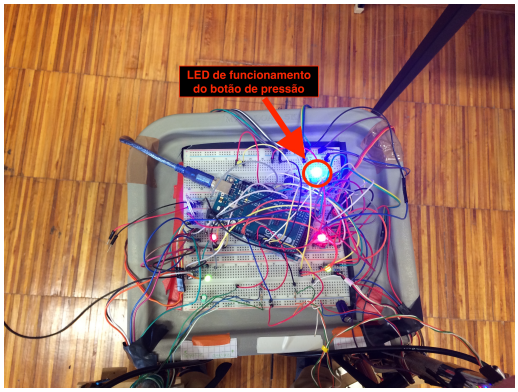
(b) Imagem do Buzzer e do LED do sistema de alarme.

Figura 4.14: Sistema de alarme do andarilho.

Também foi introduzido no sistema um botão de pressão (figura 4.15a) com um LED de funcionamento (figura 4.15b), isto é, imaginemos que o sistema de deteção de obstáculos deteta um obstáculo, imediatamente o alarme do buzzer e o LED laranja são ativados para alertar o utente, mas também o sistema de travagem automático atua, impedindo o utente de se movimentar por se encontrar na presença de um obstáculo. Assim sendo, foi introduzida uma estratégia para que o utente possa sair desse estado de travagem, carregando no botão de pressão, e sempre que o fizer, tanto o sistema de alarme como o sistema de travagem automático serão desativados.



(a) Imagem do botão de pressão no guiador direito.



(b) LED de funcionamento do botão de pressão.

Figura 4.15: Botão de pressão do andarilho.

O esquema de ligações do botão de pressão, bem como o seu LED de funcionamento estão especificados na figura 4.16.

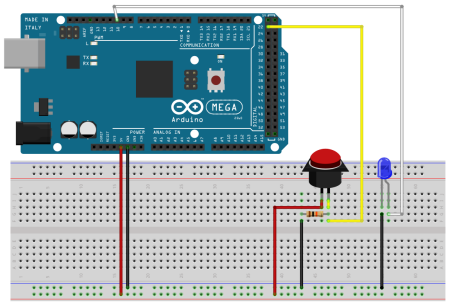


Figura 4.16: Esquema de ligações do Botão de pressão e do LED de funcionamento.

4.5.2 *Software*

Foram utilizadas dois tipos de funções, uma para lidar com o funcionamento do Buzzer e outra para o funcionamento do Botão de pressão. Função que define o funciona-

mento do botão de pressão e o seu LED de funcionamento:

```
/*----- Funcao Button ------------------------------------------------------*/
void Button() {
  ButtonState = digitalRead(PushButton);  // le o estado do botao

  if(ButtonState == HIGH){           // se o botao for pressionado
    digitalWrite(LEDButton, HIGH);   // LED acende (AZUL)
    digitalWrite(LEDobstaculo, LOW); // LED dos obstaculos apagado
    }
  else{ // caso contrario
    digitalWrite(LEDButton, LOW);    // LED do botao apagado
    }
}
/*--------------------------------------------------------------------------*/
```

Funções que definem o funcionamento do Buzzer:

```
/*----- Funcao Buzzer ON ---------------------------------------------------*/
void BuzzerON() {          // liga o buzzer (barulho de alarme)
  tone(BuzzerPin, 440); // utilizacao da funcao tone da biblioteca do arduino ↩
      (440Hz nota A)
}
/*----- Funcao Buzzer OFF --------------------------------------------------*/
void BuzzerOFF() {    // desliga o buzzer (barulho de alarme)
  noTone(BuzzerPin); // utilizacao da funcao noTone da biblioteca do arduino
}
/*--------------------------------------------------------------------------*/
```

O botão de pressão encontra-se ligado ao pino D22 do Arduino. Quanto ao Buzzer está ligado ao pino D23 do Arduino, e para o seu funcionamento correto recorreu-se ao uso das funções $tone(pino, frequencia)$ e $noTone(pino)$ da biblioteca do Arduino [24].

4.6 Modo de Funcionamento

Uma vez já conhecidos todos os sistemas implementados e em funcionamento no andarilho, vamos passar a uma breve descrição de como é que estes sistemas interagem entre si.

O objetivo principal está no funcionamento conjunto do sistema de deteção de obstáculos e no sistema de travagem automática. Sempre que um obstáculo for encontrado a menos de 30 cm do andarilho, pelo sistema de deteção de obstáculos (laterais e frente), este tem que ativar o sistema de travagem automático e fazer com que o andarilho pare. O mesmo acontece quando o utente se afasta da posição de segurança, isto é, caso o utente esteja apoiado no andarilho mas as suas pernas se afastem dele a uma distância superior a 15 cm. Do mesmo modo, o equipamento deteta o solo a uma distância de 25 cm, caso esta distância aumente abruptamente (definida como 5 cm) será também ativado o sistema de travagem automático, uma vez que o utente pode estar em perigo de cair numa escada ou varanda.

Para realizar tal tarefa, foram implementadas as seguintes restrições:

```
/*-----------------------------------------------------------------------*/
  // Condicao de funcionamento dos travoes automaticos
  if(digitalRead(PushButton)==LOW){   // se o botao de pressao nao estiver a ser pressionado
    if((DISTANCE_RIGHT>= 15 && DISTANCE_LEFT>= 15) || (DISTANCE_SLEFT<= 30) || (DISTANCE_SRIGHT<= 30) || (DISTANCE_FRONTAL<= 30) || (DISTANCE_GROUND>= 30)){
      servos(80,100);  // os servos travam
      digitalWrite(LEDobstaculo, HIGH); // LED dos obstaculos acende
      BuzzerON();  // Buzzer toca (activado ON)
    } else { // caso contrario
      BuzzerOFF(); // o Buzzer nao toca (desligado OFF)
      servos(110,80); // Posicao inicial dos servos (nao trava)
      digitalWrite(LEDobstaculo, LOW); // LED dos obstaculos apagado
     }
  }
/*-----------------------------------------------------------------------*/
```

No caso da interação do botão de pressão já definido na secção 4.5, deve desativar o sistema de travagem automático e o sistema de alarme para que o utente possa mudar de direção quando deteta um obstáculo.

Existe também uma restrição à velocidade máxima de andamento do andarilho, caso o utente ultrapasse uma velocidade máxima limite o sistema de travagem automático também é ativado. Esta velocidade máxima limite, por defeito, está definida como 1 km/h, mas pode ser alterada para 2 km/h caso o utente assim desejar. Para isso, basta carregar no botão de *reset* do arduino e no botão de pressão em simultâneo, para aumentar assim esta velocidade máxima limite para 2 km/h.

A seguinte condição é a que limita essa velocidade máxima:

```
/*-----------------------------------------------------------------------*/
  if(veloc >= veloc_maxima){ // limita a velocidade de andamento do andarilho (velocidade de segurança)
    servos(80,100);  // servos travam
    BuzzerON();   // Buzzer toca (alarme)
  }
/*-----------------------------------------------------------------------*/
```

Os restantes sistemas implementados funcionam independentemente, bastando apenas chamar as funções criadas, não tendo qualquer restrição.

```
/*-----------------------------------------------------------------------*/
void loop(){
  LEDS(1);         // LEDS dos sensores das rodas acendem (se LEDS(0) os leds ficam sempre OFF)
  Button();        // Chama a funcao do botao de pressao
  velocidade();    // Chama a funcao da velocidade das rodas
  sentido();       // Chama a funcao do sentido das rodas


```

```
9    // Sensores de Distancias
10   DISTANCE_LEFT=GetDistance_left(SENSOR_DISTANCE_LEFT);
11   DISTANCE_RIGHT=GetDistance_right(SENSOR_DISTANCE_RIGHT);
12   DISTANCE_SLEFT=GetDistance_sleft(SENSOR_DISTANCE_SLEFT);
13   DISTANCE_SRIGHT=GetDistance_sright(SENSOR_DISTANCE_SRIGHT);
14   DISTANCE_FRONTAL=GetDistance_frontal(SENSOR_DISTANCE_FRONTAL);
15   DISTANCE_GROUND=GetDistance_ground(SENSOR_DISTANCE_GROUND);
16
17   // Sensores de Forca
18   FORCE_LEFT=GetForce(SENSOR_FORCE_LEFT);
19   FORCE_RIGHT=GetForce(SENSOR_FORCE_RIGHT);
20   }
21 /*-------------------------------------------------------------------------*/
```

Todos os valores de velocidade, sentido, distâncias e forças são mostradas no monitor série do Arduino, caso o utente o desejar.

5

Testes e Resultados

Nos testes ao sistema de travagem automática, em primeiro lugar foi testada a capacidade de travagem dos servo motores. Para isso, foram dadas várias posições aos servos e testado o modo de como travavam o andarilho, empurrando o mesmo, para verificar se, de facto, as rodas bloqueavam. Os resultados estão apresentados na tabela 5.1.

Legenda para o campo da intensidade de travagem da tabela 5.1, vai de 0 a 4, sendo:

0 não trava nada; 1 trava ligeiramente; 2 trava moderadamente; 3 trava bem; 4 trava muito bem;

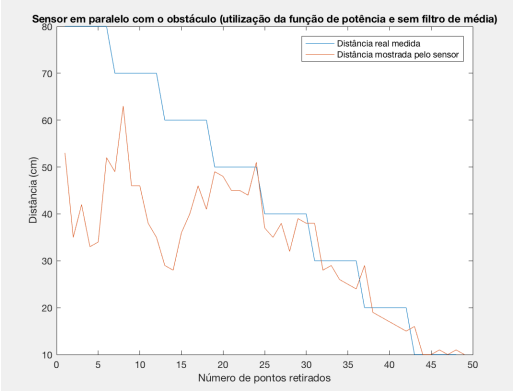
Lado	Rotação do braço do servo	Intensidade de travagem
Direito	80°	4
	90°	2
	100°	1
	110°	0
Esquerdo	80°	0
	90°	2
	100°	3
	110°	4

Tabela 5.1: Resultados dos testes aos travões.

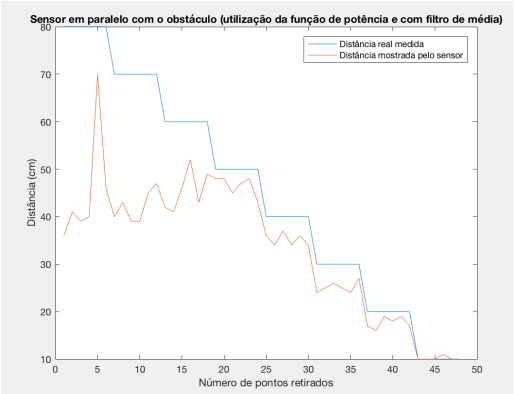
Relativamente aos testes de deteção de obstáculos, foram colocados inúmeros obstáculos ao alcance da deteção dos sensores, como cadeiras, mochilas, caixotes do lixo, paredes, pessoas, portas e escadas. O sistema de deteção foi sempre capaz de detetá-los e ativar o sistema de travagem automático e o sistema de alarme, a tempo de parar antes de embater no obstáculo.

Os gráficos 5.1, 5.2 e 5.3, contêm os resultados dos testes efetuados aos sensores de distância com a utilização dos 2 métodos usados para converter a tensão em centímetros (curva função de potência e função *map()*), com e sem utilização do filtro de média. Os testes foram efetuados com um obstáculo em paralelo ao sensor. Também foi efetuado um teste com uma inclinação do sensor a 45° relativamente ao obstáculo.

Através da análise dos resultados, conseguimos perceber o quão imprecisos estes sensores são. Todos os métodos que foram introduzidos tentaram de certa forma, suavizar os erros observados e conseguir com que o sistema de deteção de obstáculos responda da melhor maneira possível.

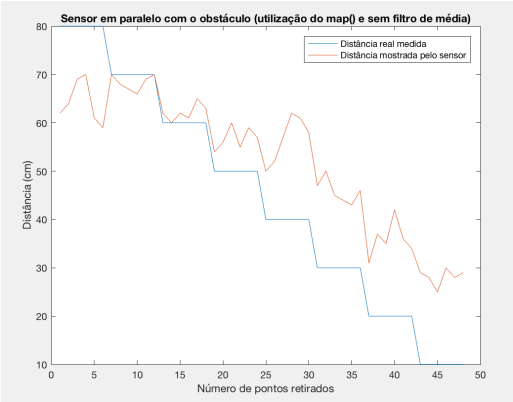


(a) Resultados com utilização da função de potência sem filtro de média

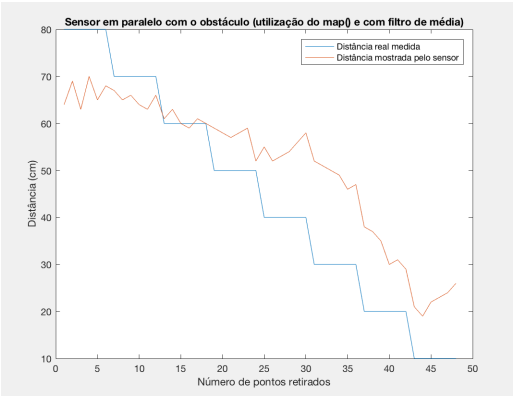


(b) Resultados com utilização da função de potência com filtro de média

Figura 5.1: Resultados com utilização da função de potência com e sem filtro de média



(a) Resultados com utilização da função map sem filtro de média



(b) Resultados com utilização da função map com filtro de média

Figura 5.2: Resultados com utilização da função map com e sem filtro de média

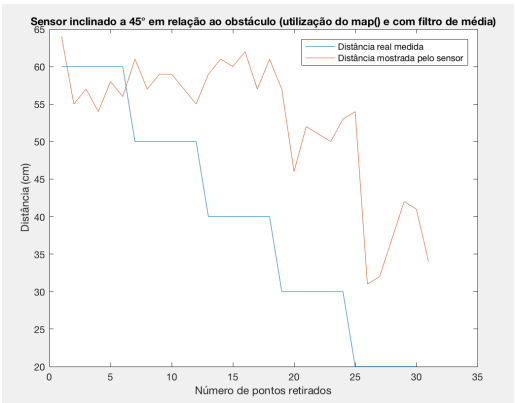
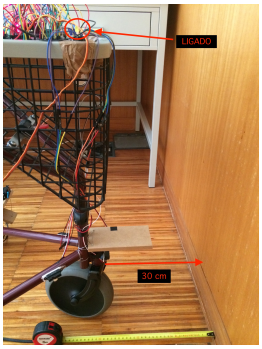


Figura 5.3: Resultados com utilização da função map com filtro de média com o sensor a uma inclinação de 45°

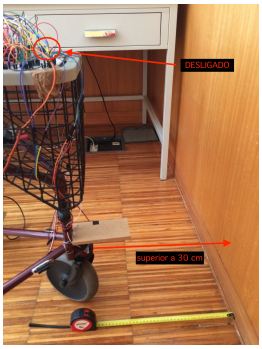
A figura 5.4 apresenta alguns exemplos de testes efetuados aos sensores de distância apresentados sob a forma de imagem, junto com o *output* que é observado no monitor série do Arduino. O nome dos sensores no monitor série do Arduino são: sensores traseiros (RIGHT e LEFT), sensores das lateriais (SRIGHT e SLEFT), sensor frontal (FRONTAL) e o sensor que aponta para o chão (GROUND).



(a) Imagem do sensor frontal a detetar um obstáculo.

```
RIGHT: 13
LEFT: 14
SRIGHT: 80
SLEFT: 80
FRONTAL: 30
GROUND: 25
```

(b) Imagem dos valores mostrados quando só o sensor frontal deteta um obstáculo.



(c) Imagem do sensor frontal quando não deteta nenhum obstáculo.

```
RIGHT: 9
LEFT: 12
SRIGHT: 80
SLEFT: 80
FRONTAL: 40
GROUND: 25
```

(d) Imagem dos valores mostrados quando não deteta nenhum obstáculo.

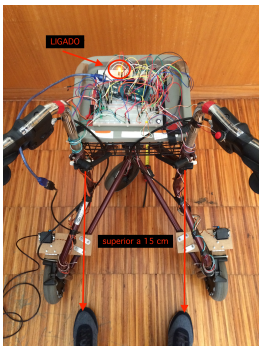
Figura 5.4: Andarilho na presença de um obstáculo frontal e os respetivos valores mostrados no monitor série do Arduino (a, b); Andarilho na ausência de qualquer obstáculo e os respetivos valores mostrados no monitor série (c, d).

Foi também feito o teste de afastar o utente do andarilho, de forma a simular uma queda por escorregamento. Este travou sempre que a distância de segurança entre o utente e o andarilho foi quebrada e o sistema de alarme ativou sempre, figura 5.5.

(a) Imagem dos sensores traseiros quando detetam o utente (distância de segurança cumprida).

```
RIGHT: 9
LEFT: 12
SRIGHT: 80
SLEFT: 80
FRONTAL: 40
GROUND: 25
```

(b) Imagem dos valores mostrados quando o sensores traseiros detetam o utente.



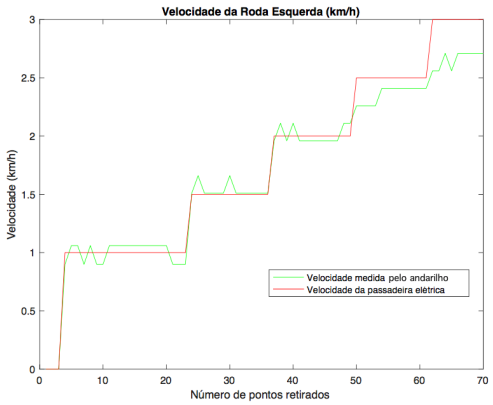
(c) Imagem dos sensores traseiros quando não detetam o utente (distância de segurança quebrada).

```
RIGHT: 50
LEFT: 50
SRIGHT: 80
SLEFT: 80
FRONTAL: 40
GROUND: 25
```

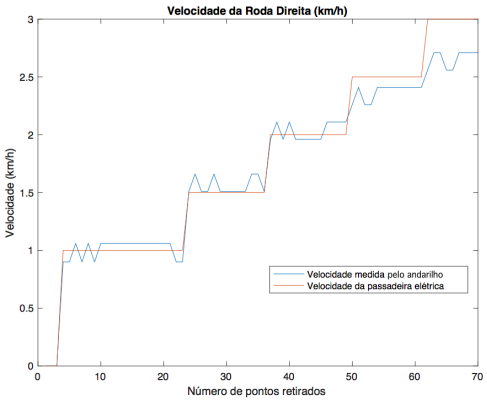
(d) Imagem dos valores mostrados quando o sensores traseiros não detetam o utente.

Figura 5.5: Andarilho quando deteta o utente e os respetivos valores mostrados no monitor série do Arduino (a, b); Andarilho quando não deteta o utente e os respetivos valores mostrados no monitor série do Arduino (c, d).

No teste de desativação dos sistemas de travagem e de alarme através do botão de pressão, os resultados foram sempre positivos, pois sempre que o botão estava pressionado, nem os travões nem o buzzer se ativavam enquanto o obstáculo estava a ser detetado. Desligando a distância de segurança entre o utente e o andarilho, foi feito o teste de empurrar o andarilho em direção à parede e este travou assim que detetou o obstáculo (a parede), não embatendo contra ele. Relativamente a testes de velocidade, para medir se realmente a velocidade em km/h que o sistema de deteção de movimento mede é preciso ou não, foram feitos testes numa passadeira eletrónica, cujos resultados se podem observar na figura 5.6.



(a) Resultados do teste da velocidade da roda esquerda.



(b) Resultados do teste da velocidade da roda direita.

Figura 5.6: Gráficos de resultados dos testes da velocidade na passadeira.

Analisando os gráficos pode chegar-se à conclusão de que os resultados são muito positivos, pois a baixas velocidades o sistema de deteção de movimento calcula de forma eficiente a velocidade, ao contrário de a velocidade superiores a 2 km/h que introduz erros na ordem de 0.5 km/h. Estes erros podem ocorrer devido à derrapagem da roda do andarilho na passadeira eletrónica, pelo que a aderência não era a mais favorável. Mas, para o tipo de aplicação e para o final deste dispositivo a gama de velocidades adequa-se perfeitamente, pois nenhuma pessoa em reabilitação vai ultrapassar o patamar de 1 km/h. Para testar se realmente a velocidade limite de segurança (1 km/h por defeito) estava a funcionar, foi efetuado o teste de correr com o andarilho enquanto se visualizava a medição da velocidade no monitor série do Arduino. Sempre que essa velocidade de segurança era ultrapassada o sistema de travagem e de alarme eram acionados.

No caso de testes ao sistema de medição da força e sentido de rotação das rodas (pertencente ao sistema de deteção de movimento), foram sempre efetuados vendo a resposta e os resultados no monitor série do Arduino. No caso do sentido de rotação sempre que

se move o andarilho para a frente a mensagem *RODA ESQUERDA FRENTE* e *RODA DIREITA FRENTE* aparece sem qualquer problema. O mesmo acontece no sentido oposto, mas com as mensagens *RODA ESQUERDA TRAS* e *RODA DIREITA TRAS*. No sistema de medição de força a mensagem que aparece é *FORCA RIGHT: X* e *FORCA LEFT: X*, onde *FORCA RIGHT* é o sensor do punho direito e *FORCA LEFT* é o sensor do punho esquerdo, e o valor que aparece à frente, neste caso designado por *X* de cada uma destas mensagens, é o valor da força exercida nos respetivos sensores em libras.

Todos os testes efetuados foram feitos por pessoas sem deficiência motora, infelizmente não houve oportunidade de testar este equipamento com pessoas que realmente necessitassem dele.

6

Conclusões e Trabalho Futuro

Ao escolher este projeto como dissertação de mestrado, o objetivo central foi sempre fazer o melhor de forma a ajudar as pessoas que necessitem deste equipamento (andarilho) para se deslocar e realizar as suas atividades diárias.

Numa fase inicial, foi bastante complicado perceber quais deveriam ser os primeiros passos a dar, por forma a evoluir na conceção de um equipamento melhor, e que de facto, trouxesse algo de útil para essas pessoas. Porém, todos os conselhos ouvidos e todas as pesquisas efetuadas deram frutos. Foi graças a toda essa informação e trabalho, que no final levou a que este magnifico projeto ficasse concluído.

De facto, utilizar um sistema de desenvolvimento como o Arduino é um aspeto bastante positivo, pois além da sua simplicidade, baixo custo e versatilidade, é utilizado em muitas soluções distintas e foi, sem dúvida, o grande ”cérebro”deste projeto.

Relativamente ao sistema de travagem automática, no início foi complicado concetualizar a sua criação. Foi de longe o mais trabalhoso mas certamente o que deu menos problemas, visto que todos os testes deram os resultados esperados. Pelo contrário, o sistema de deteção de obstáculos, foi o menos trabalhoso a nível de montagem, contudo foi o que deu mais conflitos devido à natureza dos sensores de distância que causaram alguns problemas de precisão. Apesar disso, no final, depois de várias horas de calibração e de ajustes, tudo correu como esperado.

O sistema de movimento desenvolvido para controlar a velocidade e o sentido da marcha, usando sensores de efeito Hall convertendo-o assim num funcionamento de um encoder rotacional, foi um enorme sucesso a nível de projeto.

Para colmatar, todos os sistemas criados e implementados neste equipamento de reabilitação motora, trazem uma mais valia e um enorme benefício ao utente, e por isso é uma enorme satisfação saber que todos os objetivos foram cumpridos. De um modo geral, na minha perspetiva, o projeto foi bem conseguido e cumpre os requisitos exigidos.

6.1 Trabalho Futuro

Para trabalho futuro é relevante considerar:

- Criação de um sistema de aviso externo, isto é, em caso de acidente o sistema enviar uma mensagem por email e/ou sms, ao cuidador do utente de reabilitação para o alertar que houve um acidente;
- Aplicar sensores de força no sistema de travagem automático, para se obter um *feedback* relativamente à força de travagem aplicada na roda;
- Desenvolver um sistema de travagem ABS para o andarilho;
- Aplicar um sensor ótico a apontar para o chão para que, desta forma, se consiga perceber se o andarilho se continua a deslocar no caso de as rodas derraparem enquanto estão bloqueadas;
- Adaptar sensores de inclinação ao andarilho para posteriormente funcionar em conjunto com o sistema de ABS, por exemplo, para o caso de o utente descer uma rua com uma inclinação muito acentuada, ou até mesmo saber se ocorreu uma queda;
- Utilizar um sistema para controlar o movimento do utente, imaginemos que o cuidador do utente não quer que ele saia de uma determinada zona, poderia por GPS controlar os seus movimentos. Assim caso o utente saia da zona de segurança é emitido um alerta ao seu cuidador;
- Otimizar o nível de apresentação/apelo visual exterior do equipamento. Neste momento, como foi finalizada a primeira fase de desenvolvimento do projeto não foi tido em conta o aspeto visual exterior, sendo assim gostaria de propor melhorias nesse sentido;
- Desenvolver uma interface gráfica (ANDROID, IOS) apelativa para o utente ou para o profissional que esteja a trabalhar com a pessoa em reabilitação;
- Implementar luzes de presença e iluminação noturna no andarilho também trariam algumas vantagens;
- Por último, substituir os sensores de distância por uns mais precisos, como por exemplo, sensores laser; ou tentar resolver os problemas de imprecisão e de ruído existentes, calibrando novamente os sensores implementados;

Referências

[1] R. P. Adriana and G. S. Miguel Filipe, "Andarilho de reabilitação – ANDROID," Sep. 2016.

[2] J. F. B. Barreira, "Deteção das pernas e controlo do andarilho recorrendo a um sensor laser range finder," Master's thesis, Universidade do Minho, Oct. 2014.

[3] L. F. B. da Silva, "Projeto, desenvolvimento e implementação de um "guiador" num andarilho motorizado," Master's thesis, Universidade do Minho, Oct. 2012.

[4] A. C. A. F. Ferreira, "A influência da idade nos parâmetros metabólicos durante a marcha com andarilho," 2013.

[5] C. Barbosa and I. Brandão, "Andarilho," 2014.

[6] "Smart walker: A tool for promoting mobility in elderly," 2010.

[7] R. C. J. L. P. A. A. R. R. A. Frizera, "The smart walkers as geriatric assistive device. the simbiosis purpose," 2008.

[8] T. B. A. F. R. C. C. Valadão, "Towards a smart walker controller for physiotherapy and rehabilitation purposes," 2014.

[9] (2017) What is arduino? [Online]. Available: https://www.arduino.cc/en/Guide/Introduction

[10] (2017) Arduino mega 2560 rev3. [Online]. Available: https://store.arduino.cc/arduino-mega-2560-rev3

[11] (2017) Servo motor: Veja como funciona e quais os tipos. [Online]. Available: https://www.citisystems.com.br/servo-motor/

[12] (2017) Hitec hsr-5995tg - ultra torque robot servo. [Online]. Available: https://servodatabase.com/servo/hitec/hsr-5995tg

[13] (2017) Continuous-time switch family a1101, a1102, a1103, a1104, and a1106. [Online]. Available: http://www.sycelectronica.com.ar/semiconductores/A110X.pdf

[14] (2017) Distance measuring sensor unit measuring distance: 10 to 80 cm. [Online]. Available: http://www.farnell.com/datasheets/1657845.pdf

[15] (2017) Piezo buzzer. [Online]. Available: http://www.farnell.com/datasheets/1662133.pdf

[16] (2017) Ripa aplainada 21x21x2400 mm. [Online]. Available: http://www.leroymerlin.pt/Site/Produtos/Madeiras/Ripas/Aplainadas/14125601.aspx

[17] (2017) Abraçadeira u aço inox 4x28mm. [Online]. Available: http://www.leroymerlin.pt/Site/Produtos/Ferragens/Ferragens-de-fixacao/Cordas/18607092.aspx

[18] (2017) 2 dobradiças largas zincado 25x140mm. [Online]. Available: http://www.leroymerlin.pt/Site/Produtos/Ferragens/Ferragens-para-moveis/Dobradicas-de-movel/14939505.aspx

[19] (2017) 6 cerra-cabos aço 1.5/1.8mm. [Online]. Available: http://www.leroymerlin.pt/Site/Produtos/Ferragens/Ferragens-de-fixacao/Cordas/18607344.aspx

[20] (2017) Mola de pressão 1.8x100mm. [Online]. Available: http://www.leroymerlin.pt/Site/Produtos/Ferragens/Ferragens-de-fixacao/Molas/18613420.aspx

[21] (2017) Servo library. [Online]. Available: https://www.arduino.cc/en/Reference/Servo

[22] (2017) map(value, fromlow, fromhigh, tolow, tohigh). [Online]. Available: https://www.arduino.cc/en/Reference/Map

[23] (2017) attachinterrupt(). [Online]. Available: https://www.arduino.cc/en/Reference/AttachInterrupt

[24] (2017) tone(). [Online]. Available: https://www.arduino.cc/en/Reference/Tone

7

Apêndice

Sistema de medição de força

Sensores de Força

Neste projeto foram utilizados dois sensores de força FlexyForce para medir a força que o utente faz em cada punho do andarilho.

Os FlexyForce são sensores de força que utilizam uma tecnologia resistiva. Através da pressão, ou seja, se aplicarmos um determinada força na área ativa do sensor (círculo de 9,525 mm de diâmetro, que se encontra na extremidade do sensor) este vai alterar o valor da resistência lido, esta alteração é inversamente proporcional à força aplicada.

Estes sensores são resistentes a qualquer ambiente e a força é medida entre quase todas as superfícies a que são aplicados. Os sensores são extremamente manejáveis e muito finos, figura 7.1, podendo ser utilizados e adaptados às mais variáveis aplicações.

Os sensores são constituídos por duas camadas de substrato. O substrato é composto por uma película de poliéster (ou poliamida no caso de sensores de temperatura elevada).

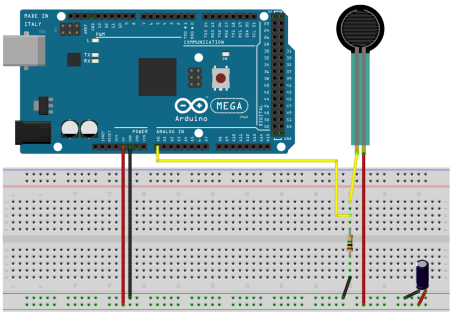
Em cada camada é aplicado um material condutor (prata), seguido de uma camada de tinta sensível à pressão. O círculo de prata em cima da tinta sensível à pressão define a área de sensibilidade ativa. A prata estende-se desde a área de sensibilidade até aos conetores na outra extremidade do sensor, formando os fios condutores [1].

Figura 7.1: Sensor de Força FlexyForce.

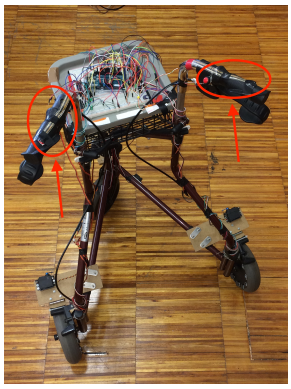
Hardware

Para determinar a força nos punhos do andarilho este sistema utiliza dois sensores de medição de força FlexiForce (um em cada punho do guiador), figura 7.2b.

A ligação com o Arduino é feita como está demonstrado na figura 7.2a, onde o sensor de força do punho direito se encontra ligado no pino A0 e o sensor de força esquerdo se encontra ligado no pino A1.



(a) Esquema de ligação dos sensores de força.



(b) Imagem dos sensores de força nos punhos do andarilho.

Figura 7.2: Sensores de força do andarilho.

Partindo do princípio que a tensão de alimentação é de $5V$, sabemos também o valor de tensão de saída lido no pino A0 e A1 e o valor da resistência é de $1M\Omega$, com o recurso a um divisor de tensão, podemos calcular a resistência do sensor, utilizando a fórmula 7.1 [1].

$$ResistenciaSensor = \frac{ResistenciaFixa \times (Vtotal - Vout)}{Vout} \tag{7.1}$$

Sabendo que o $Vtotal$ é o valor de tensão de alimentação e o $Vout$ é o valor de tensão de saída, podemos afirmar que conhecemos a resistência, portanto é necessário a determinação da condutância. Recorrendo a gráficos presentes no datasheet dos sensores observamos que, a relação entre a força e a condutância se traduz numa variação linear, ao contrário do que se verifica entre a força e a resistência, como mostram os gráficos da figura 7.3 [1].

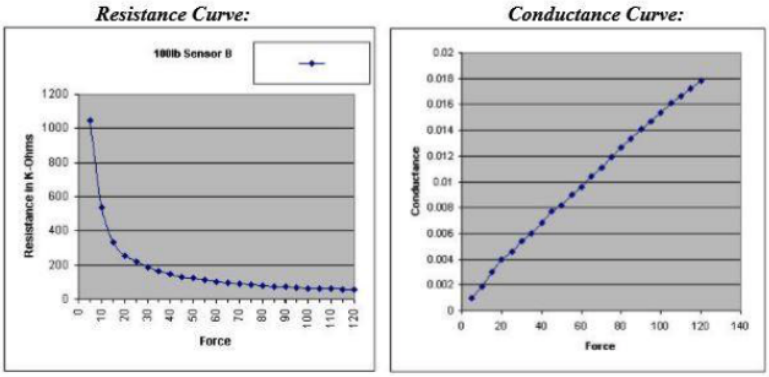


Figura 7.3: Relação entre a força e a resistência à esquerda e Relação entre a força e a condutância à direita.

Portanto, a condutância pode ser determinada com a equação 7.2 [1].

$$CondutanciaSensor = \frac{ResistenciaFixa}{ResistenciaSensor} \tag{7.2}$$

Onde a *ResistenciaSensor* foi calculada pela equação 7.1. E finalmente, para obtermos o valor da força recorremos ao valor da curva de condutância do gráfico, onde podemos determinar a equação 7.3 [1].

$$Forca(lbs) = \frac{CondutanciaSensor - 0,0012}{0,00014} \tag{7.3}$$

Onde $Forca(lbs)$ é a força obtida em libras (lbs) e a *CondutanciaSensor* é a condutância calculada na equação 7.2 [1].

Software

O *software* implementado para o funcionamento destes sensores está agrupado na função seguinte:

```
int GetForce(int pin) {
  int SensorReading;        // Variavel para guardar a leitura em int do canal analogico
  double SensorVoltage;     // Variavel para guardar o valor da voltagem do sensor
  double SensorResistance;  // Variavel para guardar o valor da resistencia do sensor
  double SensorConductance; // Variavel para guardar o valor da condutancia do sensor
  double ForceLbs;          // Variavel para guardar o valor da força lida pelo sensor
  double ForcePercent;

 SensorReading = MeanFilter(pin, 5); // recebe o valor calculado atraves do filtro de media com 5 amostras
 SensorVoltage = int(map(SensorReading,0,1023,0,5000)); // Passa o valor lido para tensao (mV).

  if (SensorVoltage < 0.0001) {  // caso o valor de voltagem lido seja inferior a 0.0001 mV
    ForceLbs=0; // Forca e devolvida como 0.
    }
    else{  // caso contrario
      // Calcula a resistencia do sensor
      // Rsensor = (Rfixed*(Vtotal-Vout))/Vout
      SensorResistance = (1000000*(5000-SensorVoltage))/SensorVoltage;

      // Calcula a condutancia do sensor
      // C = 1/R(ohms)=1000000/R(mohms)
      SensorConductance = 1000000/SensorResistance;

      // Calcula a forca em lbs.
      ForceLbs = (SensorConductance-0.0012)/0.00014;
```

```
26        // Aplicacao do modelo de regressao linear para obter o valor da forca
27        //ForceLbs = 5.3157*SensorConductance;
28        }
29  // retorna o valor da força
30  return ((int)((ForceLbs*100)/50000));
31  }
```

Toda a função traduz a explicação que foi dada para linguagem de programação.

Código Desenvolvido

Aqui se apresenta todo o código desenvolvido no Arduino IDE baseado em C/C++, devidamente estruturado e comentado.

```
1
2   #include <Servo.h> // incluir a biblioteca para os servos motores
3   /*---------- Definiçes Globais/Iniciais ----------------------------------*/
4   #define SENSOR_FORCE_RIGHT 0       // Pin analogico de entrada sensor de força ↩
        direito
5   #define SENSOR_FORCE_LEFT 1        // Pin analogico de entrada sensor de força ↩
        esquerdo
6   #define SENSOR_DISTANCE_LEFT 2     // Pin analogico de entrada sensor de distancia ↩
        perna esquerda
7   #define SENSOR_DISTANCE_RIGHT 3    // Pin analogico de entrada sensor de distancia ↩
        perna direita
8   #define SENSOR_DISTANCE_SRIGHT 4   // Pin analogico de entrada sensor de distancia ↩
        LATERAL direita
9   #define SENSOR_DISTANCE_SLEFT 5    // Pin analogico de entrada sensor de distancia ↩
        LATERAL esquerda
10  #define SENSOR_DISTANCE_FRONTAL 6 // Pin analogico de entrada sensor de distancia ↩
        FRONTAL
11  #define SENSOR_DISTANCE_GROUND 7   // Pin analogico de entrada sensor de distancia ↩
        CHAO
12
13  int DISTANCE_LEFT;     // sensor de distancia perna esquerda
14  int DISTANCE_RIGHT;    // sensor de distancia perna direita
15  int DISTANCE_SLEFT;    // sensor de distancia lateral esquerda
16  int DISTANCE_SRIGHT;   // sensor de distancia lateral direita
17  int DISTANCE_FRONTAL; // sensor de distancia frontal
18  int DISTANCE_GROUND;   // sensor de distancia que aponta para o chao
19
20  int FORCE_LEFT;    // sensor de força punho esquerdo
21  int FORCE_RIGHT; //  sensor de força punho direito
22
23  // LEDS e HALL SENSOR ENCODERS
24  const int LEDPin_ESQ_FRENTE = 8; // LED sensor de Hall roda esquerda frente
25  const int LEDPin_ESQ_TRAS = 7;   // LED sensor de Hall roda esquerda tras
26  const int LEDPin_DIR_TRAS = 6;   // LED sensor de Hall roda direita tras
27  const int LEDPin_DIR_FRENTE = 9; // LED sensor de Hall roda direita frente
28  const int SHALL_ESQ_FRENTE = 2;  // sensor de Hall esquerda FRENTE (frente LED ↩
        VERMELHO)
29  const int SHALL_ESQ_TRAS = 5;    // sensor de Hall esquerda TRAS (tras LED VERDE)
30  const int SHALL_DIR_TRAS = 4;    // sensor de Hall direita TRAS (tras LED VERDE)
31  const int SHALL_DIR_FRENTE = 3;  // sensor de Hall direita FRENTE (frente LED ↩
        VERMELHO)
32
33  const int LEDobstaculo = 11;     // LED que acende se detectar obstaculos
34
35  /*------ VARIAVEIS VELOCIDADE ---------------------------------------------*/
36  // Roda esquerda
37  int rpm_esq;                  // variavel de rotaçoes por minuto da roda esquerda
```

```
38 float veloc_esq;              // variavel de velocidade (km/h) da roda esquerda
39 volatile byte pulsos_esq;     // variavel que guarda o numero de pulsos contados na↩
       roda esquerda
40 unsigned long timeold_esq;    // variavel que guarda o tempo de rotacao para ↩
       actualizar as contagens da roda esquerda
41
42 // Roda direita
43 int rpm_dir;                  // variavel de rotaçoes por minuto da roda esquerda
44 float veloc_dir;              // variavel de rotaçoes por minuto da roda esquerda
45 volatile byte pulsos_dir;     // variavel que guarda o numero de pulsos contados na↩
       roda direita
46 unsigned long timeold_dir;    // variavel que guarda o tempo de rotacao para ↩
       actualizar as contagens da roda direita
47
48 float veloc_maxima = 1;      // 1000 m/h = 1 km/h - velocidade lenta de seguranca
49
50 // Numero de pulsos por volta do sensor de Hall
51 unsigned int pulsos_por_volta = 12; // numero de imans nas rodas
52
53 /*------ VARIAVEIS SENTIDO ------------------------------------------------------*/
54 volatile bool pronto_esq;      // variavel de interrupçao para saber se o sentido ↩
       esta definido
55 volatile bool sentido_esq;    // variavel que define o sentido da roda esquerda, se↩
       1 frente se 0 tras
56 volatile bool pronto_dir;     // variavel de interrupçao para saber se o sentido ↩
       esta definido
57 volatile bool sentido_dir;    // variavel que define o sentido da roda direita, se ↩
       1 frente se 0 tras
58 const byte encoderPinA = 18;  // sensor de Hall esquerda FRENTE (interrupçao 5) ↩
       para o sentido
59 const byte encoderPinB = 5;   // sensor de Hall esquerda TRAS
60 const byte encoderPinC = 19;  // sensor de Hall direita FRENTE (interrupçao 4) para↩
       o sentido
61 const byte encoderPinD = 4;   // sensor de Hall direita TRAS
62 static long rotacao_esq;      // variavel para contagem de rotaçoes para o sentido ↩
       (24 pulsos) roda esquerda
63 static long rotacao_dir;      // variavel para contagem de rotaçoes para o sentido ↩
       (24 pulsos) roda direita
64
65 /*------ SERVOS -----------------------------------------------------------------*/
66 Servo myservoRIGHT;    // cria o objecto do servo motor direito
67 Servo myservoLEFT;     // cria o objecto do servo motor esquerdo
68
69 /*------ BOTAO DE PRESSAO -------------------------------------------------------*/
70 const int PushButton = 22;  // define o PIN de entrada do botao de pressao (22)
71 const int LEDButton = 10;   // define o PIN de entrada do LED do botao (10)
72 int ButtonState = 0;        // Inicializa o estado do botço
73
74 /*------ BUZZER -----------------------------------------------------------------*/
75 int BuzzerPin = 23;   // define o PIN de entrada do Buzzer (23)
76
77 ///*----- MAIN SETUP -------------------------------------------------------------*/
78 void setup() {
79   Serial.begin(9600);
80
81   // Define o pin de entrada dos servos
82   myservoRIGHT.attach(25);
83   myservoLEFT.attach(24);
84
85   // Define o valor (entrada ou saida) do button e do LED
86   pinMode(PushButton, INPUT);
87   pinMode(LEDButton, OUTPUT);
88
89   // Define o pin do Buzzer como saida
90   pinMode(BuzzerPin, OUTPUT);
91
92   // Define os PINs dos LEDs dos HALL SENSORs como saidas
93   pinMode(LEDPin_ESQ_FRENTE, OUTPUT);
94   pinMode(LEDPin_ESQ_TRAS, OUTPUT);
95   pinMode(LEDPin_DIR_TRAS, OUTPUT);
```

```
96    pinMode(LEDPin_DIR_FRENTE, OUTPUT);
97
98    pinMode(LEDobstaculo, OUTPUT); // define a porta do LED que acende quando detecta↩
          obstaculos como saida
99
100   // velocidade, define os PINs dos HALL SENSORs como entradas
101   pinMode(SHALL_ESQ_FRENTE,INPUT_PULLUP);    // Interrupcao 0 - pino digital 2 ↩
          ESQUERDA FRENTE
102   pinMode(SHALL_DIR_FRENTE,INPUT_PULLUP);    // Interrupcao 1 - pino digital 3 ↩
          DIREITA FRENTE
103   pinMode(SHALL_ESQ_TRAS,INPUT);             // Entradas normais (sem interrupçao) ↩
          ESQUERDA TRAS
104   pinMode(SHALL_DIR_TRAS,INPUT);             // Entradas normais (sem interrupçao) ↩
          DIREITA TRAS
105
106   //Aciona o contador a cada pulso
107   attachInterrupt(digitalPinToInterrupt (SHALL_ESQ_FRENTE), contador_esq, FALLING);↩
          // interrupt 0 FALLING for when the pin goes from high to low.
108   attachInterrupt(digitalPinToInterrupt (SHALL_DIR_FRENTE), contador_dir, FALLING);↩
          // interrupt 1 FALLING for when the pin goes from high to low.
109   pulsos_esq = 0;   // inicializa o numero de pulsos contados da roda esquerda a 0
110   pulsos_dir = 0;   // inicializa o numero de pulsos contados da roda direita a 0
111   rpm_esq = 0;      // inicializa o numero de rotaçoes por minuto a 0 da roda ↩
          esquerda
112   rpm_dir = 0;      // inicializa o numero de rotaçoes por minuto a 0 da roda ↩
          direita
113   timeold_esq = 0;  // inicializa o tempo de rotaçao da roda esquerda a 0
114   timeold_dir = 0;  // inicializa o tempo de rotaçao da roda direita a 0
115
116
117   // SENTIDO
118   rotacao_esq = 0;  // inicializa a 0 a contagem de rotaçoes para o sentido (24 ↩
          pulsos) roda esquerda
119   rotacao_dir = 0;  // inicializa a 0 a contagem de rotaçoes para o sentido (24 ↩
          pulsos) roda direita
120
121   // Sentido, define os PINs dos HALL SENSORs como entradas
122   pinMode (encoderPinA, INPUT_PULLUP);      // Interrupcao 5 - pino digital 18 ↩
          ESQUERDA FRENTE
123   pinMode (encoderPinB, INPUT_PULLUP);
124   pinMode (encoderPinC, INPUT_PULLUP);      // Interrupcao 4 - pino digital 19 ↩
          DIREITA FRENTE
125   pinMode (encoderPinD, INPUT_PULLUP);
126   attachInterrupt (digitalPinToInterrupt (encoderPinA), interrupt_esq, CHANGE); // ↩
          interrupt 5 CHANGE to trigger the interrupt whenever the pin changes value
127   attachInterrupt (digitalPinToInterrupt (encoderPinC), interrupt_dir, CHANGE); // ↩
          interrupt 4 CHANGE to trigger the interrupt whenever the pin changes value
128
129   // le botao no arranque para atribuir velocidade maxima
130   ButtonState = digitalRead(PushButton); // verifica o estado do botao quando ↩
          inicia
131
132   if(ButtonState == HIGH){ // botao carregado, passa a velocidade rapida
133     veloc_maxima = 2;  // 2 km/h, 2000 m/h - velocidade rapida
134   }
135
136 }
137
138 ///*----- MAIN LOOP -------------------------------------------------------------*/
139 void loop() {
140
141   LEDS(1);          // LEDS dos sensores das rodas acendem (se LEDS(0) os leds ficam ↩
          sempre OFF)
142   Button();         // Chama a funcao do botao de pressço
143   velocidade();     // Chama a funcao da velocidade das rodas
144   sentido();        // Chama a funcao do sentido das rodas
145
146   //Serial.print("Velocidade Maxima: ");
147   //Serial.println(veloc_maxima);
148
```

```
149   // Sensores de Distancias
150   DISTANCE_LEFT=GetDistance_left(SENSOR_DISTANCE_LEFT);
151   DISTANCE_RIGHT=GetDistance_right(SENSOR_DISTANCE_RIGHT);
152   DISTANCE_SLEFT=GetDistance_sleft(SENSOR_DISTANCE_SLEFT);
153   DISTANCE_SRIGHT=GetDistance_sright(SENSOR_DISTANCE_SRIGHT);
154   DISTANCE_FRONTAL=GetDistance_frontal(SENSOR_DISTANCE_FRONTAL);
155   DISTANCE_GROUND=GetDistance_ground(SENSOR_DISTANCE_GROUND);
156
157   /*Serial.print("RIGHT: ");
158   Serial.println(DISTANCE_RIGHT);
159   Serial.print("LEFT: ");
160   Serial.println(DISTANCE_LEFT);
161   Serial.print("SRIGHT: ");
162   Serial.println(DISTANCE_SRIGHT);
163   Serial.print("SLEFT: ");
164   Serial.println(DISTANCE_SLEFT);
165   Serial.print("FRONTAL: ");
166   Serial.println(DISTANCE_FRONTAL);
167   Serial.print("GROUND: ");
168   Serial.println(DISTANCE_GROUND);*/
169
170   // Sensores de Forca
171   FORCE_LEFT=GetForce(SENSOR_FORCE_LEFT);
172   FORCE_RIGHT=GetForce(SENSOR_FORCE_RIGHT);
173
174   /*Serial.print("FORCA LEFT: ");
175   Serial.println(FORCE_LEFT);
176   Serial.print("FORCA RIGHT: ");
177   Serial.println(FORCE_RIGHT);*/
178
179   // Condicao de funcionamento dos travoes automaticos
180   if(digitalRead(PushButton)==LOW){                  // se o botao de pressao nao ↩
          estiver a ser pressionado
181     if((DISTANCE_RIGHT>= 15 && DISTANCE_LEFT>= 15) || (DISTANCE_SLEFT<= 30) || (↩
          DISTANCE_SRIGHT<= 30) || (DISTANCE_FRONTAL<= 30) || (DISTANCE_GROUND >= 30)↩
          ){
182       servos(80,100);                              // os servos travam
183       digitalWrite(LEDobstaculo, HIGH);            // LED dos obstaculos acende
184       BuzzerON();                                  // Buzzer toca (activado ON)
185     } else {                                       // caso contrario
186       BuzzerOFF();                                 // o Buzzer nao toca (desligado↩
              OFF)
187       servos(110,80);                              // Posiçço inicial dos servos (↩
              nao trava)
188       digitalWrite(LEDobstaculo, LOW);             // LED dos obstaculos apagado
189      }
190   }
191
192 }
193
194 /*------ Funcao SERVOS ------------------------------------------------------*/
195 void servos(int postravagemdir, int postravagemesq) { // parametros de entrada, ↩
      angulo de travagem (roda direita, roda esquerda)
196    myservoLEFT.write(postravagemesq);                  // transmite o angulo ↩
         definido para o servo esquerdo
197    myservoRIGHT.write(postravagemdir);                 // transmite o angulo ↩
         definido para o servo direito
198 }
199
200 /*------ Funcao Button ------------------------------------------------------*/
201 void Button() {                            // funcao que controla o botao de pressao
202   ButtonState = digitalRead(PushButton);   // le o estado do botao
203
204   if(ButtonState == HIGH){           // se o botao for pressionado
205     digitalWrite(LEDButton, HIGH);   // LED acende (AZUL)
206     servos(110,80);                  // os servo motores mudam a posicao (nao ↩
          travam)
207     BuzzerOFF();                     // Buzzer desligado
208     digitalWrite(LEDobstaculo, LOW); // LED dos obstaculos apagado
209     }
```

```
210   else{                                 // caso contrario
211     digitalWrite(LEDButton, LOW);       // LED do botao apagado
212     }
213 }
214
215 /*------ Funcao Buzzer ON ------------------------------------------------------------*/
216 void BuzzerON() {           // liga o buzzer (barulho de alarme)
217   tone(BuzzerPin, 440);     // utilizaçço da funcao tone da biblioteca do arduino ↩
        (440Hz nota A)
218 }
219
220 /*------ Funcao Buzzer OFF -----------------------------------------------------------*/
221 void BuzzerOFF() {     // desliga o buzzer (barulho de alarme)
222   noTone(BuzzerPin);   // utilizacao da funcao noTone da biblioteca do arduino
223 }
224
225 /*------ Funcao do filtro da media ↩
      ------------------------------------------------------------------------*/
226 float MeanFilter(int pin, int numSamples) { // parametros de entrada PIN de entrada↩
       do sensor e numero de amostras
227   int sum=0;                                // inicializacao do soma
228
229   for(int i=0;i<numSamples;i++){    // ciclo para fazer a soma de todas as amostras
230     sum+=analogRead(pin);           // faz o soma de todas as leituras da entrada ↩
          analogica pin
231   }
232  return ((int) (sum/numSamples));  // devolve a media da soma de todas as amostras
233 }
234
235 /*------ funcao do sensor de distancia perna esquerda ↩
      ------------------------------------------------------------*/
236 int GetDistance_left(int pin) {                 // funcao para definir a distancia ↩
     do sensor da perna esquerda
237 // int sensorValue = analogRead(pin);           // guarda a leitura do pin ↩
     analogico na variavel sensorValue
238 int sensorValue=MeanFilter(pin, 30);            // guarda a media calculada com 30 ↩
     amostras da leitura do pin analogico do sensor
239 float SensorOut = sensorValue * (5.0 / 1024.0); // Converte o valor lido o ADC↩
     (0-1023) para voltagem (0 - 5V)
240 float dist_cm = 11.665 * pow(SensorOut, -0.754);// regressao de potencia distancia
241   if(dist_cm>50) dist_cm=50;                    // caso a distancia for maior que ↩
        50cm devolve 50cm
242   if(dist_cm<0) dist_cm=0;                      // caso a distancia for menor que 0↩
        cm devolve 0cm
243 return ((int)dist_cm);                         // retorna a distancia em ↩
     centimetros
244 }
245
246 /*------ funcao do sensor de distancia perna direita ↩
      ------------------------------------------------------------*/
247 int GetDistance_right(int pin) {                // funcao para definir a distancia ↩
     do sensor da perna direita
248 // int sensorValue = analogRead(pin);           // guarda a leitura do pin ↩
     analogico na variavel sensorValue
249 int sensorValue=MeanFilter(pin, 30);            // guarda a media calculada com 30 ↩
     amostras da leitura do pin analogico do sensor
250 float SensorOut = sensorValue * (5.0 / 1024.0); // Converte o valor lido o ADC↩
     (0-1023) para voltagem (0 - 5V)
251 float dist_cm = 12.364 * pow(SensorOut, -0.877);// regressao de potencia distancia
252     if(dist_cm>50) dist_cm=50;                  // caso a distancia for maior que ↩
          50cm devolve 50cm
253     if(dist_cm<0) dist_cm=0;                    // caso a distancia for menor que 0↩
          cm devolve 0cm
254 return ((int)dist_cm);                         // retorna a distancia em ↩
     centimetros
255 }
256
257 /*------funcao sensor de distancia lateral direito ↩
      ------------------------------------------------------------*/
258 int GetDistance_sright(int pin) {                // funcao para definir a ↩
```

```
    distancia do sensor da lateral direita
259 //int sensorValue = analogRead(pin);              // guarda a leitura do pin ↩
    analogico na variavel sensorValue
260 int sensorValue=MeanFilter(pin, 30);              // guarda a media calculada com ↩
    30 amostras da leitura do pin analogico do sensor
261 //float SensorOut = sensorValue * (5.0 / 1024.0); // Converte o valor lido o ADC↩
    (0-1023) para voltagem (0 - 5V)
262 float dist_cm = map(sensorValue, 10, 650, 80, 10);// mapeia a voltagem guardada no ↩
    sensorValue e converte em cm entre 10 e 80cm
263 //float dist_cm = 29.982 * pow(SensorOut, -1.325);// regressao de potencia ↩
    distancia
264     if(dist_cm>=80) dist_cm=80;                   // caso a distancia for maior que↩
        80cm devolve 80cm
265     if(dist_cm<10) dist_cm=10;                    // caso a distancia for menor que↩
        10cm devolve 10cm
266 return ((int)dist_cm);                           // retorna a distancia em ↩
    centimetros
267 }
268
269 /*-----funcao sensor de distancia lateral esquerda ↩
    ------------------------------------------------------*/
270 int GetDistance_sleft(int pin) {                 // funcao para definir a distancia ↩
    do sensor da lateral esquerda
271 //int sensorValue = analogRead(pin);              // guarda a leitura do pin ↩
    analogico na variavel sensorValue
272 int sensorValue=MeanFilter(pin, 30);              // guarda a media calculada com ↩
    30 amostras da leitura do pin analogico do sensor
273 //float SensorOut = sensorValue * (5.0 / 1024.0); // Converte o valor lido o ADC↩
    (0-1023) para voltagem (0 - 5V)
274 float dist_cm = map(sensorValue, 10, 700, 80, 10);// mapeia a voltagem guardada no ↩
    sensorValue e converte em cm entre 10 e 80cm
275 //float dist_cm = 36.612 * pow(SensorOut, -1.314);// regressao de potencia ↩
    distancia
276     if(dist_cm>=80) dist_cm=80;                   // caso a distancia for maior que↩
        80cm devolve 80cm
277     if(dist_cm<10) dist_cm=10;                    // caso a distancia for menor que↩
        10cm devolve 10cm
278 return ((int)dist_cm);                           // retorna a distancia em ↩
    centimetros
279 }
280
281 /*-----funcao sensor de distancia frontal ↩
    ------------------------------------------------------*/
282 int GetDistance_frontal(int pin) {                 // funcao para definir a ↩
    distancia do sensor frontal
283 //int sensorValue = analogRead(pin);              // guarda a leitura do pin ↩
    analogico na variavel sensorValue
284 int sensorValue=MeanFilter(pin, 30);              // guarda a media calculada com↩
     30 amostras da leitura do pin analogico do sensor
285 //float SensorOut = sensorValue * (5 / 1024.0);  // Converte o valor lido o ADC↩
    (0-1023) para voltagem (0 - 5V)
286 float dist_cm = map(sensorValue, 10, 600, 80, 10);  // mapeia a voltagem guardada ↩
    no sensorValue e converte em cm entre 10 e 80cm
287 //float dist_cm = 31.528 * pow(SensorOut, -1.209);  // regressao de potencia ↩
    distancia
288     if(dist_cm>=80) dist_cm=80;                   // caso a distancia for maior ↩
        que 80cm devolve 80cm
289     if(dist_cm<10) dist_cm=10;                    // caso a distancia for menor ↩
        que 10cm devolve 10cm
290 return ((int)dist_cm);                           // retorna a distancia em ↩
    centimetros
291 }
292
293 /*-----funcao sensor de distancia do chao ↩
    ------------------------------------------------------*/
294 int GetDistance_ground(int pin) {                 // funcao para definir a ↩
    distancia do sensor do chao
295 //int sensorValue = analogRead(pin);              // guarda a leitura do pin ↩
    analogico na variavel sensorValue
296 int sensorValue=MeanFilter(pin, 30);              // guarda a media calculada com ↩
```

```
    30 amostras da leitura do pin analogico do sensor
297 //float SensorOut = sensorValue * (5.0 / 1024.0); // Converte o valor lido o ADC↩
    (0-1023) para voltagem (0 - 5V)
298 float dist_cm = map(sensorValue, 50, 300, 50, 10);// mapeia a voltagem guardada no ↩
    sensorValue e converte em cm entre 10 e 50cm
299 //float dist_cm = 29.564 * pow(SensorOut, -1.064);// regressao de potencia ↩
    distancia
300     if(dist_cm>=50) dist_cm=50;                    // caso a distancia for maior que↩
        50cm devolve 50cm
301     if(dist_cm<10) dist_cm=10;                     // caso a distancia for menor que↩
        10cm devolve 10cm
302 return ((int)dist_cm);                            // retorna a distancia em ↩
    centimetros
303 }
304
305 /*------funcao sensores de forca dos punhos ↩
    ------------------------------------------------------------*/
306 int GetForce(int pin) {
307   int SensorReading;        // Variavel para guardar a leitura em int do canal ↩
        analogico
308   double SensorVoltage;     // Variavel para guardar o valor da voltagem do sensor
309   double SensorResistance;  // Variavel para guardar o valor da resistencia do ↩
        sensor
310   double SensorConductance; // Variavel para guardar o valor da condutancia do ↩
        sensor
311   double ForceLbs;          // Variavel para guardar o valor da força lida pelo ↩
        sensor
312   double ForcePercent;
313
314  // SensorReading = analogRead(pin); // Recebe o valor do canal analogico.
315  SensorReading = MeanFilter(pin, 5); // recebe o valor calculado atraves do filtro ↩
      de media com 5 amostras
316
317  SensorVoltage = int(map(SensorReading,0,1023,0,5000)); // Passa o valor lido para ↩
      tensço (mV).
318
319   if (SensorVoltage < 0.0001) {  // caso o valor de voltagem lido seja inferior a ↩
        0.0001 mV
320     ForceLbs=0;                  // Forca ç devolvida como 0.
321     }
322     else{                        // caso contrario
323       // Calcula a resistencia do sensor
324       // Rsensor = (Rfixed*(Vtotal-Vout))/Vout
325       SensorResistance = (1000000*(5000-SensorVoltage))/SensorVoltage;
326       //Serial.print("R (mohms) = ");
327       //Serial.println(SensorResistance);
328
329       // Calcula a condutancia do sensor
330       // C = 1/R(ohms)=1000000/R(mohms)
331       SensorConductance = 1000000/SensorResistance;
332       //Serial.print("Conductance in mohms: ");
333       //Serial.println(SensorConductance);
334
335       // Calcula a forca em lbs.
336       ForceLbs = (SensorConductance-0.0012)/0.00014;
337       // Aplicacao do modelo de regressao linear para obter o valor da forca
338       //ForceLbs = 5.3157*SensorConductance;
339       //Serial.print("Force lb = ");
340       //Serial.println(ForceLbs);
341       }
342 // retorna o valor da força
343 return ((int)((ForceLbs*100)/50000));
344 }
345
346 /*---------- Funcao para calcular os rpm e a velocidade ↩
    ----------------------------------------------------*/
347 int velocidade() {
348   // RODA ESQUERDA
349   //Atualiza contador a cada 1 segundos
350   if (millis() - timeold_esq >= 1000)
```

```
351   {
352     detachInterrupt(0);                          // Desabilita interrupcao durante o calculo
353     rpm_esq = (pulsos_esq*60)/pulsos_por_volta; // numero de (pulsos obtidos * 60) / por numero de pulsos totais
354     veloc_esq = 2*PI*0.00008*rpm_esq*60;         // velocidade = 2*PI*raio*rpm*60 (km/h)
355     timeold_esq = millis();                      // temporizador e actualizado
356     pulsos_esq = 0;                              // o contador de pulsos ç reiniciado a 0
357
358    //Mostra o valor de RPM no serial monitor
359    Serial.print("RPM_esq = ");
360    Serial.print(rpm_esq, DEC);
361    //Mostra o valor da Velocidade no serial monitor
362    Serial.print("  Velocidade (km/h) = ");
363    Serial.println(veloc_esq);
364
365     // Habilita interrupcao
366     attachInterrupt(digitalPinToInterrupt (SHALL_ESQ_FRENTE), contador_esq, FALLING); // FALLING for when the pin goes from high to low.
367   }
368
369   // RODA DIREITA
370   //Atualiza contador a cada 1 segundos
371   if (millis() - timeold_dir >= 1000)
372   {
373     detachInterrupt(1);                          // Desabilita interrupcao durante o calculo
374     rpm_dir = (pulsos_dir*60)/pulsos_por_volta; // numero de (pulsos obtidos * 60) / por numero de pulsos totais
375     veloc_dir = 2*PI*0.00008*rpm_dir*60;         // velocidade = 2*PI*raio*rpm*60 (km/h)
376     timeold_dir = millis();                      // temporizador e actualizado
377     pulsos_dir = 0;                              // o contador de pulsos ç reiniciado a 0
378
379     //Mostra o valor de RPM no serial monitor
380     Serial.print("RPM_dir = ");
381     Serial.print(rpm_dir, DEC);
382     //Mostra o valor da Velocidade no serial monitor
383     Serial.print("  Velocidade (km/h) = ");
384     Serial.println(veloc_dir);
385
386     // Habilita interrupcao
387     attachInterrupt(digitalPinToInterrupt (SHALL_DIR_FRENTE), contador_dir, FALLING); // FALLING for when the pin goes from high to low.
388   }
389
390   if(veloc_esq && veloc_dir >= veloc_maxima){ // delimita a velocidade de andamento do andarilho (velocidade de segurança)
391     servos(80,100);                            // servos travam
392     BuzzerON();                                // Buzzer toca (alarme)
393   }
394
395 }
396
397 /*---------- Contadores para os pulsos da velocidade ----------*/
398 void contador_esq(){     // cada vez que uma interrupcao e ativada esta funcao e chamada
399   //Incrementa contador
400   pulsos_esq++;          // contador para os pulsos da velocidade da roda esquerda
401 }
402
403 void contador_dir(){     // cada vez que uma interrupcao e ativada esta funcao e chamada
404   //Incrementa contador
405   pulsos_dir++;          // contador para os pulsos da velocidade da roda direita
406 }
```

```
407
408 /*------------ Funcao sentido (Hall Sensor) ↩
    ------------------------------------------------------*/
409 int sentido() {                 // funcao que permite identificar o sentido de ↩
    rotacao das rodas
410   if (pronto_esq == true) {     // assim que o sentido estiver definido
411     if (sentido_esq == HIGH) {  // se a variavel sentido for HIGH roda esta a girar↩
          para a frente
412       rotacao_esq++;
413       Serial.println (" RODA ESQUERDA FRENTE ");
414     } else {                    // caso contrario (se sentido = LOW) roda gira para↩
          tras
415       rotacao_esq--;
416       Serial.println (" RODA ESQUERDA TRAS ");
417     }
418     pronto_esq = false;
419     //Serial.print (" Contagem (ESQUERDA) = ");
420     //Serial.println (rotacao_esq);
421   }
422
423   if (pronto_dir == true) {     // assim que o sentido estiver definido
424     if (sentido_dir == HIGH) {  // se a variavel sentido for HIGH roda esta a girar↩
          para a frente
425       rotacao_dir++;
426       Serial.println (" RODA DIREITA FRENTE ");
427     } else {                    // caso contrario (se sentido = LOW) roda gira para↩
          tras
428       rotacao_dir--;
429       Serial.println (" RODA DIREITA TRAS ");
430     }
431     pronto_dir = false;
432     //Serial.print (" Contagem (DIREITA) = ");
433     //Serial.println (rotacao_dir);
434   }
435 }
436
437 // INTERRUPCAO RODA ESQUERDA
438 void interrupt_esq () {                     // funcao que e chamada com a ↩
    interrupcao externa do sensor de hall
439   if (digitalRead(encoderPinA) == HIGH) {   // se a leitura do sensor de hall do ↩
        pinA dor HIGH
440     sentido_esq = digitalRead(encoderPinB); // entao o sentido da roda vai ser dado↩
          pelo estado do sensor pinB (1 ou 0)
441   } else {                                  // caso contrario
442     sentido_esq = !digitalRead(encoderPinB);// o sentido vai ser igual ao esatdo ↩
        NAO pinB (1 ou 0)
443   }
444   pronto_esq = true;                        // coloca se a variavel a true para ↩
        iniciar a funcao sentido()
445 }
446
447 // INTERRUPCAO RODA DIREITA
448 void interrupt_dir () {                     // funcao que e chamada com a ↩
    interrupcao externa do sensor de hall
449   if (digitalRead(encoderPinC) == HIGH) {   // se a leitura do sensor de hall do ↩
        pinC dor HIGH
450     sentido_dir = digitalRead(encoderPinD); // entao o sentido da roda vai ser dado↩
          pelo estado do sensor pinD (1 ou 0)
451   } else {                                  // caso contrario
452     sentido_dir = !digitalRead(encoderPinD);// o sentido vai ser igual ao esatdo ↩
        NAO pinD (1 ou 0)
453   }
454   pronto_dir = true;                        // coloca se a variavel a true para ↩
        iniciar a funcao sentido()
455 }
456
457 /*------------ Funcao controla os LEDS do HALL SENSOR ↩
    ------------------------------------------------------*/
458 void LEDS(boolean led) {
459   // LEDS para os SENSORES HALL
```

```
460     if(led == 1){
461      if((digitalRead(SHALL_ESQ_FRENTE)==LOW)){ // Quando detecta o iman
462        digitalWrite(LEDPin_ESQ_FRENTE, HIGH);  // Acende LED
463      } else {
464        digitalWrite(LEDPin_ESQ_FRENTE, LOW);   // Apaga LED
465      }
466      if((digitalRead(SHALL_ESQ_TRAS)==LOW)){   // Quando detecta o iman
467        digitalWrite(LEDPin_ESQ_TRAS, HIGH);    // Acende LED
468      } else {
469        digitalWrite(LEDPin_ESQ_TRAS, LOW);     // Apaga LED
470      }
471      if((digitalRead(SHALL_DIR_TRAS)==LOW)){   // Quando detecta o iman
472        digitalWrite(LEDPin_DIR_TRAS, HIGH);    // Acende LED
473      } else {
474        digitalWrite(LEDPin_DIR_TRAS, LOW);     // Apaga LED
475      }
476      if((digitalRead(SHALL_DIR_FRENTE)==LOW)){ // Quando detecta o iman
477        digitalWrite(LEDPin_DIR_FRENTE, HIGH);  // Acende LED
478      } else {
479        digitalWrite(LEDPin_DIR_FRENTE, LOW);   // Apaga LED
480      }
481     }
482
483     if(led == 0){
484      digitalWrite(LEDPin_ESQ_FRENTE, LOW); // TODOS OS LEDS OFF
485      digitalWrite(LEDPin_ESQ_TRAS, LOW);
486      digitalWrite(LEDPin_DIR_TRAS, LOW);
487      digitalWrite(LEDPin_DIR_FRENTE, LOW);
488     }
489 }
```