Departamento
de Engenharia Mecânica

Projeto de um Equipamento para Medição de Velocidade de Limitadores de Velocidade em Elevadores

Relatório de Projeto para a obtenção do grau de Mestre em Engenharia Mecânica

Autor

Marco Alexandre Sacramento Tomé

Orientadores

Professor Doutor Luís Manuel Ferreira Roseiro

Professor Coordenador

Professor Doutor Pedro Jorge Borges Fontes Negrão Beirão

Professor Adjunto

Coimbra, fevereiro de 2017

“No que diz respeito ao desempenho, ao compromisso, ao esforço, à dedicação, não existe meio termo. Ou você faz uma coisa bem feita ou não faz”

Ayrton Senna da Silva

AGRADECIMENTOS

Para o desenvolvimento do presente trabalho, contribuíram diversas pessoas, a quem gostaria de expressar o meu sincero agradecimento:

- Ao Professor Doutor Luís Roseiro, mais que orientador deste trabalho, foi também o responsável por este se tornar realidade. Quero agradecer-lhe por todo o incentivo para que não desistisse de fazer este mestrado e por toda a ajuda não só no desenvolvimento deste trabalho, bem como em todo o meu percurso pessoal e profissional. E ainda por toda a confiança em mim depositada.
- Ao Professor Doutor Pedro Beirão, meu orientador, por todos o conhecimento que me transmitiu sem qualquer reserva, por todos os conselhos que me deu, mesmo às vezes fora de horas e ainda por todo o incentivo e apoio, não só neste trabalho, mas também em todos os outros em que esteve envolvido comigo e que levou mesmo até ao reconhecimento público.
- Ao Professor Doutor Frederico Santos por toda a ajuda e apoio na programação e em toda a parte elétrica deste projeto. Ensinou-me muita coisa sobre *Arduino* e sobre a sua programação, sempre sem reservas.
- Aos meus Pais, Maria das Dores e José Manuel, sinto-me grato pela presença, incentivo, apoio e por todo o investimento académico que tiveram comigo. Obrigado por me educarem com os valores que fazem de mim a pessoa que hoje sou e que espero poder transmitir a outros. Não há palavras para descrever os incríveis pais que vocês foram e, mesmo na situação difícil em que nos encontrávamos, nunca me deixaram desistir e incentivaram-me para que fosse mais além. A vida não foi fácil para vocês, às vezes até foi injusta demais, mas vocês reergueram-se das cinzas e fizeram o que sou hoje. São o exemplo que pretendo conseguir atingir enquanto pessoa e ser humano.
- Ao meu Irmão Duarte, que mesmo com as injúrias da vida, me deu o lugar para que conseguisse chegar mais longe. Tiveste que ficar para trás para que eu tivesse a oportunidade de ir mais além e espero que tenhas orgulho naquilo em que me tornei.
- Ao meu Tio Vasco, mais que tio, é o exemplo de pessoa em que me quero tornar. Ensinaste-me que mesmo com todas as dificuldades que se atravessem no nosso caminho é sempre possível dar a volta por cima. Que com trabalho e dedicação tudo é possível e que o impossível só existe se acreditarmos nele.
- Ao João Ferreira, à Tânia Paiva, ao Ricardo Pereira e ao Joel Pereira, por terem sido mais que amigos, são como irmãos para mim. Obrigado por terem visto para além da aparência e me terem mostrado que existem amigos que se podem considerar família. Obrigado pelos conselhos e por me terem aturado tanta coisa e mesmo assim tentarem mostrar-me o caminho. As barreiras psicológicas quando levantadas são muito difíceis de derrubar, mas vocês ajudaram-me a que conseguisse fazer frente a isso.
- Ao Sílvio Santos, por ter sido como um segundo pai para mim. Obrigado pelas longas conversas, pelos conselhos, pelo apoio incondicional e pelo carinho que me deu.
- Ao Marcelo Sousa, por todo o apoio e amizade que me deu. Contigo havia sempre uma brincadeira, uma anedota, ou outra coisa qualquer para tornar o trabalho e o estudo mais interessantes.

- Ao Raimundo Felismina, Daniel Gonçalves e Miguel Barbosa, colegas e amigos, são aquela elite que nunca me esquecerei.
- Ao Miguel Dias e Eusébio Rodrigues, por me terem ensinado muito durante todo o meu percurso na empresa Liftime Elevadores.
- A todos os restantes colegas da empresa Liftime Elevadores por me terem recebido tão bem na empresa e me terem sempre dado alguns conselhos e dicas para a vida futura.
- À Maria João d'Aguiar, Sara Sales, Rita Santos e Ana Pinto, para além de amigas são as meninas que me dão forças para esta nova fase da minha vida que é perder peso e me incentivam a sempre fazer mais e melhor para que atinga os meus objetivos. E espero que um dia possam vir a ter orgulho no que consegui atingir.
- Ao Fernando Almeida, meu PT e amigo, por me ter ajudado nesta nova fase da minha vida e por fazer com que consiga tornar numa pessoa mais saudável.
- A toda a malta do Fitness Hut e em especial ao David Silva, ao Tó Zé, à Nádia Silva, à Manuela e ao Fábio Teixeira, por me incentivarem a perder peso e também por me darem sempre conselhos para que consiga atingir mais rapidamente os objetivos.
- Quero ainda dedicar em especial à memória do Senhor Engenheiro Carlos Coelho da empresa Liftime Elevadores, por ter feito com que este projeto fosse possível e por me ter acolhido na sua empresa. Foi em parte o meu orientador na empresa e todos os seus conselhos foram uma mais-valia para o meu desenvolvimento enquanto pessoa e trabalhador.

RESUMO

Um dos aspetos mais importantes nos elevadores é a sua segurança, do qual o limitador de velocidade é um dos equipamentos essenciais. O limitador de velocidade é responsável por reduzir o risco de acidentes derivados da velocidade excessiva do elevador.

Devido à sua importância, este equipamento é submetido a inspeções periódicas para determinar o seu bom funcionamento. O método utilizado para a realização destas inspeções é arcaico e os equipamentos atualmente empregues para esse efeito não são os mais adequados.

Pretende-se com este projeto desenvolver, projetar e construir um protótipo de um equipamento que determine, da forma mais rigorosa possível, recorrendo a sensores e uma placa de aquisição de dados, o estado de funcionamento dos limitadores de velocidade, com base nas normas em vigor. Pretende-se ainda que este equipamento efetue, de forma automática, os vários cenários (escorregamento e queda livre), permitindo que o utilizador visualize em tempo real, na interface gráfica a conceber para *tablet* ou *smartphone* (com sistema *Android*), o gráfico velocidade linear-tempo do ensaio.

Palavras-chave: Elevador; Limitador de velocidade; Inspeção

ABSTRACT

One of the most relevant aspects when dealing with elevators is their safety, from which the overspeed governor is an essential equipment. The overspeed governor is responsible for reducing the risk of accidents that could happen when the maximum velocity of the elevator is exceeded.

Because of its crucial importance, this safety equipment should be subjected to periodic inspections to assure their proper functioning. Currently, the method used to carry out these inspections is outdated and the equipment used for this purpose is not the most appropriated.

The aim of this project comprises the development, design and construction of a prototype of a device that should be able to determine, as accurately as possible, resorting to sensors and a data acquisition board, the operating status of overspeed governors, according to the governing standards. It is also intended that this device should provide, in real time, the linear speed-time graphic, after testing the various scenarios (slipping and free fall), allowing the user to view them on a graphical interface like, for instance, a tablet or a smartphone (with Android system).

Keywords: Elevator; Overspeed governor; Inspection

ÍNDICE GERAL

AGRADECIMENTOS ..... **V**

RESUMO ..... **VII**

ABSTRACT ..... **IX**

ÍNDICE GERAL ..... **XI**

LISTA DE FIGURAS ..... **XIII**

SÍMBOLOS E ABREVIATURAS ..... **XV**

UNIDADES ..... **XV**

1 INTRODUÇÃO **1**

1.1 OBJETIVOS 1

1.2 APRESENTAÇÃO DA EMPRESA 1

1.3 ENQUADRAMENTO E INTRODUÇÃO GERAL 2

1.4 ESTRUTURA DO RELATÓRIO 6

2 DESENVOLVIMENTO DO PROTÓTIPO **7**

2.1 HIPÓTESES 7

2.2 COMPONENTES UTILIZADOS 8

2.2.1 Motorredutor *8*

2.2.2 Suportes *9*

2.2.3 Roda com o-ring *11*

2.2.4 Placa Arduino *11*

2.2.5 Placa Bluetooth *12*

2.2.6 Placa de potência *12*

2.2.7 Encoder *13*

2.3 MONTAGEM FINAL 13

3 PROGRAMAÇÃO ARDUINO **17**

3.1 BIBLIOTECAS 18

3.2 VARIÁVEIS 18

3.2.1 Tipos de variáveis *19*

3.2.2 Declarar variáveis *19*

3.3 *VOID SETUP* () 20

3.4 FUNÇÕES 23

3.5 *VOID LOOP* () 24

4 PROGRAMAÇÃO *ANDROID* **31**

4.1 PÁGINA DE DESENHO DA INTERFACE DA APLICAÇÃO 31
4.2 PÁGINA DE DESENHO DE BLOCOS DA APLICAÇÃO 35
4.2.1 Declaração de variáveis 36
4.2.2 Inicialização da aplicação 37
4.2.3 Ligação Bluetooth 39
4.2.4 Programação dos botões 40
4.2.5 Obtenção de valores em tempo real 41
4.2.6 Função externa Gráfico 43
4.2.7 Resultado obtido 44
5 VERSÃO FINAL DO EQUIPAMENTO 45
6 CONCLUSÕES 51
7 BIBLIOGRAFIA 53
ANEXO 1 – PROGRAMA COMPLETO ARDUINO 55
ANEXO 2 – PROGRAMA DE COMPLETO DE BLOCOS EM APP INVENTOR 57

LISTA DE FIGURAS

Figura 1 – Esquema de elevador com casa das máquinas (Atlas Schindler, 2015) 2

Figura 2 – Esquema de elevador com casa das máquinas no topo da caixa (esquerda), em baixo (centro) e sem casa das máquinas (direita) (Gomes, 2012) 3

Figura 3 – Célula de carga de elevador 3

Figura 4 – Paraquedas de elevador (Googglet, 2015) 4

Figura 5 – Limitador de velocidade (Gervall, S.A., 2014) 4

Figura 6 – Sistema completo de limitador de velocidade com paraquedas e roda tensora (Kaskus, 2016) 5

Figura 7 – Esquema simplificado do sistema de segurança composto por limitador de velocidade e paraquedas (SETA - Shanghai Elevator Trade Association, 2015) 5

Figura 8 – Montagem da roda com *o-ring* (direita) no limitador de velocidade (esquerda) 7

Figura 9 – Motorredutor 9

Figura 10 – Chapa de suporte do protótipo 9

Figura 11 – Chapa de suporte do equipamento de medição 10

Figura 12 – Suporte para os componentes eletrónicos 10

Figura 13 – Roda com *o-ring* 11

Figura 14 – Arduino Uno SMD (Arduino, 2016) 11

Figura 15 – Placa *Bluetooth* HC-06 (ITEAD Intelligent Systems Co.Ltd., 2016) 12

Figura 16 – Placa de potência Cytron MD10 (Cytron Technologies Sdn., 2016) 12

Figura 17 – Encoder HEDS 5500 (Digi-Key Electronics, 2016) 13

Figura 18 – Protótipo final em fase de testes 14

Figura 19 – Suporte dos componentes eletrónicos 14

Figura 20 – Excerto de código com declaração de variáveis 19

Figura 21 – Exemplo de código *void setup* () (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016) 20

Figura 22 – Excerto de código *void setup* () 20

Figura 23 – Excerto de código *void setup* () relativo ao controlo do motor 21

Figura 24 – Sinal de um *encoder*, pistas A e B (Eletronica 100 limites, 2012) 21

Figura 25 – Excerto de código *void setup* () relativo ao controlo do *encoder* 22

Figura 26 – Excerto de código *void setup* () relativo ao controlo do temporizador 22

Figura 27 – Exemplo de código de uma função (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016) 23

Figura 28 –- Bloco de código com as funções 24

Figura 29 – Excerto de código *void loop* () relativo à receção de dados 25

Figura 30 – Excerto de código *void loop* () relativo ao cálculo e seguranças..... 25

Figura 31 – Sistema definido para o cálculo de velocidade..... 26

Figura 32 – Excerto de código *void loop* () relativo ao mapa de escorregamento..... 29

Figura 33 – Exemplo de página de desenho da interface (zonas *Pallets* e *Viewer*) (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016) 32

Figura 34 – Exemplo de página de desenho da interface (zona *Properties*) (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016)..... 33

Figura 35 – Página de desenho da interface de aplicação do protótipo 34

Figura 36 – Acesso à página de desenho de blocos da aplicação (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016)..... 35

Figura 37 – Exemplo de página de desenho de blocos da aplicação (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016)..... 35

Figura 38 - Exemplo de agrupamento de blocos (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016)..... 36

Figura 39 – Declaração de variáveis do programa de blocos do protótipo..... 37

Figura 40 – Bloco de inicialização da aplicação do protótipo 37

Figura 41 – Bloco de função externa "Reset" do programa do protótipo 38

Figura 42 – Interface inicial do programa do protótipo 39

Figura 43 – Blocos para ligação *Bluetooth* do protótipo 39

Figura 44 – Seletor de ligações *Bluetooth* do programa do protótipo 40

Figura 45 – Blocos de programação dos botões do programa do protótipo 41

Figura 46 – Bloco de temporizador do programa do protótipo..... 42

Figura 47 – Função externa Gráfico do programa do protótipo 43

Figura 48 – Exemplo de relatório final obtido após o ensaio 44

Figura 49 – Versão final do equipamento (modelo 3D) 45

Figura 50 – Jogo de rotação sob um pino fixo com uma mola de torção 46

Figura 51 – Suportes para fixação de vidros (Batista Gomes, Lda., 2010) 46

Figura 52 – Sistema de suporte do equipamento 47

Figura 53 – Vista explodida do sistema de suporte do equipamento..... 47

Figura 54 –- Ajuste longitudinal do equipamento 48

Abreviaturas

3D – 3 Dimensões

ABS – Acrilonitrilo Butadieno Estireno

PWM – *Pulse Width Modulation*

DC – *Direct Current*

Ft – Fator de conversão de tempo

Fa – Fator de conversão de interrupção

MIT – *Massachusetts Institute of Technology*

Unidades

Nm – Newton × metro

rpm – rotações por minuto

A – Ampere

mm – milímetro

V – Volt

mA – miliAmpere

MHz – MegaHertz

Mbps – Megabit por segundo

CPR – *Counts Per Revolution*

m/s – metro por segundo

1 Introdução

1.1 Objetivos

O presente documento tem como principal objetivo desenvolver, projetar e construir um protótipo funcional capaz de determinar, com o máximo rigor possível, a velocidade linear de um limitador de velocidade de um elevador. Para tal vai ser utilizada uma placa de aquisição de dados *Arduino* e um programa em *Android* desenvolvido no *software App Inventor* para permitir a utilização de um *tablet* ou um *smartphone* com sistema *Android*, que sirva de interface de comunicação entre o utilizador e o protótipo

Nas secções referentes aos programas (da placa *Arduino* e do programa desenvolvido em *Android*) o objetivo, para além de descrever o que foi desenvolvido, é também especificar com maior detalhe cada linha ou bloco utilizado, como se tratasse de um tutorial. O objetivo é que o leitor tenha não só a descrição do que foi feito, mas também de como foi feito, passo a passo, para que possa ter uma referência para trabalhos ou investigações futuras.

1.2 Apresentação da empresa

Este projeto foi desenvolvido a pedido da empresa Liftime Elevadores.

Criada em Janeiro de 1999, a Liftime Elevadores tem como principal atividade a comercialização, instalação e assistência técnica de elevadores, monta-cargas e escadas rolantes. Com sede em Eiras, Coimbra, a Liftime Elevadores posiciona-se no mercado local/regional, aliando a competência, o rigor e a seriedade na relação com o cliente a uma forte capacidade de resposta e flexibilidade. Com uma elevada experiência na conceção de elevadores a Liftime Elevadores aposta cada vez mais na inovação dos produtos e na formação dos seus quadros, adotando uma postura de constante desafio em relação ao mercado (Liftime Elevadores, 2006).

A ideia inicial deste projeto partiu da direção desta empresa, na pessoa do Senhor Engenheiro Carlos Coelho. A Liftime Elevadores queria destacar-se dos seus concorrentes apresentando um equipamento inovador capaz de realizar ensaios de medição de velocidade de modo mais rigoroso que os efetuados atualmente. Por questões de manutenção e segurança, esses ensaios devem ser efetuados periodicamente ao equipamento responsável por reduzir o risco de acidentes derivados da velocidade excessiva do elevador, designado por limitador de velocidade. Pretendia-se que esta medição fosse feita e acompanhada, em tempo real, através de um gráfico velocidade-tempo.

1.3 Enquadramento e introdução geral

Para melhor se entender a importância deste projeto é necessário que se perceba minimamente como funciona um elevador e a importância que o limitador de velocidade tem. Esta secção serve para ambientar o leitor no que diz respeito ao elevador em si e, mais especificamente, ao limitador de velocidade. Para tal começa-se pela sua constituição, classificação e ainda se descreve, de forma sucinta, o seu princípio de funcionamento.

Um elevador é constituído por um sistema complexo esquematizado na Figura 1. É basicamente composto por uma cabina, um contrapeso, uma máquina de tração, um quadro de comando (na Figura 1 designado por painel de controle), um sistema de guias (de cabina e de contrapeso) e os vários elementos de segurança (limitador de velocidade, para-quedas (na imagem designado como segurança), amortecedor (ou pára-choques), célula de carga, entre muitos outros.

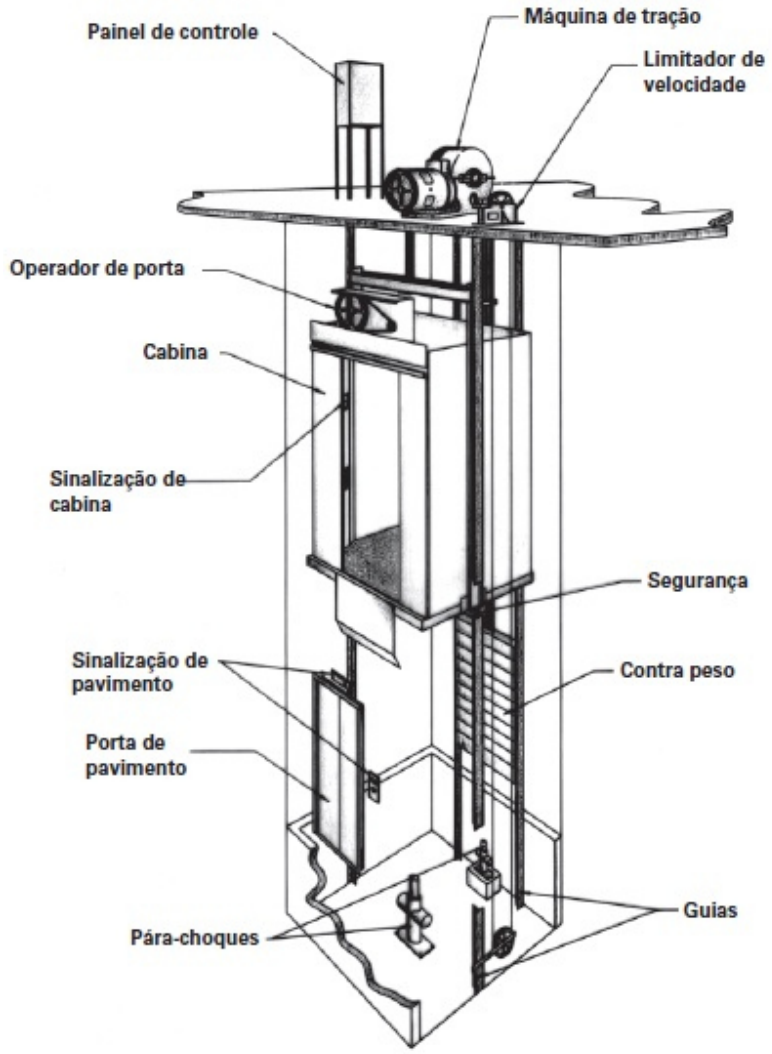


Figura 1 – Esquema de elevador com casa das máquinas (Atlas Schindler, 2015)

No que concerne à classificação quanto ao tipo de tração, os mais utilizados são:

- Elevadores elétricos de tração por roda de aderência;
- Elevadores hidráulicos.

Já quanto à localização da casa das máquinas, estes podem ser classificados (Figura 2):

- Com casa das máquinas no topo da caixa do elevador (Figura 1);
- Com casa das máquinas em baixo (tipo de construção quase restrito a elevadores de tração hidráulica para que a bomba fique o mais próxima possível do cilindro do elevador);
- Sem casa das máquinas.

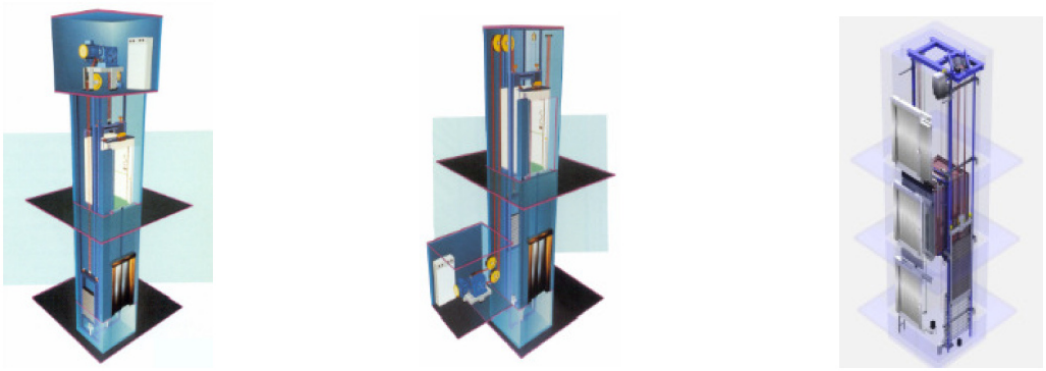


Figura 2 – Esquema de elevador com casa das máquinas no topo da caixa (esquerda), em baixo (centro) e sem casa das máquinas (direita) (Gomes, 2012)

Como se pode verificar, mesmo sendo esta uma classificação extremamente resumida, existe uma infinidade de opções e construções diferentes. No entanto, e no que diz respeito ao limitador de velocidade, este pouco muda.

Atualmente os elevadores têm uma série de sistemas de segurança que atuam em diferentes níveis, podendo mesmo ser até redundantes, sendo que os mais importantes são:

- **Célula de carga (Figura 3):** Atua parando a máquina quando o elevador se encontra com carga superior à carga definida na própria célula**;**

Figura 3 – Célula de carga de elevador

- **Paraquedas (Figura 4):** Assegura a paragem do elevador quando o limitador de velocidade é trancado por excesso de velocidade (este sistema de segurança vai ser descrito mais detalhadamente na descrição do limitador de velocidade);

Figura 4 – Paraquedas de elevador (Googglet, 2015)

- **Limitador de velocidade (Figura 5):** Dispositivo que atua quando a velocidade da cabina do elevador ultrapassa uma determinada velocidade de segurança.

Figura 5 – Limitador de velocidade (Gervall, S.A., 2014)

Um limitador de velocidade de um elevador está ligado por intermédio de um cabo à cabina do elevador, assim como ao paraquedas e à roda tensora, tal como demonstrado na Figura 6. Este cabo é animado pelo movimento da cabina do elevador e transmite esse movimento à roda do limitador de velocidade. Já o limitador de velocidade, por intermédio de um sistema de roda oscilante (idêntico ao da Figura 6), ou por qualquer outro tipo de sistema (como por exemplo os centrífugos), faz uma comparação contínua e mecânica da velocidade da cabina com a velocidade de disparo. Através da Norma Portuguesa para fabrico e montagem de elevadores (Norma Portuguesa NP EN 81-1:2000, 2000) a velocidade de disparo é, para a maioria dos elevadores, no máximo 40% superior à velocidade nominal do elevador.

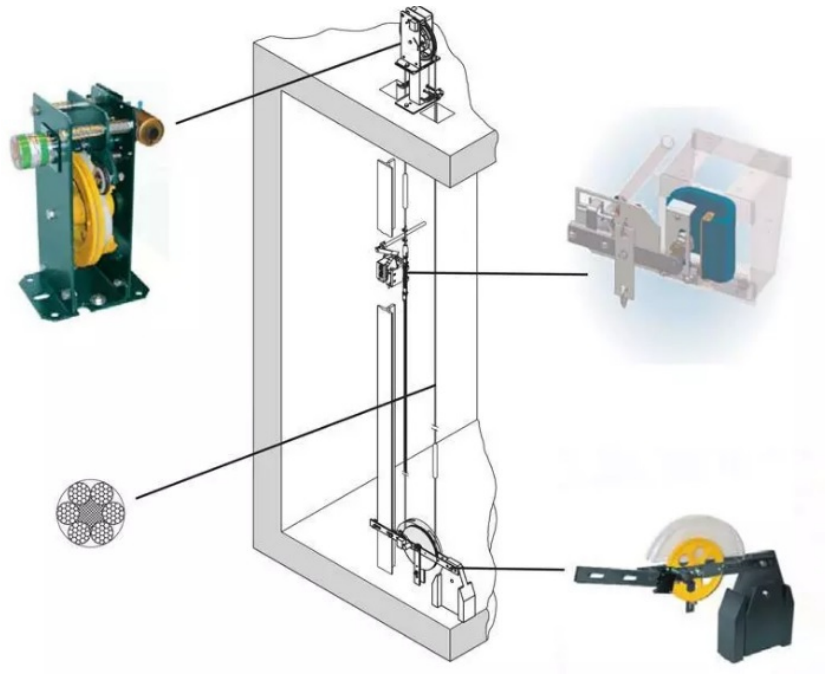


Figura 6 – Sistema completo de limitador de velocidade com paraquedas e roda tensora (Kaskus, 2016)

Quando é atingida esta velocidade o limitador de velocidade é trancado mecanicamente e faz parar o cabo que por sua vez atua o paraquedas fazendo assim com que a cabina pare. A atuação do paraquedas pode ser direta ou progressiva. Atualmente, por questões de conforto, a mais utilizada é a atuação progressiva.

A Figura 7 esquematiza de forma simples o sistema composto pelo limitador de velocidade e pelo paraquedas.

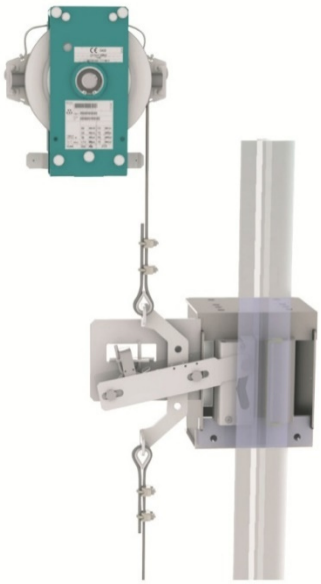


Figura 7 – Esquema simplificado do sistema de segurança composto por limitador de velocidade e paraquedas (SETA - Shanghai Elevator Trade Association, 2015)

O limitador de velocidade é submetido a inspeções periódicas e obrigatórias por parte de entidades certificadas para o efeito, as quais são acompanhadas sempre por um representante da empresa responsável pela manutenção do elevador.

Após esta breve introdução percebe-se agora a extrema importância da correta medição da velocidade de disparo do limitador de velocidade. Como já foi dito, este é submetido a inspeções periódicas e nessas inspeções é feita a medição da velocidade de disparo do limitador de velocidade, bem como é avaliado também o seu estado de conservação.

Atualmente esta medição é feita de modo muito arcaico, sendo utilizado um berbequim (ou algo semelhante) para animar a roda do limitador de velocidade, enquanto outro operador mede com um tacómetro a velocidade de rotação da roda, sendo registada a maior atingida pelo próprio tacómetro. Esta operação, para além de perigosa (operadores muito próximos do limitador de velocidade e este não tem proteções enquanto o ensaio é feito), também não garante a correta leitura da velocidade de disparo devido a vários fatores, como por exemplo:

- Não é controlada a velocidade que é inserida no limitador (a velocidade é apenas controlada através da força que o operador seleciona no berbequim (ou equivalente);
- O tacómetro tem um atraso associado (basta que o operador que controla a velocidade faça um pico de rotação instantâneo para que o limitador tranque sem que o tacómetro consiga ler essa mesma velocidade de disparo).

1.4 Estrutura do relatório

O relatório de projeto encontra-se subdividido em 6 capítulos.

No primeiro capítulo pretende-se dar a conhecer os objetivos, a apresentação da empresa para a qual este projeto foi desenvolvido e ainda um enquadramento sobre o princípio de funcionamento de um elevador e, mais detalhadamente, sobre o limitador de velocidade. É também descrita a forma de realizar a inspeção e ensaio deste componente e os problemas que daí advém. Finalmente é ainda apresentada a própria estrutura deste relatório.

Os capítulos 2, 3 e 4 abordam, respetivamente, todo o desenvolvimento do protótipo na sua componente mecânica (detalham-se os materiais, peças e componentes utilizados para a construção do protótipo), a programação em *Arduino* (descreve-se o código e algoritmo utilizado; pretende-se que o leitor tenha acesso a uma explicação mais detalhada de todo o código para o poder consultar posteriormente e poder utilizar somente as partes que lhe interessem) e por fim o programa em *Android* feito no *software App Inventor* (tendo também a mesma estrutura do capitulo anterior).

O capítulo 5 aborda a versão final do equipamento incluindo algumas melhorias a aplicar na sua construção.

O último capítulo é reservado para as conclusões a retirar do trabalho desenvolvido.

2 Desenvolvimento do protótipo

Neste capítulo vai ser abordado o desenvolvimento do protótipo, desde a construção mecânica de suportes até aos componentes elétricos e eletrónicos utilizados, nomeadamente placas eletrónicas, *encoder* e motorredutor.

Este capítulo vai ser dividido em pequenas secções de modo a enfatizar cada componente utilizado, incluindo, sempre que possível, a justificação para a utilização de tal componente.

2.1 Hipóteses

Para fazer movimentar o limitador de velocidade foram abordadas várias hipóteses, das quais se destacaram as seguintes:

- Utilização de transmissão flexível por cabo;
- Utilização de transmissão flexível por correia trapezoidal;
- Utilização de transmissão por contacto/atrito por roda com *o-ring*.

As primeiras duas hipóteses, desde que se garantisse uma correta montagem e fossem assegurados todos os apertos de modo à correia ou cabo não escorregarem em nenhuma das partes, eram as melhores. No entanto, devido à construção do limitador (ver Figura 5 ou Figura 8), eram demasiado complexas de montar e com isso a medição iria tornar-se muito dispendiosa.

A última hipótese assenta na utilização de transmissão por contacto através de uma roda com *o-ring* incorporado evitando qualquer tipo de problema na montagem, visto só ser necessário encostar a roda com *o-ring* ao limitador de velocidade, tal como demonstrado na Figura 8.

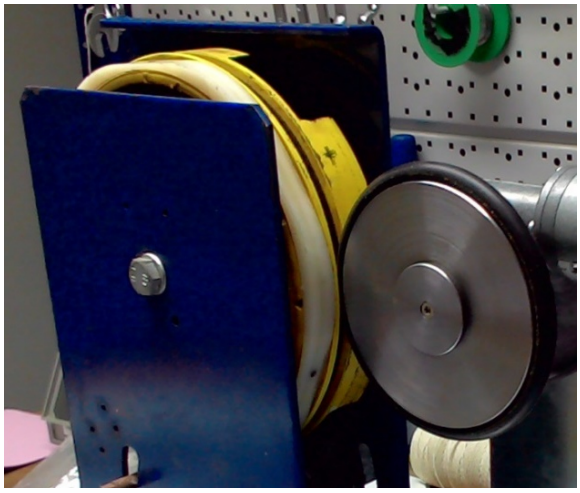


Figura 8 – Montagem da roda com *o-ring* (direita) no limitador de velocidade (esquerda)

Esta hipótese pode também sofrer do mesmo problema (escorregamento), mas desde que seja garantida uma força de atrito entre as rodas, este fenómeno pode ser quase eliminado. A utilização do *o-ring* permite ultrapassar este problema devido à sua constituição (material) e ainda reduz o ruído e as vibrações de contacto entre os componentes. Para além disso, simplifica os cálculos, porque se for garantido que não existe escorregamento, a velocidade linear do limitador é igual à velocidade linear da roda com *o-ring*, independentemente do tamanho do limitador de velocidade.

2.2 Componentes utilizados

Depois de escolhida a última das três hipóteses anteriores como sendo a solução adequada, passou-se para a construção do protótipo.

Numa primeira fase era necessário medir a velocidade da roda com *o-ring*, pelo que o protótipo a construir necessitaria dos seguintes componentes:

- Limitador de velocidade;
- Motorredutor elétrico;
- Suportes (chapas quinadas e/ou impressas em 3D);
- Roda com *o-ring*;
- *Encoder*;
- Placa *Arduino*.

Visto que o motorredutor necessita de uma tensão e intensidade de corrente superior à que a placa *Arduino* permite e que esta não tem nenhuma interface homem-máquina, foram então adicionados à lista os seguintes componentes:

- Placa de potência;
- Placa de *Bluetooth*.

2.2.1 Motorredutor

Neste protótipo foi utilizado um motorredutor de um operador de porta (motor responsável pela abertura automática das portas de cabina e patamar), com as seguintes características (KAG - Kähling Antriebstechnik GmbH, 2016):

- Binário: 1,08 Nm
- Redução: 15:1
- Rotação máxima admissível (motor): 3000 rpm
- Rotação máxima admissível (após redução): 200 rpm
- Eficiência: 72%
- Intensidade de corrente a 3000rpm (motor): 1,8 A

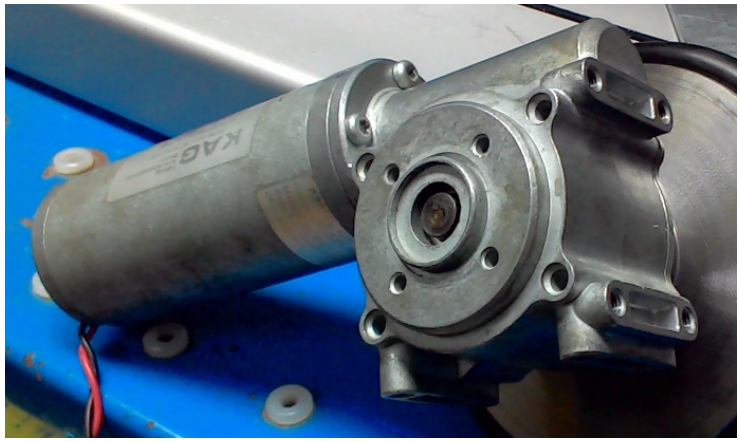


Figura 9 – Motorredutor

2.2.2 Suportes

Foi feito um suporte para todo o protótipo, bem como para o equipamento de medição e também para as placas eletrónicas. Tanto o suporte de todo o protótipo, bem como do equipamento de medição foram feitos em chapa de aço quinada para fornecer mais robustez. Já o suporte das placas eletrónicas (placa *Arduino*, placa de potência e placa *Bluetooth*) foi feito numa impressora 3D usando polímero ABS como material.

A chapa de suporte do protótipo tem como objetivo albergar todos os componentes do protótipo (limitador de velocidade, equipamento de medição e placas eletrónicas). Foi montada na placa didática já fornecida (chapa azul visível na Figura 10).

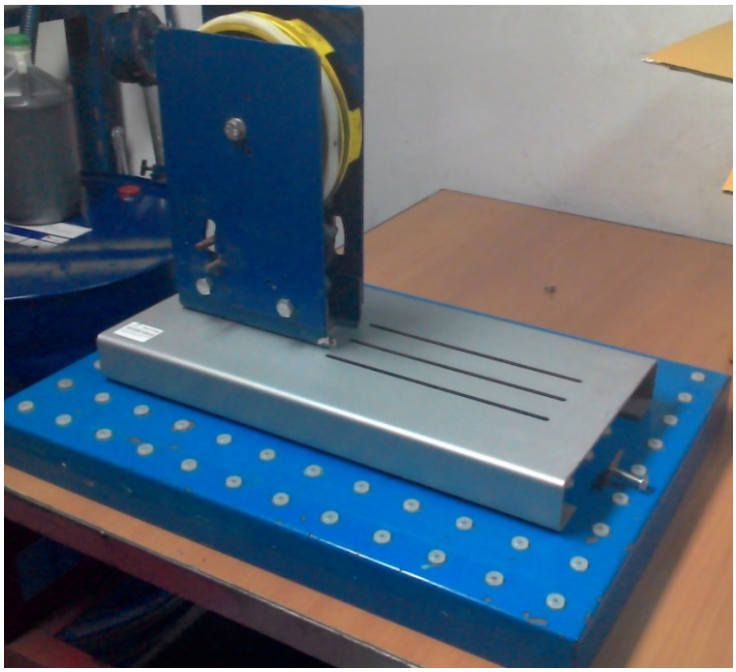


Figura 10 – Chapa de suporte do protótipo

Como se pode observar na Figura 10, a chapa de suporte do protótipo tem apenas afinação longitudinal, pelo que a chapa de suporte do equipamento de medição deve conter uma afinação transversal, para além de ter o apoio do motorredutor. Esta chapa também foi feita tendo em conta o limitador de velocidade utilizado no protótipo, pelo que a altura é fixa. Construiu-se a chapa visível na Figura 11.

Figura 11 – Chapa de suporte do equipamento de medição

Por fim foi também desenvolvida uma peça de suporte para os componentes eletrónicos. Como não era necessário que o suporte resistisse a esforços, pode ser feito usando a tecnologia de impressão 3D.

Em termos de restrições apenas se considera a localização das furações de apoio para as placas e uma redução máxima das dimensões da peça (para reduzir ao máximo o material utilizado), obtendo-se assim o suporte (já com a placa *Arduino* e com a placa *Bluetooth* montadas) visível na Figura 12.

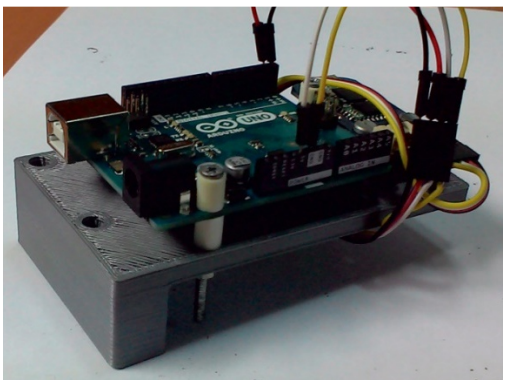


Figura 12 – Suporte para os componentes eletrónicos

2.2.3 Roda com *o-ring*

Esta é talvez o componente mais importante de todo o equipamento, visto que foi maquinada para ser adapta no motorredutor (ver secção 2.2.1) com o intuito de fazer movimentar o limitador de velocidade. Para isso foi necessário introduzir outro interveniente, o *o-ring*. Como consequência a roda foi dimensionada para se conseguir montar um *o-ring* de dimensões 10 × 100 mm (diâmetro da secção transversal × diâmetro interno do *o-ring*).

Figura 13 – Roda com *o-ring*

2.2.4 Placa *Arduino*

Para a aquisição e processamento de dados do protótipo foi utilizada uma placa *Arduino* com as seguintes características (Arduino, 2016):

- Microcontrolador: ATmega328
- Tensão de operação: 5 V
- Tensão de alimentação: 7-12 V
- Pinos digitais de entrada/saída: 14 (das quais 6 podem ter saída com sinal PWM)
- Intensidade nos pinos digitais: 40 mA
- Frequência de relógio: 16 MHz
- Pinos que permitem interrupções: 2 (pinos 2 e 3)

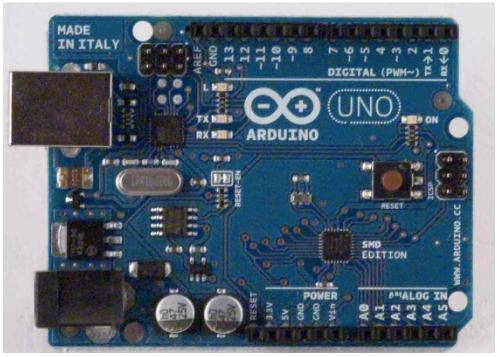


Figura 14 – Arduino Uno SMD (Arduino, 2016)

2.2.5 Placa *Bluetooth*

Para fazer a comunicação entre a paca *Arduino* e um *tablet* ou um *smartphone Android* usado como interface gráfica de comunicação entre o utilizador e o protótipo é necessário um dispositivo que seja suportado por estes equipamentos. Para isso foi utilizada uma placa *Bluetooth* com as seguintes características (ITEAD Intelligent Systems Co.Ltd., 2016):

- Versão do *Bluetooth*: 2.0 (compatível com a grande maioria dos dispositivos *Android*)
- Taxa de transferência: 3 Mbps
- Tensão de alimentação: 3,3 a 5 V

Figura 15 – Placa *Bluetooth* HC-06 (ITEAD Intelligent Systems Co.Ltd., 2016)

2.2.6 Placa de potência

Como o motorredutor necessita de uma intensidade de corrente superior à que a placa *Arduino* suporta e ainda para se conseguir controlar o motorredutor foi necessário recorrer a uma placa de potência, também chamada ponte H. Esta permite controlar um motor utilizando apenas uma saída PWM (para controlar a velocidade do motorredutor) e uma saída digital (para controlar a direção de rotação do motorredutor). Seguidamente descrevem-se as suas características (Cytron Technologies Sdn., 2016):

- Controlo bidirecional de motor DC
- Tensão de saída (controlo de motor): 7 a 30 V
- Tensão da entrada: 3,3 a 5 V
- Ligação direta na placa *Arduino* (evita cabos de ligação entre placas)

Figura 16 – Placa de potência Cytron MD10 (Cytron Technologies Sdn., 2016)

2.2.7 *Encoder*

Como um dos requisitos definidos para este protótipo consiste na medição da velocidade do limitador de velocidade, foi utilizado um *encoder*. O *encoder* não necessita de ter uma grande resolução (para não sobrecarregar a placa *Arduino*), visto que se ainda trata do desenvolvimento de um protótipo e a sua função é determinar se, na prática, a solução proposta funciona convenientemente. Para a sua escolha foi necessário ter em consideração que o *encoder* é montado no eixo do motorredutor. Este tem uma velocidade de rotação que pode atingir, no máximo, as 3000 rpm. Por isso, se, por exemplo, se acolpar um *encoder* com uma resolução de 1000 impulsos por rotação, poder-se-á ter então 6 000 000 de impulsos por minuto ou 100 000 por segundo, o que poderá ser demasiado para a placa *Arduino*. Sendo assim, foi utilizado um *encoder* com as seguintes características (Avago Technologies, 2016):

- Veio oco de 5 mm de diâmetro (para montar no eixo do motorredutor)
- Tipo: incremental
- Número de pistas: 3 (canal A, B e I)
- Resolução: 100 CPR

Figura 17 – Encoder HEDS 5500 (Digi-Key Electronics, 2016)

2.3 Montagem final

Após ter recebido todos estes componentes, passou-se para a montagem final do protótipo, tendo sempre em consideração todas as recomendações dos fabricantes dos vários componentes. Todos os esquemas fornecidos (nos componentes eletrónicos) foram rigorosamente seguidos. A Figura 18 mostra o protótipo final já em fase de testes.

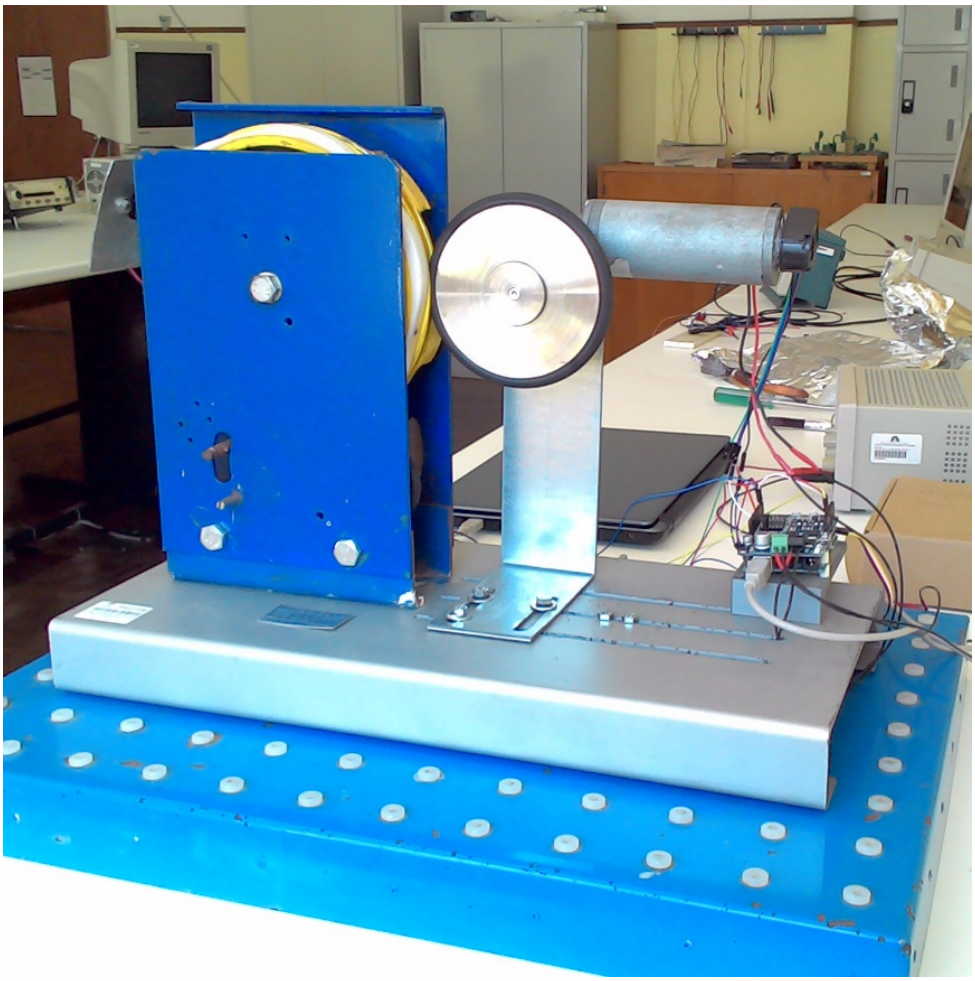


Figura 18 – Protótipo final em fase de testes

A Figura 19 mostra todos os componentes montados (placa *Arduino*, placa de potência e placa *Bluetooth*) no suporte dos componentes eletrónicos impresso em 3D.

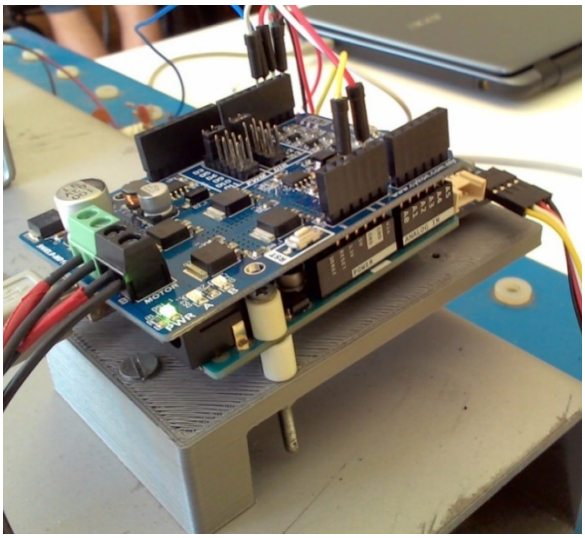


Figura 19 – Suporte dos componentes eletrónicos

Como se pode ver na Figura 19 a placa de potência é montada diretamente na placa *Arduino*, evitando assim o uso de cabos elétricos entre ambas. Como a placa de potência tem os barramentos de ligação iguais aos da placa *Arduino*, todas as ligações que tem de ser feitas na placa *Arduino* (entradas e saídas de sinal) passam agora a ser feitas na placa de potência. Existem também placas *Bluetooth* com esta característica (quando uma placa pode ser montada diretamente na placa *Arduino* é designada *shield*), mas o seu custo não compensava a sua utilização em comparação à usada neste protótipo.

Num trabalho futuro em que se construa um equipamento inspirado neste protótipo pode-se usar uma placa *shield Bluetooth* que pode ser montada por cima da placa de potência, ficando assim uma estrutura tipo sanduiche muito mais compacta e fácil de montar, para além de reduzir o número de cabos elétricos utilizados.

3 Programação *Arduino*

Neste capítulo vai ser abordada toda a programação feita para a placa *Arduino*, incluindo todos os cálculos necessários para que o equipamento forneça a leitura desejada, ou seja, a velocidade linear do limitador de velocidade. O programa completo, desenvolvido para o protótipo, encontra-se em anexo a este documento (ver Anexo 1 Programa Completo em *Arduino*).

Faz-se uma breve introdução sobre a placa *Arduino* e suas vantagens, de modo a familiarizar o leitor com esta tecnologia antes de entra na programação propriamente dita.

A placa *Arduino* é uma plataforma de prototipagem de código aberto baseado em *hardware* e *software* bastante acessível. As placas *Arduino* são capazes de ler entradas (como por exemplo, a luz incidente em um sensor, o estado de um botão ou até mesmo uma mensagem de *Twitter*) e transformá-las em uma saída (como, por exemplo, ativar um motor, ligar uma lâmpada, publicar algo *online*). Podem-se transmitir comandos à placa *Arduino* através do envio de um conjunto de instruções para o microcontrolador da mesma. Para fazer isso usa-se a linguagem de programação *Arduino* e o *software Arduino* (Arduino, 2016).

A placa *Arduino* foi desenvolvida no *Ivrea Interaction Design Institute* como uma ferramenta fácil para prototipagem rápida, destinado a estudantes sem experiência em eletrónica e programação. Assim que chegou a uma comunidade mais ampla, a placa *Arduino* começou a adaptar-se às novas necessidades e desafios, expandindo-se desde simples placas de 8 *bits* até produtos para tecnologias complexas (por exemplo impressão 3D). Todas as placas *Arduino* são completamente *open-source*, permitindo uma grande flexibilidade de utilização, isto é, os utilizadores podem utilizá-las de forma independente e, eventualmente, adaptá-las às suas necessidades específicas. O *software* também é *open-source* e está a desenvolver-se rapidamente graças às contribuições dos utilizadores de todo o mundo (Arduino, 2016).

Entrando agora na programação propriamente dita, o modo de apresentação irá mudar um pouco, passando para um estilo análogo a um tutorial. O objetivo consiste em dar a conhecer o código ao leitor e ensiná-lo (se for o caso) como fazer. Para isso vai ser utilizada a estrutura que os programadores normalmente utilizam.

Vai ser considerado que, no *Arduino*, um código é constituído por 5 elementos principais (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016):

- **Bibliotecas:** aparecem no início do programa onde são incluídas todas as bibliotecas, as quais são normalmente constituídas por funções pré-criadas por outros programadores ou até mesmo pelos criadores da plataforma *Arduino*;

- **Variáveis:** são definidas após as bibliotecas, servindo para guardar informações, como por exemplo o valor de uma velocidade ou o resultado de um cálculo;
- ***void setup* ():** normalmente usado para inicializar variáveis, modos de pinos, definir a velocidade de transmissão de dados *serial* (por exemplo, transmissão de dados por *Bluetooth*, *wifi* e outras redes);
- **Funções:** não é obrigatório que sejam colocadas abaixo das variáveis, mas para seguir um raciocínio admite-se que são colocadas abaixo das variáveis. Estas funções já dizem respeito a funções criadas pelos utilizadores e não às incluídas nas bibliotecas;
- ***void loop* ():** parte do código em que se faz um *loop*, ou seja, cria-se um ciclo de repetição para que as instruções dentro do mesmo sejam repetidas.

A explicação do código utilizado vai ser feita respeitando esta estrutura, ou seja, cada um destes elementos principais vai ser apresentado como uma secção para não se tornar demasiado confuso e facilitar o acesso à informação por parte do leitor.

3.1 Bibliotecas

Quando se cria um código/programa usando uma linguagem de programação, neste caso C/C++, existe a possibilidade de usar um conjunto de funções pré-criadas por outros programadores que já resolvem determinados problemas, poupando imenso trabalho. Esse conjunto de funções designa-se o nome de bibliotecas (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016).

Exemplos de bibliotecas a utilizar ao longo do projeto:

- **Servo:** utilizada quando existe, por exemplo, um servomotor acoplado à placa *Arduino*. Assim, e de uma forma intuitiva, consegue-se posicionar o motor numa determina posição, fornecendo apenas o ângulo como dado de entrada;
- ***Ultrasonic*:** utilizada, por exemplo, para controlo do sensor de ultrassons, facilitando a leitura da distância lida por este;
- ***TimerOne*:** utilizada para definir um temporizador no programa.

Na prática, para utilizar uma biblioteca basta incluí-la no início do código. Por exemplo, no caso do protótipo desenvolvido neste projeto a biblioteca *TimerOne* aparece no programa sob a forma `#include "TimerOne.h"`.

3.2 Variáveis

Esta secção utiliza informação disponível em (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016) e (Arduino, 2016).

Uma variável é uma forma de dar um nome e guardar um valor para usar ao longo do código, como por exemplo, dados de um sensor ou um valor intermediário de um cálculo.

3.2.1 Tipos de variáveis

Existem diversos tipos de variáveis, umas onde apenas se podem guardar números e outras onde pode ser guardado qualquer tipo de caráter, seja ele um número, uma letra ou um símbolo:

- **Int:** consegue guardar valores de -32767 até 32767 (apenas números inteiros). Exemplo: int var1 = 200;
- **unsigned long:** guarda até 32 *bits* (ou 4 *bytes*), mas não guarda valores negativos. Ou seja, este tipo de variável consegue guardar valores desde 0 até 4 294 967 295 (ou 2^{32} - 1). No caso de se utilizarem comandos de tempo, como por exemplo *micros*(), é necessário utilizar este tipo de variável;
- **float:** guarda valores entre $-3{,}4 \times 10^{-38}$ e $+3{,}4 \times 10^{+38}$, até 6 dígitos de precisão, como por exemplo, float var2 = -0.0489;
- **double:** guarda valores entre $-1{,}7 \times 10^{-308}$ e $+1{,}7 \times 10^{+308}$, até 10 dígitos de precisão, como por exemplo, double var3 = 0.478009;
- **char:** guarda caracteres, ou seja, letras, números e símbolos, como por exemplo, char var4 = 'A';
- **string:** consegue guardar conjuntos de caracteres, como por exemplo uma frase. Inicia-se como sendo do tipo *char*, mas no nome da variável, entre parêntesis retos coloca-se o número de carateres que serão colocados no seu interior, sendo essa informação colocada dentro de aspas duplas, como por exemplo char var5[20] = "Escola de Robótica!!".

3.2.2 Declarar variáveis

As variáveis devem ser declaradas antes de serem utilizadas. Declarar uma variável significa definir o seu tipo, e opcionalmente, configurar um valor inicial (iniciar a variável). As variáveis não precisam ser iniciadas quando são declaradas. No caso do protótipo desenvolvido neste projeto foram usadas bastantes, como se pode observar no excerto do código da Figura 20.

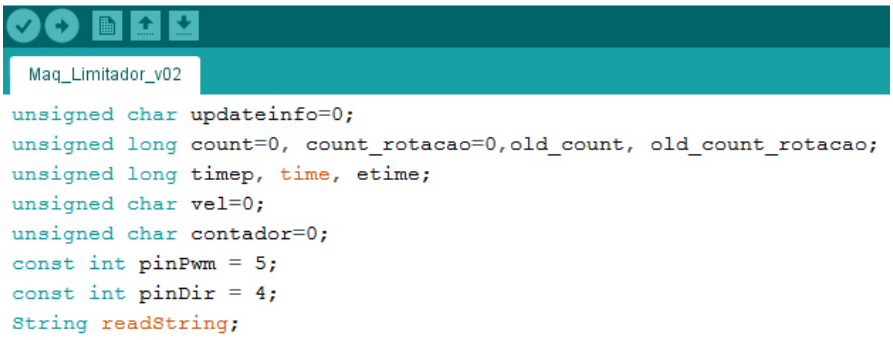


Figura 20 – Excerto de código com declaração de variáveis

Após serem declaradas ou iniciadas, as variáveis podem ser usadas de forma a atribuir, alterar ou utilizar um valor. Para atribuir ou alterar um valor de uma variável utiliza-se o operador =

que informa o código que deve atribuir o que estiver do lado direito do sinal de igual, dentro da variável que deve ficar do lado esquerdo do sinal de igual, como por exemplo:

- variavel1 = 7; // atribuir à variável o valor 7;
- variavel2 = analogRead(2); // atribuir a tensão lida da porta analógica 2 da placa *Arduino*.

3.3 *void setup* ()

Nesta secção a informação é baseada tanto no código utilizado no programa do protótipo desenvolvido neste projeto, bem como em (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016) e (Arduino, 2016).

O *void setup* () é normalmente usado para inicializar variáveis, modos de pinos e definir velocidade de transmissão *serial*. Por exemplo, caso se utilize um servomotor e se pretenda atribuir uma posição de inicialização, isto é, em que posição o servomotor deve estar situado antes do início das operações, usa-se o excerto do código apresentado na Figura 21.

```
void setup ()
{
  servomotor.attach(9); // Associar o servomotor ao pino em que esta conetado, neste caso o pino 9
  servomotor.write(posicao); // Colocar o servomotor na posicao inicial
  delay(1000);
}
```

Figura 21 – Exemplo de código *void setup* () (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016)

No caso do protótipo desenvolvido neste projeto o código é o apresentado na Figura 22.

Figura 22 – Excerto de código *void setup* ()

Como se pode observar na Figura 22, este bloco de código contém bastante informação que, por questões de compreensão, necessita de ser explicada. Assim, as primeiras 4 linhas de código (ver Figura 23) da Figura 22 são referentes ao controlo do motor, sendo que o *pinPwm* refere-se ao pino controlo de velocidade e o *pinDir* corresponde ao pino responsável pelo sentido de rotação. Destas 4 linhas, as primeiras duas definem os pinos como sendo saídas da placa *Arduino*, enquanto as outras duas inicializam esses mesmos pinos com um determinado valor. Neste caso o *pinPwm* vai ser inicializado com o valor da variável *vel* (neste caso a variável tem o valor 0, como se observa na Figura 20) e o *pinDir* vai ser inicializado com o valor *HIGH* que na placa *Arduino* corresponde ao valor de 1 (ou 5 V).

```
pinMode(pinPwm, OUTPUT);
pinMode(pinDir, OUTPUT);
analogWrite(pinPwm, vel);
digitalWrite(pinDir, HIGH);
```

Figura 23 – Excerto de código *void setup* () relativo ao controlo do motor

A próxima linha de código é referente à transmissão de dados para a consola *serial* da placa *Arduino*, que neste caso vai ser posteriormente utilizada para o envio e receção de dados para o dispositivo *Android*. Esta linha de código define a taxa de transmissão da consola *serial*, sendo que esta taxa pode ser definida com o valor de 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 ou 115200. Neste caso foi utilizado o valor de 9600.

De seguida são definidos os pinos referentes ao *encoder* (pistas A e B). Um sinal de um *encoder* é basicamente definido por uma onda retangular, sendo que o seu valor oscila entre 0 e 1, como demonstra a Figura 24.

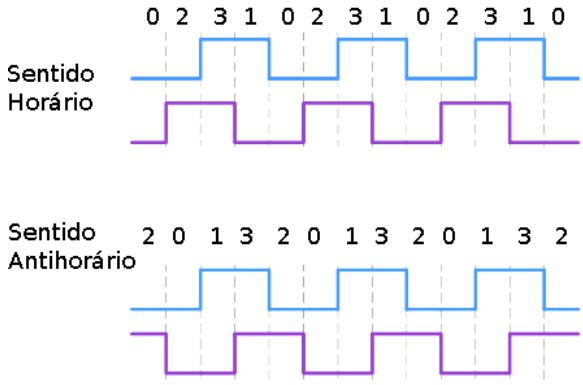


Figura 24 – Sinal de um *encoder*, pistas A e B (Eletronica 100 limites, 2012)

No protótipo é necessário apenas utilizar o sinal da pista A, mas já se encontra definido o sinal também para a pista B, como demonstra o excerto de código da Figura 25.

```
//sensor rotação pino 2 (pode-se alterar pelo Canal B do encoder)
pinMode(2, INPUT);
attachInterrupt(digitalPinToInterrupt(2),rotacao, RISING);
//Canal A no pino 3
pinMode(3, INPUT);
attachInterrupt(digitalPinToInterrupt(3),transition, CHANGE);
```

Figura 25 – Excerto de código *void setup* () relativo ao controlo do *encoder*

Para obter a velocidade é necessário calcular o número de impulsos em função do tempo. Numa primeira fase é necessário programar a placa *Arduino* para fazer a contagem desses impulsos. Para tal torna-se necessário recorrer a uma caraterística que estas possuem, que são as interrupções. No caso da placa *Arduino* utilizada (*Arduino Uno*) só os pinos 2 e 3 possuem esta característica (caso sejam necessárias mais interrupções ter-se-á de recorrer a uma placa *Arduino* mais avançada, como por exemplo a *Mega* ou a *Pro*). Quando existe uma variação positiva (*rising*), negativa (*falling*) ou ambas (*change*), as interrupções (linhas 2 e 4 do código da Figura 25) permitem executar uma função definida, neste caso a função *rotacao* e *transition* (explicadas na secção seguinte), que basicamente servem apenas para contar o número de impulsos.

Depois de se conseguir obter o número de impulsos é necessário defini-los em função do tempo Para isso são necessárias as últimas linhas do código Funçõesdo bloco *void setup* () referente à inicialização do temporizador (ver Figura 26).

```
//inicializar o tempo inicial
timep = micros();
Timer1.initialize(10000);
Timer1.attachInterrupt(int10ms);
```

Figura 26 – Excerto de código *void setup* () relativo ao controlo do temporizador

Esta parte do código é responsável pelo cálculo da velocidade a partir do *encoder*, visto que se for definido um intervalo de tempo entre contagens de impulsos, pode calcular-se o número de impulsos num determinado intervalo tempo e com esse valor calcular posteriormente um valor de velocidade. Neste caso o intervalo entre contagens é 10 000 microsegundos (ou 10 milisegundos), ou seja, obtêm-se um determinado valor de impulsos a cada 10 milisegundos. Isso é feito a partir de uma função *int10ms* que é feita exatamente a cada incremento de tempo definido no temporizador (linha 2 do código da Figura 26).

3.4 Funções

Também nesta secção a informação é baseada tanto no código utilizado no programa do protótipo deste projeto, bem como em (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016) e (Arduino, 2016).

As funções são usadas para criar pequenos pedaços de códigos, separados do programa principal. No caso do *Arduino* estas são colocadas fora do *void loop* ().

As funções são importantes pois devolvem valores, ajudam a fragmentar o código em partes menores e podem ser utilizadas mais de uma vez no mesmo programa, poupando tempo de programação e inúmeras linhas de código que fazem o mesmo.

Na Figura 27 observa-se um exemplo da criação de uma função e de como esta é utilizada no *void loop* (). Esta função soma dois valores e depois devolve o resultado ao programa principal.

```
int a,b,c;

int soma(int a,int b)
{
  c=a+b;
  return c;
}

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

void loop ()
{
  a=2;
  b=3;
  c = soma(a,b);
  Serial.print(c);
}
```

Figura 27 – Exemplo de código de uma função (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016)

No programa do protótipo foram feitas 3 funções (*rotacao*, *transition* e *int10ms*), todas elas com o objetivo de traduzir o sinal do *encoder* em velocidade. As duas primeiras servem apenas para fazer a contagem dos impulsos dos canais B e A do *encoder*, respetivamente. A terceira tem como objetivo guardar os valores das contagens em função do tempo definido (10 milisegundos) e reiniciar os contadores das duas primeiras funções. Na Figura 28 observa-se o código que permite executar o anteriormente descrito.

Maq_Limitador_v02 §

```
void int10ms()
{
  old_count=count;
  count=0;
  old_count_rotacao=count_rotacao;
  count_rotacao=0;
  updateinfo=1;
}

void transition()
{
  // incrementa os impulsos dados pelo Canal A do encoder
  count++;
}

void rotacao()
{
  // incrementa os impulsos dados pelo sensor do limitador de velocidade
  // (não é necessário, por isso, deve-se ponderar usar esta interrupção para
  // o Canal B do encoder)
  count_rotacao++;
}
```

Figura 28 –- Bloco de código com as funções

3.5 *void loop* ()

Esta é a última parte do programa, sendo que todo o código inserido nesta função é corrido repetitivamente e indefinidamente (ver Figura 27). À semelhança das secções anteriores, também esta parte se baseia em várias fontes de informação para além do código feito para este protótipo, como (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016) e (Arduino, 2016).

Esta parte do programa, por ser muito extensa e ter alguma complexidade, vai ser exposta em pequenas etapas de modo a facilitar a leitura e compreensão por parte do leitor.

É nesta parte do programa que tudo acontece, desde o cálculo da velocidade, ao envio e receção de dados do dispositivo *Android* e até ao controlo do motorredutor através dos mapas de ensaio definidos na secção 4.2.4.

Para começar foi desenvolvida uma parte de código para receber os dados do dispositivo *Android* (ver Figura 29). Esta parte de código recebe, quando disponível, informações do *Android* e guarda-as numa variável *c* que vai ser posteriormente utilizada. Esta parte é responsável pela receção dos comandos para o controlo do motorredutor (como se vai observar seguidamente).

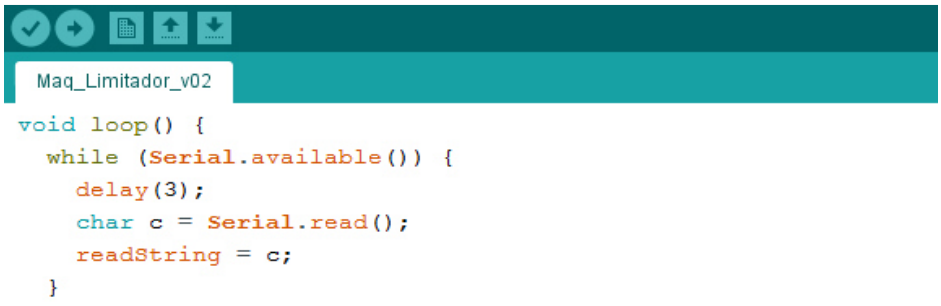


Figura 29 – Excerto de código *void loop* () relativo à receção de dados

Posteriormente é necessário definir os cálculos e as seguranças.

O excerto de código da Figura 30 representa o início das seguranças. Os dois comandos *if* que se encontram logo no princípio deste excerto de código definem que só se houver informações a serem recebidas do dispositivo *Android* (primeiro comando *if*) e se forem os comandos corretos vindos desse mesmo dispositivo (segundo comando *if*) é que todos os cálculos e mapas de controlo do motor são feitos.

```
if (readString.length() > 0){
  //Serial.println(readString);
  if (updateinfo==1 && (readString == "e" || readString == "q" || readString == "f" || readString =="d"))
  {
    updateinfo=0;
    // apresenta a conversão de impulsos por 10ms (no encoder do motor) para m/s da polia do limitador, com a seguinte fórmula:
    // ([nº impulsos/10ms]*(conversão para seg)*(1/([resol. encoder]*[rel. caixa motor]*[nº contagens/periodo]))*(d[mm]*PI/1000))
    Serial.print("V   ");
    //Serial.print(old_count*PI*0.2*100.00*120.00/(100.00*15.00*2.00*200.00));
    Serial.print(old_count*PI*0.004);
    Serial.println("    ");
        contador++;
```

Figura 30 – Excerto de código *void loop* () relativo ao cálculo e seguranças

A segunda parte de código é referente ao cálculo da velocidade a partir dos valores obtidos nas contagens de impulsos anteriores (ver secções 3.3 e 3.4). Torna-se necessário conhecer previamente o sistema que influencia a velocidade, de modo a obter a equação do cálculo de velocidade a inserir no código (aplicável a qualquer tipo de limitador de velocidade, independentemente do seu tamanho).

O sistema que influencia o cálculo da velocidade é o representado na Figura 31.

Figura 31 – Sistema definido para o cálculo de velocidade

Este sistema é composto pelo *encoder*, montado a montante do motor (localizado o mais à direita na Figura 31), pelo motorredutor e pela roda com *o-ring*.

O *encoder* é o responsável pelos valores a usar no cálculo. Como se pretende saber a velocidade da roda do limitador de velocidade, tem de se ter em conta todo este sistema, até se obter a velocidade desejada.

Para que se possa fazer o sistema independente é necessário ter em consideração que não existe escorregamento entre a roda do limitador de velocidade e a roda com *o-ring*. Mais concretamente, se não existir escorregamento entre estas duas rodas, pode-se admitir que estas vão ter exatamente a mesma velocidade linear. Desta forma não é necessário conhecer nenhum dado da roda do limitador de velocidade (à exceção da sua velocidade de disparo, definida pelo fabricante). De acordo com o que se acabou de descrever, pode-se afirmar que a velocidade linear do limitador de velocidade é igual à velocidade linear da roda com *o-ring*.

Ou seja:

$$Vel_{limitador} = Vel_{roda} \quad (1)$$

Seguidamente é necessário calcular a velocidade da roda com *o-ring*. No que diz respeito ao *encoder*, já se dispõe do valor do impulso a cada 10 milisegundos, sendo que agora é necessário descortinar todo o sistema até chegar à roda com *o-ring*. Primeiro é necessário obter, a partir deste valor, o número de rotações por segundo da roda com *o-ring*:

$$sinal\ do\ encoder\ obtido\ anteriormente\ = \frac{número\ de\ impulsos}{10\ milisegundos} \quad (2)$$

É necessário extrapolar o valor de impulsos para impulsos por segundo, sendo que então é necessário calcular o fator de conversão de tempo *Ft* (1 s corresponde a 1000 ms):

$$Ft = \frac{1000\ milisegundos}{intervalo\ entre\ medições} = \frac{1000\ milisegundos}{10\ milisegundos} = 100 \quad (3)$$

Substituindo:

$$sinal\ encoder \times Ft = \frac{número\ de\ impulsos}{10\ milisegundos} \times 100 \quad (4)$$

Assim obtém-se o valor de impulsos por segundo. De seguida é necessário aplicar outro fator de conversão que corresponde à programação da placa *Arduino* (secção 3.3) relativamente às interrupções, sendo que o fator de conversão de interrupção *Fa* pode ser:

- 1, se for utilizada a interrupção *Rising* ou *Falling*;
- 2, se for utilizada a interrupção *Change*.

Neste caso, como foi utilizada a interrupção *Change*, *Fa* é aplicado da seguinte forma:

$$sinal\ encoder \times Ft \times \frac{1}{Fa} = \frac{número\ de\ impulsos}{10\ milisegundos} \times 100 \times \frac{1}{2} \quad (5)$$

Agora é necessário traduzir este valor para rotações por segundo. Para isso é necessário recorrer à resolução do *encoder*, expressa em impulsos por rotação. Ou seja, se o valor anterior for multiplicado pelo inverso da resolução do *encoder* (100 impulsos por rotação), obtém-se o número de rotações por segundo junto ao *encoder*. Com isto pode-se escrever:

$$sinal\ encoder \times Ft \times \frac{1}{resolução\ encoder \times Fa} = \frac{nº\ rotações\ encoder}{segundo} \Leftrightarrow$$

$$\Leftrightarrow \frac{número\ de\ impulsos}{10\ milisegundos} \times 100 \times \frac{1}{100 \times 2} = \frac{nº\ rotações\ encoder}{segundo} \quad (6)$$

Desta forma, determina-se o valor de rotações por segundo junto ao *encoder*. Agora é necessário calcular o número de rotações por segundo da roda com *o-ring*. Como se pode observar na Figura 31, o único componente que ainda exerce influência é o motorredutor, sendo que a redução tem de ser aplicada neste cálculo. Sabendo que a relação do motorredutor é de 1 para 15 e que este valor deve ser multiplicado pelo seu inverso, obtém-se:

$$sinal\ encoder \times Ft \times \frac{1}{\frac{resolução}{encoder} \times Fa \times \frac{redução}{motor}} = \frac{nº\ rotações\ roda\ o-ring}{segundo} \Leftrightarrow$$

$$\Leftrightarrow \frac{número\ de\ impulsos}{10\ milisegundos} \times 100 \times \frac{1}{100 \times 2 \times 15} = \frac{nº\ rotações\ roda\ o-ring}{segundo} \quad (7)$$

Finalmente é necessário converter o valor de rotações por segundo para velocidade linear (m/s). Para isso tem-se:

$$Vel_{limitador} = Vel_{o-ring}[m/s] \Leftrightarrow$$

$$Vel_{limitador} = \frac{\pi \times \overset{diâmetro\ roda}{com\ o-ring}[mm] \times \frac{nº\ rot.roda\ o-ring}{segundo}}{1000} \quad (8)$$

Obtendo-se:

$$Vel_{limitador}[m/s] = \overset{sinal}{encoder} \times Ft \times \frac{\pi \times diâmetro\ roda\ o-ring\ [mm]}{\overset{resolução}{encoder} \times Fa \times \overset{redução}{motor} \times 1000} \quad (9)$$

Finalmente, sabendo que a roda do limitador de velocidade tem um diâmetro de 120 mm, substituem-se todos os valores na equação anterior:

$$Vel_{limitador}[m/s] = \overset{sinal}{encoder} \times 100 \times \frac{\pi \times 120}{100 \times 2 \times 15 \times 1000} \Leftrightarrow$$

$$\Leftrightarrow Vel_{limitador}[m/s] = sinal\ encoder \times \pi \times 0{,}004 \quad (10)$$

No excerto do código visível na Figura 30 é feito este cálculo e posteriormente inserido no dispositivo *Android* para envio de informação sob a forma "V [Valor da velocidade] ". Por exemplo, se a velocidade calculada for 1,32 m/s, a informação é enviada sob a forma "V 1.32 ". Os espaços em branco são propositados e servem para evitar erros na receção no dispositivo *Android*.

Finalmente, o restante código é referente ao controlo da velocidade do motor, sendo que no caso deste protótipo foram feitos 4 mapas de controlo:

- Mapa de ensaio com cenário em que o elevador se encontra em escorregamento linear e lento;
- Mapa de ensaio com cenário em que o elevador se encontra em queda livre;
- Desencravamento após o ensaio;
- Desligar o motorredutor.

Na Figura 32 observa-se o mapa de ensaio com o cenário mais usual, o de escorregamento. Este fenómeno ocorre com alguma frequência e pode ser causado por variadíssimos fatores, sendo que os mais importantes são o alongamento excessivo dos cabos e/ou roda de tração com desgaste.

```
if(contador>=50 && readString == "e")
{
  vel+=2.5;
  if(vel>=255)
  vel=255;
  analogWrite(pinPwm, vel);
  contador=0;
}
```

Figura 32 – Excerto de código *void loop* () relativo ao mapa de escorregamento

O código apresentado controla diretamente a placa de potência que converte o sinal PWM da placa *Arduino* na tensão de saída (diretamente proporcional à velocidade obtida). A placa de potência atua sobre o motorredutor recebendo um incremento constante no sinal PWM da placa *Arduino* de 2,5 por período, o que faz aumentar a velocidade do motor. O segundo bloco *if* serve apenas para segurança, porque um sinal PWM só pode ter valores entre 0 e 255, sendo que a placa *Arduino* pode dar erro quando é inserido um valor superior na saída.

Os restantes mapas têm um funcionamento semelhante e encontram-se no programa completo (ver Anexo 1 Programa Completo em *Arduino*).

4 Programação *Android*

Tal como já referido anteriormente, para o utilizador poder interagir com o protótipo criado foi necessário desenvolver uma aplicação, sob a forma de uma interface gráfica, para um dispositivo *Android*. Pretende-se ainda traçar, em tempo real, o gráfico velocidade linear-tempo. Para isso foi feito um programa em *Android* recorrendo ao *software App Inventor* desenvolvido no MIT.

O modo de apresentação deste capítulo é semelhante ao anterior e tem como objetivo mostrar a aplicação feita. Para além disso é suportado por várias fontes como (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016) e (MIT App Inventor, 2015).

Antes de mais o MIT App Inventor é um *software* "open source" que serve única e exclusivamente para criar aplicações *Android*. Faculta um editor de interface gráfico através do simples arrasto de componentes, como é o caso de botões, *labels*, recursos, listas, etc. Para utilizadores mais avançados, tem ainda um editor de blocos pré-programados que permitem executar tarefas, sem ser necessário criar um código escrito. (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016).

A programação divide-se em duas partes: *Designer* e *Blocks*. Na primeira parte é feita toda a interface gráfica, desde a colocação de botões, *labels*, textos estáticos, imagens. Na segunda fornecem-se as instruções para animar o programa. Por exemplo, quando se arrasta um botão na parte de *Designer* de seguida programa-se as ações que esse mesmo botão vai fazer quando ativado na parte de *Blocks*.

Para ser mais simples de compreender por parte do leitor separaram-se estas duas partes em duas secções distintas.

4.1 Página de desenho da interface da aplicação

A parte *Designer* ou página de desenho de interface da aplicação é constituída por 4 zonas fundamentais, a *Palette*, a *Viewer*, a *Components* e a *Properties* (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016).

Na zona *Palette* podem encontrar-se todos os tipos de componentes de uma aplicação, como é o caso dos botões, listas, caixas de seleção, as etiquetas.

A zona *Viewer* imita um ecrã de um dispositivo *Android*. Pode-se vero aspeto da futura aplicação e pode-se arrastar da zona *Palette* os componentes para o ecrã da zona *Viewer*.

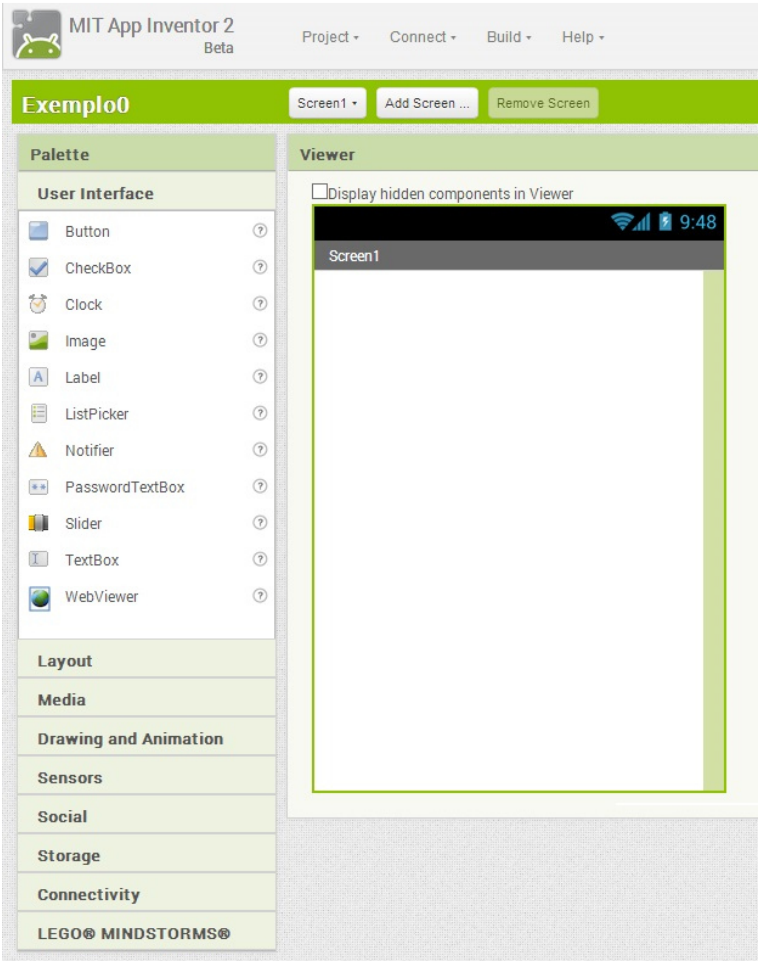


Figura 33 – Exemplo de página de desenho da interface (zonas *Pallets* e *Viewer*) (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016)

Na zona *Component*, pode-se encontrar a lista de componentes já colocadas na aplicação, sob a forma de uma hierarquia, desta forma percebe-se qual a dependência entre componentes.

Na zona *Properties* podem-se definir todas as propriedades de um componente. Por exemplo, se for um botão, vai-se à lista de componentes ou ao ecrã e seleciona-se o botão que se pretende alterar, aparecendo todas as propriedades do mesmo. Deste modo é possível alterar a cor, o texto, a posição, o tamanho.

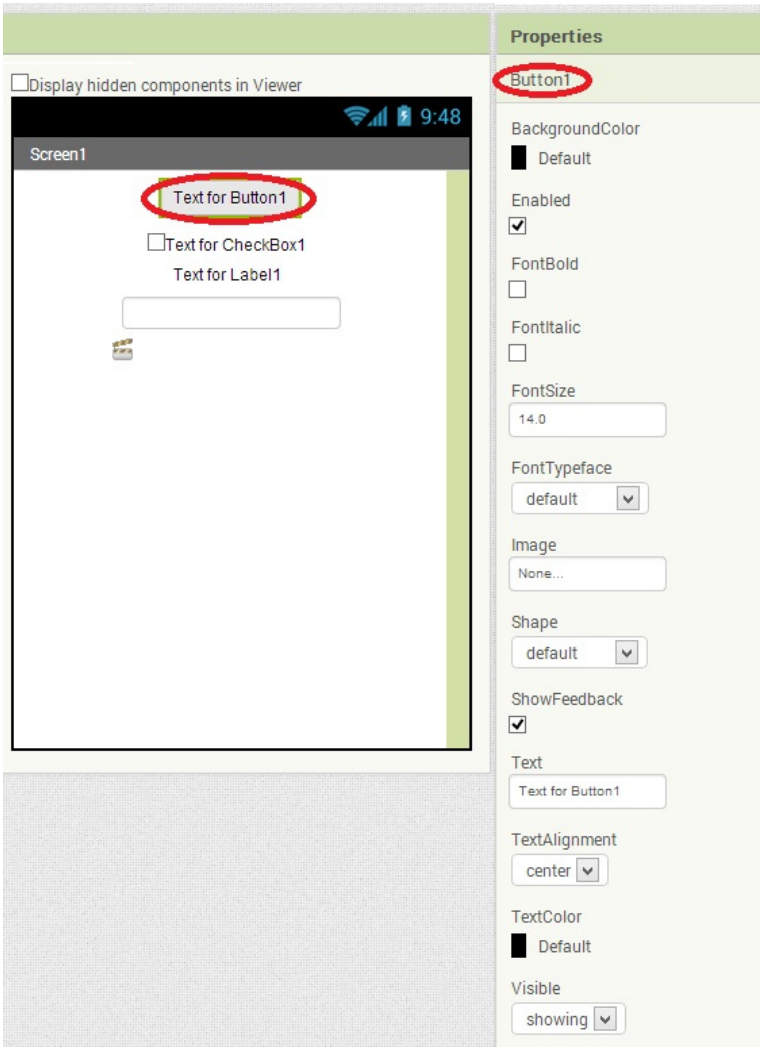


Figura 34 – Exemplo de página de desenho da interface (zona *Properties*) (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016)

No caso da aplicação desenvolvida para o protótipo, mesmo não sendo muito complexa, tem ainda assim muitos componentes, como se pode observar na Figura 35.

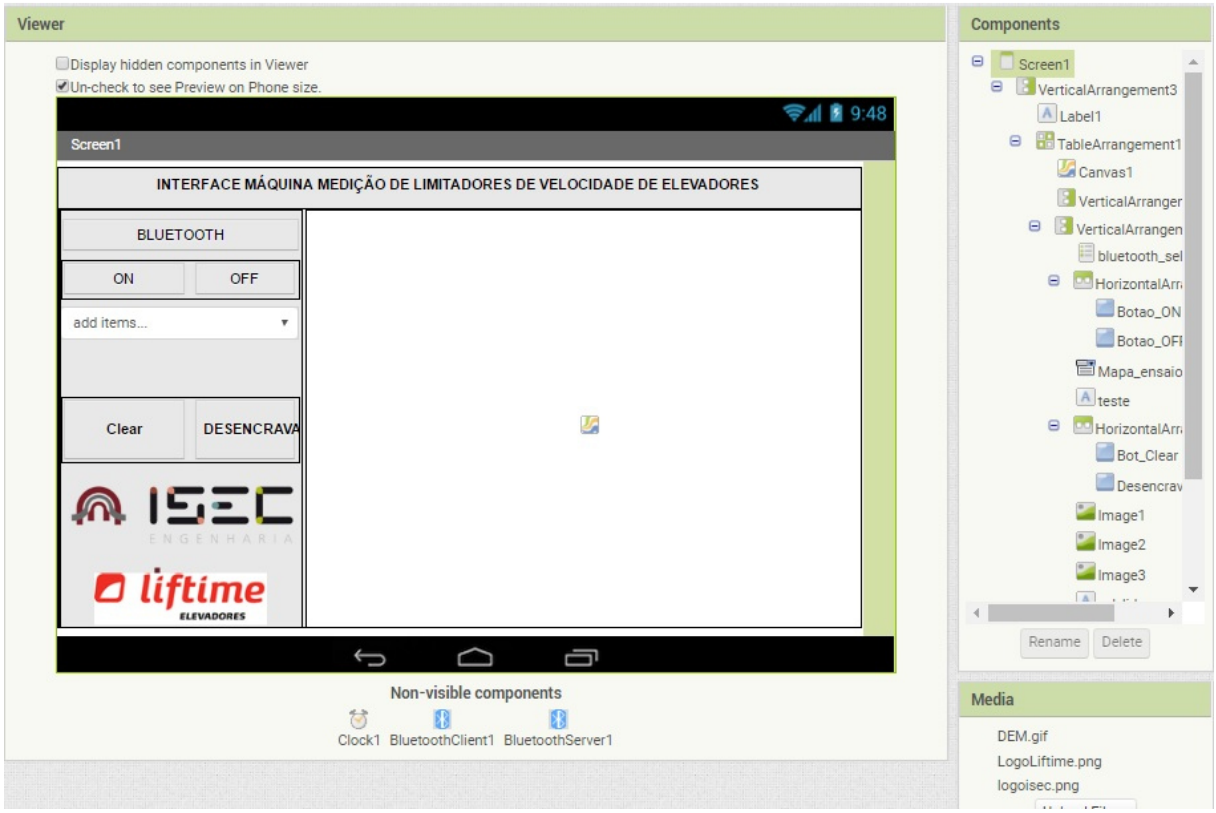


Figura 35 – Página de desenho da interface de aplicação do protótipo

A interface foi desenhada para ser bastante simples de usar pelo utilizador. Tem botões para:

- ligar o *Bluetooth*;
- ligar ou desligar o protótipo;
- escolher o mapa de ensaio;
- desencravar o motor;
- limpar a área destinada ao gráfico.

Depois tem apenas a área para fazer o gráfico velocidade linear-tempo (parte branca do lado direito na Figura 35) e por fim *labels* e imagens dos vários logotipos.

Como componentes não visíveis, tem um temporizador *clock1* e componentes para envio de dados *BluetoothServer1* e para receção de dados *BluetoothClient1* através do Bluetooth.

4.2 Página de desenho de blocos da aplicação

É nesta página que se fornecem as instruções para que tudo funcione como desejado. Para aceder a esta página seleciona-se a opção *Blocks* no canto superior esquerdo da página de desenho de interface da aplicação,como mostra a Figura 36.

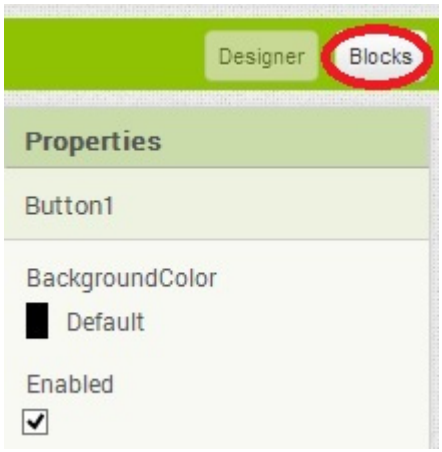


Figura 36 – Acesso à página de desenho de blocos da aplicação (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016)

Depois de selecionar esta opção acede-se à página de desenho de blocos da aplicação. Esta página é dividida em duas partes – *Viewer* e *Blocks* – como mostra a Figura 37.

Figura 37 – Exemplo de página de desenho de blocos da aplicação (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016)

Neste caso os blocos vão substituir o código escrito, sendo agrupados em vários conjuntos, dependendo do tipo (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016):

- **Control:** blocos de controlo (se isto acontecer então acontece aquilo, *if then*);
- **Logic:** blocos lógicos (verdadeiro ou falso);
- **Math:** blocos matemáticos;
- **Text:** blocos de texto (inserir, alterar, guardar texto);
- **Lists:** blocos de listas (criar listas de opções);
- **Colors:** blocos de cores (alterar a cor de um componente ou texto);
- **Variables:** blocos de variáveis (criar variáveis);
- **Procedures:** blocos de procedimentos (quando algo acontece é realizada uma ação).

Estes não são os únicos conjuntos que existem pois cada componente do interface (por exemplo um botão) também tem os seus próprios blocos.

O objetivo consiste em agrupar várias peças, como se de um *puzzle* se tratasse, para que façam aquilo que se pretende, como, por exemplo, a atribuição de um valor a uma variável, tal como demonstra a Figura 38). A programação é intuitiva e interativa, mesmo para que não esteja familiarizado com programação.

Figura 38 - Exemplo de agrupamento de blocos (Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes, 2016)

De seguida apresenta-se o programa de blocos desenvolvido para o protótipo. Como não existe uma ordem para se fazer este tipo de programas e no que diz respeito à programação destes blocos, utiliza-se o método mais lógico, para que o leitor consiga perceber a linha de pensamento utilizada.

4.2.1 Declaração de variáveis

Começando pela declaração de variáveis, foi elaborado um conjunto de blocos para a declaração das várias variáveis que irão posteriormente ser utilizadas em outros blocos, como demonstra a Figura 39.

Figura 39 – Declaração de variáveis do programa de blocos do protótipo

4.2.2 Inicialização da aplicação

É possível definir o que se pretende fazer quando a aplicação é aberta, desde aparecer determinada imagem, listas de seleção, entre outras hipóteses. No caso da aplicação desenvolvida, esta simplesmente efetua uma função externa *Reset* e coloca a lista de mapas de ensaio declarada nas variáveis no seletor.

Figura 40 – Bloco de inicialização da aplicação do protótipo

A função externa *Reset* tem como objetivo repor toda a interface num estado inicial. Ou seja, neste caso são reiniciados os valores de algumas variáveis, são executadas todas as linhas e *labels* do gráfico inicial (blocos roxos da Figura 41) é programada a configuração das linhas do gráfico (como a cor e a espessura da linha) e ainda é inserido na interface o valor da velocidade máxima.

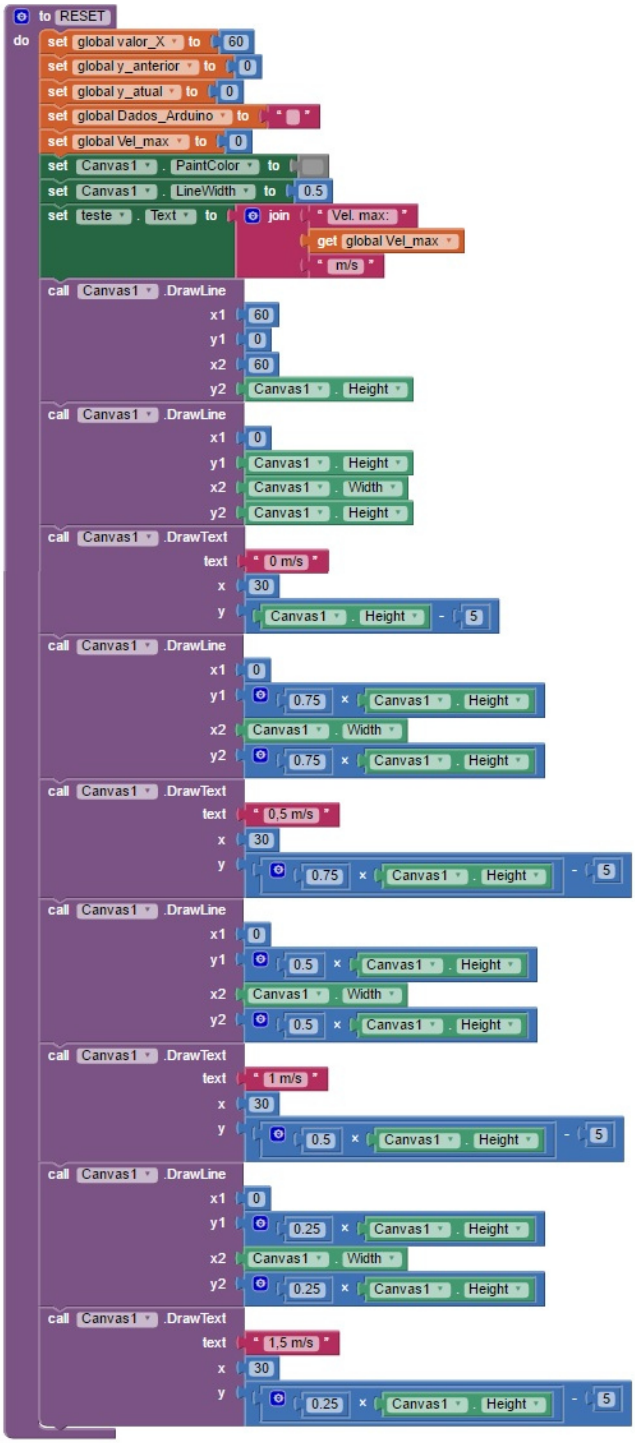


Figura 41 – Bloco de função externa "Reset" do programa do protótipo

Com isto é obtida uma interface inicial idêntica à da Figura 42, onde se pode visualizar a formatação do gráfico no que diz respeito às linhas de velocidade bem como à sua identificação, com a colocação de valores predefinidos de velocidade no ecrã (*labels* de 0, 0,5, 1 e 1,5 m/s).

Figura 42 – Interface inicial do programa do protótipo

4.2.3 Ligação *Bluetooth*

Como já referido, o protótipo funciona através de uma ligação por *Bluetooth* entre a placa *Arduino* e o dispositivo *Android*. Então, neste programa de blocos, é necessário informar o programa para efetuar essa ligação. Os blocos apresentados na Figura 43 servem para programar a apresentação dos dispositivos ativos e para fazer a ligação aquele que for selecionado.

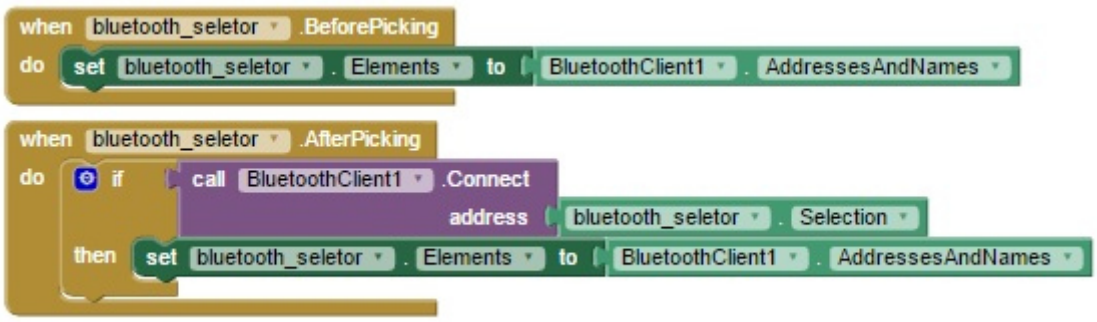


Figura 43 – Blocos para ligação *Bluetooth* do protótipo

Quando o botão de *Bluetooth*, presente na página de desenho da interface, é pressionado e devido à programação feita com os blocos anteriores, deverá aparecer um seletor idêntico ao da Figura 44.

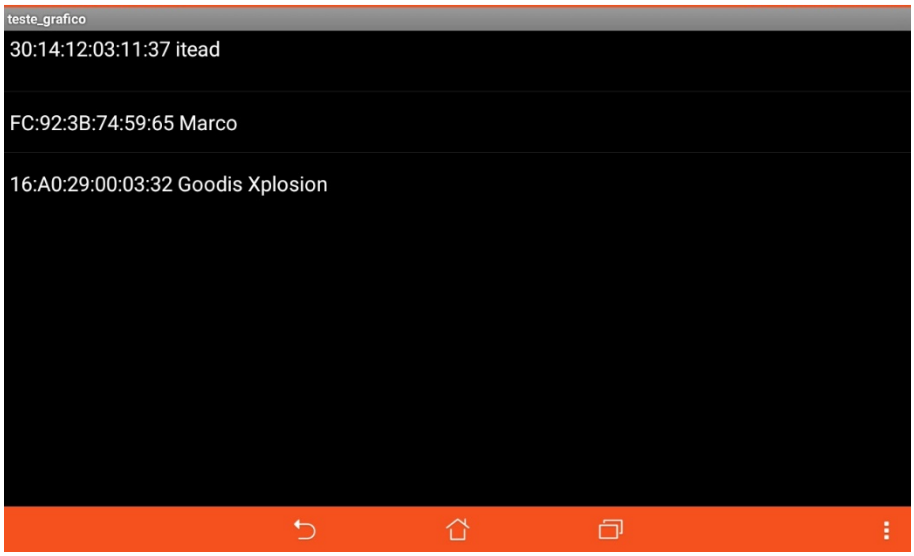


Figura 44 – Seletor de ligações *Bluetooth* do programa do protótipo

4.2.4 Programação dos botões

Como se pode ver na interface inicial da Figura 42, para além do botão para a ligação do *Bluetooth*, o programa possui mais alguns botões, sendo que estes são os seguintes:

- Botão *ON*;
- Botão *OFF*;
- Lista de mapas de ensaio;
- *Clear*;
- Desencravar.

Todos estes botões têm ações internas que necessitam de ser programadas. Para isso utilizaram-se os blocos da Figura 45.

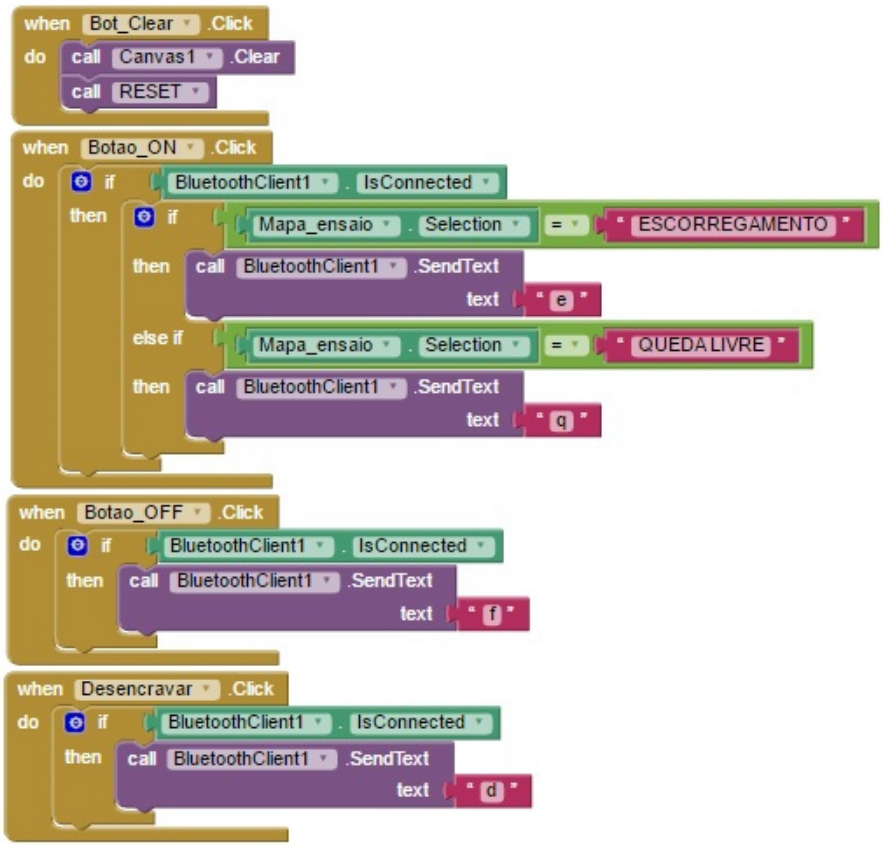


Figura 45 – Blocos de programação dos botões do programa do protótipo

Nestes blocos pode-se observar que o botão *ON* envia a informação para a placa *Arduino* com a letra que corresponde ao mapa selecionado (*e* para o mapa de escorregamento e *q* para queda livre). O botão *OFF* envia também a letra que corresponde a desligar o motor no programa do *Arduino* (letra *f*). O botão *Clear* limpa a área do gráfico e posteriormente executa a função externa *Reset* (explicada na secção 4.2.2). Finalmente o botão Desencravar envia, à semelhança dos dois anteriores, a letra correspondente ao comando responsável por efetuar o desencravamento do limitador após o ensaio estar concluído.

4.2.5 Obtenção de valores em tempo real

Como já referido anteriormente, o programa deve ser capaz de efetuar em tempo real o gráfico velocidade linear-–tempo. Para isso é necessário inicialmente programar o dispositivo *Android* para que receba e trate os dados enviados pela placa *Arduino*, sendo que para isso foi utilizado um temporizador *Clock1*. Este temporizador tem a função de receber e tratar os valores enviados pela placa *Arduino*, bem como inseri-los posteriormente na área do gráfico. Para isso foi necessário programar o bloco referente ao temporizador, conforme se pode observar na Figura 46.

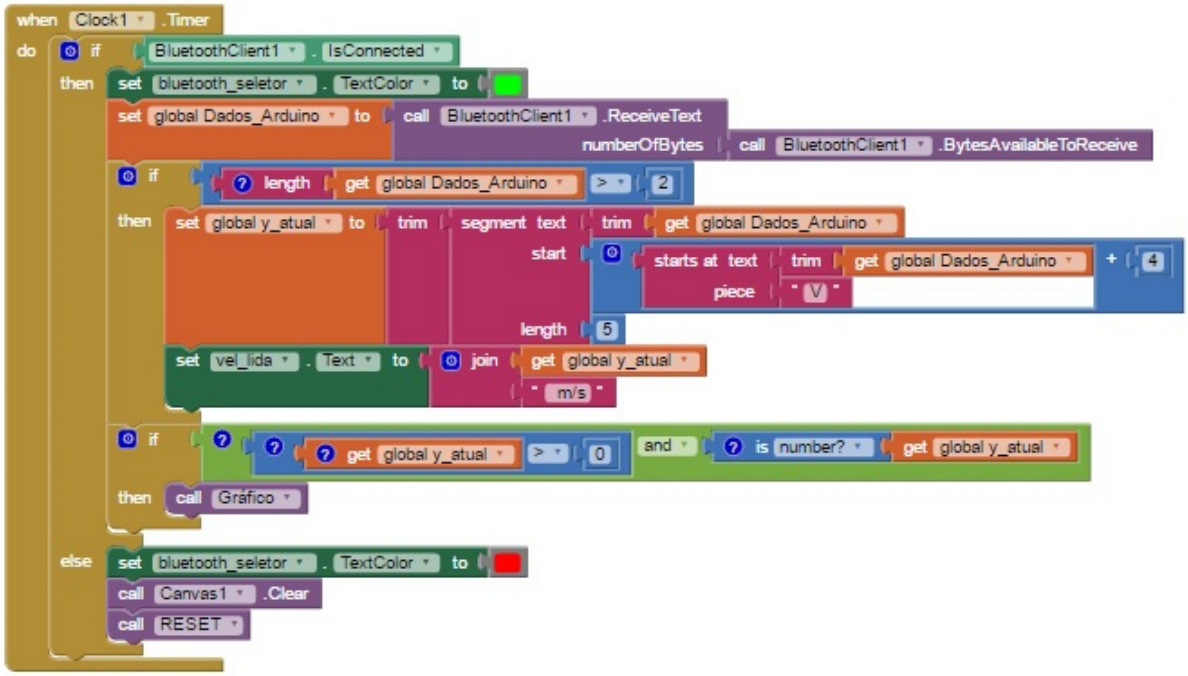


Figura 46 – Bloco de temporizador do programa do protótipo

Este temporizador é regido por um comando *if then else* que controla se existe ou não uma ligação ativa de *Bluetooth*. Se não existir é executada a parte referente ao comando *else*, onde apenas é limpa a área do gráfico e posteriormente é efetuada a função externa *Reset* (ver secção 4.2.4). Se existir uma ligação ativa então é executado todo o restante conjunto de comandos que integram o comando *then*. Como consequência, é feito, pela ordem que se apresenta na Figura 46, o seguinte:

- Recebe os dados em bruto da placa *Arduino* (como descrito na secção 3.5), logo após a obtenção da equação de velocidade);
- Como existe um atraso entre o envio dos dados da placa *Arduino* e a posterior receção por parte do dispositivo *Android*, existe a possibilidade de perder parte do número referente à velocidade calculada. Para o evitar foi necessário incorporar (antes e depois) do valor de velocidade um conjunto de espaços em branco para que, mesmo que desaparecessem alguns espaços em branco não desaparecesse o valor da velocidade. Com esta solução existe um problema, a *string* pode ser tão desfasada por parte do programa *Android* que o valor da velocidade pode se perder. Para solucionar isso foi inserido no início da *string* de envio de informação proveniente da placa *Arduino* a letra *V*. Com isto, o que o programa faz nesta fase é procurar a letra *V* e corta 5 caracteres após 4 da letra *V* e posteriormente retira os espaços em branco que ainda podem restar;
- Após isso é chamada a função externa Gráfico, explicada na secção seguinte.

4.2.6 Função externa Gráfico

Esta função tem como objetivo traçar o gráfico e fazer o cálculo obtenção da velocidade máxima recebida através da placa Arduino. A função é composta pelos blocos que se podem observar na Figura 47

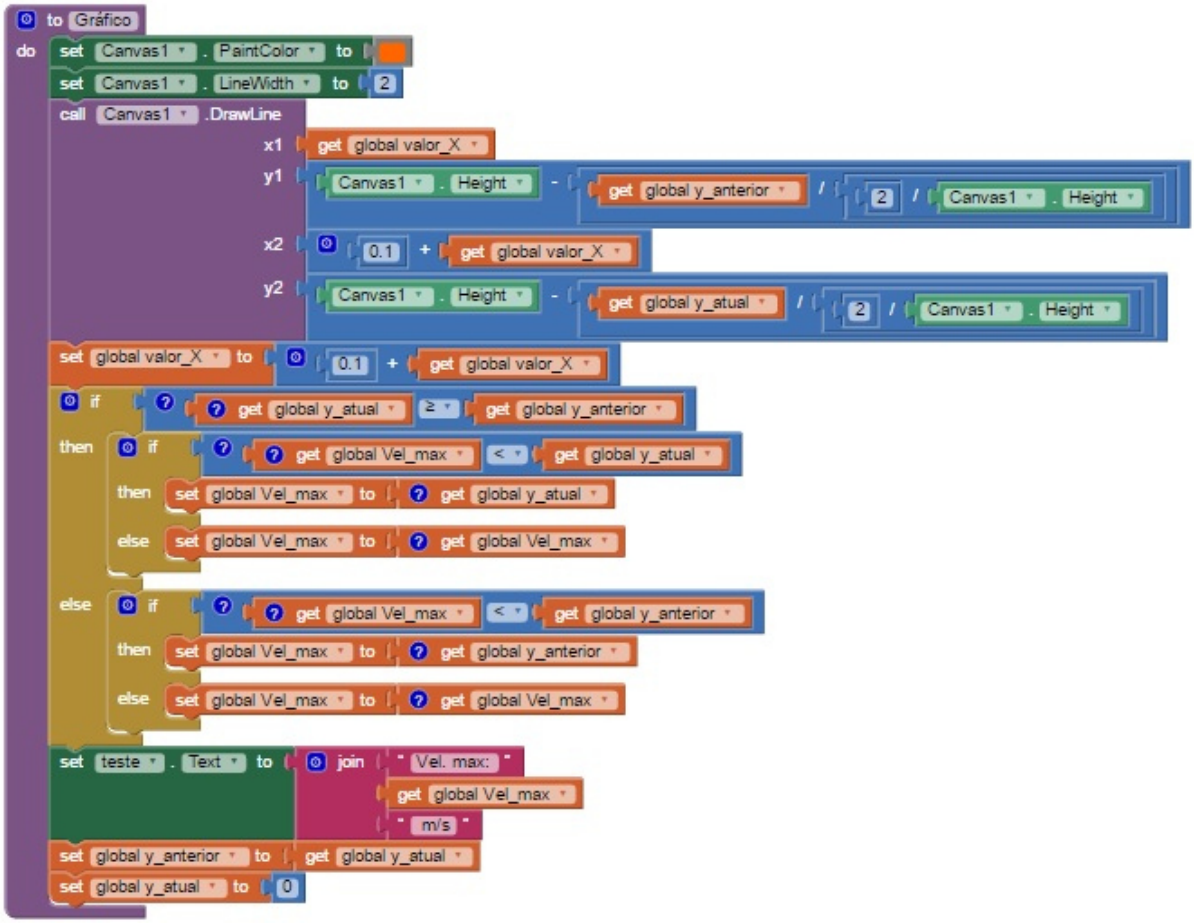


Figura 47 – Função externa Gráfico do programa do protótipo

O comando responsável por traçar o gráfico é *Canvas1.DrawLine* (bloco roxo na Figura 47) sendo que este tem um ligeiro contratempo, pois necessita de um conjunto de coordenadas iniciais (*x1* e *y1*) e outro conjunto de coordenadas finais (*x2* e *y2*) Na realidade este comando traça uma sucessão de pequenos segmentos de reta ligados entre si. Sabendo que o eixo horizontal corresponde ao tempo, este é definido por incrementos constantes e, sendo assim, só é necessário acrescentar uma constante ao valor anterior, ou seja, $x2 = x1 + c$ (c é uma constante igual a 0,1).

Já no eixo vertical, que corresponde ao da velocidade linear, este pressuposto não pode ser aplicado, visto que não é conhecido nenhum dos valores. Então, para solucionar este problema foram criadas duas variáveis para guardar valores de velocidade (y_atual e y_anterior). A primeira variável guarda a velocidade lida em tempo real e a outra guarda a velocidade lida imediatamente antes. A variável y_atual é obtida diretamente do temporizador

(ver secção 4.2.5) e y_anterior é obtida após se fazer um segmento de reta e imediatamente antes de receber outro valor de y_atual e é feito igualando uma à outra, ou seja, é atribuído o valor de y_atual à variável y_anterior e assim guarda-se o valor. Posteriormente é recebido um novo valor proveniente da placa *Arduino* que é atribuído a y_atual, ficando assim com os valores de velocidade real e a lida imediatamente antes, permitindo assim fazer um gráfico contínuo.

Devido a vários fatores, tais como o possível escorregamento, falhas de comunicação e/ou programação e ainda erros de cálculo, não se pode garantir a 100% que a velocidade máxima seja sempre a lida em tempo real, por isso é necessário fazer uma comparação para descobrir a velocidade máxima. Isto é feito através de dois blocos *if* que têm como objetivo comparar a variável velocidade máxima Vel_max com y_atual e com y_anterior de maneira a obter a maior entre as três e atribuindo esse valor a Vel_max. Entretanto é enviado para a interface o valor de velocidade máxima mais recente, de modo que, quando o limitador de velocidade tranca, se disponha da maior velocidade obtida para posteriormente ser comparada com a velocidade de disparo do limitador de velocidade.

4.2.7 Resultado obtido

Mediante toda esta programação, tanto de *Arduino* bem como de *Android*, obtém-se um relatório final, após o ensaio terminar, do qual a Figura 48 é um exemplo.

Figura 48 – Exemplo de relatório final obtido após o ensaio

5 Versão final do equipamento

Este capítulo descreve uma possível versão final do equipamento para avaliação de velocidade de limitadores de velocidade de elevadores que poderá ser construído no futuro partir deste protótipo.

Os principais requisitos a observar neste equipamento são:

- **Portabilidade:** para o instalar diretamente no limitador de velocidade de modo que não seja necessário retirar o limitador de velocidade do elevador para se fazer o ensaio numa bancada;
- **Compacidade:** devido ao local especifico, de difícil acesso, onde se encontra montado o limitador de velocidade no elevador;
- **Facilidade de instalação**: para reduzir ao máximo o tempo de ensaio (e com isso o seu custo),

Da análise destes requisitos obteve-se uma versão final do equipamento, o qual recorre a suportes diretos no limitador de velocidade e um sistema de fácil instalação. Na Figura 49 observa-se o equipamento completo e instalado num limitador de velocidade genérico.

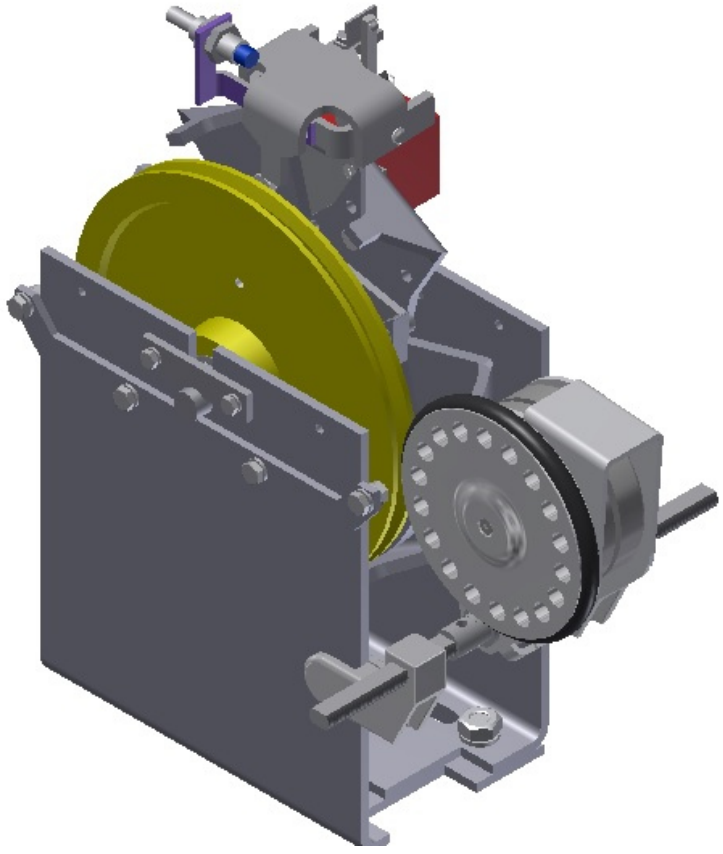


Figura 49 – Versão final do equipamento (modelo 3D)

O equipamento é composto por vários sistemas de fixação. Antes desses sistemas serem explicados tem de se definir qual seu o princípio de funcionamento. Foi idealizado para funcionar rodando num pino fixo que albergasse uma mola de torção, ou seja, fizesse o jogo de rotação que pode ser visto na Figura 50.

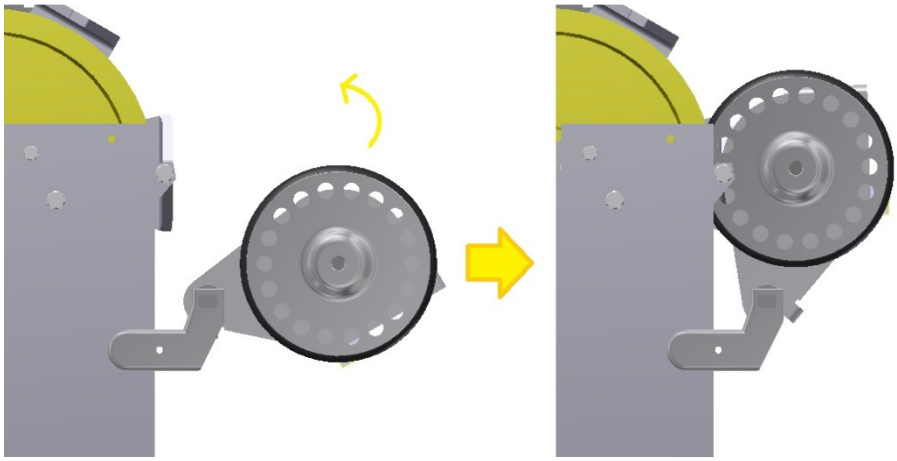


Figura 50 – Jogo de rotação sob um pino fixo com uma mola de torção

A mola de torção e o pino fixo são usados para garantir que a roda do equipamento vá de encontro à roda do limitador de velocidade e exerça a força necessária para que não ocorra escorregamento entre elas.

Para além disso deve existir a possibilidade de fazer uma afinação longitudinal. Para isso foi idealizado um conjunto de fixação baseado em componentes de fixação utilizados nos vidros, como se pode observar na Figura 51.

Figura 51 – Suportes para fixação de vidros (Batista Gomes, Lda., 2010)

É necessário ter em consideração que o sistema de suporte do equipamento tem de conter o pino de rotação (e fazer a ligação com o suporte montado no limitador de velocidade), tivesse afinação longitudinal e fosse fácil de montar. Obteve-se o sistema visível na Figura 52.

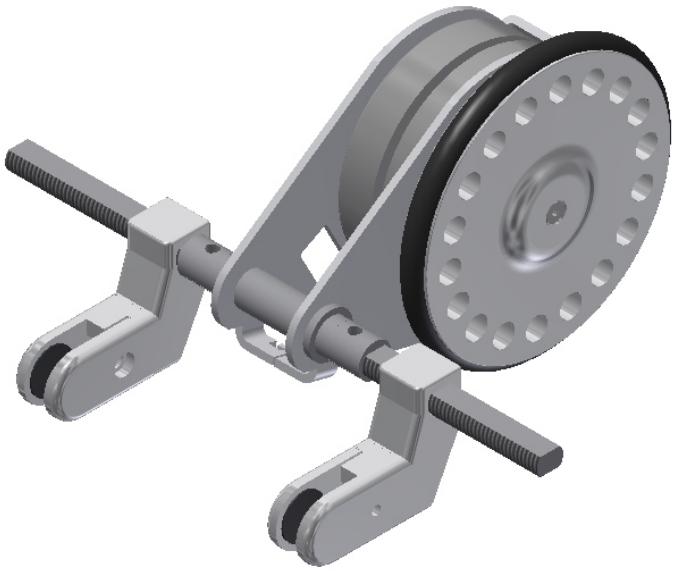


Figura 52 – Sistema de suporte do equipamento

Para ser mais fácil de entender, a Figura 53 mostra o mesmo sistema de suporte, mas numa vista explodida. O pino de rotação da máquina é acoplado a duas peças roscadas e faceadas por intermédio de chavetas. Estas peças roscadas têm esta configuração para que só permitam a rotação da máquina em si e não do pino de rotação da máquina e, ao simultaneamente, permitam um ajuste longitudinal.

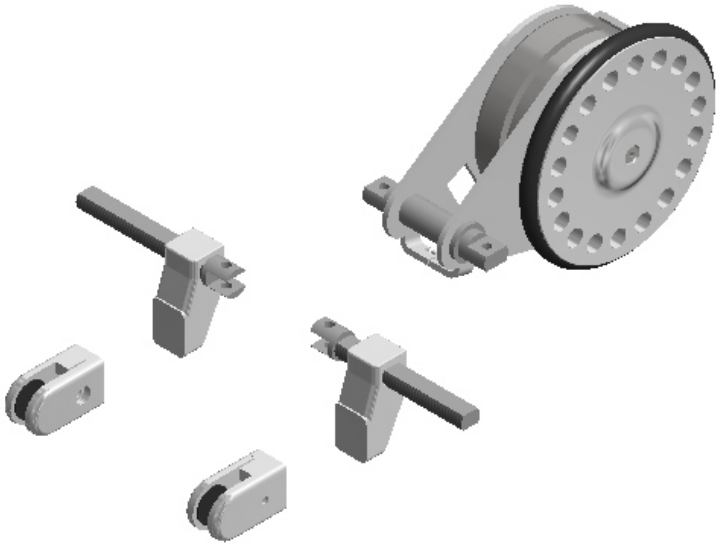


Figura 53 – Vista explodida do sistema de suporte do equipamento

O ajuste longitudinal é, como já referido, obtido através das peças roscadas. O ajuste efetua-se fazendo deslizar porcas por estes varões. Deste modo, o equipamento desloca-se longitudinalmente em ambos os sentidos, como demonstrado na Figura 54.

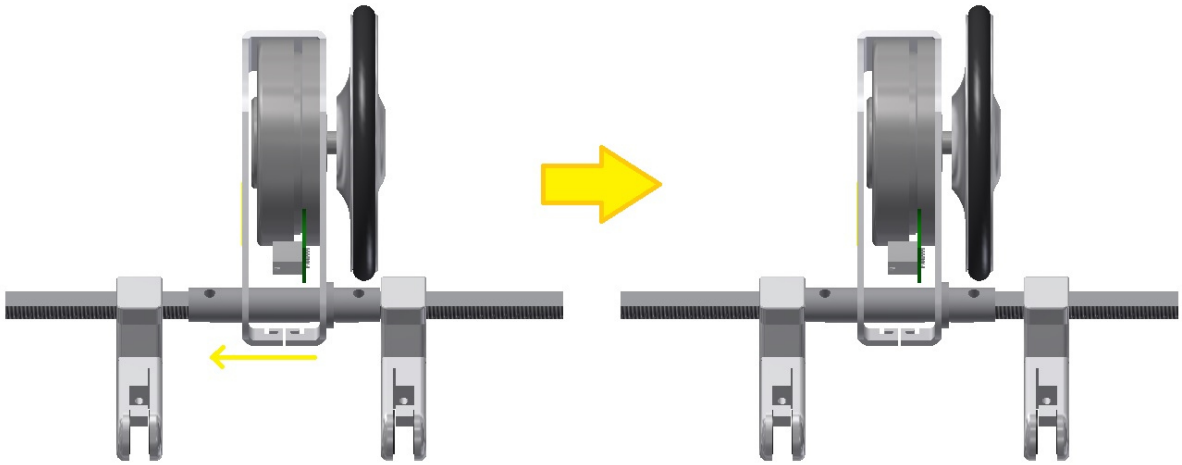


Figura 54 –- Ajuste longitudinal do equipamento

Seguidamente descrevem-se algumas sugestões de melhorias a efetuar aquando da construção da versão final do equipamento.

Melhorara programação, tanto em *Arduino* como em *Android*, para resolver alguns possíveis erros, para melhorar a interface gráfica e também para introduzir novas funcionalidades, como por exemplo:

- Incluir novos mapas de ensaio;
- Incluir mapas para permitir o ensaio nos dois sentidos de rotação (ensaiar descida/subida do elevador devido ao facto de já existirem no mercado limitadores de velocidade que disparam nos dois sentidos);
- Elabora relatórios que poderiam ser enviados automaticamente para o *email* geral da entidade responsável pelo ensaio, para posteriormente ser encaminhado para a entidade certificadora do elevador e assim agilizar todo o processo burocrático;
- Incorporar no programa *Android* um tutorial relativo à instalação do equipamento nos vários limitadores existentes nos elevadores;
- Criar no programa *Android* uma base de dados com as características dos limitadores de velocidade, para que o equipamento obtenha logo um resultado em que o limitador de velocidade é aprovado ou reprovado sem que haja intervenção do operador, evitando deste modo erros de leitura.

Reduzir erros de medição referentes à construção mecânica do equipamento, como por exemplo:

- Inserir um coeficiente que afete a velocidade obtida devido à força aplicada entre as rodas em contacto (obtida no equipamento pela mola de torção). Pode ser calculado experimentalmente através da colocação de um potenciómetro que meça o deslocamento da mola e com isso a

força aplicada. Posteriormente efetua-se um estudo em que a roda com *o-ring* é submetida a vários ensaios com aplicação de várias forças contra uma roda com diâmetro calibrado. Desta forma obtém-se um gráfico de diâmetro da roda com *o-ring* em função da força aplicada;

- O motorredutor deverá ter uma relação de redução o mais eficiente possível ou mesmo nem ter redução para assim eliminar possíveis erros devido às folgas entre o motor e a roda com *o-ring*. Para evitar isto pode também ser aplicado o *encoder* diretamente no eixo da roda com *o-ring*, mas tem de ser considerada a aplicação de um *encoder* com uma resolução muito elevada (idealmente acima de 1000 – 2000 CPR), o que pode fazer com que a que a placa *Arduino* não tenha capacidade suficiente para o suportar.

Como explicado neste documento, a obtenção da velocidade é efetuada a cada 10 milisegundos, fazendo com que possam existir erros na medição, uma vez que idealmente deveria ser medido o tempo entre impulsos e não ter uma estimativa a cada 10 milisegundos. Isto pode ser reduzido com um *encoder* de maior resolução, aplicado da mesma forma que no protótipo, e assim poder reduzir o intervalo entre medições de velocidade. Em alternativa se for usado o mesmo intervalo de tempo é melhorada a estimativa, visto que há um aumento de impulsos entre cada medição. A ideia a reter é que quanto maior for o número de impulsos entre cada leitura, menor será o erro da medição.

6 Conclusões

Este capítulo tem como finalidade descrever as principais conclusões obtidas no desenvolver deste projeto.

Como já referido, o objetivo do projeto era o desenvolvimento e construção de um protótipo de um equipamento que medisse a velocidade de um limitador de velocidade de um elevador, melhorando assim os métodos de ensaio utilizados atualmente. Após a construção do protótipo e efetuados alguns ensaios do mesmo, verificou-se que a solução apresentada resultou em pleno. De facto obtém uma medição da velocidade com bastante precisão (é necessário ter em consideração que o protótipo foi elaborado com alguns componentes mecânicos (motorredutor e o limitador de velocidade) usados e em condições que já não permitiam a sua utilização normal.

Para além disso, e como se pode concluir da construção do protótipo, o contacto entre as duas rodas era apenas garantido através do suporte de fixação do motorredutor. Como este tem uma construção arcaica, nem sempre se conseguia uma medição correta, visto que por vezes a roda escorregava (patinava) após o encravamento do limitador.

Este projeto foi uma experiência que fortaleceu, não só as capacidades profissionais e intelectuais, visto que foi necessário aprender a programar em *Arduino* e em *App Inventor* e portanto aumentou as competências profissionais. Também foi uma experiência fantástica no que diz respeito à ligação com a empresa Liftime Elevadores.

Em termos pessoais considero que fui acolhido de braços abertos pela empresa e esta permitiu-me ter uma visão da vida profissional, tendo estado em contacto permanente com muitas empresas de engenharia e tendo aprendido métodos de trabalho e competências que complementam tudo o que obtive na minha formação escolar ao longo de todos estes anos.

Em jeito de conclusão final, foi uma honra ter tido a oportunidade de poder ter feito este projeto, tendo também sido um desgosto não ter sido possível completar totalmente o projeto e construir uma versão final do equipamento. Ainda assim foi muito desafiante e gratificante o caminho até à obtenção do protótipo funcional, visto que não tinha qualquer noção de programação em nenhum dos programas utilizados, o que permitiu aprender bastante sobre a programação tanto de *Arduino* como de *App Inventor*.

Bibliografia

Arduino. (2016). Arduino Uno SMD Rev3. Obtido em 08 de Setembro de 2016, de https://www.arduino.cc/en/Main/ArduinoBoardUnoSMD

Arduino. (2016). What is Arduino? Obtido em 10 de Setembro de 2016, de https://www.arduino.cc/en/Guide/Introduction

Atlas Schindler. (2015). *Elevadores Atlas Schindler*. Obtido em 20 de Julho de 2016, de http://www.schindler.com/content/dam/web/br/PDFs/NI/manual-transporte-vertical.pdf

Avago Technologies. (2016). Encoder HEDS 5500#C14. Obtido em 9 de Setembro de 2016, de http://www.avagotech.com/products/motion-control-encoders/incremental-encoders/transmissive-encoders/heds-5500c14#specifications

Batista Gomes, Lda. (2010). Suportes para painéis em vidro. Zona Indústrial Sul, Variante Recardães-Barrô - Águeda, Aveiro, Portugal. Obtido em 14 de Setembro de 2016, de http://www.batista-gomes.pt/produtos.htm?idcont=66

Cytron Technologies Sdn. (2016). Placa de Potência Cytron MD10. Obtido em 09 de Setembro de 2016, de http://www.cytron.com.my/p-shield-md10

Digi-Key Electronics. (2016). Encoder HEDS 5500. Obtido em 9 de Setembro de 2016, de http://www.digikey.com/product-detail/en/broadcom-limited/HEDS-5500-A12/516-2755-ND/2219628

Eletronica 100 limites. (3 de Agosto de 2012). Precisando de um encoder? Monte o seu! Obtido em 10 de Setembor de 2016, de http://eletronica100limites.blogspot.pt/2012/08/precisando-de-um-encoder-monte-o-seu.html

Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes. (2016). Tutorial de Programação Android. Castelo Branco, Castelo Branco, Portugal. Obtido em 11 de Setembro de 2016, de http://escoladerobotica.ipcb.pt/?page_id=439

Escola Superior de Tecnologia – Laboratório de Robótica e Equipamentos Inteligentes. (2016). Tutorial de Programação Arduino – Lição 3. Castelo Branco, Castelo Branco, Portugal. Obtido em 10 de Setembro de 2016, de http://escoladerobotica.ipcb.pt/?page_id=297

Gervall, S.A. (2014). Espanha. Obtido em 6 de Setembro de 2016, de http://www.efalift.com/imgs/produtos/125259_1_3980_limitador-Gerval.jpg

Gomes, C. F. (Setembro de 2012). Manual de Manutenção Preventiva de Sistemas de Elevação Vertical para Transporte de Pessoas . Obtido em 26 de Maio de 2016, de http://repositorio.ipl.pt/bitstream/10400.21/2374/2/Disserta%C3%A7%C3%A3o.pdf

Googglet. (2015). Obtido em 7 de Setembro de 2016, de http://image.made-in-china.com/2f0j00DsQabBArHTqC/Elevator-Safety-Gear-001.jpg

ITEAD Intelligent Systems Co.Ltd. (2016). Obtido em 9 de Setembro de 2016, de https://www.itead.cc/electronic-brick-hc06-serial-bluetooth-brick.html

KAG - Kählig Antriebstechnik GmbH. (Fevereiro de 2016). *Kählig Antriebstechnik GmbH.* Obtido em 20 de Janeiro de 2016, de http://www.kag-hannover.com/wp-content/uploads/datasheets/en/Standard/Motoren_Schneckengetriebe/M48_SN31_Eng.pdf

Kaskus. (6 de Fevereiro de 2016). Obtido em 4 de Setembro de 2016, de http://s.kaskus.id/images/2016/02/06/3339886_20160206023858.jpg

Liftime Elevadores. (2006). *Empresa/Página Liftime Elevadores*. Obtido em 1 de Junho de 2016, de Página oficial Liftime Elevadores: http://www.liftime.pt/

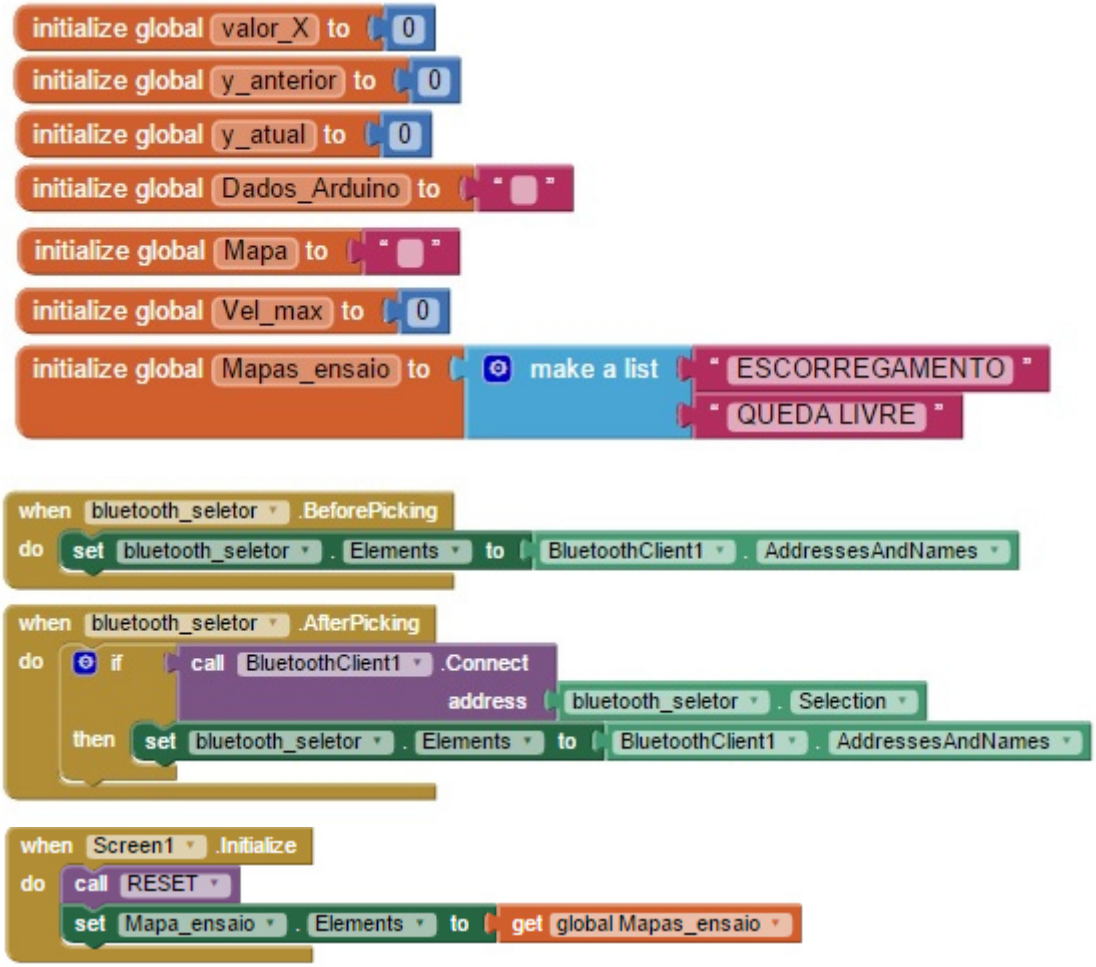
MIT App Inventor. (2015). About Us. (M. I. Technology, Ed.) USA. Obtido em 6 de Setembro de 2016, de http://appinventor.mit.edu/explore/about-us.html

Norma Portuguesa NP EN 81-1:2000. (2000). *Regras de segurança para o fabrico e instalação de ascensores - Parte.* IPQ - Instituto Português da Qualidade. Obtido em 9 de Junho de 2016, de http://www.thyssenkrupp-elevadores.pt/pdf/pt/en81_1_2000ascensoreselectricos.pdf

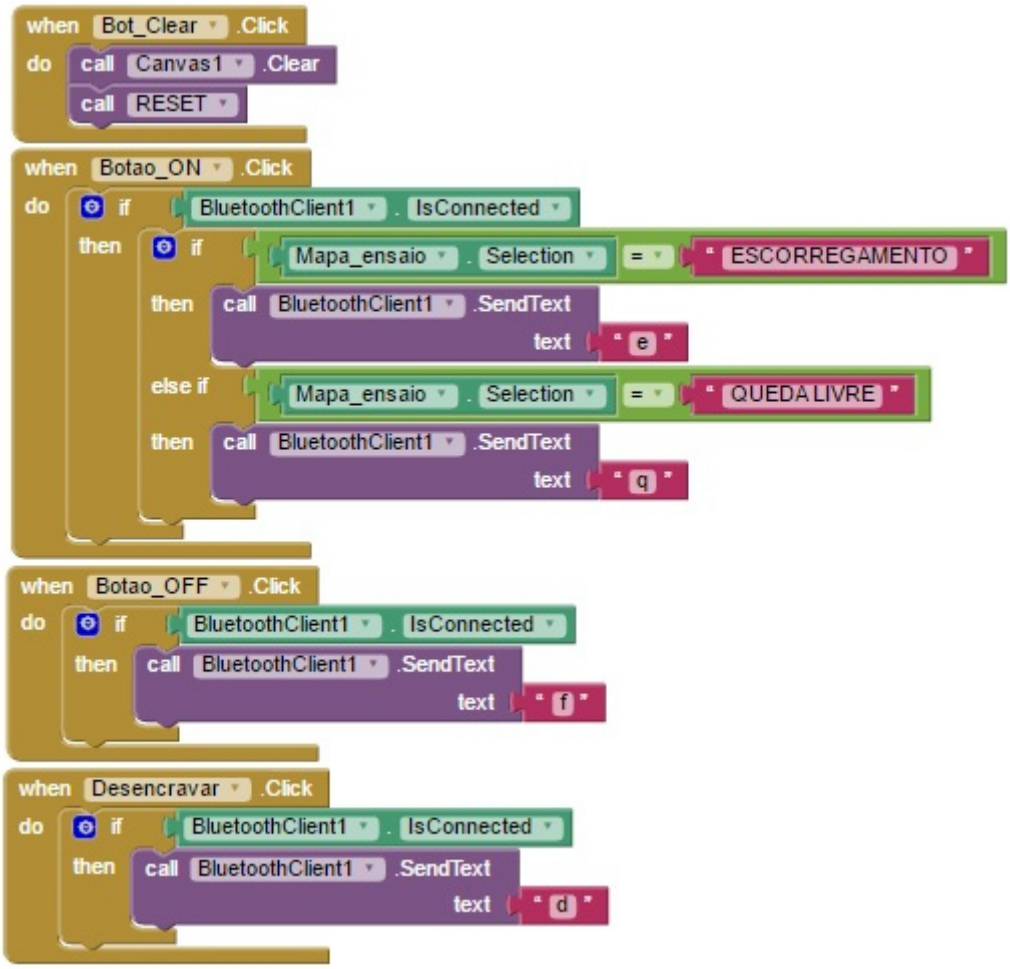
SETA - Shanghai Elevator Trade Association. (2015). Wittur UCM Solution. Shanghai. Obtido em 8 de Setembro de 2016, de http://b2b.sh-ea.org/UploadFiles/201207/201207111448478906.jpg

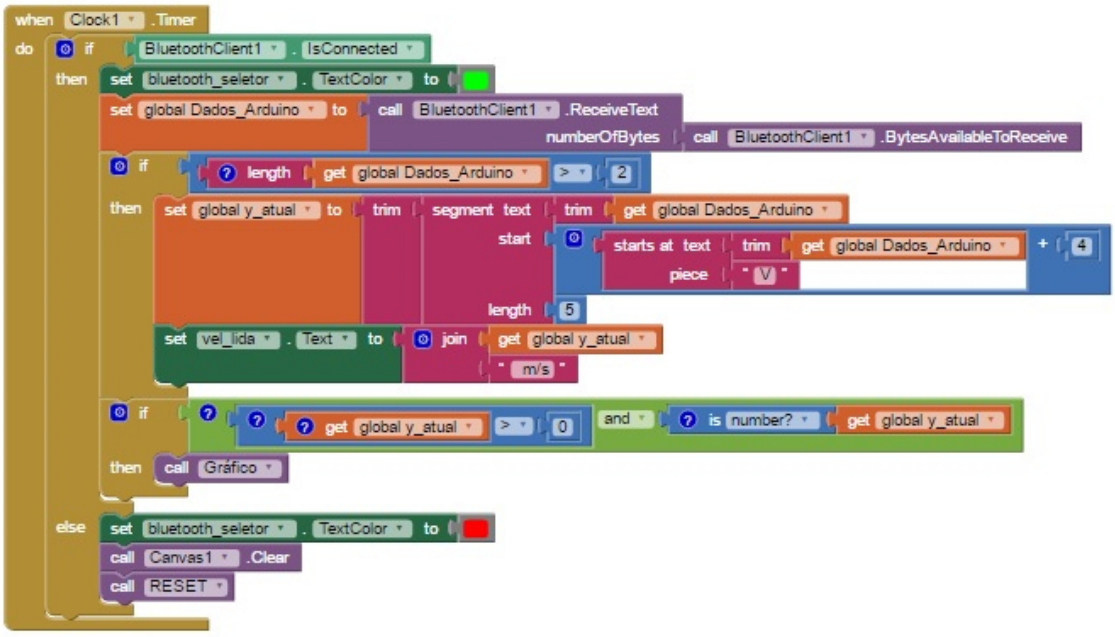
Anexo 1 Programa Completo em *Arduino*

Anexo 2 Programa Completo de Blocos em *App Inventor*

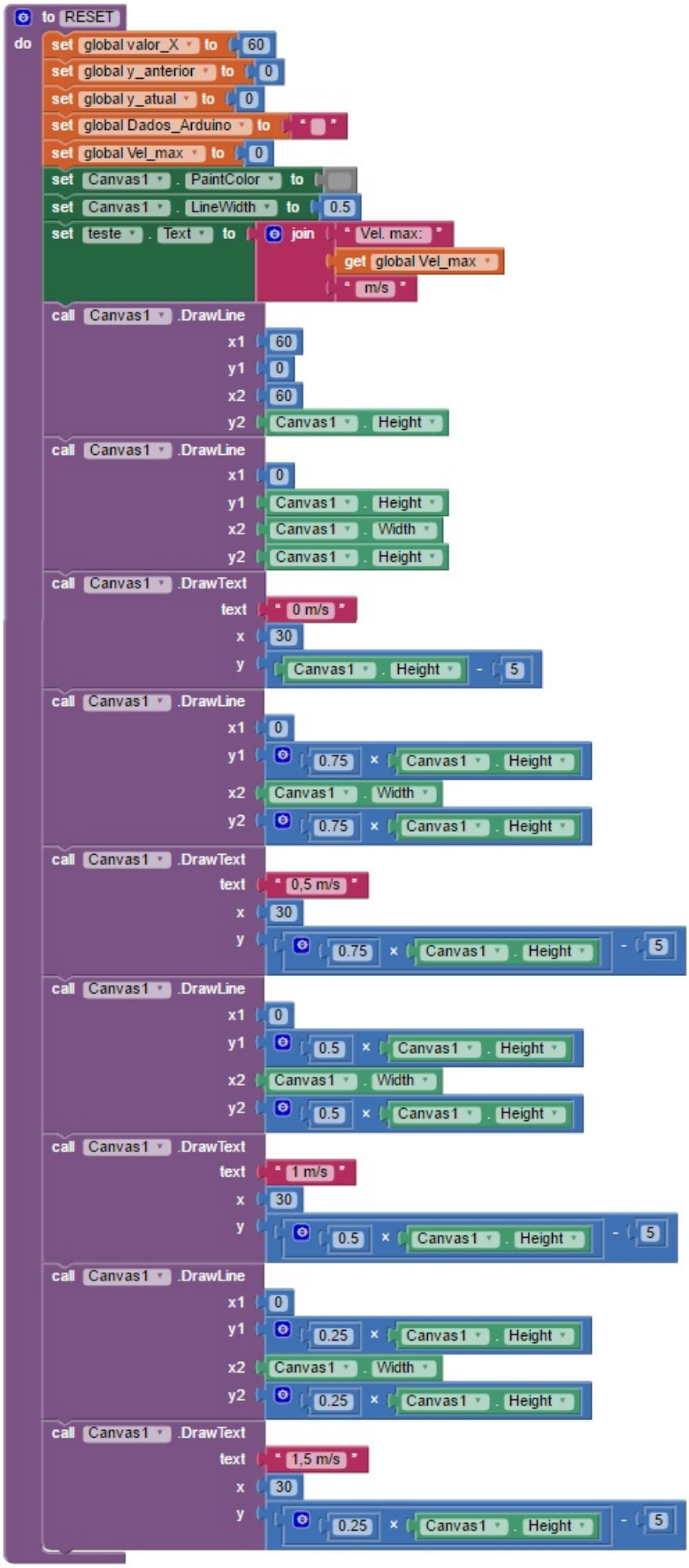


(Continua na próxima página)

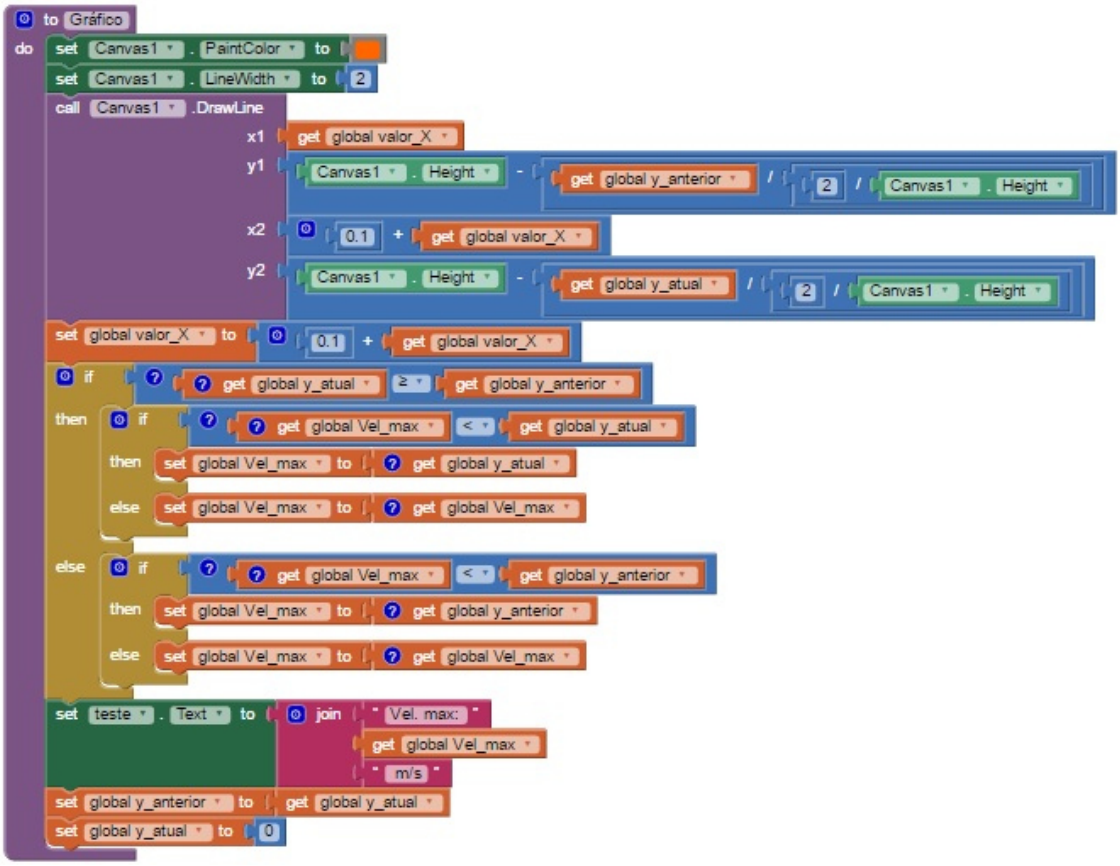




(Continua na próxima página)



(Continua na próxima página)



(Fim do código)