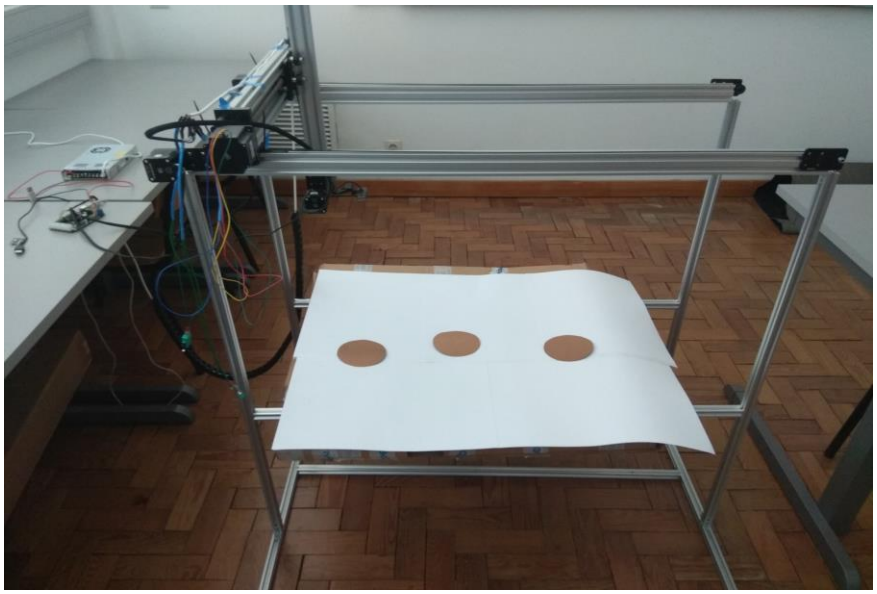
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia Mecânica

ISEL



Sistema de Posicionamento Robotizado Segundo o Conceito da Indústria 4.0

NUNO MIGUEL LEITÃO BEITES
(Licenciado em Engenharia Mecânica)

Trabalho Final de Mestrado para obtenção do grau de Mestre
em Engenharia Mecânica

Orientadores:
Doutor João Manuel Ferreira Calado
Doutor Mário José Gonçalves Cavaco Mendes
Doutor Francisco Mateus Marnoto de Oliveira Campos

Júri:
Presidente: Doutor Silvério João Crespo Marques
Vogais:
Doutora Alexandra Bento Moutinho
Doutor Mário José Gonçalves Cavaco Mendes

Outubro de 2018

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia Mecânica

ISEL

Sistema de Posicionamento Robotizado Segundo o Conceito da Indústria 4.0

NUNO MIGUEL LEITÃO BEITES

(Licenciado em Engenharia Mecânica)

Trabalho Final de Mestrado para obtenção do grau de Mestre em Engenharia Mecânica

Orientadores:

Doutor João Manuel Ferreira Calado

Doutor Mário José Gonçalves Cavaco Mendes

Doutor Francisco Mateus Marnoto de Oliveira Campos

Júri:

Presidente: Doutor Silvério João Crespo Marques

Vogais:

Doutora Alexandra Bento Moutinho

Doutor Mário José Gonçalves Cavaco Mendes

Outubro de 2018

Agradecimentos

Em primeiro lugar, quero agradecer especialmente aos meus pais, ao meu irmão e ao meu tio por me terem dado um apoio incondicional durante todas as fases do meu percurso académico e pessoal.

Quero agradecer à minha namorada que sempre me apoiou e me ajudou a motivar para concluir esta etapa.

Quero também agradecer a todos os meus amigos e colegas que me apoiaram durante esta jornada.

Quero agradecer aos Professores João Calado, Mário J. G. C. Mendes e Francisco Campos pela orientação, apoio e disponibilidade ao longo da realização deste Trabalho Final de Mestrado.

Quero agradecer também a todos os Professores que me aconselharam e acompanharam durante a minha formação académica.

Nota:

Este trabalho foi apoiado pela segunda edição do Concurso Anual para Projectos de Investigação, Desenvolvimento, Inovação e Criação Artística (IDI&CA) do Instituto Politécnico de Lisboa, sob a concessão do projecto IPL/2017/ROBOTCLEAN/ISEL

Resumo

Com a evolução da automação industrial, existe a necessidade de automatizar até os processos mais simples, de forma a manter as empresas industriais competitivas. O objectivo deste Trabalho Final de Mestrado é automatizar o processo de posicionamento do sistema de lavagem interior dos tanques dos camiões-cisterna, com supervisão a partir da *Internet*.

Neste trabalho de projecto desenvolveu-se um sistema de posicionamento robotizado com controlo de posição através da imagem de câmaras de vídeo. O algoritmo de visão foi programado no software *Matlab* com auxílio de bibliotecas de visão computacional e de comunicação com um controlador de motores de passo desenvolvido em arquitectura de *Arduino*.

O desenvolvimento foi feito dentro do conceito da indústria 4.0, ou seja, todo o sistema é totalmente automático com excepção da introdução do número de bocas pelo operador. Foram testados controladores com comunicação à base do protocolo *Ethernet*, nomeadamente o PC com o cliente do *Matlab* a comunicar com um servidor Apache para que seja possível estabelecer a conexão entre o sistema e a supervisão na *Internet*. Todo o sistema pode ser supervisionado através de um smartphone ou tablet a partir do *browser* instalado.

Construiu-se também um protótipo à escala, seleccionado de modo a validar o algoritmo de visão e as simulações efectuadas em *Matlab.* O protótipo é constituído por uma estrutura em perfis *V-slot*, motores de passo, o controlador e câmara USB, que embora tenha uma menor qualidade de imagem, foi suficiente para validar o algoritmo de visão implementado.

Palavras-chave

Sistema de Posicionamento Automático, Visão Computacional, Indústria 4.0, Sistema de Supervisão e Controlo, *Internet of Things*.

Abstract

With the evolution of industrial automation, there is a need to automate even the simplest processes, to keep industrial companies competitive. The objective of this Master's Thesis is to automate the positioning system of an internal washing process of tanker trucks, with supervision from the Web.

In this project work, a robot positioning system was developed with position control through the image of video cameras. The vision algorithm was programmed in Matlab software with the help of computer vision toolboxes and communication with a stepper motor controller developed in Arduino architecture.

The development was done within the concept of the industry 4.0, that is, the whole system is totally automatic except for the introduction of the number of holes by the operator. Controllers were tested with communication based on the Ethernet protocol, namely the PC with the Matlab client to communicate with an Apache server so that it is possible to establish the connection between the system and the supervision on the web. The entire system can be supervised through a smartphone or tablet from the installed browser.

A prototype was also built to scale, selected to validate the vision algorithm and the simulations performed in Matlab. The prototype consists of a structure in V-slot profiles, as well as the controller and USB camera that although having a lower quality of image, was enough to validate the implemented vision algorithm.

Keywords

Automatic Positioning System, Computer Vision, Industry 4.0, System of Supervision and Control, Internet of Things.

Índice

Índice de Figuras

Índice de Tabelas

Lista de Abreviaturas

AI	Artificial Inteligence
CAD	Computer Assisted Design
CCD	Charged-Couple Device
CPS	Cyber Physical Systems
CPU	Central Processing Unit
DRC	Design Rule Check
DSLR	Digital Single Lens Reflex
GPU	Graphics Processing Unit
HT	Hough Transform
HTTP	Hypertext Transfer Protocol
IBVS	Image-Based Visual Servo Control
IDE	Integrated Development Environment
IED	Intelligence Electronic Devices
IIoT	Industrial Internet of Things
IoT	Internet of Things
MAC	Message Authentication Code
MSU	Master Station Unit
MTU	Master Terminal Unit
PBVS	Position-Based Visual Servo Control
PC	Personal Computer
PLC	Programmable Logic Controller
RGB	Red Green Blue
RGB-D	Red Green Blue - Depth

RV	Realidade Virtual
RTU	Remote Terminal Unit
SCADA	Supervisory Control and Data Acquisition
SSU	Slave Station Unit
TCP/IP	Transfer Control Protocol / Internet Protocol
XAMPP	X-Cross-Platform; A-Apache; M-MariaDB; P-PHP; P-Perl.

Lista de Símbolos

A	Área da secção do módulo linear
b	Medida horizontal da secção equivalente do módulo linear
h	Medida vertical da secção equivalente do módulo linear
l	Lado do quadrado da secção equivalente do perfil estrutural
Iy_m	Segundo momento de inércia segundo o eixo Y do módulo linear
Iy_e	Segundo momento de inércia segundo o eixo Y do perfil estrutural
σ_p	Tensão máxima aplicada nos perfis estruturais
σ_y	Tensão de cedência da liga de alumínio 1060 *Alloy*
Fz max	Força máxima admissível em Z
Fy max	Força máxima admissível em Y
Mt max	Momento máximo admissível em torno do eixo X
ML max	Momento máximo admissível em torno do eixo Y e Z
Mx	Momento aplicado em torno do eixo X
My	Momento aplicado em torno do eixo Y
Mz	Momento aplicado em torno do eixo Z
Fy	Força aplicada em Y
Fz	Força aplicada em Z
Mt	Momento aplicado em torno do eixo X
ML	Momento aplicado em torno do eixo Y
Fcomb	Força combinada
C	Capacidade de carga dinâmica
L	Vida útil estimada [m]
Lh	Vida útil estimada [h]

θ	Ângulo da instrução do motor de passo
$Cref$	Coordenadas do centro da imagem (referência)
Cm	Coordenadas do centro do círculo detectado (medido)
E	Erro do sistema
Kp	Ganho
V	Vector velocidade dos motores X e Y
$e(t)$	Erro do sistema genérico
k	Função das características detectadas
$m(t)$	Medições provenientes da imagem
a	Parâmetro de informação adicional do sistema
s	Características detectadas
s'	Características de referência
vel_{RPM}	Velocidade de rotação do motor de passo
vel_{linear}	Velocidade linear do motor de passo
a_m	Aceleração do motor de passo
D_m	Distância percorrida equivalente a uma dada instrução em graus
x_{cd}	Coordenada X do círculo detectado
y_{cd}	Coordenada Y do círculo detectado

1. Introdução

Com a constante e rápida evolução das tecnologias de informação e comunicação, observou-se um grande impacto nos campos da automação industrial e robótica, nomeadamente nos protocolos de comunicação entre *software* e *hardware*, no aumento da disponibilidade, eficiência e flexibilidade dos processos automáticos. Esta evolução tecnológica deu origem ao termo "*Internet* das Coisas" (IoT – *Internet of Things*), que se refere à possibilidade de se ligarem dispositivos com a capacidade de partilharem informação relacionada com o ambiente que os rodeia [1].

A visão computacional é também uma tecnologia importante na automação de processos, uma vez que aumenta a capacidade sensorial das máquinas para, por exemplo, realizar tarefas de inspecção [2]. Com os avanços neste campo, tornou-se possível programar os robôs para executarem tarefas de uma maneira mais autónoma, eficaz e flexível. Isto porque os sistemas de posicionamento com visão não necessitam de muita programação relativamente aos pontos de referência e movimentos pré-programados, comparativamente aos sistemas sem visão. A capacidade de se adicionar visão artificial aos robôs permite que as instruções dadas para o movimento dos actuadores sejam adequadas a cada situação, dependendo da imagem que o robô recebe e das decisões programadas nos algoritmos. Assim, é possível alterar o processo em questão com muito mais facilidade, dentro das limitações físicas do mesmo e dos robôs que executam as tarefas.

1.1. Motivação

Cada vez mais, observa-se uma evolução rápida e significativa da automação industrial. Actualmente, até os sistemas mais simples começam a ser automatizados, com os objectivos de aumentar a produtividade e a segurança dos sistemas industriais. A conectividade e comunicação entre dispositivos periféricos e controladores são factores chave para a implementação de sistemas segundo o conceito de Indústria 4.0, que tem como objectivo impulsionar a reformulação da indústria, pois combina os aspectos do mundo físico, virtual, de tecnologias de informação e do sistema cibernético para ajudar a criar um novo ambiente de trabalho de produtividade integrada entre trabalhador e máquina [3].

O caso de estudo para este Trabalho Final de Mestrado está relacionado com o projecto do IPL: ROBOTCLEAN (ver os Agradecimentos), nomeadamente na automação do processo de posicionamento do sistema de lavagem dos camiões cisterna.

Actualmente, os sistemas de lavagem de camiões cisterna são manuais, ou seja, o operador tem de introduzir a cabeça de lavagem, manualmente, numa boca de cada vez para efectuar a sua lavagem (Figura 1.1). Isto implica riscos para a segurança dos operadores, uma vez que o processo é efectuado a altas temperaturas, contém produtos tóxicos e existe o risco de queda da parte do operador.

Figura 1.1 : Método Manual de Lavagem de Camiões Cisterna [4]

O que se pretende é automatizar o processo, para que este se torne mais eficiente, fiável e seguro para o operador. Também se pretende que as variáveis do processo sejam

visíveis através de um HMI (Human Machine Interface) que seja acessível a partir de um *browser*.

Figura 1.2 : Human Machine Interface (HMI) - Exemplo Genérico [5]

Os HMI's são uma excelente ferramenta para que os operadores/supervisores tenham acesso às variáveis intrínsecas do processo enquanto este é executado, de modo a ter a possibilidade de intervir caso algum aspecto do processo não esteja em conformidade ou para alterar ordens de trabalho (Figura 1.2). Também se podem gerar bases de dados com base na informação recolhida nos sensores de modo a optimizar-se o processo numa fase posterior ou planear operações de manutenção de forma mais eficaz [6].

A configuração pensada para este sistema automático, é a de um robô cartesiano, exemplificada na Figura 1.3 [7].

Figura 1.3 : Exemplo de um sistema de posicionamento em forma de robô cartesiano [7]

1.2. Objectivos

Com a necessidade de se projectar um sistema automático de lavagem, para que o processo se torne mais eficaz, seguro e menos dispendioso, neste Trabalho Final de Mestrado pretende-se desenvolver e implementar um sistema automático de controlo, com realimentação baseada em visão computacional, para o posicionamento do sistema de lavagem nas diversas bocas de acesso dos camiões cisterna. Mais detalhadamente, os objectivos deste Trabalho Final de Mestrado são os seguintes:

- Indicação de uma estrutura real, com sensores e actuadores a aplicar em contexto industrial nesta área.
- Projectar um sistema de posicionamento automático com controlo por visão computacional.
    - Desenvolver algoritmos de visão computacional que consigam dar resposta ao desafio que é proposto.
- Construir um protótipo do sistema à escala.
    - Validação do algoritmo de visão desenvolvido aplicação dos conhecimentos teóricos adquiridos neste trabalho.
- Supervisionar o processo através de uma página de *Internet.*
    - Criação de um HMI para este projecto focando a possibilidade de o operador necessitar de interromper o processo e de supervisionar as variáveis do processo, tais como temperatura e humidade da envolvente, estado dos motores e qualidade da imagem capturada quanto à iluminação, resolução e taxa de fotogramas necessários ao bom funcionamento do sistema.

Os primeiros dois objectivos foram atingidos, uma vez que o algoritmo de visão foi implementado com sucesso no protótipo à escala e o sistema funciona como intencionado. O terceiro objectivo foi atingido parcialmente, pois o HMI não está na sua forma final. Contudo, a ligação entre o sistema físico e o HMI está estabelecida, tendo sido testados botões para actuar sobre o sistema e também a recepção de informação enviada pelo mesmo. O algoritmo permite ver em tempo real as imagens da câmara com uma taxa de fotogramas aceitável e, também, aceder ao HMI a partir de um *browser*.

Um dos resultados deste Trabalho Final de Mestrado foi um artigo publicado na primeira edição da conferência ibérica TEMM2018 (Theoretical and Experimental Mechanics and Materials), cuja realização efectuou-se entre os dias de 4 a 7 de Novembro de 2018 [8].

1.3. Estrutura do Documento

O presente Trabalho Final de Mestrado encontra-se dividido em seis capítulos. Neste primeiro capítulo introdutório, pretende-se enquadrar o trabalho desenvolvido.

O capítulo dois é referente ao levantamento do Estado da Arte nas áreas de estudo em que este trabalho se insere.

No capítulo três, é feita uma selecção da estrutura a utilizar no caso real para a lavagem interior de camiões cisterna.

No capítulo quatro é apresentado em pormenor o algoritmo de visão desenvolvido, bem como a simulação que foi elaborada de modo a validar o mesmo.

O capítulo cinco é relativo ao protótipo construído em ambiente laboratorial com o objectivo de validar o algoritmo programado e a simulação efectuada. Este capítulo também apresenta os aspectos essenciais da implementação de um sistema de supervisão e controlo. Neste caso, o HMI é uma página de *Internet* que pode ser acedida por qualquer dispositivo que possua um *browser* instalado.

O sexto e último capítulo é referente às conclusões retiradas ao longo da realização deste Trabalho Final de Mestrado, assim como possíveis implementações futuras neste mesmo projecto.

Nota: Este documento foi elaborado segundo o acordo ortográfico anterior ao que está actualmente em vigor.

2. Estado da Arte

Tendo em conta a motivação e os objectivos deste Trabalho Final de Mestrado, foi necessário definir-se que tecnologias e métodos utilizar, de modo a tornar o processo de lavagem de camiões cisternas automático. Como existem vários tipos de camião cisterna no que toca a números de bocas/depósitos, surgiu a necessidade de se investigar sobre sistemas robóticos que sejam controlados por visão. Isto devido à necessidade de se automatizar o processo de uma maneira flexível, isto é, que não seja necessário programar o sistema para cada tipo de cisterna diferente, e evitar o uso de um operador para ajustar o actuador em função da posição do camião.

2.1. Sistemas robóticos guiados por visão

A tendência que os processos industriais apresentam actualmente é de exigir cada vez mais a simplicidade, adaptabilidade, e a capacidade de reconfiguração dos sistemas no ambiente industrial [9]. Isto acontece devido à tendência que existe da personalização em massa e do elevado ritmo com que os produtos são introduzidos no mercado, impulsionada pelo avanço rápido das tecnologias. As soluções de automação tradicionais de montagem de componentes com baixas tolerâncias, usadas, por exemplo, na montagem de placas de circuito impresso, requerem o desenvolvimento de *hardware* e/ou *software* específico, o que faz com que estes tipos de linhas de montagem sejam difíceis de reconfigurar. Devido à sua rigidez, este tipo de solução não respeita os requisitos industriais atrás referidos [9].

Com os recentes avanços na tecnologia de visão, consegue-se que este tipo de solução de automação cumpra os requisitos industriais acima mencionados, devido à sua flexibilidade, à rapidez com que a tecnologia é desenvolvida e à consequente tendência para os custos de desenvolvimento e implementação diminuirem. Neste momento, é possível adicionar um sistema de visão a um robô por um preço competitivo, com um baixo custo de mão-de-obra humana e permitindo uma maior flexibilidade e capacidade de reimplementação do sistema em causa [9].

Nota: A partir deste ponto do documento, existem termos não traduzidos e escritos em inglês, isto por se tratarem de termos universais para a designação de certos equipamentos e/ou técnicas.

2.2. Controlo de robôs guiados por visão (*Visual servo control*)

Para que seja possível o controlo de um robô guiado por visão, é necessário obter informação através de uma ou mais câmaras. Os sensores visuais podem estar montados no próprio robô, o que faz com que o movimento do robô implique movimento na câmara, ou podem estar fixos num ponto externo ao sistema. Matematicamente, o objectivo destes sistemas é minimizar o erro e(t), dado pela equação (1).

$$e(t) = \boldsymbol{k}(\boldsymbol{m}(t), \boldsymbol{a}) - \boldsymbol{s}' \quad (1)$$

Esta é uma expressão bastante geral, e engloba muitas aplicações [10]. Os parâmetros desta equação são os seguintes: O vector $\boldsymbol{m}(t)$ é um conjunto de medições provenientes da imagem, como por exemplo, as coordenadas de pontos de interesse ou de um centroide de um objecto. Estas medições nas imagens são utilizadas para gerar um vector $\boldsymbol{s}$ com as características visuais das imagens obtidas, como se pode observar na equação (2).

$$\boldsymbol{s} = \boldsymbol{k}(\boldsymbol{m}(t), \boldsymbol{a}) \quad (2)$$

Dentro da função de tranformação $\boldsymbol{k}$, $\boldsymbol{a}$ é um conjunto de parâmetros que representam potencial informação adicional sobre o sistema. Caso o objecto alvo esteja estático, o parâmetro $\boldsymbol{s}'$ considera-se constante e as únicas mudanças que ocorrem em $\boldsymbol{s}$ são devidas ao movimento da câmara. O que varia principalmente entre diferentes sistemas guiados por visão, é o parâmetro $\boldsymbol{s}$, que consiste num conjunto de características que são obtidas de formas diferentes, dependendo do tipo de abordagem em causa. Existem duas abordagens diferentes para este tipo de problemas. A primeira é o denominado Controlo Baseado em Imagem (IBVS - *Image-Based Visual Servo Control*), e a segunda abordagem denomina-se Controlo Baseado em Posição (PVBS – *Position-Based Visual Servo Control*) [8,9].

2.2.1. Controlo Baseado em Imagem

Os sistemas de controlo baseados em imagens definem o parâmetro $\boldsymbol{s}$, como sendo características disponíveis imediatamente a partir do plano da imagem. As medições $\boldsymbol{m}$ da imagem normalmente são as coordenadas dos pixéis de um conjunto de pontos da imagem, e os parâmetros $\boldsymbol{a}$ referem-se aos parâmetros intrínsecos da câmara que se

inserem na função $\boldsymbol{k}$ para se transformarem as medições obtidas a partir da imagem (expressas em pixéis) em características que se estão a tentar encontrar, como por exemplo, círculos, linhas, ou outras formas.

Neste tipo de sistemas, o referencial de coordenadas é relativo à imagem obtida da câmara, ou seja, as coordenadas coincidem com as do plano da imagem. Deste modo, para garantir a estabilidade em sistemas de posicionamento baseados em imagem, é necessário que a imagem presente e a desejada/futura estejam próximas, isto é, não pode existir muita discrepância espacial entre duas imagens adquiridas consecutivamente. Este requisito pode não ser cumprido quando o robô necessita de efectuar grandes deslocamentos em pouco tempo. Pode-se observar na Figura 2.1, um exemplo de um esquema de controlo de um sistema IBVS. Esta abordagem considera-se mais indicada para aplicações em que seja necessário ultrapassar problemas como erros de calibração, devido à baixa sensibilidade que apresenta relativamente a este tipo de situações [8,10,11,12,13,14,15,16].

Como existem situações em que os parâmetros da câmara não estão disponíveis ou não são claros, foram desenvolvidos métodos de calibração [19] para este tipo de sistemas cumprirem os requisitos sem ter que saber ao certo todos os parâmetros necessários para o caso da abordagem clássica. Como por exemplo, a distância focal da câmara, que é assumida dentro de um intervalo de valores e de seguida é re-estimada sucessivamente em cada iteração do ciclo de controlo. Estes métodos de calibração conseguem garantir uma boa estabilidade do sistema a partir de uma abordagem mais simples no que toca à calibração da câmara [17,9].

Cada vez mais se procura incorporar outras tecnologias complementares dentro dos sistemas robóticos guiados por visão, tais como, a aprendizagem por demonstração, proposto por [20], que consiste num robô equipado com visão que grava um conjunto de movimentos resultantes de tarefas executadas por mão humana, e de seguida é capaz de reproduzir essas tarefas com uma dada velocidade pretendida.

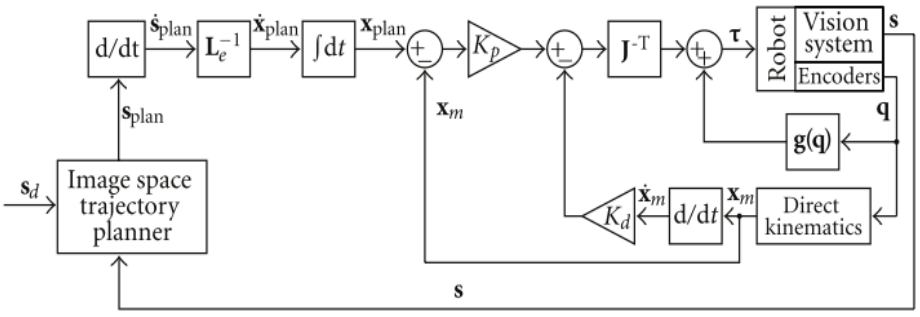


Figura 2.1: Controlo de um sistema IBVS (Dynamic Look and Move) [14]

Hoje em dia existem inúmeras aplicações para a tecnologia de visão. Um dos ramos que tem beneficiado com este tipo de tecnologia é o da medicina, sendo agora possível, por exemplo, programar robôs para manusear ferramentas cirúrgicas [21]. Preparar todas as ferramentas necessárias num ambiente pré-operatório é das tarefas mais importantes e das que mais consomem recursos num hospital. O desempenho desta tarefa afecta directamente a segurança do paciente e o orçamento de uma cirurgia, como tal, uma automatização eficaz deste processo tem um grande impacto na gestão de recursos de uma instituição de saúde que efectue este tipo de tarefas. A função deste tipo de robôs é identificar um conjunto de instrumentos cirúrgicos a partir de alguns pontos de referência e de seguida organizá-los em vários recipientes de acordo com as suas características, como é esquematizado na Figura 2.2. Neste caso de aplicação foi utilizada uma câmara DSLR (*Digital Single Lens Reflex*), montada no manipulador. Este é um exemplo de um sistema IBVS [21].

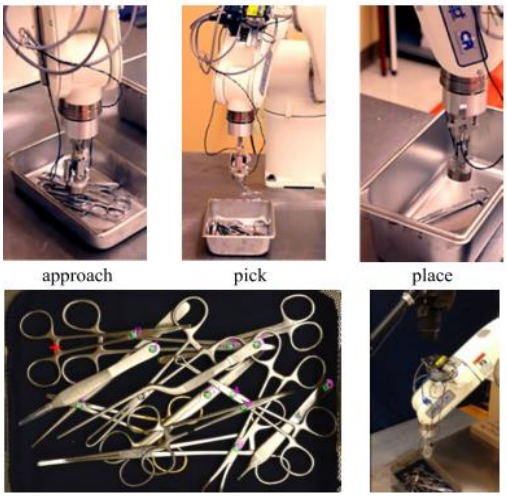


Figura 2.2 : *Pick and place* de instrumentos cirúrgicos com auxílio de um sistema de visão e actuação robótica [21]

2.2.2. Controlo Baseado em Posição

Nos sistemas de controlo baseados em posição utiliza-se a posição da câmara relativamente a um referencial de coordenadas para que seja possível definir o parâmetro $\boldsymbol{s}$. Para que seja possível definir a posição da câmara a partir de um conjunto de medições obtidas através de uma imagem, é necessário que os parâmetros instrínsecos da câmara e o modelo 3D do objecto observado sejam conhecidos e definidos. É frequente, nestes casos, definir-se $\boldsymbol{s}$ como a parametrização utilizada para representar a posição da câmara. Os parâmetros $\boldsymbol{a}$, são neste caso, os parâmetros intrínsecos da câmara e o modelo 3D do objecto. Os parâmetros extrínsecos indicam a posição do sistema de coordenadas da câmara em relação a um sistema de coordenadas global. Assume-se que o sistema de coordenadas global é coincidente com o referencial do objecto a detectar, sendo que assim, os parâmetros extrínsecos fornecem a informação directa da posição da câmara em relação ao objecto [8,10,11,12,13,14,15,16]. Pode-se observar na Figura 2.3, um exemplo de um esquema de controlo de um sistema PBVS.

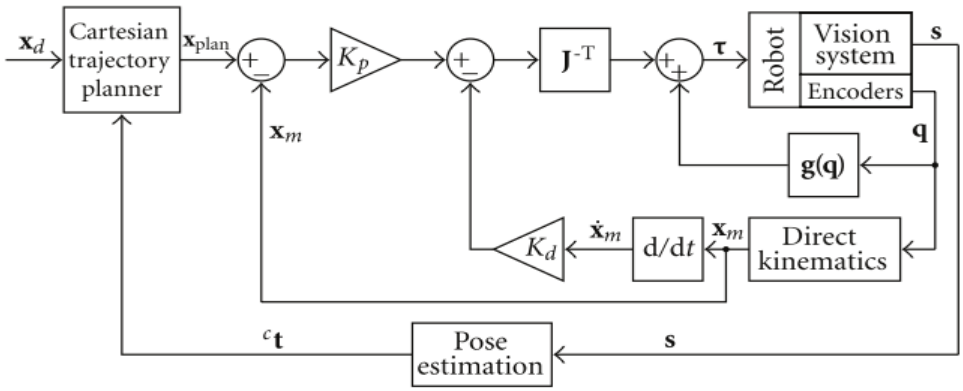


Figura 2.3: Controlo sistema PBVS (*Dynamic Look and Move*) [14]

Uma aplicação possível com este tipo de tecnologia é a inspecção de furos na fase do controlo de qualidade de um processo de fabrico. O exemplo que se apresenta de seguida foi desenvolvido por Biegelbauer et al.[2] e consiste num robô com 6 graus de liberdade que movimenta um endoscópio com o objectivo de inspecionar cada furo numa dada peça (Figura 2.4). Para parametrizar o sistema de forma a efectuar as tarefas propostas em novas peças, escolheu-se uma maneira rápida e simples de programar as trajectórias básicas do sistema através de um *software* de CAD (*Computer Assisted*

Design). O *software* CAD é onde são definidos os furos a inspecionar na peça em questão e isto permite ao utilizador programar o robô de uma maneira eficiente. O endoscópio está ligado a uma câmara CCD (*Charged-Couple Device*), de modo a capturar imagens da superfície do furo para uma posterior inspecção de qualidade automatizada baseada no processamento das imagens. Em caso da detecção e localização do furo falharem, há um mecanismo de segurança programado de modo a evitar colisões do endoscópio com a peça a inspecionar. Se tal acontecer, o endoscópio é imediatamente retirado. Neste exemplo, como o sistema é guiado com base no posicionamento dos furos definidos no modelo 3D, é um exemplo de um PBVS.

Figura 2.4 : Protótipo do sistema de inspecção de furos [2]

O exemplo que se segue, é um projecto de um robô que resulta da combinação do conceito de um robô industrial e de um robô móvel (Figura 2.5). Os sistemas que resultam desta combinação são denominados Manipuladores Móveis Industriais Autónomos (AIMMs - *Autonomous Industrial Mobile Manipulators*). Este tipo de sistemas apresenta várias vantagens, tais como, elevada flexibilidade, funcionalidade e também mobilidade. O caso em estudo foi proposto por Cheng et al. [22] e o objectivo definido para o robô foi o de executar as seguintes funções: 1) identificar o objecto; 2) mover-se em direcção ao objecto enquanto acompanha a posição do mesmo; 3) coordenar a base móvel com o manipulador e por fim, parar e manipular o objecto em questão. Neste projecto é utilizada uma câmara RGB-D (*Red Green Blue – Depth*), o que permite um fornecimento de imagens RGB (*Red Green Blue*) com informação de

profundidade, conseguindo-se assim, calcular a distância entre a câmara e os objectos. A localização do objecto alvo é obtida através da correspondência das características das cores existentes. Ao contrário de um sistema IBVS, este sistema (PBVS) é projectado para estabilizar a distância e ângulo de visão do objecto alvo, ao ajustar a velocidade linear e angular da plataforma móvel [22].

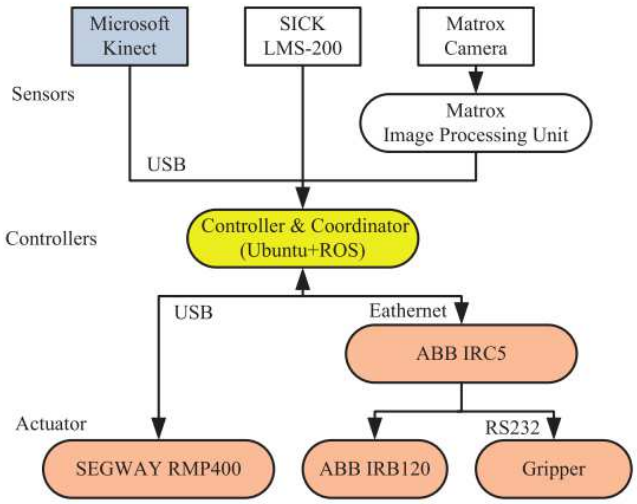


Figura 2.5: Arquitectura da plataforma AIMM (Sensor utilizado (1): Microsoft *Kinect*) [22].

2.2.3. Calibração de Câmaras

Existem dois tipos de parâmetros que têm de ser considerados no processo de calibração de uma câmara, os parâmetros intrínsecos e extrínsecos. Os parâmetros intrínsecos dizem respeito à geometria interna e características ópticas do sensor de imagem, indicando de que forma a luz é projectada, através da lente, no plano da imagem do sensor. Os parâmetros extrínsecos medem a posição e orientação da câmara em relação a um sistema de coordenadas global definido [23].

A calibração de câmaras é dos passos mais importantes quando se projecta um sistema de visão, seja este um microscópio digital, instrumento de metrologia ou um robô guiado por visão [24]. Como a precisão da calibração influencia directamente no desempenho dos sistemas, foram desenvolvidos vários métodos de calibração, que se podem classificar em duas categorias: calibração baseada em padrões de calibração, e auto-calibração [24]. Quanto aos padrões de calibração, existem três tipos de objectos de acordo com as suas dimensões, nomeadamente, alvo 3D [25], alvo 2D [26], alvo 1D

[27]. A auto calibração não necessita de nenhum padrão conhecido, porém, a sua precisão pode ser afectada se houver ruído na imagem adquirida [28].

Os métodos de calibração também podem ser classificados quanto à técnica utilizada para determinar os parâmetros instrínsecos e extrínsecos da câmara [23]:

- Técnicas de optimização não-linear: uma técnica de calibração torna-se não linear quando se contabiliza qualquer tipo de imperfeição da lente no modelo matemático da câmara. Nesse caso, os parâmetros são obtidos através de um processo iterativo com o objectivo de minimizar uma determinada função, que normalmente é a distância entre os pontos da imagem e as projecções modeladas obtidas através das iterações [29].
- Técnicas lineares: estas técnicas utilizam o método dos mínimos quadrados para obter a matriz de transformação que relaciona os pontos 3D com as suas projecções 2D [30].
- Técnicas de dois passos: estas técnicas utilizam como primeiro passo a optimização linear para determinar alguns dos parâmetros da câmara, e como segundo passo, os restantes parâmetros são calculados iterativamente. Nestes métodos, a convergência das iterações é quase garantida devido aos valores calculados linearmente no primeiro passo [31].

2.2.4. Dynamic Look and Move e Direct Visual Servoing

Os sistemas guiados por visão também podem ser classificados com base na sua arquitectura, nos seguintes dois tipos: *Dynamic Look and Move* – O sistema de visão fornece, por intermediário de uma unidade externa de processamento de imagem, um *input* pré-processado (coordenadas, presença de formas, etc.) ao ciclo fechado de controlo da articulação/actuador do robô que estabiliza o sistema, como está referido por Xin et al. [32]; *Direct Visual Servoing* – As informações provenientes do sistema de visão são directamente utilizadas, no mesmo formato em que são adquiridas, no ciclo de controlo para calcular os *inputs* das articulações, estabilizando autonomamente o robô, como proposto por Garcia et al. [16] e Alepuz et al. [33]. A frequência com que são processadas as imagens dos sistemas de visão é normalmente bastante inferior à frequência com que o ciclo de controlo calcula os *outputs* necessários para os motores e/ou articulações do robô. Isto causa uma restrição que pode causar problemas na

estabilidade do sistema, na medida em que as imagens não são obtidas a uma frequência de amostragem suficientemente elevada, o que faz com que haja a possibilidade de serem enviadas instruções para os motores com base em informação retirada de imagens desactualizadas [31,13]. Esta diferença de frequências de processamento faz com que, para que sejam implementados sistemas de *Direct Visual Servoing*, seja necessário *hardware* com um elevado poder de processamento para que o valor da diferença das frequências seja o menor possível, ou nulo. Por esta razão, a maior parte das aplicações de controlo de movimento baseado em visão são do tipo *Dynamic Look and Move*, devido ao menor poder de processamento exigido [31,13].

2.2.5. Sistemas de Visão Monoculares e *Stereo*

Um sistema que disponha de apenas uma câmara, é capaz de detectar características específicas de um dado objecto, como por exemplo, arestas, círculos, linhas, entre outros. A partir da relação entre as coordenadas da imagem e as coordenadas do "mundo real", se forem conhecidas, é possível reconstruir a posição 3D de um dado objecto, através de um conjunto de expressões matemáticas [35]. Podem ser encontrados alguns métodos de calibração de câmaras e respectiva comparação propostos por Shi et al. [36].

A técnica de visão *Stereo* consiste em extrair informações 3D a partir de várias imagens 2D obtidas através de duas câmaras, e é tipicamente utilizada quando se pretende ter uma perspectiva tridimensional do espaço, a partir da qual se podem estimar dimensões (lineares e angulares). O maior desafio neste tipo de sistemas é encontrar pontos correspondentes em cada imagem do par *stereo* que representam a projecção do mesmo objecto nas duas imagens. Uma vez tendo essa informação, através dos dados de calibração das câmaras e da triangulação *stereo*, é possível obter as coordenadas 3D do objecto [32,34]. Existem vários métodos para encontrar pontos correspondentes nas imagens, como por exemplo, a procura de áreas ou características semelhantes nas duas imagens, como descrito por Scharstein et al. [38].

2.2.6. Ferramentas de extracção de características em imagens

Normalmente, quando se utilizam ferramentas de extracção de características em imagens, é necessário que se efectue um ou mais processamentos prévios da imagem a analisar, mais conhecidos como técnicas de processamento de imagem [39]. Por

exemplo, a conversão de RGB para Escala de Cinzentos (*Grayscale*) é uma técnica utilizada maioritariamente porque os tempos de processamento de imagens em escala de cinzentos são inferiores aos que se verificam para imagens RGB, pelo facto de possuírem um número muito menor de variedade de cores [40]. Estão descritas outras técnicas de processamento de imagem utilizadas por Corke [39].

Uma das ferramentas de extracção de características utilizadas é a análise de regiões/formas (*Blob Analysis*), que é utilizada para calcular dados estatísticos em regiões de interesse. Vários exemplos dos dados calculados por esta ferramenta são os seguintes: centróide; caixa limite (*Bounding Box*); matriz de rótulos (*Label Matrix*) e contagem de formas (*Blob Count*). O objectivo desta ferramenta é detectar regiões correspondentes a versões a outra escala da mesma imagem. Para o objectivo ser cumprido, é também necessário que exista um mecanismo de escolha de escala de modo a encontrar o tamanho característico da região que varia proporcionalmente com a transformação da imagem [38,39].

Outra ferramenta que é utilizada na extracção de características de imagens é a correspondência de padrões (*Pattern Matching*), que consiste em encontrar correspondências entre um padrão predefinido e as imagens que se estão a adquirir. Existem vários métodos de correspondência de padrões, como por exemplo: correspondência de padrões baseada em *Machine-learning;* correspondência de padrões baseada em modelos; correspondência de padrões baseada em linhas (*strings)* ou correspondência de padrões baseada na Verificação da Regra de *Design* (*DRC - Design Rule Check*) [40].

De maneira a detectar formas como linhas ou circunferências, pode-se usar uma ferramenta matemática denominada por Transformada de *Hough* (HT - *Hough Transform*). A HT é uma técnica de extracção de características frequentemente utilizada na área do processamento de imagem. O conceito baseia-se na detecção de arestas nas imagens, transformando cada uma das arestas na imagem para o espaço de coordenadas de *Hough*. Uma das vantagens deste método é a robustez do resultado da imagem segmentada, ou seja, este algoritmo não é muito sensível a informação incompleta ou ruído. Contudo, pode-se utilizar a detecção de arestas como ferramenta de pré-processamento antes da transformação para o espaço de *Hough*, de modo a aumentar a precisão de detecção [40,41]. A detecção de círculos é um aspecto

importante em aplicações de visão, uma vez que é uma forma geométrica bastante frequente na indústria. A maior parte dos algoritmos de detecção de círculos baseiam-se na HT, tratando-se, neste caso, da HT circular, ferramenta que necessita de uma grande quantidade de memória disponível e demora um tempo considerável a calcular. Quanto ao output criado pela HT circular, as duas primeiras dimensões no espaço de *Hough* correspondem às coordenadas do centro do círculo detectado e a terceira dimensão corresponde ao raio do mesmo [41,42].

Todos os métodos referidos de extracção de características ou de processamento de imagem são altamente dependentes das condições de iluminação, que influenciam a qualidade da imagem adquirida e a taxa de fotogramas adquiridos por segundo, entre outros factores. López-Franco et al. [46], propôs um algoritmo de visão que apresenta melhores resultados quanto à robustez à variação de luminosidade e contraste, relativamente a outros métodos conhecidos.

Existem, porém, outros problemas associados a este tipo de sistemas, como por exemplo, o tempo elevado de processamento necessário para tratar as imagens adquiridas, a resolução da imagem e a taxa de fotogramas (*frame rate*). Embora existam algoritmos de visão relativamente rápidos no processamento, geralmente as taxas de amostragem *(sample rate*) das medições das imagens continuam a ser inferiores à frequência dos *encoders* de posição e dos sensores de ângulo de juntas robóticas. Para que este problema seja mitigado, é necessário que a taxa de amostragem da imagem seja alta o suficiente e que o tempo de processamento da imagem seja o mais baixo possível. Ao combinar as medições obtidas a partir da imagem e as altas frequências da informação de posição, consegue-se que o anel de controlo consiga ser executado com maior frequência, logo, consegue-se também uma melhor estabilidade assimptótica e convergência do sistema [18]. Existem outras opções para acelerar o processamento de imagem, como por exemplo, o método proposto por Patil et al.[47], que tira proveito do poder de processamento do GPU (*Graphics Processing Unit*) para processar as características (linhas) das imagens adquiridas utilizando a HT. Os resultados deste método indicaram que o tempo de processamento da mesma imagem utilizando o GPU é seis vezes inferior ao que se obteria com o CPU (*Central Processing Unit*) [47].

2.3. IIoT – *Industrial Internet of Things* (*Internet* Industrial das Coisas)

A *Internet* Industrial das Coisas (IIoT - *Industrial Internet of Things*) é a aplicação do conceito da *Internet* das coisas (IoT – *Internet of Things*) na indústria. A IIoT é um conceito em expansão crescente que oferece uma vasta infraestrutura de comunicações aos processos e fábricas de vários tipos de indústria [48]. A IIoT refere-se aos objectos unicamente identificáveis e suas respectivas representações virtuais numa rede como pequenos pacotes de informação num conjunto de nós (dispositivos electrónicos que estão ligados a uma rede e que são capazes de gerar, receber ou transmitir informações através de um canal de comunicação), de modo a integrar, automatizar e controlar todos os componentes numa fábrica, por exemplo, como proposto por Sasajima et al. [49].

O termo Sistemas Ciber Físicos (CPS – *Cyber Physical Systems*) é bastante importante quando se fala de IoT. Um CPS significa "todas as coisas" (sendo estas dispositivos periféricos) no mundo físico que estão ligadas a um espaço cibernético (*Cyber space)* através de uma rede de sensores [49], como esquematizado na Figura 2.6. Os CPS's permitem aceder aos dispositivos periféricos (*Edge Devices)* através da nuvem em qualquer ponto do mundo ou explorar as capacidades da computação em nuvem.

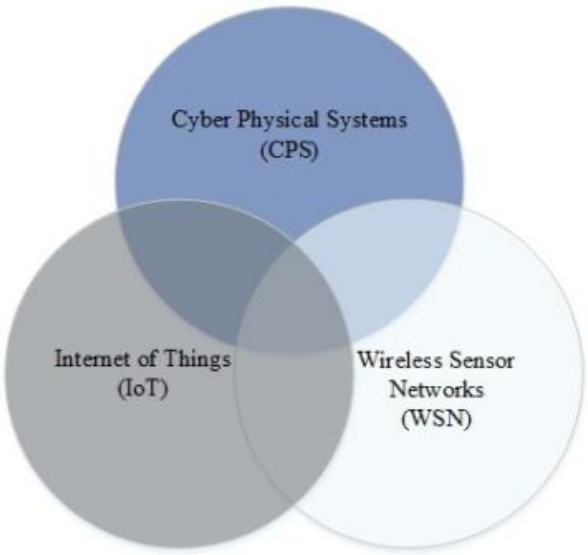


Figura 2.6 : Modelo de um CPS [50]

Com serviços de comunicações através da nuvem disponibilizados por várias empresas, como por exemplo, Microsoft ou Amazon, nunca foi tão fácil e acessível conectar até o mais pequeno dispositivo (seja este um sensor, actuador, etc.) a uma rede, seja esta uma rede local ou uma rede que seja possível aceder em qualquer ponto do mundo. A IoT consegue providenciar um grande poder de processamento para as indústrias que a implementam. O poder de processamento, que também pode ser encontrado na nuvem é expansível na medida, em que, em vez de se ter apenas uma unidade de processamento central (CPU- *Central Processing Unit*) com memória e espaço de disco limitados, por exemplo, pode-se ter um conjunto de CPU's ou servidores a realizar a mesma tarefa em muito menos tempo. Se este poder de processamento for utilizado com eficácia, pode ser aplicado para analisar dados, reportar eventos ou resultados e resolver problemas que nunca foram resolvidos anteriormente [48].

Quando se implementa o conceito de IIoT, por exemplo, numa instalação fabril, o primeiro passo é conseguir adquirir informação dos sensores e actuadores, que por sua vez estão ligados a outros componentes e equipamentos. Estes componentes podem ser, por exemplo, motores, bombas, válvulas, entre muitos outros componentes de automação que se encontram num ambiente fabril. Todos estes equipamentos mencionados acima podem-se classificar como dispositivos periféricos. O próximo passo é adquirir a informação destes dispositivos, tarefa que é normalmente efectuada pelo controlador lógico programável (PLC-*Programmable Logic Controller*) e outros controladores avançados. Uma vez que a informação é adquirida, esta pode ser partilhada com utilizadores locais e remotos via *Internet* e redes locais [6] .

2.3.1. Software

Muitas empresas que tencionam implementar o conceito de IoT procuram soluções prontas a utilizar (ou seja, sem que seja preciso criar a solução particularmente para um caso específico), que sejam expansíveis com o objectivo de monitorizar, analisar e visualizar dados em tempo real. Devido à normalização dos sistemas de comunicação emergentes, muitas das aplicações de IoT estão disponíveis para que permitam a operacionalidade com a mínima configuração possível e pouco ou nenhum *software* intermédio (*middleware)*. Para se conseguirem estes benefícios requere-se que estejam

disponíveis *gateways* de baixo custo, também conhecidos por "*IoT Gateways*". Estes mecanismos comunicam com os dispositivos periféricos e providenciam a ponte entre uma rede local de comunicação e poder de computação e visualização de uma aplicação baseada na nuvem. As *IoT Gateways* devem possuir a habilidade de comunicar com protocolos de comunicação normalizados da indústria.

A conectividade é a chave, sem se conseguirem interligar os dispositivos periféricos atrás de uma *firewall* e publicar dados seguramente em aplicações baseadas na nuvem, as organizações não serão capazes de atingir os objectivos pretendidos quanto a cálculos complexos através do poder computacional na nuvem [48]. Os benefícios para organizações e clientes podem ser alcançados sob a forma de redução de custos, novos fluxos de receitas, ou uma experiência melhorada do cliente.

2.3.2. Aplicações

As vantagens competitivas que são proporcionadas pela crescente e consistente visibilidade, precisão e acções baseadas em informações dos sensores são demasiado importantes para as empresas ignorarem, daí o crescimento de aplicações baseadas em IIoT. Estas aplicações têm sido cada vez mais utilizadas na indústria devido às suas vastas aplicações e vantagens como por exemplo [51] :

- Soluções de Inteligência Artificial (AI – *Artificial Intelligence*), relacionadas com a monitorização e visibilidade avançadas.
- Actualização e melhoria de sistemas fabris já existentes com auxílio das tecnologias IIoT com o objectivo de expandir a visibilidade de activos e processos.
- Serviços de monitorização de activos de fornecedores que estão a tirar vantagem da IIoT para providenciar capacidades de analítica preventiva, predictiva e remota para os seus activos instalados no *website* do cliente.

2.4. Sistemas SCADA – *Supervisory Control and Data Acquisition*

As redes de Sistemas de Supervisão e Acquisição de Dados (SCADA - *Supervisory Control and Data Acquisition*) são redes computadorizadas especiais concebidas para monitorizar e controlar infraestruturas e indústrias. Nas redes SCADA, os sistemas de aquisição de dados, os sistemas de transmissão de dados e os *softwares* de Interface

Homem-Máquina (HMI – *Human Machine Interface*) são integrados de modo a fornecer a monitorização centralizada e controlo do sistema para processar entradas e saídas. Estes sistemas são também utilizados para adquirir informações no "campo", transferindo as mesmas para uma instalação central de processamento, sendo de seguida tratadas e apresentadas aos utilizadores (através do HMI) sob a forma de texto, gráficos, entre outros. Como resultado, isto permite que os utilizadores monitorizem e controlem um certo sistema remotamente em tempo real [52].

2.4.1. Distribuição da informação dos HMI

A informação presente nos HMI pode ser distribuída para uma variedade de clientes/utilizadores (*clients*) com auxílio de protocolos normalizados, como por exemplo, o protocolo *Ethernet*. Assim, surge o termo *thin client*, que se refere a um dispositivo projectado e utilizado para comunicações remotas com um servidor. No caso do *software* de HMI/SCADA, um dado projecto é desenvolvido e a aplicação que corre em tempo real é instalada no PC servidor. A informação pode então ser consultada directamente a partir do PC onde a aplicação se encontra instalada, ou a partir de um PC que se encontre dentro da mesma rede utilizando uma ferramenta chamada *Secure-viewer thin client,* que faz com que a aplicação possa ser implementada num PC remoto. Se se quiser aceder às informações que são adquiridas dentro da rede interna a partir de uma rede externa, pode-se utilizar uma aplicação que é designada por *Web thin client*. Esta aplicação corre dentro do *browser* e não é independente como o *Secure-viewer thin client*. Isto dá ao utilizador mais flexibilidade, pois permite acesso a partir de qualquer dispositivo ligado à *Internet* que seja capaz de correr um *browser de Internet*, por exemplo, PC's, *tablets* e *smartphones* [6].

2.4.2. Hardware e Arquitectura

Quanto ao *hardware* necessário para que uma rede SCADA seja implementada, têm-se os seguintes componentes: MSU/MTU – *Master Station Unit/Master Terminal Unit*; sub-MSU's; IED's – *Intelligence Electronic Devices* e RTU's – *Remote Terminal Unit*. É de notar que em algumas aplicações de sistemas SCADA, os sub-MSU's podem não ser utilizados, nestes casos, o MSU está directamente ligado a cada *slave station* RTU ou IED, e estas fornecem uma interface directa que permite controlar e monitorizar

equipamento e sensores. As SSU - *Slave Station Units*, nestes casos, podem ser directamente controladas pelo MSU/MTU, porém, têm que conter algum tipo de programação local para que as mesmas consigam fazer acções sem as instruções directas do MSU ou MTU. Estes componentes acima mencionados encontram-se esquematicamente apresentados na Figura 2.7 que esquematiza a arquitectura de um sistema SCADA genérico simples[52]. Os MSU's ou MTU's armazenam e processam a informação proveniente dos *inputs* e *outputs* das unidades *slave*, RTU's ou IED's, enquanto estas controlam o processo local.

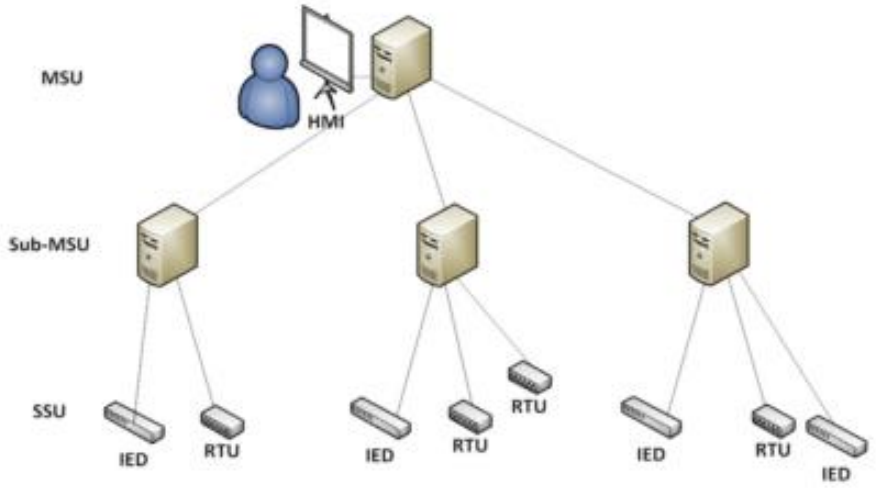


Figura 2.7 : Arquitectura de uma rede SCADA simplificada [52]

Por outro lado, com a evolução da tecnologia e o surgimento da IoT, alguns sistemas SCADA sofreram alterações na sua arquitectura. Na Figura 2.8, podem-se observar os aspectos e componentes principais de um sistema SCADA integrado com IoT. O facto de se conseguir comunicar com o ambiente industrial remotamente a partir de um simples computador portátil ou *smartphone* oferece muitas vantagens às empresas, porém, a integração deste tipo de sistemas implica vulnerabilidades de rede [50].

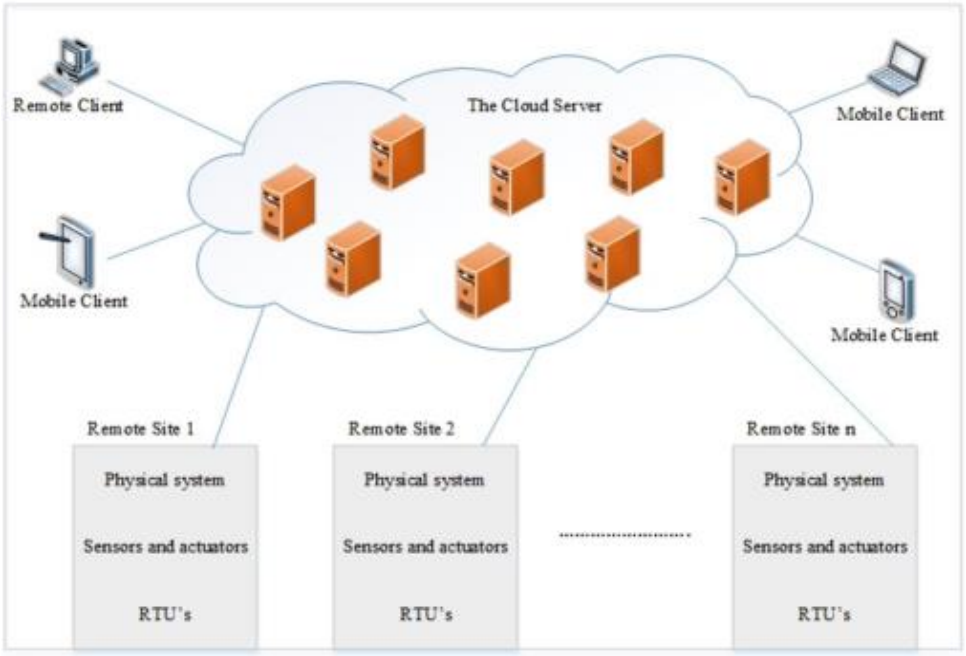


Figura 2.8 : Arquitectura geral de um sistema SCADA num ambiente Nuvem-IoT [50]

Tendo em conta a pesquisa bibliográfica efectuada, pode-se concluir que todas estas tecnologias estão, muitas vezes, interligadas entre si e podem ser frequentemente utilizadas em conjunto para oferecerem soluções industriais. A partir do seguinte capítulo irão abordar-se soluções para o problema enunciado com base em algumas destas áreas.

3. Estrutura do Robô Cartesiano

Esta secção diz respeito à escolha da estrutura a utilizar no robô cartesiano real para o sistema de posicionamento automático de lavagem do interior de camiões cisterna. As dimensões dos camiões cisterna considerados são de aproximadamente: 2,5 metros de largura e 12 metros de comprimento. Pretende-se que o sistema seja capaz de efectuar um ajuste nos diferentes eixos de modo a fazer coincidir o actuador de lavagem com as várias bocas do camião, sequencialmente, para que seja possível concluir o processo de lavagem. No início do processo de lavagem, é também necessário que o camião esteja adequadamente posicionado em relação à estrutura, de maneira a que o camião não esteja fora da zona de actuação do robô de lavagem

3.1. Configuração

Para este sistema foram selecionados produtos Bosch Rexroth [53] com a configuração cartesiana representada na Figura 3.1.

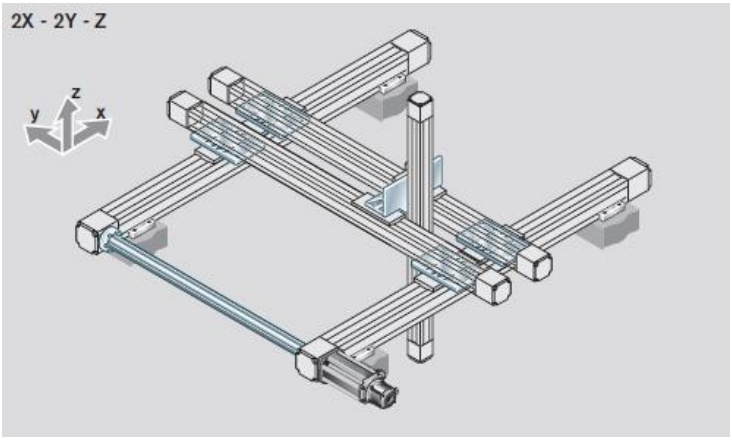


Figura 3.1 : Conceito da configuração do sistema [53]

Nesta gama de produtos, os módulos lineares possuem uma tecnologia patenteada que permite utilizar os mesmos como estão representados na Figura 3.1 ou no sentido inverso da coordenada z. Como tal, confere-se um carácter versátil à solução quanto à fixação ao chão ou ao tecto.

Neste caso, considerar-se-á uma solução fixa ao tecto, de modo a que os esforços aplicados sejam menores devido ao carácter mais compacto da solução, em que cada módulo linear longitudinal estará multiplamente apoiado a um perfil estrutural de modo

a reduzir a sua flecha máxima. O mesmo não se aplica aos módulos transversais devido ao facto destes serem móveis. Quanto à fixação ao tecto, esta será efectuada com perfis estruturais da mesma marca para garantir a compatibilidade da solução.

Quanto a dimensões máximas, considerando o caso a aplicar, assumiu-se que o comprimento dos módulos lineares longitudinais e dos módulos transversais serão de 12 m e 3 m, respectivamente (aproximadamente as dimensões de um camião-cisterna). O comprimento que define a distância entre o tecto e a estrutura será de 2 m (em altura), que corresponderá ao comprimento dos perfis estruturais de suporte.

3.2. Escolha dos módulos lineares

Antes da escolha dos módulos lineares longitudinais, efectuou-se um estudo dos esforços envolvidos, pois um dos pontos críticos da estrutura são as ligações entre os módulos lineares e os transversais, principalmente quanto a momentos flectores. Como tal, fez-se uma simulação simplificada em *Solidworks* com elementos de viga para se determinarem os esforços envolvidos. Consideraram-se os apoios encastrados de modo a maximizar os esforços devido à maior rigidez estrutural imposta (pior caso possível). A carga aplicada é de 1500 N, isto pois assume-se uma massa de 150 kg a transportar nos eixos, correspondente ao robô de lavagem e aos componentes de actuação necessários.

Para efeitos de simulação é necessário definir uma secção no *software* de simulação *Solidworks*. Como fundamentalmente se trata de uma estrutura a sofrer flexão, é necessário definir o seu segundo momento de área bem como a área da secção transversal [54]. Como nos catálogos da Bosch estes parâmetros estavam especificamente indicados, mas a maioria das dimensões da secção transversal não se encontrava disponível para consulta, calculou-se então uma altura e largura equivalente da secção (fazendo uma aproximação conservativa a uma secção rectangular, considerando as cararacterísticas de flexão dos perfis segundo o eixo Y). Estes cálculos efectuaram-se a partir das equações (3) e (4), para se poder simular uma secção com um comportamento à flexão semelhante aos perfis Bosch a selecionar, e consequentemente, descobrir os esforços envolvidos em cada ligação [53].

$$Iy_m = \frac{b^3 \times h}{12} = 3527 * 10^{-8}\ m^4 \quad (3)$$

$$A = b \times h = 0{,}28\ m^2 \tag{4}$$

A partir da equação (3) podem-se calcular o "*b*" (medida horizontal da secção equivalente) e o "*h*" (medida vertical da secção equivalente), e tendo em conta que a secção é aproxidamadente quadrada, tem-se assim a secção equivalente rectangular (quadrada): b = 143.43 mm (Horizontal); h = 143.43 mm (Vertical), como se pode observar no referencial da Figura 3.2. Tendo estes valores, é possível simular os esforços envolvidos na estrutura.

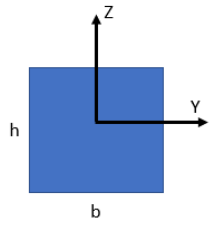


Figura 3.2 : Referencial para o cálculo da secção equivalente

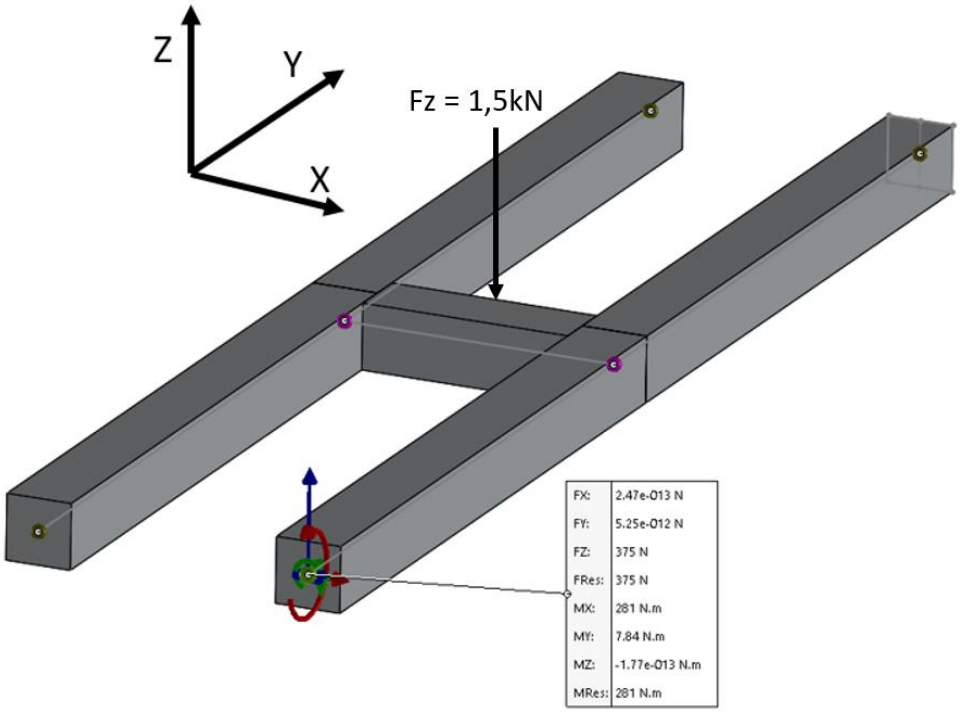


Figura 3.3 : Esforços máximos nas ligações dos módulos lineares

As variáveis Mt Max e ML max estão definidas segundo um referencial do catálogo Bosch [53], presente na Figura 3.4, e correspondem aos momentos flectores My e Mx no referencial da Figura 3.3, respectivamente.

Como se pode observar na Figura 3.3 e na Tabela 3.1, os esforços máximos aplicados não ultrapassam os valores máximos admissíveis do catálogo para as ligações dos módulos lineares MKR-165, portanto considerar-se-ão estes módulos na estrutura a utilizar, tanto longitudinal, como transversalmente. Só os módulos lineares MKR-165 apresentam configurações até 12m de comprimento.

Tabela 3.1 : Cargas admissíveis vs cargas aplicadas das ligações

	Fz max[N] / Fz[N]	**Fy max[N] / Fy[N]**	**Mt max[N.m] /Mt[N.m]**	**ML max [N.m]/ ML[N.m]**
Carga admissível	34100	34100	1445	4160
Carga aplicada	375	0	7.84	281

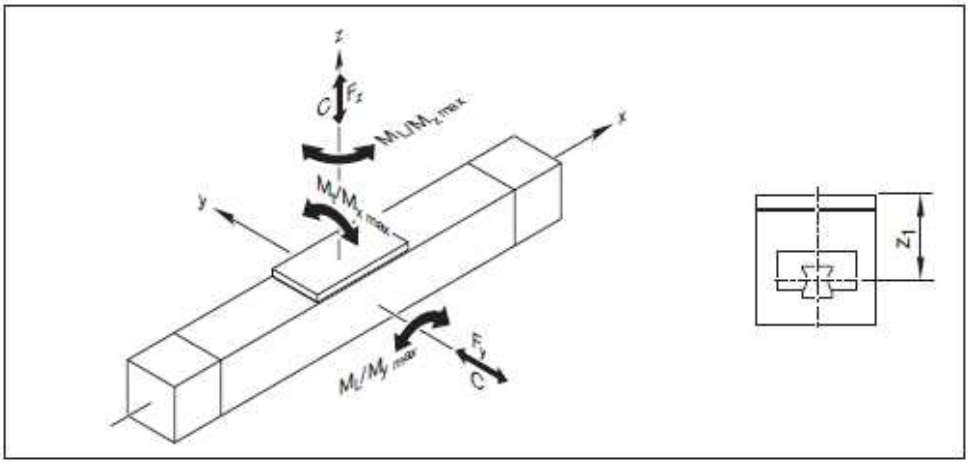


Figura 3.4 : Referencial utilizado no catálogo Bosch [53]

Quanto às deformações máximas admissíveis, analisaram-se os gráficos disponibilizados no catálogo Bosch (Figura 3.5) e concluiu-se qual a distância entre apoios para que os módulos lineares se encontrem em segurança.

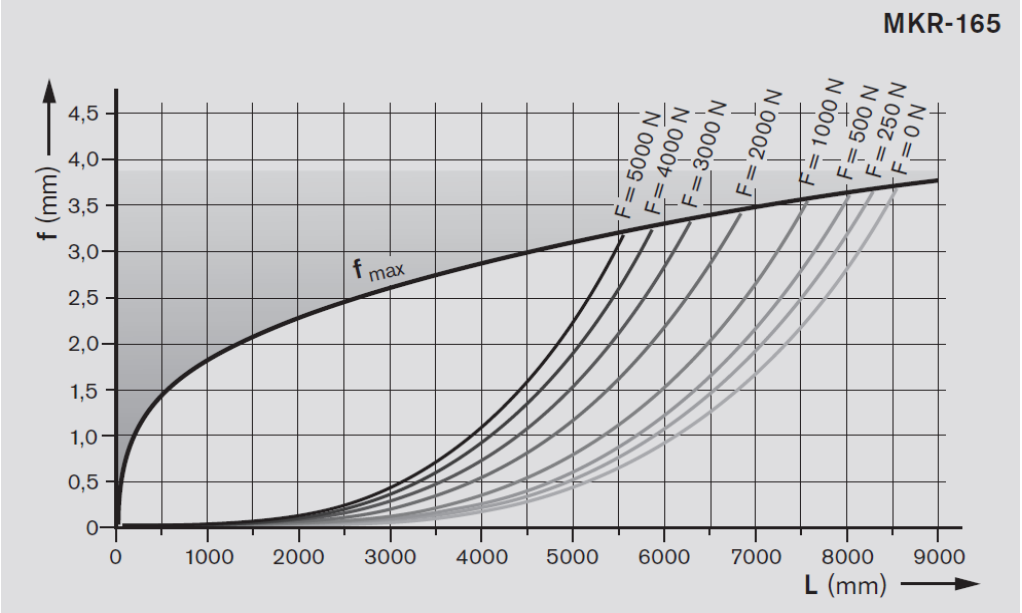


Figura 3.5 : Flecha máxima admissível

Para a situação de carga presente nesta estrutura, quer nos módulos longitudinais, quer nos módulos transversais, o esforço máximo aplicado em Z é de 750 N (curva de 1000 N). Como se pode observar na Figura 3.5, se se colocarem apoios de 3 em 3 metros nos módulos lineares longitudinais, estes encontram-se em segurança quanto à deformada máxima, que é 0,3 mm em 2,5 mm admissíveis. Minimizam-se assim os esforços sofridos pelo módulo linear de forma a prolongar a sua vida útil. Este raciocínio é válido também para os módulos lineares transversais, uma vez que para o mesmo comprimento (3 m) sofrem a mesma flecha máxima.

3.3. Actuação vertical

Para a actuação vertical foi escolhido o tipo de ligação representado na Figura 3.6.

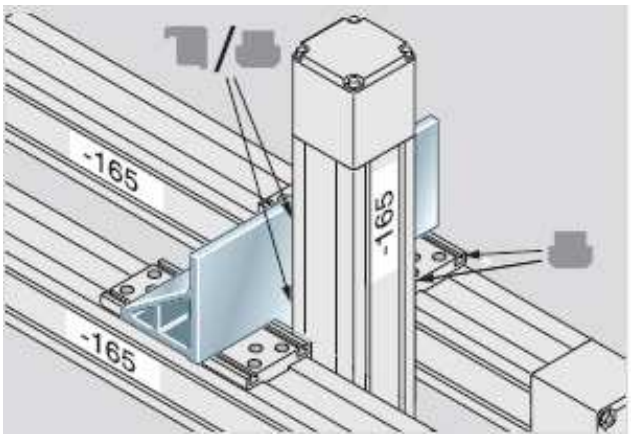


Figura 3.6 : Ligação para a actuação vertical (*Angle Bracket* R0391 210 50)

Este tipo de ligação não possui nenhum valor tabelado quanto a cargas máximas admissíveis. Foram então consideradas as cargas admissíveis para um tipo de ligação (Figura 3.7) que suporta uma força no eixo Z (5200 N), menor do que a ligação escolhida para a actuação vertical (informação cedida pela Bosch).

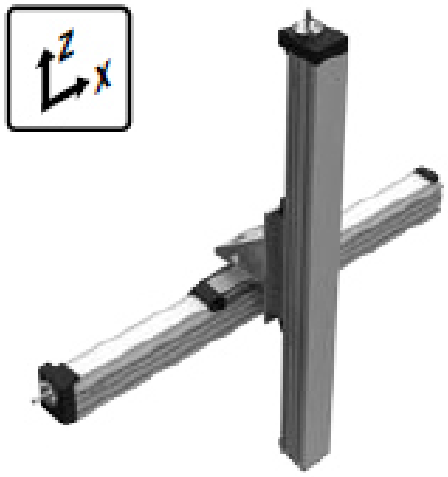


Figura 3.7 : Ligação semelhante à Figura 3.6 com menor carga admissível

3.4. Escolha de perfis estruturais

Na escolha de perfis estruturais verticais considerou-se o perfil de alumínio 100x100L do catálogo da Bosch.

Foi efectuada uma análise de tensões para os perfis estruturais, através de uma simulação em *Solidworks,* simulando as condições críticas de carga para os perfis verticais, quando o actuador está mais próximo da extremidade transversal. Esta simulação foi efectuada com elementos de viga e as ligações consideram-se como se estivessem soldadas (caso mais conservativo) [54]. A carga aplicada é de 1500 N.

Como neste caso existem tensões axiais e também tensões resultantes de esforços de flexão da estrutura, aproximou-se a secção do perfil estrutural a uma secção quadrada, a partir do seu momento de inércia relativamente ao eixo X, que neste caso é igual ao momento de inércia do eixo Y: $Iy_e = \frac{l^4}{12} = 318.3 * 10^{-8}\, m^4$, sendo $l = 78.61mm$, a dimensão lateral da secção quadrada calculada. Este raciocínio é análogo ao efectuado na secção 3.2 (Escolha dos módulos lineares).

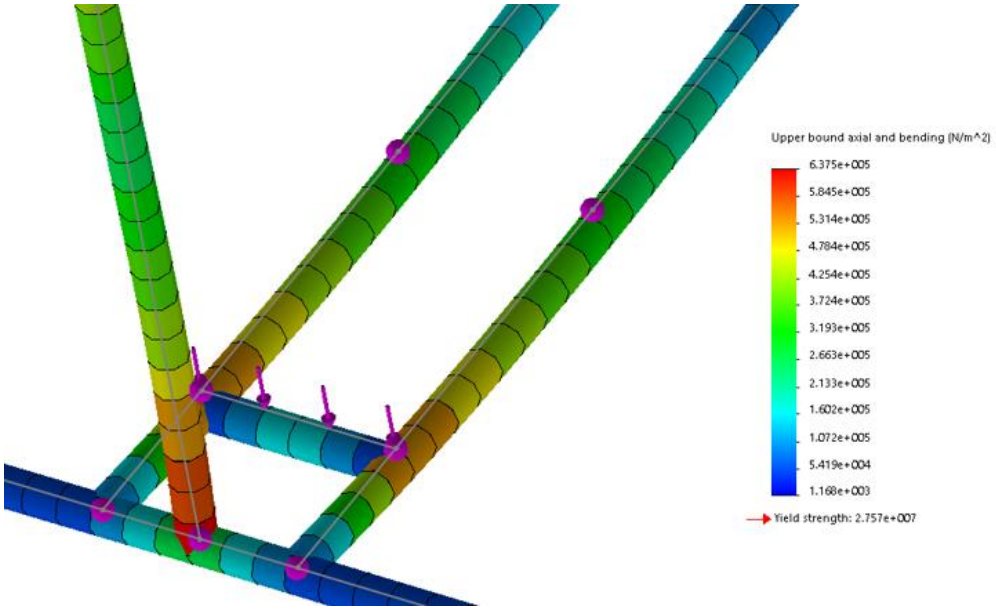


Figura 3.8 : Tensões normais nos perfis estruturais

Na Figura 3.8, pode-se observar que o valor da tensão máxima aplicada em cada perfil estrutural vertical é de $\sigma_p = 637$ kPa, muito inferior ao valor da tensão de cedência da liga 1060 *Alloy* ($\sigma_y = 27.57\ MPa$). Considera-se então que os perfis estruturais estão em segurança.

3.5. Escolha dos actuadores eléctricos

Para o sistema operar correctamente são necessários actuadores eléctricos adequados. Como tal, consideraram-se os motores eléctricos recomendados pela Bosch para mover a carga pretendida, compatíveis com os módulos lineares MKR-165. O motor recomendado cumpre os requisitos pretendidos quanto à movimentação horizontal e vertical da carga (Figura 3.9).

Vertical operation

MSK 076C, HCS02.1E-W0070, 3 x 400 V

i		i = 8					i = 12					i = 16				
m_{ex}	(kg)	20	40	60	80	100	20	40	80	120	160	25	50	100	150	200
t_a	(ms)	103	132	166	205	251	131	154	210	288	401	103	122	171	247	383
s_a	(mm)	155	198	249	308	377	131	154	211	288	402	52	61	85	123	191
a	(m/s^2)	29.2	22.8	18.2	14.7	12.0	15.3	13.0	9.5	7.0	5.0	9.7	8.2	5.9	4.0	2.6
v_{dc}	(m/s)	3.0					2.0					1.0				
*	(mm)	± 0.1					± 0.1					± 0.1				

Horizontal operation

MSK 076C, HCS02.1E-W0070, 3 x 400 V

i		i = 8					i = 12					i = 16				
m_{ex}	(kg)	20	40	60	80	100	50	100	150	200	250	100	200	300	400	500
t_a	(ms)	125	150	175	200	225	170	210	245	280	320	116	146	176	206	236
s_a	(mm)	250	300	350	400	450	215	260	305	350	400	58	73	88	103	119
a	(m/s^2)	32.0	26.0	22.0	20.0	18.0	15.0	12.0	10.3	8.9	7.9	8.6	6.8	5.7	4.8	4.2
v_{dc}	(m/s)	4.0					2.5					1.0				
*	(mm)	± 0.1					± 0.1					± 0.1				

Figura 3.9 : Servo motores indicados para instalar no módulo MKR-165

A velocidade máxima de movimentação da carga é então de 1 m/s, porém, para o caso em estudo, arbitrou-se uma velocidade de 0.2 m/s. Este valor foi arbitado considerando a distância total a ser percorrida pelo robô (12 metros), para que o tempo total correspondende a ser dispendido na deslocação do mesmo durante o processo seja igual a 60 segundos. Deste modo, os módulos lineares têm uma vida útil muito superior à que se obteria com uma velocidade de deslocamento de 1 m/s.

3.6. Vida útil estimada da estrutura

É possível obter uma estimativa da vida útil dos módulos lineares a partir das expressões fornecidas no catálogo da Bosch. Em primeiro lugar, calcula-se a força combinada aplicada ao módulo linear transversal, através da equação (5).

$$Fcomb = Fy + Fz + \frac{C*My}{Mt} + \frac{C*Mx}{ML} + \frac{C*Mz}{ML} \quad (5)$$

Tabela 3.2 : Esforços máximos aplicados na estrutura

Esforço	Valor
$\boldsymbol{Mx}$	$282 \times 1.2 = 338.4\ N.m$
$\boldsymbol{My}$	$7.81 \times 1.2 = 9.37\ N.m$
$\boldsymbol{Mz}$	0
$\boldsymbol{Fy}$	0
$\boldsymbol{Fz}$	$(\frac{150kg}{4}) \times 9.81 \times 1.2 = 441\ N$
$\boldsymbol{Mt}$	$1445\ N.m$
$\boldsymbol{ML}$	$4160\ N.m$
$\boldsymbol{Fcomb}$	$6.432\ kN$

Nota: Os cálculos dos esforços contam com um coeficiente de segurança de 1.2 (Tabela 3.2).

Após o cálculo de *Fcomb*, calcula-se então a vida útil estimada destes módulos em metros, a partir da equação (6), ou em horas, partir da equação (7).

$$L\ [m] = \left(\frac{C}{Fcomb}\right)^3 * 10^5 = 1.19 * 10^8\ metros \quad (6)$$

$$Lh\ [h] = \frac{L[m]}{3600*Vm} = 165601\ h = 6900\ dias = 18.9\ anos \quad (7)$$

Na equação (6), C representa a capacidade de carga dinâmica do módulo linear ($C = 68{,}2kN$). Estes valores foram obtidos considerando uma velocidade média de movimentação do módulo linear igual a 0.2 m/s. Também se considera que o

equipamento está em funcionamento 24h por dia, 365 dias por ano. Este valor de vida útil só contempla o tempo em que os módulos estão em movimento, pois a maior parte do tempo será dispendido na lavagem, enquanto a estrutura se encontra estática.

3.7. Segurança do Sistema

Para se ter a certeza que o camião está devidamente posicionado debaixo da estrutura de lavagem automática, de modo a garantir a segurança do processo de lavagem, foram selecionados sensores fotoeléctricos para o efeito. Estes sensores serão colocados na extremidade frontal da estrutura, como se pode observar na Figura 3.10.

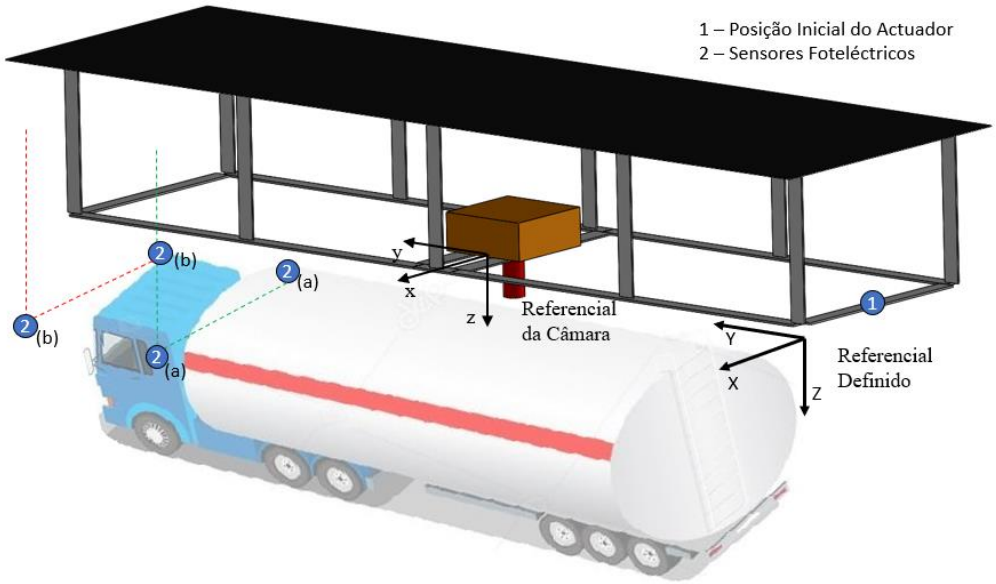


Figura 3.10 : Esquema dos sensores auxiliares e Referencial Definido

No momento do posicionamento do camião, o operador do veículo deve passar pelo sensor fotoeléctrico 2(a), obtendo uma mensagem num painel luminoso (por exemplo): "Avance lentamente até obter o aviso de paragem". Em seguida, quando a cabine do veículo passar pelo sensor 2(b), o operador obterá a mensagem: "STOP", que será quando o tanque cisterna estará na posição correcta para iniciar a lavagem. Assume-se que o sensor 2(b) é colocado numa posição que contemple uma margem de erro que seja razoável tendo em conta o tempo de reacção do operador de modo a parar o veículo.

Como medida de segurança, seleccionaram-se sensores fim de curso com o objectivo de se possuirem variáveis capazes de interromper o processo quando um limite do eixo longitudinal é atingido, criando assim uma redundância caso o

mecanismo de paragem do algoritmo não funcione. Estes sensores fim de curso serão colocados nas duas extremidades longitudinais da estrutura, nas extremidades dos módulos transversais e no eixo z (de modo a limitar o movimento em x, y e z).

Os sensores fim de curso selecionados são da marca OMRON [55] (não é estritamente necessário serem desta marca, têm apenas de ser compatíveis com a solução apresentada), modelo D4B-2115N em aço inoxidável com acção instantânea, de modo a obter-se uma precisão elevada quanto à segurança do sistema (Figura 3.11)

Quanto aos sensores fotoeléctricos, selecionou-se o modelo com feixe de barreira E3Z (dois sensores opostos) da OMRON, com um alcance máximo de 30 metros (Figura 3.11). No caso dos sensores fotoeléctricos é necessário um cuidado de limpeza mínimo para que o funcionamento dos mesmos não seja comprometido devido à sujidade e/ou vapores de água. De qualquer modo, os sensores E3Z da OMRON possuem uma protecção contra substâncias contaminantes, como as referidas anteriormente, aumentando assim a fiabilidade do sistema.

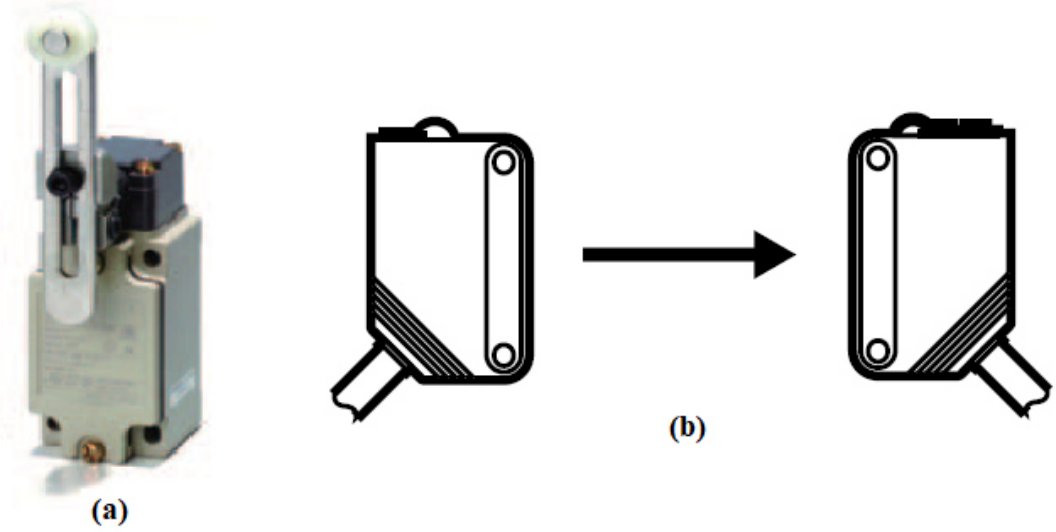


Figura 3.11 : Sensores fim de curso (a) e fotoeléctricos (b) [55]

4. Controlo Baseado em Visão Computacional

Neste projecto, surgiu a necessidade de se incorporar visão computacional no sistema, de modo a tornar o processo flexível, na medida em que funcione correctamente independentemente do número de bocas/depósitos que os camiões-cisterna possam apresentar e da posição do camião quando é estacionado. Como tal, foi programado um algoritmo no editor de *Matlab*, de modo a implementar-se o controlo do sistema a partir dos dados extraídos a partir da visão de uma câmara, neste caso, uma câmara USB.

4.1. Algoritmo de Visão

O algoritmo descrito a seguir refere-se ao controlador do protótipo, que por sua vez é descrito no capítulo 4. Quando se corre o programa, a primeira opção que é apresentada ao utilizador é se pretende entrar no modo Manual ou Automático. Se escolher o modo Manual, então o utilizador tem a liberdade de mover qualquer um dos 3 motores nas 2 direcções correspondentes a cada um, com um conjunto de comandos apresentado a seguir:

- w: O motor Y move-se no sentido positivo (frente) do referencial definido (Figura 4.5);
- s: O motor Y move-se no sentido negativo (trás) do referencial definido;
- a: O motor X move-se no sentido positivo (esquerda) do referencial definido;
- d: O motor X move-se no sentido negativo (direita) do referencial definido;
- q: O motor Z move-se no sentido positivo (cima) do referencial definido;
- e: O motor Z move-se no sentido negativo (baixo) do referencial definido;
- x: Interrompe o modo Manual, para todos os motores e acaba o programa.

Nas Figuras 4.1 e 4.2, pode-se observar de que forma foi implementado este modo no *Matlab,* tendo em conta que a função utilizada para registar as teclas pressionadas (*getkey*) foi programada por terceiros (Jos van der Geest) e publicada no fórum de utilizadores do *Matlab* [56].

```
function manual(cam,ctr)

img=getsnapshot(cam);
f = figure('Name','Visão','NumberTitle','Off');
imshow(img);

char=0;

    while char ~='x';

    char1=getkey(1,'non-ascii');
    char=num2str(char1);

        switch char

            case 'w'
            ctr.moveForwardApproach();
            pause(1);
            ctr.stop();

            case  's'
            ctr.moveBackwardApproach();
            pause(1);
            ctr.stop();

            case 'a'
            ctr.moveLeft();
            pause(1);
            ctr.stop();

            case 'd'
            ctr.moveRight();
            pause(1);
            ctr.stop();
```

Figura 4.1 : Implementação modo “Manual” (1ª parte)

```
35
36 -                 case 'q'
37 -                 ctr.moveUp();
38 -                 pause(1);
39 -                 ctr.stop();
40
41 -                 case 'e'
42 -                 ctr.moveDown();
43 -                 pause(1);
44 -                 ctr.stop();
45
46 -             end
47 -         end
48 -         ctr.stop();
49 -     end
```

Figura 4.2 : Implementação modo “Manual” (2ª parte)

Este modo “Manual” foi implementado de maneira a que o operador tenha controlo nos movimentos dos motores, se necessário. Caso exista alguma avaria que seja necessário resolver, será mais fácil se houver controlo manual do sistema, e não só o automático.

As instruções que são enviadas para os motores são discretas, ou seja, cada vez que se carrega numa tecla, o respectivo motor executa a instrução correspondente durante 1 segundo. Só ao fim desse tempo é que o algoritmo aceita outro *input*. Esta foi a maneira mais expedita de se chegar a este resultado com o *hardware* disponível, que não permite instruções contínuas, como por exemplo, o motor mover-se enquanto se carrega na tecla e quando se solta a mesma, o motor parar.

Quando se carrega na letra “x”, os motores param todos completamente, o modo “Manual” é interrompido e o programa chega ao fim. Para se voltar a utilizar o modo automático, terá que se correr outra vez o programa.

```
1   function [vid] = initializeCamera()
2 -     vid=videoinput('winvideo',2,'YUY2_320x240');
3 -     src = getselectedsource(vid);
4 -     vid.FramesPerTrigger = Inf;
5 -     src.Brightness = -15;
6 -     src.Contrast = 35;
7       %src.FrameRate= 15;
8 -     start(vid);
9 -   end
```

Figura 4.3 : Função Inicialização da Câmara

Ao iniciar o algoritmo de posicionamento automático, a primeira função a ser executada é a denominada *videoinput* (função do *Matlab*), como se pode observar na Figura 4.3, que adquire, em tempo real, o vídeo proveniente da câmara USB. As imagens são obtidas através da função *getsnapshot* do *Matlab (Image Acquisition Toolbox)*. A partir do momento em que são adquiridas imagens com sucesso, efectua-se a detecção de círculos, esta é efectuada pela função *imfindcircles*, pertencente à toolbox de visão computacional do *Matlab*.

A função *Imfindcircles* utiliza o princípio da Transformada de *Hough* Circular para detectar círculos numa imagem. Esta detecção pode ser efectuada de duas formas, sendo a primeira a detecção de círculos escuros em imagens claras e a segunda o oposto. Neste caso utiliza-se a primeira opção, assumindo que as bocas de lavagem têm profundidade suficiente para serem "escuras" em contraste com a cor do camião.

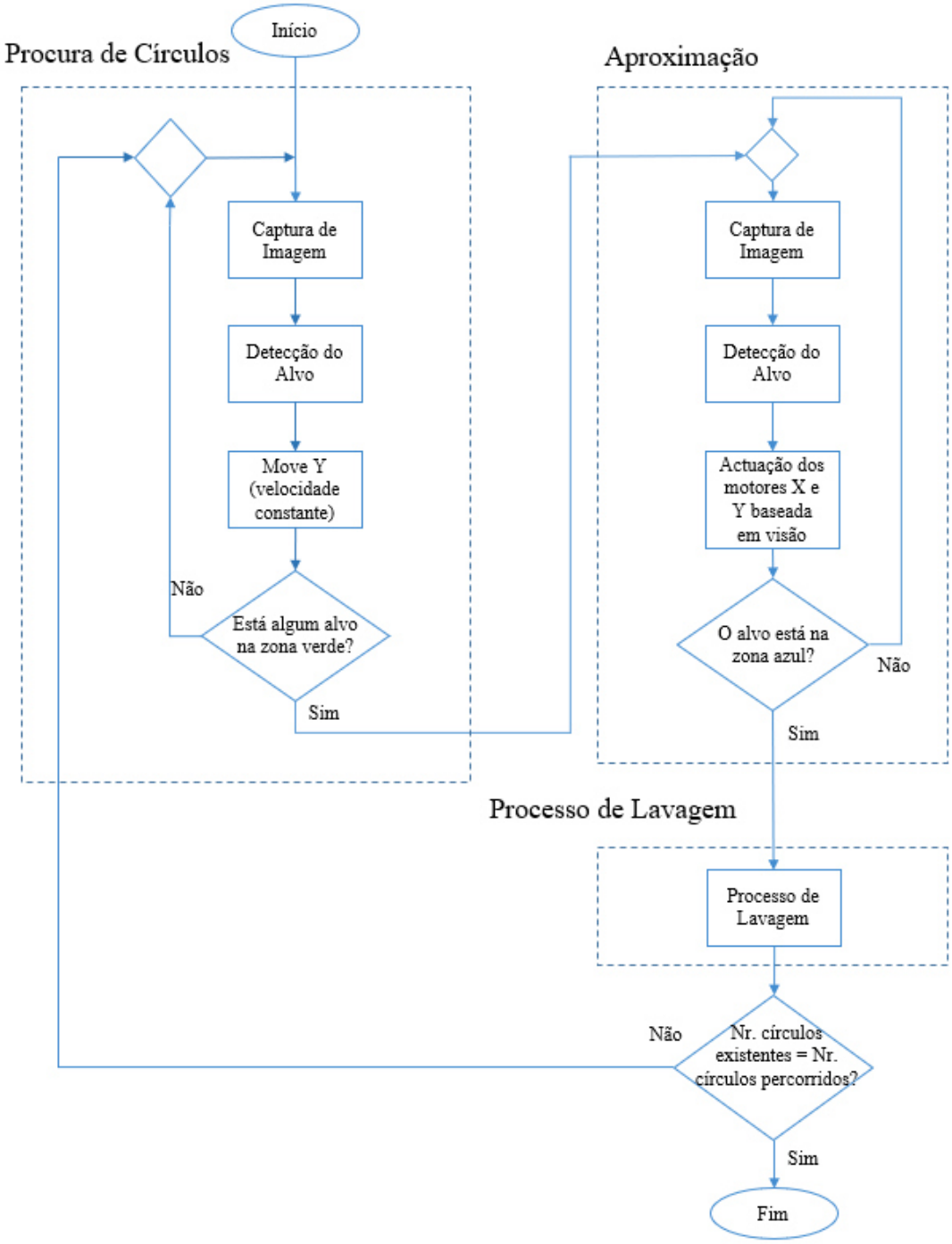


Figura 4.4 : Fluxograma do Algoritmo de Visão

Para que esta detecção se faça com sucesso, é necessário estudar as condições de iluminação em cada caso. Após a detecção de círculos, se for mais que um detectado, o furo escolhido é sempre o que tem a coordenada Y mais próxima, segundo o referencial definido (Figura 4.5).

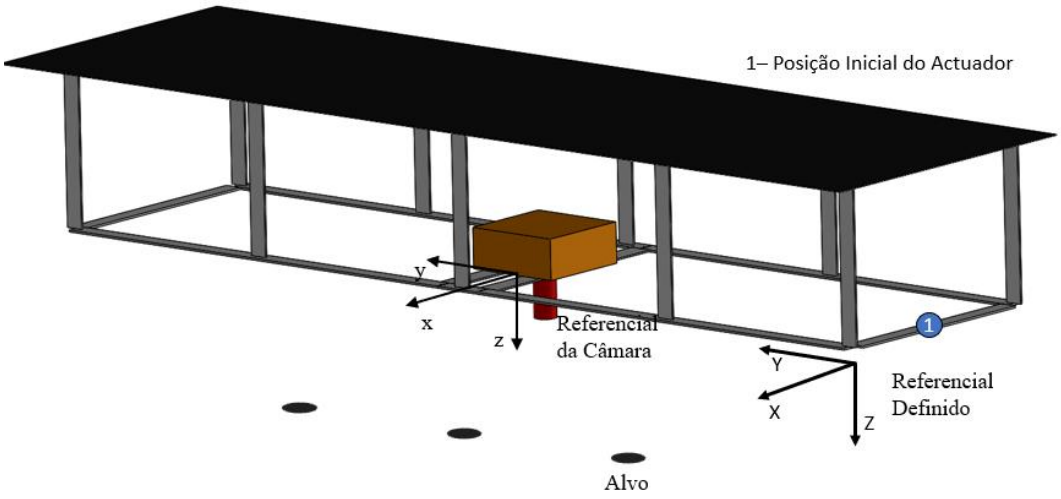


Figura 4.5 : Representação dos alvos e Referencial Definido

Quando se inicia o algoritmo (fluxograma da Figura 4.4), o modo de Procura de Círculos, descrito de seguida, é o primeiro a ser executado assim que se obtêm imagens da câmara. Os dois modos possíveis de controlo deste sistema são os seguintes:

- Procura de círculos: quando não há nenhum círculo detectado ou os centros das bocas de lavagem não estão dentro da zona verde representada na Figura 4.6, o motor X permanece parado e o motor Y movimenta-se no sentido positivo do movimento até encontrar um círculo dentro da zona verde representada na mesma figura. Quando isto acontece, o sistema entra no modo de Aproximação.

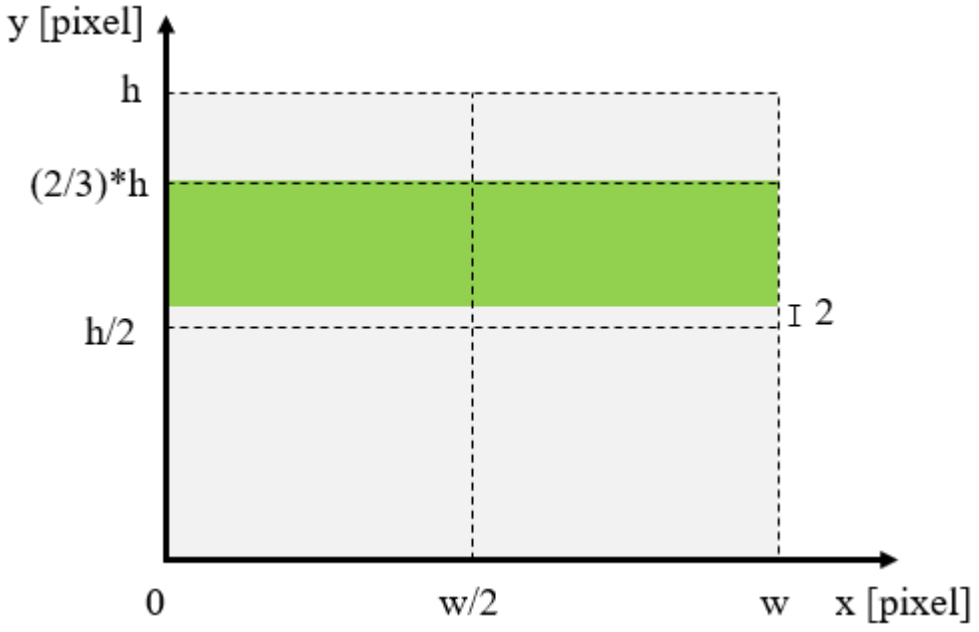


Figura 4.6 : Áreas de “decisão” do modo de Procura de Círculos

```
function [next] = idle(resolution,hole,ctr,t)
    next=State.Idle;
    ctr.moveForwardRoutine();
    t.UserData.dirY=1;
    if ~isempty(hole)
        if hole(2)<(resolution(2)/2)-2 && hole(2)>(resolution(2)/3)
            ctr.stopY();
            next=State.Approach;
        end
    end
end
```

Figura 4.7 : Função “Procura de círculos”

- Aproximação e Actuação: os motores actuam de acordo com o erro (diferença entre o centro da imagem e o centro da boca de lavagem, na equação (8)). Quando esses pontos coincidem (com uma tolerância de 1 pixel, Figura 4.8), é efectuada a actuação do sistema de lavagem (eixo Z), e também uma pausa que representa o tempo de lavagem. Nessa altura, e depois da actuação em z ser efectuada, o sistema entra novamente no modo “Procura de Círculos” (Figura 4.7). A actuação da lavagem implementou-se ao enviar uma instrução para o motor Z se movimentar uma certa distância arbitrada, simulando a descida do dispositivo de lavagem na situação real.

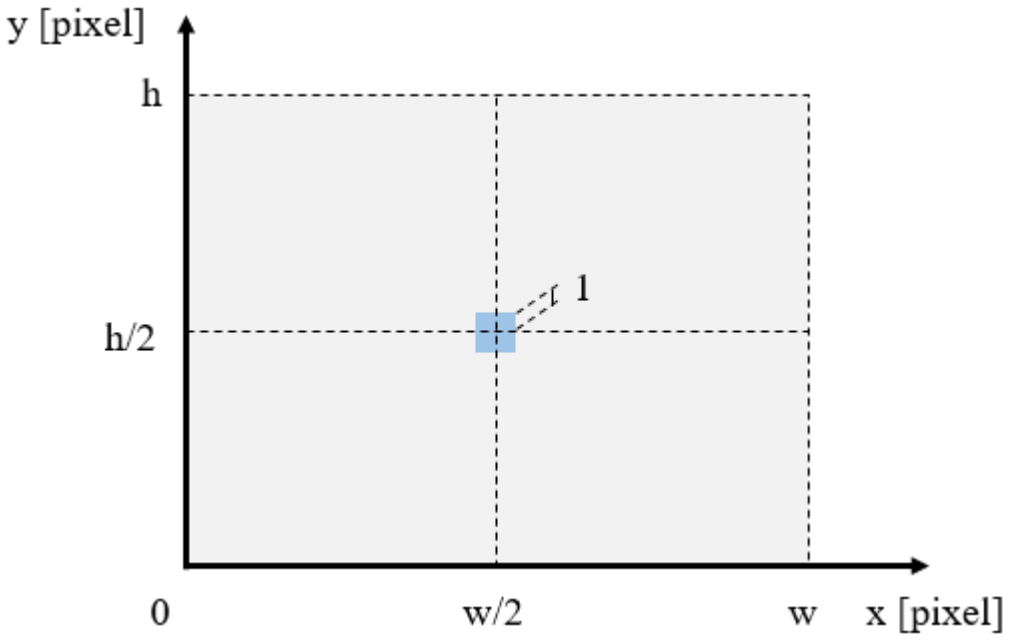


Figura 4.8 : Áreas de “decisão” do modo de Aproximação

$$E = \left[\frac{w}{2}, \frac{h}{2}\right] - [x_{cd}, y_{cd}] \quad (8)$$

A equação (8) diz respeito ao erro do sistema (distância do centro do círculo detectado ao centro da imagem), o objectivo é minimizar este erro de modo a fazer coincidir o centro do círculo detectado $[x_{cd}, y_{cd}]$, com o centro da imagem $\left[\frac{w}{2}, \frac{h}{2}\right]$.

```
function [next] = approach(resolution,hole,ctr,t)
    next=State.Approach;
    y=0;
    x=0;
    pixel_interval=1;
    if hole(2)<(resolution(2)/2)+pixel_interval && hole(2)>(resolution(2)/2)-pixel_interval
        ctr.stopY();
        y=1;
        t.UserData.dirY=0;
    elseif hole(2)>=(resolution(2)/2)+pixel_interval
        t.UserData.dirY=-1;
        ctr.moveBackwardApproach();
    elseif hole(2)<=(resolution(2)/2)-pixel_interval
        t.UserData.dirY=1;
        ctr.moveForwardApproach();
    end

    if hole(1)<(resolution(1)/2)+pixel_interval && hole(1)>(resolution(1)/2)-pixel_interval
        ctr.stopX();
        x=1;
        t.UserData.dirX=0;

    elseif hole(1)>=(resolution(1)/2)+pixel_interval
        if y<=0
            t.UserData.dirX=-1;
            ctr.moveSlowRight();
        else
            t.UserData.dirX=-1;
            ctr.moveRight();
        end
    else
        if y<=0
            t.UserData.dirX=1;
            ctr.moveSlowLeft();
        else
            t.UserData.dirX=1;
            ctr.moveLeft();
        end
    end

    if x>0 && y>0
        next=State.Clean;
    end
```

Figura 4.9 : Função “Aproximação” e “Actuação”

O sistema muda de modo de funcionamento, através de um *switch-case* implementado no *Matlab*, como se pode observar na Figura 4.10.

Nota: Nas Figuras 4.7, 4.9 e 4.10 podem-se observar comandos relativos à interface com a *Internet*. Estes comandos foram desenvolvidos no âmbito de um trabalho em conjunto com o bolseiro de investigação Manuel Dias no projecto do IPL:ROBOTCLEAN [57], que auxiliou a criação da interface do algoritmo com a *Internet*, bem como a parte das ligações eléctricas e conexão com a placa controladora.

```
try
    Mode=input('Modo Manual - 0 ; Modo automático - 1 :  ');
    camera=initializeCamera();
    resolution=camera.VideoResolution;
    state=State.Idle;
    count=0;
    numHoles=input('Introduza Nr Furos :  ');
if Mode==0
    manual(camera,ctr);
else
    t=createTimer(ctr);
    server=Server(8081);
    start(t);
    while count<numHoles
        pause(0.01);
        if ~isempty(server.Connections)
            [c,img]=getCircles(camera);
            sendData(server,t.UserData,img,state);
            if server.isOnline()==1
                nearestHole=findNearestHole(c);
                switch state
                    case State.Calibration
                        state=calibration(resolution,ctr);
                    case State.Idle
                        state=idle(resolution,nearestHole,ctr,t);
                    case State.Approach
                        state=approach(resolution,nearestHole,ctr,t);
                    case State.Clean
                        pause(0.5);
                        sendData(server,t.UserData,img,state);
                        clean(ctr);
                        state=State.Idle;
                        count=count+1;
                        fprintf('Cleaned holes: %d\n',count);
                        pause(2);
                end
            else
                ctr.stop();
            end
        else
            ctr.stop();
        end
    end
```

Figura 4.10 : *Script* principal da implementação do algoritmo de controlo baseado em visão

Como se pode observar na Figura 4.10, antes do sistema começar a mover-se, é definida uma variável **ctr** referente ao controlador. Para se identificar o controlador, foi definida uma classe Controller(), cuja única propriedade é a comunicação com a placa do controlador (*serial*), e os métodos são as várias acções que os motores de passo podem tomar (Tabela 4.1) [57].

Tabela 4.1 : Métodos da classe Controller()

Método	Descrição
GetState(obj,axis)	Obtém a informação do estado actual do motor do eixo "axis" (variável de entrada)
moveForwardRoutine(obj)	Move-se 117 mm (700 graus) no sentido positivo com uma velocidade de 90 cm/min [15 rpm]
moveForwardApproach(obj)	Move-se 16.75 mm (100 graus) no sentido positivo com uma velocidade de 30 cm/min [5 rpm]
moveBackwardApproach(obj)	Move-se 16.75 mm (100 graus) graus no sentido negativo com uma velocidade de 30 cm/min [5 rpm]
moveDown(obj)	Move-se 33.5 mm (200 graus) no sentido positivo com uma velocidade de 60 cm/min [10 rpm]
moveUp(obj)	Move-se 33.5mm (200 graus) no sentido negativo com uma velocidade de 60 cm/min [10 rpm]
moveLeft(obj)	Move-se 22 mm (1000 graus) no sentido positivo com uma velocidade de 48 cm/min [60 rpm]
moveSlowLeft(obj)	Move-se 11 mm (500 graus) no sentido positivo com uma velocidade de 30 cm/min [37.5 rpm]
moveRight(obj)	Move-se 22mm (1000 graus) no sentido negativo com uma velocidade de 48cm/min [60rpm]
moveSlowRight(obj)	Move-se 11 mm (500 graus) no sentido negativo com uma velocidade de 30 cm/min [37.5 rpm]
stop(obj)	Pára imediatamente o funcionamento de todos os motores
stopX(obj)	Pára imediatamente o funcionamento do motor X
stopY(obj)	Pára imediatamente o funcionamento do motor Y
stopZ(obj)	Pára imediatamente o funcionamento do motor Z
disconnect(obj)	Desliga a conexão do *Matlab* com o controlador

O processo de lavagem chega ao fim quando o robô de lavagem acaba de actuar na última boca da cisterna. De modo a que o algoritmo tenha a informação do número total de bocas presentes no camião, foi definido um parâmetro a introduzir pelo operador antes do processo começar. Este é o único parâmetro introduzido pelo operador.

4.2. Simulação do Algoritmo de Visão

De modo a replicar as condições do processo a automatizar, elaborou-se um modelo de realidade virtual que contém, em ambiente de simulação, uma câmara e um cilindro com vários furos a representar as bocas do camião-cisterna. Através do *Matlab/Simulink*, e com ligação à *Toolbox* de Realidade Virtual foi implementado um algoritmo de comando do pórtico de modo a simular um ciclo de lavagem de uma cisterna. Esta simulação foi elaborada com os objectivos de validar o mecanismo de detecção das bocas da cisterna, o servo-controlo das juntas baseado em imagem e o algoritmo de comando do ciclo de lavagem. O modelo de Realidade Virtual tem as seguintes características: O raio do tanque é 1.2m, o comprimento total da cisterna é 7m e as bocas de lavagem têm um raio de 0.25m. Pode-se observar o esquema de controlo da simulação na Figura 4.11.

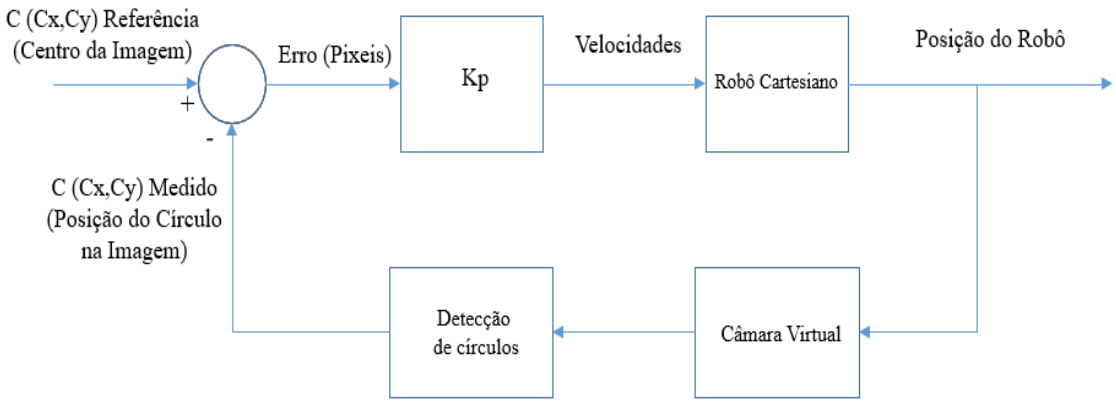


Figura 4.11 : Esquema de Controlo da Simulação

Tabela 4.2 : Variáveis do Processo

Variável	Descrição	Tipo	Dimensão	Unidade
Cref	Coordenadas do centro da imagem (referência)	Vector	1x2	Pixéis
Cm	Coordenadas do centro do círculo detectado (medido)	Vector	1x2	Pixéis
Erro	Cref - Cm	Vector	1x2	Pixéis
Kp	Ganho	Escalar	1x1	Adimensional
V	Velocidade (Kp*Erro)	Vector	1x2	Pixéis

Após a imagem ser captada pela câmara virtual, esta é processada no bloco do *Simulink* denominado "Processamento de Imagem" (Figura 4.13). Neste bloco, a imagem é convertida de cores para escala de cinzentos, transformada do formato *uint8* para *double*, e de seguida, é introduzida no bloco denominado "Bloco de Controlo". Neste bloco estão implementados os mecanismos de decisão relativamente à actuação dos motores e também está presente a função de reconhecimento de círculos *Imfindcircles* (função do *Matlab*) que tem como *outputs* as coordenadas de um círculo (Cm_x,Cm_y) (se presente), no referencial da imagem. Esta função tem como parâmetros de entrada o raio mínimo e o raio máximo de detecção (em pixéis), portanto para cada caso de aplicação deste algoritmo tem que se estabelecer uma correspondência entre distâncias reais e distâncias em pixéis. As variáveis do processo estão descritas na Tabela 4.2.

Depois das velocidades dos motores X e Y serem calculadas através da expressão $V = Kp * Erro$, em que o Kp é o ganho, estes sinais são integrados com as condições iniciais (x=0.5; y=4), obtendo-se uma posição que é realimentada no bloco *Mux*, de modo a actualizar a posição da câmara virtual no *VR Sink* do *Simulink*. Na Figura 4.12

pode-se observar uma sequência de acontecimentos da simulação, sendo que a Figura 4.12 (c) representa o momento em que o centro da câmara está coincidente com a boca de lavagem. Na mesma figura, a transição de (a) para (b) corresponde ao ajuste proporcional efectuado pelos motores. O Kp implementado foi de 0.006, e foi obtido através de tentativa e erro, de modo a obter-se uma simulação com uma velocidade adequada para a ilustração da cadeia cinemática do processo.

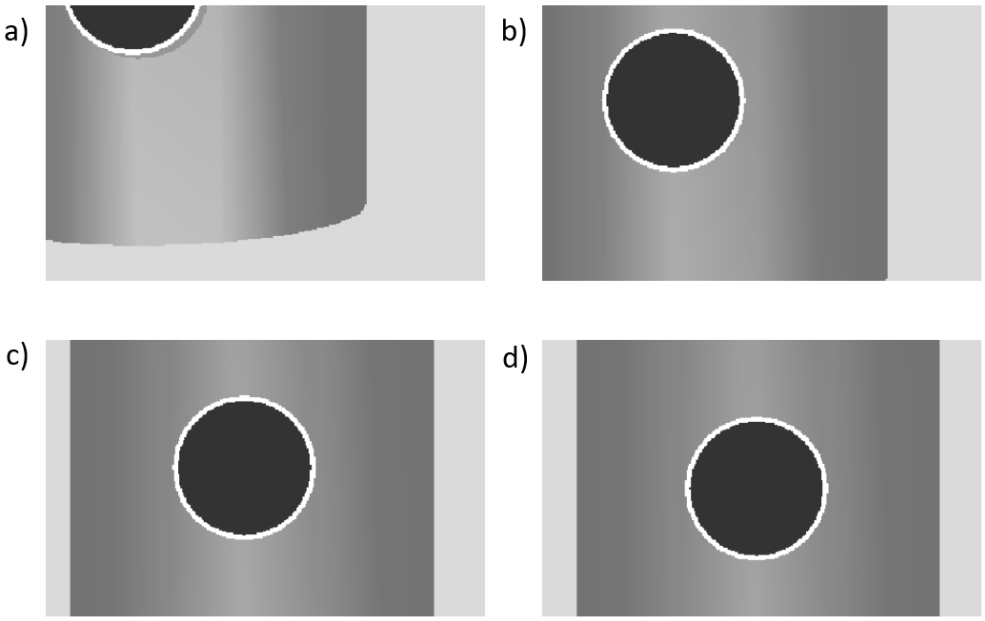


Figura 4.12 : Sequência cronológica de imagens retiradas a partir da câmara virtual

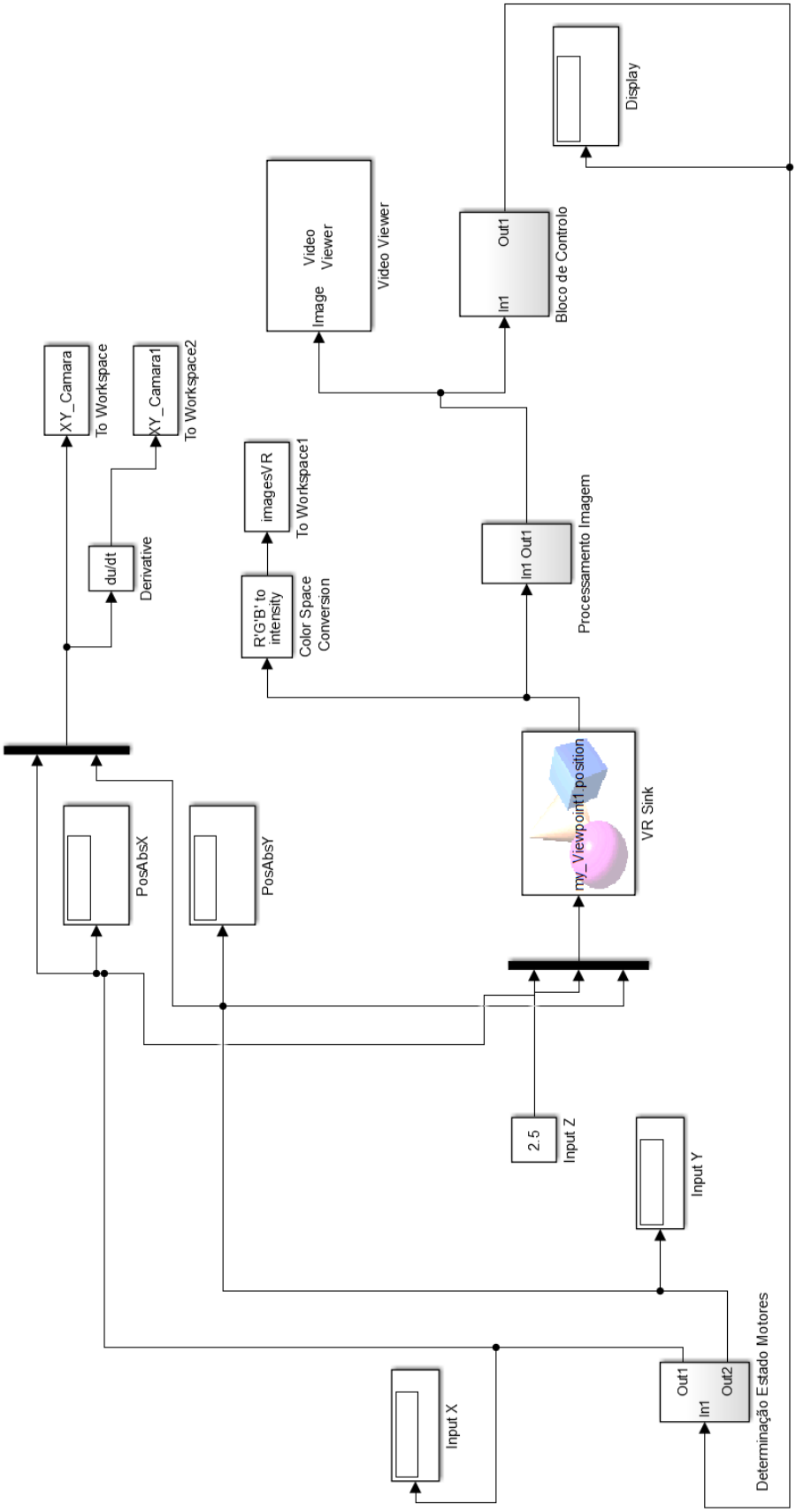


Figura 4.13 : Esquema de Simulink da simulação efectuada

O funcionamento dos motores X e Y, durante a execução de uma operação de lavagem, está evidenciado nas Figuras 4.14 - 4.17. Como se pode observar no funcionamento do motor que actua em X (Figura 4.14), a zona (1) representa a fase de procura de círculos e a zona (2) representa os ajustes efectuados para que a câmara se mantenha colinear em Y em relação ao círculo detectado. A zona (3) diz respeito à fase de processamento, ou seja, o processo de lavagem. A distância que está indicada na ordenada no gráfico da Figura 4.14, representa a distância percorrida em X até ao ajuste com o centro do círculo detectado.

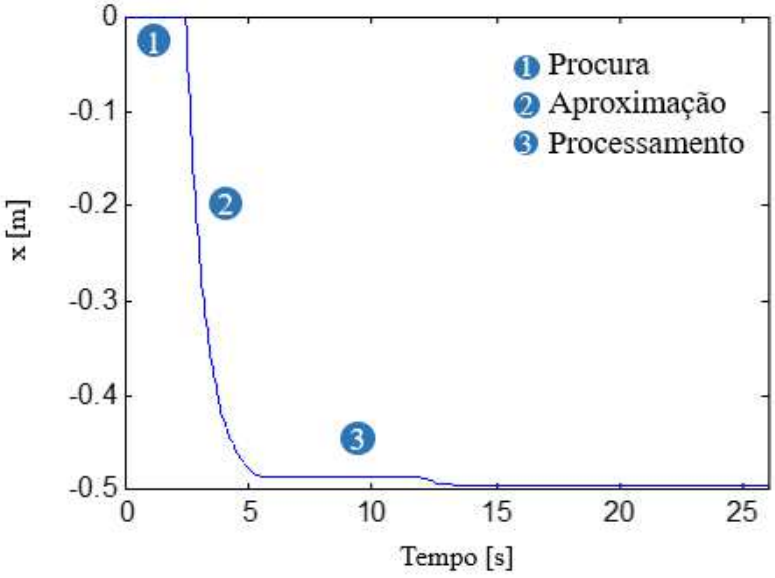


Figura 4.14 : Posições do Motor X na Simulação (Distância Percorrida vs Tempo [s])

Na Figura 4.15 observa-se um pico de velocidade na zona (1), que corresponde à aceleração que o motor X imprime para que o eixo do x fique centrado com o centro do círculo detectado e escolhido. As zonas (2) correspondem às alturas em que o motor X está parado, nomeadamente no início da execução do algoritmo e quando já está centrado com o círculo escolhido. Na zona (3) mais à direita na mesma figura, observam-se oscilações de velocidade, que correspondem a pequenos ajustes do motor X.

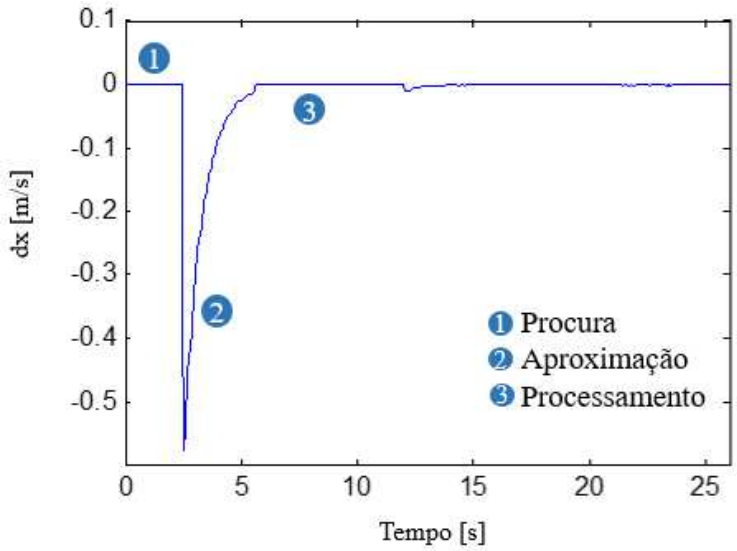


Figura 4.15 : Velocidades do Motor X na Simulação (Velocidade vs Tempo[s])

Quanto ao funcionamento do motor Y (Figura 4.16), as zonas (1) representam o avanço constante do robô cartesiano até encontrar o próximo círculo, seja este o primeiro a ser detectado ou não. As zonas (2) correspondem à fase de aproximação do motor em Y. Os pontos (3) representam a actuação da lavagem quando o centro da imagem e do círculo detectado estão coincidentes dentro de uma tolerância definida (1 pixel). Na fase de aproximação são observadas curvas na imagem, isto é, devido à acção proporcional implementada na simulação, que faz com que à medida que o alvo está mais próximo do objectivo, a velocidade vai diminuindo proporcionalmente.

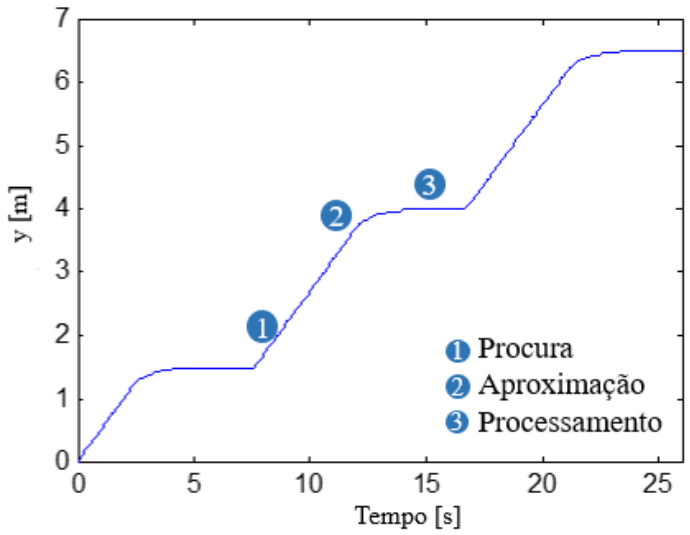


Figura 4.16 : Posições do Motor Y na Simulação (Distância Percorrida vs Tempo[s])

Na Figura 4.17, as zonas (1) correspondem à velocidade máxima que o motor Y atinge quando está no modo "Procura de círculos". As zonas (2) correspondem à desaceleração gradual do motor Y quando o sistema encontra o círculo alvo e entra no modo de "Aproximação". Logo de seguida, como volta a entrar no modo "Procura de Círculos", atinge a sua velocidade máxima novamente até detectar o próximo círculo.

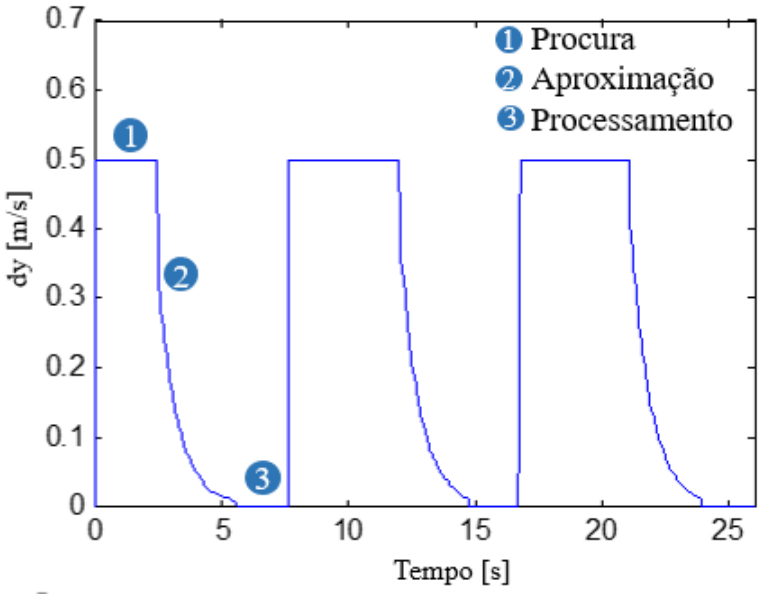


Figura 4.17 : Velocidades do Motor Y na Simulação (Velocidade vs Tempo[s])

5. Protótipo do Robô Cartesiano

De modo a que seja possível a validação do algoritmo de visão e respectiva simulação, foi construído um protótipo à escala aproximada de 1:10 em ambiente laboratorial ao abrigo do projecto IPL/2017/ROBOTCLEAN/ISEL [8].

5.1. Hardware Protótipo

5.1.1. Estrutura

Os elementos estruturais escolhidos para a construção do protótipo à escala foram selecionados no catálogo da Vslot-Europe [58]. Os perfis utilizados foram os seguintes:

- 2 *Vslot Linear Actuator Bundle* (Correia) – 1 metro – Eixo Y (*Nema 17 Stepper Motor*) (Figura 5.1).

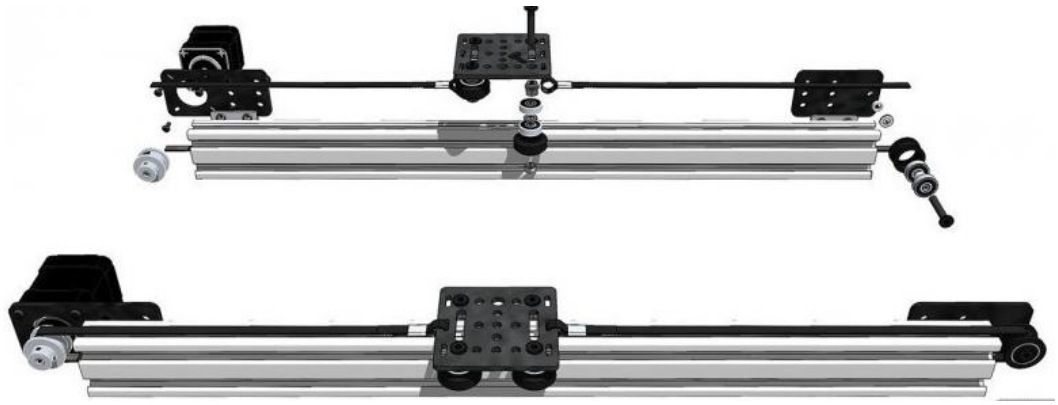


Figura 5.1 : Vslot Linear Actuator Bundle

- 1 *Vslot Linear Actuator Bundle* (Correia) – 0,5 metro – Eixo Z (Nema 17 *Stepper Motor*).
- 1 *C-Beam Linear Actuator Bundle* (Fuso) – 0,5 metro – Eixo X (Nema 23 *Stepper Motor*) (Figura 5.2).

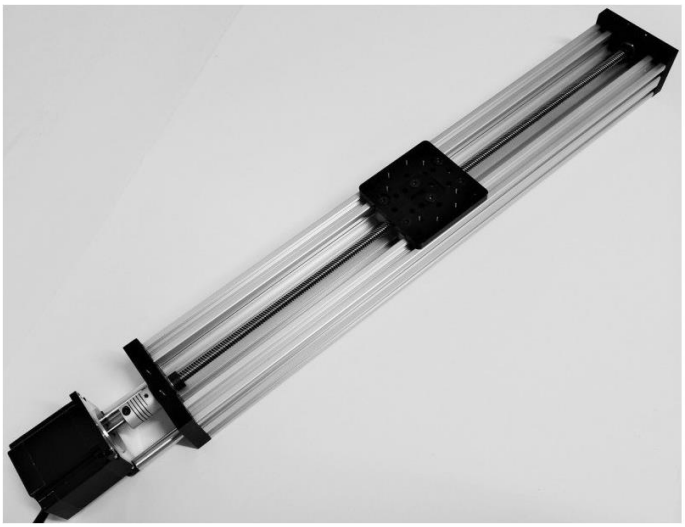


Figura 5.2 : C-Beam Linear Actuator Bundle

5.1.2. Electrónica

Quanto a componentes electrónicos, foi utilizada uma fonte de alimentação de 24V; 17A; 400W (Figura 5.3).

Figura 5.3 : Fonte de Alimentação

Para o controlo dos motores de passo, foi utilizado o controlador CNC xPRO *Controller Stepper Driver* V3. Este controlador é baseado na arquitectura no *Arduino ATmega328p* e permite enviar directamente instruções em ângulos de rotação aos motores a partir do Ambiente de Desenvolvimento (IDE - *Integrated Development Environment*) do *Arduino*. Este controlador (Figura 5.4) possui 4 *drivers* DRV 8825, de modo a proteger os motores de picos de corrente. Foi também desenvolvido *software*

para enviar instruções a partir do *Matlab* (Anexo B), através de um objecto de software de comunicação com o controlador (**ctr**).

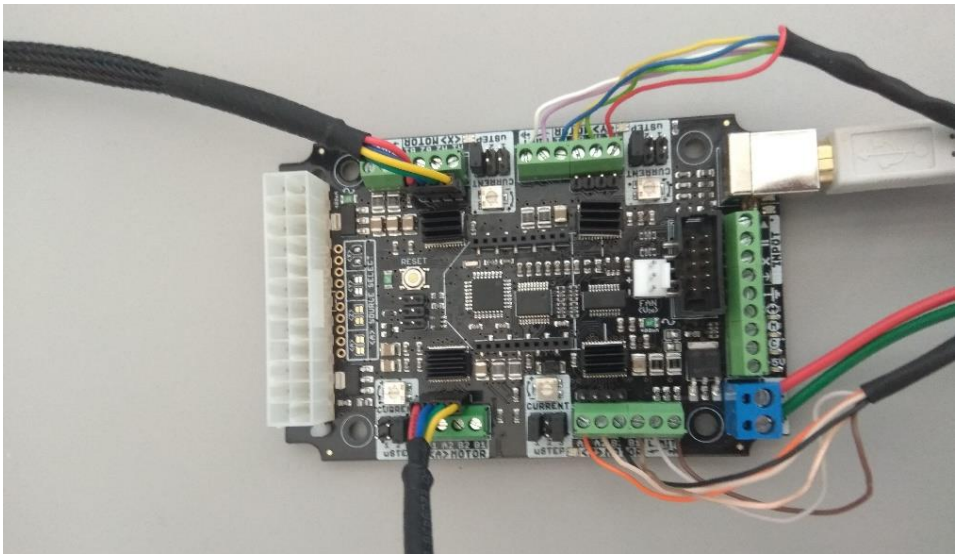


Figura 5.4 : Controlador CNC xPRO Controller Stepper Driver

Estes objectos, da classe Controller dispõem de métodos para enviar vários tipos de instruções para os motores (Tabela 5.1).

Tabela 5.1 : Comandos do Controlador

Descrição do Comando	Comando	Exemplo
Pedido do estado dos drivers	<m'eixo'>	<mx>
Mover o motor em ângulo	<r'eixo''ângulo'>	<ry-360>
Mover o motor em milímetros	<'eixo''milimetros'>	<z15>
Definir velocidade do motor	<s'eixo''velocidade em RPM'>	<sx60>
Pedido da velocidade instantânea	<v'eixo'>	<vy>
Parar o motor	<S'eixo'>	<Sx>

A câmara utilizada foi uma *Webcam* da HP com conexão por USB (Figura 5.5). A câmara têm 5MP e uma taxa de fotogramas máxima de 30.

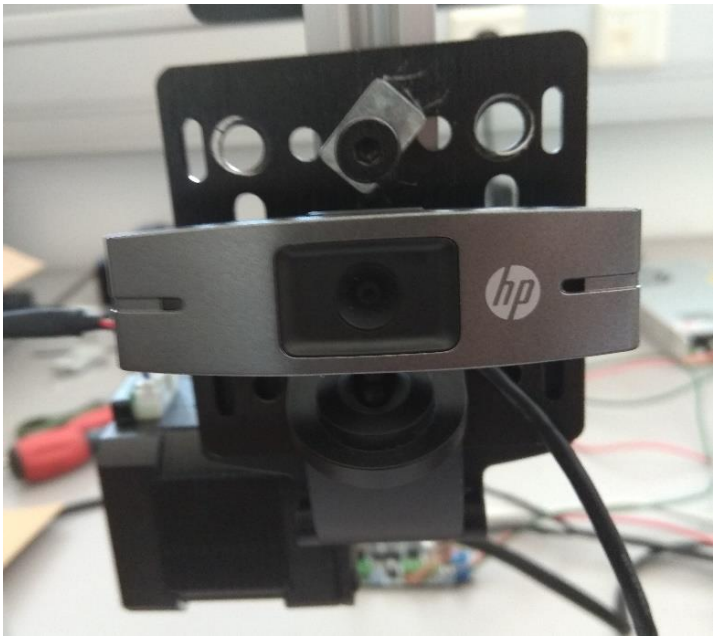


Figura 5.5 : *Webcam* HP

Os motores utilizados para a actuação do sistema foram motores de passo, *Nema* 17 e *Nema* 23, para os actuadores com correia e com fuso, respectivamente.

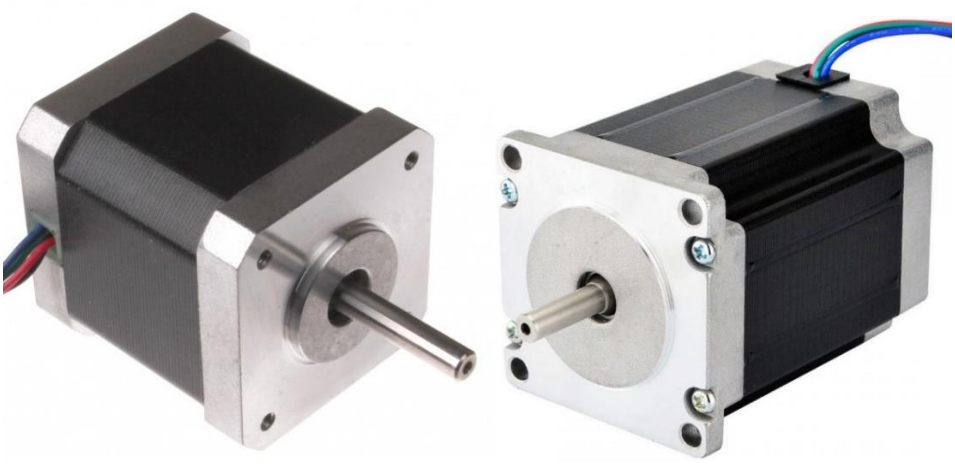


Figura 5.6 : Motores de Passo Nema 17 (Esquerda) e Nema 23 (Direita)

Na Figura 5.7 pode-se observar a montagem efectuada do protótipo no laboratório de Robótica da Área Departamental de Engenharia Mecânica.

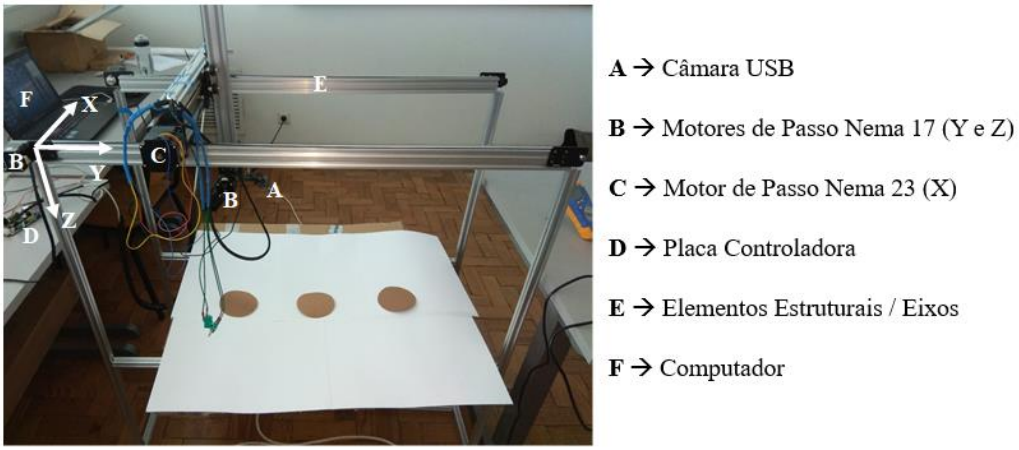


Figura 5.7 : Fotografia da Montagem do Protótipo

5.2. Arquitectura do Protótipo Experimental

A arquitectura do sistema pode-se observar na Figura 5.7. Esta foi desenvolvida com os objectivos de controlar os motores de passo, processar as imagens e permitir a supervisão local e remota (utilizando um *browser*). Os motores de passo são controlados pela placa de controlo, que recebe informação do computador, que por sua vez é o que processa as imagens obtidas através da câmara, ligada por USB. A comunicação com a *Internet* é efectuada através de um servidor, que recebe e envia informação ao computador, através do protocolo de comunicação *WebSocket*. O Rx/Tx representa a comunicação por série. Na Figura 5.8 observa-se o motor A, que representa o motor clone do motor Y, ou seja, o segundo motor actuante no eixo Y.

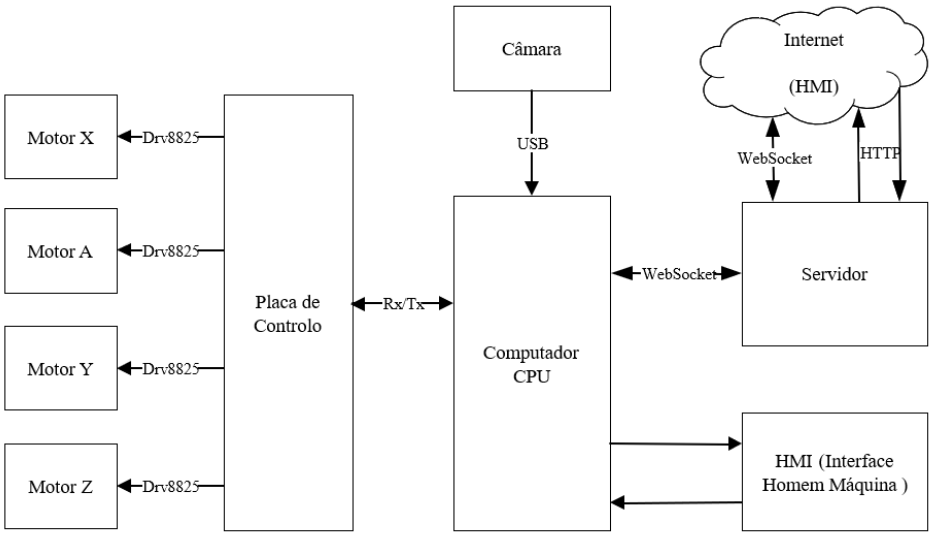


Figura 5.8 : Arquitectura do Sistema [57]

5.3. Configuração do sistema

No protótipo que foi montado, não foi possível implementar a acção proporcional inicialmente pensada (como se pode observar na Figura 5.9, o controlador não é proporcional, é simplesmente do tipo *ON/OFF*), pois sempre que se envia uma instrução para o controlador, é definida a velocidade e a distância a percorrer (parâmetros do movimento) antes do motor começar a actuar. O motor só acaba de actuar quando a instrução é cumprida até ao fim, a menos que a instrução seguinte seja contrária ao movimento que está a ser efectuado. Para o controlo proporcional ser possível, é necessária uma alteração de velocidade no mesmo sentido cada vez que a imagem seguinte é analisada, e para a alteração de velocidade ser efectuada, o controlador necessita de interromper a instrução anterior, mudar de parâmetros e voltar a enviar outra instrução. Isto faz com que a acção proporcional resulte num movimento descontínuo, como tal, optou-se por se enviarem instruções sempre à mesma velocidade (uma velocidade constante superior no caso do modo "Procura de Círculos", e uma velocidade inferior, também constante, no modo de "Aproximação"), com a condição de garantir que o centro da imagem e o círculo detectado coincidam no modo de aproximação, com uma dada margem de erro (1 pixel). Isto faz com que o movimento efectuado pelo robô de lavagem seja contínuo, como requerido.

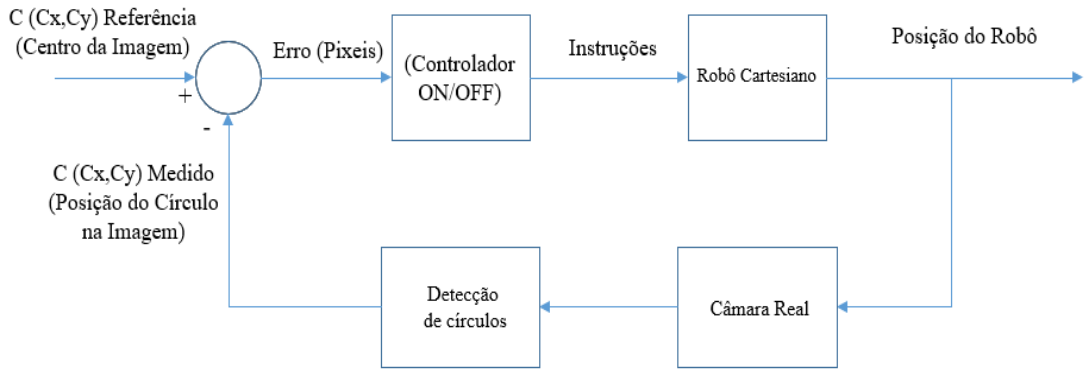


Figura 5.9 : Esquema de Controlo do Protótipo

5.3.1. Posição, Velocidade e Aceleração dos Motores

A aceleração e a desacerelação lineares foram calculadas de modo a que se demore 0,5s a atingir a velocidade máxima ou a desacelerar até a uma velocidade nula. Este tempo foi obtido através de tentativa e erro com vista a que a aceleração não seja

demasiado elevada, para não provocar movimentos bruscos, nem demasiado baixa, para que o robô não demore muito tempo a atingir a velocidade máxima. A aceleração dos motores é dada pela equação (9).

$$a_m = \frac{vel_{RPM} * 200}{60} \left[\frac{passos}{minuto \times segundo}\right] \quad (9)$$

A velocidade, dada em rotações por minuto, quando multiplicada por 200 (número de passos para uma volta completa), dá o número de passos que o motor avança em 1 minuto. Ao dividir esse valor por 60 segundos, obtém-se o número de passos por minuto que o motor acelera por cada segundo. O tempo estabelecido para a aceleração/desaceleração total dos motores foi de meio segundo.

Dentro da biblioteca do *driver* DRV8825, existem duas funções para mover o motor com os seguintes parâmetros de entrada: o número de passos ou o ângulo que o utilizador desejar. Para se conseguir adquirir um valor de distâncias percorridas em milímetros, executou-se uma instrução para os motores darem uma volta completa (360º), e assim mediram-se as distâncias que cada um dos eixos percorreu. Estas relações foram obtidas empíricamente por se terem tido dificuldades em encontrar todos os parâmetros geométricos das transmissões. No caso do eixo Y e Z (correia), uma volta completa corresponde a um deslocamento de 60mm e no eixo X (fuso) corresponde a um deslocamento de 8mm. Com esta relação, e assumindo que é directamente proporcional, chega-se então à equação (10), que relaciona a instrução do ângulo do motor com a distância a que corresponde esse ângulo (D_m) e a distância de uma volta completa (360º) do motor.

$$D_m\,[mm] = \frac{Instrução[º] \times VoltaCompleta[mm]}{360} \quad (10)$$

$$vel_{linear}\left[\frac{mm}{min}\right] = VoltaCompleta[mm] \times vel_{RPM} \quad (11)$$

A equação (11) diz respeito ao cálculo da velocidade linear dos motores de passo.

5.4. Resultados Experimentais

Foram produzidos gráficos a partir do protótipo que representam o funcionamento dos motores, tal como foi feito na secção 4.2. Podem-se observar os resultados obtidos nas Figuras 5.10 – 5.13.

Na Figura 5.10, as zonas (2) dizem respeito à fase de procura de círculos, e as zonas (1) representam o ajuste que o motor X efectua, de modo a aproximar o centro do círculo do objectivo (centro da imagem). O eixo das ordenadas representa a distância percorrida pelo robô de lavagem de modo a fazer o ajuste do centro da imagem com o centro do círculo detectado em X. Neste caso, foram feitos dois ajustes, o primeiro a 45mm à direita da posição inicial, e o segundo, no sentido contrário, cerca de 25mm.

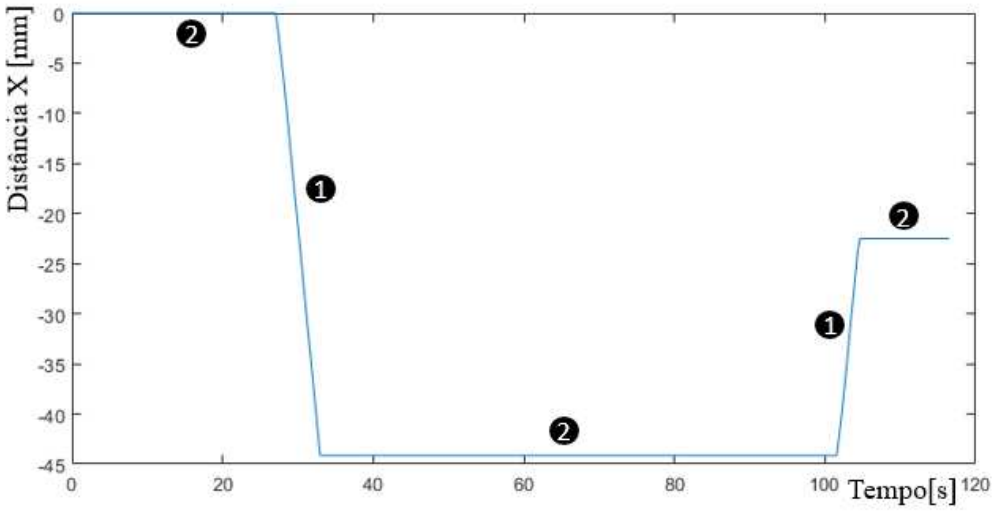


Figura 5.10 : Posições do Motor X no Protótipo (Distância Percorrida vs Tempo[s])

Na Figura 5.11, as zonas (1) dizem respeito à fase de ajuste do motor X, e as zonas (2) representam a fase de procura de círculos.

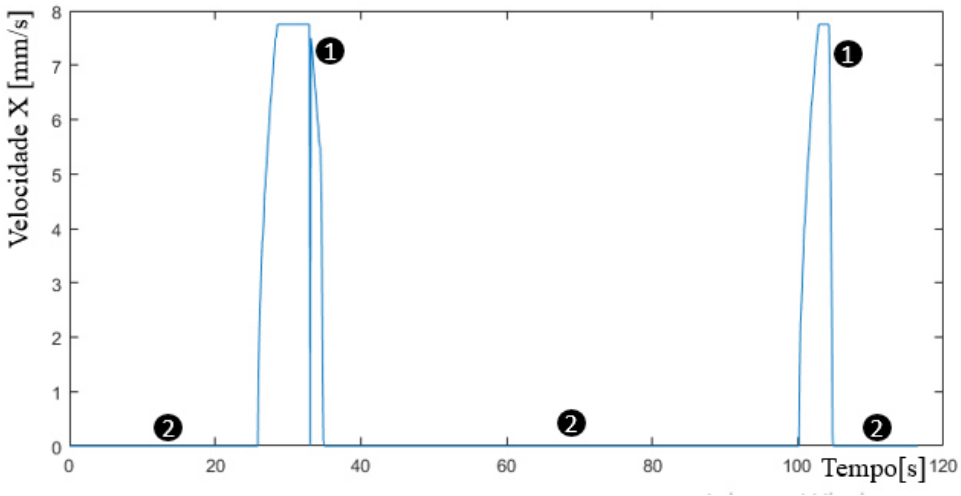


Figura 5.11 : Velocidades do Motor X no Protótipo (Velocidade vs Tempo[s])

Na Figura 5.12, estão representadas as várias fases do movimento dos motores em Y, as zonas (1) representam a fase de procura de círculos, as zonas (2) representam a fase de aproximação do círculo em relação ao centro da imagem e as zonas (3) representam a paragem dos motores relativos ao eixo Y enquanto o robô efectua o processo de lavagem.

Como se pode observar na Figura 5.12, na fase de aproximação, a distância percorrida aumenta linearmente, isto porque no caso do protótipo, a actuação dos motores é linear (devido aos motivos explicados anteriormente), ao contrário da simulação, em que a velocidade é proporcional à distância restante entre o centro da imagem e o centro do círculo detectado.

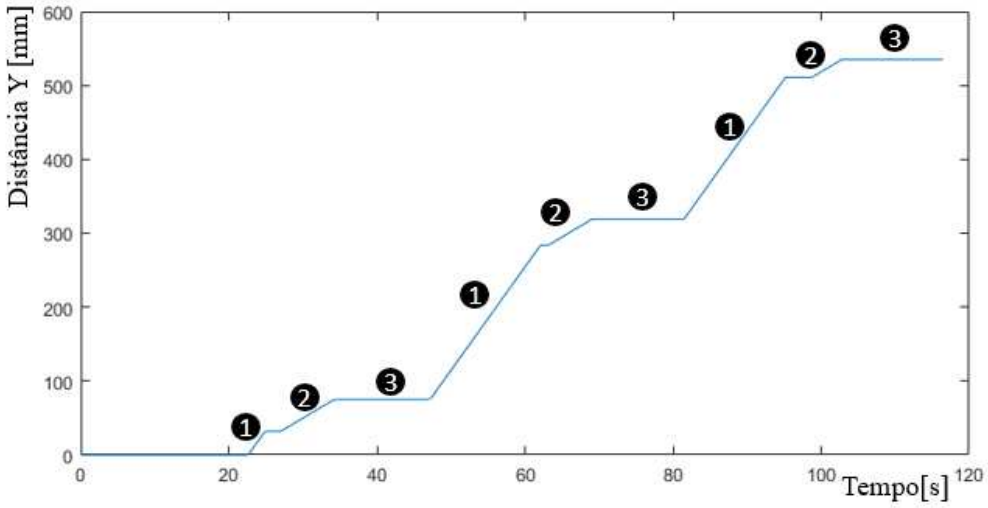


Figura 5.12 : Posições do Motor Y no Protótipo (Distância Percorrida vs Tempo[s])

Na Figura 5.13, as zonas (1) representam a fase de procura de círculos, as zonas (2) representam a fase de aproximação do círculo detectado em relação à imagem capturada, e as zonas (3) representam o momento da lavagem. A descontinuidade que se observa na zona (2) corresponde a um ajuste efectuado pelo motor Y.

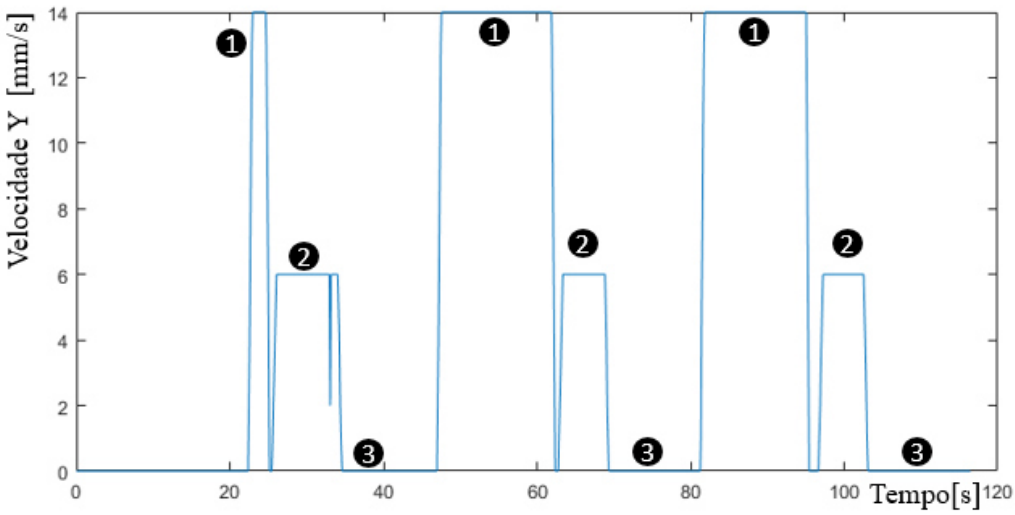


Figura 5.13 : Velocidades do Motor Y no Protótipo (Velocidade vs Tempo[s])

5.5. Supervisão e Controlo do Processo

No âmbito do conceito da indústria 4.0, na realização deste Trabalho Final de Mestrado, decidiu-se explorar a comunicação do *Matlab* (Controlo) com uma página de *Internet* [57]. O objectivo é supervisionar e controlar o processo a partir de qualquer dispositivo que tenha um browser instalado e ligação à *Internet*, como por exemplo, um computador, *tablet* ou *smartphone*.

A maneira escolhida para a transmissão dos dados do *Matlab* foi por TCP/IP (*Transfer Control Protocol / Internet Protocol*), utilizando o protocolo de comunicação *WebSocket*. Este protocolo permite que tanto o servidor, como o cliente, possam transmitir dados entre si sem precisarem de confirmação de que podem transmitir. Assim sendo, a recepção e transmissão de dados são feitas por eventos, isto é, o servidor não tem que estar constantemente a ler da porta correspondente, mas sim espera por cada evento.

Para colocar os dados na *Internet*, foi necessário criar um servidor HTTP (*Hypertext Transfer Protocol*), e o servidor escolhido para o efeito foi o *Apache*. Optou-se por este servidor, porque é gratuito e é compatível com o protocolo HTTP, o que permite a comunicação por *WebSocket*.

Para a implementação da comunicação *WebSocket*, foi criado um servidor no mesmo computador em que corre o algoritmo de visão (implementado no *Matlab*). Na biblioteca *WebSocket* do *Matlab*, o servidor é um objecto da linguagem *JavaScript*, portanto foi necessário adicionar bibliotecas do *JavaScript* ao *Matlab* [57]. Neste protótipo, o servidor *WebSocket* só recebe dois comandos por parte do cliente, sendo o primeiro fazer com que o processo se inicie, e o segundo fazer com que o processo pare.

Para a transmissão de dados, foi necessário definir identificadores (Tabela 5.2), de maneira a que o cliente consiga reconhecer que foram transmitidos dados, sejam estes uma imagem, estado do algoritmo ou botões, distâncias ou velocidades. Se o cliente receber alguma informação em que nenhum destes identificadores estejam presentes, este ignora os dados recebidos.

Tabela 5.2 : Identificadores de Dados

Descrição do identificador	Identificador
Uma imagem	img:
Estado atual do botão de comando	Button:
Distância percorrida pelo eixo X	X_Distance:
Distância percorrida pelo eixo Y	Y_Distance:
Velocidade instantânea do eixo X	X_Speed:
Velocidade instantânea do eixo Y	Y_Speed:
Estado do algoritmo	State:

5.5.1. Apache WebServer

Para se criar o servidor *Apache*, foi utilizado o ambiente de desenvolvimento XAMPP (***X-****Cross-Platform;* ***A****pache;* ***M****ariaDB;* ***P****HP;* ***P****erl*). Para ligar o servidor, abre-se o painel de controlo do XAMPP e carrega-se no botão "*Start*". Depois deste passo, o cliente pode-se ligar ao servidor, fazendo um pedido de página de *Internet* alocada na porta em que o servidor *Apache* está ligado.

Antes de se ligar o servidor *Apache*, é necessário garantir que a porta TCP que o servidor está a abrir, não está ocupada por outro processo do computador. Uma das maneiras de verificar se uma porta está ocupada ou não, é escrever na linha de comandos do sistema operativo o seguinte: "netstat -na". Conseguem-se ver assim, todas as portas ocupadas pelo computador [57].

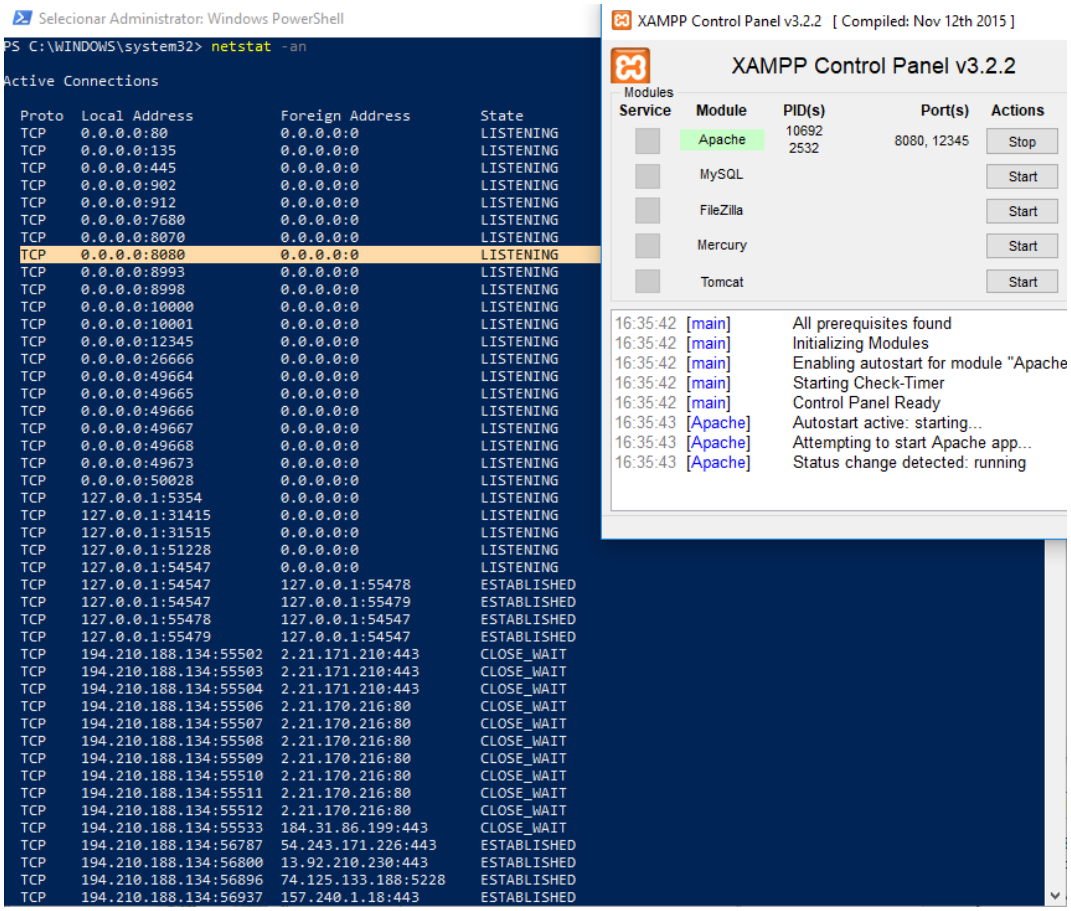


Figura 5.14 : Estado das Portas do Computador

Pode-se observar no XAMPP (Figura 5.14) que as portas a serem ocupadas pelo servidor são as 8080 e 12345. Como se pode verificar na mesma figura (linha de comandos do *Windows*), as portas 8080 e 12345 estão abertas a comunicação, portanto a ligação com o servidor pode ser estabelecida.

5.5.2. Página de *Internet*

A página de *Internet* é composta por três ficheiros: index.html, cliente.js e template.css (Anexo C). O ficheiro index.html, serve para formatar a estrutura e divisão dos componentes da página. O ficheiro cliente.js, tem como objectivo garantir a lógica da transmissão e recepção de dados via *WebSocket*. O ficheiro template.css é onde estão programados os estilos e formatações da página de *Internet* [57].

A página está dividida em três partes, o comando para a ligação ao servidor *WebSocket*, as informações do robô e uma caixa de texto.

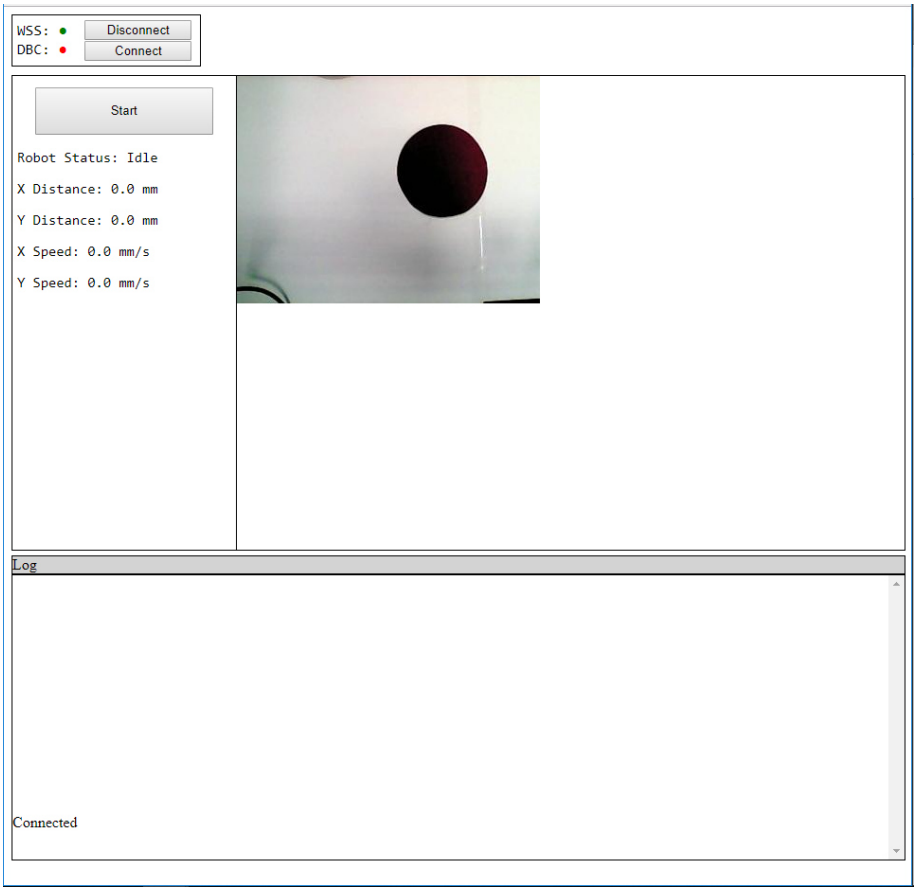


Figura 5.15 : Página Web de Controlo e Supervisão

Como se pode observar na Figura 5.15, é possível assistir às imagens a serem transmitidas em tempo real, assim como enviar instruções para o *Matlab* (*Start* e *Pause*), assim como a informação do estado em que o robô se encontra.

6. Conclusões

O capítulo 6 expõe, na secção 6.1, as conclusões retiradas sobre o presente Trabalho Final de Mestrado, analisando os objectivos iniciais do trabalho e se foram cumpridos. Na secção 6.2 são enumeradas algumas implementações futuras que se poderiam efectuar neste projecto.

6.1. Trabalho Final

Este Trabalho Final de Mestrado propôs uma abordagem para controlar a posição de um robô cartesiano utilizando um sistema de visão computacional. A abordagem visa tarefas em que um número de objectos é exibido num plano horizontal dentro do espaço de trabalho do robô, e cada objecto deve ser processado pelo robô. Ao adoptar algumas suposições, sobre o posicionamento dos alvos e a orientação da câmara, foi desenvolvido um sistema de controlo simples.

A simulação e os resultados experimentais validaram a abordagem em relação à lógica de controlo e aos anéis de controlo baseados em imagem. Devido à sua simplicidade, este tipo de sistema de controlo pode ser implementado sem muito esforço em aplicações industriais onde o posicionamento automático e preciso é necessário. Além disso, a introdução de um sistema de visão para controlar uma operação industrial tem a vantagem de que o mesmo sistema também possa ser usado para realizar tarefas de controlo de qualidade e segurança, reduzindo assim os riscos para os operadores.

Tendo em conta os objectivos inicialmente definidos para este Trabalho Final de Mestrado, pode-se dizer que o primeiro foi cumprido, uma vez que se conseguiu programar um algoritmo com base em visão computacional para fazer actuar os motores. A actuação é dependente da posição dos círculos detectados em relação ao referencial da imagem (câmara).

O segundo objectivo também foi cumprido. Foi construído um protótipo aproximadamente à escala (1:10), que simula o funcionamento do sistema real de posicionamento. Este protótipo foi construído com material da *V-slot Europe*, nomeadamente os perfis de automação, os motores e a placa de controlo dos mesmos.

O terceiro objectivo também se cumpriu. A partir de um *browser*, como por exemplo, o *Google Chrome*, é possível supervisionar o processo em tempo real, nomeadamente a imagem da câmara e o estado dos motores. Também se podem enviar instruções para começar ou parar o processo. O *Matlab* comunica com a página de *Internet* através de um servidor grátis criado para o efeito (*Apache*).

As principais dificuldades na realização deste Trabalho Final de Mestrado reflectiram-se na programação do algoritmo de visão, de modo a que se cumprissem os objectivos propostos, e também na implementação da página de *Internet* que conseguisse comunicar com o *Matlab*, servindo de interface de utilização e apresentação de dados.

Este projecto é um sistema de posicionamento baseado em imagem (IBVS), porém, não contém a vertente de calibração.

6.2. Implementações Futuras

O projecto que se efectuou no âmbito deste Trabalho Final de Mestrado contém ainda muitas limitações e melhoramentos que se podem desenvolver em trabalhos futuros. Como implementações futuras, sugerem-se as seguintes:

- Implementação de um sistema de calibração que consiga estabelecer uma correspondência precisa entre pixéis e distância real, de modo a ser possível a criação de um referencial global do sistema, ao invés de apenas um referencial local da câmara.
- Análise de imagem de modo a detectar o fim do camião-cisterna. Assim, em vez de se necessitar de introduzir o número de bocas que o camião possui de modo a acabar o processo de lavagem, este pode finalizar com uma instrução enviada automaticamente quando a câmara detecta o fim do camião.
- Pode ser criada uma base de dados para guardar todas as informações possíveis de obter sobre o processo, de modo a ser mais fácil resolver avarias que aconteçam ao longo do tempo, com base na informação recolhida.
- É necessário um servidor dedicado se se pretender que vários utilizadores consigam supervisionar e/ou controlar o processo.

- Implementação de vários tipos de contas com diferentes permissões, como por exemplo, operário, director, administrador, de modo a obter-se uma maior segurança no processo.
- Instalação de uma Firewall no servidor, de modo a diminuir a probabilidade de sucesso de ataques informáticos.
- Implementação de um botão de emergência no modo Automático.
- Desenvolver uma função que faça com que o robô retorne à sua posição inicial após o processo estar completo.

Referências Bibliográficas

[1] F. Civerchia, S. Bocchino, C. Salvadori, E. Rossi, L. Maggiani, and M. Petracca, "Industrial Internet of Things monitoring solution for advanced predictive maintenance applications," *J. Ind. Inf. Integr.*, vol. 7, pp. 4–12, 2017.

[2] G. Biegelbauer and M. Vincze, "3D vision-guided bore inspection system," *Proc. Fourth IEEE Int. Conf. Comput. Vis. Syst. ICVS'06*, vol. 2006, no. Icvs, p. 22, 2006.

[3] M. Albert, "Seven Things to Know about the Internet of Things and Industry 4.0," *Mod. Mach. Shop*, vol. 88, no. 4, pp. 74–81, 2015.

[4] Carmona, "Carmona," 2018. [Online]. Available: http://www.carmona.pt/carmona/home. [Accessed: 06-Jun-2018].

[5] Library.AutomationDirect, "Tips for a Better HMI Layout." [Online]. Available: https://library.automationdirect.com/tips-better-hmi-layout/. [Accessed: 29-Nov-2018].

[6] B. J. Payne, "A UTOMATION S OLUTIONS in controllers , HMIs," no. April, 2016.

[7] LANGHAMMER, "Gantry Robots PRO03, PRO04," 2018. [Online]. Available: https://www.langhammer.de/en/products/gantry-robots-pro03pro04.html. [Accessed: 05-Aug-2018].

[8] Nuno Beites, Manuel Dias, Mário J.G.C. Mendes, F. Carreira, F. Campos, J.M.F. Calado, "A Gantry Robot Automatic Positioning System using Computational Vision". In Proceedings of the 1st Iberic Conference on Theoretical and Experimental Mechanics and Materials/11th National Congress on Experimental Mechanics, Porto/Portugal 4-7 November 2018. Ed. J.F. Silva Gomes. INEGI/FEUP (2018); ISBN: 978-989-20-8771-9; pp. 1031-1042.

[9] B. Zhang, J. Wang, G. Rossano, C. Martinez, and S. Kock, "Vision-guided robot alignment for scalable, flexible assembly automation," *2011 IEEE Int. Conf. Robot. Biomimetics, ROBIO 2011*, pp. 944–951, 2011.

[10] F. Chaumette and S. Hutchinson, "Visual servo control. I. Basic approaches," *IEEE Robot. Autom. Mag.*, vol. 13, no. 4, pp. 82–90, 2006.

[11] Q. Chen, S. Zhu, X. Wang, and W. Wu, “Analysis on an uncalibrated image-based visual servoing for 6 DOF industrial welding robots,” *2012 IEEE Int. Conf. Mechatronics Autom. ICMA 2012*, no. 1, pp. 2013–2018, 2012.

[12] T. Shen, J. Yang, Y. Cai, D. Li, and G. Chesi, “Visual servoing with cylinders: Reaching the desired location following a straight line,” *Chinese Control Conf. CCC*, no. 1, pp. 11183–11188, 2017.

[13] M. Gridseth, K. Hertkorn, and M. Jagersand, “On Visual Servoing to Improve Performance of Robotic Grasping,” *Proc. -2015 12th Conf. Comput. Robot Vision, CRV 2015*, pp. 245–252, 2015.

[14] G. Palmieri, M. Palpacelli, M. Battistelli, and M. Callegari, “A comparison between position-based and image-based dynamic visual servoings in the control of a translating parallel manipulator,” *J. Robot.*, vol. 2012, 2012.

[15] S. Huang, Y. Yamakawa, T. Senoo, and M. Ishikawa, “A direct visual servo scheme based on simplified interaction matrix for high-speed manipulation,” *2012 IEEE Int. Conf. Robot. Biomimetics, ROBIO 2012 - Conf. Dig.*, pp. 1950–1955, 2012.

[16] G. J. Garcia, C. A. Jara, J. Pomares, and F. Torres, “Direct visual servo control of a robot to track trajectories in supervision tasks,” *11th Int. Conf. Control. Autom. Robot. Vision, ICARCV 2010*, no. December, pp. 1434–1439, 2010.

[17] C. Cai, E. Dean-le, N. Somani, and A. Knoll, “3D Image-based Dynamic Visual Servoing with uncalibrated Stereo Cameras.”

[18] M. T. Hussein, “SURVEY PAPER A review on vision-based control of flexible manipulators,” *Adv. Robot.*, vol. 29, no. 24, pp. 1575–1585, 2015.

[19] A. Santamaria-Navarro and J. Andrade-Cetto, “Uncalibrated image-based visual servoing,” *2013 IEEE Int. Conf. Robot. Autom.*, pp. 5247–5252, 2013.

[20] B. Huang, M. Ye, S.-L. Lee, and G.-Z. Yang, “A Vision-Guided Multi-Robot Cooperation Framework for Learning-by-Demonstration and Task Reproduction,” pp. 4797–4804, 2017.

[21] Y. Xu, X. Tong, Y. Mao, W. B. Griffin, B. Kannan, and L. A. Derose, “A vision-guided robot manipulator for surgical instrument singulation in a cluttered environment,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 3517–3523, 2014.

[22] H. Cheng, H. Chen, and Y. Liu, “Object handling using autonomous industrial mobile manipulator,” *2013 IEEE Int. Conf. Cyber Technol. Autom. Control Intell. Syst. IEEE-CYBER 2013*, pp. 36–41, 2013.

[23] J. Salvi, X. Armangué, and J. Batlle, “A comparative review of camera calibrating methods with accuracy evaluation,” *Pattern Recognit.*, vol. 35, no. 7, pp. 1617–1635, 2002.

[24] B. Cai, Y. Wang, K. Wang, M. Ma, and X. Chen, “Camera calibration robust to defocus using phase-shifting patterns,” *Sensors (Switzerland)*, vol. 17, no. 10, 2017.

[25] M. Wilczkowiak, P. Sturm, and E. Boyer, “Using geometric constraints through parallelepipeds for calibration and 3D modeling,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 2, pp. 194–207, 2005.

[26] Z. Zhang, “A Flexible New Technique for Camera Calibration (Technical Report),” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 11, pp. 1330–1334, 2002.

[27] O. Objects, Z. Zhang, and S. Member, “Camera Calibration with,” vol. 26, no. 7, pp. 892–899, 2004.

[28] F. C. Wu, Z. Y. Hu, and H. J. Zhu, “Camera calibration with moving one-dimensional objects,” *Pattern Recognit.*, vol. 38, no. 5, pp. 755–765, 2005.

[29] E. L. Hall, J. B. K. Tio, C. A. McPherson, and F. A. Sadjadi, “Measuring Curved Surfaces for Robot Vision,” *Computer (Long. Beach. Calif).*, vol. 15, no. 12, pp. 42–54, 1982.

[30] R. Y. Tsai, “A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses,” *IEEE J. Robot. Autom.*, vol. 3, no. 4, pp. 323–344, 1987.

[31] J. Salvi and M. A. S. Isbn, *AN APPROACH TO CODED STRUCTURED LIGHT TO OBTAIN THREE PhD . Thesis UNIVERSITY OF GIRONA European PhD in Industrial Engineering Joaquim Salvi*. 2001.

[32] G. Xin, J. Qingxuan, S. Hanxu, C. Gang, Z. Qianru, and Y. Yukun, “Real-time dynamic system to path tracking and collision avoidance for redundant robotic arms,” *J. China Univ. Posts Telecommun.*, vol. 23, no. 1, pp. 73–85, 2016.

[33] J. P. Alepuz, M. R. Emami, and J. Pomares, “Direct image-based visual servoing of

free-floating space manipulators," *Aerosp. Sci. Technol.*, vol. 55, pp. 1–9, 2016.

[34] M. Palpacelli, G. Palmieri, and L. Carbonari, "Analysis of an Experimental Setup for Direct Visual Servoing," 2017.

[35] P. Ferrara *et al.*, "Wide-angle and long-range real time pose estimation: A comparison between monocular and stereo vision systems," *J. Vis. Commun. Image Represent.*, vol. 48, pp. 159–168, 2017.

[36] W. Shi, L. Zhang, Y. Yao, J. Zuo, and X. Yao, "Linear calibration for robot vision," *Proc. - 2016 8th Int. Conf. Intell. Human-Machine Syst. Cybern. IHMSC 2016*, vol. 2, pp. 431–435, 2016.

[37] J. C. Rodríguez-Quiñonez *et al.*, "Improve a 3D distance measurement accuracy in stereo vision systems using optimization methods' approach," *Opto-electronics Rev.*, vol. 25, no. 1, pp. 24–32, 2017.

[38] D. Scharstein, S. Richard, and Z. Ramin, "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms," 2001.

[39] P. Corke, *Robotics, Vision and Control*, vol. 73. 2011.

[40] J. W. Park, R. Todd, and X. Song, "Geometric Pattern Match Using Edge Driven Dissected Rectangles and Vector Space," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 35, no. 12, pp. 2046–2055, 2016.

[41] T. Ganokratanaa and S. Pumrin, "The vision-based hand gesture recognition using blob analysis," *2nd Jt. Int. Conf. Digit. Arts, Media Technol. 2017 Digit. Econ. Sustain. Growth, ICDAMT 2017*, pp. 336–341, 2017.

[42] A. Chantakamo and M. Ketcham, "The multi vehicle recognition using hybrid blob analysis and feature-based," *Proc. - 2015 7th Int. Conf. Inf. Technol. Electr. Eng. Envisioning Trend Comput. Inf. Eng. ICITEE 2015*, pp. 453–457, 2015.

[43] H. Ye, G. Shang, L. Wang, and M. Zheng, "A new method based on hough transform for quick line and circle detection," *Proc. - 2015 8th Int. Conf. Biomed. Eng. Informatics, BMEI 2015*, no. Bmei, pp. 52–56, 2016.

[44] N. H. Lestriandoko and R. Sadikin, "Circle detection based on hough transform and Mexican Hat filter," *Proceeding - 2016 Int. Conf. Comput. Control. Informatics its Appl. Recent Prog. Comput. Control. Informatics Data Sci. IC3INA 2016*, no. 1, pp.

153–157, 2017.

[45] Z. Yao and W. Yi, “Curvature aided Hough transform for circle detection,” *Expert Syst. Appl.*, vol. 51, pp. 26–33, 2016.

[46] L. Carlos, L. Michel, M. Reynoso, and E. Nu, “Real-Time Image Template Matching Algorithm based on Differential Evolution,” *2016 IEEE-RAS Int. Conf. Humanoid Robot. (Humanoids 2016)*, 2016.

[47] P. R. Patil, M. D. Patil, and V. A. Vyawahare, “Acceleration of hough transform algorithm using Graphics Processing Unit (GPU),” *Int. Conf. Commun. Signal Process. ICCSP 2016*, pp. 1584–1588, 2016.

[48] G. O. Online and C. This, “IoT gateways: Industrial automation’s path to Industrie 4.0,” no. March, pp. 36–38, 2016.

[49] H. Sasajima, T. Ishikuma, and H. Hayashi, “Future IIOT in process automation-Latest trends of standardization in industrial automation, IEC/TC65,” *2015 54th Annu. Conf. Soc. Instrum. Control Eng. Japan, SICE 2015*, pp. 963–967, 2015.

[50] A. Sajid, H. Abbas, and K. Saleem, “Cloud-Assisted IoT-Based SCADA Systems Security: A Review of the State of the Art and Future Challenges,” *IEEE Access*, vol. 4, no. c, pp. 1375–1384, 2016.

[51] A. Young, A. G. Separators, and B. M. Risse, “IIoT arrives :,” no. February, 2017.

[52] A. Rezai, P. Keshavarzi, and Z. Moravej, “Key management issue in SCADA networks: A review,” *Eng. Sci. Technol. an Int. J.*, vol. 20, no. 1, pp. 354–363, 2017.

[53] Bosch, “Rexroth,” 2018. [Online]. Available: https://www.boschrexroth.com/en/xc/. [Accessed: 25-Jul-2018].

[54] D. Beer, F; Russell Johnston, E; DeWolf, J; Mazurek, *Mechanics of Materials*. Mc Graw Hill, 2015.

[55] OMRON, “OMRON Automação Industrial,” 2018. [Online]. Available: https://industrial.omron.pt/pt/products. [Accessed: 20-Oct-2018].

[56] J. van der Geest, “Getkey.” [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/7465-getkey. [Accessed: 06-Jun-2018].

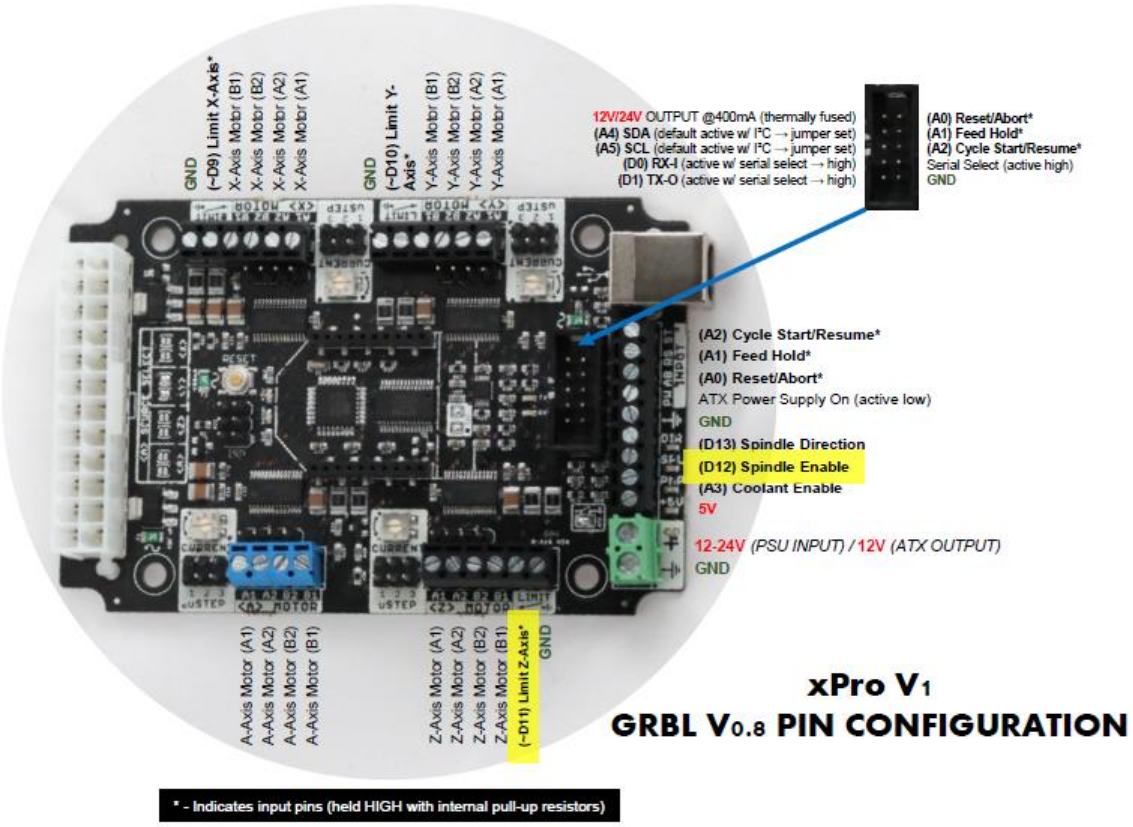
[57] M. Dias, “Indústria 4.0 Aplicada à Lavagem Robotizada de Camiões Cisterna para Transporte de Produtos Tóxicos,” Lisboa, IPL/2017/ROBOTCLEAN/ISEL, 2018.

[58] V. Europe, “Vslot Europe - By Open Technologies,” 2018. [Online]. Available: http://vsloteurope.com. [Accessed: 25-Jul-2018].

Anexos

Anexo A

Configuração de Pinos do Controlador



Anexo B

Algoritmo de Visão (Script Principal) (Switch-case)

```
clear all
imaqreset
clc
ctr=Controller();
try
    Mode=input('Modo Manual - 0 ; Modo automáticoco - 1 :  ');
    camera=initializeCamera();
    resolution=camera.VideoResolution;
    state=State.Idle;
    count=0;
    numHoles=input('Introduza Nr Furos :  ');
if Mode==0
    manual(camera,ctr);
else
    t=createTimer(ctr);
    server=Server(8081);
    start(t);
    while count<numHoles
        pause(0.01);
        if ~isempty(server.Connections)
            [c,img]=getCircles(camera);
            sendData(server,t.UserData,img,state);
            if server.isOnline()==1
                nearestHole=findNearestHole(c);
                switch state
                    case State.Calibration
                        state=calibration(resolution,ctr);
                    case State.Idle
                        state=idle(resolution,nearestHole,ctr,t);
                    case State.Approach

state=approach(resolution,nearestHole,ctr,t);
                    case State.Clean
                        pause(0.5);
                        sendData(server,t.UserData,img,state);
                        clean(ctr);
                        state=State.Idle;
                        count=count+1;
                        fprintf('Cleaned holes: %d\n',count);
                        pause(2);
                end
            else
                ctr.stop();
            end
        else
            ctr.stop();
        end
    end
    x=t.UserData;
    save('dados.mat','x');
    stop(t);
    pause(1);
    delete(server);
end
```

```
    stop(camera);
    imaqreset;
    ctr.disconnect();
catch
    stop(t);
    stop(camera);
    imaqreset;
    ctr.stop();
    ctr.disconnect();
    delete(server);
end
```

Função Approach

```
function [next] = approach(resolution,hole,ctr,t)
    next=State.Approach;
    y=0;
    x=0;
    pixel_interval=1;
    if hole(2)<(resolution(2)/2)+pixel_interval &&
hole(2)>(resolution(2)/2)-pixel_interval
        ctr.stopY();
        y=1;
        t.UserData.dirY=0;
    elseif hole(2)>=(resolution(2)/2)+pixel_interval
        t.UserData.dirY=-1;
        ctr.moveBackwardApproach();
    elseif hole(2)<=(resolution(2)/2)-pixel_interval
        t.UserData.dirY=1;
        ctr.moveForwardApproach();
    end

    if hole(1)<(resolution(1)/2)+pixel_interval &&
hole(1)>(resolution(1)/2)-pixel_interval
        ctr.stopX();
        x=1;
        t.UserData.dirX=0;

    elseif hole(1)>=(resolution(1)/2)+pixel_interval
        if y<=0
            t.UserData.dirX=-1;
            ctr.moveSlowRight();
        else
            t.UserData.dirX=-1;
            ctr.moveRight();
        end
    else
        if y<=0
            t.UserData.dirX=1;
            ctr.moveSlowLeft();
        else
            t.UserData.dirX=1;
            ctr.moveLeft();
        end
    end

    if x>0 && y>0
```

```
        next=State.Clean;
    end
end
```

Função Idle

```
function [next] = idle(resolution,hole,ctr,t)
    next=State.Idle;
    ctr.moveForwardRoutine();
    t.UserData.dirY=1;
    if ~isempty(hole)
        if hole(2)<(resolution(2)/2)-2 &&
hole(2)>(resolution(2)/3)
            ctr.stopY();
            next=State.Approach;
        end
    end
end
```

Função Clean

```
function clean(ctr)
    pause(1);
    ctr.moveDown();
    pause(5);
    ctr.moveUp();
    pause(5);
end
```

Função initializeCamera

```
function [vid] = initializeCamera()
    vid=videoinput('winvideo',2,'YUY2_320x240');
    src = getselectedsource(vid);
    vid.FramesPerTrigger = Inf;
    src.Brightness = -15;
    src.Contrast = 35;
    %src.FrameRate= 15;
    start(vid);
end
```

Função getCircles

```
function [c,img] = getCircles(cam)
    img=getsnapshot(cam);
    img = ycbcr2rgb(img);
    %imshow(img);
    [c,r,m] = imfindcircles(img,[30,80],'ObjectPolarity','dark');
End
```

Função getAvailableComPort

```
function lCOM_Port = getAvailableComPort()
% function lCOM_Port = getAvailableComPort()
% Return a Cell Array of COM port names available on your computer

try
    s=serial('IMPOSSIBLE_NAME_ON_PORT');fopen(s);
catch
    lErrMsg = lasterr;
end

%Start of the COM available port
lIndex1 = findstr(lErrMsg,'COM');
%End of COM available port
lIndex2 = findstr(lErrMsg,'Use')-3;

lComStr = lErrMsg(lIndex1:lIndex2);

%Parse the resulting string
lIndexDot = findstr(lComStr,',');

% If no Port are available
if isempty(lIndex1)
    lCOM_Port{1}='';
    return;
end

% If only one Port is available
if isempty(lIndexDot)
    lCOM_Port{1}=lComStr;
    return;
end

lCOM_Port{1} = lComStr(1:lIndexDot(1)-1);

for i=1:numel(lIndexDot)+1
    % First One
    if (i==1)
        lCOM_Port{1,1} = lComStr(1:lIndexDot(i)-1);
    % Last One
    elseif (i==numel(lIndexDot)+1)
        lCOM_Port{i,1} = lComStr(lIndexDot(i-1)+2:end);
    % Others
    else
        lCOM_Port{i,1} = lComStr(lIndexDot(i-1)+2:lIndexDot(i)-1);
    end
end
```

Classe Controller

```
classdef Controller

    properties (Access=private)
        s %communicate via serial with controller
        xSpeed double
    end
```

```
    methods
        function obj = Controller()

            serialPorts=seriallist; %receive serial port list
            while(isempty(serialPorts))
                disp("No ports available.");
                serialPorts=seriallist;
            end
            obj.s=serial(serialPorts(1));
            set(obj.s,'BaudRate',2000000); %setting transmition
speed. %Tem que ser a mesma do controlador
            fopen(obj.s);
            [out,count,msg]=fgetl(obj.s);
        end

        function startTimer(obj)

        end

              function out=getState(obj,axis)
                  fprintf(obj.s,'%s',strcat('<m',axis,'>'));
                  [out,count,msg]=fgetl(obj.s);
                  out=out(1);
              end

              function out=getSpeed(obj,axis)
                  fprintf(obj.s,'%s',strcat('<v',axis,'>'));
                  [out,count,msg]=fgetl(obj.s);

              end

              function moveForwardRoutine(obj)
                  if(obj.getState('y')=='0')
                      fprintf(obj.s,'%s','<sy15>');
                      fprintf(obj.s,'%s','<y700>');
                  end
              end

              function moveForwardApproach(obj)
                  if(obj.getState('y')=='0')
                      fprintf(obj.s,'%s','<sy5>');
                      fprintf(obj.s,'%s','<ry100>');
                  end
              end

              function moveBackwardApproach(obj)
                  if(obj.getState('y')=='0')
                      fprintf(obj.s,'%s','<sy5>');
                      fprintf(obj.s,'%s','<ry-100>');
                  end
              end

              function moveDown(obj)
                  obj.stopZ();
                  fprintf(obj.s,'%s','<sz10>');
                  fprintf(obj.s,'%s','<rz-200>');
              end

              function moveUp(obj)
```

```
    obj.stopZ();
    fprintf(obj.s,'%s','<sz10>');
    fprintf(obj.s,'%s','<rz200>');
end

function moveSlowLeft(obj)
    obj.xSpeed=9;
    if(obj.getState('x')=='0')
        fprintf(obj.s,'%s','<sx37.5>');
        fprintf(obj.s,'%s','<500>');
    end
end

function moveSlowRight(obj)
    obj.xSpeed=9;
    if(obj.getState('x')=='0')
        fprintf(obj.s,'%s','<sx37.5>');
        fprintf(obj.s,'%s','<rx-500>');
    end
end

function moveLeft(obj)
    if obj.xSpeed==9
        stopX(obj);
    end
    obj.xSpeed=104;
    if(obj.getState('x')=='0')
        fprintf(obj.s,'%s','<sx60>');
        fprintf(obj.s,'%s','<rx1000>');
    end
end

function moveRight(obj)
    if obj.xSpeed==9
        stopX(obj);
    end
    obj.xSpeed=104;
    if(obj.getState('x')=='0')
        fprintf(obj.s,'%s','<sx60>');
        fprintf(obj.s,'%s','<rx-1000>');
    end
end

function home(obj)
end

function stop(obj)
    fprintf(obj.s,'%s','<S>');
end

function stopX(obj)
    fprintf(obj.s,'%s','<Sx>');
end

function stopY(obj)
    fprintf(obj.s,'%s','<Sy>');
end

function stopZ(obj)
    fprintf(obj.s,'%s','<Sz>');
```

```
            end

            function disconnect(obj)
                fclose(obj.s);
            end

            function obj=set.xSpeed(obj,value)
                obj.xSpeed=value;
            end
        end
    end
```

Classe State

```
classdef State

    enumeration
        Calibration,Idle,Approach,Clean
    end
end
```

Função "Manual"

```
function manual(cam,ctr);

img=getsnapshot(cam);
f = figure('Name','Visão,'NumberTitle','Off');
imshow(img);

char=0;

while char ~='x';

char1=getkey(1,'non-ascii');
char=num2str(char1);

switch char

        case 'w'
        ctr.moveForwardApproach();
        pause(1);
        ctr.stop();

        case  's'
        ctr.moveBackwardApproach();
        pause(1);
        ctr.stop();

        case 'a'
        ctr.moveLeft();
        pause(1);
        ctr.stop();

        case 'd'
        ctr.moveRight();
        pause(1);
```

```
        ctr.stop();

        case 'q'
        ctr.moveUp();
        pause(1);
        ctr.stop();

        case 'e'
        ctr.moveDown();
        pause(1);
        ctr.stop();

end
end
ctr.stop();
end
```

Função sendData

```
function sendData(server,data,img,state)
    imwrite(img,'temp.jpeg');
    c=base64file('temp.jpeg');
    c=strcat('img:',c);
    server.sendToAll(c);
    c=strcat('Button:',server.message);
    server.sendToAll(c);
    c=strcat('X_Distance:',num2str(data.dx(end)));
    server.sendToAll(c);
    c=strcat('Y_Distance:',num2str(data.dy(end)));
    server.sendToAll(c);
    c=strcat('X_Speed:',num2str(data.speedX(end)));
    server.sendToAll(c);
    c=strcat('Y_Speed:',num2str(data.speedY(end)));
    server.sendToAll(c);
    c=strcat('State:',char(state));
    server.sendToAll(c);
end
```

Função createTimer

```
function t = createTimer(ctr)
    t=timer;
    set(t,'executionMode','fixedRate');
    set(t,'StartFcn','tic');
    t.TimerFcn={@countDistance,ctr};

t.UserData=struct('t',[0],'stateX',[0],'stateY',[0],'dx',[0],'dy',
[0],'dirX',[0],'dirY',[0],'speedX',[0],'speedY',[0]);
    t.Period=0.1;
end
```

Função countDistance

```
function countDistance(obj,event,ctr)
    obj.UserData.t=[obj.UserData.t toc];
    state=ctr.getState('x');
```

```
    obj.UserData.stateX=[obj.UserData.stateX state];
    if state=='1'
        obj.UserData.speedX=[obj.UserData.speedX
str2double(ctr.getSpeed('x'))];
        obj.UserData.dx=[obj.UserData.dx
obj.UserData.dx(end)+obj.UserData.dirX*(obj.UserData.speedX(end))*
(obj.UserData.t(end)-obj.UserData.t(end-1))];
    elseif state=='2'
        obj.UserData.speedX=[obj.UserData.speedX
str2double(ctr.getSpeed('x'))];
        obj.UserData.dx=[obj.UserData.dx
obj.UserData.dx(end)+obj.UserData.dirX*(obj.UserData.speedX(end))*
(obj.UserData.t(end)-obj.UserData.t(end-1))];
    elseif state=='3'
        obj.UserData.speedX=[obj.UserData.speedX
str2double(ctr.getSpeed('x'))];
        obj.UserData.dx=[obj.UserData.dx
obj.UserData.dx(end)+obj.UserData.dirX*(obj.UserData.speedX(end))*
(obj.UserData.t(end)-obj.UserData.t(end-1))];
    else
        obj.UserData.speedX=[obj.UserData.speedX 0];
        obj.UserData.dirX=0;
        obj.UserData.dx=[obj.UserData.dx obj.UserData.dx(end)];
    end
    state=ctr.getState('y');
    obj.UserData.stateY=[obj.UserData.stateY state];
    if state=='1'
        obj.UserData.speedY=[obj.UserData.speedY
str2double(ctr.getSpeed('y'))/8];
        obj.UserData.dy=[obj.UserData.dy
obj.UserData.dy(end)+obj.UserData.dirY*(obj.UserData.speedY(end))*
(obj.UserData.t(end)-obj.UserData.t(end-1))];
    elseif state=='2'
        obj.UserData.speedY=[obj.UserData.speedY
str2double(ctr.getSpeed('y'))/8];
        obj.UserData.dy=[obj.UserData.dy
obj.UserData.dy(end)+obj.UserData.dirY*(obj.UserData.speedY(end))*
(obj.UserData.t(end)-obj.UserData.t(end-1))];
    elseif state=='3'
        obj.UserData.speedY=[obj.UserData.speedY
str2double(ctr.getSpeed('y'))/8];
        obj.UserData.dy=[obj.UserData.dy
obj.UserData.dy(end)+obj.UserData.dirY*(obj.UserData.speedY(end))*
(obj.UserData.t(end)-obj.UserData.t(end-1))];
    else
        obj.UserData.speedY=[obj.UserData.speedY 0];
        obj.UserData.dirY=0;
        obj.UserData.dy=[obj.UserData.dy obj.UserData.dy(end)];
    end
end
```

Função calculateDistance

```
function calculateDistance(obj,event)
    obj.UserData.dx=zeros(size(t));
    obj.UserData.dy=zeros(size(t));
end
```

Classe Server

```
classdef Server < WebSocketServer
    %ECHOSERVER Summary of this class goes here
    %   Detailed explanation goes here

    properties (Access=public)
        message char
    end

    methods
        function obj = Server(varargin)
            %Constructor
            obj@WebSocketServer(varargin{:});
            obj.message='Start';
            disp('Server is online');
        end
    end

    methods(Access =public)
        function ans=isEmpty(obj)
            ans=length(obj.message);
        end

        function emptyMessages(obj)
            obj.message='';
        end

        function ret=isOnline(obj)
            if strcmp(obj.message,'Start')
                ret=0;
            elseif strcmp(obj.message,'Stop')
                ret=1;
            else
                ret=0;
            end
        end
    end

    methods (Access = protected)
        function onOpen(obj,conn,message)
            fprintf('%s\n',message)
        end

        function onTextMessage(obj,conn,message)
            %fprintf('Received from %d, the message:
%s\n',conn.HashCode,message);
            if strcmp(message,'Start')
                obj.message='Stop';
            elseif strcmp(message,'Stop')
                obj.message='Start';
            end
        end

        function onBinaryMessage(obj,conn,bytearray)
            % This function sends an echo back to the client
            conn.send(bytearray); % Echo
        end
```

```
        function onError(obj,conn,message)
            fprintf('%s\n',message)
        end

        function onClose(obj,conn,message)
            fprintf('%s\n',message)
        end


    end
end
```

Código Implementado no Controlador (*Arduino*) (Linguagem C)

```
#include <Arduino.h>
#include "DRV8825.h"

#define MOTOR_STEPS 200

#define DEFAULTRPM 60


#define XDIR 5
#define XSTEP 2
#define YDIR 6
#define YSTEP 3
#define ZDIR 7
#define ZSTEP 4
#define ENABLE 8
#define XLIM 9
#define YLIM 10
#define ZLIM 12

#define MODE0 10
#define MODE1 11
#define MODE2 12
DRV8825 stepperX(MOTOR_STEPS, XDIR, XSTEP, ENABLE);
DRV8825 stepperY(MOTOR_STEPS, YDIR, YSTEP, ENABLE);
DRV8825 stepperZ(MOTOR_STEPS, ZDIR, ZSTEP, ENABLE);
const byte numChars = 10;
char input[numChars];

struct Command {
  char coordinate;
  int rotation;
};

void setup() {
  setupSerial();
  setupDrivers();
}

void setupSerial() {
  Serial.begin(2000000);
  Serial.println("<Controller is ready>");
}
```

```
void setupDrivers() {
  pinMode(XLIM, INPUT_PULLUP);
  pinMode(YLIM, INPUT_PULLUP);
  pinMode(ZLIM, INPUT_PULLUP);
  stepperX.begin(DEFAULTRPM, 16);
  stepperY.begin(DEFAULTRPM, 16);
  stepperZ.begin(30, 32);
  stepperX.enable();
  stepperX.setSpeedProfile(DRV8825::LINEAR_SPEED, 200, 200);
  stepperY.setSpeedProfile(DRV8825::LINEAR_SPEED, 200, 200);
  stepperZ.setSpeedProfile(DRV8825::LINEAR_SPEED, 200, 200);
}

void loop() {
  checkLimits();
  stepperAction();
  if (Serial.available() > 0) {
    receiveCommand();
    String in = String(input);
    if (in.charAt(0) == 'r')
      moveRotation(in.substring(1));
    else if (in.charAt(0) == 'm')
      getState(in.substring(1));
    else if (in.charAt(0) == 'v')
      getSpeed(in.substring(1));
    else if (in.charAt(0) == 's')
      updateSpeed(in.substring(1));
    else if (in.charAt(0) == 'h')
      goHome();
    else if (in.charAt(0) == 'S') {
      stopFunction(in.substring(1));
    }
    else
      moveDist(in);
  }
}

void stepperAction() {
  if (stepperX.nextAction() + stepperY.nextAction() +
stepperZ.nextAction() <= 0)
    stepperX.enable();
}

void receiveCommand() {
  static boolean recvInProgress = false;
  static byte ndx = 0;
  char startMarker = '<';
  char endMarker = '>';
  char rc;
  boolean newData = false;
  while (Serial.available() > 0 && newData == false) {
    rc = Serial.read();
    if (recvInProgress == true) {
      if (rc != endMarker) {
        input[ndx] = rc;
        ndx++;
        if (ndx >= numChars) {
          ndx = numChars - 1;
        }
```

```
      }
      else {
        input[ndx] = '\0'; // terminate the string
        recvInProgress = false;
        ndx = 0;
        newData = true;
      }
    }

    else if (rc == startMarker) {
      recvInProgress = true;
    }
  }
}

void getState(String s) {
  int state = 0;
  if (s.charAt(0) == 'x')
    state = stepperX.getCurrentState();
  else if (s.charAt(0) == 'y')
    state = stepperY.getCurrentState();
  else if (s.charAt(0) == 'z')
    state = stepperZ.getCurrentState();
  Serial.println(state);
}

void getSpeed(String s) {
  short rpm = 0;
  if (s.charAt(0) == 'x')
    rpm = stepperX.getCurrentRPM();
  else if (s.charAt(0) == 'y')
    rpm = stepperY.getCurrentRPM()+1;
  else if (s.charAt(0) == 'z')
    rpm = stepperZ.getCurrentRPM();
  Serial.println(rpm);
}

void stopFunction(String s) {
  if (s.charAt(0) == 'x')
    stepperX.startBrake();
  else if (s.charAt(0) == 'y')
    stepperY.startBrake();
  else if (s.charAt(0) == 'z')
    stepperZ.startBrake();
  else {
    stepperX.startBrake();
    stepperY.startBrake();
    stepperZ.startBrake();
  }
  stepperX.nextAction();
  stepperY.nextAction();
  stepperZ.nextAction();
}

void goHome() {
  moveStepper(&stepperY,-15000); //move backwards until the button
is pressed;
  while(digitalRead(YLIM) == HIGH)stepperAction();
  stepperY.stop(); //when the button is pressed the motor stops
```

```
  changeSpeed(&stepperY,4); // adjusts slowly and accuratly until
the button is no longer pressed
  moveStepper(&stepperY,360);
  while(digitalRead(YLIM) == LOW)stepperAction();
  stepperY.stop();
  changeSpeed(&stepperY,DEFAULTRPM);
}

void checkLimits() {
  if (digitalRead(XLIM) == LOW) {
    stepperX.stop();
  }
  if (digitalRead(YLIM) == LOW) {
    stepperY.stop();
  }
  if (digitalRead(ZLIM) == LOW) {
    stepperZ.stop();
  }
}

void updateSpeed(String s) {
  if (s.charAt(0) == 'x')
    changeSpeed(&stepperX, s.substring(1).toInt());
  else if (s.charAt(0) == 'y')
    changeSpeed(&stepperY, s.substring(1).toInt());
  else if (s.charAt(0) == 'z')
    changeSpeed(&stepperZ, s.substring(1).toInt());
  else {
    int aux = s.substring(0).toInt();
    stepperX.setRPM(aux);
    stepperY.setRPM(aux);
    stepperZ.setRPM(aux);
  }
}

void changeSpeed(DRV8825* stepper, int vel) {
  stepper->setRPM(vel);
  short accel = vel * 200 / 30;
  stepper->setSpeedProfile(DRV8825::LINEAR_SPEED, accel, accel);
}

void moveRotation(String in) {
  Command cmd[3] = {{' ', -1}, {' ', -1}, {' ', -1}};
  getCommands(in, cmd);
  for (byte i = 0; i < 3 && cmd[i].coordinate != ' '; i++) {
    if (cmd[i].coordinate == 'x')
      moveStepper(&stepperX, cmd[i].rotation);
    else if (cmd[i].coordinate == 'y')
      moveStepper(&stepperY, cmd[i].rotation);
    else if (cmd[i].coordinate == 'z')
      moveStepper(&stepperZ, cmd[i].rotation);
  }
}

void moveStepper(DRV8825* stepper, int rotation) {
  Serial.println(rotation);
  stepper->disable();
  stepper->startRotate(rotation);
}
```

```
void moveDist(String s) {
  Command cmd[3] = {{' ', -1}, {' ', -1}, {' ', -1}};
  getCommands(s, cmd);
  for (byte i = 0; i < 3 && cmd[i].coordinate != ' '; i++) {
    if (cmd[i].coordinate == 'x')
      moveStepper(&stepperX,
convertDistToAngleX(cmd[i].rotation));
    else if (cmd[i].coordinate == 'y')
      moveStepper(&stepperY, convertDistToAngle(cmd[i].rotation));
    else if (cmd[i].coordinate == 'z')
      moveStepper(&stepperZ, convertDistToAngle(cmd[i].rotation));
  }
}

int convertDistToAngle(int dist) {
  return (dist * 360) / 60;
}

int convertDistToAngleX(int dist) {
  return (dist * 360) / 8;
}

void getCommands(String in, Command* cmd) {
  byte idx = 0;
  for (byte i = 0; i < in.length(); i++) {
    if (in.charAt(i) == 'x' || in.charAt(i) == 'y' || in.charAt(i)
== 'z' ) {
      cmd[idx].coordinate = in.charAt(i);
      cmd[idx].rotation = in.substring(i + 1).toInt();
      idx += 1;
    }
  }
}
```

Anexo C

Código HTML

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">

    <!-- Always force latest IE rendering engine or request Chrome
Frame -->
    <meta content="IE=edge,chrome=1" http-equiv="X-UA-Compatible">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <link rel="stylesheet" type="text/css" href="templateV1.css">
    <!--<script type="text/javascript"
src="interface.js"></script>-->
    <script type="text/javascript" src="client.js"></script>
    <title>RobotClean</title>
  </head>

  <body class="index">
    <div class="servers">
      <div class="serverStatus">
        <pre id="serverStatus" >WSS: <span
id="WebSocket">&#9679</span></pre>
        <pre id="serverStatus">DBC: <span
id="DBConnection">&#9679</span></pre>
      </div>
      <div class="serverButtons">
        <button id="WebSocketButton"
onclick="connection()">Connect</button>
        <button id="DBButton">Connect</button>
      </div>
    </div>

    <div class="infoBox">
      <div class="infoAndCommandBox">
        <button  id="StartButton" onclick="StartPauseClick()"
></button>
        <div class="RobotInfoBox">
          <pre id="RobotStatus" >Robot Status: <span
id="RobotStatusText"></span></pre>
          <pre id="X_Distance" >X Distance: <span
id="XDistanceText">0.0</span> mm</pre>
          <pre id="Y_Distance" >Y Distance: <span
id="YDistanceText">0.0</span> mm</pre>
          <pre id="X_Speed" >X Speed: <span
id="XSpeedText">0.0</span> mm/s</pre>
          <pre id="Y_Speed" >Y Speed: <span
id="YSpeedText">0.0</span> mm/s</pre>
        </div>
      </div>
      <div class="photoBox">
        <img id="photo"/>
      </div>
    </div>
```

```
    <div class="logBox">
      <div class="log" >Log</div>
      <div class="messages" id="messages">
         <p id="logText"></p>
      </div>
    </div>
  </body>
</html>
```

Código CSS

```
div.servers{
    height: 5%;
    max-width: 20%;
    border: 1px solid black;
    margin-bottom: 1%;
    padding:0.5%;
    display:flex;
}

div.serverButtons{
    max-width:60%;
    height:100%;
    margin-left: 10%;
}

div.serverStatus{
    font-size: 16px;
    font-style: normal;
    text-align: left;
    max-width:40%;
    height:100%;
}

div.infoBox{
    width:100%;
    height: 500px;
    max-height: 500px;
    border:1px solid black;
    display: flex;
    margin-bottom: 5px;
}

div.infoAndCommandBox{
    width: 25%;
    height: 100%;
    display: :flex;
    border-right:1px solid black;
}

div.RobotInfoBox{
    width: 100%;
    height: 75%;
    margin-left: 5px;
    font-size: 15px;
    font-style: normal;
    text-align: left;
```

```
}

div.photoBox{
    max-width: 100%;
    height:100%;
}

div.messages{
    width: 100%;
    height: 300px;
    display:flex;
    flex-direction: column-reverse;
    overflow:auto;
    overflow-y: scroll;
    border:1px solid black;
}

div.log{
    width: 100%;
    max-height: 300px;
    background-color:rgb(211,211,211);
    border:1px solid black;
    margin:0px;
}


#serverStatus{
    width:100%;
    margin:2% 1% 0% 0%;
}

#WebSocket{
    color:rgb(255,0,0);
}

#DBConnection{
    color:rgb(255,0,0);
}

#WebSocketButton{
    width:100%;
    height:10%;
    margin-bottom:1%;
}

#DBButton{
    width:100%;
    height:10%;
}

#StartButton{
    width:80%;
    height:10%;
    margin-left: 10%;
    margin-top:5%;
}


#logText{
    font-size: 15px;
```

```
    font-style: normal;
    text-align: left;
}

img{
    max-width: 100%;
    max-height: 100%;
}
```

Código .JS

```
var conn=null;
var state=null;

function connection(){
    startWebSocketConnection();
}

function startWebSocketConnection(){
    if(conn==null)
        conn= new WebSocket('ws://localhost:8081');
    else if(conn.readyState!=conn.OPEN)
        conn= new WebSocket('ws://localhost:8081');

    conn.onopen = function(e) {
        document.getElementById("WebSocket").style.color="green";

document.getElementById("WebSocketButton").onclick=function(){clos
eConnection()};

document.getElementById("WebSocketButton").innerHTML="Disconnect";
        addLine("Connected");
    };


    conn.onmessage= function(e){
        var x="";
        if(e.data.startsWith("X_Distance:")){
            x=e.data.substring(11,e.data.length);
            document.getElementById("XDistanceText").innerHTML=x;
        }
        else if(e.data.startsWith("Y_Distance:")){
            x=e.data.substring(11,e.data.length);
            document.getElementById("YDistanceText").innerHTML=x;
        }
        else if(e.data.startsWith("X_Speed:")){
            x=e.data.substring(8,e.data.length);
            document.getElementById("XSpeedText").innerHTML=x;
        }
        else if(e.data.startsWith("Y_Speed:")){
            x=e.data.substring(8,e.data.length);
            document.getElementById("YSpeedText").innerHTML=x;
        }
        else if(e.data.startsWith("State:")){
            x=e.data.substring(6,e.data.length);

document.getElementById("RobotStatusText").innerHTML=x;
        }
        else if(e.data.startsWith("Button:")){
            x=e.data.substring(7,e.data.length);
```

```
                document.getElementById("StartButton").innerHTML=x;
                state=x;
            }
            else if(e.data.startsWith("img:")){
                x=e.data.substring(4,e.data.length);
                document.getElementById("photo").src =
'' + x;
            }
            else
                addLine(e.data);
        };

        conn.onclose=function(e){
            document.getElementById("WebSocket").style.color="red";
            document.getElementById("DBConnection").style.color="red";
            addLine("Disconnected");

document.getElementById("WebSocketButton").onclick=function(){star
tWebSocketConnection()};

document.getElementById("WebSocketButton").innerHTML="Connect";
        };

        conn.onerror=function(e){
            addLine("No connection found");
        };
};

function StartPauseClick(){
    if(conn!=null){
        if(conn.readyState==conn.OPEN){
            conn.send(state);
        }
    }
}

function addLine(message){
    var newCon1=document.createElement('br');
    document.getElementById("messages").appendChild(newCon1);
    var field=document.getElementById("messages");
    var newCon=document.createElement('p');
    newCon.innerHTML=message;
    field.appendChild(newCon.firstChild);
};

function closeConnection(){
    conn.close();

document.getElementById("WebSocketButton").onclick=function(){star
tWebSocketConnection()};

document.getElementById("WebSocketButton").innerHTML="Connect";
}

/*
function hexToBase64(str) {
    return btoa(String.fromCharCode.apply(null,
str.replace(/\r|\n/g, "").replace(/([\da-fA-F]{2}) ?/g, "0x$1
").replace(/ +$/, "").split(" ")));
}*/
```