FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

DEPARTAMENTO DE
ENGENHARIA MECÂNICA

Desenvolvimento de um sensor de docagem com aplicação em robôs holonómicos

Dissertação apresentada para a obtenção do grau de Mestre em Engenharia Mecânica na Especialidade de Energia e Ambiente

Autor

Rafael Tiago Neto Salvado Martins da Conceição

Orientador

Norberto Pires

Júri

Presidente	**Professor Doutor Pedro Neto** **Professor Auxiliar da Universidade de Coimbra**
	Professor Doutor Nuno Mendes **Investigador Auxiliar da Universidade do Porto**
Vogais	
Orientador	**Professor Doutor J. Norberto Pires** **Professor da Universidade de Coimbra**

Colaboração Institucional

Budapest University of Technology and Economics

Coimbra, Fevereiro, 2015

Murphy era um otimista.

O'Toole

Aos meus pais e avô.

Agradecimentos

O trabalho que aqui se apresenta só foi possível graças à colaboração e apoio de algumas pessoas, às quais não posso deixar de prestar o meu reconhecimento.

Em primeiro lugar, ao Professor Norberto Pires, por ter aceite orientar a minha Dissertação de Mestrado e me ter ajudado a levá-la a bom porto.

Ao Tajti Ferenc, por me ter dado a oportunidade de desenvolver o trabalho que deu origem a esta dissertação, e também ao Kovács Bence pela ajuda prestada.

À Professora Ana Amaro e ao Professor Lezsovits Ferenc, por me terem ajudado a evitar a bancarrota durante o Erasmus.

A todos os que me ajudaram através da Internet, por partilharem o vosso conhecimento.

À minha companheira. Obrigado por seres quem és.

À minha família, muito especialmente aos meus pais e ao meu avô António por me apoiarem sempre.

A todos os meus amigos, em especial ao Pedro Baptista e ao Paulo Silva, por serem mais para mim do que irmãos de sangue, e também ao Bruno Pires, por todos os bons momentos passados ao longo do curso.

Por fim, ao Joaquim Salgueiro, por toda a inspiração.

Resumo

Cada vez mais, a área da robótica se tem dedicado ao desenvolvimento de robôs móveis destinados a prestar serviços a humanos. Desta forma, são muitas as situações em que um robô móvel necessita de realizar uma acoplagem mecânica, fazendo-o através de um processo de docagem.

O objetivo deste trabalho é o desenvolvimento de um sensor de docagem de baixo custo e complexidade com aplicação em robôs holonómicos, uma vez que a maior parte das soluções encontradas não cumpriam esses parâmetros.

Assim, a implementação final do sistema é composta por três LEDs de infravermelhos instalados na estação.de docagem, e por uma câmara Pixart associada a um computador de placa única Arduino, colocados no robô.

O robô consegue calcular a sua posição relativamente à estação de docagem com precisão de aproximadamente 1 centímetro a 1 metro de distância através da imagem observada. Com base nessa informação, toma decisões quanto ao seu movimento. O algoritmo de movimento foi desenvolvido com a robustez em mente, para que o computador central do robô tenha o mínimo de intervenção no processo.

Infelizmente, não foi possível testar eficazmente o algoritmo de docagem, uma vez que todas as formas para o fazer apresentaram grandes obstáculos à sua implementação.

No entanto, a minha expetativa e da equipa com quem trabalhei é de que, resolvidos os problemas com os simuladores e/ou com a plataforma robótica, seja possível testar do ponto de vista prático o algoritmo desenvolvido e confirmar o seu potencial.

Palavras-chave: Docagem, Holonómico, Robô, Sensor, Algoritmo, Sistema embebido.

Abstract

Increasingly, the area of robotics has been dedicated to the development of mobile robots designed to serve humans. Thus, there are many situations where a mobile robot needs to perform a mechanical coupling, doing so through a docking process.
The objective of this work is to develop a low-cost and low complexity docking sensor with application in holonomic robots, since most of the solutions found to this problem did not meet these qualities.

Thus, the final implementation of the system is composed of three infrared LEDs installed in the docking station, and a Pixart camera associated with an Arduino single board computer, placed on the robot.

The robot can calculate its position relative to the docking station with an accuracy of approximately 1 cm at 1 meter distance, from the observed image. Based on this information, it makes decisions about its movement. The motion algorithm was developed with robustness in mind, so that the robot's central computer has minimal intervention in the process.

Unfortunately it was not possible to effectively test the docking algorithm, since all the ways to do this presented great implementation hurdles.

However, my spectation, which is shared with the team I worked with, is that once the problems with the simulators and/or robot platform have been solved, it will be possible to test the algorithm from a practical point of view, and therefore assess its potential.

Keywords Docking, Holonomic, Robot, Sensor, Algorithm, Embedded system.

Índice

ÍNDICE DE FIGURAS

SIMBOLOGIA E SIGLAS

Simbologia

$abc_{LEDs\ v}$ – abcissa, em píxeis, dos LEDs verticais, no plano da imagem

r, R – distância do robô relativamente aos LEDs verticais

r_a – distância do robô relativamente ao LED A

r_a – distância do robô relativamente ao LED A

r_a – distância do robô relativamente ao LED A

r_b – distância do robô relativamente ao LED B

r_{c_1} – distância entre a câmara e o LED C, caso a câmara esteja à esquerda da estação (do ponto de vista desta)

r_{c_2} – distância entre a câmara e o LED C, caso a câmara esteja à direita da estação (do ponto de vista desta)

$Campo\ de\ visão$ – distância angular entre a câmara e os pontos limítrofes do plano da imagem

C_h– distância entre a câmara e chão

L – distância, em píxeis, entre dois pontos no plano da imagem

P_l – comprimento da pala

P_y – distância entre o centro do LED em questão e a pala

$Resolução$ – número de píxeis que preenchem uma imagem, desde uma extremidade à outra

$SensX$ – distância horizontal entre o LED C e os LEDs verticais

$SensY$ – distância entre os LEDs verticais (A e B)

α – distância angular entre um ponto qualquer do espaço, a câmara, e outro ponto

γ – ângulo entre o LED A, a câmara e o LED B

δ – ângulo entre o LED C, os LEDs verticais, e a câmara

δ_1 – ângulo entre o LED C, os LEDs verticais, e a câmara, caso a câmara esteja à esquerda da estação (do ponto de vista desta)

δ_2 – ângulo entre o LED C, os LEDs verticais, e a câmara, caso a câmara esteja à direita da estação (do ponto de vista desta)

ϵ – ângulo entre o LED C, a câmara, e os LEDs verticais

θ – ângulo entre o robô, a estação de docagem, e a reta perpendicular a esta

λ – desvio angular entre o eixo ótico da câmara e os LEDs

ϕ_b – ângulo entre o ponto resultante da interseção da reta vertical que atravessa os LEDs verticais e a reta horizontal que atravessa a câmara, a câmara, e o LED B, para quando a câmara está abaixo do LED B

ϕ_c – ângulo entre o ponto resultante da interseção da reta vertical que atravessa os LEDs verticais e a reta horizontal que atravessa a câmara, a câmara, e o LED A, para quando a câmara está acima do LED A

ϕ_m– ângulo entre o ponto resultante da interseção da reta vertical que atravessa os LEDs verticais e a reta horizontal que atravessa a câmara, a câmara, e o LED B, para quando a câmara está entre os LEDs verticais

ω_b – ângulo entre o LED B, o LED A e a câmara, para quando esta está acima do LED B

ω_c – ângulo entre o LED A, o LED B e a câmara, para quando esta está acima do LED A

Siglas

2D – Duas Dimensões

3D – Três Dimensões

4D – Quatro Dimensões

AASIC - Application Specific Integrated Circuit

API – Application Programming Interface

CCD – Charge-Coupled Device

CMOS – Composite Metal-Oxide Semiconductor

CPP – Ficheiro de código C++

DIP – Dual Inline Package

FPGA – Field Programmable Gate Array

GCC – GNU Compiler Collection

GNU – GNU's Not Unix!

H – Ficheiro de Cabeçalhos C

I2C – Inter-Integrated Circuit

ICSP – In-Circuit Serial Programming

IDE – Integrated Development Environment

INO – Ficheiro de entrada de projetos Arduino

IR – Infravermelhos

Isp – In-System Programming

LED – Light Emitting Diode

MHz - Megahertz

MinGW – Minimalist GNU for Windows

OCD – On Chip Debug

PLC – Programmable Logic Controller

PTZ – Pan-Tilt-Zoom

SMD – Surface Mount Device

SOC – System-On-a-Chip

UART – Universal Asynchronous Receiver/Transmitter

UML – Unified Modelling Language

USB – Universal Serial Bus

1. INTRODUÇÃO

1.1. Enquadramento e Motivação

Nos últimos anos, a pesquisa na área da robótica tem-se dedicado cada vez mais a aplicações orientadas à prestação de serviços diretamente a humanos (Kotoku, 2006). Assim, uma maior quantidade de robôs móveis está a ser desenvolvida para funções tão diversas como o desmantelamento de centrais nucleares, a produção de cocktails, o apoio a idosos, ou a realização de tarefas domésticas. De facto, nos tempos recentes, temos visto robôs de limpeza de relativo baixo custo à venda para o público geral, tais como o Roomba da iRobot Co., ou o Trilobite da Electrolux Co. Só em 2013 foram vendidos cerca de 4 milhões de robôs de serviço para uso pessoal e doméstico, um aumento de 28% face a 2012 (International Federation of Robotics, 2014).

Consecutivamente, para qualquer robô móvel, as situações em que é necessário que este se mova para um determinado ponto de interesse e realizar uma acoplagem mecânica por forma a realizar uma certa tarefa, ou seja, docar, são inúmeras. A docagem surge portanto como uma função essencial para um robô. Mais comummente, este processo é relevante no recarregamento das baterias da máquina, que deve ser feito de forma independente, por forma a garantir-se autonomia de longo termo.

Os robôs modernos têm usualmente processadores centrais bastante potentes, onde são tomadas as decisões comportamentais mais complexas. Desta forma, é importante que certas tarefas de baixo nível sejam executadas autonomamente por sistemas embebidos (Tajti, Szayer, Kovács, & Korondi). Este princípio é semelhante ao que acontece nos mamíferos, em que a maior parte das funções vitais é levada a cabo sem intervenção direta do neocórtex, responsável por tarefas complexas, tais como a perceção sensorial, raciocínio espacial e comandos motores (Lui, Hansen, & Kriegstein). Desta maneira, a implementação das instruções de docagem no próprio sensor liberta o processador central de ter que realizar esta tarefa.

Adicionalmente, convém que este sensor seja simples e eficiente para se reduzirem custos e para que a manutenção e melhorias sejam facilitadas. A maior parte das

soluções encontradas atualmente tem elevada complexidade, o que leva a muitos pontos de falha e a uma menor escalabilidade em aplicações viradas para o consumidor. Dado este problema, surgiu a ideia deste projeto, que visa encontrar um meio mais simples e eficiente de se resolver o problema.

1.1. Objetivo

Nesta dissertação pretende-se desenvolver um sensor de docagem simples e de baixo custo que possa ser aplicado em qualquer robô holonómico no plano horizontal. O sistema do sensor de docagem em desenvolvimento deverá ser capaz de, depois do robô ser levado a uma posição próxima da estação de docagem (através da memória espacial do computador central do robô), tomar conta do robô e levá-lo de forma eficiente e eficaz ao acoplamento com a estação de docagem. Adicionalmente, deverá calcular a sua posição relativamente à estação de docagem com base na informação visual disponível, que o computador central do robô cruzará com outras fontes de informação espacial por forma a deduzir a sua localização.

Para atingir os objetivos definidos, fez-se inicialmente uma análise teórica matemática do problema, e concebeu-se o algoritmo de docagem. Seguidamente, procedeu-se ao desenvolvimento de *software* para que a implementação destes desenvolvimentos através da programação em C++ de um microcontrolador fosse possível. Tendo-se escrito este último código, foi necessário proceder ao seu teste no terreno. Não se tendo conseguido proceder a este como estava previsto, optou-se pelo teste do código em *software* que simulasse o funcionamento de robôs.

Ao longo de todo o processo, foi necessário igualmente fazer uma revisão bibliográfica de sistemas semelhantes e do conhecimento pertinente ao assunto, à medida que tal se anunciava como necessário.

1.2. Estrutura da dissertação

A presente dissertação encontra-se estruturada em 5 capítulos principais.

O capítulo 1, *Introdução*, ostenta a importância da temática patente neste trabalho, com base no enquadramento e motivação, onde se apresenta a relevância social e científica do tema e, portanto, as razões que levam à necessidade da concretização deste

trabalho. São identificados, de modo conciso, os objetivos a que se propõe esta dissertação e, por fim, apresenta-se a presente estrutura da dissertação.

No capítulo 2, *Estado da arte*, apresenta-se uma revisão bibliográfica de estudos e conhecimentos relevantes ao presente trabalho.

O capítulo 3, *Metodologia*, expõe o procedimento adotado a fim de se atingir os objetivos propostos. Aqui são expostos, em relativo detalhe, os métodos escolhidos.

No capítulo 4, *Resultados e Discussão*, é revelado o produto final da dissertação bem como a análise crítica deste.

No capítulo 5, *Conclusões*, indicam-se as principais conclusões obtidas no trabalho e são sugeridos algumas recomendações para desenvolvimentos futuros.

2. ESTADO DA ARTE

O capítulo presente pretende analisar o conhecimento atual relevante ao problema em questão. É efetuado um enquadramento teórico a nível de conceitos que vão ser usados, e são apresentadas soluções semelhantes encontradas para os problemas enfrentados, fundamentando-se assim as escolhas feitas quanto aos métodos usados

2.1. Robôs de serviço

Um robô de serviço pode ser definido como um robô que executa tarefas úteis para humanos ou equipamentos, excluindo-se as aplicações de automação industrial. Desta forma, a classificação de um robô em robô industrial ou de serviço depende da aplicação pretendida. No entanto, devido à variedade de formas e áreas de aplicação, é difícil definir explicitamente o conceito de robôs de serviço (International Federation of Robotics, s.d.).

2.2. Soluções de docagem

O processo de docagem de um robô móvel pode ser definido como a deslocação deste para a menor distância possível de uma superfície, sem colidir com esta, orientando-se com a direção normal à dita superfície (Questa & Sandini, 1996). Existem várias solução para garantir que um robô efetue o processo de docagem corretamente. Grande parte destas utiliza um elevado número de sensores dispendiosos, o que aumenta apreciavelmente o custo do robô.

Por exemplo, o sistema de docagem proposto por Oh *et al.* para o robô Nomad XR4000 baseia-se no princípio que os aviões utilizam na aterragem em aeroportos: o alinhamento com a pista só é feito depois de o avião entrar na zona primária de controlo próxima do aeroporto. O *hardware* em si contém quarenta e oito sensores IR (infravermelhos) de curto alcance, um telémetro[1] a laser com um ângulo de visão de cento e oitenta graus e um sistema de sonar[2]. Na estação de docagem está instalado um farol de

[1] Dispositivo que mede a distância entre o observador e um ponto distante.
[2] Aparelho que deteta contornos à distância através da emissão de ondas sonoras, e posterior análise das ondas refletidas pelos ditos contornos.

longo alcance constituído por um conjunto de LEDs (*Light Emitting Diode*) de infravermelhos, para ajudar na orientação a grandes distâncias, efetuada com o apoio do sonar e do telémetro.

Para a fase de aproximação a curta distância com a estação de docagem, foi colocada uma grelha refletiva cujo perfil o telémetro a laser identificará, aproveitando o facto de a grelha ser constituída por superfícies afastadas e superfícies mais próximas. Isto permite o alinhamento correto do robô, dado que, dependendo da posição deste, o perfil observado será diferente. Assim, a grelha tem que estar junto a uma parede, caso contrário os dados do telémetro não são viáveis.

Esta grelha, visível na Figura 2.1, é constituída pelas seguintes partes:

- quatro buracos suficientemente longos para que seja possível gerar pontos de dados, com superfícies difusas no fundo;
- duas superfícies difusas largas, uma de cada lado, para melhorar a deteção a ângulos oblíquos
- duas superfícies brilhantes nas extremidades que não difundem o laser do telémetro, para marcar o início e o fim da grelha;

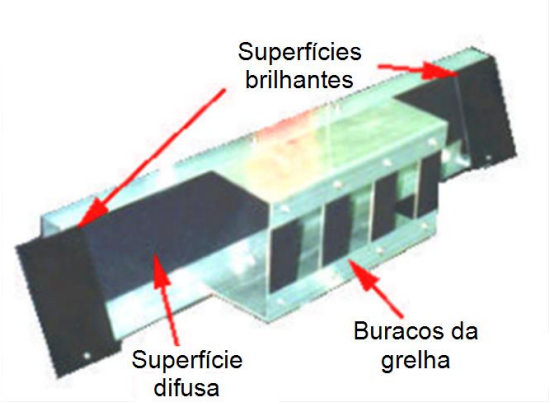


Figura 2.1. Grelha que o telémetro digitaliza remotamente; FONTE: alterado de **(Oh, Zelinsky, & Taylor)**

Outro sistema de docagem é o desenvolvido por Cassinis *et al.* (s.d.) que se baseia num princípio muito utilizado na navegação marítima, o das luzes de alinhamento: imaginem-se dois pontos não alinhados verticalmente, em quaisquer pontos do espaço. Se definirmos o plano de alinhamento como sendo o plano vertical definido por estes dois pontos, saberemos que estamos nesse plano quando conseguirmos visualizar os dois pontos como estando alinhados (Marinha do Brasil). Para tirar partido deste princípio, na estação de docagem foram instaladas duas lâmpadas a diferentes alturas e não alinhadas

verticalmente, como se pode ver na Figura 2.2. No robô foi instalada uma simples *webcam*, sendo as imagens processadas a preto e branco. A utilização de marcadores ativos (lâmpadas) facilitou na identificação dos pontos de alinhamento. Tendo em conta a distância horizontal observada entre as lâmpadas, conseguir-se-á assim deduzir o alinhamento do robô. Desta forma, quando as duas luzes estão uma por cima da outra, sabemos que o robô está alinhado, aumentando a precisão desta medida consoante o robô se aproxima da estação, por considerações geométricas.

Figura 2.2. Robô, com webcam em cima, e luzes de alinhamento; FONTE: **(Cassinis, Tampalini, Bartolini, & Fedrigotti)**

Alternativamente, o sistema de docagem apresentado por Silverman *et al.* (2002) para o robô Pioneer 2DX usa uma câmara a cores PTZ (*pan-tilt-zoom*)[3] para localizar a estação de docagem. Como é ilustrado na Figura 2.3, esta câmara procura um quadrado cor-de-laranja colocado acima da estação de docagem sempre que o robô queira docar, atraindo-o para a estação. Junto ao quadrado colorido foi colocada uma superfície refletiva com um padrão distinto, para que o telémetro a laser do robô consiga saber a orientação do robô relativamente à estação, bem como a distância à estação.

Este robô doca de "costas", pelo que terá que fazer a aproximação final cegamente. Quando a distância à estação é menor que um certo valor, o robô dá meia volta

[3] Câmara que permite a realização automática do movimento de rotação no plano horizontal e vertical da câmara, e ampliação da imagem. Estas câmaras são usualmente controladas remotamente em sistemas de segurança.

com ajuda da informação dos sensores de odometria[4], dado que a bússola instalada não é suficientemente precisa para esta manobra. Apesar dos sensores, é difícil dar uma volta de exatamente 180º, mas o cone onde o espigão do robô entra corrige estes erros mecanicamente.

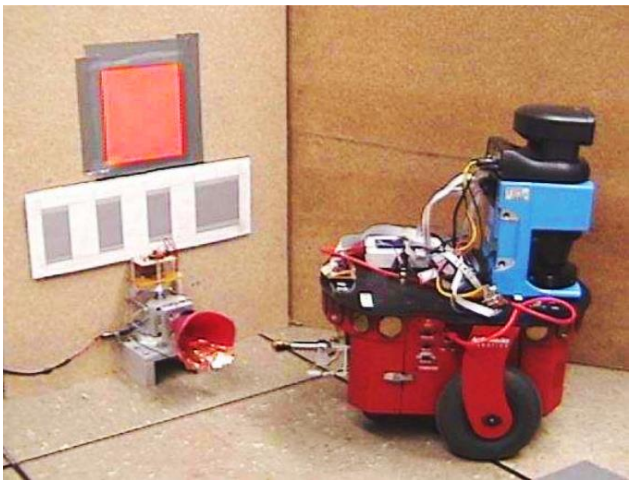


Figura 2.3. Estação de docagem, com o quadrado laranja em cima e a superfície refletiva abaixo deste, e robô com a câmara PTZ em cima; FONTE: **(Silverman, 2002)**

Uma solução mais económica foi apresentada por Kim, *et al.* (2005) que funciona com recurso a seis sensores no robô e cinco LEDs de infravermelhos na estação de docagem, apresentando resultados bastante satisfatórios. Tanto uns como os outros encontram-se dispostos à volta da plataforma que os suporta, radialmente, como se ilustra na Figura 2.4. Cada LED emite radiação intermitentemente de forma diferente, semelhante aos códigos emitidos pelos comandos de televisão. Dependendo do código (ou códigos) que se observa, pode-se saber qual dos LEDs se está a ver, conseguindo-se inferir a posição do robô relativamente à estação de docagem. Adicionalmente, dependendo de qual dos sensores (ou de quais) observa esses sinais, é possível inferir a orientação do robô relativamente à estação de docagem.

O algoritmo de docagem é constituído pelas seguintes fases:

- Inicialmente, o robô é levado para as proximidades da estação de docagem;
- seguidamente, o robô gira sobre o seu eixo vertical até que os dois sensores frontais capturem radiação dos LEDs;

[4] Sistema de cálculo de posição de um robô, relativamente a um ponto inicial, com base na análise de uma certa propriedade ao longo do tempo, tal como a velocidade das rodas em cada instante (como acontece num rato mecânico dum computador), ou a imagem capturada por uma câmara em cada instante (rato ótico)

- depois, gira novamente até que os sensores do seu lado esquerdo ou direito estejam virados para a estação, movendo-se em frente até que estes detetem a região central (vermelha);
- aqui, o robô vira-se para a estação, e ziguezagueia pela região central, corrigindo o trajeto sempre que sai desta, aproximando-se da estação, acabando por docar;

Apesar de o sistema ser pouco preciso (dado só existirem 9 posições possíveis a inferir, uma para cada combinação de regiões possível, e 7 orientações, de forma homóloga), o sistema funciona bem porque a estação de docagem corrige os erros de orientação mecanicamente.

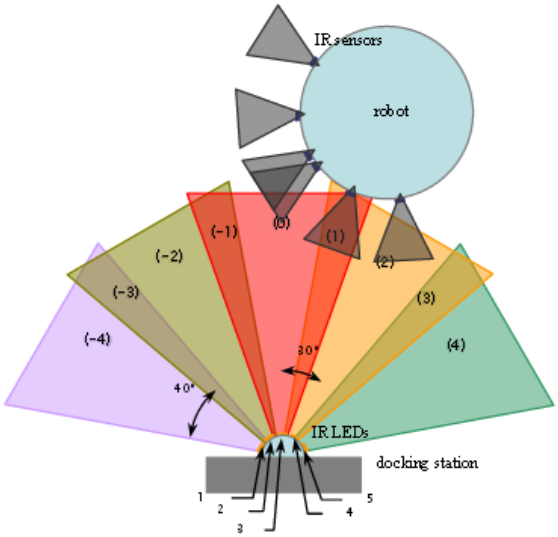


Figura 2.4. Esquema do sistema de orientação do robô, com as diferentes zonas de posicionamento a cores, e os sensores a cinzento; FONTE: **(Kim, et al., 2005)**

2.3. Sistemas de locomoção de robôs

O padrão de movimento que um robô segue quando doca depende da sua mobilidade. O facto de um robô conseguir executar rotações sobre o próprio eixo vertical, por exemplo, dá-lhe mais hipóteses de movimentos do que um que não o consiga. Consequentemente, o algoritmo de docagem de um robô deverá aproveitar ao máximo as capacidades deste, para que este processo se dê de forma fluida e eficiente. A agilidade de um robô depende do seu sistema de locomoção, existindo para o efeito vários sistemas diferentes.

A maior parte destes consistem em rodas simples ou em patas (Böttcher, s.d.). Os primeiros estão exemplificados pelos robôs visíveis nas figuras 2.2 e 2.3. Um exemplo de um segundo é o *Asimo* da Honda, na Figura 2.5. (a). Uma pequena quantidade de robôs utiliza métodos mais exóticos: uns tentam, similarmente aos robôs movidos por patas, imitar sistemas naturais, como as cobras (Figura 2.5. (b)); outros usam a deslocação por rodas móveis (Figura 2.5. (c)), ou por lagartas (Figura 2.5. (d)).

(a)

(b)

(c)

(d)

Figura 2.5. Exemplos de modos de locomoção:
(a) Robô bípede *Asimo* demonstrando a sua grande capacidade de equilíbrio; FONTE: **(Honda, 2015)**
(b) Robô Modular Snake Robot trepando um poste; FONTE: **(Spice, 2013)**
(c) Robô *Shrimp*, cujas rodas estão colocadas em braços que se ajustam ao terreno; FONTE: **(Bluebotics, s.d.)**
(d) Robô *Nanokhod*, movido por lagartas; FONTE: **(Hoerner, Sulger, & Max Planck Institute, s.d.)**

Sendo os mecanismos baseados em patas inspirados em sistemas biológicos, acabam por ser excelentes na sua capacidade de atravessarem eficazmente terrenos

inóspitos, se bem implementados. No entanto, copiar estes mecanismos é extremamente difícil e por várias razões. Os principais obstáculos são a complexidade mecânica dos mecanismos, a estabilidade do robô e o consumo energético. Assim, a roda revela-se como a melhor solução para a maior parte dos problemas de locomoção de máquinas, o que leva à sua frequente utilização como meio de locomoção para robôs. Isto devido à simplicidade mecânica desta e dos mecanismos envolventes, à mais fácil solução do problema da estabilidade do robô e à sua maior eficiência energética. No entanto, estes sistemas têm mais dificuldades em percorrer terreno acidentado. Posto isto, para a maior parte das situações em que a área de funcionamento do robô é criada pelo homem (terreno plano e previsível), a roda é efetivamente a melhor solução (Böttcher, s.d.).

Contudo, a maior parte dos robôs com locomoção por rodas simples tem mobilidade limitada. Isto porque para que o robô mude de direção, é necessário que o eixo de rotação da roda rode sobre o eixo vertical. De facto, basta imaginar o sistema de direção de um carro comum, em que a rotação das rodas de direção é limitada, o que leva a que deslocações para o lado levem a movimentos longos e complexos (p.e., avançar o carro, e recuar em marcha-atrás com as rodas viradas). Por forma a contrariar esta situação, podem-se tornar todas as rodas móveis em torno do eixo vertical. Isto faz efetivamente com que o robô se possa deslocar em todos os três graus de liberdade que possuiu (deslocação nos dois eixos, e rotações segundo o eixo vertical). Por outro lado, os mecanismos para tornar isto possível (rodas motorizadas com rotação completa do seu eixo vertical) tornam-se complexos e pouco ágeis, pelo que, quando se pretende um robô com estas capacidades, se costuma usar outro tipo de rodas. As chamadas *omni-wheels* são efetivamente rodas em cuja circunferência se colocam cilindros de eixo tangente a esta. Isto permite que as rodas transmitam movimento eficazmente, mas que também possam deslizar na direção perpendicular à direção de rotação. Posto isto, com a utilização de três rodas em triângulo, naquilo a que se chama um *kiwi-drive*, é possível obter um robô com três graus de liberdade no plano do chão. (Tajti, Szayer, Kovács, & Korondi).

Figura 2.6. *Kiwi-drive,* constituído por três *Omni-wheels;* FONTE: **(unclejoe, 2011)**

Outra forma de se utilizarem estas rodas é a apresentada no robô *BallIP*, visível na Figura 2.7. Aqui, com a configuração engenhosa de três *omni-wheels* e de uma esfera, consegue-se igualmente criar um robô holonómico, através da transmissão de movimento das rodas para a esfera. No entanto, os algoritmos de controlo da estabilidade do robô têm que ser bastante eficientes para evitar que o robô tombe (Kumagai & Ochiai, 2009).

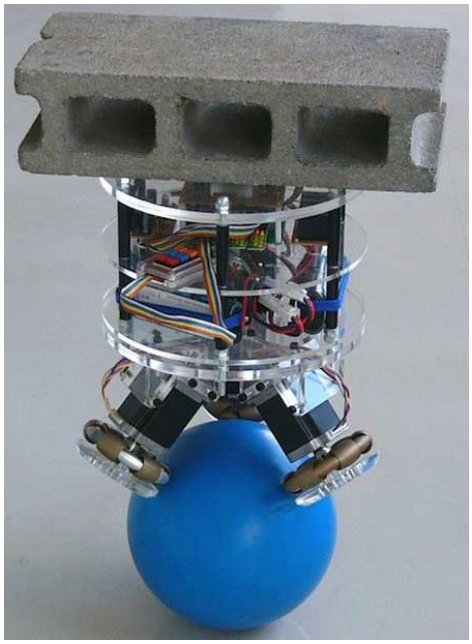


Figura 2.7. Robô *BallIP* suportando um tijolo de 10 quilogramas; FONTE: **(Tadakuma, 2006)**

Existem também rodas semelhantes em que os eixos dos cilindros estão a 45º com a circunferência da roda (Böttcher, s.d.), chamadas rodas *Mecanum*, utilizadas em grupos de quatro, tal como se pode ver no robô *Uranus*, desenvolvido na Universidade Carnegie Mellon (Podnar W. G.) .

Figura 2.8. Robô *Uranus,* com quatro rodas *Mecanum*; FONTE: **(Podnar W. G., 1985)**

Outro sistema utilizado para o efeito, mas muito mais raro, consiste na utilização da chamada roda esférica (ou *omni-ball)*, onde o robô é apoiado em três ou quatro esferas, como pode ser visto na Figura 2.9. Cada uma delas é constituída por duas rodas hemisféricas, rodando estas passivamente sobre um eixo comum que passa no centro das respetivas faces planas. Uma vez que o raio de cada roda hemisférica é igual ao da própria roda esférica, este sistema exibe uma ótima capacidade de se mover em terreno acidentado e de subir degraus, quando comparado com *omni-wheels* normais (Tadakuma, 2006).

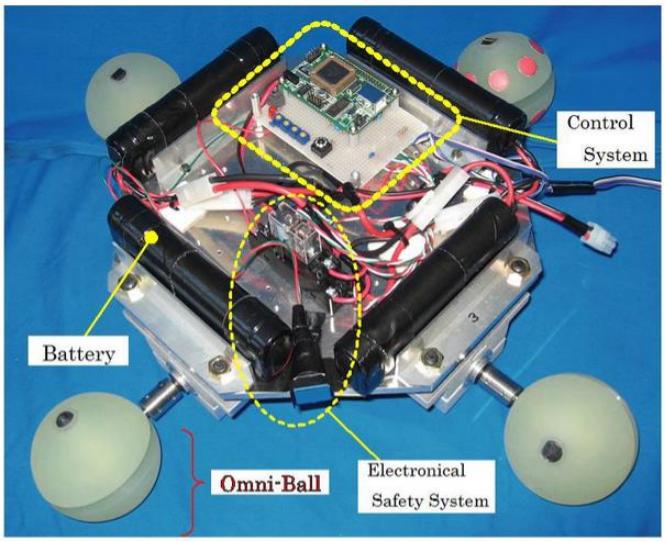


Figura 2.9. Robô que utiliza quatro *omni-balls* para a sua locomoção; FONTE: **(Tadakuma, 2006)**

Desta forma, estes últimos modos de locomoção são efetivamente sistemas holonómicos de locomoção: o número de graus de liberdade controláveis é igual ao número de graus de liberdade totais no espaço em questão (Hlaváč). Isto é, estando o robô

no plano do chão, conseguem-se controlar as translações nos eixos horizontais e as rotações sobre o eixo vertical.

2.4. Visão de máquina

Para que um robô consiga docar, este tem que adquirir a noção de onde está localizado espacialmente relativamente ao objeto com que se pretende docar. A forma mais usual de obter esta noção é através da deteção de radiação, tal como acontece nos olhos dos animais. Nos robôs, esta função é usualmente desempenhada por câmaras, constituindo um sistema de visão de máquina (MV – *Machine Vision*). Estes sistemas aplicam tecnologia e métodos baseados em visualização de imagens a sistemas de inspeção automática, controlo de processos, ou, no nosso caso, orientação de robôs (Ulrich, Wiedemann, & Steger, 2008).

2.4.1. Tipos de visão de máquina

Enquanto que a visualização 2D de luz visível é a mais utilizada em MV, outras alternativas são a visualização 2D de diversas bandas de infravermelhos (Wilson, 2011), o rastreamento linha a linha, a visualização 3D de superfícies, ou até mesmo a visualização 2D por raios-X (Turek, 2011). Pra o presente estudo, a visualização por raios-X não fazia sentido, nem o rastreamento linha a linha, uma vez que este pressupõe movimentos lineares previsíveis da câmara ou do objeto em estudo (Huang). No propósito de se identificar a estação de docagem, poder-se-ia utilizar a visualização 3D de superfícies, mas este processo é complexo e o equipamento necessário é mais dispendioso do que a visualização 2D. A visualização 2D de luz visível seria também igualmente complexa, uma vez que os dados de entrada (uma imagem a cores da estação de docagem) teriam que ser sujeitos a um processamento relativamente complexo. Estas duas hipóteses talvez levassem a que fosse necessário utilizar um sistema de processamento de imagem mais poderoso (e consequentemente mais dispendioso) do que aquele que acabou por ser utilizado neste projeto. Assim, a deteção de radiação infravermelha proveniente de marcadores ativos (LEDs) acaba por se tornar uma solução viável e simples para o problema.

2.4.2. Métodos de reconstrução especial

Para sabermos a posição de um robô relativamente ao objeto observador, é necessário possuir métodos que nos permitam reconstruir a cena no espaço tridimensional que gerou uma certa imagem. Esta cena é constituída pelos parâmetros da câmara (usualmente conhecidos) e sua posição no espaço relativamente ao objeto, e pelas propriedades geométricas e óticas deste. Infelizmente, quando uma imagem é criada a partir de uma cena, perde-se informação de profundidade. Isto faz com que seja difícil e muitas vezes impossível inferir a geometria 3D do objeto e a posição da câmara, dado que um número potencialmente infinito de superfícies 3D pode produzir a mesma imagem (Oswald & Klodt).

Para resolver este problema, existem várias soluções práticas que podem ser aplicadas a um robô móvel. Em primeiro lugar, temos o sistema de visão binocular utilizado nalguns animais com visão binocular, como os humanos. De facto, os humanos conseguem calcular bastante bem a geometria de um objeto observado, bem como a distância a que este se encontra. Na visão binocular, dois olhos capturam a mesma cena de pontos ligeiramente diferentes. Isto dá origem a duas imagens ligeiramente diferentes, devido ao efeito da paralaxe[5]. Com base nestas duas imagens, é possível inferir informação espacial sobre a cena em questão, efetuando uma triangulação[6]. Em visão de máquina, tais sistemas são conseguidos com a utilização de duas câmaras, constituindo um sistema de visão estereoscópica (Mattoccia, 2013). Alternativamente, se a câmara se mover, pode-se capturar uma imagem num instante, e outra um tempo depois. Se conhecermos precisamente a distância percorrida pela câmara, podemos utilizar as duas imagens como se tivessem sido capturadas por duas câmaras (Nevatia, 1975). Com efeito, pensa-se que o movimento alternado da cabeça de muitas espécies de pássaros tenha como fim a perceção de profundidade, dado que cada olho está em lados opostos da cabeça (Bruckstein, Holt, Katsman, & Rivlin, 2005).

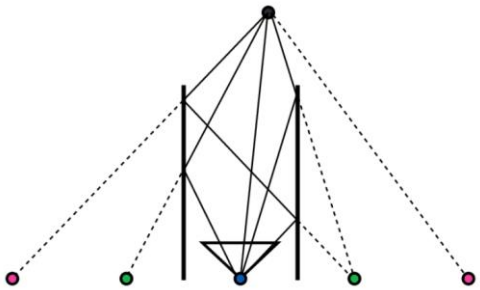
Outras formas que tradicionalmente se têm usado para recuperar informação de profundidade com uma só câmara baseiam-se também na análise da paralaxe observada,

[5] Alteração do posicionamento dum objeto numa imagem causado pelo deslocamento do ponto de observação. (Lathrop)

[6] Processo através do qual se determina a localização de um ponto criando um triângulo em que se define a distância entre dois pontos e se medem os ângulos entre essa distância e o ponto cuja localização se quer saber. Assim, conhece-se um dos lados do triângulo e dois dos seus ângulos, pelo que se podem calcular os outros dois lados e o outro ângulo. (Merriam-Webster, 2015)

mas desta vez devido à ampliação da imagem pela lente da câmara, cujos parâmetros se conhecem (Baba, Asada, Oda, & Migita).

Ainda outra maneira de se resolver este problema será através da utilização de sistemas catadióptricos. Nestes, espelhos posicionados à volta de uma só câmara permitem a condensação de várias imagens em uma, o que permite que só uma câmara efetue o trabalho de várias. Nestes moldes, existem inúmeras configurações de espelhos planos e curvos com as quais é possível implementar visão estereoscópica. Uma forma simples de se implementar este sistema consiste na colocação de dois espelhos planos, um de cada lado da câmara, como se mostra na Figura 2.10. (a) Isto leva a que existam duas "câmaras virtuais" (a verde na mesma figura). Se rodarmos estes espelhos um pouco e os generalizarmos para uma superfície curva à volta da câmara, obtemos o sistema da Figura 2.10 (b), cujo resultado pode ser visto na Figura 2.11. Como se pode verificar, os olhos do sujeito aparecerem três vezes, bem como todos os outros pontos da imagem. Com esta informação consegue-se fazer uma reconstrução 3D do modelo (Sturm, Ramalingam, Tardif, Gasparini, & Barreto, 2011)**.**



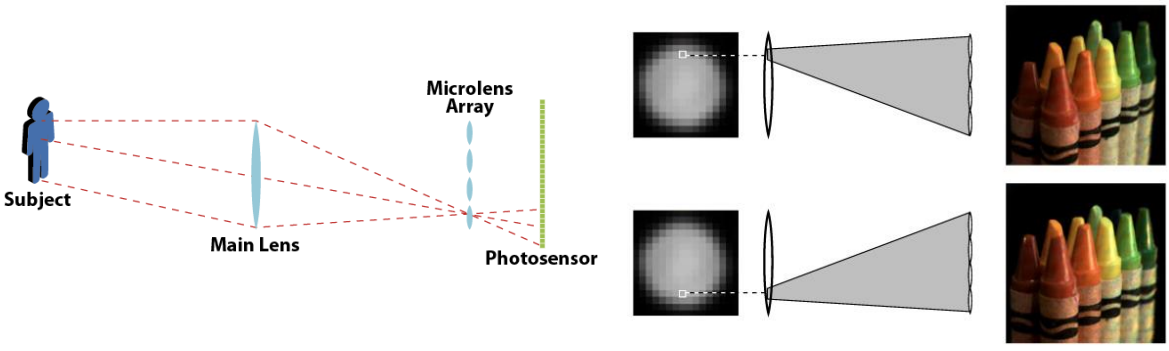
(a)

(b) (c)

Figura 2.10. (a) Sistema catadióptrico constituído por dois espelhos planos paralelos ao eixo focal da câmara (preto – objeto; azul – câmara; rosa – câmara real que capturaria os raios exteriores; verde – câmaras virtuais que capturam os raios exteriores;
(b) Sistema catadióptrico constituído por um espelho cónico em frente à câmara;
(c) Imagem capturada pelo sistema em (b);
FONTE: **(Sturm, Ramalingam, Tardif, Gasparini, & Barreto, 2011)**

Outro método curioso que também permite retirar informação de profundidade com um sistema de visão é o das câmaras plenópticas. Aqui, uma grelha de microlentes é posta em frente ao sensor ou filme da câmara. Isto faz com que se forme um número de pequenas fotografias igual ao número de microlentes, como se mostra na Figura 2.11 (a). Retirando informação de cada uma destas pequenas fotografias, é possível reconstruir uma imagem, de muitas à escolha. Efetivamente, com esta tecnologia é possível reconstruir o campo de luz 4D[7] em questão.

Atente-se na Figura 2.11 (b). As duas imagens da direita foram reconstruídas a partir da mesma "fotografia". Note-se que são ligeiramente diferentes, estando a câmara aparentemente a alturas diferentes em cada uma delas. À esquerda temos uma hipotética imagem capturada por uma microlente. Para reconstruir a foto de cima por exemplo, retira-se um píxel de cada microlente, neste caso, o que está na localização assinalada, e juntam-se todos. Para a imagem de baixo, retira-se o outro píxel assinalado. Desta forma, o número de imagens possíveis é igual ao número de píxeis capturados por cada microlente. Alternativamente, com a utilização cuidada de píxeis diferentes de cada microlente, é mesmo possível alterar o foco da fotografia depois (Ng, et al., 2005)



(a) (b)

Figura 2.11. Esquemas explicativos do funcionamento de uma câmara plenóptica:
(a) Projeção do objeto através de cada microlente;
(b) Correspondência entre dois píxeis específicos de cada microlente e a respetiva imagem reconstruída;
FONTE: **(Ng, et al., 2005)**

Todos estes métodos partem do princípio de que não conhecemos a forma do objeto que estamos a observar. No entanto, se possuirmos essa informação à partida, o

[7] Representação dos raios de luz presente num espaço livre, parametrizados com quatro dimensões. Basicamente, dados quaisquer dois pontos no espaço, consegue-se ter informação sobre o raio de luz que o atravessa. Contrastantemente, numa câmara normal ideal, para cada ponto no espaço, só se consegue ter informação sobre um único raio de luz (Wanner, Fehr, & Jähne)

problema de retirar informação espacial a partir duma imagem torna-se muito mais simples. De facto, foi o que se acabou por fazer no presente projeto, tendo-se usado como objeto da câmara marcadores ativos (LEDs) a distâncias pré-estabelecidas uns dos outros. Sabendo isso, torna-se moderadamente simples inferir a posição da câmara relativamente a estes com base numa imagem.

2.4.3. Modelos de câmara

Para que se consiga tirar conclusões sobre a disposição da cena que deu origem a uma certa imagem, é necessário um sistema lógico que permita relacionar a imagem obtida com a cena. Existem modelos matemáticos, chamados modelos de câmara, que permitem tal. Estes modelam a geometria básica da projeção de pontos 3D, curvas e superfícies numa superfície 2D, o chamado plano da imagem (Fleet & Hertzman, 2005).

Existem vários modelos de câmara, uns mais simples que outros. De facto, se se pretender modelar lentes compostas de câmaras reais, o problema pode-se tornar bastante complexo, dado que é preciso ter em conta efeitos de distorção da lente. (Sigal, 2008).

A maior parte das câmaras usa lentes para focar a luz no plano da imagem. Isto é feito para que se consiga capturar luz suficiente num espaço curto de tempo, a fim de que o objeto não se mova apreciavelmente nesse instante. Assim, o caso simplista destes sistemas é o modelo da lente fina, mostrado na Figura 2.12 (a). Neste, raios de luz emitidos a partir de um ponto no espaço viajam através da lente, convergindo noutro ponto atrás desta (Fleet & Hertzman, 2005).

Outro modelo, mais simples, é o modelo da câmara *pinhole*. Este pode ser deduzido através do caso específico de um modelo de câmara fina em que a abertura[8] da câmara tende para zero (Fleet & Hertzman, 2005). Ou seja, a luz proveniente do objeto, viajando em linha reta, passa toda por um buraco estreito (o *pinhole)*, projetando o objeto no plano da imagem, como se ilustra na Figura 2.12 (b). De facto, pela sua simplicidade, as primeiras câmaras inventadas pelo homem foram câmaras *pinhole*, devidas ao filósofo chinês Mozi (470-390 AC) (Phan, 2013).

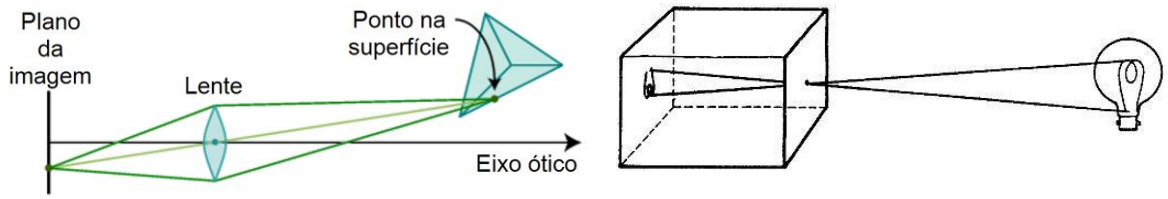
Os modelos usados em visão de máquina baseiam-se usualmente no modelo *pinhole.* Matematicamente, é equivalente ter o plano da imagem atrás do *pinhole* ou à

[8] Tamanho da abertura por onde passa a luz que chega à lente de uma câmara (Nikon, 2015)

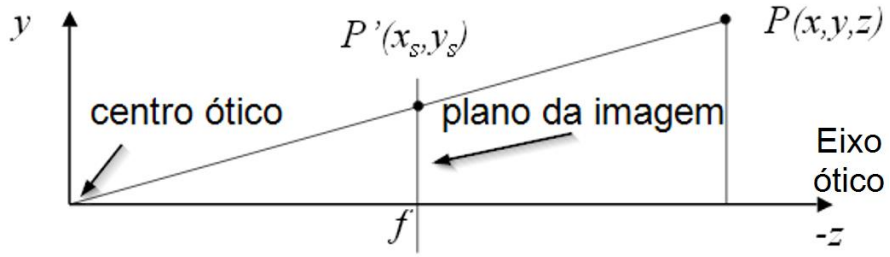
frente, como se exemplifica na Figura 2.12 (c). Desta maneira, estes modelos usualmente colocam esse plano à frente da câmara, a uma distância chamada *distância focal*, passando a origem da câmara a chamar-se *centro ótico*. Então, por semelhança de triângulos, e observando a Figura 2.12 (c), para qualquer ponto no espaço $P(x, y, z)$ a projeção de perspetiva no plano da imagem $P'(x_s, y_s)$ pode ser obtida pela transformação:

$$x_s = f \cdot \frac{x}{z} \; ; \; y_s = f \cdot \frac{y}{z} \Rightarrow P(x, y, z) \rightarrow P'\left(f \cdot \frac{x}{z}, f \cdot \frac{y}{z}\right) \tag{2.1}$$

Em que x_s é a abcissa do ponto P' projetado no plano da imagem, y_s é a ordenada do ponto projetado no plano da imagem, x, y, z são as coordenadas do ponto P no espaço, e f é a distância focal.



(a) (b)



(c)

Figura 2.12. Esquemas de modelos de câmara e projeções:
(a) Câmara de lente fina; FONTE: alterado de **(Fleet & Hertzman, 2005)**
(b) Câmara *pinhole*; FONTE: **(Nutffield Foundation, 2014)**
(c) Projeção de perspetiva; FONTE: alterado de **(Fleet & Hertzman, 2005)**

Este sistema foi o utilizado para modelar o funcionamento da câmara usada no presente estudo, uma vez que a lente desta é bastante pequena. Além disso, só é necessário um modelo preciso para situações em que o objeto esteja centrado na imagem, e é exatamente aqui que o modelo *pinhole* tem maior correlação com a realidade (Fusiello, s.d.),

2.4.4. Deteção de *blobs*

Em visão de máquina, um *blob* pode ser definido como uma região duma imagem associada com pelo menos um extremo local (Lindeberg, 1993). Por exemplo, um conjunto de píxeis de uma certa cor podem constituir um *blob*.

Por forma a extrair *blobs* a partir de uma imagem (i.e., saber que píxeis correspondem ao tal extremo local) existem vários métodos. Estes métodos de extração de *blobs* podem ser divididos em dois tipos: recursivos e iterativos. Sendo os métodos recursivos pouco eficientes para imagens relativamente grandes, optou-se pela utilização de um método iterativo. Este consiste na construção de um grafo[9] que contém todos os píxeis ligados uns aos outros que partilham uma certa propriedade. Os métodos usados para a construção deste grafo dividem-se em três grupos: algoritmos de uma passagem, de duas passagens ou de múltiplas passagens. A eficiência de cada método depende do tipo de imagem que se está a considerar (Wu, Otoo, & Suzuki). Uma vez que a implementação de extração de *blobs* não estava no âmbito inicial do trabalho, optou-se pela utilização do algoritmo mais simples de implementar: o algoritmo de uma passagem, tendo-se recorrido a uma busca em profundidade[10] Para o levar a cabo.

2.5. Sistemas de controlo

Para que o robô consiga docar, é necessário que algo tome em consideração a informação disponível ao robô, e tome decisões quanto ao seu movimento em função desta. Existem várias formas práticas possíveis de realizar essa função. Seguidamente, apresentam-se algumas delas, todas elas baseadas em soluções eletrónicas, já que é a melhor maneira de realizar operações lógicas com baixo custo e complexidade.

Em primeiro lugar, temos a hipótese de utilização de um chip FPGA (*Field Programmable Gate Array*). Estes chips, ao contrário dos microcontroladores, podem ser customizados facilmente pelo utilizador. Consequentemente, é possível obter um chip talhado à medida para a tarefa, sem ser necessário tê-lo produzido de raiz na fábrica. Este facto traz vantagens, uma vez que em grande parte das aplicações (nomeadamente em visão de robô) são bastante mais rápidos do que microcontroladores, devido à possibilidade

[9] Estrutura de dados constituída por um conjunto de nós ligados entre si. (Etessami)
[10] Método de travessia de um grafo em que se começa num nó arbitrário e se percorre o grafo o mais longe possível ao longo de cada ramo antes de se voltar para trás. (Eppstein, 1996)

de computação paralela e mais eficiente. No entanto, os chips FPGA são significativamente mais caros que os microcontroladores, e para a aplicação em específico o aumento de velocidade de processamento não se revelaria vantajoso, além de que acarretaria bastante mais trabalho de desenvolvimento. (National Instruments, 2011)

Uma solução similar seria o desenvolvimento de um ASIC (*Application Specific Integrated Circuit*), ou seja, a criação de um circuito integrado de propósito para o efeito. Estes circuitos oferecem vantagens semelhantes aos FPGA, mas a menor custo unitário, com ainda mais versatilidade de *design* com maior velocidade e com menor consumo energético (Roosta). No entanto, só se tornam viáveis em grandes quantidades, e as vantagens que oferecem acabam por não ser relevantes para o nosso projeto.

Outra opção seria a utilização de um PLC (*Programmable Logic Controller*). Estes controladores são bastantes mais robustos que a maior parte dos computadores de placa única, mas para a aplicação em questão não será necessário essa característica. Logo, dados os seus custos e peso superiores (Lipták, 2005), esta opção também foi rapidamente descartada

Em teoria seria possível resolver o problema ainda de outra maneira, com recurso a portas lógicas discretas. No entanto, dada a complexidade do problema, isto traduzir-se-ia em placas de circuito gigantes (Whitaker, 2006), extremamente complicadas, pelo que essa opção foi igualmente de imediato eliminada.

Outra forma de resolver o problema seria através da utilização de um computador de placa única, tal como o Arduino, baseado num microcontrolador AVR, ou o Raspberry Pi, baseado num microprocessador ARM. Dada a relativa simplicidade do nosso sistema, o poder deste último seria desperdiçado, acarretando simplesmente mais custos e complexidade de funcionamento. A solução que se acabou por escolher foi a utilização de um Arduino, que possui uma interface USB que permite a fácil comunicação com um computador, bem como a programação através do mesmo cabo, sem ser necessário retirar o microcontrolador da placa (Radioshack, 2014).

Outra opção teria sido a utilização de uma placa de circuito criada especificamente para esta aplicação, com base no microcontrolador do Arduino (ATmega328), ou outro. Desta forma, o custo seria mais baixo, uma vez que se eliminam os circuitos relativos a pinos não utilizados, bem como o módulo de interface USB, realizando-se agora a programação através do conector ICSP (*In-Circuit Serial Programming)*, com a ajuda de

um programador, tal como o AVRISP MKII, no caso de se optar por utilizar o ATmega328 (ATmel Corporation, 2008). No entanto, por uma questão de comodidade, bem como de maior rapidez de teste, o Arduino revela-se mais desejável nesta fase de prototipagem. De qualquer das formas, depois de desenvolvido, o programa a instalar no Arduino pode ser imediatamente transferido para uma placa de circuito com base no ATmega328.

2.5.1. Programação do microcontrolador

A seguir apresentam-se as considerações que se tiveram em conta relativamente ao método de programação do sensor.

2.5.1.1. Linguagens de programação

Originalmente, os microcontroladores eram programados somente em linguagem *Assembly* (Mann). Hoje em dia, várias linguagens de programação de alto nível estão disponíveis para utilização em microcontroladores, se bem que com algumas restrições, bem como com aperfeiçoamentos destinados a melhor suportar características específicas de certos microcontroladores (Carter, 1997). Assim, o programa a implementar no Arduino foi desenvolvido na linguagem de programação Arduino, sendo esta baseada na linguagem *Wiring*, por sua vez baseada na linguagem *Processing* (Arduino, s.d.). A linguagem Arduino aceita código C++, uma vez que o processo de compilação é efetuado pelo compilador AVR-GCC com a ajuda da AVR *Libc*, que efetivamente permite a utilização de código C e C++ praticamente puros em microcontroladores AVR (NonGNU). Desta forma, a grande maior parte do código escrito acaba por tratar-se de C++, tendo-se aproveitado as suas capacidades de orientação a objetos[11] por forma a estruturar o código mais claramente. Este facto facilitou a implementação de uma máquina de estados finitos hierárquica, que acabou por ser utilizada para modelar o problema em causa.

2.5.1.2. Máquina de estados finitos

Existem várias maneiras possíveis de organizar um código de programação. Um deles, a máquina de estados finitos, é particularmente útil para descrever o comportamento de um robô. Assim, não só se acelera o desenvolvimento do programa, como se facilitam

[11] Orientação a objetos, em linguagens de programação, é um tipo de programação em que não só é definido o tipo de dados de uma estrutura de dados, como também o tipo de operações (funções) que podem ser aplicadas sobre essa estrutura. Desta forma, a estrutura de dados transforma-se num *objeto* que inclui dados e funções, podendo-se criar relações entre diversos objetos. Através deste processo, a programação torna-se modular. (Beal, 2015)

futuras alterações, dado que o código produzido é mais intuitivo e estruturado (Carter, 1997).

Uma máquina de estados finitos é um modelo de computação, construído por uma máquina abstrata constituída por um conjunto finito de estados em que a máquina pode estar, um conjunto de eventos de entrada, um conjunto de eventos de saída e uma função de transição de estado. Esta função toma o estado atual e o evento de entrada e retorna um conjunto de eventos de saída e o estado seguinte (Free On-Line Dictionary of Computing, 2014). Mais concretamente, num determinado instante a máquina está num certo estado, o *estado corrente*, de entre um conjunto finito de estados. Quando ocorre um certo acontecimento, a máquina passa para um outro estado, mantendo-se neste até que ocorra outro evento. Uma máquina de estados finitos pode ser usada para descrever o comportamento de muitos aparelhos utilizados na sociedade moderna, bem como de conceitos mais abstratos, que efetuem uma sequência de ações pré-determinada dependendo de eventos que lhe são apresentados. Por exemplo, um semáforo teria três estados (no caso mais simples): vermelho, amarelo e verde. As condições de transição entre os estados seriam o tempo passado desde o início do estado corrente. As máquinas de estado finito têm aplicações em automação de projeto de circuitos integrados (Sherwood & Calder, 2001), em protocolos de comunicação (Rosier & Gouda, 1983), para modelação de sistemas neurológicos (Natschläger & Maass, 2012) e até para descrever a gramática de linguagens naturais (James, 2014).

2.5.1.2.1. Implementação de uma máquina de estados finitos

Relativamente à implementação duma máquina de estado em C++, existem várias formas de o fazer, como é explicado por Grosberg (2003) e parafraseado a seguir.

Em primeiro lugar, temos a implementação com recurso a uma *lookup table*[12]. Aqui, cada estado é identificado por um número inteiro único, sendo o estado corrente guardado numa variável inteira. Adicionalmente, as regras de transição de estado são guardadas num vetor bidimensional (uma tabela), em que um dos eixos contém todos os estados possíveis, e o outro eixo contém todos os eventos que provocam uma transição de estado.

[12] Tabela que contém valores pré-calculados de uma variável. Por exemplo, por vezes usam-se *lookup tables* trigonométricas que contêm o resultado de funções trigonométricas por forma a acelerar os cálculos (Jones, 2010)

Máquinas de estado implementadas desta maneira têm várias desvantagens. Primeiramente, ocupam muito espaço, dado que para cada combinação de entrada e de estado tem que haver um registo na tabela. Outra desvantagem é o facto de ser difícil de manter à mão, e, finalmente, de não poderem ocorrer ações durante as transições de estados, sem o recurso a ainda outra *lookup table*.

Para resolver estes problemas, muitas vezes recorre-se a blocos *switch-case* aninhados (um bloco para o estado corrente e outro para o evento de transição). No entanto, isto leva a um código difícil de ler, e, caso se pretenda uma máquina de estados dentro de outra máquina de estados (que foi o que se fez neste trabalho), o código torna-se um verdadeiro "esparguete". Além do mais, a execução do código é mais lenta, dado que o programa demora mais tempo a descobrir o número do estado corrente.

Felizmente, existe uma outra solução, que resolverá todos estes problemas. Esta passa por guardar o endereço do local do programa onde a execução deva começar a seguir. Isto pode ser fácil e eficientemente implementado em C++ (e em C) através da utilização de apontadores para funções, que guardam o endereço de funções. Isto leva a máquinas de estado fáceis de modificar, cujo código é relativamente fácil de ler, pelo que se acabou por seguir esta solução.

Para facilitar a escrita do código, considerou-se a utilização de um conversor UML (*Unified Modeling Language)*[13]- C++, que nos permite desenhar a máquina de estado com recurso a um diagrama UML, sendo o código C++ gerado a partir desse diagrama. No entanto, optou-se pela escrita do código manualmente, tendo-se criado diagramas simplesmente para organizar o raciocínio. Isto porque, apesar de em teoria ser mais fácil criar os diagramas UML e gerar o código automaticamente, este facto levaria a menos controlo sobre o código criado. Por outro lado, não se tendo experiência com esse modo de programação, que implica o conhecimento de várias regras para a criação dos diagramas (KDE, 2014), provavelmente acabar-se-ia por usar mais tempo e esforço seguindo esta linha de ação.

[13] Linguagem utilizada em engenharia de *software* que permite visualizar sob a forma de diagrama o design e o funcionamento de um sistema (KDE, 2014)

3. METODOLOGIA

Nesta secção, primeiro vai ser apresentado o caso de estudo que se usou para desenvolver o sensor de docagem, mostrando-se o robô onde seria instalado, e explicando-se o funcionamento do sistema. Seguidamente, descreve-se o desenvolvimento teórico matemático que se utilizou para calcular o posicionamento do robô. Em terceiro lugar, explica-se o funcionamento do algoritmo de docagem desenvolvido. A implementação da análise matemática e do algoritmo é analisada na quarta secção, onde se explicitam as várias fases seguidas. Numa primeira fase, explica-se a necessidade de monitorizar o funcionamento do microcontrolador usado com recurso a um programa criado para o efeito. A seguir analisa-se a implementação do código em si, bem como a forma como este está estruturado. Nesta sequência, enumeram-se os ajustes ao *hardware* que se consideraram necessários à correta implementação do processo. Por fim, discursa-se sobre o teste do código no robô e em simuladores, explicitando-se os diversos problemas encontrados nestas fases.

3.1. Caso de estudo

O desenvolvimento do sensor de docagem foi feito tendo por base um robô real, no qual seria instalado o dito sensor. Seguidamente apresenta-se o robô, e depois explica-se o funcionamento do sistema de docagem.

3.1.1. Apresentação do robô

O sensor foi desenvolvido com vista à aplicação numa base de robô para o setor dos serviços que tem vindo a ser construído pelo Departamento de Mecatrónica, Ótica e Engenharia Informática, da Universidade de Budapeste de Tecnologia e Economia, podendo a base ser usada para suportar diferentes robôs. Está correntemente a ser utilizada para suportar um robô usado em estudos de inteligência artificial com base no comportamento animal, visível na figura Figura 3.1, levados a cabo pelo *Comparative Ethology Group* da Universidade *Eötvös Loránd* de Budapeste. Mais especificamente,

pretende-se desenvolver modelos que permitam a um robô interagir socialmente com humanos de forma significativa e emocional. Entendendo-se que tal é importante para robôs de serviço, desenhados para trabalharem juntos a pessoas. Para tal, têm-se vindo a basear na relação ancestral entre humanos e cães.

Figura 3.1. Robô onde a base está a ser usada, visível em baixo; FONTE: **(Tajti, Szayer, Kovács, & Korondi)**

Trata-se de uma base holonómica, conseguindo o robô mover-se em qualquer direção imediatamente numa superfície horizontal, bem como rodar sobre o seu próprio eixo vertical.

O robô consegue ser holonómico devido à utilização de *omni-wheels* na base, em triângulo, em *kiwi-drive* (mostrado na Figura 3.1).

Uma vez que a bateria deste robô só o consegue abastecer durante quatro horas de funcionamento contínuo, é imprescindível que o robô recarregue a sua bateria autonomamente. Isto pode ser conseguido através de um processo de docagem automática com uma estação de carregamento de bateria.

3.1.2. Funcionamento do sistema de docagem

O fluxo de informação do Sistema de docagem pode ser visto na Figura 3.2.

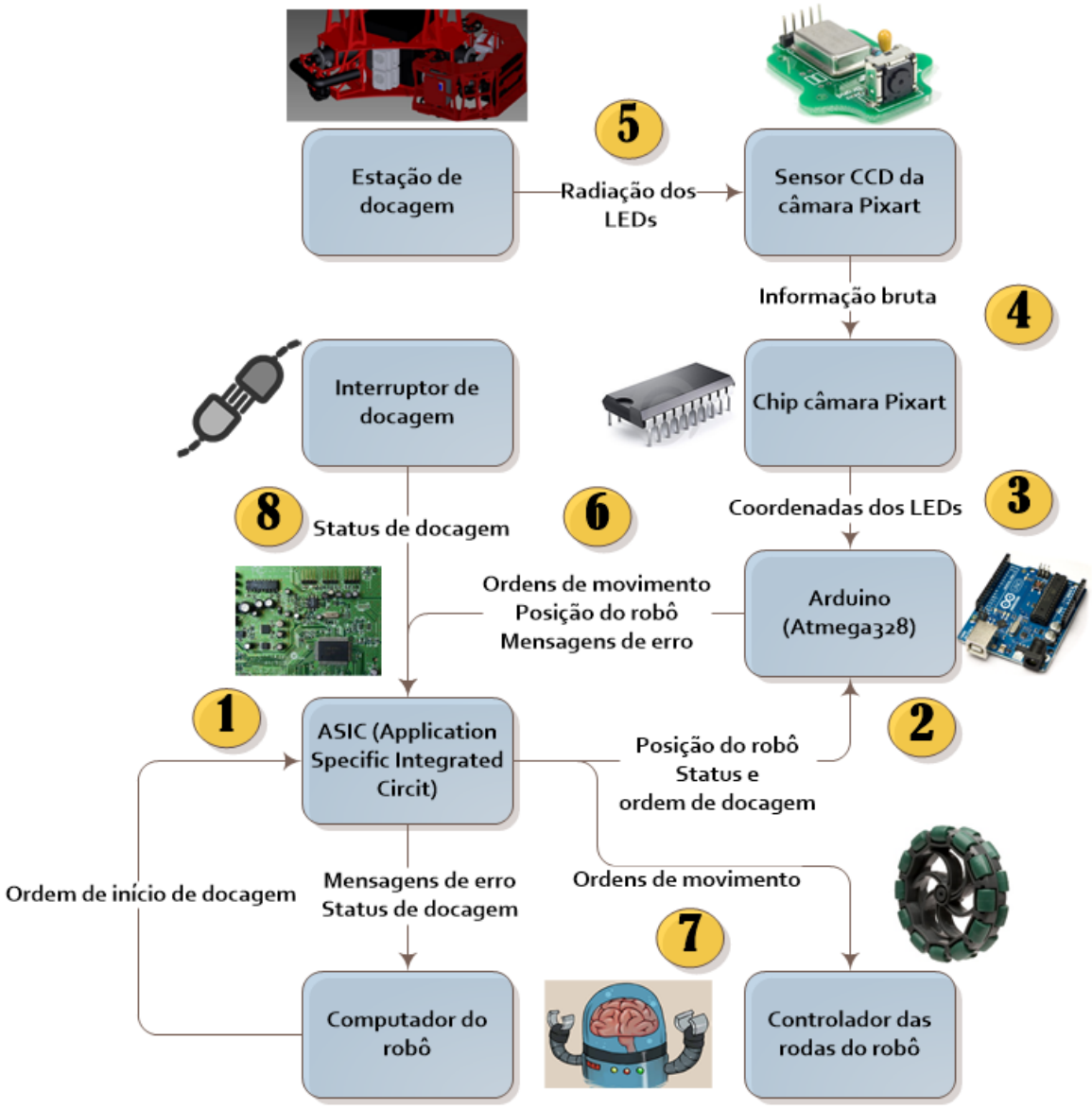


Figura 3.2. Esquema representativo do fluxo de informação do sistema de docagem;
FONTES: alterado com **(Tajti, Szayer, Kovács, & Korondi)**, **(Robotix, 2012)**, **(Plaxen & Desalu)**, **(Dreamstime, 2015)**, **(Embedded Systems Portal, s.d.)**, **(Arduino, s.d.)**, **(Ragab, s.d.)**, **(Clipart Panda, s.d.)**

Para compreender essa figura, imaginemos que desejamos docar o robô numa certa estação de docagem à nossa escolha, de entre quatro hipóteses de estações:

Em primeiro lugar, o computador central do robô coloca-o nas proximidades da estação de docagem em questão (graças ao seu sentido de orientação e com ajuda do mapa do espaço em que se encontra, que tem na memória).

- Seguidamente, o computador central envia a ordem de docar e a informação sobre em qual das estações se pretende docar ao ASIC,

onde o controlador das rodas (não visível na imagem) e o Arduino estão ligados. (1)

- Ao receber esta ordem, o ASIC reencaminha para o Arduino a informação recebida do computador central. Aqui, o controlo do robô é tomado pelo Arduino. (2)
- Agora, o Arduino recebe as coordenadas dos LEDs visíveis, enviadas pelo circuito integrado da câmara Pixart. (3)
- Este processou a informação visual bruta vinda do sensor CCD (*Charge-Coupled Device*) da câmara Pixart. (4)
- Por sua vez, este detetou a radiação de três LEDs colocados sobre a estação de docagem. (5)
- Com base nas coordenadas recebidas, o Arduino envia ordens de movimento de volta ao ASIC, bem como as coordenadas do robô calculadas com base na informação visual obtida, e eventuais mensagens de erro, se for esse o caso. (6)
- O ASIC envia os dados de movimento ao controlador dos motores das rodas. (7) Ao mesmo tempo, o ASIC envia continuamente para o Arduino informação sobre o movimento e posição do robô, bem como se a docagem já se efetuou ou não. (2)
- Esta última informação vem de um interruptor colocado no robô que se fecha quando o robô tiver docado com a estação. (8)

Seguidamente, lançar-se-á um olhar mais profundo sobre as partes mais relevantes do nosso sistema de docagem.

3.1.2.1. Câmara Pixart do Wiimote

No caso deste projeto, utilizou-se a câmara de infravermelhos da Pixart que é utilizada no comando da consola Wii da Nintendo. O comando Wiimote da consola Wii da Nintendo tem uma câmara de infravermelhos na sua frente que observa um conjunto de LEDs colocados perto da televisão. Com base na posição relativa destes LEDs, a consola calcula a posição aproximada do comando relativamente à televisão (WiiMote Physics, 2010).

Efetivamente, optou-se pela visualização 2D de infravermelhos, com a instalação de LEDs de infravermelhos na estação de docagem. Isto leva a que o processamento a realizar seja muito mais simples e consistente do que se se usasse visualização de luz visível por exemplo. Adicionalmente, para que o processamento básico da imagem não tivesse que ser feito pelo microcontrolador principal, utilizou-se um "sensor inteligente". Este não só captura a imagem, como ainda realiza algum processamento sobre esta, devolvendo os dados processados através de uma interface de comunicação (WiiMote Physics, 2010).

O sensor PixArt em si trata-se de um SOC (*System on a Chip*), ou seja, para além do sensor CMOS (*Composite Metal-Oxide-Semiconductor*), semelhante aos das câmaras digitais normais, com um filtro de infravermelhos posto à frente da lente, tem também um chip embebido que pré-processa a imagem. Desta forma, não temos acessos aos dados brutos do CMOS: o sistema fixa automaticamente até quatro fontes de infravermelhos e envia-nos as coordenadas destes pontos (que posteriormente serão chamados de *blobs*), bem como o "tamanho" de cada um, através do protocolo de transmissão de dados em série I^2C (*Inter-Integrated Circuit*). A câmara tem uma resolução real de 128 x 96 píxeis, mas interpola esses dados para a resolução de 1024 x 768 (Wiibrew, s.d.).

Não existindo documentação para este sensor, a engenharia reversa levada a cabo pela comunidade de *hacking* demonstrou-se imprescindível na utilização do dito sensor. Desta forma, a interface de baixo nível com o sensor foi levada a cabo com recurso à biblioteca *Pvision* para Arduino previamente desenvolvida por *Hobley* (Hobley, 2009), fortemente baseada no trabalho de Kako (Kako, 2007) no dito sensor, que se fundamentou no trabalho de muitos anónimos da comunidade Wiibrew que desvendaram os pormenores específicos do funcionamento do *Wiimote* (Wiibrew, s.d.).

3.1.2.2. Arduino

O "cérebro" do sistema de docagem deste projeto é um computador de placa única Arduino Uno, que utiliza o microprocessador de 8 bits ATmega 328P. O Arduino Uno é facilmente programável por USB com o *software* Arduino, que converte um ficheiro em linguagem de programação do Arduino em código máquina. Como já foi referido, é possível programar o ATmega328A utilizando praticamente todas as capacidades da linguagem C++. Adicionalmente, o Arduino coloca à disposição várias bibliotecas que

implementam funções de baixo nível, tais como a comunicação de dados em série. O Arduino Uno consegue interagir com um dispositivo através do protocolo I^2C, como é o caso do sensor Pixart, pelo que funciona perfeitamente no âmbito do projeto.

3.1.2.3. Estação de docagem

A estação de docagem em si foi desenvolvida previamente no âmbito doutros projetos, estando ainda em fase de melhoria. Esta deve realizar o acoplamento mecânico com o robô, com uma certa margem de erro, isto é, o robô não tem que acertar precisamente com a estação: deve-se realizar um encaixe correto a partir de posições diferentes do robô, dentro de certas margens. Nesse sentido, a estação tem características físicas que visam a que a inserção do robô seja relativamente fácil. Na Figura 3.3 podemos ver o robô acoplado à dita estação (à direita, a vermelho mais escuro).

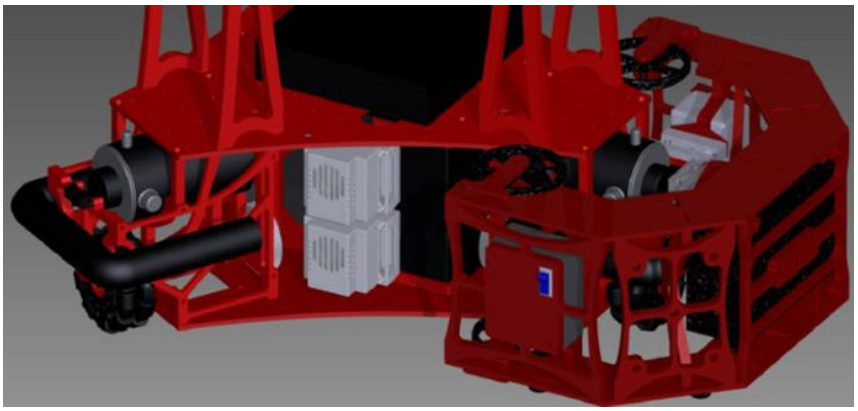


Figura 3.3. Renderização da estação de docagem e da base do robô FONTE: **(Tajti, Szayer, Kovács, & Korondi)**

Em cima desta estação será colocado o conjunto de três LEDs que o sensor visualizará por forma a deduzir a posição do robô. Dois LEDs um por cima do outro e outro LED ao lado, formando um triângulo retângulo, como se pode ver na Figura 3.4.

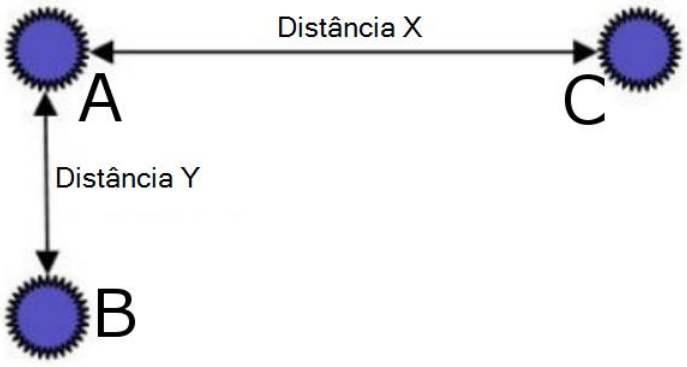


Figura 3.4. Configuração dos LEDs de infravermelhos; FONTE: alterado de **(Tajti, Szayer, Kovács, & Korondi)**

Desta forma e no plano de captura de imagem, a distância Y é proporcional à distância entre o sensor e os LEDs. Com esta informação, o ângulo relativo do robô é obtido através da distância X visualizada. Adicionalmente, esta solução simples permite a definição de mais três configurações de LEDs, visíveis na Figura 3.5, que o robô saberá distinguir, caso se deseje que o robô faça a docagem noutras hipotéticas estações de docagem.

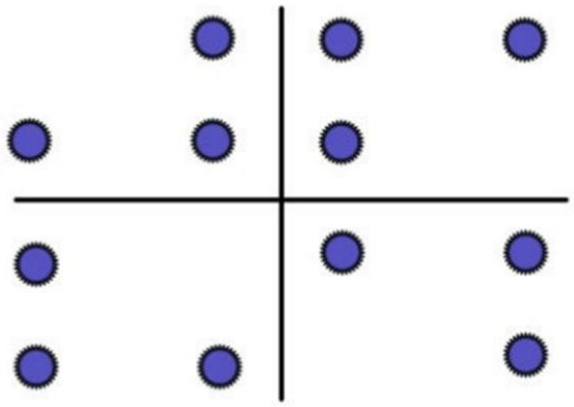


Figura 3.5. Configurações possíveis dos LEDs de infravermelhos FONTE: **(Tajti, Szayer, Kovács, & Korondi)**

Na Figura 3.6 podemos ver a plataforma onde os LEDs estão instalados. Podemos ver os LEDs acesos uma vez que as câmaras digitais são sensíveis à radiação infravermelha. Esta plataforma é um protótipo do produto final, tanto que possui outro LED C à direita, que pode alternar com o LED C aceso no meio, através do interruptor visível no meio. Isto fez-se para testar se a distância X, entre os LEDs A e C, influenciava significativamente a precisão do cálculo da posição do robô, tendo-se obtida uma resposta negativa a esta pergunta. Nesta plataforma, as distâncias X e Y são iguais, sendo ambas de 81 mm.

Figura 3.6. Plataforma onde estão instalados os LEDs de infravermelhos

3.2. Análise matemática do problema

Por forma a que o robô consiga calcular a sua posição com base na imagem observada, é necessário estabelecer a formulação matemática para tal.

O sistema de coordenadas do robô relativamente à plataforma dos LEDs pode ser visto na Figura 3.7, onde está representada uma vista de cima do sistema. Aqui, os LEDs verticais (A e B na Figura 3.4) estão no ponto roxo do lado direito, o LED C é o ponto roxo do lado esquerdo, o robô está representado pelo círculo azul e a câmara está representada pelo triângulo azul acoplado ao robô.

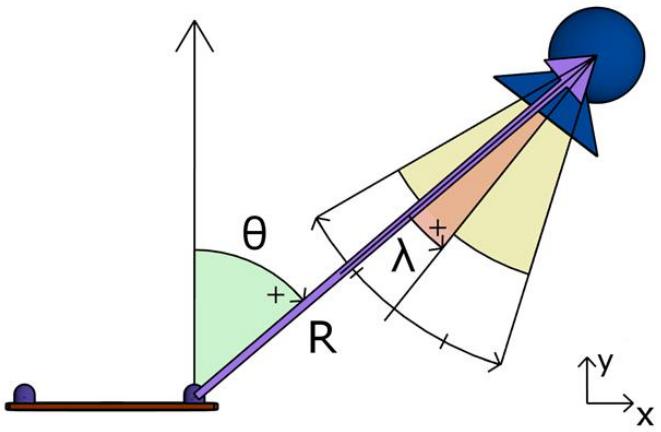


Figura 3.7. Sistema de coordenadas do robô

Na Figura 3.7, R é a distância do robô relativamente aos LEDs verticais, θ é a distância angular à linha perpendicular à plataforma de LEDs e λ é o desvio angular relativamente ao eixo da câmara. Isto é, quando a câmara está perfeitamente alinhada com os LEDs verticais, temos $\lambda = 0$.

Posto isto, sabendo as distâncias reais entre os LEDs e determinados ângulos, é possível ficarem-se a conhecer as grandezas acima descritas com uma análise geométrica. Para cada caso de posicionamento do sensor relativamente aos LEDs, em que *SensY* representa a distância vertical real entre os LEDs, temos:

$$\omega_c = 90^o - \gamma - \phi_c \quad (3.1)$$

$$r_a = \frac{\sin(90^o - \gamma - \phi_c) \cdot SensY}{\sin\gamma} \quad (3.2)$$

$$r = \cos\phi_c \cdot r_a \quad (3.3)$$

$$r_b = \frac{r}{\cos(\phi_c + \gamma)} \quad (3.4)$$

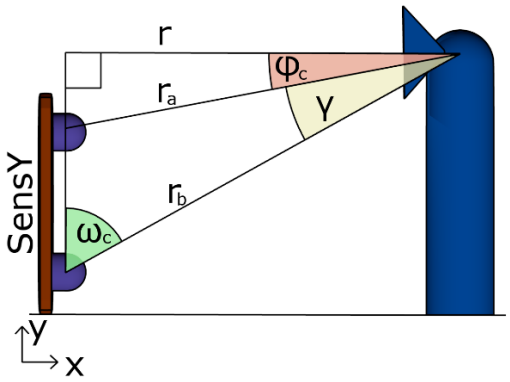


Figura 3.8. Esquema para o cálculo da distância para o sensor acima do LED A

$$\omega_b = 90^o - \gamma - \phi_b \quad (3.5)$$

$$r_b = \frac{\sin(90^o - \gamma - \phi_b) \cdot SensY}{\sin\gamma} \quad (3.6)$$

$$r = \cos\phi_b \cdot r_b \quad (3.7)$$

$$r_a = \frac{r}{\cos(\phi_b + \gamma)} \quad (3.8)$$

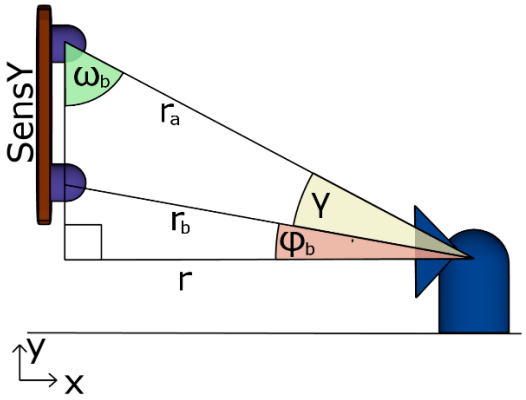


Figura 3.9. Esquema para o cálculo da distância para o sensor abaixo do LED B

$$r_a = \frac{\frac{\phi_m}{\gamma} \cdot SensY}{\sin\phi_m} \quad (3.9)$$

$$r = \cos\phi_m \cdot r_a \quad (3.10)$$

$$r = \cos\phi_m \cdot r_a \quad (3.11)$$

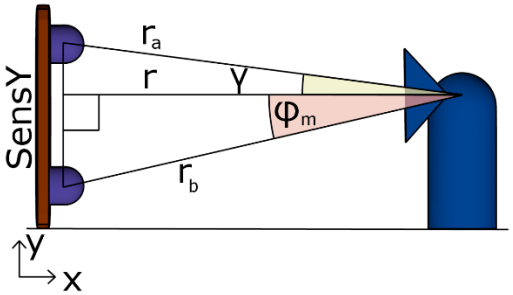


Figura 3.10. Esquema para o cálculo da distância para quando sensor se encontra entre os LEDs A e B

Esta formulação parte do pressuposto de que sabemos os ângulos γ e ϕ. Por forma a obtê-los, temos que modelar o funcionamento da câmara, isto é, a forma como transforma objetos reais em imagens.

Utilizando uma projeção de perspetiva, obtém-se a Figura 3.11.

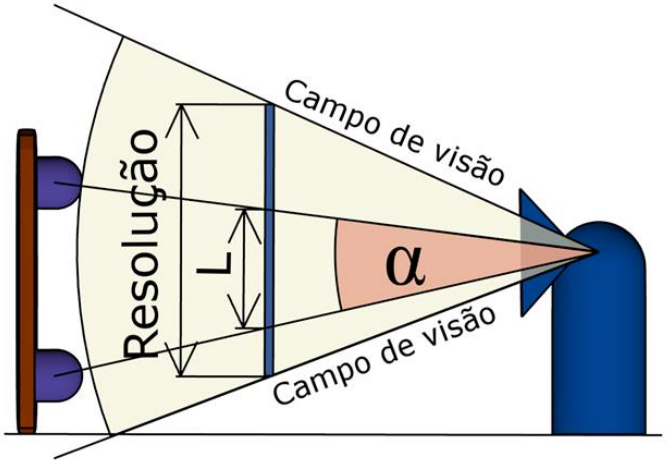


Figura 3.11. Esquema para o cálculo dos ângulos de visão

Sendo a linha azul o plano da imagem.

Consequentemente, pode-se aproximar que, para quaisquer dois pontos visíveis, e numa certa direção:

$$\frac{\tan(\alpha\ [graus])}{\tan(Campo\ de\ visão\ [graus])} = \frac{L[píxeis]}{Resolução[píxeis]} \tag{3.12}$$

Onde:

$$\alpha = \arctan\left(\frac{\tan(Campo\ de\ visão) \times L}{Resolução}\right) \tag{3.13}$$

Em que α é o ângulo de visão entre quaisquer dois pontos, $Campo\ de\ visão$ é o ângulo máximo de imagem que a câmara consegue capturar na direção em questão (horizontal ou vertical), L é a distância em píxeis entre esses pontos na imagem, e $Resolução$ é a resolução da câmara em píxeis na direção em questão (horizontal ou vertical). Esta equação é mais precisa quando α é mais baixo, e quando a imagem está no

centro da câmara. Uma vez que é nestas situações (longe da estação e com esta centrada) em que tais cálculos são mais importantes, entende-se que a aproximação é válida

Posto isto, temos o método para calcular os ângulos anteriormente referidos.

Relativamente ao ângulo θ, o cálculo deste revela-se um pouco mais complicado. De facto, para uma mesma imagem podem existir dois posicionamentos que geram a mesma imagem. Repare-se na Figura 3.12.

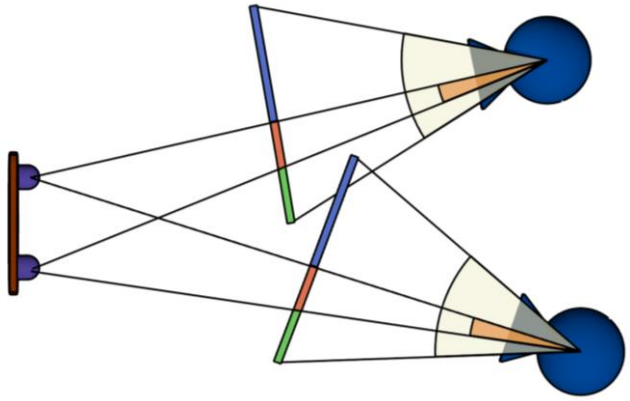


Figura 3.12. Esquema das posições possíveis do sensor para a mesma imagem

Na imagem acima, ambas as câmaras estão à mesma distância do LED de baixo. Adicionalmente, as duas imagens são iguais, como se pode ver pela correspondência entre a cor dos retângulos no plano da câmara. Logo, temos duas posições possíveis para a mesma imagem, não se conseguindo distinguir qual delas é a correta por simples observação da imagem. Faz-se uma discussão mais aprofundada sobre este problema na discussão sobre a implementação do código no Arduino.

Logo, existem dois θ a ser calculados, o que será feito com base na Figura 3.13.

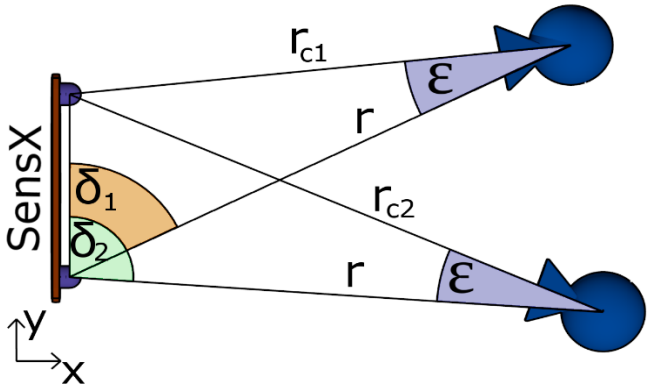


Figura 3.13. Esquema para o cálculo de teta

Primeiro definimos que:

$$\theta = \delta - 90^o \tag{3.14}$$

Seguidamente, com recurso à lei dos cossenos, obtemos:

$$SensX^2 = r^2 + r_c^2 - 2 \cdot r \cdot r_c \cdot \cos\epsilon \tag{3.15}$$

Recorrendo à fórmula resolvente, obtemos:

$$r_{c1} = \frac{2 \cdot r \cdot \cos\epsilon + \sqrt{4 \cdot r^2 \cos^2\epsilon - 4 \cdot (-SensX^2 + r^2)}}{2} \tag{3.16}$$

$$r_{c2} = \frac{2 \cdot r \cdot \cos\epsilon - \sqrt{4 \cdot \epsilon^2 \cos^2\epsilon - 4 \cdot (-SensX^2 + r^2)}}{2} \tag{3.17}$$

Para descobrir δ, aplicamos novamente a lei dos cossenos, agora que já sabemos r:

$$r_c^2 = SensX^2 + r^2 - 2 \cdot SensX \cdot r \cdot \cos\delta \tag{3.18}$$

$$\delta_1 = \text{acos}\frac{(-r_{c1} + SensX^2 + r^2)}{2 \cdot SensX \cdot r} \tag{3.19}$$

$$\delta_2 = \text{acos}\frac{(-r_{c2} + SensX^2 + r^2)}{2 \cdot SensX \cdot r} \tag{3.20}$$

Em que: $SensX$ é a distância real entre os LEDs; r é a distância do sensor aos LEDs verticais (A e B); r_c é a distância ao LED C; ϵ é o ângulo de visão entre o LED C, o sensor, e os LEDs verticais (A e B); δ é o ângulo entre o LED C, os LEDs verticais (A e B) e o sensor; e os índices 1 e 2 identificam o caso possível de o sensor estar à direita ou à esquerda da estação de docagem, respetivamente (do ponto de vista do robô).

Falta ainda calcular o ângulo λ (desvio angular entre o eixo da câmara e os LEDs verticais). Este é calculado com uma aplicação direta do método usado para calcular

os ângulos de visão, uma vez que esse ângulo acaba por ser a distância angular entre o centro da imagem e os ditos LEDs. Posto isto, obteremos:

$$\lambda = \frac{Campo\ de\ visão}{Resolução} \times \left(abc_{LEDs\ v} - \frac{Campo\ de\ visão}{2} \right) \tag{3.21}$$

Em que $abc_{LEDs\ v}$ é a abcissa (em píxeis) dos LEDs verticais.

Com recurso às equações expostas nesta secção, podemos calcular as coordenadas do robô relativamente à estação de docagem.

3.3. Algoritmo de docagem

Na Figura 3.14 pode-se ver a sequência de movimentos que um robô teria que seguir num processo de docagem. As cores das setas correspondem às cores dos estados da Figura 3.15, onde se pode ver o esquema de alto nível do algoritmo de docagem do robô. A seguir ao diagrama, explicita-se melhor o que se está a passar.

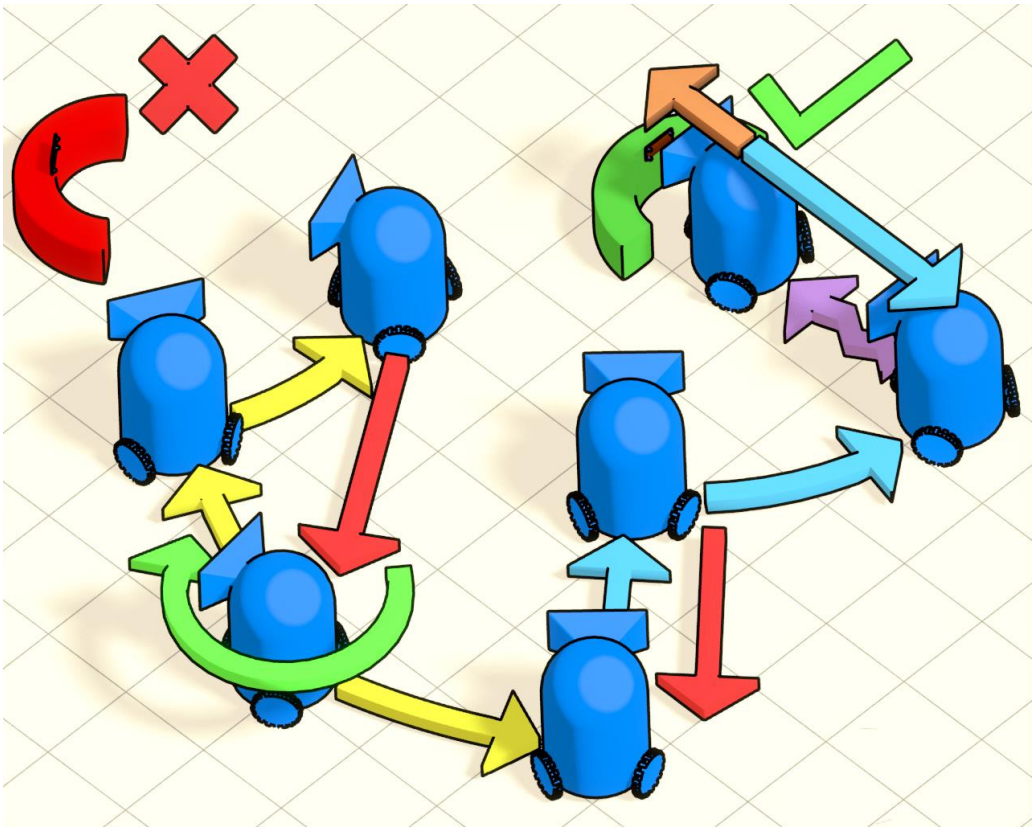


Figura 3.14. Esquema da sequência de movimentos subjacente a uma hipotética docagem. A estação vermelha é uma com a qual não se pretende docar. A estação verde é a estação correta.

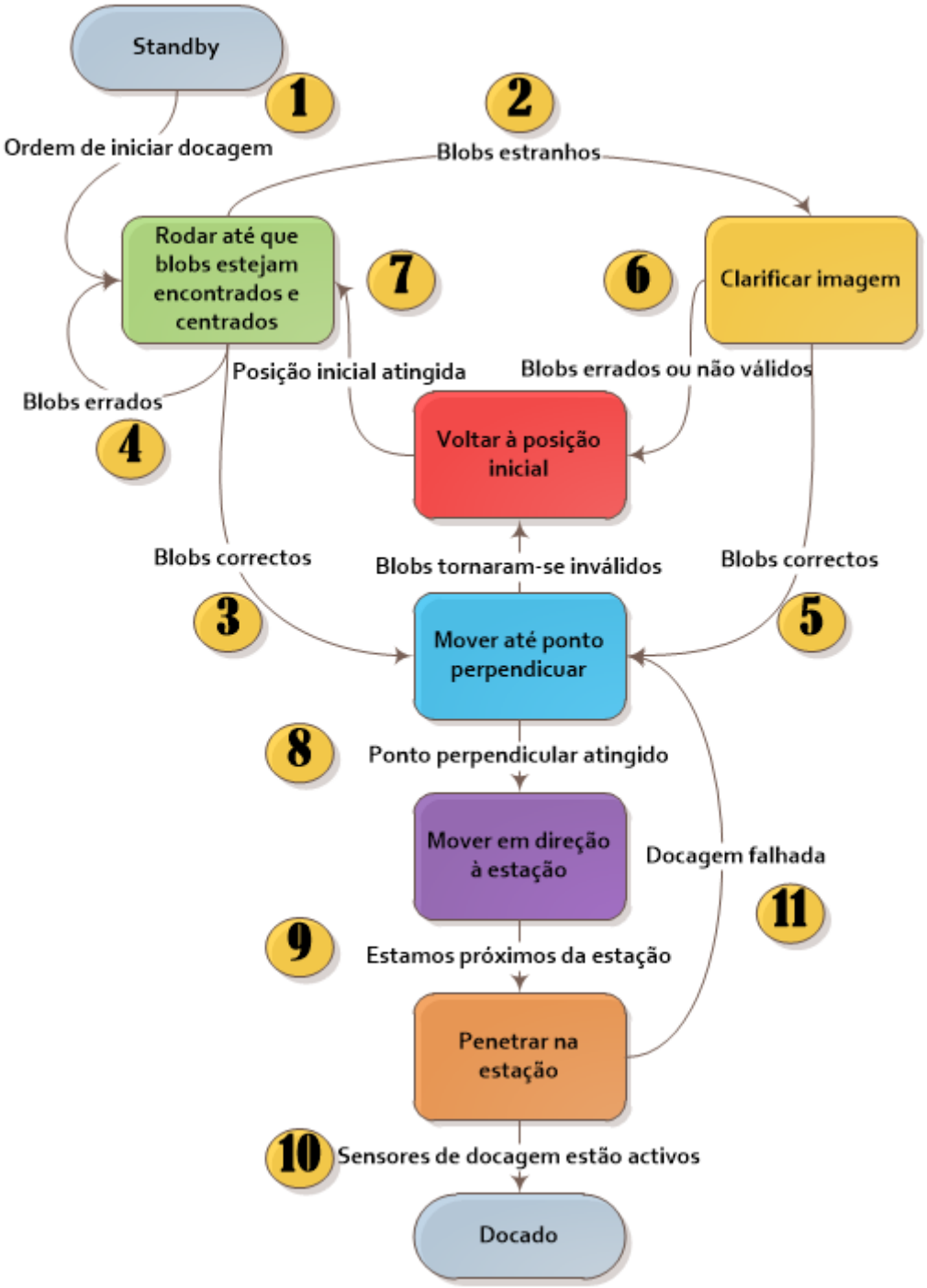


Figura 3.15. Algoritmo de docagem de alto nível

Inicialmente, o programa está em standby, à espera de ordens. No momento em que o computador central do robô dê a ordem de docagem, o nosso programa toma o controlo (1). Antes disto acontecer, o computador central deve levar o robô a uma posição que seja relativamente próxima da estação de docagem, de modo a que o sensor consiga ver alguns LEDs.

Desta forma, inicialmente o robô começa a rodar sobre o seu próprio eixo até que encontre alguns *blobs*, e estes estejam centrados na imagem. Aqui, pode ocorrer um de três casos:

1. Caso os *blobs* encontrados não correspondam a nenhuma configuração das possíveis (i.e., não sejam válidos), o robô tentará clarificar a imagem, deslocando-se até que possa ter uma visão mais clara da situação (2).
2. Caso os *blobs* encontrados correspondam à configuração de *blobs* desejada, proceder-se-á à docagem com a estação (3).
3. Caso os *blobs* encontrados não correspondam à configuração de *blobs* desejada, o robô continuará a rodar à procura de mais grupos de *blobs*, rodando sobre o seu eixo vertical (4).

O caso 1 pode levar a duas situações, depois de o robô obter uma visão mais clara dos *blobs*:

1.1. Caso estes correspondam à configuração correta, proceder-se-á à docagem normalmente (5).

1.2. Caso não correspondam à configuração de *blobs* desejada, o robô voltará à posição onde estava quando se começou a deslocar para clarificar a imagem (6) (uma vez que teve que se deslocar por forma a obter uma visão mais clara da situação). Quando a posição inicial tiver sido atingida, o robô continuará a girar até encontrar outro grupo de *blobs* (7).

Assim que o robô encontra os *blobs* que acha serem os corretos, este começa efetivamente a docar. Primeiramente, o robô deslocar-se-á até um ponto que esteja na linha perpendicular à estação de docagem. Para tal, primeiro desloca-se em linha reta em direção à estação até que a distância a esta seja inferior a um certo valor. A seguir, descreve um arco de circunferência, estando sempre virado para a estação de docagem. Esta trajetória é possível graças ao facto de o robô conseguir rodar sobre o seu eixo vertical, ie, ser holonómico.

Recorde-se que, com base na imagem que o robô tem dos LEDs, este não consegue saber qual dos dois, θ_1 ou θ_2, é real. Ou seja, o robô acaba por não saber em que direção se deve mover para chegar à perpendicular à estação. Por este facto, assim que o robô se começa a mover, compara a distância horizontal entre os LEDs observada com a distância observada quando estava parado. Se esta tiver diminuído, ou se um LED desaparecer, isto significa

que o robô se está a mover na direção incorreta. Aqui, o robô corrige o curso, começando a mover-se na direção oposta, e o θ correto é selecionado.

Quando se atinge o dito ponto na linha perpendicular à estação, o robô começará a mover-se a direito em direção a esta (8), corrigindo eventuais desvios na rota. Quando o robô estiver suficientemente perto da estação, deixará de se guiar pela informação visual (uma vez que a curta distância os LEDs poderão sair do seu ângulo de visão, e avançará cegamente em frente (9). No instante em que os sensores de docagem deem sinal, a docagem dá-se por terminada (10). Caso estes não deem sinal, passado um certo tempo de se iniciar a manobra cega, o robô mover-se-á em linha reta para trás até atingir o ponto perpendicular inicial (11). Daí, iniciará outra tentativa de docagem.

Não se tema que o robô entre num ciclo vicioso de tentativas em qualquer um dos passos, pois a maior parte da deteção de erros está omitida do diagrama acima por forma a não o sobrecarregar.

3.4. Implementação do processo

Na primeira fase, desenvolveu-se um programa de computador para se monitorizar o funcionamento do Arduino, por forma a que o desenvolvimento posterior de código neste fosse possível. Seguidamente, a escrita do código do Arduino foi sendo feita em C++. Por fim, dado que não foi possível testar o código no robô real, tentou-se fazer isto num programa que simule fisicamente o funcionamento de robôs, sem sucesso.

3.4.1. Monitorização do funcionamento do Arduino

Em primeiro lugar, foi preciso desenvolver uma forma de monitorizar o funcionamento do código implementado no Arduino, por forma a conseguir identificar e corrigir erros de programação, inevitáveis num programa desta complexidade, bem como eventuais erros de funcionamento. Uma solução de *debugging* possível teria sido a utilização de software que emulasse o Arduino, o que permitiria a utilização de *breakpoints*. Estes deixar-nos-iam compreender mais facilmente o funcionamento do código. No entanto, não se encontrou um emulador que suportasse a câmara PixArt, o que seria indispensável dada a fulcralidade desta no nosso programa. Outra hipótese seria a utilização de um plugin para o *software* Microsoft Visual Studio desenvolvido pela

VisualMicro que permite fazer o *debugging* do Arduino com recurso a *breakpoints*. No entanto, ambos os *softwares* são pagos, pelo que não se teve acesso aos mesmos. Outra forma possível de se fazer este tipo de *debugging* seria com o recurso a uma placa de desenvolvimento tal como a AVR Dragon, que utiliza as capacidades OCD (*On Chip Debug*) do microcontrolador Atmega328 do Arduino. No entanto, tão pouco se tinha acesso a uma solução do género. Consequentemente, teve que se proceder ao *debugging* da maneira tradicional, com recurso a *printf*s. Este método é bastante mais trabalhoso, mas acaba por ser igualmente eficaz. O *standard output* (para onde vão os dados provenientes dos *printfs*) do Arduino é, por design, o seu UART (*Universal Asynchronous Receiver/Transmitter*), que pode ser acedido através da ficha USB instalada na placa. Consequentemente, é possível utilizar um computador como recetor do *standard output*.

Felizmente, existe uma linguagem de programação, a linguagem *Processing*, que possui bibliotecas de raiz que facilitam a receção de dados do Arduino, bem como a visualização dos mesmos. Assim, desenvolveu-se um pequeno programa (255 linhas) em *Processing 2.1.1* que permitiu a receção e visualização do valor das variáveis que se quisessem monitorizar. Na Figura 3.16 podemos ver o programa em funcionamento, onde é visível, à esquerda a imagem que a câmara envia ao Arduino, bem como as posições e orientações possíveis do robô (a vermelho), relativamente à estação de docagem (a azul), à esquerda, numa vista de cima.

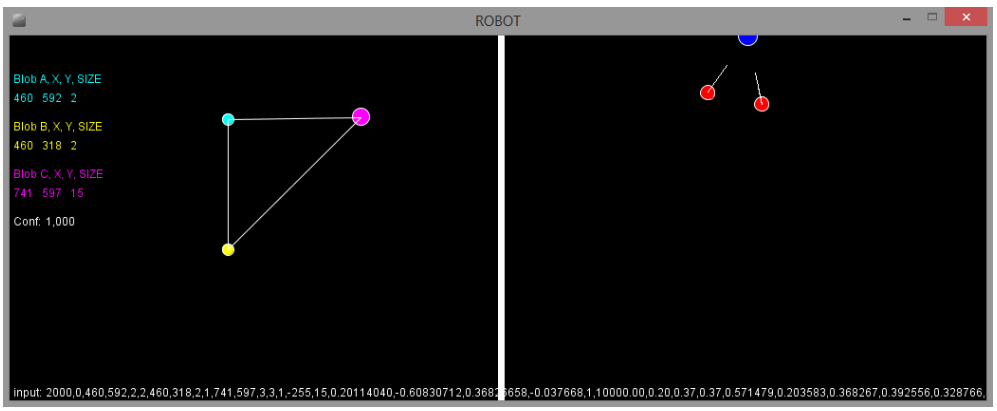


Figura 3.16. Programa desenvolvido em *Processing* para monitorizar o Arduino

3.4.2. Implementação do código no Arduino

O código foi escrito efetivamente em C++, utilizando-se somente algumas funções da linguagem Arduino. Numa primeira fase, o código estava a ser escrito em C. No entanto, dada a complexidade crescente deste, e aquando da adoção do modelo da

máquina de estado finito para a organização do algoritmo, optou-se pela linguagem C++ pelas suas capacidades de orientação a objetos. Para se tirar partido destas, foi necessário reorganizar o código existente e aprender a usar esta linguagem, mas a escrita posterior tornou-se desta maneira simples, tendo-se produzido código mais claro e estruturado.

3.4.2.1. Estruturação do código

Como já foi insinuado, o algoritmo de docagem foi implementado com recurso a uma máquina de estados finitos.

Nestes termos, o algoritmo explicado na secção 3.3 - Algoritmo de docagem deu origem a uma classe, chamada *mainStateMachine*, que contém todos os estados visíveis no diagrama dessa secção, sendo cada estado uma função com as instruções a realizar. Adicionalmente, as diferentes transições são guardadas num tipo de dados enumerado – basicamente, cada transição pode ser identificado por um inteiro. No início de cada função de estado, é feito um teste lógico em que se verifica se a transição corrente guardada em memória implica uma mudança de estado. Se não for o caso, o resto da função é executado, onde se decide se se deve ou não atribuir um valor à variável transição corrente, o que levará a uma transição de estado no ciclo seguinte.

Dado que uma boa maneira de implementar o controlo do movimento do robô também é com uma máquina de estado, criou-se outra máquina de estados, chamada *moveStateMachine,* que contém vários estados possíveis de movimento de mais baixo nível, tais como "alinhar com estação de docagem". Esta máquina está efetivamente "dentro" da anterior, funcionando dentro de cada um dos estados da mesma, formando uma máquina de estados hierárquica.

Posto isto, a função *Loop* do Arduino contém unicamente quatro linhas:

- Em primeiro lugar, são adquiridos os dados da câmara Pixart, e seguidamente são calculadas as coordenadas do robô e outras variáveis relevantes;
- a seguir, é executada o código correspondente ao estado corrente da instância da *mainStateMachine*, onde se determina o estado da *moveStateMachine* a executar, através de um apontador para a função de estado;
- depois, é passada à instância da *moveStateMachine* o seu novo estado;

- por fim, é corrido o código correspondente ao estado corrente da instância da *moveStateMachine*, onde o movimento do robô é efetivamente executado, igualmente através de um apontador para a função de estado.

O código encontra-se organizado em 13 ficheiros – 6 ficheiros *.h* (cabeçalhos de bibliotecas) e 6 ficheiros *.cpp* (código C++), e 1 ficheiro *.ino*, que é o ficheiro de entrada do programa, como é standard para projetos do Arduino. No total, temos 85 funções distribuídas por 8 classes, perfazendo 2015 linhas de código.

3.4.3. Ajustes ao hardware

Durante a execução do projeto, notaram-se certas limitações do hardware utilizado, que levaram a que fosse necessário fazer alterações ao equipamento que se tinha à disposição.

3.4.3.1. Plataforma dos LEDs

Em primeiro lugar, teve que se modificar a plataforma que alberga os LEDs original. De facto, a distância original entre os LEDs A e B era mais baixa, de aproximadamente 40 mm. Isto levava a que, devido à baixa resolução da câmara Pixart, esta não conseguisse distinguir entre os dois LEDs a distâncias superiores a 1,5 metros, juntando-os num único *blob*. Por forma a corrigir esta situação, teve que se construir uma segunda plataforma para albergar os LEDs, em que a distância entre os LEDs A e B fosse superior.

3.4.3.2. Palas anti-refletoras

Em segundo lugar, verificou-se que a radiação de infravermelhos é facilmente refletida por diferentes tipos de pavimento utilizados em espaços interiores (madeira, cerâmica, plástico). Estes reflexos são detetados pela câmara, o que faz com que sejam tidos em conta mais *blobs* do que os três devidos. Em princípio, isto seria simples de resolver se simplesmente programássemos o sensor para só ter em conta os três primeiros *blobs* a contar de cima para baixo (eliminando assim os refletidos). No entanto, uma vez que a câmara Pixart só consegue seguir quatro *blobs* de cada vez, quando lhe são apresentados mais do que esse número, escolhe quatro aleatórios desse grupo, enviando as coordenadas desses para o Arduino. Esta aleatoriedade não permite a criação de uma rotina

fidedigna que elimine os *blobs* indesejáveis, já que não há forma de saber se um *blob* é um reflexo ou não.

A solução prática acaba por ser a instalação de palas sob os LEDs na estação de docagem, como se ilustra na Figura 3.17.

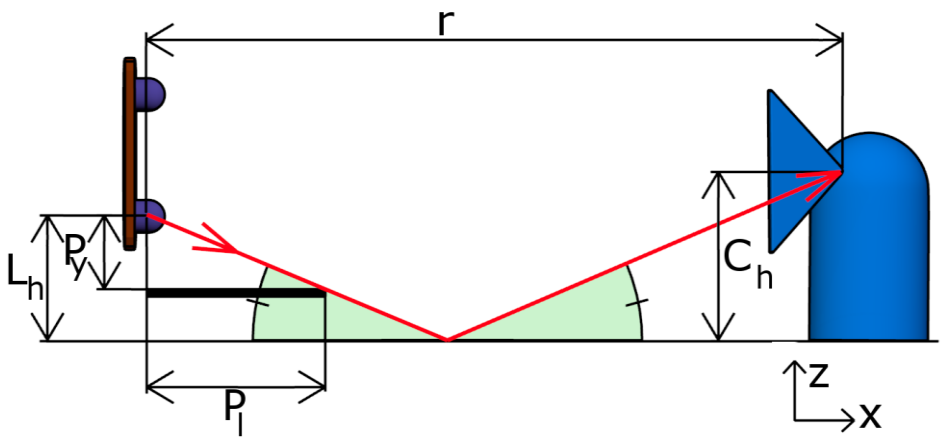


Figura 3.17. Esquema para a colocação de uma pala sob os LEDs

Em que L_h é a distância entre o centro do LED em questão e o chão, P_y é a distancia entre o centro do LED em questão e a pala, P_l é o comprimento da pala, r é a distância entre a estação de docagem e a câmara, e C_h é a distância entre a câmara e o chão.

Pensar-se-ia que o facto de o robô se aproximar automaticamente da estação por forma a distinguir melhor os LEDs sempre que a imagem é duvidosa seria suficiente para resolver este problema. No entanto, para um caso em que se utilizem LEDs de infravermelhos de ângulo largo (~40º), e se $Py = Ch = 20cm$, o que são valores próximos dos reais, o reflexo só desaparece da visão da câmara quando esta está a 54 centímetros da estação de docagem, o que é demasiado perto. No caso de existirem várias estações com configurações de LEDs diferentes próximas umas das outras, isto levaria a que potencialmente o robô andasse para a frente e para trás à volta de cada estação até que as distinguisse corretamente.

Desta forma, o comprimento da pala é dado por:

$$P_l = \frac{P_y \cdot r}{C_h + L_h} \tag{3.22}$$

Se se pretender que o robô consiga distinguir a estação a uma distância de 4 metros por exemplo, e para $C_h = L_h = 20cm$ e $P_y = 0.3cm$[14], o comprimento da pala será de $P_l = 3cm$, o que é um valor bastante razoável. Para a pala dos LEDs de cima, $P_l = 2.4cm$.

Relativamente ao material para o revestimento da pala, testaram-se algumas soluções. A melhor passou pelo revestimento da pala com tecido escuro (ganga), que absorve praticamente toda a radiação infravermelha emitida pelos LEDs.

3.4.4. Teste do funcionamento do robô

Quando chegou a fase de testar o código desenvolvido no robô, não se conseguiu estabelecer uma ligação mínima entre o Arduino e a placa do robô que alberga o ASIC. Depois de se testar com um osciloscópio e com outro computador, verificou-se que o Arduino estava a enviar a ordem corretamente. Tal dever-se-ia provavelmente ao facto de o Arduino enviar o sinal a 5V, mas mesmo depois de utilizar um divisor de voltagem para converter o sinal lógico para 3.3V, esperado pelo ASIC, não se obteve sucesso. Os responsáveis pelo robô em Budapeste já não tinham mais ideias sobre o assunto, e a disponibilidade deles era pouca, pelo que se acabou por não ter conseguido resolver este problema.

Estas dificuldades podem possivelmente ser explicadas pelo facto de o subcircuito, da placa que alberga o ASIC, destinado a comunicar com o Arduino, provavelmente nunca ter sido testado corretamente. De facto, inclusivamente foi necessário soldar os pinos do conector através do qual a interface com o Arduino é feita. Uma vez que não foi possível usar o robô real, optou-se pela simulação de um robô holonómico num programa de computador para testar o algoritmo.

3.4.5. Simulação do funcionamento do robô

Por forma a simular a docagem, teve que se escolher um simulador. Existem vários programas de computador destinados a simular o funcionamento de robôs. No entanto, a maior parte deles é bastante básico, muitos funcionando somente em 2D. Para

[14] $0.3cm$ é um pouco mais de metade do diâmetro de um LED comum de 5 mm, para que se tenha em consideração que a luz não vem exatamente do centro do deste. Se considerarmos a pala diretamente encostada ao LED, obtém-se aproximadamente esse valor.

que se conseguisse simular eficazmente o problema em questão, o simulador teria que ter as seguintes características:

- Capacidade de simular ambientes 3D, recorrendo a um motor físico 3D, renderizando a simulação em tempo real;
- interface relativamente intuitiva, por forma a não se perder muito tempo a aprender a trabalhar com o programa;
- possibilidade de utilização da linguagem C++ para a programação do controlo do robô. Muitos simuladores ou usam uma linguagem própria, ou só aceitam C;
- disponibilidade, ou seja, poder ser obtido da internet gratuitamente ou por tempo suficiente;
- pré-existência de modelos de robôs holonómicos no programa, bem como de sistemas de visão para o robô, já que a criação destes se afastaria muito do âmbito deste trabalho;
- existência de binários compatíveis com o sistema operativo Windows XP ou seguintes.

Seguindo estes critérios, de entre mais de trinta simuladores selecionaram-se dois: o Webots, desenvolvido pela empresa Cyberbotics, e o V-REP, desenvolvido pela Coppelia Robotics. Ambos os programas se conformam com as especificações acima, suportando ambos a utilização de código C++ sob a forma de scripts externos.

3.4.5.1. V-REP

Uma vez que o Webots apenas disponibiliza uma versão *trial* com duração de um mês, inicialmente optou-se pela utilização do V-REP, que tem uma versão gratuita para estudantes.

Assim, na sequência da familiarização com o funcionamento do programa, tentou-se utilizar código C++ neste, através da criação de uma biblioteca externa. A documentação relativa a este processo é pouca e esparsa, pelo que se avançou lentamente, por tentativa e erro, mesmo com a ajuda do fórum online de apoio do programa.

A biblioteca tem que ser criada com recurso a um compilador externo. Os desenvolvedores do programa disponibilizam duas maneiras de a criar, fornecendo projetos pré-configurados a ser utilizados quer com o ambiente de desenvolvimento MinGW

("Minimalist GNU for Windows"), quer com o Microsoft Visual Studio. Tentou-se usar o Visual Studio com recurso às funções de importação do programa Code::Blocks sem sucesso. Por alguma razão, tal não funcionou pelo que, se utilizou o projecto para o MinGW. No entanto, apesar de o projeto teoricamente já vir pré configurado, por defeito estava a selecionar outro compilador que não o MingGW, o que acabou por ser descoberto e corrigido. Posteriormente, o plugin estava a ser compilado em modo *debug* em vez de *release*, o que curiosamente evitava que o *plugin* sequer compilasse. Aqui, ao corrigir-se o modo de compilação, finalmente conseguiu-se compilar o *plugin* mais simples que o V-REP traz instalado, como forma de teste. No entanto, pretendia-se um *plugin* muito mais complexo, pelo que se optou pela modificação do código de um outro *plugin*, cujo funcionamento se aproximava mais dos nossos fins, até porque a documentação relativa à criação de um *plugin* de raiz é praticamente não existente. Infelizmente, apesar de aparentemente se ter conseguido compilar o *plugin* corretamente, este simplesmente não funcionava dentro do programa V-REP, tendo-se inclusive testado em dois computadores com sistemas operativos diferentes (Windows 8.1 e Windows XP Service Pack 3). Dado que os responsáveis pelo programa não conseguiram identificar qual seria o causador deste problema, optou-se pela utilização de outro simulador: o Webots.

3.4.5.2. Webots

No Webots, os problemas começaram mais cedo. De facto, a documentação existente relativa aos funcionamentos básicos do programa estava desatualizada. Isto, aliado ao facto de por alguma razão as dependências não estarem a ser extraídas da internet (procedimento que não existia nas versões anteriores do programa), levou a que este problema se atribuísse a problemas do próprio computador. Depois de se identificar o problema, tentaram-se várias maneiras de o resolver sem sucesso. Finalmente, sem razão aparente, ao fim de uma nova instalação, o programa finalmente fez o *download* das dependências necessárias. Assim, por fim se pode começar a trabalhar com o programa.

No entanto, a compilação do *plugin*, para ser possível usar código C++, de novo levantou problemas, devido a documentação deficiente. No Webots, o robô holonómico padrão deve ser controlado com recurso a bibliotecas escritas em C por defeito. Dado que se pretendia criar um controlador em C++, este teria que utilizar a API de C++ do Webots. Para que este conseguisse usar bibliotecas em C, foi necessário fazer ajustes a estas, bem como alterar o *Makefile* do projeto, tendo havido inclusive problemas

com o facto de as variáveis que definiam os caminhos dos ficheiros no *Makefile* por alguma razão não funcionarem, tendo que ser substituídas.

As deficiências da documentação existente até levaram a que as idiossincrasias da invocação correta de um simples *printf*, para fins de *debug*, levantassem problemas.

Finalmente, conseguiu-se compilar corretamente um controlador de teste. Nesta fase, procedeu-se à modificação do robô holonómico padrão que o Webots traz, por forma a colocar-lhe uma câmara, que faria a função da câmara Pixart no robô real. No entanto, ao contrário do V-REP que já possui uma câmara que identifica *blobs*, o Webots só possui câmaras normais. Por este fato, foi preciso proceder à criação de código de identificação de *blobs*, algo que inicialmente era feito pela câmara Pixart.

Para tal, recorreu-se a um método de extração de *blobs* um *blob* de cada vez, executando-se uma busca em profundidade. Entretanto, expiraram os dois meses de utilização do Webots (em dois computadores). Tentou-se utilizar um terceiro computador, mas neste novamente não se conseguiu proceder à compilação do *plugin*.

4. RESULTADOS E DISCUSSÃO

O código relativo ao cálculo do posicionamento do robô com base na visualização da câmara no Arduino foi corretamente implementado. De facto, testou-se o mesmo com recurso ao programa desenvolvido em *Processing*, visível na Figura 3.16, e os resultados foram bastante satisfatórios, tendo-se conseguido calcular a posição e orientação da câmara com uma precisão de aproximadamente 1 centímetro a 1 metro de distância. Isto prova que o sistema simples e de baixo custo desenvolvido é uma solução viável para problemas do género.

Tendo-se desenvolvido o algoritmo de docagem e feito a sua implementação em C++, disponível em anexo, infelizmente não se conseguiu proceder ao seu teste, pelas razões explicadas na metodologia. No entanto, salvo eventuais pequenos ajustes necessários no próprio algoritmo, que só conseguiriam ser inferidos através de testes práticos, o código só necessitaria de ser "limpo" de *bugs*, sendo este processo facilitado pela forma modular como foi escrito. Desta forma, pretender-se-ia aqui apresentar uma análise odométrica do robô, que deveria revelar resultados semelhantes ao padrão de movimentos visível na Figura 3.14. Adicionalmente, uma análise estatística da eficiência do processo de docagem (tempo despendido, número de tentativas, etc.) seria útil para não só validar o algoritmo, como também para ajudar a melhorá-lo. Espera-se que a restante equipa e futuros membros levem a cabo estas tarefas.

Em anexo podemos encontrar os fluxogramas de cada estado do robô, que permitem desvendar o funcionamento do robô mais profundamente. No entanto, não é possível colocar-se todo o trabalho desenvolvido nesses diagramas, pelo que se disponibiliza igualmente a totalidade do código escrito. Dado que a colocação deste em anexo faria com que se ultrapassasse o número de páginas limite para a dissertação, o código está disponível *online* em:

https://www.dropbox.com/sh/71eklilxz15pivo/AADw7KJog6oLVLJm6Vuc4fWya?dl=0.

5. CONCLUSÕES

5.1. Considerações finais

Com este trabalho conseguiu-se cumprir os objetivos propostos no que diz respeito ao cálculo da posição de um robô relativamente a uma estação de docagem, com base num mecanismo de reconstrução espacial simples e de baixo custo. Além disso, desenvolveu-se um algoritmo de docagem robusto, capaz de distinguir entre várias estações, tendo-se feito a sua implementação em C++.

Este sistema pode ser aplicado em qualquer robô holonómico que funcione num pavimento plano (ou irregular), o que englobará a maior parte das situações em que seja necessário que um robô faça a docagem. Adicionalmente, dada a simplicidade do sistema desenvolvido, o seu custo é bastante baixo, além de que quaisquer objetivos de melhoria ou de manutenção se tornam mais simples de serem concretizados.

O facto de o código ter sido escrito em C++, uma linguagem comum, aliado à maneira modular como foi implementado, faz com que futuras alterações e desenvolvimentos sobre ele se tornem mais fáceis de implementar.

5.2. Limitações

Infelizmente, não foi possível testar o código escrito relativo ao algoritmo em si. Não se tendo conseguido comunicar com o robô real, procedeu-se à simulação em *software* da docagem do robô. No entanto, apesar de se terem testado dois simuladores diferentes, ambos comerciais, cada um apresentou diferentes problemas, tanto devido a *bugs*, ou comportamentos incompreensíveis dos próprios programas, como a documentação bastante deficiente. Estes efeitos conjugados contribuíram para dificultar o trabalho e prolongar o seu desenvolvimento. De facto, seria de esperar que a simulação de um robô com um programa escrito em C++ se tratasse de um terreno mais maduro, estando nós no ano de 2014.

5.3. Recomendações para trabalhos futuros

Dadas as limitações encontradas, o primeiro desenvolvimento a seguir seria fazer o teste do algoritmo de docagem, quer num robô holonómico real, quer num simulador. Isto é, validar não só o princípio do algoritmo, como também fazer o eventual *debug* do código escrito (qualquer código minimamente complicado tem inevitavelmente *bugs*). Por fim, uma análise da eficiência do algoritmo (tempo de docagem, número de tentativas, etc) seria também indispensável.

Apesar de o trabalho desenvolvido nesta dissertação ter como objetivo a aplicação em robôs holonómicos, os princípios aqui apresentados podem facilmente ser aplicados a outras situações. Nomeadamente, a outros tipos de robô, e a bastantes situações em que seja necessário proceder a um acoplamento mecânico existindo grande liberdade de movimentos, tal como pode acontecer em linhas de montagem industrial ou construção civil, por exemplo. Desta forma, seria interessante ver este tipo de sistemas simples aplicados a situações onde métodos mais exóticos, por uma razão ou outra, não se tomem como apropriados.

Nesta linha de pensamento, um desenvolvimento interessante seria a criação de uma série de sistemas compactos de baixo custo praticamente prontos a serem usados para resolver este tipo de problemas.

REFERÊNCIAS BIBLIOGRÁFICAS

Arduino Libraries. (s.d.). Obtido em 10 de 06 de 2014, de http://arduino.cc/en/Reference/Libraries

Arduino. (s.d.). *What is Arduino*. Obtido em 20 de 11 de 2014, de Arduino: http://arduino.cc/en/guide/introduction

ATmel Corporation. (2008). *AVR910: In-System Programming.* Obtido em 30 de 12 de 2014, de Atmel: http://www.atmel.com/images/doc0943.pdf

Baba, M., Asada, N., Oda, A., & Migita, T. (s.d.). *A Thin Lens Based Camera Model for Depth Estimation from Defocus and Translation by Zooming.* Obtido em 8 de 1 de 2015, de CiteseerX: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.3.7697&rep=rep1&type=pdf

Beal, V. (2015). *Objet Oriented Programming.* Obtido em 6 de 1 de 2015, de Webopedia: http://www.webopedia.com/TERM/O/object_oriented_programming_OOP.html

Bluebotics. (s.d.). *Shrimp - On Rough Terrain*. Obtido de Bluebotics - Mobile Robots at Your Service: http://www.bluebotics.com/mobile-robotics/shrimp-3/

Böttcher, S. (s.d.). *Principles of robot locomotion.* Obtido em 10 de 10 de 2014, de http://www2.cs.siu.edu/~hexmoor/classes/CS404-S09/RobotLocomotion.pdf

Bruckstein, A., Holt, R. J., Katsman, I., & Rivlin, E. (2005). Head Movements for Depth Perception: Praying Mantis versus Pigeon. *Autonomous Robots*, pp. 21-42. Obtido em 8 de 1 de 2015, de Technion - Insrael Institute of Technology - Computer Science Department: http://www.cs.technion.ac.il/~freddy/papers/Paper113.pdf

Carter, E. F. (1 de Fevereiro de 1997). *Robots and Finite-State Machines.* Obtido em 20 de 01 de 2015, de Dr. Dobb's: http://www.drdobbs.com/parallel/robots-and-finite-state-machines/184410132

Cassinis, R., Tampalini, F., Bartolini, P., & Fedrigotti, R. (s.d.). *Docking and Charging System for Autonomous Mobile.* University of Brescia, Department of Electronics for Automation, Brescia. Obtido de http://www.ing.unibs.it/~arl/docs/papers/05_008.pdf

Clipart Panda. (s.d.). *Connection Clipart*. Obtido em 10 de 2 de 2015, de Clipart Panda: http://www.clipartpanda.com/categories/connection-20clipart

Dreamstime. (2015). *Computer Chip*. Obtido em 15 de 2 de 2015, de Dreamstime: http://www.dreamstime.com/photos-images/chip.html

Embedded Systems Portal. (s.d.). Obtido em 15 de 2 de 2015, de Embedded Systems Portal: http://www.embedded-systems-portal.com/CTB/ASIC,1005.html

Eppstein, D. (15 de Fevereiro de 1996). *Breadth first search and depth first search.* Obtido em 20 de 12 de 2014, de Donald Bren - School of Information and Computer Science: http://www.ics.uci.edu/~eppstein/161/960215.html

Etessami, K. (s.d.). *Discrete Mathematics & Mathematical Reasoning Chapter 10: Graphs.* Obtido em 20 de 12 de 2014, de University of Endiburgh: http://www.inf.ed.ac.uk/teaching/courses/dmmr/slides/13-14/Ch10.pdf

Fleet, D., & Hertzman, A. (2005). *Camera Models - Lectures Notes for CSC418 / CSCD18 / CSC2504.* Obtido em 23 de 1 de 2015, de Toronto University - Computer Science Department: http://www.cs.toronto.edu/~jepson/csc2503/readings/Camera.pdf

Free On-Line Dictionary of Computing. (2014). Obtido em 15 de 6 de 2014, de http://foldoc.org/

Fusiello, A. (s.d.). *Elements of Geometric Computer Vision*. Obtido de http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/FUSIELLO4/tutorial.html#x1-30003

Grosberg, M. (2003). *Implementing Efficient State Machines.* Obtido em 24 de 10 de 2014, de http://www.conman.org/projects/essays/states.html

Hlaváč, V. (s.d.). *Path planning in Robotics.* Praga: Czech Technical University in Prague. Obtido de http://cmp.felk.cvut.cz/~hlavac/TeachPresEn/55IntelligentRobotics/090PathPlanningInRobotics.pdf

Hobley, S. (22 de Fevereiro de 2009). *Pixart Sensor and Arduino*. Obtido em 27 de Fevereiro de 2014, de http://www.stephenhobley.com/blog/2009/02/22/pixart-sensor-and-arduino/

Hoerner, Sulger, & Max Planck Institute. (s.d.). *Nanokhod Project*. Obtido em 21 de 10 de 2014, de http://asl.epfl.ch/research/projects/

Honda. (2015). Asimo. Obtido de http://asimo.honda.com/

Huang, M. (s.d.). *PC-Based Line-Scan Imaging Systems.* Obtido em 15 de Julho de 2014, de http://www.adlinktech.com/industrial_automation/PC-Based_Line-Scan.php?source=

International Federation of Robotics. (2014). *Service Robot Statistics*. Obtido de International Federation of Robotics.

International Federation of Robotics. (s.d.). *Service Robots*. Obtido em 30 de 1 de 2015, de International Federation of Robotics: http://www.ifr.org/service-robots/

James, M. (2014). *Finite State Machines: grammar and machines*. Obtido em 24 de 10 de 2014, de http://www.i-programmer.info/babbages-bag/223-finite-state-machines.html?start=1

Jones, N. (11 de Janeiro de 2010). *A tutorial on lookup tables in C.* Obtido em 10 de 1 de 2015, de Embedded Gurus: http://embeddedgurus.com/stack-overflow/2010/01/a-tutorial-on-lookup-tables-in-c/

Kako. (Janeiro de 2007). *Infrared wiimote sensor analysis*. Obtido em 27 de Fevereiro de 2014, de www.kako.com/neta/2007-001/2007-001.html

KDE. (20 de 12 de 2014). *Fundamentos do UML.* Obtido de KDE Docs: https://docs.kde.org/stable/pt_BR/kdesdk/umbrello/uml-basics.html

Kim, K. H., Choi, H. D., Soon, S., Lee, K. W., Ryu, H. S., Woo, C. K., & Kwak, Y. K. (2005). Development of Docking System for Mobile Robots Using Cheap Infrared Sensors. *1st International Conference on Sensing Technology.* Palmerston North, New Zealand.

Kotoku, T. (2006). Robot Middleware and its Standardization in OMG. *International Conference on Intelligent Robots and Systems (IROS'06)* . Beijing.

Kumagai, M., & Ochiai, T. (2009). *Development of a Robot Balanced on a Ball - First Report, Implementation of the Robot and Basic Control* -. Tohoku Gakuin University. Obtido de http://www.fujipress.jp/finder/xslt.php?mode=present&inputfile=ROBOT002200030013.xml

Lathrop, R. (s.d.). *Principles of Photogrammetry: Stereoscopic Parallax.* Obtido em 17 de 1 de 2015, de Rutgers - The State University of New Jersey: http://www.crssa.rutgers.edu/courses/airphoto/airphoto7_files/frame.htm

Lindeberg, T. (1993). Detecting Salient Blob-Like Image Strucutres and Their Scales with a Scale-Space Primal Sketch: A method for Focus-of-Attention. *International Journal of Computer Vision, 11*(3), 283-318. Obtido de http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.140.5212&rep=rep1&type=pdf

Lipták, B. G. (2005). *Instrument Engineers' Handbook - Process Control and Optimization.* CRC Press.

Lui, H. J., Hansen, V. D., & Kriegstein, R. A. (s.d.). *Development and Evolution of the Human Neocortex.* San Francisco.

Mann, R. (s.d.). How to Program an 8-bit Microcontroller using C language. *Atmel Applications Journal*, pp. 13-16. Obtido em 15 de 1 de 2015, de http://www.atmel.com/images/avr_3_04.pdf

Marinha do Brasil. (s.d.). Auxílios visuais à navegação: faróis, faroletes, barcas-faróis, boías, balizas e sistemas de balizamento. Em *Navegação costeira, estimada, e em águas restritas.* Obtido em 1 de 12 de 2014, de https://www.mar.mil.br/dhn/bhmn/download/cap13.pdf

Mattoccia, S. (12 de Janeiro de 2013). *Stereo Vision: Algorithms and Applications.* Obtido de University of Bologna - Department of Computer Science: http://vision.deis.unibo.it/~smatt/Seminars/StereoVision.pdf

Merriam-Webster. (2015). *Triangulation.* Obtido em 21 de 1 de 2015, de Merriam-Webster: http://www.merriam-webster.com/dictionary/triangulation

MTA-ELTE. (s.d.). *Etho-robotics.* Obtido em 4 de 2 de 2015, de MTA-ELTE Comparative Ethology Group: http://mta-etologia.elte.hu/?p=2&s=1

National Instruments. (21 de Dezembro de 2011). *Introdução à tecnologia FPGA.* Obtido em 28 de 12 de 2014, de National Instruments: http://www.ni.com/white-paper/6984/pt/

Natschläger, T., & Maass, W. (2012). Spiking neurons and the induction of finite state machines. *Theoretical Computer Science - Natural computing, 287*(1), 251-265.

Nevatia, R. (1975). *Depth Measurement by Motion Stereo.* University of Southern California, Image Procesing Institute, Los Angeles. Obtido em 20 de 1 de 2015, de http://iris.usc.edu/Outlines/papers/1970/nevatia-cgip-76.pdf

Ng, R., Levoy, M., Brédif, M., Duval, G., Horowitz, M., & Hanrahan, P. (2005). *Light Field Photographt with a Hand-held Plenoptic Camera.* Obtido em 23 de 1 de 2015, de https://graphics.stanford.edu/papers/lfcamera/lfcamera-150dpi.pdf

Nikon. (2015). *Understanding Maximum Aperture.* Obtido em 29 de 1 de 2015, de Nikon USA: http://www.nikonusa.com/en/Learn-And-Explore/Article/g3cu6o1r/understanding-maximum-aperture.html

NonGNU. (s.d.). Obtido em 21 de 10 de 2014, de AVR Libc FAQ: http://www.nongnu.org/avr-libc/user-manual/FAQ.html

Nutffield Foundation. (2014). *From Pinhole Camera to Lens Camera.* Obtido em 16 de 3 de 2014, de Nutffield Foundation: http://www.nuffieldfoundation.org/practical-physics/pinhole-camera-lens-camera

Oh, S., Zelinsky, A., & Taylor, K. (s.d.). Autonomous battery recharging for indoor mobile robots. Obtido de

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.9145&rep=rep1&type=pdf

Oswald, M., & Klodt, M. (s.d.). *Image-based 3D Reconstruction.* Obtido de Technische Universität München - Computer Vision Group: http://vision.in.tum.de/research/image-based_3d_reconstruction

Phan, K. (9 de Julho de 2013). *From Mozi to Polaroid: A Concise History of Photography.* Obtido em 20 de 1 de 2015, de Optical Collimator: http://opticalcollimator.com/2013/07/09/from-mozi-to-polaroid-a-concise-history-of-photography/

Plaxen, B., & Desalu, A. (s.d.). *Infrared Motion Tracking.* Obtido em 20 de 2 de 2015, de cmuhumanoids.

Podnar, W. G. (1985). *Uranus*. Obtido em 20 de 1 de 2015, de Carnegie Mellon University: School of Computer Science: http://www.cs.cmu.edu/~gwp/robots/Uranus.html

Podnar, W. G. (s.d.). *The Uranus Mobile Robot.* Obtido em 21 de 10 de 2014, de http://www.cs.cmu.edu/~gwp/papers/Podnar_URANUS_1085.pdf;

Questa, P., & Sandini, G. (1996). Time to contact computation with a space-variant ret-ina-line c-mos sensor. *Proceedings of the International Conference on Intelligent Robots and Systems.* Osaka, Japan.

Radioshack. (2014). *Single Board Computers Overview.* Obtido em 27 de 12 de 2014, de Radioshack Online: http://uk.rs-online.com/web/generalDisplay.html?id=infozone&file=electronics/single-board-computers

Ragab, H. (s.d.). *Draw a Brain*. Obtido em 15 de 2 de 2015, de Learn Drawing: http://paintingxxx.blogspot.pt/2013/05/draw-brain.html

Robotix. (2012). *Mecanum Wheel.* Obtido em 15 de 2 de 2015, de Robotix - Technology Robotix Society: http://robotix.in/rbtx13/tutorials/category/mechanical/wheelstut

Roosta, R. (s.d.). *ASICs VS PLDs.* Obtido em 29 de 12 de 2014, de California State University Northridge: http://www.csun.edu/edaasic/roosta/ECE595_Chap5.pdf

Rosier, L. E., & Gouda, M. G. (1983). *Deciding Progress for a Class of Communicating Finite State Machines.* Austin: University of Texas at Austin.

Sherwood, T., & Calder, B. (2001). Automated design of finite state machine predictors for customized processors. *Computer Architecture, 2001. Proceedings. 28th Annual International Symposium on* . Goteborg.

Sigal, L. (2008). *Camera Models Part 1, Computer Graphics Lecture Notes, CSCD18.* Obtido de Brown University: http://cs.brown.edu/~ls/teaching08/LS07_Cameras_Part1.pdf

Silverman, M. N. (2002). *Staying alive: a docking station for autonomous robot recharging.* Obtido de http://robotics.usc.edu/publications/media/uploads/pubs/237.pdf

Spice, B. (2013). *Press Release: Carnegie Mellon Snake Robot Winds Its Way Through Pipes, Vessels of Nuclear Power Plant.* Obtido em 10 de 1 de 2015, de Carnegie Mellon University - Carnegie Mellon News: http://www.cmu.edu/news/stories/archives/2013/july/july9_snakerobot.html

Sturm, P., Ramalingam, S., Tardif, J.-P., Gasparini, S., & Barreto, J. (Janeiro de 2011). *Camera Models and Fundamental Concepts Used in Geometric Computer Vision.* Obtido em 19 de 1 de 2015, de MITSUBISHI ELECTRIC RESEARCH LABORATORIES: http://www.merl.com/publications/docs/TR2011-069.pdf

Tadakuma, K. (2006). *Spherical Omnidirectional Wheel: "Omni-Ball"*. Obtido em 20 de 1 de 2015, de Kaneko Higashimori Laboratory, Osaka University: http://www-hh.mech.eng.osaka-u.ac.jp/robotics/Omni-Ball_e.html

Tajti, F., Szayer, G., Kovács, B., & Korondi, P. (s.d.). *Robot base with holonomic drive.* Budapest, Hungary.

Turek, F. D. (Junho de 2011). *Machine Vision Fundamentals, How to Make Robots See.* Obtido em 15 de Julho de 2014, de www.techbriefs.com/privacy-footer-69/10531

Ulrich, M., Wiedemann, C., & Steger, C. (2008). *Machine Vision Algorithms and Applications.* Wiley.

unclejoe. (2011). *BOTZ Gateway Design Teaser.* Obtido em 30 de 9 de 2014, de VEX Robotics: http://www.vexforum.com/showthread.php?t=38089

Wanner, S., Fehr, J., & Jähne, B. (s.d.). *Generating EPI Representations of 4D Light Fields with a Single Lens Focused Plenoptic Camera.* Heidelberg: Heidelberg Collaboratory for Image Processing. Obtido em 21 de 1 de 2015, de http://hci.iwr.uni-heidelberg.de/HCI/Research/LightField/paper/wfj2011_isvc.pdf

Whitaker, J. C. (2006). *Microelectronics.* CRC Press. Obtido de https://books.google.pt/books?id=n-fh3wIPZbkC&pg=SA7-PA15&lpg=SA7-PA15&dq=discreet+logic+gates+complicated&source=bl&ots=QHlW0VN2_R&sig=e-m0QN74SrJvdKxYTdXX9w2uk3c&hl=pt-PT&sa=X&ei=BVbaVNDEGouzUbnYgKgB&ved=0CGQQ6AEwCA#v=onepage&q=discreet%20logic%20gates%

Wiibrew. (s.d.). Obtido em 14 de Abril de 2014, de http://wiibrew.org/

WiiMote Physics. (2010). Obtido em 10 de 06 de 2014, de http://wiiphysics.site88.net/physics.html

Wilson, A. (Abril de 2011). *The Infrared Choice.* Obtido em 17 de Julho de 2014, de http://www.vision-systems.com/articles/print/volume-16/issue-4/features/the-infrared-choice.html

Wu, K., Otoo, E., & Suzuki, K. (s.d.). *Optimizing Two-Pass Connected-Component Labeling Algorithms.*

ANEXO A - FLUXOGRAMAS DAS MÁQUINAS DE ESTADO

A.1. Fluxogramas da máquina de estados principal

A seguir apresentam-se os fluxogramas de cada função correspondente aos estados da máquina de estados principal explicitada na Figura 3.15.

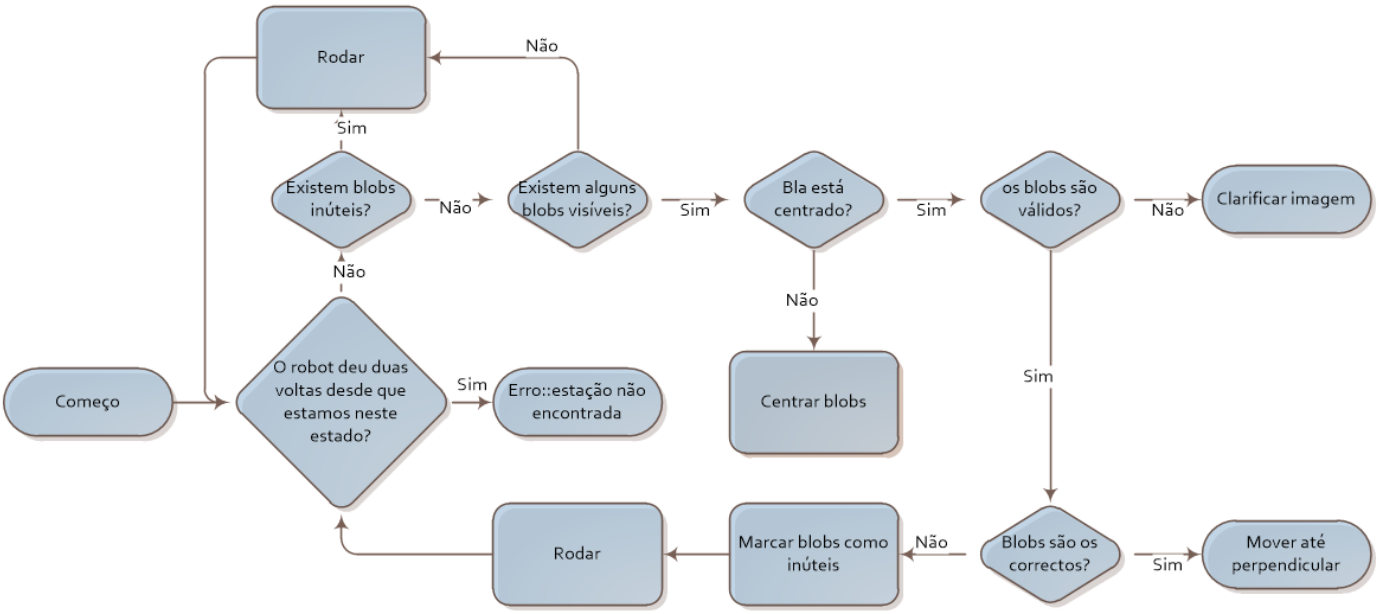


Figura A.1. Fluxograma do estado "Rodar até que blobs estejam encontrados e centrados"

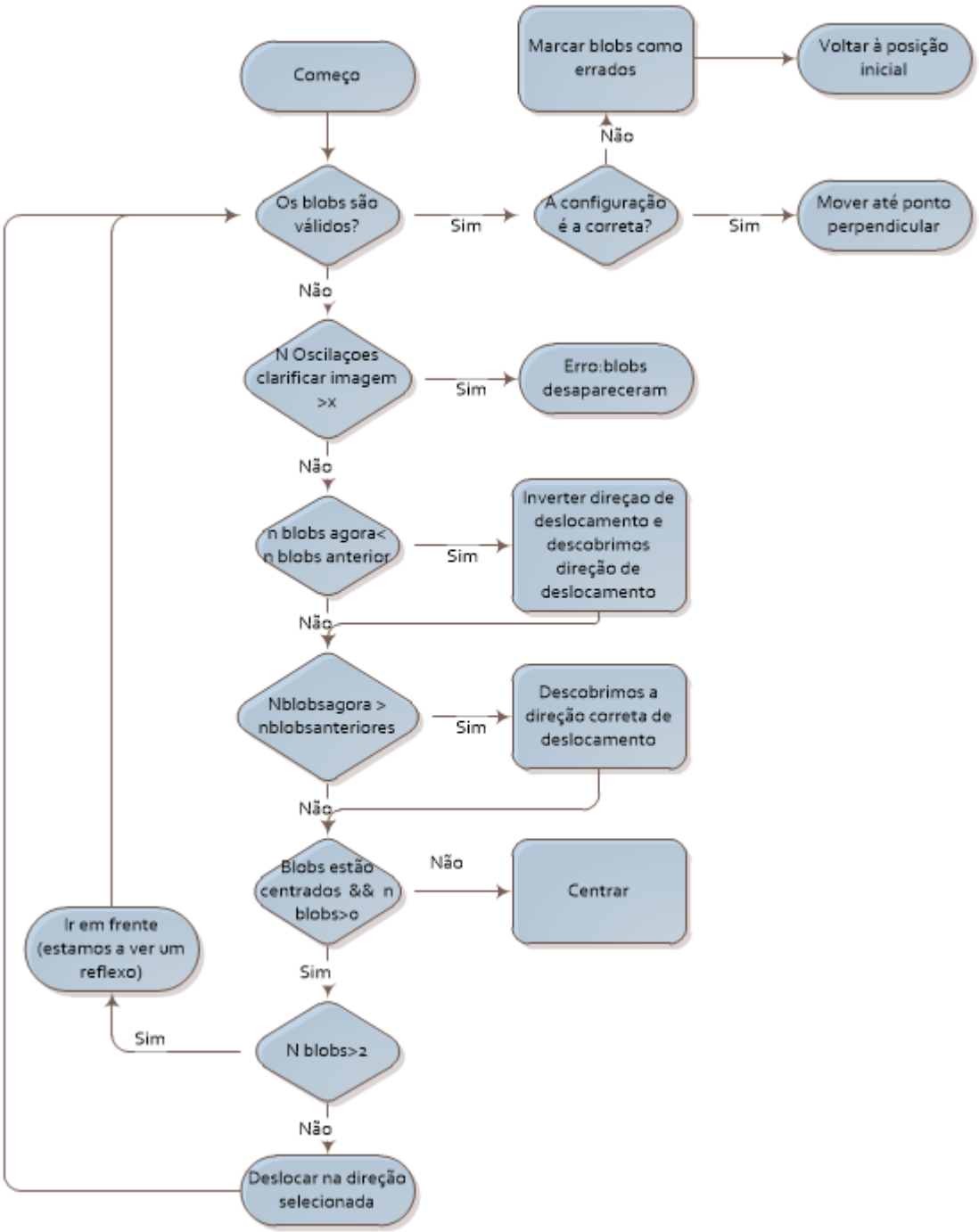


Figura A.2. Fluxograma do estado “Clarificar Imagem”

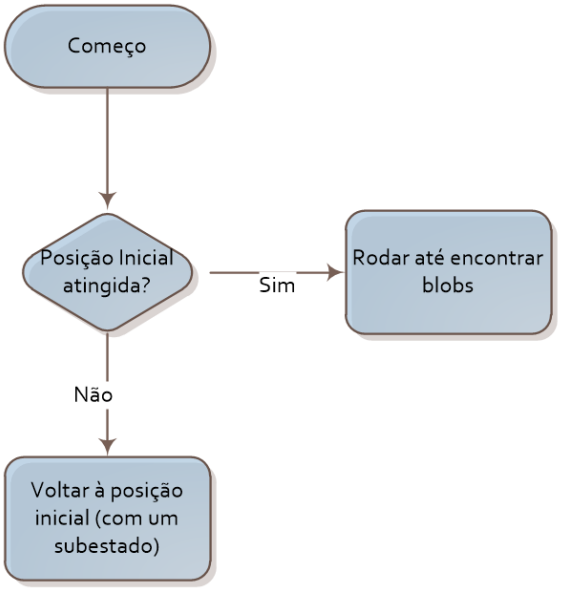


Figura A.3. Fluxograma do estado "Voltar à posição inicial"

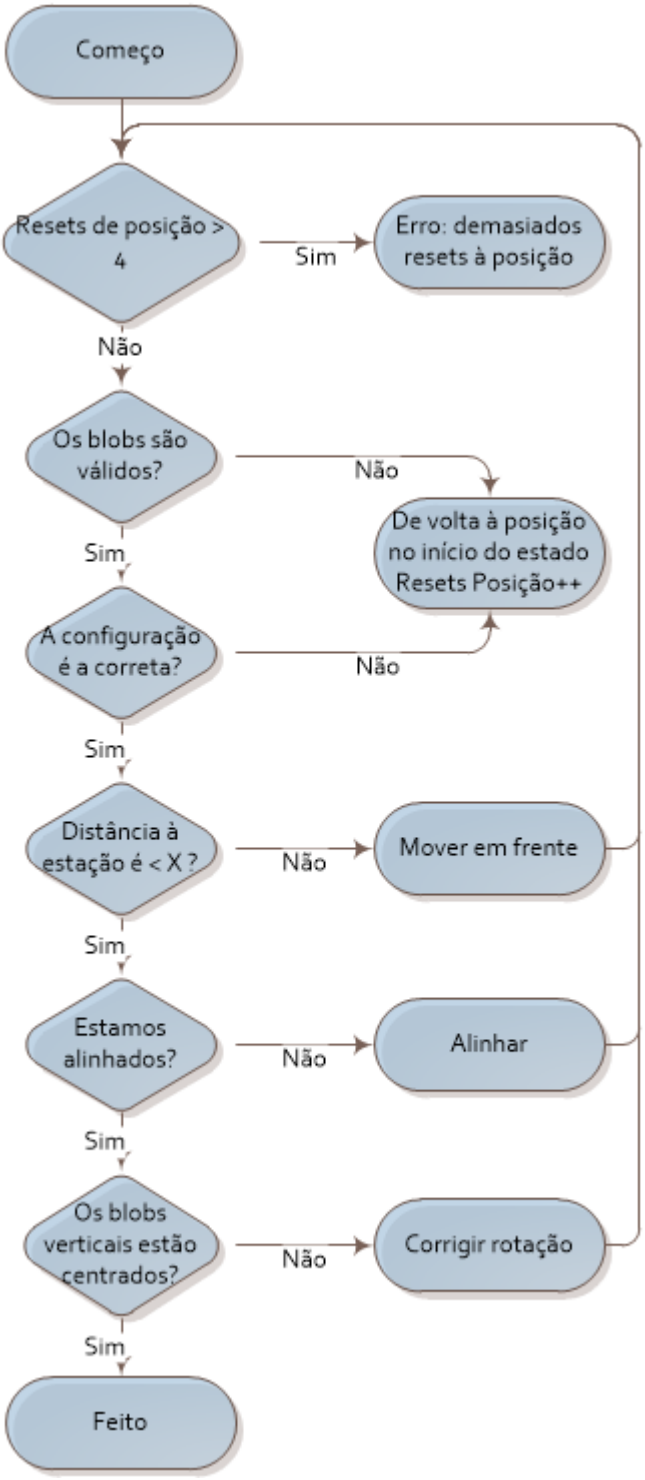


Figura A.4. Fluxograma do estado "Mover até perpendicular"

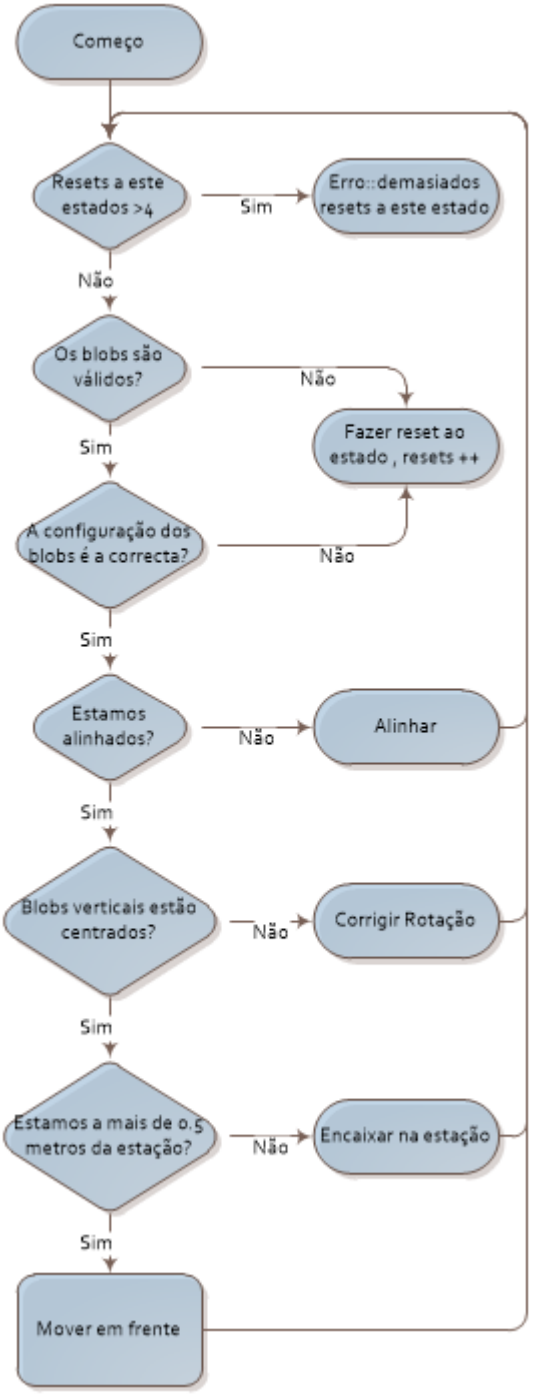


Figura A.5. Fluxograma do estado “Mover em direção à estação”

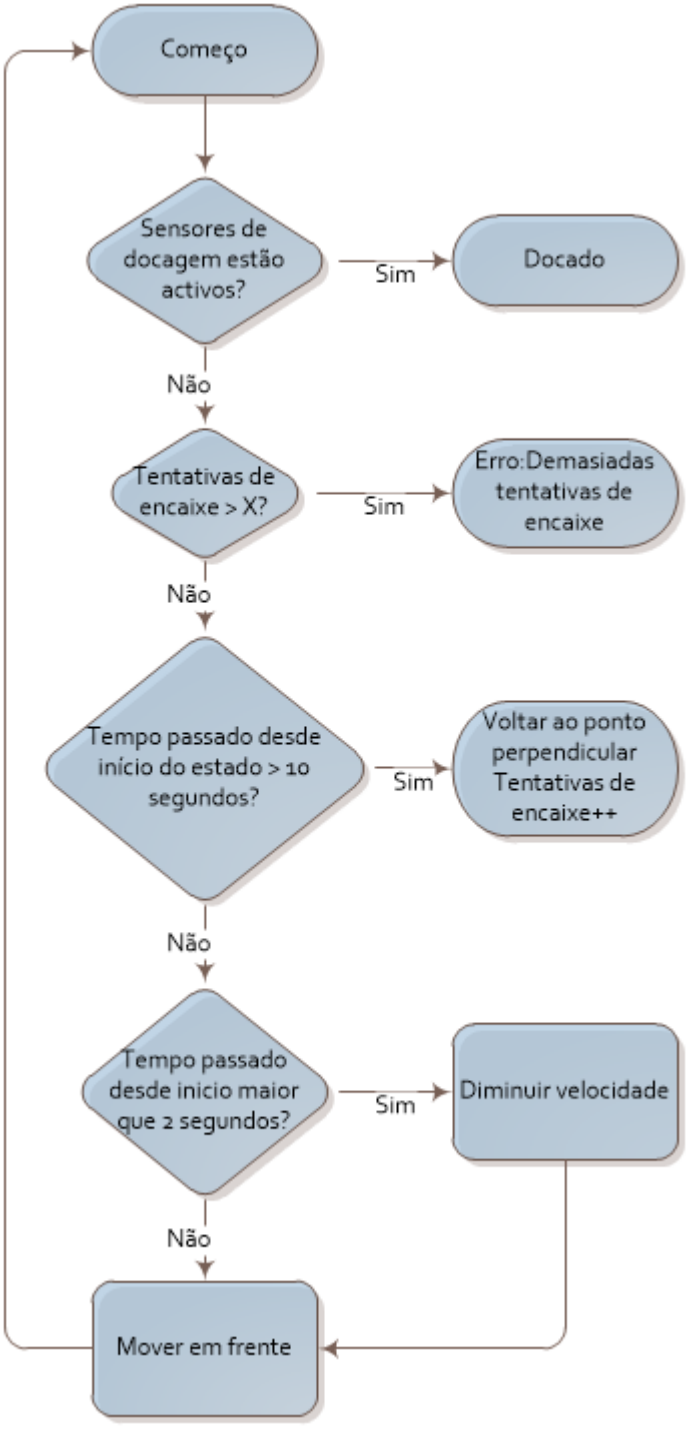


Figura A.6. Fluxograma do estado “Penetrar na estação”

A.2. Fluxogramas da máquina de estados de movimento

Agora, explicitar-se-ão os diagramas correspondentes ao subestados de movimento, que as funções principais chama variadas vezes:

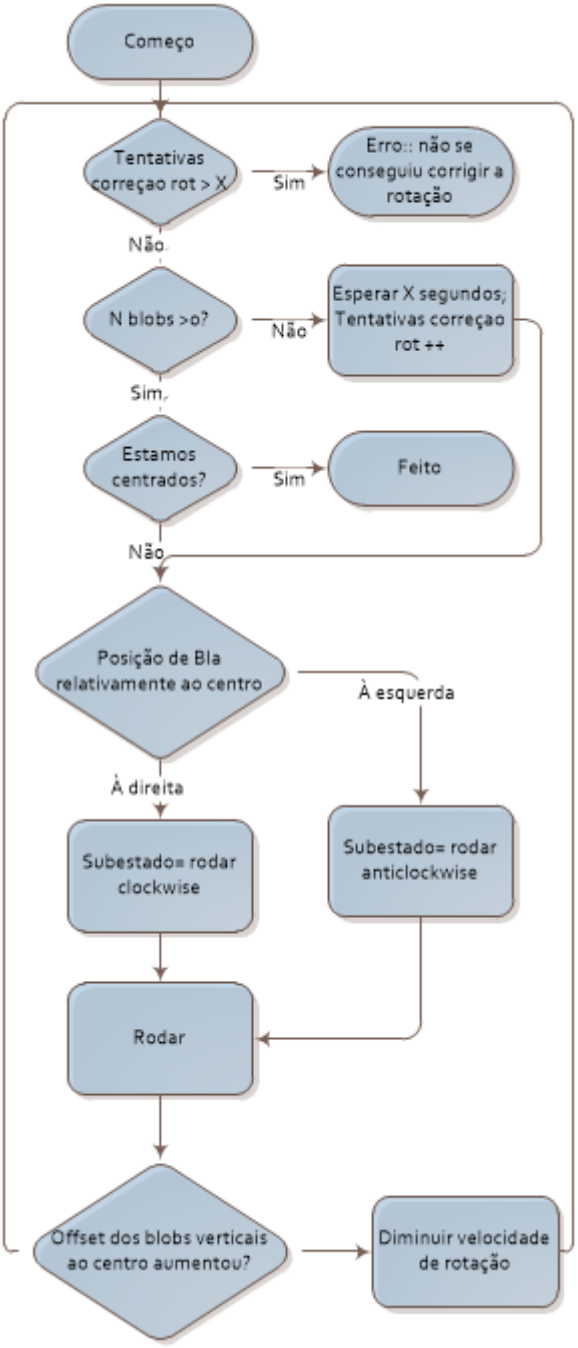


Figura A.7. Fluxograma do subestado “Corrigir rotação”

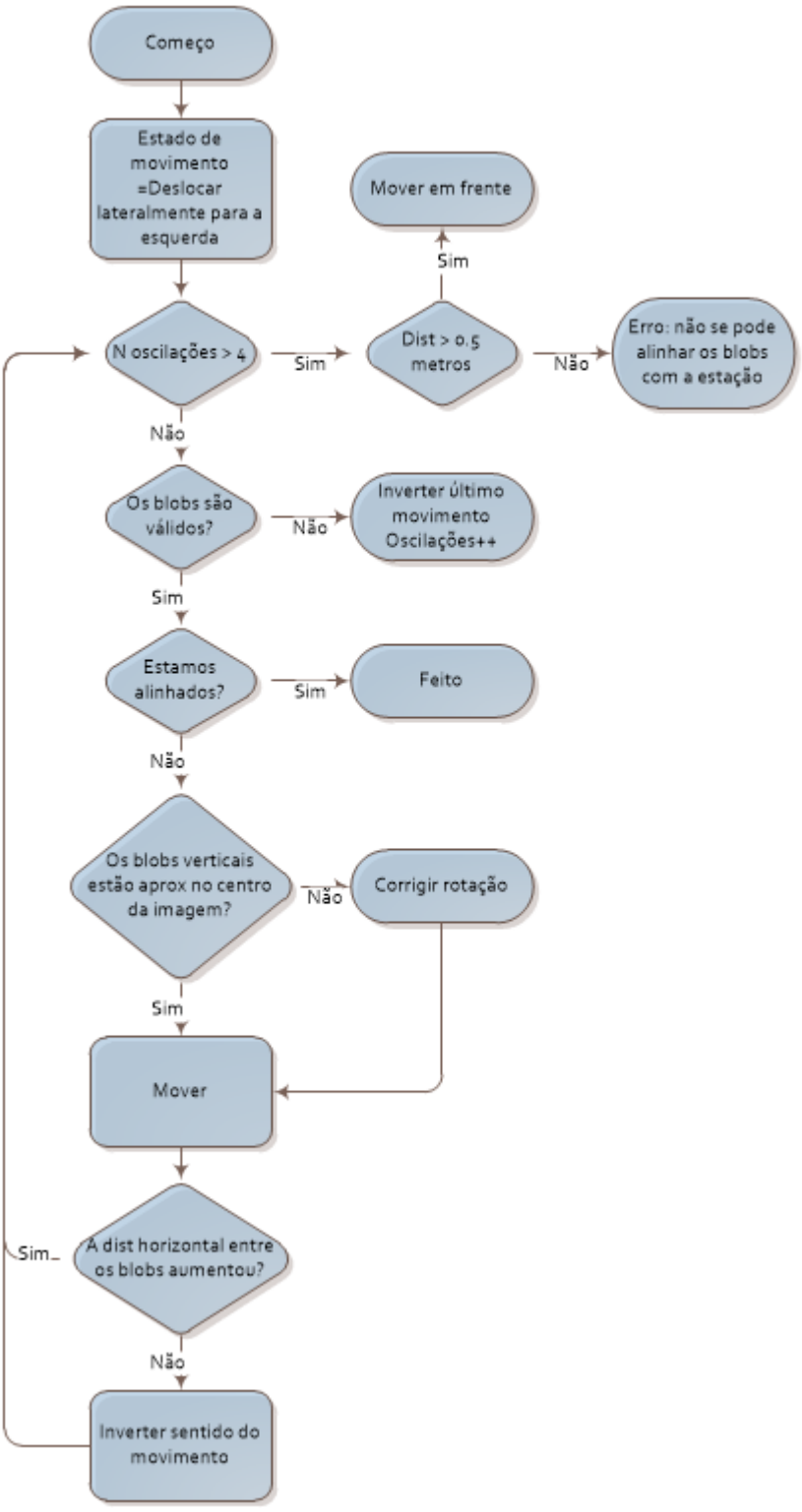


Figura A.8. Fluxograma do subestado "Alinhar com a estação"

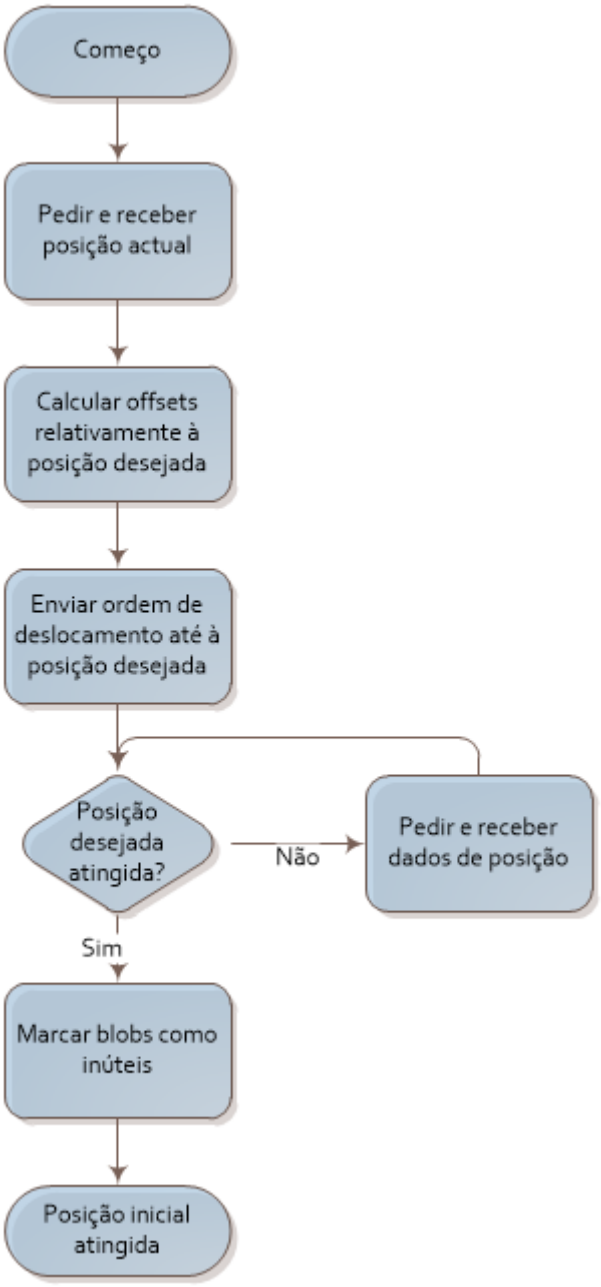


Figura A.9. Fluxograma do subestado "Voltar à posição inicial"