ISEL — Instituto Superior de Engenharia de Lisboa

DEETC — Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

MEIM - Mestrado em Engenharia Informática e Multimédia

Navegação autónoma assistida por visão

Fábio Domingues Vieira
(Licenciado em Engenharia Informática e Multimédia)

Trabalho Final de Mestrado para obtenção de grau de Mestre em Engenharia Informática e Multimédia

Orientadores

Professor Doutor	Pedro Miguel Torres Mendes Jorge Instituto Superior de Engenharia de Lisboa – ISEL
Professor Doutor	Pedro Daniel Dinis Teodoro Escola Superior Náutica Infante D. Henrique – ENIDH

Júri

Presidente	*Professor Doutor* Pedro Viçoso Fazenda Instituto Superior de Engenharia de Lisboa – ISEL
Vogais	*Professor Doutor* André Ribeiro Lourenço Instituto Superior de Engenharia de Lisboa – ISEL
	Professor Doutor Pedro Daniel Dinis Teodoro Instituto Superior de Engenharia de Lisboa – ISEL

Dezembro, 2022

Resumo

Neste trabalho, foi desenvolvido um algoritmo de processamento de imagem que permite obter a posição e orientação de uma embarcação autónoma (*Unmanned Surface Vehicle* ou USV) e um algoritmo para a controlar remotamente, permitindo que siga uma trajetória pré-definida de forma autónoma. Ambos os algoritmos executam em tempo real um modelo de embarcação (USV-enautica1) a operar em ambiente interior (piscina).

O algoritmo de processamento de imagem incluiu a calibração de uma câmara com uma lente *fish-eye*, o cálculo de uma matriz de homografia para observar o campo de visão da câmara numa vista virtual e a deteção da embarcação através de 3 conjuntos de *Light-Emitting Diodes* (LED) colocados na embarcação. Foi ainda desenvolvido um algoritmo que utiliza uma *Lookup Table* (LUT) para melhorar o tempo de processamento do algoritmo de localização da embarcação alcançando uma complexidade $O(1)$ independentemente do tamanho da imagem.

Os resultados experimentais obtidos mostram que a utilização da LUT diminui o tempo de processamento permitindo alcançar um ritmo de imagens por segundo igual ao disponibilizado pela câmara (tempo real). São também apresentados os erros de posição e orientação obtidos ao longo da navegação de várias trajetórias pré-determinadas. Para testar o controlador, foi desenvolvido um simulador. Neste simulador, os resultados demonstram que um controlador PD permite que a embarcação estabilize sobre a trajetória e que a siga sem oscilações.

Palavras-chave: *Lookup Table*; GPS interior; calibração de câmara; homgrafia; deteção de caraterísticas; rastreio em tempo real; estimação de pose de USV; controlo;

Abstract

In this project, an image processing algorithm was developed to obtain the position and orientation of an autonomous vessel (*Unmanned Surface Vehicle* or USV) and a remote control algorithm, allowing the vessel to follow a pre-defined trajectory autonomously. Both algorithms execute in real time a vessel model (USV-enautica1) operating in an indoor environment (pool).

The image processing algorithm includes the calibration of a *fish-eye* lens camera, the computation of an homography matrix to observe the camera's field of view in a virtual view and the detection of the vessel by using 3 sets of *Light-Emitting Diodes* (LED) placed on the vessel. An algorithm which uses a *Lookup Table* (LUT) was also developed to improve the run time of the algorithm achieving $O(1)$ complexity independently of the image size.

The experimental results show the use of LUT decreases the run time allowing a rate of images per second equal to that provided by the camera (real-time). The position and orientation errors obtained while navigating several pre-defined trajectories are also presented. To test the controller, a simulator was developed. In this simulator, the results show that a PD controller allows the vessel to stabilize on the trajectory and follow it without oscillations.

Keywords: *Lookup Table*; indoor GPS; camara calibration; homography; feature detection; real time tracking; USV pose estimation; control;

Agradecimentos

Gostaria de agradecer aos meus orientadores, o Prof. Dr. Pedro Teodoro e o Prof. Dr. Pedro Mendes Jorge por toda a ajuda e apoio que me deram durante o desenvolvimento deste trabalho.

Gostaria também de agradecer à Escola Superior Náutica Infante D. Henrique por ter disponibilizado as suas instalações e aos parceiros e patrocinadores da *Sea2Future*.

Por último mas não menos importante, gostaria de agradecer aos meus familiares, amigos e colegas pelo seu apoio durante todo o meu percurso.

Dedico este trabalho à minha mãe e ao meu pai, por me darem esta oportunidade através dos seus próprios esforços e por todo o seu apoio e paciência ao longo do meu percurso.

Índice de Conteúdos

Índice de Tabelas

Índice de Figuras

Índice de Códigos

Lista de Acrónimos

D Derivativo. 11

ENIDH Escola Superior Náutica Infante D. Henrique. v, 1

GNSS *Global Navigation Satellite System*. 5

HSV Hue Saturation Value. xiv, 30, 31

I Integral. 11

ISEL Instituto Superior de Engenharia de Lisboa. 1

KNN *K-Nearest Neighbors*. 20, 21

LED *Light-Emitting Diodes*. i, iii, xiv, 4, 14, 23, 24, 27–30, 32, 49

LUT *Lookup Table*. i, iii, xi, xiii, xiv, xvii, 3, 4, 13–16, 23, 25–28, 32, 35, 36, 49, 50

MAE *Mean Absolute Error*. xi, 43, 44

OpenCV *Open Source Computer Vision Library*. 10, 17, 18, 20, 24, 31

P Proporcional. xiv, 11, 12, 44–47, 50

PD Proporcional Derivativo. i, iii, xv, 3, 12, 15, 44, 46, 47, 49, 50

PI Proporcional Integral. 12

PID Proporcional Integral Derivativo. 12, 50

RFID *Radio Frequency Identification*. 5, 6

ROS *Robot Operating System*. 2

SIFT *Scale-Invariant Feature Transform*. xiii, 10, 20, 21

SURF *Speeded-up Robust Features*. 10

USV *Unmanned Surface Vehicle*. i, iii, 1

WLAN *Wireless Local Area Network*. 5

Capítulo 1

Introdução

Este trabalho provém de uma parceria entre o Instituto Superior de Engenharia de Lisboa (ISEL) e a Escola Superior Náutica Infante D. Henrique (ENIDH) e enquadra-se na linha de projetos *Sea2Future* [1] que visam a investigação e o desenvolvimento em robótica marítima aplicada. Esta linha de projetos pertence ao Centro de Investigação e Desenvolvimento da ENIDH e desenvolve-se em torno de USV, nomeadamente a embarcação *USV-enautica1* (ver Fig. 1.1), utilizada no decorrer deste trabalho.

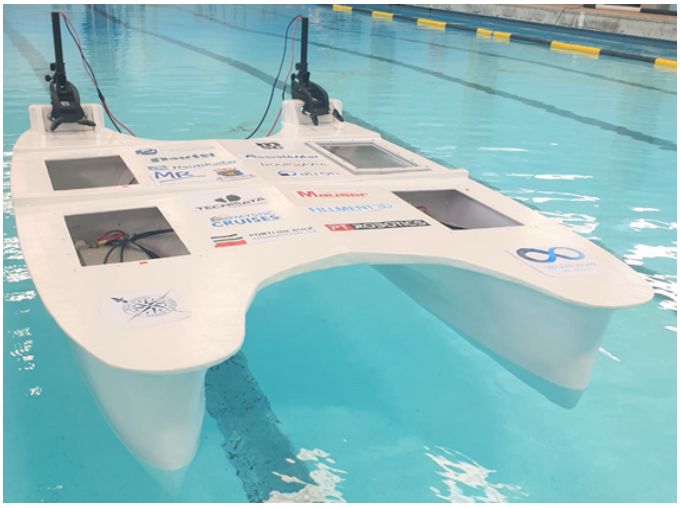


Figura 1.1: USV-enautica1

A *Sea2Future* surge da necessidade de criar uma embarcação autónoma que navegue em oceanos e rios sem a necessidade da presença humana. Atual-

mente, existem veículos autónomos disponíveis no mercado para a deslocação no meio terrestre mas as mesmas opções não são encontradas para o meio aquático.

1.1 Motivação

Antes de introduzir a embarcação no oceano ou rio, esta deve ser colocada num ambiente controlado para efetuar os testes necessários ao seu normal funcionamento.

Por esta razão, a embarcação *USV-enautica1* foi introduzida numa piscina interior onde surgiu a necessidade de a controlar de forma autónoma e remota. Este controlo requer o conhecimento da posição e orientação da embarcação em tempo real. Para obter estes dados, são necessárias técnicas próprias para ambientes interiores uma vez que as técnicas utilizadas em ambientes exteriores não produzem os resultados desejados. Como referido, esta operação é efetuada em tempo real, por essa razão, é necessário otimizar a aplicação para cumprir este requisito.

Em resumo, este trabalho visa a obtenção dos dados da embarcação num ambiente interior e a sua utilização para a controlar.

1.2 Enquadramento

Este trabalho foi desenvolvido com recurso à embarcação *USV-enautica1* [2], equipada com um *Raspberry Pi* [3] juntamente com a *framework Robot Operating System* (ROS) [4]. Esta *framework* é utilizada na troca de dados utilizando o conceito *Publish/Subscribe*, tornando possível a comunicação com a embarcação ao publicar dados num determinado tópico. Estes dados são obtidos ao subscrever esse mesmo tópico.

A embarcação possui dois motores instalados na sua traseira que permitem o seu controlo (ver Fig. 1.2).

Figura 1.2: Motores da embarcação

A potência dos motores pode ser controlada publicando dois valores: o primeiro (AV) indica a potência base para ambos os motores e o segundo (AC) permite controlar a rotação da embarcação. As potências dos dois motores, compreendidas entre 0 e 100, são dadas pelas seguintes equações:

$$M_{DIREITO} = AV + AC \tag{1.1}$$

$$M_{ESQUERDO} = AV - AC \tag{1.2}$$

onde $M_{DIREITO}$ representa a potência do motor direito e $M_{ESQUERDO}$ a potência do motor esquerdo.

Se $AC > 0$, a potência do motor direito irá ser superior à do motor esquerdo e a embarcação curva para a esquerda. Se $AC < 0$, a embarcação curva para a direita. Se $AC = 0$, a embarcação desloca-se em linha reta.

A embarcação possui ainda baterias e diversos sensores e atuadores que contribuem para o seu normal funcionamento.

1.3 Metodologia utilizada

Para alcançar os objetivos apresentados na Motivação (secção 1.1), são necessários vários passos: o primeiro é a calibração de uma câmara equipada com uma lente *fish-eye*; em seguida aplica-se uma transformação homográfica para obter uma vista virtual de cima da piscina; no passo seguinte é criada uma LUT para melhorar o desempenho da aplicação; em seguida, a embarcação é detetada e a posição e orientação da embarcação na vista virtual é obtida com recurso à LUT; no último passo é desenvolvido um simulador com uma embarcação virtual na qual é utilizado um controlador PD para que esta siga uma trajetória pré-definida autonomamente.

1.4 Contribuições

No decorrer deste trabalho, foi desenvolvido um método para diminuir o tempo de processamento da aplicação. O método consiste na criação de uma LUT que permite substituir múltiplos passos de processamento de imagem por um simples passo de indexação. Esta substituição ocorre em cada imagem adquirida para obter a posição dos marcadores LED colocados na embarcação. Desta forma, é possível reduzir a complexidade temporal da aplicação de $O(n^2)$ para uma complexidade $O(1)$.

Foi ainda escrito o artigo "*Real-time GPS indoor for USV tracking using Lookup Table*". Este apresenta o processo utilizado para rastrear a embarcação num ambiente interior, aplicando a LUT.

1.5 Organização

O documento está organizado nos seguintes capítulos: o capítulo 2 explora o estado da arte dos principais temas abordados neste trabalho: comunicações em ambiente interior, algoritmos de imagem e estratégias de controlo; no capítulo 3 é apresentado o desenvolvimento efetuado, nomeadamente, a metodologia abordada, a solução proposta e a implementação; em seguida, no capítulo 4, são discutidos os resultados obtidos, tanto no rastreio da embarcação, como no seu controlo remoto; por fim, são apresentadas as conclusões no capítulo 5, bem como o trabalho futuro que pode ser desenvolvido para complementar ou melhorar o trabalho já efetuado.

Capítulo 2

Estado da Arte

O trabalho desenvolvido aborda os temas localização em ambiente interior, processamento de imagem e controlo. Para obter a posição e orientação da embarcação, é necessário estudar técnicas e tecnologias adequadas para ambientes interiores. A abordagem escolhida inclui a utilização de uma câmara e torna-se necessário obter as informações da embarcação através das imagens adquiridas. Por último, depois de calcular a posição e orientação da embarcação, procede-se ao controlo da mesma e por isso são analisadas diferentes estratégias de controlo.

2.1 Localização em Ambiente Interior

Localização em ambientes interiores é um tópico de pesquisa muito interessante com vários tipos de aplicações para rastrear pessoas, animais ou objetos. *Global Navigation Satellite System* (GNSS) é, provavelmente, a tecnologia mais conhecida e utilizada para localização mas é mais adequada para ambientes exteriores. Várias alternativas têm sido estudadas e propostas para ambientes interiores, sendo as mais comuns infravermelhos, ultrassom, *Wireless Local Area Network* (WLAN), identificação por radiofrequência (*Radio Frequency Identification* ou RFID) e visão computacional (baseado em imagem) [5].

2.1.1 Infravermelhos

Infravermelhos é uma das tecnologias mais comuns e diversas aplicações desta tecnologia são estudadas em [6–9]. Os estudos apresentados demonstram que

a tecnologia apresenta bons resultados para aplicações em tempo real mas a sua precisão é afetada pela distância percorrida pelo sinal e pelo *multipath error*. Este erro ocorre quando um sinal bate e reflete em objetos no ambiente e o recetor recebe o sinal original e os vários sinais refletidos, afetando a sua precisão.

2.1.2 Ultrassom

Ultrassom é outra alternativa que apresenta bons resultados em aplicações que necessitam de operar em tempo real mas é suscetível a interferência de ruído e a sua precisão pode ser afetada por condições ambientais (e.g. temperatura do ar e humidade) [10–12].

2.1.3 WLAN

Várias aplicações aproveitam a infraestrutura WLAN instalada para rastrear a posição de um alvo em ambientes interiores [13–15]. É, no entanto, uma tecnologia com pouca precisão que tem de ser aumentada com a adição de *routers*/pontos de acesso extra, aumentando a complexidade do sistema.

2.1.4 RFID

RFID é outra alternativa utilizada para localização interior. Devido às particularidades desta tecnologia, é necessário utilizar múltiplos marcadores ou leitores RFID espalhados pelo ambiente para melhorar a precisão do sistema [16–18].

2.1.5 Visão

Por último, localização baseada em imagem/visão é outra técnica muito utilizada para rastrear alvos em ambientes interiores devido à sua versatilidade. Esta tecnologia funciona utilizando marcadores [19], deteção de caraterísticas [20], segmentação [21] ou fluxo óptico [22].

2.2 Algoritmos de Imagem

Uma solução comum para rastrear objetos em ambientes interiores utilizando visão, é através do uso de câmaras fixas. Esta metodologia consiste em calibrar a câmara, para remover a distorção introduzida em cada imagem adquirida, e reprojetar a região de interesse da imagem [23–26]. Para esta reprojeção, deve ser calculada a matriz de transformação homográfica que necessita de múltiplos pontos da imagem. Estes pontos podem ser obtidos de uma forma dinâmica utilizando as características da imagem. Por último, para detetar a embarcação em cada imagem adquirida, é necessário aplicar segmentação de imagem.

2.2.1 Calibração

Para remover as distorções introduzidas em cada imagem pela câmara, é necessário calcular os parâmetros da mesma [27]. Este processo é conhecido como calibração da câmara. Os parâmetros da câmara podem ser definidos como intrínsecos e extrínsecos (ver Fig. 2.1).

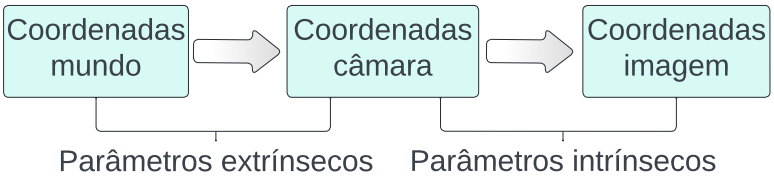


Figura 2.1: Parâmetros extrínsecos e intrínsecos

Os parâmetros extrínsecos consistem na matriz de rotação e no vetor de translação que relacionam os eixos de coordenadas do mundo e da câmara. Estes parâmetros permitem transformar um ponto no referencial do mundo num ponto no referencial da câmara. Para tal, é utilizada a equação 2.1.

$$P_c = RP_w + T \quad \equiv \quad \lambda \begin{bmatrix} X^C \\ Y^C \\ Z^C \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X^W \\ Y^W \\ Z^W \\ 1 \end{bmatrix} \tag{2.1}$$

onde P_c e P_w são os pontos no referencial da câmara e do mundo respetivamente, R é a matriz de rotação, T o vetor de translação e λ é um fator

de escalamento relacionado com as coordenadas homogéneas.

Os parâmetros intrínsecos incluem os coeficientes de conversão de coordenadas métricas em pixeis (a_u, a_v) e o ponto principal (coordenadas do centro óptico projetado na imagem - u_0, v_0) da câmara (ver equação 2.2).

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = C.P_c = \begin{bmatrix} a_u & 0 & u_0 \\ 0 & a_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{2.2}$$

É ainda necessário calcular os coeficientes de distorção da lente. Estes coeficientes são utilizados para corrigir a distorção radial e tangencial que é introduzida em cada imagem pela lente, que provoca uma alteração ao modelo de projeção ideal.

Para calibrar a câmara é necessário corresponder pontos no referencial da imagem com pontos conhecidos no referencial do mundo. Concatenando as equações 2.1 e 2.2 temos:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a_u & 0 & u_0 \\ 0 & a_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} (R|t) \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{2.3}$$

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = P \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{2.4}$$

Para cada correspondência entre o ponto no referencial do mundo e a sua projeção na imagem, são obtidas duas equações:

$$u = \frac{p_{11}X + p_{12}Y + p_{13}Z + p_{14}}{p_{31}X + p_{32}Y + p_{33}Z + p_{34}} \tag{2.5}$$

$$v = \frac{p_{21}X + p_{22}Y + p_{23}Z + p_{24}}{p_{31}X + p_{32}Y + p_{33}Z + p_{34}} \tag{2.6}$$

Uma vez que existem 12 incógnitas e 2 equações por ponto, são necessários, no mínimo, 6 pontos correspondentes para se obter 12 equações.

Atualmente, existem várias bibliotecas que disponibilizam algoritmos de auto-calibração e remoção de distorções [28–30].

2.2.2 Homografia

Para obter a posição e orientação da embarcação no plano da água, é necessário ter uma vista de cima da piscina sem distorções. Para obter esta vista, é necessário aplicar uma transformação homográfica. Uma transformação homográfica pode ser definida como uma transformação entre dois planos [31]. Esta transformação é uma simplificação da projeção apresentada na sub-secção 2.2.1, onde é escolhido um referencial do mundo em que os pontos tenham $Z = 0$ (ver equação 2.7).

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{14} \\ p_{21} & p_{22} & p_{24} \\ p_{31} & p_{32} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{2.7}$$

A matriz 3x3 resultante é a matriz homográfica. Esta matriz é muitas vezes apresentada como:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{2.8}$$

A variável h_{33} é um fator de escalamento e pode ser assumido que $h_{33} = 1$, restando 8 incógnitas. Aplicando a mesma lógica da sub-secção 2.2.1 à equação 2.7, cada correspondência dá origem a 2 equações. São por isso necessárias 4 correspondências para calcular as 8 incógnitas restantes.

A biblioteca *Open Source Computer Vision Library* (OpenCV) possui métodos que permitem calcular a matriz homográfica [32] e reprojetar a imagem [33].

2.2.3 Características

De modo a que a transformação homográfica seja feita de forma autónoma, é necessário que as correspondências a serem utilizadas nesta transformação também sejam obtidas autonomamente.

A obtenção destas correspondências pode ser auxiliada através de vários algoritmos, como *Canny* [34] ou *Sobel* [35] para detetar os contornos da piscina, ou [36] ou [37] para detetar os cantos diretamente.

Outra alternativa muito interessante é a extração de características da imagem. Estas características podem ser obtidas através de algoritmos como *Scale-Invariant Feature Transform* (SIFT) [38] ou *Speeded-up Robust Features* (SURF), uma variação do algoritmo SIFT.

A biblioteca OpenCV também providencia algoritmos para obter estas características [39, 40]

2.2.4 Segmentação

Segmentação de imagem é o processo de decompor uma imagem em múltiplas regiões, com o objetivo de isolar as zonas de interesse dessa imagem. Este processo pode ser realizado com diferentes técnicas [41], tal como *thresholding*, *clustering* ou técnicas de inteligência artificial que utilizam, por exemplo, redes neuronais.

Uma das técnicas utilizadas na segmentação de imagem é o método *clustering* que divide a imagem em vários conjuntos (*clusters* ou classes) consoante uma medida de semelhança entre os pixeis baseado em características, como por exemplo, cor, intensidade ou textura.

Outra alternativa consiste na utilização de redes neuronais. Estas redes são constituídas por camadas de neurónios que aprendem ao ajustar os pesos entre camadas até que os valores de saída sejam os desejados.

Thresholding é outro método utilizado na segmentação de imagem. Este método consiste em converter uma imagem de tons de cinzento numa imagem

binária consoante um valor de limiar (*threshold*).

2.3 Estratégias de Controlo

Para que a embarcação siga uma trajetória de forma autónoma, é necessário algum tipo de controlador. Existem dois tipos principais de controladores: discretos e contínuos [42, 43]. Os controladores discretos têm valores de saída limitados a valores discretos enquanto os controladores contínuos possuem uma variação maior no valor de saída.

Os controladores contínuos podem ser divididos principalmente em 3 tipos distintos:

- Proporcional (P);
- Derivativo (D);
- Integral (I);

Um controlador P é um controlador simples de implementar. Este tipo de controlador produz um resultado diretamente proporcional ao valor do erro. Pode ser descrito pela seguinte equação em função do tempo (t):

$$O(t) = K_p \, e(t) \tag{2.9}$$

onde O representa o valor de saída, e representa o erro e K_p o ganho.

Um controlador D tenta minimizar a alteração do erro para reduzir oscilações no sistema. Os valores de saída deste controlador são calculados a partir das alterações do erro ao longo do tempo. Pode ser descrito pela seguinte equação em função do tempo (t):

$$O(t) = K_d \, \frac{de(t)}{dt} \tag{2.10}$$

onde O representa o valor de saída, e representa o erro e K_d o ganho.

Por fim, um controlador I permite manter alguma estabilidade num determinado ponto e por isso é comum ser utilizado em conjunto com outro tipo de controlador. No entanto, pode ser descrito pela seguinte equação em função do tempo (t):

$$O(t) = K_i \int e(t)\ dt \tag{2.11}$$

onde O representa o valor de saída, e representa o erro e K_i o ganho.

Estes tipos de controladores nem sempre apresentam os resultados desejados individualmente e por isso podem ser combinados para criar novos controladores com comportamentos diferentes:

- Proporcional Derivativo (PD);
- Proporcional Integral (PI);
- Proporcional Integral Derivativo (PID);

Estes controladores juntam as propriedades de cada controlador num único. Os controladores PD, PI e PID podem ser descritos, respetivamente, pelas seguintes equações:

$$O(t) = K_p\ e(t) + K_d\ \frac{de(t)}{dt} \tag{2.12}$$

$$O(t) = K_p\ e(t) + K_i \int e(t)\ dt \tag{2.13}$$

$$O(t) = K_p\ e(t) + K_i \int e(t)\ dt + K_d\ \frac{de(t)}{dt} \tag{2.14}$$

Os artigos [44–46] apresentam os resultados da aplicação destes controladores em diferentes problemas. O controlador P possui uma elevada oscilação antes de estabilizar e demora mais até esta ocorrer comparado com os restantes controladores. Tanto os controladores PD, PI e PID apresentam resultados melhores nos casos em estudo sendo o controlador PID aquele que mais se destaca.

Capítulo 3

Desenvolvimento

Neste capítulo são detalhadas as várias etapas para o desenvolvimento deste trabalho. A primeira secção descreve a metodologia abordada transmitindo uma visão geral das diferentes vertentes do sistema implementado. Em seguida é apresentada a solução proposta onde é descrito o método de criação de uma LUT que permite melhorar o tempo de execução do sistema. Por último, é descrita a implementação do sistema onde é explicado com maior detalhe as suas várias vertentes, desde a configuração do mesmo até o controlo da embarcação de modo a que esta siga uma trajetória pré-definida de forma autónoma.

3.1 Metodologia Abordada

Para obter a posição e orientação da embarcação em tempo real, e a sua posterior navegação autónoma, são necessárias várias etapas. A Fig. 3.1 ilustra as principais etapas necessárias para obter a posição e orientação da embarcação começando com uma imagem adquirida com distorções pela câmara, posterior correção de distorções, reprojeção da zona de interesse da imagem, aplicação de uma LUT para adquirir a posição e orientação da embarcação e por fim a simulação de uma trajetória seguida pela embarcação de forma autónoma.

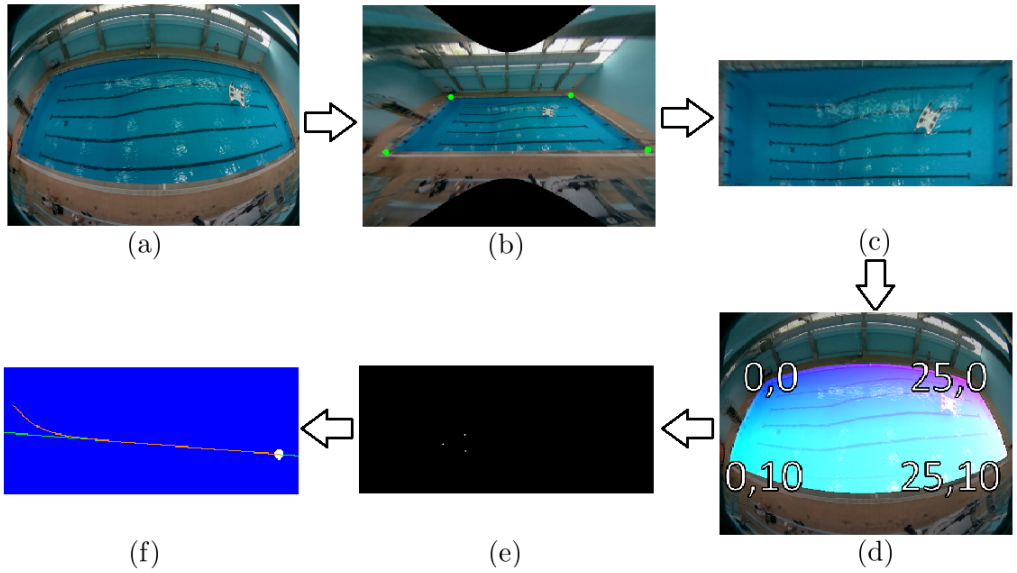


Figura 3.1: (a) Imagem adquirida. (b) Imagem com distorção corrigida e pontos de referências para a transformação homográfica. (c) Imagem reprojetada e redimensionada para a proporção da piscina. (d) Imagem adquirida com LUT sobreposta. (e) Pontos obtidos aplicando a LUT. (f) Simulação de uma trajetória.

O sistema pode ser dividido em 3 partes.

A primeira parte corresponde à inicialização do sistema. Nesta parte, é feita a calibração de uma câmara, equipada com uma lente *fish-eye*, apontada para a piscina onde a embarcação navega. É ainda calculada a matriz de transformação homográfica para transformar o campo de visão da câmara numa vista virtual de cima. Por fim, é criada uma LUT que permite melhorar o processo efetuado na segunda parte.

Na segunda parte, procede-se à análise da imagem para detetar os marcadores LED colocados na embarcação e à utilização da LUT logo em seguida para obter a posição dos marcadores na imagem final (corrigida e reprojetada). Para operar em tempo real, a aplicação deve ser otimizada para executar em dispositivos com recursos limitados, como um *Raspberry Pi*. A LUT é utilizada com o propósito de otimizar a aplicação ao substituir os passos de correção de distorções e reprojeção da imagem (ver Fig. 3.2). A posição dos marcadores na imagem final é utilizada para calcular a posição (x, y) e orientação (θ) da embarcação.

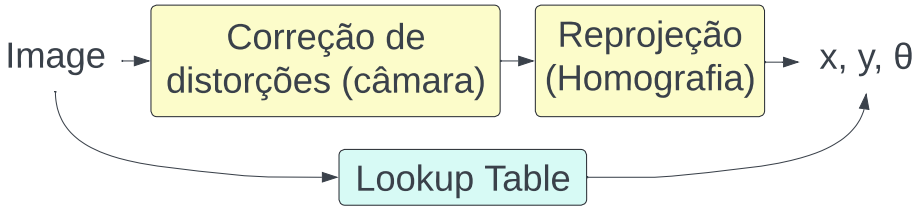


Figura 3.2: Utilização de *Lookup Table*

Esta substituição permite diminuir a complexidade temporal do sistema para $O(1)$ e aumentar, por consequência, as imagens processadas por unidade de tempo, independente da resolução da imagem adquirida.

A terceira parte diz respeito ao controlo da embarcação. Nesta parte, foi desenvolvido um simulador de modo a evitar o uso da embarcação. É utilizado um controlador PD, aplicado à simulação, juntamente com a posição e orientação do objeto da simulação, obtida em cada *frame*, para que este siga uma trajetória pré-definida de forma autónoma.

3.2 Implementação

Nesta secção é descrita a implementação do sistema começando com a configuração do mesmo, o processo de calibração da câmara utilizada para corrigir as distorções que esta adiciona à imagem, o cálculo da matriz de projeção para obter uma vista virtual de cima da piscina, o modo como é feito o rastreio e cálculo da posição e da orientação da embarcação e por fim, o controlo da mesma de modo a que esta siga uma trajetória pré-definida de forma autónoma.

3.2.1 Configuração do sistema

Como mencionado na secção 3.1, é necessária uma câmara com um campo de visão que inclua toda a área da piscina. Assim, foi decidido utilizar uma câmara com uma lente grande angular (*fish-eye*). Além disso, devido ao ambiente interior de uma piscina ser húmido e nocivo para a câmara, foi utilizado um sistema de proteção. Quando esta não está a ser utilizada, é aplicada uma rotação através de um motor servo que permite recolher a câmara e protegê-la melhor do ambiente da piscina. (ver Fig. 3.3).

(a) Câmara conectada ao motor servo

(b) Câmara e servo instalados no teto da piscina

Figura 3.3: Instalação da câmara

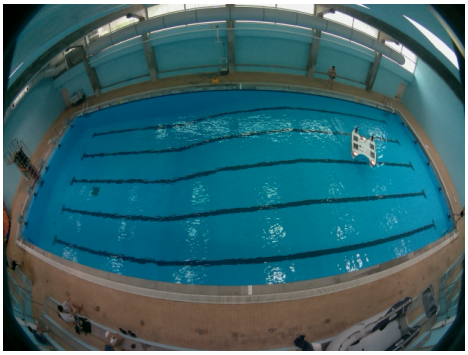
Estas opções provocam alguns desafios que necessitam ser resolvidos para que o sistema tenha a precisão necessária. A lente *fish-eye* provoca uma distorção nas imagens adquiridas (efeito "barril") que necessita ser corrigida. A rotação da câmara pode provocar um desalinhamento na posição da operação (esta não é sempre a mesma) que necessita ser compensada, uma vez que altera a transformação homográfica e, consequentemente, a LUT.

A inicialização do sistema e a análise de imagem com utilização da LUT (a primeira e segunda parte mencionadas na secção 3.1, respetivamente) são executadas num *Raspberry Pi*, fora da piscina, conectado à câmara. O controlo da embarcação deve ser executado noutro *Raspberry Pi* conectado à embarcação.

3.2.2 Calibração da câmara

Para obter a posição e orientação da embarcação em relação à piscina, é necessária uma câmara com um ângulo de visão amplo o suficiente para incluir toda a piscina em cada imagem adquirida. Uma câmara com uma lente *fish-eye* permite observar toda a piscina mas introduz distorções em cada imagem. A Fig. 3.4(a) mostra a câmara com lente *fish-eye* e o *Raspberry Pi* utilizados enquanto a Fig. 3.4(b) apresenta uma imagem adquirida pela câmara.

(a) *Raspberry Pi* e câmara *fish-eye*



(b) Imagem adquirida

Figura 3.4: (a) *Raspberry Pi* e câmara com lente *fish-eye*. (b) Imagem da piscina adquirida pela câmara

Como pode ser observado na Fig. 3.4(b), a câmara introduz distorção à imagem adquirida transformando linhas retas em linhas curvas sendo mais proeminente nos cantos da imagem e menos no centro (efeito "barril").

Como mencionado no estado da arte, na subsecção 2.2.1, para a calibração da câmara (determinação dos parâmetros intrínsecos e extrínsecos) são necessárias pelo menos 6 correspondências entre os pontos com coordenadas no referencial do mundo e as respetivas coordenadas no referencia da imagem. Para obter estas correspondências pode ser utilizado um determinado padrão que permita relacionar os pontos entre os dois referenciais. Foi utilizado o procedimento de calibração de câmara do OpenCV que utiliza um padrão calibrador quadriculado igual ao apresentado na Fig. 3.5.

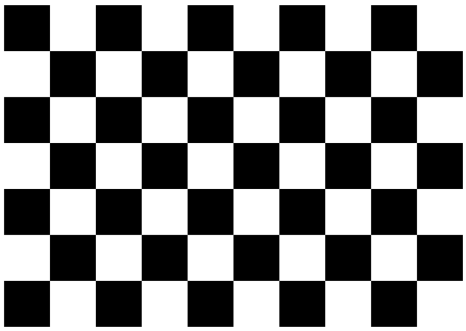


Figura 3.5: Padrão de calibração

Para otimizar a calibração, procurou-se adquirir imagens em múltiplas posições de modo a cobrir o campo de visão da câmara. A Fig. 3.6 apresenta alguns exemplos das imagens adquiridas.

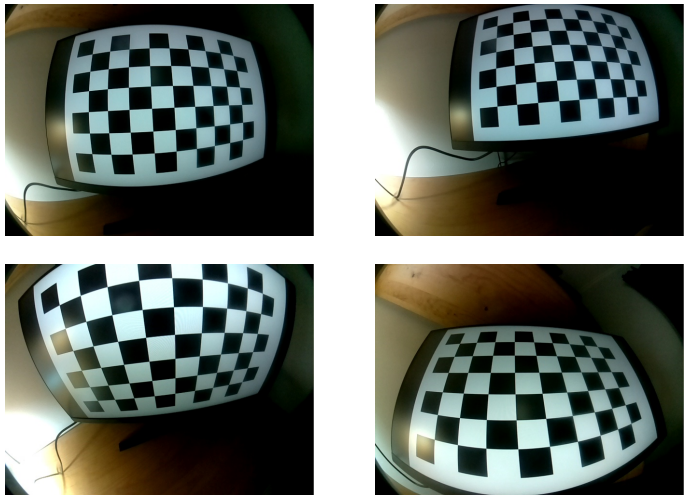


Figura 3.6: Exemplo de imagens utilizadas na calibração da câmara

A biblioteca OpenCV disponibiliza algoritmos para calibrar câmaras com lentes *fish-eye* e corrigir as distorções que introduzem [29], nomeadamente o métodos `calibrate`, `estimateNewCameraMatrixForUndistortRectify` e `initUndistortRectifyMap`. A Fig. 3.7 apresenta uma imagem adquirida pela câmara (Fig. 3.4(b)), com as distorções corrigidas, utilizando os parâmetros da câmara já determinados.

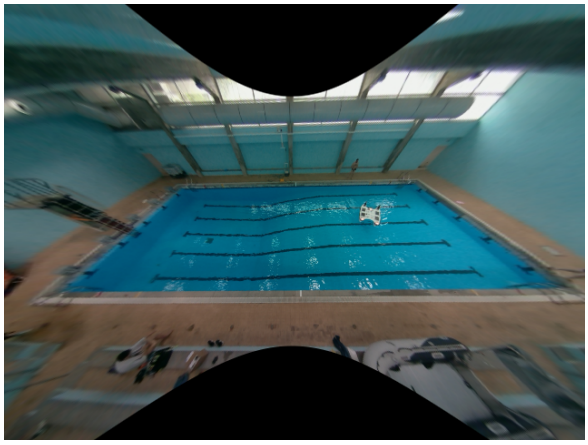
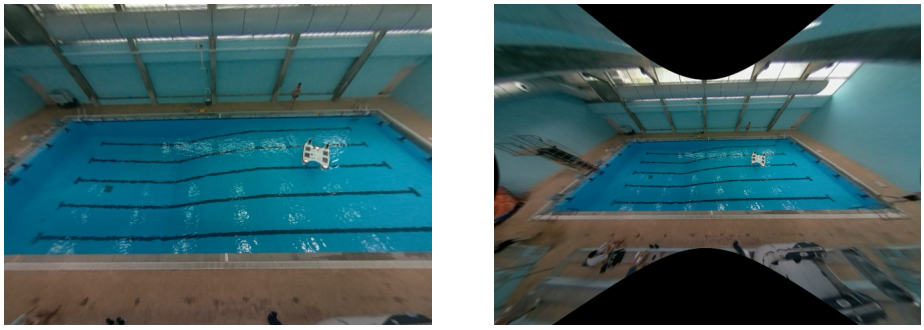


Figura 3.7: Fig. 3.4(b) corrigida

Analisando a Fig. 3.7, é possível observar pixeis que não pertencem à imagem original, adicionados para compensar a sua correção. O método utilizado, `estimateNewCameraMatrixForUndistortRectify` possui um parâmetro de entrada, `balance`, com valores compreendidos entre 0 e 1, que permite controlar o número desses pixeis. A Fig. 3.8 mostra a mesma imagem adquirida, corrigida, com `balance = 0` e `balance = 1`.



(a) `balance = 0` (b) `balance = 1`

Figura 3.8: Comparação de diferentes valores de `balance`

Como se pode analisar, com `balance = 0` a imagem contém apenas pixeis "válidos", mas parte da piscina é cortada. Com `balance = 1`, a imagem contém todos os pixeis da imagem original e pixeis pretos fora da imagem para compensar a correção. Foi escolhido o valor de `balance` de 0.9 que permite obter uma imagem com a totalidade da piscina enquanto, ao mesmo tempo, reduz os pixeis pretos (ver Fig. 3.7).

3.2.3 Transformação homográfica

Para obter a posição e orientação da embarcação no plano da água, é necessário ter uma vista de cima da piscina. Assim, é possível localizar a embarcação no plano da piscina de acordo com um referencial local, localizado no seu canto superior esquerdo. A câmara não pode ser colocada diretamente por cima da piscina, por isso, para obter esta vista, é necessário aplicar uma transformação homográfica.

Como mencionado no estado da arte, na subsecção 2.2.2, são necessários pelo menos 4 correspondências para calcular a matriz de transformação homográfica. A zona da piscina é a única área de interesse nas imagens. Por

esta razão, os pontos (de origem) utilizados no cálculo da matriz são os 4 cantos da piscina. Os pontos de destino são os 4 cantos da imagem para que a estimação da posição e orientação da embarcação seja direta.

Na subsecção 3.2.1 refere-se que a câmara está conectada a um motor servo, que a roda para a posição pretendida, de modo a que esta possa ser protegida quando não está a ser utilizada. Os 4 pontos de origem utilizados no cálculo da matriz de transformação homográfica são registados manualmente através de uma imagem de referência, adquirida na posição de normal funcionamento da câmara. Apesar do ângulo de rotação do motor servo ser sempre o mesmo, alguns desvios na sua posição podem ocorrer. Estes desvios devem ser compensados para não alterar a posição dos pontos de origem.

Para corrigir estes desvios, é adquirida uma nova imagem que é comparada com a imagem de referência. O objetivo desta comparação é estimar um vetor diferença que represente a diferença entre as duas imagens. Para realizar esta comparação, optou-se por utilizar caraterísticas locais SIFT e o algoritmo *K-Nearest Neighbors* (KNN) para associar as caraterísticas entre imagens.

Os desvios são pequenos, por essa razão a procura por caraterísticas pode ser limitada a pequenas janelas em cada canto da piscina. A Fig. 3.9 mostra as caraterísticas locais SIFT obtidas nas janelas nos cantos do topo da piscina.

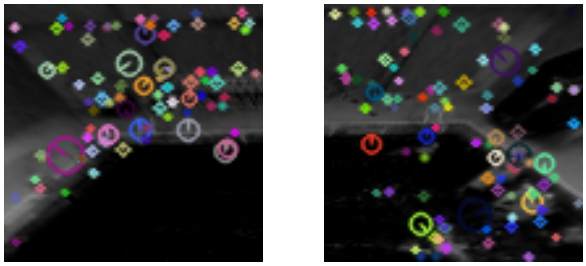


Figura 3.9: Caraterísticas locais SIFT obtidas nos cantos do topo da piscina

A biblioteca OpenCV disponibiliza algoritmos para calcular estas características [39].

Ao obter as caraterísticas em cada imagem, na de referência e na nova imagem adquirida, estas podem ser comparadas. Esta comparação pode

ser efetuada utilizando, por exemplo, um algoritmo KNN. Este algoritmo compara os descritores (obtidos no cálculo das caraterísticas locais) de cada caraterística da imagem de referência com os descritores de cada caraterística da nova imagem. As `k` características mais semelhantes são retornadas. A Fig. 3.10 apresenta as múltiplas correspondências.

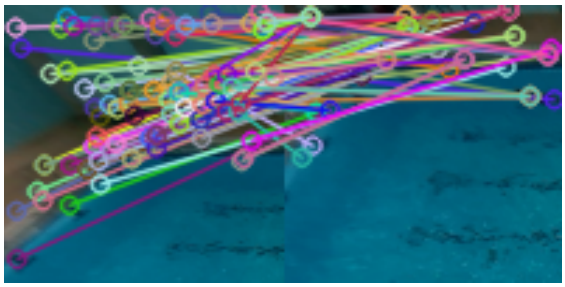


Figura 3.10: Correspondências das caraterísticas locais SIFT entre duas imagens: imagem de referência à esquerda e a nova imagem à direita

Analisando a Fig. 3.10, é possível perceber que esta apresenta correspondências incorretas. Estas correspondências incorretas podem ser filtradas aplicando um *Ratio Test* [38]. Este teste consiste em comparar a distância entre o vizinho mais próximo da caraterística SIFT em estudo, com a distância ao segundo vizinho mais próximo. Se a distância ao vizinho mais próximo for semelhante à distância ao segundo mais próximo, então a caraterística é descartada. É necessário haver uma diferença mínima entre o primeiro e segundo melhor vizinho para que o primeiro seja aceite. Este teste permite assim filtrar correspondências incorretas.

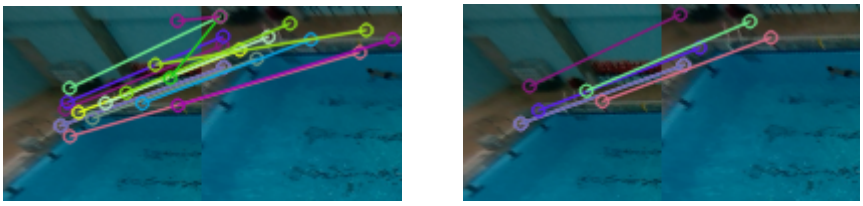
Para aplicar este teste é necessário haver 2 vizinhos para cada caraterística. Por essa razão, é utilizado `k = 2` no algoritmo KNN. O código 1 apresenta a aplicação do algoritmo KNN aos descritores das caraterísticas de cada imagem e a implementação do *Ratio Test*.

Código 1: KNN e *Ratio Test*

```
matches = bf.knnMatch(des1, des2, k=2)
for m, n in matches:
    if m.dist < ratio_dist*n.dist:
        # Valid features
```

A variável `ratio_dist` representa o grau de semelhança das duas distâncias. Quanto menor o valor desta variável, maior tem de ser a diferença entre

as distâncias, ou seja, maior é a filtragem. A Fig. 3.11 mostra as correspondências restantes da Fig. 3.10, após da aplicação do *Ratio Test*.



(a) `ratio_dist = 0.75` (b) `ratio_dist = 0.5`

Figura 3.11: Correspondências depois de aplicar o *Ratio Test*

Como se pode ver na Fig. 3.11, com `ratio_test = 0.5` é possível filtrar as correspondências incorretas.

Com as correspondências restantes, calcula-se o vetor médio entre elas. Este vetor representa a diferença/erro entre as duas imagens e é utilizado para calcular o ponto referência na nova imagem, somando o vetor ao ponto da imagem de referência (ver Fig. 3.12).

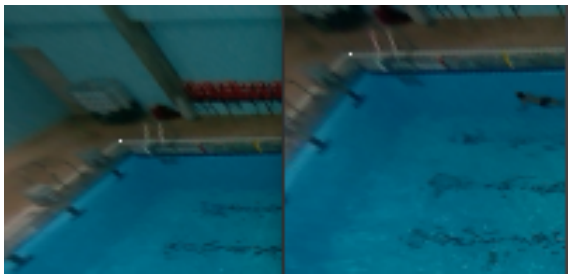


Figura 3.12: Ponto de referência com pixel branco nas imagens base e atual

Aplicando o mesmo algoritmo a todos os 4 pontos, obtém-se os 4 pontos de origem para o cálculo da matriz de transformação homográfica. Esta matriz é então utilizada para reprojetar a imagem. A Fig. 3.13 mostra o resultado da reprojeção com um posterior redimensionamento para manter a mesma proporção da piscina.

Figura 3.13: Imagem reprojetada a partir da transformação homográfica estimada e do contributo dos pontos associados

3.2.4 *Lookup Table*

Apesar da calibração da câmara e do cálculo da matriz homográfica ser realizado apenas uma vez, a correção de distorções e a transformação homográfica têm de ser efetuadas em cada imagem. Em aplicações onde é necessário a execução em tempo real, existe a necessidade de diminuir a sua complexidade computacional.

Durante o desenvolvimento deste trabalho, foi implementado um método que permite diminuir o tempo de processamento da aplicação. Este método consiste na criação de uma LUT para substituir os diversos passos de processamento de imagem por um simples passo de indexação (ver Fig. 3.2). Esta utilização ocorre para cada imagem adquirida para obter a posição dos marcadores LED da embarcação na imagem de uma forma direta. Desta forma, é possível reduzir a complexidade computacional da aplicação de $O(n^2)$ para uma complexidade $O(1)$.

A Fig. 3.14 resume os diferentes passos para criar a LUT.

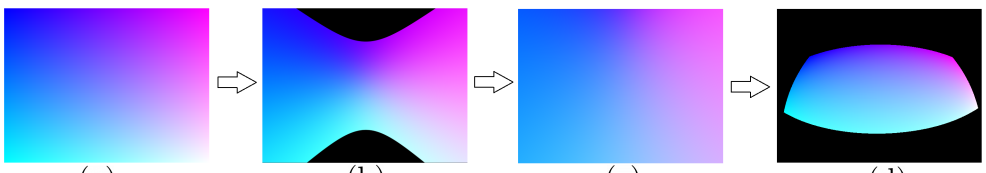


Figura 3.14: (a) Imagem normalizada. (b) Aplicação da correção de distorções da câmara. (c) Aplicação da transformação homográfica. (d) LUT criada

Para obter a posição dos marcadores LED sem aplicar a correção de distorções e a transformação homográfica, devemos associar os pixeis da imagem adquirida (imagem original) aos pixeis da imagem corrigida. Para definir a posição de um marcador é necessário saber o seu x e y e por isso são necessários dois planos. Em vez de dois planos, podemos ter uma imagem onde dois canais de cor representam os planos.

O primeiro passo é criar uma imagem onde cada pixel corresponde ao respetivo índice (ver código 2). Uma vez que são apenas necessários dois dos três canais de cor da imagem, o canal restante pode ter, por exemplo, o valor 1. Para efeitos de visualização, normalizamos os valores.

Código 2: Criação da imagem normalizada

```
1 normalized_image=np.zeros([height,width,3], dtype=np.float32)
2 for i in range(len(normalized_image)):
3     for j in range(len(normalized_image[i])):
4         normalized_image[i][j]=(1, i/(height-1), j/(width-1))
```

O OpenCV utiliza o formato *BGR* para os canais de cor e por isso a ordem dos canais no código 2 está ao contrário.

A Fig. 3.15 corresponde à imagem normalizada criada, no formato *RGB*, que representa a localização de cada pixel.

Figura 3.15: Imagem normalizada

Para uma imagem com uma resolução de 640 por 480 pixeis, por exemplo, o pixel na coluna com índice 10 e na linha com índice 10, possui o seguinte valor:

$$(10/479, 10/639, 1) = (0.02087683, 0.01564945, 1)$$

Em seguida, são aplicados os mesmos passos de processamento de imagem que são aplicados às imagens da câmara: correção de distorções e transformação homográfica (ver código 3).

Código 3: Aplicar transformações à imagem normalizada

```
undistorted = camera.un_distort(normalized_image)
reprojected = homography.apply_homography(undistorted)
```

O valor de cada pixel na imagem resultante contém informação referente aos índices que lhe deram origem e por isso é possível transformar o valor de cada pixel na posição respetiva como se pode ver no código 4.

Código 4: Transformar os valores dos pixeis em posições

```
reprojected[:, :, 1] *= (height - 1)
reprojected[:, :, 2] *= (width - 1)
reprojected = np.round(reprojected).astype(int)
```

Por fim, a LUT é criada correspondendo as posições guardadas na imagem normalizada transformada (índices da LUT) com os valores da imagem normalizada original (valores da LUT) como se pode ver no código 5.

Código 5: Criação da LUT

```
lut = np.zeros([height, width, 3], dtype=np.float32)
for x in range(len(reprojected)):
    for y in range(len(reprojected[x])):
        pix = reprojected[x][y]
        lut[pix[1]][[pix[2]]] = normalized_image[x][y]
```

A Fig. 3.16 representa a LUT obtida. Contudo, como se pode observa na Fig. 3.16, a LUT apresenta algumas falhas devido aos pixeis da imagem original não cobertos.

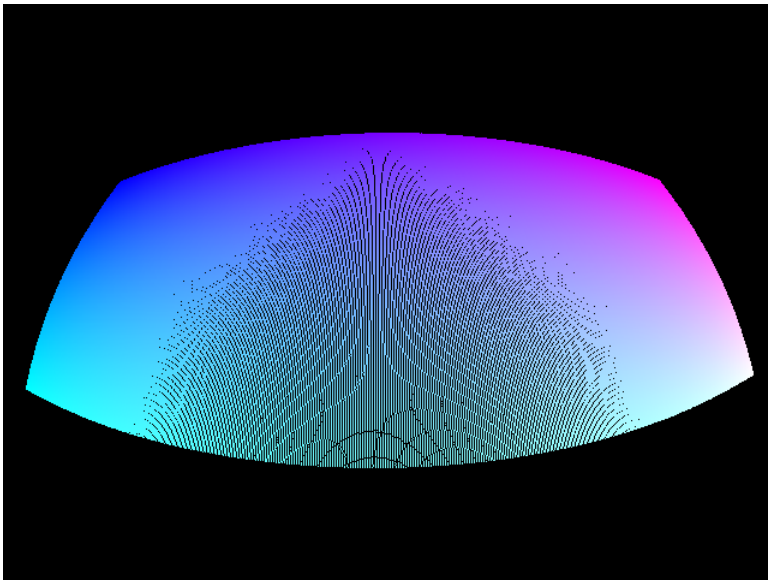


Figura 3.16: LUT com algumas falhas

Estas falhas podem ser corrigidas aplicando uma dilatação de modo a preencher os pixeis a preto, seguido de uma erosão para que a região original não se altere (operação *Morphological Closing*), obtendo a LUT final apresentada na Fig. 3.17.

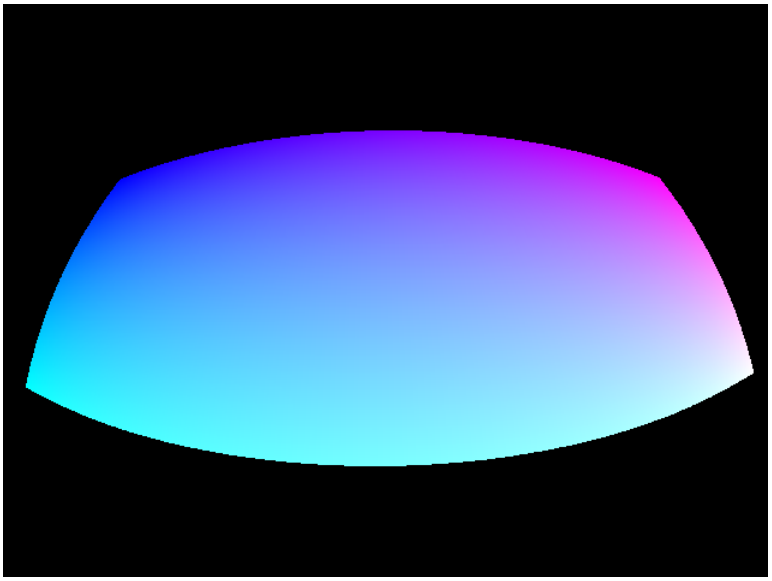


Figura 3.17: *Lookup Table*

As dimensões da LUT devem corresponder às dimensões da piscina. Por isso, uma maneira de validar a correta criação da LUT, é sobrepor a LUT com uma imagem adquirida pela câmara (ver Fig. 3.18).

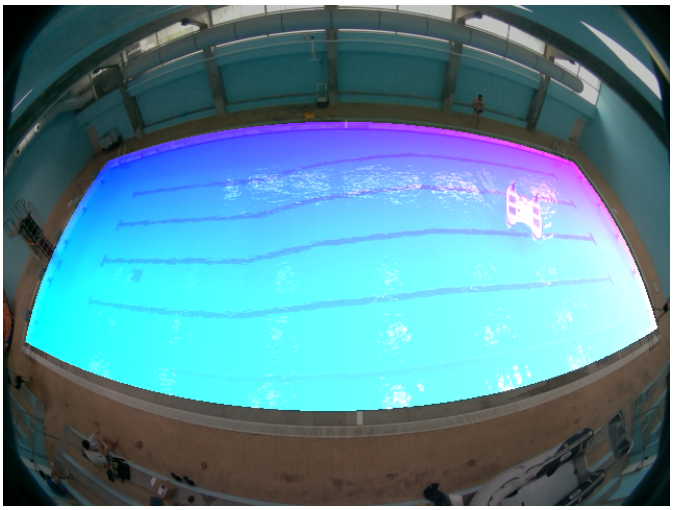


Figura 3.18: Sobreposição da LUT com a piscina

Ao detetar os pixeis correspondentes aos marcadores LED da embarcação na imagem original, é possível obter a sua posição na imagem corrigida e reprojetada utilizando a LUT.

3.2.5 Deteção do USV

Para calcular a posição e orientação da embarcação são necessáriasas etapas apresentadas na Fig. 3.19, nomeadamente, a deteção a deteção dos marcadores, a obtenção dos pontos através da LUT e o cálculo da posição e da orientação da embarcação.

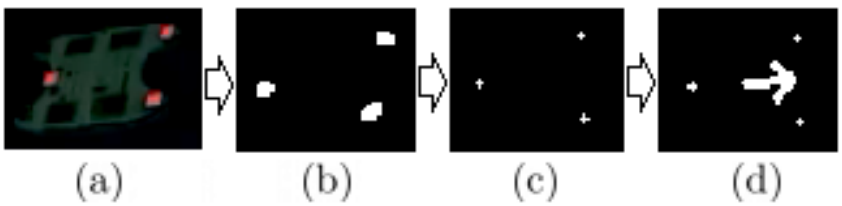


Figura 3.19: (a) Marcadores LED instalados na embarcação. (b) Deteção dos marcadores. (c) Pontos obtidos através da aplicação da LUT. (d) Cálculo do centro e da orientação da embarcação.

O primeiro passo realiza a deteção dos marcadores LED instalados na embarcação em cada imagem adquirida. Em seguida, através da aplicação da LUT, obtém-se a posição dos marcadores na imagem final (imagem sem distorções e reprojetada). Esta posição é convertida para coordenadas métricas com base nas dimensões reais da piscina (25 metros de comprimento e 10 metros de largura). Por fim, calcula-se o centro e a orientação da embarcação através da posição dos marcadores, considerando o canto superior esquerdo da imagem como origem do eixo de coordenadas.

Para facilitar a deteção e localização de um veículo móvel (como é a embarcação utilizada) com base em visão computacional, é comum colocar marcadores fiduciais, relativamente fáceis de detetar devido ao seu padrão característico. Contudo, a resolução da câmara não permite utilizar marcadores passivos uma vez que os marcadores colocados na embarcação, que necessitariam de ter um tamanho semelhante ao de uma folha A4, não seriam reconhecidos quando a embarcação se encontrasse em zonas mais afastadas da câmara. Além disso, a iluminação natural da piscina não produz o contraste necessário para detetar os marcadores de forma robusta. Assim, optou-se por utilizar marcadores ativos, compostos por 3 conjuntos de LED, 2 colocados na frente e 1 colocado na traseira da embarcação (ver Fig. 3.20), que permitem a sua deteção em todas as condições de utilização deste trabalho.

Figura 3.20: USV-enautica1 equipada com LED

Os LED instalados na frente da embarcação têm a 1 metro de distância entre eles enquanto os LED instalados na traseira encontram-se a 1.5 metros da dianteira (ver Fig. 3.21).

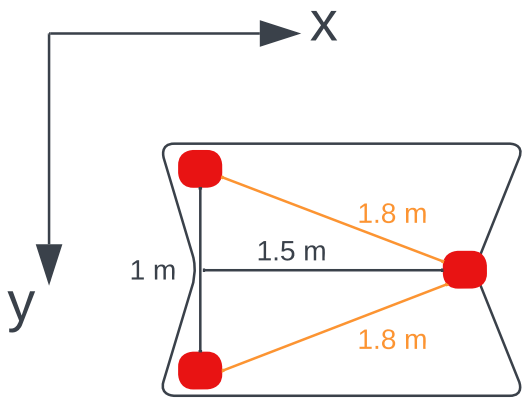


Figura 3.21: USV-enautica1 com LED: medidas

Para detetar os pixeis correspondentes aos marcadores da embarcação, é necessário distingui-los dos restantes pixeis da imagem. Uma vez que os marcadores são constituídos por múltiplos LED, estes podem ser detetados através da sua luminosidade. Para facilitar essa deteção, a sensibilidade da câmara à luz é ajustada (ver Fig. 3.22). Desta forma, é possível distinguir mais facilmente os marcadores LED.

Figura 3.22: Imagem adquirida com sensibilidade da câmara à luz ajustada

Como se pode ver na Fig. 3.22, devido ao ambiente onde a embarcação está inserida, para além nos marcadores, existem outras zonas de alta luminosidade. Por exemplo, a Fig. 3.23 apresenta o resultado obtido depois de converter a imagem para tons de cinzento e aplicação de um *thresholding* com um limiar de 100.

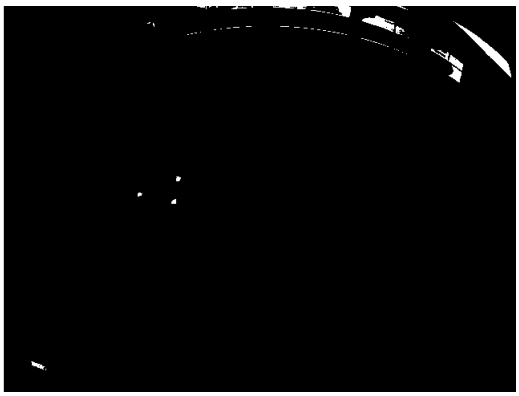


Figura 3.23: Imagem binária para um *threshold* de 100

Para remover as zonas não desejadas, optou-se por filtrar também utilizando a informação de cor na zona do vermelho (cor dos LED utilizados). Para realizar a filtragem, a imagem é convertida para o formato *Hue Saturation Value (HSV)* e são obtidos os pixeis que se encontram no intervalo correspondente à cor vermelha (ver Fig. 3.24).

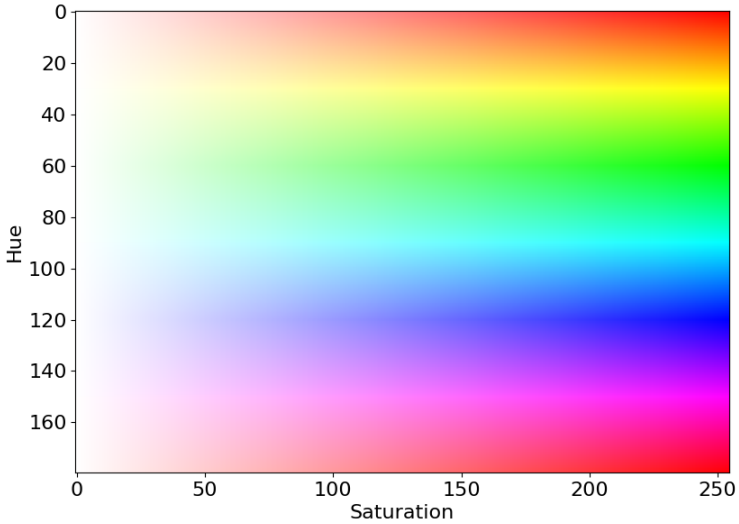


Figura 3.24: Espetro de cor para HSV com V=255

O valor de *Hue* está compreendido entre 0 e 360 graus. A figura apresenta-o entre 0 e 180 pois estes são os valores que a biblioteca OpenCV utiliza (de modo a não representar valores superiores a 255 - limite da estrutura de dados *uint8*)

Analisando a imagem, verifica-se que o vermelho pode ser representado tanto por valores de *Hue* de 0, como por valores de 180, que são contíguos no formato HSV. Por esta razão, são necessários dois intervalos para representar a cor vermelha. O primeiro intervalo está compreendido entre 0 e 20 enquanto o segundo intervalo está compreendido entre 175 e 180. Ainda analisando a imagem, é possível verificar que os intervalos escolhidos, representam cores pouco semelhantes à cor dos marcadores para valores de *Saturation* mais pequenos. Por esta razão, filtra-se todas as cores que tenham valores de *Saturation* menores que 80 em cada um dos intervalos. Para adicionar o filtro de luminosidade aos intervalos, inclui-se o valor de *Value* obtendo-se os dois intervalos (H,S,V): (0, 80, 100) até (20, 255, 255) e (175, 80, 100) até (180, 255, 255).

A Fig. 3.25 apresenta a máscara obtida aplicando os intervalos à imagem no formato HSV.

Figura 3.25: Máscara obtida aplicando os intervalos

Os pixeis correspondentes a cada marcador podem apresentar regiões separadas. Para tornar a deteção mais robusta, aplicou-se uma dilatação à imagem com um elemento estruturante de 3x3. O resultado é apresentado na Fig. 3.26.

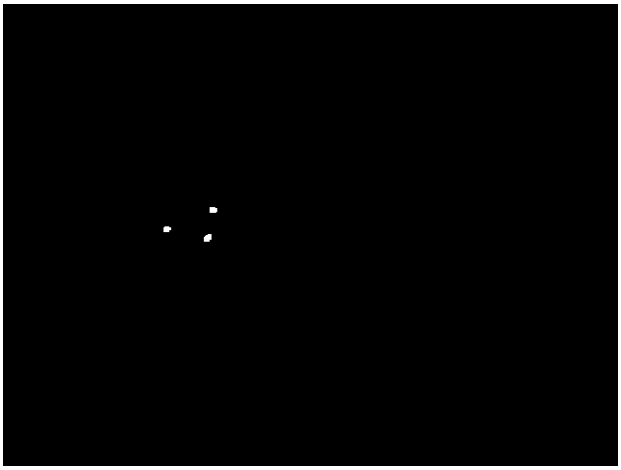


Figura 3.26: Máscara após dilatação

Com os pixeis correspondentes aos marcadores detetados, são obtidas as regiões ativas e é calculado o centróide de cada região. O centróide representa a posição de cada marcador.

Com a posição dos marcadores calculada, é utilizada a LUT para obter a posição correspondente na imagem final. Esta posição pode então ser convertida para valores referentes à piscina.

Com base nos pontos obtidos através da LUT, referentes aos marcadores de LED na imagem final, é possível estimar a posição e orientação da embarcação.

A embarcação possui 2 marcadores na frente da embarcação, separados por 1 metro de distância. Um terceiro marcador encontra-se na traseira da embarcação a 1.5 metros de distância da frente da embarcação. Para além da posição, esta colocação permite calcular a orientação da embarcação pois é possível saber quais marcadores representam a frente da embarcação e qual representa a traseira ao comparar as distâncias entre eles (ver Fig. 3.21 referente às distâncias entre marcadores).

A média entre os pontos da frente é calculada para obter um único ponto que representa a frente da embarcação. Com estes dois pontos é possível calcular a posição (ponto médio entre eles) e a orientação da embarcação.

Para que o percurso da embarcação ao longo do tempo seja o mais suave possível e para que o seu cálculo seja mais robusto, é utilizada uma média temporal móvel com as últimas 10 posições dos pontos da embarcação. Para

além disso, a aplicação verifica se cada ponto é um *outlier*. Esta verificação é efetuada comparando a distância do ponto atual ao último ponto obtido. Se esta distância for muito grande, o ponto é considerado como um *outlier* e descartado.

3.2.6 Controlo do USV

Depois de calcular a posição e orientação, estas podem ser utilizadas para controlar a embarcação. O objetivo, nesta fase, é controlar a embarcação de forma autónoma para que esta siga uma trajetória pré-determinada. Para atingir este objetivo, é utilizado um controlador que em função do erro, diferença entre a posição da embarcação e a posição da trajetória mais próxima, produz um valor de saída para minimizar esse erro.

Para calcular o erro, obtém-se o ponto da trajetória mais próximo da embarcação e calcula-se o vetor entre os dois pontos. A Fig. 3.27 apresenta um exemplo da distância (`d`) entre a embarcação com posição no ponto `P` e orientação igual à da seta contínua e a trajetória desejada a tracejado com orientação igual à da seta tracejada.

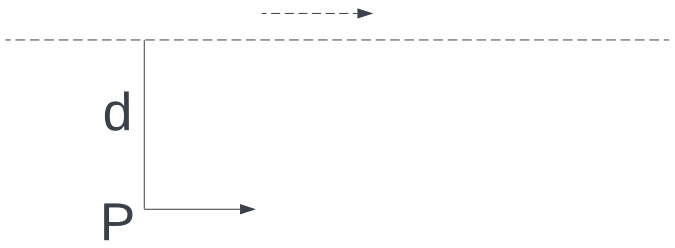


Figura 3.27: Erro de posição (`d`)

A embarcação irá navegar apenas na horizontal. Por essa razão, podemos simplificar a implementação e controlar os motores consoante o sinal da ordenada do erro. Para tal, obtém-se o sinal de ordenada do vetor diferença entre os dois pontos e calcula-se o erro (ver código 6).

Código 6: Cálculo do erro

```
1 distance_vec = (target[0]-position[0], target[1]-position[1])
2
3 s = np.sign(distance_vec[1])
4 d = math.sqrt((distance_vec[0]**2) + (distance_vec[1]**2))
```

```
5 e = s * d
```

O erro calculado é utilizado pelos controladores. O valor de saída dos controladores (`AC`) possui o mesmo sinal do erro e pode ser utilizado para controlador os motores (ver código 7).

Código 7: Cálculo da potência dos motores

```
1 left_engine = AV - AC
2 right_engine = AV + AC
```

Desta forma, quando a embarcação se encontra abaixo da trajetória, o erro será positivo e o motor direito irá ter uma potência maior que o motor esquerdo, fazendo com que a embarcação se aproxime da trajetória. O oposto acontece se a embarcação se encontrar acima da trajetória.

Para haver um maior controlo da embarcação, o valor de `AC` é limitado a um valor máximo, tal como o ângulo máximo que a embarcação pode alcançar.

Capítulo 4

Resultados Práticos

4.1 Rastreamento

Um dos principais objetivos deste trabalho é o cálculo da posição e orientação da embarcação, em tempo real, para que esta possa ser controlada de forma autónoma.

A utilização da LUT é um ponto chave deste trabalho. Para validar o método proposto, este foi comparado com as diferentes etapas do processo normal: remoção de distorções e reprojeção. Neste teste, cada abordagem é executada 10 vezes num *Raspberry Pi 2* para diferentes resoluções e o tempo de processamento necessário é registado (ver Fig. 4.1).

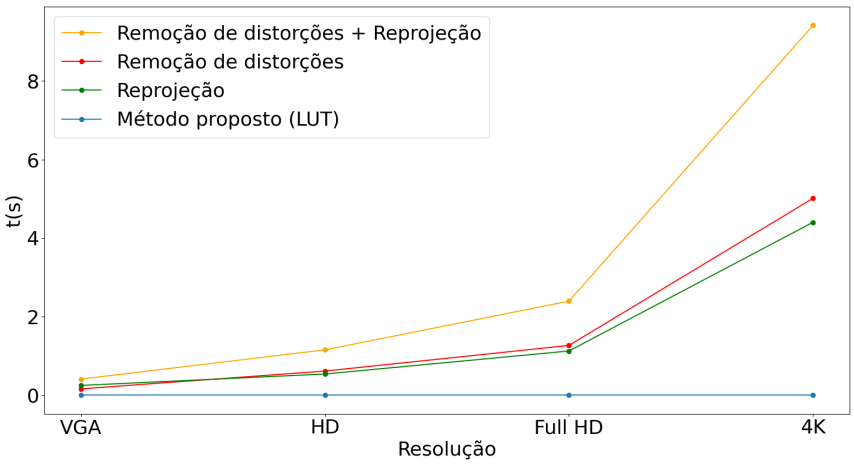


Figura 4.1: Tempo de processamento das diferentes abordagens

Analisando o gráfico, observa-se que o tempo de processamento para o

processo normal aumenta de forma quadrática, complexidade $O(n^2)$, com o aumento da resolução de imagem. Ao mesmo tempo, é possível perceber que o tempo utilizado pelo método proposto permanece constante independentemente da resolução, complexidade $O(1)$. A tabela 4.1 apresenta, com maior detalhe, os tempos de cada etapa para cada resolução.

	VGA	**HD**	**FHD**	**4K**
Remoção de distorções	0.1620 +-0.0007	0.6153 +-0.0046	1.2685 +-0.0038	5.0127 +-0.0097
Re-projeção	0.2496 +-0.0004	0.5404 +-0.0014	1.1266 +-0.0038	4.4036 +-0.0373
Remoção de distorções + Re-projeção	0.4116 +-0.0011	1.1557 +-0.0060	2.3951 +-0.0076	9.4163 +-0.0470
Método proposto LUT	0.0012 +-6.4e-05	0.0012 +-0.0001	0.0012 +-0.0002	0.0012 +-0.0002

Tabela 4.1: Tempo para as diferentes etapas do processo normal e para o método LUT (valores em segundos)

Para o processo sem LUT, a aplicação necessita de 0.4116 segundos para executar 10 iterações para imagens com resolução VGA, que se pode traduzir em 24 *frames* por segundo. Utilizando imagens com resolução 4K, são necessários 9.4163 segundos, ou seja, 1 *frame* por segundo. Utilizando o método proposto (LUT), a aplicação necessita 0.0012 segundos, para qualquer resolução de imagem utilizada, traduzindo-se em 8333 *frames* por segundo.

Estes resultados provam que a abordagem com a LUT apresenta resultados melhores quando comparada com a abordagem normal, atingindo uma complexidade $O(1)$ independentemente da resolução de imagem utilizada.

Devido a questões logísticas, a câmara não está colocada diretamente por cima da piscina e por isso existe a necessidade de aplicar uma transformação homográfica. Pela mesma razão, existem menos pixeis para representar os lados da piscina mais distantes da câmara.

Ao aplicar os passos de remoção de distorções e reprojeção à imagem normalizada, é criada uma imagem que representa a piscina sem distorções e reprojetada. Cada valor desta imagem pode ser transformado numa posição válida (ver código 4). Depois de aplicar estes passos, é possível obter o número de pixeis ao calcular a distância entre cada posição. Assumindo

uma piscina com 25m de comprimento por 10m de largura, os eixos x e y podem ter um tamanho de 250 por 100 pixeis, respetivamente. Desta forma, é possível calcular os mm por pixel dividindo 100 (1000 se fossem utilizadas as dimensões 25 por 10 pixeis) pela distância calculada em cada posição. A Fig. 4.2 e 4.3 apresentam os mm por pixel calculados para cada eixo.

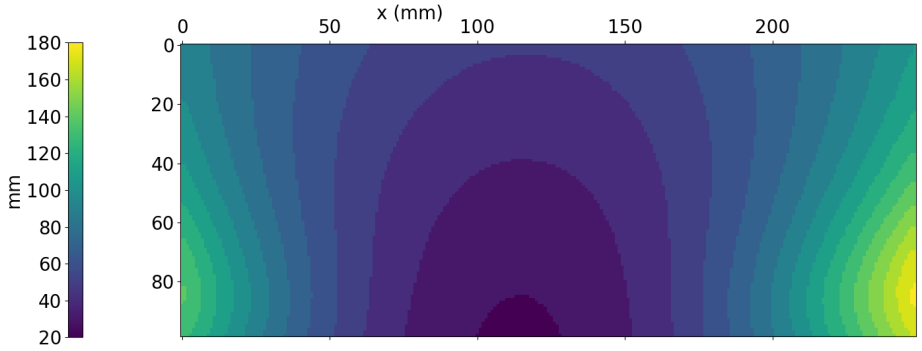


Figura 4.2: mm por pixel para o eixo x

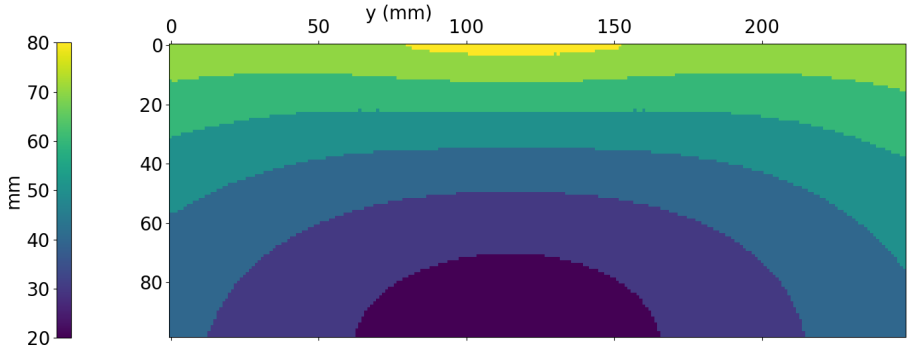


Figura 4.3: mm por pixel para o eixo y

Na Fig. 4.2, a área central, onde os valores do eixo das ordenadas são mais elevados, apresenta o menor valor de mm por pixel, ou seja, existem mais pixeis para representar uma área da piscina menor. Esta é a área mais perto da câmara. Nos extremos do eixo x, é onde existem mais mm por pixel, uma vez são as zonas mais distantes da câmara. O valor é maior de um dos lados devido ao facto da câmara não estar centrada. A mesma análise pode ser feita na Fig. 4.3 para o eixo y.

A interseção dos dois gráficos permite perceber melhor quais as áreas da

piscina onde o erro de localização seria mais reduzido. O gráfico resultante é apresentado na Fig.4.4.

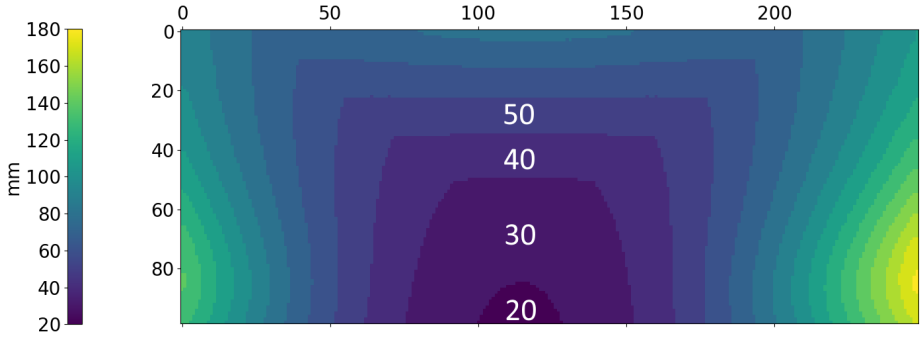


Figura 4.4: Interseção entre os gráficos das Fig. 4.2 e Fig. 4.3

Na área com menos mm por pixel, um erro de um pixel na deteção da posição da embarcação significa um erro menor que 20mm. Outras zonas possuem uma proporção de mm por pixel maior ou são zonas muito próximas do limite da piscina que devem ser evitadas para não ocorrerem colisões com esses limites. Assim, a zona central da piscina é a mais adequada para navegar.

São ainda estudados os erros de posição e orientação. A embarcação navega por trajetórias pré-determinadas e a sua posição e orientação são obtidas através dos algoritmos descritos anteriormente. Estes valores são comparados com os valores das trajetórias ideais. A Fig. 4.5 apresenta uma trajetória real navegada pela embarcação, dividida em 3 rotas assinaladas a vermelho.

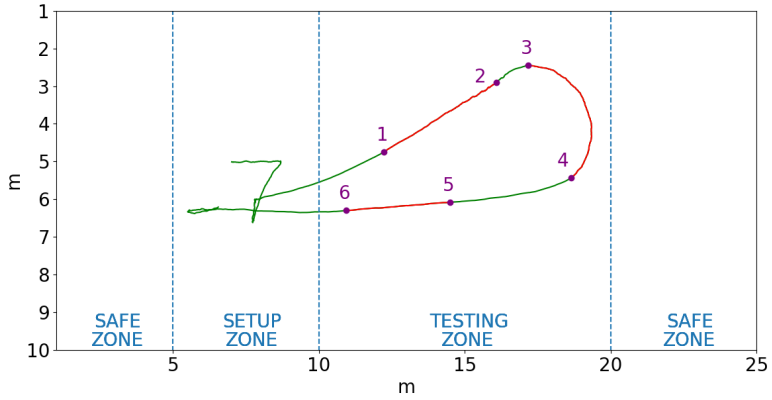


Figura 4.5: Trajetória real (verde), dividida em 3 rotas (vermelho)

A piscina pode ser dividida em 4 zonas: 2 zonas de segurança nos limites da piscina para evitar colisões (*safe zone*), 1 zona para posicionar e orientar a embarcação em preparação para os testes (*setup zone*) e 1 zona de testes onde a embarcação navega (*testing zone*). Na Fig. 4.5 é possível observar as 3 rotas em estudo.

O objetivo é a embarcação navegar uma primeira linha reta (rota 1-2), em seguida uma curva (rota 3-4) e por último outra linha reta (rota 5-6).

A trajetória ideal para as rotas 1-2 e 5-6 é obtida utilizando o ponto inicial e final de cada uma, obtendo uma reta entre os dois pontos. Para a rota 3-4, é calculado um arco que passa no ponto inicial e no ponto final da rota.

Para cada ponto de cada rota navegada, o ponto mais próximo da trajetória ideal é calculado e os erros de posição e orientação são calculados.

A Fig. 4.6 apresenta a rota 1-2, a vermelho, e a rota ideal (reta entre o primeiro e último ponto) a laranja.

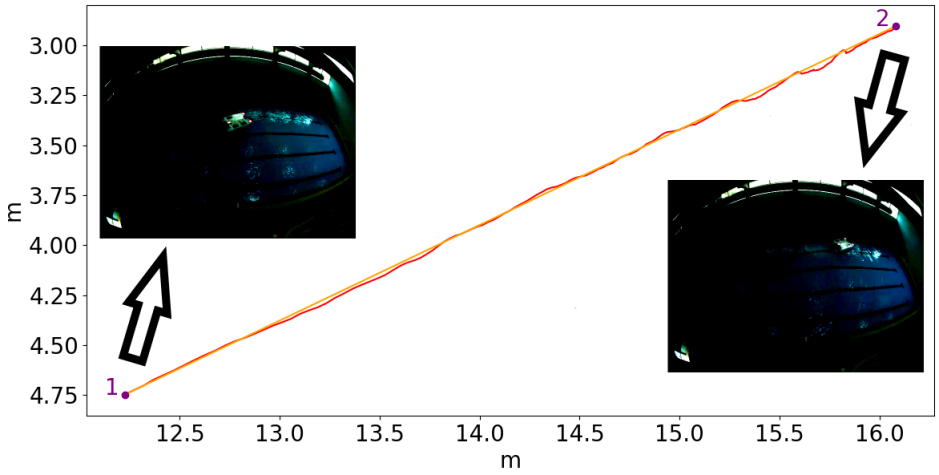


Figura 4.6: Trajetória 1-2

Os erros de posição e orientação são apresentados nas Fig. 4.7 e 4.8, respetivamente.

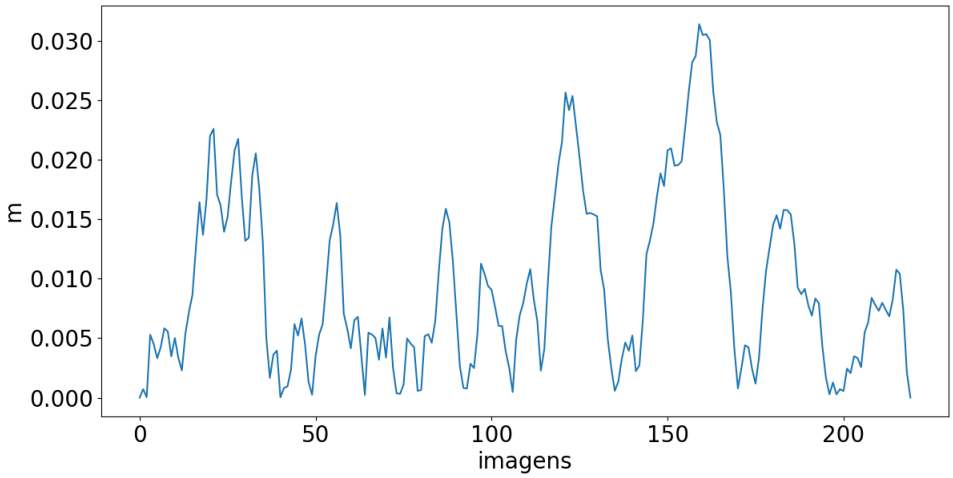


Figura 4.7: Erros de posição na trajetória 1-2

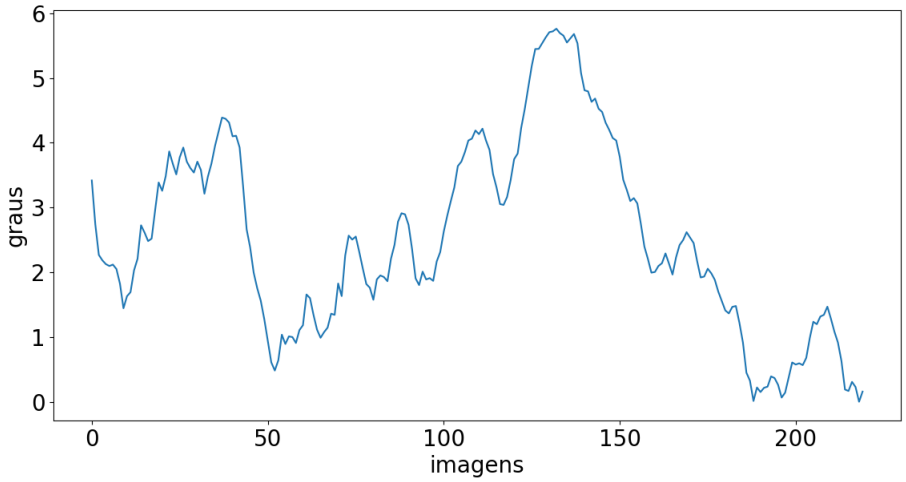


Figura 4.8: Erros de orientação na trajetória 1-2

A segunda trajetória (rota 3-4) é apresentada na Fig. 4.9 juntamente com a trajetória ideal, a vermelho e laranja respetivamente.

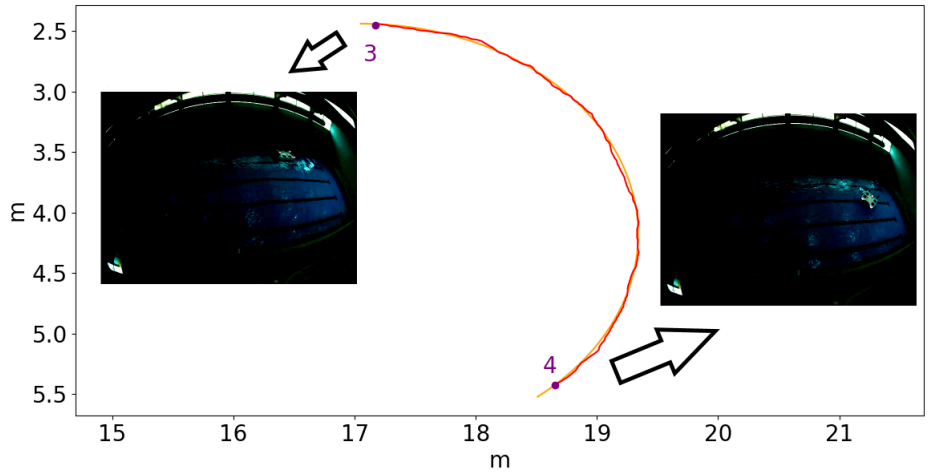


Figura 4.9: Trajetória 3-4

Os erros desta trajetória também são calculados da mesma forma. Na Fig. 4.10 são apresentados os erros de posição.

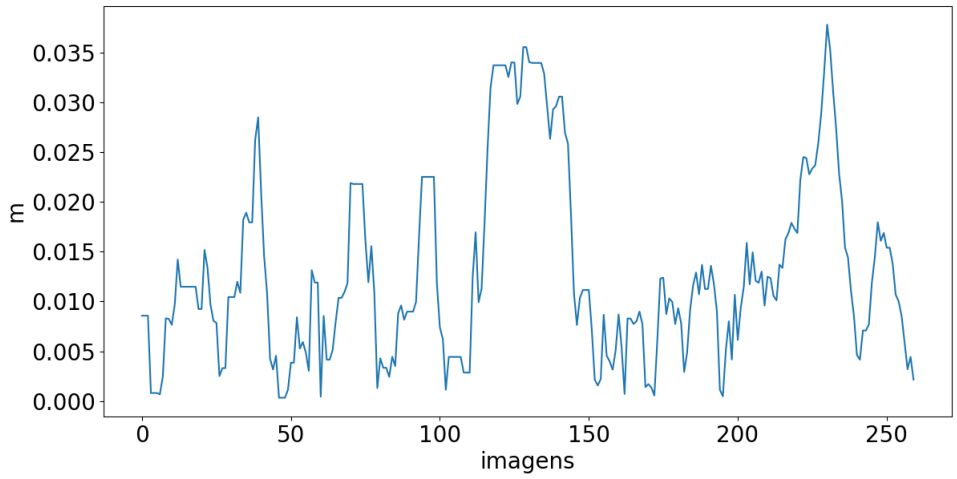


Figura 4.10: Erros de posição na trajetória 3-4

Na Fig. 4.11 são apresentados os erros de orientação.

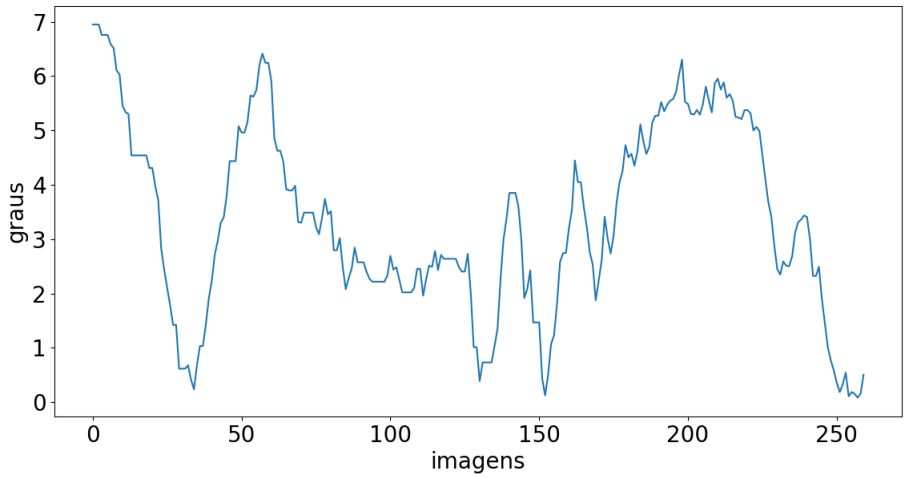


Figura 4.11: Erros de orientação na trajetória 3-4

A Fig. 4.12 apresenta, a vermelho, a última trajetória (rota 5-6) e, a laranja, a trajetória ideal.

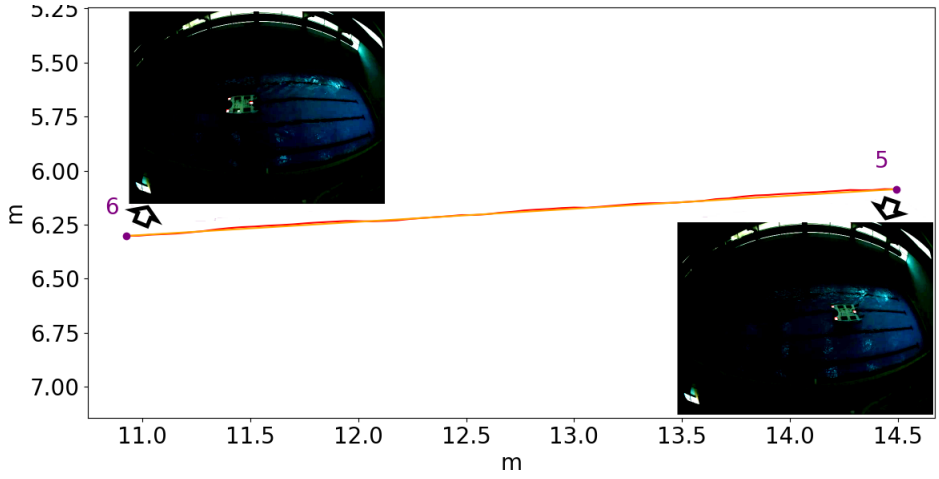


Figura 4.12: Trajetória 5-6

Tal como nas trajetórias anteriores, o erros de posição (ver Fig. 4.13) e orientação (ver Fig. 4.14) são calculados.

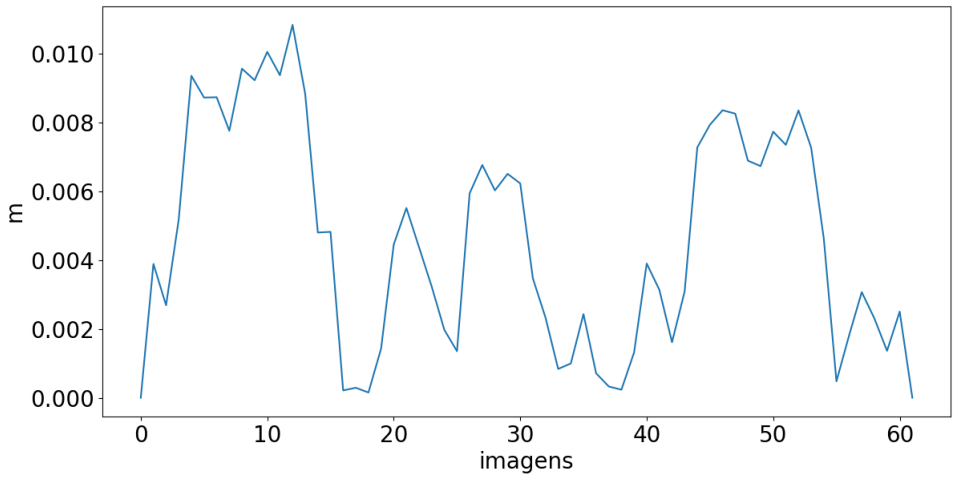


Figura 4.13: Erros de posição na trajetória 5-6

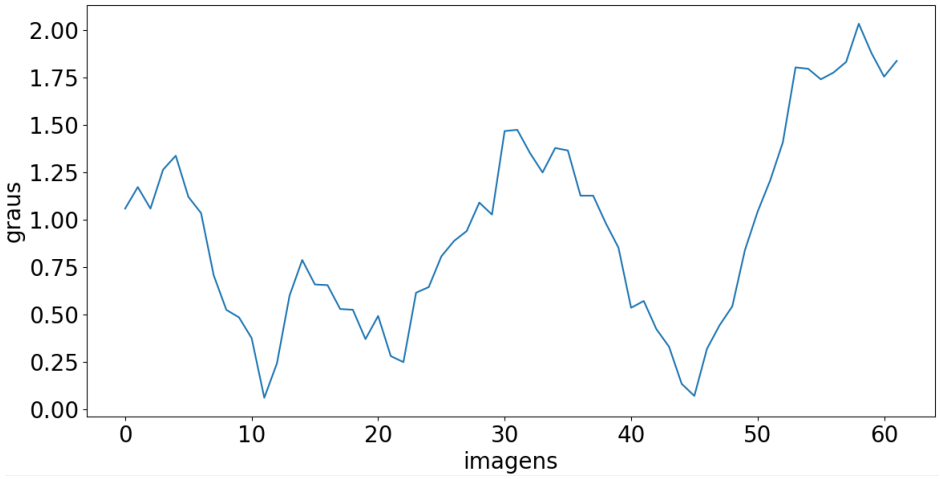


Figura 4.14: Erros de orientação na trajetória 5-6

Adicionalmente, foi calculada a métrica *Mean Absolute Error* (MAE), com os erros de posição (tabela 4.2) e orientação (tabela 4.3), para mais facilmente comparar as diferentes trajetórias.

Trajetória	**1-2**	**3-4**	**5-6**
MAE	9.6 mm	12.92 mm	4.60mm

Tabela 4.2: MAE com os erros de posição

Trajetória	**1-2**	**3-4**	**5-6**
MAE	2.51^{o}	3.35^{o}	0.94

Tabela 4.3: MAE com os erros de orientação

Analisando os resultados obtidos, é possível perceber que as primeiras duas rotas, 1-2 e 3-4, são as que apresentam valores de erros mais elevados, tanto para a posição como para a orientação.

A Fig. 4.15 apresenta uma sobreposição entre a Fig. 4.4 e a trajetória da embarcação (Fig. 4.5) para comparar esta com as zonas da piscina onde um erro mais elevado pode ocorrer.

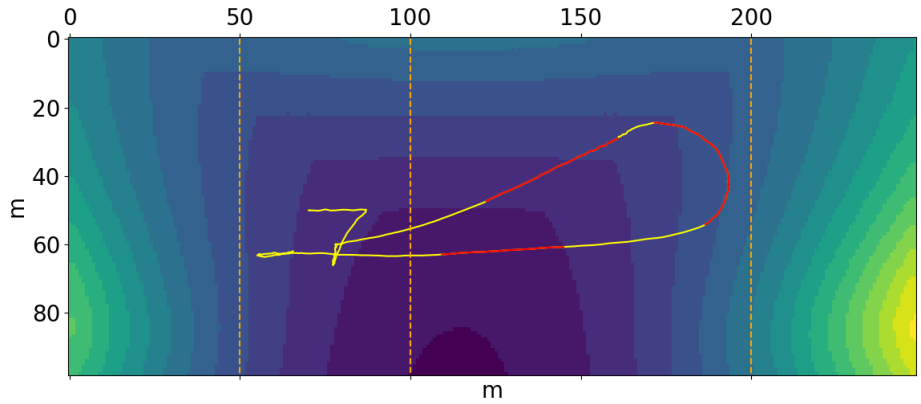


Figura 4.15: Sobreposição entre a Fig. 4.4 e a trajetória da embarcação

Analisando os resultados, é possível afirmar que a zona da piscina mais adequada para a navegação da embarcação, é a zona central, o mais próximo possível da câmara, onde a trajetória 5-6 ocorreu. Além disso, há mais erro na trajetória 3-4 em relação à 1-2, pelo mesmo motivo.

4.2 Controlo

Por último, temos os resultados da utilização dos dois controladores: controlador P e controlador PD. De modo a evitar o uso da embarcação, foi desenvolvido um simulador, através da biblioteca *Pygame*, que permite a aplicação dos controladores a um objeto virtual que representa a embarcação.

O ambiente do simulador implementado possui a mesma proporção, em termos de comprimento por largura, da piscina onde a embarcação se encon-

tra. A velocidade de deslocação da embarcação virtual é limitada a 2 m/s e a rotação a 30 graus/s, para que o comportamento desta seja mais realista.

Nos testes efetuados, os controladores são aplicados à embarcação virtual que deve seguir uma trajetória pré-definida. As trajetórias são representadas na simulação a laranja, o percurso percorrido pela embarcação virtual a vermelho e a embarcação virtual é representada como um círculo branco com um linha preta a representar a dianteira, que permite perceber melhor qual a orientação da embarcação virtual num determinado instante.

O controlador P apresenta um comportamento oscilatório. Como se pode ver na Fig. 4.16, a embarcação virtual aproxima-se da trajetória mas ultrapassa-a e afasta-se da mesma. Este comportamento repete-se sem a aproximação da embarcação virtual à trajetória estabilizar.

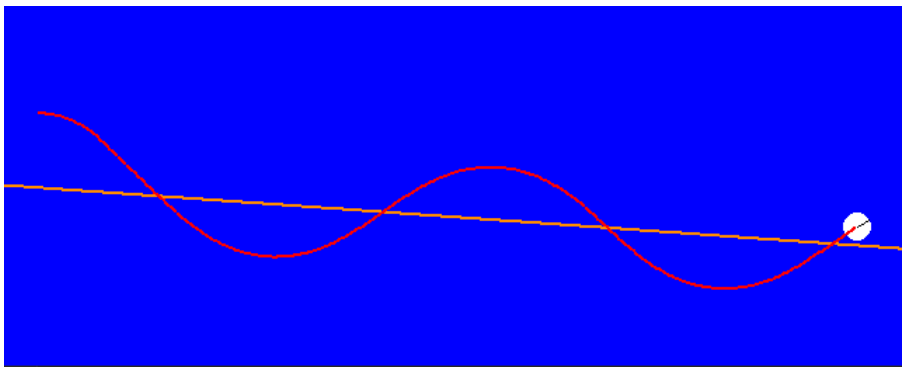


Figura 4.16: Controlador P com $k_p = 0.4$

Aumentando o valor de k_p (ganho), observa-se uma diminuição da distância da embarcação virtual à trajetória mas o comportamento oscilatório mantém-se, como se pode verificar na Fig. 4.17.

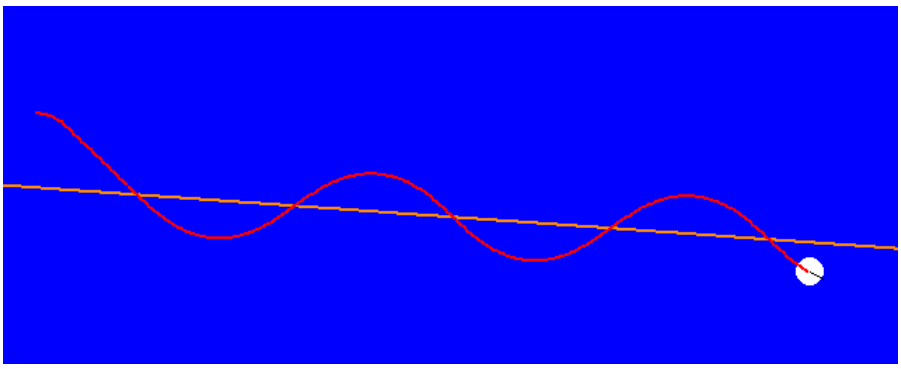


Figura 4.17: Controlador P com $k_p = 0.8$

Este comportamento significa que a embarcação virtual não percorre a trajetória, e, em vez disso, tem um movimento oscilatório ao redor desta. O aumento do valor de k_p permite reduzir a oscilação mas não a elimina.

Por sua vez, o controlador PD permite corrigir este comportamento oscilatório. Como se pode ver na Fig. 4.18, a embarcação virtual aproxima-se da trajetória, ultrapassa-a e afasta-se da mesma mas, em cada correção, a embarcação virtual afasta-se menos da trajetória até estabilizar sobre a trajetória.

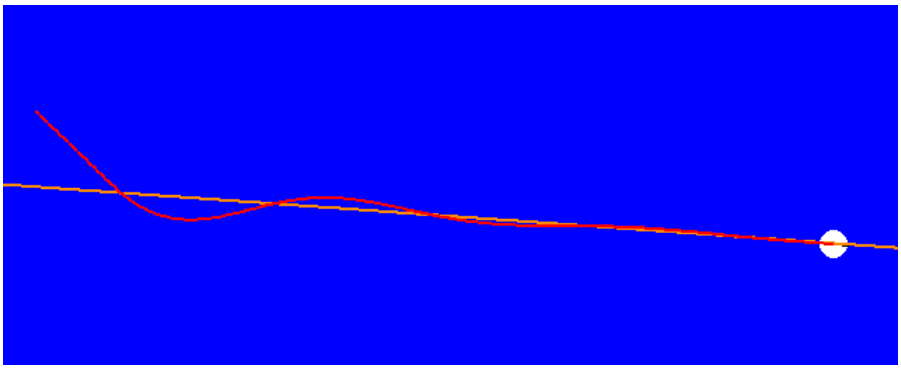


Figura 4.18: Controlador PD com $k_p = 1.0$ e $k_d = 0.1$

Com o aumento do valor de k_d, é possível evitar que a embarcação virtual ultrapasse a trajetória e a sua posterior reaproximação à mesma, removendo a oscilação inicial, como se pode ver na Fig. 4.19.

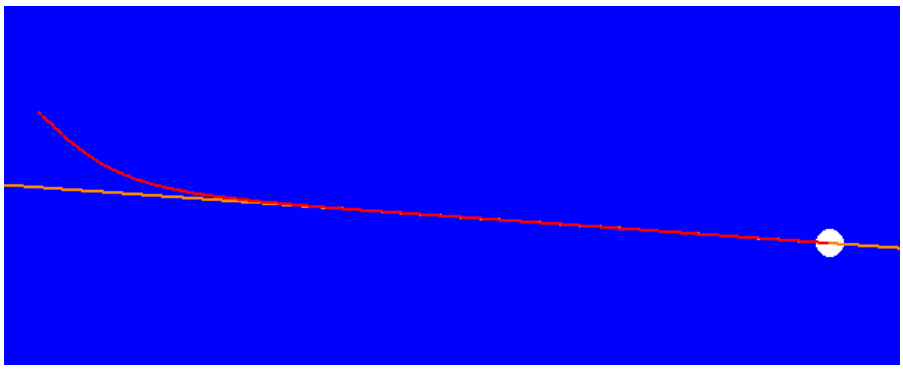


Figura 4.19: Controlador PD com $k_p = 1.0$ e $k_d = 0.4$

Para valores de k_d ainda mais elevados, verifica-se uma estabilização mais lenta, com a embarcação virtual a demorar mais tempo a alcançar a trajetória pretendida, como se verifica na Fig. 4.20.

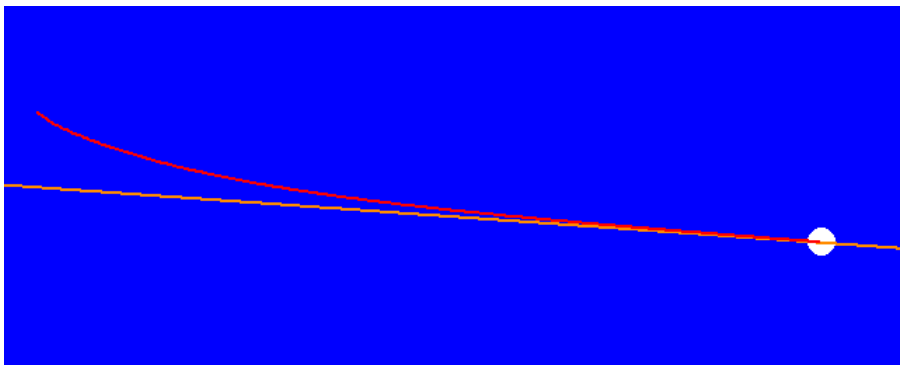


Figura 4.20: Controlador PD com $k_p = 1.0$ e $k_d = 0.8$

Analisando os resultados obtidos, é possível afirmar que o controlador PD apresenta um desempenho melhor que o controlador P. O controlador P apresenta um comportamento oscilatório, não permitindo que siga a trajetória corretamente. A alteração do valor de k_p permite modificar essa oscilação mas não a elimina. O controlador PD demonstrou resultados positivos ao atingir e percorrer a trajetória. É, no entanto, necessário encontrar os parâmetros mais favoráveis para maximizar o seu desempenho.

Capítulo 5

Conclusões e Trabalho Futuro

Neste capítulo são apresentadas as conclusões do trabalho desenvolvido e dos resultados obtidos. É também sugerido algum trabalho futuro a desenvolver com o objetivo de evoluir o projeto *Sea2future*, onde esta tese se insere.

5.1 Conclusões

Este trabalho descreve um sistema para calcular a posição e orientação de uma embarcação, equipada com marcadores LED, num ambiente interior, através do uso de uma câmara fixa com uma lente *fish-eye* ligada a um *Raspberry Pi*.

São descritos os processos de calibração da câmara para a posterior correção de distorções e a aplicação de uma transformação homográfica para reprojetar a imagem com o intuito de obter virtualmente uma visão de cima da piscina.

É ainda apresentado um método baseado no uso de uma LUT para melhorar o desempenho da aplicação ao substituir os passos de correção de distorções e de reprojeção da imagem.

Em seguida, é descrito o algoritmo de deteção dos marcadores da embarcação para calcular a posição e orientação desta.

Por último, é utilizado um controlador PD, com recurso a um simulador implementado através da biblioteca *Pygame*, para estudar e estimar o comportamento da embarcação ao seguir uma trajetória pré-determinada, utilizando a posição e orientação calculadas.

Os resultados obtidos provam que a utilização da LUT melhora o processo, apresentando valores de execução quase instantâneos para qualquer resolução, complexidade $O(1)$. É também apresentado o número de mm que representam cada pixel da piscina, bem como os erros obtidos referentes à posição e orientação calculadas para diferentes rotas. Ambos permitem concluir que a zona central da piscina, próxima à câmara, é a mais adequada para a navegação da embarcação.

A aplicação dos controladores no simulador desenvolvido permitiu concluir que o controlador PD apresenta um desempenho superior ao do controlador P. A utilização do controlador P provoca um comportamento oscilatório na embarcação virtual da simulação, fazendo com que esta não siga a trajetória desejada. O controlador PD demonstrou resultados superiores, uma vez que com a sua aplicação, a embarcação virtual foi capaz de seguir a trajetória desejada, desde que os seus parâmetros sejam definidos corretamente.

5.2 Trabalho Futuro

Um dos próximos passos é substituir a simulação implementada pela aplicação do controlador PD à embarcação real na piscina. Para que esta substituição seja completa, o módulo do controlo deve utilizar a posição e orientação da embarcação e por isso deve ser integrado com o módulo de rastreio, onde a posição e orientação da embarcação são calculadas em tempo real. Os resultados obtidos no ambiente da piscina serão certamente diferentes dos obtidos na simulação realizada devido a fatores como o próprio comportamento da embarcação que será diferente do comportamento do objeto da simulação (e.g. velocidade) e por isso é necessário estudar os parâmetros a utilizar no controlador para maximizar o seu desempenho.

Ainda no mesmo tópico, a utilização da embarcação na piscina irá introduzir novas condições como, por exemplo, a ondulação da água. Estas novas condições podem afetar o desempenho do controlador PD e por isso seria interessante implementar e testar outros tipos de controladores como, por exemplo, um controlador PID, mais robusto ao ruído.

Pode ainda ser dado um maior ênfase à navegação autónoma da embarcação para que esta siga trajetórias mais complexas para preparar melhor a introdução da embarcação no oceano/rio.

Bibliografia

[1] *SEA2FUTURE*. URL: https://sea2future.pt/. (Acedido a 20 de Julho de 2022).

[2] Mário Assunção et al. «Design of an Underactuated USV Catamaran». Em: *APCA International Conference on Automatic Control and Soft Computing*. Springer. 2022, pp. 656–666.

[3] *Raspberry Pi*. URL: https://www.raspberrypi.com/documentation/. (Acedido a 15 de Agosto de 2022).

[4] *ROS - Robot Operating System*. URL: https://www.ros.org/. (Acedido a 20 de Agosto de 2022).

[5] Luca Mainetti, Luigi Patrono e Ilaria Sergi. «A survey on indoor positioning systems». Em: *2014 22nd international conference on software, telecommunications and computer networks (SoftCOM)*. IEEE. 2014, pp. 111–120.

[6] Taiga Arai et al. «Evaluation of indoor positioning system based on attachable infrared beacons in metal shelf environment». Em: *2019 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE. 2019, pp. 1–4.

[7] Erwin Aitenbichler e Max Muhlhauser. «An IR local positioning system for smart items and devices». Em: *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.* IEEE. 2003, pp. 334–339.

[8] Thibaut Raharijaona et al. «Local positioning system using flickering infrared leds». Em: *Sensors* 17.11 (2017), p. 2518.

[9] Jens-Steffen Gutmann et al. «Challenges of designing a low-cost indoor localization system using active beacons». Em: *2013 IEEE conference on technologies for practical robot applications (TePRA)*. IEEE. 2013, pp. 1–6.

[10] Seong Jin Kim e Byung Kook Kim. «Dynamic ultrasonic hybrid localization system for indoor mobile robots». Em: *IEEE Transactions on Industrial Electronics* 60.10 (2012), pp. 4562–4573.

[11] Faheem Ijaz et al. «Indoor positioning: A review of indoor ultrasonic positioning systems». Em: *2013 15th International Conference on Advanced Communications Technology (ICACT)*. IEEE. 2013, pp. 1146–1150.

[12] Carlos Medina, José Carlos Segura e Angel De la Torre. «Ultrasound indoor positioning system based on a low-power wireless sensor network providing sub-centimeter accuracy». Em: *Sensors* 13.3 (2013), pp. 3501–3526.

[13] José Luis Carrera et al. «A real-time indoor tracking system in smartphones». Em: *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. 2016, pp. 292–301.

[14] Weixiao Meng et al. «Optimized access points deployment for WLAN indoor positioning system». Em: *2012 IEEE wireless communications and networking conference (WCNC)*. IEEE. 2012, pp. 2457–2461.

[15] Fan Wang et al. «EESM-based fingerprint algorithm for Wi-fi indoor positioning system». Em: *2013 IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE. 2013, pp. 674–679.

[16] Luca Catarinucci, Salvatore Tedesco e Luciano Tarricone. «Customized ultra high frequency radio frequency identification tags and reader antennas enabling reliable mobile robot navigation». Em: *IEEE Sensors Journal* 13.2 (2012), pp. 783–791.

[17] Charalampos Tsirmpas et al. «An indoor navigation system for visually impaired and elderly people based on Radio Frequency Identification (RFID)». Em: *Information Sciences* 320 (2015), pp. 288–305.

[18] Lei Yang et al. «A hybrid method for achieving high accuracy and efficiency in object tracking using passive RFID». Em: *2012 IEEE International Conference on Pervasive Computing and Communications*. IEEE. 2012, pp. 109–115.

[19] Dawar Khan, Sehat Ullah e Syed Nabi. «A generic approach toward indoor navigation and pathfinding with robust marker tracking». Em: *Remote Sensing* 11.24 (2019), p. 3052.

[20] Jason Zhi Liang et al. «Image based localization in indoor environments». Em: *2013 Fourth International Conference on Computing for Geospatial Research and Application*. IEEE. 2013, pp. 70–75.

[21] Jae-Hong Shim e Young-Im Cho. «A mobile robot localization using external surveillance cameras at indoor». Em: *Procedia Computer Science* 56 (2015), pp. 502–507.

[22] Daniel Gehrig et al. «Asynchronous, photometric feature tracking using events and frames». Em: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 750–765.

[23] João Dias e Pedro Mendes Jorge. «People tracking with multi-camera system». Em: *Proceedings of the 9th International Conference on Distributed Smart Cameras*. 2015, pp. 181–186.

[24] Mikhail Mozerov et al. «A simple method of multiple camera calibration for the joint top view projection». Em: *Computer Recognition Systems 2*. Springer, 2007, pp. 164–170.

[25] Long Sha et al. «End-to-end camera calibration for broadcast videos». Em: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 13627–13636.

[26] Anthony Cioppa et al. «Camera calibration and player localization in soccernet-v2 and investigation of their representations for action spotting». Em: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 4537–4546.

[27] Juho Kannala e Sami S Brandt. «A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses». Em:

IEEE transactions on pattern analysis and machine intelligence 28.8 (2006), pp. 1335–1340.

[28] *OpenCV: Camera Calibration and 3D Reconstruction*. URL: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html. (Acedido a 01 de Junho de 2022).

[29] *OpenCV: Camera Calibration and 3D Reconstruction*. URL: https://docs.opencv.org/4.x/db/d58/group__calib3d__fisheye.html. (Acedido a 01 de Junho de 2022).

[30] *Matlab: Fisheye Calibration Basics*. URL: https://www.mathworks.com/help/vision/ug/fisheye-calibration-basics.html. (Acedido a 01 de Junho de 2022).

[31] Elan Dubrofsky. «Homography estimation». Em: *Diplomová práce. Vancouver: Univerzita Britské Kolumbie* 5 (2009).

[32] *OpenCV: Camera Calibration and 3D Reconstruction*. URL: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html. (visited: 01/06/2022).

[33] *OpenCV: Geometric Image Transformations*. URL: https://docs.opencv.org/4.x/da/d54/group__imgproc__%20transform.html. (visited: 01/06/2022).

[34] *Canny Edge Detection*. URL: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html. (Acedido a 30 de Agosto de 2022).

[35] *Sobel Derivatives*. URL: https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html. (Acedido a 30 de Agosto de 2022).

[36] *Harris Corner Detection*. URL: https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html. (Acedido a 11 de Fevereiro de 2023).

[37] *Shi-Tomasi Corner Detector*. URL: https://docs.opencv.org/3.4/d4/d8c/tutorial_py_shi_tomasi.html. (Acedido a 11 de Fevereiro de 2023).

[38] David G Lowe. «Distinctive image features from scale-invariant keypoints». Em: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[39] *OpenCV: Feature Detection and Description*. URL: `https://docs.opencv.org/4.x/d7/d60/classcv_1_1SIFT.html`. (Acedido a 01 de Junho de 2022).

[40] *OpenCV: Feature Detection and Description*. URL: `https://docs.opencv.org/4.x/d0/d13/classcv_1_1Feature2D.html`. (Acedido a 01 de Junho de 2022).

[41] Nida M Zaitoun e Musbah J Aqel. «Survey on image segmentation techniques». Em: *Procedia Computer Science* 65 (2015), pp. 797–806.

[42] *Types of Controllers*. URL: `https://www.electrical4u.com/types-of-controllers-proportional-integral-derivative-controllers/`. (Acedido a 16 de Outubro de 2022).

[43] *Types of Controllers*. URL: `https://electronicscoach.com/types-of-controllers.html`. (Acedido a 16 de Outubro de 2022).

[44] Sumukh Surya e DB Singh. «Comparative study of P, PI, PD and PID controllers for operation of a pressure regulating valve in a blow-down wind tunnel». Em: *2019 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)*. IEEE. 2019, pp. 1–3.

[45] JS Madugu, PG Vasira et al. «Modeling and performance evaluation of P, PI, PD and PID temperature controller for water bath». Em: *American Academic Scientific Research Journal for Engineering, Technology, and Sciences* 47.1 (2018), pp. 186–200.

[46] Paulo Moura Oliveira e John D Hedengren. «An APMonitor temperature lab PID control experiment for undergraduate students». Em: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. 2019, pp. 790–797.

Apêndice A

UML

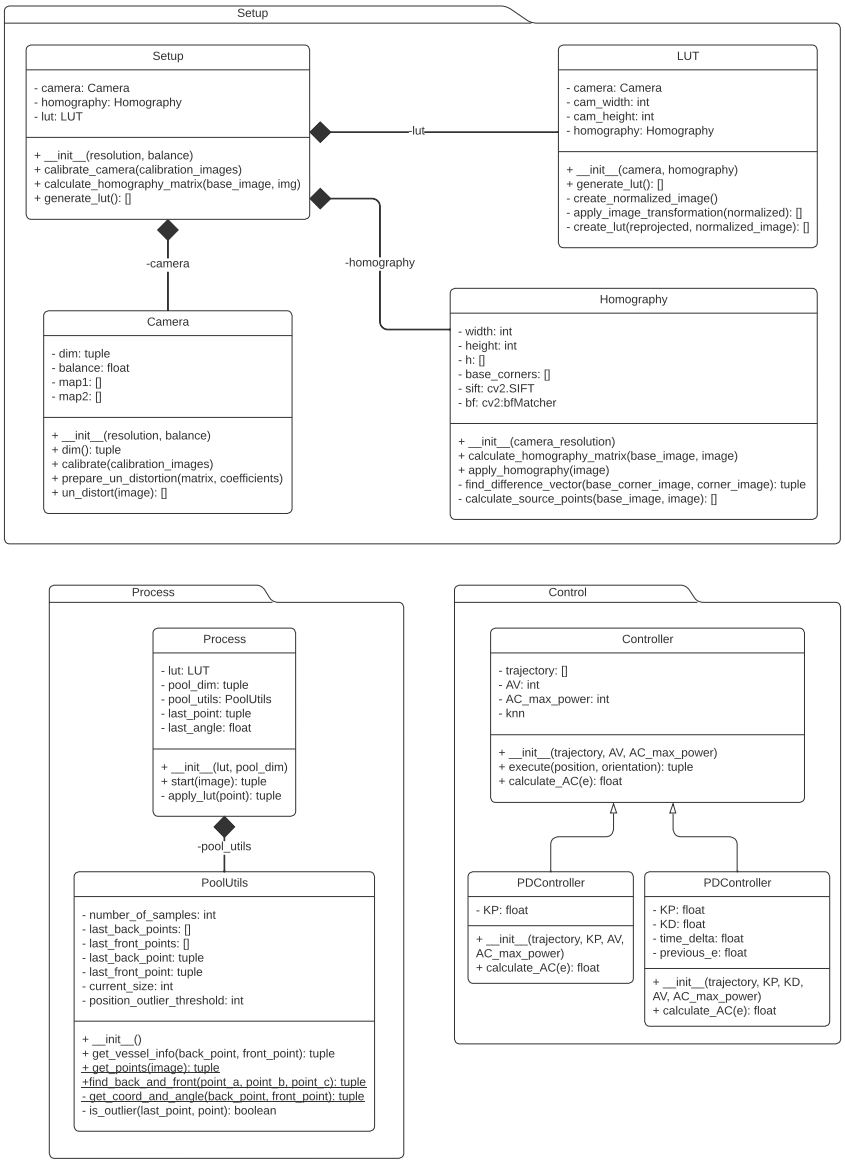


Figura A.1: UML