



ISEL – INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
ADEETC – ÁREA DEPARTAMENTAL DE ENGENHARIA DE
ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES

LEIM

LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA
UNIDADE CURRICULAR DE PROJETO

Algoritmo para avaliação da prestação de atletas de patinagem artística

André Ferreira nº45124
David Valente nº45129

Orientador(es)

Engenheiro André Lourenço

Engenheiro Tiago Dias

Setembro, 2021

Agradecimentos

Queremos agradecer aos nossos orientadores por nos acompanharem no desenvolvimento do projeto como também, Bárbara Pires e Edgar Santinhos por terem desenvolvido as ferramentas das quais estamos a utilizar para continuar este projeto.

Resumo

Este projeto consiste no desenvolvimento de um algoritmo que seja capaz de avaliar a prestação de atletas de patinagem artística de forma automática recorrendo a imagens recolhidas nos patins. Como tal, várias imagens irão ser captadas, e, usando os conhecimentos adquiridos em diversas cadeiras do curso (como, por exemplo, Processamento de Imagem e Visão), serão posteriormente tratadas e analisadas.

A realização de figuras obrigatórias pelo patinador pressupõe o seguimento de linhas com os patins, sendo que as imagens adquiridas pelos patins instrumentados são a base deste projeto. Esta tarefa de seguimento e linha é estudada de forma sistemática usando a cadeia de processamento típica de análise automática de imagem:

1. Pré-Processamento
2. Binarização
3. *Thresholding*
4. Representação da região desejada

O algoritmo desenvolvido pressupõe uma adaptação às características onde o patinador vai realizar as figuras obrigatórias incluindo uma fase de calibração onde o sistema terá que se adaptar ás características físicas do espaço (como a cor do chão e da linha). O critério usado para determinar a performance do patinador é baseado na centralidade da linha na imagem.

No fim, foram realizados testes usando várias sequências de imagens recolhidas por patinadores que permitiram validar o algoritmo desenvolvido.

Abstract

This project consists in the development of an algorithm that can evaluate the performance of Artistic Roller Skating athletes, in an automatic way. It resorts in a collection of previously obtained images via a sensor attached to the roller skates.

Using all the knowledge obtained in diverse course units (such as *Processamento de Imagem e Visão*), this images will be treated and processed on future blocks.

The execution of the so called figures by the athletes states that the athlete must follow the lines with the roller skates. In this way, we can obtain the collection of images that are going to be the foundation of this project.

This line following task is studied in a sistematic way using the processing chain tipically used in the automatic image analysis:

1. Pre-Processing
2. Binarization
3. Thresholding
4. Representation of the desired region

In a calibration phase, the developed algorithm will be adapted with the physical characteristics of the environment the athlete is in (such as the colors of the floor and the line).

The criteria used to calculate the performance of the athletes is based in the centralization of the line in the image.

Latter on, some tests using different images collected by the athletes were realized so that it was possible to validate the developed algorithm.

Índice

| | |
|--|------------|
| Agradecimentos | i |
| Resumo | iii |
| Abstract | v |
| Índice | vii |
| Lista de Tabelas | ix |
| Lista de Figuras | xi |
| 1 Introdução | 1 |
| 1.1 Análise do Problema | 5 |
| 2 Trabalho Relacionado | 7 |
| 2.1 "VisionRace", por Club de Robótica-Mecatrónica de la UAM . | 7 |
| 2.2 "Sensorização de patins para monitorização de movimentos", pela aluna Bárbara Pires, 2019 | 11 |
| 2.3 Deteção de desvios em figuras obrigatórias na Patinagem Artística por Edgar Santinhos, 2020 | 15 |
| 2.4 Automatic contrast and brightness adjustment of a color photo of a sheet of paper with OpenCV | 17 |
| 2.5 Automatic Thresholding of Gray-Level Pictures Using Two- Dimensional Otsu Method por Liu Jianzhuang. Li Wenqing. e Tian Yupeng, 1991 | 17 |
| 3 Modelo Proposto | 19 |
| 3.1 Requisitos | 19 |

| | | |
|----------|--|-----------|
| 3.2 | Fundamentos | 20 |
| 3.2.1 | Espaços de cor | 20 |
| 3.2.2 | Métodos de binarização | 25 |
| 3.2.3 | Pré-Processamento | 25 |
| 3.2.4 | Canny Edge Detector | 26 |
| 3.3 | Abordagem | 28 |
| 4 | Implementação do Modelo | 31 |
| 4.1 | Captação de imagem | 32 |
| 4.2 | Testes Iniciais - Espaço de Cores | 32 |
| 4.3 | Calibração e Validação da mesma | 35 |
| 4.4 | Pré-Processamento | 36 |
| 4.4.1 | Blur | 36 |
| 4.4.2 | Ajuste automático de contraste | 37 |
| 4.5 | Classificador para identificar a região desejada | 39 |
| 4.5.1 | Binarização | 39 |
| 4.5.2 | Deteção de linha | 40 |
| 4.6 | Avaliador com base na distância da linha ao centro | 41 |
| 5 | Validação e Testes | 43 |
| 5.1 | Validação Calibração | 43 |
| 5.2 | Validação avaliação | 44 |
| 6 | Conclusões e Trabalho Futuro | 47 |
| 7 | Referências | 49 |
| | Bibliografia | 51 |

Lista de Tabelas

Listas de Figuras

| | | |
|------|---|----|
| 1.1 | Uma patinadora a executar uma figura desta modalidade. | 1 |
| 1.2 | Parágrafo do Laço (Boucle) | 2 |
| 1.3 | Exemplo das regras das dimensões do traçar das linhas | 2 |
| 1.4 | Exemplo de uma das imagens obtidas pelo protótipo. | 6 |
| 2.1 | Exemplo do resultado obtido do projeto ”VisionRace” | 8 |
| 2.2 | Troço de código realizado pela equipa do ”VisionRace”, reali- zando um <i>threshold</i> adaptativo. | 9 |
| 2.3 | Troço de código realizado pela equipa do ”VisionRace”, para determinar a altura inicial para realizar a média. | 9 |
| 2.4 | Troço de código realizado pela equipa do ”VisionRace”, a cal- cular o ponto médio da linha. | 10 |
| 2.5 | Apresentação do fluxo de processamento realizado pela aluna Bárbara Pires. | 11 |
| 2.6 | Troço de código realizado pela aluna para atribuir algum pro- cessamento ás imagens obtidas. | 12 |
| 2.7 | Troço de código realizado pela aluna para executar a bina- rização das imagens obtidas e processadas. | 12 |
| 2.8 | Troço de código realizado pela aluna para calcular o centroide da imagem. | 13 |
| 2.9 | Troço de código realizado pela aluna para calcular a região de interesse. | 13 |
| 2.10 | Troço de código realizado pela aluna para calibrar o valor de <i>threshold</i> | 14 |
| 2.11 | Ciclo de desenvolvimento da framework ESP-32 | 15 |
| 2.12 | Printscreen do funcionamento da pagina disponibilizada | 16 |
| 2.13 | Prótotipo fixado no patim | 16 |

| | | |
|------|--|----|
| 3.1 | Exemplo ilustrativo da mistura de cores da componente RGB. | 21 |
| 3.2 | Exemplo ilustrativo da mistura do funcionamento do espaço HSV. | 22 |
| 3.3 | Ferramenta online onde podemos escolher uma cor, retornando valores nos diferentes espaços de cor. | 23 |
| 3.4 | Exemplo ilustrativo da transformação do espaço RBG em YCbCr. | 24 |
| 3.5 | Exemplo da aplicação do algoritmo <i>Canny Edge</i> , depois e antes. | 26 |
| 3.6 | Comparação entre um <i>Thresholding</i> normal, e um <i>Thresholding</i> de histerese. | 27 |
| 3.7 | Diagrama de blocos referente ao nosso sistema | 28 |
| 4.1 | Diagrama de blocos referente ao nosso sistema | 31 |
| 4.2 | Histogramas das componentes RGB,HSV E YCBCR | 33 |
| 4.3 | Fração da imagem das quais foram feitos os histogramas de todos os espaços de cor. | 34 |
| 4.4 | Histogramas de cor para a fração da imagem. | 35 |
| 4.5 | Testes de blur | 36 |
| 4.6 | Snippet de código criado por Nathancy | 37 |
| 4.7 | Output gerado pelo código | 38 |
| 4.8 | Testes de ajuste de gamma | 38 |
| 4.9 | Teste do método otsu com base na variação interclasse | 39 |
| 4.10 | Snippet de código criado por Michael Burdinov | 40 |
| 4.11 | Teste detecção de linha | 40 |
| 5.1 | Output gerado no teste da calibração | 43 |
| 5.2 | Output gerado no teste da avaliação | 44 |
| 5.3 | Testes de imagem com condições mais difícieis | 45 |

Capítulo 1

Introdução

A patinagem artística é uma modalidade desportiva de competição que apresenta várias vertentes.



Figura 1.1: Uma patinadora a executar uma figura desta modalidade.

Na vertente Individual existe a especialidade de Figuras Obrigatórias onde é colocada à prova a perícia técnica do/a patinador/a que, para ser bem-sucedido/a, tem que realizar corretamente figuras obrigatórias, como por exemplo a figura Parágrafo do laço (Boucle) ilustrada na Figura 1.1.

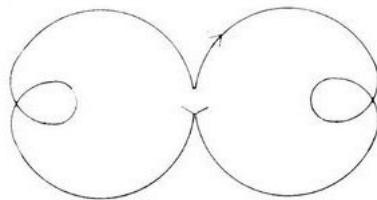


Figura 1.2: Parágrafo do Laço (Boucle)

Cada figura é delineada através de linhas pintadas no pavimento, sendo que a sua marcação obedece a regras específicas [1]. Existem características obrigatórias a seguir, como dimensões e raio das linhas, tal como se pode observar na figura 1.2 (presente na página seguinte), e outras deixadas em aberto, como a cor, que a única regra presente aqui, é haver contraste suficiente com o solo.

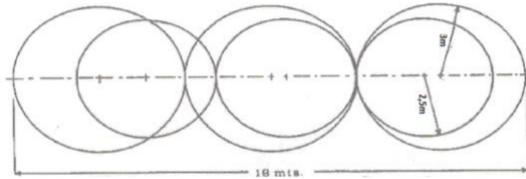


Figura 1.3: Exemplo das regras das dimensões do traçar das linhas

Para além da perícia técnica do/a patinador/a, também são avaliados outros critérios, como por exemplo:

- A postura dos atletas mantida durante a execução da figura;
- O tamanho e simetria apresentado nas curvas realizadas;
- A velocidade e o ritmo da execução;
- A precisão nas mudanças de rodado e prosseguimentos;
- O rodado e cumprimento do traçado;

Ainda, nesta modalidade a faixa etária também é um fator a ter em conta. A cada atleta, as figuras que terão que realizar serão sorteadas consoante o seu escalão, de uma lista pré-definida.

Como qualquer outra modalidade desportiva de alta competição, é requerido dos/das atletas treino intensivo, acompanhado pelos seus/suas treinadores/as, que os/as observam de forma incansável, de forma a avaliar a sua prestação, para que mais tarde seja possível debater sobre o seu desempenho.

Como é óbvio, o acompanhamento dos/das treinadores/as nem sempre é possível, quer seja por motivos de deslocação, ou por outros fatores externos, que possam eventualmente privar o/a atleta de ter o seu/sua treinador/a presente, o que dificulta significativamente o/a atleta ter noção do seu desempenho.

Mesmo assim, e dados os atuais avanços tecnológicos, o treino não acompanhado/supervisionado continua a ser uma opção, tendo como exemplo outros desportos como: no ténis [2], em que é colocado um sensor na extremidade do punho da raquete; no skate [3], o uso de sensores como acelerómetros nos ténis dos atletas e nas tábuas de skate; e em vários outros desportos.

Ora, com base nestes exemplos, a atleta Catarina Silva, também aluna do nosso instituto, apresentou na edição de 2018 do concurso Born from Knowledge Ideas, da Associação Nacional de Inovação, uma ideia para fazer uso da tecnologia com o propósito de fomentar e auxiliar a prática individual da vertente de Figuras Obrigatórias da Patinagem Artística, que venceu na categoria de Turismo e Indústrias Criativas [4].

Tendo essa ideia como base, foi iniciado um projeto IDICA [5] de um protótipo para a obtenção de imagens enquanto o/a atleta realiza as Figuras Obrigatórias, que será essencial ao desenvolvimento do nosso algoritmo.

Este projeto tem como principal objetivo o estudo e a implementação de um sistema de sensorização para patins. Este sistema deverá permitir adquirir e facultar, de forma automática, a informação de que o/a atleta necessita para compreender o seu grau de sucesso no cumprimento dos traçados das figuras.

O nosso trabalho consistirá, como o nome do projeto sugere, desenvolver um algoritmo que avalie a prestação dos/as atletas desta modalidade.

Como tal, o tema principal a estudar/abordar será o processamento automático de imagem, dado que o nosso projeto incidirá no tratamento de imagens obtidas do protótipo referido anteriormente.

Como é óbvio, nem todas as imagens obtidas serão iguais. As mesmas irão variar em cor, luz, posição da linha, etc. De forma a determinar com exatidão a posição da linha em relação ao centro da imagem, será necessário, em primeiro lugar, detetar onde está a mesma, e distingui-la do pavimento. Isto só é possível, computacionalmente, através do processamento de imagem, onde serão extraídas diversas características da imagem (como a cor de cada pixel) de forma a que o resultado final seja obtido como desejado.

1.1 Análise do Problema

Na modalidade de Figuras Obrigatórias da Patinagem Artística o/a patinador/a tem de seguir uma linha circular traçada no solo (como representado na Figura 1.2), descrevendo com o patim uma figura selecionada de uma lista regulamentada, tendo em conta determinadas regras [1].

Como é óbvio, as provas e treinos não são executados sempre em pavilhões iguais, com características como a cor do pavimento, e a cor da linha, semelhantes. As linhas, por vezes, encontram-se mais desgastadas. A iluminação do pavilhão também nem sempre é a mesma.

Tudo isto, são fatores físicos externos que o/a atleta não pode controlar. De forma a desenvolvermos um sistema que estude e analise a prestação dos/das patinadores/as de forma a não ser obrigatória a presença de um/a treinador/a, é necessário projetar um sistema de sensorização que meça a taxa de seguimento da linha e a posição dos patins. Também será necessário que esse mesmo sistema se adapte ao meio em que se encontra inserido, visto que nem sempre as condições físicas são as mesmas.

Este projeto irá adquirir informação de um sensor realizado em outros projetos desta unidade curricular, e irá tratar toda a informação necessária para que seja possível apresentar e interpretar o desempenho dos/as atletas ao terminarem a execução de cada Figura Obrigatória.

Cada imagem irá apresentar uma série de características que irão ser diferentes das restantes imagens. Características como a luminosidade do pavilhão, a cor do chão, a linha e a sua posição, irão variar de imagem para imagem, pelo que será necessário aplicar os filtros e processamento adequado a cada situação.



Figura 1.4: Exemplo de uma das imagens obtidas pelo protótipo.

Aqui, temos um exemplo de uma das imagens que irão ser tratadas pelo nosso algoritmo.

O nosso principal objetivo é distinguir a linha do pavimento, com sucesso, ou seja, que a cada imagem que seja apresentada, esta distinção seja bem feita. No entanto, existem diversos fatores que irão dificultar esta tarefa. Por exemplo, se olharmos para o lado esquerdo da imagem a cima, observamos que o mesmo é mais claro que o lado direito da mesma. Este é um fator que irá dificultar imenso a distinção.

Outro, como podemos observar, existem duas linhas sobrepostas, uma de cor amarela, e outra de cor preta. Neste caso, o nosso algoritmo terá que saber distinguir qual é a linha que pretende analisar, e os limites da mesma, tarefa que é dificultada pela existência de sobreposição de linhas.

Estes tipos de dificuldades só serão ultrapassados ao aplicar o processamento devido. Teremos que fazer um ajuste na luminosidade da imagem, uma calibração para a cor da linha e do pavimento (de modo a não haver dúvidas na distinção) e, quando estes passos já forem executados e a linha já seja distinguida como desejado, teremos que calcular o quão centrada a mesma se encontra na imagem, dados os seus limites.

Como referido antes, nem todos os pavilhões e linhas apresentarão estas mesmas características. A linha poderá não apresentar contraste suficiente para o pavimento, por exemplo, o que será também outro problema.

O nosso sistema terá que ser capaz de ultrapassar estas dificuldades de forma automática, o que irá requerer que o devido processamento seja aplicado, e de forma correta.

Capítulo 2

Trabalho Relacionado

De forma a realizar o nosso projeto de forma mais otimizada possível, foi necessário o estudo de outros projetos já realizados na área. Os exemplos apresentados a seguir serão os trabalhos em que o nosso projeto foi baseado.

2.1 ”VisionRace”, por Club de Robótica-Mecatrónica de la UAM

Este projeto foi a ”base” para podermos desenvolver o nosso.

Baseia-se num algoritmo onde um robot, que através da captação de imagem por uma câmara de vídeo, irá seguir uma linha captando vários ”frames” e, utilizando um algoritmo de análise de imagem, irá traçar o seu caminho a seguir, que é muito semelhante ao que queremos executar no nosso projeto.



Figura 2.1: Exemplo do resultado obtido do projeto ”VisionRace”

À semelhança do nosso projeto, este usa bibliotecas como o OpenCV e Numpy para executar funções de processamento de imagem (como por exemplo, o `threshold()` do OpenCV).

Dentro do repositório do GitHub referente a este projeto, podemos encontrar vários ficheiros Python, no entanto, o mais importante para o nosso desenvolvimento será o de nome `”line_follow.py”`.

Num processo inicial, este trabalho começa por adquirir várias imagens (”*frames*”, no caso) onde, pela reprodução do vídeo disponibilizado pelos autores, podemos observar que a linha já se encontra presente na câmara, e centrada. Após algum tratamento das imagens obtidas, os projetistas procedem então para algo que será crucial para o nosso projeto: um *”threshold”* adaptativo, que nos irá retornar a imagem binarizada, distinguindo o que é linha, e o que não é linha.

Como o nome indica, um *”thresholding”* é algo utilizado para binarização, onde resumidamente é tido em conta um valor (o chamado *”threshold”*). Tudo o que estiver a baixo desse valor é tomado como valor 0, enquanto o que estiver a cima é considerado 1, realizando assim uma binarização da imagem. No caso de cada imagem, como é sabido, cada pixel apresenta um determinado valor. É com esse mesmo valor que é comparado com o *”threshold”*.

Este é um passo importantíssimo para o nosso trabalho. No nosso caso, queremos também distinguir o que é linha e o que é chão, de forma a mais tarde, conseguirmos analisar a posição da mesma.

```
thresh = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY,11,2)
```

Figura 2.2: Troço de código realizado pela equipa do "VisionRace", realizando um *threshold* adaptativo.

Adicionalmente, ainda houveram alguns aspetos importantes deste projeto que podemos observar.

Os desenvolvedores do trabalho "VisionRace" também nos apresentam um troço de código que irá calcular o centro da linha, fazendo uma média entre o ponto mais à direita da linha no ecrã, e o ponto mais à esquerda da linha no ecrã.

Ora, entrando um pouco mais em detalhe, os projetistas começaram por definir a "altura" em que iam começar a calcular o ponto médio da linha, sendo este algo muito perto da linha mais a baixo no ecrã (para evitar eventuais erros, como por exemplo, a linha não aparecer imediatamente no ponto mais a baixo do ecrã, ou algum tipo de ruído).

```
(w, h) = (640,240) # Resolution
bytesPerFrame = w * h
fps = 40 # setting to 250 will request the maximum framerate possible

lateral_search = 20 # number of pixels to search the line border
start_height = h - 5 # Scan index row 235
```

Figura 2.3: Troço de código realizado pela equipa do "VisionRace", para determinar a altura inicial para realizar a média.

Aqui, podemos observar que a equipa começa por definir uma largura e altura para a imagem, e define a altura em que vão começar a calcular a média.

Depois de algum tratamento de imagem, e de já terem a imagem binarizada (através do *threshold* mencionado anteriormente), a equipa progrediu então para calcular o ponto médio da linha, através dos seguintes cálculos:

```

signed_thresh = thresh[start_height].astype(np.int16) # select only one row
diff = np.diff(signed_thresh) #The derivative of the start_height line

points = np.where(np.logical_or(diff > 200, diff < -200)) #maximums and minimums of derivative

cv2.line(frame_rgb,(0,start_height),(640,start_height),(0,255,0),1) # draw horizontal line where scanning

if len(points) > 0 and len(points[0]) > 1: # if finds something like a black line
    if GetSpeed() == 0: # if is stopped but finds a line
        BaseSpeed(Speed)

    middle = (points[0][0] + points[0][1]) / 2

    cv2.circle(frame_rgb, (points[0][0], start_height), 2, (255,0,0), -1)
    cv2.circle(frame_rgb, (points[0][1], start_height), 2, (255,0,0), -1)
    cv2.circle(frame_rgb, (middle, start_height), 2, (0,0,255), -1)

```

Figura 2.4: Troço de código realizado pela equipa do ”VisionRace”, a calcular o ponto médio da linha.

Como podemos ver na Figura 2.4, após calcular os extremos da linha, é calculado o ponto médio da mesma.

É ainda importante ter em conta que a imagem é dividida por linhas, e processada a cada uma dessas linhas horizontais, de cima para baixo.

Tendo então a linha bem detetada, bem como o ponto médio calculado, é possível afirmar que este projeto nos será imensamente útil.

2.2 ”Sensorização de patins para monitorização de movimentos”, pela aluna Bárbara Pires, 2019

À semelhança do projeto apresentado anteriormente, este também nos ajudou bastante no que toca ao estudo da teoria necessária para o desenvolvimento do nosso trabalho.

Como projeto final de curso da Licenciatura em Engenharia Informática e de Computadores, a aluna Bárbara Pires desenvolveu um trabalho muito semelhante ao nosso, com algumas diferenças.

Num processo inicial, a aluna começa por nos apresentar a peça de hardware utilizada neste projeto, que, essencialmente, fará a captação de imagem do patim.

No entanto, o nosso foco foi mais na parte de software.

Nesta componente, a aluna começa por nos apresentar o seguinte fluxo de processamento:



Figura 2.5: Apresentação do fluxo de processamento realizado pela aluna Bárbara Pires.

Podemos concluir que este projeto começa pela aquisição de imagens, seguida pelo processamento, depois a análise, e numa fase final, a apresentação dos resultados obtidos. O fluxo do nosso trabalho será algo muito semelhante ao desta aluna, pelo que é bastante interessante observar e analisar este trabalho em detalhe.

A primeira fase, o processo de obtenção das imagens, é muito simples, no entanto, e visto que iremos utilizar uma linguagem diferente da que a aluna usou, o nosso será bastante diferente, pelo que este ainda não será o nosso foco principal.

A segunda fase, o processamento, já nos é um pouco mais interessante. Aqui, a aluna recorre à biblioteca *open-source* "OpenCV", que disponibiliza algumas ferramentas essenciais ao processamento de imagem.

```
void grayscaleImage(Mat* src) {
    Mat image = *src;
    cvtColor(image, image, COLOR_BGR2GRAY);
    *src = image;
}
```

Figura 2.6: Troço de código realizado pela aluna para atribuir algum processamento ás imagens obtidas.

A aluna recorre a uma transformação da imagem original em *BGR* para *Grayscale*, transformando assim uma imagem a cores, numa imagem em tons de cinzento.

Isto deve-se ao facto de existir um elevado contraste entre a linha e o pavimento. Como é suposto, e estabelecido pelas normas da Patinagem Artística, a linha tem que apresentar um contraste elevado em relação ao pavimento em que se encontra. Como tal, se esse contraste for assim tão elevado como necessário, se convertermos a imagem para tons de cinzento, apesar de não termos toda a informação da cor, temos a informação necessária para distinguir a linha e o pavimento, o que tornará a binarização da imagem muito mais fácil.

De seguida, e como esperado, a aluna efetua a tão mencionada binarização, como podemos observar no troço de código a seguir:

```
void binarizeImage(Mat* src, int value) {
    Mat image = *src;
    threshold(image, image, value, 255, THRESH_BINARY);
    image = ~image;
    *src = image;
}
```

Figura 2.7: Troço de código realizado pela aluna para executar a binarização das imagens obtidas e processadas.

Como falámos no projeto anterior, a aluna recorreu também a um *threshold* para binarizar a imagem, como era esperado.

Encerrando então a fase de processamento das imagens, a aluna avançou para a próxima fase: a análise, que também é bastante interessante para o nosso trabalho.

A aluna começa por encontrar as coordenadas do centroide da imagem, que deverão corresponder ao centro geométrico da linha. Esta informação é relevante pois através disto, é possível determinar a região em que é suposto a linha se encontrar, caso o desvio do patim fosse nulo (situação em que a performance do/da atleta equivale a 100%).

Nos troços de código a seguir apresentados, a aluna calcula as coordenadas do centroide, e da região de interesse.

```
Moments m = moments(thresh, true);
int centroid_x = m.m10 / m.m00;
int centroid_y = m.m01 / m.m00;
```

Figura 2.8: Troço de código realizado pela aluna para calcular o centroide da imagem.

```
int range_left = centroid_x - lineWidth / 2;
if (range_left < 0) range_left = 0;

int range_right = centroid_x + lineWidth / 2;
if (range_right > thresh.cols) range_right = thresh.cols;

Mat imageROI = thresh.colRange(range_left, range_right);
```

Figura 2.9: Troço de código realizado pela aluna para calcular a região de interesse.

Esta fase é crucial para o nosso trabalho pois à semelhança do que a colega fez, também teremos que saber onde a linha se encontra em relação ao centro da imagem.

A fase seguinte a aluna não mencionou no fluxo de processamento, no entanto, também nos é interessante pois será focada em calibrar o sistema com as condições físicas do meio em que o patim se encontra inserido.

Aqui, a chamada ”Fase de Calibração do Sistema”, a aluna começa por encontrar uma imagem de calibração, onde a linha se encontra no meio da imagem, e que equivale a um desvio quase nulo do patim em relação à mesma. Depois de obtida esta imagem, é necessário encontrar um valor de ”*threshold*” adequado, para que a binarização seja feita com êxito.

Desta forma, a aluna desenvolveu o seguinte troço de código:

```

int calibrateThresholdValue(Mat* src) {
    namedWindow("Original image", WINDOW_KEEP_RATIO);
    resizeWindow("Original image", 400, 300);
    moveWindow("Original image", 10, 10);
    imshow("Original image", *src);

    namedWindow("Processed image", WINDOW_KEEP_RATIO);
    resizeWindow("Processed image", 400, 300);
    moveWindow("Processed image", 450, 10);

    int value = 120;
    int option;
    Mat thresh;

    do {
        thresh = *src;
        processImage(&thresh, value);
        imshow("Processed image", thresh);
        waitKey(1);
        cout << "Is the line well defined in the processed image?" << endl;
        cout << "0 - Yes, it's similar to the original" << endl;
        cout << "1 - No, it's larger than the original" << endl;
        cout << "2 - No, it's smaller than the original" << endl;
        cin >> option;
        switch (option) {
            case 1:
                value -= 10;
                if (value < 0) value = 0;
                break;
            case 2:
                value += 10;
                if (value > 255) value = 255;
                break;
        }
    } while (option != 0);

    destroyWindow("Original image");
    destroyWindow("Processed image");

    return value;
}

```

Figura 2.10: Troço de código realizado pela aluna para calibrar o valor de *threshold*.

Como podemos observar, através da interação com o utilizador, é possível obter um valor de ”*threshold*” ótimo para o projeto.

Estas são as fases que serão essenciais ao desenvolvimento do nosso trabalho, referentes ao trabalho da nossa colega. Foi com base nelas que foi possível a realização de uma grande parte do nosso projeto.

2.3 Deteção de desvios em figuras obrigatórias na Patinagem Artística por Edgar Santinhos,2020

Este projeto foca-se maioritariamente no desenvolvimento do *hardware* do equipamento utilizado na captação de imagens como adicionalmente o desenvolvimento do *software* para a transmissão das imagens via *wi-fi*. O aluno deu uso à *framework* ESP-32 em que possui o seguinte ciclo de desenvolvimento.

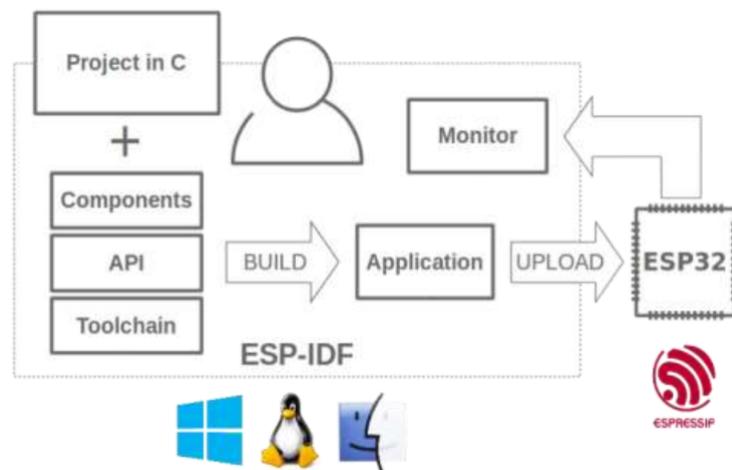


Figura 2.11: Ciclo de desenvolvimento da framework ESP-32

Com o uso adicional do ambiente Visual Studio Code que permite o uso de diversas bibliotecas foi desenvolvido a aplicação que irá devolver ao utilizador as imagens captadas pelo sensor.

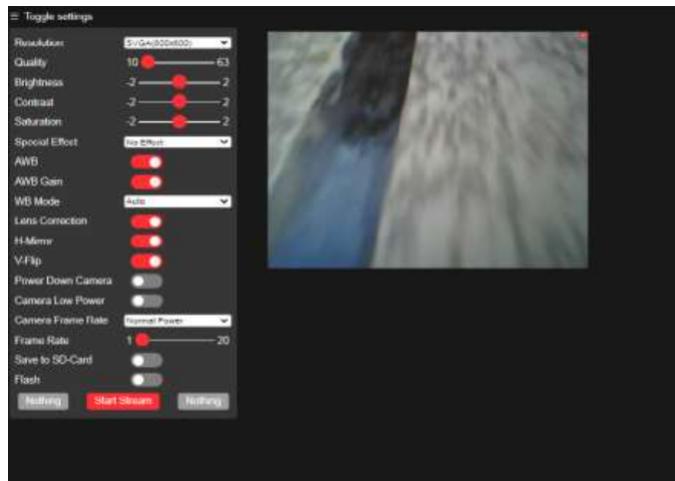


Figura 2.12: Printscreen do funcionamento da pagina disponibilizada

Como é possível observar pelo *print* a aplicação irá devolver a imagem em conjunto com varias informações do sensor pelo que na nossa aplicação iremos cortar a imagem de modo só ficar o frame captado pelo sensor.

Por fim foi desenvolvido o protótipo do sensor que irá ser colocado na frente do patim de uma maneira conveniente.

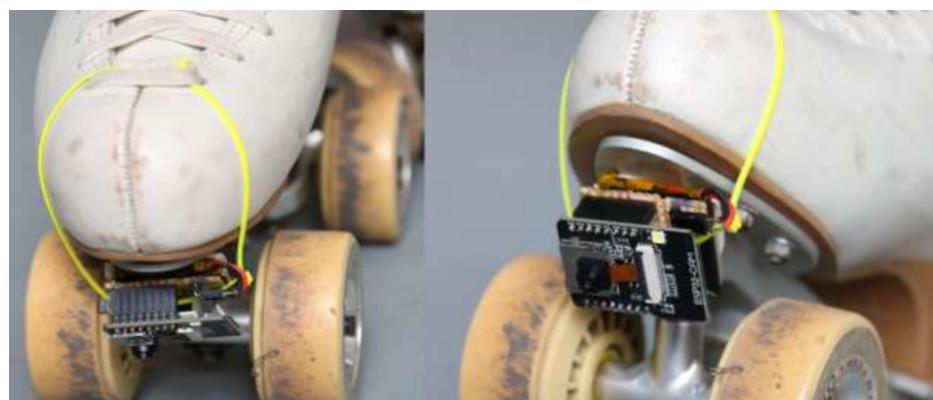


Figura 2.13: Prótotipo fixado no patim

2.4 Automatic contrast and brightness adjustment of a color photo of a sheet of paper with OpenCV

Este trabalho é uma *thread* no *stackOverflow* em que consiste na deteção de texto que esteja marcado com marcadores tipicamente utilizados no dia a dia, como o objetivo deste trabalho consistia também no processamento de uma dada imagem e a subsequente classificação dos componentes da mesma testamos algumas das ideias que os utilizadores do site responderam no thread.

2.5 Automatic Thresholding of Gray-Level Pictures Using Two-Dimensional Otsu Method por Liu Jianzhuang. Li Wenqing. e Tian Yupeng, 1991

Este projeto concentra-se maioritariamente nos benefícios que a aplicação do método otsu 2D possui ao invés do método otsu unidimensional, para o nosso trabalho no entanto o uso para método otsu unidimensional prova ter sucessos na binarização automática de imagens, como tal usamos este artigo para estudar o uso do método para aplicar no nosso projeto.

Capítulo 3

Modelo Proposto

Como visto anteriormente, o nosso sistema irá captar uma sequência de imagens, e irá determinar o quanto centrada essa linha se encontra com o patim. Desta forma, existe a necessidade de definir um modelo que analise os dados obtidos de forma concisa e eficaz. Para tal, é necessário estabelecer os requisitos que o nosso sistema terá que ter, os nossos fundamentos teóricos para tal efeito, e ainda a abordagem que realizámos, que serão apresentados a seguir.

3.1 Requisitos

O nosso sistema terá que ser analisar com o maior detalhe possível, a posição da linha no ecrã, para, futuramente, avaliar a prestação do/da atleta que executa a Figura Obrigatória.

Assim sendo, o nosso sistema necessitará de cumprir os seguintes requisitos:

- Calibrar o sistema com as características físicas do pavimento e da linha;
- Processar cada imagem de forma a distinguir a linha;
- Retornar a performance do/da atleta em cada instante;

Desta forma, todos estes requisitos são essenciais ao desenvolvimento do nosso projeto. Se um falhar, então os seguintes não serão executados com tanto sucesso (ou em alguns casos, não serão executados de todo).

3.2 Fundamentos

Após observarmos e termos cientes todos os requisitos necessários para o nosso projeto, ainda era preciso realizar algum estudo teórico sobre alguns temas fundamentais para o desenvolvimento deste trabalho. Dito isto, explorámos alguns temas na área de processamento de imagem, que serão mencionados a seguir.

3.2.1 Espaços de cor

O primeiro tema a abordar, serão os espaços de cor. Como é óbvio, qualquer processamento de imagem está associado com a cor, e como a mesma se representa no mundo da tecnologia.

RGB

O primeiro espaço testado foi o RGB, em que cada letra corresponde a uma cor. A cor Vermelho (*Red*), verde (*Green*) e azul (*Blue*).

Para cada cor estão reservados 8 bits, ou seja, o valor máximo que cada componente poderá possuir será 255 ($2^8 - 1$).

Este espaço foi escolhido devido a ser o espaço mais comum. É conveniente referir também que o `OpenCV` utiliza o espaço BGR em que o componente *Red* e a componente *Blue* são trocados.

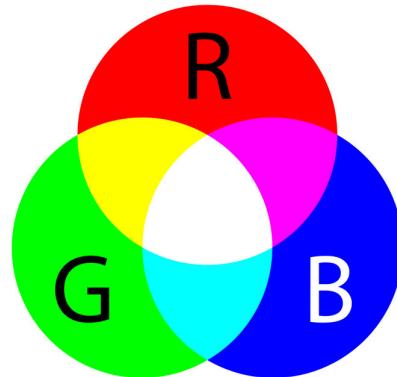


Figura 3.1: Exemplo ilustrativo da mistura de cores da componente RGB.

Num ecrã, por exemplo, cada pixel na tela pode ser representado com valores para vermelho, verde e azul.

Cada cor será uma combinação de diferentes valores de Red, Green e Blue. Com este sistema, e dadas as gamas referidas anteriormente, é possível representar uma vasta quantidade de cores diferentes.

HSV

O segundo espaço escolhido foi o espaço HSV, que, como no espaço RGB, o nome é uma sigla em que cada letra corresponde a uma componente.

Hue ou Tonalidade, em que a escala é de 0 a 360, *Saturation* ou saturação com escala de 0 a 100 %, *Value* ou Brilho também com a escala de 0 a 100 %, no entanto o OpenCV utiliza a escala de [0,179],[0,255],[0,255]. Este espaço foi escolhido devido ao facto de ser uma transformação não linear do espaço RGB, em que teoricamente será possível obter valores de cor mais distintos.

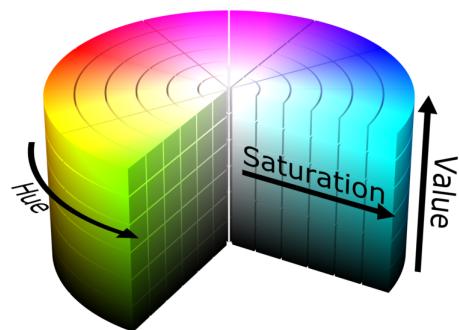


Figura 3.2: Exemplo ilustrativo da mistura do funcionamento do espaço HSV.

A imagem apresentada a baixo é um programa online onde podemos escolher uma cor, deslizando o cursor na vertical e na horizontal. Se o deslizarmos na horizontal, iremos variar os valores da Saturação. Se imaginarmos apenas um segmento horizontal dessa área, podemos observar que o que varia na cor, é, de facto, a saturação, o que se traduz numa cor mais acinzentada para valores de saturação mais baixos, e uma cor mais viva, para valores de saturação altos.

Se deslizarmos o cursor apenas na vertical, podemos observar uma variação nos valores do Value, o que se traduz numa cor mais escura, quando mais para baixo deslizarmos, e numa cor mais clara, quanto mais para cima o utilizador deslizar.

Para o valor Hue, temos a barra mais à direita. Ao fazermos deslizar essa barra para cima e para baixo, podemos escolher então a tonalidade que queremos, que será influenciada pelos valores de Saturation e Value que escolhemos no quadrado mais à direita.



Figura 3.3: Ferramenta online onde podemos escolher uma cor, retornando valores nos diferentes espaços de cor.

YCbCr

O último espaço escolhido é o YCbCr.

YCbCr, ou Y Pb/Cb Pr/Cr, escolhido devido ao facto de ser usado tipicamente em *pipelines* fotográficos ou de vídeo, em que o Y' representa a *luma* que é uma codificação não linear da luminosidade (representada pela letra Y), C_B e C_R é a diferença da chroma azul e vermelho respetivamente.

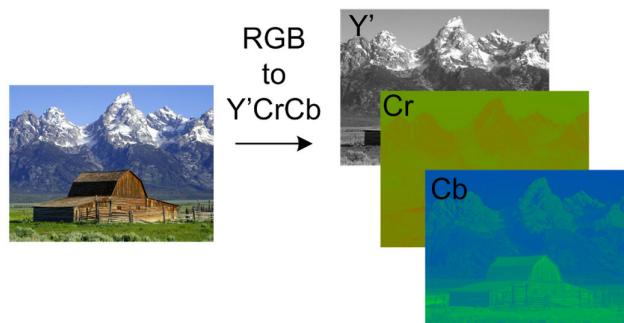


Figura 3.4: Exemplo ilustrativo da transformação do espaço RGB em YCbCr.

Como sabemos, num ecrã, as cores são representadas por sinais de tensão vermelhos, verdes e azuis, como vimos anteriormente para o espaço RGB, no entanto, este espaço não é eficiente no que toca a armazenamento e transmissão, uma vez que tem muita redundância.

Este espaço é mais conveniente que RGB na medida em que pode ser usado para separar um sinal de *luma* que pode ser armazenado com alta resolução ou transmitido em alta largura de banda, e dois componentes de crominância (CB e CR) que podem ser reduzidos de outra forma tratados separadamente para melhorar a eficiência do sistema.

Desta forma, este espaço de cores atribui mais largura de banda à primeira componente, visto que os humanos são mais sensíveis à informação em preto e branco, e preserva um pouco mais nas outras duas componentes.

3.2.2 Métodos de binarização

De forma a conseguirmos separar, com eficiência, a linha do pavimento, é necessário recorrer à binarização.

Binarização é a conversão de uma data imagem, num dado espaço de cores, para apenas tons de preto e branco, ou seja, de valores 1 e 0. Neste processo, é definido um determinado valor, ao que se dá o nome *threshold*, e é feita uma decisão através desse valor.

No caso das imagens, a cada pixel, é feita a comparação do valor atribuído a esse pixel, com o valor de decisão. Caso este esteja a baixo do *threshold*, esse é tomado a 0, e, caso esteja a cima, tomado a 1.

Como é óbvio, a binarização não é algo que seja feito sempre da mesma forma. Existem vários métodos para a mesma, que serão explorados mais à frente no projeto.

3.2.3 Pré-Processamento

Qualquer imagem que seja obtida através de qualquer tipo de lente, possui um determinado nível de ruído, o que dificultará processos como a binarização.

Este passo é utilizado de forma a que seja possível preparar, ao máximo, uma determinada imagem para que a mesma possa ser analisada. Por exemplo, se quisermos que algo seja mais saliente numa imagem, como, por exemplo, uma linha, podemos proceder a diferentes métodos diferentes como a adição de ruído, ou *blur*, uma dilatação, uma erosão, etc, dependendo sempre do resultado que pretendemos obter.

Uma erosão, por exemplo, é utilizada quando queremos subtrair alguma informação que consideremos "a mais" numa determinada imagem. Considerando que estes métodos se baseiam não apenas no pixel presente, mas nos seus vizinhos também, ou seja, se tivermos um pavimento cinzento, e no meio do mesmo, um pixel que apresente uma cor um pouco mais avermelhada, dado que os seus vizinhos apresentam a cor cinzenta, ao realizar uma erosão esse pixel irá ficar mais semelhante aos seus vizinhos.

Métodos como uma dilatação, uma abertura, um fecho, etc, são semelhantes à erosão, onde também é tido em conta o valor dos pixéis vizinhos, no en-

tanto, a decisão tomada é diferente, consoante o método a executar.

3.2.4 Canny Edge Detector

Este foi um dos métodos de binarização que estudámos inicialmente, dado que vai bastante ao encontro com aquilo que pretendíamos para a linha. O *Canny Edge Detector*, desenvolvido em 1986 por John F. Canny, é um detector de "edges", ou bordas, que através de um algoritmo "*multi-stage*"(constituído por várias fases), extrairá informação sobre a estrutura de vários objetos que se encontram numa dada imagem, ou seja, resumidamente, o *Canny Edge* retornará uma imagem em tons de preto com as bordas dos objetos detetados a branco.

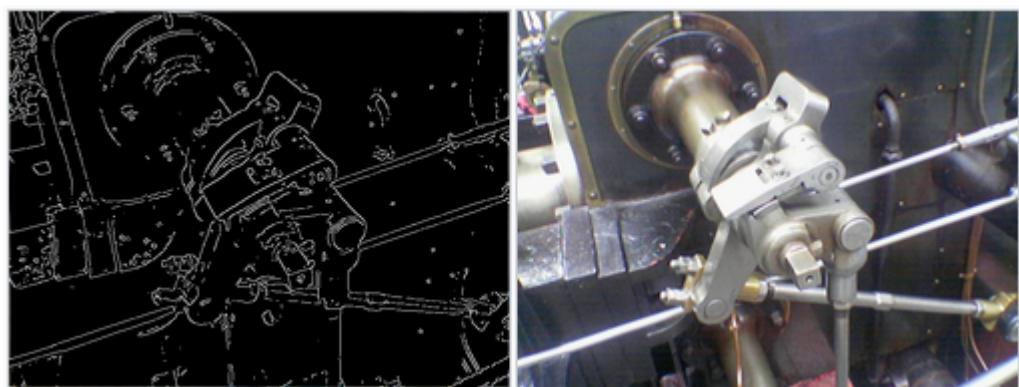


Figura 3.5: Exemplo da aplicação do algoritmo *Canny Edge*, depois e antes.

Este algoritmo divide-se em 4 fases principais:

1. Redução de ruído;
2. Detetar a intensidade do gradiente da imagem;
3. Remover pixeis que não pertençam ao gradiente;
4. *Thresholding* da histerese da imagem;

Estes são os quatro passos essenciais à execução deste algoritmo.

No primeiro, é necessário recorrer a um filtro Gaussiano de 5x5, de forma

a remover uma parte do ruído que possa eventualmente estar presente na imagem.

No segundo, como é sabido, um gradiente é uma alteração progressiva nos tons de uma cor, numa imagem. Dado isto, é necessário achar o mesmo, bem como a sua intensidade e direção.

No terceiro passo, logo após encontrar o gradiente, é necessário remover quaisquer pixéis que estejam incorretamente fora desse gradiente. Por exemplo, dado um gradiente de um azul escuro para um azul claro, caso, no meio, exista um pixel que apresente uma cor vermelha, dado que os seus vizinhos são de cor azul, à partida esse pixel é considerado como ruído.

Por fim, é necessário proceder a uma remoção da histerese da imagem. A histerese é uma espécie de inércia. Um *Thresholding* de histerese é quando algumas áreas abaixo do valor de *Threshold* são consideradas a cima do mesmo, caso estas estejam conectadas a outras áreas a cima desse mesmo valor. Em baixo, podemos observar a comparação entre um *Thresholding* normal, e um *Thresholding* de histerese:

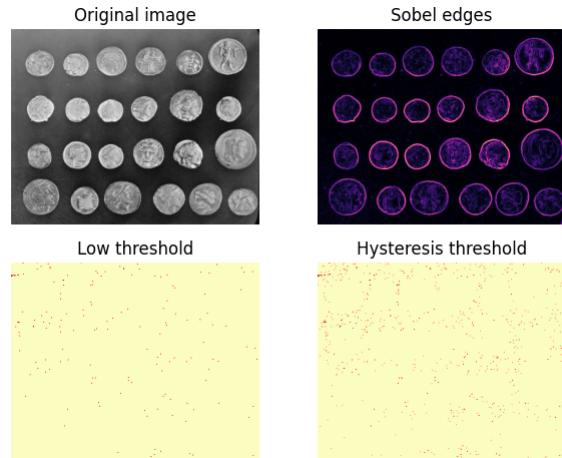


Figura 3.6: Comparaçāo entre um *Thresholding* normal, e um *Thresholding* de histerese.

Dado que o *Canny Edge* é um elemento bastante interessante para o nosso trabalho, visto que, à partida, podemos detetar bastante bem os limites da minha com este algoritmo, o grupo considerou o estudo do mesmo bastante relevante.

3.3 Abordagem

Inicialmente decomposemos o trabalho em componentes que poderão ser desenvolvidos em paralelo ou sequencialmente.

Para o desenvolvimento do projeto foi criado o seguinte diagrama de blocos:

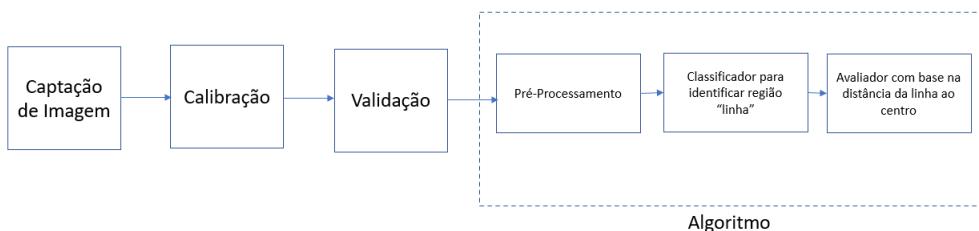


Figura 3.7: Diagrama de blocos referente ao nosso sistema

Inicialmente procede-se o modulo de captação de imagem que irá obter o vídeo seguido de uma validação em que permitirá ao utilizador centrar o patim na configuração inicial e adicionalmente permitirá ao código extrair características da imagem.

Após calibrado o patim existe um componente de validação, em que é verificado se o patim está centrado e reduzir erros humanos na calibração do patim.

Segue-se então para o algoritmo em que está dividido em três componentes, Pré-processamento que realiza qualquer operação necessária para melhor a qualidade do *frame* antes de iniciar o processamento do mesmo, um classificador que irá binarizar a imagem e obter a região da linha e por fim o avaliador com base no *output* gerado pelo classificador irá avaliar a prestação do patinador.

O cálculo da avaliação será com base no centro da linha e no ponto em que a linha se encontra mais perto do patim (na ultima linha do *frame*), deste modo é possível obter uma avaliação precisa mesmo que o patinador esteja a realizar uma curva.

Após definidos os blocos foram definidos um elevado nº de testes para obter quais os melhores parâmetros a serem utilizados em cada componente que serão realizados antes de qualquer avanço em cada componente.

Capítulo 4

Implementação do Modelo

Como visto no capítulo anterior, o primeiro passo para a execução deste projeto foi projetar como o desenvolvimento do mesmo se iria fazer, dado que foi desenvolvido o devido diagrama de blocos.

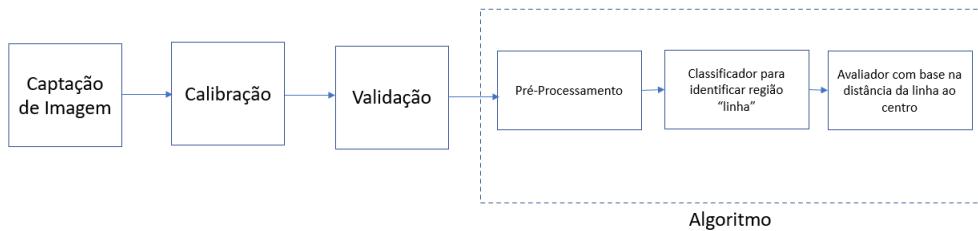


Figura 4.1: Diagrama de blocos referente ao nosso sistema

Com este realizado, é então possível começar a desenvolver o algoritmo de forma sistemática e metódica.

Cada um dos blocos será visto em detalhe nas próximas secções.

É ainda relevante salientar que o desenvolvimento do projeto não seguiu diretamente o diagrama de blocos, pelo que preferimos desenvolver primeiro certos blocos que outros.

4.1 Captação de imagem

Foi fornecido pelos nossos orientadores um vídeo de exemplo, obtido através do protótipo desenvolvido pelos nossos colegas, mencionado anteriormente, em que inicialmente foi armazenado dentro de um *array*, onde posteriormente foram obtidos alguns *frames* que iremos denominar de *Corner Cases*.

Corner Cases são casos específicos que apresentam-se segundo certas condições. Estes frames serão analisados para o desenvolvimento de cada modulo.

4.2 Testes Iniciais - Espaço de Cores

Foi observado em cada *Corner Case* o histograma de cores de cada componente de cada espaço.

O objetivo será o de analisar componentes cujo *spread* de valores sejam mais distintos dessa maneira quando se proceder para o modulo de binarização, o algoritmo ter mais facilidade a binarizar o frame.

De seguida, foram analisados os histogramas de três espaços de cores.

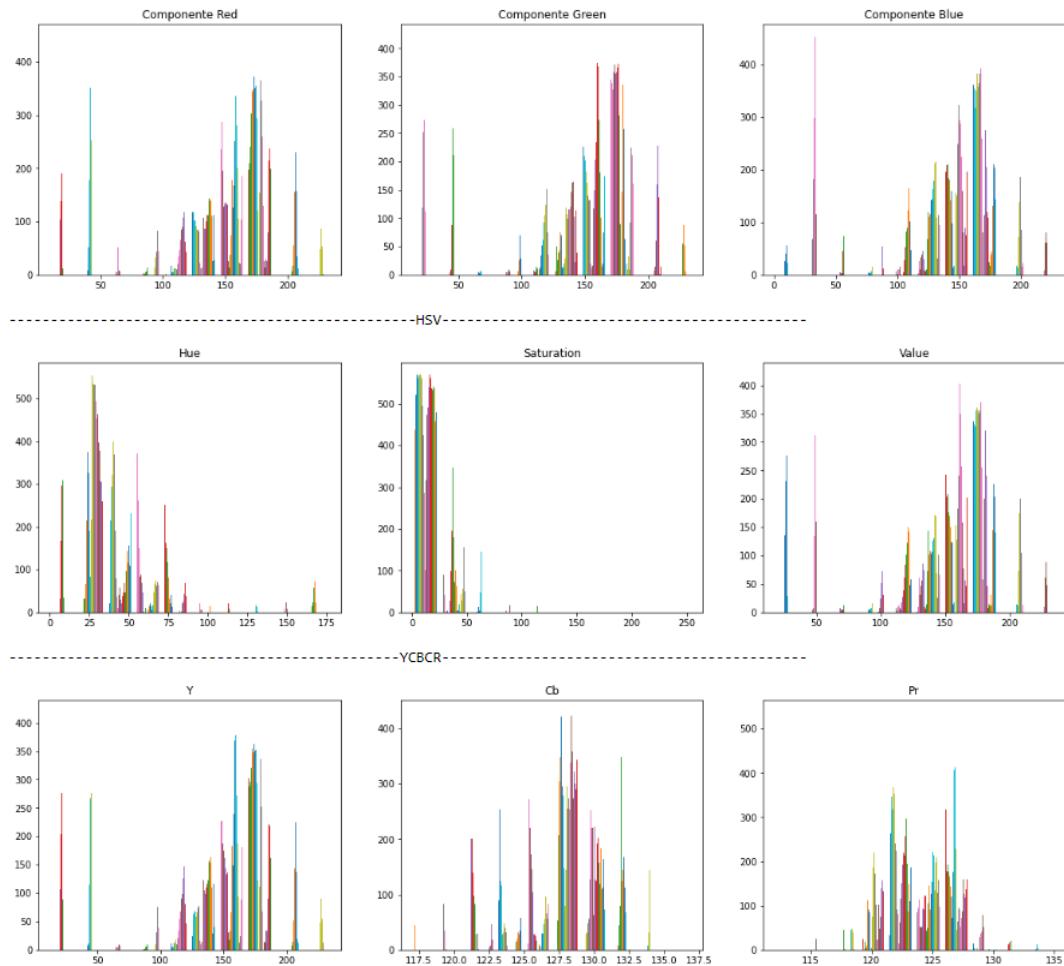


Figura 4.2: Histogramas das componentes RGB,HSV E YCBCR

Como é possível observar no espaço HSV possui valores mais distintos entre o chão e a linha.

Adicionalmente foi feito um teste equivalente, no entanto é filtrado para apenas uma linha do *frame* de modo a ter uma perspetiva mais precisa. A opção de analisar apenas uma linha foi devido a melhorar tempo de processamento sendo que o mesmo método foi usado para o projeto *Vision-Race*.

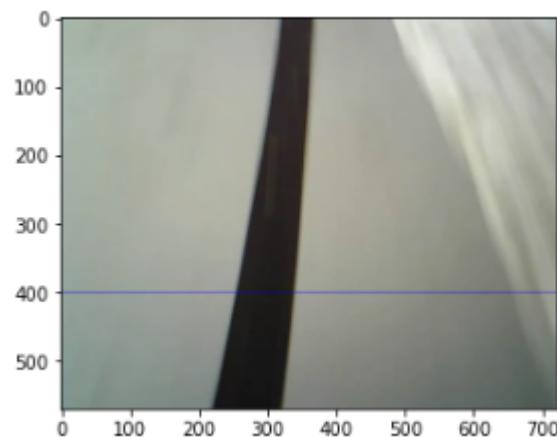


Figura 4.3: Fração da imagem das quais foram feitos os histogramas de todos os espaços de cor.

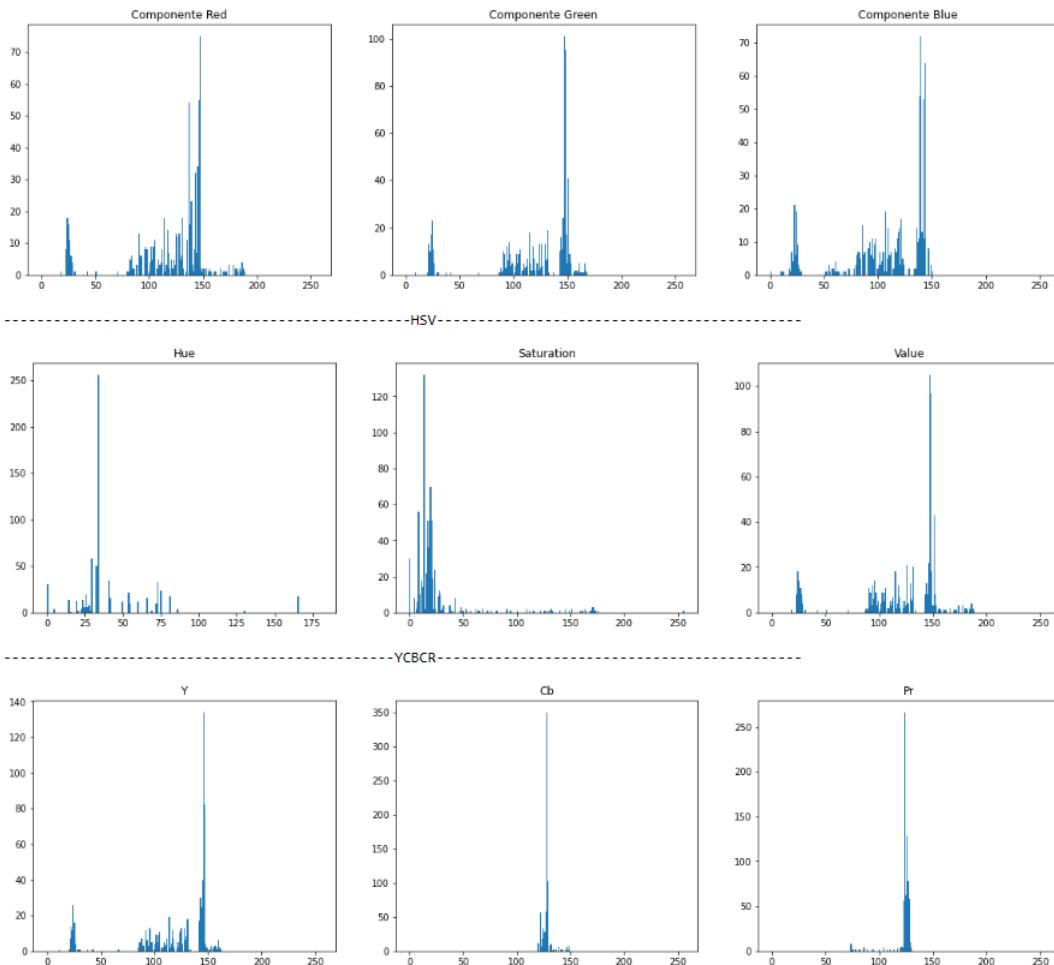


Figura 4.4: Histogramas de cor para a fração da imagem.

Observando os histogramas foi concluído que o espaço HSV é o espaço mais otimizado para realizar a binarização da imagem.

4.3 Calibração e Validação da mesma

Nesta secção iremos referir dois blocos. Dado que um depende do outro, é pertinente referir ambos na mesma secção do documento.

Para validar a calibração será introduzido *live feed*, ou seja o vídeo em direto, o utilizador terá um indicador visual composto por duas cores, verde e vermelho em que verde corresponde a que a linha está na posição adequada e vermelho caso o oposto se verifique. Também é possível introduzir um ajuste

na `gamma` que poderá ser benéfico atendendo à situação. A função irá devolver o componente cuja variação interclasse seja superior.

4.4 Pré-Processamento

Para melhorar a eficácia da binarização foram desenvolvidos módulos de pré-processamento, no entanto existem inúmeras técnicas de pré-processamento como tal foram realizados testes para obter quais as técnicas ideais para o problema.

4.4.1 Blur

Em primeiro lugar foi testado um *blur*, este efeito é concretizado através de alterar o valor de cada pixel através de uma mistura dos pixéis envolventes, no entanto existem diversos algoritmos que concretizam o mesmo efeito.

Como tal foram testadas três versões. Uma versão da binarização sem *blur*, uma com um *blur* por média em que cada pixel é calculado através de uma média dos pixéis adjacentes, e por fim um frame binarizado com um *blur* gaussiano em que o ruído introduzido pelo *blur* segue o valor de uma função Gaussiana.

É de notar que ambos os testes foram feitos com máscara (9,9) que é a maior máscara possível de aplicar pelo *blur* gaussiano, deste modo se houver uma diferença notável será mais perceptível do que uma máscara inferior.

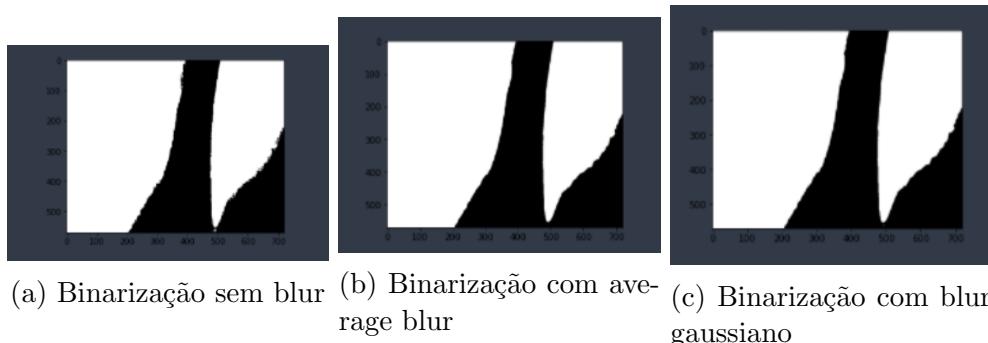


Figura 4.5: Testes de blur

Foi concluído então com este teste que realizar o *blur* não introduz diferenças significativas na binarização realizada, pelo que sendo assim será desnecessário realizar o algoritmo de modo a tentar manter o tempo de processamento o mais reduzido possível.

4.4.2 Ajuste automático de contraste

Para realizar o ajuste automático de contraste, foi utilizado um *snippet* de código escrito pelo utilizador **Nathancy**, no *StackOverflow*[8]

```

def convertScale(img, alpha, beta):
    new_img = img * alpha + beta
    new_img[new_img < 0] = 0
    new_img[new_img > 255] = 255
    return new_img.astype(np.uint8)

def automatic_brightness_and_contrast(image, clip_hist_percent=25):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Calculate grayscale histogram
    hist = cv2.calcHist([gray],[0],None,[256],[0,256])
    hist_size = len(hist)

    # Calculate cumulative distribution from the histogram
    accumulator = []
    accumulator.append(float(hist[0]))
    for index in range(1, hist_size):
        accumulator.append(accumulator[index - 1] + float(hist[index]))

    # Locate points to clip
    maximum = accumulator[-1]
    clip_hist_percent *= (maximum/100.0)
    clip_hist_percent /= 2.0

    # Locate left cut
    minimum_gray = 0
    while accumulator[minimum_gray] < clip_hist_percent:
        minimum_gray += 1

    # Locate right cut
    maximum_gray = hist_size - 1
    while accumulator[maximum_gray] >= (maximum - clip_hist_percent):
        maximum_gray -= 1

    # Calculate alpha and beta values
    alpha = 255 / (maximum_gray - minimum_gray)
    beta = -minimum_gray * alpha
    auto_result = convertScale(image, alpha=alpha, beta=beta)
    return (auto_result, alpha, beta)

```

Figura 4.6: Snippet de código criado por Nathancy

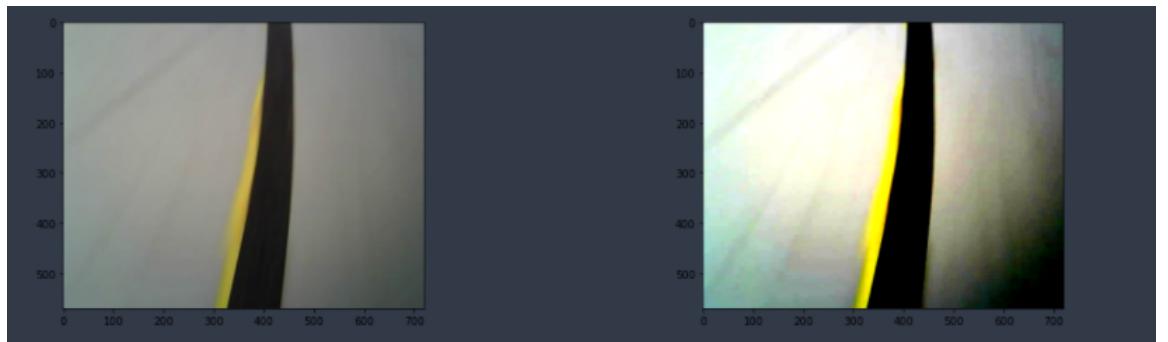


Figura 4.7: Output gerado pelo código

Para observar cada componente procedeu-se à visualização de cada componente através de *heatmaps*. Os *heatmaps* permitem à visualização, com a ajuda de uma escala de cores, os valores que cada componente possui no dado *frame*.

Para comparação da efetividade do código utilizado procedeu-se à comparação dos *heatmaps* antes e após o contraste automático, adicionalmente também foi executado a binarização.

Abaixo encontram-se as comparações realizadas:

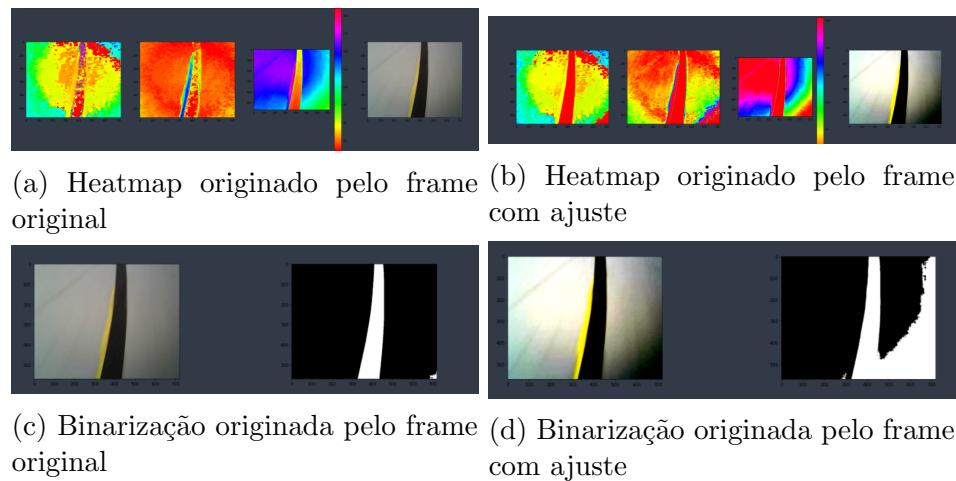


Figura 4.8: Testes de ajuste de gamma

Como observado nas duas comparações realizadas, o aumento do contraste traz ruído adicional à binarização sendo que não é aconselhável o uso

no nosso projeto.

4.5 Classificador para identificar a região desejada

4.5.1 Binarização

Método Otsu

Para a binarização das imagens obtidas foi desenvolvida uma função que realizará o método otsu.

O método otsu, inventado por Nobuyuki Otsu, é usado para obter um valor de thresholding automático em que o uso comum é binarizar uma imagem em escala de cinzento e é calculado primeiro dois pesos através do histograma e depois com base na média e no valor dos pesos é obtido a variação interclasse de cada valor possível na determinada escala apresentada. Por exemplo no caso de uma escala de 8 bits será obtido 256 valores de variação distintos e após calculado é obtido o índice que apresenta mais variação. Após alguns testes foi concluído que o componente com maior variação interclasse apresenta melhores resultados na binarização da mesma.

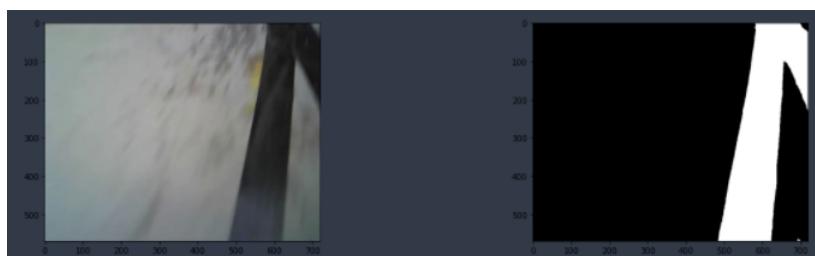


Figura 4.9: Teste do método otsu com base na variação interclasse

Threshold de mediana

Também foi testado comparativamente uma tentativa threshold de mediana sugerida por Michael Burdinov [6], no entanto os testes demonstraram que esta opção não é a mais ótima devido ao facto de não ser um algoritmo

robusto, ou seja que para cada frame seria necessário um ajuste de parâmetros de modo a obter uma binarização precisa.

```
def backgroundFilterMichael(img,median,mad,k = 1):
    imgCopy = np.zeros((len(img),len(img[0])))
    for linha in range(len(img)):
        for coluna in range(len(img[linha])):
            if img[linha,coluna] >= (median - k * mad) and img[linha,coluna] <= (median + k * mad) :
                imgCopy[linha,coluna] = 0

            else:
                imgCopy[linha,coluna] = 255

    return imgCopy
```

Figura 4.10: Snippet de código criado por Michael Burdinov

4.5.2 Deteção de linha

Para a deteção de linha foi usado um troço de código do projeto *Vision Race* e foi executado testes para observar a efetividade do mesmo.



Figura 4.11: Teste deteção de linha

Foi concluído que caso a binarização seja bem sucedida o algoritmo deteta bastante bem, no entanto caso haja duas linhas presentes no dado frame o algoritmo deteta apenas a linha que se encontra mais à esquerda.

4.6 Avaliador com base na distância da linha ao centro

Para avaliação foi desenvolvido uma função que irá calcular com base no centro da imagem à semelhança da divisão feita na calibração, é feita uma divisão entre o centro da linha e o centro da imagem a dividir por 10, esta conta irá devolver um valor de 0 - 20 em que os pontos que ultrapassam a linha central do *frame* terão valores superiores 10, para resolver essa exceção, caso a condição anterior seja presente é retirado o máximo que é possível ser obtido (20) e é subtraído a conta referida anteriormente.

Capítulo 5

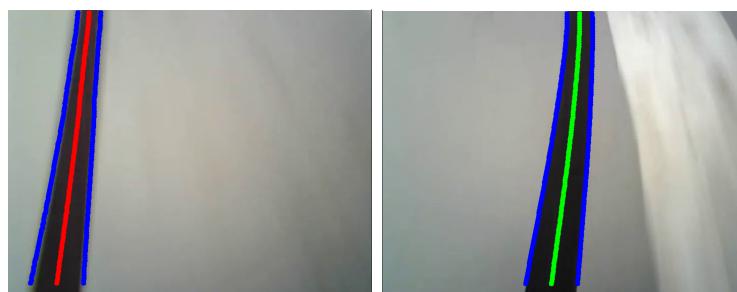
Validação e Testes

5.1 Validação Calibração

Para testar a calibração foi inserido um ficheiro vídeo, esse ficheiro, já previamente segmentado, servirá como substituto ao *live feed* que irá ser introduzido na aplicação final.

O centro da imagem foi calculado a partir do tamanho da *frame* original em que a coluna central do mesmo será metade do comprimento do *frame* e delimitamos uma área que será a que o utilizador terá de introduzir a linha, o *range* ou alcance da área foi delimitada através de um cálculo entre o valor obtido do centro da linha binarizada e o centro da imagem dividido por 10 sendo esse 10 a escala que irá ser utilizada na avaliação do patinador.

É possível observar os resultados do teste na imagem seguinte:



(a) Linha incorreta para calibração (b) Linha correta para calibração

Figura 5.1: Output gerado no teste da calibração

5.2 Validação avaliação

Após feita a função que irá calcular a performance do/a atleta, a mesma é implementada dentro da função de processamento e armazenado de modo a poder devolver métricas como a média de performance, etc. Abaixo encontram-se exemplos de avaliações geradas em vários cenários.

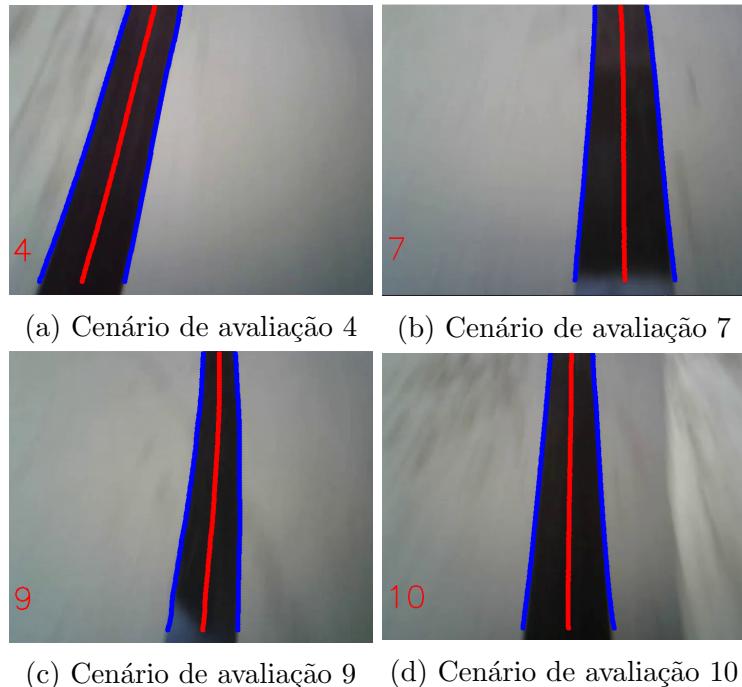
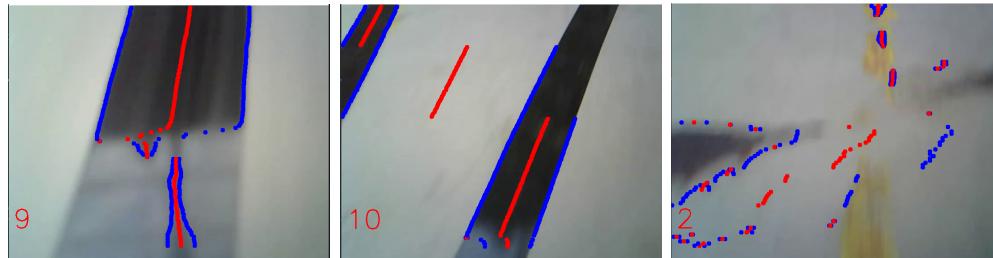


Figura 5.2: Output gerado no teste da avaliação

Também foi utilizado o mesmo algoritmo em imagem com condições de detecção mais difícil, no entanto apesar de as condições não proporcionarem uma binarização muito precisa, ainda é possível obter o contorno da linha em certos casos.

Seguem abaixo alguns casos em que o algoritmo possuiu algumas dificuldades a obter o contorno.



(a) Imagem com excesso de luminosidade (b) Imagem com duas linhas (c) Imagem com qualidade péssima

Figura 5.3: Testes de imagem com condições mais difícieis

Capítulo 6

Conclusões e Trabalho Futuro

Este trabalho foi apresentado com o intuito de projetar e desenvolver um sistema que permitisse aos/ás atletas treinarem sem a presença de um/a treinador/a, da forma mais otimizada e eficiente possível.

Como visto anteriormente, as imagens obtidas através do protótipo colocado nos patins dos atletas, passaram por vários segmentos de processamento onde foram realizadas diversas tarefas, como a calibração do sistema com as características físicas do ambiente, o pré-processamento, de forma a facilitar os passos seguintes, a binarização, o *thresholding*, e, no fim, a execução de um avaliador de performance baseado na centralidade da linha na imagem.

Existem alguns pontos chave que podiam ser melhorados, como por exemplo a nossa binarização, no entanto, em ambientes onde haja um elevado contraste da linha com o pavimento (que é esse o objetivo) o nosso algoritmo apresenta resultados bastante precisos.

Os resultados que foram obtidos onde observámos que foi feita uma boa binarização, foram bastante satisfatórios, dado que é possível delimitar bem a linha.

Para trabalhos futuros, pode indicar-se o estudo de novos métodos de análise de imagem, bem como outras abordagens para o pré-processamento e processamento das "frames".

Ainda, poderá incluir-se sensores de outros tipos diferentes, que obtenham outros tipos de informação para além da imagem (como por exemplo, um sensor que devolvesse a direção do patim).

Ainda, poderíamos ter várias imagens a ser analisadas de uma só vez, em vez

de só uma. Desta forma, ao compararmos a *frame* do momento com a anterior e a posterior, poderíamos obter resultados com mais precisão, mesmo em frames onde não é tão fácil realizar uma boa binarização.

Capítulo 7

Referências

- [1] Federação Portuguesa de Patinagem, “Regulamento Técnico da Patinação Artística,” [Online]. Available: <http://partistico.pt/Documentos/documentos1.php?dir=/2021/Regulamentos>. [Acedido em 17 de Agosto de 2021].
- [2] Sony, “Melhore o seu jogo com o Smart Tennis Sensor da Sony,” 14 Maio 2015. [Online]. Available: <http://forum.pplware.com/showthread.php?tid=21104.YTaL-45KhhE>. [Acedido em 18 de Agosto de 2021].
- [3] N. K. Corrêa, J. C. M. d. Lima, T. Russomano e M. A. d. Santos, “Development of a skateboarding trick classifier using accelerometry and machine learning,” [Online]. Available: <https://www.scielo.br/j/reng/a/sgsxHt4HffBYxDhqj9QD3dS/abstract/?lang=en> [Acedido em 21 de Agosto de 2021]
- [4] Instituto Politécnico de Lisboa, “IPLisboa vence concurso Born From Knowledge Ideas,” [Online]. Available: <https://www.ipl.pt/noticias/iplisboa-vence-concurso-born-knowledge-ideas>. [Acedido em 21 de Agosto de 2021].
- [5] ”IDICA — Instituto Politécnico de Lisboa”, [Online]. Available: <https://www.ipl.pt/id-inovacao/idica> [Acedido a 21 de Setembro de 2021].
- [6] ”How to chose the optimal HSV values for InRange thresholding in OpenCV”, [Online]. Available:

<https://stackoverflow.com/questions/26671027/how-to-choose-the-optimal-hsv-values-for-inrange-thresholding-in-opencv>. [Acedido em 10 de Junho de 2021].

[7] ”OpenCV Gamma Correction”, [Online] Available:
<https://www.pyimagesearch.com/2015/10/05/opencv-gamma-correction> [Acedido em 7 de Julho de 2021].

[8] ”Automatic contrast and brightness adjustment of a color photo of a sheet of paper with OpenCV”, [Online] Available:
<https://stackoverflow.com/questions/56905592/automatic-contrast-and-brightness-adjustment-of-a-color-photo-of-a-sheet-of-paper>. [Acedido a 23 de Julho de 2021].

Bibliografia

- https://docs.opencv.org/master/da/d22/tutorial_py_canny.html
- https://en.wikipedia.org/wiki/Gamma_correction
- https://scikit-image.org/docs/dev/auto_examples/filters/plot_hysteresis.html
- <https://pt.wikipedia.org/wiki/Histerese>
- <https://en.wikipedia.org/wiki/YCbCr>
- <https://pt.wikipedia.org/wiki/RGB>
- <https://pt.wikipedia.org/wiki/HSV>
- <https://colorpicker.me/5e7070>
- <https://learnopencv.com/otsu-thresholding-with-opencv/>
- <https://create.arduino.cc/projecthub/Aasai/lane-following-robot-using-opencv-da3d45>
- <https://const-toporov.medium.com/line-following-robot-with-opencv-and-contour-based-approach-417b90f2c298>
- https://www.researchgate.net/publication/346311568_Vision-Based_Line_Following_Robot_in_Webots
- <https://github.com/CRM-UAM/VisionRace>
- ”Sensorização de Patins para monitorização de movimentos”, por Bárbara Pires, Setembro 2019

- ”Deteção de desvios em percursos de Figuras Obrigatórias na Patinação Artística”, por Edgar Santinhos, em Novembro de 2020
- ”Automatic Thresholding of Gray-Level Pictures Using Two-Dimensional Otsu Method”, por Liu Jianzhuang. Li Wenqing. e Tian Yupeng 1991