UNIVERSIDADE FEDERAL DO PARANÁ

MARIA LUIZA SAMPAIO LOGRADO

MODELAGEM DE FEIÇÕES FACIAIS COM *SPLINES* PARA RECONHECIMENTO POR *DYNAMIC TIME WARPING*

CURITIBA

2025

MARIA LUIZA SAMPAIO LOGRADO

MODELAGEM DE FEIÇÕES FACIAIS COM *SPLINES* PARA RECONHECIMENTO POR *DYNAMIC TIME WARPING*

Trabalho apresentado como requisito parcial à coclusão do curso de graduação em Matemática Industrial no Setor de Exatas da Universidade Federal do Paraná.

Orientador: Prof Thiago de Oliveira Quinelato, DSc.

CURITIBA

2025

RESUMO

Este trabalho investiga o uso de *splines* cúbicas como método de representação de características faciais em sistemas biométricos. A proposta tem como objetivo avaliar se a modelagem das formas faciais por meio de curvas suaves pode contribuir para uma identificação mais eficiente e interpretável de indivíduos. O processo inicia-se com a extração automática de pontos a partir de imagens faciais, utilizando técnicas como detecção com Haar Cascades, contornos via operador de Canny e redução de pontos com base em grafos e árvores geradoras mínimas. Em seguida, as curvas obtidas são modeladas com *splines* cúbicas, que fornecem uma representação contínua e compacta das feições. Para realizar a comparação entre diferentes indivíduos, emprega-se o algoritmo *Dynamic Time Warping*, capaz de alinhar sequências mesmo com variações espaciais. Os resultados mostram que a abordagem é viável e apresenta bom desempenho na tarefa de reconhecimento facial, destacando o potencial de combinar métodos geométricos com algoritmos clássicos de comparação temporal. O trabalho também aponta caminhos para futuras melhorias, como o uso de expressões faciais variadas, ajustes nos hiperparâmetros de pré-processamento e testes com bases de dados mais desafiadoras.

Palavras-chaves: reconhecimento facial, *splines* cúbicas, DTW, biometria.

ABSTRACT

This work investigates the use of cubic splines as a method for representing facial features in biometric systems. The aim is to assess whether modeling facial shapes using smooth curves can contribute to more efficient and interpretable individual identification. The process begins with the automatic extraction of key points from facial images, using techniques such as Haar Cascade detection, contour extraction via the Canny operator, and point reduction based on graph structures and minimum spanning trees. Next, the resulting curves are modeled with cubic splines, which provide a continuous and compact representation of facial features. To compare different individuals, the Dynamic Time Warping algorithm is employed, as it can align sequences even in the presence of spatial variations. The results show that the proposed approach is feasible and performs well in the task of facial recognition, highlighting the potential of combining geometric methods with classical temporal comparison algorithms. The work also outlines paths for future improvements, such as incorporating varied facial expressions, adjusting preprocessing hyperparameters, and testing with more challenging datasets.

Key-words: facial recognition, cubic splines, DTW, biometrics

SUMÁRIO

1 INTRODUÇÃO

O reconhecimento facial é uma das áreas mais promissoras e desafiadoras dentro do campo da biometria. Sua aplicação abrange desde sistemas de segurança até autenticação em dispositivos móveis, tornando-se cada vez mais presente no cotidiano. Nos últimos anos, métodos baseados em aprendizado profundo, especialmente redes neurais convolucionais (CNNs), têm dominado esse campo devido à sua alta acurácia. No entanto, tais métodos apresentam algumas limitações, especialmente no que diz respeito à interpretabilidade dos modelos e à necessidade de grandes volumes de dados para treinamento.

Nesse contexto, a busca por alternativas mais eficientes e interpretáveis tem ganhado relevância. Uma dessas alternativas é o uso de *splines*, funções matemáticas capazes de gerar curvas suaves a partir de um conjunto reduzido de pontos. Ao representar as características faciais por meio de curvas, é possível reduzir significativamente a redundância dos dados, além de tornar o processo de extração e análise mais estruturado e compreensível.

Este trabalho propõe uma abordagem híbrida que combina a expressividade das *splines* com técnicas de aprendizado de máquina para o reconhecimento facial. A metodologia baseia-se na extração de contornos das principais características do rosto, sua modelagem com *splines* e posterior classificação utilizando algoritmos de comparação de sequências, como o *Dynamic Time Warping* (DTW) (Sakoe; Chiba, 1978). Espera-se, com isso, alcançar uma representação mais compacta e interpretável das feições faciais, sem comprometer o desempenho do sistema de identificação.

A introdução dessa nova abordagem se justifica não apenas pela busca por eficiência computacional, mas também pela necessidade de modelos que ofereçam maior transparência quanto às decisões tomadas. Dessa forma, o presente trabalho contribui para a ampliação das possibilidades de representação biométrica e propõe caminhos para o desenvolvimento de sistemas mais acessíveis, leves e explicáveis.

Os códigos implementados neste trabalho estão disponíveis no repositório do GitHub Lograd (2025) e foram desenvolvidos utilizando a linguagem de programação Python.

1.1 OBJETIVO

O objetivo deste trabalho é investigar a viabilidade do uso de *splines* como método de representação de características faciais em sistemas biométricos, avaliando sua eficácia na melhoria do desempenho da identificação de indivíduos. A abordagem

proposta visa combinar a expressividade das *splines* com a capacidade do DTW para reconhecimento facial, buscando uma solução que seja ao mesmo tempo eficiente e interpretável.

1.2 JUSTIFICATIVA

Métodos convencionais baseados em CNNs e aprendizado profundo apresentam altos níveis de acurácia, mas muitos desses modelos operam como “caixas-pretas”, dificultando a interpretação das características extraídas e utilizadas para a tomada de decisão.

Diante desse cenário, a abordagem baseada no uso de *splines* surge como uma alternativa para representação de características faciais de forma mais estruturada e interpretável. As *splines* são funções matemáticas que permitem a modelagem de curvas suaves a partir de um conjunto reduzido de pontos de controle, garantindo uma representação eficiente das características faciais sem a necessidade de armazenar grandes quantidades de dados redundantes.

Além da compactação da informação, o uso de *splines* facilita a extração de características relevantes, permitindo que apenas as estruturas mais significativas do rosto sejam utilizadas como entrada para o modelo de aprendizado. Isso pode melhorar a eficiência do DTW, aumentando a precisão na identificação.

Dessa forma, este trabalho se justifica pela necessidade de explorar métodos alternativos para reconhecimento facial que combinem eficiência computacional e interpretabilidade.

1.3 METODOLOGIA

A abordagem proposta para o reconhecimento facial baseada em *splines* segue um fluxo estruturado, dividido em cinco etapas principais: detecção de características faciais, extração de contornos, redução de pontos, modelagem com *splines* e classificação por DTW.

É importante esclarecer que a primeira versão das 4 primeiras etapas do método proposto neste trabalho foi desenvolvida durante o Programa de Voluntariado Acadêmico (PVA) em 2024 e neste TCC será desenvolvida a última etapa, que é a classificação por DTW, além de algumas melhorias nas etapas anteriores.

Por fim, trechos da pesquisa neste trabalho utilizam o banco de dados FERET (Phillips *et al.*, 1998; Phillips *et al.*, 2000) de imagens faciais, coletado sob o programa FERET, patrocinado pelo Escritório do Programa de Desenvolvimento de Tecnologia Antidrogas do Departamento de Defesa dos EUA (DOD).

1.3.1 Detecção de Características Faciais

Inicialmente, são identificadas as principais regiões do rosto, como olhos, boca e nariz. Para isso, é utilizado o modelo Haar Cascade (Viola; Jones, 2001; OpenCV, s.d.[b]) como a técnica de detecção. O modelo é ajustado para reconhecer padrões de características faciais em imagens, permitindo a localização das regiões de interesse.

1.3.2 Extração de Contornos

Após a identificação das regiões faciais, são extraídos os contornos dessas características por meio da técnica de processamento de imagens de detecção de bordas utilizando o operador de Canny (segmentação baseada em gradientes) (Canny, 1986; OpenCV, s.d.[a]). O objetivo é obter um conjunto inicial de pontos que descrevem as características faciais.

1.3.3 Simplificação dos Contornos com Estruturas de Grafos

Para reduzir a complexidade computacional, os pontos extraídos nos contornos são submetidos a um processo de simplificação. Estruturas de dados como grafos são utilizadas para armazenar os pontos e suas vizinhanças. A partir desses grafos, é possível aplicar algoritmos para identificação de seus componentes conexos (Virtanen *et al.*, 2020), mantendo aqueles que possuem os maiores nós, ou seja, os pontos mais significativos.

1.3.4 Otimização Estrutural com Árvore Geradora Mínima

Para garantir que as curvas não apresentem ciclos, é aplicada a técnica de árvore geradora mínima. Essa técnica permite selecionar um subconjunto de arestas que conecta todos os pontos sem formar ciclos, resultando em uma representação mais limpa e eficiente das características faciais. Após essa etapa, é implementado um algoritmo para encontrar o maior caminho entre os pontos e remover os pontos intermediários, resultando em uma curva mais simplificada.

1.3.5 Classificação por *Dynamic Time Warping*

Por fim, os pontos gerados a partir dessas curvas são utilizados como entrada para um algoritmo de DTW (Sakoe; Chiba, 1978; Tavenard, 2021). O DTW é um algoritmo que mede a similaridade entre duas sequências temporais, permitindo o alinhamento não-linear entre elas. Essa abordagem é especialmente útil para lidar com variações na velocidade e, neste trabalho, na forma das características faciais.

1.4 REFERENCIAL TEÓRICO

A ideia de representar características faciais por meio de *splines* é discutida por Maggini *et al.* (2007), que apresentam uma abordagem baseada em *splines* Catmull-Rom para modelagem de curvas suaves.

O uso do algoritmo de DTW para reconhecimento facial é abordado por Levada *et al.* (2008), que discutem a eficácia do DTW na comparação de sequências temporais, destacando sua aplicabilidade em tarefas de reconhecimento facial. No entanto, a abordagem proposta no artigo difere da adotada neste trabalho, uma vez que os autores concentram-se na análise das variações das cores dos pixels da imagem, em vez de explorar a forma ou a geometria das características faciais.

1.5 CRONOGRAMA

A Tabela 1 apresenta o cronograma de atividades para o desenvolvimento do trabalho.

TABELA 1 – Cronograma de Atividades

Atividade	**Mês 1**	**Mês 2**	**Mês 3**	**Mês 4**
Refatoração do código	X			
Revisão Bibliográfica	X	X	X	X
Estudo sobre *Dynamic Time Warping*		X	X	
Implementação do DTW		X	X	
Escrita da Monografia	X	X	X	X
Defesa do Trabalho				X

2 EXTRAÇÃO DE PONTOS DE FORMA AUTOMÁTICA

Neste capítulo será apresentado o método de extração automática de pontos, de forma a facilitar a modelagem de *splines* para reconhecimento facial. O método proposto consiste em um algoritmo que utiliza técnicas de processamento de imagem e otimização para extrair pontos relevantes em imagens faciais. A abordagem é baseada na detecção de características faciais, seguida pela extração de contornos e filtragem dos pontos obtidos.

Inicialmente, realizamos uma pesquisa para identificar as principais características faciais a serem extraídas, decidindo focar na detecção dos olhos, nariz e boca. Para isso, utilizamos o algoritmo Haar Cascade (Viola; Jones, 2001) para detectar o rosto da pessoa na imagem, concentrando a análise nessa área específica e facilitando a detecção das características menores.

Em seguida, utilizamos o método Canny do OpenCV (OpenCV, s.d.[a]; Canny, 1986) para extrair os contornos das características faciais. O Canny é um algoritmo de detecção de contornos eficiente que nos ajuda a identificar os limites das características faciais com maior precisão.

Após a extração dos contornos, suprimimos alguns pontos fazendo o uso de grafos e árvores geradoras mínimas. Essa etapa é crucial para reduzir a quantidade de pontos a serem utilizados na modelagem das *splines*, mantendo apenas os pontos mais significativos que representam as características faciais. A simplificação dos pontos é realizada utilizando o algoritmo construído de forma autoral e implementado em Python, que utiliza a biblioteca *Scipy* (Virtanen *et al.*, 2020) para encontrar a árvore geradora mínima. Essa abordagem garante que os pontos extraídos sejam representativos e relevantes para a modelagem das *splines*.

Por fim, ainda é possível aplicar um filtro de suavização para diminuir a quantidade de pontos, fazendo isso de forma completamente aleatória, mantendo apenas os pontos iniciais e finais de cada segmento, o que pode ser útil para melhorar a qualidade da modelagem das *splines*.

2.1 MODELOS PARA DETECÇÃO DE CARACTERÍSTICAS

Foi utilizada uma abordagem popular de detecção chamada Haar Cascade, implementada com OpenCV e Python. Este método, introduzido e estudado no artigo (Viola; Jones, 2001), permite a detecção eficaz das características faciais.

O classificador Haar Cascade já foi calibrado e validado com um vasto conjunto

de dados de rostos humanos, eliminando a necessidade de calibração adicional. Basta carregar o classificador da biblioteca e utilizá-lo para detectar rostos em uma imagem de entrada.

Para distinguir com precisão entre amostras que contêm ou não um rosto humano, utilizamos um classificador forte, resultante da combinação de diversos classificadores. Esta técnica envolve a utilização de uma cascata de classificadores para identificar diferentes características em uma imagem.

Os seguintes classificadores foram utilizados:

- `haarcascade_frontalface_default.xml`
- `haarcascade_eye.xml`
- `haarcascade_mcs_nose.xml`
- `haarcascade_mcs_mouth.xml`

Todos os classificadores mencionados podem ser encontrados no repositório do OpenCV no GitHub (OpenCV, s.d.[b]).

2.1.1 Parâmetros

Após carregar os classificadores, podemos utilizá-los aplicando quatro parâmetros específicos para cada um.

O método `detectMultiScale()` é utilizado para identificar faces de diferentes tamanhos na imagem de entrada. Os quatro parâmetros principais deste método são detalhados a seguir:

- `image`:
  - O primeiro parâmetro é a imagem em tons de cinza, utilizada como entrada do método por facilitar a detecção das características faciais.
- `scaleFactor`:
  - Este parâmetro é usado para reduzir o tamanho da imagem de entrada, facilitando a detecção de faces maiores pelo algoritmo. Especificamos um fator de escala de 1.1, indicando que queremos reduzir o tamanho da imagem em 10%.
- `minNeighbors`:

  - O classificador em cascata aplica uma janela deslizante através da imagem para detectar faces. Essas janelas são representadas como retângulos. Inicialmente, o classificador captura um grande número de falsos positivos. O parâmetro `minNeighbors` especifica o número de retângulos vizinhos que precisam ser identificados para que um objeto seja considerado uma detecção válida, ou seja, valores pequenos como 0 ou 1 resultam em muitos falsos positivos, enquanto valores grandes podem levar à perda de verdadeiros positivos. É necessário encontrar um equilíbrio que elimine falsos positivos e identifique com precisão os verdadeiros positivos.

- `minSize`:

  - Este parâmetro define o tamanho mínimo do objeto a ser detectado. O modelo ignorará faces menores do que o tamanho mínimo especificado.

Foram colocados como *default* para todos os classificadores os seguintes valores:

```
detectMultiScale(image, scaleFactor=1.1, minNeighbors=5, minSize=(40, 50))
```

2.1.2 Resultado da Detecção das Características

Para realizar a detecção das características faciais, inicialmente identificamos o rosto na imagem. Esse processo retorna um *array* com quatro valores: as coordenadas x e y do ponto onde o rosto foi detectado, além de sua largura e altura. Em seguida, recortamos a imagem nessa região, de modo a isolar a face da pessoa.

Com o rosto isolado, aplicamos classificadores específicos para detectar o nariz, a boca e os olhos.

Essa abordagem, que realiza a detecção passo a passo, garante maior precisão na identificação das características menores, pois concentra a análise na área previamente delimitada pelo rosto.

Considere a FIGURA 1 , onde a detecção das características faciais é realizada:

1. **Detecção do Rosto:** Para identificar e isolar o rosto na imagem usamos o classificador `haarcascade_frontalface_default.xml`.

2. **Detecção dos Olhos:** Aplicamos o classificador `haarcascade_eye.xml` para localizar os olhos dentro da área do rosto previamente detectada.

3. **Detecção do Nariz:** Utilizamos o classificador `haarcascade_mcs_nose.xml` para identificar o nariz na mesma área delimitada.

4. **Detecção da Boca:** Finalmente, aplicamos o classificador `haarcascade_mcs_mouth.xml` para localizar a boca.

2.2 EXTRAÇÃO DE CONTORNOS

A detecção de contornos é essencial para a análise de imagens, pois permite a identificação de contornos e a segmentação de objetos em uma cena. O algoritmo de Canny (Canny, 1986) é um dos métodos mais populares devido à sua eficiência e precisão. Esta seção irá explorar o funcionamento do algoritmo de Canny e demonstrará sua aplicação prática com a biblioteca OpenCV (OpenCV, s.d.[a]).

2.2.1 Algoritmo de Canny

O algoritmo de detecção de contornos de Canny é composto por várias etapas, conforme descrito a seguir:

1. **Redução de Ruído**: A imagem é suavizada com um filtro Gaussiano para minimizar a interferência do ruído na detecção de contornos.
2. **Cálculo do Gradiente de Intensidade**: Os gradientes de intensidade da imagem são calculados nas direções horizontal e vertical, utilizando os operadores de Sobel. As componentes do gradiente são representadas por $G_x = \frac{\partial I}{\partial x}$ e $G_y = \frac{\partial I}{\partial y}$, onde I denota a imagem original. A magnitude do gradiente é então dada por:

$$Contorno\_Gradiente(G) = \sqrt{G_x^2 + G_y^2}$$

3. **Supressão Não Máxima**: Elimina *pixels* que não são máximos locais no gradiente, preservando apenas os que representam contornos fortes.
4. **Rastreamento de Contornos por Histerese**: Classifica os contornos detectados usando dois limiares, *minVal* e *maxVal*:
   - Contornos com gradiente maior que *maxVal* são fortes.
   - Contornos com gradiente menor que *minVal* são descartados.
   - Contornos entre *minVal* e *maxVal* são fracos e mantidos apenas se conectados a contornos fortes.

Por exemplo, na FIGURA 2, o contorno A está acima do *maxVal*, sendo assim considerado um contorno forte. Embora o contorno C esteja abaixo do *maxVal*, ele está conectado ao contorno A, de modo que também é considerado como contorno válido,

FIGURA 1 – DETECÇÕES.

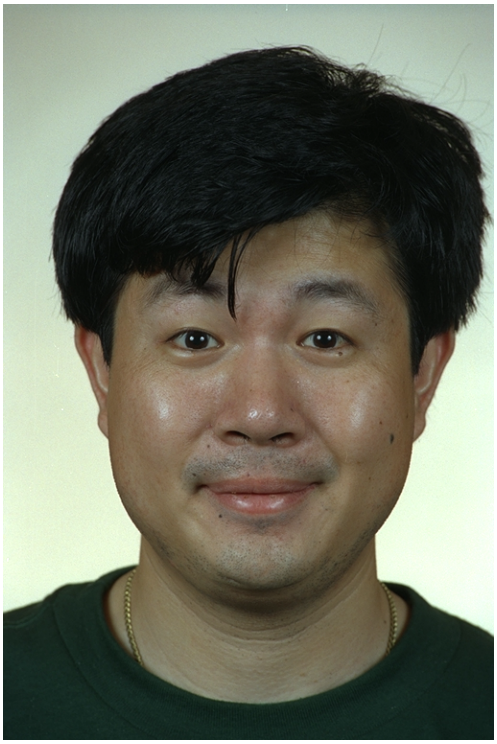
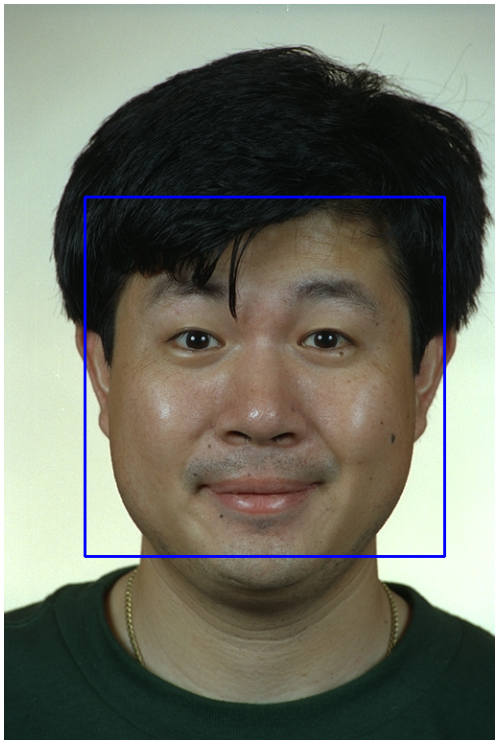
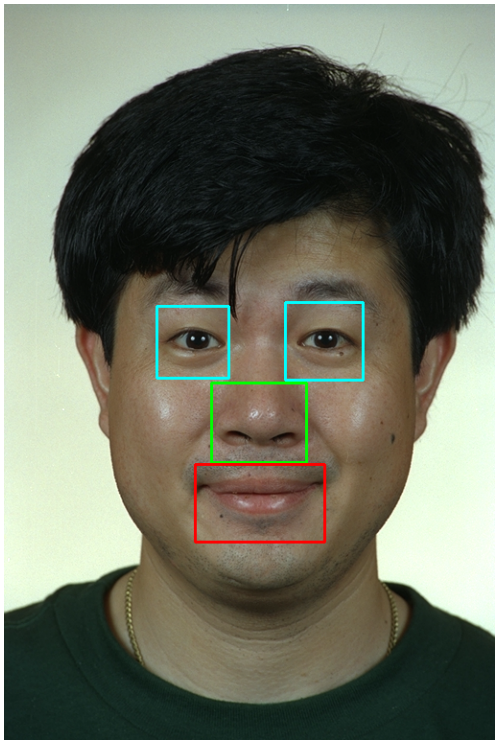


Imagem Original.



1º Detecção da face.

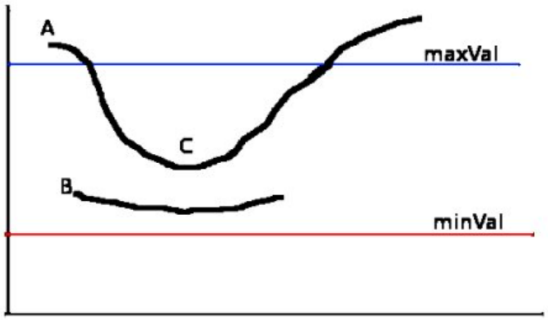


2º Detecção das características.

FONTE: (Phillips *et al.*, 1998; Phillips *et al.*, 2000)

resultando em uma curva completa. Já o contorno B, embora esteja acima do *minVal*, não está conectado a nenhum contorno forte e, portanto, é descartado.

FIGURA 2 – RASTREAMENTO DE CONTORNO POR HISTERESE



FONTE: (OpenCV, s.d.[a])
LEGENDA: No eixo horizontal estão as curvas parametrizadas e no eixo vertical estão os valores de intensidade do gradiente.

É crucial selecionar adequadamente os valores de *minVal* e *maxVal* para obter resultados precisos. Esse estágio também ajuda a remover pequenos ruídos de *pixels*, assumindo que os contornos são linhas longas e contínuas.

2.2.2 Resultado da Detecção de Contornos

Ao aplicar o algoritmo de Canny para a detecção de contornos, obtemos um *array* binário onde os valores ‘0’ e ‘255’ representam diferentes tipos de *pixels*. Os *pixels* com valor ‘0’ são considerados descartáveis, ou seja, não fazem parte dos contornos detectados. Em contraste, os *pixels* com valor ‘255’ são os que definem os contornos, indicando regiões de mudança de intensidade significativa na imagem original.

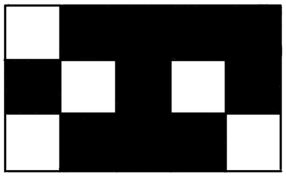
A FIGURA 3 ilustra a matriz Canny e os contornos resultantes.

FIGURA 3 – MATRIZ CANNY E CONTORNOS.

Matriz Canny

$$\begin{bmatrix} 255_1 & 0 & 0 & 0 & 0 \\ 0 & 255_2 & 0 & 255_3 & 0 \\ 255_4 & 0 & 0 & 0 & 255_5 \end{bmatrix} \rightarrow$$

Contornos

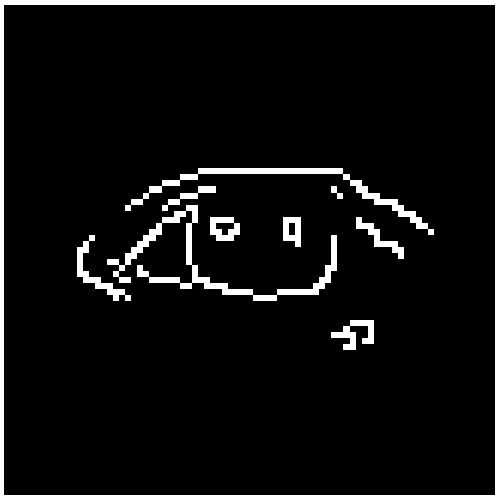


A FIGURA 4 apresentada os resultados da aplicação do algoritmo de Canny em imagens de características faciais.

Esses resultados demonstram como o algoritmo de Canny é eficaz para realçar os contornos em diferentes tipos de imagens, permitindo uma análise visual clara e detalhada das características principais em cada exemplo.

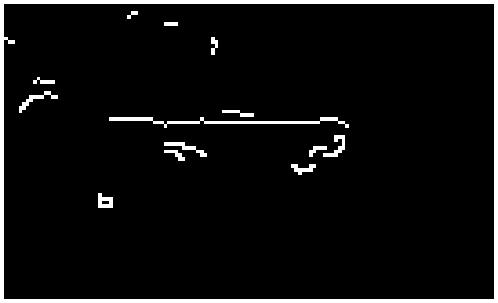
FIGURA 4 – APLICAÇÃO DO ALGORITMO DE CANNY.

Olho esquerdo.



Olho direito.

Nariz.



Boca.

2.3 REDUÇÃO DE PONTOS EM DETECÇÃO DE CONTORNOS

Embora o algoritmo de Canny seja excelente para capturar os contornos, nem sempre é necessário utilizar todos os pontos detectados ao passar uma *spline*. Para a modelagem de contornos suaves e contínuos, é suficiente selecionar um subconjunto representativo dos pontos detectados. Esse processo pode envolver a amostragem ou a simplificação de alguns desses pontos, garantindo que a *spline* mantenha a forma geral do contorno sem a complexidade de lidar com todos os pontos individuais.

Esta parte é realizada em cinco etapas principais:

1. **Construção do Grafo:** Os pontos de contorno detectados são organizados em um grafo, onde cada ponto é um nó e as vizinhanças entre eles representam as arestas. Essa estrutura facilita a análise e a manipulação dos pontos.
2. **Identificação de Componentes Conexos:** A partir do grafo, são identificados os componentes conexos, mantendo apenas os de maior tamanho. Isso é feito

para garantir que apenas as partes mais significativas dos contornos sejam consideradas, eliminando ruídos e pontos irrelevantes.

3. **Árvore Geradora Mínima:** A partir dos maiores componentes conexos, uma árvore geradora mínima é construída. Essa árvore conecta todos os pontos de forma a remover ciclos, resultando em uma representação mais limpa e eficiente dos contornos.

4. **Otimização com Poda da Árvore:** A árvore geradora mínima é otimizada, removendo mais uma vez ruídos e mantendo apenas os pontos mais significativos.

5. **Filtragem Aleatória:** Por fim, uma filtragem aleatória é aplicada para suavizar ainda mais a representação, mantendo apenas os pontos iniciais e finais de cada segmento. Essa etapa garante que a *spline* resultante seja suave e contínua.

2.3.1 Grafos

Para a construção do grafo, utilizamos os pontos de contorno detectados pelo algoritmo de Canny. Cada ponto é representado como um nó no grafo, e as vizinhanças entre os nós são representadas como arestas. Neste trabalho, a matriz de adjacências é utilizada para armazenar essas vizinhanças.

O código foi feito de forma autoral em Python, utilizando a fórmula Manhattan para calcular a distância entre os nós. Essa fórmula é adequada para o nosso caso, pois considera apenas as distâncias horizontais, verticais e diagonais, o que é suficiente para que o grafo represente adequadamente a estrutura dos contornos faciais.

Sejam os pontos $i = (x_1, y_1)$ e $j = (x_2, y_2)$. Definimos a distância Manhattan $d(i, j)$ e a vizinhança $v(i, j)$ como:

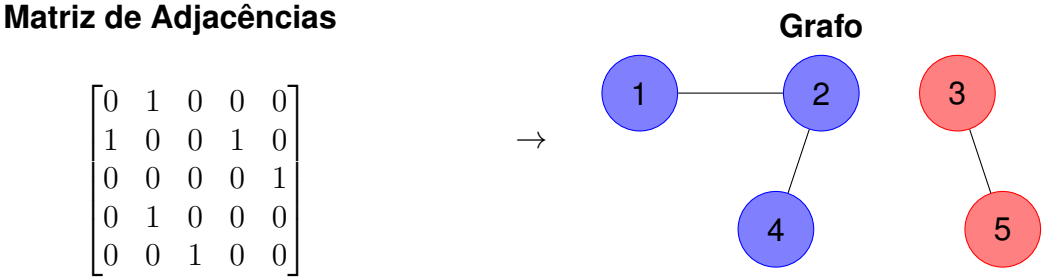
$$d(i,j) = \begin{cases} 1, & \text{se } (x_1 = x_2 \vee y_1 = y_2) \text{ e } (x_1, y_1) \neq (x_2, y_2) \\ 2, & \text{se } |x_1 - x_2| = 1 \text{ e } |y_1 - y_2| = 1 \\ \infty, & \text{caso contrário} \end{cases} \tag{2.1}$$

$$v(i,j) = \begin{cases} 1, & \text{se } d(i,j) = 1 \text{ ou } d(i,j) = 2 \\ 0, & \text{caso contrário} \end{cases} \tag{2.2}$$

A partir da matriz resultante da detecção de bordas por Canny (FIGURA 3), é possível construir a matriz de adjacências com base na distância de Manhattan (2.1) e na definição de vizinhança adotada (2.2). Essa matriz representa, de forma matricial, as conexões entre os pixels considerados como nós do grafo, onde cada elemento a_{ij} indica a existência (ou não) de uma aresta entre os nós i e j.

A FIGURA 5 ilustra tanto a matriz de adjacências quanto o grafo correspondente aos contornos identificados na FIGURA 3. Na matriz, cada linha e coluna correspondem a um nó, e os valores registrados indicam a presença ou ausência de conexões entre os pares de nós.

FIGURA 5 – MATRIZ DE ADJACÊNCIAS E GRAFO.



$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

2.3.2 Grafos Conexos e Desconexos

Os grafos podem ser classificados em dois tipos principais: grafos conexos e grafos desconexos. Um grafo é considerado conexo se existe pelo menos um caminho entre qualquer par de nós do grafo. Por outro lado, um grafo desconexo possui pelo menos um par de nós que não estão conectados por um caminho.

Um grafo desconexo pode ser dividido em componentes conexos, que são subgrafos conexos. Cada componente conexo é um subconjunto do grafo original onde há pelo menos um caminho conectando cada par de nós, mas não há conexão com os nós de outros componentes. Essa distinção é importante para entender a estrutura do grafo e como ele pode ser analisado e manipulado.

FIGURA 6 – COMPONENTES CONEXOS REPRESENTADOS POR GRAFOS.

Nesta etapa do trabalho, o objetivo principal é reduzir ruídos nas imagens por meio da identificação de componentes conexas. Essa abordagem permite isolar regiões com alta densidade de *pixels* conectados – geralmente associadas a estruturas relevantes – e descartar pequenas componentes, que tendem a representar ruído ou informações irrelevantes.

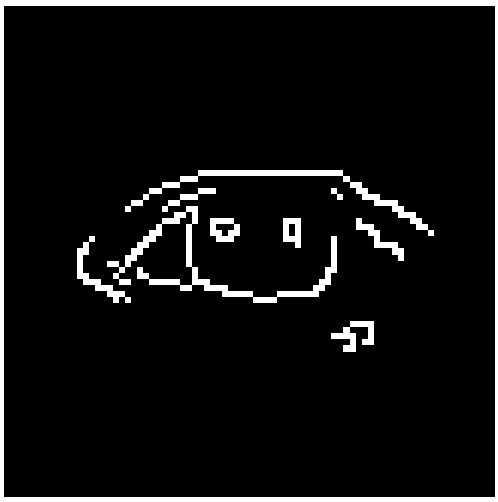
2.3.3 Resultado da Construção do Grafo

Podemos observar que a imagem original é composta por um grande número de pontos, porém utilizando a filtragem de *pixels* através de grafos com os maiores

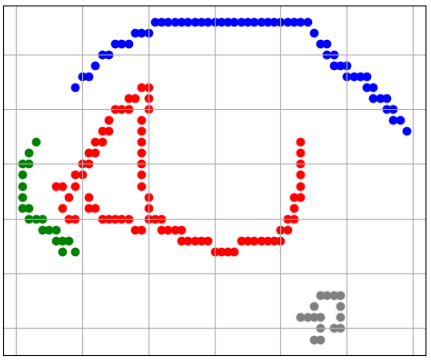
componentes conexos, conseguimos reduzir a quantidade de pontos, mantendo apenas os mais relevantes. A FIGURA 7 mostra o resultado da construção do grafo e a filtragem dos pontos.

FIGURA 7 – COMPONENTES CONEXOS OLHO DIREITO.

Olho direito identificado.



Ênfase nos contornos.



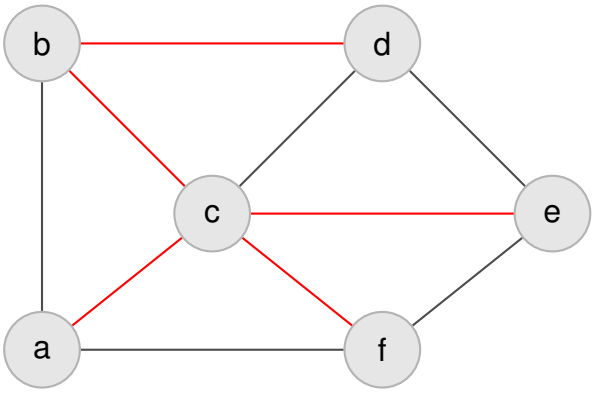
Remoção de alguns pontos através dos maiores compenentes conexos.

2.3.4 Árvore Geradora Mínima

A árvore geradora mínima (AGM) é uma estrutura de dados fundamental em teoria dos grafos, que conecta todos os nós de um grafo com o menor custo total possível. Essa estrutura é especialmente útil em aplicações de processamento de imagens, onde a simplificação de contornos e a redução do número de pontos são necessárias para facilitar a modelagem e a análise. Neste trabalho foi utilizada a biblioteca *Scipy* (Virtanen *et al.*, 2020) para encontrar a AGM.

Cada nó na AGM tem uma relação hierárquica com outros nós:

FIGURA 8 – ÁRVORE GERADORA MÍNIMA.



Em vermelho é destacado o caminho da AGM.

- O nó raiz é o nó inicial a partir do qual a árvore é construída ou percorrida.
- Um nó filho é aquele que se origina de outro nó – chamado de pai – seguindo essa ideia de crescimento ou expansão da árvore.
- A folha é um nó que não tem filhos, ou seja, ela está no fim de um caminho na árvore

Para este trabalho, a AGM é utilizada para conectar os pontos de contorno detectados, formando uma estrutura que representa as características faciais de forma eficiente removendo ciclos. A FIGURA 8 ilustra a Ávore Geradora Mínima construída entre os pontos de contorno, removendo os ciclos.

2.3.5 Otimização da Árvore Geradora Mínima

Para otimizar a análise, aplicamos uma poda da árvore na AGM para identificar o caminho principal e eliminar os pontos que não contribuem significativamente para a forma geral do contorno. Essa poda é realizada com base em critérios de comprimento e relevância, garantindo que apenas os pontos mais significativos sejam mantidos na representação final.

Veja abaixo a implementação do método `prunning_tree`. Ele percorre recursivamente a árvore para calcular a altura dos nós e determinar o caminho de maior comprimento entre dois nós folha.

Algoritmo 2.1 – IMPLEMENTAÇÃO DO MÉTODO `prunning_tree`.

```
1 def prunning_tree(self, raiz):
2     # Se o nó atual não tem filhos, é uma folha
3     if len(raiz.filhos) == 0:
4         return 1, [raiz.idx], 1, [raiz.idx]
5     else:
6         # Para cada filho, executa a função recursivamente
```

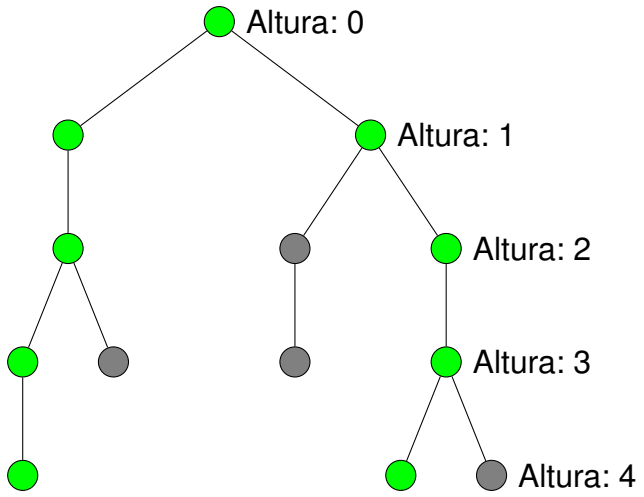
```
7           for filho in raiz.filhos:
8               filho.altura, filho.caminho_altura_maxima, filho.
                    tamanho_caminho_maximo, filho.caminho_maximo = self.
                    prunning_tree(filho)
9
10          if len(raiz.filhos) == 1:
11              # Verifica se o caminho que passa pelo nó atual é maior que
                    o maior caminho encontrado até o momento
12              if raiz.filhos[0].altura + 1 > raiz.filhos[0].
                    tamanho_caminho_maximo:
13                  tamanho_caminho_maximo = raiz.filhos[0].altura + 1
14                  caminho_maximo = [raiz.idx] + raiz.filhos[0].
                        caminho_altura_maxima
15              else:
16                  # Usa o maior caminho já encontrado no filho
17                  tamanho_caminho_maximo = raiz.filhos[0].
                        tamanho_caminho_maximo
18                  caminho_maximo = raiz.filhos[0].caminho_maximo
19          else:
20              # Ordena os filhos pela altura (do maior para o menor)
21              raiz.filhos.sort(key=lambda x: x.altura, reverse=True)
22
23              # Calcula o maior caminho passando por este nó (soma das
                    duas maiores alturas + o nó atual)
24              tamanho_caminho_maximo = raiz.filhos[0].altura + raiz.filhos
                    [1].altura + 1
25
26              # Monta o caminho correspondente
27              uma_parte_do_caminho = raiz.filhos[0].caminho_altura_maxima.
                    copy()
28              uma_parte_do_caminho.reverse()
29              caminho_maximo = uma_parte_do_caminho + [raiz.idx] + raiz.
                    filhos[1].caminho_altura_maxima
30
31              # Verifica se algum filho tem um caminho maior do que esse
32              for filho in raiz.filhos:
33                  if filho.tamanho_caminho_maximo > tamanho_caminho_maximo
                        :
34                      tamanho_caminho_maximo = filho.
                            tamanho_caminho_maximo
35                      caminho_maximo = filho.caminho_maximo
36
37          # Pega o filho com maior altura (o primeiro da lista, já
                ordenada)
38          no_altura_maxima = raiz.filhos[0]
39
40          return no_altura_maxima.altura + 1, [raiz.idx] +
```

```
no_altura_maxima.caminho_altura_maxima,
tamanho_caminho_maximo, caminho_maximo
```

O funcionamento do algoritmo pode ser resumido nos seguintes passos:

1. **Cálculo da Altura da Árvore** medindo a profundidade máxima a partir da raiz até as folhas.
2. **Cálculo do Maior Caminho** dentro dos nós da árvore.
3. O maior desses dois valores é considerado o **caminho principal**, capturando a estrutura mais relevante e eliminando nós de menor importância.

FIGURA 9 – PODA DA AGM.



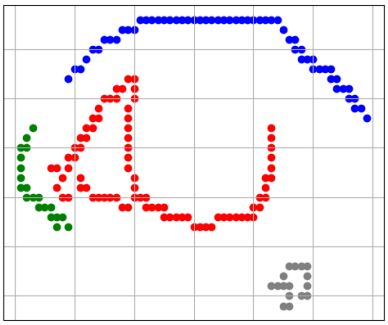
LEGENDA: Em verde: nós que fazem parte do maior caminho. Em cinza: nós que não fazem parte do maior caminho.

Pela FIGURA 9 podemos observar como funciona o algoritmo que realiza a poda da AGM, mantendo apenas os nós que fazem parte do maior caminho. Esta etapa é essencial para simplificar a representação dos contornos e facilitar a modelagem das *splines*.

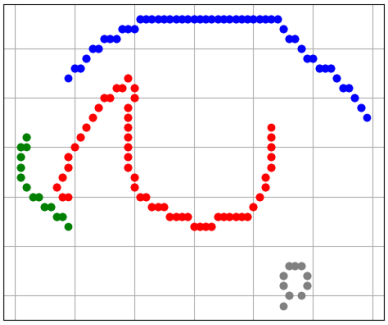
2.3.6 Resultado da AGM

A FIGURA 10 mostra os pontos removidos que não fazem parte do maior caminho e logo após isso, aplicamos mais um algoritmo autoral que elimina pontos de forma aleatória, mantendo apenas os pontos iniciais e finais de cada segmento.

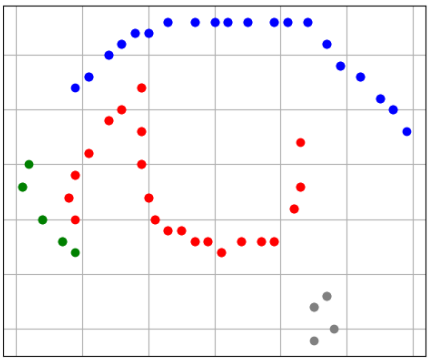
FIGURA 10 – FILTRAGEM DE PONTOS PELA AGM.



Maiores componentes conexos.



Aplicação da poda da AGM.



Remoção de alguns pontos de forma aleatória.

3 *SPLINES* CÚBICAS

As *splines* são funções definidas por partes que permitem a construção de curvas suaves a partir de um conjunto de pontos de controle. Elas são amplamente utilizadas em contextos onde é necessário representar formas complexas de maneira precisa e contínua, como na computação gráfica, modelagem geométrica e análise de dados. Dentre os diversos tipos, as *splines* cúbicas se destacam por proporcionarem um bom equilíbrio entre flexibilidade e suavidade, garantindo continuidade até a segunda derivada e, consequentemente, uma transição harmoniosa entre os segmentos da curva.

Na área de biometria facial, a representação precisa das características do rosto – como contornos dos olhos, nariz e lábios – é essencial para o desempenho de sistemas de reconhecimento. Utilizar *splines* para descrever essas formas permite capturar variações sutis da geometria facial com suavidade e fidelidade, o que é especialmente importante na comparação entre diferentes imagens. Além disso, a estrutura matemática das *splines* facilita tanto a manipulação quanto a análise das curvas faciais, tornando-as uma ferramenta eficiente e robusta nesse tipo de aplicação.

Neste capítulo, serão abordadas as *splines* cúbicas, suas propriedades e aplicações. A sua construção envolve a definição de um conjunto de pontos de controle e a determinação dos coeficientes que definem os polinômios cúbicos entre esses pontos. A suavidade da curva é garantida pela imposição de condições de continuidade e diferenciabilidade, resultando em uma representação suave e flexível.

3.1 DEFINIÇÃO DE *SPLINES* CÚBICAS

Uma *spline* cúbica é uma função definida por partes, onde cada parte é um polinômio cúbico. Seguimos então para os polinômios de grau 3, que são especificados na forma de Hermite:

$$p(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3, \tag{3.1}$$

onde cada c_i é um vetor no $\mathbb{R}^2$.

A função $p(t)$ representa uma curva parametrizada com variável independente $t \in [0, 1]$. Essa função descreve a posição ao longo da curva no plano bidimensional, sendo essa posição o elemento de interesse ao desenharmos a curva, já que o processo de plotagem envolve a determinação de pontos com coordenadas (x, y). As curvas de Hermite são definidas por equações paramétricas, ou seja, ambas as componentes x e y são expressas como funções da variável t.

Expressar uma curva cúbica na forma de Hermite consiste em fornecer quatro valores fundamentais – conhecidos como valores de Hermite – que contêm todas as informações necessárias para descrever completamente a curva. Esses valores também permitem calcular os coeficientes da variável t na equação polinomial cúbica padrão, resultando assim na expressão completa da curva como um polinômio.

A principal motivação para utilizar a forma de Hermite está na possibilidade de determinar os coeficientes do polinômio diretamente a partir de informações conhecidas: a posição do ponto inicial da curva, $p_0 = p(0)$, que corresponde ao valor da curva no instante inicial; a posição do ponto final, $p_1 = p(1)$, que representa o valor da curva no instante final; além das tangentes nesses pontos, ou seja, a taxa de variação da curva em p_0, denotada por $v_0 = v(0)$, e em p_1, denotada por $v_1 = v(1)$.

Agora que sabemos como se dão os valores de Hermite, podemos resolvê-los através do seguinte sistema de equações:

$$p(0) = c_0 + c_1(0) + c_2(0)^2 + c_3(0)^3 = c_0 \tag{3.2}$$
$$p(1) = c_0 + c_1 + c_2 + c_3 \tag{3.3}$$

Para encontrar v_0 e v_1 é necessário calcular a derivada da posição. Portanto:

$$v(t) = p'(t) = c_1 + 2c_2t + 3c_3t^2 \tag{3.4}$$
$$v(0) = c_1 \tag{3.5}$$
$$v(1) = c_1 + 2c_2 + 3c_3 \tag{3.6}$$

Então é possível obter a fórmula:

$$c_0 = p_0 \tag{3.7}$$
$$c_1 = v_0 \tag{3.8}$$
$$c_2 = -3p_0 - 2v_0 - v_1 + 3p_1 \tag{3.9}$$
$$c_3 = 2p_0 + v_0 + v_1 - 2p_1 \tag{3.10}$$

Escrevendo de forma matricial, temos:

$$p(t) = \begin{pmatrix} 1 & t & t^2 & t^3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ v_0 \\ v_1 \end{pmatrix} \tag{3.11}$$

3.1.1 Criação da *Spline*

Supondo que existam 4 pontos e é desejado criar uma *spline* que passe por todos eles, é necessário criar 3 segmentos de curvas cúbicas, cada uma passando por dois pontos consecutivos. Assim, a *spline* será composta por 3 segmentos cúbicos, cada um definido por 4 pontos de controle (Catmull; Rom, 1974).

A primeira curva irá de p_0 a p_1, a segunda de p_1 a p_2 e a final de p_2 a p_3. Então aplicaremos a fórmula que derivamos acima para p_t, mas em 3 variantes diferentes, cada variante com um par diferente de pontos iniciais e finais. A fórmula da primeira curva terá pontos iniciais e finais p_0 e p_1, a segunda em p_1 e p_2 e a terceira em p_2 e p_3, como é possível ver na FIGURA 11.

No que diz respeito às tangentes, Edwin Catmull e Raphael Rom propuseram uma abordagem baseada na diferença entre pontos adjacentes: a tangente em um ponto pode ser obtida subtraindo-se o ponto anterior do ponto seguinte. Por exemplo, a tangente no ponto p_2 é dada por $(p_3 - p_1)$. É comum, ainda, dividir esse resultado por 2, pois essa operação tende a produzir curvas visualmente mais suaves e esteticamente agradáveis.

Logo é possível reescrever a equação em termos de 4 pontos: p_0 a p_3.

Outra complicação que surge é que, ao utilizarmos o ponto anterior e o ponto seguinte para o cálculo das tangentes, surge a dúvida sobre qual deve ser considerado o ponto anterior a p_i e o ponto seguinte a p_f. Uma solução comum para esse problema é a adição de *pontos fantasmas*, que não fazem parte da curva, mas são utilizados apenas para o cálculo das tangentes.

Então ficamos com:

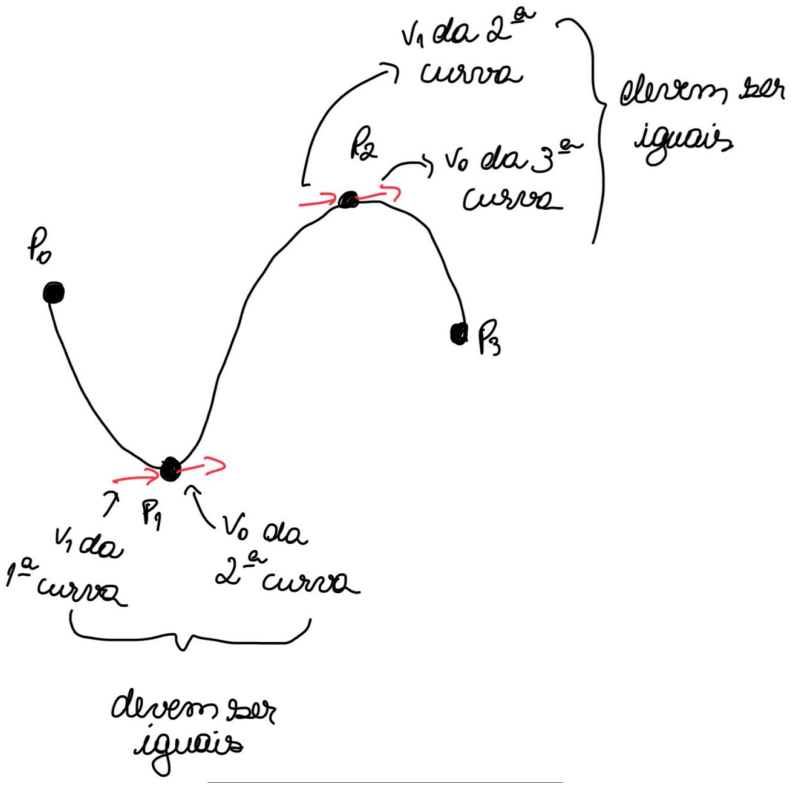
$$p(t) = \begin{pmatrix} 1 & t & t^2 & t^3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ \frac{p_2 - p_0}{2\tau} \\ \frac{p_3 - p_1}{2\tau} \end{pmatrix}, \tag{3.12}$$

onde τ é um fator de escala que pode ser ajustado para controlar a suavidade da curva. O valor de τ pode ser definido com base na distância entre os pontos de controle ou em outras considerações geométricas.

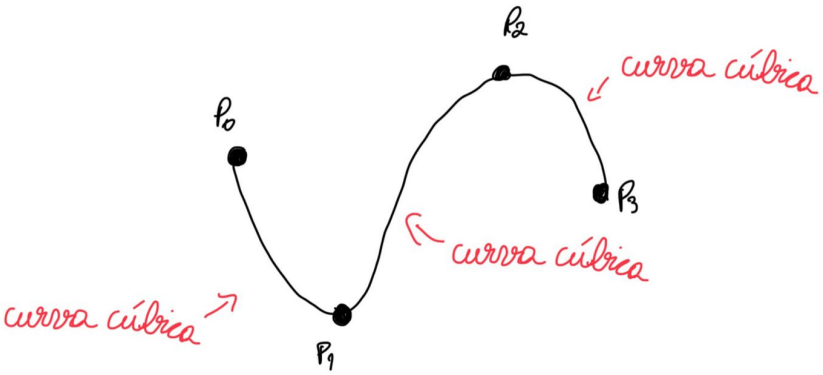
Com isso é possível chegar no formato de Catmull-Rom:

$$p(t) = \frac{1}{2} \begin{pmatrix} 1 & t & t^2 & t^3 \end{pmatrix} \begin{pmatrix} 0 & 2 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau - 6 & -2(\tau - 3) & -\tau \\ -\tau & 4 - \tau & \tau - 4 & \tau \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix} \tag{3.13}$$

FIGURA 11 – CURVAS CÚBICAS DE CATMULL-ROM.



Spline cúbica com 4 pontos de controle



Tangentes coincidentes nas conexões das cúbicas em cada ponto.

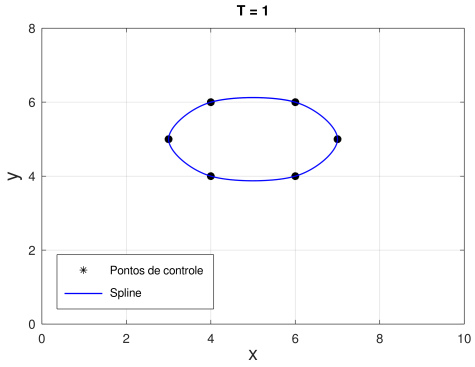
Na FIGURA 12 é possível observar que quanto menor a tensão τ, menos suave fica a curva:

3.1.2 Resultados

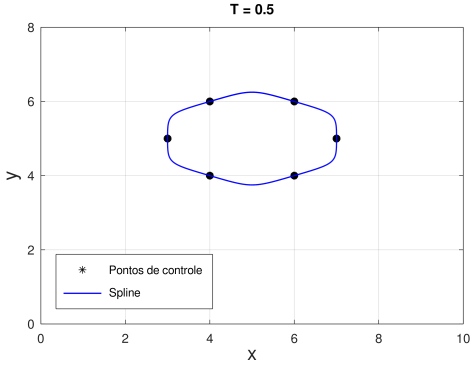
Foi implementado um algoritmo autoral que gera uma *spline* cúbica a partir de um conjunto de pontos de controle. O algoritmo utiliza a fórmula de Catmull-Rom para calcular os coeficientes da curva, garantindo suavidade e continuidade.

A FIGURA 13 apresenta os resultados obtidos a partir de conjuntos de pontos

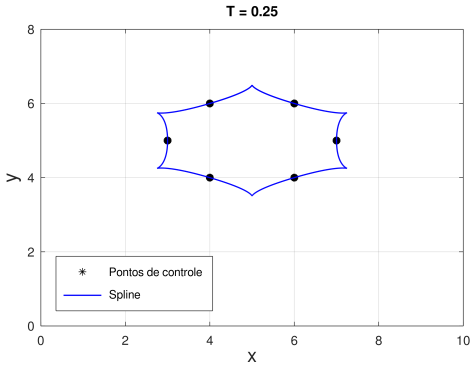
FIGURA 12 – TENSÃO DA CURVA.



$\tau = 1$



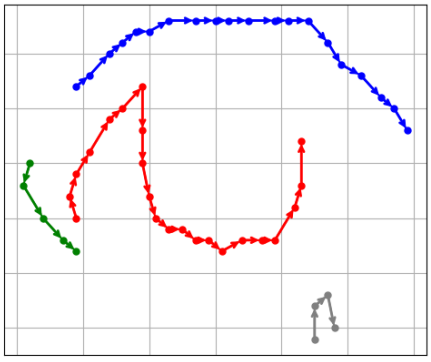
$\tau = 0.5$



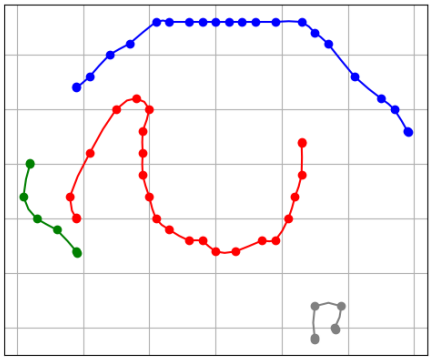
$\tau = 0.25$

de controle extraídos da imagem, conforme o processo descrito no Capítulo 2. É importante ressaltar que os pontos foram reordenados de modo a garantir que a curva resultante seja traçada sempre da esquerda para a direita, o que é essencial para o bom desempenho do algoritmo de DTW.

FIGURA 13 – *SPLINES* NOS PONTOS EXTRAÍDOS.



Reordenação dos pontos.



Splines.

4 *DYNAMIC TIME WARPING*

O *Dynamic Time Warping* (DTW) (Tavenard, 2021) é um algoritmo amplamente utilizado para medir a similaridade entre duas sequências temporais que podem variar em velocidade, tempo ou comprimento. Sua principal aplicação está em áreas como reconhecimento de fala, análise de séries temporais, bioinformática e visão computacional. Neste trabalho, essa técnica é utilizada para comparação entre as curvas que representam as características faciais dos indivíduos, permitindo identificar as mesmas pessoas em diferentes momentos ou condições.

4.1 INTRODUÇÃO

Este algoritmo surgiu como uma solução para o problema de comparação entre sequências temporais que apresentam distorções no eixo do tempo. Diferente da distância Euclidiana, que compara ponto a ponto, o DTW permite alinhar dinamicamente duas sequências, encontrando o menor custo de distorção temporal necessário para que elas se assemelhem.

Considere duas sequências temporais:

$$X = (x_1, x_2, \ldots, x_n) \quad \text{e} \quad Y = (y_1, y_2, \ldots, y_m), \tag{4.1}$$

onde X e Y são as sequências a serem comparadas, com n e m sendo seus respectivos comprimentos.

O algoritmo de DTW constrói uma matriz de custo acumulado D de dimensão $n \times m$, na qual cada elemento $D(i, j)$ representa o custo mínimo acumulado para alinhar os primeiros i elementos de X com os primeiros j elementos de Y.

O alinhamento entre os primeiros elementos das sequências ocorre diretamente na célula $D(1, 1)$, definida como:

$$D(1, 1) = d(x_1, y_1), \tag{4.2}$$

onde $d(x_1, y_1)$ é a distância entre os primeiros elementos de cada sequência, usualmente calculada por uma métrica como a distância Euclidiana.

Em seguida, são preenchidas a primeira linha e a primeira coluna da matriz. Nessas bordas, o caminho de alinhamento é único (ou apenas vertical ou apenas horizontal), o que resulta em uma soma cumulativa de distâncias. Assim, a inicialização segue as seguintes fórmulas:

$$D(i,1) = d(x_i, y_1) + D(i-1,1), \quad \text{para } i = 2,\ldots,n, \tag{4.3}$$

$$D(1,j) = d(x_1, y_j) + D(1,j-1), \quad \text{para } j = 2,\ldots,m. \tag{4.4}$$

Essa etapa garante que todas as possíveis trajetórias de alinhamento que passam pelas extremidades da matriz possam ser consideradas no cálculo recursivo subsequente.

Preenchimento recursivo da matriz

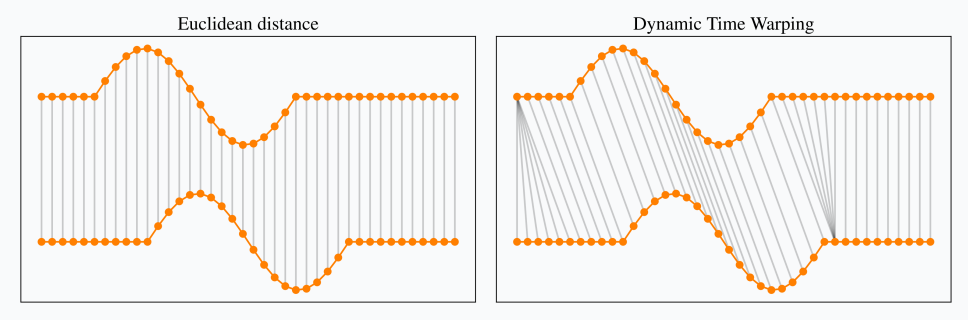
A partir de $i = 2$ e $j = 2$, a matriz é preenchida utilizando a fórmula recursiva do DTW:

$$D(i,j) = d(x_i, y_j) + \min \begin{cases} D(i-1,j), \\ D(i,j-1), \\ D(i-1,j-1), \end{cases} \quad \text{para } i \geq 2 \text{ e } j \geq 2. \tag{4.5}$$

Essa relação computa o custo acumulado de alinhar x_i com y_j. O valor final $D(n,m)$ representa o custo total do alinhamento ótimo entre as duas sequências temporais.

A FIGURA 14 ilustra o conceito de alinhamento dinâmico, mostrando como o DTW pode distorcer o eixo do tempo para alinhar duas sequências temporais que, à primeira vista, parecem diferentes.

FIGURA 14 – ALINHAMENTO DINÂMICO.



FONTE: (Tavenard, 2021)

4.2 VANTAGENS E DESVANTAGENS

O método apresenta diversas vantagens em relação a outras métricas de distância, como a distância Euclidiana. Entre elas, destacam-se:

- **Robustez a variações de tempo**: É capaz de lidar com sequências que apresentam variações na velocidade ou no comprimento, permitindo comparações mais precisas.
- **Alinhamento dinâmico**: Encontra o melhor alinhamento entre as sequências, minimizando o custo total de distorção temporal.
- **Versatilidade**: Funciona bem em sinais unidimensionais e multidimensionais, sendo aplicável em diversas áreas como reconhecimento de padrões e análise de séries temporais.

No entanto, o DTW também possui desvantagens:

- **Complexidade computacional**: O algoritmo tem complexidade $O(n \cdot m)$, o que pode ser um limitante para sequências muito longas.
- **Sensibilidade a ruídos**: Pode ser afetado por ruídos nas sequências, o que pode levar a alinhamentos incorretos.

4.3 JUSTIFICATIVA PARA O SEU USO

Neste trabalho, o DTW foi utilizado por ser uma técnica adequada para comparar sequências que representam características espaciais extraídas de contornos faciais utilizando um algoritmo simples. As curvas resultantes dessas características podem variar em comprimento e forma devido a diferentes condições de iluminação, expressões faciais ou ângulos de captura. Esse método permite alinhar essas curvas de forma dinâmica, possibilitando uma comparação mais precisa entre elas.

Além disso, este trabalho baseia-se na abordagem proposta por Levada *et al.* (2008), que demonstraram a eficácia do uso do algoritmo DTW na comparação da variação das cores dos *pixels* ao longo do vetor. Os autores obtiveram acurácias de 100%, 94% e 70% em diferentes cenários experimentais, evidenciando a robustez do método. Observa-se, contudo, uma diminuição progressiva na acurácia à medida que ruídos são introduzidos nas imagens, o que reforça a importância de técnicas complementares de pré-processamento.

Entretanto, uma limitação relevante observada nesse estudo foi o alto custo computacional do DTW, o que compromete sua aplicabilidade em cenários que exigem processamento em tempo real. Embora o tempo exato de execução dos experimentos não tenha sido reportado pelos autores, implementamos um *script* com base na proposta do artigo, cujo tempo de execução foi de aproximadamente 10 horas para comparar duas imagens representadas por vetores de dimensão 77.284 (correspondentes às intensidades de cinza).

Foi utilizada a biblioteca `dtaidistance` (Meert *et al.*, 2020) para a implementação do DTW para reproduzir os resultados do artigo, pois os pixels das imagens são representados como vetores unidimensionais. Porém para o nosso trabalho foi preciso adaptar o código para lidar com coordenadas, o que exigiu uma abordagem diferente, visto que as curvas não são vetores unidimensionais, mas sim representações contínuas de pontos no espaço.

5 RESULTADOS

Para a avaliação dos métodos propostos neste trabalho, foi utilizada uma amostra do banco de dados FERET (Phillips *et al.*, 1998; Phillips *et al.*, 2000). A seleção compreende um total de 24 imagens faciais pertencentes a quatro indivíduos distintos. Ressalta-se que há um desbalanceamento na distribuição de imagens por indivíduo, sendo 10 imagens do primeiro sujeito, 3 do segundo, 6 do terceiro e 5 do quarto. Todas as imagens foram capturadas em condições controladas, com orientação frontal e sem variações significativas de ângulo ou expressões faciais.

Foram segmentadas quatro regiões distintas: olhos, nariz e boca. Cada uma é representada por um *array* bidimensional que armazena as coordenadas dos pontos gerados pela aplicação das *splines* em cada região.

O DTW foi aplicado separadamente para cada uma dessas quatro representações, permitindo avaliar a similaridade local (por região) e utilizando o sistema de votação para determinar a identidade do indivíduo. Além disso, uma outra versão com todas as características concatenadas foi utilizada como referência global da face.

É importante ressaltar que devido a uma aleatoriedade introduzida de propósito no algoritmo para retirar alguns pontos (subseção 2.3.6), os resultados podem variar entre as execuções. Portanto, para garantir a consistência dos resultados, foi utilizada a semente `random.seed(42)` em todas as execuções, assegurando que os mesmos pontos fossem selecionados em cada teste.

A máquina utilizada para os testes possui as seguintes especificações: processador Intel Core i7-8550U, 16 GB de memória RAM, 4 núcleos e 8 processadores lógicos.

5.1 PARÂMETROS DE CONFIGURAÇÃO

Para a realização dos experimentos, foram definidos alguns parâmetros de configuração que influenciam diretamente o desempenho do DTW e a qualidade das curvas geradas pelas *splines*. Esses parâmetros incluem:

- **Passo da *Spline*:** Define a densidade dos pontos ao longo da curva. Um passo menor resulta em curvas mais detalhadas, mas com maior custo computacional.
- **Tensão:** Controla a suavidade das curvas geradas pelas *splines*. Valores mais altos resultam em curvas mais flexíveis, enquanto valores mais baixos produzem curvas mais rígidas.

- **Translação:** Indica se as curvas devem ser transladadas para alinhar melhor as características faciais.

Para cada combinação desses parâmetros, foram realizados testes com as características faciais concatenadas e segmentadas, permitindo uma análise comparativa dos resultados obtidos.

5.2 RESULTADOS OBTIDOS

As tabelas 2 e 3 apresentam os resultados obtidos para cada uma das abordagens, considerando diferentes parâmetros de configuração.

TABELA 2 – Testes realizados para as características concatenadas.

Experimento	Passo *Spline*	Tensão	Translação	Acurácia (%)	Tempo
1	0.5	1	Não	50	4min 21s
2	0.5	0.5	Não	54.16	4min 05s
3	0.5	0.25	Não	50	4min 13s
4	1	1	Não	50	2min 02s
5	1	0.5	Não	54.16	1min 48s
6	1	0.25	Não	45.83	2min 00s
7	0.5	1	Sim	16.66	4min 33s
8	0.5	0.5	Sim	41.66	3min 57s
9	0.5	0.25	Sim	37.5	5min 05s
10	1	1	Sim	33.33	1min 58s
11	1	0.5	Sim	33.33	2min 10s
12	1	0.25	Sim	33.33	2min 04s

TABELA 3 – Testes realizados para as características segmentadas.

Experimento	Passo *Spline*	Tensão	Translação	Acurácia (%)	Tempo
13	0.5	1	Não	41.66	2min 12s
14	0.5	0.5	Não	41.66	2min 39s
15	0.5	0.25	Não	41.66	2min 21s
16	1	1	Não	45.83	1min 07s
17	1	0.5	Não	37.5	1min 08s
18	1	0.25	Não	41.66	1min 08s
19	-	-	Não	37.5	0min 21s
20	0.5	1	Sim	79.16	2min 43s
21	0.5	0.5	Sim	75	2min 39s
22	0.5	0.25	Sim	70.83	2min 27s
23	1	1	Sim	75	1min 04s
24	1	0.5	Sim	79.16	1min 09s
25	1	0.25	Sim	79.16	1min 13s
26	-	-	Sim	70.83	0min 17s

5.2.1 Características Concatenadas

Na Tabela 2, observa-se que a introdução de translação causou uma queda significativa na acurácia. Enquanto os experimentos sem translação atingiram até 54,16% de acurácia (Experimentos 2 e 5), aqueles com translação tiveram desempenho inferior, com valores que variaram entre 16,66% e 41,66%. Isso sugere que, ao utilizar características concatenadas, a translação dificulta a extração de padrões discriminativos.

Além disso, ao comparar os valores de tensão, não há uma tendência clara de melhora ou piora na acurácia. No entanto, o passo da *spline* igual a 1 reduziu consideravelmente o tempo de execução, com destaque para o Experimento 5 (1min 48s), mantendo a mesma acurácia de 54,16% que o Experimento 2 (4min 05s), cujo passo era 0.5. Isso indica que um passo maior pode ser preferível por reduzir o tempo computacional sem comprometer o desempenho.

5.2.2 Características Segmentadas

Na Tabela 3, o cenário se inverte: os melhores resultados foram obtidos com a presença de translação. Os experimentos 20, 24 e 25 atingiram 79,16% de acurácia, superando significativamente os demais. Isso indica que, no caso das características segmentadas, a translação introduz variações benéficas ao aprendizado, provavelmente por aumentar a robustez das representações faciais.

Entre os experimentos sem translação, a acurácia ficou abaixo de 46%, indicando que a segmentação sozinha não é suficiente para gerar boas representações sem esse tipo de transformação. Já o tempo de execução foi consistentemente mais baixo nas segmentadas em comparação às concatenadas, sugerindo uma vantagem adicional em termos de eficiência computacional.

Os experimentos 19 e 26 utilizaram apenas os pontos extraídos automaticamente, sem aplicação de *splines*. Esses testes funcionam como uma linha de base para comparação com as demais abordagens. Observa-se que, ao comparar o Experimento 19 com o Experimento 26, a acurácia salta de 37,5% para 70,83%, evidenciando o impacto positivo da translação mesmo quando se utiliza uma representação simples baseada apenas nos pontos faciais. Isso reforça a relevância da translação como uma etapa de pré-processamento importante, especialmente no contexto das características segmentadas.

5.2.3 Considerações Finais

De forma geral:

- A translação impacta negativamente nas características concatenadas, mas melhora significativamente os resultados nas características segmentadas.
- Um passo da *spline* maior (1) reduz o tempo computacional sem grandes perdas de desempenho.
- As características segmentadas com translação se mostraram a melhor combinação, atingindo a maior acurácia com tempos de execução relativamente baixos.

Esses resultados indicam que a segmentação, aliada a transformações geométricas simples como a translação, pode ser uma estratégia promissora para a extração de características mais robustas em sistemas biométricos baseados em curvas *spline*.

6 CONCLUSÃO

O objetivo central deste trabalho foi investigar a viabilidade do uso de *splines* cúbicas como método de representação de características faciais em sistemas biométricos, buscando unir a expressividade matemática das curvas suaves com a necessidade de precisão e eficiência nos processos de identificação de indivíduos. Diferentemente de abordagens tradicionais baseadas em *pixels*, essa proposta parte da premissa de que a forma das feições faciais contém informações ricas e discriminativas que podem ser representadas de forma mais compacta, interpretável e flexível através de curvas. A metodologia também buscou avaliar o comportamento dessa representação ao ser combinada com uma técnica de comparação temporal não-linear, o *Dynamic Time Warping*, para verificação da similaridade entre diferentes amostras faciais.

Para que essa abordagem fosse possível, foi necessário desenvolver uma etapa robusta de extração automática de pontos. A *pipeline* adotada iniciou-se com a detecção de rostos utilizando classificadores Haar Cascade, seguida pela aplicação do algoritmo de Canny para realçar contornos faciais relevantes, como os olhos, nariz e boca. Esses contornos, ainda com excesso de pontos, foram então refinados por meio de técnicas de processamento baseadas em grafos: utilizando componentes conexas, árvores geradoras mínimas e poda de nós, obteve-se uma versão reduzida e estruturada dos pontos. Essa etapa foi fundamental para garantir que a modelagem por *splines* posteriormente ocorresse de forma eficiente e significativa, preservando os principais traços geométricos do rosto.

A modelagem com *splines* cúbicas foi escolhida pela sua capacidade de representar curvas suaves que passam por um conjunto de pontos com continuidade e derivadas bem definidas. As *splines* atuaram como intermediárias entre a imagem e a abstração matemática das formas faciais, convertendo conjuntos discretos de pontos em curvas contínuas que descrevem com precisão a geometria das feições. Além disso, a representação por *splines* facilita tanto a visualização quanto a análise matemática, o que a torna vantajosa frente a representações puramente estatísticas ou baseadas em aprendizado de máquina de “caixa-preta".

Para comparar diferentes curvas, foi utilizado o algoritmo DTW, uma técnica que mede a similaridade entre sequências que podem variar em comprimento ou sofrer pequenas distorções espaciais. O método foi aplicado diretamente sobre os pontos das *splines*, buscando identificar o alinhamento ótimo entre duas formas faciais distintas. Essa abordagem permitiu lidar com variações naturais entre amostras de um mesmo indivíduo – como leve inclinação do rosto – sem comprometer a acurácia da comparação. Assim, o DTW mostrou-se especialmente útil em contextos onde as

formas das *splines* preservam a identidade, mas podem diferir ligeiramente em suas parametrizações.

Os resultados experimentais demonstraram que a proposta é viável e promissora. A representação com *splines* cúbicas possibilitou capturar de forma eficiente os principais contornos das feições faciais, mesmo após a redução dos pontos extraídos. Além disso, o uso do DTW como medida de similaridade entre curvas permitiu diferenciar indivíduos com boa acurácia, mesmo em um cenário com variações naturais entre capturas faciais. Embora o trabalho tenha utilizado uma base de dados limitada, os testes iniciais indicam que a combinação entre representação geométrica e alinhamento temporal pode oferecer uma solução eficiente e interpretável para aplicações biométricas.

Além dos avanços apresentados, há diversas possibilidades de aprimoramento que podem ser exploradas em trabalhos futuros. Etapas de pré-processamento adicionais, como a rotação e o alinhamento dos rostos com base em pontos de referência (por exemplo, olhos e boca), são práticas comuns na literatura de reconhecimento facial e podem contribuir para maior robustez em bases mais desafiadoras – embora neste trabalho isso não tenha sido necessário, dada a qualidade e o alinhamento já presentes nas imagens utilizadas. Outro caminho promissor seria testar a abordagem em cenários mais realistas, com expressões faciais variadas, como caretas, sorrisos ou estados emocionais distintos, para avaliar a capacidade das *splines* em representar feições mais dinâmicas. Também é válido considerar uma análise mais aprofundada sobre os hiperparâmetros da etapa de detecção de bordas, como os limiares do algoritmo de Canny, e seu impacto na acurácia final do sistema. Por fim, etapas de pós-processamento, como a adição controlada de ruídos ou perturbações nas imagens, poderiam ser utilizadas para testar a resiliência do modelo frente a distorções comuns em aplicações práticas, contribuindo para uma avaliação mais abrangente da robustez da abordagem proposta.

REFERÊNCIAS

CANNY, J. A Computational Approach to Edge Detection. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, PAMI-8, n. 6, p. 679–698, 1986. DOI: 10.1109/TPAMI.1986.4767851. Citado 3 vezes nas páginas 8, 10, 13.

CATMULL, E.; ROM, R. A CLASS OF LOCAL INTERPOLATING SPLINES. *In:* BARNHILL, R. E.; RIESENFELD, R. F. (ed.). **Computer Aided Geometric Design**. [*S. l.*]: Academic Press, 1974. p. 317–326. ISBN 978-0-12-079050-0. Disponível em: https://doi.org/10.1016/B978-0-12-079050-0.50020-5. Citado 1 vez na página 26.

LEVADA, A. L. M.; CORREA, D. C.; SALVADEO, D. H. P.; SAITO, J. H.; MASCARENHAS, N. D. A. Novel approaches for face recognition: Template-matching using Dynamic Time Warping and LSTM neural network supervised classification. *In:* 2008 15th International Conference on Systems, Signals and Image Processing. [*S. l.: s. n.*], 2008. p. 241–244. DOI: 10.1109/IWSSIP.2008.4604412. Citado 2 vezes nas páginas 9, 31.

LOGRADO, M. L. [*S. l.: s. n.*], 2025. https://github.com/MariaLuizaLogrado/spline-dtw-biometric-id-code. Citado 1 vez na página 6.

MAGGINI, M.; MELACCI, S.; SARTI, L. Representation of Facial Features by Catmull-Rom Splines. *In:* p. 408–415. ISBN 978-3-540-74271-5. DOI: 10.1007/978-3-540-74272-2_51. Citado 1 vez na página 9.

MEERT, W.; HENDRICKX, K.; VAN CRAENENDONCK, T.; ROBBERECHTS, P.; BLOCKEEL, H.; DAVIS, J. **DTAIDistance**. [*S. l.: s. n.*], ago. 2020. DOI: 10.5281/zenodo.3981067. Disponível em: https://github.com/wannesm/dtaidistance. Citado 1 vez na página 32.

OPENCV. **Canny Edge Detection**. Disponível em: https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html. Citado 3 vezes nas páginas 8, 10, 13, 15.

OPENCV. **FaceDetection. opencv/opencv Wiki**. Disponível em: https://github.com/opencv/opencv/wiki/FaceDetection. Citado 2 vezes nas páginas 8, 11.

PHILLIPS, P.; MOON, H.; RIZVI, S.; RAUSS, P. The FERET evaluation methodology for face-recognition algorithms. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 22, n. 10, p. 1090–1104, 2000. DOI: 10.1109/34.879790. Citado 2 vezes nas páginas 7, 14, 33.

PHILLIPS, P.; WECHSLER, H.; HUANG, J.; RAUSS, P. J. The FERET database and evaluation procedure for face-recognition algorithms. **Image and Vision Computing**, v. 16, n. 5, p. 295–306, 1998. ISSN 0262-8856. DOI: https://doi.org/10.1016/S0262-8856(97)00070-X. Disponível em: https://www.sciencedirect.com/science/article/pii/S026288569700070X. Citado 2 vezes nas páginas 7, 14, 33.

SAKOE, H.; CHIBA, S. Dynamic programming algorithm optimization for spoken word recognition. **IEEE Transactions on Acoustics, Speech, and Signal Processing**, v. 26, n. 1, p. 43–49, 1978. DOI: 10.1109/TASSP.1978.1163055. Citado 2 vezes nas páginas 6, 8.

TAVENARD, R. **An introduction to Dynamic Time Warping**. [*S. l.: s. n.*], 2021. https://rtavenar.github.io/blog/dtw.html. Citado 2 vezes nas páginas 8, 29, 30.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. *In:* PROCEEDINGS of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001. [*S. l.: s. n.*], 2001. v. 1. DOI: 10.1109/CVPR.2001.990517. Citado 3 vezes nas páginas 8, 10.

VIRTANEN, P.; GOMMERS, R.; OLIPHANT, T. E.; HABERLAND, M.; REDDY, T.; COURNAPEAU, D.; BUROVSKI, E.; PETERSON, P.; WECKESSER, W.; BRIGHT, J.; VAN DER WALT, S. J.; BRETT, M.; WILSON, J.; MILLMAN, K. J.; MAYOROV, N.; NELSON, A. R. J.; JONES, E.; KERN, R.; LARSON, E.; CAREY, C. J.; POLAT, İ.; FENG, Y.; MOORE, E. W.; VANDERPLAS, J.; LAXALDE, D.; PERKTOLD, J.; CIMRMAN, R.; HENRIKSEN, I.; QUINTERO, E. A.; HARRIS, C. R.; ARCHIBALD, A. M.; RIBEIRO, A. H.; PEDREGOSA, F.; VAN MULBREGT, P.; SCIPY 1.0 CONTRIBUTORS. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. **Nature Methods**, v. 17, p. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2. Citado 3 vezes nas páginas 8, 10, 19.