UNIVERSIDADE FEDERAL DO PARANÁ

MARIA LUIZA SAMPAIO LOGRADO

SPLINES PARA REPRESENTAÇÃO DE CARACTERÍSTICAS FACIAIS COM APLICAÇÃO EM BIOMETRIA

CURITIBA

2025

MARIA LUIZA SAMPAIO LOGRADO

SPLINES PARA REPRESENTAÇÃO DE CARACTERÍSTICAS FACIAIS COM APLICAÇÃO EM BIOMETRIA

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

O resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. (. . .) As palavras-chave devem figurar logo abaixo do resumo, antecedidas da expressão Palavras-chave:, separadas entre si por ponto e finalizadas também por ponto.Ter no máximo 500 palavras!!! As palavras chave são separadas por ponto e vírgula.

Palavras-chaves: latex; abntex; editoração de texto.

ABSTRACT

This is the english abstract.

Key-words: latex. abntex. text editoration.

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.

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.

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 Cascades (Viola; Jones, 2001; OPENCV. . . , s.d.[a]) 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.[b]). O objetivo é obter um conjunto inicial de pontos que descrevem as características faciais.

1.3.3 Redução de Pontos

Para evitar redundância e reduzir a complexidade computacional, os pontos extraídos nos contornos são submetidos a um processo de simplificação. Estruturas de dados como grafos e árvore geradora mínima são utilizadas para manter apenas os pontos mais significativos. Nesta etapa utilizamos o algoritmo que encontra a árvore geradora mínima através da biblioteca *Scipy* (Virtanen *et al.*, 2020).

1.3.4 Modelagem com *splines*

Com os pontos reduzidos, são traçadas curvas suaves utilizando *splines* Catmull-Rom (Catmull; Rom, 1974; Twigg, 2003; Maggini *et al.*, 2007). Essa modelagem permite representar as feições faciais de forma contínua e diferenciável, garantindo uma estrutura mais compacta e interpretável.

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, com as datas previstas para cada etapa.

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 para detectar o rosto da pessoa na imagem, concentrando a análise nessa área específica e facilitando a detecção das características menores (Viola; Jones, 2001).

Em seguida, utilizamos o método `Canny` do OpenCV (OpenCV, s.d.[b]) 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 (Canny, 1986).

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.[a]).

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. Essa imagem serve como entrada para o método e com ela em tons de cinza facilita a detecção de 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.

  Foi colocado 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. Este 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.

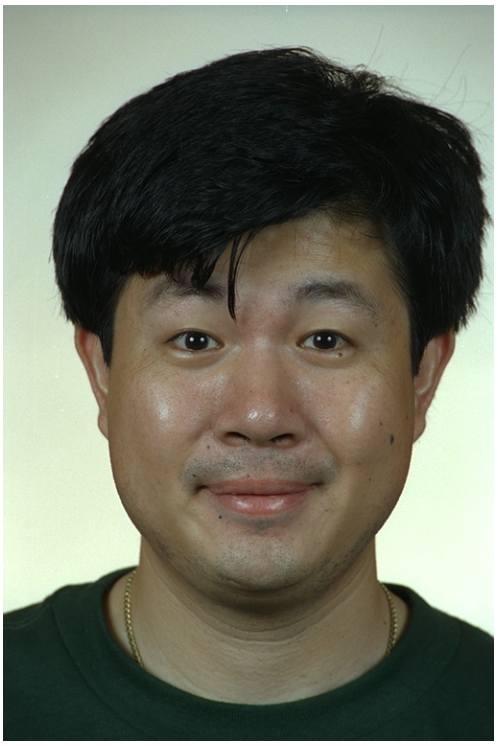
Essa abordagem, que realiza a detecção passo a passo, garante maior precisão na identificação das características menores, pois concentra a análise na área previamente delimitada pelo rosto.

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

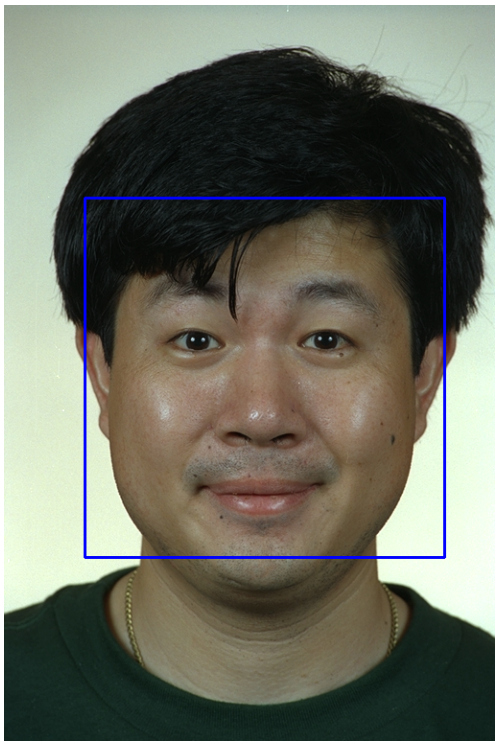
1. **Detecção do Rosto:** Para identificar e isolar o rosto na imagem usamos o classificador `haarcascade_frontalface_default.xml`.
2. **Detecção dos Olhos:** Aplicamos o classificador `haarcascade_eye.xml` para localizar os olhos dentro da área do rosto previamente detectada.
3. **Detecção do Nariz:** 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.

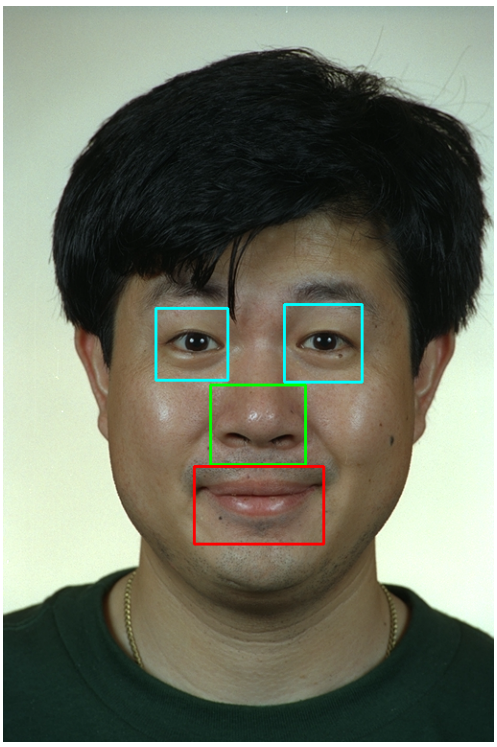
FIGURA 1 – DETECÇÕES.



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



1º Detecção da face.



2º Detecção das características.

2.2 EXTRAÇÃO DE CONTORNOS

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

2.2.1 Algoritmo de Canny

O algoritmo de detecção de bordas 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 bordas.
2. **Cálculo do Gradiente de Intensidade**: Os gradientes de intensidade são calculados nas direções horizontal G_x e vertical G_y usando operadores Sobel. A magnitude do gradiente é dada por:

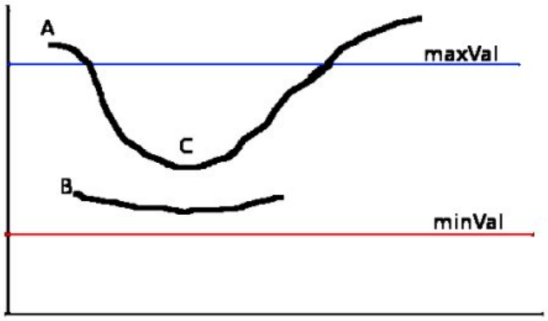
$$G_x = \frac{\partial I}{\partial x} \quad G_y = \frac{\partial I}{\partial y}$$

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

3. **Supressão Não Máxima**: Elimina *pixels* que não são máximos locais no gradiente, preservando apenas os que representam bordas fortes.
4. **Rastreamento de Bordas por Histerese**: Classifica as bordas detectadas usando dois limiares, *minVal* e *maxVal*:
    - Bordas com gradiente maior que *maxVal* são fortes.
    - Bordas com gradiente menor que *minVal* são descartadas.
    - Bordas entre *minVal* e *maxVal* são fracas e mantidas apenas se conectadas a bordas fortes.

Por exemplo, na Figura FIGURA 2, a borda A está acima do *maxVal*, sendo assim considerada uma *sure-edge*. Embora a borda C esteja abaixo do *maxVal*, ela está conectada à borda A, de modo que também é considerada como borda válida, resultando em uma curva completa. Já a borda B, embora esteja acima do *minVal*, não está conectada a nenhuma *sure-edge* e, portanto, é descartada.

FIGURA 2 – RASTREAMENTO DE CONTORNO POR HISTERESE



FONTE: (OpenCV, s.d.[b])
LEGENDA: Calculada a magnitude do gradiente ao longo de uma curva parametrizada.

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 as bordas são linhas longas e contínuas.

2.2.2 Resultado da Detecção de Bordas

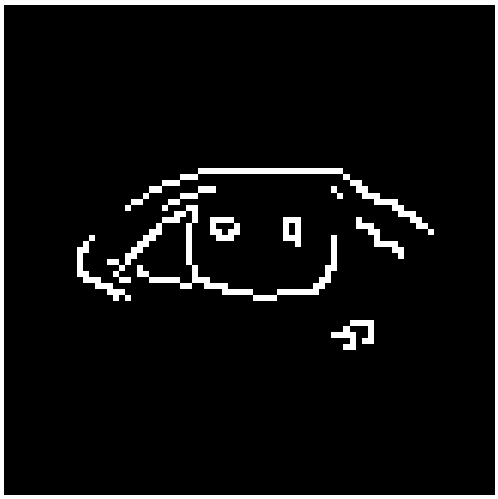
Ao aplicar o algoritmo de Canny para a detecção de bordas, obtemos um array binário onde os valores ’0’ e ’255’ representam diferentes tipos de pixels. Os pixels com valor ’0’ são considerados descartáveis, ou seja, não fazem parte das bordas detectadas. Em contraste, os pixels com valor ’255’ são os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

A seguir são apresentados os resultados da aplicação do algoritmo de Canny em imagens de exemplo. A FIGURA 3 mostra as características com as bordas detectadas.

A detecção de bordas de Canny é uma técnica poderosa e eficiente para identificar contornos em imagens. Esses resultados demonstram como o algoritmo de Canny é eficaz para realçar as bordas em diferentes tipos de imagens, permitindo uma análise visual clara e detalhada das características principais em cada exemplo.

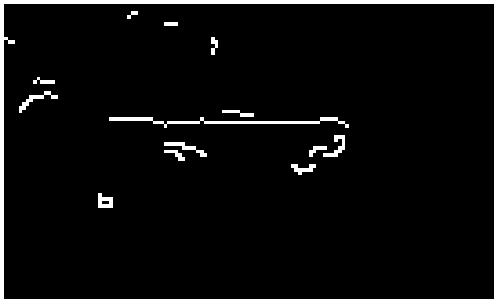
FIGURA 3 – APLICAÇÃO DO ALGORITMO DE CANNY.

Olho esquerdo.



Olho direito.

Nariz.



Boca.

2.3 REDUÇÃO DE PONTOS EM DETECÇÃO DE BORDAS

Embora o detector de bordas de Canny seja excelente para capturar as bordas, 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 dos pontos de borda, 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 quatro etapas principais:

1. **Construção do Grafo:** Os pontos de borda detectados são organizados em um grafo, onde cada ponto é um nó e as conexões entre eles representam as arestas. Essa estrutura facilita a análise e a manipulação dos pontos.
2. **Árvore Geradora Mínima:** A partir do grafo, uma árvore geradora mínima é construída. Essa árvore conecta todos os pontos de forma a minimizar o custo

total das arestas, resultando em uma representação simplificada dos contornos.

3. **Otimização com Poda da Árvore:** A árvore geradora mínima é otimizada, removendo pontos que não contribuem significativamente para a forma geral do contorno. Essa poda reduz o número de pontos, mantendo apenas os mais relevantes.

4. **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 borda detectados pelo algoritmo de Canny. Cada ponto é representado como um nó no grafo, e as conexões entre os nós são representadas como arestas. Essa estrutura é chamada de matriz de adjacência, onde cada elemento indica a presença ou ausência de uma aresta entre dois nós.

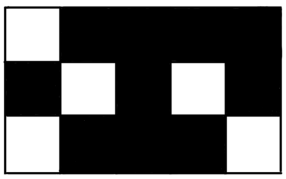
O código foi feito de forma autoral em Python, utilizando a fórmula Manhattan para calcular a distância entre os nós. Essa fórmula é adequada para o nosso caso, pois considera apenas as distâncias horizontais e verticais, o que é suficiente para que o grafo represente adequadamente a estrutura dos contornos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

No processo de detecção de contornos, a matriz Canny e a matriz de adjacências são conceitos fundamentais para entender a construção dos grafos. O primeiro passo é aplicar a matriz Canny, que resulta na detecção de contornos da imagem. A seguir, apresentamos uma representação da matriz Canny e da matriz de adjacências.

FIGURA 4 – MATRIZ CANNY E CONTORNOS.

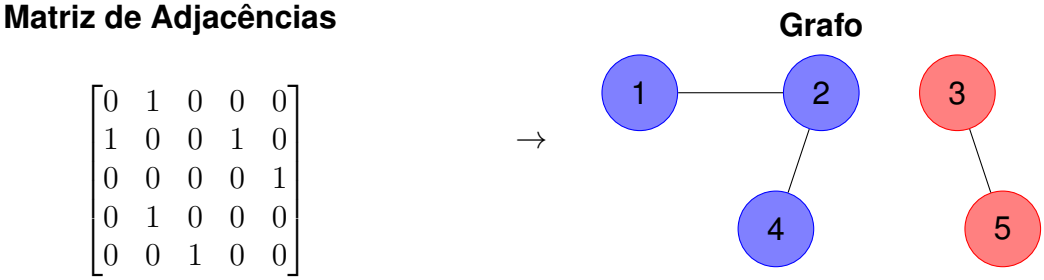
Matriz Canny **Contornos**

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



O próximo passo é a conversão dos contornos em grafos utilizando a matriz de adjacências.

FIGURA 5 – MATRIZ DE ADJACÊNCIAS E GRAFO.



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 todos os nós estão conectados entre si, 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 REPRESENTADAS POR GRAFOS.

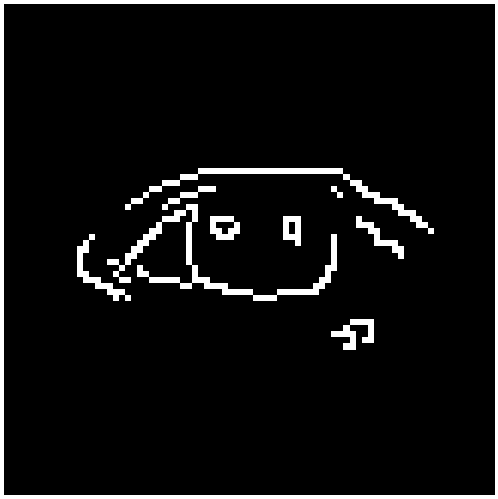
Em problemas de processamento de imagens, frequentemente são utilizados grafos que representam os pixels da imagem. O objetivo é identificar componentes conexos, permitindo a seleção dos caminhos mais significativos e a filtragem de ruídos. Componentes com um grande número de nós indicam regiões de interesse, correspondendo a áreas com maior quantidade de pixels conectados. Por outro lado, componentes menores podem ser interpretadas como ruído ou regiões irrelevantes, que podem ser descartadas durante o processamento.

2.3.3 Resultado da Construção do Grafo

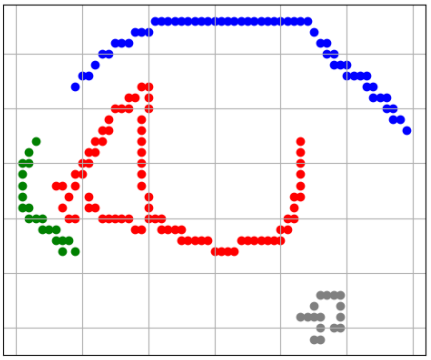
Podemos observar que a imagem original é composta por um grande número de pontos, porém utilizando a filtragem de pixels através de grafos com as componentes conexos, conseguimos reduzir a quantidade de pontos, mantendo apenas os mais relevantes. A FIGURA 7 mostra o resultado da construção do grafo e 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:

- 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.

- É um nó que não tem filhos, ou seja, ele está no fim de um caminho na árvore

Para a construção adequada das splines, é essencial definir uma ordem consistente para os pontos que descrevem as características faciais. Sem uma ordem definida, as splines podem ser traçadas de maneira incoerente, resultando em representações visuais incorretas ou deformadas. Além disso, a ordenação contribui para evitar ciclos ou conexões redundantes entre os pontos, o que poderia comprometer a suavidade e continuidade das curvas. Foi utilizada a biblioteca *Scipy* (Virtanen *et al.*, 2020) para encontrar a AGM.

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 distância 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.

Listing 2.1 – IMPLEMENTAÇÃO DO MÉTODO `prunning_tree`.

```
def prunning_tree(self, raiz):
    # Se o nó atual não tem filhos, é uma folha
    if len(raiz.filhos) == 0:
        return 1, [raiz.idx], 1, [raiz.idx]
    else:
        # Para cada filho, executa a função recursivamente
        for filho in raiz.filhos:
            filho.altura, filho.caminho_altura_maxima, filho.
                tamanho_caminho_maximo, filho.caminho_maximo = self.
                prunning_tree(filho)

        if len(raiz.filhos) == 1:
            # Verifica se o caminho que passa pelo nó atual é maior que
                o maior caminho encontrado até o momento
            if raiz.filhos[0].altura + 1 > raiz.filhos[0].
                tamanho_caminho_maximo:
                tamanho_caminho_maximo = raiz.filhos[0].altura + 1
                caminho_maximo = [raiz.idx] + raiz.filhos[0].
                    caminho_altura_maxima
            else:
                # Usa o maior caminho já encontrado no filho
                tamanho_caminho_maximo = raiz.filhos[0].
                    tamanho_caminho_maximo
                caminho_maximo = raiz.filhos[0].caminho_maximo
        else:
```

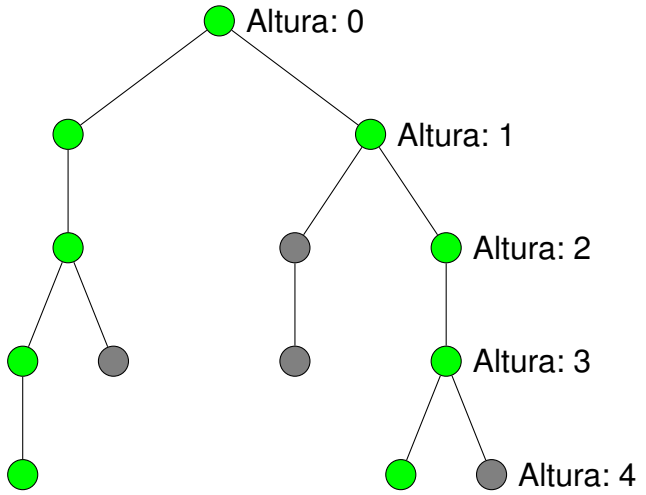
```
20              # Ordena os filhos pela altura (do maior para o menor)
21              raiz.filhos.sort(key=lambda x: x.altura, reverse=True)
22
23              # Calcula o maior caminho passando por este nó (soma das
                    duas maiores alturas + o nó atual)
24              tamanho_caminho_maximo = 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. **Calculando a Altura da Árvore** medindo a profundidade máxima a partir da raiz até as folhas.
2. **Calculando o 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.

Pela FIGURA 8 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*.

FIGURA 8 – PODA DA AGM.

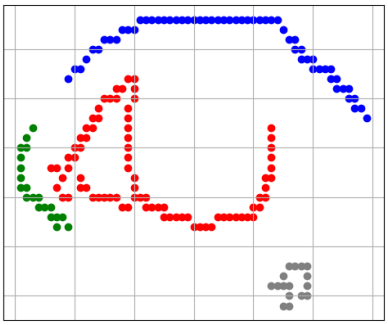


LEGENDA: Em verde: nós que fazem parte do maior caminho. Em cinza: nós que não fazem parte do maior caminho.

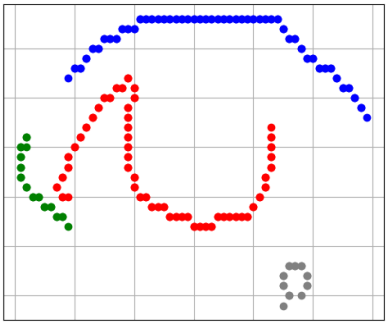
2.3.5 Resultado da AGM

A FIGURA 9 mostra o resultado da AGM, onde os pontos que não fazem parte do maior caminho foram removidos e logo após a AGM, aplicamos mais um algoritmo autoral que elimina pontos de forma aleatória, mantendo apenas os pontos iniciais e finais de cada segmento.

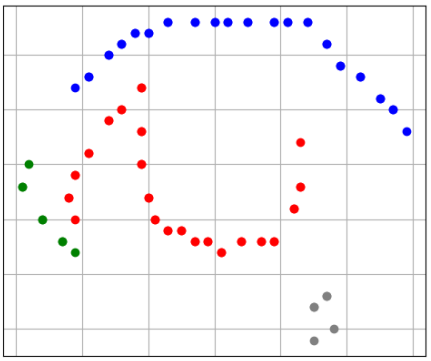
FIGURA 9 – FILTRAGEM DE PONTOS PELA AGM.



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, será abordado 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 derivabilidade, 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 representa uma curva parametrizada cuja variável independente é $t \in [0, 1]$. 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. Para ilustrar esse conceito, iniciamos definindo x em função de t:

$$x(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 \tag{3.2}$$

A função $y(t)$ será expressa de forma análoga à função $x(t)$, embora possivelmente com coeficientes diferentes. A variável t representa o parâmetro que indica o progresso ao longo da curva, normalizado no intervalo $[0, 1]$, sendo $t = 0$ correspondente ao início da curva e $t = 1$ ao seu final.

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 é a possibilidade de determinar os coeficientes do polinômio diretamente a partir das informações das posições dos pontos inicial e final da curva, bem como das tangentes nesses pontos:

$$p_0 = p(0), \tag{3.3}$$

posição do ponto inicial da curva.

$$p_1 = p(1), \tag{3.4}$$

posição do ponto final da curva.

$$v_0 = v(0), \tag{3.5}$$

a taxa de variação em p_0.

$$v_1 = v(1), \tag{3.6}$$

a taxa de variação em p_1.

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

$$p(0) = c_0 + c_1(0) + c_2(0)^2 + c_3(0)^3 = c_0 \tag{3.7}$$
$$p(1) = c_0 + c_1 + c_2 + c_3 \tag{3.8}$$

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

$$v(t) = p'(t) = c_1 + 2c_2t + 3c_3t^2 \tag{3.9}$$
$$v(0) = c_1 \tag{3.10}$$
$$v(1) = c_1 + 2c_2 + 3c_3 \tag{3.11}$$

Então é possível obter a fórmula:

$$c_0 = p_0 \tag{3.12}$$
$$c_1 = v_0 \tag{3.13}$$
$$c_2 = -3p_0 - 2v_0 - v_1 + 3p_1 \tag{3.14}$$
$$c_3 = 2p_0 + v_0 + v_1 - 2p_1 \tag{3.15}$$

Escrevendo de forma matricial, temos:

$$p(t) = \begin{pmatrix} 1 & t & t^2 & t^3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ v_0 \\ v_1 \end{pmatrix} \tag{3.16}$$

3.1.1 Criação da *Spline*

Supondo que existam 4 pontos e é desejado criar uma *spline* que passe por todos eles. Para isso, 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)

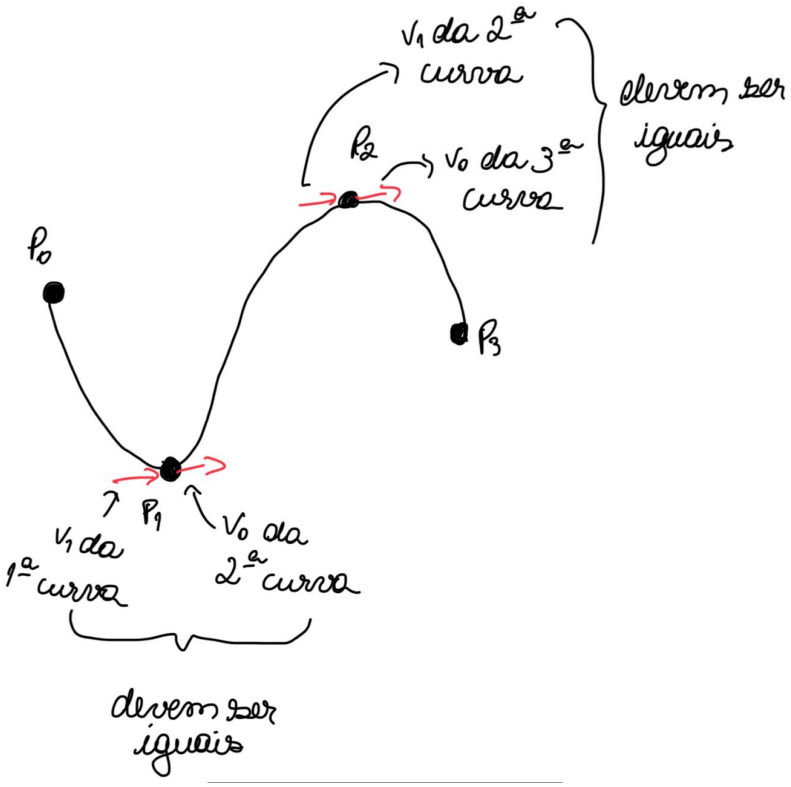
A primeira curva irá de p_0 a p_1, a segunda de p_1 a p_2 e a final de $p2$ a p_3. Então aplicaremos a fórmula que derivamos acima para p_t, mas em 3 variantes diferentes, cada variante com um par diferente de pontos finais. A fórmula da primeira curva tera pontos finais p_0 e p_1, a segunda em p_1 e p_2 e a terceira em p_2 e p_3, como é possível ver na FIGURA 10.

Para as tangentes, Edwin Catmull e Raphael Rom descobriram que uma forma de fazer isso é subtrair o ponto anterior do ponto seguinte. Então para calcular a tangente para o ponto p_2 é só fazer $(p_3 - p_1)$. Também é comum dividir por 2, pois tende a gerar curvas esteticamente agradáveis.

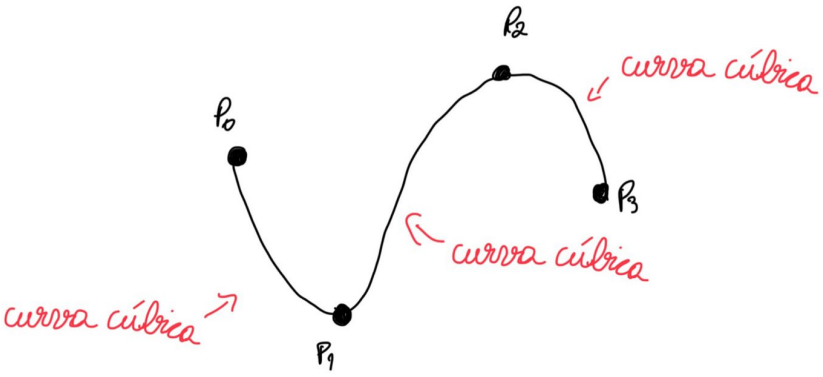
Logo é possível reescrever a equação em termos de 4 pontos p_0 a p_3.

Outra complicação que surge é que se precisar usar o ponto anterior e o próximo para calcular a tangente, surge a dúvida de qual será o ponto anterior de p_i e

FIGURA 10 – CURVAS CÚBICAS DE CATMULL-ROM.



Spline cúbica com 4 pontos de controle



Tangentes coincidentes nas conexões das cúbicas em cada ponto.

o próximo ponto de p_f. Uma solução para isso é adicionar pontos fantasmas que não fazem parte da curva.

Então ficamos com:

$$p(t) = \begin{pmatrix} 1 & t & t^2 & t^3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ \frac{p_2 - p_0}{2\tau} \\ \frac{p_3 - p_1}{2\tau} \end{pmatrix} \tag{3.17}$$

, onde τ é um fator de escala que pode ser ajustado para controlar a suavidade da

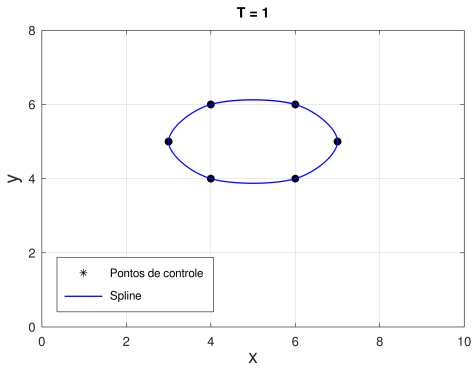
curva. O valor de τ pode ser definido com base na distância entre os pontos de controle ou em outras considerações geométricas.

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

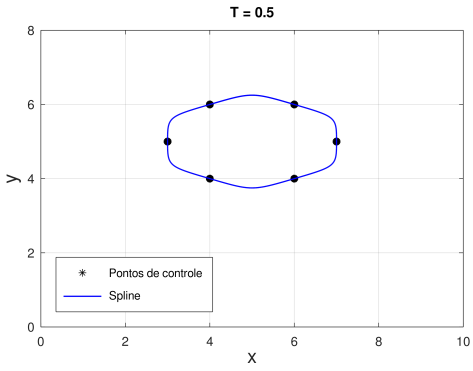
$$p(t) = \frac{1}{2} \begin{pmatrix} 1 & t & t^2 & t^3 \end{pmatrix} \begin{pmatrix} 0 & 2 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau - 6 & -2(\tau - 3) & -\tau \\ -\tau & 4 - \tau & \tau - 4 & \tau \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix} \tag{3.18}$$

Nas seguintes figuras é possível observar que quanto menor a tensão τ, menos suave fica a curva:

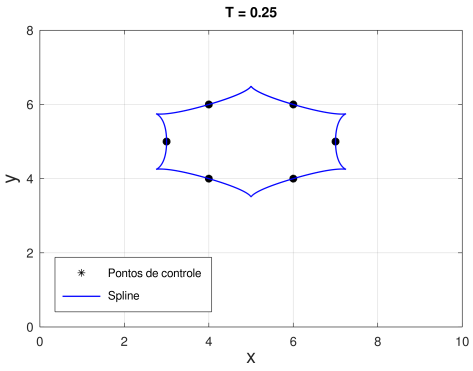
FIGURA 11 – TENSÃO DA CURVA.



τ = 1



τ = 0.5



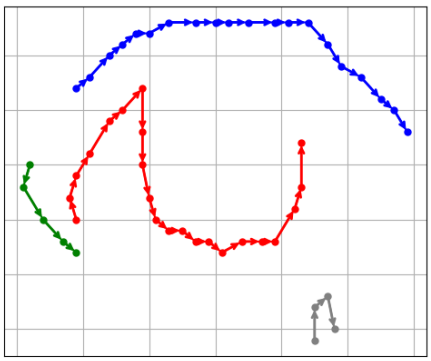
τ = 0.25

3.1.2 Resultados

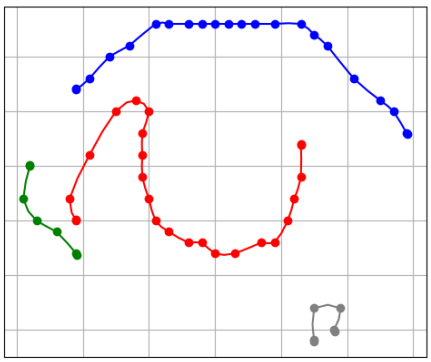
Foi implementado um algoritmo autoral que gera uma *spline* cúbica a partir de um conjunto de pontos de controle. O algoritmo utiliza a fórmula de Catmull-Rom para calcular os coeficientes da curva, garantindo suavidade e continuidade.

A seguir, são apresentados os resultados obtidos em conjuntos de pontos de controle que foram extraídos da imagem pelo processo explicado no Capítulo 2.

FIGURA 12 – SPLINES NOS PONTOS EXTRAÍDOS.



Reordenação dos pontos.



Splines.

4 *DYNAMIC TIME WARPING*

O Dynamic Time Warping (DTW) 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, o DTW é utilizado como ferramenta para comparação entre as curvas que representam as características faciais dos indivíduos, permitindo identificar semelhanças de mesmas pessoas em diferentes momentos ou condições.

4.1 INTRODUÇÃO AO DTW

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

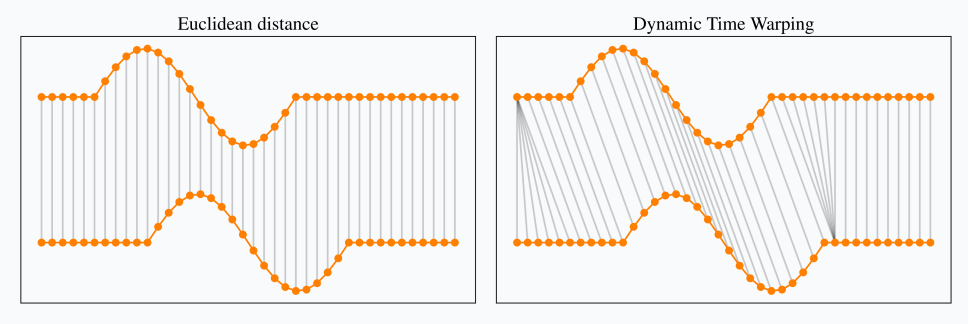
A matriz de custo é preenchida de forma recursiva, considerando o custo mínimo para cada posição $D(i, j)$:

$$D(i,j) = d(x_i, y_j) + \min \begin{cases} D(i-1, j) \\ D(i, j-1) \\ D(i-1, j-1) \end{cases} \tag{4.2}$$

onde $d(x_i, y_j)$ é a distância entre os pontos x_i e y_j. O custo total do alinhamento é obtido em $D(n, m)$.

A FIGURA 13 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 13 – ALINHAMENTO DINÂMICO.



FONTE: (Tavenard, 2021)

4.2 VANTAGENS E DESVANTAGENS DO DTW

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

- **Robustez a variações de tempo**: O DTW é capaz de lidar com sequências que apresentam variações na velocidade ou no comprimento, permitindo comparações mais precisas.
- **Alinhamento dinâmico**: O DTW 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**: O DTW pode ser afetado por ruídos nas sequências, o que pode levar a alinhamentos incorretos.

4.3 JUSTIFICATIVA PARA O USO DO DTW

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. O DTW permite alinhar essas curvas de forma dinâmica, possibilitando uma comparação mais precisa entre elas.

Além disso, este trabalho se fundamenta na abordagem proposta por Levada *et al.* (2008), que demonstrou a eficácia do uso do algoritmo DTW na comparação da variação das cores dos pixels, obtendo acurácias de 100%, 94% e 70%, conforme as especificidades de cada experimento em tarefas de reconhecimento facial.

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, foi implementado um script com base na proposta do artigo, cujo tempo de execução estimado foi de aproximadamente 10 horas para comparar duas imagens representadas por vetores de dimensão 77.284 (correspondentes às cores dos pixels).

Diante disso, uma das propostas deste trabalho é otimizar o processo de comparação por meio da utilização de curvas suaves representadas por splines, com o objetivo de reduzir a complexidade computacional sem comprometer a acurácia e a eficiência da técnica.

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 consistiu em um total de 24 imagens, pertencentes a quatro indivíduos distintos. É importante ressaltar que há um desbalanceamento na quantidade de imagens por indivíduo e todas as imagens foram capturadas sob condições controladas em orientação frontal, sem grandes variações de ângulo ou expressões faciais.

Para comparar sequências de características faciais extraídas de imagens, utilizei duas formas distintas. Uma delas foi segmentar três regiões distintas: olhos, nariz e boca, cada uma representada por um array bidimensional contendo as coordenadas dos pontos relevantes do contorno contendo as *splines*.

O DTW foi aplicado separadamente para cada uma dessas quatro representações, permitindo avaliar a similaridade local (por região).

Além disso, uma outra versão com todas as características concatenadas foi utilizada como referência global da face.

A seguir, estão apresentadas as tabelas com os valores de distância DTW para cada comparação.

TABELA 2 – Testes realizados para as características concatenadas.

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

TABELA 3 – Testes realizados para as características segmentadas.

Passo Spline	**Translação**	**Tensão**	**Acurácia (%)**	**Tempo**
0.5	Não	1	41.66	2min 12s
0.5	Não	0.5	41.66	2min 39s
0.5	Não	0.25	41.66	2min 21s
1	Não	1	45.83	1min 07s
1	Não	0.5	37.5	1min 08s
1	Não	0.25	41.66	1min 08s
0.5	Sim	1	79.16	2min 43s
0.5	Sim	0.5	75	2min 39s
0.5	Sim	0.25	70.83	2min 27s
1	Sim	1	75	1min 04s
1	Sim	0.5	79.16	1min 09s
1	Sim	0.25	79.16	1min 13s

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 7, 9, 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 2 vezes nas páginas 7, 25.

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 7, 31.

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 2 vez na página 7.

OPENCV. [*S. l.: s. n.*]. https://github.com/opencv/opencv/tree/master/data. Acessado em: 24-05-2024. Citado 2 vezes nas páginas 6, 10.

OPENCV. **Canny Edge Detection**. Disponível em: https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html. Citado 3 vezes nas páginas 7, 9, 13, 14.

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 1 vezes nas páginas 12, 32.

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 1 vezes nas páginas 12, 32.

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 5, 7.

TAVENARD, R. **An introduction to Dynamic Time Warping**. [*S. l.: s. n.*], 2021. https://rtavenar.github.io/blog/dtw.html. Citado 1 vezes nas páginas 7, 30.

TWIGG, C. **Catmull-Rom splines**. Pittsburg: CMU school of computer science, 2003. Disponível em: http://graphics.cs.cmu.edu/nsp/course/15-462/Fall04/assts/catmullRom.pdf. Citado 1 vez na página 7.

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 6, 9.

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 4 vezes nas páginas 7, 9, 18, 19.