

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
INSTITUTO DE INFORMÁTICA  
TRABALHO DA DISCIPLINA INF01030 - FUNDAMENTOS DE VISÃO  
COMPUTACIONAL

## **Implementação de algoritmo de detecção de parábolas**

Alunos: Leonardo Oliveira Wellausen

Matheus Fernandes Kovaleski

Orientador: Prof. Dr. Cláudio Rosito Jung

Porto Alegre, Novembro de 2019

# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Metodologia</b>	<b>2</b>
2.1	Algoritmo . . . . .	2
<b>3</b>	<b>Resultados Obtidos</b>	<b>5</b>
<b>4</b>	<b>Conclusão</b>	<b>11</b>

# Lista de Figuras

1.1	Exemplos de execução normal do programa. . . . .	1
3.2	Resultado obtido para o exemplo 1. . . . .	5
3.3	Resultado para o exemplo 1 exibindo todas as retas encontradas pela transformada de <i>Hough</i> (amarelo). A reta azul é o eixo 2 escolhido pelo programa, antes de ele ser rotacionado e transformado na reta rosa. Por ser a imagem com traços mais fortes nos eixos, é a que mais retorna retas; e também a mais fácil de se trabalhar. Foi a imagem de teste durante toda a implementação das técnicas . . . . .	6
3.4	Resultado obtido para o exemplo 2. . . . .	7
3.5	Resultado obtido para a segmentação por <i>K-means</i> no exemplo 2. Nota-se que o eixo vertical, apesar de fraquíssimo na imagem original, foi bastante acentuado pela dilatação e pela própria clusterização. Esta foi a imagem que nos obrigou a abandonar <i>thresholding</i> e implementar <i>K-means</i> , pois o algoritmo sempre acaba selecionando a "perninha" da parábola como eixo vertical. . . . .	8
3.6	Resultado obtido para o exemplo 3. . . . .	9
3.7	Resultado obtido para o exemplo 3 após remoção dos eixos encontrados pela transformada de <i>Hough</i> . Estes são os pontos (brancos!) que serão passados ao <i>RANSAC</i> . . . . .	10

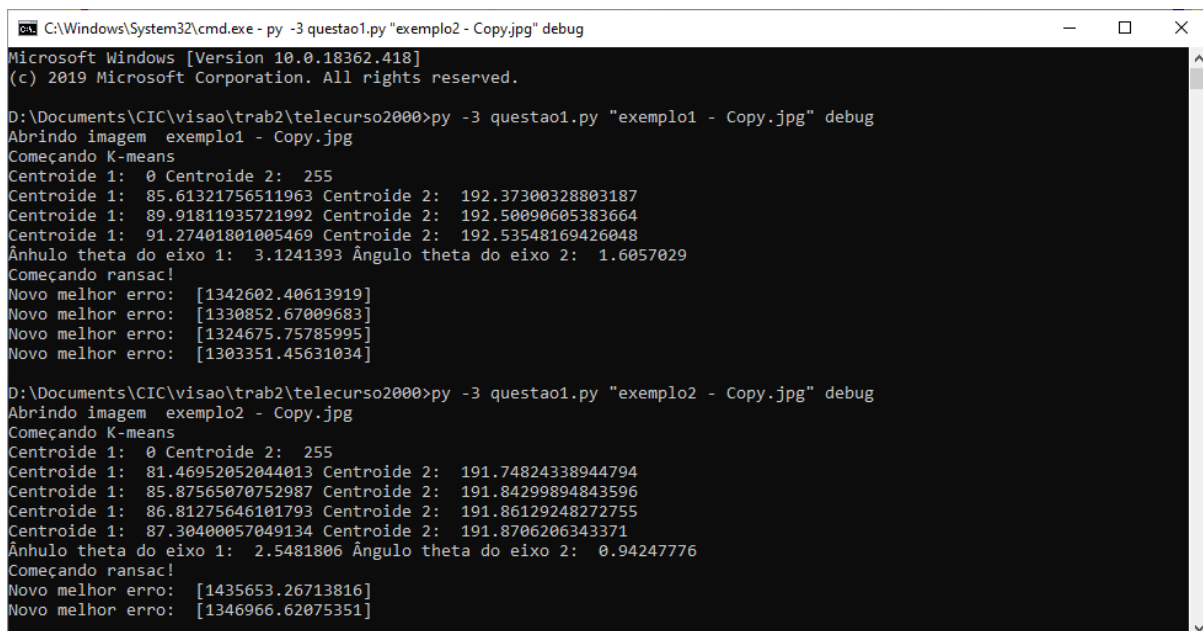
# 1 Introdução

O objetivo deste trabalho de implementação é a realização de um programa capaz de identificar e reproduzir uma parábola desenhada com canetão em um quadro branco.

As ferramentas utilizadas para cumprir tal objetivo foram a linguagem de programação *Python 3* com o uso das seguintes bibliotecas não padrão: *opencv-python* para leitura e exibição de imagens; *numpy* para cálculo matricial, incluindo resolução de sistema linear.

Foi desenvolvido um programa utilizando as ferramentas citadas acima para atender aos requisitos exigidos na definição do trabalho. O funcionamento do programa é como segue:

- *questao1.py* - Pode receber dois parâmetros posicionais; o primeiro sendo o nome do arquivo de imagem a ser aberto, e o segundo, opcional, se é desejado que sejam exibidas imagens intermediárias para debug (para ativar esta opção deve se inserir apenas "debug" após o nome da imagem). Exemplos de como chamar o programa podem ser vistos em 1.1



```
C:\Windows\System32\cmd.exe - py -3 questao1.py "exemplo2 - Copy.jpg" debug
Microsoft Windows [Version 10.0.18362.418]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\Documents\CIC\visao\trab2\telecurso2000>py -3 questao1.py "exemplo1 - Copy.jpg" debug
Abrindo imagem exemplo1 - Copy.jpg
Começando K-means
Centroide 1: 0 Centroide 2: 255
Centroide 1: 85.61321756511963 Centroide 2: 192.37300328803187
Centroide 1: 89.91811935721992 Centroide 2: 192.50090605383664
Centroide 1: 91.27401801005469 Centroide 2: 192.53548169426048
Ânhulo theta do eixo 1: 3.1241393 Ângulo theta do eixo 2: 1.6057029
Começando ransac!
Novo melhor erro: [1342602.40613919]
Novo melhor erro: [1330852.67009683]
Novo melhor erro: [1324675.75785995]
Novo melhor erro: [1303351.45631034]

D:\Documents\CIC\visao\trab2\telecurso2000>py -3 questao1.py "exemplo2 - Copy.jpg" debug
Abrindo imagem exemplo2 - Copy.jpg
Começando K-means
Centroide 1: 0 Centroide 2: 255
Centroide 1: 81.46952052044013 Centroide 2: 191.74824338944794
Centroide 1: 85.87565070752987 Centroide 2: 191.84299894843596
Centroide 1: 86.81275646101793 Centroide 2: 191.86129248272755
Centroide 1: 87.30400057049134 Centroide 2: 191.8706206343371
Ânhulo theta do eixo 1: 2.5481806 Ângulo theta do eixo 2: 0.94247776
Começando ransac!
Novo melhor erro: [1435653.26713816]
Novo melhor erro: [1346966.62075351]
```

Figura 1.1: Exemplos de execução normal do programa.

## 2 Metodologia

Esta seção discutirá sobre a metodologia adotada para se atingir o resultado final. Discutiremos sobre técnicas escolhidas e suas implementações.

### 2.1 Algoritmo

Descrição alto nível das etapas realizadas para implementação dos requisitos:

1. Carregar imagem **img**
2. Transformar a imagem colorida **img** em uma imagem **grey-img** em tons de cinza.
3. Segmentar a imagem **grey-img** utilizando *K-means*.
  - Criamos 2 *clusters*, um deve representar o quadro (fundo, branco) e o outra representa os traços de interesse (canetão, preto). *Cluster* de interesse acaba contendo ruído. A imagem **grey-img** é binarizada com base nos *clusters* retornados. Como sabemos que as cores (centroides) almejadas para cada cluster são **0 (preto)** e **255 (branco)**, representando os traços e o quadro branco.
4. Aplicar uma transformação morfológica de dilatação.
  - Aplicamos a dilatação pois algumas das imagens fornecidas apresentam traços muito fracos nos desenhos dos eixos cartesianos, criando uma dificuldade no estabelecimento de valores padrão para o algoritmo de *Hough*.
5. Aplicar a *Transformada de Hough* na imagem **grey-img**.
  - A transformada é utilizada para encontrar as duas retas principais que formam um eixo cartesiano na imagem fornecida. A implementação utilizada é a fornecida pelo *OpenCV* e os parâmetros são: **rhô** variando em 1; **theta** variando em 1°; **número de votos** em uma reta para considerá-la uma reta, de fato, na imagem é 500.
6. Definir as duas retas que formam o eixo cartesiano.
  - A transformada retorna diversas retas para cada traço representando um eixo, devido à espessura dos traços. É preciso selecionar uma reta representando um eixo e, após, procurar (na lista de retas retornadas por *Hough*) por uma segunda reta que forme um ângulo o mais próximo de 90° com a primeira, este será o segundo eixo. Chamaremos estes eixos de **eixo 1** (primeiro encontrado) e **eixo 2** (segundo encontrado).
7. Utilizar as duas retas recém encontradas para apagar os eixos na imagem **grey-img**.
  - Utilizamos as retas encontradas e as desenhamos na imagem **grey-img**, porém com uma grande espessura e de cor preta. Assim ficamos com a imagem contendo apenas os pontos pertencentes à parábola (+ ruído).

8. Encontrar o **ponto de intersecção** entre os eixos. Desenhemos um quadrado **azul** neste ponto.
9. Gerar um novo **eixo 2 ajustado**, semelhante ao **eixo 2** encontrado anteriormente, porém formando exatamente  $90^\circ$  com o **eixo 1**.
  - Geramos um vetor **v1** a partir do **eixo 1**, rotacionamos esse vetor em  $90^\circ$  gerando, assim, um vetor **v2** perpendicular ao **v1**. O nosso novo **eixo 2 ajustado** é definido a partir de 2 pontos: o próprio **ponto de intersecção** calculado anteriormente; um novo ponto definido como o mesmo ponto de intersecção, porém com uma translação em direção ao vetor **v2** calculado.
10. Desenhar os eixos na imagem **img**. **Eixo 1** em vermelho e **eixo 2 ajustado** em rosa.
11. Definir todos os pontos brancos em **grey-img** como sendo uma parábola **parabola**.
12. Rotacionar **parabola** de forma que ela esteja alinhada horizontalmente com a imagem.
  - O eixo de rotação **alpha** é definido como o ângulo dentre o **eixo 2** e o eixo horizontal (exatamente  $90^\circ$ ) da imagem. Esta rotação é feita, primeiramente transladando **parabola** para a origem da imagem (com base na posição central dos pontos que definem **parabola**); seguida da rotação de fato, definida por uma matriz de rotação; e então a parábola rotacionada é transladada de volta para a sua posição de origem.
13. Encontrar a equação que representa **parabola** utilizando a técnica *RANSAC*.
  - (a) Define variáveis de melhor erro e melhor equação zeradas. Partimos com o conjunto de pontos **parabola**.
  - (b) Escolhemos **3 pontos** aleatórios de **parabola**.
  - (c) *Fittamos* uma equação de parábola com base nesses 3 pontos. Para encontrarmos esta equação geramos um sistema linear  $\mathbf{Ax} = \mathbf{B}$  (representação matricial de  $\mathbf{ax}^2 + \mathbf{bx} + \mathbf{c} = \mathbf{y}$ ) com os pontos selecionados; a partir deste sistema geramos o sistema  $\mathbf{A}^T \mathbf{Ax} = \mathbf{A}^T \mathbf{B}$  que representa a solução por *mínimos quadrados* do sistema inicial. Este sistema é resolvido com a função *linalg.solve()* do pacote *numpy*.
  - (d) Utilizado a equação encontrada no passo anterior, geramos uma lista de **inliers**; isto é, todos os pontos em **parab** que são suficientemente bem representados pela equação encontrada.
  - (e) Se *inliers* não contém uma quantidade significativa dos pontos de **parabola**, voltamos para (b); caso contrário, *fittamos* uma nova equação, desta vez gerando um sistema linear muito maior, que contém todos os pontos em *inliers*.
  - (f) Calculamos o **erro** da nova equação gerada, isto é: para todo ponto de **parabola** usamos sua coordenada  $x$  na nossa equação e calculamos a diferença absoluta entre o valor retornado e o valor  $y$  original do ponto; essa diferença é somada ao **erro**.

- (g) Se o **erro** calculado é menor que o menor erro já obtido, definimos este erro como o **menor erro** e esta equação como a **melhor equação**.
  - (h) Voltamos ao passo (b) e repetimos por um número fixo de vezes (arbitrado aqui, em 100).
14. Utilizando a equação final devolvida pelo *RANSAC*, gerar os pontos da parábola definida por tal equação, nominalmente **fitted-parabola**.
  15. Rotacionar os pontos de **fitted-parabola** por um ângulo de **-alpha**.
    - Todo o procedimento de rotação é como o já feito anteriormente (estamos, basicamente, desfazendo a rotação anterior). Estas rotações foram necessárias para que a parábola recuperada pela análise da imagem seja de fato uma *função* nas coordenadas da imagem, de forma que o *solver* utilizado funcione corretamente.
  16. Desenhar a parábola **fitted-parabola** na imagem **img** em cor **verde**.
  17. Exibir a imagem **img** como resultado!

### 3 Resultados Obtidos

Nesta seção executamos o programa para as **3** imagens de exemplo fornecidas na definição do trabalho. Serão exibidos e discutidos os resultados finais, assim como alguns resultados intermediários.

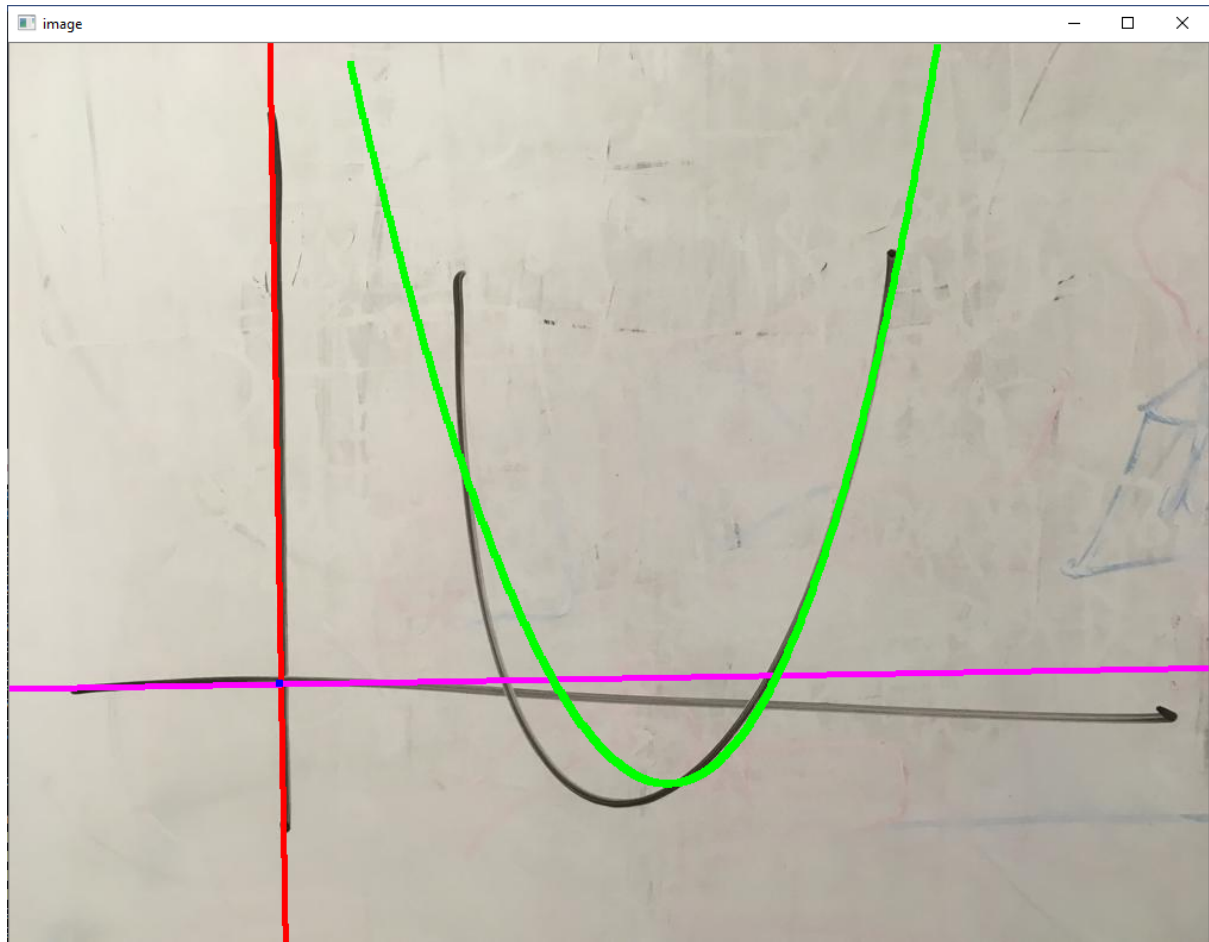


Figura 3.2: Resultado obtido para o exemplo 1.



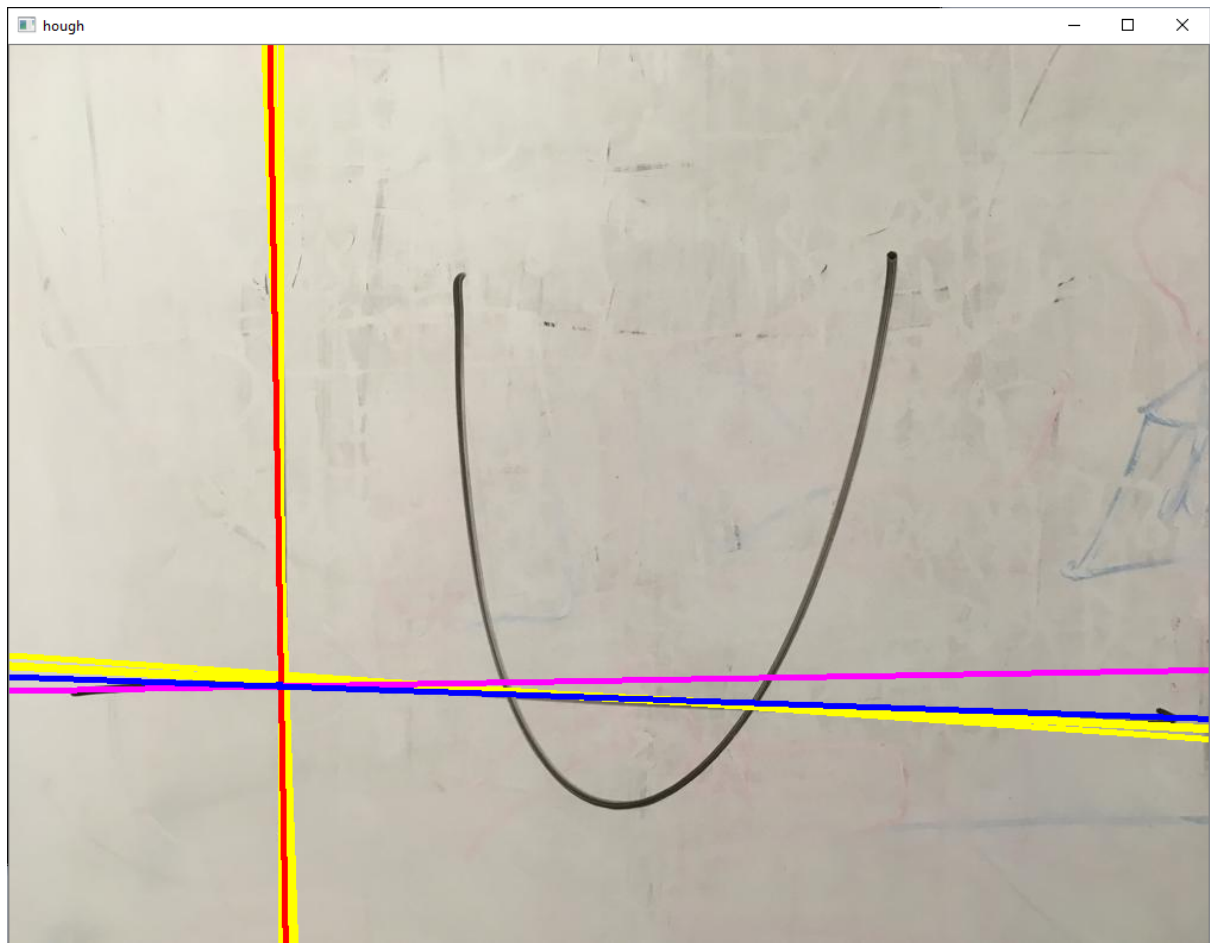


Figura 3.3: Resultado para o exemplo 1 exibindo todas as retas encontradas pela transformada de *Hough* (amarelo). A reta azul é o eixo 2 escolhido pelo programa, antes de ele ser rotacionado e transformado na reta rosa. Por ser a imagem com traços mais fortes nos eixos, é a que mais retorna retas; e também a mais fácil de se trabalhar.

Foi a imagem de teste durante toda a implementação das técnicas

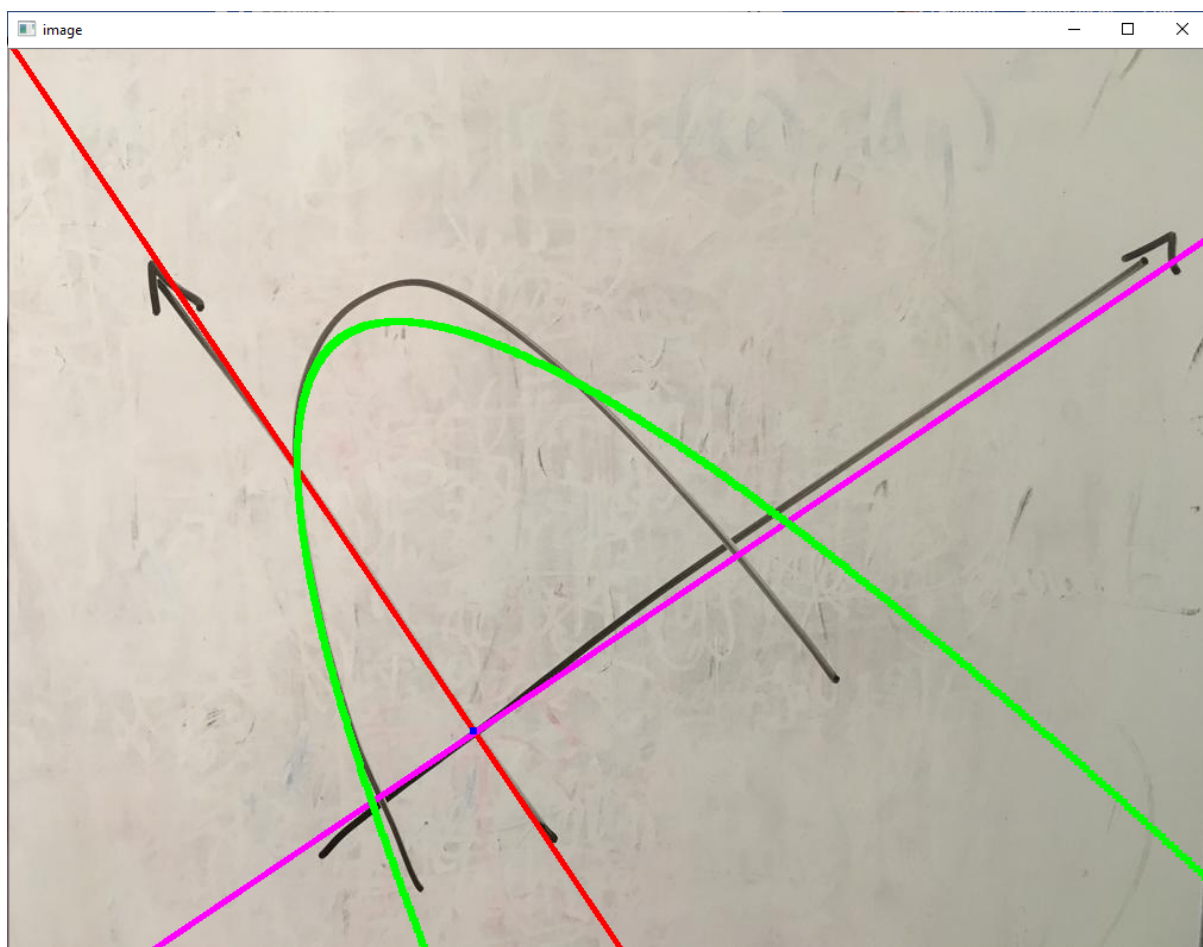


Figura 3.4: Resultado obtido para o exemplo 2.

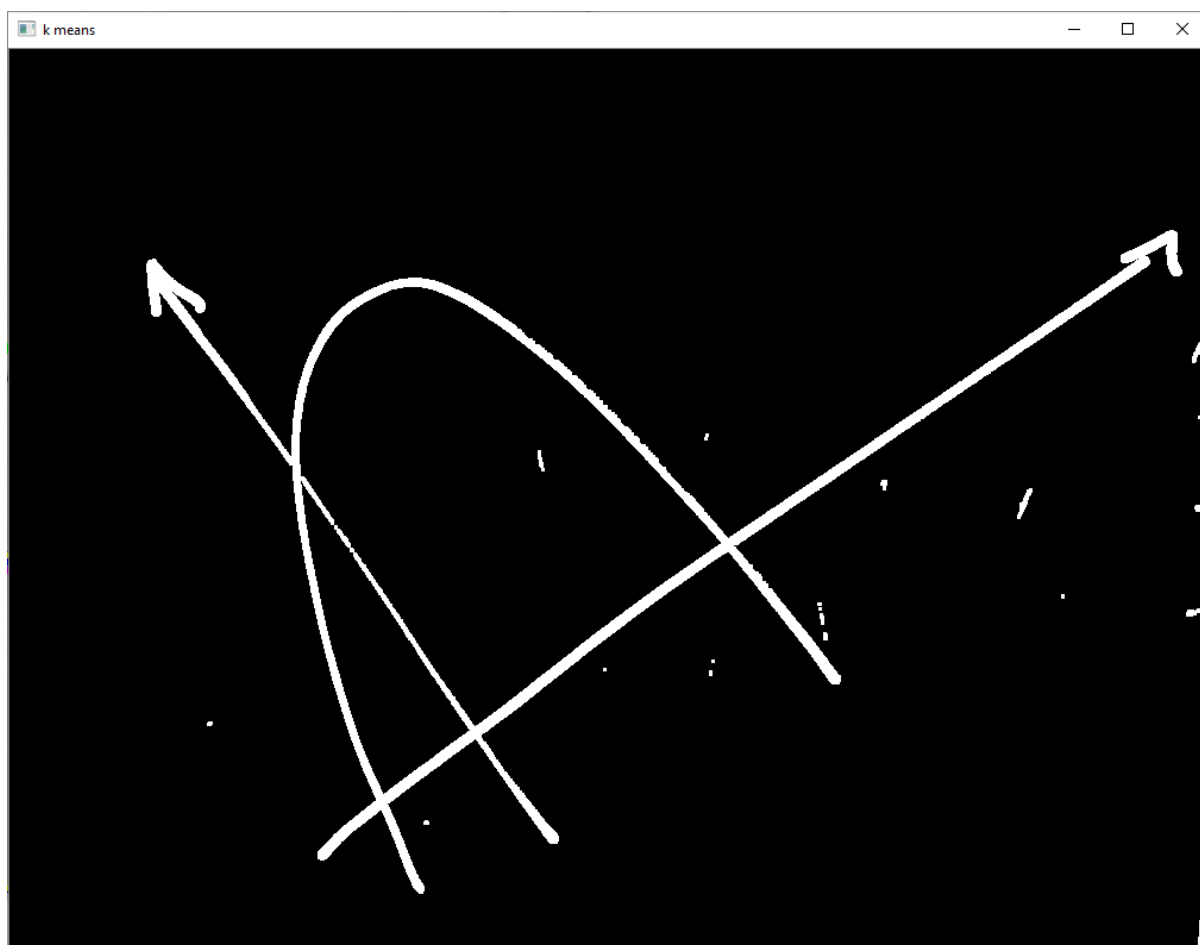


Figura 3.5: Resultado obtido para a segmentação por *K-means* no exemplo 2. Nota-se que o eixo vertical, apesar de fraquíssimo na imagem original, foi bastante acentuado pela dilatação e pela própria clusterização. Esta foi a imagem que nos obrigou a abandonar *thresholding* e implementar *K-means*, pois o algoritmo sempre acaba selecionando a "perninha" da parábola como eixo vertical.

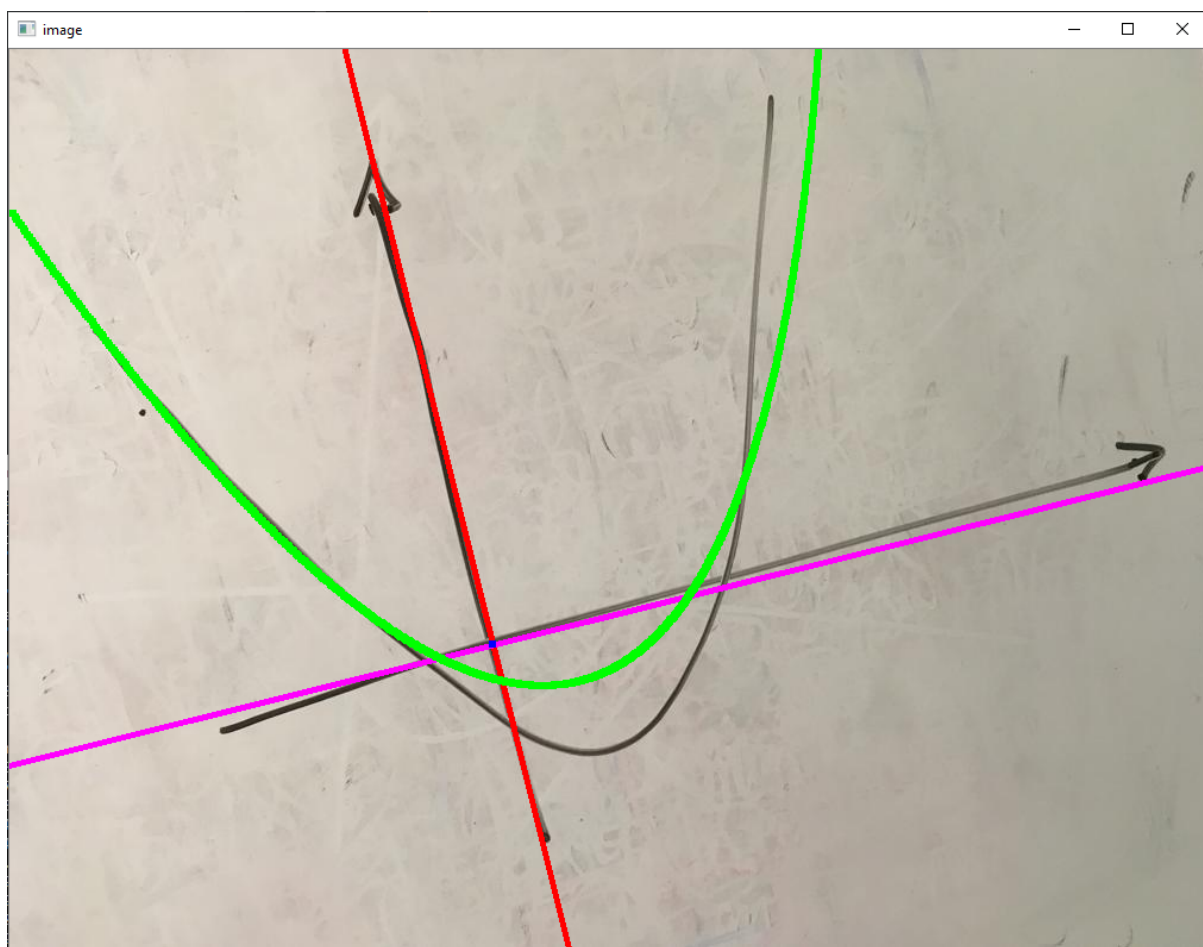


Figura 3.6: Resultado obtido para o exemplo 3.



Figura 3.7: Resultado obtido para o exemplo 3 após remoção dos eixos encontrados pela transformada de *Hough*. Estes são os pontos (brancos!) que serão passados ao *RANSAC*.

## 4 Conclusão

Após a implementação deste trabalho notamos o quanto técnicas muito simples, quando combinadas, podem ser utilizadas para aplicações interessantes de mundo real.

Um desafio interessante proposto pelo trabalho é a dificuldade de produzir uma técnica que funcione bem para exemplos que não são bem comportados. Por mais que as técnicas implementadas não fossem de grande complexidade técnica, foi desafiador conectá-las e parametrizá-las de forma que o programa sempre funcionasse para todas as imagens; diversas vezes chegamos em pontos onde o programa reotornava resultados bons para duas imagens, porém, ao rodar a terceira...tudo quebrava!

Por fim, as decisões foram guiadas por fatores como: a diferença de força no traço entre as imagens, resolvido com *K-means*; a diferença na retilineidade dos eixos e das próprias parábolas, resolvido com métodos mais robustos para a escolha de quais retas retornadas por *Hough* devem ser de fato consideradas como os eixos principais; e quantidade de ruído no quadro, resolvido com *RANSAC*.