Detecção de Marcadores para Estimativa de Posição Relativa

Rodrigo Passos Sousa

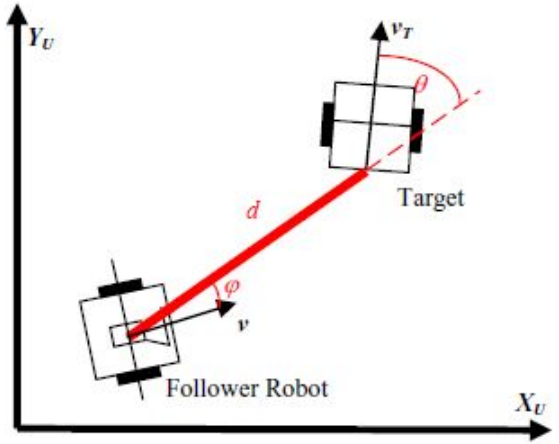
Sumário

- Introdução
- Objetivo
- Fluxograma do Projeto
- Redimensionamento da Imagem
  - Pirâmide de Imagens
- Segmentação da Imagem
  - Algoritmo de Suzuki e Abe
  - Algoritmo de Douglas Peucker
- Transformação Geométrica
- Comparação com o Dicionário
- Refinamento da Estimativa dos Cantos
- Estimação da Pose
- Conclusões

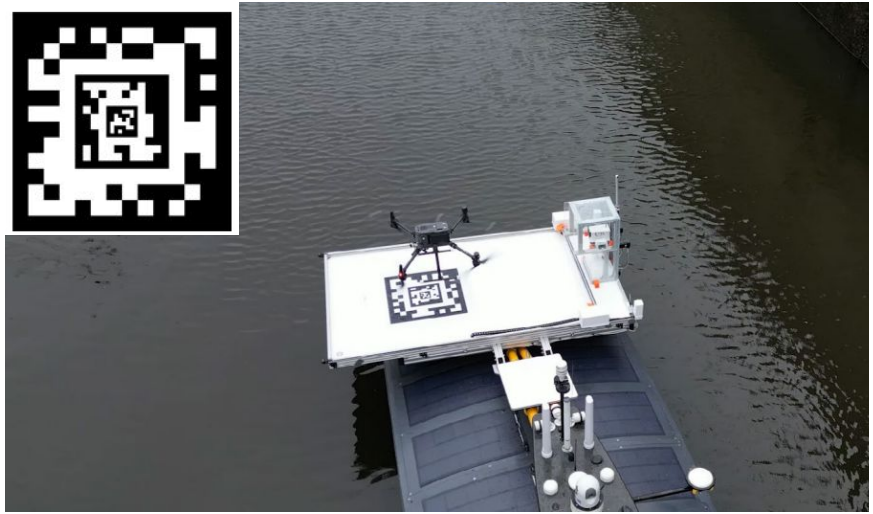
Introdução : : Motivação

Vision-Based Tracking Control for Mobile Robots*

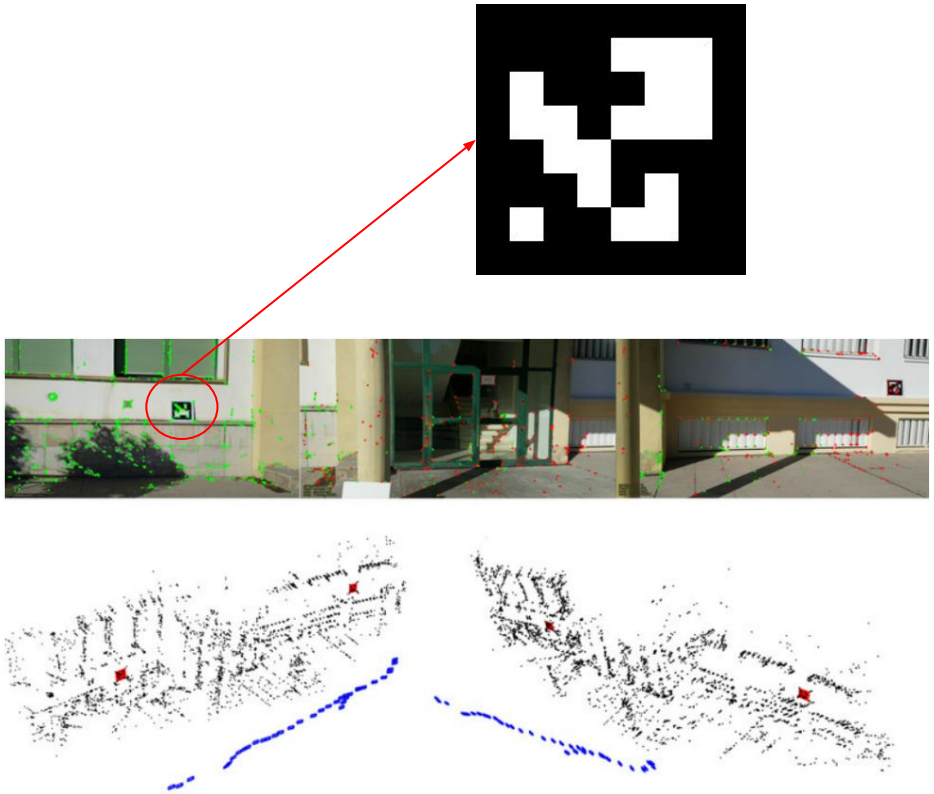
Ricardo Carelli, Carlos M. Soria and Beatriz Morales
Instituto de Automática, Universidad Nacional de San Juan
Av. San Martín Oeste, 5400 San Juan, Argentina
{rcarelli,csoria,bmorales}@inaut.unsj.edu.ar



Introdução : : Marcadores

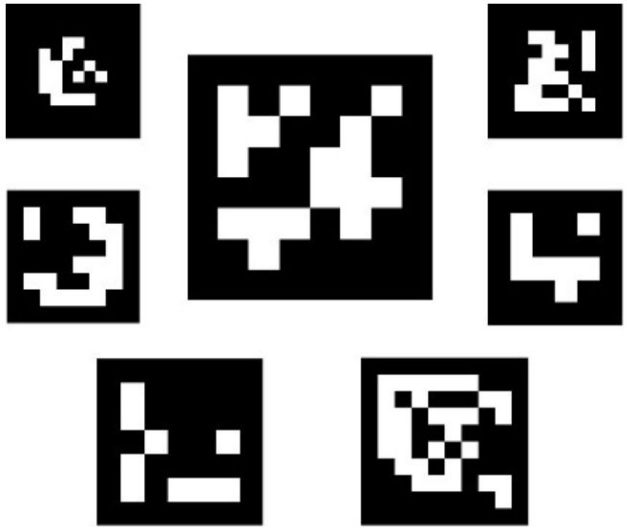


Navegação Robótica

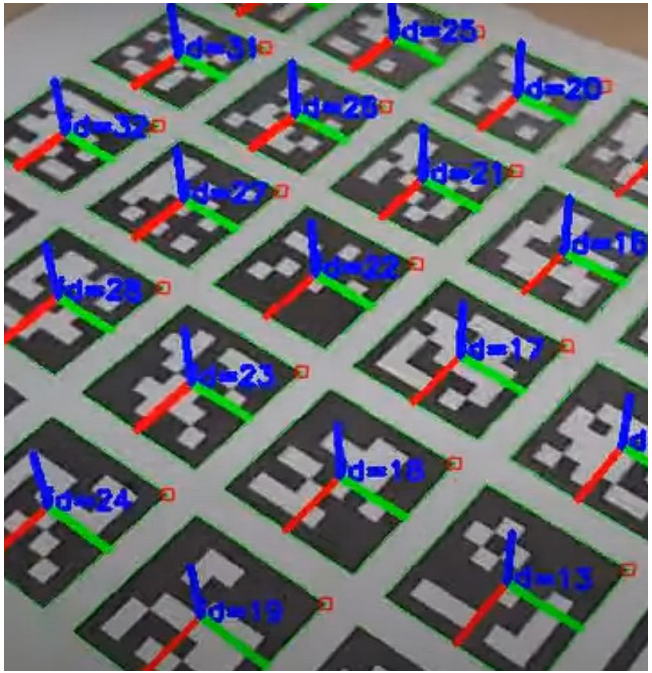


Localização e Mapeamento Simultâneos

Introdução : : Biblioteca Aruco



Dicionário de Marcadores



Identificação e Estimação
da Pose

Objetivo

- O objetivo desse projeto é implementar a solução apresentada pela biblioteca ArUco para **identificação de marcadores**, e com isso obter a **posição relativa de um objeto** com relação ao sistema de coordenadas da câmera. Para tal foi usando como base, o seguinte artigo:

Contents lists available at ScienceDirect

Image and Vision Computing

journal homepage: www.elsevier.com/locate/imavis

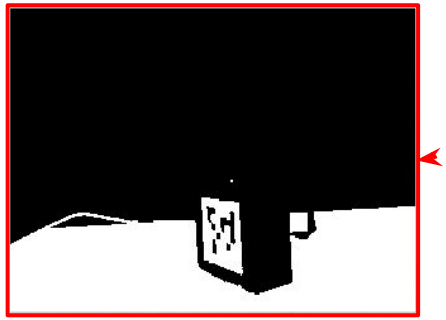
Speeded up detection of squared fiducial markers☆

Francisco J. Romero-Ramirez[a], Rafael Muñoz-Salinas[a,b,*], Rafael Medina-Carnicer[a,b]

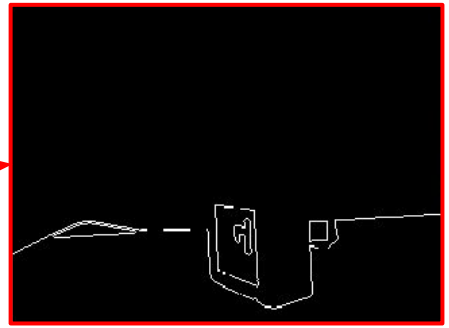
[a]Departamento de Informática y Análisis Numérico, Edificio Einstein, Campus de Rabanales, Universidad de Córdoba, 14071 Córdoba, Spain
[b]Instituto Maimónides de Investigación en Biomedicina (IMIBIC), Avenida Menéndez Pidal s/n, 14004 Córdoba, Spain

Fluxograma do Projeto

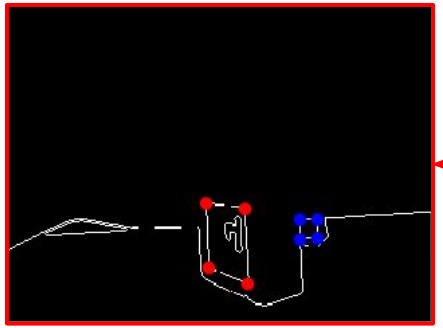
Binarização da Imagem (Otsu)



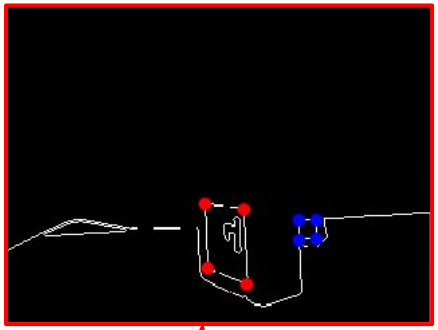
Detecção de Borda e Segmentador (Suzuki e Abe)

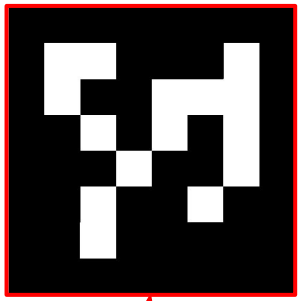


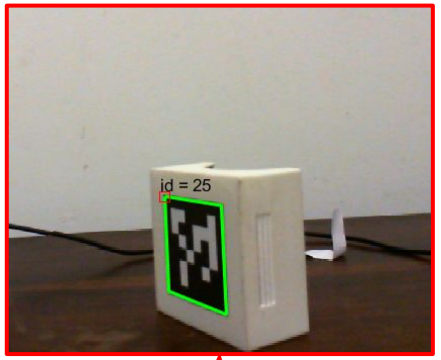
Filtragem dos Objetos (Douglas Peucker)



Fluxograma do Projeto

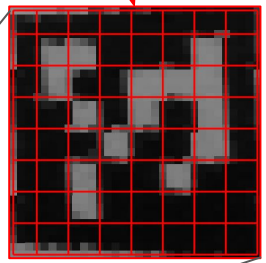






Obtenção do Código (Matriz Binária)

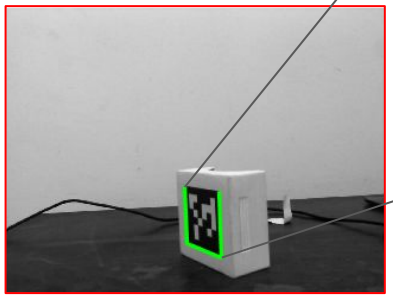
Transformação Geométrica (Pseudo Inversa)



Identificação do Padrão

Refina a posição dos cantos (Tamanho original da imagem)

Estima a posição relativa

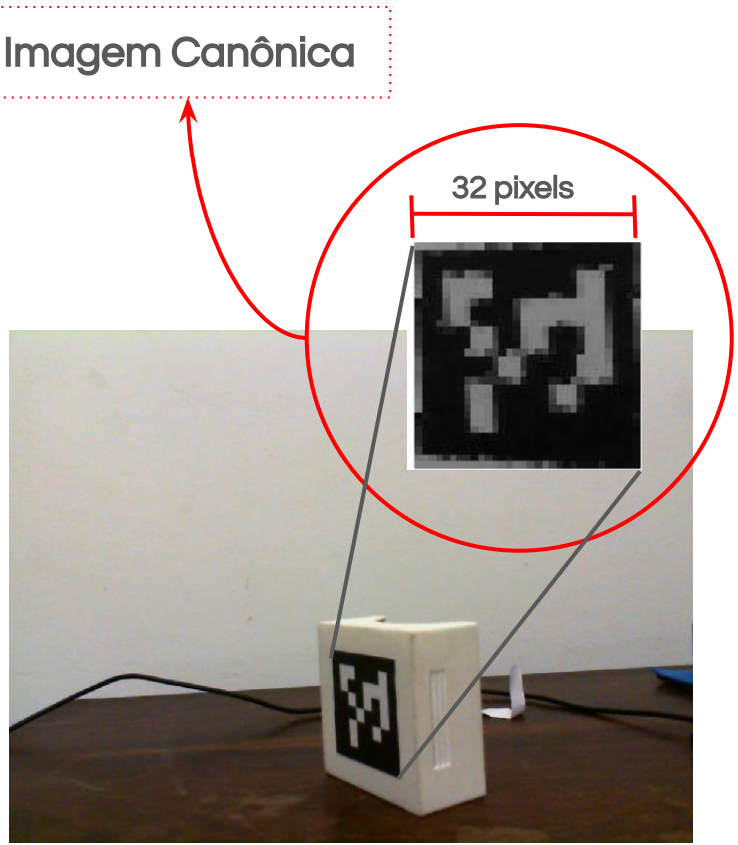


Redimensionamento da Imagem

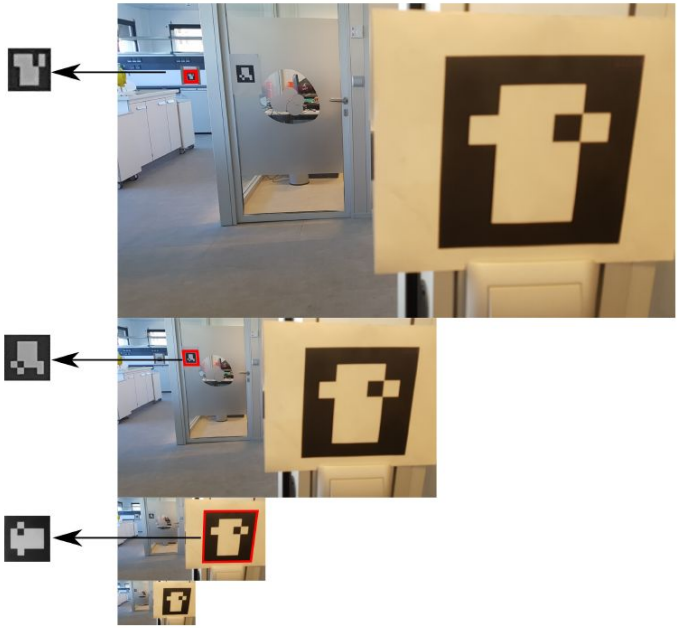
- Reduzir a quantidade de cálculos, reduzindo a quantidade de pixels

$$\text{Porcentagem de Redução} = \frac{32}{\text{Menor Lado}}$$

- Menor Lado = 100;



Redimensionamento da Imagem : : Pirâmide de Imagens

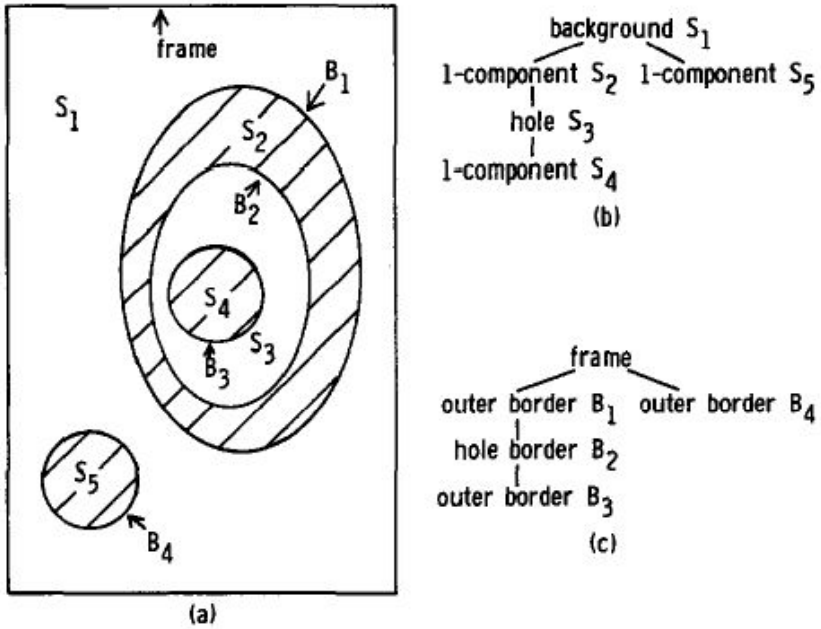


Fonte: Romero-Rodríguez *et. al*

- Sequência de N imagens redimensionadas com tamanhos diferentes.
- A quantidade N de imagens é tal que a menor imagem, tenha dimensões maiores que 32x32 pixels, em que as dimensões da imagem são sequencialmente divididas por 2.

Segmentação da Imagem

- Seguidor de Borda de Suzuki e Abe (1985)

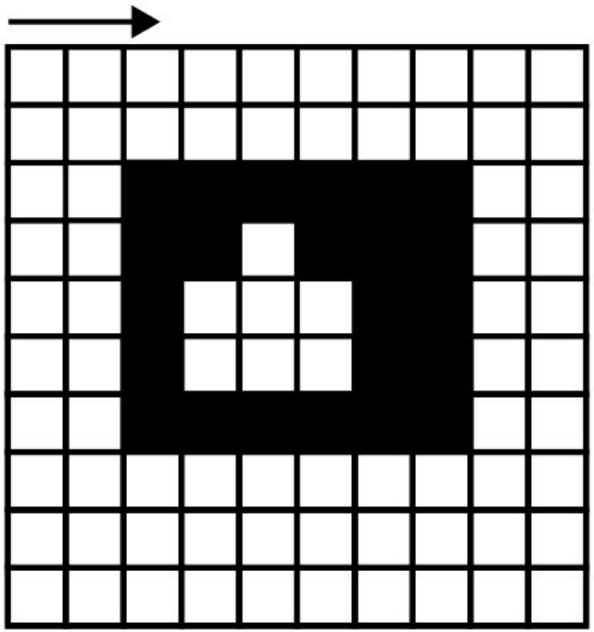


Algoritmo de Suzuki e Abe

(0) Inicializaçao

- NBD ← 0

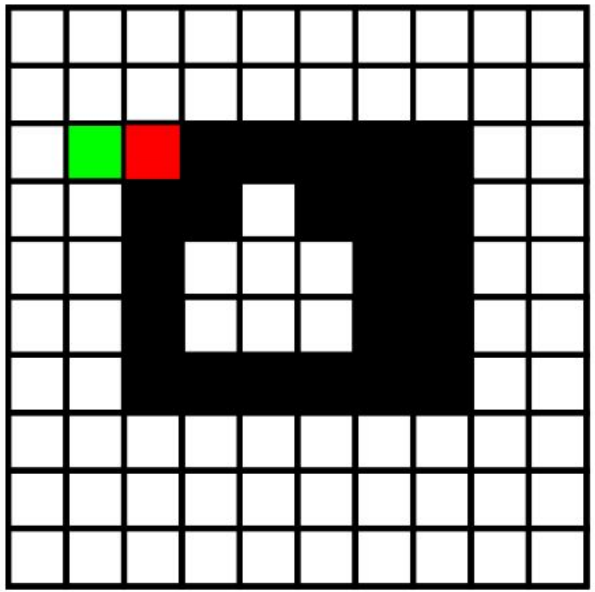
(1) Inicia a Varredura da Imagem

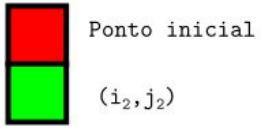


(2) Se f_{ij}=1 e $f_{i,j-1}$=0,

- o pixel (i,j) é o Ponto Inicial
- NBD ← NBD + 1
- (i_2,j_2) ← (i,j-1)

Caso contrário, vai direto para o (4)

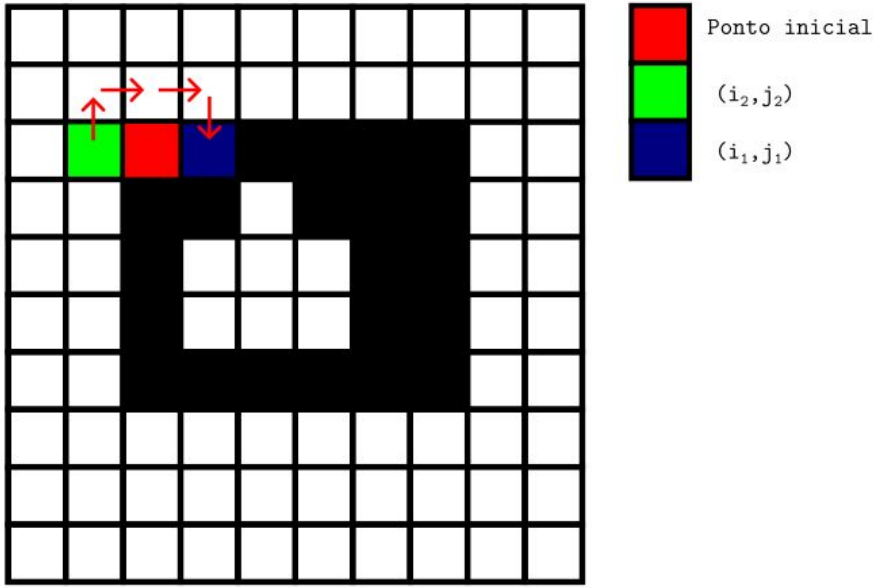




Algoritmo de Suzuki e Abe

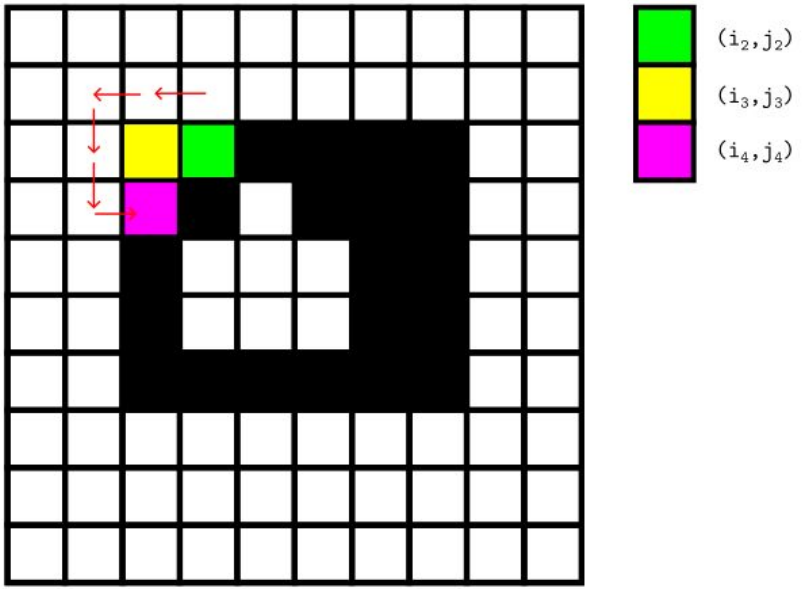
(3) Começando em (i,j), siga a borda:

(3.1) Começando de (i_2,j_2) visite os vizinhos de (i,j), no sentido horário. O primeiro pixel não nulo será denotado por (i_1,j_1). Se não houver, $f_{ij} \leftarrow$ -NBD, e vai para (4)



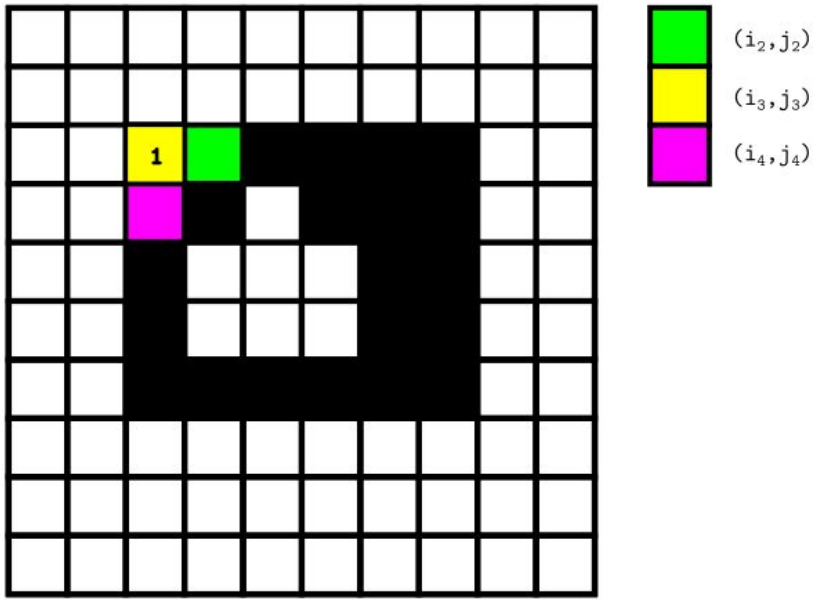
(3.2) $(i_2,j_2) \leftarrow (i_1,j_1)$ e $(i_3,j_3) \leftarrow (i,j)$

(3.3) Começando pelo próximo a (i_2,j_2), no sentido anti-horário, vizitar os vizinhos de (i_3,j_3), até encontrar um pixel não nulo (i_4,j_4).

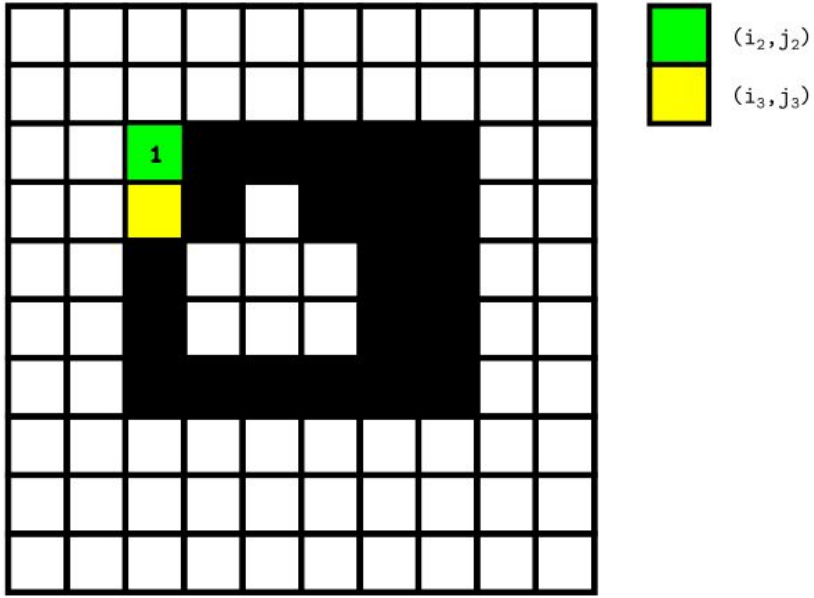


Algoritmo de Suzuki e Abe

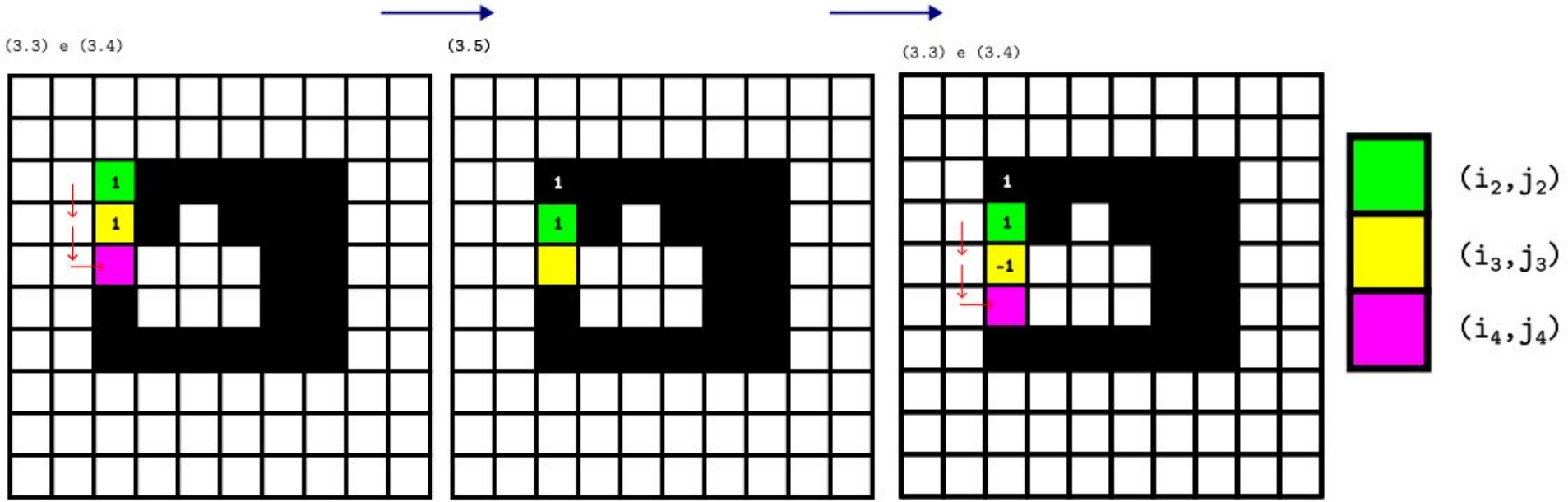
(3.4) Mude o valor fi3,j3 do pixel (i_3, j_3):
(a) Se o pixel (i_3, j_3+1) é nulo, $f_{i3,j3} \leftarrow$ -NBD
(b) Se o pixel (i_3, j_3+1) é não nulo, $f_{i3,j3} \leftarrow$ NBD
(c) Caso contrário, não mude $f_{i3,j3}$.



(3.5) Se (i4,j4) = (i,j) e (i3,j3)=(i1,j1) (Completou o Objeto), vá para o item (4).
Caso contrário, (i2,j2) ← (i3,j3) e (i3,j3) ← (i4,j4) e vai para o item (3.3).

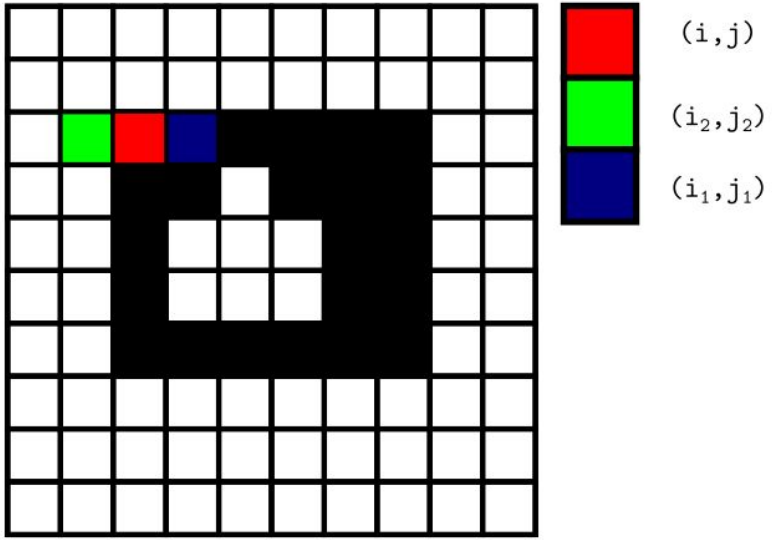


Algoritmo de Suzuki e Abe

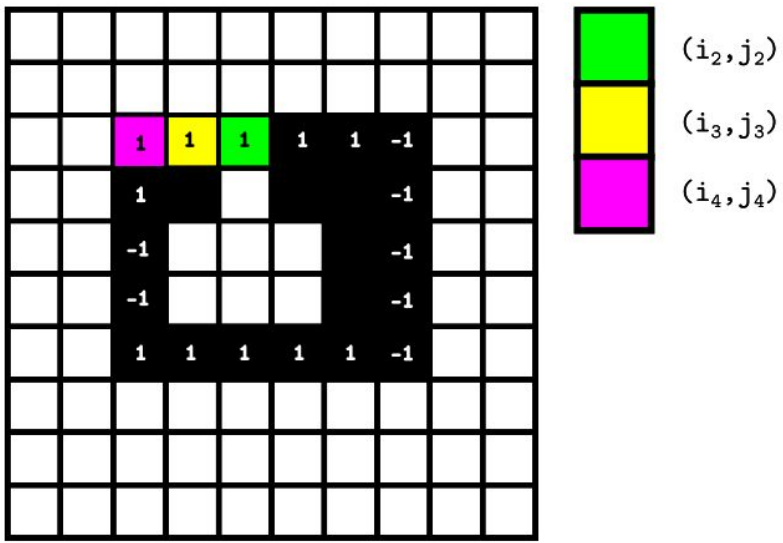


Algoritmo de Suzuki e Abe

(3.5) Se (i4,j4) = (i,j) e (i3,j3)=(i1,j1) (Completou o Objeto), vá para o item (4).
Caso contrário, (i2,j2) ← (i3,j3) e (i3,j3) ← (i4,j4) e vai para o item (3.3).



(3.3), (3.4) e (3.5)

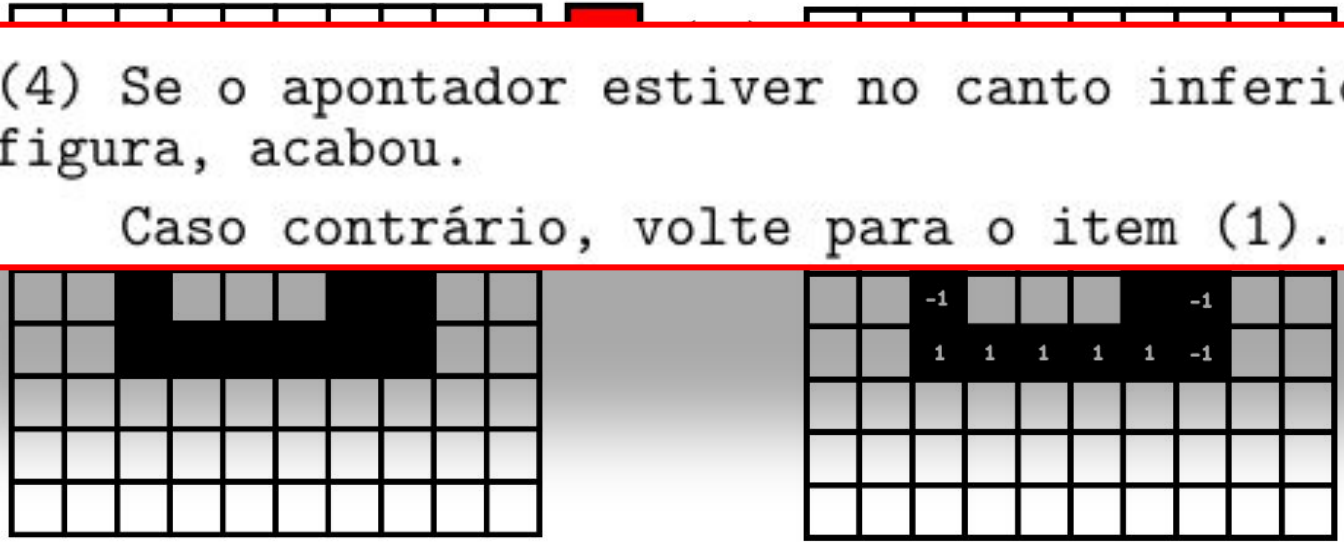


Algoritmo de Suzuki e Abe

(3.5) Se (i4,j4) = (i,j) e (i3,j3)=(i1,j1) (Completou o Objeto), vá para o item (4).
Caso contrário, (i2,j2) ← (i3,j3) e (i3,j3) ← (i4,j4) e vai para o item (3.3).

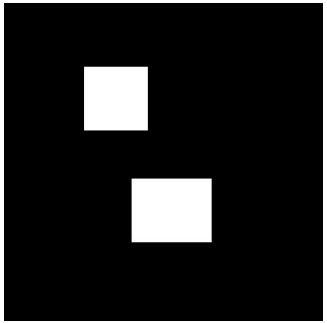
(3.3), (3.4) e (3.5)

(4) Se o apontador estiver no canto inferior direito da figura, acabou.
Caso contrário, volte para o item (1).



Algoritmo de Suzuki e Abe

Entrada 20x20 pixels:

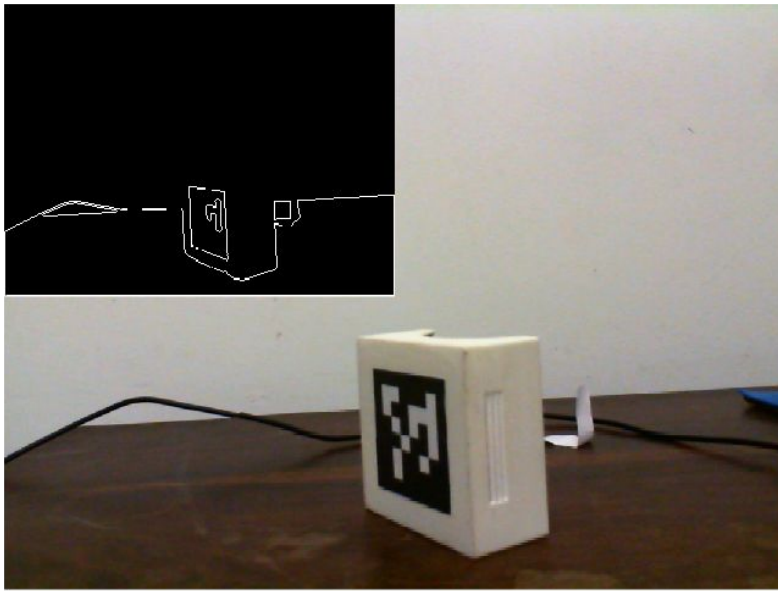


Resultado:

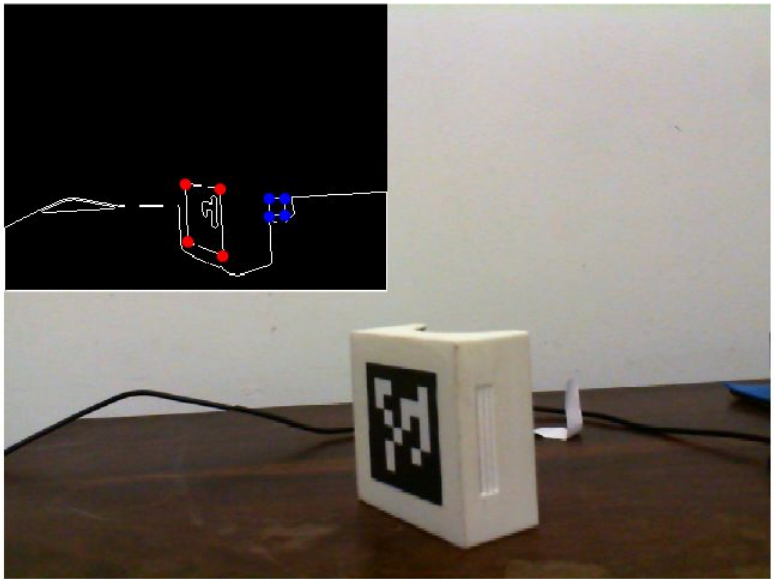
```
ImSeg =

     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     1     1     1    -1     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     1     0     0    -1     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     1     0     0    -1     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     1     1     1    -1     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     2     2     2     2    -2     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     2     0     0     0    -2     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     2     0     0     0    -2     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     2     2     2     2    -2     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
```

Filtragem dos Objetos : : Algoritmo de Douglas Peucker

Resultado da segmentação:

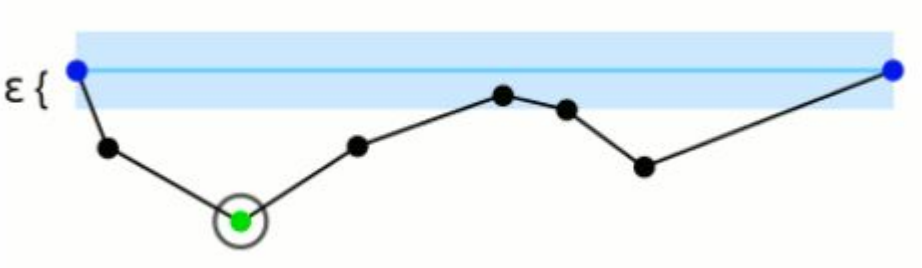


Resultado da Filtragem:



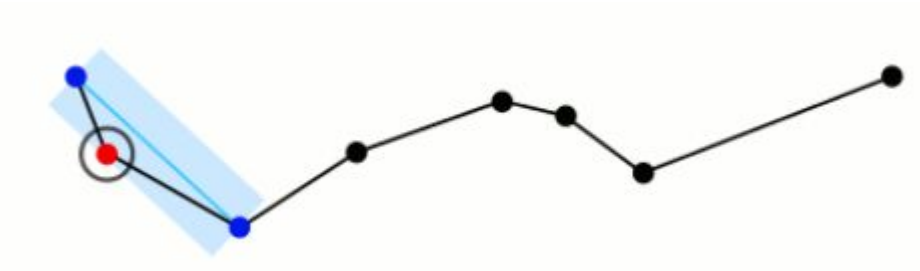
Filtragem dos Objetos : : Algoritmo de Douglas Peucker

- Usado para simplificação de curvas
- Algoritmo recursivo com apenas 1 (um) parâmetro: ε



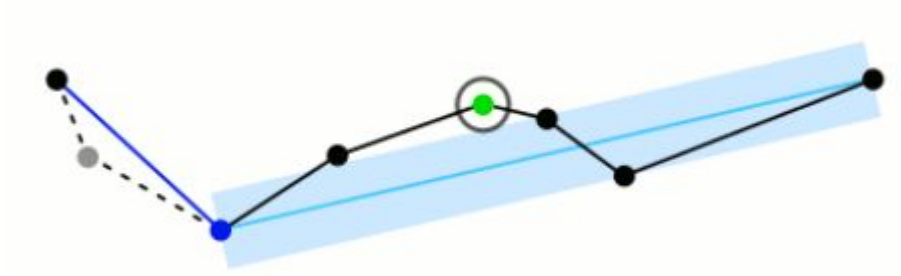
Filtragem dos Objetos : : Algoritmo de Douglas Peucker

- Usado para simplificação de curvas
- Algoritmo recursivo com apenas 1 (um) parâmetro: **ε**



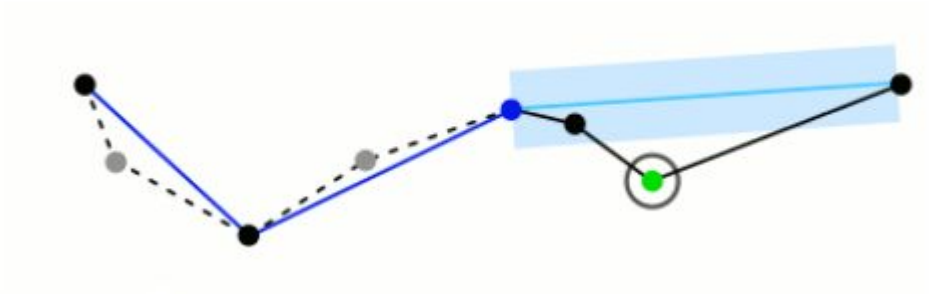
Filtragem dos Objetos : : Algoritmo de Douglas Peucker

- Usado para simplificação de curvas
- Algoritmo recursivo com apenas 1 (um) parâmetro: ε



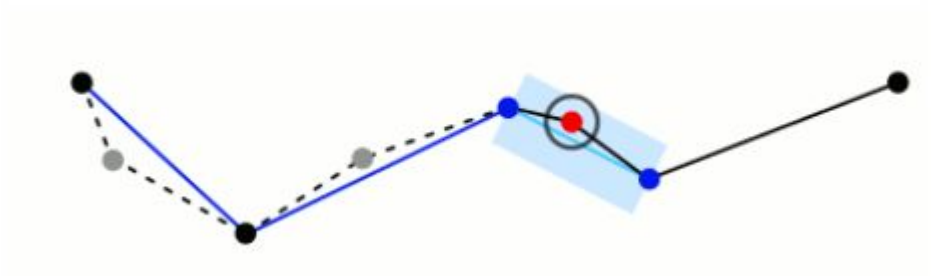
Filtragem dos Objetos : : Algoritmo de Douglas Peucker

- Usado para simplificação de curvas
- Algoritmo recursivo com apenas 1 (um) parâmetro: **ε**



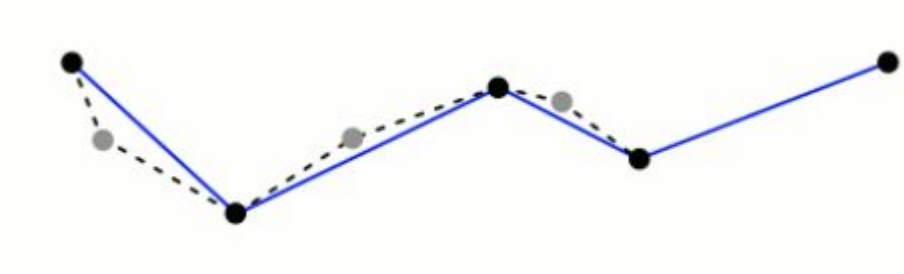
Filtragem dos Objetos : : Algoritmo de Douglas Peucker

- Usado para simplificação de curvas
- Algoritmo recursivo com apenas 1 (um) parâmetro: **ε**



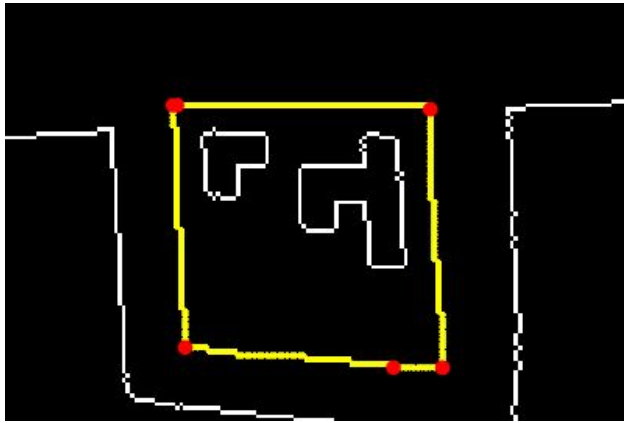
Filtragem dos Objetos : : Algoritmo de Douglas Peucker

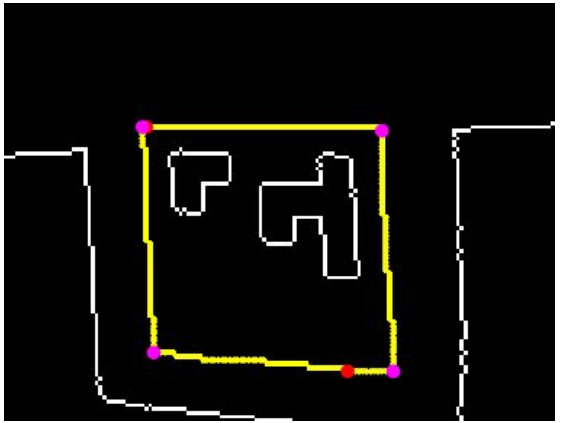
- Usado para simplificação de curvas
- Algoritmo recursivo com apenas 1 (um) parâmetro: ε



Filtragem dos Objetos : : Algoritmo de Douglas Peucker

- Pós processamento:

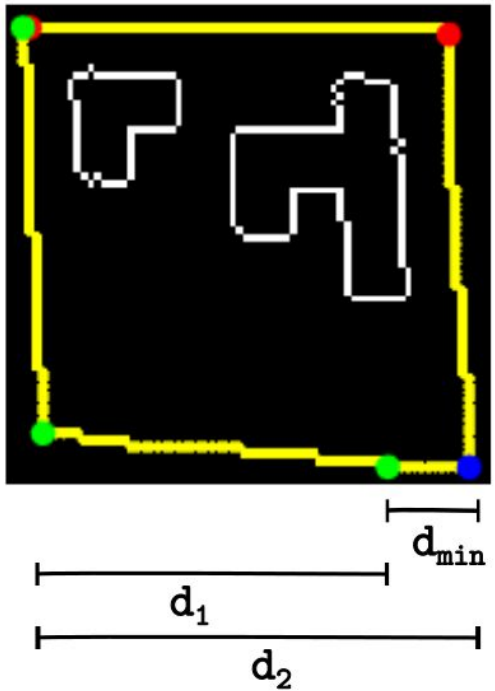


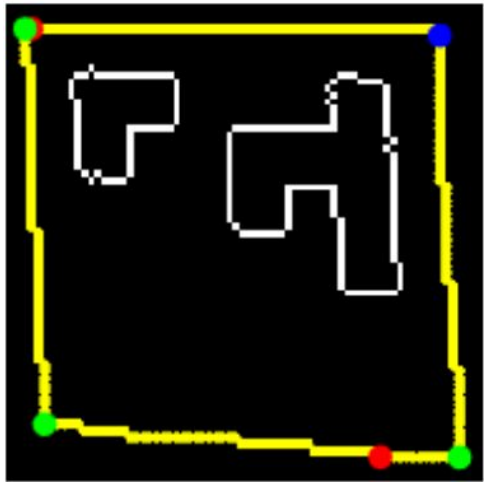


Filtragem dos Objetos : : Algoritmo de Pós-Processamento

Filtragem dos Objetos : : Algoritmo de Pós-Processamento

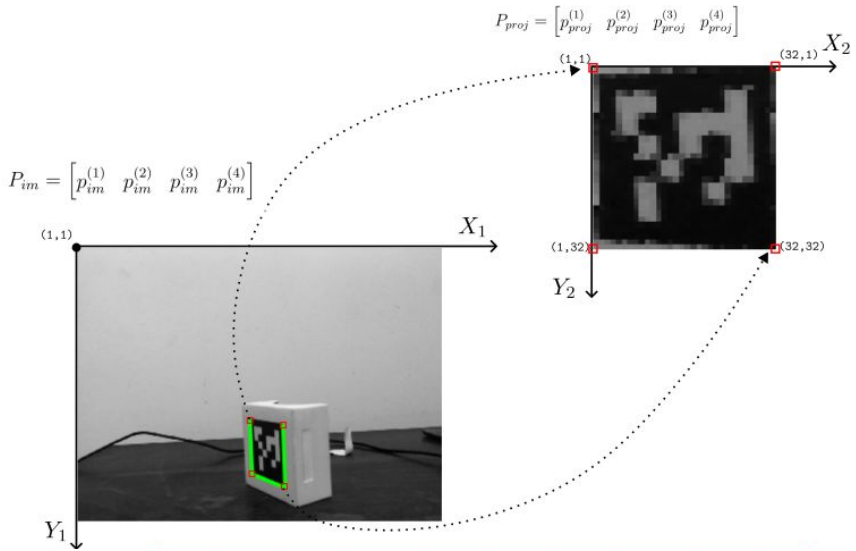
Se d_{min} < limiar **então**
  Se $d_2 > d_1$ **então**
    marca p4 no lugar de p3



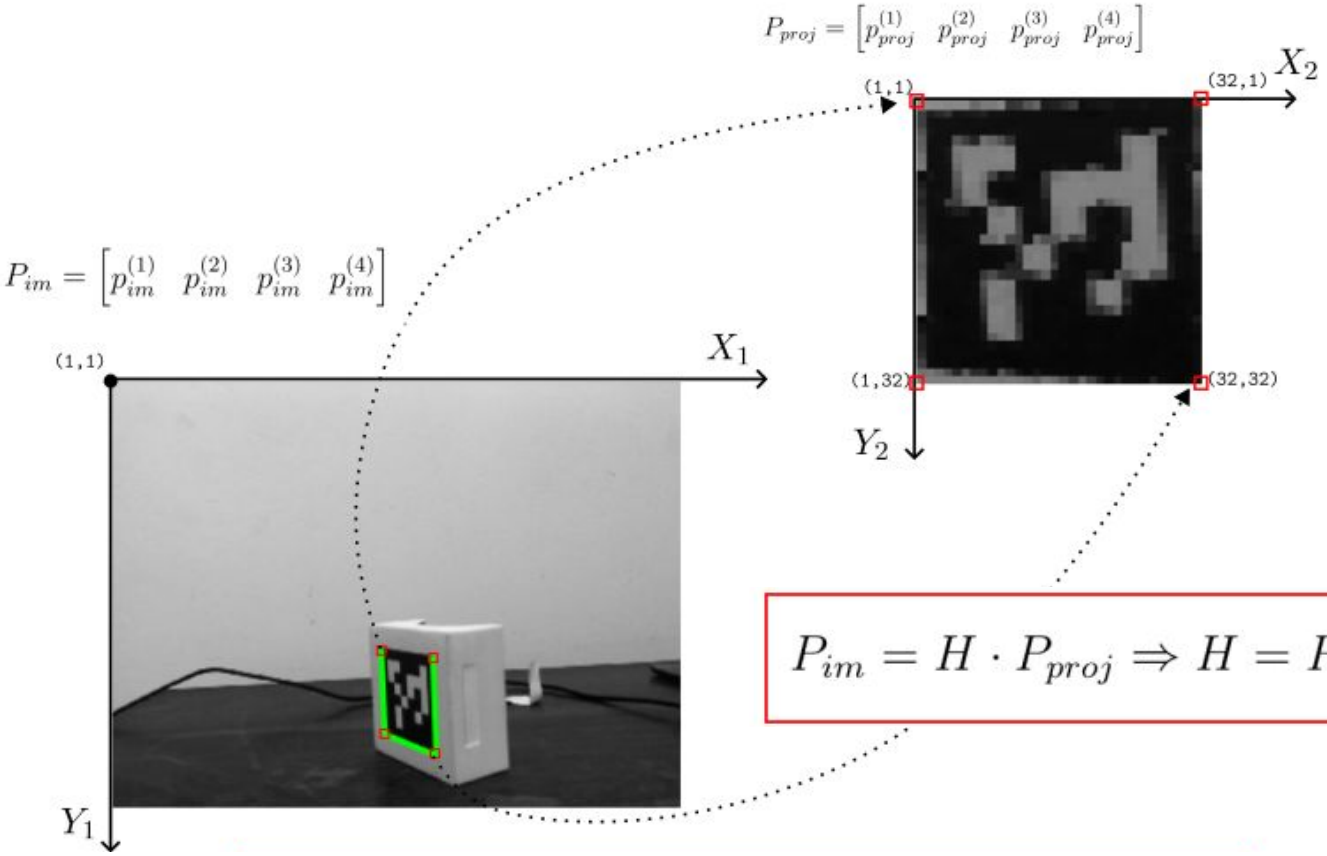


Transformação Geométrica

- A transformação é obtida a partir da **imagem redimensionada** , da pirâmide de imagens, em cujo o perímetro do marcador é mais próximo de 4x32 pixels.

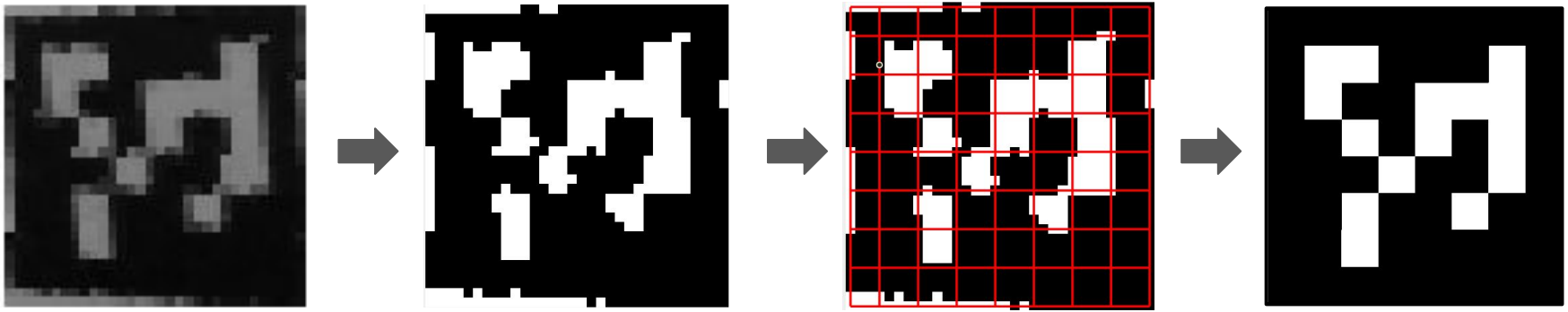


Transformação Geométrica



$$P_{im} = H \cdot P_{proj} \Rightarrow H = P_{im} P_{proj}^T (P_{proj} P_{proj}^T)^{-1}$$

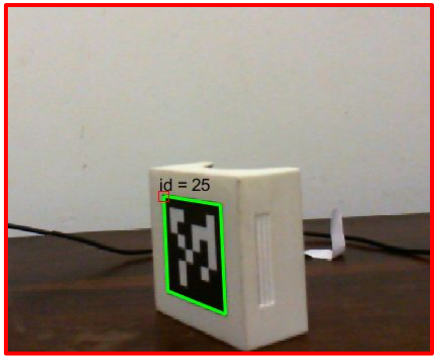
Obtenção do código



32x32 32x32 32x32 8x8

Comparação com o dicionário

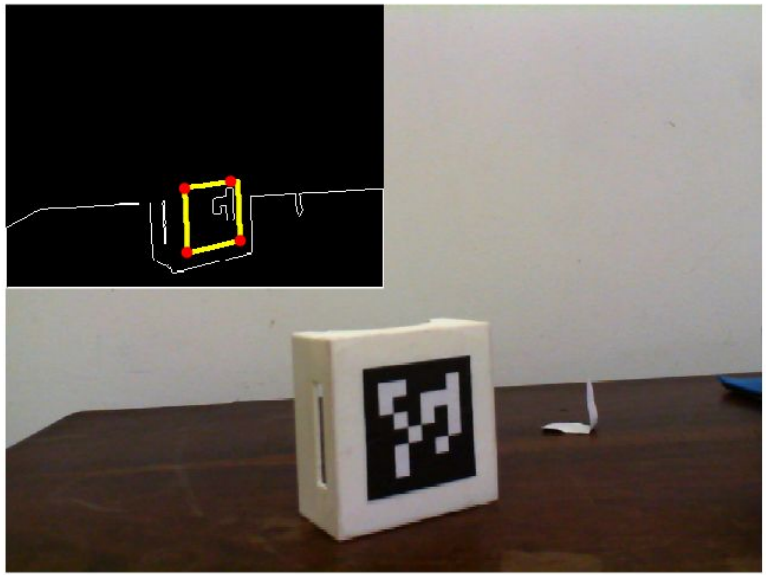
- Distância de Edição
- A comparação considerando 4 orientações diferentes.
- Um limiar de distância é usado para classificar o marcador.
- limiar=4.



Algumas Imagens presentes no dicionário usado

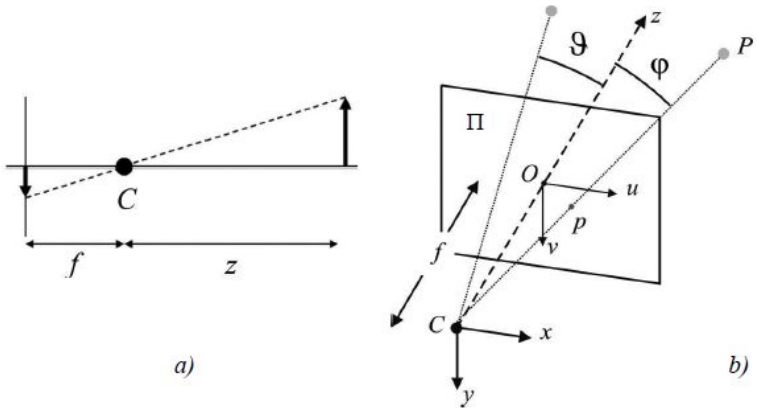
Refinamento da Estimativa dos Cantos

- Itera sobre a pirâmide de imagem, e obtém novas estimativas para os cantos do marcador.
- Detector de Harris numa janela de pixels em torno de cada canto.



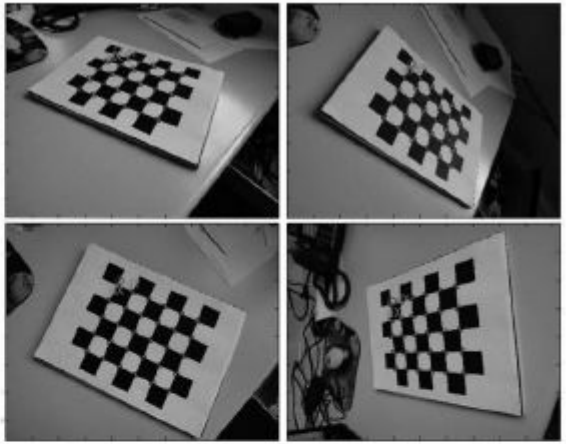
Estimação da Pose

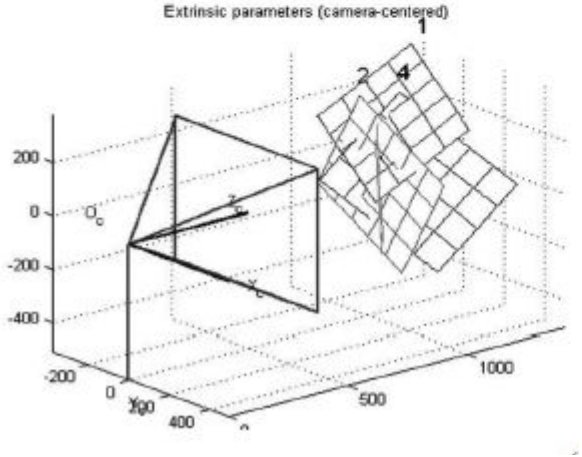
- Modelo *pinhole*



$$\frac{f}{z} = \frac{u-u_0}{x} = \frac{v-v_0}{y}$$

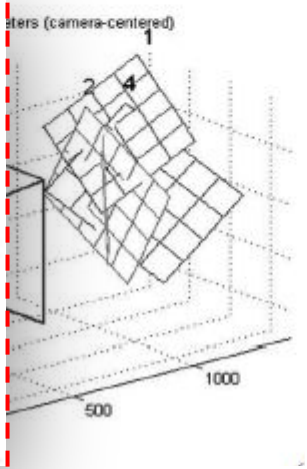
Estimação da Pose : : Calibração da Câmera



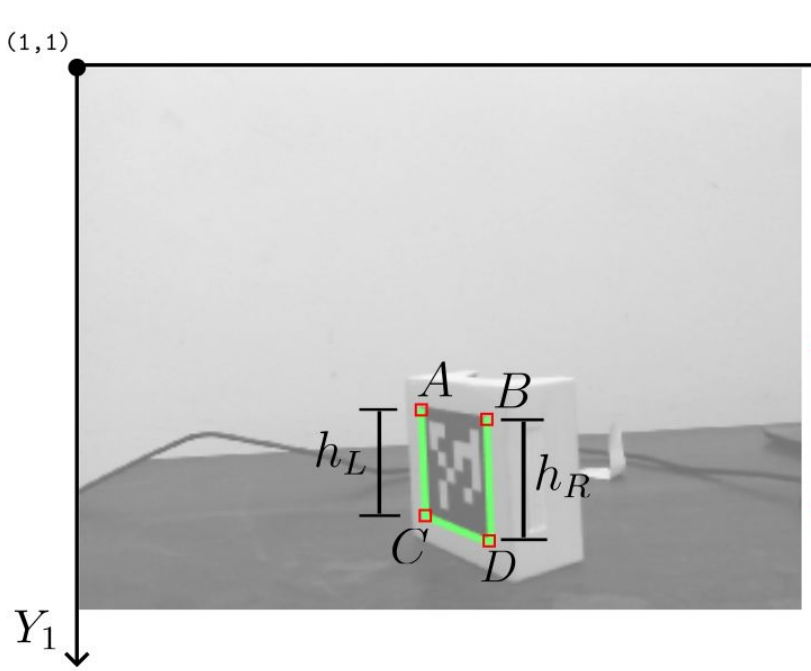


Estimação da Pose : : Calibração da Câmera

$$\begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix},$$



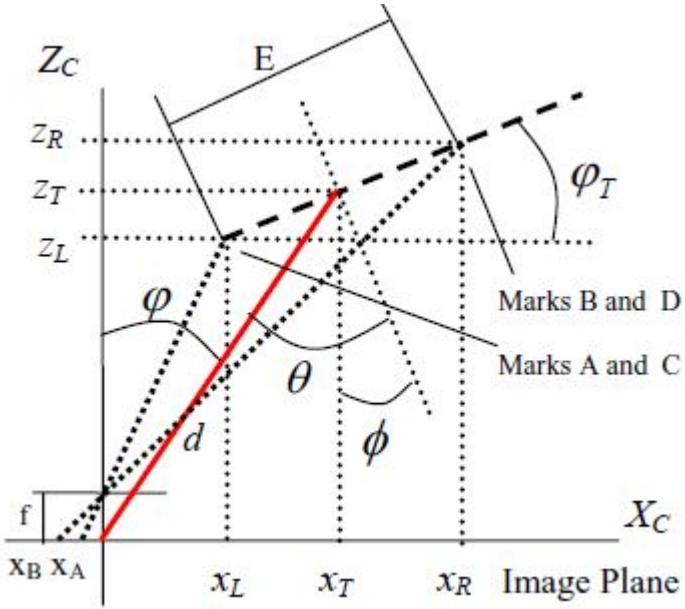
Estimação da Pose

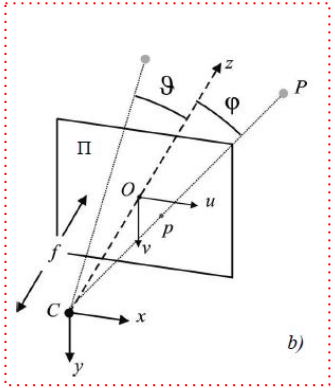


$$z_L = f \cdot \frac{E}{hL}$$
$$z_R = f \cdot \frac{E}{hR}$$

$$x_L = \frac{z_L}{f} \cdot (x_A - u_0)$$
$$x_R = \frac{z_R}{f} \cdot (x_B - u_0)$$

Estimação da Pose





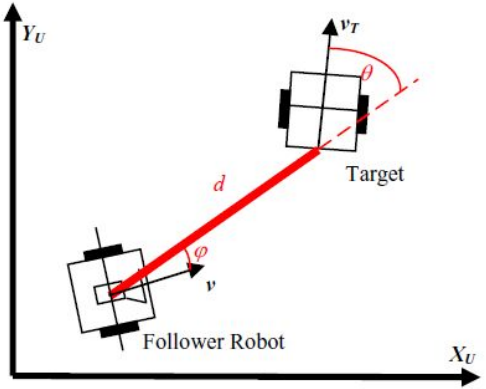
Equações:

$$\phi = \tan^{-1}\left(\frac{z_R - z_L}{x_R - x_L}\right)$$

$$\varphi = \tan^{-1}\left(\frac{x_T}{z_T}\right)$$

$$\theta = \varphi + \phi$$

$$d = \sqrt{x_T^2 + z_T^2}$$



Conclusões

- A detecção de marcadores encontra grande aplicabilidade em robótica, sendo potencialmente mais robusta que a detecção por cores.
- Implementação em MatLab com processamento em torno de 7 quadros por segundo.
- Uma implementação em C é mais viável, sob o ponto de vista prático.

Obrigado!