

**UNIVERSIDADE FEDERAL DE SANTA CATARINA  
CAMPUS BLUMENAU**

# **Trabalho I**

Aluno      Eduardo Mafra Pereira

Professores    Leonardo Mejia Rincon  
                  Marcos Vinicius Matsuo

Blumenau, Junho de 2019

# **Conteúdo**

<b>1</b>	<b>Identificação da proposta</b>	<b>1</b>
<b>2</b>	<b>Introdução</b>	<b>2</b>
<b>3</b>	<b>Desenvolvimento</b>	<b>3</b>
3.1	Algoritmo . . . . .	3
3.1.1	Algoritmo das referências . . . . .	3
3.1.2	Algoritmo Principal . . . . .	4
<b>4</b>	<b>Resultados</b>	<b>19</b>
<b>5</b>	<b>Referências</b>	<b>23</b>

# 1 Identificação da proposta

Implemente no Matlab um programa capaz de reconhecer cartas de um baralho. Neste trabalho será considerado o baralho clássico composto por 52 cartas, onde cada carta é diferenciada através do seu naipe e de seu valor. O trabalho está dividido em quatro níveis de dificuldade, a saber:

1. No primeiro nível, o programa deve ser capaz de reconhecer as cartas fornecidas individualmente. Ou seja, com cada imagem fornecida ao programa contendo apenas uma carta (conforme ilustrado na Figura 1).
2. No segundo nível de dificuldade, o programa deve reconhecer todas as cartas presentes em uma imagem, com cartas orientadas em posição vertical e sem variações de escalas (conforme ilustrado na Figura 2).
3. No terceiro nível, o programa deve ser capaz de reconhecer todas as cartas presentes em uma imagem, com cartas apresentando orientações e escalas variadas (conforme ilustrado na Figura 3).
4. No quarto nível de dificuldade, o programa deve ser capaz de reconhecer cartas obtidas via webcam ou câmera equivalente.



Figura 1: Exemplo 1. Fonte: Profº Marcos Vinicius Matsuo.



Figura 2: Exemplo 2. Fonte: Profº Marcos Vinicius Matsuo.

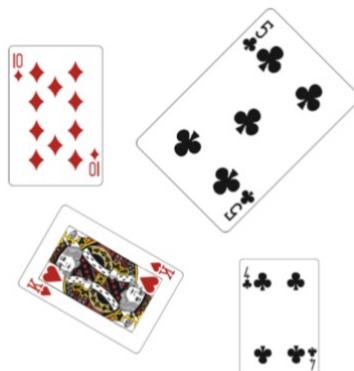


Figura 3: Exemplo 3. Fonte: Profº Marcos Vinicius Matsuo.

## 2 Introdução

Para realizar os objetivos propostos foi construído um algoritmo que realiza a homografia de todas as cartas presentes em uma imagem e as detecta. Este algoritmo foi desenvolvido utilizando o software de programação Matlab.

Na realização da homografia de cada carta foi utilizada a transformada “Hough”(que possibilitou encontrar os pontos periféricos das cartas) e as funções “fitgeotrans” e “imwarp”. Já para a detecção das cartas é aplicada a técnica de “tamplate matching - SAD”.

O algoritmo desenvolvido atende os 3 primeiros níveis com excelência, o nível 4 não foi realizado neste trabalho.

## 3 Desenvolvimento

Nesta seção será apresentado o algoritmo descrevendo todas suas etapas de processamento.

### 3.1 Algoritmo

#### 3.1.1 Algoritmo das referências

O algoritmo utiliza a técnica de “tamplate matching”, comparando as referências com as formas encontradas nas cartas. Para isto foi necessário processar as imagens de referências, padronizando suas dimensões (altura e largura). Este processo é extremamente necessário para que o “tamplate matching” seja realizado com sucesso. O processamento das referências é realizado em um código separado e só precisa ser executado uma única vez, pois utiliza-se o comando “save” para salvar as imagens de referência após o processamento e a padronização. Todas as imagens devem ser salvas em preto e branco.

As figuras 4 à 6 representam o algoritmo realizado para processar os naipes, os números e as letras a serem utilizadas para a comparação. O restante deste algoritmo será enviado aos orientadores junto ao programa principal. Estas figuras contém o processo de padronização realizado para o naipe de “espadas”, pois este processo se repetirá para o restante das referências.

As referências são transformadas em imagens em preto e branco, no qual é extraída da imagem apenas a localização do elemento de interesse (naipe, número ou letra), através de sua ”bounding box”. Após é criada uma nova imagem contendo apenas este elemento de interesse, ela é redimensionada com as dimensões presentes nas variáveis “tanY” e “tamX” através da função “imresize” e finalmente este novo corte é salvo utilizando a função “save”.

```
% Código nipes:
tamY = 24; % Tamanho de comparação padronizados
tamX = 21;
% Espadas:

espada = imread('espada.png', 'double');
espada = rgb2gray(espada);
espadal = otsu(espada);
espadabw = not(espada >= spadal);
[esp_label, m, ~, cls] = ilabel(espadabw);
prop = regionprops(esp_label,'ConvexArea','BoundingBox');
j = 0;
indice = [];
idisp(espadabw);
for i = 1:m
    b = cls(i)
    if 50 < prop(i).ConvexArea < 900 && b == 1
        j = j+1;
        indice = i;
    end
end
a = prop(indice).BoundingBox
xMin = a(1)
```

Figura 4: Algoritmo das Referências.

```
xMax = xMin + a(3)
yMin = a(2)
yMax = yMin + a(4)
hold on
plot_box(xMin,yMin, xMax,yMax, 'r')

espadabw = spadabw(yMin - 1: yMax, xMin - 1 : xMax);
espadabw = imresize(espadabw, [tamY,tamX]);
```

Figura 5: Algoritmo das Referências.

```
save ('espadabw');
save ('copasbw');
save ('ourobw');
save ('pausbw');
```

Figura 6: Algoritmo das Referências.

### 3.1.2 Algoritmo Principal

O Algoritmo desenvolvido utilizará a técnica de “tamplete matching” com as referências já processadas, para isto é preciso realizar alguns métodos anteriores de processamento de imagem para garantir que a comparação seja realizada com sucesso.

Inicialmente são carregadas as imagens de referências através da função “load”, estas matrizes serão tratadas ao decorrer do algoritmo. As referências são carregadas no início do algoritmo para evitar que sejam carregadas mais de uma vez ao longo do código.

```
clear all; clc;
close all;
warning('off');

% Referências para os naipes:
tamYa = 24;
tamXa = 21;

load('espadabw.mat');
load('copasbw.mat');
load('ourobw.mat');
load('pausbw.mat');

% Referências para os numeros (ou letras):

tamYl = 30;
tamXl = 20;
|
load('asbw');
load('doisbw');
load('tresbw');
load('quatrobw');
load('cincobw');
```

Figura 7: Carregando as referências.

```
load('seisbw');
load('setebw');
load('oitobw');
load('novebw');
load('vatebw');
load('damabw');
load('reibw');
```

Figura 8: Carregando as referências.

Posteriormente, dois vetores de “strings” são criados, estas “strings” contém os nomes das cartas e naipes, e serão utilizados ao longo do algoritmo para mostrar no “Command Window” do software de programação Matlab o nome e o naipe da carta detectada. Vale ressaltar que não há uma “string” referente ao valor 10, a explicação será apresentada no decorrer desse relatório.

```
% Strings para os printf

str = ["Ás", "Dois", "Três", "Quatro", "Cinco", "Seis", ...
"Sete", "Oito", "Nove", "Valete", "Dama", "Rei"];
strl = ["espada", "copas", "ouro", "paus"];
```

Figura 9: ”Strings” para nomeação das cartas.

Após carregadas as referências e definidas as “strings” para nomear as cartas, o processamento da imagem principal é iniciado. Primeiramente é capturada a imagem a ser processada por meio da função “imread”, esta imagem é transformada para a escala de cinza e para a escala “preto e branco”, após passa-se um filtro de suavização a fim de evitar que os pixels fiquem isolados dos objetos. As cartas presentes na imagem “preto e branco” são preenchidas através do comando “bwconvhull”, este processo de preenchimento é necessário para que ao capturar as ”labels” da imagem, sejam encontrados apenas o número de objetos referentes as cartas, se esta função não fosse utilizada, uma variedade de objetos seriam encontrados na imagem, e estes não corresponderiam necessariamente as cartas. Por fim, são determinadas as ”labels” para cada carta.

```
% Inicio código:

im = imread('fig5_n3.png', 'double');
figure(); idisp(im);

tic % Inicia a contagem do tempo de processamento do código;

img = rgb2gray(im); % Converte a imagem para escala de cinza;
imbw = not((img >= 200/255)); % imagem negada;
Se = ones(2); % Máscara de suavização;
Imbw = imdilate(imbw, Se); % Filtro de Suavização;
preençimento = bwconvhull(Imbw, 'objects'); % Imagem com cartas preenchidas;
[lab, m, ~, cls] = ilabel(preençimento); % Aplica Label nas cartas;
```

Figura 10: Tratamento da imagem principal.

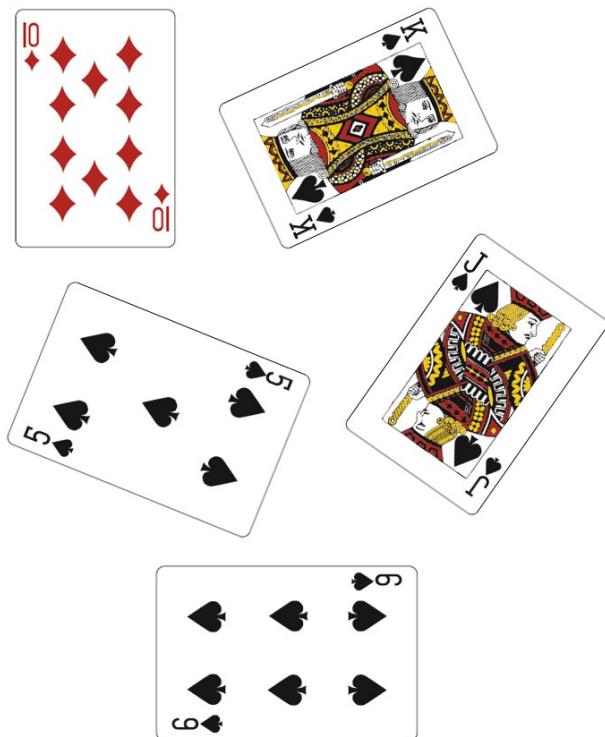


Figura 11: Imagem utilizada.

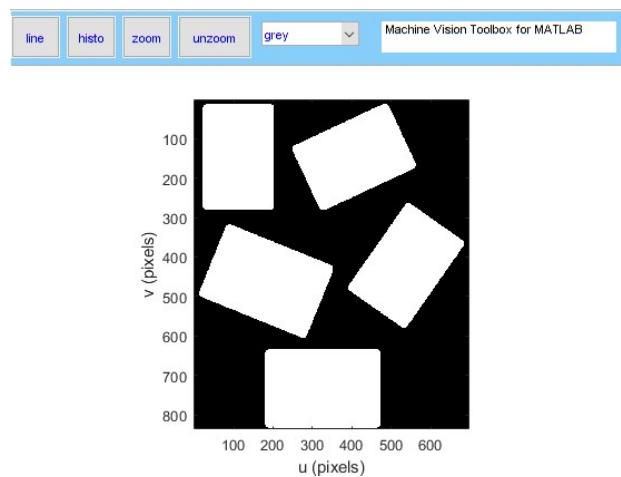


Figura 12: Imagem utilizada.

Com as “labels” de cada carta determinadas, as detecções podem ser iniciadas, estas serão processadas individualmente através de um laço de repetição, ou seja, a cada laço uma nova carta será processada até que o número

de repetições se iguale ao número de objetos da imagem.

```
%% Tratamento das cartas (uma por uma):
for p = 1:m % Laço para percorrer todas as cartas do baralho;
    if cls(p,1) == 1 % Verifica se é carta ou fundo;
        Im = (lbl==p); % Tratamento da carta com label == p
        %figure; idisp(Im);
```

Figura 13: Laço de repetição.

Como o algoritmo foi desenvolvido para atender os 3 níveis de dificuldade propostos é necessário realizar a homografia a cada laço, não importando para qual nível de dificuldade esta sendo tratado. Com o objetivo de realizar a homografia, a carta é recortada da imagem principal criando duas novas imagens que contém apenas a carta processada, sendo um recorte realizado na imagem RGB e outro na imagem após o preenchimento. O corte RGB será utilizado apenas para aplicar na homografia e proporcionar uma imagem RGB da carta redirecionada, já o corte do preenchimento será utilizado para extrair os vértices da carta.

Para o processo de extração dos vértices através da transformada “Hough”, é necessário aplicar inicialmente um filtro de detecção de bordas, com o intuito de limitar as linhas geradas a partir da transformada, obtendo assim apenas as linhas de perspectiva externa da carta.

```
% Separa as cartas:
[x,y] = find(Im); % Posição das cartas;
xmin = min(x);
xmax = max(x);
ymin = min(y);
ymax = max(y);
% Corte na imagem contendo apenas as cartas:
ibw = Im(xmin: xmax, ymin : ymax); % Carta preto e branco;
irgb = im(xmin: xmax, ymin : ymax, :); % Carta RGB;

exp =100; % Padroniza margem de cada carta;

% Carta para extrair os corners:
Inova = double(zeros(size(ibw,1) + exp, size(ibw,2) + exp));
Inova(exp/2: exp/2 + size(ibw,1) - 1, exp/2 : exp/2 + size(ibw,2) - 1) = ibw;

% Carta onde é realizada a homografia:
InRGB = double(ones(size(irgb,1) + exp, size(irgb,2) + exp, 3));
InRGB(exp/2 : exp/2 + size(irgb,1) - 1, exp/2 : exp/2 + size(irgb,2) - 1, :) = irgb;
```

Figura 14: Corte das cartas.

```

edges = icanny(Inova); % Filtro de detecção de bordas;
Inova = bwconvhull(edges); % realiza precimento da imagem ;
edges = icanny(double(Inova)); % filtro de detecção de bordas;

Se = ones(3);
edges = idilate(edges, Se); % Engrossar as bordas para utilizar a
% transformada Hough;
I2 = edges;
h = Hough(edges,'suppress',10);
linhas = h.lines(); % linhas de perspectiva;
% hold on;
% linhas.plot;
linhas2 = linhas.seglength(I2); % linhas de perspectiva (com informação adicional);
% figure();idisp(InRGB);
% linhas2.plot('g');

```

Figura 15: Transformada "Hough".

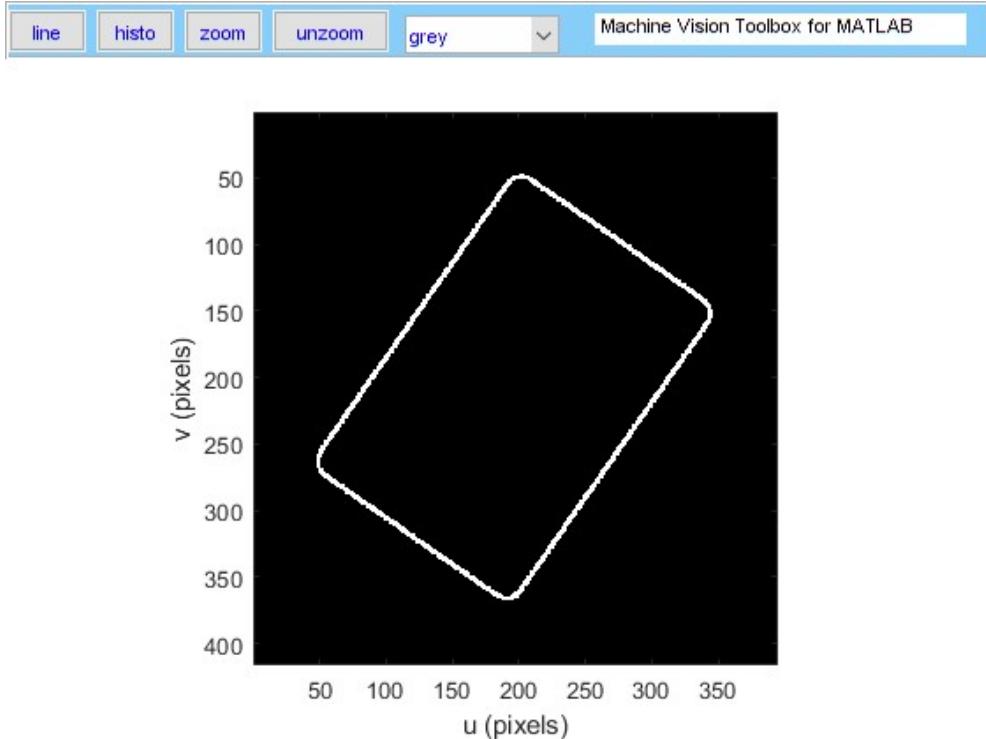


Figura 16: Detecção de Bordas.

Os pontos de interesse ("corners") de cada carta são os pontos de intersecção entre as linhas obtidas através da transformada "Hough". Suas coordenadas podem ser encontradas utilizando os laços de repetição da figura 18. Nestes laços os pontos de intersecção são atribuídos a uma matriz, na qual a primeira coluna representa as coordenadas pertencentes ao eixo das abscissas da imagem recortada e a segunda representa as coordenadas pertencentes ao eixo das ordenadas.

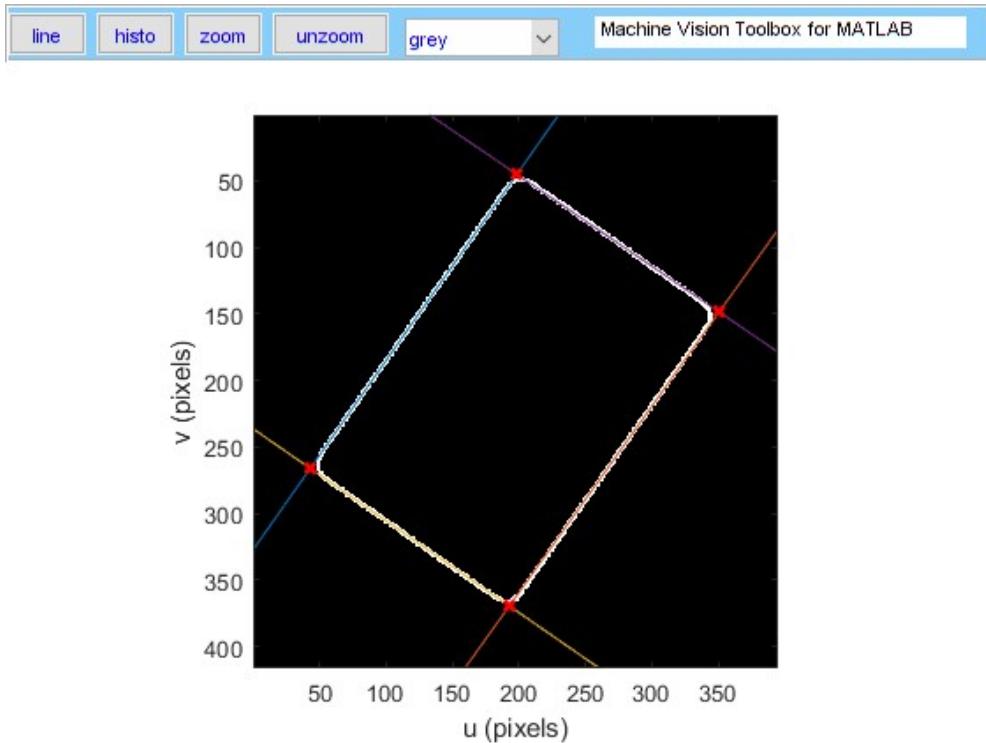


Figura 17: Linhas obtidas através da transformada "Hough".

```
%> Obtenção dos corners:
[m,n] = size(linhas2);
k = 0;
a = 0;
for i = 1:n
    for j = i+1:n
        k=k+1;
        u(k) = ((linhas2(i).rho/cos(linhas2(i).theta)) - ...
                  (linhas2(j).rho/cos(linhas2(j).theta)))/...
                  (tan(linhas2(i).theta)-tan(linhas2(j).theta));
        v(k) = -u(k)*tan(linhas2(j).theta) + ...
                  (linhas2(j).rho/cos(linhas2(j).theta));
        if u(k)>0 && v(k)>0 && u(k)<1000 && v(k)<1000
            a=a+1;
            M(a,1)=u(k); % Pontos de interseção;
            M(a,2)=v(k);
        end
    end
end
```

Figura 18: Obtenção dos "corners".

Com os pontos de interesses obtidos, é necessário ordena-los antes serem utilizados no processo de homografia. Sendo assim, inicialmente foi ordenada a Matriz M do menor elemento da segunda coluna para o maior elemento, utilizando a função “sortrows”, isto possibilita identificar quais são os elementos superiores e inferiores dos ”corners”, porém ainda é necessário descobrir quais são os elementos da esquerda e da direita. Para isto é calculada a localização do centroide da carta preenchida através dos momentos do objeto processado, os elementos que estiverem a esquerda do centroide serão portanto os elementos da esquerda e os que estiverem a direita serão os elementos da direita. Isto possibilita padronizar a ordem dos pontos para que estes possam ser aplicados sempre da mesma maneira, quando realizada a homografia.

```
% Ordenando os corners:

M = sortrows(M,2); % Ordena pontos do menor v para o maior v.
% portanto M(1,1) será o ponto mais acima (basta saber se esta na esquerda
% ou na direita).

m00 = mpq(Inova, 0, 0);
m10 = mpq(Inova, 1, 0);
m01 = mpq(Inova, 0, 1);
centro = [m10/m00, m01/m00]; % Centroide da carta;

j=0;
for i = 2:size(M,1) % Percorre os outros 3 corners;
    if M(i,2)<centro(2) % Procura o outro ponto acima do centroide;
        if M(1,1)<M(i,1)
            conjuntol = [M(1,1),M(1,2);M(i,1),M(i,2)];
        end
        if M(1,1)>M(i,1)
            conjuntol = [M(i,1),M(i,2);M(1,1),M(1,2)];
        end
    end
end
```

Figura 19: Ordenação dos ”corners”.

```

    else
        j = j+1;
        baixo(j) = i; % Pontos restante são os ponto de baixo.
    end
end

if M(baixo(1),1) < M(baixo(2),1) % Verifica entre os pontos de baixo
    % qual é o da esquerda e o da direita.
    conjunto2 = [M(baixo(1),1),M(baixo(1),2);M(baixo(2),1),M(baixo(2),2)];
end
if M(baixo(1),1) > M(baixo(2),1)
    conjunto2 = [M(baixo(2),1),M(baixo(2),2);M(baixo(1),1),M(baixo(1),2)];
end

```

Figura 20: Ordenação dos ”corners”.

O Algoritmo presente nas figuras 19 e 20 não foi pensado para as cartas que se encontram deitadas, e para atender este tipo de condição foi necessário realocar os pontos ordenados anteriormente. Para identificar se há necessidade de realocar os pontos, é criada uma condição que verifica se as distâncias dos pontos superiores, são maiores que as distâncias entre os pontos da esquerda, caso esta condição for verdadeira haverá a necessidade de realocar a ordem atribuída aos ”corners”.

```

% Verifica se a distância entre os pontos de cima são maiores que a
% distância entre o primeiro ponto e o ponto de baixo da esquerda (caso
% a resposta seja verdadeira, uma inversão nos corners será feita):
if norm(conjunto1(1,:)-conjunto1(2,:)) > norm(conjunto1(1,:)-conjunto2(1,:))
    novo = conjunto1(1,:);
    conjunto1(1,:) = conjunto1(2,:);
    conjunto1(2,:) = conjunto2(2,:);
    conjunto2(2,:) = conjunto2(1,:);
    conjunto2(1,:) = novo;
end

```

Figura 21: Reordenando os ”corners”.

Os pontos ordenados são atribuídos a uma nova matriz, com o propósito de organizar estes pontos, e são utilizados para realizar a homografia efetivamente. Neste código foi necessário padronizar as dimensões da imagem gerada a partir do processo de homografia, isto porque o canto superior esquerdo da imagem gerada será tratado e o recorte deste elemento emprega sempre uma janela de dimensões fixas, ou seja, se o tamanho da homografia diferenciar a cada carta, a janela pode não contemplar os elementos de interesse (naipe e número ou naipe e letra). Destaca-se que apesar da imagem com preenchimento ser utilizada para encontrar os pontos de interesse, é a imagem RGB que é atribuída a homografia.

```

%% Realiza a Homografia:
B(1,:) = conjunto1(1,:); % Conjuntos atribuidos a uma matriz B.
B(2,:) = conjunto1(2,:);
B(3,:) = conjunto2(1,:);
B(4,:) = conjunto2(2,:);

x1=[B(4,1);B(3,1);B(2,1);B(1,1)];% Pontos para a realização da homografia
y1=[B(4,2);B(3,2);B(2,2);B(1,2)];

% Padroniza o tamanho das imagens após a homografia:
dimX = 260;
dimY = dimX*0.70;
x2 = [1; dimY; 1; dimY];
y2 = [1; 1; dimX; dimX];

% Realiza a homografia das imagens:
T = fitgeotrans([x1 y1], [x2 y2], 'projective');
Im2= imwarp(InRGB,T,'OutputView',imref2d(size(InRGB)));

```

Figura 22: Aplicando a homografia.

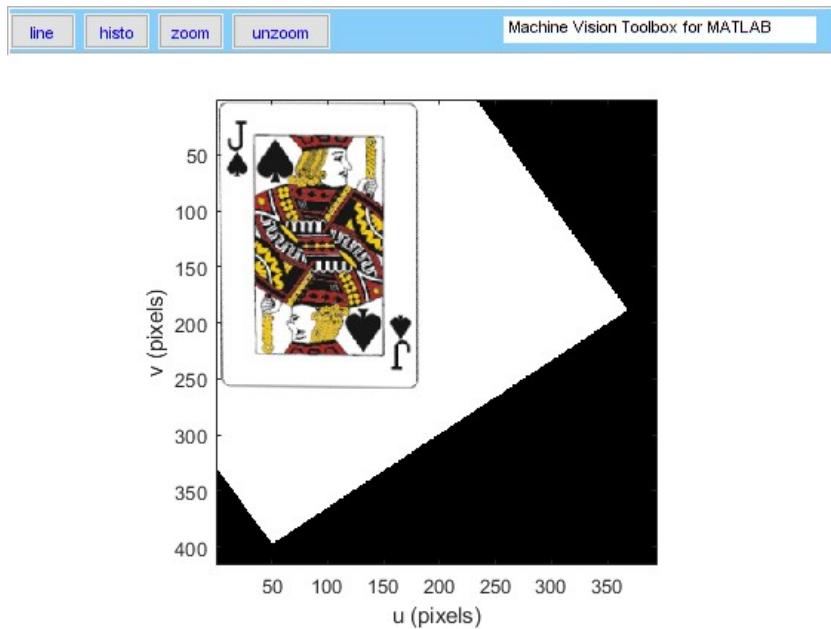


Figura 23: Resultado após homografia.

Com a homografia concluída, inicia-se a fase de detecção das cartas na prática. Nesta etapa é onde utiliza-se finalmente o "tamplate matching", para concluir qual o nome e o naipe da carta tratada. Porém antes de realizar a comparação por "tamplate matching" entre a carta e as referências,

é necessário tratar a imagem obtida e extrair apenas os objetos de interesse. Sendo assim, inicialmente é realizado um corte contendo apenas o canto inferior esquerdo desta imagem, este canto sempre deverá conter os objetos de interesse, devido a padronização das cartas após a homografia. O corte obtido é transformado para escala de cinza. Com a escala de cinzas obtida utiliza-se a função “otsu” para suavizar e melhorar os objetos contidos na imagem, além de realizar a transformação para imagem binária. A imagem em preto e branco é negada com a finalidade de transformar os objetos encontrados na imagem em branco e o fundo em preto, e por fim são atribuídas ”labels” aos objetos pertencentes a imagem negada.

---

#### **% Seção de Detecção:**

```
corte = Im2(16: 80, 7 : 35,:); % Corte nas imagem após a homografia;
cortel = rgb2gray(corte);
corte2 = otsu(cortel); % Suaviza a imagem
cortebw = not(cortel >= corte2);
[im_label, m, ~, cls1] = ilabel(cortebw); % Aplica labels aos objetos
% encontrados no corte;
```

Figura 24: Região de interesse.

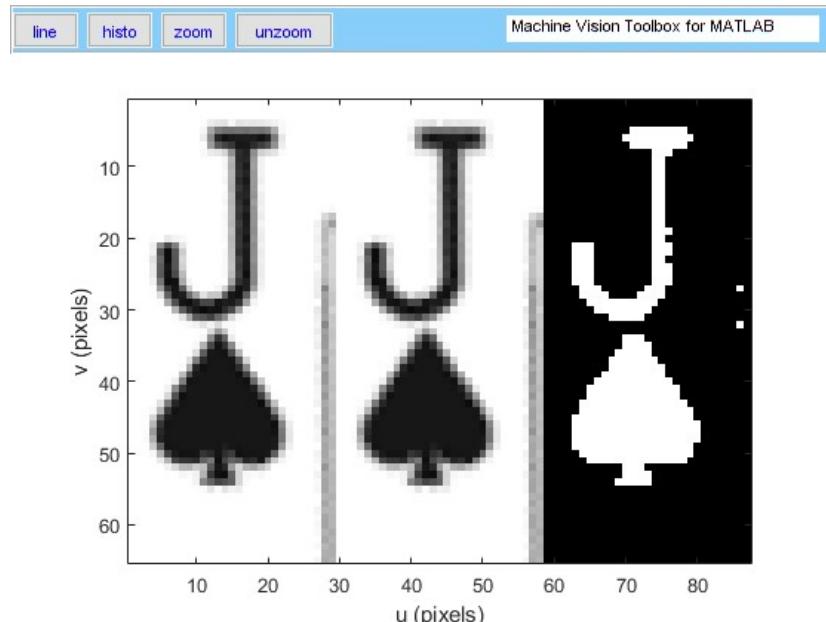


Figura 25: Corte gerado para as 3 escalas.

Com o corte gerado deve haver uma separação dos naipes e dos números, ou letras, para que os objetos de interesse possam ser comparados as referências de forma separada. Sendo assim, foi utilizada a função “regionprops”, em que esta possibilita identificar quantos objetos há na imagem, bem como os “boundingboxs” de cada objeto e quais serão suas áreas. Logo após, é criado um laço de repetição para obter apenas os elementos em que a área possui um tamanho de intervalo determinado, eliminando desta forma possíveis ruídos e o fundo, com o objetivo de não serem encontrados como objetos.

```

prop = regionprops(im_label,'Area','BoundingBox'); % Extrai propriedades dos objetos
% encontrados.

indice=[];
j = 0;
for i = 1:m
    a = prop(i).Area;
    b = cls1(i);
    if a > 40 && a < 500 && b==1
        j = j+1;
        indice(j) = i; % Seleciona apenas o numero (ou letra) e o naipe.
    end
end
% idisp(cortebw)

```

Figura 26: Encontrando os objetos de interesse.

Como mencionado anteriormente, não há nenhum elemento na “string” correspondente a carta de número 10, isto porque a carta de número 10 será a única carta que gera 3 elementos encontrados e não apenas 2. Sendo assim, para a carta de número 10 não há necessidade de aplicar o ”template matching” no número da carta, pois basta saber quantos elementos serão encontrados no corte superior. Se o número de elementos for igual a 3, a carta será definida como sendo de valor 10 e então é realizada apenas a comparação do naipe. No algoritmo portanto, haverá uma condição para determinar se a carta é a carta de número 10 ou não.

A comparação de naipes para as cartas de número 10 é feita utilizando o terceiro elemento encontrado, este necessariamente será o naipe partindo da premissa que o Matlab percorre uma imagem da esquerda para a direita, linha após linha. Sendo assim, é extraída a “boundingbox” deste terceiro elemento para identificar a posição do elemento na imagem, e partir da “boundingbox” é criada uma nova imagem contendo apenas o elemento presente nela com uma margem de 1 pixel para cada lado. Esta nova imagem é redimensionada seguindo exatamente o dimensionamento aplicado as referências para que o ”template matching” necessite de apenas uma única interação.

```

if size(indice,2) == 3 % É a carta 10;

    a = prop(indice(3)).BoundingBox; % Tratamento do naipe.

    xMin = a(1);
    xMax = xMin + a(3);
    yMin = a(2);
    yMax = yMin + a(4);

    %
    % hold on
    % plot_box(xMin,yMin, xMax,yMax,'b')

    % Recorta apenas o naipe e padroniza ele com o mesmo tamanho da
    % referência:
    naipe = cortebw(yMin - 1 : yMax, xMin - 1 : xMax);
    naipe = imresize(naipe,[tamYa,tamXa]);

```

Figura 27: Recorte e redimensionamento do naipe.

Com o naipe extraído e as referências redimensionadas é possível realizar o “tamplete matching”, o método utilizado foi o SAD e seus resultados foram atribuídos a um vetor de elementos. O valor de menor elemento representará o naipe da carta detectada e então é mostrado no “Command Window” qual elemento foi detectado, para isto é necessário extrair a posição do menor valor encontrado no vetor de elementos, pois esta posição indica a “string” a ser mostrada no vetor de “strings”.

```

% Template Matching (SAD):

nl = [];
nl(1) = sum((sum(sum((naipe(:,:,1) - espadabw(:,:,1)).^2)))); % Espada
nl(2) = sum((sum(sum((naipe(:,:,1) - copasbw(:,:,1)).^2)))); % Copas
nl(3) = sum((sum(sum((naipe(:,:,1) - ourobw(:,:,1)).^2)))); % Ouro
nl(4) = sum((sum(sum((naipe(:,:,1) - pausbw(:,:,1)).^2)))); % Paus

i = find(nl == min(nl)); % Menor valor será o naipe correto;

fprintf('10 de %s \n',str1(i)); % Print mensagem.

```

Figura 28: ”Tamplete Matching”.

O processo anterior repete-se para as cartas que não forem a carta de número 10, porém com algumas modificações. O naipe será agora o segundo elemento encontrado, e o primeiro elemento que será o índice da carta (letra ou número), também deverá ser redimensionado para passar por um “tamplate matching” de identificação.

```

else % Não é a carta 10;

    a = prop(indice(2)).BoundingBox; % Tratamento naipe;

    xMin = a(1);
    xMax = xMin + a(3);
    yMin = a(2);
    yMax = yMin + a(4);

    %
    % hold on
    % plot_box(xMin,yMin, xMax,yMax,'b')

    % Recorta apenas o naipe e padroniza ele com o mesmo tamanho da
    % referência:
    naipe = cortebw(yMin - 1 : yMax, xMin - 1 : xMax);
    naipe = imresize(naipe,[tamYa,tamXa]);

```

Figura 29: Redimensiona o naipe.



Figura 30: Naipe redimensionado.

```

% Template Matching (SAD):

n2 = [];
n2(1) = sum((sum(sum((naipe(:,:,1) - espadabw(:,:,1)).^2)))); % Espada
n2(2) = sum((sum(sum((naipe(:,:,1) - copasbw(:,:,1)).^2)))); % Copas
n2(3) = sum((sum(sum((naipe(:,:,1) - ourobw(:,:,1)).^2)))); % Ouro
n2(4) = sum((sum(sum((naipe(:,:,1) - pausbw(:,:,1)).^2)))); % Paus

n = find(n2 == min(n2)); % Menor valor será o naipe correto;

```

Figura 31: "Tamplete Matching".

```
% Detecção do numero (ou letra):

a = prop(indice(1)).BoundingBox; % Tratamento numero (ou letra);

xMin = a(1);
xMax = xMin + a(3);
yMin = a(2);
yMax = yMin + a(4) ;

% hold on
plot_box(xMin,yMin, xMax,yMax,'b')

% Recorta apenas o numero e padroniza ele com o mesmo tamanho da
% referência:
number = cortebw(yMin - 1 : yMax, xMin - 1 : xMax);
number = imresize(number,[tamY1,tamX1]);
```

Figura 32: Redimensiona o índice.

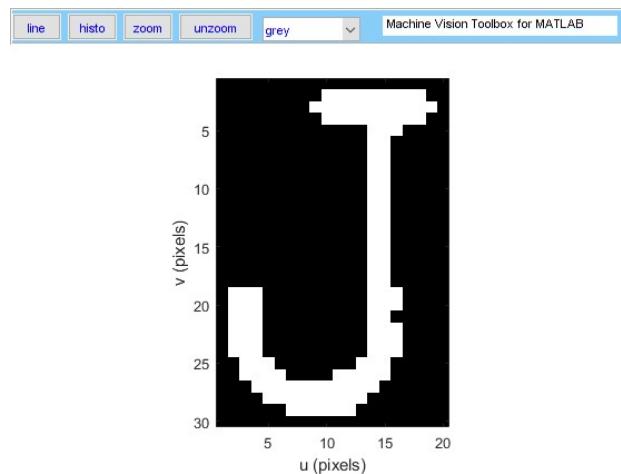


Figura 33: Índice redimensionado.

```

% Template Matching (SAD):
num = [];
num(1) = sum((sum(sum((number(:,:,1) - asbw(:,:,1)).^2)))); % Ases
num(2) = sum((sum(sum((number(:,:,1) - doisbw(:,:,1)).^2)))); % Dois
num(3) = sum((sum(sum((number(:,:,1) - tresbw(:,:,1)).^2)))); % Três
num(4) = sum((sum(sum((number(:,:,1) - quatrobw(:,:,1)).^2)))); % Quatro
num(5) = sum((sum(sum((number(:,:,1) - cincobw(:,:,1)).^2)))); % Cinco
num(6) = sum((sum(sum((number(:,:,1) - seisbw(:,:,1)).^2)))); % Seis
num(7) = sum((sum(sum((number(:,:,1) - setebw(:,:,1)).^2)))); % Sete
num(8) = sum((sum(sum((number(:,:,1) - oitobw(:,:,1)).^2)))); % Oito
num(9) = sum((sum(sum((number(:,:,1) - novebw(:,:,1)).^2)))); % Nove
num(10) = sum((sum(sum((number(:,:,1) - valetebw(:,:,1)).^2)))); % Valete
num(11) = sum((sum(sum((number(:,:,1) - damabw(:,:,1)).^2)))); % Dama
num(12) = sum((sum(sum((number(:,:,1) - reibw(:,:,1)).^2)))); % Rei

i = find(num == min(num));

% Printa o nome da carta:

fprintf('%s de %s \n',str(i),strl(n));

```

Figura 34: Tamplete Matching.

```

end
    end
end
toc

```

Figura 35: Fim do algoritmo.

## 4 Resultados

O algoritmo atendeu os 3 níveis de forma satisfatória e os resultados encontram-se a seguir:

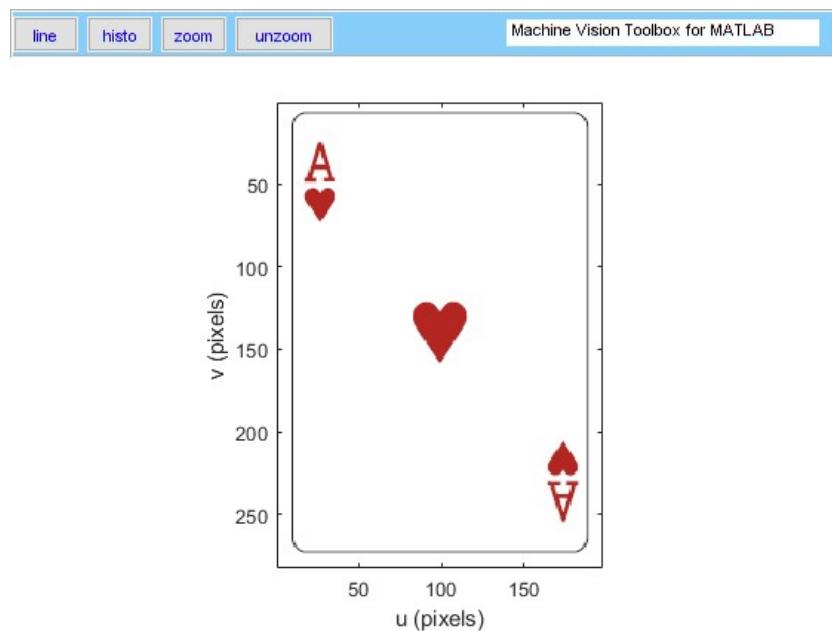


Figura 36: Imagem processada nível 1.

```
Ás de copas
Elapsed time is 0.900920 seconds.
>>
```

Figura 37: Resultado nível 1.

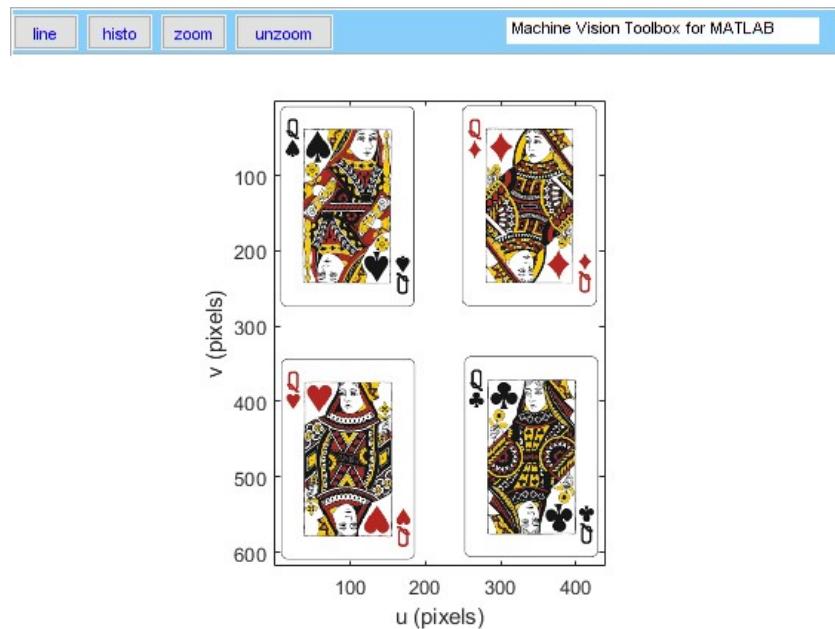


Figura 38: Imagem processada nível 2.

```
Dama de ouro
Dama de espada
Dama de paus
Dama de copas
Elapsed time is 1.557057 seconds.
```

Figura 39: Resultado nível 2.

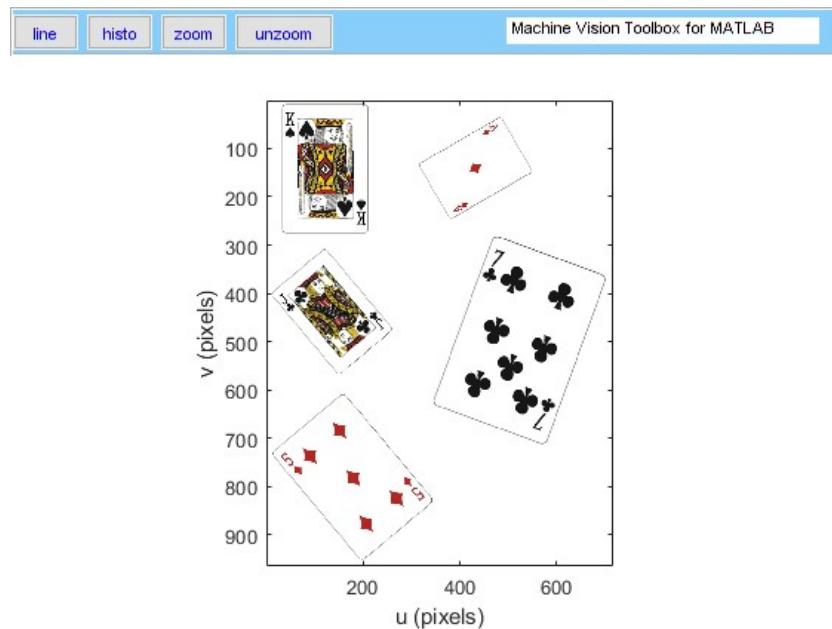


Figura 40: Imagem processada nível 3.

```

Rei de espada
Ás de ouro
Sete de paus
Valete de paus
Cinco de ouro
Elapsed time is 1.879662 seconds.

```

Figura 41: Resultado nível 3.

## **5 Referências**

### **Referências**

- [1] CORKE, Peter. Robotics, Vision and Control: Fundamental Algorithms in MATLAB. Springer Verlag NY, 1, 2011. ISBN: 3642201431.
- [2] RINCON, Leonardo Mejia. Visão Computacional em robótica: Notas de aula. Blumenau, SC, 2019.
- [3] MATSU, Marcos. Visão Computacional em robótica: Notas de aula. Blumenau, SC, 2019.