Relatório Trabalho 3 – IPI

Gustavo Pereira Chaves
Departamento de Ciência da Computação
Universidade de Brasília
Brasília, Brasil
gustavopchaves@hotmail.com

Keywords— imagem, processamento de imagem, binarização, algoritmos morfológicos, xadrez, detecção de peças

I Resumo

O seguinte trabalho teve como objetivo a implementação de um algoritmo que detecta a posição de peças de xadrez no tabuleiro através de técnicas de processamento de imagem.

Todo o processo foi dividido em três partes. A primeira refere-se ao processo de cortar somente a região do tabuleiro da foto em questão. Já a segunda consiste na detecção das peças em si, que busca através da análise de canal RGB da imagem, identificar o que possibilita a melhor binarização e posterior aplicação de algoritmos morfológicos para a detecção das peças em cada caso que será discutido a seguir. Enquanto que a terceira redimensiona a foto para uma imagem 8x8, que corresponde ao tamanho de um tabuleiro de xadrez comum.

Os códigos foram todos desenvolvidos em Python, utilizando as bibliotecas OpenCV e Numpy, havendo sido testados em todas as imagens fornecidas.

II Introdução

Este trabalho teve por finalidade a aplicação dos conhecimentos obtidos em sala de aula com relação a binarização de imagens e a aplicação de algoritmos morfológicos nessas imagens.

Inicialmente, para entender todo o processo é preciso discutir o processo de binarização de uma imagem, que consiste em tornar todos os pixeis desta em preto ou branco. Para tal, define-se o que é chamado limiar, que será o brilho mínimo necessário de um pixel para este ser transformado em branco, tornando-se preto caso o contrário.

Em segunda instância necessita-se entender o funcionamento dos algoritmos morfológicos que serão apresentados. Estes tratam da transformação de uma imagem baseado nas formas do objetos contidos nesta, sendo baseados em dois processos básicos, a erosão e a dilatação. Em imagens binárias, a erosão reduz o tamanho dos objetos brancos através da redução das bordas. Já a dilatação provoca o efeito contrário, aumentando o tamanho do objeto. Esses algoritmos funcionam através de um elemento estruturante (também chamado “kernel”) que percorre toda a imagem realizando a operação, sendo este definido como uma matriz de zeros e uns definida pelo usuário da função. A partir disso, é possível definir os algoritmos de abertura e fechamento, também utilizados no programa. A abertura consiste em uma erosão seguida por uma dilatação, o que provoca o desaparecimento de objetos menores que o kernel. Já o fechamento é uma erosão após uma dilatação, que provoca o fechamento de regiões menores que o kernel.

De posse dessas informações, pode-se dar continuidade a exposição do experimento.

III Metodologia

III.A Detecção do Tabuleiro

Inicialmente, dada uma foto, é preciso identificar a posição do tabuleiro, de forma a aplicar o restante do algoritmo somente na área de interesse.

Para esse processo foi criada a função CropImage, que recebe como parâmetros a imagem a ser cortada e o número de iterações. O funcionamento baseia-se na soma de cada coluna e cada linha da imagem separadamente, e definindo um valor de corte que foi calculado através da média dessas somas. Todos os valores acima dessa média são cortados do vetor de soma, e logo após tomando-se os índices do primeiro e do último elemento, têm-se a região a ser cortada da imagem.

A partir desse algoritmo verifica-se que são necessárias várias iterações para o corte corresponder a região do tabuleiro. Assim, foi definido de forma fixa que para o tabuleiro sem peças o algoritmo deve ser aplicado três vezes, enquanto que para o tabuleiro com peças dez vezes.

III.B Redimensionamento

Ao fim da detecção das peças que será detalhada a seguir, torna-se necessário a criação de uma função que redimensiona a imagem, de forma a reduzir ruídos e tornar o resultado final em um tamanho padronizado. Definiu-se então uma função chamada Resize, que dada uma imagem de entrada, a reduz para o tamanho 8x8.

Para tal, é utilizada a função resize do OpenCV com uma interpolação linear, de forma a reduzi-la inicialmente para o tamanho 32x32. Dessa forma, cada casa do tabuleiro passa a corresponder 16 pixeis (4x4). Após isso, passa-se a imagem resultante para a função FillWhite, que ao verificar cada casa, caso encontre pelo menos dois pixeis com brilho máximo, preenche os demais também com o brilho máximo. Dessa forma, evita-se a perda de uma peça que poderia não ser detectada caso fosse feito um redimensionamento diretamente para o tamanho desejado.

Por fim, basta redimensionar a imagem para o tamanho 8x8, utilizando a mesma interpolação e função do OpenCV já descritos.

III.C Detecção das Peças

Analisando as imagens, verificou-se que era necessário identificar peças de determinada cor em quadrados de cores diferentes separadamente, já que determinados canais RGB favoreciam certas casos e detrimento de outros. Dessa forma, definiram-se quatro etapas: a identificação de peças pretas nos quadrados brancos, peças brancas nos quadrados brancos, peças brancas nos quadrados pretos e peças pretas nos quadrados pretos.

Antes da realização dos processos a seguir, todas as imagens foram aplicadas ao algoritmo de corte descrito anteriormente. Definiu-se então a subtração das seguintes imagens:

subtracted_oc = Tabuleiro_Vazio - Tabuleiro_Com_Peças

subtracted_co = Tabuleiro_Com_Peças – Tabuleiro_Vazio

E analisando cada uma, verificou-se que os seguintes canais eram mais adequados para encontrar peças de cores específicas:

	R (vermelho)	G (verde)	B (azul)
subtracted_oc	Peças Brancas nos Quadrados Brancos	Peças Pretas nos Quadrados Pretos	Peças Pretas nos Quadrados Brancos
subtracted_co	Peças Brancas nos Quadrados Pretos	x	x

Para identificar as peças pretas nos quadrados brancos foi feita então uma binarização utilizando limiar = 90 no canal especificado. Após isso foi aplicada uma abertura com o kernel sendo uma cruz em uma matriz 40x40. Fez-se então um fechamento com o kernel sendo uma matriz 50x50, e por fim uma dilatação com kernel sendo uma matriz 20x20. Aplicada a função de redimensionamento, obtém-se a imagem resultante.

Posteriormente, torna-se possível identificar as peças brancas nos quadrados brancos. Binarizou-se então o canal descrito com um limiar = 20. Fez-se uma abertura com kernel em forma de cruz de tamanho 50x50. Posteriormente um fechamento com uma matriz 50x50 como elemento estruturante, e uma dilatação com o kernel sendo uma matriz 20x20. Fazendo o redimensionamento, verifica-se que ao fim, foram detectadas todas as peças que estavam em casas brancas, logo basta subtrair as peças pretas encontradas anteriormente.

Para encontrar as peças brancas em quadrados pretos binarizou-se o canal com um limiar = 50. Aplicou-se uma abertura com kernel sendo uma cruz de tamanho 40x40, e após isso um fechamento com um kernel em forma de matriz 20x20. Redimensionada a imagem, têm-se o resultado final.

Por fim, falta reconhecer as peças pretas nos quadrados pretos. Binarizando o canal descrito anteriormente com um limiar = 15, obtém-se uma imagem com várias peças demarcadas. Aplica-se então uma abertura com kernel em forma de cruz em uma matriz 70x70, um fechamento com kernel em forma de matriz 30x30 e uma dilatação com kernel sendo uma matriz 20x20. Ao fim, basta subtrair da imagem final todas as imagens já encontradas, o que resulta nas peças desejadas.

III.D Geração do Tabuleiro Digital

Finalizando o algoritmo, basta juntar todos os resultados encontrados e gerar o tabuleiro.

Criou-se então uma função chamada CreateChessBoard que inicializa um tabuleiro vazio, com as casas em tom de ciano. Ao fim de cada detecção, os quadrados com peças são identificados em branco, enquanto que os vazios em preto. Basta então adicionar no tabuleiro vazio as cores correspondestes nas posições encontradas, resultando no tabuleiro final digitalizado.

IV RESULTADOS

Após a descrição da metodologia utilizada, a seguir serão apresentadas as imagens geradas em cada etapa do algoritmo desenvolvido para as imagens “3.png” e “4.png”:

Tabuleiro vazio utilizado no algoritmo:

1 *Imagem de entrada “original.jpg”*

Tabuleiro com peças:

2 *Imagem de entrada “3.jpg”*

IV.A Detecção do tabuleiro:

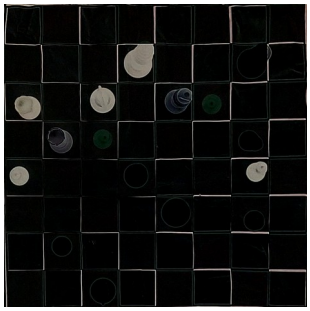
Aplicado o algoritmo de corte nas imagens de entrada:

3 *Imagem "original.jpg" cortada*

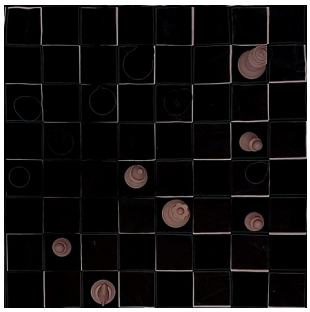
4 *Imagem "3.jpg" cortada*

IV.B Detecção das Peças:

Subtraindo as imagens cortadas:

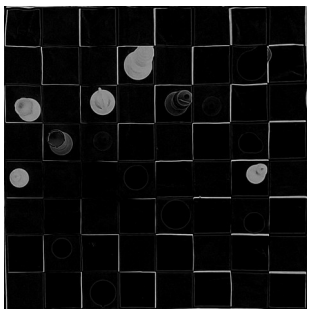


5 *Imagem "subtracted_oc"*

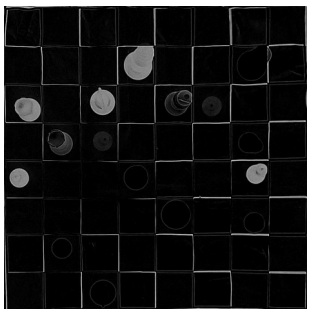


6 *Imagem "subtracted_co"*

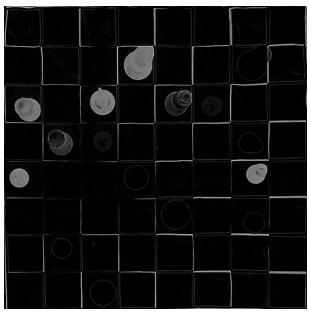
Separando os canais RGB da imagem "subtracted_oc":



7 *Canal Vermelho (Red) da imagem "subtracted_oc"*

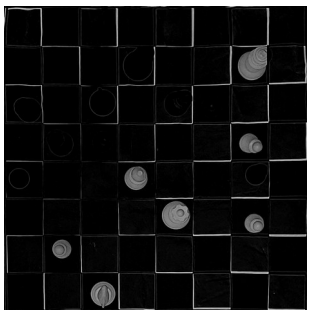


8 *Canal Verde (Green) da imagem "subtracted_oc"*

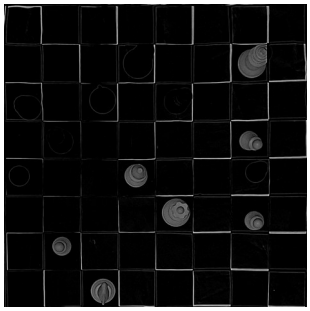


9 *Canal Azul (Blue) da imagem "subtracted_oc"*

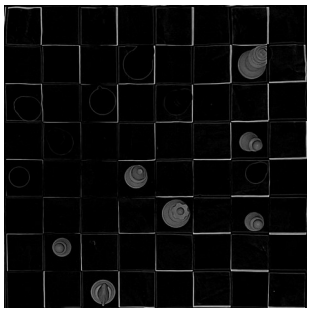
Separando os canais RGB da imagem "subtracted_co":



10 Canal Vermelho (Red) da imagem "subtracted_co"

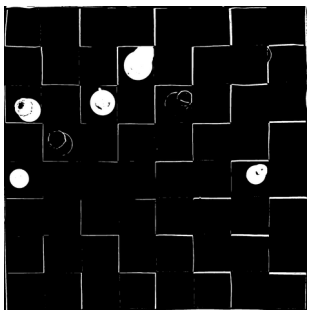


11 *Canal Verde (Green) da imagem “subtracted_co”*



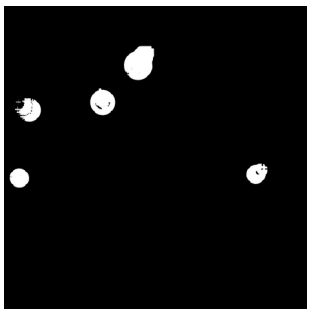
12 *Canal Azul (Blue) da imagem “subtracted_co”*

Começando pela detecção das peças pretas nas casas brancas, binarizando o canal escolhido:



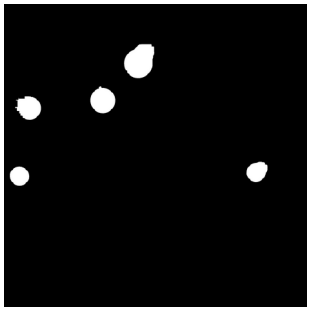
13 *Canal azul da imagem “subtracted_co” binarizado*

Aplicando a abertura:



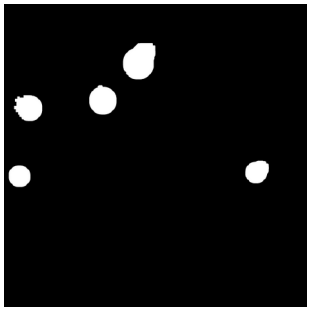
14 *Imagem binária após a abertura*

Realizando o fechamento:



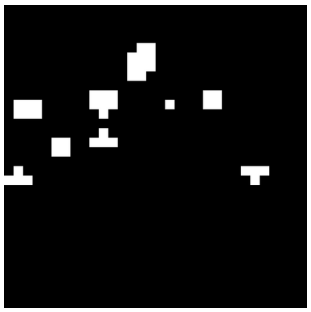
15 *Imagem binária após o fechamento*

Aplicando a dilatação:

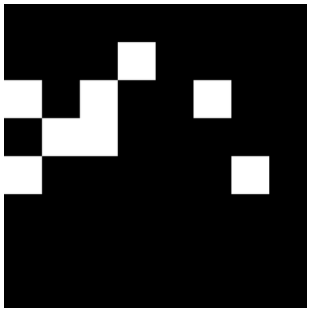


16 *Imagem binária após a dilatação*

Após isso, a imagem será redimensionada, inicialmente para 32x32, e após isso para 8x8 (esses passos intermediários serão omitidos para as demais imagens):

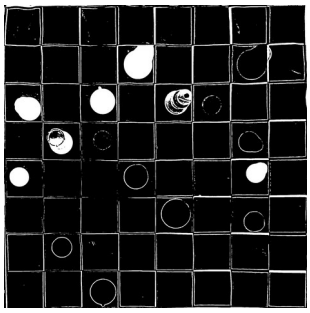


17 *Imagem redimensionada para 32x32 pixeis*



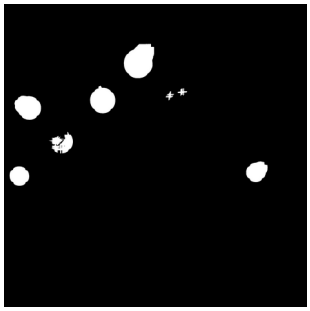
18 *Imagem final das peças pretas nas casas brancas*

Encontrando agora as peças brancas em quadrados brancos. Primeiro binariza-se o canal descrito:



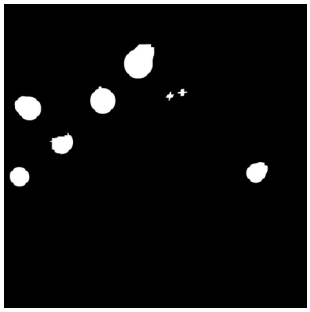
19 *Canal vermelho da imagem "subtracted_co" binarizado*

Após isso uma abertura:



20 *Imagem binária após a abertura*

Realizando o fechamento:

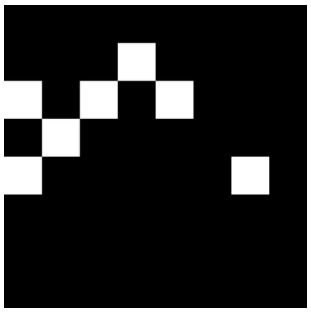


21 *Imagem binária após o fechamento*

Aplicando a dilatação:

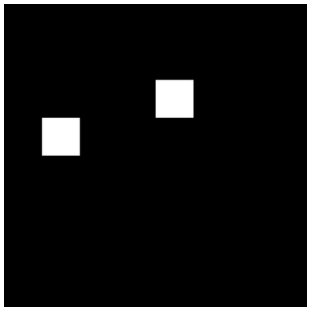
22 *Imagem binária após a dilatação*

Redimensionando a imagem e aplicando a função FillWhite:



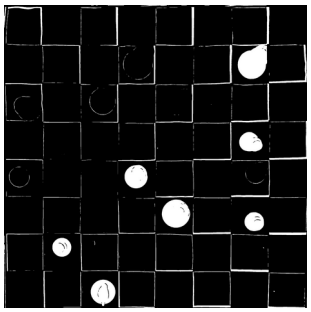
23 *Imagem após a função FillWhite e transformada em 8x8 pixeis*

Removendo as peças já encontradas:



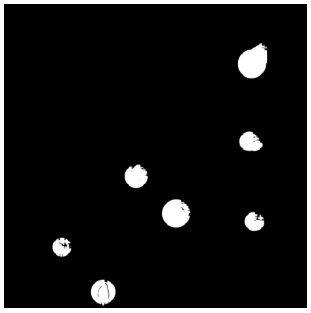
24 Imagem final das peças brancas nas casas brancas

Passando para a identificação das peças brancas nos quadrados pretos. Binarizando o canal:



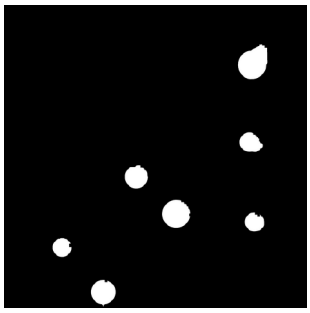
25 *Canal vermelho da imagem "subtracted_oc" binarizado*

Após isso uma abertura:



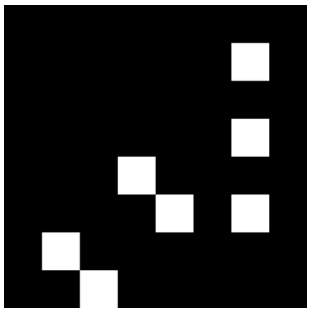
26 *Imagem binária após a abertura*

Realizando o fechamento:



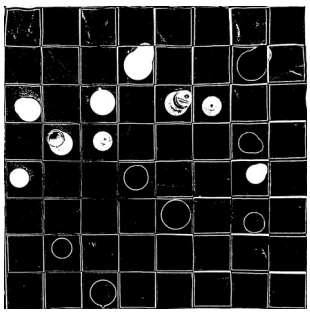
27 Imagem binária após o fechamento

Redimensionando a imagem e aplicando a função FillWhite:



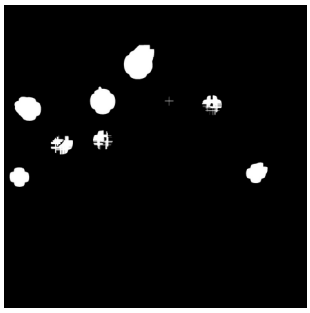
28 *Imagem final das peças brancas nas casas pretas*

Por fim, faltam encontrar as peças pretas nas casas pretas. Novamente binariza-se o canal escolhido:



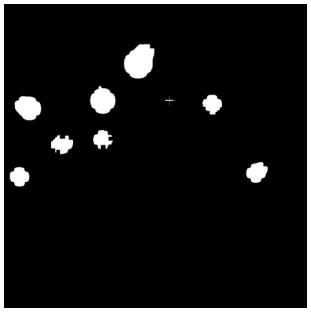
29 Canal Verde da imagem “subtracted_oc” binarizado

30 Após isso uma abertura:



31 *Imagem binária após a abertura*

Realizando o fechamento:

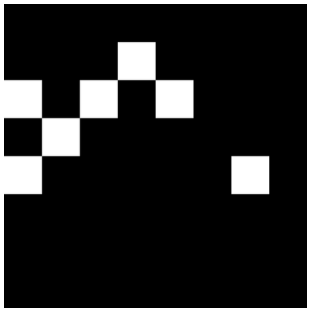


32 *Imagem binária após o fechamento*

Aplicando a dilatação:

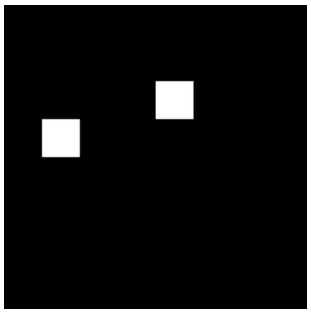
33 *Imagem binária após a dilatação*

Redimensionando a imagem e aplicando a função FillWhite:



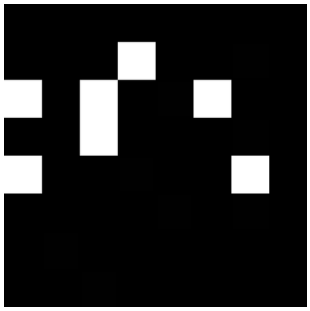
34 *Imagem após a função FillWhite e transformada em 8x8 pixeis*

Removendo as peças já encontradas:

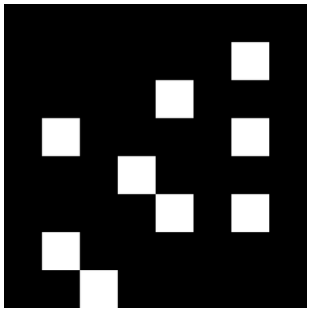


35 *Imagem final das peças pretas nas casas pretas*

Somando os resultados obtidos das peças pretas e brancas encontradas:

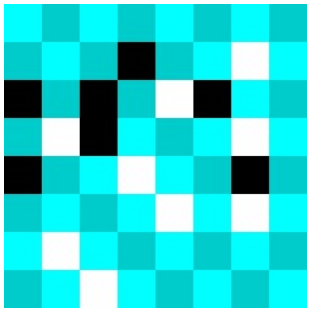


36 *Soma das imagens finais das peças pretas*



37 *Soma das imagens finais das peças brancas*

Convertendo então para as cores do tabuleiro digitalizado:



38 Resultado final

Aplicando esse processo para várias imagens, obteve-se o seguinte resultado para as peças pretas:

Imagem	Total de Peças Pretas	Peças Pretas Corretas	Falsos Positivos Peças Pretas	Falsos Negativos Peças Pretas
1.jpg	16	16	0	0
2.jpg	16	16	0	0
3.jpg	7	7	0	0
4.jpg	16	16	0	0
5.jpg	16	16	1	0
6.jpg	16	16	7	0
7.jpg	16	16	1	0
8.jpg	16	16	0	0
9.jpg	8	8	0	0
10.jpg	16	16	1	0
11.jpg	16	15	0	1
12.jpg	16	16	0	0
13.jpg	16	15	5	1
14.jpg	16	15	12	1
15.jpg	16	15	0	1
16.jpg	16	7	11	9

E o seguinte para peças brancas:

Imagem	Total de Peças Brancas	Peças Brancas Corretas	Falsos Positivos Peças Brancas	Falsos Negativos Peças Brancas
1.jpg	16	16	0	0
2.jpg	16	16	0	0
3.jpg	9	9	0	0
4.jpg	16	16	0	0
5.jpg	15	15	0	0
6.jpg	16	16	7	0
7.jpg	15	15	0	0
8.jpg	16	16	0	0
9.jpg	16	16	0	0
10.jpg	16	16	0	0
11.jpg	16	13	0	3
12.jpg	16	16	0	0
13.jpg	16	16	0	0
14.jpg	16	16	17	0
15.jpg	16	16	1	0
16.jpg	16	10	24	6

V Conclusão

Analisando os resultados de cada imagem, vê-se que para a maioria dos casos a taxa de acerta é satisfatória, ocorrendo poucos erros de detecção, exceto pelas imagens “6.jpg”, “14.jpg” e “16.jpg” que foram muito imprecisas.

Tratando primeiramente da imagem “6.jpg”, conclui-se que o nível de brilho da imagem afetou certamente o resultado, tendo em vista que a imagem “8.jpg” está com as peças nas mesmas posições, porém com uma iluminação melhor.

Já com relação as imagens “14.jpg” e “16.jpg”, observando as imagens de corte geradas, percebe-se que o algoritmo não cortou os tabuleiros corretamente, resultando em posições totalmente aleatórias. Devido a isso, acabam ocorrendo, ao acaso, uma série de acertos e consequentemente uma série de erros. Logo são considerados resultados ruins, pois não são produto da análise correta da imagem.

De forma geral, o algoritmo possui uma precisão descente em condições adequadas de brilho, tendo em vista que este pode afetar tanto o algoritmo de corte quanto a detecção das peças. Logo, os resultados ocorreram dentro do esperado ao utilizar somente algoritmos morfológicos para tal detecção.

Referências

1 *Slides do professor Bruno Macchiavello.*

2 Site de documentação do OpenCV

3 G. D Illeperuma, “Using Image Processing Techniques to Automate Chess Game Recording”, Sri Lankan Journal of Physics, vol. 27, pp. 76-83, January 2011.