Jogo da Velha em Realidade Aumentada

Introdução ao Processamento de Imagens

CIC-117242 – Turma A

Caio Yuri da Silva Costa – 09/0108213

Ciência da Computação, Departamento de Ciência da Computação

Universidade de Brasília

Brasília-DF, Brasil

caio.costa@outlook.com

Herisson Neves de Carvalho – 13/0054852

Engenharia de Computação, Departamento de Engenharia Elétrica

Universidade de Brasília

Brasília-DF, Brasil

herisson.carvalho96@outlook.com

Matheus Braga Almeida – 15/0141025

Engenharia de Computação, Departamento de Engenharia Elétrica

Universidade de Brasília

Brasília-DF, Brasil

matheus.mtb7@gmail.com

*Resumo*—Neste trabalho é mostrado como foi implementado um Jogo da Velha em Realidade Aumentada. Processamento morfológico e detecção de bordas são aplicadas na imagem capturada e as linhas são extraídas com o algoritmo *HoughLines*, de modo a detectar as linhas que correspondem ao jogo da velha. O quadro é dividido em subquadros para reconhecimento das jogadas pelo usuário, e as jogadas do computador são sobrepostas na imagem capturada.

Palavras-chave—Processamento Digital de Imagens; Operações morfológicas; Implementação lógica; OpenCV; Leitura e interpretação de quadros de vídeo; Jogo da velha;

# Introdução

O jogo da velha (em inglês, *Tic Tac Toe*) é um meio de entretenimento amplamente conhecido e utilizado como forma a prover lazer.

Todavia, sob o ponto de vista do Processamento Digital de Imagens, o referido jogo faz-se como objeto de estudo a fim de que se obtenha uma interação computacional ao ponto de permitir que a máquina seja capaz de se comportar como um jogador adversário ao jogador humano (usuário) incluindo elementos de realidade aumentada.

Data vênia, a consecução do acima assinalado em nada desaguaria senão pelo estudo das diversas operações morfológicas que circundam a análise de sinais bidimensionais, ou seja, de imagens. Os mecanismos morfológicos dizem respeito ao tratamento de imagens sob o intuito de que os resultados de suas aplicações satisfaçam determinados anseios e/ou finalidades.

Conseguinte, e ainda basilar, sabe-se que as referidas operações podem, de fato, ser utilizadas em diversas linguagens de programação ou, ainda, em várias Interfaces de Desenvolvimento. Dentre o amplo espaço de escolhas proporcionado, uma das linguagens que mais se adapta aos intuitos externados é a C++ (linguagem de programação orientada a objetos), uma vez que a base de conhecimento aplicada ao caso concreto se mostra mais vasta quanto a outras.

Ainda assim, quanto às IDEs (Ambientes de Desenvolvimento Integrado), afasta-se o critério opinativo a seu respeito, a fim de que seja utilizada aquela que seja apta a tornar viável a compilação e execução de implementações em OpenCV, ou seja, lançar-se-á mão de Ambiente de Desenvolvimento que seja compatível com as operações em OpenCV, fazendo-se uso do C++ para suas implementações.

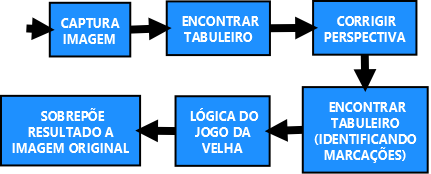
Afastando-se a discussão quanto às ferramentas utilizadas, e focando-se ao conteúdo do desenvolvido, retorna-se à elucidação da criação de elemento estruturante, das operações de dilatação, erosão, filtro gaussiano, *Canny*, uso do esqueleto da imagem, subtração de imagens, fechamento morfológico, da função *HoughLinesP*.

O bojo material não se limita tão somente àquelas aplicações, mas, em contrário, se expande em diversas funções utilizadas a fim de se alterar o sinal original, modificando-o em suas características ou adicionando elementos ao mesmo, sejam essas funções nativas ou desenvolvidas juntamente ao algoritmo principal.

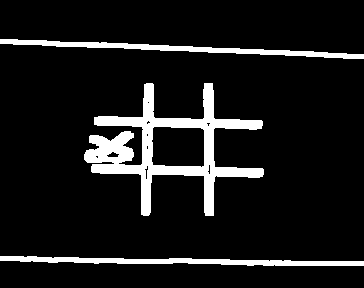
As referidas funções dizem respeito à análise de imagem ou vídeo, sob um foco de pixel, de bloco, ou de sinal por inteiro, objetivando-se que sejam encontrados determinados elementos, sujeitando-se à ajuda das operações morfológicas para tanto, e, uma vez encontrados, sejam comparados a padrões esperados, e, a partir de então, sejam adicionados ou modificados elementos a fim de que se alcance o objetivo do processamento, qual seja fazer com que o computador (*PC*) se comporte como um adversário no jogo da velha.

# Metodologia

Antes de mais nada, para uma melhor visualização dos procedimentos que se desencadeiam em função do objetivo aqui traçado, veja-se o fluxograma da Fig. 1:

Fig. 1.    Diagrama resumindo as operações implementadas para cada *frame* capturado.

Ao olhar para o problema, a solução se mostra um tanto trivial até que se analise a fundo a problemática trazida, onde se observa sua complexidade. O primeiro passo executado foi tentar extrair o tabuleiro da imagem, para isso foi aplicado o detector de bordas *Canny*. Conquanto, para se realizar tal procedimento, faz-se necessário aplicar um filtro anteriormente, qual seja o *Gaussian Blur.* No final, após as aplicações, obtém-se uma imagem com apenas as bordas e alguns elementos desconexos, os quais são retirados por um fechamento morfológico.

Fig. 2.    Resultado após operações morfológicas e detector de bordas *Canny*.

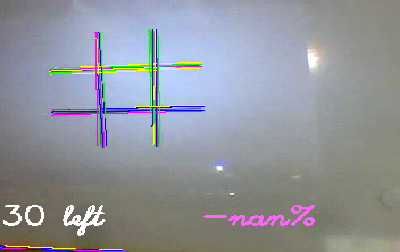
Nota-se, a partir da Fig. 2, que o problema dessa abordagem tem origem quando se vislumbra a qualidade das linhas apresentadas, ou seja, quando verifica se tratar de duas linhas paralelas em cada outra linha integrante do tabuleiro.

Nisso, cogitou-se preencher os buracos existentes, porém isto iria afetar outros elementos que não podem sofrer tal tipo de alteração, o que não seria adequado ao presente estudo, motivando novas tentativas de solução.

Na primeira tentativa, a operação de dilatação tomou lugar duas vezes, seguida da obtenção do esqueleto da imagem. Em resultado, obtinha-se uma imagem que representava as linhas, com espessuras variantes entre 1 e 2 pixels. Para que se detectasse as linhas, usou-se a transformada *HoughLinesP*, que é uma forma da *HoughLines* padrão, mas, esta, sob uma ótica probabilística. Tal solução não demonstrou a funcionalidade esperada, pois, mesmo obtendo claramente uma linha continua, em alguns casos, nenhuma linha era identificada pela transformada.

Durante alguns experimentos, foi retirada a função de obtenção do esqueleto da imagem, e foi aplicado, de forma direta, a *HoughLinesP* na imagem previamente dilatada (Fig. 2).

Com a aplicação da *HoughLinesP*, não intermediada pela obtenção de esqueleto, várias linhas foram aglutinadas na região que representa as quatro linhas integrantes do tabuleiro do jogo (Fig. 3). No qual é a implementação atual e que proporcionou melhores resultados.

Fig. 3.    Resultado da *HoughLinesP* sem obtenção de esqueleto.

Nesse momento, depara-se com o maior desafio do presente estudo: encontrar uma linha média para cada grupo de linhas, contendo linhas com ângulos levemente diferentes, tamanhos variáveis, e começos e fins diversos.

Para que se trespasse o referido desafio, uma análise linear merece destaque ao fim de que a solução seja alcançada.

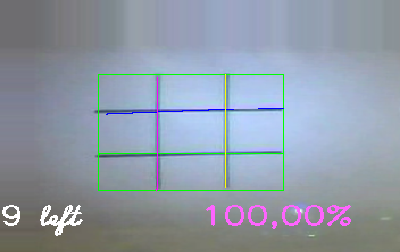
Primeiramente, as linhas são separadas em grupos, tendo como critério o ângulo de inclinação e a distância entre elas. O coeficiente de inclinação de cada linha é determinado, e, então, procede-se ao cálculo do arco-tangente (*arctg*) desse coeficiente. O ângulo, em radianos, varia entre *–π/2* e *π/2*. Dada as finalidades do algoritmo, os ângulos *–π/2* e *π/2* são equivalentes, haja vista que ambos são verticais. Isto posto, são calculadas as diferenças: | a – b |, | a + π – b | e | a – b – π |, onde *a* e *b* são os ângulos, em radianos, das duas linhas sendo comparadas. Se qualquer uma dessas diferenças (em módulo) for menor do que *π/8*, as linhas são consideradas como tendo inclinação similar. Partindo de que suas inclinações são similares, considera-se que distância entre as linhas é igual à distância entre o ponto médio de uma linha à reta definida pelas extremidades da outra linha. Com isso, as linhas que tenham inclinação similar e que distem menos de 32 (trinta e dois) pixels entre si são agrupadas.

O agrupamento acima apontado é realizado de maneira correlata a um *selection sort*, construindo uma nova lista à medida que elementos “iguais” sejam encontrados.

Para cada grupo de linhas, calcular-se-á uma única linha que represente aquele agrupamento (Fig. 4) da seguinte maneira:

Determina-se qual linha no grupo possui maior comprimento, e

Determinam-se as linhas do grupo cujos extremos estão a menos de 50 (cinquenta) pixels de distância dos extremos da linha determinada em *(a)*. É calculada uma nova linha, cujas extremidades são o centroide das extremidades das linhas obtidas em *(b)*.

Fig. 4.    Detecção de linha média para cada grupo de linhas.

Tendo em vista que se obtinha uma linha com um bom grau de certeza ao ponto de confirmar a existência de uma linha, ou seja, uma vez vencido aquele desafio, fez-se possível seguir para ao próximo passo, qual seja a identificação do tabuleiro dentre as linhas.

Sabe-se que o tabuleiro é composto de quatro linhas, teoricamente do mesmo tamanho, necessariamente cruzadas. Tendo a informação de que uma linha é intersectada sempre por outras duas, e também intersecta essas mesmas duas linhas, a partir de operações com vetores foi feita uma lista de quais linhas uma dada linha intersecta.

A intersecção de retas pode ser concluída a partir da análise da colinearidade entres três pontos, de forma que, sendo dois desses pontos integrantes exclusivos de uma reta *x* e um deles integrante simultâneo das retas *x* e *y*, caso os três pontos sejam, de fato, colineares entre si, e o ponto simultâneo se encontre dentro do intervalo do ponto exclusivo, a intersecção é a qualidade que se impõe[3].

Salientando-se que o acima enunciado funciona como um grafo, e dado que um elemento de um grafo só pode ter duas conexões, e que essas conexões têm um terceiro membro que deve conter o quarto fechando, em fim, o ciclo, satisfeita tal condição, o jogo é dado como encontrado. Ou seja, para que o jogo seja encontrado, uma linha deve ser cruzada por não menos, não mais, que duas linhas, e essas últimas devem ambas cruzar uma terceira que deve atender, igualmente, os mesmos requisitos da primeira linha.

Enfim, essa checagem funciona para todo o conjunto de linhas que, de fato, contenha o jogo (tabuleiro) e nenhuma outra linha que intersecte aquelas quatro linhas.

Após a checagem, e a detecção do jogo for efetivada, inicia-se o processo de homografia, que serve para aplicar uma transformada na imagem e redimensioná-la a fim de que pareça que ela esteja de frente. Isso é, “remover” ou compensar a perspectiva.

Outro grande desafio foi realizar o procedimento de achar a homografia da imagem, ou seja, analisar o sinal capturado e posicionar a imagem adaptando a perspectiva. No estudo base[1], tal procedimento é feito por meio de funcionalidades implementadas ao *OpenCV* (algoritmo SURF). No presente estudo, todavia, foram implemementadas funções próprias que, partindo-se de uma imagem capturada em qualquer posição, sejam aptas a dimensionar e posicionar a imagem a fim de, mesmo que originalmente esteja torta, inclinada, deitada, dentre outras posições, em resultado, a mesma se apresente como se estivesse plana.

Para tal, cada interseção de duas linhas é interpretada como uma origem, e as extremidades mais próximas dessas linhas, são interpretadas como vetores a partir dessa origem. A soma desses vetores é calculada, e conforme a Fig , resultam no quadrilatéro que delimita o tabuleiro, em qualquer perspectiva que esteja.

A partir da função *findHomography* do *OpenCV*, calcula-se a matriz de transformação linear necessária para transformar os pontos do quadrilátero em perspectiva, para um quadrado regular. Essa matriz é então utilizada na função *warpPerspective*, que efetua a distorção correspondente na imagem, corrigindo a perspectiva.

Feita a homografia, as transformadas anteriores e detecções são aplicadas novamente na imagem resultante da homografia, de modo a determinar a posição do jogo na imagem ajustada. Em seguida após todas as checagens e a certeza de que o jogo está em um determinado quadro, um contador que se inicia em 30 (trinta), é decrementado em 1 (um). Passa-se para o próximo quadro, onde, se for encontrado novamente o jogo, o decremento em 1 (um) é repetido, e, caso o mesmo não seja detectado, é incrementado em 0,5 (cinco décimos). Esse contador permite que o jogador regule e/ou posicione o dispositivo de gravação de vídeo, proporcionando uma melhor detecção e, consequentemente, um melhor funcionamento do algoritmo. Quando esse contador decremente o suficiente a ponto de zerá-lo, o jogo de fato inicia.

No início do jogo, são criadas 9 (nove) submatrizes, uma para cada célula do jogo sendo definidas pelo tamanho total do tabuleiro. O tabuleiro é definido pela maior e menor posição das retas. As células têm um terço do tamanho do tabuleiro em largura e em altura. Caso o tabuleiro seja simétrico e o ajuste de homografia tenha sido bem-sucedido, o resultado em tela ter-se-á por ideal, caso não haja qualquer deslocamento da imagem desenhada, para onde de fato o desenho deveria estar, ou da câmera.

Após iniciado o jogo, delimitadas as células e criadas as submatrizes, salientando-se que foi previamente criada uma máscara de tamanho previamente definido com um recorte em forma de “X” (Fig. 5), suas dimensões são adaptadas a fim de se ajustar ao tamanho de cada célula do tabuleiro do jogo. É utilizada a imagem binária que foi usada para detectar as linhas.

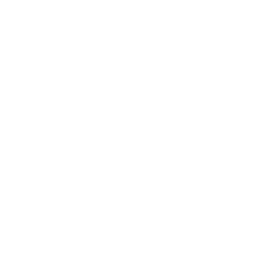


Fig.5.   Máscara utilizada.

Durante a execução, o programa continua buscando o tabuleiro, e se necessário, reposiciona o mesmo caso a câmera ou o tabuleiro tenham sofrido alguma alteração de posição.

Quando o tabuleiro deixa de ser identificado, inicia-se um pequeno contador, neste caso, 10 *frames*, quando esse contador chega ao fim, o estado do jogo muda para “O jogador está escrevendo”. Essa mudança de estado se dá em razão de que, quando o jogador está escrevendo (realizando jogada), sua mão, consequentemente, estará em cima do tabuleiro, bloqueando a leitura da imagem.

Quando o tabuleiro volta a ser identificado, o programa sobrepõe a máscara redimensionada para o tamanho da célula em questão, comparando em qual célula a marcação é concomitante ao recorte da máscara utilizando a operação binária *AND*, e computando a porcentagem de combinação.

Tal porcentagem é dada pela quantidade de pixels não zeros da máscara redimensionada, e o resultado de não zeros da operação.

Caso a referida porcentagem seja significativa (aos testes do presente estudo, cinquenta por cento foi suficiente), a célula é dada como marcada pelo jogador, e, então, é impresso em tela sua marcação (“X”). Dito isso, a vez, agora, é do computador, o qual seleciona uma célula vazia de forma randômica e realiza sua jogada, imprimindo em tela a mesma (“O”).

Em caso de o jogador marcar uma célula, durante o contador, e o estado não tiver sido alterado para “O jogador está escrevendo”, é feita uma média ponderada do valor. No início do jogo, o valor de cada célula é armazenado e a seguinte fórmula é executada para cada submatriz:

onde *NewP* é o novo valor a ser armazenado ao contador, *OldP* é o valor anterior e *ReadP* é o valor lido direto da medição.

Dada essa média ponderada, em nossos testes foi suficiente um tempo de 2 segundos para que o programa detectasse a célula em questão. Tal célula é marcada quando *NewP* ultrapassa o valor definido de cinquenta por cento.

A mudança na forma de checagem de célula se deu por conta de que no estudo base[1], para a consecução deste, restou um tanto aparente a existência de um contratempo com a orientação do “X” marcado pelo jogado humano, e, considerando que o jogador sempre joga primeiro, e quanto aos quesitos de apenas identificar onde ele jogou somado com as áreas de decisão, apenas uma marcação no local seria suficiente.

Após cada jogada, é checado se o alguém ganhou. Se verdadeiro, o jogo termina com a mensagem de quem venceu a partida, ao modo que, se “X”, o jogador venceu; se “O”, o computador venceu.

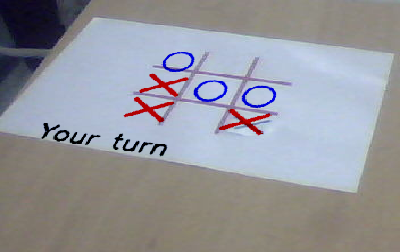
Por fim após todo processamento do frame atual, um último passo é feito, exibir corretamente na tela. Para isso, uma imagem preta de mesmo tamanho da imagem ajustada é criada. Nessa imagem é desenhado todas as marcações e textos do jogo. Então é feita uma função de homografia, utilizando a mesma matriz de transformação utilizada anteriormente, porém desta vez de forma reversa. Basicamente usa os mesmos pontos da primeira homografia, mas a função é chamada com um parâmetro dizendo para fazer a função inversa.

E no final a imagem da câmera é subtraída da imagem com apenas as marcações. E tem-se o jogo na tela, vide Fig 6.

# Resultados

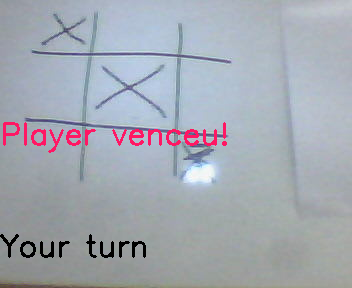
Após as diversas detecções explanadas, veja-se a Fig. 6, a qual apresenta um jogo em sua terceira rodada ilustrando o funcionamento do algoritmo desenvolvido.

Dada a Imagem anterior, os “X” constantes dessa foram escritos em um pedaço de papel e colados ao tabuleiro. O tabuleiro em tela é um tabuleiro ideal, cujas dimensões são 14x14 centímetros e cuja uma inclinação, mesmo que leve, se deu em razão da câmera, a qual estava posicionada a 60 (sessenta) centímetros do tabuleiro, ressaltando-se que a resolução da mesma é de 352x288 pixels.

Fig. 6.    Algoritmo em funcionamento na terceira rodada.

Nada obstante, depara-se com um problema que, infelizmente, não foi solucionado: o reflexo. O dispositivo de gravação de vídeo usado possui 6 (seis) LEDs que auxiliam na iluminação do tabuleiro. Nesse teste foi usado uma lousa que, por ser polida ou envernizada, reflete alguma onda luminosa. Nesse exemplo, é possível verificar que o “X” está marcado, mas, por conta do reflexo, o programa não o detecta.

Para contornar esse infortuno, uma área maior foi preenchida, a fim de se evadir do reflexo, assim como consta da Fig. 7.

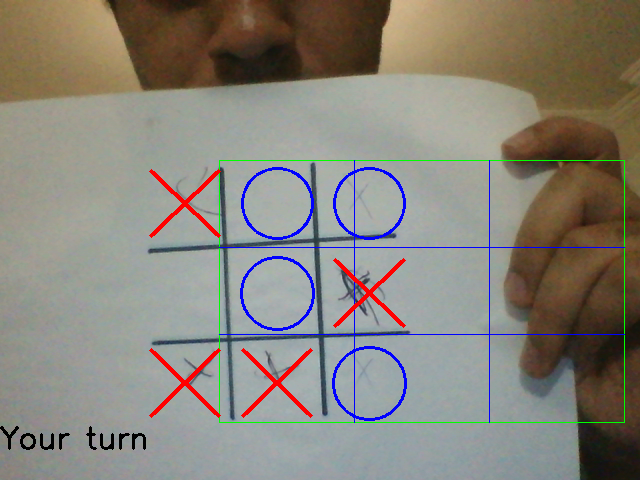
Fig. 7.    Vitória após várias jogadas.

Em outro teste, utilizou-se uma câmera com resolução mais alta e posicionada mais próxima ao tabuleiro: resolução de 640x480 pixels, tabuleiro de 14x14 centímetros e distância de 20 centímetros entre a câmera e o tabuleiro. Nisso, algumas vezes, por conta da *HoughLinesP*, e sua qualidade probabilística, algumas linhas ficam mais compridas do que deveriam. Em virtude disso, o tamanho do tabuleiro aumenta (Vide Fig. 8).

TABELA I.             Porcentagem de frames onde o jogo foi localizado, e falsos positivos e falsos negativos na detecção do X marcado.

|  |  |  |  |
| --- | --- | --- | --- |
| Vídeo | % frames | Falsos positivos | Falsos negativos |
| Teste1.webm | 100% | 1 | 1 |
| Teste2.webm | 100% | 0 | 3 |
| Teste3.webm | 82% | 0 | 0 |

De modo a possibilitar a reprodutibilidade dos testes, foram gravados 3 vídeos, para análise de performance e confiabilidade da solução apresentada. Os resultados são listados na Tabela 1.

Fig. 8.    Tabuleiro comprido em câmera com maior resolução, incluindo imagem tremendo.

# Conclusões

A partir do desenvolvido nesse projeto e nos testes aplicados aqui anteriormente e devidamente explanados, verificou-se que o procedimento de aplicação de diversas operações morfológicas não se limita a alcançar melhoramentos de sinal bidimensional findando-se em si mesmos, seja em *MatLab* ou em *OpenCV*, mas sim em proporcionar um objeto alhures daquele constante da entrada, ao ponto de, ao resultado, propiciar algo computacionalmente melhor interpretável, ou ainda, em outros termos, aplicando-se ao presente estudo, caso as operações morfológicas não tivessem tomado lugar importantíssimo nos primeiros momentos, toda a análise lógica e matemática subsequente seria impossível, ou ainda que possível, dificultosa.

Demais disso, após todo o desenrolar necessário, é veemente a satisfação do objetivo traçado ao início do estudo, qual seja desenvolver uma análise de imagens e vídeo capaz de interpretar um tabuleiro do Jogo da Velha, bem como as jogadas nele realizadas, a fim de que, dado isso, o computador seja capaz de interagir no referido tabuleiro, se comportando como o jogador adversário do jogador humano.

As implementações lógicas constantes do algoritmo ostentam, intrinsecamente, diversas manipulações que abarcam a teoria circundante de sinais bidimensionais e/ou tridimensionais.

Por fim, como sugestão de evolução ao aqui apresentado, se coloca quanto à inteligência do programa, ao ponto de que a jogada do computador tenda a evitar que o jogador humano vença.

##### Referências

1. Maguire, Joe; Saltzman, David. Augmented Reality Tic-Tac-Toe. Stanford: Final Project for Autumn 2013-2014, Student-Proposed Projects. < https://stacks.stanford.edu/file/druid:np318ty6250/Maguire\_Saltzman\_Augmented\_Reality\_Tic\_Tac\_Toe.pdf>; Acesso em 27 jun. 2016. (references)
2. Apresentações de Slides disponibilizadas pelo Profº Bruno Luiggi Macchiavello Espinoza.
3. How do you detect where two line segments intersect? Fórum online Stack overflow. <http://stackoverflow.com/questions/563198/how-do-you-detect-where-two-line-segments-intersect/565282#565282>; Acesso em 27 jun. 2016.
4. Base de conhecimento do OpenCV. <http://docs.opencv.org/>; Acesso em 27 jun. 2016.