UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS BLUMENAU
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Leonardo dos Santos Schmitt

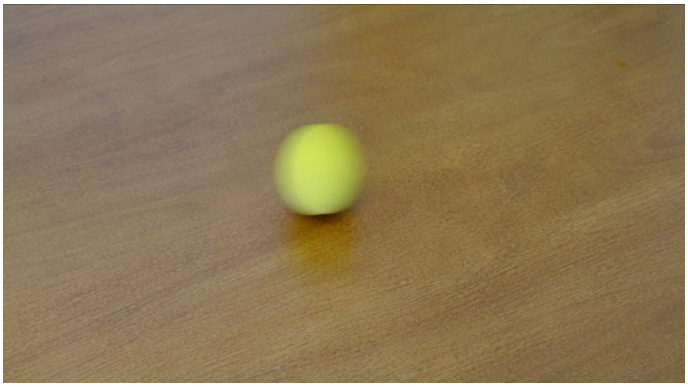
Laboratório 1: Visão Computacional em Robótica

Blumenau
2024

1 ATIVIDADE 1

A primeira atividade consiste no desenvolvimento de um sistema de visão computacional capaz de processar os *frames* de um vídeo disponibilizado pelo Dr. Prof. Marcos Matsuo, chamado `sequencia1.mp4`, para realizar a contagem de um objeto. Na Figura 1 é possível visualizar um *frame* do vídeo no qual o objeto (uma bolinha amarela) se encontra no centro do ambiente. Destaca-se ainda, que é necessário gravar um vídeo de autoria própria para realizar o mesmo processo.

Figura 1 – *Frame* contendo o objeto no centro do ambiente.



Fonte: Dr.Prof.Marcos Matsuo, 2024.

O sistema a ser desenvolvido é composto por etapas, sendo estas: a segmentação de cor do objeto, a extração da linha central da imagem binária e a detecção do objeto. Na segmentação de cor, o *frame* no espaço de cor BGR é convertido para o espaço de cor binário (preto e branco), visando isolar a região correspondente à bola amarela, conforme apresentado na Figura 2.

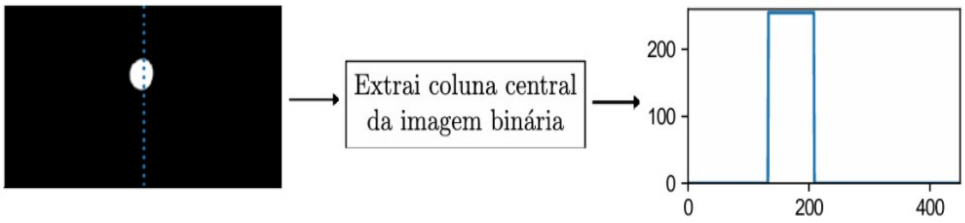
Figura 2 – Processo de segmentação de cor.

Fonte: Dr.Prof.Marcos Matsuo, 2024.

Com a etapa de segmentação de cor concluída, para o próximo passo é necessário extrair a coluna central da imagem binária, conforme mostrado na Figura 3. Após a extração, deve-se analisar a curva obtida, visando detectar a passagem de novos objetos.

Entretanto, salienta-se que o objeto não pode ser detectado enquanto estiver passando pela coluna central, ou seja, desde o momento de sua chegada até a sua saída, deve-se contabilizar apenas um objeto por vez.

Figura 3 – Extração da coluna central para contagem do objeto.



Fonte: Dr.Prof.Marcos Matsuo, 2024.

1.1 SOLUÇÃO DA ATIVIDADE 1

Conforme elucidado anteriormente, esta atividade solicita o desenvolvimento de um sistema que realiza a contagem do objeto por meio da segmentação de cor e da extração da coluna central. Esta seção apresentará a solução desenvolvida pelo autor, a qual abrange um contexto geral do processo, destacando detalhadamente cada fase e como ela pode ser implementada.

De primeiro momento, a solução consiste na segmentação de cor, que é a conversão da cor do *pixel* no espaço BGR para o espaço binário. Para realizar tal procedimento, deve-se definir as cores de referência do objeto, que podem ser representadas pelas componentes b_r, g_r e r_r, sendo azul, verde e vermelho, respectivamente. Essas componentes, quando aplicadas em código, são dispostas em um vetor e armazenadas em uma variável, como por exemplo `cor_referencia = [b_r, g_r, r_r]`, sendo que cada cor pode ser referenciada pelo seu respectivo índice na variável armazenada.

A Figura 4 apresenta um exemplo de segmentação de cor, que para este caso, por exemplo, podem ser definidos os valores de referência `[28, 34, 163]`, haja vista que a tonalidade do objeto que se deseja separar possui um tom mais avermelhado. Ressalta-se que esses valores são de escolha do projetista, que pode ajustar as cores de referência conforme sua necessidade.

Neste trabalho, a cor de referência utilizada pelo autor foi `[0, 255, 255]`, haja vista que ao utilizar as cores de referência diferente deste formato, a segmentação de cor não é eficaz, fazendo com que o objeto identificado e mapeado para a cor branca (espaço de cor binária) seja apenas o seu reflexo. Este problema pode ser causado pelo mau funcionamento do *VScode*.

Após definidas as cores de referência, deve-se realizar o armazenamento dos *frames* da imagem de entrada em uma variável definida como **I**. Esta, por sua vez, será utilizada

Figura 4 – Exemplo de segmentação de cor para uma imagem com tomates.

Fonte: Dr.Prof.Marcos Matsuo, 2024.

em conjunto com as cores de referência para realizar a segmentação do objeto. Existem funções já prontas que conseguem realizar este processo, entretanto, para este trabalho, o método utilizado é a distância euclidiana, que é uma forma de verificar a similaridade de cores. Os *pixels* com cores mais semelhantes, no cálculo da distância entre **I** e as cores de referência, terão uma distância euclidiana menor.

Neste caso, a distância euclidiana pode ser armazenada em uma variável **D**, que irá representar uma imagem, onde cada *pixel* **D**[y, x] é dado pela distância euclidiana entre o *pixel* **I**[y, x], da imagem de entrada, e a cor de referência. Na Equação (1), é possível visualizar o cálculo em questão, onde **B**[y, x], **G**[y, x] e **R**[y, x] referem-se aos *pixels* da camada BGR da imagem original.

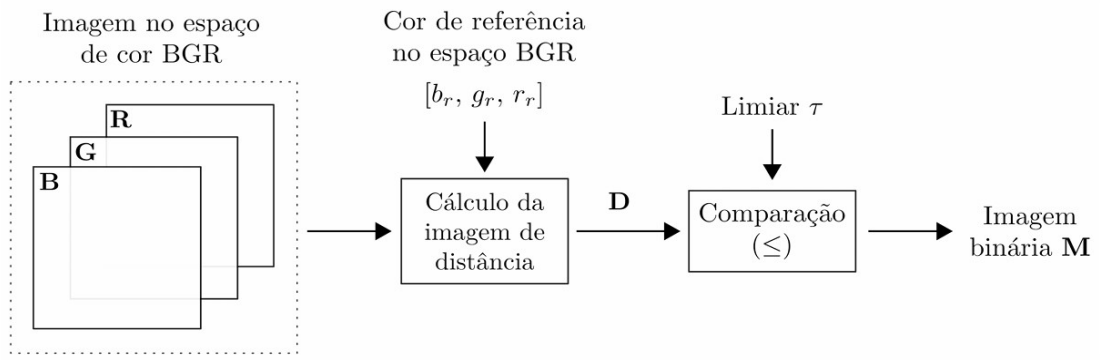
$$D[y,x] = \sqrt{(B[y,x] - b_r)^2 + (G[y,x] - g_r)^2 + (R[y,x] - r_r)^2}. \tag{1}$$

Com a nova imagem **D** obtida, é possível realizar a limiarização dos *pixels* da mesma com o limiar τ definido pelo projetista (que para este caso foi 170), sendo que o resultado será armazenado em uma imagem binária **M**, conforme apresenta a Figura 5, que ilustra o fluxo que o algoritmo deve seguir. Em outras palavras, um limiar definido através de experimentação faz um *threshold* dos limites em que o *pixel* pode estar. Se o valor do *pixel* **D**[y, x] for menor ou igual ao valor do limiar, o *pixel* resultante na imagem **M** será 255 (representando a cor branca no espaço de cor binário com valor `uint8`), caso contrário, será 0 (representando a cor preta no espaço de cor binário).

A Equação (2) representa o que foi brevemente mencionado. Além disso, esse tipo de implementação, pode ser dada através de um laço de repetição, no qual cada pixel da imagem D passa pelo condicional da equação abaixo, que é representado por `if` e `else`.

$$M[y,x] = \begin{cases} 255, & \text{se } D[y,x] \leq \tau \\ 0, & \text{caso contrário} \end{cases} \tag{2}$$

Figura 5 – Fluxo a ser seguindo pelo Algoritmo de segmentação baseada em cor e em medida de distância.



Fonte: Dr.Prof.Marcos Matsuo, 2024.

Por fim, além de obter a imagem binária M, é necessário extrair a coluna central da imagem, que é representada por `M[0:449, 399]`, e realizar a contagem de objetos. Esse valor da coluna central está atrelado ao fato de que a imagem tem 450 linhas e 800 colunas, contudo, a contagem dos índices começa em zero.

Ao analisar os *pixels* da coluna central, se pelo menos um deles tiver o valor 255, isso significa que um objeto está presente. Quando um pixel com valor 255 é detectado, um contador, que é uma variável inteira, é incrementado em uma unidade. Essa variável é responsável por manter o registro da quantidade de objetos que passam pela coluna. Para garantir que o contador não registre o mesmo objeto mais de uma vez, utiliza-se uma variável booleana chamada `flag`. Inicialmente a `flag` é definida como `False` e, quando um objeto é detectado, a mesma é alterada para `True`, evitando que o contador seja incrementado novamente enquanto o objeto ainda estiver na coluna. Assim que o objeto passa completamente e não há mais *pixels* com valor 255, a `flag` é revertida para `False`. Isso permite que em uma próxima detecção, o sistema possa contar novamente a passagem de novos objetos.

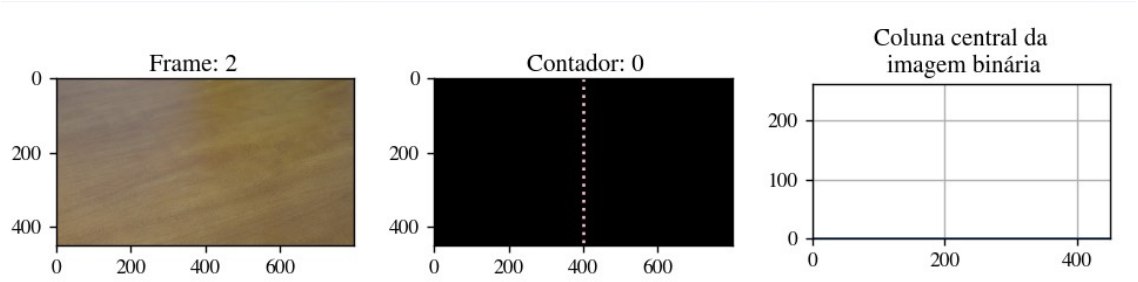
Ao final, é possível mostrar na tela todas as etapas do procedimento. Sendo assim, é exibida uma interface contendo o vídeo original, com a contagem de *frames*, a imagem binária, com a contagem de elementos, e a coluna central, que será mostrada na próxima subseção.

1.2 RESULTADO OBTIDOS NA ATIVIDADE 1

Nesta seção, serão apresentados os resultados obtidos na atividade 1, por meio de figuras. Além disso, será apresentado o tempo médio de processamento de cada quadro do vídeo, no qual foram utilizadas as funções `getTickCount()` e `getTickFrequency()` do *OpenCV*.

Primeiramente, observa-se que o segundo *frame* do vídeo, conforme mostrado na Figura 6, não há nenhum objeto visível em cena. Todo o fundo foi mapeado para a cor preta no espaço de cor binário, e a coluna central também não apresenta qualquer elemento. É importante ressaltar que a representação da coluna central na imagem binária não impacta a contagem do objeto, ela foi incluída apenas para facilitar a visualização do usuário.

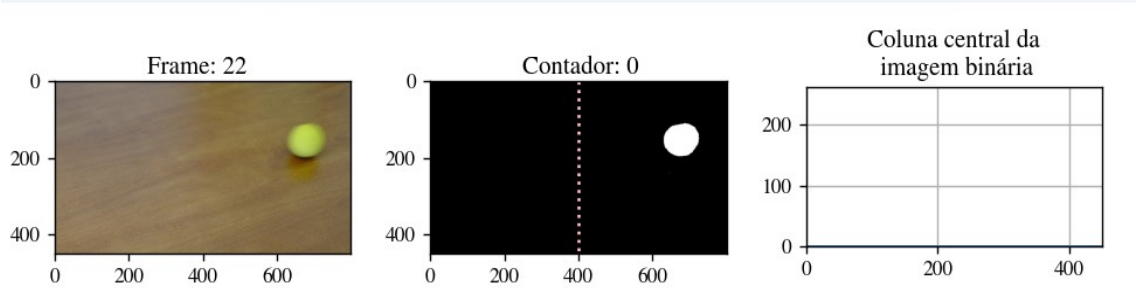
Figura 6 – Segundo *frame* do vídeo, onde não há presença do objeto.



Fonte: Autor, 2024.

Na Figura 7, é possível visualizar que o objeto (bolinha amarela) já está visível no ambiente. Além disso, o mapeamento do espaço de cor BGR para binário foi realizado de acordo com o projeto, atendendo ao limiar. Nota-se também que no terceiro gráfico, que representa a extração da coluna central, não houve detecção de nenhum pixel na cor branca (255).

Figura 7 – Vigésimo terceiro *frame* do vídeo, onde há presença do objeto.

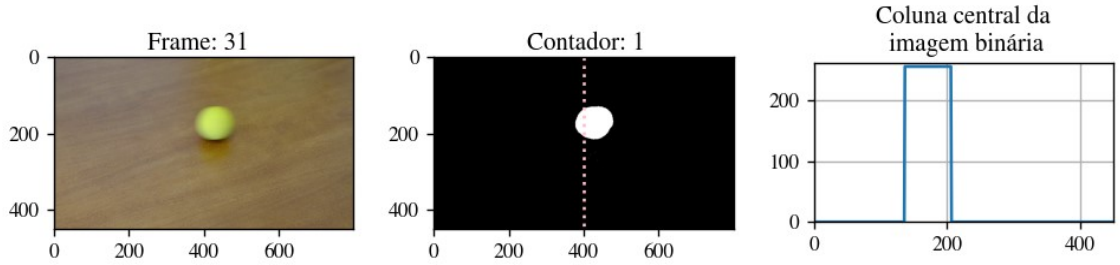


Fonte: Autor, 2024.

Em seguida, conforme os *frames* vão passando, o objeto se encontra na coluna central. Nesse momento, é possível visualizar que o contador é incrementado para 1, e o terceiro gráfico apresenta os *pixels* da coluna central que estão com o valor de 255, conforme mostra a Figura 8. Contudo, apenas analisando este *frame*, não é possível afirmar que o algoritmo está realizando o processo por completo, sendo que é necessário ainda, analisar o próximo *frame* para verificar se o contador não é contabilizado mais de uma vez.

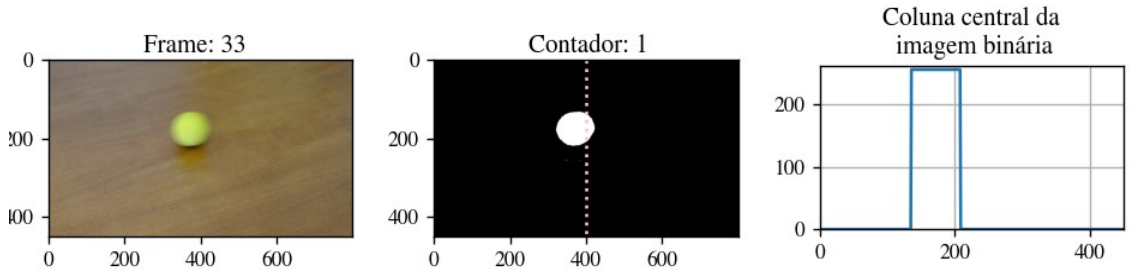
Ao analisar a Figura 9, é possível observar que o condicional implementado, que faz com que a variável booleana `flag` mude de valor, está funcionando de acordo com o planejado, pois o objeto não é contabilizado mais vezes.

Figura 8 – Primeira objeto contado e *pixels* apresentados na coluna central.



Fonte: Autor, 2024.

Figura 9 – Verificação do funcionamento da variável *flag* para contabilização do primeiro objeto.



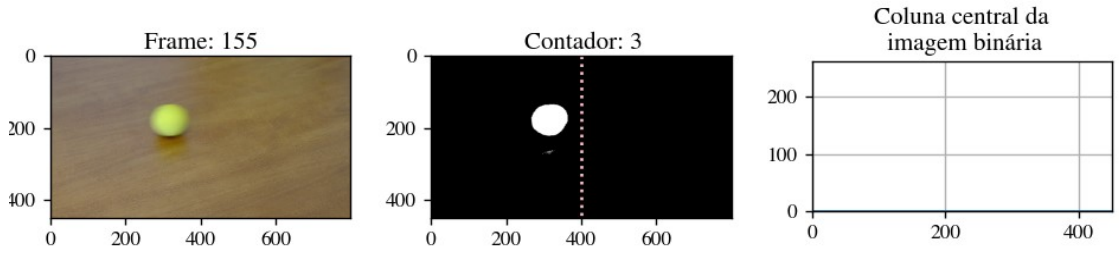
Fonte: Autor, 2024.

Além disso, pode-se avaliar se mais de um elemento pode ser contabilizado na mesma iteração. A figura Figura 10 ilustra a contagem de três elementos que já passaram pela coluna central da imagem.

Conforme visto, conclui-se que o algoritmo desenvolvido atende aos requisitos do projeto, permitindo que todas as etapas sejam realizadas, desde a segmentação de cor até a contagem do objeto. Em vista disso, deve-se aplicar o algoritmo a outro objeto. Por opção do autor, o objeto escolhido está apresentado na Figura 11. Este objeto, por sua vez, possui predominância na cor azul, contudo, há algumas partes que apresentam outras tonalidades no espaço de cor BGR. Sendo assim, para este caso, defini-se as cores de referência como `[255, 0, 0]` e o limiar como 180.

Nota-se na Figura 12 que a contagem do objeto, assim como a extração da coluna central, apresenta uma diferença em comparação com a coluna central da bolinha amarela.

Figura 10 – Contabilização de três objetos detectados.



Fonte: Autor, 2024.

Figura 11 – Objeto escolhido pelo autor para novo ensaio do código.

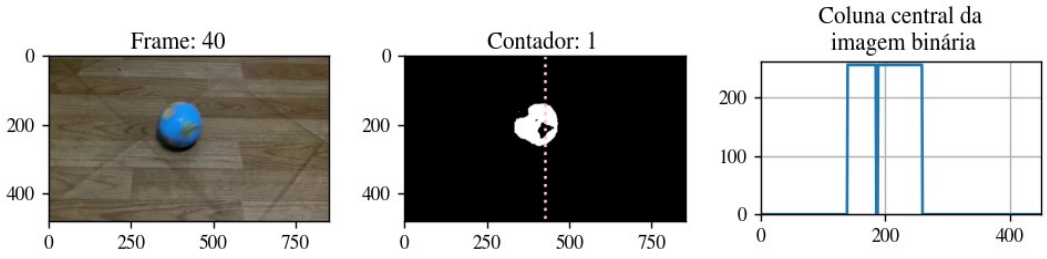
Fonte: Autor, 2024.

Isso se deve ao fato de que alguns pontos do objeto escolhido pelo autor (globo) não foram mapeados para a cor branca, pois são diferentes da cor azul. Contudo, o algoritmo consegue contar um objeto por vez.

Quando o limiar é alterado para um valor menor, como 140, o objeto passa a ser contado duas vezes, conforme mostram as Figura 13 e Figura 14. Isso ocorre porque o objeto não é mapeado corretamente para o espaço de cor BGR, e, quando ele passa pela coluna central, é como se houvesse dois objetos passando. Para resolver esse tipo de problema, deve-se manter o limiar conforme projetado ou realizar outro processo que ainda não é conhecido pelo autor.

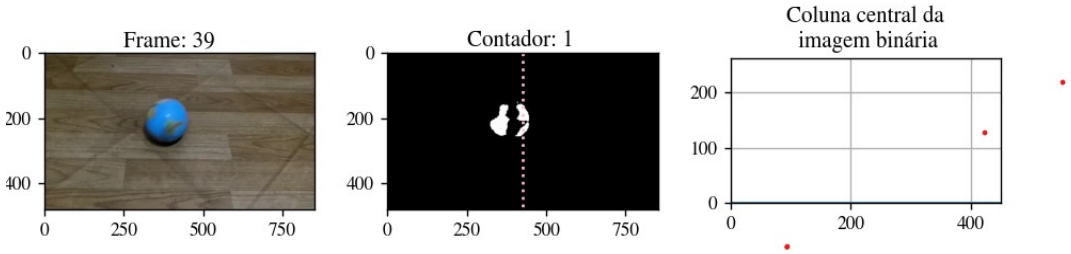
Por fim, a análise do tempo de execução do algoritmo para verificar se o código é capaz de operar em tempo real, é realizado com a contagem de tempo em segundos por *frame*, sendo que esse valor foi obtido a partir da média de todos os *frames* do vídeo. Após isso, é necessário realizar a avaliação do tempo de processamento, conforme apresentado na Equação (3). Na Tabela 1, visualiza-se os valores obtidos tanto para o vídeo da bolinha

Figura 12 – Primeira contagem do novo objeto escolhido pelo autor.



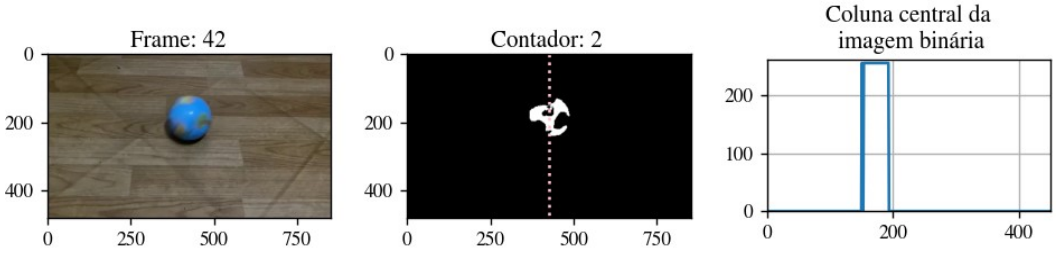
Fonte: Autor, 2024.

Figura 13 – Primeira contagem do novo objeto escolhido pelo autor com τ de 140



Fonte: Autor, 2024.

Figura 14 – Segunda contagem instantes após a primeira contabilização do novo objeto.



Fonte: Autor, 2024.

amarela quanto para o globo.

$$FPS = \frac{1}{\text{tempo em segundos por frame}} \tag{3}$$

Tabela 1 – Comparação Entre os Tempos de Processamento dos Vídeos.

Tempos de Processamento e FPS		
Vídeo	**Tempo de Execução (s)**	**FPS**
sequencia1	0.133	7.47
globoseq	0.203	4.92

Fonte: Autor, 2024.

Conforme apresentado na tabela acima, é possível verificar que não é viável aplicar o código em tempo real, uma vez que a maioria das aplicações requer uma taxa de 30 FPS para que o usuário possa enxergar sem nenhum atraso. Ressalta-se que o vídeo realizado pelo autor com o globo possui uma qualidade maior, pois levou mais tempo para ser processado.

2 ATIVIDADE 2

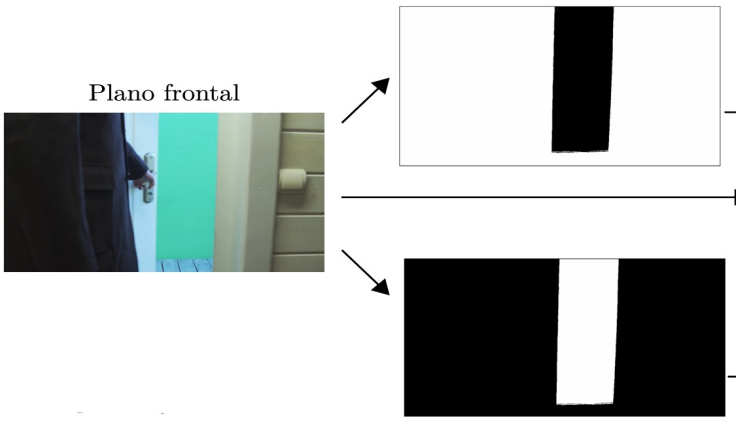
A segunda atividade envolve a aplicação do efeito *chromakey*, para combinar duas imagens, sendo estas: a Figura 15a e a Figura 15b. Neste contexto, a Figura 15b deve atuar como o fundo da imagem Figura 15a, substituindo a área verde. Ambas as imagens foram extraídas de vídeos disponibilizados pelo Dr. Prof. Marcos Matsuo, sendo `Chromakey.mp4` utilizado para a Figura 15a e `Clouds.mp4` para a Figura 15b.

(a) Plano frontal. (b) Plano de fundo.

Figura 15 – Imagens para aplicação de *chromakey*.

Em conjunto com os vídeos disponibilizados, deve-se utilizar o arquivo `lab1_atividade2.py` e o algoritmo de segmentação de cor da atividade 1 para gerar as máscaras binárias necessárias, apresentadas na Figura 16. Essas máscaras serão utilizadas para realizar operações entre o plano frontal e o plano de fundo, possibilitando a aplicação do efeito de *chromakey*.

Figura 16 – Máscaras binárias do plano frontal.



Fonte: Dr.Prof.Marcos Matsuo, 2024.

Por fim, espera-se obter a imagem apresentada na Figura 20 através da aplicação

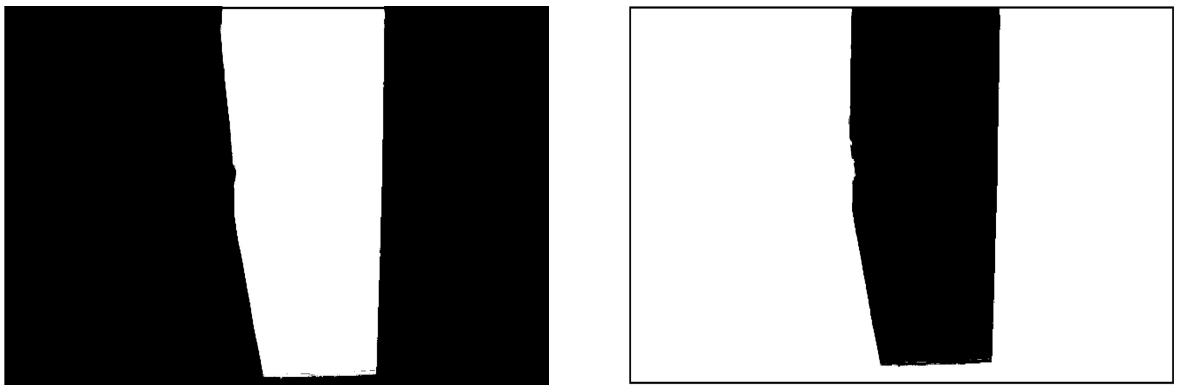
do efeito de *chromakey*. Ressalta-se, contudo, que a aplicação pode não ser 100% eficaz, uma vez que são necessários outros tratamentos para realizar o refinamento desta técnica.

2.1 SOLUÇÃO DA ATIVIDADE 2

A solução da Atividade 2 é composta por elementos já discutidos neste relatório, especialmente no Capítulo 1, onde foi utilizada a segmentação de cor para o mapeamento do espaço de cor BGR para o formato binário. Além de aplicar a segmentação de cor, é necessário realizar operações entre as imagens e as máscaras, possibilitando assim a criação do efeito de *chromakey*.

De início, é necessário realizar a segmentação de cor, ou seja, deve-se definir a cor de referência do objeto que será mapeado para o espaço binário. Para este caso, o valor utilizado para as cores b_r, g_r e r_r é `[150, 255, 44]`. Após definir a cor de referência, pode-se utilizar o conceito de distância euclidiana, apresentado na Equação (1), para realizar o mapeamento do *pixel* através do valor de um limiar, que neste caso é 95 e, é representado pela variável `delta`. Isso permitirá mapear o objeto para o *pixel* esperado.

A nova imagem resultante, apresentada na Figura 17a, está no espaço de cor binário e é uma máscara, pois destaca o objeto em branco (valor de 255). Após obter a máscara na qual o objeto em questão é referenciado como branco, é necessário realizar o processo de subtração de cada pixel por 255. Por exemplo, `M2 = 255 - M1` no qual resulta em outra máscara, em que o objeto passa a ser preto e o restante da imagem é branco, conforme apresentado na Figura 17b.



(a) Máscara binária M1 (b) Máscara binária M2

Figura 17 – Máscaras binárias do plano frontal.

A obtenção de duas máscaras se faz necessária para facilitar as operações de combinação de imagens durante o efeito de *chromakey*. Em outras palavras, a segunda máscara é utilizada em uma operação *AND* com a imagem Figura 15a, fazendo com que a região de interesse compreenda todas aquelas que não possuem as cores de referência. Dessa forma, o fundo é considerado preto, enquanto o restante da imagem, que seria branco, se torna a imagem apresentada na Figura 18a.

Por outro lado, quando utilizamos a primeira máscara em conjunto com a Figura 15b e realizamos também a operação *AND*, o objeto em questão se transforma nas nuvens, conforme apresentado na Figura 18b. Ressalta-se que, para realizar tais operações, pode-se utilizar a função `bitwise_and` do *OpenCV*.

(a) Máscara aplicada no plano frontal (b) Máscara aplicada no plano de fundo

Figura 18 – Aplicação das máscaras nos fundos.

Por fim, após a aplicação das máscaras, basta realizar uma operação de soma, utilizando a função `add()` do *OpenCV*, para obter a imagem resultante. A imagem resultante pode ser visualizada na próxima seção.

2.2 RESULTADOS OBTIDOS NA ATIVIDADE 2

O resultado obtido na atividade 2 foi alcançado por meio da aplicação do código previamente abordado na seção anterior, bem como através de testes das cores de referência e do limiar `delta`. Esses dois critérios são de extrema importância para melhorar os resultados da aplicação do *chromakey*. No entanto, conforme mencionado anteriormente, a aplicação não será 100% eficiente, pois existem outros métodos para aperfeiçoar os resultados.

A Figura 19 apresenta um resultado no qual não foram utilizadas os valores dos parâmetros mencionados na seção anterior, mas sim um `delta` de 40 e cores de referência de `[180, 227, 89]`. Nota-se que ainda existem partes com tons da cor de fundo que deveriam ter sido eliminados completamente. Contudo, sempre há variações nas bordas que não são mapeadas por completo para o espaço de cor binária. Além disso, o valor do `delta` pode influenciar o mapeamento, fazendo com que ele não seja realizado de forma correta.

Ao utilizar os valores de cores de referência e de `delta` mencionados anteriormente, é possível visualizar uma melhoria, especialmente na parte inferior, perto da madeira, onde não há mais partes derivadas do fundo antigo. O resultado final pode ser visualizado na Figura 20. Assim, conclui-se que os resultados são satisfatórios e que o código foi implementado conforme solicitado.

Figura 19 – Aplicação do *chromakey* com delta igual 40.

Fonte: Autor, 2024.

Figura 20 – Aplicação do *chromakey* com delta igual 90.

Fonte: Autor, 2024.