Detetor de alterações em tempo real usando OpenCV

Tiago Dias 88896
Martim Neves 88904

***Resumo* – O presente artigo apresenta o desenvolvimento de uma aplicação que permita detetar, em tempo-real, modificações significativas numa sequência de vídeo obtida por uma câmara fixa, usando a subtração do background, implementada através da biblioteca C/C++ OpenCV.**

O artigo começa por estabelecer alguns conceitos básicos relativos a detecção de alterações em vídeo e a técnicas de subtração do background. De seguida, são apresentados os principais objetivos da aplicação considerados ao longo do desenvolvimento do projeto e como foi feita a sua implementação. Finalmente, descrevem-se e discutem-se os resultados obtidos por diferentes algoritmos e diferentes configurações dos algoritmos.

I. INTRODUÇÃO

A deteção de alterações é fundamental em vários domínios, com particular relevância para os modernos sistemas de vigilância. Os sistemas atuais têm como tarefa inferir informação sobre o ambiente que estão a monitorizar. Para alcançar o objetivo do sistema, muitas vezes é necessário executar um *pipeline* de instruções sobre os dados obtidos do ambiente (como um *stream* de vídeo de uma câmara fixa).

Normalmente, um dos primeiros passos nesse *pipeline* corresponde à obtenção de frames de uma câmara fixa e, após processamento desses frames, detetar a existência de alterações. Associado à deteção pode estar a exibição de alterações

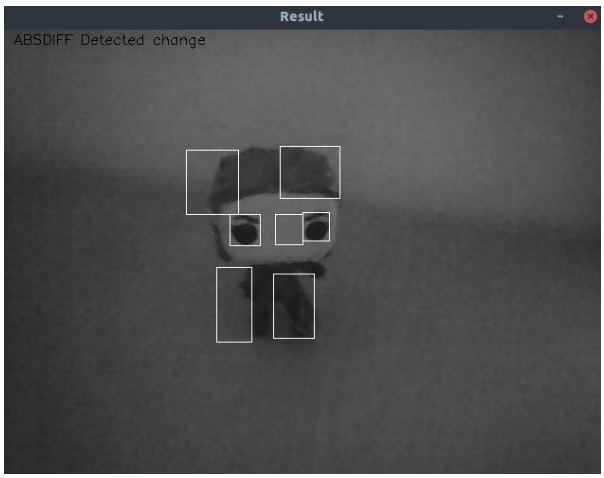

Figura 1 – Exemplo de deteção de alterações numa cena

O objetivo da aplicação desenvolvida foi a implementação de um sistema que detete alterações num vídeo obtido através de uma câmara fixa (embora possa ser usado em situações em que a câmara não se encontra fixa) através de 3 algoritmos distintos, 1 baseado numa abordagem simples e outros 2 baseados em algoritmos *built-in* do OpenCV.

A. *Background subtraction [1]*

Backgroud subtraction é uma técnica muito usada para gerar uma máscara de *foreground.* Essa máscara corresponde a uma imagem binária que contém os pixéis que definem as alterações numa dada cena (Figura 2).

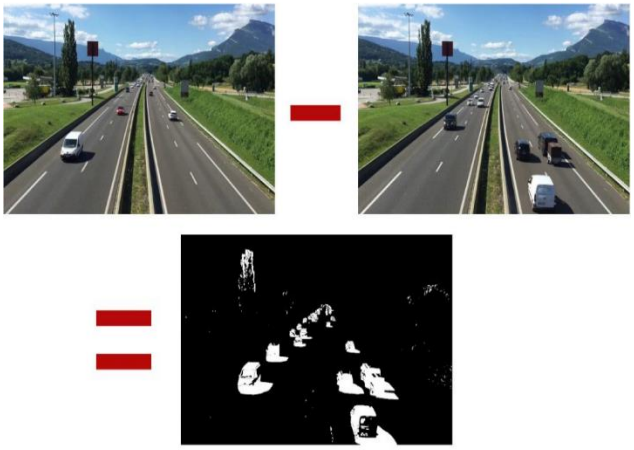

Figura 2 – Processo de *Background Subtraction*. Neste exemplo, os carros correspondem à alteração na cena.

A máscara de *foregroud* é calculada através da subtração da *frame* atual e de uma *frame* anterior, um *background*, que corresponde à parte estática da cena.

No desenvolvimento de um algoritmo de *background subtraction* é necessário considerar os seguintes aspetos:

- Inicialização do *background*;
- Atualização do *background*;
- Cálculo da diferença entre a *frame* atual e o *background*.

II. OBJETIVOS E IMPLEMENTAÇÃO

A. *Obtenção de vídeo a partir de uma câmara*

Para a implementação do sistema pretendido, era necessário, em primeiro lugar, obter *frames* de uma câmara. Para tal foi utilizada a classe `VideoCapture` do OpenCV.

Embora a classe `VideoCapture` permita a utilização de ficheiros de vídeo em vez de uma câmara, optou-se por testar o sistema sempre com câmaras.

B. *Deteção de alterações utilizando um algoritmo simples (Versão AbsDiff)*

Para a deteção das alterações foi desenvolvido em primeiro lugar um algoritmo simples baseado na diferença entre 2 *frames*:

1. Obter *frame* inicial, o *background*;
2. Enquanto a câmara está ligada:
   I. Obter frame atual;
   II. Fazer a diferença entre o *background* e a frame atual;
   Utilizando a função `absdiff`, que calcula a diferença entre 2 frames (arrays de pixéis) por elemento.
   III. Procurar contornos na diferença obtida; Utilizando a função `findContours`.
   Esses contornos vão corresponder aos contornos de zonas em que ocorreram alterações.
   IV. Desenhar retângulos com base nos contornos obtidos;
   Só rectângulos com área superior a um valor fixo são desenhados, permitindo filtar ruido associado a pequenas alterações de cor entre *frames* (que podem ocorrer por exemplo por cintilação da imagem).
   V. [Opcional]
   Periodicamente, atualizar o *background.*

A atualização do *background* de forma periódica torna possível a utilização do sistema numa situação real, já que permite que ele se adapte a novos cenários sem ser necessário reiniciá-lo. Isto também permite que o sistema seja usável com câmaras não fixas, onde o *background* não é expectável que seja constante.

C. *Utilização de algoritmos built-in para background subtraction (Versão MOG2/KNN)*

O algoritmo descrito em *Background subtraction [1]*, embora funcional, necessita de um passo adicional de actualização do *background*. Embora seja de implementação simples, os critérios para decidir se o *background* deve ser actualizado (por exemplo actualização ao fim de um intervalo de tempo fixo) não são universais e dependem do contexto em que a aplicação irá ser utilizada.

Dos diferentes algoritmos *built-in* na biblioteca OpenCV [1], e tendo em conta que não era desejável que aplicação desenvolvida dependensse da existência de GPUs com suporte à API NVIDIA CUDA, foram escolhidos 2 algoritmos para implementar sistemas com funcionalidade equivalente ao que o algoritmo descrito em *Background subtraction [1]* possibilita: o MOG2 (*Mixture of Gaussians 2*) e o KNN (*k-nearest neighbour*).

Para a deteção das alterações utilizando esses algoritmos *built-in*, o algoritmo inicial foi ligeiramente alterado:

1. Inicializar o *Background Subtractor*;
   Na inicialização, podem ser definidos valores (ver Parâmetros de configuração).
2. Enquanto a câmara está ligada:
   I. Obter frame atual;
   II. Obter a máscara de *foreground;*
   Utilizando a função `apply`.
   III. Procurar contornos na diferença obtida; Utilizando a função `findContours`.
   Esses contornos vão corresponder aos contornos de zonas em que ocorreram alterações.
   IV. Desenhar retângulos com base nos contornos obtidos. Ruído contina a ser filtrado como anteriormente.

É de notar que, ao contrário da primeira abordagem, deixa de ser necessário actualizar o *background* para a aplicação funcionar adequadamente em situações em que a câmara não está fixa, que constitui a principal vantagem desta abordagem (para além da reutilização de algoritmos já implementados).

Algoritmo MOG2

O algoritmo MOG (*Mixture of Gaussians*) baseia-se na mistura de *k* distribuições guassianas, que modelam cada pixel do *background*. Cada distribuição representa cada cor diferente do *background* ou do *foreground*.

O peso de cada uma dessas distribuições usados no modelo é proporcional à quantidade de tempo que cada cor está no pixel, o que significa que quando o peso de uma distribuição de pixels é baixo, o pixel é classificado como sendo pertencente ao *foreground*.

O algoritmo MOG2 resolve uma das limitações do algoritmo MOG, introduzindo a aleatoriedade do número *k* de distribuições guassianas.

Na implementação da biblioteca OpenCV, o algoritmo MOG2 (`cv:: createBackgroundSubtractorMOG2`) pode ser configurado com 3 parâmetros:

- *history*: número de *frames* utilizadas para modelar o *background*;
- *varThreshold*: correlação entre o peso dos pixéis na *frame* actual e valores no modelo;
- *detectShadows*: activar/desactivar a detecção de sombras;

A partir destes parâmetros é possível perceber que a modelação do background é muito mais sofisticada do que a abordagem descrita *Background subtraction [1]*, em que o background era apenas 1 *frame*. Isto permite que o algoritmo possa adaptar-se mais suavemente a mudanças no *background* e sem necessidade de passos adicionais.

Algoritmo KNN [3]

O algoritmo KNN (*k-nearest neighbour*) baseia-se na aplicação do algoritmo de classificação KNN. No algoritmo KNN, cada elemento a ser classificado é classificado tendo em conta a classificação dos *k* elementos que lhe estão mais próximos. Para evitar empates, o valor de *k* deve ser ímpar.

Na implementação da biblioteca OpenCV, o algoritmo KNN (`cv:: createBackgroundSubtractorKNN`) pode ser configurado com 3 parâmetros:

- *history*: número de *frames* utilizadas para modelar o *background*;
- *dist2Threshold*: correlação entre o peso dos pixéis na *frame* actual e valores no modelo;
- *detectShadows*: activar/desactivar a detecção de sombras;

III. APLICAÇÃO DESENVOLVIDA

A. *Compilar e executar*

Para executar o sistema implementado, é necessário instalar o OpenCV 3.2.0 ou superior e compilar o ficheiro `change_detector.cpp`:

```
g++ change_detector.cpp -o change_detector `pkg-config --cflags --libs opencv`
```

Para o executar basta executar o programa compilado

```
$ ./change_detector
```

B. *Parâmetros de configuração*

Tendo em vista a fácil ateração dos algoritmos, foram definidas algumas constantes no início do código-fonte do programa, que se encontram descritas na Tabela 1.

Parâmetro	**Propósito**	**Default**
MIN_AREA	Área mínima do rectângulos que podem ser desenhado (permitindo filtrar ruído)	500
UPDATE_AREA	Taxa de actualização do *background* (apenas para *Background subtraction [1]*)	100
MOG2history *KNNhistory*	Número de *frames* para modelar *background* (apenas para MOG2/KNN)	500 500
MOG2varThreshold *KNNdist2Threshold*	Correlação entre o peso dos pixéis na frame actual e valores no modelo (apenas para MOG2/KNN)	16.0 400.0

Tabela 1 – Parâmetros de configuração

IV. RESULTADOS & DISCUSSÃO

A implementação de 3 algoritmos diferentes e a alteração dos parâmetros de configuração permitem uma fácil alteração do comportamento do programa.

A. *Comparação entre algoritmos*

Para comparar a eficácia dos diferentes algoritmos, foi executado um teste em que os algoritmos foram configurados com valores por omissão. Alterações na cena foram marcadas utilizando rectângulos e com o texto *Change/No Change* no canto superior esquerdo.

Figura 3 – Situação inicial

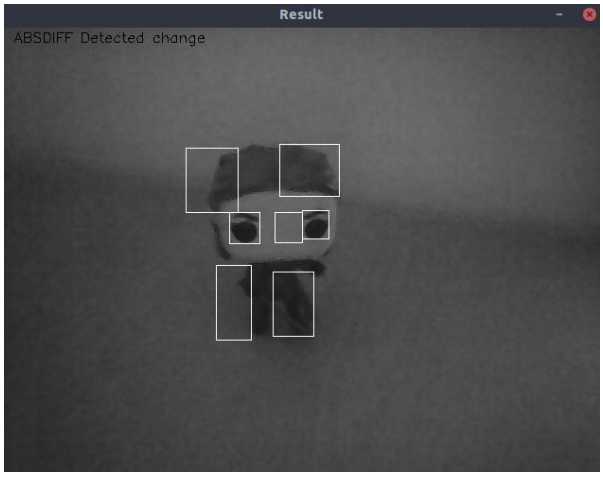

Figura 4 – Deteção de alterações (versão *AbsDiff)*

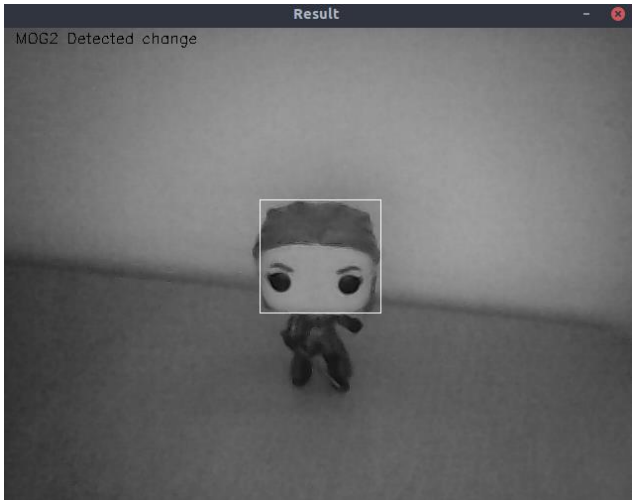

Figura 5 – Deteção de alterações (versão *MOG2*)

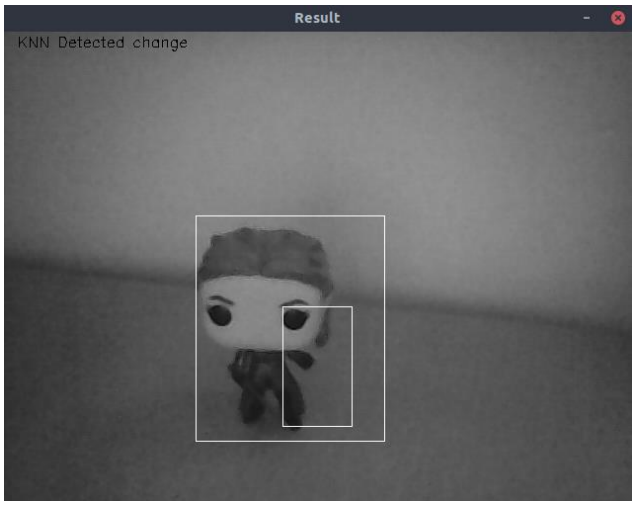

Figura 6 – Deteção de alterações (versão *KNN*)

As 3 versões do sistema conseguem, com sucesso considerável, detectar alterações na cena. Contudo, já é possível ver algumas diferenças na eficácia:

Na versão *AbsDiff*, o cubo não é detectado completamente mas só partes deste, enquanto que os algoritmos *MOG2* e *KNN* detectam correctamente o cubo. Entre os 2 algoritmos, o MOG2 destaca-se por conseguir detectar exactamente o cubo, enquanto que a versão KNN consegue detectar não só o cubo mas também a sua sombra.

B. Alteração dos parâmetros do algoritmo MOG2 e KNN

Para perceber o impacto da alteração dos parâmetros descritos em Parâmetros de configuração, foi executado um teste com o algoritmo MOG2, onde foram alterados os parâmetros *MOG2History* e *MOG2varThreshold*.

Figura 7 – Deteção de alterações com valor de *history* e *threshold* por omissão (500 e 16.0)

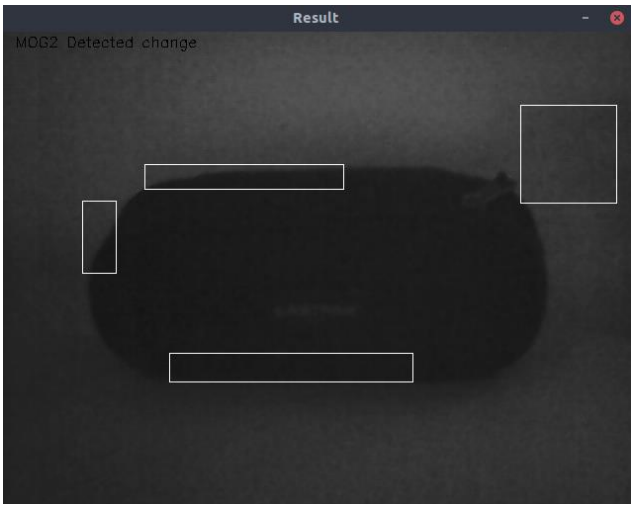

Figura 8 – Deteção de alterações com valor de *history* maior (5×10^6)

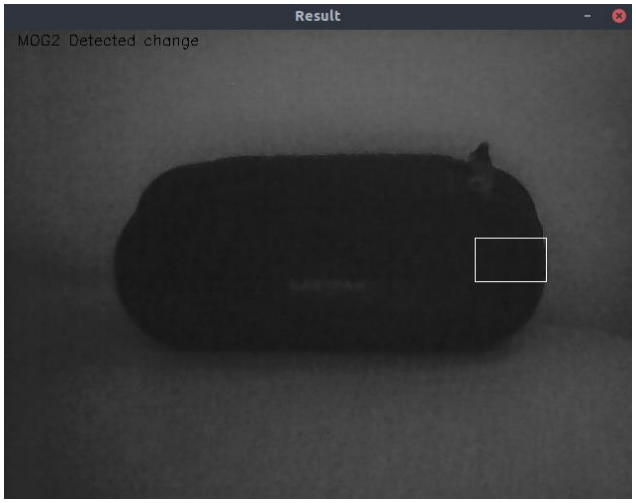

Figura 9 – Deteção de alterações com valor de *history* menor (5)

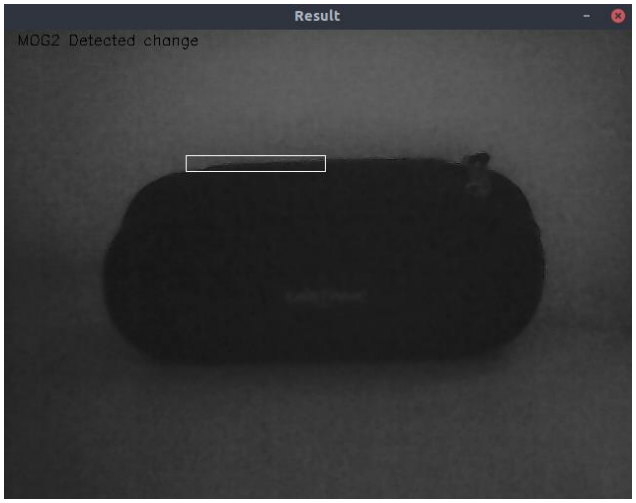

Figura 10 – Deteção de alterações com valor de *threshold* maior (100.0)

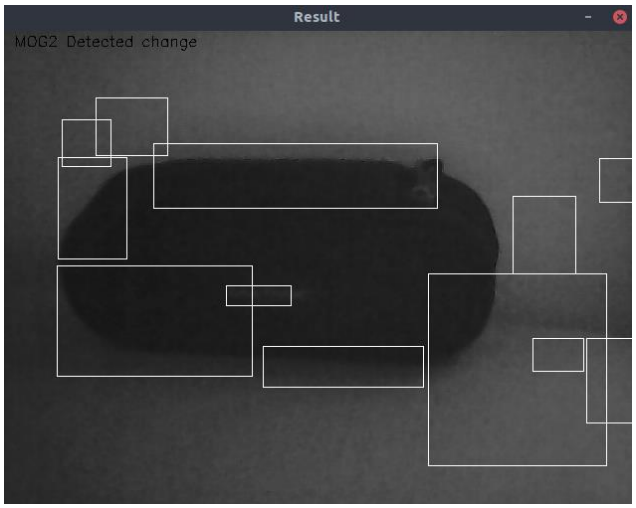

Figura 11 – Detecção de alterações com valor de *threshold* menor (5)

É possível concluir que a alteração dos valores de *history* não tem qualquer impacto na sensibilidade do programa a detectar alterações mas sim na rapidez com que as alterações deixam de ser detectadas e passam a ser consideradas como *background*.

Valores muito grandes de *history* aumentam o intervalo de tempo que uma alteração não faz parte do *background*, enquanto que valores muito pequenos diminuem o intervalo de tempo e podem até tornar humanamente impossível ver que foram detectadas alterações.

A alteração dos valores de *threshold* já têm impacto na detecção das alterações. Valores de *threshold* menores permitem detectar cada vez mais alterações mais pequenas, enquanto que um valor de *threshold* maior torna cada vez mais difícil detectar alterações pequenas, podendo no máximo ser possível apenas detectar que existe 1 alteração (na cena toda, como na Figura 11).

Conclui-se que o valor de *history* e *threshold* deve portanto sofrer ajustes, tendo em conta o contexto em que o sistema vai ser aplicado mas verificou-se que os valores por omissão funcionam bem na maioria das situações.

C. Filtragem de ruído

Para perceber o impacto da alteração do parâmetro de *MIN_AREA*, necessário para a filtragem de ruído, foi executado um teste com o algoritmo KNN com os parâmetros *KNNhistory* e *KNNdist2Threshold* com os valores por omissão, onde o parâmetro *MIN_AREA* foi alterado.

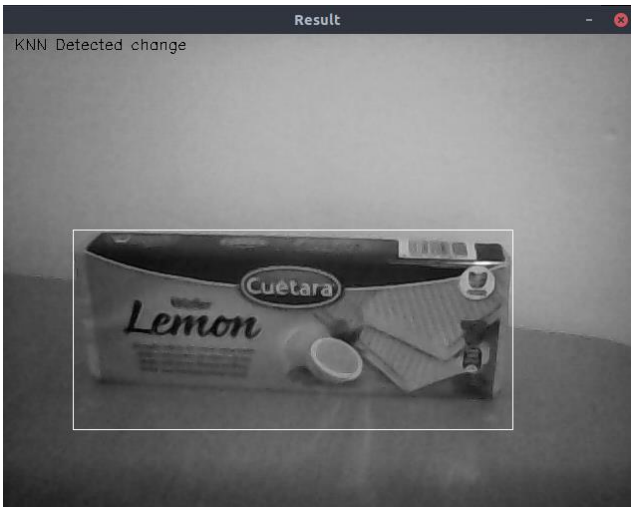

Figura 12 – Deteção de alterações com valor de *MIN_AREA* por omissão (500)

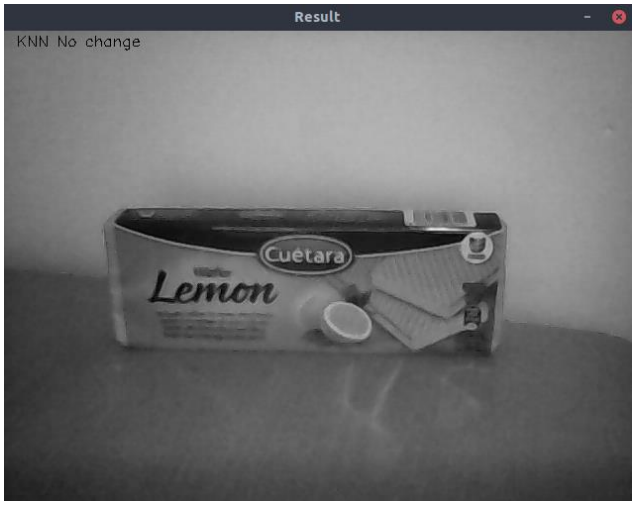

Figura 13 – Deteção de alterações com valor de *MIN_AREA* maior (5 x 10^6)

Figura 14 – Detecção de alterações com valor de *MIN_AREA* menor (5)

Valores maiores de *MIN_AREA* filtram mais alterações, podendo, se forem valores extremamente grandes, filtrar todas as alterações (como na Figura 13).

Por oposição, valores mais pequenos vão filtrar menos alterações, podendo, se for demasiado baixo, levar não só ao aparecimento de muito ruido (como na Figura 14), mas a muita instabilidade na detecção do mesmo.

V. CONCLUSÕES

Neste artigo foram apresentadas várias abordagens para a detecção de alterações numa cena: uma abordagem simples baseada na diferença entre *frames* (versão *AbsDiff*) e 2 abordagens mais complexas, baseadas em algoritmos de *background subtraction* implementados no OpenCV (versão *MOG2* e *KNN*).

Embora com limitações claras, sobretudo ao nível da actualização do *background* (que tem de ser feita manualmente), a abordagem mais simples baseada na diferença entre *frames* (versão *AbsDiff*) conseguiu resultados aceitáveis nos cenários testados. Contudo, continua a ser evidente os melhores resultados que os algoritmos *MOG2* ou *KNN* apresentaram.

Os testes efectuados permitem concluir que, embora sejam possíveis ajustes nos parâmetros dos algoritmos de *background subtraction*, os resultados ideais para a maioria dos cenários são obtiveis com valores próximos dos valores por omissão.

Já a necessidade de filtar ruído, através da filtragem de rectângulos de alterações de área inferior a um dado valor foi crítica para a obtenção de resultados aceitáveis. A determinação desse valor também é crítica para a obtenção de bons resultados.

Destaca-se por último a facilidade, em termos de implementação, de soluções de detecção de alterações numa cena com bom desempenho e eficácia, utilizando a biblioteca OpenCV.

REFERÊNCIAS

[1] How to Use Background Subtraction Methods
https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html
Acedido em Janeiro de 2021

[2] cv::BackgroundSubtractor Class Reference
https://docs.opencv.org/3.4/d7/df6/classcv_1_1BackgroundSubtractor.html
Acedido em Janeiro de 2021

[3] Understanding k-Nearest Neighbour
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_ml/py_knn/py_knn_understanding/py_knn_understanding.html#knn-understandingc
Acedido em Janeiro de 2021