DM

OpenCV Studio

MASTER DISSERTATION

Juan Aurélio Soares da Silva
MASTER IN INFORMATICS ENGINEERING

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

September | 2024

OpenCV Studio

MASTER DISSERTATION

Juan Aurélio Soares da Silva
MASTER IN INFORMATICS ENGINEERING

SUPERVISOR
Leonel Domingos Telo Nóbrega

Faculdade de Ciências Exatas e da Engenharia

Mestrado em Engenharia Informática

OpenCV Studio

Juan Aurélio Soares da Silva

Orientado por:

Leonel Domingos Telo Nóbrega

11 de janeiro de 2025

Resumo

A visão computacional tem gerado cada vez mais interesse devido às suas aplicações no campo da inteligência artificial. Para suportar estes sistemas que utilizam a visão computacional, surgiram bibliotecas que visam facilitar a implementação de comportamentos e funcionalidades, como é o caso da biblioteca ***OpenCV***. Esta é uma biblioteca complexa, constituída por diversos módulos que disponibilizam inúmeros métodos, funções e algoritmos.

Foi desenvolvida uma plataforma *low-code*, intitulada ***OpenCV Studio***, que tem como principal objetivo elevar o nível de abstração da biblioteca ***OpenCV*** a fim de permitir o rápido desenvolvimento de sistemas no campo da visão computacional de uma forma simples e intuitiva.

A avaliação da plataforma desenvolvida consistiu na realização de quatro tarefas por três participantes. Estas realizadas em programação tradicional, recorrendo à linguagem de programação C++, e posteriormente recorrendo à plataforma desenvolvida. Na maioria das tarefas verificou-se uma diminuição de cerca de 50% no tempo necessário à implementação e conclusão das tarefas através da plataforma *low-code*.

Durante a avaliação foram recolhidos os comentários e as sugestões dos participantes, através da metodologia *Think Aloud*. Os participantes responderam também a um formulário *SUS*, para se obter uma classificação de usabilidade da plataforma.

Ambas as avaliações apresentaram resultados positivos. Os participantes consideraram a plataforma intuitiva e fácil de utilizar. A avaliação quantitativa, utilizando a *System Usability Scale*, confirmou estes resultados apresentando uma pontuação de 84 pontos.

Keywords: Visão Computacional · OpenCV · Low-code · Desenvolvimento Visual · Citizen Developers

Abstract

Computer vision has been generating increasing interest due to its applicability in the field of artificial intelligence. To support systems that use computer vision, libraries have emerged aimed at facilitating the implementation of behaviors and functionalities, such as the ***OpenCV*** library. This is a complex library, consisting of several modules that provide numerous methods, functions, and algorithms.

A *low-code* platform, called ***OpenCV Studio***, has been developed with the main goal of raising the level of abstraction of the ***OpenCV*** library, enabling the rapid development of computer vision systems in a simple and intuitive way.

The evaluation of the developed platform consisted of completing four tasks by three participants. These tasks were first carried out using traditional programming in C++, and then using the developed platform. In most cases, the time required to implement and complete the tasks using the low-code platform was reduced by approximately 50%.

During the evaluation, participants' comments and suggestions were collected using the *Think Aloud* methodology. Participants also completed a *SUS* questionnaire to assess the platform's usability.

Both evaluations yielded positive results. The participants found the platform intuitive and easy to use. The quantitative evaluation, using the *System Usability Scale*, confirmed these findings with a score of 84 points.

Keywords: Computer Vision · OpenCV · Low-code · Visual Development · Citizen Developers

Agradecimentos

Em primeiro lugar, expresso a minha mais profunda gratidão ao meu orientador, professor Leonel Nóbrega, pela sua disponibilidade, apoio, e especialmente agradeço pelos conhecimentos que me transmitiu ao longo desta jornada, essenciais para a conclusão desta dissertação.

Aos meus colegas de trabalho, deixo o meu sincero agradecimento por motivarem-me e por proporcionarem-me tempo para dedicar a este projeto.

Sou grato aos professores e alunos de doutoramento, que generosamente disponibilizaram o seu tempo para a realização das tarefas de avaliação da plataforma desenvolvida nesta dissertação.

Um agradecimento especial para a minha família, especialmente aos meus pais, Dorita e Aurélio Soares, e à minha tia por todo o apoio incondicional e por nunca permitirem que eu desistisse. Ao meu irmão, Toni Silva, que, sem se aperceber, me motivou de uma forma especial a continuar esta dissertação, pois mostrou que, mesmo com dificuldades, não devemos desistir daquilo de que gostamos.

Aos meus amigos da banda *Os Infantes*, em particular ao chefe Marco Correia, o meu obrigado por motivar-me, à sua maneira, a persistir no curso mesmo quando a vontade de desistir era forte.

Por fim, agradeço à minha namorada, Jéni Monroy, por estar ao meu lado nos momentos de maior *stress* e nervosismo, sempre pronta a acalmar-me e a mostrar-me o lado positivo de tudo. Agradeço também pela sua paciência e por dedicar o seu tempo a motivar a acompanhar-me na escrita desta dissertação.

À minha madrinha, o meu anjo da guarda.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

API Application Programming Interface

BGR Blue Green Red

DSL Domain-Specific Languages

GML General Modeling Languages

GPL General-Purpose Programming Languages

GPML General-Purpose Modeling Languages

GPU Graphics Processing Unit

HDR High Dynamic Range

HSV Hue Saturation Value

HTML HyperText Markup Language

IPL Image Processing Library

JPG Joint Photographic Experts Group

JSON JavaScript Object Notation

MBE Model-Based Engineering

MDA Model-Driven Architecture

MDD Model-Driven Developement

MDE Model-Driven Engineering

OCR Optical Character Recognition

OMG Object Management Group

OpenCV Open Source Computer Vision Library

PNG Portable Network Graphics

RGB Red Green Blue

SOLID Princípios de software

SUS System Usability Scale

SVG Scalable Vector Graphics

UCI University of California Irvine

UI User Interface

UML Unified Modeling Language

URL Uniform Resource Locator

WYSIWYG What You See Is What You Get

1 Introdução

A visão computacional é uma área interdisciplinar da inteligência artificial, cujo o propósito principal é extrair informações úteis do mundo físico por meio de análise de imagens e vídeos utilizando recursos computacionais [8, 9].

Os primeiros trabalhos datam da década de 1970 e as tecnologias já desenvolvidas possuem aplicações em áreas como a robótica, automação industrial, processamento de documentos, deteção remota, navegação, microscopia, imagens médicas, entre outras [8]. Atualmente, fruto da combinação com a área das redes neuronais, são desenvolvidos sistemas inteligentes que utilizam a visão computacional para percecionar o ambiente onde operam, como é o caso dos veículos autónomos [10], ou dos robôs de exploração espacial [11], por exemplo, o caso do robô Curiosity [12].

Existem dois objetivos no campo da visão computacional: o primeiro diz respeito à criação de sistemas informáticos que consigam "ver" e o segundo diz respeito à compreensão da visão biológica [8, 13]. O primeiro objetivo, que se encontra explicitamente relacionado com os sistemas informáticos, é onde assenta esta dissertação. De forma a facilitar o desenvolvimento e a criação destes sistemas informáticos, surgiram diversas bibliotecas de suporte como o ***OpenCV***, o ***SimpleCV***, o ***BoofCV***, entre outras [14, 15].

Como referido no parágrafo anterior, o ***SimpleCV*** trata-se de uma biblioteca, na linguagem de programação *Python*, que agrupa diversos algoritmos e ferramentas *open-source* dedicados à visão computacional. Esta utiliza por base a biblioteca ***OpenCV***, introduzindo uma camada de abstração aos recursos e às funcionalidades da mesma. A abstração entregue por esta biblioteca fornece uma *interface* de utilização mais simples para os utilizadores com capacidades de programação, reservando-os da necessidade de aprender cada detalhe de implementação do ***OpenCV***. Esta biblioteca apresenta também funcionalidades próprias além das funcionalidades entregues pelo ***OpenCV*** [14].

O ***BoofCV***, é outra biblioteca destinada aos desafios que envolvem a visão computacional. Porém, esta foi desenvolvida totalmente na linguagem *Java*. São disponibilizadas diversas ferramentas e algoritmos eficazes e acessíveis aos utilizadores que pretendem desenvolver sistemas dedicados à visão computacional, utilizando a linguagem de programação *Java*.

Neste trabalho de dissertação, foi considerada apenas a biblioteca ***OpenCV***. Primeiro devido à sua popularidade, por ser gratuita, *open-source* e suportada por grandes e conhecidas empresas, das quais se destacam a Intel, o Google, a Microsoft, entre outras [16]. Outra forte característica importante surgiu do facto desta biblioteca ser a base para muitas de outras bibliotecas conhecidas como, por exemplo, a biblioteca ***SimpleCV***, introduzida no parágrafo anterior. Por fim, uma das capacidades mais relevantes consiste na diversidade de linguagens de programação disponibilizadas. Ao contrário da biblioteca ***BoofCV***, a biblioteca ***OpenCV*** não tem foco exclusivamente em *Java*, fornecendo *interfaces* de programação em *C*, *Python*, *Java*, entre outros.

1.1 OpenCV

A biblioteca ***OpenCV*** (*Open Source Computer Vision Library*) é amplamente utilizada no campo da visão computacional e processamento de imagem. A sua principal característica centra-se na disponibilização de implementações de inúmeros algoritmos, módulos e funções, cobrindo uma vasta gama de funcionalidades. Além disso, a biblioteca é fortemente suportada por uma vasta

comunidade ativa de programadores, o que dá origem a atualizações recorrentes, geralmente a cada seis meses [16].

Por tratar-se de uma biblioteca que engloba o conhecimento de várias áreas, como matemática, física e tecnologias específicas de sensores, o ***OpenCV*** representa a integração de diferentes tecnologias. Entre estas, destacam-se os sensores físicos e técnicas de processamento de imagem [9]. Além disso, a biblioteca oferece suporte a diversas linguagens de programação, como *C++*, *Python*, *Java*, *MATLAB* e, mais recentemente, *JavaScript*. Essa versatilidade é complementada pelo suporte a múltiplos sistemas operativos, incluindo *Windows*, *MacOS*, *Linux* e até mesmo *Android* [9, 17, 18].

1.2 Motivação

Por tratar-se de uma biblioteca que engloba o conhecimento de várias áreas, como matemática, física e tecnologias específicas de sensores, o ***OpenCV*** representa a integração de diferentes tecnologias. Entre estas, destacam-se os sensores físicos e técnicas de processamento de imagem [9]. Além disso, a biblioteca oferece suporte a diversas linguagens de programação, como *C++*, *Python*, *Java*, *MATLAB* e, mais recentemente, *JavaScript*. Essa versatilidade é complementada pelo suporte a múltiplos sistemas operativos, incluindo *Windows*, *MacOS*, *Linux* e até mesmo *Android* [9, 17, 18].

1.3 Objetivos

Para mitigar as dificuldades descritas anteriormente, é importante elevar o nível de abstração da biblioteca, através do redesenho da sua interface de programação (*API*) ou criando uma notação alternativa que seja posteriormente mapeada na *API* atual. Esta dissertação parte da hipótese de que a integração da ***OpenCV***. numa plataforma *low-code*, em combinação com uma abordagem *Model-Driven Engineering*, pode abstrair a complexidade associada à sua utilização, permitindo o desenvolvimento de sistemas de visão computacional de forma ágil, eficiente e acessível. A solução final deverá ser flexível e intuitiva para que qualquer utilizador, com ou sem experiência de programação, consiga utilizar as funcionalidades da biblioteca, sendo apenas necessário conhecer minimamente as funcionalidades gerais da biblioteca.

Contudo, é fundamental reforçar que a plataforma a desenvolver não se trata de uma aplicação de visão computacional suportada pelo *OpenCV*. Trata-se, em vez disso, de um gerador de código em C++ direcionado para a biblioteca *OpenCV*, com o objetivo de oferecer aos utilizadores da biblioteca uma ferramenta que lhes permita criar as suas próprias aplicações na linguagem nativa, conseguindo proveito da sua eficiência, reduzindo significativamente a curva de aprendizagem e a complexidade associada ao uso direto da biblioteca na linguagem C++.

Este protótipo deverá permitir o desenvolvimento de algumas sequências de filtros e funcionalidades de uma forma simples e intuitiva, reduzindo as dificuldades iniciais para novos utilizadores da biblioteca. O protótipo deverá então realizar a geração de código numa linguagem de programação, *C++*, de forma a permitir a sua utilização no ambiente de produção com o mínimo esforço, reduzindo ou eliminando a necessidade de reescrita de código além do estritamente necessário à integração do mesmo. Toda esta abordagem deverá reduzir a exigência de conhecimento de programação de baixo nível e da sintaxe detalhada da biblioteca.

De forma a se obter resultados sobre a eficácia e a utilidade do protótipo, este deverá ser alvo de avaliação por utilizadores que conhecem ou já fizeram uso desta biblioteca.

1.4 Contribuições

A principal contribuição consiste no desenvolvimento de um protótipo de uma plataforma *low-code*, no navegador *web*, que permite a geração de código, em *C++*, para algumas funcionalidades da biblioteca ***OpenCV***. Este protótipo permite ainda a pré-visualização das funcionalidades aplicadas, através da versão em *JavaScript* da biblioteca, sem a necessidade de realizar qualquer instalação ou configuração.

Outro contributo reside na escrita desta dissertação, onde são introduzidos os principais fundamentos, tais como a visão computacional, a biblioteca ***OpenCV***, o conceito *low-code* e a abordagem *Model-Driven Engineering*, sendo ainda descrito o método de desenvolvimento do protótipo, desde a definição dos seus modelos à sua respetiva avaliação, resultante da experiência realizada com utilizadores selecionados.

Pretende-se que esta plataforma *low-code* apresente valor aos programadores e potenciais utilizadores e que, com o crescimento e com o amadurecimento da plataforma, permita auxiliar e acelerar o desenvolvimento de sistemas envolvendo a biblioteca ***OpenCV*** e a visão computacional.

1.5 Estrutura do Documento

Esta dissertação está estruturada em seis capítulos, cada um abordando uma fase fundamental do processo realizado. A organização destes capítulos foi pensada de forma a introduzir os conceitos antes de se apresentar todo o processo de implementação prática do protótipo.

No capítulo 1, está definida a introdução, onde são apresentados a motivação e os objetivos desta dissertação. É neste capítulo que se introduz a complexidade da biblioteca ***OpenCV*** bem como o objetivo fundamental de se elevar o seu nível de abstração.

Os fundamentos estão descritos ao longo do segundo capítulo. Inicia-se o capítulo com a introdução aos conceitos e típicas aplicações da visão computacional. São também introduzidas algumas das bibliotecas que permitem e facilitam o desenvolvimento de aplicações nesta área, focando-se em especial na biblioteca ***OpenCV***. Ainda neste capítulo é apresentada a abordagem *Model-Driven Engineering*, introduzindo-se os diversos paradigmas, linguagens de modelação e os níveis de abstração presentes nesta abordagem. Finaliza-se este capítulo com referência ao conceito *low-code* e restantes termos associados, mencionando-se também as suas fases de desenvolvimento, vantagens e algumas das plataformas existentes.

No terceiro capítulo é realizada toda a análise aos desafios e definidos os requisitos de implementação à plataforma *low-code* a desenvolver. Para tal são comparadas diversas plataformas de suporte às interações típicas de *low-code* e estudada a estratégia de abstração a ser utilizada. Termina-se este capítulo descrevendo as decisões tomadas para a implementação do protótipo, incluindo os blocos e funcionalidades a implementar, e as *frameworks* ou tecnologias a utilizar.

Todo o processo de desenvolvimento do protótipo encontra-se detalhado etapa por etapa no capítulo seguinte. Inicia-se o quarto capítulo com uma análise profunda à plataforma de suporte escolhida, descrevendo-se as suas principais classes e funcionalidades. No fim deste capítulo são descritas as dificuldades e as limitações das tecnologias utilizadas e no processo de implementação, e as soluções encontradas para as superar.

O capítulo cinco, a avaliação, introduz os conceitos relacionados às metodologias de avaliação de sistemas. São introduzidas as técnicas ***Think Aloud*** e a escala de usabilidade *SUS*. Neste capítulo são descritos os resultados da avaliação e apresentadas a suas classificações para cada uma das afirmações do questionário *SUS*, bem como a classificação final. Termina-se este capítulo ao referir as limitações da avaliação realizada.

Por fim, no último capítulo são apresentadas as principais conclusões, acompanhadas pelos objetivos futuros, que incluem as sugestões obtidas no processo de avaliação do protótipo e as diversas funcionalidades que foram identificadas durante o desenvolvimento.

2 Revisão de Literatura

Para se enquandrar os conceitos fundamentais à concretização dos objetivos mencionados na secção anterior, começa-se por introduzir a visão computacional e as suas principais definições. Passando-se, posteriormente, a contextualizar a biblioteca alvo desta dissertação, ***OpenCV***, desde o seu surgimento, tendo em consideração a sua estrutura, alguns dos seus algoritmos e das suas funcionalidades. É ainda introduzida a abordagem *Model-Driven Engineering*, dando ênfase à sua importância no processo de abstração de um sistema através de modelos e de atividades como a modelação e meta-modelação. Por fim, é introduzido o termo Low-code e as respetivas plataformas, que são geralmente inspiradas na abordagem *Model-Driven Engineering*, descrevendo os seus principais objetivos e casos de utilização, apresentando os principais benefícios e limitações provenientes da utilização destas plataformas.

2.1 Visão Computacional

A Visão Computacional é a ciência de criar uma capacidade semelhante, ou se possível melhor, ao olho humano em conjunto com o processamento do cérebro [14].

A visão computacional tem associado a si diversas terminologias, tais como *machine vision*, *computer vision*, *image understanding*, *robot vision*, *image analysis* e *scene analysis*. Cada uma destas nomenclatura possui uma diferente perspetiva histórica, por exemplo, o *Machine Vision* é frequentemente utilizado em disciplinas de engenharia, sendo assim mais orientado para a engenharia e para as aplicações. Já o termo de Visão Computacional surge da investigação de várias áreas, como da física e da neurociência [8].

A visão é um exemplo clássico de um problema no qual o ser humano consegue superar a capacidade limitada das máquinas. O olho humano consegue recolher uma enorme quantidade de informação visual, que é processa pelo cérebro sem a necessidade do ser humano pensar no processamento destas informações [14]. Considerando a leitura de um livro como exemplo, enquanto o olho captura informações visuais, o cérebro tem como responsabilidade todo o processamento destas informações, começando pela distinção entre a representação do livro e o ambiente ao redor, que deve ser ignorado. Após o foco do cérebro se centrar no livro, é necessário transformar o conteúdo do mesmo em informação relevante, reconhecendo letras, palavras e frases. Através deste exemplo, torna-se explícito toda a capacidade do ser humano, principalmente considerando esta tarefa de leitura como uma constante recolha de informação ao longo do tempo [14].

A visão computacional possuí algumas restrições, a nível do ambiente onde opera, que comprometem a sua flexibilidade, tais como a luminosidade do cenário sendo necessário a que os objetos tenham que ser posicionados antes da captura da imagem. Isto porque o mundo real encontra-se exorbitante de variações que afetam a captura de imagens, o que acaba por desafiar os melhores algoritmos de visão computacional na tarefa de extrair a "essência" ou as características invariantes dos objetos [19].

Esta área, no que diz respeito à investigação, é extremamente desafiante, devido ao sistema visual do ser humano ser extremamente complexo e apresentar capacidades notáveis em diversas tarefas. O ser humano consegue reconhecer rostos em diversas condições de iluminação, pontos de vista, expressões, entre outros. Muitas vezes não apresenta dificuldade no reconhecimento de uma pessoa próxima presente numa fotografia antiga, retirada há muitos anos atrás. A capacidade

humana aparenta também não possuir limite ao número de rostos que consegue memorizar para reconhecimento futuro.

Toda esta capacidade da visão humana diminuí drásticamente a perspetiva de criação de um sistema autónomo com semelhante desempenho ao ser humano. Uma das maiores dificuldades prende-se com os algoritmos desenvolvidos, que funcionam em determinados casos mas não em todos, principalmente quando devem ser consideradas diversas condições como a iluminação, os ângulos e os pontos de vista [13].

Porém, existem tarefas na visão computacional em que os computadores desempenham um melhor aproveitamento que o ser humano. Tais tarefas envolvem olhar para um mesmo objeto durante uma infinidade de horas, conseguem determinar as bordas (mais concretamente *edges*), isto é, conseguem identificar numa imagem onde os pixéis seguidos possuem diferentes cores. Conseguem ainda identificar onde os pixéis partilham cores semelhantes, fornecendo medições de formas, e conseguem comparar imagens e identificar de forma precisa as diferenças existentes entre elas [14].

Existem diversas operações que podem ser realizadas sobre imagens. Estas operações podem ser agrupadas em diferentes categorias dependendo da estrutura, do nível e do propósito. Algumas operações podem ter o simples propósito de melhorar a imagem de forma a que se torne nítida ao ser humano, outras podem ter o objetivo de extrair informação para depois ser utilizada em sistemas de inteligência artificial [19].

2.1.1 Cronologia

Foi no início dos anos 70 que surgiu a visão computacional, vista na época como a componente de perceção visual capaz de imitar a inteligência humana, dotando robôs com comportamentos inteligentes. O que distinguia esta área do processamento de imagem era o objetivo da recolha de estruturas tridimensionais do mundo a partir de imagens e como utilizá-las para a compreensão da cena[1] [20].

Nas duas décadas seguintes, focaram-se em técnicas matemáticas que permitissem realizar análises quantitativas de imagem e de cena. Começaram a surgir técnicas para a junção de imagens e começaram também a se dedicar à identificação de bordas e contornos. Dedicaram-se também a otimização das técnicas já existentes, surgindo os algoritmos *stereo* que permitiam produzir superfícies completas em três dimensões. Para tal, recorrendo a duas ou mais fotografias de um espaço e, sabendo a posição da camera em cada fotografia, tornou-se possível encontrar correspondências nas fotografias. Estes algoritmos conseguem estimar profundidades detalhadas, o que permite a criação de modelos 3D das superfícies presentes nas fotografias [20].

No final da década de 90, começaram a surgir técnicas de aprendizagem estatística que permitiam realizar toda uma análise a utilizar nas técnicas de reconhecimento facial. Ainda de realçar que, nesta época, a ideia de manipulação de imagens do mundo real para a criação de novas animações ganhou relevo devido ao surgimento de técnicas de transformação de imagem e, mais tarde, devido à interpolação de vistas e à união de imagens panorâmicas. Na mesma década ainda, estavam sendo introduzidas técnicas de modelação à base de imagens, que permitiam criar modelos 3D realísticos a partir de conjuntos de imagens [20].

Já no novo milénio, deu-se o surgimento na aprendizagem de técnicas com base em características, utilizadas para o reconhecimento de objetos, dominando ainda outras tarefas de reconhe-

[1] Entenda-se por cena como o contexto em que ocorre o objeto de uma determinada imagem.

cimento, entre as quais o reconhecimento de cena e de localização. A principal preponderância atualmente centra-se em técnicas sofisticadas de aprendizagem automática aplicados a problemas de visão computacional [20].

Olhando para o passado da visão computacional, torna-se relevante o espectro das potenciais aplicações, que tipicamente une a visão computacional com outras áreas [13]. A psicologia humana da perceção é uma das áreas considerada muito importante, pois o estudo da capacidade do ser humano em compreender uma imagem pode guiar o desenvolvimento dos algoritmos da computação visual. Outras áreas de extrema importância são a Física, devido ao estudo da ótica e da ciência das cores, a Matemática, pois são utilizados nos algoritmos uma diversidade de modelos matemáticos e estatísticos, probabilidades, geometria analítica, entre outros [19]. Existem diversas áreas técnológicas, que se relacionam com a visão computacional, tais como a área de *Image Processing*, que envolve transformações de imagens, de *Computer Graphics*, que envolve descrições às transformações realizadas em imagens, e de *Pattern Recognition*, que envolve transformações de padrões em classes [8].

2.1.2 Níveis de Processamento

Como referido no parágrafo anterior, a Visão Computacional está fortemente relacionada com diversas áreas, sendo importante distinguir o conceito de *Image Processing*, ou processamento de imagem, do conceito de *image understanding*, ou compreensão de imagem. O primeiro conceito, *image processing*, destina-se à transformação de imagens em imagens, sendo utilizado frequentemente para dar suporte ao segundo conceito, compreensão de imagem. Por sua vez o conceito de *image understanding* destina-se à tomada de decisões com base em imagens, construindo as descrições de cenário necessárias à essa tomada de decisão [19].

O processamento da visão computacional é geralmente dividido em dois níveis:

- **Visão inicial**: também conhecida como visão de baixo nível, envolve as primeiras fases de processamento necessárias para uma tarefa visual. Um aspeto desta primeira fase é a análise de características, em que é extraída informação sobre cor, movimento, forma, textura, profundidade estéreo, e intensidade dos contornos. Outro aspeto da visão inicial é a segmentação da imagem, em que a informação das características é utilizada para segmentar a imagem em regiões que têm uma elevada probabilidade de ter surgido a partir de uma única causa física.

- **Análise de cena**: também referida como visão de alto nível, envolve mais processamento com base no conhecimento resultante da visão inicial. Nesta segunda fase é realizada a recolha dos descritores característicos gerados pela visão inicial e são construídas descrições, com um nível mais elevado, da cena. Algumas das aplicações deste nível envolvem a análise de formas, o reconhecimento de objetos e a localização de objetos.

Esta divisão entre a visão inicial e a análise de cena não é consensual. Alguns investigadores referem a existência de três níveis de visão, sendo de baixo, intermédio e alto nível [8]. Devido a esta alternativa, alguns especialistas referem que o objetivo da visão computacional consiste na construção de descrições de cenas a partir de imagens [19].

2.1.3 Principais Aplicações

Por fim é de expor que a visão computacional é utilizada numa variedade de aplicações de casos reais atualmente, sendo de destacar algumas das mais conhecidas em seguida, realçando que existem várias e diversas outras aplicações [20]:

- ***Optical character recognition (OCR)***: que é aplicado no reconhecimento de texto manuscrito e na identificação automática de números de matrícula em veículos, entre outras [20].
- ***Medical imaging***: que consiste por exemplo no registo de imagens antes e entre operações, ou até mesmo utilizado para longos estudos envolvendo a morfologia do cérebro de um paciente ao longo da sua vida [20, 21].
- ***Segurança automotiva***: consiste na deteção de obstáculos inesperados, como pedestres nas ruas em condições onde as técnicas de visão ativa não funcionem, isto é, visão que tem como fundamento sensores como radar ou lidar que, como qualquer tecnologia, têm as suas limitaçõe [20].
- ***Match Move***: consiste essencialmente na união de imagens geradas por computador a filmagens reais através de pontos característicos, para se estimar o movimento tridimensional da câmera e do ambiente (muito utilizado em filmes de cinema) [20].
- ***Reconhecimento de biométrica e de impressão digital***: utilizado para autenticação automática e em aplicações forenses [20].
- ***Vigilância***: permite monitorar intrusos, analisar tráfego nas autoestradas e até mesmo identificar pessoas a se afogar em piscinas [20].
- ***Deteção facial***: utilizado para melhorar o foco das câmeras para as pessoas, muito atual hoje em dia nos smartphones [20].

2.2 OpenCV

O ***OpenCV*** trata-se de uma biblioteca *open-source* de visão computacional desenvolvida pela Intel Corporation em 1999. Esta iniciativa teve o intuito de acelerar a visão computacional e a inteligência artificial, fornecendo uma infraestrutura solida, de forma a tornar a visão computacional mais acessível aos programadores e aos utilizadores na área de interação humano computador em tempo real, e na robótica [18, 22]. Esta biblioteca foi desenhada tendo em vista a eficiência computacional e tendo como o foco de aplicação as aplicações de tempo real. Para isso, a biblioteca é escrita em *C++* otimizado, podendo obter maior proveito devido aos processadores *multicore* [18].

O ***OpenCV*** é mais direcionado a fornecer as ferramentas básicas necessárias à resolução de problemas de visão computacional. As funcionalidades de mais alto nível são tipicamente suficientes para resolver problemas mais complexos, mas quando isto não acontece, os elementos de mais baixo nível são completos, ao ponto de permitirem o desenvolvimento de uma solução [18].

2.2.1 Cronologia

Como referido anteriormente, a Intel Corporation desenvolveu e apresentou a primeira versão, em *C*, da biblioteca ***OpenCV*** em 1999 [18].

Em 2009 foi introduzido a segunda versão do ***OpenCV***, responsável pelas grandes mudanças devido à introdução de implementações em *C++*. No lançamento da versão 2.2 são introduzidos

diversos módulos, ficando esta biblioteca com uma estrutura mais modular. Desde esta versão, a biblioteca encontra-se dividida em vários módulos, sendo que cada um destes é dedicado a um conjunto específico de desafios na área da visão computacional. Estes módulos tratam-se de ficheiros de biblioteca localizados na diretoria lib [17, 23].

No surgimento da terceira versão do ***OpenCV***, foram introduzidos novos algoritmos e novos módulos reorganizando determinados algoritmos e funcionalidades em diferentes módulos [17, 18, 23]. Um exemplo destas alterações foram as funcionalidades de escrita e leitura de imagens, que estavam presentes no módulo ***higui***, passaram a integrar o módulo ***imgcodecs*** [24]. É ainda considerável o surgimento de novos módulos entre a versão 3 e a versão atual, versão 4.6.0, sendo que já se encontra em fase de desenvolvimento a versão 5 [25].

A partir do ano de 2012, a fundação sem fins lucrativos OpenCV.org assumiu a tarefa de manter e atualizar o *website* de suporte aos programadores e aos utilizadores [24].

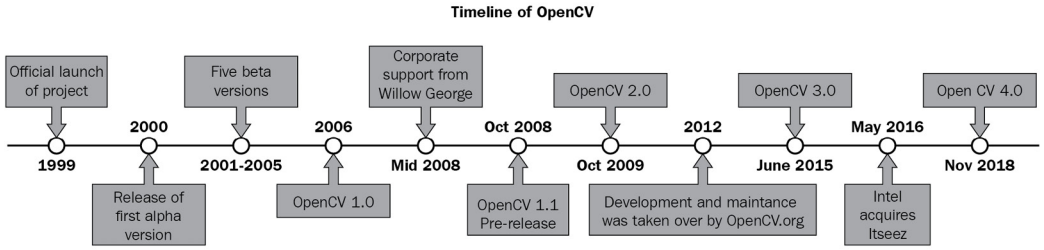

Figura 1. Linha cronológica das versões da biblioteca OpenCV até 2018 [1].

Na Figura 1 são apresentadas as versões introduzidas até ao ano 2018, que corresponde ao ano de lançamento da versão 4.0 [26], sendo que a biblioteca encontra-se atualmente na versão 4.6.0, e com a versão 5 em fase de desenvolvimento [16, 18].

Atualmente a biblioteca encontra-se disponível nos sistemas operativos mais populares, tais como *Windows*, *Linux*, *MacOS*, *iOS*, *Android* e outros. Possui ainda toda uma interface para outras linguagens de programação além do *C++*, tais como *Java*, *Python*, *MATLAB*, entre outras, e possuí rotinas para outras linguagens como *C#*, *Ruby* e *Perl* [24].

Existe ainda atualmente uma versão em *JavaScript* da biblioteca, *OpenCV.js*, que se trata de um vínculo a um subconjunto de funcionalidades da biblioteca para a *web*. Foi inicialmente criado no Parallel Architectures and Systems Group da Universidade da Califórnia, Irvine (UCI), como um projeto de pesquisa financiado pela Intel, tendo sido aprimorado e integrado o projeto oficial da biblioteca ***OpenCV*** como parte do programa Google Summer of Code de 2017 [27].

2.2.2 Estrutura

Todas as classes e métodos estão definidas no *namespace* cv, onde o principal objeto de toda a biblioteca é a classe ***Mat***, uma matriz que contém os valores dos pixéis de determinada imagem. Esta matriz é ainda constituída por um tamanho e por um tipo, este último possuí diversos valores, tais como *CV_8U* que se refere a imagens de 1 byte (8 bits), *CV_8UC3* que se refere a imagens coloridas com 3 canais, entre outros. De notar que a biblioteca utiliza os canais de cor na ordem contrária à tradicional, sendo Azul, Verde e Vermelho (*BGR*) [17].

Tal como descrito anteriormente, existem diversos valores para se aplicar ao tipo de uma matriz, onde os mais comumente utilizados são: *CV_8UC1*, *CV_8UC3*, *CV_8UC4*, *CV_32FC1*, *CV_32FC3*, *CV_32FC4*. Este tipo é especificado da seguinte forma:

$$CV\_numero\_tipoC(n)$$

onde **numero_tipo** pode estar entre 8 bits do tipo *unsigned* (8U) e 64 bits do tipo *float* (64F), **c** é constante, e **n** é o número de canais permitidos, desde 1 até a constante definida **CV_CN_MAX**, tipicamente limitada a 512

A biblioteca ***OpenCV*** permite então a criação de matrizes de diferentes tipos, como já mencionado, sendo importante realçar que determinadas operações apenas podem ser aplicadas de acordo com o tipo da matriz. Para combater esta limitação existem operações que permitem a conversão do tipo da matriz e, no caso do número de canais de cor na imagem, existem operações para separar (*split*) a matriz em 3 instâncias, cada uma das instâncias com um canal de cor (*Blue*, *Green*, *Red*) e para unir (*merge*) as 3 instâncias constituídas por um único canal numa única instância de 3 canais de cor [17].

Além disto, é fundamental destacar a presença de outros objetos e classes de extrema relevância dentro desta biblioteca e nos algoritmos que esta oferece. Nos próximos parágrafos, atribuiu-se atenção especial à exploração e descrição de algumas destas entidades essenciais.

Já foi introduzido nesta secção a noção de matriz e o seu funcionamento base, passando-se agora à classe ***Vec*** que é direcionada para vetores numéricos. Podem ser definidos qualquer tipo de vetor, com a sua capacidade de componentes variando desde os números inteiros até aos números reais, e com quantas dimensões forem precisas. Além disto, a classe ***Vec*** implementa operações aritméticas, incluindo somas, subtrações e multiplicações entre vetores [1].

Referindo a classe ***Vec*** torna-se necessário mencionar a classe ***Scalar***, por ser derivada desta com quatro dimensões. Esta é tipicamente utilizada para passar e ler os valores de um pixel, que normalmente são constituídos pelos três canais de cor: azul, verde e vermelho, além de uma propriedade auxiliar denominada *alpha*, onde é definida a opacidade do *pixel* [1].

Uma outra classe comumente utilizada é a classe ***Point***, que permite a definição de um ponto utilizando duas dimensões especificado pelas coordenadas x e y. Vale ressaltar que também existe a classe ***Point3*** que permite o mesmo comportamento, adicionando uma terceira dimensão e coordenada [1].

De realçar que existem outros tipos de objeto que não são aqui descritos, como a classe ***Size***, que é a última a ser mencionada e que permite especificar o tamanho de uma imagem ou retângulo. Esta classe recorre a dois membros, altura e largura, e a uma funcionalidade importante que permite calcular a respetiva área [1].

Em termos de funcionalidades, esta biblioteca implementa ferramentas para a manipulação de imagens, contendo mais de 2500 algoritmos destinados à deteção de rostos, identificação de objetos, classificação de ações humanas em vídeos, localização de objetos em movimento, extração de modelos 3D de objetos do mundo físico, identificação de imagens duplicadas, remoção de olhos vermelhos de fotografias tiradas com flash, reconhecimento de cenários, deteção e seguimento de movimentos oculares, interface de utilizador, calibração de câmera, entre outros [16, 17, 28].

Estes algoritmos têm por base estruturas de dados flexíveis, designadas por *Dynamic Data Structures*, agregadas com estruturas de dados complexas, presentes na *Image Processing Library* (*IPL*) [17, 28].

Com todas estas funcionalidades suportadas por inúmeros algoritmos, a biblioteca encontra-se, como já mencionado anteriormente, distribuída em módulos, cujos mais utilizados e conhecidos são descritos de seguida de forma muito breve e superficial [18, 23, 24]:

- O módulo principal, ***Core***, define as funções básicas e as operações aritméticas utilizadas pelos outros módulos e as estruturas de dados fundamentais.
- O módulo ***HighGui*** fornece simples interfaces de utilizador, *UI*, que podem ser usadas para apresentar imagens e para receber input dos utilizadores.
- O módulo ***ImgProc*** contém as funções para o processamento de imagem tais como filtros, transformações geométricas, conversão do espaço de cores, histogramas, entre outros.
- O módulo ***ImgCodecs*** trata-se de uma interface simples de utilizar para ler e escrever imagens.
- O módulo ***Photo*** trata-se de um módulo recente destinado à fotografia computacional incluindo *Inpainting*, que permite a conservação e recuperação de partes de imagens, redução de ruído, *High Dynamic Range* (*HDR*), entre outros.
- O módulo ***Stitching*** trata-se de um módulo recente que possuí algoritmos para a união de imagens.
- O módulo ***VideoIO*** trata-se de uma interface simples de utilizar para capturar e escrever vídeos.
- O módulo ***Video*** fornece funções de análise de vídeo tais como estimativa de movimento, extração de primeiro plano, rastreamento de objetos, entre outros.
- O módulo ***Features2d*** possuí funções para a deteção, descrição e correspondência de elementos, tais como, cantos e objetos planos.
- O módulo ***ObjDetect*** contém algoritmos para a deteção de objetos, tais como rostos, olhos, sorrisos, pessoas, carros, entre outros.
- O módulo ***Calib3d*** possuí algoritmos necessários à calibração de câmeras e funções *stereo*.
- O módulo ***ML*** trata-se de uma biblioteca de aprendizagem automática, *Machine Learning*, com algoritmos focados no reconhecimento estatístico de padrões e em ferramentas de *clustering* para o agrupamento dos dados.
- O módulo ***FLANN***, sigla para *Fast Library for Approximate Nearest Neighbors*, possuí métodos que são utilizados por outros módulos para procurar vizinhos mais próximos em grandes conjuntos de dados. Estes algoritmos possuem como objetivo acelerar a procura por correspondências, sacrificando apenas uma pequena parte da precisão [29].

Além destes módulos mais conhecidos e utilizados, módulos mais recentes foram e continuam sendo introduzidos com o lançamento de novas versões, dos quais se destacam os seguintes [24]:

- O módulo ***Shape*** que desvincula e combina formas.

- O módulo ***SuperRes*** que consiste na super resolução de imagens, isto é, no aprimoramento de resolução das imagens. Este aprimoramento consiste na combinação de uma sequência de *frames* de baixa resolução de forma a produzir uma imagem de resolução mais elevada [30].
- O módulo ***VideoStab*** que consiste na estabilização de vídeo.

À parte destes módulos mais conhecidos, mais utilizados ou recentemente introduzidos, existe ainda um repositório com a nomenclatura *"opencv_contribute"*, onde podem ser encontrados os módulos de contribuição. Estes são módulos modificados, desenvolvidos e mantidos pela comunidade da biblioteca e visam fornecer novas e diferentes funcionalidades. Módulos estes que devem ser totalmente compreendidos antes da sua utilização, uma tarefa que pode ser dificultada por nem sempre existir uma documentação completa [18, 23, 24].

2.2.3 Funcionalidades

A utilização da biblioteca ***OpenCV*** permite realizar praticamente qualquer tarefa de visão computacional, através da utilização de diferentes algoritmos e módulos em conjunto disponibilizados na mesma. Algumas destas tarefas encontram-se descritas abaixo, mencionando os módulos que dão o principal suporte a essas tarefas [1].

Antes de mais, para qualquer algoritmo de visão computacional, existem operações de processamento de imagens que serão utilizadas e reutilizadas inúmeras vezes. A maioria destas operações básicas encontram-se no módulo ***imgproc*** que, como já referido anteriormente, permitem a execução de tarefas envolvendo filtragem de imagem, transformações geométricas, conversões de cor, desenho em imagens, histogramas, análise de formas, identificação de características, entre outras [1].

Outra funcionalidade extremamente necessária em qualquer tarefa que envolva a visão computacional trata-se da criação de janelas que permitam a visualização do resultado das operações realizadas na imagem. O módulo que permite esta funcionalidade é o ***highgui***, que lida com todas as operações da interface de utilizador. Outras funcionalidades presentes neste módulo permitem detetar os eventos do cursor, extraordinariamente útil para o desenvolvimento de aplicações interativas, permitindo até desenhar ou selecionar regiões diretamente na imagem apresentada numa janela [1].

Passando de funcionalidades envolvendo imagens para funcionalidades envolvendo vídeos, existe um módulo chamado ***video***. Este módulo permite analisar movimentos entre *frames* sucessivos num vídeo, acompanhar diferentes objetos, criar modelos para vídeo vigilância, entre outras tarefas. Um outro módulo relacionado a videos, denominado por ***videostab***, trabalha com a estabilização de vídeo. Isto é importante pois muitos dos vídeos são gravados segurando a câmera com as mãos, existindo a necessidade de correção de algum movimento involuntário [1].

Uma das tarefas mais comuns quando se fala em visão computacional é a identificação de objetos, esta dedica-se, como o nome indica, ao posicionamento de um determinado objeto numa dada imagem. Este processo de identificação não tem em conta o tipo de objeto, isto é, no caso de uma cadeira não são feitas distinções entre a sua cor nem o seu encosto, sendo apenas identificada a cadeira e a sua posição na imagem. Esta tarefa precisa de realizar diversos cálculos para a identificação de objetos, sendo por isso uma tarefa de computação intensa. A biblioteca oferece os módulos ***objdetect*** e ***xobjdetect***, que fornecem ferramentas para esta identificação de objetos [1].

A noção de forma é crucial na visão computacional. A análise visual é feita recorrendo ao reconhecimento de diferentes formas numa imagem, sendo isto muito importante em múltiplos algoritmos. O módulo ***shape*** fornece algoritmos necessários à extração de diferentes formas, à transformação de formas em objetos, às medições de similaridades entre as formas, entre outros [1].

Por fim, não podia deixar de ser referida a tarefa de fotografia computacional, que se dedica à utilização de técnicas avançadas de processamento de imagem para o melhoramento de imagens capturadas por câmeras, isto é, são utilizados algoritmos para manipular os dados visuais tais como a transformação de uma imagem em *HDR*, numa foto panorâmica, na alteração da luminosidade, entre outros. Estas manipulações conseguem entregar fotografias com cores mais vivas, quando se aplica o *HDR*, o que não seria possível com as técnicas de imagem convencionais por serem necessárias várias capturas da mesma cena utilizando diferentes configurações de imagem, como a exposição, e ainda no fim unificá-las numa única imagem final. Os módulos que fornecem estas técnicas mais avançadas para ser utilizadas em fotografia são os módulos ***photo*** e ***xphoto***, existindo ainda o módulo ***stitching*** que fornece algoritmos para a criação de imagens panorâmicas [1].

Existem muitas outras funcionalidades e módulos, como já mencionado anteriormente, onde aqui se pretendia descrever de forma mais detalhada apenas algumas das mais populares e essenciais funcionalidades permitidas com a utilização da biblioteca ***OpenCV***.

2.3 Model-Driven Engineering

A abordagem *Model-Driven Engineering* (*MDE*), tem como princípio de que tudo é um modelo [31]. Os modelos fornecem abstrações de um sistema que permitem aos engenheiros raciocinar sobre esse sistema, ignorando detalhes irrelevantes, enquanto se concentram nos relevantes [2]. Esta abstração diz respeito à compactação e à redução ao essencial, constituindo um fator decisivo para o aumento da velocidade e da qualidade no desenvolvimento de *software* [24]. Estes modelos são utilizados de várias maneiras, tais como para prever as qualidades do sistema, raciocinar sobre propriedades específicas quando aspetos do sistema são alterados e comunicar as principais características do sistema aos seus *stakeholders* [2].

A modelação é uma atividade complexa, que pode implicar desenho gráfico, mas que pode ser substituído por notações textuais. Esta não se limita a retratar ideias genéricas, podendo conter semânticas implícitas que permitem a troca de informação, e ainda garante um conjunto de vantagens tais como a validação sintática, a verificação, a simulação, a transformação, a execução e a depuração do modelo. A execução do modelo pode acontecer por geração de código ou pela interpretação do modelo [31].

As linguagens de modelação são ferramentas que permitem especificar os modelos para os sistemas a serem desenvolvidos, isto é, permitem a definição de uma representação de um modelo conceptual [31]. Podem ser identificadas 2 grandes classes de linguagens:

- ***Domain-Specific Languages* (*DSLs*)**: são linguagens desenhadas especificamente para um certo domínio, contexto ou empresa e têm por objetivo facilitar a tarefa de descrever o conteúdo nesse domínio [31, 32]. Estas são as linguagens necessárias para a formalização dos modelos na abordagem *MDE* [2]. Existem milhares de *DSLs*, sendo as seguintes algumas das mais conhecidas e utilizadas: *BNF*, *SQL*, *HTML*, *MatLab*, *Excel*, *LaTeX*, entre outros [33, 34].
- ***General-Purpose Modeling Languages* (*GPMLs*, *GMLs*, ou *GPLs*)**: são linguagens que podem ser aplicadas a qualquer setor ou domínio para fins de modelação [31]. O exemplo

mais conhecido e utilizado de uma *GPML* é o *UML*, já nas *General-Purpose Programming Languages* inserem-se o *C/C++*, o *Java*, o *Python*, entre outros [35].

A definição de uma linguagem de modelação pode ser vista como um modelo em si, denomina-se a este procedimento de *metamodeling* [31]. A abstração de um modelo, que contém a abstração de fenómenos do mundo real, denomina-se de *metamodel*. Este, por se tratar de mais uma abstração, destaca as propriedades do modelo, definindo os constructos de uma linguagem de modelação, as suas relações, as suas restrições e as suas regras de modelação sem definir a sintaxe concreta da linguagem [31, 32].

Metamodelos e modelos têm uma relação classe-instância em que cada modelo é uma instância específica do metamodelo que o define. Para definir um metamodelo é necessária uma linguagem de *metamodeling*, que fornece os elementos e as regras para definir metamodelos. Esta linguagem, por sua vez, é descrita por um meta-metamodelo. Assim, este funciona como o último nível da hierarquia [32].

Esta abstração em cascata pode ser infinitamente recursiva, isto é, modelar um *metamodel* ou descrever todos os metamodelos significa definir um *meta-metamodel* [31, 32]. Em teoria é possível definir níveis infinitos de modelação, já na prática foi demonstrado que *meta-metamodels* podem ser definidos com base neles mesmos não fazendo sentido ir além deste nível de abstração, é descrito em seguida um exemplo destes níveis de modelação suportado pela Figura 2 [2, 31].

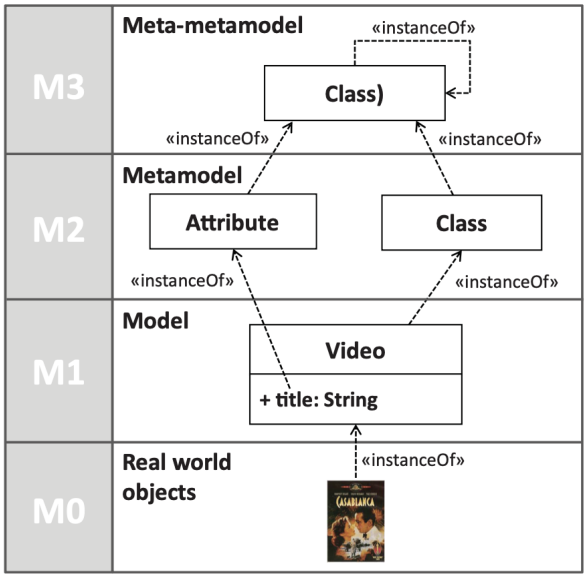

Figura 2. Exemplificação dos níveis de abstração [2].

Na Figura 2 é então demonstrado um exemplo de metamodelação, começando-se pelo objeto do mundo real, nível M0, que neste caso é um filme. Este é modelado numa representação, no nível M1 ,na qual o modelo descreve o conceito de vídeo com os seus atributos, que no exemplo é apenas considerado o título. Este modelo presente no nível M1 é metamodelado conforme representado no nível M2, descrevendo os conceitos utilizados para definir o modelo no nível anterior, nomeadamente Classe, Atributo e Instância. Por fim, no nível M3 está representado a meta-metamodelação que define novamente os conceitos definidos no nível anterior, desta vez o nível M2, em que todos

os conceitos colapsam num conceito único de classe neste exemplo, deixando claro que não há necessidade de continuar a se realizar novas metamodelações para além deste nível, como referido anteriormente, uma vez que o metamodelo de uma única classe seria novamente uma única classe [2].

Resumindo, podem ser definidos modelos, que consistem na abstração da realidade, metamodels, que são modelos que descrevem modelos, e meta-metamodels que consistem na descrição recursiva dos metamodels [31].

Quando é mencionado esta abordagem, *Model-Driven Engineering*, o primeiro desafio a encarar deve-se à imensa diversidade de acrónimos associados a esta abordagem que conseguem gerar alguma confusão. A Figura 3 demonstra, de forma visual, as relações entre os diferentes acrónimos e posteriormente é esclarecido os diferentes níveis de abstração associado a cada nível da abordagem.

A abordagem *Model-Driven Engineering* proporciona uma base sólida para abstrair e organizar as funcionalidades da biblioteca ***OpenCV***, permitindo que a plataforma *low-code* a implementar entregue uma experiência intuitiva e eficiente aos utilizadores, independentemente do seu nível técnico.

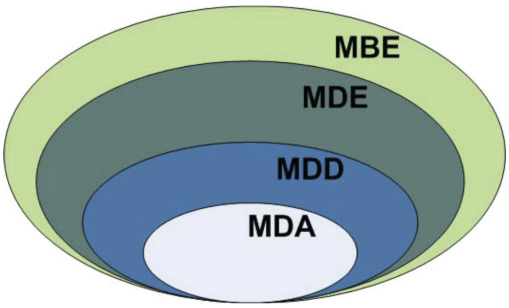

Figura 3. Representação gráfica das relações dos vários paradigmas Model-Driven Engineering [2].

O acrónimo *MDD*, *Model-Driven Developement*, é um paradigma que utiliza modelos como uma primeira abordagem no processo de desenvolvimento. Neste paradigma a implementação é tipicamente gerada a partir dos modelos de forma semiautomática [2, 31].

O *MDA*, *Model-Driven Architecture*, é uma visão particular do paradigma anterior, o *MDD*, proposta pela *Object Management Group* (*OMG*) dependendo assim da utilização dos padrões da mesma. O *MDA* pode ser considerado um subconjunto do *MDD*, no qual as modelações e as linguagens de modelação estão padronizadas pela *OMG* [2, 31].

O termo *Model-Driven Engineering*, *MDE*, é considerado um superconjunto do *MDD* pois vai além do puro desenvolvimento englobando tarefas de um processo completo de engenharia de *software* [2].

Por fim o *Model-Based Engineering*, *MBE*, é considerada uma versão mais simples do *MDE*, sendo um superconjunto do mesmo. É um processo em que os modelos de *software* desempenham um papel importante, embora não sejam necessariamente os artefactos principais do desenvolvimento, isto é, não conduzem (*Driven*) o processo como no *MDE*, sendo utilizados mais como planos ou rascunhos do sistema [2].

De notar ainda que existem diversas variantes da abordagem *Model-Driven Engineering*, *MD**, em que todas as abordagens *MD*E* podem ser consideradas como subconjuntos do *MDE* [2].

Todos os processos *Model-Driven Engineering* são *Model-Based* mas não ao contrário, isto é, os processos *Model-Driven Engineering* têm por fundamento um modelo, mas nem todos os modelos provém do processo *Model-Driven Engineering* [2].

2.4 Low-Code

O conceito *"Low-Code"* foi introduzido em 2014 pela *Forrester Research*, afirmando que as empresas preferem utilizar plataformas *low-code*, para uma entrega rápida e contínua [36]. Este conceito trata-se de um paradigma que permite o desenvolvimento de *software* com a mínima escrita de código, utilizando programação visual com interface gráfica e *design* orientado por modelos, isto é, *Model-Driven Engineering* [3].

De acordo com a mesma fonte, é estimado que o mercado de plataformas *low-code* deverá ser de 21 mil milhões de dólares até ao final de 2022, sendo esperado que até 2024 cerca de 65% das grandes empresas utilizem as plataformas *low-code* [3].

O número de plataformas de desenvolvimento *low-code* está a crescer, uma vez que têm sido muito solicitadas pelos *Citizen Developers*. Estes são especialistas de domínio que, tipicamente, não possuem conhecimentos de programação, mas que contribuem para os processos de desenvolvimento e de testes de *software*. Por serem os especialistas das funcionalidades do sistema, a definição dos requisitos é da sua responsabilidade. Com a utilização das plataformas *low-code*, este torna-se um utilizador final que consegue gerar aplicações empresariais para a utilização de outros e por si mesmo, tudo isto com um mínimo de esforço na escrita de código [3, 37–41]. Com a contribuição destes especialistas torna-se possível manter os procedimentos de desenvolvimento de *software* sem sacrificar a produtividade dos programadores profissionais, lidando assim com a escassez destes profissionais altamente qualificados [37, 38].

Atualmente existem mais de 200 plataformas, fornecidas maioritariamente por grandes empresas como a Google e a Sales-Force, o que acaba por tornar a escolha da plataforma a utilizar mais difícil, pois a escolha errada pode levar a um desperdício de tempo e de recursos. Algumas das plataformas mais populares são a Google App Maker, a Microsoft Power Apps, a SalesForce Lightning, a OutSystems, entre outras [37].

Um dos líderes no campo das plataformas *low-code* é a OutSystems. Esta permite a rápida construção e validação de aplicações *web* e móveis, permitindo que os seus utilizadores consigam modelar diferentes perspetivas tais como interfaces, modelos de dados e lógicas de negócio personalizadas. Por terem como fundamento o utilizador para as suas aplicações, isto é, devido às suas soluções *low-code* serem focadas nas necessidades dos utilizadores, tornaram-se numa das plataformas de topo. A OutSystems é destacada nesta secção por ter sido fundada em Portugal em 2001 e ser atualmente uma das líderes no mundo das plataformas *low-code* [42, 43].

As plataformas *low-code* são inspiradas na abordagem *Model-Driven Engineering*, onde o desenvolvimento é orientado pelas representações abstratas do conhecimento em vez de se focar nos algoritmos. Estas plataformas têm como propósito a abstração da complexidade existente nas metodologias de desenvolvimento tradicionais, reduzindo a complexidade associada ao desenvolvimento, aos testes e à manutenção [37]. Além disso, essas plataformas permitem que programadores de diferentes domínios e com diferentes capacidades técnicas desenvolvam aplicações completas e prontas para produção. Tais aplicações são desenvolvidas através dos princípios da abordagem

Model-Driven Engineering, já introduzida, as plataformas *low-code* utilizam a modelação e a metamodelação como fundamentos para alcançar essa abstração [38].

Uma plataforma *low-code* trata-se de um sistema que visa migrar o estilo de desenvolvimento de aplicações, que envolve a escrita manual de código através de uma linguagem de programação, para um estilo de interação com interfaces gráficas que faz uso de componentes previamente desenvolvidos. Estas interfaces gráficas são construídas através de diagramas visuais, de um elevado nível de abstração, de linguagens declarativas e em determinados casos programação manual [37].

As plataformas de *low-code* são normalmente descritas pela baixa quantidade de escrita de código, ou até mesmo a ausência de código, para a criação de *software*. Outros termos são associados a estas plataformas, tais como [36]:

- *drag-and-drop*;
- programação visual;
- interface gráfica do utilizador, que fornece uma interface gráfica para o *frontend*, para a lógica de *backend* e para a ligação às *APIs* externas;
- modelos pré-desenhados, isto é uma implementação de alto nível mais fácil de aprender e utilizar;
- *WYSIWYG* (*What You See Is What You Get*), para construir tudo de forma visual e possuindo diversos modelos bem construídos.

A principal motivação das plataformas *low-code* consiste na construção de aplicações, nas sugestões dos utilizadores e na incorporação rápida dessas mesmas sugestões. Desta forma, tanto a metodologia do desenvolvimento *Agile* como estas plataformas têm como fundamento a satisfação do utilizador final e a entrega contínua e incremental da aplicação. O círculo interior, na Figura 4, apresenta as principais e fundamentais fases do desenvolvimento de uma aplicação através de uma plataforma *low-code*, enquanto que o circulo exterior demonstra as fases do desenvolvimento *Agile* tradicional [3].

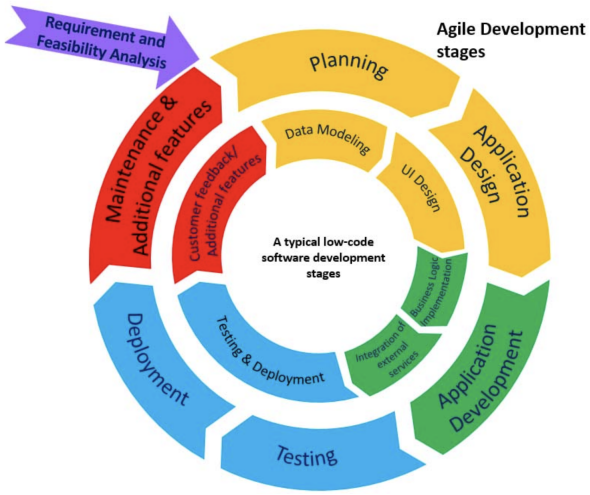

Figura 4. Fases de desenvolvimento tradicional vs low-code [3].

Estas plataformas, embora sejam mais acessíveis do que aprender uma linguagem de programação, continuam a possuir uma curva de aprendizagem ao serem utlizadas, especialmente nas que oferecem funcionalidades mais complexas. No entanto, devido à redução do tempo de desenvolvimento e à possibilidade de permitir que os *Citizen Developers*, referidos anteriormente, construam soluções, pode gerar uma maneira eficaz de reduzir certos custos. Algumas plataformas comerciais podem se tornar muito dispendiosas consoante a equipa de desenvolvimento cresce, devido aos seus preços elevados para fornecer um serviço completo. Para muitos utilizadores, estas plataformas são consideradas intuitivas, pois apenas recorrem às operações de *drag-and-drop*. No entanto, quando é necessário implementar alguma funcionalidade mais personalizada e flexível, torna-se necessário recorrer à escrita de código, originando complicações e frustrações [36].

De forma resumida, a utilização destas plataformas origina benefícios, tais como [3]:

- flexibilidade e agilidade;
- menor tempo de desenvolvimento;
- correção de *bugs* reduzida;
- menor esforço de implementação;
- manutenção mais fácil.

Sendo que o maior benefício destas plataformas a ter em grande consideração é a facilidade de criação de aplicações completas de forma rápida e com menor esforço, tornando o desenvolvimento mais ágil, por fornecerem unidades de implementação prontas a utilizar [36].

2.5 Conclusão

Esta dissertação parte da hipótese de que a integração da biblioteca ***OpenCV*** numa plataforma *low-code*, em combinação com uma abordagem *Model-Driven Engineering* capaz de permitir a geração de código numa linguagem de programação definida, tem o potencial de originar uma plataforma capaz de abstrair toda a complexidade associada à utilização da biblioteca ***OpenCV***. Esta abstração, por sua vez, deverá permitir o desenvolvimento de sistemas que envolvem a visão computacional de uma forma mais ágil, rápida, eficiente e acessível, reduzindo significativamente a barreia de utilização desta biblioteca por parte dos programadores e dos profissionais de diversas áreas. A hipótese assenta também na possibilidade de facilitar a manutenção e tornar as atualizações dos sistemas desenvolvidos mais direta. Além disto, pretende-se que a abordagem *Model-Driven Engineering* proporcione uma alternativa intuitiva e eficaz de modelar sistemas de visão computacional, reduzindo-se o conhecimento necessário em programação de baixo nível e da sintaxe da própria biblioteca.

A validação desta hipótese é o objetivo central nesta dissertação, explorando-se a implementação e a avaliação de uma plataforma *low-code* para a biblioteca ***OpenCV***. Devem ser realizados avaliação perante utilizadores com diferentes experiências com objetivo de se demonsrar como esta abordagem pode acelerar o desenvolvimento de sistemas de visão computacional, tornado-os mais acessíveis a um amplo conjunto de utilizadores.

3 Análise

Nesta secção serão descritos os requisitos que se consideram necessários ao desenvolvimento da plataforma *low-code* proposta anteriormente. Estes requisitos devem envolver a escolha de uma plataforma que forneça suporte às técnicas *low-code*, isto é, que forneça uma representação visual e que permita diferentes tipos de interação com esses elementos visuais. Para esta escolha é necessária a pesquisa, exploração, avaliação e comparação entre algumas das diversas plataformas existentes. Devem ser considerados ainda outros requisitos, desde a definição de uma framework para a implementação até à necessidade da definição de uma estrutura flexível para a facilidade nos desenvolvimentos futuros. Ainda nesta secção, são abordadas as estratégias adotadas para o desenvolvimento do protótipo bem como as decisões tomadas de forma a se cumprir os requisitos descritos, justificando-se cada uma das decisões desde a escolha da plataforma de suporte à framework de implementação até à escolha das funcionalidades a implementar.

3.1 Requisitos

Antes de qualquer decisão quanto à implementação do protótipo, alguns requisitos foram definidos para servirem como guia na tomada de decisão. Cada requisito deve ser objeto de pesquisa e análise, a fim de definir o caminho e a abordagem a serem seguidos, de modo a satisfazer as necessidades e a alcançar o objetivo.

3.1.1 Compreensão da Biblioteca OpenCV

O primeiro passo consistiu em compreender o funcionamento prático da biblioteca ***OpenCV***, identificada como a tarefa mais impactante para todo o desenvolvimento. Através da literatura, foi possível perceber a estrutura da biblioteca e que funcionalidades são fornecidas de uma forma meramente teórica, sendo essencial a perceção da sua aplicação e implementação na prática. Para tal foram implementados alguns dos exemplos e dos tutoriais, utilizando linguagens de programação como o C++ e o C#, permitindo uma compreensão mais aprofundada da aplicação da biblioteca.

3.1.2 Funcionalidades para Abstração

Através de toda esta análise e aprendizagem mais prática da biblioteca ***OpenCV***, pretende-se passar ao requisito seguinte. O próximo requisito tem por base a escolha das funcionalidades a implementar no protótipo de modo a elevar o seu nível de abstração. As funcionalidades escolhidas não precisam ser demasiado complexas mas também não devem ser demasiado simples, devem apresentar uma complexidade equilibrada. Pretende-se implementar algumas funcionalidades em que seja notável a elevação do nível de abstração, originando uma representação mais simples de uma funcionalidade, sem gerar uma correspondência direta entre o código necessário a escrever e essa representação.

3.1.3 Plataforma de suporte às técnicas Low-Code

Para esta elevação do nível de abstração através de uma plataforma *low-code*, existe a necessidade de utilização de uma plataforma capaz de fornecer suporte a representações visuais e a interações com estes elementos visuais. É necessária uma plataforma que permita a manipulação destes elementos visuais através de ações essenciais a numa plataforma *low-code*, como o *drag-and-drop*. Este requisito visa evitar o desenvolvimento manual de cada representação e interação. Esta escolha deve

ser alvo de uma pesquisa minuciosa por diferentes soluções, em que cada plataforma deve sofrer de uma análise e uma comparação prática, sendo estas diferentes soluções descritas em maior detalhe na subsecção seguinte.

3.1.4 Framework de suporte à aplicação web

O protótipo deverá se encontrar disponível em todos os sistemas operativos, ampliando o alcance a uma maior diversidade de utilizadores. Idealmente deve ser considerada a construção de uma aplicação *web*, disponível e acessível em qualquer navegador de *internet* moderno. Para a construção de uma aplicação *web* surge um novo requisito: a utilização de uma *framework* capaz de fornecer suporte à construção de aplicações *web*. Existem diversas *frameworks* disponíveis, tais como ***React***, ***Vue***, ***Angular***, entre outras. Ao avaliar esta escolha, é importante considerar a compatibilidade com o requisito anterior.

3.1.5 Boas Práticas de Desenvolvimento

Por fim, o protótipo deverá ser desenvolvido respeitando as boas práticas de programação, , cumprindo sempre que possível os princípios *SOLID* e fazendo uso de padrões de desenho. Isto com o objetivo de ser futuramente criado um repositório em que outros programadores podem contribuir com novas funcionalidades de forma fácil e intuitiva, obtendo-se assim uma estrutura flexível, estável e bem desenhada para a plataforma *low-code*.

3.2 Estratégias

Antes de explorar a elevação do nível de abstração na biblioteca ***OpenCV***, é importante compreender o conceito de **"bloco"** que será utilizado ao longo desta abordagem. Um **bloco** pode ser entendido como uma representação simplificada de uma ou mais funcionalidades do ***OpenCV***, projetado para encapsular a complexidade técnica e facilitar o uso da biblioteca por parte dos utilizadores.

A elevação do nível de abstração da biblioteca ***OpenCV*** entrega um maior valor ao conseguir transformar a implementação de funcionalidades complexas de forma simples e intuitiva. No entanto, ao implementar cada funcionalidade como um bloco independente, acabamos por criar uma correspondência direta, isto é, cada bloco representa apenas uma funcionalidade da biblioteca. Deste modo, o utilizador precisa de um conhecimento prévio da biblioteca ***OpenCV*** para identificar quais os blocos necessários para a implementação de uma determinada funcionalidade.

A verdadeira vantagem da abstração torna-se evidente ao conseguir-se encapsular funcionalidades complexas, que envolvem uma série de métodos e recursos do ***OpenCV***, num único bloco de alto nível. Neste contexto, a abstração não se limita apenas na associação entre blocos e funcionalidades, superando esta correspondência direta e permitindo uma simplificação substancial na interação com a biblioteca. Esta abordagem possibilita que os utilizadores desenvolvam funcionalidades complexas e completas com um esforço substancialmente reduzido, sobretudo para aqueles que não possuem domínio sobre a biblioteca ***OpenCV***.

Para atingir esse grau desejado de abstração, existem duas estratégias principais: a estratégia "*white box*"e a estratégia "*black box*". A estratégia "*white box*"envolve a geração de código a partir dos blocos de alto nível, que se traduzem em métodos que são intrinsecamente parte da biblioteca ***OpenCV***. Essa abordagem é adequada para os utilizadores que procuram um controlo preciso

sobre a funcionalidade e o desempenho das suas aplicações, pois permite interagir diretamente com os métodos da biblioteca.

Por outro lado, a estratégia "*black box*"requer o desenvolvimento de uma biblioteca open-source que encapsula as funcionalidades necessárias para tarefas mais complexas da biblioteca ***OpenCV***. O código gerado a partir destes blocos de alto nível limita-se a invocar os métodos desta nova biblioteca, ocultando-se os detalhes internos do ***OpenCV***. Embora essa abordagem simplifique significativamente o processo de desenvolvimento, tornando-o ideal para utilizadores que procuram um alto nível de abstração e que não pretendem interagir diretamente com os métodos do ***OpenCV***, adiciona-se uma nova dependência, criando-se a necessidade de instalar esta biblioteca auxiliar, que implementa as funcionalidades encapsuladas do ***OpenCV***.

Em resumo, a necessidade de um maior nível de abstração da biblioteca ***OpenCV*** é fundamental para tornar as funcionalidades complexas mais acessíveis e produtivas a uma mais ampla diversidade de utilizadores. As estratégias "*white box*"e "*black box*"oferecem abordagens complementares para alcançar este objetivo, cada uma com vantagens e desvantagens em termos de simplicidade e gestão de dependências.

3.3 Plataformas

Nesta secção serão descritas, de uma forma breve, algumas das plataformas encontradas que possuem o potencial pretendido e a capacidade para suportar a parte interativa do protótipo a implementar, isto é, a representação dos blocos, as suas relações, e a manipulação dos mesmos de forma gráfica. Apenas uma destas plataformas possui alguma literatura, sendo que a descrição realizada nas restantes deve-se à exploração da documentação, dos seus repositórios, dos seus exemplos e de alguma implementação prática.

3.3.1 Blockly

A plataforma ***Blockly*** tem como objetivo fundamental a construção de editores gráficos dedicados à programação visual. É um projeto *open-source* da Google que entrega uma enorme diversidade de blocos visuais já construídos. Além disso, permite a construção de novos tipos de blocos, desde os primitivos, ou folhas, até aos mais compostos, conhecidos como *containers* [44].

Esta plataforma foi inicialmente lançada em maio de 2012 continuando atualmente em desenvolvimento. O ***Blockly*** fornece um editor de blocos gráfico e uma ferramenta para a geração de código em linguagens de programação textuais, incluindo geradores para *JavaScript*, *Lua*, *PHP*, *Dart* e *Python*, podendo ser construídos geradores de código personalizados [45].

Fundamentalmente, esta biblioteca foi desenvolvida em *JavaScript*, podendo ser utilizada como parte de um *website* ou até mesmo numa *WebView*. Existem ainda versões nativas da biblioteca para *Android* e *iOS*, fornecendo um subconjunto de características para a construção de aplicações móveis de alto desempenho [45].

Como biblioteca, o ***Blockly*** não está pronto para ser utilizado pelos utilizadores finais. Não é totalmente uma linguagem, fornecendo apenas uma gramática e uma representação para a programação que os programadores podem utilizar nas suas aplicações. O código é representado por um bloco, que pode ser arrastado pelo ecrã, possuindo pontos de ligação utilizados para acoplar os blocos uns aos outros [45]. O ***Blockly*** não fornece um vocabulário completo de blocos nem um

ambiente de execução, sendo da responsabilidade do programador integrar uma forma de saída, construir um vocabulário próprio e decidir como é que o código gerado irá ser executado [45].

Como referido anteriormente, trata-se um projeto *open-source* Google, no qual ao explorar o seu repositório pode-se verificar que se trata de uma biblioteca em constante desenvolvimento, disponibilizando atualizações de forma recorrente. Pode-se ainda verificar pelas estatísticas do repositório, tais como a quantidade de *issues*, de *forks*, de *pull requests* e de visualizações, que a biblioteca possuí uma comunidade extremamente ativa, sendo de referir que a biblioteca ainda possui um fórum onde muitos utilizadores se ajudam mutuamente [46]. Por fim, no *website* dedicado aos programadores, está presente a sua documentação, sendo esta completa por apresentar os passos necessários às funcionalidades fornecidas e ainda por dar suporte recorrendo a alguns exemplos [47].

Foi então desenvolvido um exemplo muito simples utilizando esta plataforma, apresentado na Figura 5, com o objetivo de se adquirir uma perceção prática da curva de aprendizagem e da dificuldade de implementação dos blocos. Por possuir uma página dedicada à criação de blocos personalizados, notou-se que esta plataforma não é muito complicada de utilizar, requerendo um pouco mais de empenho no que se refere à geração de código.

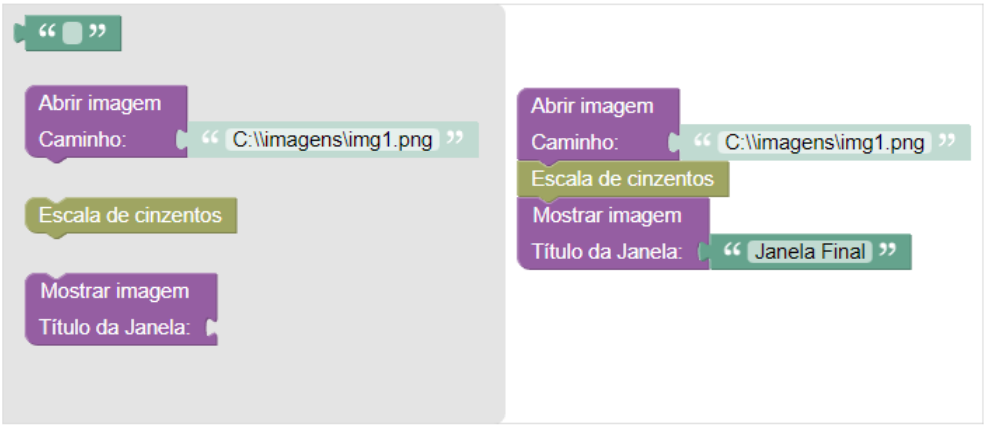

Figura 5. Implementação utilizando a plataforma *Blockly*.

O desenvolvimento deste exemplo começou pelo uso da página dedicada à criação de blocos, em que foi construído cada um dos blocos, sendo gerado o respetivo código em *JavaScript*, ou opcionalmente em *JSON*. Este código define apenas a representação do bloco, isto é, os espaços de entrada para o acoplamento de outros blocos, a cor e toda a parte visual. Também foi gerado na linguagem pretendida, neste caso o *JavaScript*, o código que apenas define a entrada dos valores

dos blocos acoplados, cujos espaços foram definidos no passo anterior. Neste código também foi declarada uma variável que deve conter o código que o bloco deve gerar tendo em consideração os valores recebidos pelo bloco.

3.3.2 GoJS

Esta é uma biblioteca para a *web* em *JavaScript* e em *TypeScript* que permite a construção rápida de diagramas interativos. Inclui diversos e diferentes exemplos de *layouts* personalizados que podem ser utilizados e adaptados conforme o necessário. O ***GoJS*** consegue ainda manter os modelos automaticamente sincronizados com o ecrã e vice-versa. Apresenta compatibilidade com *frameworks* modernas tais como ***Vue***, ***Angular*** e ***React***, permitindo uma simples e mais rápida integração do ***GoJS*** nestas *frameworks* [48].

Existem inúmeros exemplos de implementações disponíveis, que abrangem desde diagramas de fluxo, mapas mentais (*mind maps*), diagramas *UML*, entre outros. São ainda fornecidos tutoriais detalhados que descrevem cada etapa necessária à utilização da biblioteca, recorrendo ainda a vídeos para clarificar cada funcionalidade e cada passo. Já a sua documentação é igualmente abrangente, existindo explicações detalhadas, exemplos práticos e vídeos para praticamente todas as funcionalidades fornecidas por esta biblioteca [48].

A comunidade em torno desta plataforma não é tão ativa quanto na plataforma anterior, existindo um fórum que não apresenta uma grande atividade. Porém, o seu repositório, apesar de não apresentar *issues* nem *pull requests*, possuí ainda uma quantidade considerável de *forks* e visualizações. A sua versão gratuita possuí uma marca de água, sendo que para esta desaparecer é necessário adquirir uma licença, que apresenta um custo demasiado elevado para o pretendido neste protótipo. Esta biblioteca mantém-se atualizada, sendo que ainda este ano recebeu funcionalidades direcionadas ao *Windows Forms*, tendo a última atualização ocorrida, até à data de escrita, no final de julho de 2022 [48, 49].

Também foi desenvolvido um exemplo, desta vez utilizando esta plataforma *GoJS*, como se pode verificar na Figura 6. O objetivo foi o mesmo do exemplo da plataforma anterior: adquirir uma perceção mais prática sobre o custo de aprendizagem e implementação. Porém, neste exemplo, seguiu-se uma abordagem diferente, seguindo uma abordagem típica de *pipes & filters*. Esta plataforma embora não represente as portas, ou nós, de entrada e de saída de cada bloco, como era esperado na abordagem *pipes & filters*, segue a sua estratégia, pois o resultado do primeiro filtro é propagado ao seguinte.

Esta plataforma possui um recurso que se considerou interessante, pois permite agrupar elementos já interligados entre si, ou seja, torna-se possível definir uma tarefa como um bloco, em que este bloco contem definidos no seu interior diversos blocos mais simples que em conjunto implementam a tarefa principal, como poderá ser visto também na 6.

Ao contrário da plataforma anterior, ***Blockly***, não foi possível realizar a geração de código da biblioteca ***OpenCV***, o que mostra que se trata de uma tarefa mais complexa que no ***Blockly***. Conseguindo-se gerar uma representação em formato *JSON* que poderia ser posteriormente interpretada e convertida de forma a se gerar código, na linguagem C++, da biblioteca ***OpenCV***.

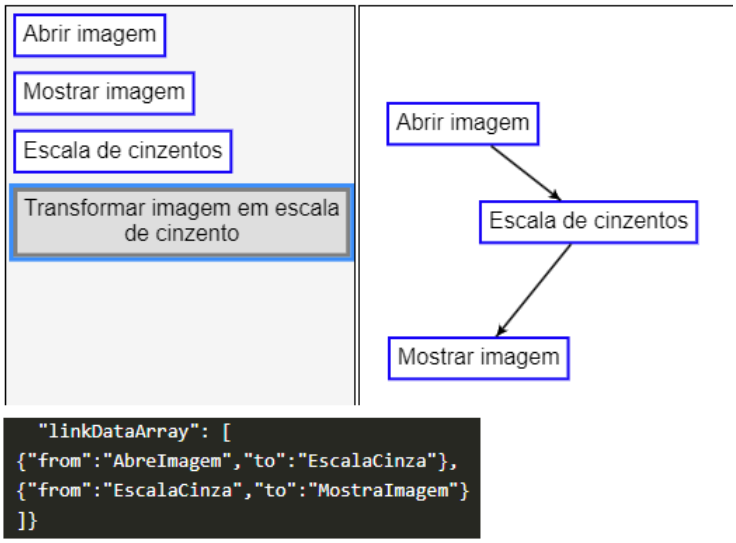

Figura 6. Implementação utilizando a plataforma *GoJS*.

3.3.3 InteractJS

A plataforma ***InteractJS*** tem uma abordagem diferentes das outras plataformas, permitindo um maior controlo sobre os eventos mantendo a sua *API* simples e flexível tanto quanto o possível. Devido a esta abordagem, torna-se necessário definir cada interação para cada elemento visual, não existindo nenhuma interação previamente definida [50].

A sua comunidade é moderadamente ativa, existindo ainda diversos *forks* no seu repositório e pouco mais que uma centena de visualizações. Ao nível de atualizações, é possível verificar ser existente, tendo a última atualização sido realizada no início de julho 2022, tendo em conta a data da escrita deste documento [51].

A sua utilização torna-se demasiado complexa para uma plataforma de suporte, pois como referido anteriormente, é necessário especificar para cada elemento cada um dos seus comportamentos, tais como as funcionalidades de arrastar um bloco para uma determinada zona, movimentar um bloco, os toques aplicados ao elemento, entre outros.

3.3.4 Draw2D

Draw2D é uma plataforma *web* que proporciona interações com gráficos e diagramas. Permite a criação de diagramas utilizando apenas o navegador de internet e a sua *API* [52]. A sua documentação é extensa e repleta de exemplos, tornando-se em alguns casos confuso devido à organização da mesma. Possuí ainda representações gráficas interessantes tais como portas ou nós de entrada e de saída, típicas da abordagem *pipes & filters* [52]. As suas atualizações não são tão frequentes como as anteriores, sendo que a última atualização até à data foi realizada no início de abril 2022. Apesar do seu repositório possuir alguns *forks*, não possuí grandes visualizações, tendo assim uma comunidade não muito ativa.

Não são descritos mais detalhes desta plataforma, pois a mesma é mais direcionada à renderização de diagramas em *SVG*, não respeitando o que era o objetivo definido [52, 53].

3.3.5 Diagram-Maker

Diagram-Maker trata-se de uma plataforma, desenvolvida também em *TypeScript*, sendo que pode ser utilizada juntamente com outras *frameworks* como ***React*** e ***Vue***, permitindo criar e interagir com elementos visuais. É totalmente personalizável, tanto em termos de aparência como de comportamento, permitindo aos consumidores adaptar a biblioteca a uma variedade de casos de utilização, apresentando por padrão portas de entrada e de saída, como na abordagem *pipes & filters* [54].

Fornece uma documentação bem organizada e exemplificada, tendo um aspeto muito importante devido a apresentar a sua arquitetura. Tal como a biblioteca anterior, a sua comunidade não é muito ativa apresentando pouco mais que uma centena de *forks* e algumas dezenas de visualizações. As suas atualizações também não são recorrentes, pois a última atualização remete a meados de agosto, sendo que a atualização anterior remete a meados de maio 2022 [54].

3.3.6 Vue Blocks

O ***VueBlocks*** apresenta uma representação muito típica da abordagem *pipes & filters*, como se pode verificar na Figura 7, permitindo a interligação de um bloco a vários blocos. Este repositório apresentava aquilo que se tinha em mente para a implementação do protótipo, tendo por pontos negativos não fornecer documentação nem ter atualizações desde novembro de 2020 [4].

A ponto de se analisar novas alternativas, foi questionado ao autor deste repositório se este se encontrava descontinuado e se existem alternativas semelhantes ativas, ao qual se obteve a resposta que este está abandonado devido à falta de tempo do autor, sendo indicadas 3 novas plataformas a se ter em consideração [55]. Estas novas plataformas a se considerar serão as seguintes a serem mencionadas.

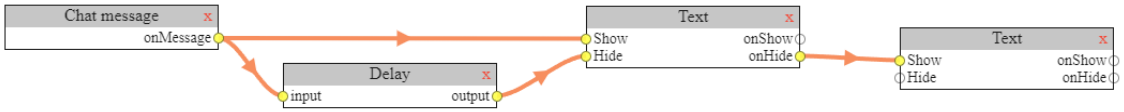

Figura 7. Representação dos blocos na plataforma *Vue Blocks* [4].

3.3.7 TotalJS

Uma das alternativas mencionadas pelo autor da plataforma anterior é o ***TotalJS***. Esta trata-se de uma plataforma *JavaScript* que fornece diversas bibliotecas, componentes de interface e produtos completos escritos em *JavaScript*. Uma dessas bibliotecas denomina-se ***Flow***, e consiste numa interface acessível para programação visual, integrando, processando e transformando vários eventos e dados em tempo real [56].

É uma biblioteca muito completa, que fornece uma inúmera diversidade de serviços e aplicações, tendo como limitação a criatividade do utilizador. A sua documentação é praticamente inexistente estando ainda em desenvolvimento. Apresenta exemplos simplistas, complicando a aprendizagem desta biblioteca tão completa e complexa [56].

Esta, apesar de toda a sua potencialidade, apresenta um número de *forks* relativamente reduzido, pouco mais que meia centena, e uma quantidade modesta de visualizações, de aproximadamente duas dezenas. Ao nível de atualizações, à data da escrita, é de notar que estas são muito frequentes e que remontam ao fim do mês de outubro de 2022 [57].

3.3.8 BaklavaJS

Este repositório ***Baklavajs*** trata-se de um editor de gráficos para a *web*, fornecendo a habilidade de criar os próprios nós. A sua documentação é limitada, mas apresenta os conceitos essenciais para a utilização da plataforma. Não possuí uma comunidade muito ativa, com um número muito reduzido de *forks*. Quanto às ocorrências de atualizações, é de notar acontecem com alguma regularidade, fornecendo no mínimo uma atualização por mês, sendo que a última remete a meados do mês de outubro de 2022 [5].

Apesar disto experimentou-se esta plataforma, pois apresentava uma representação e uma interação gráfica que está muito próxima ao que era pretendido inicialmente. Ao nível do comportamento, notaram-se alguns aspetos não favoráveis, desde à atualização dos valores introduzidos nos respetivos campos à geração do código. Já a nível gráfico notou-se mais tarde que alguns elementos ficavam desfocados devido ao *zoom* realizado, sendo que foi pedido ajuda no repositório da plataforma. Foi reconhecido que se tratava de um problema devido à renderização, que estava ainda por resolver.

Posto isto, implementou-se uma possível solução para este problema. Esta solução foi comunicada ao responsável do repositório, sendo descritas todas as alterações realizadas a nível do código. A solução implementada foi aceite, sendo oficialmente implementada pelo autor do repositório na versão seguinte, v1.9.3, conforme ilustrado na Figura 8 [5, 58].

Figura 8. Lançamento da versão 1.9.3 que implementa a solução partilhada.

Como se pode verificar pelas Figuras 9 e 10, a representação dos elementos é muito interessante, pois possui diversas alternativas e formas para a introdução de valores, como caixas de seleção,

caixas de verificação e de seleção de cor. A ligação entre os blocos também é intuitiva, conseguindo-se facilmente identificar onde se encontram e o intuitivamente perceber o seu funcionamento.

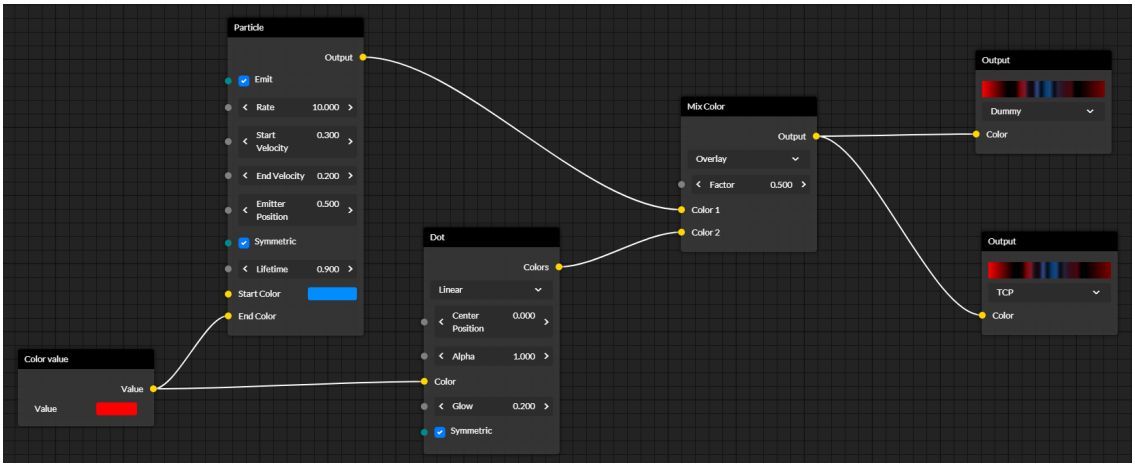

Figura 9. Representação dos blocos na plataforma *BaklavaJS* [5].

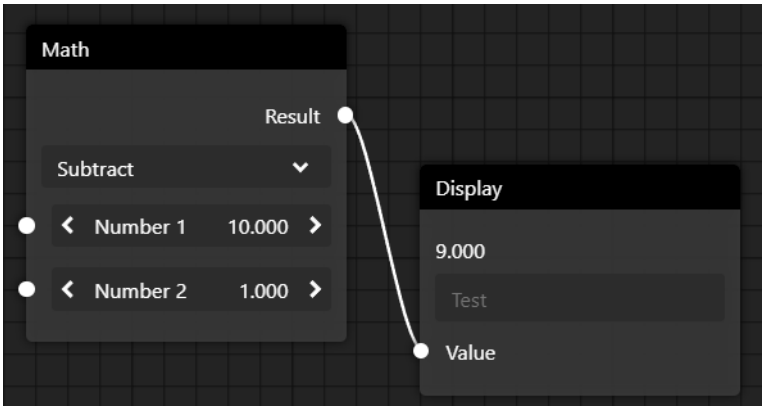

Figura 10. Representação mais detalhada dos blocos na plataforma *BaklavaJS* [5].

3.3.9 LiteGraph

Esta é uma plataforma em *JavaScript*, existindo suporte para *TypeScript*, que permite a criação de gráficos no *browser*. Os seus elementos são desenhados num *Canvas2D* e possuem diversas funcionalidades tais como, o *drag-and-drop*, a pesquisa pelos diversos nós, menus, caixas de seleção, sub grafos, entre outras. Estes nós, após inseridos no *canvas*, podem ser executados uma única vez, ou executados indefinidamente, permitindo que alterações geradas sejam visualizadas em tempo real [6].

A plataforma oferece uma imensa diversidade de nós que podem ser úteis para diversos cenários ou até mesmo para o desenvolvimento de nós personalizados. Estes nós possuem uma representação semelhante ao esperado inicialmente, sendo interessante por permitir tanto a representação na forma horizontal como na forma vertical, como se pode ver na Figura 11. Entre os nós oferecidos, são de destacar os nós de inserção e visualização de texto, de números, de imagens, e até sons [6].

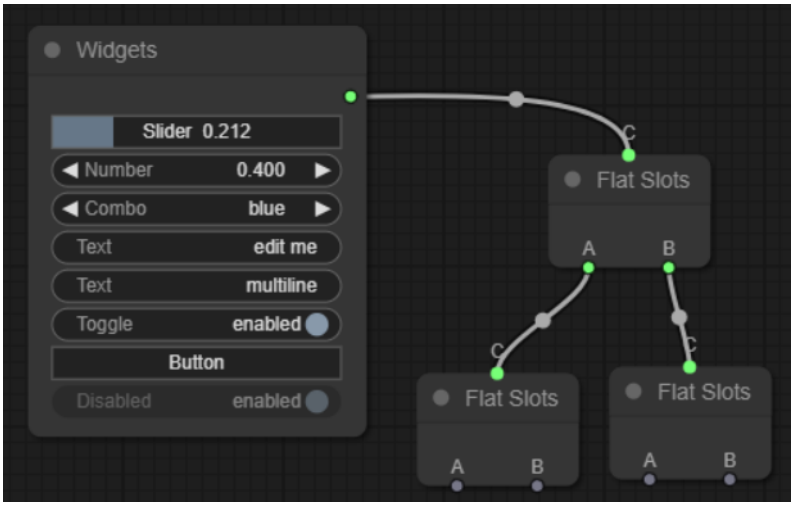

Figura 11. Representação dos blocos na plataforma *LiteGraph* [6].

Possuí uma boa documentação, descrevendo as principais classes e os métodos pertencentes às mesmas, e apresentando alguns exemplos práticos. Trata-se de uma biblioteca com uma comunidade relevante, existindo alguns *forks*, algumas *issues* e algumas visualizações. Ao nível de atualizações, são disponibilizadas algumas correções periodicamente e geralmente não afetam o comportamento das funcionalidades já implementadas, acabando por dar alguma estabilidade ao desenvolvimento utilizando esta plataforma tão completa [6].

3.4 Decisões

Definidos os requisitos, começou-se pela exploração e aprendizagem prática da biblioteca ***OpenCV***, implementando exemplos e tutoriais em C++ e em C#. Desta exploração descobriu-se a existência de uma versão da biblioteca em *JavaScript*, já mencionada na secção anterior. Esta versão em *JavaScript* chamou à atenção devido à possibilidade de se utilizar as funcionalidades da biblioteca no *browser*, o que acabaria por permitir a pré-visualização das manipulações realizadas a uma imagem ou vídeo antes mesmo da geração de código. Aprofundando e procurando por mais exemplos, encontrou-se uma derivação comunitária desta versão *JavaScript* da biblioteca. Esta versão encontra-se em constante desenvolvimento, pois trata-se de uma reestruturação da biblioteca ***OpenCV*** para *TypeScript*. Decidiu-se que seria interessante implementar-se as funcionalidades do protótipo recorrendo a esta versão da biblioteca, visto que esta já disponibiliza o *módulo* core na sua totalidade e praticamente todo o módulo de processamento de imagem.

Entre as duas estratégias apresentadas anteriormente, decidiu-se que a estratégia "*white box*"é a mais indicada. Esta decisão baseia-se na sua simplicidade e abordagem direta, evitando-se assim a necessidade de desenvolver uma biblioteca adicional.

A estratégia "*white box*"foi escolhida por permitir a geração de código recorrendo aos métodos nativos da biblioteca ***OpenCV***. Esta abordagem mantém assim uma ligação direta com a biblioteca, o que é vantajoso para os utilizadores que desejam um controlo preciso e explícito sobre as funcionalidade a implementar. Além disso, a escolha por esta estratégia elimina a necessidade de criar e manter uma nova biblioteca, simplificando assim o processo de desenvolvimento.

3.4.1 Funcionalidades

Ao se considerar o requisito seguinte, que consiste na escolha das funcionalidades a implementar, começou-se por definir que é necessário implementar as funcionalidades mais básicas, que envolvem a escolha e carregamento de uma imagem e a pré-visualização do resultado das manipulações. No fundo, trata-se no desenvolvimento da funcionalidade na qual todo o processo de manipulação se inicia, e na funcionalidade onde todo o processo de manipulação termina. Este bloco onde o processo se inicia deve ser representado por um bloco sem portas ou nós de entrada e com apenas uma porta ou nó de saída [0:1]. Já outro bloco, onde todo o processo de manipulação termina, deve ser representado com apenas uma porta de entrada e nenhuma porta de saída [1:0].

Outras funcionalidades devem ser consideradas de forma a que se consiga implementar pelo menos um tipo de bloco diferente, isto é, um bloco com uma porta de entrada e uma porta de saída [1:1], um bloco com várias entradas e uma única saída [n:1], um bloco com uma única entrada e diversas saídas [1:n] e por fim um bloco com várias entradas e várias saídas [n:m], onde as entradas e as saídas podem ser de diferente dimensão.

A partir dos requisitos acima mencionados e de toda a exploração, aprendizagem e implementação de alguns exemplos práticos utilizando a biblioteca ***OpenCV***, definiram-se as seguintes funcionalidades e respetivos blocos a implementar.

Começou-se pelas funcionalidades consideradas mais básicas, são as funcionalidades fundamentais ao funcionamento e que permitem a leitura das imagens que o utilizador pretende manipular e aplicar filtros utilizando a biblioteca, e a escrita das imagens resultantes de toda a manipulação realizada, permitindo a sua apresentação ao utilizador. Surgiu assim a motivação de serem implementados os blocos *"Capture Image"*, *"Show Image"* e ainda o bloco *"Camera Capture"*.

Este primeiro bloco, ***"Capture Image"***, consiste apenas na transformação de uma determinada imagem, selecionada pelo utilizador, numa matriz típica da biblioteca ***OpenCV*** [1,23,24,59], sendo esta matriz partilhada pelos restantes blocos desejados através da sua porta de saída. Trata-se assim de um bloco que não possui portas de entrada, pois é neste que se inicia todo o fluxo de manipulação, e apenas possui uma porta de saída, implementando-se assim um bloco do tipo [0 : 1]. De realçar que apesar deste bloco possuir apenas uma porta de saída, esta pode ser interligada a múltiplos blocos em simultâneo permitindo à mesma imagem ser utilizada em diferentes fluxos de manipulação, através de cópias.

O bloco ***"Show Image"***, tal como o nome indica, tem como responsabilidade a representação gráfica da matriz recebida pelo bloco anterior, permitindo a apresentação da imagem resultante das manipulações aplicadas à imagem inicial ao utilizador [1, 23, 24, 59]. Consiste então num bloco sem portas de saída, pois é neste bloco que um fluxo de manipulação termina, e possuí uma porta de entrada por onde é recebida a matriz da biblioteca vinda de um outro bloco, implementando-se um bloco do tipo [1 : 0]. De notar que, ao contrário do que foi realçado sobre a porta de saída no bloco anterior, cada porta de entrada só pode se encontrar associada a um e apenas um bloco. Neste caso, para se representar graficamente a mesma imagem diversas vezes, é necessário a criação de diversas instâncias deste bloco.

Por fim, o bloco ***"Camera Capture"*** possui uma responsabilidade semelhante ao primeiro bloco, *"Capture Image"*, cuja única diferença deve-se ao facto deste bloco não utilizar uma imagem determinada pelo utilizador, mas utilizar uma câmera disponível no dispositivo selecionando *frames* do vídeo recebido por essa mesma câmera [1, 23, 24, 59], e realizando o comportamento do primeiro

bloco. Cada *frame* é transformado numa matriz da biblioteca ***OpenCV*** e esta é propagada aos blocos associados à porta de saída, tratando-se novamente de um bloco sem portas de entrada e com apenas uma porta de saída, implementando-se um bloco do tipo [0:1], que pode ser associado a diversos blocos.

Estes são considerados os blocos básicos e fundamentais ao funcionamento da biblioteca ***OpenCV***. As outras funcionalidades a implementar consistem em manipulações ou filtros simples que se podem aplicar às matrizes geradas nos blocos básicos. Os próximos blocos, tais como o bloco *"Linear Blender"* e *"HSV Filter"*, inserem-se nesta categoria e são detalhados nos parágrafos seguinte.

O bloco **"HSV Filter"** tem como fundamento a transformação dos canais de cor típicos de uma imagem, BGR nas convenções biblioteca ***OpenCV***, para o modelo de cor *HSV* no qual, *H* para *Hue* representa a cor, *S* para *Saturation* representa a saturação, e *V* para Value representa o brilho da imagem [1, 23, 24, 59]. Esta transformação é muito utilizada nos problemas de visão computacional, pois o espaço de cor no modelo *HSV* produz um aumento no desempenho quando comparado ao modelo de cor tradicional [1, 23, 24, 59]. Este bloco é constituído essencialmente por uma porta de entrada, destinada à matriz representativa da imagem a se realizar a transformação, e por uma porta de saída onde será propagada a matriz resultante, surgindo assim o primeiro bloco com ambos os tipos de portas em simultâneo, porta de entrada e porta de saída, implementando-se um bloco do tipo [1:1]. De notar ainda que os valores atribuídos à cor, à saturação e ao brilho são definidos geralmente no bloco.

O bloco ***"Linear Blender"*** tem como objetivo a combinação e a sobreposição de duas imagens numa única imagem, podendo ser controlado através de uma quantidade percentual qual das imagens contribuí em maior quantidade para a imagem resultante [24, 59]. Trata-se conceitualmente de um bloco que possui duas portas de entrada, cada porta dedicada a uma matriz da biblioteca ***OpenCV*** originada a partir de uma imagem, e apenas uma porta de saída destinada à partilha da matriz resultante pelos restantes blocos, implementando-se assim um bloco do tipo [2:1].

O bloco ***"Split"*** tem como principal responsabilidade a separação dos canais de cor de uma determinada imagem, que se encontra tipicamente no formato *BGR*, em imagens contendo apenas um dos canais de cor, ou seja, uma imagem com apenas os tons em azul, outra com apenas os tons em verde e outra com os tons vermelhos [1, 23, 24, 59]. Isto é, recebendo uma matriz de uma imagem que possui 3 canais de cor pela única porta de entrada do bloco, três imagens com apenas 1 canal de cor deverão ser geradas e associadas à porta de saída correspondente. De forma a validar a definição de blocos com mais que uma porta de saída, ou seja blocos do tipo [1:n], torna-se ainda mais importante a implementação desta funcionalidade, definindo-se assim um bloco do tipo [1:3], possuindo então uma única porta de entrada e 3 diferentes portas de saída.

De forma a se reverter a funcionalidade do bloco anterior, *"Split"*, que separa uma imagem de 3 cores em 3 imagens de uma cor surgiu o bloco ***"Merge"***. Este bloco possui o comportamento inverso ao bloco *"Split"* pois recebe 3 imagens por cada uma das portas de entrada e une-as numa única imagem com os 3 canais de cor [1, 23, 24, 59], sendo partilhada a outros blocos pela única porta de saída do bloco. Este bloco torna-se importante para se voltar a reconstruir a imagem separada pelo bloco anterior, e simultaneamente validar a implementação de blocos com múltiplas portas de entrada, ou seja, blocos do tipo [n:1]. Define-se assim um bloco do tipo [3:1], constituído por 3 portas de entrada diferentes cujo resultado é partilhado aos restantes blocos interligados com a sua única porta de saída.

Considerando estas funcionalidades e, sendo importante para algumas destas funcionalidades a definição de valores numéricos ou de texto, decidiu-se ser importante implementar alguns blocos complementares que permitam definir estes valores e propagá-los por diversos blocos. Isto pode ser útil quando os mesmos valores são utilizados múltiplas vezes, sendo apenas necessário definir um valor apenas uma única vez num único bloco e associá-lo aos diversos blocos em que é pretendido utilizar esse valor, evitando-se assim a redefinição de valores e permitindo a reutilização de blocos.

Por fim é importante realçar que, concetualmente não foram descritos nos parágrafos anteriores a necessidade de implementação de blocos do tipo [n: m], isto é, blocos com múltiplas portas de entrada e múltiplas portas de saída em simultâneo, pois como são realizadas manipulações a imagens apenas é tipicamente necessário propagar a imagem resultante, sendo que esta pode ser propagada por diversas portas de entrada de outros blocos.

Como referido anteriormente, uma porta de entrada apenas pode ter associado a si um bloco, já uma porta de saída pode ser associada a vários blocos. Na Figura 12 está representado um diagrama que exemplifica estes blocos e as associações entre as portas de saída e as portas de entrada.

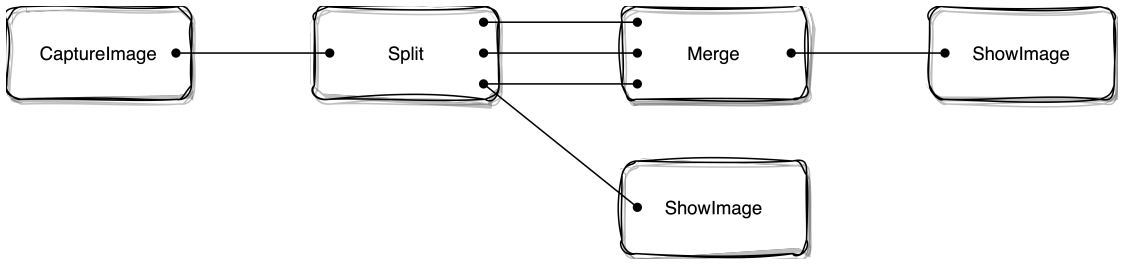

Figura 12. Exemplo de associações entre diferentes blocos.

É exemplificado, no diagrama da Figura 12, a ideia de representação de alguns dos blocos acima introduzidos. É possível verificar-se que as portas de entrada encontram-se na lateral esquerda de cada bloco e que as portas de saída estão posicionadas na lateral direita. Ainda é notável como é realizada a propagação de uma saída de um bloco para a porta de entrada de um ou mais blocos. Neste diagrama em específico pode-se verificar que a pré-visualização das manipulações e filtros aplicados pode ser utilizada em blocos intermédios, como é o caso do bloco *"Linear Blender"* e do bloco *"Split"*.

3.4.2 Plataforma

Considerando-se os requisitos definidos e as decisões já tomadas acima nesta secção, decidiu-se que a plataforma de suporte às técnicas *low-code* deve possuir, além das técnicas fundamentais, formas de se conseguir representar o resultado das manipulações e dos filtros aplicados às imagens e vídeos.

Considerando ainda a funcionalidade ou bloco ***"Camera Capture"***, descrita anteriormente, será importante que a plataforma consiga fornecer suporte à pré-visualização de vídeo. Seria ainda importante uma funcionalidade de execução para se conseguir capturar os *frames* do vídeo apenas após todo o fluxo de blocos estar definido, permitindo parar ou colocar em pausa todas as manipulações aplicadas ao vídeo.

Da análise e comparação anteriormente realizada às plataformas, decidiu-se que a plataforma mais completa, tendo em conta todos estes requisitos e decisões, é a plataforma ***LiteGraph***. Esta, como referido anteriormente, oferece uma vasta diversidade de exemplos e implementações, onde alguns envolvem o carregamento e a pré-visualização de imagem e de vídeo, sendo isto algo que facilitaria na pré-visualização das manipulações aplicadas, utilizando a versão *web* da biblioteca ***OpenCV***. A plataforma oferece também uma boa representação para os blocos e para as interligações entre os mesmos, e oferece ainda a possibilidade de se executar os blocos uma vez ou manter em execução durante um determinado intervalo de tempo.

3.4.3 Framework

Escolhida a plataforma, disponível em *JavaScript* e em *TypeScript*, pretende-se cumprir o requisito seguinte que consiste na escolha de uma *framework* que forneça suporte à criação de uma aplicação *web*. Esta *framework*, como referido nos requisitos, deve ter compatibilidade com a plataforma escolhida, ou seja o LiteGraph, tendo que ser compatível com *JavaScript* ou *TypeScript*. Preferencialmente pretendia-se utilizar *TypeScript* devido a ser fortemente tipado, devido às suas vantagens que não serão aqui descritas, e por se ter encontrado uma versão comunitária da biblioteca ***OpenCV*** em *TypeScript*, esta acaba por herdar as vantagens desta linguagem o que ajuda na implementação das funcionalidades da biblioteca principalmente por ser tipada e orientada por objetos, facilitando no requisito de implementar o protótipo recorrendo às boas práticas e aos padrões de desenho.

A *framework* escolhida foi o ***Angular***, esta é uma *framework* mantida pelo Google, o que acaba por gerar segurança e estabilidade às aplicações desenvolvidas. O ***Angular*** consegue ainda garantir, pelo que se verifica pelas versões anteriores, o funcionamento da aplicação *web* com o evoluir da *framework* [60].

Ao contrário do ***Vue***, que se trata de uma *framework* mais recente, e que por isso gera alguma insegurança e incerteza, sobretudo devido ao lançamento da sua terceira versão. Esta atualização introduziu mudanças significativas na arquitetura e no funcionamento da *framework*, tornando necessário realizar alterações no código existente e configurações adicionais para migrar aplicações das versões anteriores para a nova versão. De referir ainda que as versões mais antigas não recebem novas atualizações, sendo que o termo oficial da segunda versão do ***Vue*** está destinado para o fim de 2023 [60, 61]. Um outro aspeto considerado na escolha da *framework* ***Angular*** consiste na sua adoção ao *TypeScript*, que é essencialmente uma versão tipada do *JavaScript*. Esta versão tipada e orientada a objetos acaba por facilitar e acelerar o desenvolvimento devido à capacidade de serem verificadas, durante o desenvolvimento, as atribuições de tipos corretos às variáveis [62, 63].

Quantas mais funcionalidades são necessárias mais bibliotecas são instaladas, levando ao aumento da complexidade da *framework*, da sua utilização e da aplicação *web* desenvolvida [64, 65].

Um último aspeto a considerar na escolha desta *framework* consiste no facto de que o ***Angular*** é uma *framework* completa, pois fornece todas as ferramentas necessárias à criação de uma aplicação. Esta integra, de forma nativa, funcionalidades essenciais como a gestão de *routes*, a comunicação com servidores através de serviços como o *HttpClient*, e um sistema de injeção de dependências avançado. Estas funcionalidades reduzem a necessidade de dependências externas, permitindo que os programadores trabalhem num ambiente homogéneo e bem documentado. Quantas mais funcionalidades são necessárias mais bibliotecas são instaladas, levando ao aumento da complexidade da *framework*, da sua utilização e da aplicação *web* desenvolvida [64, 65].

Por fim, de forma a manter a estrutura do protótipo bem estruturada e flexível cumprindo e utilizando sempre que possível os princípios *SOLID* e os padrões de desenho, decidiu-se ser importante definir uma arquitetura para a plataforma, recorrendo à criação de um modelo de dados. Estes aspetos devem de ter em consideração a estrutura da plataforma escolhida, ***LiteGraph***, e as suas funcionalidades disponíveis. Tendo-se ainda em consideração as funcionalidades e a integração da biblioteca ***OpenCV*** na *web*. Todos os detalhes mais práticos destas decisões e da implementação serão descritos na secção seguinte referente ao desenvolvimento.

3.5 Conclusão

Após toda esta análise, exploração e implementação de exemplos utilizando a biblioteca ***OpenCV*** e realizada a comparação das diversas plataformas de suporte às técnicas *low-code* conseguiu-se definir os principais objetivos e as decisões fundamentais. Conclui-se esta secção com as decisões fundamentais desde as funcionalidades a implementar, as ferramentas e as plataformas de suporte a utilizar para a implementação do protótipo da plataforma *low-code*. Ficando assim nesta secção definido a utilização da versão comunitária da biblioteca ***OpenCV*** em *TypeScript* com vista a implementar as funcionalidades já definidas. Estas funcionalidades e esta versão da biblioteca ***OpenCV*** serão implementadas utilizando as funcionalidades adquiridas pela utilização da plataforma ***LiteGraph*** de suporte ao *low-code*. Isto tudo transformado numa aplicação *web* suportada pela *framework* ***Angular***.

4 Desenvolvimento

Instaladas todas as ferramentas e plataformas definidas na secção anterior, dá-se início à fase de desenvolvimento do protótipo proposto. Toda a fase deste desenvolvimento encontra-se descrita nesta secção, começando-se por introduzir a arquitetura da plataforma de suporte escolhida, passando-se à implementação do protótipo recorrendo-se a exemplos práticos e, finalizando-se esta secção com as limitações e dificuldades encontradas ao longo do desenvolvimento.

4.1 Arquitetura da Plataforma LiteGraph

Esta plataforma encontra-se dividida essencialmente em 4 níveis, que se encontram descritos nos seguintes parágrafos. Antes da descrição destes níveis é importante introduzir-se o conceito de *node* e de *slot*. Anteriormente foram mencionados os blocos que se pretendiam implementar, estes blocos são referidos pela plataforma como *nodes*. Já a noção de *slot* refere-se às portas de entrada e de saída que são utilizadas a interligação dos diversos blocos.

4.1.1 LGraphNode

O primeiro nível da arquitetura aqui descrito é o mais complexo, consiste na classe ***LGraphNode***. Esta classe tem como principal responsabilidade a definição dos nós e toda a sua funcionalidade. É constituída por um conjunto de elementos que permitem instanciar, controlar e atribuir comportamentos a cada um dos nós. O principal elemento presente nesta classe é o ***Node***, que representa tipicamente o bloco. Este possuí diversas propriedades que permitem definir as características visuais do bloco, tais como o nome, o tamanho, a cor, entre outras.

Um bloco também pode possuir diferentes comportamentos de acordo com a interação realizada com o mesmo, através de métodos como os seguintes, sendo de realçar que estão disponíveis outros métodos para além destes:

- ***onAdded()***: executado quando o bloco é instanciado.
- ***onRemoved()***: executado quando o bloco é removido.
- ***onExecute()***: executado de forma recorrente quando o ambiente entra em modo de execução.
- ***onConnectionsChange()***: executado sempre que as associações das suas portas de entrada ou de saída sofrem alterações.
- ***onDrawBackground()***: permite personalizar o aspeto gráfico do conteúdo do bloco.

Cada bloco, ou *node*, pode possuir uma ou mais portas de saída, ou de entrada. Introduz-se assim uma nova classe, que estende a classe principal, a classe ***Slot***. Estas portas de entrada e de saída são adicionadas ao bloco invocando os métodos ***addInput()*** e ***addOutput()*** respetivamente. A classe ***Slot*** possuí algumas propriedades, como o nome atribuído à porta. Já outras propriedades dependem se a porta é de entrada ou de saída. Uma porta de entrada possuí propriedades como o tipo de dados que deve receber e uma *source* ou origem, que contém uma referência ao *slot* do bloco que se encontra associado a esta porta. Por outro lado, uma porta de saída possuí também uma propriedade destinada ao tipo de dados que está a ser enviado, e uma lista denominada por *targets* ou origem, que contém cada uma das instâncias de ***Slot*** de todos os blocos a que esta porta se encontra associada. Para as portas de entrada obterem os dados enviados pelo bloco associado precisam invocar o método ***getInputData()***, já para as portas de saída definirem os

dados a enviar às portas de entrada dos blocos, associados a esta porta de saída, precisam invocar o método ***setOutputData()***.

Uma das propriedades dos blocos que pode ser definida consiste no seu tipo de valor, quer numéricos quer de texto. Para estes existe uma outra subclasse de ***Node***, denominada por ***Widgets***. Estes *widgets* são adicionados aos blocos invocando o método ***addWidget()***, sendo necessário indicar como parâmetro o tipo de *widget* que se pretende instanciar. Uma característica importante dos *widgets* centra-se na disponibilização de um aspeto mais apelativo e de uma forma de interação diferente e mais visual. Existem alguns tipos de *widgets* já fornecidos pela plataforma, sendo estes os seguintes tipos:

- ***Number***: permite alterar um valor numérico incrementando ou decrementando-o de forma constante, simulando a utilização de botões físicos.
- ***Text***: permite introduzir um valor textual
- ***Slider***: permite alterar também um valor numérico, mas recorrendo a um elemento gráfico deslizante.
- ***Combo***: permite selecionar um dos valores predefinidos disponíveis como lista.
- ***Toggle***: permite ativar ou desativar um valor, semelhante a um interrutor.
- ***Button***: permite adicionar um simples botão, cuja funcionalidade deve ser definida durante a instanciação do mesmo.

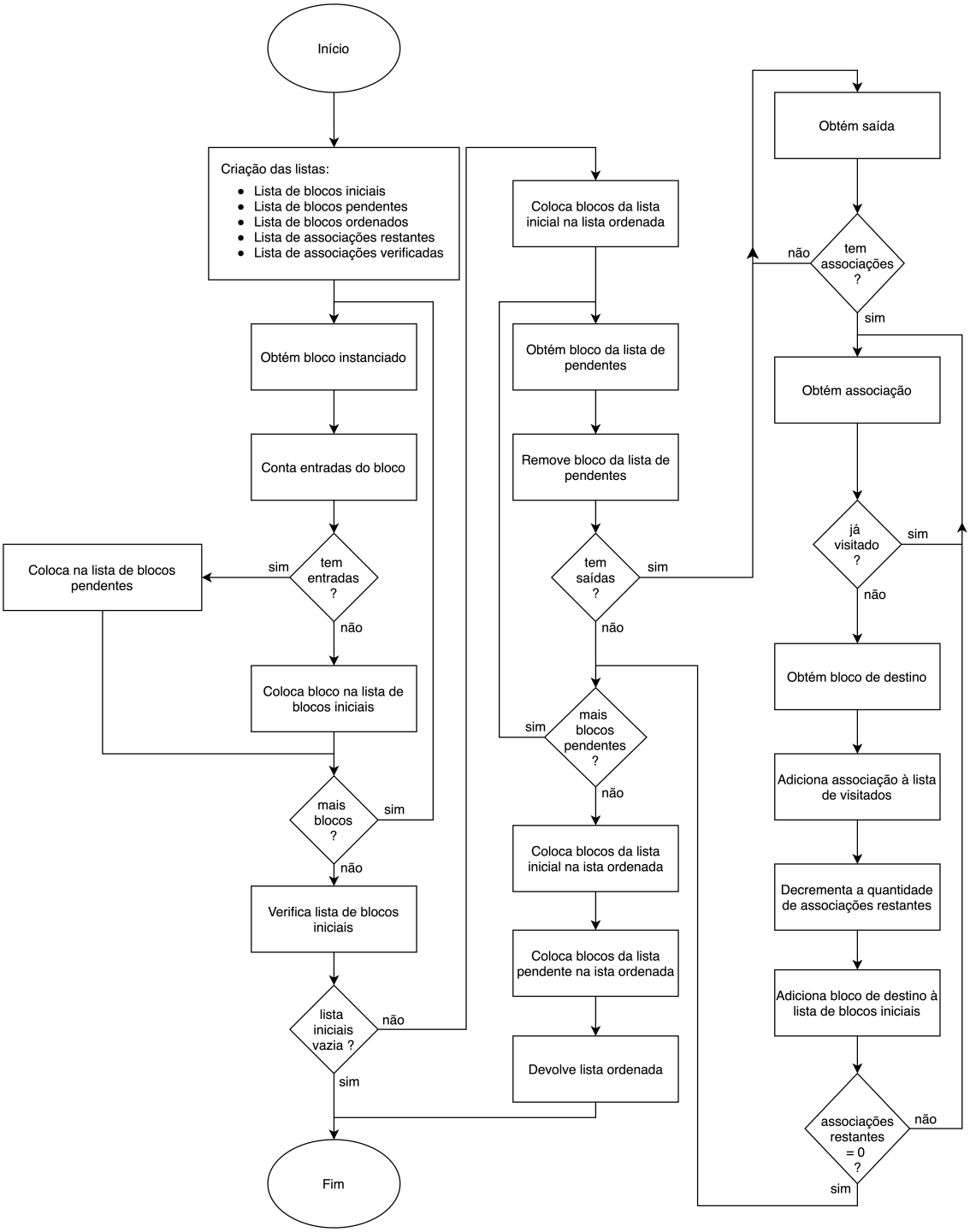

Figura 13. Fluxograma de ordenação de blocos pela plataforma *LiteGraph*.

4.1.2 LGraph

Um outro nível da arquitetura consiste na classe ***LGraph***. Esta classe contém todo o gráfico completo. Deve ser apenas instanciada uma única vez, em que são adicionados a esta todos os blocos instanciados. A partir desta classe com todos os elementos associados, torna-se possível a execução dos blocos de forma ordenada. Ao serem adicionados os blocos ao *canvas*, esta classe atribuí um identificador único a cada bloco, sendo que no momento da execução do gráfico é gerada uma lista contendo os blocos numa ordem específica.

Os blocos para execução são ordenados de acordo com a sua relevância, em que um bloco com apenas portas de saída possuem maior relevância que os blocos com apenas porta de entrada, sendo estes blocos com apenas portas de saída os blocos iniciais. A partir das associações destes blocos são obtidos os seguintes blocos a se executar até convergir para os blocos finais.

De uma forma mais detalhada, esta ordenação dos blocos é conseguida através da realização de uma busca em profundidade. Como o fluxograma da Figura 13 representa, o algoritmo começa por criar cinco listas, sendo as seguintes: lista de blocos iniciais, lista de blocos pendentes, lista de blocos ordenados, lista de associações restantes e lista de associações visitadas. A cada bloco é contabilizado a quantidade de entradas do mesmo, sendo que se não existirem entradas é considerado um bloco inicial, introduzido assim na lista de blocos iniciais, e, para o caso em que existam entradas, é introduzido na lista de pendentes.

Após estarem todos os blocos nestas listas, todos os blocos da lista inicial passam para a lista ordenada e inicia-se o tratamento aos blocos pendentes. Para cada bloco na lista de pendentes é verificada se possuí portas de saída, e em caso de possuir o bloco é removido da lista de pendentes e passam a ser verificadas cada uma das associações para cada uma destas saídas, marcando-se a verificação de cada associação utilizando a lista de associações visitadas. A cada associação visitada é obtido o bloco de destino, sendo este adicionado ao fim da lista de blocos iniciais e marcados como visitados na lista destinada a esse fim.

Finalizado este comportamento a todos os blocos da lista de pendentes, verificando-se as suas saídas e respetivas associações, colocam-se estes novos blocos da lista inicial na lista ordenada e por fim os restantes blocos da lista de pendentes, considerados os blocos finais, são colocados também na lista ordenada.

Todo este fluxo consegue assegurar que os blocos que dependem da saída de outros blocos sejam executados após estes o serem, garantindo assim a execução dos blocos na ordem correta, e minimizando-se a quantidade de vezes que cada bloco é executado.

Descritos os principais componentes, é necessário ainda descrever como são realizadas as instanciações visuais dos blocos e onde é permitida toda a funcionalidade e interação entre os elementos das classes anteriormente referidos.

4.1.3 LGraphCanvas

Um terceiro nível da arquitetura implementa a classe ***LGraphCanvas***, que é responsável pela renderização do *canvas* e de todos os seus componentes, começando pelos blocos e recursivamente pelos seus *slots* e *widgets*. Esta classe é ainda responsável por todas as diferentes formas de interação disponíveis e pelos eventos, isto incluí as funcionalidades de *drag-and-drop*, de *zoom* e de desenhar os blocos e os seus respetivos elementos.

4.1.4 LiteGraph

Este último nível possuí o mesmo nome que a plataforma, ***LiteGraph***, isto devido a se tratar da sua classe global. Nesta classe são registados os blocos, de forma a permitir a sua instanciação e utilização pelas restantes classes da plataforma mencionadas imediatamente acima. Sem a instanciação desta classe global toda a plataforma não funciona.

4.2 Implementação do Protótipo

Introduzida a arquitetura da plataforma a utilizar para dar suporte às técnicas *low-code* no protótipo proposto, são descritas nos seguintes parágrafos todo o processo de desenvolvimento e de implementação de cada um dos principais componentes, acompanhado por exemplos e ilustrações.

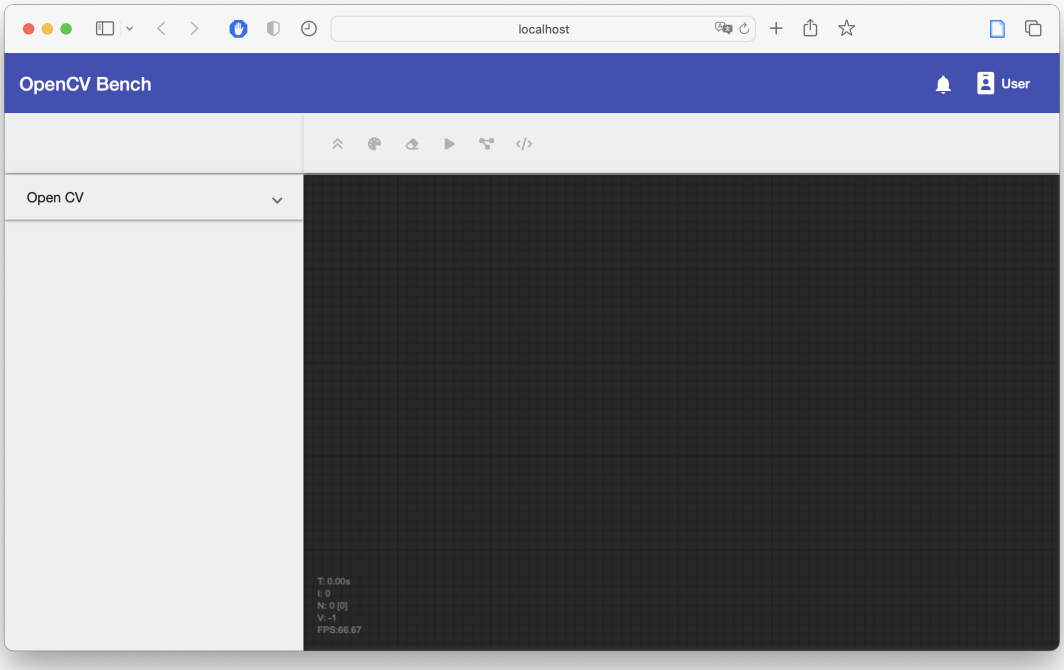

Figura 14. Editor completo.

4.2.1 Editor

Começa-se por descrever o desenvolvimento do elemento mais fundamental de todo o protótipo, o ***Editor***, Figura 14. Neste ***Editor*** definiu-se 3 elementos fundamentais em que o principal elemento consiste no ***Canvas***, Figura 15, uma área onde é realizada toda a atividade, é neste ***Canvas*** que o utilizador realiza todo o seu desenvolvimento sem recorrer à escrita de código.

Um segundo elemento consiste numa palete, Figura 16, que contém todos os blocos registados na classe ***LiteGraph***, introduzida na secção 4.1.4, permitindo a rápida seleção e posterior instanciação no ***Canvas***. A implementação destes blocos é descrita nas secções seguintes.

Por fim, o último elemento, Figura 17, trata-se de uma ***Toolbar*** ou barra de ferramentas responsável por disponibilizar ações a serem realizadas ao ***Canvas*** e aos blocos instanciados no

Figura 15. Elemento Canvas do Editor.

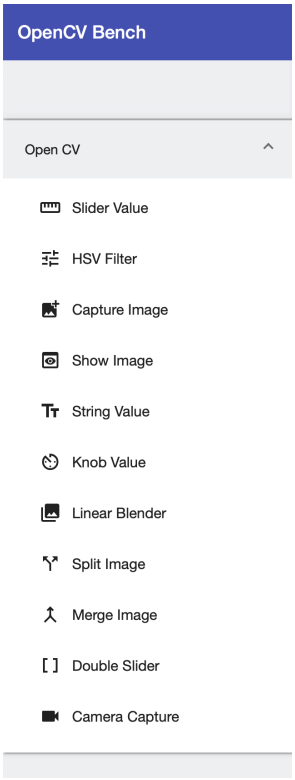

Figura 16. Palete de blocos do Editor.

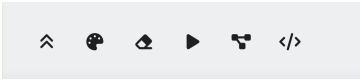

Figura 17. Barra de Ferramentas do editor.

mesmo. É através desta barra de ferramentas que é possível executar cada um dos blocos, gerando posteriormente o seu modelo e o código correspondente na linguagem de programação C++.

De realçar ainda que, é também neste componente ***Editor*** que se inicializa a plataforma ***LiteGraph***, redefinindo-se as propriedades necessárias e implementando-se um *override* às funcionalidades que não possuem o comportamento pretendido, evitando-se aplicar alterações à plataforma de suporte. As principais propriedades alteradas consistem na desativação das funcionalidades de interação, tais como o menu para a instanciação dos blocos, pois optou-se por desenvolver uma palete como referido acima, e a redefinição do aspeto de alguns *widgets* que não se enquadravam bem nos blocos. Como se pretendia funcionalidade acima de tudo, não foram alteradas propriedades relativas ao aspeto gráfico dos elementos da plataforma de suporte, sendo de realçar que estas podem ser facilmente alteradas neste componente.

4.2.2 Blocos

Esta secção foca-se no desenvolvimento dos blocos, nas suas propriedades e elementos. Cada bloco implementado corresponde a uma subclasse que herda os atributos e os métodos da classe ***LGraphNode***, introduzida na secção 4.1.1.

No início de cada bloco devem ser definidas as suas propriedades essenciais, isto é, deve-se definir o título, o tamanho, a descrição e o ícone do bloco. Já no construtor, devem ser instanciadas

as portas de entrada e de saída, recorrendo aos métodos ***addInput()*** e ***addOutput()***, disponíveis através da plataforma de suporte ***LiteGraph***. Na instanciação destas portas, devem ser introduzidas como parâmetros um nome para identificação da porta, o tipo de dados que deve receber ou enviar e, opcionalmente, uma *label* para se apresentar um nome diferente no bloco. Ainda no construtor, devem ser instanciados os *widgets*, já mencionados acima, atribuindo-se um nome, valores iniciais, valor máximo e valor mínimo, e a sua respetiva função. Caso o *widget* pretendido não esteja implementado nativamente na plataforma ***LiteGraph***, é possível implementar um *widget* personalizado no bloco, como é o caso do bloco *"Double Slider"*, descrito na secção 4.2.2.1.

Após a introdução do construtor dos blocos, um outro método presente em todos os blocos trata-se do método ***onExecute()***, herdado da classe ***LGraphNode*** da plataforma ***LiteGraph***. Neste método encontra-se definida toda a funcionalidade e comportamento do bloco quando todo o ***Canvas*** é colocado em execução, começando por processar os dados recebidos pelas portas de entrada, aplicando as funcionalidades implementadas, e atribuindo os dados com as devidas alterações às portas de saída, como pretendido.

Outro método importante é o ***onConnectionsChange()***. Este método permite definir diferentes formas para a atribuição de um valor a um bloco, por exemplo, um bloco que necessita de um valor numérico pode, com a devida implementação neste método, definir este valor num *widget* instanciado no próprio bloco ou receber esse valor de um outro bloco através de uma porta de entrada, dando-se sempre prioridade à porta de entrada e atualizando-se o valor do *widget* e do bloco de acordo com o valor presente na porta de entrada.

Os restantes métodos dedicam-se às interações com cada bloco, que não serão descritos nesta dissertação. Existindo ainda métodos que se dedicam à geração de código e à implementação da biblioteca ***OpenCV*** utilizando a sua versão em *TypeScript*, sendo estes descritos nas secções seguintes.

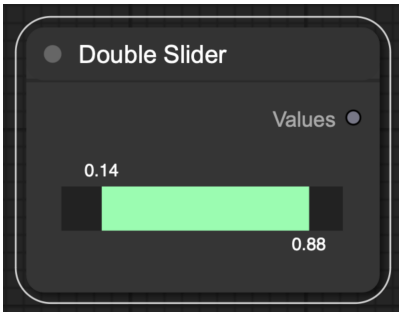

Figura 18. Bloco *Double Slider*.

4.2.2.1 Bloco Double Slider

Este bloco surgiu da necessidade de se obter um valor mínimo e um valor máximo, evitando-se a instanciação e a utilização de 2 *sliders*. Implementou-se então o bloco *"Double Slider"*, Figura 18, que permite selecionar um valor máximo e um valor mínimo simultaneamente, sendo estes valores enviados em forma de vetor.

No desenvolvimento desde bloco foi necessário definir todo o comportamento do cursor, desde o clique até ao movimento do cursor, estando limitado ao *slider* e não permitindo a troca dos

valores máximo e mínimo. Já o elemento gráfico correspondente ao *"Double Slider"* é definido no método ***onDrawBackground()***, Figura 19, verificando-se a criação de 2 marcadores dedicados a delimitar o valor mínimo e o valor máximo.

```
override onDrawBackground(ctx: any) {
    if (this.flags.collapsed) {
        return;
    }

    const margin = 20;
    const widgetHeight = 25;
    const h = LiteGraph.NODE_SLOT_HEIGHT + LiteGraph.NODE_TITLE_HEIGHT;
    const w = this.size[0] - margin;

    //background color
    ctx.fillStyle = this.doubleSlider.bgcolor;
    ctx.fillRect(margin, h, w - margin, widgetHeight);

    //range color
    const minMarker = margin + this.doubleSlider.minValue * (w - margin);
    const maxMarker = this.doubleSlider.maxValue * (w - margin) - this.doubleSlider.minValue * (w - margin);
    ctx.fillStyle = this.doubleSlider.color;
    ctx.fillRect(minMarker, h, maxMarker, widgetHeight);

    ctx.textAlign = "center";
    ctx.fillStyle = this.doubleSlider.text_color;
    //min value
    ctx.fillText(this.doubleSlider.minValue.toPrecision(2), minMarker, h * 0.9);
    //max value
    ctx.fillText(this.doubleSlider.maxValue.toPrecision(2), minMarker + maxMarker, h * 1.6);
}
```

Figura 19. Método *onDrawBackground()* definido no bloco *Double Slider*.

Este método, ***onDrawBackground()***, tem como principal responsabilidade a alteração e a atualização do aspeto de cada bloco, devendo atualizar o aspeto de acordo com os dados recebidos pelas portas de entrada, bem como pela manipulação de um elemento gráfico do bloco, como o exemplo apresentado em cima.

4.2.2.2 Bloco Linear Blender

Tendo em consideração toda esta descrição dos componentes que definem um bloco, decidiu-se exemplificar a definição de bloco. O bloco alvo deste exemplo é o *"Linear Blender"*, cuja responsabilidade consiste na sobreposição de 2 imagens numa única imagem. Este bloco servirá de exemplo, pois permite introduzir grande parte dos conceitos acima mencionados. Os restantes blocos podem ainda ser analisados recorrendo ao código que acompanha esta dissertação.

Para a implementação deste bloco, *"Linear Blender"*, começou-se por definir uma subclasse com este mesmo nome que herda as propriedades e os métodos da classe ***LGraphNode*** da plataforma de suporte ***LiteGraph***, como mencionado anteriormente.

No seu construtor definiram-se as propriedades específicas, Figura 20, no qual se define o título do bloco e são instanciadas as portas de entrada e de saída conforme foi declarado na secção na qual constam as decisões tomadas sobre os blocos a implementar. Instanciou-se assim uma porta de saída, que deve conter a matriz da imagem resultante, duas portas de entrada para receber as duas

imagens a serem sobrepostas e uma outra porta de entrada, que se encontra associada a um *widget* e que poderá receber um valor numérico, *alpha*, responsável por distribuir o peso de sobreposição de cada imagem de entrada na imagem resultante. Este *widget* também deve ser instanciado e atualizado caso a sua porta de entrada receba uma associação.

```
constructor() {
    super(LinearBlender.title);

    this.addOutput("Image", "image", { label: "Image" });

    this.addInput("Image 1", "image", { label: "First Image" });
    this.addInput("Image 2", "image", { label: "Second Image" });

    this.addInput("Alpha", "number");
    this.widget_alpha = this.addWidget(
        "slider",
        "Alpha: ",
        0.1,
        (v) => {
            this.widget_alpha.value = v;
            this.isAlphaChanged = true;
        },
        { min: 0, max: 1 }
    );
}
```

Figura 20. Construtor do bloco *Linear Blender*.

É no método ***onConnectionsChange()***, Figura 21, que é realizada toda a gestão da origem do valor do *alpha*, pois ao surgir uma associação à sua respetiva porta de entrada o valor de *alpha* deve ser atualizado consoante o valor recebido através dessa porta e a interação com o respetivo *(*widget) deve ser desativada. A atualização gráfica do *widget* de acordo com o valor recebido, através da porta de entrada, é realizada no método ***onDrawBackground()***. É ainda neste método, Figura 22, que é atribuído o identificador único ao nome da variável, a ser utilizada aquando da geração de código.

```
override onConnectionsChange(connection, slot, connected, link_info) {
    if (link_info == null) {
        return;
    }

    this.widget_alpha.disabled = false;

    if (this.isInputConnected(2)) {
        this.widget_alpha.disabled = true;
        this.widget_alpha.value = this.getInputNode(slot)?.properties["value"];
    }

    this.inputs[slot].name = this.getInputNode(slot)?.outputs[link_info.origin_slot].name as string;
}
```

Figura 21. Método *onConnectionsChange()* do bloco *Linear Blender*.

Na Figura 23 pode-se verificar o comportamento do *widget* destinado ao *alpha* com e sem a associação à sua porta de entrada, realçando-se que devido à associação o *widget* do bloco encontra-se desativo, mostrando apenas o valor passado, sem possibilidade de interação com o mesmo.

Ao ser iniciado o processo de execução no ***Editor***, é invocado a dada altura o método ***onExecute()*** deste bloco, que começa por verificar se as imagens a serem sobrepostas existem e se as suas propriedades, tais como tamanho e formato, são equivalentes para se conseguir sobrepor as imagens. Caso não sejam preenchidas a totalidade das condições para a sobreposição das duas imagens, a execução é parada e a cor do bloco ganha uma tonalidade encarnada, Figura 24. Resolvidas as incompatibilidades e voltando a executar o ***Editor***, desta vez sem problemas, as imagens de entrada são manipuladas recorrendo à biblioteca ***OpenCV*** em *TypeScript* sendo a imagem resultante colocada na respetiva porta de saída. É neste método, ***onExecute()***, que também são realizadas verificações a possíveis alterações nas imagens de entrada ou no valor de *alpha*, em tempo de execução.

```
override onDrawBackground(ctx: any) {
    this.outputs[0].name = "image" + this.id;

    //update slider when using slot
    if (this.isInputConnected(2)) {
        if (this.widget_alpha.value != this.getInputNode(2)?.properties["value"]) {
            this.isAlphaChanged = true;
        }
        this.widget_alpha.value = this.getInputNode(2)?.properties["value"];
    }
}
```

Figura 22. Método *onDrawBackground()* do bloco *Linear Blender*.

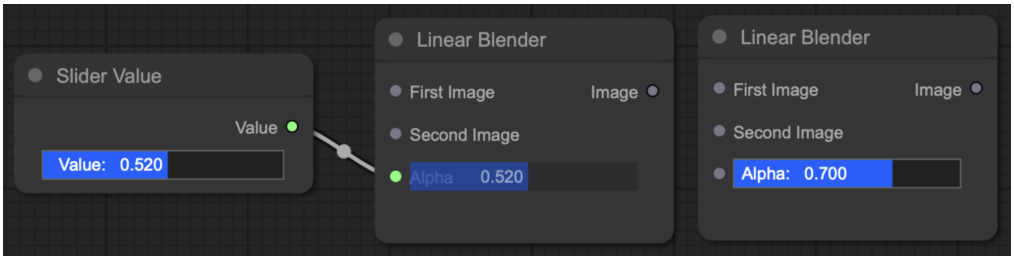

Figura 23. *Widget* do valor *alpha* com e sem associação no bloco *Linear Blender*.

Depois de definido cada um dos blocos, estes devem ser importados e referenciados no módulo ***BlocksModule***, permitindo assim o registo dos blocos na classe ***LiteGraph***. Após isto, cada bloco presente no módulo é carregado na palete de blocos no ***Editor***, mencionado anteriormente, ficando assim disponível para instanciação e utilização no ***Canvas***.

4.2.3 Geração de Modelo

A geração do modelo trata-se de um passo intermédio necessário para geração de código, para tal foi desenvolvido um modelo de dados, tendo em consideração os objetivos e os requisitos definidos e a arquitetura da plataforma de suporte ***LiteGraph***, descrita anteriormente.

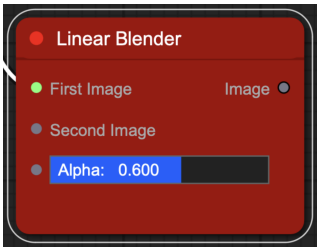

Figura 24. Erro no bloco *Linear Blender* por falta de uma imagem de entrada.

Considerando o modelo desenvolvido, Figura 25, é possível verificar-se que este possui uma classe base, a classe ***Package***. Esta classe possui todas as referências das restantes classes, sendo esta classe a responsável pela geração do ficheiro em formato (JSON) contendo todas as instâncias da sua subclasse, a classe ***Block***.

Esta classe ***Block*** apenas possui como seus atributos, um nome e um identificador único, que são herdados pelas suas subclasses ***Value*** e ***Filter***. Todas estas classes, ***Block*** e as suas subclasses ***Filter*** e ***Value***, possuem ainda uma associação à classe ***Property***.

A classe ***Property*** destina-se a representar propriedades associadas a cada instância de ***Block***, possuindo como atributos um nome e um respetivo valor. Um tipo específico desta classe é a classe ***Slot***, que se destina a representar as portas de entrada e de saída e as suas respetivas associações.

Esta classe ***Slot*** possuí 2 tipos de relação de autorreferência, uma denominada de *"Target"* e outra de *"Source"*. Esta primeira possuí uma cardinalidade de um para muitos, isto é, cada instância de ***Slot*** tem de estar associada a uma outra instância de ***Slot***, garantindo que uma porta de saída está associada a pelo menos uma outra porta de entrada. Já a segunda relação, *"Source"*, possui uma cardinalidade de zero para um, indicando que cada instância de ***Slot*** deste tipo deve estar associado, no máximo, a uma outra instância de *Slot*. Garantindo assim que a cada porta entrada está associada, no máximo, uma porta de saída de um outro bloco.

Introduzido o modelo de dados desenvolvido, descreve-se em seguida a sua relação com a plataforma de suporte ***LiteGraph*** e consequentemente a relação com o protótipo e as suas instâncias no ***Canvas***.

4.2.3.1 Classe Modeler

A transformação das instâncias presentes no ***Editor***, seguindo o modelo de dados descrito acima, apenas é conseguido através de uma classe definida como *Modeler*. Esta classe é a responsável por instanciar a classe *Package*, atribuindo-lhe um nome, disponibilizando assim um método que tem por objetivo toda a transformação necessária. Este método começa por percorrer uma lista de blocos ordenada por ordem de execução, disponibilizada internamente pela plataforma de suporte, como já referido anteriormente.

Desta forma, cada bloco instanciado no ***Editor*** corresponde a uma instância da classe ***Block***, ou uma das suas subclasses ***Filter*** e ***Value*** dependendo das suas portas de entrada e saída. Tipicamente se um bloco só é constituído por apenas portas de saída, é criada uma instância da classe ***Value***, com exceção aos blocos que se destinam ao início de todo fluxo. As propriedades de cada bloco tornam-se instâncias da classe ***Property***, e cada porta de entrada e de saída corresponde a uma instância da classe ***Slot***, que é subclasse de ***Property***.

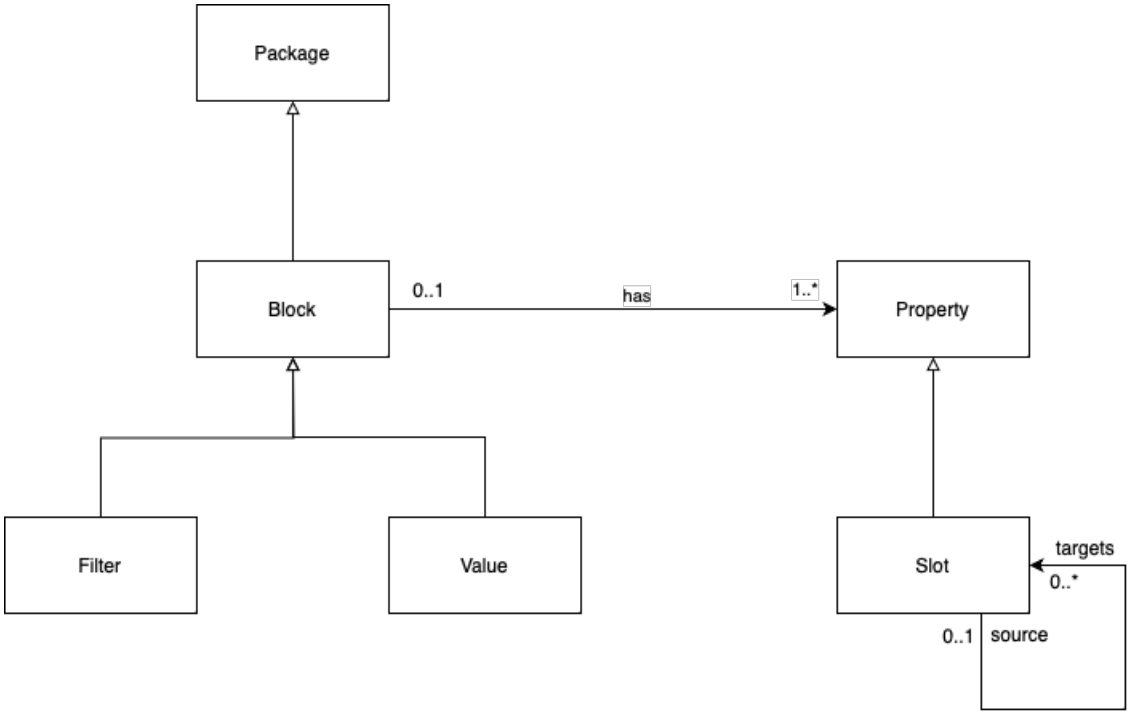

Figura 25. Modelo desenvolvido.

Após a instanciação de todos os blocos e das suas propriedades e respetivas portas de entrada e de saída, é invocado o método ***createAttatchments()*** da classe ***Modeler*** que tem como objetivo a criação das relações de autorreferência entre as instâncias da classe ***Slot***. Para tal é necessário obter a instância da classe ***Block*** à qual se pretende associar cada *slot*, e a partir desta instância de ***Block*** obter a referência da instância de ***Slot*** pretendida das suas propriedades associadas.

Todas estas instâncias, associações e referências encontram-se, por fim, na instância da classe ***Package*** criada antes de todo este processo, estando acessível ainda na classe ***Modeler*** um método responsável pela geração do modelo e de toda esta informação em formato *JSON*, como referido inicialmente. Na Figura 26 apresenta-se um fluxograma, onde é ilustrado todo este processo de transformação das instâncias da plataforma de suporte em instâncias considerando o modelo definido e anteriormente mencionado.

4.2.3.2 Exemplo Completo

Tendo em conta este modelo, apresenta-se o seguindo exemplo prático como forma de clarificar toda a transformação referida acima. Considerando que foi anteriormente utilizado como objeto de exemplo o bloco *"Linear Blender"*, este exemplo deve considerar todo um fluxo que engloba este mesmo bloco. Para tal utilizou-se 2 blocos *"Capture Image"* e 1 bloco *"Show Image"*, além do referido bloco *"Linear Blender"*, como representado na Figura 27.

Na geração do modelo para este exemplo começa-se por instanciar a classe ***Modeler***, que tem como responsabilidade a instanciação da classe ***Package***, sendo posteriormente invocado o método ***createModel()*** da classe ***Modeler*** após esta instanciação.

Este método ***createModel()*** inicia o seu comportamento percorrendo uma lista ordenada contendo todos os blocos instanciados utilizando a plataforma ***LiteGraph***, por ordem de execução.

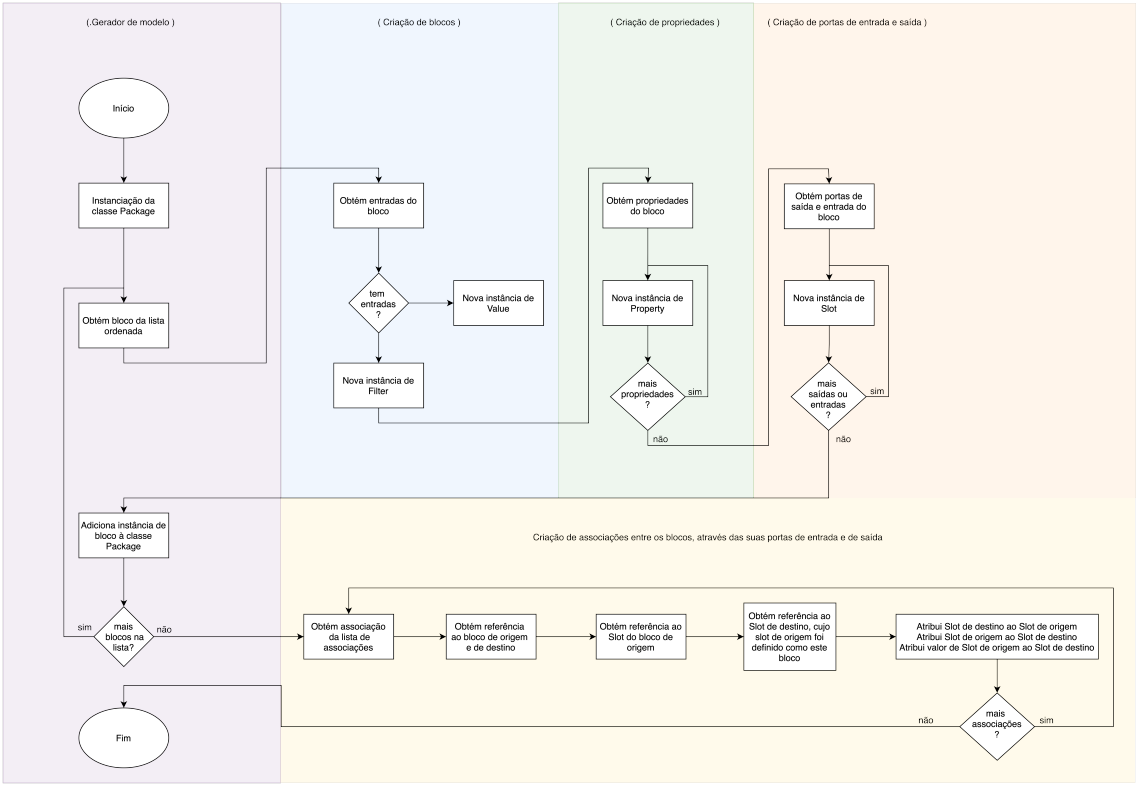

Figura 26. Fluxograma de transformação para o modelo desenvolvido.

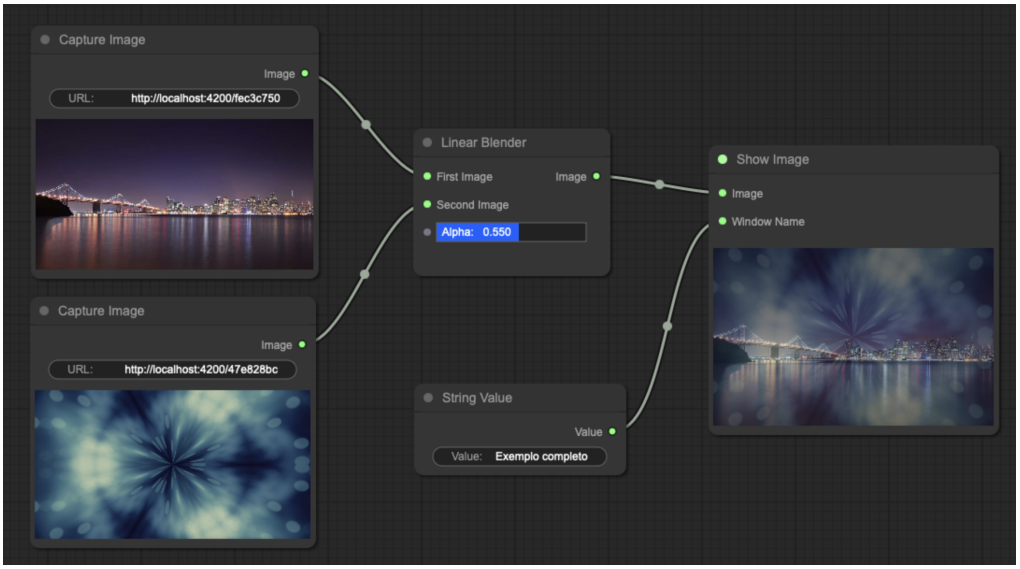

Figura 27. Exemplo completo utilizando o bloco *Linear Blender*.

Sendo esta lista gerada pela plataforma de suporte, como já descrito anteriormente na secção de análise à arquitetura. Para cada um dos blocos nessa lista, são invocados os métodos ***createBlock()***, ***createProperties()*** e ***createSlots()***, que transformam as instâncias da plataforma de suporte ***LiteGraph*** em instâncias do modelo de dados desenvolvido, invocando no fim o método ***addBlock()*** definido na classe ***Package***, conseguindo-se obter na única instância desta classe todos os blocos e respetivas propriedades.

Considerando o bloco alvo deste exemplo *"Linear Blender"*, é criada pelo método ***createBlock()***, uma instância de ***Filter***, que é subclasse de ***Block***, contendo o título e o identificador único do bloco. De seguida é invocado o método ***createProperties()*** que instancia a classe ***Property*** para cada propriedade definida no bloco, que para este bloco consiste essencialmente no valor de *alpha*. Por fim, é invocado o método ***createSlots()*** de forma a se instanciar todas as portas de saída e de entrada deste bloco recorrendo à subclasse ***Slot***, sendo instanciadas 3 portas de entrada e uma porta de saída. Todas estas instâncias são atribuídas à instância do bloco, sendo este atribuído à classe ***Package*** instanciada inicialmente através do método ***addBlock()***. Todo este comportamento é realizado para cada bloco na lista ordenada disponibilizada internamente pela plataforma de suporte.

Finalizados todos os blocos, procede-se à criação das associações entre os mesmos invocando o método ***createAttachments()***. Este método utiliza os identificadores únicos definidos aos blocos e a lista de *links*, ou associações, também disponível internamente na plataforma de suporte ***LiteGraph***, permitindo associar as instâncias da classe ***Slot*** de cada instância da classe ***Block***.

Esta lista de *links* trata-se no fundo, de uma lista que armazena toda a informação de todas as associações existentes entre os diferentes blocos instanciados. A modificação e atualização desta lista acontece de diferentes formas. Sempre que é realizada uma nova associação entre 2 portas de diferentes blocos, é instanciado um novo objeto da classe ***Link*** definida internamente na plataforma de suporte. Este objeto contém toda a informação relevante dos *slots* e dos respetivos blocos associados, sendo adicionado à lista em detalhe neste parágrafo de forma imediata. Esta lista ainda sobre alterações sempre que é adicionada, removida ou alterada uma qualquer associação, tratando-se assim de uma lista dinâmica que se encontra sempre à espera de que surjam alterações às portas de algum dos blocos, garantindo o registo correto de todas as associações definidas no ***Editor***.

De forma a se descarregar o modelo em formato *JSON*, é invocado o método ***save()*** definido na classe ***Package***. Este ficheiro *JSON* contem todas as informações das instâncias, incluindo propriedades, portas de entrada e de saída e as suas respetivas associações, sendo necessário mencionar que, devido a existirem algumas relações cíclicas nomeadamente nas associações entre as instâncias da classe ***Slot***, foram utilizadas referências nestas situações com o objetivo de se evitar ciclos infinitos. Na Figura 28 está representado o resultado em *JSON* referente apenas ao bloco *"LinearBlender"*. O modelo completo adiciona, para cada bloco instânciado, um objeto semelhante ao apresentado na figura 28.

4.2.4 Geração de Código

É a partir do modelo, descrito na subsecção anterior, que se dá a geração do código correspondente em linguagem C++. O modelo, que contém todas as referências a todas as instâncias dos elementos utilizados no ***Editor***, consegue obter acesso aos métodos de cada bloco instanciado no ***Canvas***. Como referido anteriormente, cada bloco possuí um método, ***generateCPP()***, que tem por objetivo devolver o seu respetivo código em C++ incluindo já as suas propriedades definidas, como o nome e o valor das suas variáveis únicas, evitando conflitos quando um determinado bloco é instanciado diversas vezes.

À semelhança do que acontece na geração de código mas aplicado às instâncias do modelo gerado, para se invocar o método ***generateCPP()*** de cada bloco começa-se por percorrer cada instância da classe ***Block***, anteriormente adicionada numa instância da classe ***Package***. Para

```
"_name": "OpenCVStudio",
"_blocks": [
  {
    "_name": "Linear Blender",
    "_id": 3,
    "_properties": [
      {
        "_name": "image3",
        "_value": null,
        "id": 0,
        "_source": null,
        "_targets": [
          {
            "_name": "image3",
            "_value": null,
            "id": 0,
            "_source": {
              "$ref": "$[\"_blocks\"][3][\"_properties\"][0]"
            },
            "_targets": []
          }
        ]
      },
      {
        "$ref": "$[\"_blocks\"][1][\"_properties\"][0][\"_targets\"][0]"
      },
      {
        "$ref": "$[\"_blocks\"][0][\"_properties\"][0][\"_targets\"][0]"
      },
      {
        "_name": "Alpha",
        "id": 2,
        "_source": null,
        "_targets": []
      }
    ],
    "_script": ""
  }
]
```


Figura 28. Bloco *Linear Blender* em formato *JSON*.

cada instância da classe ***Block*** é procurada a sua respetiva instância na plataforma de suporte ***LiteGraph***, através do identificador único atribuído no momento da sua instanciação, conseguindo-se invocar assim o seu método para a geração de código.

Complementando o bloco alvo dos exemplos utilizados anteriormente, descreve-se em seguida o comportamento do bloco *"Linear Blender"* definido na linguagem C++. Como se pode verificar na Figura 29, que contém código do método ***generateCPP()*** deste bloco, começa-se por encontrar a definição de uma matriz da biblioteca ***OpenCV*** com referência ao nome atribuído à porta de saída do bloco, que como referido anteriormente é unido ao identificador único do bloco de forma a não existirem conflitos. De seguida são ainda definidas 2 variáveis destinadas ao valor de *alpha* e ao valor de *beta*, que consiste este último no valor restante à primeira, ou seja, (1-*alpha*). Por fim é definida uma invocação a um método da biblioteca ***OpenCV*** na linguagem C++, o método ***addWeighted()***, que recebe como argumentos as variáveis definidas imediatamente acima e as referências às variáveis definidas provenientes do bloco anterior, associadas pelas portas de entrada deste bloco.

Todo este código na linguagem C++ é retornado à classe responsável pelo início da geração de código, a classe ***Generator***, que no fim acaba por apresentar ao utilizador todo o código concatenado e, com as devidas alterações, pronto a utilizar. Como o modelo é gerado seguindo a ordem de execução, disponibilizada pela plataforma de suporte ***LiteGraph***, e a geração de código fundamenta-se neste mesmo modelo, está assegurada a ordem correta na geração de código. Considerando o exemplo completo utilizado na clarificação do processo de geração do modelo na secção 4.2.3.2, foi gerado o respetivo código para a linguagem C++, como se verifica na Figura 30.

```
generateCPP(): string {
    let cppInputImage_1 = this.inputs[0].name;
    let cppInputImage_2 = this.inputs[1].name;
    let cppOutputImage = this.outputs[0].name;
    let alpha = `alpha${this.id}`;
    let beta = `beta${this.id}`;

    let code = `\n`;
    code += `Mat ${cppOutputImage};` + `\n`;
    code += `double ${alpha} = ${this.alpha};` + `\n`;
    code += `double ${beta} = ( 1.0 - ${alpha} );` + `\n`;

    code +=
        `addWeighted( ${cppInputImage_1}, ${alpha}, ${cppInputImage_2}, beta${this.id}, 0.0, ${cppOutputImage});` + `\n`;

    return code;
}
```

Figura 29. Método de geração de código em C++ para o bloco *Linear Blender*.

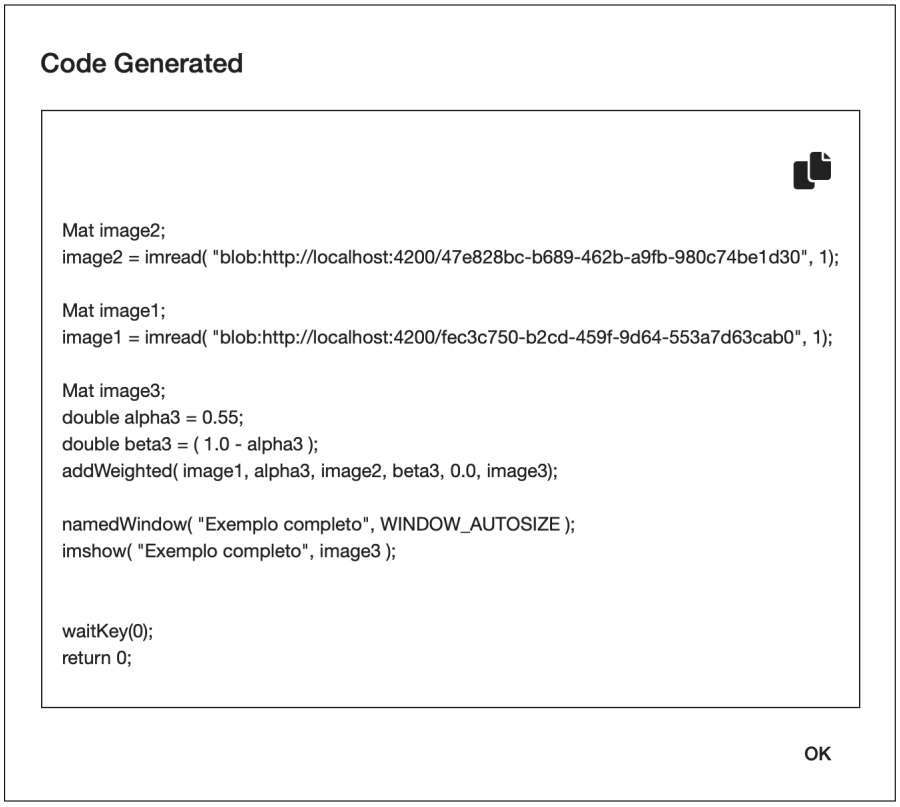

Figura 30. Código gerado na linguagem C++ pelo editor desenvolvido.

Este código foi posteriormente utilizado num ambiente de desenvolvimento em C++ já preparado, com todas as dependências e com a biblioteca ***OpenCV*** instalada, sendo necessário alterar o caminho para as imagens a utilizar. Utilizando as mesmas imagens tanto no ***Editor*** como no ambiente de desenvolvimento em C++, que o resultado é o mesmo, como se pode verificar nas Figuras 27 e 31, reforçando que apenas se alterou o caminho das imagens no código gerado.

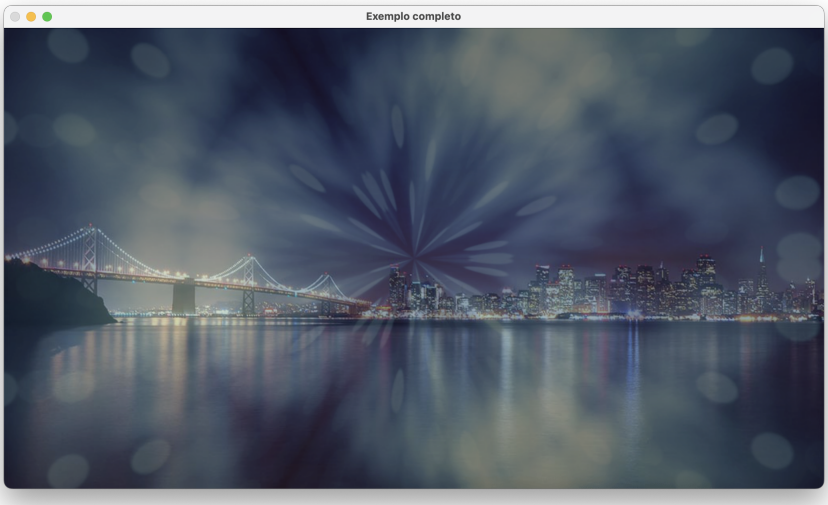

Figura 31. Aplicação do código gerado num ambiente C++.

4.2.5 OpenCV para Web

Como se verificou na subsecção anterior, é apresentado um resultado obtido no ***Canvas*** que é exatamente o mesmo resultado obtido quando executado o código gerado no ambiente de desenvolvimento utilizando a biblioteca ***OpenCV***.

Isto é possível, como já referido, através da utilização de uma versão para *web* da biblioteca ***OpenCV***, que oficialmente encontra-se disponível em *JavaScript*, existindo uma versão reestruturada em *TypeScript* com a designação de ***opencv-ts***.

Utilizou-se esta versão reestruturada em *TypeScript* da biblioteca ***OpenCV*** nos blocos que têm por objetivo elevar a abstração desta mesma biblioteca. Para tal implementou-se um método designado ***openCVWeb()*** para cada um desses blocos que, como já foi referido nos exemplos anteriores, é invocado no método ***onExecute()*** do respetivo bloco. Garante-se assim que as funcionalidades da biblioteca ***OpenCV*** apenas são aplicadas após a execução e a propagação dos valores dos blocos, conseguindo-se ainda atualizar os valores em tempo de execução quando existirem alterações no ***Editor***.

Voltando ao exemplo utilizado anteriormente e considerando o bloco *"Linear Blender"*, pode-se verificar na Figura 32, que é no método ***onExecute()*** do bloco onde é invocado o método ***openCVWeb()*** responsável pela aplicação das funcionalidades da biblioteca ***OpenCV*** representadas no ***Canvas***. Este método para o bloco em consideração tem um comportamento semelhante ao método para geração de código, descrito anteriormente. Como se verifica na Figura 33, apenas é instanciada um matriz da biblioteca ***OpenCV***, sendo posteriormente invocado um outro método, o método ***addWeighted()*** responsável por sobrepor as duas imagens recebidas pelo bloco de acordo com o valor de alpha defindo e colocando a imagem resultante na matriz instanciada, que é propagada pela porta de saída a outro bloco como é possível verificar na Figura 32 respetiva ao método ***onExecute()***.

A pré-visualização da imagem resultante trata-se de uma funcionalidade mais complexa devido à biblioteca ***OpenCV*** apenas representar a matriz da imagem num elemento de *HTML*

```
override onExecute() {
    if (this.updateMats()) {
        this.setOutputData(0, this.matOut);
        return;
    }

    if (this.isInputsCompatible()) {
        this.openCVWeb();
        this.setOutputData(0, this.matOut);
        return;
    }

    this.setOutputData(0, null);
}
```

Figura 32. Método *onExecute()* do bloco *Linear Blender*.

```
openCVWeb(): void {
    this.alpha = this.isInputConnected(2) ? Number(this.getInputData(2)) : this.widget_alpha.value;
    this.isAlphaChanged = false;

    this.matOut = new cv.Mat(this.mats[0].rows, this.mats[0].cols, this.mats[0].type());
    cv.addWeighted(this.mats[0], this.alpha, this.mats[1], 1 - this.alpha, 0, this.matOut);
}
```

Figura 33. Método que aplica a biblioteca *OpenCV* para *web* do bloco *Linear Blender*.

denominado por *canvas*. Esta matriz renderizada nesse *canvas* precisa de um processamento que transforme o conteúdo do mesmo numa imagem de formato comum, como *PNG* ou *JPG*, sendo então redimensionada e desenhada no respetivo bloco, tipicamente no bloco *"Show Image"*, através do método ***onDrawBackground()*** disponibilizado pela plataforma de suporte ***LiteGraph*** como já referido ao longo desta secção.

4.3 Limitações no desenvolvimento

Nesta subsecção encontram-se descritas as dificuldades e as limitações encontradas ao longo da implementação deste protótipo e as soluções implementadas para combater as mesmas. Estas limitações encontram-se pela ordem em que surgiram e foram tratadas, sendo de mencionar que determinadas soluções implementadas deram a origem a novas limitações e, em certas situações facilitaram a implementação de uma solução para outras limitações já existentes.

4.3.1 Geração de código

As primeiras limitações que surgiram foram em relação à geração de código da biblioteca ***OpenCV*** para a linguagem de programação C++. Como já referido na secção 4.2.4, o código gerado tem de aceder a instância de cada bloco e agregar o respetivo código no fim, sendo que para tal acedia-se à instância de cada bloco a partir do seu tipo e do seu título. O primeiro problema identificado está relacionado à repetição na instanciação de blocos com diferentes responsabilidades e blocos associados. Embora os blocos tenham finalidades distintas, compartilham os mesmos atributos, o que dá origem a conflitos na geração de código. Estes conflitos ocorriam devido à falta de distinção entre as variáveis de cada bloco, levando à repetição e redefinição de variáveis.

Para combater esta limitação definiu-se uma nova propriedade em cada um dos blocos, um identificador único, que seria responsável por diferenciar cada uma das instâncias de um mesmo tipo de bloco. Através deste identificador conseguiu-se garantir que para cada instância de bloco é apenas gerado uma única vez o seu respetivo código.

Com esta limitação superada, constatou-se que as variáveis das diferentes instâncias de um mesmo bloco possuíam a mesma identificação, fazendo com que o último bloco gerado em código seja o responsável pelo valor atribuído à variável. Esta segunda limitação foi superada aproveitando o identificador único definido para cada instância de bloco, sendo este identificador concatenado ao nome das variáveis de cada bloco. Esta abordagem foi aplicada aos nomes atribuídos às portas de saída e de entrada de cada bloco, dando origem assim a mais uma limitação: a variável de uma porta saída de um bloco possui um identificador diferente da variável de uma porta de entrada de outro bloco. Corrigiu-se esta nova limitação com recurso a um dos métodos descrito anteriormente, ***onConnectionChange()***, tornado-se possível, durante a associação de dois blocos, obter o identificador do bloco associado à porta de entrada e atualizar o respetivo nome de acordo com este identificador.

Após estas implementações todas, verificou-se que a geração de código já não se repetia e que já existia a correta distinção e associação das diferentes variáveis. Um novo problema foi encontrado na geração de código que surge quando os blocos não são instanciados na ordem que devem ser executados, pois os identificadores encontram-se pela ordem de instanciação dos blocos. Para tal voltou-se a analisar a plataforma de suporte, encontrando-se a solução ideal para esta limitação que consiste na utilização de uma lista dos blocos instanciados já pela ordem de execução, esta descrita na subsecção 4.1.2.

4.3.2 OpenCV para Web

A implementação da biblioteca ***OpenCV*** para *web*, através da versão comunitária escrita em *TypeScript*, trouxe desafios e limitações. A principal limitação tem impacto no desempenho do processamento realizado pois, por se tratar de uma versão *web*, não existe suporte a aceleração de *hardware* recorrendo à GPU. Outra limitação consiste no leque limitado de leitura de formatos de imagem e de vídeo, e na quantidade ainda limitada de funcionalidades da biblioteca. Embora determinadas funcionalidades não se encontrem implementadas, é possível definir-se os blocos dessas funcionalidades com o intuito de apenas se gerar código para a linguagem C++, sacrificando-se apenas a visualização do seu resultado no ***Editor***.

A utilização desta versão para *web* da biblioteca ***OpenCV*** trouxe ainda algumas dificuldades na utilização das funcionalidades, isto devido aos parâmetros diferentes que a mesma funcionalidade recebe. Como o método ***Scalar()*** utilizado no bloco *"HSV Filter"*, que na sua versão em C++ recebe 3 parâmetros para a cor, saturação e brilho, enquanto que o mesmo método para a versão *web* recebe um quarto parâmetro destinado à opacidade. Outra diferença mais fundamental centra-se nos canais de cor que, como referido múltiplas vezes, a versão em C++ utiliza o formato *BGR*, enquanto a versão *web* já utiliza o formato mais comum, *RGB*, devendo-se ter especial atenção na utilização da funcionalidade para *web*, na respetiva geração de código, e no envio de cada canal de cor através das portas dos blocos, em especial os blocos *"Split"* e *"Merge"* que trabalham a partir destes canais de cor.

Outra das principais limitações da utilização da biblioteca ***OpenCV*** para *web* acontece na pré-visualização da matriz resultante de todas as transformações aplicadas. Esta limitação surge

devido a esta matriz apenas ser visível através de um elemento *canvas* do *HTML*, sendo necessário a criação de um elemento deste tipo escondido. Após definido um *canvas* onde a biblioteca ***OpenCV*** pode carregar a sua matriz, torna-se necessário colocá-la no bloco *"Show Image"* para o utilizador conseguir visualizar o resultado. Para tal é necessário transformar o conteúdo do *canvas* criado num objeto em formato de imagem, tipicamente em *PNG*, obtendo-se o respetivo caminho associado, ou seja, o seu *URL*. A partir deste *URL* é carregado num elemento de imagem do *HTML* este objeto que representa o resultado das transformações da biblioteca ***OpenCV***. Recorrendo ao método introduzido anteriormente, ***onDrawBackground()***, torna-se possível desenhar a imagem resultante na respetiva instância do bloco *"Show Image"*.

4.3.3 Double Slider

Ao dedicar-se à verificação da funcionalidade do ***OpenCV*** para a *web* e da geração de código, notou-se que um dos blocos implementados, *"HSV Filter"*, possuí 3 variáveis que devem ter um valor mínimo e um valor máximo, e que a utilização de 6 *sliders* para cada um destes valores tornava o ***Editor*** confuso e repetitivo. Decidiu-se que seria interessante implementar um bloco cujo *slider* permita definir um valor mínimo e um valor máximo de forma simultânea, dando assim origem ao bloco *"Double Slider"*. A maior dificuldade na implementação deste bloco consiste na definição da interação com o *slider*, pois teve-se de considerar limites para o valor máximo ser sempre superior ao valor mínimo, implementando-se marcadores escondidos no próprio *slider*. A interação do cursor com este *slider* e com estes marcadores foi definida através da coordenada do cursor dentro do próprio bloco, onde as coordenadas mais à esquerda no bloco refere-se à definição do valor mínimo e as coordenadas mais à direita refere-se ao valor máximo, nunca deixando os marcadores se inverterem. Estes valores correspondentes ao mínimo e máximo são colocados num vetor na sua porta de saída, sendo recebido pela porta de entrada do bloco de destino na mesma forma de vetor.

4.3.4 Camera Capture

Como referido na secção 3.4, foi decidido implementar-se um bloco que fizesse uso de vídeo, ou imagens, em tempo real através de uma câmera, denominando-se este bloco por *"Camera Capture"*. Por se pretender que este bloco faça a recolha e o processamento de imagem em tempo real, uma nova estratégia de execução dos blocos foi implementada. Esta estratégia garante que imediatamente após uma execução de todos os elementos no ***Canvas*** ocorre uma nova execução, até que algum problema seja detetado ou que o utilizador decida parar a execução.

Esta estratégia gerou problemas ao nível de processamento e transformação da imagem capturada, devido a esta transformação não ser imediata e precisar de algum tempo de processamento. Para superar este problema, decidiu-se limitar a quantidade de *frames* recolhidos e processados pelo bloco, conseguindo-se assim manter a matriz resultante sincronizado com a imagem capturada pela câmera.

Esta redução na quantidade de *frames* capturados juntamente com a nova abordagem de execução dos blocos deu origem a um problema quer na imagem capturada pela câmera, quer pela imagem resultante das transformações aplicadas. Este problema é conhecido como flickering, que consiste na cintilação constante das imagens, que ocorre devido à atualização dos *frames* capturados e processados. Para superar este problema utilizou-se uma técnica denominada por *double buffering*, que apenas armazena a imagem capturada e a imagem resultante em memória. Estas

imagens armazenadas são as que se encontram visíveis ao utilizador no ***Editor***, sendo que apenas são substituídas após concluído todo o processo de transformação dos diversos blocos instanciados. Conseguindo-se assim garantir o correto processamento e a completa apresentação das transformações da biblioteca ***OpenCV*** em tempo de execução.

De realçar ainda que, devido à implementação da nova abordagem de execução para este bloco, os restantes blocos conseguem atualizar instantaneamente o seu resultado quando aplicadas alterações às propriedades de cada bloco, sem a necessidade de executar novamente o ***Editor***.

4.3.5 Problemas de memória

Devido à implementação da nova abordagem de execução, cada vez que era aplicada uma mudança, ou a cada *frame* da câmera processado, existia a criação de uma nova matriz da biblioteca ***OpenCV*** acompanhada da sua representação num elemento *canvas* do *HTML*, escondido. Esta criação e instanciação de *canvas* e de matrizes da biblioteca ***OpenCV*** deu origem a problemas de memória, devido à essencialmente haver uma nova instância a cada segundo.

Para combater este problema de memória começou-se por destruir as instâncias depois de utilizadas, garantindo a existência mínima de instâncias. Esta solução deu origem a um novo problema, pois a instanciação de uma matriz que ocorria num primeiro bloco precisava ser enviada para o bloco seguinte, através das associações. Sendo que a instância era destruída, quando se aplicava uma alteração em algum dos blocos com o intuito de se aplicar determinados filtros à matriz, esta matriz por já não se encontrar em memória acabava por gerar um novo problema.

Para superar este novo problema começou-se por aplicar as funcionalidades dos blocos através de clones, isto é, um bloco ao receber uma matriz, faz de imediato uma cópia da mesma eliminando a original. Sobre esta cópia são então aplicadas as funcionalidades e os filtros da biblioteca ***OpenCV*** definidas no bloco, sendo esta cópia modificada a enviada para os blocos seguintes, em que cada bloco que recebeste esta matriz aplica todo o comportamento novamente, realizando logo de início uma cópia à matriz recebida.

Visto que o bloco *"Camera Capture"* é o que mais atualizações consegue realizar num curto período de tempo e, por sua vez, é o que mais rapidamente consegue gerar problemas de memória, instanciou-se este bloco juntamente com alguns filtros e deixou-se o ***Editor*** em execução durante algum tempo de forma a se verificar a ocorrência de problemas de memória. Como se pode verificar pela Figura 34, foram processados mais de 27 mil *frames* durante cerca de 1 hora sem apresentar de nenhum problema de memória.

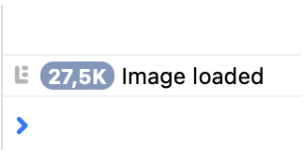

Figura 34. *Debug* de frames processados.

4.3.6 Geração de modelo em formato *JSON*

Como mencionado na secção 4.2.3, todas as instâncias realizadas no ***Editor*** são transformadas de acordo com o modelo definido anteriormente, permitindo descarregar e guardar todas as instâncias

num ficheiro de formato *JSON*. Uma limitação nesta transformação surgiu devido às associações entre diferentes blocos, pois uma porta de entrada possui como origem a porta de saída do bloco anterior, e esta porta de saída possuí como destino a referida porta de entrada. Estas propriedades e associações acabam por tornar a estrutura circular, existindo ciclos de objetos repetidos de forma recursiva. Utilizou-se uma biblioteca *open-source*, denominada por ***json-decycle***, que resolve estes ciclos de objetos através da utilização de referências em *JSON* [66].

5 Avaliação

Nesta secção encontra-se descrito todo o procedimento realizado para a avaliação da plataforma desenvolvida. Foi realizada uma experiência com três utilizadores, com o objetivo de se avaliar a usabilidade da plataforma *low-code* desenvolvida e de se validar a hipótese de que a abstração implementada sobre a biblioteca ***OpenCV***, através desta plataforma *low-code*, daria origem a um ciclo de desenvolvimento mais ágil, rápido e simples aos utilizadores que já possuem algum conhecimento sobre a respetiva biblioteca.

A experiência definida consistiu em quatro tarefas. A primeira centrou-se num tutorial que teve por único objetivo a introdução do fluxo de utilização da biblioteca ***OpenCV***, quer através de programação tradicional quer através da plataforma *low-code* desenvolvida. Nesta tarefa foram descritos cada um dos passos que o utilizador tinha de realizar para a conclusão da mesma. Para as restantes tarefas, a dificuldade aumentou, pois, o utilizador deveria concluir as mesmas utilizando apenas alguma documentação disponibilizada e uma breve descrição das etapas necessárias, Anexo A.1.

Cada uma destas tarefas encontra-se dividida em 2 fases. A primeira fase consiste em concluir a tarefa na linguagem de programação C++. A fase seguinte tem como principal foco a utilização da plataforma *low-code* desenvolvida para a conclusão da mesma tarefa.

A realização de cada uma destas tarefas, por cada um dos utilizadores, foi registada em vídeo de forma a ser analisado posteriormente. Procurou-se registar e identificar os comentários, os comportamentos, os erros e os tempos de realização e conclusão de cada tarefa.

Por fim, cada utilizador preencheu um simples questionário que visa medir a usabilidade da plataforma de abstração desenvolvida, Anexo C. Para tal recorreu-se à principal escala de usabilidade, a ***System Usability Scale (SUS)***. Esta permite medir, de uma forma quantitativa, a usabilidade de um sistema, através de uma avaliação entre 0 e 100. Este valor é calculado através de uma fórmula específica que é aplicada aos valores atribuídos a cada uma das afirmações do questionário. O utilizador deve atribuir a cada afirmação um valor entre 1 e 5, conforme está de acordo ou não com a sua opinião.

5.1 Participantes

Nesta avaliação participaram três utilizadores com idades compreendidas entre os 27 e os 37 anos. O critério de inclusão estabelecido especifica que os participantes devem possuir experiência de programação, em especial na linguagem de programação C++. O critério fundamental consiste no conhecimento ou utilização da biblioteca ***OpenCV***, ou similares com foco na visão computacional.

Estes participantes afirmaram possuir mais do que cinco anos de experiência de programação. Todos já programaram em C++, apesar de não ser a linguagem de programação em que estão mais confortáveis. Em relação à biblioteca ***OpenCV***, é conhecida e já foi utilizada por todos os participantes.

5.2 Metodologia

Os testes de usabilidade foram divididos, como já referido, em duas abordagens. A primeira abordagem consistiu na execução das tarefas propostas utilizando a linguagem de programação C++. Na

outra abordagem, os utilizadores tiveram que concluir as mesmas tarefas utilizando a plataforma *low-code* desenvolvida, a fim de gerar o respetivo código na linguagem de programação C++.

Foram propostas as seguintes quatro tarefas aos utilizadores em que a tarefa inicial, tarefa 0, foi utilizada como tutorial de forma a introduzir a biblioteca ***OpenCV*** na linguagem C++ e a introduzir o funcionamento base da plataforma desenvolvida. Para esta tarefa foram descritos e detalhados cada um dos passos necessários à conclusão da mesma, Anexo A.1.1.

Para as três tarefas restantes, os utilizadores apenas tiveram acesso à documentação das funcionalidades e respetivas assinaturas, na linguagem C++, consideradas úteis. Tiveram ainda uma tabela resumo dos blocos implementados na plataforma *low-code*, Anexo A.3. Cada uma destas tarefas apenas enumerava os principais passos para a conclusão das mesmas.

- **Tarefa 0: Troca de canais de cor**: também conhecida como visão de baixo nível, envolve as primeiras fases de processamento necessárias para uma tarefa visual. Um aspeto desta primeira fase é a análise de características, em que é extraída informação sobre cor, movimento, forma, textura, profundidade estéreo, e intensidade dos contornos. Outro aspeto da visão inicial é a segmentação da imagem, em que a informação das características é utilizada para segmentar a imagem em regiões que têm uma elevada probabilidade de ter surgido a partir de uma única causa física.

- **Tarefa 1: Sobreposição de duas imagens**: esta tarefa consistiu numa simples sobreposição linear de duas imagens, a fim de preparar os utilizadores para as seguintes, e mais complexas, tarefas.

- **Tarefa 2: Identificar objetos verdes a partir da câmera**: a tarefa número dois consistiu na utilização da câmera do dispositivo, para a deteção e identificação de objetos de cor verde, em tempo real. Nesta tarefa os participantes tiveram de detetar e identificar, através do desenho de retângulos vermelhos, diversos objetos de cor verde capturados pela câmera do dispositivo.

- **Tarefa 3: Identificar o logótipo da UMa**: nesta última tarefa, foi solicitado aos participantes a identificação da presença de um logótipo específico em três imagens selecionadas. O logótipo fornecido para este propósito foi o da Universidade da Madeira, que serviu como referência para a deteção. Os participantes tiveram então de identificar o logótipo da universidade nas três imagens fornecidas através da definição de um retângulo nas respetivas correspondências.

A realização destas tarefas foi registada em vídeo, de forma a se conseguir analisar os comportamentos, comentários, dúvidas e sugestões dos utilizadores. Para tal recorreu-se à técnica *Think Aloud*, em que se solicitou aos utilizadores a projetar os seus pensamentos, a questionar o sistema e a sugerir melhorias à plataforma. A avaliação finalizou-se através de um formulário *SUS*, Anexo C, em que os utilizadores atribuíram uma pontuação a cada uma das dez afirmações de avaliação.

5.2.1 Think Aloud

No método de investigação ***Think Aloud***, os participantes dizem em voz alta aquilo que pensam enquanto completam a tarefa proposta. O participante não é interrompido com questões durante a realização das tarefas, nem é distraído do principal objetivo de concluir a tarefa. As dúvidas e sugestões são enunciadas no decorrer da realização das tarefas, de uma forma natural.

Esta técnica é uma das formas mais eficazes de avaliar os processos de pensamento, que envolvem memória. Esta pode ainda ser utilizada para estudar as diferenças individuais dos diversos participantes, na execução da mesma tarefa [7].

Porém, os protocolos desta técnica não são completos. Os utilizadores podem não projetar tudo o que pensam e todas as dúvidas que surgiram no decorrer da tarefa. No entanto, é considerado um método fácil para os participantes, por terem toda a liberdade de expressão e de utilização da sua própria linguagem [7].

5.2.2 System Usability Scale

A ***System Usability Scale (SUS)*** é uma das principais ferramentas utilizadas para a avaliação da usabilidade de um produto, ou sistemas, tais como *sites* e aplicações. Esta escala possuí um valor de avaliação entre 0 e 100, cujo um valor inferior a 68 pontos indica uma fraca usabilidade do sistema [7].

Para se obter este valor, cada utilizador deve atribuir uma avaliação entre 1 e 5, a cada uma das dez afirmações que compõem o formulário desta escala. O valor um, o mais baixo da escala, corresponde a completa discordância com a afirmação. Por outro lado, o valor mais elevado da escala (cinco), indica uma forte aceitação da afirmação [7].

Dos dez elementos a serem avaliados, metade constituem uma afirmação negativa à usabilidade do sistema. As restantes cinco afirmações entregam pontos positivos à usabilidade. No formulário, estas afirmações encontram-se intercaladas entre si. As afirmações ímpares (1, 3, 5, 7 e 9) contribuem para o valor de usabilidade final, com a subtração de uma unidade à respetiva avaliação atribuída pelo utilizador. Já as afirmações pares (2, 4, 6, 8 e 10) contribuem de forma ligeiramente diferente, pois é subtraído a avaliação atribuída um valor constante de cinco pontos [7].

Finaliza-se esta avaliação através do somatório de todas as contribuições, multiplicando-se ainda pelo valor de 2,5. Esta avaliação obtida permite inferir uma classificação qualitativa, considerando a escala na Figura 35 [7].

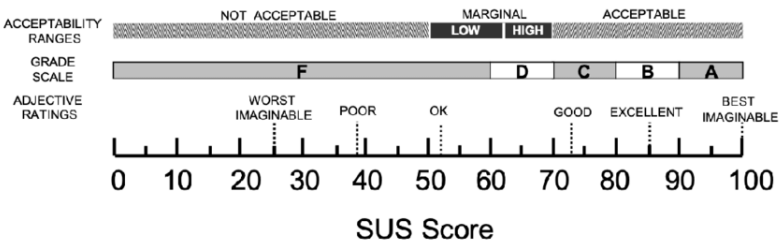

Figura 35. Escala de Usabilidade [7].

Utilizou-se esta escala na sua versão portuguesa, devido a todo o guia e descrição das tarefas se encontrarem em português. Esta versão da escala foi traduzida e adaptada culturalmente, e devidamente validada em 2015 [67].

5.3 Avaliação Qualitativa

Nesta subsecção apresenta-se a avaliação qualitativa realizada através da técnica ***Think Aloud***, introduzida anteriormente. Esta avaliação foi dividida por tarefa, devido ao facto dos participantes partilharem alguns dos comportamentos e opiniões sobre a ferramenta desenvolvida.

A análise de cada tarefa encontra-se dividida em duas partes, uma dedicada à realização da mesma recorrendo à programação tradicional, através da linguagem de programação C++, e outra com foco na experiência de utilização da plataforma *low-code* desenvolvida nesta dissertação. Após a análise de cada tarefa, são apresentados os tempos médios dos participantes, em ambas as abordagens.

5.3.1 Tarefa 0 - Tutorial

Definiu-se esta tarefa tutorial com o objetivo de apresentar aos participantes o objetivo e o fluxo de funcionamento da plataforma *low-code* implementada, de uma forma prática. Esta trata-se de uma tarefa simples que tem por objetivo a troca dos canais de cor de uma determinada imagem, Figura 36.

Nesta tarefa o utilizador teve à sua disposição a solução proposta, para a abordagem em programação tradicional, além da documentação de um conjunto de funcionalidades consideradas úteis. Para a realização desta tarefa, recorrendo à plataforma desenvolvida, foi disponibilizado uma tabela resumo dos blocos implementados e um guia detalhado que descreve cada um dos passos necessários à conclusão da tarefa.

Verificou-se uma recorrente utilização da solução e da documentação fornecida, para a implementação de cada uma das funcionalidades necessárias à tarefa. Apenas um dos participantes não sentiu necessidade de recorrer à documentação nem à solução para a implementação de algumas das funcionalidades, mencionando que considerou que a utilização seria semelhante à linguagem Python, em que está mais ambientado a utilizar a biblioteca ***OpenCV***.

Já na utilização da ferramenta *low-code* proposta, todos os participantes começaram por assumir que a instanciação dos blocos deveria ser realizada através do arrastar do mesmo da palete de blocos para o ***Editor***. Rapidamente perceberam que deveriam selecionar o bloco e então selecionar a posição onde instanciar o mesmo no ***Editor***. Cada um dos participantes percebeu, de uma forma intuitiva, como realizar a interligação entre os diversos blocos instanciados. Identificaram corretamente os botões destinados à execução dos blocos, à geração do modelo de dados e à geração de código na linguagem de programação C++.

Foi registado um tempo médio necessário para a conclusão desta tarefa tutorial, através da programação na linguagem C++, de cerca de 17 minutos. Através da plataforma proposta, este tempo médio reduziu-se a 9 minutos.

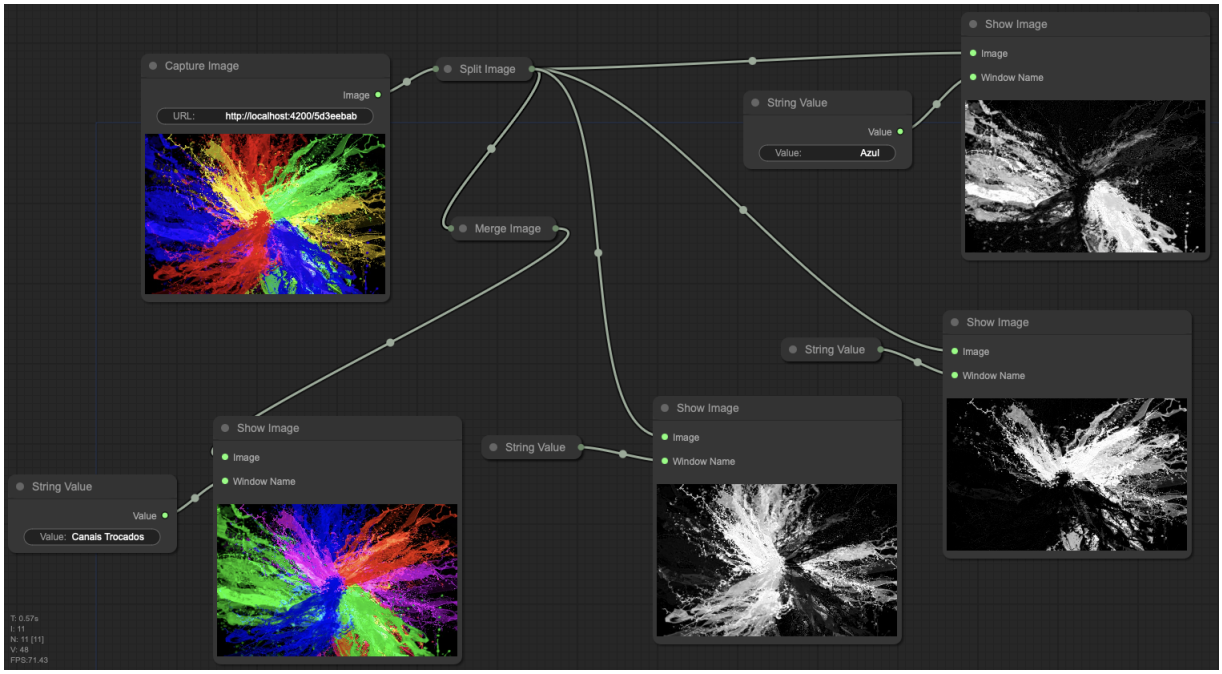

Figura 36. Resultado da tarefa tutorial.

5.3.2 Tarefa 1 - Sobreposição de 2 imagens

A tarefa mais simples, consistiu na sobreposição de 2 imagens de uma forma linear, Figura 37. Para esta tarefa foi apenas disponibilizado, aos participantes, a documentação das funcionalidades em linguagem C++ e a descrição dos blocos implementados na ferramenta *low-code* desenvolvida, ao contrário da tarefa anterior, não foi disponibilizada uma proposta de solução.

Para a implementação desta tarefa, através do desenvolvimento tradicional na linguagem de programação C++, os utilizadores recorreram à documentação fornecida, para a implementação de cada funcionalidade. Apesar de se notar alguma resistência, todos os participantes concluíram sem dificuldade esta tarefa.

Através da plataforma *low-code* desenvolvida, verificou-se que os participantes reconheceram de forma instantânea os blocos a utilizar para a conclusão desta tarefa. Porém, um dos participantes sugeriu alterar nome do bloco "*Linear Blender*"para outro mais intuitivo, indicando o nome "*Merge Image*"como exemplo. Foi também indicado um outro aspeto a ter em atenção, a interação com os *sliders* para a definição de valores numéricos. Alguns participantes referiram sentir dificuldade para definir um valor exato nestes elementos e, sugeriram a introdução de botões ou de uma caixa de entrada que permitam um controlo mais refinado do valor a definir pelo utilizador.

Esta simples tarefa precisou de, em média, 8 minutos, considerando o desenvolvimento tradicional. Através da ferramenta utilizando os blocos, verificou-se uma redução para 4 minutos, sensivelmente metade do tempo necessário na abordagem tradicional.

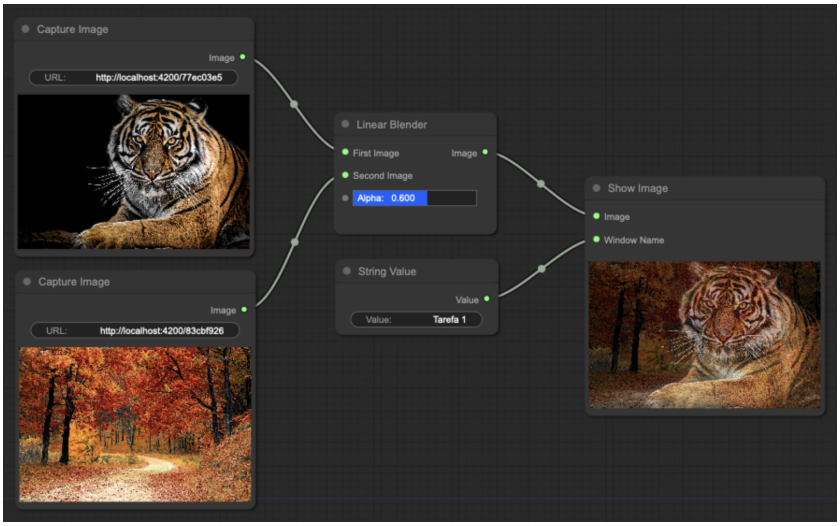

Figura 37. Resultado da tarefa 1.

5.3.3 Tarefa 2 - Identificar e marcar objetos verdes a partir da câmera

Na tarefa seguinte os participantes tinham por principal objetivo identificar e marcar, através de um retângulo vermelho, todos os objetos de cor verde colocados à frente da câmera do dispositivo, Figura 38.

A sua implementação através de programação na linguagem C++ mostrou ser complexa, com os participantes a recorrer ativamente à documentação fornecida. Foi disponibilizado a secção de código de acesso à câmera do dispositivo e restantes validações e autorizações associadas à mesma, devido a este ser um código extenso que não introduz valor à avaliação. Os participantes mostraram enorme dificuldade na definição dos limites inferiores e superiores de cor. Esta dificuldade surgiu do facto de que os limites devem ser aplicados, de forma individual a cada elemento do formato HSV. Surgiu outra dificuldade na definição dos retângulos sobre os objetos identificados, pois os participantes consideraram confuso a necessidade de passar argumentos por referência de memória.

A implementação desta tarefa, através da ferramenta *low-code*, foi concluída num período inferior, contudo, apesar de notar-se uma maior agilidade no desenvolvimento, os participantes apresentaram alguma dificuldade na identificação do bloco "*Double Slider*", que permite a definição dos limites mínimos e máximos no mesmo elemento visual. Novamente, um dos participantes realçou que a atribuição dos valores deste mesmo bloco poderia ser mais intuitivo. Este participante sugeriu a utilização de etiquetas e marcadores, ou outro elemento visual, que facilite a correta identificação e interação dos valores mínimo e máximo do bloco. Um outro participante revelou alguma confusão nos limites de cor a definir, devido a não corresponder diretamente com o intervalo típico, pois o intervalo habitual é definido entre 0 e 255, enquanto que o bloco implementado apresenta os valores entre 0 e 1, realizando-se internamente a sua transformação.

Esta tarefa, realizada na linguagem C++, durou em média 28 minutos por cada um dos 3 participantes. Através da plataforma desenvolvida, verificou-se uma redução significativa para apenas 7 minutos, registando-se uma grande vantagem na agilidade de implementação.

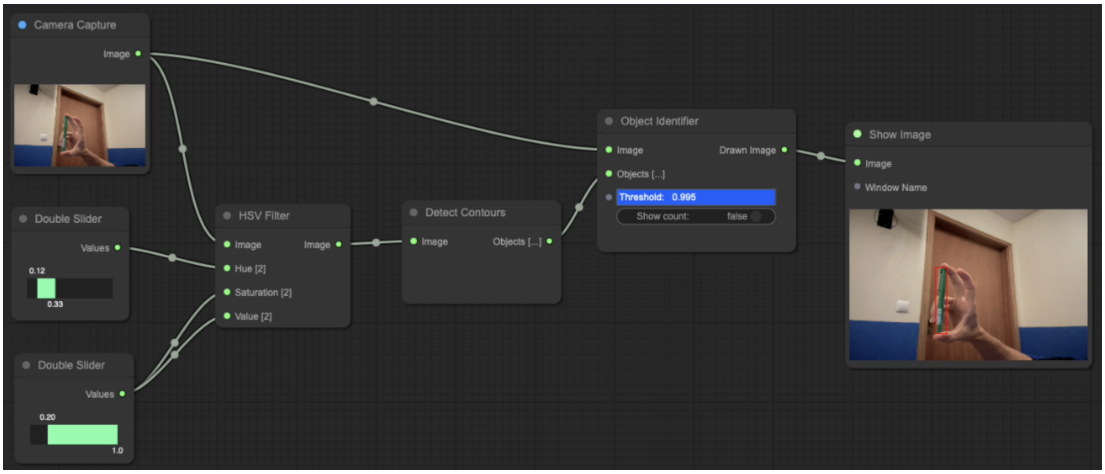

Figura 38. Resultado da tarefa 2.

5.3.4 Tarefa 3 - Identificar e marcar o logótipo da UMa

Como tarefa final, os participantes tinham por objetivo identificar, através de um retângulo vermelho, o logótipo da universidade da Madeira em cada uma das três imagens disponibilizadas 39. A funcionalidade a implementar consistia essencialmente na procura por correspondências da imagem que representa o logótipo da universidade nas respetivas 3 imagens.

O desenvolvimento deste código para a conclusão desta tarefa deu origem a muitas das dificuldades já identificadas nas tarefas anteriores. Estas dificuldades obrigaram à consulta recorrente da documentação fornecida, incluindo para a implementação das funcionalidades básicas como carregar e mostrar as imagens. A interação sobre as correspondências encontradas e a definição dos retângulos nas mesmas geraram novamente alguma dificuldade, pois nesta tarefa a técnica de instanciação dos retângulos consiste num processo mais complexo. Processo no qual cada correspondência identificada apenas fornece um ponto, o que obriga ao programador a criar um retângulo a partir deste mesmo ponto em conjunto com o tamanho da imagem que contém o logótipo.

Os participantes, ao recorrerem à plataforma desenvolvida para a realização desta mesma tarefa, não mencionaram nenhuma dificuldade na identificação dos blocos a utilizar. As interligações entre os diversos blocos foram aplicadas de uma forma intuitiva. Um dos participantes mencionou a enorme facilidade e utilidade da identificação de um dado logótipo numa nova imagem de forma instantânea, precisando apenas de arrastar as novas imagens para os blocos correspondentes. Outro participante sugeriu a atribuição do nome "*Input Image*"para o bloco definido como "*Capture Image*", devido à confusão que pode surgir com o bloco de captura de vídeo da câmera, que possui o nome "*Camera Capture*".

A realização desta tarefa durou, em média, cerca de 16 minutos recorrendo à linguagem de programação C++. Já na plataforma disponibilizada, verificou-se novamente uma maior rapidez e agilidade no processo de desenvolvimento, registando-se uma duração de aproximadamente 5 minutos, em média, para cada um dos três participantes.

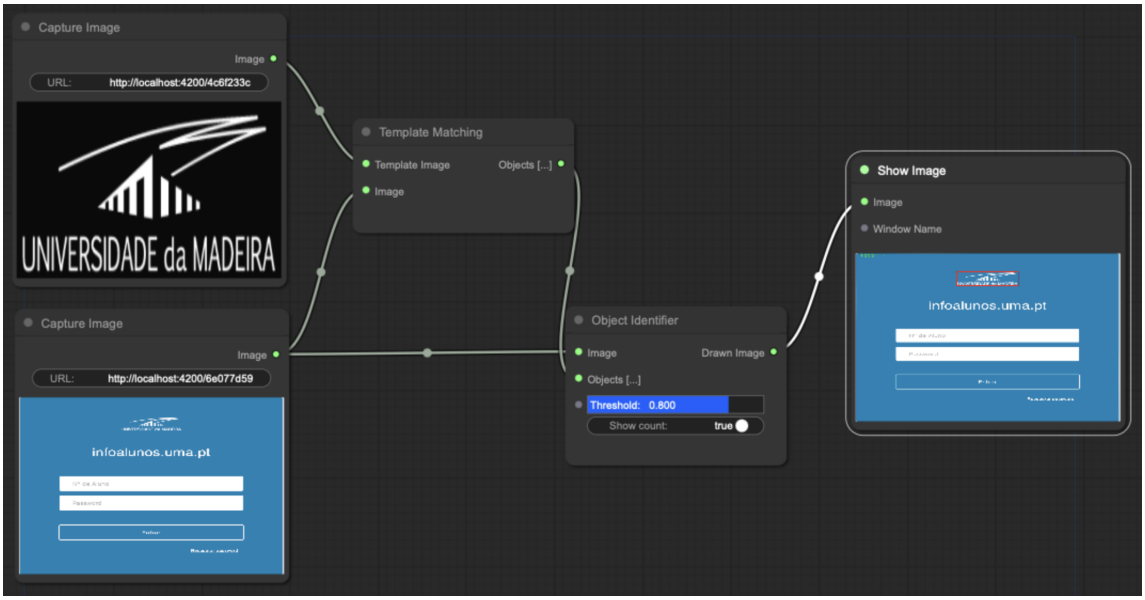

Figura 39. Resultado da tarefa 3.

5.4 Avaliação Quantitativa

Nesta subsecção encontra-se descrita a avaliação quantitativa, fundamentada pelos resultados do questionário *SUS*. Começa-se esta subsecção por apresentar e interpretar as pontuações médias atribuídas às dez afirmações que constituem este questionário. No fim é realizado o cálculo da pontuação obtida por cada participante, concluindo-se a avaliação da plataforma desenvolvida com a pontuação média de usabilidade.

Através do gráfico da Figura 40, que representa a pontuação média atribuída pelos participantes a cada uma das dez afirmações de avaliação, é possível obter-se seguintes conclusões para cada afirmação.

Considerando a primeira afirmação, na qual os participantes devem avaliar o quanto gostariam de utilizar o produto com frequência, consegue-se observar que os participantes concordam com esta afirmação. Um dos participantes manteve-se neutro nesta questão justificando que não é um utilizador assíduo da biblioteca ***OpenCV*** nem um programador dedicado à visão computacional.

Já na segunda questão, que avalia a excessiva complexidade da plataforma, verificou-se uma forte discordância. Com a pontuação mais baixa possível, pode-se concluir que os participantes consideram a plataforma desenvolvida simples e intuitiva de utilizar.

A terceira afirmação destina-se a avaliar a facilidade de utilização da plataforma, verificando-se uma total concordância de todos os participantes. Estes consideram que se trata de uma ferramenta intuitiva de utilizar, com uma baixa curva de aprendizagem. Isto está em concordância com a avaliação qualitativa, na subsecção anterior, em que os participantes conseguiram utilizar a plataforma e concluir as tarefas num curto intervalo de tempo.

Considerando-se a quarta questão, cujo objetivo é avaliar a necessidade de apoio técnico para a utilização da ferramenta desenvolvida, verifica-se um valor reduzido. Isto indica que não existe necessidade de explicações adicionais. Porém, um dos participantes atribuiu uma pontuação elevada

a esta afirmação, por considerar demasiado técnico alguns dos nomes atribuídos a determinados blocos implementados.

Para a quinta questão, observou-se que os participantes concordam que as funcionalidades da plataforma apresentada estavam bem integradas, pois permitiram concluir todas as tarefas propostas. Os participantes realçaram ainda que a funcionalidade responsável por verificar, em tempo real, as transformações aplicadas é uma vantagem valiosa, principalmente na definição e validação dos limites mais adequados a utilizar.

Na afirmação seguinte, os participantes discordam do facto da plataforma apresentar muitas inconsistências. Porém, um dos participantes atribuiu uma pontuação de 4 valores, indicando que concorda que existem inconsistências. Segundo as suas sugestões podemos concluir que estes problemas ocorrem devido aos nomes atribuídos a alguns dos blocos implementados.

Na sétima afirmação, em que é avaliada a rápida aprendizagem de utilização da plataforma, verificou-se que um dos participantes manteve uma postura neutra. Os restantes participantes concordam com esta afirmação, pois ao considerar que esta plataforma se destina aos programadores que utilizam ou pretendem utilizar as funcionalidades da biblioteca ***OpenCV***, esta plataforma *low-code* deverá ser simples de aprender.

Em relação à afirmação que pretende avaliar a complexidade de utilização da ferramenta, verifica-se uma total discordância por todos os participantes. Isto enquandra-se nos comentários obtidos durante a avaliação qualitativa, descrita na subsecção anterior, na qual os participantes realçam a facilidade e agilidade de desenvolvimento através desta plataforma *low-code*.

Os participantes indicaram muita confiança na utilização da plataforma desenvolvida, dado que todos os participantes atribuíram uma pontuação de cinco valores a esta penúltima afirmação.

A última afirmação, que tem por objetivo avaliação a curva de aprendizagem da plataforma, apresenta uma pontuação de dois valores. A maioria dos participantes discordou completamente do facto desta plataforma exigir uma elevada aprendizagem, enquanto um participante concordou com esta afirmação. De referir que a avaliação deste participante tem maior foco na aprendizagem das funcionalidades da biblioteca ***OpenCV*** e de como estas se interligam entre si, e não na utilização da plataforma *low-code*.

Através das pontuações atribuídas por cada um dos participantes a cada afirmação, é possível verificar-se a pontuação de usabilidade. Esta pontuação varia entre 0 e 100 pontos e é obtida conforme os cálculos introduzidos na subsecção 5.2.2.

Através da Figura 41, é possível verificar-se que a pontuação mais elevada foi de 92,5 pontos. Os restantes participantes atribuíram uma pontuação ligeiramente inferior, de 80 pontos. Isto indica que a plataforma desenvolvida é aceitável, possuindo uma pontuação média de 84,167. Esta traduz-se como "Excelente"considerando as classificações por adjetivos [67, 68].

5.5 Limitações

Apesar dos resultados positivos obtidos nas subsecções anteriores, é importante reconhecer as limitações associadas a esta experiência.

A principal limitação desta experiência tem origem na amostra de participantes da mesma. Devido aos critérios de seleção muito específicos, trata-se de uma amostra muito reduzida, contando

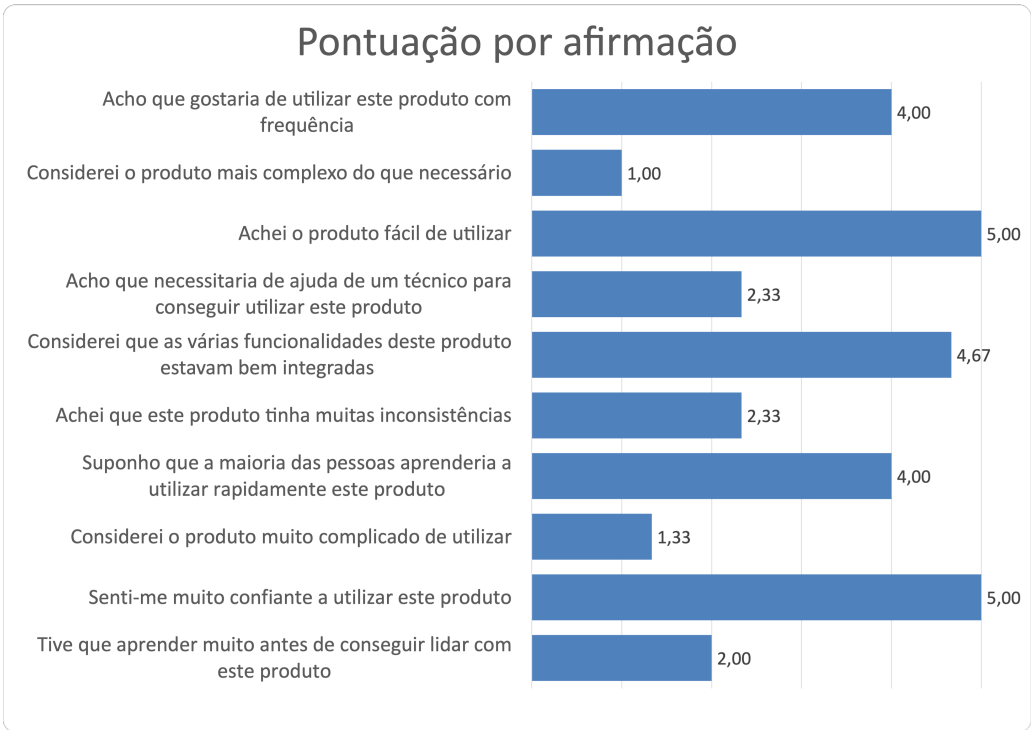

Figura 40. Pontuações por afirmação.

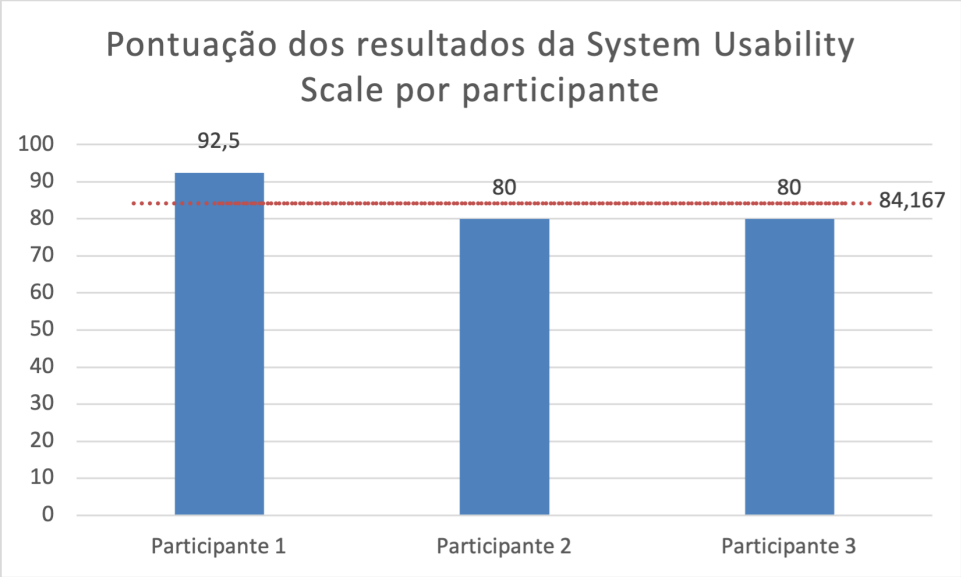

Figura 41. Pontuações por participante.

apenas com três participantes. Porém, estes participantes fazem estritamente parte dos utilizadores alvo desta plataforma *low-code*, pois são utilizadores que conhecem e utilizam, ou já utilizaram, a biblioteca ***OpenCV***.

Outra limitação da experiência deve-se à linguagem de programação escolhida. Embora a linguagem de programação C++ se encontre no núcleo da biblioteca ***OpenCV***, muitos dos programadores que a utilizam recorrem à linguagem de programação *Python*. Esta linguagem é muito utilizada devido a ser atualmente considerada mais simples. É importante referir que um dos par-

ticipantes na experiência comentou que uma determinada funcionalidade da biblioteca ***OpenCV*** pode ser implementada com uma simples invocação em *Python*, enquanto que em C++ é um processo um pouco mais complexo.

5.6 Considerações Finais

A utilização da técnica ***Think Aloud***, que incentiva aos participantes a verbalizar os seus pensamentos no decorrer da execução das tarefas propostas, permitiu identificar aspetos que visam melhorar a usabilidade e a experiência de utilização da plataforma *low-code* implementada.

O *feedback* positivo incide essencialmente na *interface* simples e intuitiva de utilizar. Os participantes consideram que esta plataforma entrega uma facilidade e uma rapidez no desenvolvimento de funcionalidades que envolvem a visão computacional e a utilização da biblioteca ***OpenCV***. A possibilidade de se visualizar as transformações aplicadas em tempo real no navegador, sem a necessidade de compilação nem de instalação de *software* específico, foi um aspeto que agradou ao ser elogiado por todos os participantes.

As sugestões recebidas realçam os aspetos da plataforma que necessitam de algum cuidado ou melhoria, nomeadamente a nomenclatura atribuída a alguns dos blocos implementados, e a possibilidade de se arrastar intuitiva e diretamente o bloco da palete para o respetivo editor. Um dos participantes mencionou que poderia ser interessante fornecer o código gerado num ficheiro, no respetivo formato, pronto a ser importado e executado no ambiente de desenvolvimento do utilizador. Outros participantes indicaram que a incorporação de geração de código para outras linguagens de programação, como *Python*, seria uma funcionalidade de elevado valor de ser implementada.

6 Conclusão

Nesta dissertação foi desenvolvida uma plataforma *low-code* que tem por objetivo a abstração da biblioteca ***OpenCV***. A complexidade associada à utilização da biblioteca de visão computacional ***OpenCV***, devido aos seus inúmeros algoritmos e funcionalidades, apresenta desafios na sua utilização e na sua curva de aprendizagem.

Para combater esta curva de aprendizagem e respetiva complexidade, foi implementada uma plataforma *low-code* com o objetivo de abstrair algumas das funcionalidades desta biblioteca, representando-as através de blocos. Tornou-se possível, através desta abordagem, a utilização conjunta e sequencial de diversas funcionalidades da biblioteca através da interligação entre blocos. Estes blocos, interligados entre si, entregam aos utilizadores o seu código correspondente na linguagem de programação C++, facilitando e agilizando o desenvolvimento de aplicações que envolvem a visão computacional e recorrem à biblioteca ***OpenCV***.

O processo realizado antes da fase de implementação do protótipo da plataforma *low-code* foi fundamental para a definição da solução e da estratégia implementada. Este processo iniciou-se pela fase de investigação, na qual teve de se perceber os conceitos associados à visão computacional e as bibliotecas existentes nesta vertente. O ***OpenCV*** foi a biblioteca de maior foco nesta investigação, devido às limitações das restantes e a dependência de diversas bibliotecas nesta. Atribuiu-se assim um maior nível de detalhe à biblioteca ***OpenCV***, através do estudo da sua estrutura interna, dos seus algoritmos e das suas funcionalidades.

Um outro ponto de investigação de grande relevância centrou-se na compreensão da metodologia de desenvolvimento orientada por modelos, *Model-Driven Engineering*. Através desta metodologia torna-se possível aplicar, através de modelos, todo um processo de abstração de um sistema. As plataformas *low-code* são inspiradas nesta abordagem, em que a abstração é realizada ao nível do conhecimento. Estas plataformas entregam vantagens na flexibilidade, agilidade, tempo e esforço no desenvolvimento de aplicações.

Outra etapa deste processo anterior à implementação do protótipo consistiu na análise de ferramentas de suporte e na elicitação de requisitos. Foram analisadas diversas *frameworks* de suporte à criação de aplicações *web*, verificando-se as suas respetivas vantagens, facilidade de implementação e compatibilidade com as restantes ferramentas. Exploraram-se também plataformas de apoio às interações típicas das ferramentas *low-code*, com o objetivo de facilitar a implementação de elementos gráficos e de formas de interação com os mesmos. Através da investigação realizada sobre a biblioteca alvo de abstração, analisaram-se tutoriais e exemplos, afim de se definir os blocos a implementar e algumas das suas propriedades. Verificou-se também a existência de uma versão *web* da biblioteca ***OpenCV***, que permite a aplicação de algumas funcionalidades diretamente no navegador do utilizador. Desta forma surgiu um novo requisito que consistiu na utilização desta versão *web* como forma de apresentar ao utilizador uma pré-visualização do resultado obtido pela utilização das funcionalidades da biblioteca.

Por fim, após a seleção das plataformas e ferramentas de suporte a utilizar e das funcionalidades a implementar, foi realizada uma análise aprofundada à plataforma *LiteGraph*. Verificou-se toda a sua estrutura, as suas classes e os seus métodos, com o intuito de se desenvolver uma abstração da biblioteca ***OpenCV*** sobre esta plataforma, de uma forma eficiente e estruturada.

Foram definidas três tarefas, além da tarefa tutorial, que foram realizadas por três participantes, com o intuito de se avaliar a utilidade e a usabilidade da plataforma implementada. Estas tarefas foram realizadas através da programação tradicional, na linguagem de programação C++, e através da plataforma *low-code* implementada de forma a se comparar as duas abordagens e de se avaliar as vantagens da plataforma proposta.

Verificou-se, como descrito na subsecção 5, que a plataforma *low-code* apresentada permitiu aos participantes concluírem mais rapidamente as tarefas propostas. A utilização da plataforma gerou um ganho de eficiência, facilidade e agilidade na implementação de funcionalidades que envolvem a área da visão computacional e a utilização da biblioteca ***OpenCV***.

Na avaliação de usabilidade, os participantes atribuíram pontuações positivas, confirmando-se que consideram a plataforma fácil e intuitiva de utilizar. Porém, existem alguns aspetos que carecem de atenção, nomeadamente no que diz respeito à interação e à definição de valores numéricos através dos *sliders*, e nos nomes atribuídos a alguns dos blocos implementados.

A maior limitação no decorrer desta dissertação deveu-se à avaliação da plataforma através dos testes de usabilidade. Apenas se conseguiu três participantes para realizar toda a avaliação, devido aos requisitos necessários para a seleção dos participantes e à disponibilidade dos mesmos. Requisitos estes que envolviam ter conhecimento e feito uso da biblioteca ***OpenCV***, e ainda possuir alguma experiência em programação nomeadamente na linguagem C++. Em relação ao desenvolvimento, surgiram algumas dificuldades que conseguiram ser superadas como, por exemplo, a falta de memória devido à utilização da funcionalidade responsável por captar imagens da câmera do dispositivo. Outras limitações surgiram devido às diferenças existentes nas funcionalidades e argumentos entre a versão *web* e a versão na linguagem C++ da biblioteca ***OpenCV***.

6.1 Trabalho Futuro

A próxima fase de implementação da plataforma tem especial foco na abstração de mais funcionalidades e na agregação de funcionalidades que são utilizadas regularmente em conjunto. Uma abordagem interessante a utilizar para estas funcionalidades agregadas consiste na utilização de subgrafos, ou seja, diversos blocos mais simples contidos num único bloco. Desta forma o utilizador conseguirá analisar e alterar o comportamento das funcionalidades auxiliares que a funcionalidade principal necessita.

Uma outra fase de implementação deverá estar focada na adição ao suporte de novas linguagens de programação. Isto de modo que os utilizadores consigam utilizar a plataforma para gerar código para a linguagem de programação que utiliza, como C++, Python ou Java. A implementação desta funcionalidade é de elevado interesse, uma vez que possibilita o aumento dos potenciais utilizadores da plataforma.

Será importante também definir-se a implementação de sessões de utilizador, nas quais poderão ser armazenadas os modelos implementados, evitando que seja necessário o utilizador instanciar e interligar novamente cada um dos blocos. Outro aspeto relacionado com este consiste na possibilidade de importação do modelo *json*, visto que este pode ser descarregado seria interessante também permitir a sua importação.

Finalmente, a fim de permitir uma constante evolução da plataforma, seria fundamental tornar a plataforma *Open-Source*, com o propósito de dar a oportunidade aos utilizadores de implementar

novos blocos e novas abstrações sobre a biblioteca ***OpenCV***, tornando a plataforma desenvolvida cada vez mais completa.

Referências

[1] D. M. Escrivá, P. Joshi, V. G. Mendonça, and R. Shilkrot, *Building Computer Vision Projects with OpenCV 4 and C++: Implement complex computer vision algorithms and explore deep learning and face detection.* Packt Publishing Ltd, Mar. 2019, google-Books-ID: naOPDwAAQBAJ.

[2] M. Brambilla, J. Cabot, and M. Wimmer, "Model-Driven Software Engineering in Practice," *Synthesis Lectures on Software Engineering*, vol. 1, no. 1, pp. 1–182, Sep. 2012. [Online]. Available: http://www.morganclaypool.com/doi/abs/10.2200/S00441ED1V01Y201208SWE001

[3] M. A. Al Alamin, S. Malakar, G. Uddin, S. Afroz, T. B. Haider, and A. Iqbal, "An Empirical Study of Developer Discussions on Low-Code Software Development Challenges," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, May 2021, pp. 46–57, iSSN: 2574-3864.

[4] "vue-blocks." [Online]. Available: https://ghost.im/vue-blocks/index.html

[5] "Getting Started." [Online]. Available: https://newcat.github.io/baklavajs/#/

[6] J. Agenjo, "litegraph.js," Aug. 2022, original-date: 2013-09-26T17:39:05Z. [Online]. Available: https://github.com/jagenjo/litegraph.js

[7] E. Charters, "The Use of Think-aloud Methods in Qualitative Research An Introduction to Think-aloud Methods," *Brock Education Journal*, vol. 12, no. 2, Jul. 2003, number: 2. [Online]. Available: https://journals.library.brocku.ca/brocked/index.php/home/article/view/38

[8] D. Walters, "Computer vision," in *Encyclopedia of Computer Science.* GBR: John Wiley and Sons Ltd., Jan. 2003, pp. 431–435.

[9] A. F. Villán, *Mastering OpenCV 4 with Python: A practical guide covering topics from image processing, augmented reality to deep learning with OpenCV 4 and Python 3.7.* Packt Publishing Ltd, Mar. 2019, google-Books-ID: w86PDwAAQBAJ.

[10] J. Manigel and W. Leonhard, "Computer control of an autonomous road vehicle by computer vision," in *Control and Instrumentation Proceedings IECON '91: 1991 International Conference on Industrial Electronics*, Oct. 1991, pp. 19–24 vol.1.

[11] L. Matthies, M. Maimone, A. Johnson, Y. Cheng, R. Willson, C. Villalpando, S. Goldberg, A. Huertas, A. Stein, and A. Angelova, "Computer Vision on Mars," *International Journal of Computer Vision*, vol. 75, no. 1, pp. 67–92, Jul. 2007. [Online]. Available: http://link.springer.com/10.1007/s11263-007-0046-z

[12] Y. Ding, L. Hua, and S. Li, "Research on computer vision enhancement in intelligent robot based on machine learning and deep learning," *Neural Computing and Applications*, vol. 34, no. 4, pp. 2623–2635, Feb. 2022. [Online]. Available: https://link.springer.com/10.1007/s00521-021-05898-8

[13] C. E. Vandoni, Ed., *Proceedings / 1996 CERN School of Computing: Egmont aan Zee, The Netherlands, 8 September - 21 September 1996*, ser. CERN. Geneva: CERN, 1996, no. 1996,8, meeting Name: School of Computing.

[14] K. Demaagd, A. Oliver, N. Oostendorp, and K. Scott, *Practical Computer Vision with SimpleCV: The Simple Way to Make Technology See.* "O'Reilly Media, Inc.", Jul. 2012, google-Books-ID: 4st9l6QlkpUC.

[15] "BoofCV." [Online]. Available: https://boofcv.org/index.php?title=Main_Page

[16] "About." [Online]. Available: https://opencv.org/about/

[17] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek, "A brief introduction to OpenCV," in *2012 Proceedings of the 35th International Convention MIPRO*, May 2012, pp. 1725–1730.

[18] A. Kaehler and G. Bradski, *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*, 2016.

[19] G. Stockman and L. G. Shapiro, *Computer Vision*, 1st ed. USA: Prentice Hall PTR, 2001.

[20] R. Szeliski, *Computer Vision: Algorithms and Applications.* Springer Science & Business Media, Sep. 2010, google-Books-ID: bXzAlkODwa8C.

[21] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach. (Second edition).* Prentice Hall, Nov. 2011. [Online]. Available: https://hal.inria.fr/hal-01063327

[22] F. Llorens, F. J. Mora, M. Pujol, R. Rizo, and C. Villagrá, "Working with OpenCV and Intel Image Processing Libraries. Processing image Data Tools." p. 6.

[23] R. Laganiere, *OpenCV Computer Vision Application Programming Cookbook*, 2nd ed. Packt Publishing, 2014.

[24] G. B. García, Ed., *Learning Image Processing with OpenCV: exploit the amazing features of OpenCV to create powerful image processing applications through easy-to-follow examples*, 1st ed., ser. Open Source: community experience distilled. Birmingham: Packt Publ, 2015.

[25] "OpenCV: Modules." [Online]. Available: https://docs.opencv.org/3.0.0/modules.html

[26] "Preface | Raspberry Pi Computer Vision Programming - Second Edition." [Online]. Available: https://subscription.packtpub.com/book/data/9781800207219/1/ch01lvl1sec02/understanding-computer-vision

[27] "OpenCV: Introduction to OpenCV.js and Tutorials." [Online]. Available: https://docs.opencv.org/3.4/df/d0a/tutorial_js_intro.html

[28] "OpenCV Reference Manual," p. 417.

[29] "FAST APPROXIMATE NEAREST NEIGHBORS WITH AUTOMATIC ALGORITHM CONFIGURATION:," in *Proceedings of the Fourth International Conference on Computer Vision Theory and Applications.* Lisboa, Portugal: SciTePress - Science and and Technology Publications, 2009, pp. 331–340. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0001787803310340

[30] S. Farsiu, D. Robinson, M. Elad, and P. Milanfar, "Fast and robust super-resolution," in *Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429)*,

vol. 3. Barcelona, Spain: IEEE, 2003, pp. II–291–4. [Online]. Available: http://ieeexplore.ieee.org/document/1246674/

[31] S. Beydeda, M. Book, and V. Gruhn, Eds., *Model-driven software development*. Berlin ; New York: Springer, 2005.

[32] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, Jun. 2013, google-Books-ID: 9ww_D9fAKncC.

[33] A. Deursen, P. Klint, and J. Visser, "Domain-Specific Languages: An Annotated Bibliography," *SIGPLAN Notices*, vol. 35, pp. 26–36, Jan. 2000.

[34] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM Computing Surveys*, vol. 37, no. 4, pp. 316–344, Dec. 2005. [Online]. Available: https://doi.org/10.1145/1118890.1118892

[35] D. S. Wile, "Supporting the DSL Spectrum," *CIT. Journal of Computing and Information Technology*, vol. 9, no. 4, pp. 263–287, Oct. 2004, number: 4. [Online]. Available: http://cit.fer.hr/index.php/CIT/article/view/1462

[36] Y. Luo, P. Liang, C. Wang, M. Shahin, and J. Zhan, "Characteristics and Challenges of Low-Code Development: The Practitioners' Perspective," in *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ser. ESEM '21. New York, NY, USA: Association for Computing Machinery, Oct. 2021, pp. 1–11. [Online]. Available: https://doi.org/10.1145/3475716.3475782

[37] F. Khorram, J.-M. Mottu, and G. Sunyé, "Challenges & opportunities in low-code testing," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS '20. New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 1–10. [Online]. Available: https://doi.org/10.1145/3417990.3420204

[38] A. Sahay, A. Indamutsa, D. Di Ruscio, and A. Pierantonio, *Supporting the understanding and comparison of low-code development platforms*, Aug. 2020.

[39] M. Oltrogge, E. Derr, C. Stransky, Y. Acar, S. Fahl, C. Rossow, G. Pellegrino, S. Bugiel, and M. Backes, "The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators," in *2018 IEEE Symposium on Security and Privacy (SP)*, May 2018, pp. 634–647, iSSN: 2375-1207.

[40] C. Baumgarten, A. Simeon, and M. C. Wilhelm, "Citizen Developers Driving the Digital Campus," p. 12.

[41] D. Thacker, V. Berardi, V. Kaur, and G. Blundell, "Business Students as Citizen Developers: Assessing Technological Self-Conception and Readiness," *Information Systems Education Journal*, vol. 19, pp. 15–30, Oct. 2021.

[42] K. Arai, *Proceedings of the Future Technologies Conference (FTC) 2021, Volume 3*. Springer Nature, 2021, google-Books-ID: QzZKEAAAQBAJ.

[43] "OutSystems," Jun. 2022, page Version ID: 1094734089. [Online]. Available: https://en.wikipedia.org/w/index.php?title=OutSystems&oldid=1094734089

[44] A. Savidis and C. Savaki, "Complete Block-Level Visual Debugger for Blockly," in *Human Systems Engineering and Design II*, ser. Advances in Intelligent Systems and Computing, T. Ahram, W. Karwowski, S. Pickl, and R. Taiar, Eds. Cham: Springer International Publishing, 2020, pp. 286–292.

[45] E. Pasternak, R. Fenichel, and A. N. Marshall, "Tips for creating a block language with blockly," in *2017 IEEE Blocks and Beyond Workshop (B&B)*, Oct. 2017, pp. 21–24.

[46] "Blockly," Aug. 2022, original-date: 2013-10-25T21:13:33Z. [Online]. Available: https://github.com/google/blockly

[47] "Blockly." [Online]. Available: https://developers.google.com/blockly?hl=pt

[48] "GoJS Introduction – Northwoods Software." [Online]. Available: https://gojs.net/latest/intro/

[49] "GitHub - NorthwoodsSoftware/GoJS: JavaScript diagramming library for interactive flowcharts, org charts, design tools, planning tools, visual languages." [Online]. Available: https://github.com/NorthwoodsSoftware/GoJS

[50] "interact.js - JavaScript drag and drop, resizing and multi-touch gestures for modern browsers." [Online]. Available: https://interactjs.io/docs/

[51] taye, "taye/interact.js," Aug. 2022, original-date: 2012-10-21T10:05:19Z. [Online]. Available: https://github.com/taye/interact.js

[52] A. Herz, "Draw2D," Aug. 2022, original-date: 2018-06-29T13:36:45Z. [Online]. Available: https://github.com/freegroup/draw2d

[53] "Draw2D touch." [Online]. Available: http://www.draw2d.org/draw2d/home.html

[54] A. W. S. Labs, "Diagram Maker · A library to display an interactive editor for any graph-like data." Aug. 2020. [Online]. Available: https://awslabs.github.io/diagram-maker/

[55] "Similar active projects · Issue #29 · ghostiam/vue-blocks." [Online]. Available: https://github.com/ghostiam/vue-blocks/issues/29

[56] "About the platform - Total.js Platform." [Online]. Available: https://www.totaljs.com/platform/

[57] "totaljs/framework: Node.js framework." [Online]. Available: https://github.com/totaljs/framework

[58] "Zoom causes nodes to Blur · Issue #178 · newcat/baklavajs." [Online]. Available: https://github.com/newcat/baklavajs/issues/178

[59] A. Pajankar, *Raspberry Pi Computer Vision Programming*. Packt Publishing, 2015.

[60] "Vue 2.7 is Now in Beta | The Vue Point." [Online]. Available: https://blog.vuejs.org/posts/vue-2-7-beta.html

[61] "Introduction | Vue.js." [Online]. Available: https://vuejs.org/guide/introduction.html

[62] "Angular - What is Angular?" [Online]. Available: https://angular.io/guide/what-is-angular

[63] "Difference between TypeScript and JavaScript," Jun. 2018, section: Web Technologies. [Online]. Available: https://www.geeksforgeeks.org/difference-between-typescript-and-javascript/

[64] "React vs Vue vs Angular: qual escolher? - Geek Blog |." [Online]. Available: https://blog.geekhunter.com.br/react-vs-vue-vs-angular-qual-escolher/

[65] "Getting Started – React." [Online]. Available: https://reactjs.org/docs/getting-started.html

[66] Y. Chebotaev, "json-decycle," Mar. 2023, original-date: 2015-10-10T06:19:36Z. [Online]. Available: https://github.com/YChebotaev/json-decycle

[67] A. I. Martins, A. F. Rosa, A. Queirós, A. Silva, and N. P. Rocha, "European Portuguese Validation of the System Usability Scale (SUS)," *Procedia Computer Science*, vol. 67, pp. 293–300, Jan. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050915031191

[68] A. Bangor, "Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale," vol. 4, no. 3, 2009.

A Guia de Avaliação do OpenCV Studio

Encontram-se descritas neste guia 3 tarefas que os utilizadores devem realizar de modo a se conseguir avaliar o impacto que a abstração aplicada à biblioteca ***OpenCV*** tem no desenvolvimento de sistemas por utilizadores com e sem experiência em programação.

A.1 Tarefas

A.1.1 Tarefa 0 - Tutorial: Troca dos 3 canais de cor (protótipo low-code)

– Na plataforma, abrir a lista de blocos, selecionar o bloco **Capture Image** e clicar posteriormente no espaço mais escuro ao lado, chamamos este espaço de editor.

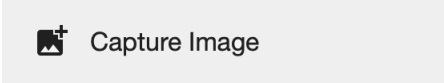

Figura 42. Bloco Capture Image

– Neste bloco arrastamos a imagem fornecida *tarefa0.jpg* para este respetivo bloco.

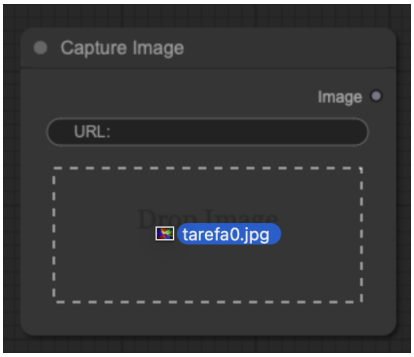

Figura 43. Carregar imagem no bloco Capture Image

– Colocar no editor um bloco **Split**.

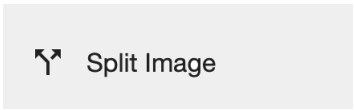

Figura 44. Bloco Split Image

– Interligar a saída *Image* do primeiro bloco, com a entrada *Image* deste novo bloco.

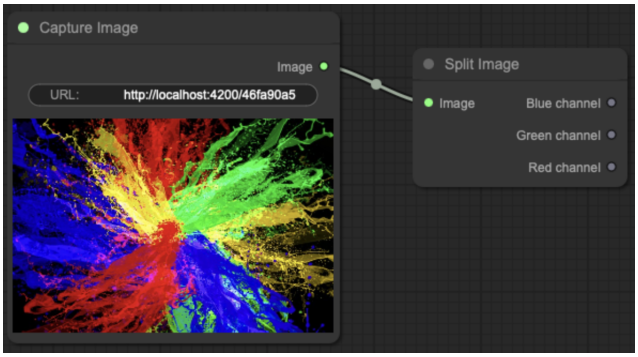

Figura 45. Associação para separação de canais

– Criar 3 novos bloco do tipo *Show Image*.

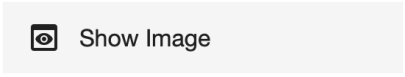

Figura 46. Bloco Show Image

– Interligar cada uma das saídas do bloco *Split Image* aos 3 novos blocos no editor.

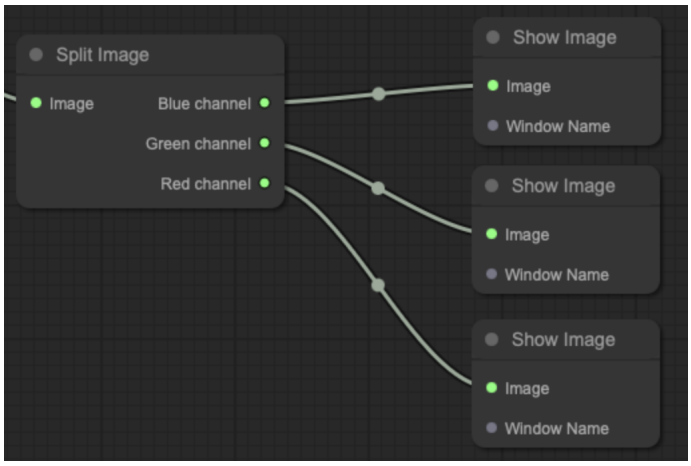

Figura 47. Janela para cada canal de cor

– Criar 3 novos blocos do tipo *String Value*.

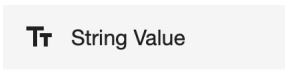

Figura 48. Bloco String Value

– Interligar a saída de cada um destes blocos à entrada dos blocos **Show Image**, preenchendo o respetivo nome do canal de cor (utilizar ctrl+c e ctrl+v para copiar e colar o mesmo bloco).

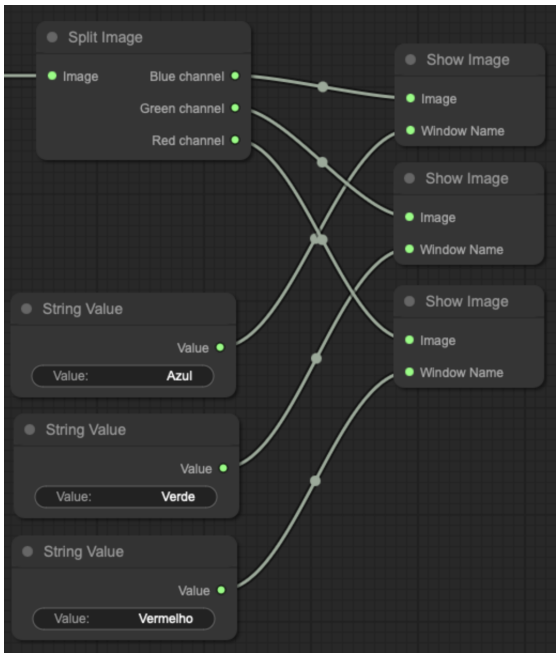

Figura 49. Associar título de janela a cada cor

– Para se ganhar um pouco mais de espaço, selecionamos estes blocos **String Value** e escondemos o conteúdo carregando no respetivo *icon* da barra de ferramentas. (Utilizar o *shift* para múltiplas seleções).

Figura 50. Botão de colapsar blocos

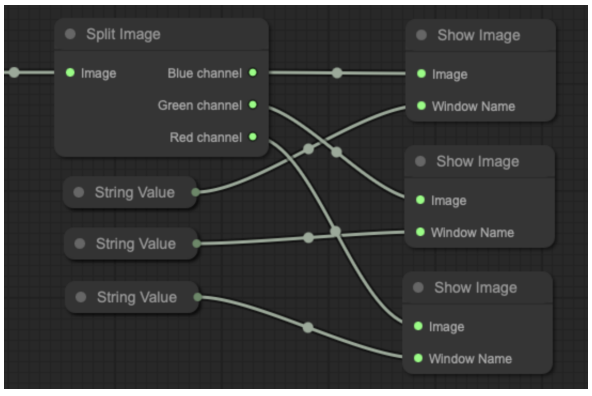

Figura 51. Colapsar blocos

– Vamos colocar um novo bloco **Merge Image**, trocando os canais de cor.

- Na cor azul (B): utilizar o canal vermelho;
- Na cor verde (G): utilizar o canal azul;
- Na cor vermelha (R): utilizar o canal verde.

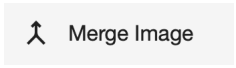

Figura 52. Bloco Merge Image

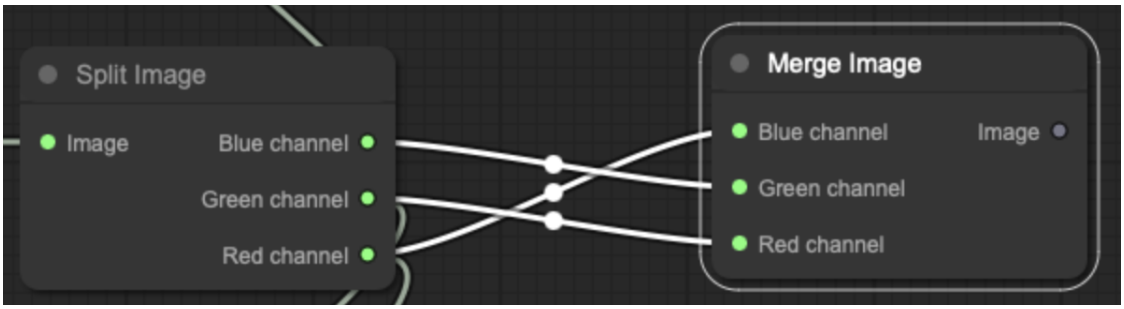

Figura 53. Canais de cor trocados

– Colocamos um novo bloco do tipo **Show Image** e um do tipo **String Value**, onde deve constar a imagem final com os canais trocados.

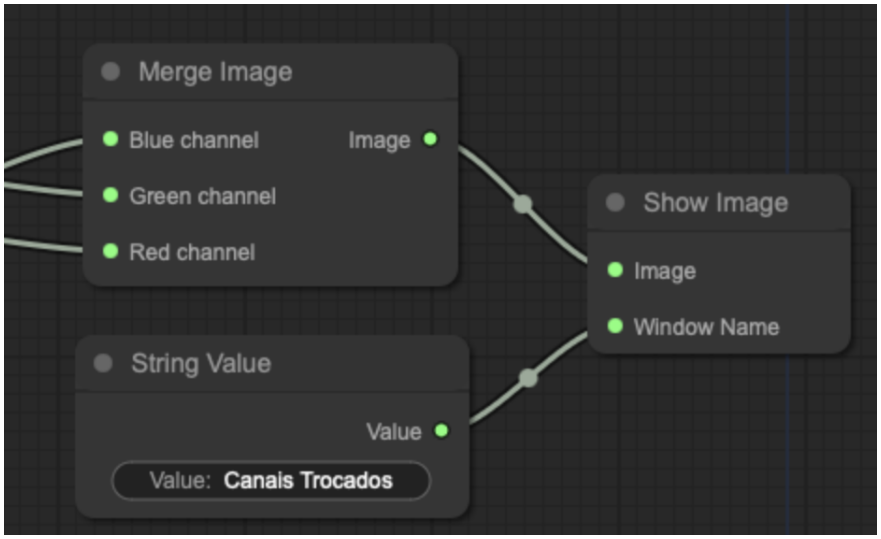

Figura 54. Janela com os canais trocados

– O resultado de todos os blocos interligados deverá ser semelhante ao seguinte:

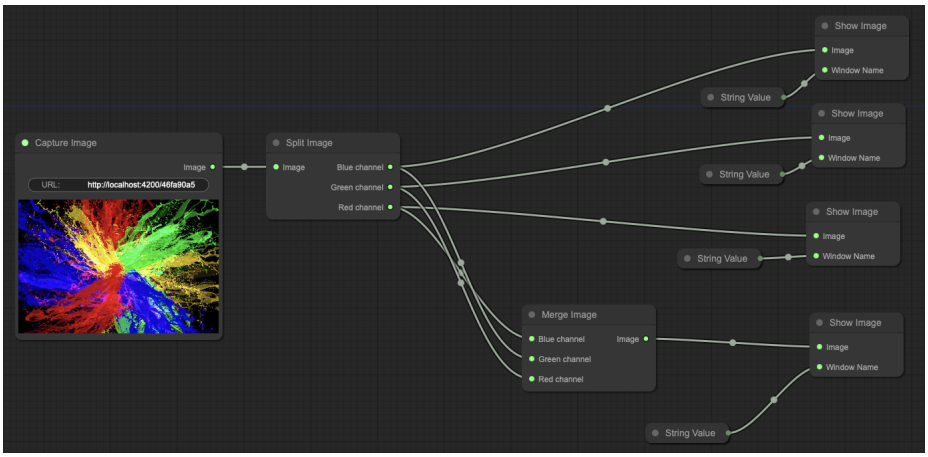

Figura 55. Todos os blocos desta tarefa e as suas associações

– Com todos os blocos interligados, pode-se proceder à execução dos mesmos, carregando no *icon* de *play* na barra de ferramentas. Este depois pode ser parado através do *icon* de *stop*.

Figura 56. Botão play e stop

– Instantaneamente serão visíveis os resultados das transformações, verificando-se nos blocos **Show Image** os 3 canais de cor individualmente e como resultado final as cores trocadas como mencionado anteriormente.

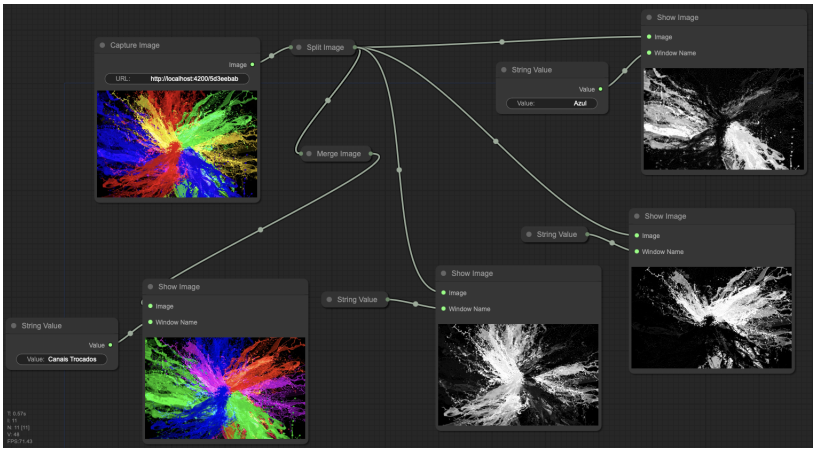

Figura 57. Resultado após execução

Nota: Naturalmente as imagens correspondentes a cada canal de cor encontram-se em escala de cinzentos, pois se trata de apenas 1 canal de cor. Considerando a cor azul, pode-se verificar que onde está presente esta cor na imagem original, o seu respetivo canal entrega um tom mais claro.

Figura 58. Botão de geração de modelo e geração de código

– O seu modelo, em formato *JSON*, pode ser gerado e transferido através do respetivo *icon* mais à esquerda na Figura 58. Este modelo pode ser utilizado para se carregar novamente estes blocos e associações.

– A respetiva geração de código, para a linguagem de programação C++, é realizada através do *icon* ao lado. Surgindo um *popup* contendo o código respetivo aos blocos e às associações criadas.

Figura 59. Popup contendo o código gerado

Nota: Este código está pronto a utilizar, devendo-se apenas substituir o caminho da imagem utilizada, pelo caminho da imagem relativo ao respetivo projeto em c++.

– Copiando e colocando este código no respetivo ambiente de programação em C++, com o respetivo caminho das imagens alterado, obtém-se 4 janelas. Das quais 3 janelas contendo um canal de cor e outra contendo o resultado final. Como título das janelas, encontram-se os valores definidos nos blocos **String Value**.

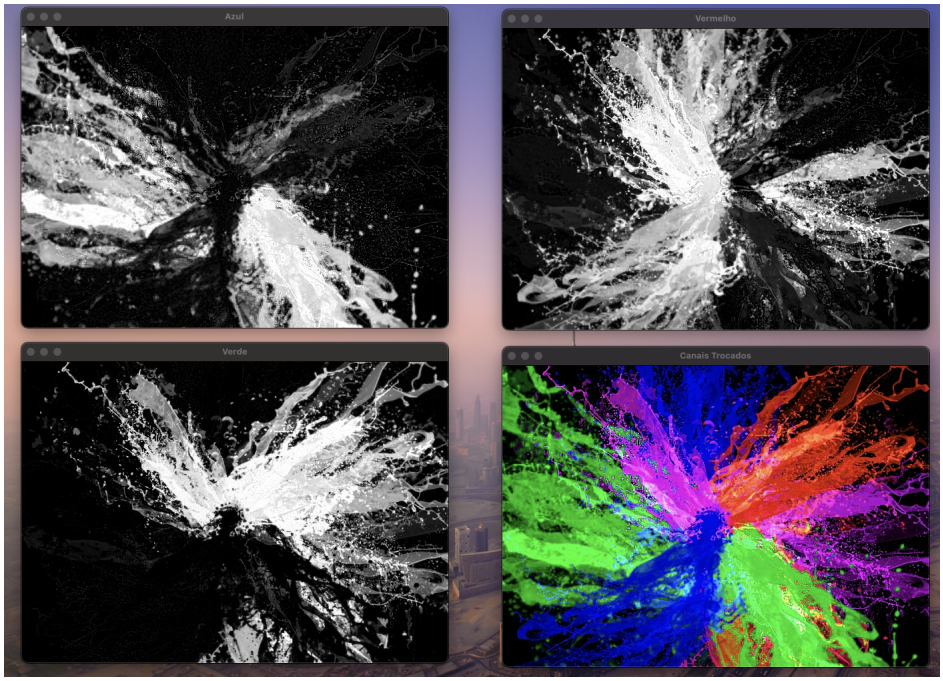

Figura 60. Resultado da execução do código

A.1.2 Tarefa 0 - Tutorial: Troca dos 3 canais de cor (C++)

O objetivo desta tarefa consiste na troca das cores da imagem *tarefa0.jpg*. De forma resumida, pretende-se que a cor azul se transforme em vermelho, a cor verde em azul e a cor vermelha em verde. Por ser a tarefa zero encontra-se descrito detalhadamente os passos a seguir quer através da linguagem de programação C++, quer utilizando a plataforma desenvolvida.

```
1  int main() {
2      Mat sourceImage;
3      sourceImage = imread( "tarefa0.jpg", 1);
4
5      vector<Mat> channels(3);
6      Mat& blueChannel = channels.at(0);
7      Mat& greenChannel = channels.at(1);
8      Mat& redChannel = channels.at(2);
9
10     split(sourceImage, channels);
11
12     namedWindow("Azul", WINDOW_AUTOSIZE);
13     namedWindow("Verde", WINDOW_AUTOSIZE);
14     namedWindow("Vermelho", WINDOW_AUTOSIZE);
15
16     imshow("Azul", blueChannel);
17     imshow("Verde", greenChannel );
18     imshow("Vermelho", redChannel);
19
20     vector<Mat> channelsSwitched = {redChannel, blueChannel, greenChannel};
21
22     Mat merged;
23     merge(channelsSwitched, merged);
24
25     namedWindow( "Canais Trocados", WINDOW_AUTOSIZE );
26     imshow( "Canais Trocados", merged );
27
28     waitKey(0);
29 }
```

Figura 61. Código completo da tarefa 0 - Tutorial

- Considera-se o ambiente de programação já preparado, contendo os *header files* e as imagens necessárias na raiz do projeto. Começa-se por definir, na função principal *main*, uma matriz da biblioteca ***OpenCV*** e a respetiva leitura da imagem *tarefa0.jpg* (linhas 2 e 3 respetivamente).
- Após a leitura da imagem, pretende-se separar os canais de cor da mesma. Para tal é necessário definir-se um vetor de 3 matrizes, que deverá conter cada um dos 3 canais de cor (linhas de 5 a 8).

  Nota: A ordem tradicional dos canais de cores nesta biblioteca é invertido, isto é, BGR (azul, verde e vermelho)
- Aplica-se então, na linha 10, a separação dos canais de cor utilizando a funcionalidade **Split**.

– O código entre as linhas 12 e 18, são responsáveis pela criação de 3 novas janelas, nas quais encontram-se representados os 3 canais de cor de forma individual.

– Procedendo-se à troca dos canais, é definido na linha 20, um novo vetor de matrizes considerando a seguinte troca de cores:

  - Na cor azul (B): utilizar o canal vermelho;
  - Na cor verde (G): utilizar o canal azul;
  - Na cor vermelha (R): utilizar o canal verde.

  Nota: A ordem tradicional encontra-se assim alterada de BGR para RBG, isto é, vermelho, azul e verde.

– Nas linhas 22 e 23, recorre-se à funcionalidade Merge cujo objetivo é unir os 3 canais de cor na ordem definida no vetor anterior.

– Para se concluir esta tarefa, é necessário definir uma nova janela contendo a imagem resultante com os canais de cor trocados. Para tal procede-se à criação de uma nova janela onde fica representado esta nova matriz, linhas 25 e 26.

– De modo a se evitar o fim da execução, deve ser adicionado um *wait* no final da função principal, linha 28.

– Por fim, a execução deste código, apresenta como resultado 4 janelas. Destas, 3 representam cada canal de cor de forma individual. A janela restante contem a imagem final com as cores trocadas.

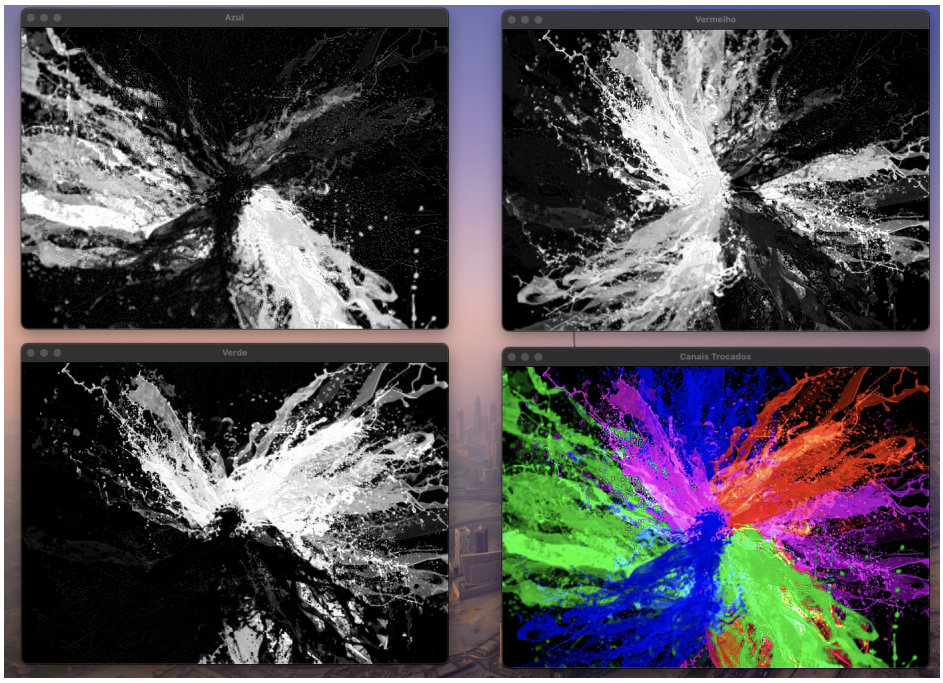

Figura 62. Resultado da execução do código

A.1.3 Tarefa 1: Sobreposição de 2 imagens disponibilizadas

Para esta tarefa pretende-se sobrepor a imagem de um tigre numa imagem de fundo, para que o tigre ganhe tonalidades presentes nesta última. É pretendido que seja uma sobreposição equilibrada, onde cada imagem contribuí 50% para a imagem final.

Seguir os seguintes passos para concretizar o pretendido:

- Carregar ambas as imagens disponibilizadas *tigre.jpg* e *background.jpg*.
- Aplicar sobreposição de imagens
- Mostrar imagem final.
- Manipular a quantidade de sobreposição das imagens.
- Definir um valor *alpha* de 0.5.
- Gerar o modelo em formato json.
- Gerar o código em C++.
- Aplicar no ambiente C++ disponibilizado e verificar o resultado final.

Nota: Para ajustar a quantidade de sobreposição, deve-se alterar o valor de *alpha* entre 0 e 1, onde 0 corresponde à primeira imagem e 1 corresponde à segunda imagem. Um valor de **0.5** traduz-se numa sobreposição equilibrada de ambas as imagens.

Critério de sucesso: Aplicar a sobreposição de 2 imagens e verificar o mesmo resultado no protótipo e no ambiente de programação em C++.

A.1.4 Tarefa 2: Identificar, marcar e contar objetos de cor verde em tempo real a partir da câmera

Esta é uma tarefa mais robusta onde pretende-se desenhar retângulos nos objetos verdes que serão colocados em frente à câmera do dispositivo.

Esta é uma tarefa que precisa:

- utilizar a captura da câmera em tempo real;
- aplicar um filtro de cores para considerar apenas os tons verdes;
- identificar a existência e a posição de elementos dentro deste filtro de cores;
- desenhar na imagem resultante um retângulo nesses elementos identificados;

Os passos para se concluir esta tarefa consistem essencialmente em:

- Iniciar a captura de câmera em tempo real.
- Filtrar as cores pela cor verde.
  - Ajuda:

    - H = [30, 85] (ou normalizado [0.12, 0.33]), que representam a cor verde
    - S = [50, 255] (ou normalizado [0.20, 1.0]), representa a saturação da cor
    - V = [50, 255] (ou normalizado [0.20, 1.0]), representa o brilho da cor
- Detetar os contornos dos objetos dessa cor.
- Identificar os respetivos objetos na **imagem original**.
- Verificar resultado final através do bloco de visualização.
- Aplicar limitação aos objetos identificados.
- Gerar o código em C++.
- Aplicar no ambiente C++ disponibilizado e comparar as saídas com o obtido na ferramenta.

Nota: A limitação aos objetos identificados, na linguagem de programação C++, é realizada na lista resultante da funcionalidade *findContours*. Devendo-se iterar por esta lista de forma a se filtrar os elementos de acordo com um determinado critério. Tipicamente o critério utilizado é a área, obtida através da função ***contourArea***.

Critério de sucesso: Identificar, em tempo real, objetos verdes colocados em frente à câmera.

A.1.5 Tarefa 3: Encontrar, e identificar o logótipo da UMa

Esta tarefa consiste em utilizar-se o logótipo da Universidade da Madeira de forma a que este seja identificado nas imagens fornecidas. Deverá ser identificado o logótipo da UMa:

- numa captura de ecrã realizada ao website da univeridade
- numa captura de ecrã realizada à página de *login* do infoAlunos
- num Cartaz de um curso de inglês da universidade

Os passos a seguir para a conclusão desta tarefa centram-se nos seguintes:

- Carregar a imagem do logótipo da universidade da Madeira.
- Carregar a imagem onde se pretende encontrar o logótipo: (carregar uma imagem à vez)
    - Imagem do website
    - Imagem do infoAlunos
    - Cartaz
- Aplicar o mesmo espaço de cor a ambas as imagens (escala de cinzentos).
- Procurar correspondências do logótipo as imagens.
- Identificar e marcar as correspondências na imagem original.
- Aplicar uma limitação de 65% (ou 0.65) às correspondências encontradas.

- Verificar resultado final no respetivo bloco.
- Trocar para a imagem seguinte.
- Gerar o código em C++.
- Aplicar no ambiente C++ disponibilizado e comparar as saídas com o obtido na ferramenta.

Nota: A limitação aos objetos identificados, na linguagem de programação C++, é realizada recorrendo à funcionalidade ***minMaxLoc***. Esta funcionalidade permite encontrar o valor mínimo e máximo atribuído às correspondências entre o logótipo e a imagem. Permite ainda identificar o respetivo ponto mínimo e máximo (utilizado para o desenho do retângulo).

Critério de sucesso: Identificar o logótipo da universidade da Madeira nas 3 imagens fornecidas.

A.2 Conclusão

Após a conclusão das tarefas delineadas neste guia, pretende-se recolher informações sobre a interação dos utilizadores com a plataforma *low-code* desenvolvida. Para tal irão ser analisadas as gravações para se contabilizar a quantidade de erros e o tempo dedicado a cada uma das tarefas. Através de um pequeno questionário, que visa avaliar a usabilidade da plataforma desenvolvida, pretende-se identificar potenciais áreas de melhoria e compreender a experiência geral do utilizador.

A.3 Descrição Blocos Disponíveis

	**Capture Image**
Este trata-se de um bloco que permite carregar uma imagem a ser processada pelos restantes blocos. A imagem pode ser carregada através de um *link* ou realizando *drag-and-drop* da imagem no bloco.	

	**Camera Capture**
Como o próprio nome indica, este bloco utiliza os *frames* de uma das câmeras disponíveis no dispositivo.	

	**Show Image**
Este é o bloco final de uma tarefa, onde é representado o resultado visual final. Este bloco apenas recebe uma imagem e um título.	

	**Slider Value**
O slider consiste simplesmente na definição de um valor entre 0 e 1, utilizado tipicamente para definição de *thresholds*.	

	**Knob Value**
Possui a mesma responsabilidade que o bloco **Slider Value**, apresentando uma representação visual diferente.	

	**Double Slider**
Este **Double Slider** é semelhante ao **Slider Value**. Porém, o seu resultado consiste em 2 valores, permitindo definir um valor mínimo e um valor máximo na mesma representação.	

	**String Value**
Este bloco consiste simplesmente na definição de uma *string*, isto é, simples texto.	

	**Gray Scale**
Como o nome indica, este bloco transforma a imagem de entrada em escala de cinzentos.	

	**HSV Filter**
Este bloco divide a representação de cores tradicionais, como o RGB, de uma forma mais intuitiva. Neste caso concreto em: – **Hue**: representa simplesmente a cor, onde 0^{o} é vermelho, 120^{o} verde e 240^{o} azul. – **Saturation**: representa a saturação, é o que mede a intensidade da cor, onde 0 é uma tonalidade cinza, desprovida de cor. Consoante este valor aumenta, a cor torna-se mais viva. – **Value**: representa a luminosidade ou brilho da cor. Um valor baixo determina que se trata de uma cor escura, enquanto um valor elevado representa uma cor clara e brilhante. O valor mínimo (zero) e o valor máximo representa as cores preto e branco respetivamente. De forma resumida, este bloco permite a aplicação de filtros de cor às imagens.	

	**Linear Blender**
Este trata-se de um bloco que permite realizar sobreposição de 2 imagens.	

	**Split Image**
Este bloco recebe uma imagem a cores e divide os seus canais de cor em azul, verde e vermelho.	

	**Merge Image**
Este bloco realiza a transformação oposto ao **Split Image**, recebendo os 3 canais de cores e unindo-os numa única imagem a cores.	

	**Detect Contours**
Neste bloco são detetados os contornos dos objetos presentes na imagem de entrada. Como resultado deste bloco são fornecidas as coordenadas dos contornos detetados.	

	**Template Matching**
Neste bloco entram 2 imagens, uma das quais consiste no *template*, isto é, no objeto que se pretende procurar na outra imagem recebida.	

	**Object Identifier**
Este bloco recebe uma imagem e uma lista de objetos. Este bloco desenha um quadrado vermelho à volta de cada objeto na lista.	

Tabela 1: Descrição dos blocos desenvolvidos

A.4 Descrição de funcionalidades em C++

Abaixo encontram-se descritas as funcionalidades consideradas da biblioteca ***OpenCV***. Deve-se considerar a utilização do *namespace* ***cv***.

Ao utilizar a biblioteca ***OpenCV*** na linguagem C++, é essencial considerar a utilização do *namespace* ***cv***, que encapsula todas as classes, métodos e estruturas fornecidas pela biblioteca. O uso deste *namespace* facilita o acesso e a utilização dessas funcionalidades, evitando conflitos de nomes e garantindo uma implementação coerente e organizada.

Encontram-se assim descritas as funcionalidades do ***OpenCV*** fundamentais às tarefas definidas, considerando a utilização do respetivo *namespace*, definindo a utilização da biblioteca de maneira eficiente e consistente.

A.4.1 Leitura de imagem

Esta funcionalidade permite carregar uma imagem de um determinado ficheiro numa matriz da biblioteca ***OpenCV***.

```
1  Mat imagem = imread("path/imagem.jpg");
```

Figura 63. Leitura de imagem

A.4.2 Apresentação de imagem

Esta é a funcionalidade responsável por mostrar ao utilizador a matriz processada pela biblioteca ***OpenCV***, representando uma imagem numa determinada janela (que deve ser definida com recurso à sua respetiva funcionalidade).

```
1  imshow("Título da janela", imagem);
```

Figura 64. Apresentação de imagem

A.4.3 Captura de câmera

Esta funcionalidade permite carregar um vídeo ou utilizar a captura em tempo real de uma câmera presente no dispositivo.

```
1  int main() {
2      VideoCapture captura(0);
3
4      while (true) {
5          captura.open(0);
6
7          if (captura.isOpened()) {
8              cout << "Camera opened successfully." << endl;
9              break;
10         } else {
11             cerr << "Error: Unable to open the camera. Retrying..." << endl;
12             waitKey(1000);
13         }
14     }
15
16     Mat frame;
17
18     while (true) {
19         captura >> frame;
20
21         if (frame.empty()) {
22             cerr << "Error: Failed to capture frame." << endl;
23             break;
24         }
25
26         process_frame(frame);
27
28         if (waitKey(1) == 27) {
29             break;
30         }
31     }
32
33     captura.release();
34     destroyAllWindows();
35     return 0;
36 }
37
38 void process_frame(Mat frame){
39     //processamento a cada frame capturado ...
40 }
```

Figura 65. Funcionalidade geral de captura de câmera

Para se iniciar a captura é necessário definir-se uma instância da classe VideoCapture, linha 2.

É importante realçar que, antes de ser iniciado o processo de captura de *frames* é necessário realizar a verificação de permissão de acesso à câmera do dispositivo e se esta encontra-se devidamente pronta para a captura. Para tal é comum utilizar-se um ciclo, para serem realizadas verificações até obter-se de acesso (linhas 4 a 14).

Uma vez obtido acesso à captura da câmera, é iniciado um novo ciclo que é responsável pela captura recorrente de *frames* da câmera (linhas 18 a 31). É neste ciclo que se deve ser realizado o

processamento a cada *frame* capturado, para tal encontra-se definida a chamada à função responsável pelo processamento e a sua respetiva assinatura, nas linhas 26 e 38.

O fim deste ciclo de captura de vídeo em tempo real é realizada ao ser premida a tecla *Escape (ESC)*. Condição definida nas linhas 28 a 30.

Por fim, é necessário libertar a utilização da câmera após o utilizador decidir por término à tarefa. Para tal recorre-se à funcionalidade *release*, linha 33.

A.4.4 Criação de janela

Nesta funcionalidade é definida a janela onde deverá ser carregada uma imagem através da funcionalidade anterior. Especial atenção no título da janela a utilizar que deve corresponder ao título definido na funcionalidade de apresentação de imagem.

```
1  namedWindow("Título da janela", WINDOW_NORMAL);
```

Figura 66. Criação de janela

Esta funcionalidade tem como segundo argumento a definição das propriedades da janela a ser criada. Estas podem ser:

- **WINDOW_NORMAL**: Permite o redimensionamento da janela.
- **WINDOW_AUTOSIZE**: Define a janela com um tamanho estático, definido pelas propriedades da imagem apresentada.
- **WINDOW_OPENGL**: Utiliza a tecnologia *OpenGL* para a apresentação da janela.
- **WINDOW_FULLSCREEN**: Define a janela em modo de ecrã completo.
- **WINDOW_FREERATIO**: permite o redimensionamento da janela de forma livre, alterando a relação largura e altura.
- **WINDOW_KEEPRATIO**: Permite o redimensionamento da janela, mas garantindo a proporção entre a largura e altura original.

A.4.5 Divisão dos canais de cor

Para se aplicar a separação dos 3 canais de cor (azul, verde e vermelho) é necessário definir-se um vetor de matrizes da biblioteca ***OpenCV***. É este o vetor a ser utilizado como segundo argumento da funcionalidade *split*. O primeiro argumento trata-se apenas da matriz original com todos os canais de cor.

```
1  vector<Mat> canais;
2  split(imagem, canais);
```

Figura 67. Divisão dos canais de cor

Para se aplicar transformações a cada canal de cor de forma individual, deve-se aceder ao endereço dos elementos do vetor, por exemplo. Ao canal azul corresponde o primeiro elemento que se encontra na posição 0. Porém, para se utilizar devidamente este canal de cor, deve-se utilizar o seu respetivo endereço de memória, utilizando-se o símbolo *&* para esse efeito.

```
1  Mat& canal_azul = canais.at(0);
```

Figura 68. Acesso individual ao canal de cor

A.4.6 União dos canais de cor

A união dos canais de cor é realizada através da função *merge*. Esta funcionalidade deve receber como primeiro argumento os 3 canais de cor respeitando a ordem BGR (Azul, Verde e Vermelho).

```
1  vector<Mat> canais;
2  Mat imagem_merged;
3  merge(canais, imagem_merged);
```

Figura 69. União dos canais de cor

A.4.7 Transformação do espaço de cor

Ao utilizar esta funcionalidade torna-se possível a conversão do espaço de cor de uma imagem. O primeiro argumento corresponde à imagem de origem, o segundo argumento à imagem resultante desta conversão do espaço de cor, e o terceiro argumento define o tipo de transformação aplicar. Introduz-se uma lista com os 3 tipos de conversão mais comuns para este último argumento:

```
1  Mat imagem_cinza;
2  cvtColor(imagem_origem, imagem_cinza, COLOR_BGR2GRAY);
```

Figura 70. Transformação do espaço de cor em escala de cinzentos

- **COLOR_BGR2GRAY**: Converte o espaço de cor azul, verde e vermelho para escala de cinzentos.
- **COLOR_BGR2RGB**: Converte o espaço de cor azul, verde e vermelho para o espaço típico vermelho, verde e azul.
- **COLOR_BGR2HSV**: Converte o espaço de cor azul, verde e vermelho para *Hue*, *Saturation*, *Value*.
  - **Hue**: representa simplesmente a cor, onde 0^{o} é vermelho, 120^{o} verde e 240^{o} azul.

- **Saturation**: representa a saturação, é o que mede a intensidade da cor, onde 0 é uma tonalidade cinza, desprovida de cor. Consoante este valor aumenta, a cor torna-se mais viva.
- **Value**: representa a luminosidade ou brilho da cor. Um valor baixo determina que se trata de uma cor escura, enquanto um valor elevado representa uma cor clara e brilhante. O valor mínimo (zero) e o valor máximo representa as cores preto e branco respetivamente.

A.4.8 Criação de uma cor (3 canais)

A classe **Scalar** é a utilizada para representar valores com múltiplos componentes, como o caso das cores que, tipicamente, possuem 3 canais correspondentes a cada cor (azul, verde e vermelho) e outro valor opcional que representa a opacidade de cor. A seguinte linha de código define a cor vermelha:

```
1  Scalar cor_vermelha(0, 0, 255); // BGR
```

Figura 71. Criação da cor vermelha

A.4.9 Filtro por cor

A funcionalidade *inRange* é a que permite filtrar a cor num determinado intervalo de valores. É frequentemente utilizada para filtrar píxeis com base em critérios de cor. A seguinte linha apresenta os argumentos da funcionalidade, e logo de seguida a sua aplicação considerando a filtragem da cor azul, utilizando-se o espaço de cor HSV na imagem de origem.

```
1  inRange(imagem_origem, limite_inferior_cor, limite_superior_cor, imagem_destino);
```

Figura 72. Filtro por cor

Os argumentos relativos aos limites inferior e superior devem ser definidos com recurso à classe **Scalar**, introduzida acima. A ordem destes valores consiste em H, S, V.

```
1  Mat imagem_destino;
2  Scalar limite_inferior_cor_azul(100, 100, 100);
3  Scalar limite_superior_cor_azul(120, 255, 255);
4  inRange(imagem, cor_limite_inferior, cor_limite_superior, mascara);
```

Figura 73. Filtro pela cor azul

A.4.10 Sobreposição de imagens

A função *addWeighted* é utilizada para realizar uma combinação linear de duas imagens, dando a cada imagem um determinado peso. Esta operação é frequentemente utilizada para a criação de efeitos de transparência ou para sobreposição de duas imagens.

```
1  addWeighted(imagem_origem_1, double alpha, imagem_origem_2, double beta, double gamma, imagem_destino);
```

Figura 74. Combinação de imagens

O argumento *alpha* representa o peso que a primeira imagem possui. Este valor deve ser definido entre 0 e 1. Já o valor de *beta* corresponde ao peso atribuído à segunda imagem, tipicamente corresponde a *1 - valor de alpha*. O valor de *gamma* é tipicamente zero, pois corresponde ao valor adicionado a cada píxel após a combinação linear.

A.4.11 Contours

A função *findContours* é utilizada para encontrar os contornos na imagem. Estes contornos são armazenados num vetor de vetores de pontos.

```
1  void findContours( InputArray image, OutputArrayOfArrays contours,
2                     int mode, int method, Point offset = Point() );
```

Figura 75. Delinear contornos

O argumento *mode* permite definir o modo de recuperação dos contornos. Pode-se definir os seguintes valores:

- **RETR_EXTERNAL**: recuperar apenas os contornos externos;
- **RETR_LIST**: recuperar todos os contornos sem hierarquia;
- **RETR_TREE**: recuperar todos os contornos com uma hierarquia completa;

Já o argumento *method* destina-se à definição do método de aproximação dos contornos, devendo tomar um dos seguintes valores, sendo que o mais comum consiste na aproximação simples:

- **CHAIN_APPROX_NONE**: Não realiza nenhuma aproximação e armazena todos os pontos do contorno;
- **CHAIN_APPROX_SIMPLE**: aproximar os contornos por compressão de segmento (**recomendado**);
- **CHAIN_APPROX_TC89_L1**: algoritmo de aproximação *Tangent-Chordal (Taubin)*;

A.4.12 Área de contornos

Para se calcular a área de um contorno, utiliza-se a função ***countArea***. Estes contornos podem assumir qualquer forma, desde que sejam um conjunto de pontos contínuos. Esta função recebe como

argumento um contorno, definido tipicamente através de um vetor de pontos (*std::vector<cv::Point>*), e, opcionalmente, um valor booleano que define se é utilizada a orientação do contorno no cálculo da área.

```
1  double area = contourArea(InputArray contour, bool oriented = false);
```

Figura 76. Área de um contorno

A.4.13 Template Matching

Esta funcionalidade é utilizada para encontrar a correspondência de um modelo (ou *template*) numa determinada imagem. Este *template* é comparado com diferentes regiões da imagem principal, gerando um mapa de correspondência.

```
1  void matchTemplate( InputArray image, InputArray templ, OutputArray result,
2                      int method, InputArray mask = noArray() );
```

Figura 77. Template Matching

Em *result* ficará armazenado o mapa de correspondências encontrado entre a imagem e o*template*. No argumento *method* deve ser definido o algoritmo de comparação a utilizar como, por exemplo:

- **TM_CCOEFF**: consiste no coeficiente de correlação;
- **TM_CCOEFF_NORMED**: coeficiente de correlação normalizado, entre 0 e 1;
- **TM_CCORR**: correlação cruzada;
- **TM_CCORR_NORMED**: correlação cruzada normalizada (**recomendado**);
- **TM_SQDIFF**: quadrado da soma das diferenças;
- **TM_SQDIFF_NORMED**: quadrado da soma das diferenças normalizado;

A.4.14 Draw Image

A função *rectangle* permite desenhar retângulos numa dada posição definida pelos argumentos *point1* e *point2*. O argumento *lineType* permite definir o tipo de linha a ser utilizada para desenhar o retângulo. Este pode tomar os valores *LINE_8*, *LINE_4*, ou *LINE_AA* para*antialiasing*.

```
1  void rectangle( InputOutputArray img, Point point1, Point point2,
2                  const Scalar& color, int thickness = 1,
3                  int lineType = LINE_8, int shift = 0 );
```

Figura 78. Desenho de um retângulo

Este retângulo pode ser definido através de instâncias da classe ***Point***, ou através da função *boundingRect*. Esta função recebe como argumento um vetor de pontos. É tipicamente utilizado juntamente com contornos.

```
1  boundingRect(InputArray array)
```

Figura 79. Desenho de um retângulo utilizando a função boundingRect

Outra funcionalidade utilizada para se desenhar nas imagens é a função *putText*. Esta função permite escrever texto na imagem pretendida. É possível definir-se o texto, a posição, a fonte, o tamanho da letra, a cor entre outros parâmetros.

```
1  void putText( InputOutputArray img, const String& text, Point org,
2                int fontFace, double fontScale, int thickness = 1,
3                int lineType = LINE_8, bool bottomLeftOrigin = false );
```

Figura 80. Desenho de texto

O argumento *fontFace* tem como responsabilidade a definição do estilo e aparência a aplicar à letra. Não sendo um argumento importante, são enumerados de seguida alguns dos tipos disponíveis:

- **FONT_HERSHEY_SIMPLEX**: Fonte simples;
- **FONT_HERSHEY_PLAIN**: Fonte simples, mais fina que a anterior;
- **FONT_HERSHEY_COMPLEX**: Fonte com detalhes nas extremidades;

B Soluções das tarefas propostas em C++

Nesta secção encontram-se uma das possíveis soluções na linguagem de programação C++, para cada uma das tarefas definidas.

B.1 Tarefa 1: Sobreposição de 2 imagens disponibilizadas

```
1  int main(){
2
3      Mat image1 = imread( "background.jpg", 1);
4
5      Mat image2 = imread( "tigre.jpg", 1);
6
7      Mat imagemResultado;
8      double alpha = 0.5;
9      double beta = ( 1.0 - alpha );
10     addWeighted( image1, alpha, image2, beta, 0.0, imagemResultado);
11
12     namedWindow( "Resultado Combinado", WINDOW_AUTOSIZE );
13     imshow( "Resultado Combinado", imagemResultado);
14
15     waitKey(0);
16 }
17
```

Figura 81. Solução tarefa 1

B.2 Tarefa 2: Identificar, marcar e contar objetos de cor verde em tempo real a partir da câmera

```cpp
int main() {
    VideoCapture captura(0);

    while (true) {
        captura.open(0);

        if (captura.isOpened()) {
            cout << "Camera opened successfully." << endl;
            break;
        } else {
            cerr << "Error: Unable to open the camera. Retrying..." << endl;
            waitKey(1000);
        }
    }

    Mat frame, hsvFrame, mask;
    vector<vector<Point>> contornos;
    Scalar hsvLow(30, 50, 50), hsvHigh(85, 255, 255);

    while (true) {
        captura >> frame;

        if (frame.empty()) {
            cerr << "Error: Failed to capture frame." << endl;
            break;
        }

        cvtColor(frame, hsvFrame, COLOR_BGR2HSV);
        inRange(hsvFrame, hsvLow, hsvHigh, mask);

        findContours(mask, contornos, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);

        Mat detetarFrame = frame.clone();
        int totalCount = 0;
        int totalImageArea = frame.rows * frame.cols;
        for (const auto& contorno : contornos) {
            double area = contourArea(contorno);
            double areaPercentage = area / totalImageArea;

            if (areaPercentage >= 0.027) {
                totalCount++;
                Rect retangulo = boundingRect(contorno);
                rectangle(detetarFrame, retangulo, Scalar(0, 0, 255), 2);
            }
        }

        namedWindow("Objetos Verdes", WINDOW_AUTOSIZE);
        imshow("Objetos Verdes", detetarFrame);

        if (waitKey(1) == 27) {
            break;
        }
    }

    captura.release();
    destroyAllWindows();
    return 0;
}
```

Figura 82. Solução tarefa 2

B.3 Tarefa 3: Encontrar, e identificar o logótipo da UMa

```cpp
const double MATCH_THRESHOLD = 0.5;
int main() {
    // Leitura das imagens
    Mat imageTemplate = imread("logo_uma.png", IMREAD_COLOR);
    Mat image = imread("imagem3.png", IMREAD_COLOR);

    // Transforma em escala de cinzentos
    Mat grayimageTemplate, grayImage;
    cvtColor(imageTemplate, grayimageTemplate, COLOR_BGR2GRAY);
    cvtColor(image, grayImage, COLOR_BGR2GRAY);

    // Template matching
    Mat result;
    matchTemplate(grayImage, grayimageTemplate, result, TM_CCORR_NORMED);
    Point maxLoc;
    double maxVal;
    minMaxLoc(result, nullptr, &maxVal, nullptr, &maxLoc);

    // Aplica threshold
    vector<Point> matches;
    if (maxVal >= MATCH_THRESHOLD) {
        matches.push_back(maxLoc);
    }

    // Desenha retângulos na imagem
    Mat displayImage = image.clone();
    Scalar rectangleColor(0, 0, 255);
    for (const auto& match : matches) {
        Point end(match.x + grayimageTemplate.cols, match.y + grayimageTemplate.rows);
        rectangle(displayImage, match, end, rectangleColor, 2);
    }

    // Mostra imagem
    namedWindow("Matched Image", WINDOW_AUTOSIZE);
    imshow("Matched Image", displayImage);

    waitKey(0);
    return 0;
}
```

Figura 83. Solução tarefa 3

C Formulário da System Usability Scale

Profissão: _____ Idade: _____

Anos de experiência em programação:

Não tem ☐ 1-2 anos ☐ 3-5 anos ☐ 5+ anos ☐

Experiência de programação na Linguagem C++: Não ☐ Sim ☐

1. Acho que gostaria de utilizar este produto com frequência.

Discordo Totalmente	1	2	3	4	5	Concordo Totalmente

2. Considerei o produto mais complexo do que necessário.

Discordo Totalmente	1	2	3	4	5	Concordo Totalmente

3. Achei o produto fácil de utilizar.

Discordo Totalmente	1	2	3	4	5	Concordo Totalmente

4. Acho que necessitaria de ajuda de um técnico para conseguir utilizar este produto.

Discordo Totalmente	1	2	3	4	5	Concordo Totalmente

5. Considerei que as várias funcionalidades deste produto estavam bem integradas.

Discordo Totalmente	1	2	3	4	5	Concordo Totalmente

6. Achei que este produto tinha muitas inconsistências.

Discordo Totalmente	1	2	3	4	5	Concordo Totalmente

7. Suponho que a maioria das pessoas aprenderia a utilizar rapidamente este produto.

Discordo Totalmente	1	2	3	4	5	Concordo Totalmente

8. Considerei o produto muito complicado de utilizar.

Discordo Totalmente	1	2	3	4	5	Concordo Totalmente

9. Senti-me muito confiante a utilizar este produto.

Discordo Totalmente	1	2	3	4	5	Concordo Totalmente

10. Tive que aprender muito antes de conseguir lidar com este produto.

Discordo Totalmente	1	2	3	4	5	Concordo Totalmente