Correção Automática

Cristiano Vagos 65169, Miguel Brás 49977

***Resumo* – O artigo que se segue pretende apresentar o projeto desenvolvido para a Correção Automática de um exame de escolha múltipla, fazendo uso de manipulação e transformação de uma imagem obtida a partir de uma câmara de computador, ou a partir de um ficheiro.**

O artigo começa por fazer uma apresentação teórica sobre o problema, sendo o tema seguinte a discussão e apresentação das soluções encontradas para a realização do programa relacionado com Correção Automática.

I. INTRODUÇÃO

O programa de Correção Automática foi desenvolvido para a unidade curricular Computação Visual (4º ano, 1º semestre) do Mestrado Integrado em Engenharia de Computadores e Telemática, na qual foi usado o OpenCV[1] na linguagem C++ [2]. O objetivo é permitir a deteção e reconhecimento das respostas dadas a um exame de escolha múltipla, sendo uma imagem do exame preenchido obtida a partir de uma câmara de computador.

Este projeto foi desenvolvido utilizando a biblioteca OpenCV (Open Source Computer Vision Library), desenvolvida pela Intel em 2000, de uso gratuito [3]. O OpenCV é multiplataforma, e está disponível para C++, Python, entre outras linguagens de programação, permitindo o desenvolvimento de aplicativos na área da Computação Visual. Uma vez que a linguagem abordada nas aulas foi o C++, utilizamos essa mesma linguagem no projeto que aqui se apresenta.

O projeto Correção Automática está interligado a várias áreas da Computação Visual, tais como a deteção de objetos, análise de imagens e extração de dados, bem como realidade aumentada. São áreas que estão em expansão na computação e que têm cada vez mais importância, pois existem várias áreas de aplicação da mesma tais como a Medicina, que podem, por exemplo melhorar o diagnóstico dos pacientes. É portanto uma área interessante, que certamente será útil e de grande importância no futuro, daí o nosso interesse em efetuar um projeto abordando esta área da Computação Visual para esta unidade curricular.

II. ABORDAGEM AO PROBLEMA

Numa primeira abordagem ao problema, foi nos fornecido um espécime de um exame, um exame modelo pelo qual nos baseámos para podermos fazer a sua correção automática.

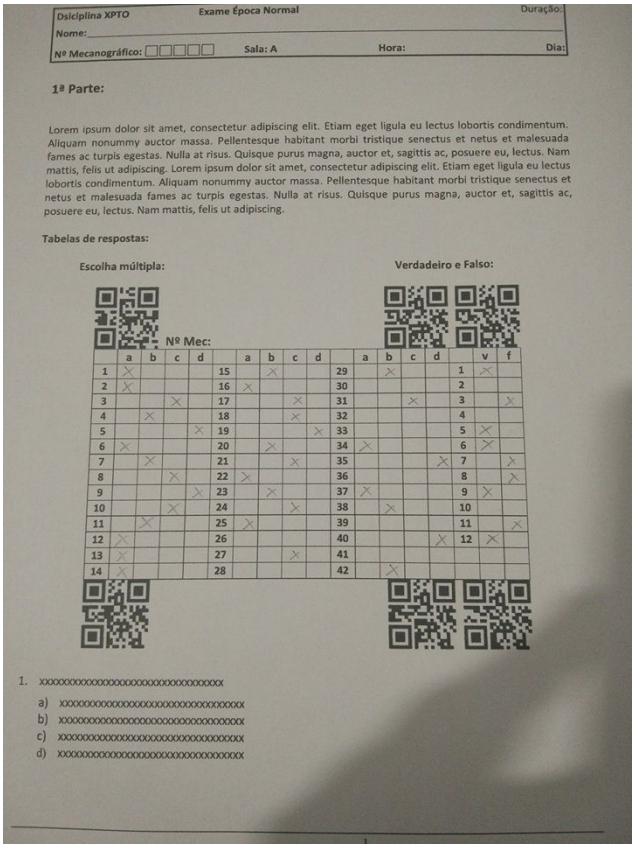

Figura 1 - Exame modelo fornecido completo

De acordo com o trabalho efetuado e numa análise inicial ao problema, o procedimento para a correção automática passa por vários passos ao longo da dita correção. Passos esses que são os seguintes:

- Captura de imagem
- Pré-processamento
- Deteção da tabela matriz de resposta
- Transformação da imagem
- Reconhecimento das respostas
- Mostrar resultado ao utilizador

Obtivemos algumas dificuldades na obtenção da tabela/matriz de respostas utilizando o exame fornecido num dos métodos que implementámos para a deteção da mesma, pelo que iremos apresentar duas formas de obtenção da mesma, uma de forma automática, em que a tabela é capturada e obtida a partir de várias transformações até chegar ao resultado final, bem como

uma extração manual, em que são seleccionados 4 pontos-limite da tabela a ser analisada.

Para a deteção de respostas resolvemos utilizar o preenchimento a cheio de um quadrado para simbolizar a resposta seleccionada no exame. Tentámos implementar outras abordagens e técnicas para a deteção de uma resposta, nomeadamente OCR (Optical Character Recognition) e Template Matching, mas acabámos por escolher o método de preenchimento de um quadrado para simbolizar a escolha de uma resposta e a podermos detetar. Ainda neste artigo explicamos o porquê de termos tomado esta decisão.

Iremos agora descrever cada um dos passos a tomar até chegar ao resultado final que é pretendido.

III. CAPTURA DE IMAGEM

Para capturar inicialmente a imagem foi utilizada uma câmara de telemóvel para então obtermos ficheiros relativos a imagens da tabela de exame, para testarmos durante o desenvolvimento a partir de uma forma estática (o que também é possível de ser utilizado no programa aqui descrito). No entanto, o objetivo foi também passar a captura de imagem para uma câmara de computador, o que foi conseguido. É capturado um frame a partir da câmara e de seguida a imagem é então processada de acordo com os passos definidos anteriormente.

De forma a obter melhores resultados na imagem capturada, a câmara de computador deve ter boa resolução/qualidade de imagem, e as condições de iluminação do espaço onde a imagem é capturada devem ser apropriadas de forma à imagem capturada poder ter boa qualidade e poder ser analisada de uma melhor forma, sendo os seus limites mais nítidos. O ângulo da folha relativamente à câmara deve ser também o mais perpendicular possível, de forma à imagem estar alinhada perfeitamente (ainda que com alguma margem de erro, o que é natural) e podermos obter melhores resultados na sua análise e manipulação.

Após testes realizados com a câmara de computador, estes não foram de todo o esperado, uma vez que a câmara utilizada para o efeito não tem qualidade nem resolução suficiente para que os resultados possam ser de acordo com o esperado. Durante o desenvolvimento foram utilizadas imagens simulando o resultado esperado da captura de uma imagem via câmara de computador, tiradas de um telemóvel. Depois de efetuado o desenvolvimento, passámos a adoptar o método de captura de imagem a partir de uma câmara de computador, cujos resultados não foram positivos. Uma vez que os limites da tabela são finos, aliados à baixa resolução e qualidade de imagem da câmara onde foram feitos os testes, nem todos os limites da tabela são capturados, o que faz com que não seja possível detetar as respostas, ou até mesmo apresentar respostas erradas.

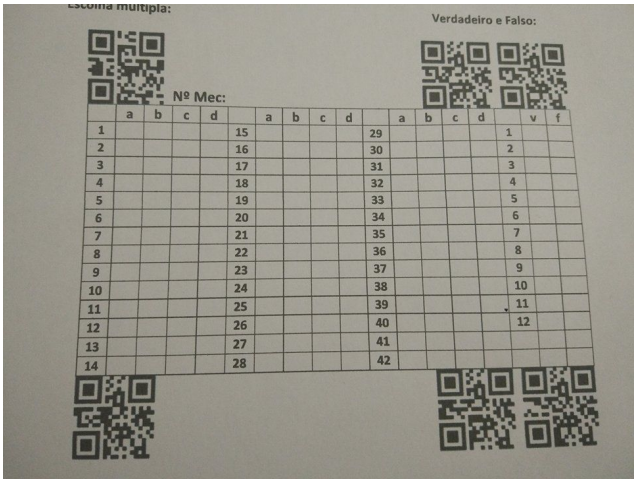

Figura 2 - Imagem de teste capturada pela câmara de telemóvel

IV. PRÉ-PROCESSAMENTO

Após a obtenção da imagem é necessário fazer um pré-processamento da mesma, para que seja mais fácil podermos fazer a deteção que queremos para então se transformar a imagem conforme pretendido, aumentando o desempenho na execução dos algoritmos e procedimentos descritos para análise da imagem.

Inicialmente, criamos uma imagem com o mesmo tamanho da imagem capturada, em escala de cinzentos (CV_8UC1), que será manipulada para obtermos o resultado pretendido. A seguir aplicamos à imagem capturada Desfocagem Gaussiana (Gaussian Blur) para suavizar a imagem, ajudando a remover eventuais ruídos que esta possa ter aquando a sua captura, facilitando a sua manipulação para a deteção adiante [4].

De seguida limitamos os valores de cor da imagem utilizando o tipo de "threshold" adaptável (Adaptive Threshold), pois este é adequado a imagens que tenham vários tipos de iluminação, ao contrário do tipo de "threshold" normal/simples [5]. A principal diferença entre estes dois tipos de "threshold" é que o "threshold" normal trata a imagem como um todo, aplicando a mesma limitação a toda a imagem, enquanto que o "threshold" adaptável vai-se adaptando à imagem de acordo com os valores no elemento da imagem da sua vizinhança a analisar. A diferença pode ser visualizada na Figura 3, onde é notória e permite-nos tomar uma decisão relativamente ao tipo de "threshold" a utilizar no âmbito do nosso projeto.

A Figura 3 demonstra também um dos tipos de imagem que iremos trabalhar, ainda que em situações diferentes e com objetivos diferentes. Na verdade, precisamos de detectar uma tabela, e os valores da mesma. O jogo Sudoku é um exemplo de uma tabela/matriz, que se equipara com a tabela matriz das respostas a um exame, que é o nosso caso de utilização.

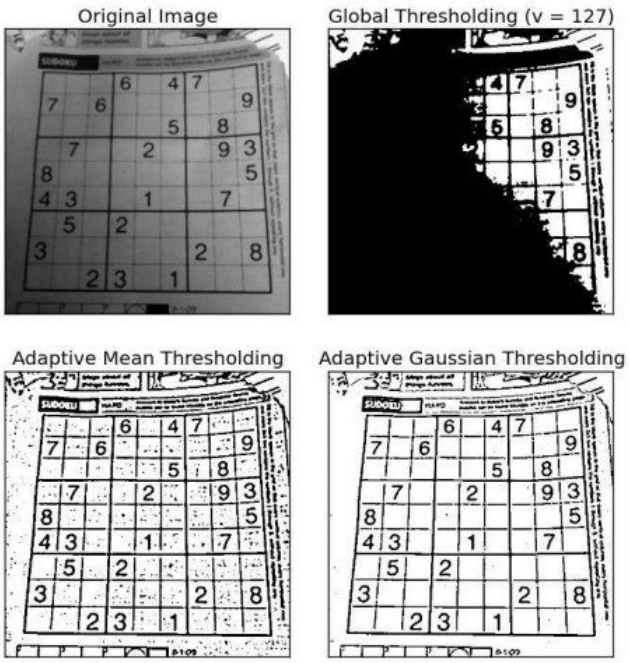

Figura 3 - "Threshold" normal vs "Threshold" adaptável [4]

Após o "threshold", invertemos as cores da imagem, aplicando a operação "bitwise_not" de forma a que os seus contornos fiquem a branco, sendo mais fácil para analisar, conforme iremos mostrar adiante.

O pré-processamento é assim terminado, temos a imagem minimamente trabalhada para que possamos mais facilmente detetar o que queremos. E assim entramos na fase seguinte, a deteção da tabela de respostas.

V. DETEÇÃO AUTOMÁTICA DA TABELA DE RESPOSTAS

De acordo com o exame modelo fornecido, numa primeira análise ao mesmo espera-se que a área da tabela de respostas seja a maior área da imagem capturada, uma vez que a tabela tem o maior tamanho, e é bem visível pelo facto da folha ter apenas as cores preto e branco (e outras cores intermédias), após a conversão feita para escala de cinzentos na fase de pré-processamento. Partindo deste princípio temos agora um método para podermos detetar a tabela de respostas, um método não-convencional (existem outros métodos para encontrar contornos de imagem no OpenCV) mas que nos pareceu adequado no desenvolvimento do programa. Para tal, aplicamos dilatação (uma das propriedades morfológicas mais comuns [6] e importantes, também referida nas aulas) para que os contornos da imagem sejam mais notórios. O elemento estruturante escolhido para esta operação morfológica é o em forma de cruz (MORPH_CROSS) [7], para obtermos maior definição nos contornos da tabela a ser detetada. Após testes com várias imagens decidimos usar o tamanho "3x3" para o elemento estruturante pois foi o que obteve melhores resultados de acordo com o pretendido.

Uma vez que temos os contornos da tabela com maior definição, podemos começar a tentar descobrir qual o elemento com maior área na imagem obtida após a fase de pré-processamento.

Para tal, iteramos toda a imagem e procuramos as maiores áreas pintadas a branco (valor acima de 128 na escala) e cobrimos as áreas encontradas com uma cor mais escura para se distinguirem das outras, sendo o objetivo encontrar a maior área possível (a tabela), o que demonstra a Figura 4. De seguida, pintamos essa área obtida a branco, e cobrimos todas as áreas que não estão a preto, para que fiquemos apenas com a área pretendida. Deste modo ficamos com a maior área encontrada pintada a branco, o que podemos verificar na Figura 5.

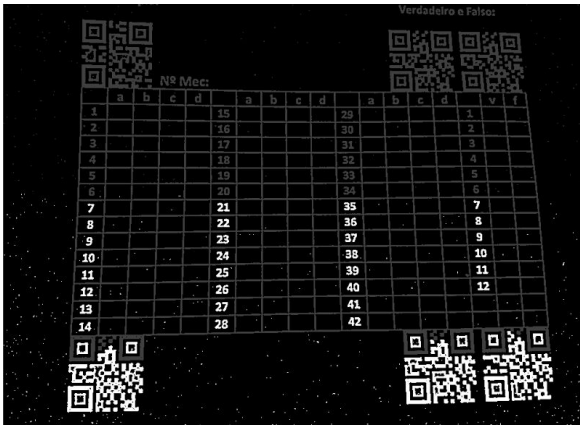

Figura 4 - Processo de procura da maior área no objeto

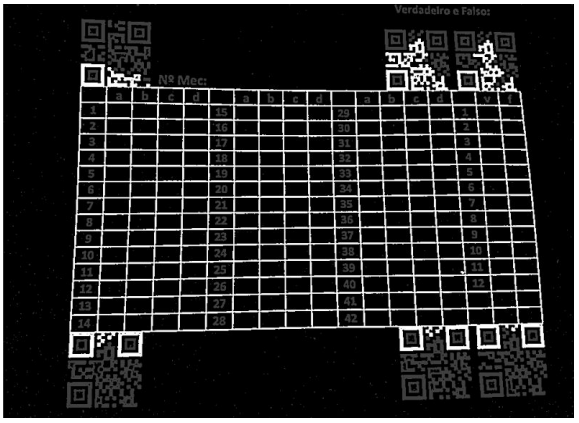

Figura 5 - Maior área possível da imagem encontrada

Com esta operação concluída, obtemos agora a maior área possível a branco, e o restante a preto, ficando apenas com a tabela. Aplicamos de seguida a operação morfológica de erosão à imagem com o mesmo elemento estruturante para que a imagem volte ao seu estado inicial. No exemplo ilustrado acima, é possível ver que tivemos algumas dificuldades com a obtenção da tabela, devido aos códigos QR disponíveis ao longo da tabela. Para tal criámos também outro método para a deteção da tabela de respostas, um método manual.

VI. DETEÇÃO MANUAL DA TABELA DE RESPOSTAS

Conforme podemos visualizar na Figura 5, algumas partes da imagem detetada ficam com pedaços dos códigos QR obtidos nas mesmas. Mais à frente no desenvolvimento a partir da deteção automática da tabela de respostas, não conseguimos obter a tabela com um tamanho perfeito e alinhado, o que era importante para o passo seguinte: transformar a imagem para podermos extrair as respostas na grelha, pelo que criámos um método de deteção manual da tabela de respostas, selecionando através do clique na imagem os 4 pontos-limite (cantos da imagem) para que se possa extrair a tabela a partir dos pontos selecionados.

Após a obtenção da imagem, a aplicação mostra a imagem capturada e fica à espera que o utilizador escolha quatro pontos clicando com o rato, por uma ordem específica (canto superior esquerdo - canto superior direito - canto inferior esquerdo - canto inferior direito).

Usa-se a maior distância horizontal e vertical entre esses pontos e faz-se uma transformação de perspectiva de modo a obter a Tabela isolada.

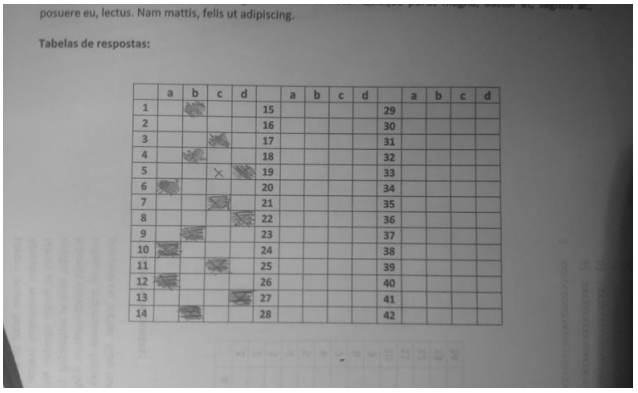

Figura 6 -Imagem capturada

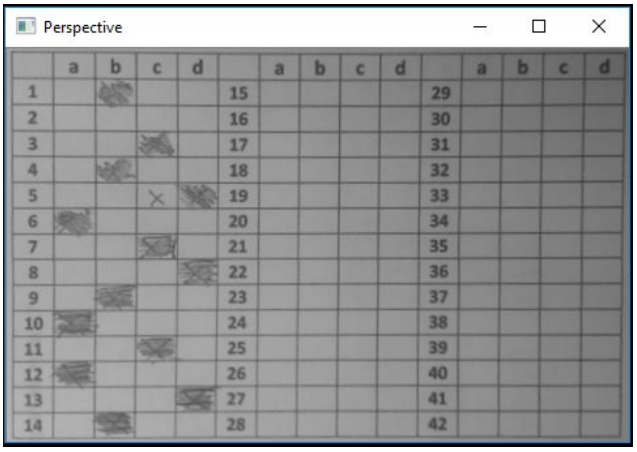

Figura 7 -Imagem após a seleção dos quatro pontos

VII. TRANSFORMAÇÃO DA IMAGEM (AUTOMÁTICA)

A partir da imagem mostrada na Figura 5 e após o preenchimento a preto das áreas mais escuras, ficando apenas com a área preenchida a branco, já temos praticamente a tabela toda definida. No entanto vemos alguns pedaços da imagem a branco que estão a mais, e que não fazem parte da tabela. Precisamos de detetar então as linhas que compõem o exterior da tabela, que sabemos que é um retângulo. Para a deteção de linhas usamos a função HoughLines do OpenCV [8], que é um dos métodos de deteção de linhas disponível, através da transformada de Hough.

Este processo cria várias linhas na imagem a analisar, sendo necessário reduzir o número de linhas a serem consideradas. Analisando a imagem obtemos linhas diagonais, que não é de todo o pretendido, como podemos ver na Figura 8.

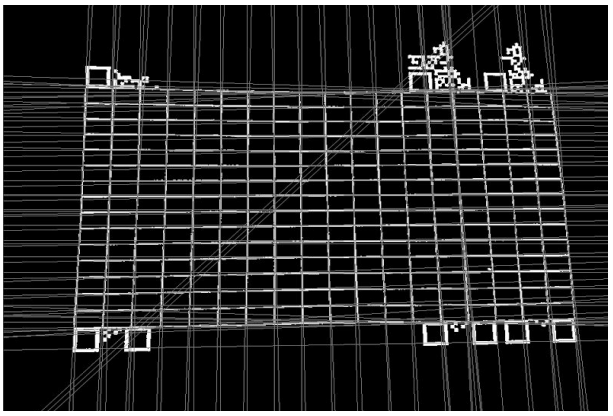

Figura 8 - Deteção de linhas com a função HoughLines

Para obtermos apenas as linhas que queremos vamos então percorrer todas as linhas que temos, e remover as linhas cujo ângulo não é praticamente horizontal (com valores a rondar aproximadamente 180 graus ou sem ângulo) ou vertical (com valores entre os 80 e os 100 graus), comparando as linhas com maior potencial de interseção na tabela. No final deste processo ficamos com as 4 linhas pretendidas (topo, base e laterais), conforme descrito na Figura 9.

Agora é necessário saber quais os pontos onde as linhas se intersectam, de forma a podermos extrair apenas a tabela de respostas. Para tal, recorremos às fórmulas matemáticas de interseção com duas retas, conforme explícito em [9] e [10].

Figura 9 - Linhas da imagem final após deteção de linhas

Obtidos os pontos de interseção, podemos cortar a imagem obtendo apenas o que queremos, a tabela de respostas final obtida através desta transformação de imagem aqui explicada, e visível na Figura 10.

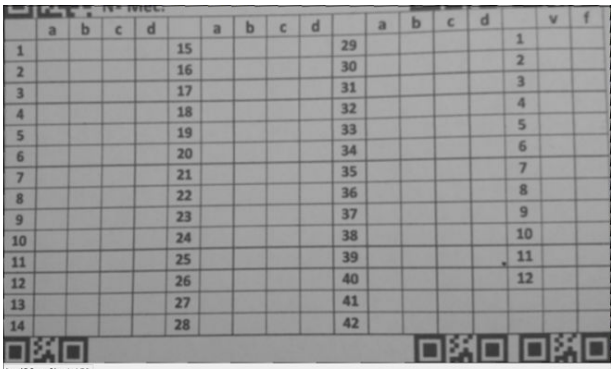

Figura 10 - Imagem final, tabela extraída a partir da transformação

VII. DETEÇÃO E EXTRAÇÃO DAS RESPOSTAS

A forma de resposta que o grupo inicialmente idealizou para a grelha contava com a deteção de cruzes (letra X) em cada quadrado da grelha/tabela. No entanto, para reconhecimento de texto o método recomendado é utilizar uma biblioteca de OCR (Optical Character Recognition)[11] de forma a poder reconhecer o texto (caracter) introduzido, que seria manuscrito, o que é também difícil de detetar. Uma vez que tivemos dificuldades na instalação do mesmo na linguagem C++ e termos de investigar e pesquisar formas de poder implementar tal reconhecimento no nosso programa, tendo em conta o tempo disponível para o desenvolvimento do mesmo descartámos essa hipótese, tendo depois o grupo decidido optar pela técnica de Template Matching, uma forma de poder detetar pequenas partes de uma imagem (template) numa outra imagem [12]. Mas após algumas experiências também se descobriu que o Template Matching é muito limitado para aquilo que se pretende. Criou-se um template de uma cruz, mas não obtivemos resultados positivos. Então decidiu-se fazer uma correção de testes com a resposta escolhida a ser preenchida no quadrado respetivo.

Chegou-se à conclusão que seria preciso separar as linhas horizontais das verticais e guardar as suas coordenadas, então aplicou-se sucessivas transformações de erosão e dilatação aplicando um elemento estruturante para se obter só as linhas verticais e horizontais separadas [13].

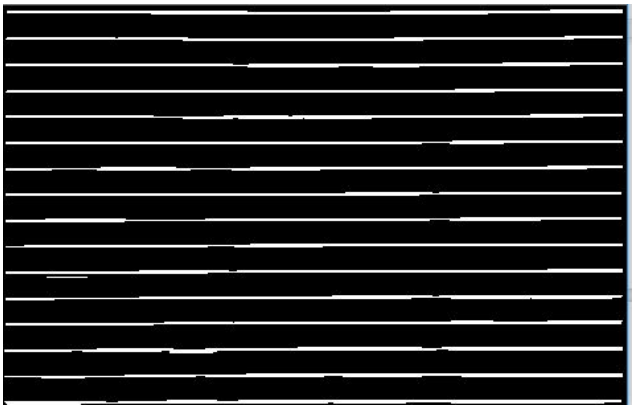

Figura 11 - Linhas Horizontais

Aqui o elemento estruturante escolhido foi o de elipse (MORPH_ELLIPSE).

Após esta operação, procura-se então os contornos usando a aproximação simples (para reduzir pontos a mais) das linhas que serão gravados num array de Pontos. Depois é preciso ordenar os pontos de cima para baixo e da esquerda para a direita. Obtém-se assim uma grelha ordenada em que facilmente descobrimos as coordenadas de cada linha.

Com isto em mente, vamos percorrer todas os quadrados na grelha, usando as coordenadas das linhas.

Cada linha horizontal corresponde a uma combinação de pergunta e resposta. Sabemos que a primeira linha está reservada ao enunciado (A, B, C, D) e que a primeira coluna está reservada para o número da questão. Como cada pergunta tem 5 colunas, então se houver mais colunas (por exemplo, 10), então significa que existe outra tabela de perguntas e respostas. Usando uma média do comprimento da linha, obteve-se o comprimento médio de cada rectângulo de resposta. E a partir da linha vertical esquerda, construímos um rectângulo. Com esse rectângulo fazemos uma sub-matriz da imagem principal, e aplicando um threshold, vemos a densidade de pontos negros usando a função countNonZero [14], e dividindo o resultado dessa operação pelo número total de pixeis da submatriz. Se esse valor for superior a 30% e maior que um valor já detectado, então essa é a resposta selecionada.

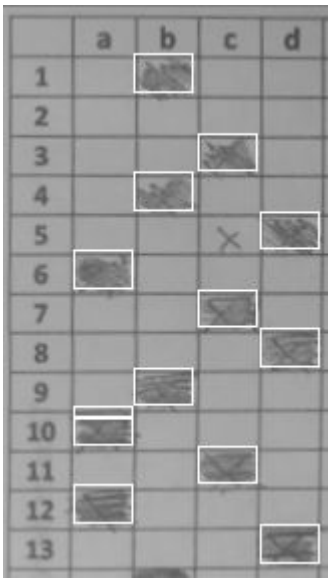

Figura 12 - Resultados

IX. ANÁLISE DOS RESULTADOS OBTIDOS

Como descrito, obtemos resultados diferentes de acordo com o método de deteção a seleccionar (automático / manual). A partir do método automático é possível retirar a tabela, no entanto esta não aparece completa, o que poderá dificultar na deteção/extração das respostas, com o método que abordámos. Os resultados obtidos e mostrados nas Figuras deste artigo sobre a deteção automática foram otimizados tendo em conta o resultado esperado, ajustando os valores de "threshold" da função do OpenCV HoughLines [13].

Já no método manual em algumas situações foram obtidos os resultados esperados tendo em conta uma imagem captada a partir de um telemóvel. Foram efetuados testes com a câmara do computador, mas devido à qualidade de resolução da mesma e pelo facto dos limites da tabela de resultados (linhas e colunas) serem finos não permite a obtenção de bons resultados conforme esperado. O uso de uma câmara com melhor qualidade e com maior resolução irá permitir captar uma melhor imagem e consequentemente, os resultados obtidos serão melhores. Em conjunto com o programa, fornecemos algumas imagens que utilizámos no desenvolvimento do programa para teste. A funcionalidade de captura de imagem via câmara de computador também é possível, sendo que, de acordo com os testes efetuados nenhuma extração obteve resultados positivos.

X. CONCLUSÃO

Aqui foi apresentado e descrito o tema escolhido para o segundo trabalho da unidade curricular Computação Visual: o tema Correção Automática. Também foram descritos os métodos, técnicas e ferramentas utilizadas para obtermos os resultados pretendidos. O programa foi criado na linguagem C++, uma das linguagens disponíveis para o OpenCV, biblioteca extensivamente utilizada no desenvolvimento deste trabalho, e leccionada na segunda parte das aulas da dita unidade curricular.

Muitos dos métodos e técnicas aqui usados foram retirados através de alguma investigação e pesquisa da nossa parte em vários websites relacionados com a Computação Visual / Visão por Computador, entre eles os tutoriais do OpenCV e da comunidade, retirando ideias e abordagens ao problema que tínhamos em mão. Sentimos muitas dificuldades no desenvolvimento e implementação do programa, pelo facto de não termos abordado alguns problemas do género nas aulas práticas, e também pelo facto da maior parte dos utilizadores de OpenCV usarem a linguagem Python ao invés da linguagem C++, o que facilita o desenvolvimento de programas usando o OpenCV, pois existem vários exemplos pela Internet feitos nesta linguagem sobre diversos problemas relacionados com a deteção de objetos, reconhecimento, entre outros, o que nos leva a sugerir o uso desta linguagem nesta unidade curricular no futuro.

Os resultados que obtivemos no final não foram de todo satisfatórios, pois na deteção automática (um dos objetivos do trabalho) não conseguimos extrair no final toda a tabela, sem excessos, cortando o que está a mais na imagem que não seja a tabela, deixando apenas a dita tabela, podendo depois detetar e extrair a informação relativa às respostas de uma forma mais simples. A solução encontrada para reverter esta situação foi uma abordagem manual na deteção da tabela, definindo os cantos da imagem para melhor interpretação da tabela, de acordo com o método de deteção/extração das respostas, mas também esta solução não é muito satisfatória porque depende muito da iluminação e do ângulo em que a imagem é captada, sendo que em imagens particularmente escuras, irá detectar respostas onde estas não existem, pelo método que escolhemos.

Apesar das dificuldades encontradas, acreditamos que esta área da computação é muito importante e interessante, e que terá grande notoriedade no futuro, sendo primordial a sua aprendizagem e prática, tal como foi feito. O trabalho em si é interessante do ponto de vista pedagógico, mas a falta de compreensão e de prática das ferramentas necessárias para desenvolver este trabalho aliada ao tempo disponível do grupo para o desenvolvimento do mesmo foram as principais dificuldades. Haveria mais trabalho pela frente e muito por onde melhorar, tal como outras técnicas e métodos possíveis de serem aplicados para uma análise, deteção e extração de dados mais eficiente e eficaz nos resultados. No entanto o trabalho que aqui se apresenta foi o possível, sabendo que demos o nosso melhor para apresentar uma solução ao problema sugerido.

Consideramos assim que o OpenCV é uma excelente ferramenta para o desenvolvimento de aplicativos e programas na área da Computação Visual.

REFERÊNCIAS

[1] Joaquim Madeira, http://sweet.ua.pt/jmadeira/OpenCV/ em <http://sweet.ua.pt/jmadeira/OpenCV/>, acesso em Janeiro 2018

[2] OpenCV, página oficial em <https://opencv.org/>, acesso em Janeiro 2018

[3] C++, em Wikipedia <https://en.wikipedia.org/wiki/C%2B%2B>, acesso em Janeiro 2018

[4] OpenCV: Smoothing Images, em <https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html>, acesso em Janeiro 2018

[5] OpenCV: Image Thresholding, em <https://docs.opencv.org/3.3.1/d7/d4d/tutorial_py_thresholding.html>, acesso em Janeiro 2018

[6] OpenCV: Morphological Transformations, em <https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html>, acesso em Janeiro 2018

[7] Open CV: Image Filtering (cv::MorphShapes), em <https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html#gac2db39b56866583a95a5680313c314ad>, acesso em Janeiro 2018

[8] Hough Line Transform - Open CV, em <https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html>, acesso em Janeiro 2018

[9] Line-line intersection - Wikipedia, em <https://en.wikipedia.org/wiki/Line%E2%80%93line_intersection>, acesso em Janeiro 2018

[10] How to calculate the point of intersection between two lines - Stack Overflow, em <https://stackoverflow.com/questions/16524096/how-to-calculate-the-point-of-intersection-between-two-lines>, acesso em Janeiro 2018

[11] Optical character recognition - Wikipedia, em <https://en.wikipedia.org/wiki/Optical_character_recognition>, acesso em Janeiro 2018

[12] Template Matching - Wikipedia, em <https://en.wikipedia.org/wiki/Template_matching>, acesso em Janeiro 2018

[13] OpenCV: Feature Detection (cv::HoughLines), em <https://docs.opencv.org/master/dd/d1a/group__imgproc__feature.html#ga46b4e588934f6c8dfd509cc6e0e4545a>, acesso em Janeiro 2018

[14] code by Theodore, at How extract tables from an image? <http://answers.opencv.org/question/63847/how-to-extract-tables-from-an-image/>, acesso em Janeiro 2018

[15] OpenCV: Operations on Arrays em <https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html>, acesso em Janeiro 2018