Projecto de Computação Visual – Extração e Leitura de Códigos QR

Nuno Humberto Paula

Resumo **– Este relatório descreve a concepção e funcionamento de uma aplicação que permite a deteção e extração de códigos QR em tempo real, através da obtenção de uma stream de vídeo com recurso à webcam do utilizador. Após o processamento e extração da zona do frame correspondente ao código QR, é efectuado um ajuste de perspectiva de forma a restaurar o código à sua forma plana. Da imagem resultante, é extraída uma matriz de bits, que é posteriormente interpretada. Os estágios da deteção do código, juntamente com a interpretação da sua matriz de bits, são apresentados ao utilizador.**

Abstract **– This report describes the conception and operation of an application that allows the real-time detection and extraction of QR codes by obtaining a live video feed from the local webcam. After processing every frame from the live feed and extracting the QR code segment from it, a perspective transformation is applied to the segment, restoring the QR code to its original flat shape, from which a bit matrix is then extracted and decoded. The stages of code detection and bit matrix decoding are presented to the user.**

I. Introdução

Um código QR (Quick Response) consiste num código de barras bidimensional, cuja leitura óptica pode ser efectuada por diferentes tipos de dispositivos como, por exemplo, smartphones.

A deteção de um código QR presente nos frames obtidos da webcam do utilizador é efectuada através da deteção de contornos, com recurso a um detector de Canny, cuja hierarquia é posteriormente verificada, possibilitando a enumeração dos seus três elementos de posição.
A estrutura de um código QR é descrita com maior pormenor na secção II.

Para a transformação de perspectiva, é necessário calcular a posição dos quatro vértices do código QR na imagem de origem, aos quais irão corresponder os quatro vértices da imagem de destino, quadrada, numa matriz 500 por 500.

Após a aplicação da transformação de perspectiva, são verificados todos os elementos de posição, timing e alinhamento para, em seguida, ser efectuada uma tentativa de interpretação do conteúdo do código QR.

Na aplicação desenvolvida é mostrado ao utilizador, em tempo real, o vídeo obtido através da sua webcam, juntamente com a imagem plana do código QR obtido e o resultado da interpretação de cada código QR,
É também mostrada ao utilizador a deteção dos quatro cantos do código.

Todo o processamento de imagem é efectuado com recurso à utilização de OpenCV 3.1. Não são usadas quaisquer bibliotecas adicionais.

II. Estrutura

Os códigos QR dispõem de elementos de posição, alinhamento e timing.
Estes elementos facilitam a sua deteção e ajuste de perspectiva.

Fig. 1 - Exemplo de código QR (link para a Wikipedia), com destaque dos elementos de timing (a azul)

Para a validação da perspectiva correcta de um código QR, a correcta localização de cada um destes elementos deve ser verificada antes de proceder à interpretação do seu conteúdo.

Na Fig. 2 segue-se a representação de cada um destes elementos, presentes num código QR.

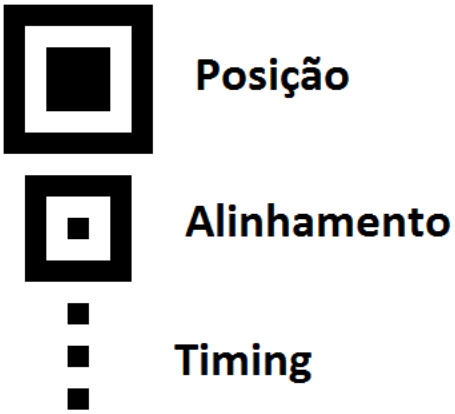

Fig. 2 - Aspecto dos diferentes elementos presentes num código QR

III. Deteção e transformação do Código QR

A identificação de cada elemento de posição é efectuada através de uma deteção de contornos. Cada um destes elementos deverá dispor de 5 contornos descendentes. Esta hierarquia é verificada no momento de deteção do código QR.

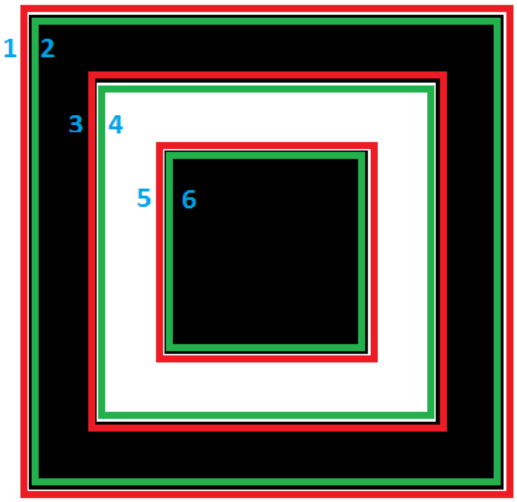

Fig. 3 – Hierarquia de contornos de um elemento de posição

A existência de exactamente quatro vértices no contorno exterior de cada elemento de posição é verificada através da aproximação do contorno a um polígono. Tanto o centro de massa do contorno como os seus quatro vértices são armazenados para processamento posterior.

Em seguida, são calculadas as distâncias entre os centros de massa para os três elementos de posição, com o objectivo de identificar quais os dois elementos de posição que participam na hipotenusa do triângulo rectângulo formado pelos três elementos.

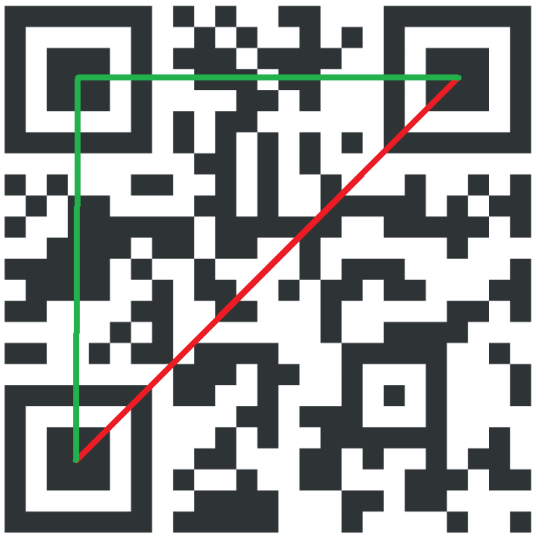

Fig. 4 – Identificação da hipotenusa (a vermelho)

O elemento de posição que não pertencer à hipotenusa, será sempre o elemento que deverá estar localizado no canto superior esquerdo do código QR. Esta deteção permite a rotação (se necessária) do código QR obtido.

Resta identificar qual dos dois elementos de posição contidos na hipotenusa pertence ao canto superior direito ou ao canto inferior esquerdo.

Para tal, é calculada a distância perpendicular da hipotenusa (definida pela recta à qual pertencem os pontos (x1, y1) e (x2, y2)) ao ponto (x0, y0) correspondente ao canto superior esquerdo (Fig. 5) e o declive da recta correspondente à hipotenusa (Fig. 6).

$$\mathrm{d} = \frac{(y_2 - y_1)x_0 - (x_2 - x_1)y_0 + x_2 y_1 - y_2 x_1}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}$$

Fig. 5 – Fórmula utilizada para o cálculo da distância perpendicular

$$\mathrm{k} = \frac{y_2 - y_1}{x_2 - x_1}$$

Fig. 6 – Fórmula utilizada para o cálculo do declive

Após o cálculo destes valores, através do sinal dos mesmos, é possível deduzir a posição relativa (à esquerda ou á direita da recta da hipotenusa) do elemento de posição em que a hipotenusa não participa, que deverá corresponder ao canto superior esquerdo do código QR e, desta forma, também é possível deduzir em que canto deverá estar cada um dos restantes elementos de posição.

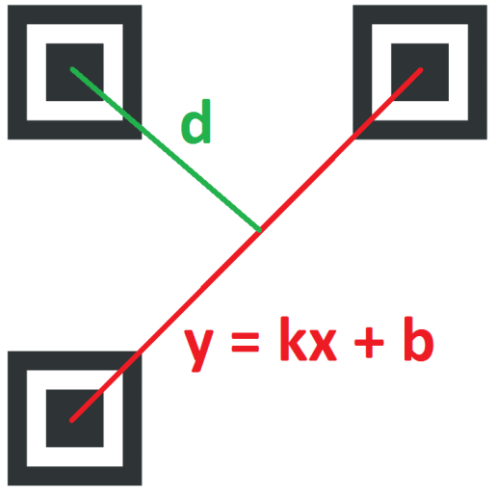

Fig. 7 – Representação da hipotenusa e da distância perpendicular calculada

Após a identificação da correcta posição de todos os elementos de posição, é necessário o cálculo da posição do canto inferior direito do código QR.

Para tal, é necessário calcular o ponto de intersecção entre a recta em que estão contidos os pontos A1 e A2 e a recta em que estão contidos os pontos B1 e B2, como mostra a Fig. 8.

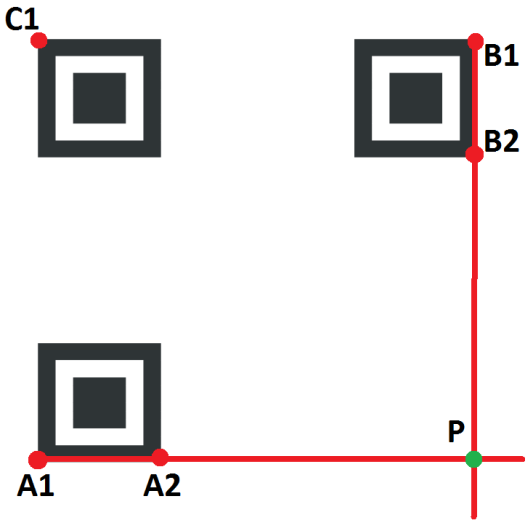

Fig. 8 – Cálculo do ponto de interseção de duas rectas

Para o cálculo deste ponto de intersecção é utilizado o método descrito por Gareth Rees [1].

Os pontos A1, B1 e C1 correspondem aos vértices dos elementos de posição (armazenados anteriormente, como descrito imediatamente após a Fig. 3) mais afastados do centro do código QR, escolhidos através da comparação das suas distâncias ao centro do código.

Os pontos A2 e B2 são obtidos iterando de forma circular, no sentido horário (para B2) ou anti-horário (para A2) os vectores onde estão armazenados os vértices dos elementos de posição.

Com o cálculo dos pontos A1, B1, C1 e P, já é possível calcular uma matriz de transformação para ser utilizada na transformação de perspectiva.

Para tal, é efectuada uma chamada à função *getPerspectiveTransform()*, à qual são fornecidas como coordenadas de origem os pontos {C1, B1, P, A1} e como coordenadas de destino os pontos: {(0,0) , (500, 0) , (500,500) , (0, 500)} correspondentes a uma nova imagem de dimensões 500 por 500, que apenas contém o código QR.

Utilizando a matriz obtida, a transformação é efectuada através de uma chamada à função warpPerspective().

Ao resultado, é aplicado um threshold de forma a que este possua apenas duas cores.

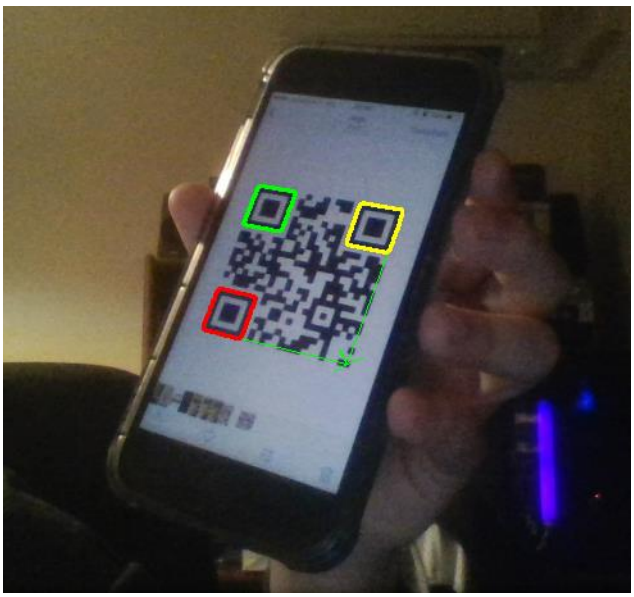

Fig. 9 - Imagem de entrada

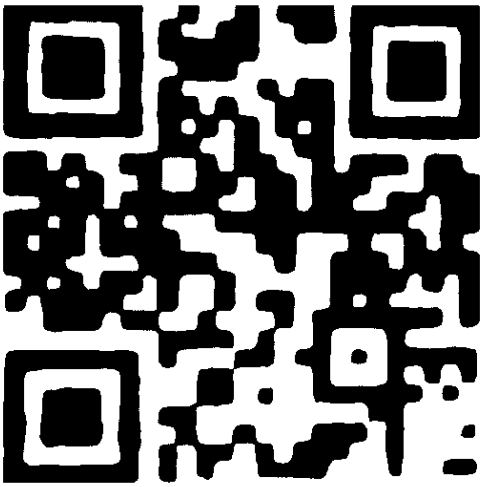

Fig. 10 – Resultado da transformação

IV. Extração Da Matriz de Dados

Após ser efectuada a transformação, cada pixel do código QR (pixel como menor elemento do código QR) deve ser transferido para uma matriz (*array* bidimensional, 25 por 25).

Para tal, a imagem é convertida numa grelha, em que a cor do ponto central de cada célula decidirá o valor do respectivo pixel, sendo 1 para preto e 0 para branco.

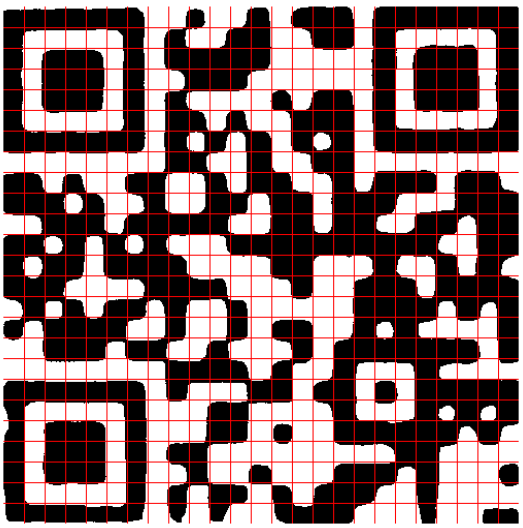

Fig. 11 – Divisão dos pixéis do código QR

Assim que a matriz é preenchida, são imediatamente verificados todos os elementos de posição, timing e alinhamento. Se os valores da matriz nos locais desses respectivos elementos não forem exactamente iguais ao apresentado na Fig. 12, o resultado é rejeitado e a matriz é extraída novamente da transformação, desta vez com um valor de threshold diferente.

Os valores utilizados para o threshold variam entre 64 e 192.

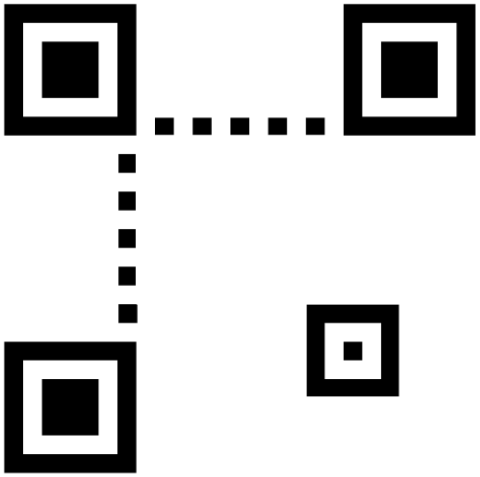

Fig. 12 – Posição correcta de todos os elementos de verificação

Se a matriz passar em todas as verificações, é iniciado o processo de interpretação da matriz de dados.

V. Interpretação da Matriz de Dados

A qualquer código QR está associada uma máscara, aplicada sobre todos os seus bits (pixéis) com excepção dos seus elementos de posição, alinhamento e timing. A máscara corresponde a um padrão segundo o qual os bits do código devem ser invertidos. Existem 8 padrões diferentes para a máscara e, para saber qual deve ser aplicado, devem ser interpretados três bits do código QR, como apresentado na Fig. 13.

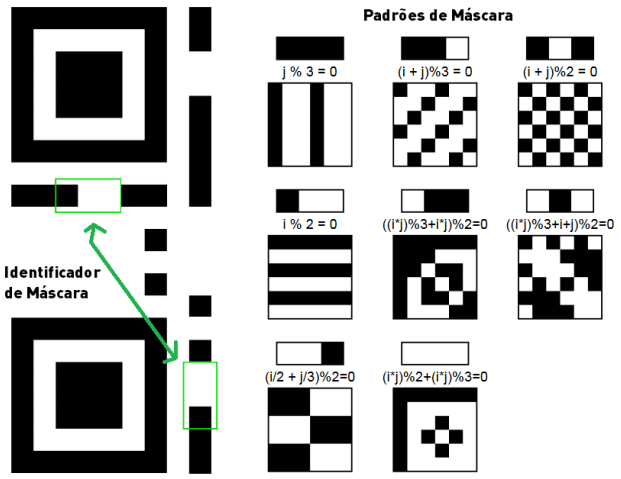

Fig. 13 – Diferentes padrões de máscara

No caso da Fig. 13, o identificador de máscara detectado corresponde ao quarto padrão, em que todas as linhas ímpares do código QR deverão ser invertidas.

A aplicação desenvolvida suporta todos os tipos de máscara.

Após a aplicação da máscara à matriz de dados, já é possível interpretar o conteúdo do código QR.

O primeiro passo deverá ser a interpretação do *encoding* e do número de bytes a ler, esta informação está armazenada no canto inferior direito do código QR.

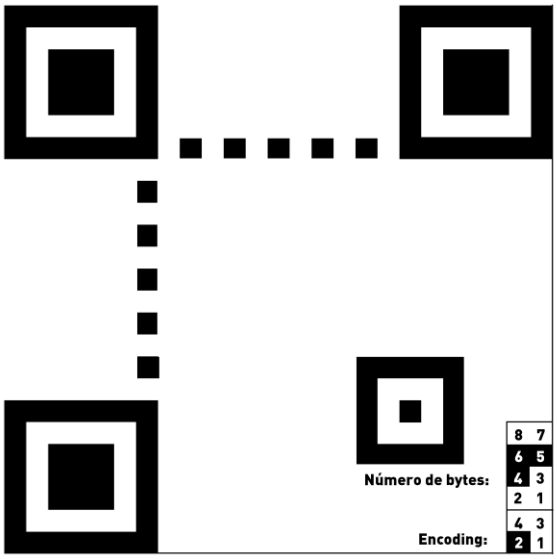

Fig. 14 – Informação de número de bytes a ler e *encoding*

Os bits das regiões de *encoding* e número e bytes deverão ser lidos num padrão de ziguezague, como apresentado na Fig. 14. Como mencionado anteriormente, um bit preenchido a preto corresponde a um 1 e um bit preenchido a branco corresponde a um 0.

Para o *encoding*, tendo em conta que o bit 1 é o mais significativo e o bit 4 é o menos significativo, obtemos o *nibble* 0100, correspondente ao número 4.

Diferentes tipos de *encoding* são suportados nos códigos QR, mas para esta aplicação apenas serão considerados os códigos com o tipo 4, que correspondem à grande maioria dos códigos QR encontrados diariamente. O tipo 4 corresponde a *byte encoding*, e significa que cada byte extraído poderá conter qualquer caracter da tabela ASCII.

Em seguida, é interpretada a zona relativa ao número de bytes a ler utilizando o mesmo métodos. Tendo em conta que o bit 1 é o mais significativo e o bit 8 é o menos significativo, obtemos o byte 00011100, correspondente ao valor 28, significando que no código QR estão armazenados 28 bytes de dados.

Existem 40 versões diferentes de códigos QR, sendo que nesta aplicação é considerada a versão 2, com capacidade para até 32 bytes de dados.

Após a interpretação do *encoding* e do número de bytes a ler, a leitura de cada byte de dados do código QR pode ser iniciada.

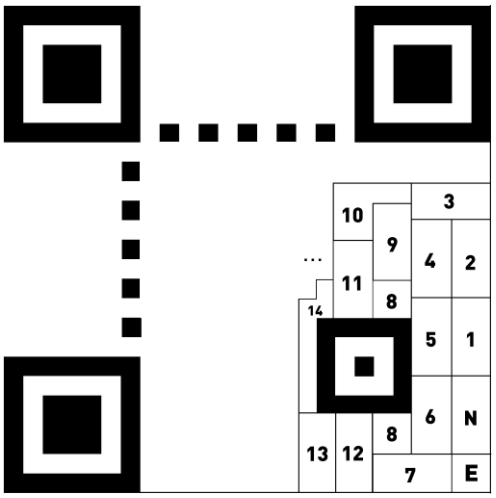

Fig. 15 – Exemplo da ordem de leitura do conteúdo do código QR

A leitura dos bytes do código QR deve ser sempre efectuada da direita para a esquerda, iniciando-se no sentido ascendente e invertendo o mesmo ao atingir um limite, como mostra a Fig. 15.

A leitura dos bits de cada byte deve ser sempre feita no padrão de ziguezague indicado na Fig. 14. O padrão também é invertido verticalmente sempre que é invertido o sentido de leitura.

Cada um dos bytes interpretados da matriz de dados é convertido para um caracter e adicionado a uma *string*, que é devolvida no final da leitura da matriz.

Somente para efeitos de demonstração, sempre que o conteúdo do código QR se tratar de um URL, este é aberto no browser do utilizador através de uma chamada da função system(), utilizando como parâmetro a *string* "start chrome URL".

Por outro lado, sempre que se tratar de uma mensagem simples, esta será mostrada na consola da aplicação.

VI. Conclusão

Com a elaboração deste projecto conseguiu-se a criação de uma aplicação intuitiva e interessante, que interage com o utilizador através da câmera do seu computador. Dependendo da qualidade e resolução dos frames obtidos e da iluminação do local, a leitura do código QR pode ser quase instantânea.

Foi também possível a aplicação dos conhecimentos de computação visual adquiridos durante o semestre.

Um possível próximo passo que tornaria a aplicação mais versátil, embora não relacionado com OpenCV, seria possibilitar também a criação de um código QR com o conteúdo escolhido pelo utilizador.

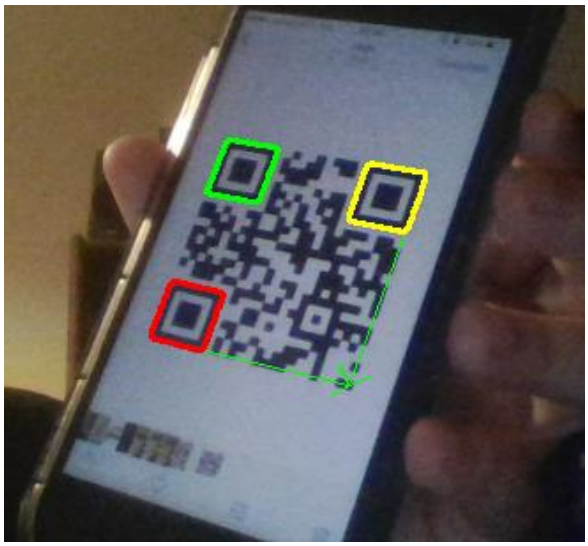

Fig. 16 – Exemplo de imagem de origem

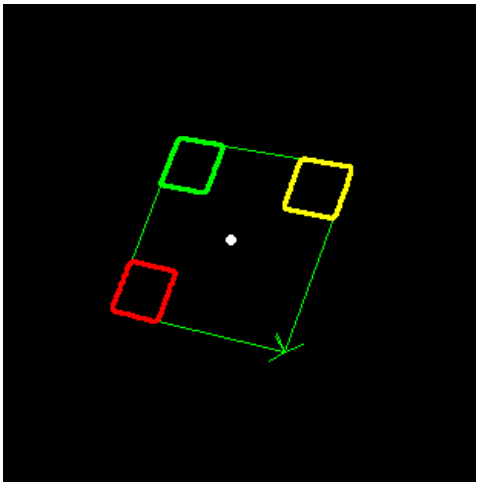

Fig. 16 – Processamento

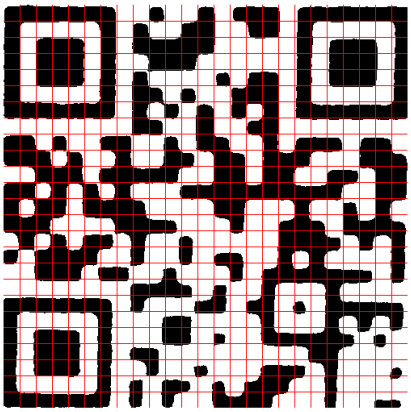

Fig. 17 – Aquisição de matriz

Fig. 18 – Resultado

Referências

[1] Gareth Rees: How do you detect where two line segments intersect?, StackOverflow, visto 6 Janeiro 2017, http://stackoverflow.com/questions/563198/how-do-you-detect-where-two-line-segments-intersect/.