Trabalho 1

Visão Computacional

**Ricardo Silva**

2192447@my.ipleiria.pt

**José Rosa**

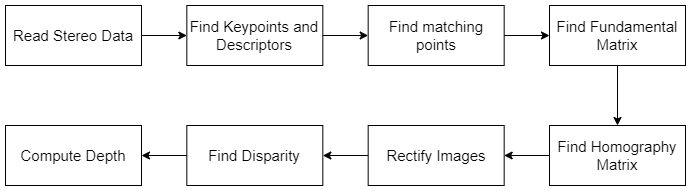
2190383@my.ipleiria.pt

# Introdução

# Tarefa 1 – Visão Estéreo

A primeira tarefa do trabalho de visão computacional tem como objetivo implementar todo o fluxo necessário para se computar o mapa de profundidade a partir das imagens esquerda e direita captadas pelas camaras infravermelhas.

O fluxo implementado de modo a completar a tarefa proposta é ilustrado na figura X. Cada uma das tarefas ilustradas será detalhada num subcapítulo próprio.



## Read Stereo Data

A aquisição das imagens *stereo* infravermelhas (IR) é feita através da leitura dos *streams* disponibilizados no ficheiro [*bag*](http://wiki.ros.org/Bags) fornecido pelo docente. O dispositivo que foi usado para capturar os streams fornecidos no ficheiro *bag* foi a camara de profundidade [Intel® RealSense™ D435](https://www.intelrealsense.com/depth-camera-d435/).

Inicialmente é criada e configurada uma *pipeline* *realsense* que permite a interação com o dispositivo e os seus módulos. Esta *pipeline* é configurada para que um dispositivo seja criado a partir de um ficheiro, no nosso caso, o dispositivo é criado a partir do ficheiro *bag* fornecido. Os diversos *streams* são depois configurados e habilitados para leitura, sendo a aquisição das *frames* dos mesmo feita num ciclo infinito. O ciclo infinito onde as *frames* são adquiridas é onde é feito todo o processamento das *frames* infravermelhas esquerda e direita para cálculo do mapa de profundidade e a comparação da mesmo com o mapa de profundidade fornecido pelo dispositivo *emulado* pelo ficheiro.

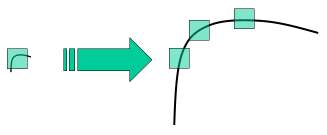
A implementação do código necessário à extração das *frames* teve como base o código fornecido pelo docente.

## Find Keypoints and Descriptors

<https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html>

De modo a, posteriormente, fazermos corresponder pontos entre a imagem esquerda e a imagem direita necessitamos de determinar os pontos de interesse (*keypoints*) e as características que os descrevem (*descriptors*) em ambas as imagens.

Usou-se o algoritmo Scale-Invariant Feature Transform (SIFT) proposto por David Lowe em 2004 para computar os *keypoints* e o seus *descriptors*. Este algoritmo é amplamente usado pois este é invariante a diferentes fatores de escala. Ao usar SIFT um canto é sempre detetado mesmo que este tenha uma escala diferente. Isto não acontece com os detetores de cantos comuns, como o detetor de *Harris*, onde o canto só é detetado como canto se o tamanho da janela for o adequado [Figura X].



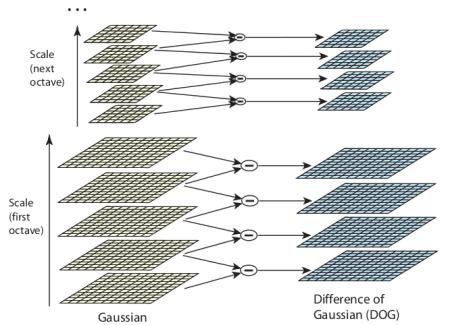
De referir que para além de este algoritmo ser invariante a fatores de escala este também é robusto a rotações, distorções afins (*affine* *distortion*), modificações do ponto de vista (*viewpoint*) e modificações na iluminação devido ao método de implementação do mesmo.

O algoritmo SIFT é composto por duas partes, a primeira é o detetor de *keypoints* e a segunda descreve os *keypoints* usando *descriptors*.

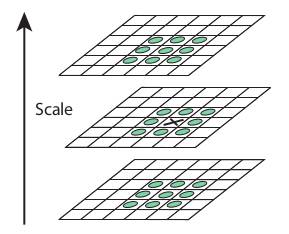
A determinação de *keypoints* é conseguida através da aplicação do d*ifference* *of* *gaussian* (DoG) na imagem, usando diferentes valores de sigma (σ) e subtraindo duas imagens consecutivas no espaço de escalas (*scale space*), de modo a que o resultado seja uma aproximação do *laplacian of gaussian* que usa menos recursos.

A variação de σ faz com que seja possível determinar *keypoints* de vários tamanhos (*blobs*) sendo que quando maior o valor de σ maior o tamanho do *blob* encontrado.

Em termos práticos a determinação de um *keypoint* é feita aplicando um filtro gaussiano com diferentes valores de σ à imagem em diferentes escalas, sendo que, para cada uma das escalas, duas imagens consecutivas (com diferentes σ) são subtraídas resultando numa imagem/matriz que representa a diferença gaussiana entre ambas [Figura X].



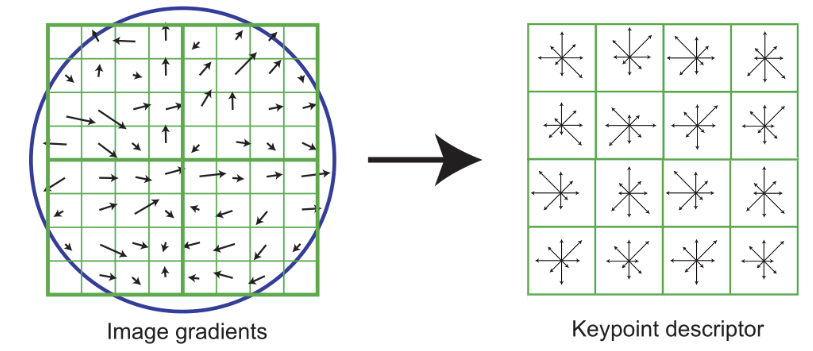
Após a determinação das matrizes DoG é feita a comparação de todos os elementos (pixéis) com os elementos contidos na janela de vizinhança da mesma matriz e das matrizes DoG superior e inferior. Se o elemento a ser avaliado tiver um valor extremo (mínimo ou máximo) quando comparado com os outros elementos então este é considerado um *keypoint*.



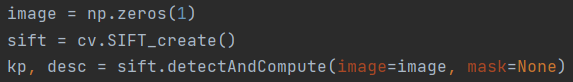
Após os keypoints serem determinados é aplicado um *threshold*, semelhante ao aplicado no algoritmo de Harris,de modo a diferenciar as zonas com pouco contraste e os contornos da imagem de modo a obtermos apenas os pontos de interesse fortes.

Para além da aplicação do *threshold* é calculada a orientação do *keypoint* tendo em conta o sentido e magnitude do gradiente nessa região, isto é, para todos os pixéis da região é calculado o gradiente. Após o cálculo dos gradientes estes são divididos em 36 categorias que caracterizam o sentido do mesmo em intervalos de 10° ao longo de 360°. Cada gradiente tem uma contribuição proporcional à sua magnitude para a sua categoria. À representação do sentido e magnitude dos diversos gradientes numa região dá-se o nome de histograma. Após o cálculo do histograma o valor mais alto do mesmo e os que correspondem a pelo menos a 80% do maior pico são considerados para o cálculo e atribuição do sentido ao *keypoint*.

A determinação dos *descriptors* proposta por David Lowe consiste em criar uma área de 16x16 pixéis em torno do *keypoint*, sendo que esta área é dividida em 16 blocos de 4x4 pixéis. Para cada bloco de 4x4 pixéis é calculado um histograma de 8 valores com o sentido dos mesmos [Figura X]. Além do sentido de cada bloco são também guardados outros dados de modo a descrever um *keypoint*.



O algoritmo descrito foi implementado em *python* de modo a obtermos os *keypoints* e *descriptors* das imagens infravermelhas esquerda e direita, sendo que, para isso usou-se a função *detectAndCompute* do *openCV* aplicada a um objeto do tipo SIFT com os parâmetros padrão [Figura X]. Esta função apenas nos devolve os *keypoints* e o *descriptors*, pelo que, de modo a implementar o algoritmo SIFT ainda se tem de achar os pontos correspondentes e aplicar o teste de rácio proposto por David Lowe.



## Finding Matching Points

<https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html>

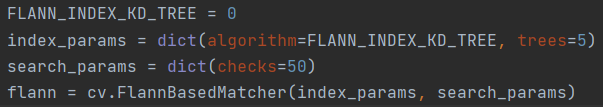
A correspondência entre os *keypoints* nas duas imagens é feita através da escolha de um par de *descriptors* com características semelhantes, sendo que as melhores correspondências são as que apresentam uma distância menor, o que permite que a pesquisa seja feita usando o *nearest* *neighbour*.

A implementação deste processo em *python* pode ser feita de diversas formas, das quais:

1. Implementação do algoritmo de raiz;
2. Usando um objeto que descenda da classe *DescriptorMatcher*:
   1. Brute-Force Matcher ([BFMatcher](https://docs.opencv.org/4.4.0/d3/da1/classcv_1_1BFMatcher.html))
   2. *Fast Library for Approximate Nearest Neighbors* (*[FlannBasedMatcher](https://docs.opencv.org/4.4.0/dc/de2/classcv_1_1FlannBasedMatcher.html)*)

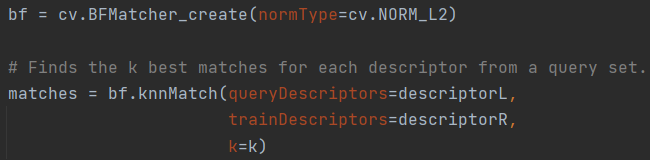
O *Brute-Force* *Matcher* procura a correspondência de um ponto num conjunto de pontos usando como métrica a distância entre pontos. Esta métrica pode ser definida quando se instância o objeto, sendo que para o caso de SIFT é recomendado usar como métrica a distância euclidiana definida pela constante [*NORM\_L2*](https://docs.opencv.org/4.4.0/d2/de8/group__core__array.html#gad12cefbcb5291cf958a85b4b67b6149f) do *openCV*.

A [*FLANN*](https://github.com/mariusmuja/flann) é uma biblioteca que contém uma coleção de algoritmos otimizados para efetuar a pesquisa por o *nearest* *neighbour* num conjunto de dados de grandes dimensões. De modo a instanciar este objeto é necessário passar como parâmetros dois dicionários, o primeiro que define qual o tipo de algoritmo a usar e os seus parâmetros, e o segundo que define os parâmetros de pesquisa do algoritmo. No caso do *SIFT* os parâmetros recomendados são ilustrados na figura X.



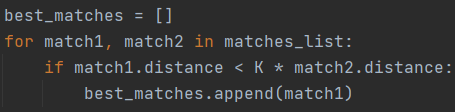
Configuração e instanciação de um objeto do tipo FLANN

Após a criação de um objeto do tipo *FLANN* ou *BFMatcher* é chamada a função *knnMatch* de modo a obtermos as correspondências entre os keypoints. Esta função recebe como parâmetros os descritores de ambas as imagens e um inteiro que representa o número de melhores resultados que pretendemos obter por correspondência e devolve uma estrutura de dados que contem informação acerca das correspondências.



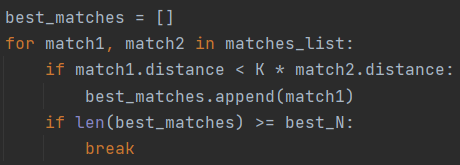
Exemplo de pesquisa usando um objeto do tipo BFMatcher

De acordo com o algoritmo SIFT devemos obter as duas melhores correspondências e aplicar um teste de rácio de modo a validarmos as mesmas. Se a melhor correspondência for inferior a 80% da segunda melhor correspondência então é considerada a melhor correspondência como válida. David Lowe recomenda um rácio de 80%, no entanto este pode ser alterada de modo a melhorar os resultados [Figura X].



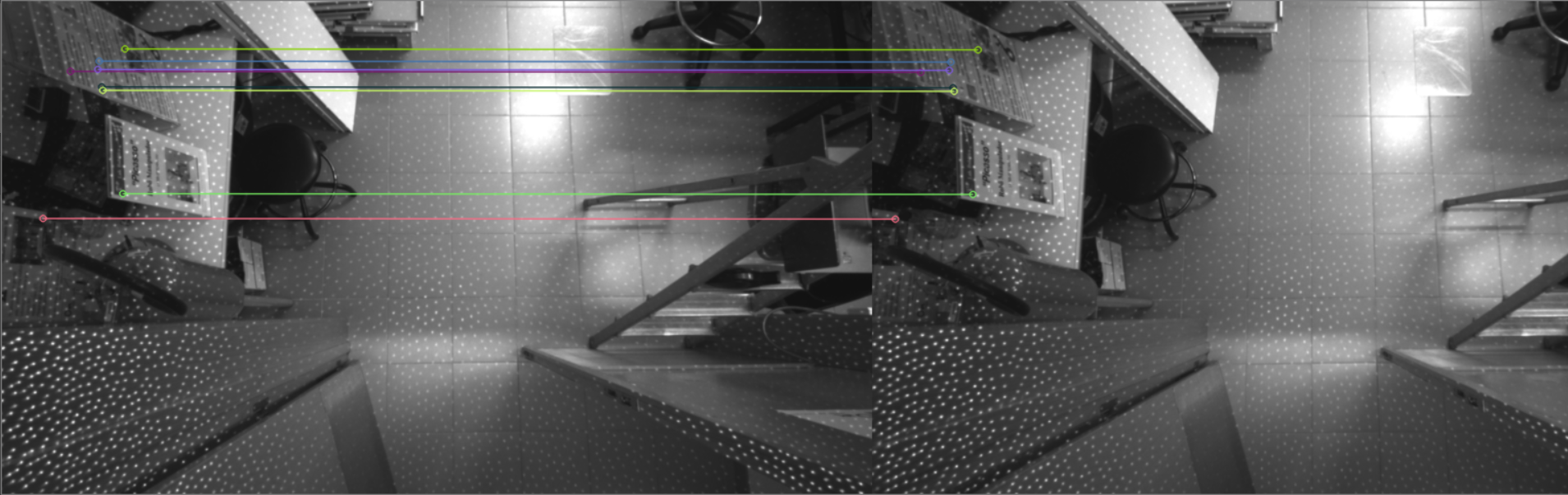
Implementação do teste de rácio proposto por David Lowe

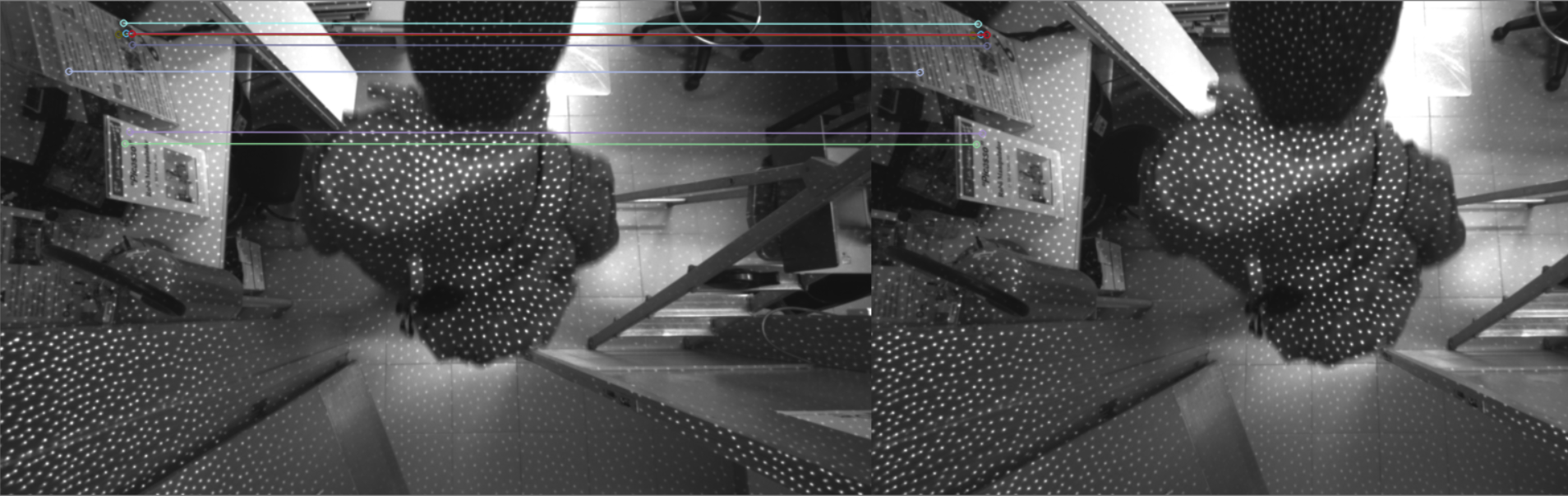
Verificou-se que após a implementação do teste de rácio os resultados continuavam a contar algum erro. Após alguma pesquisa verificámos que a melhor solução era usar apenas as melhores *N* correspondências, o que foi facilmente implementado pois a função *knnMatch* retorna as correspondências por ordem crescente de distância, ou seja, das melhores para as piores correspondências [Figura X]. Decidiu-se reter apenas as 8 melhores correspondências pois este é o número mínimo necessário para computar a matriz fundamental usando o algoritmo RANSAC.

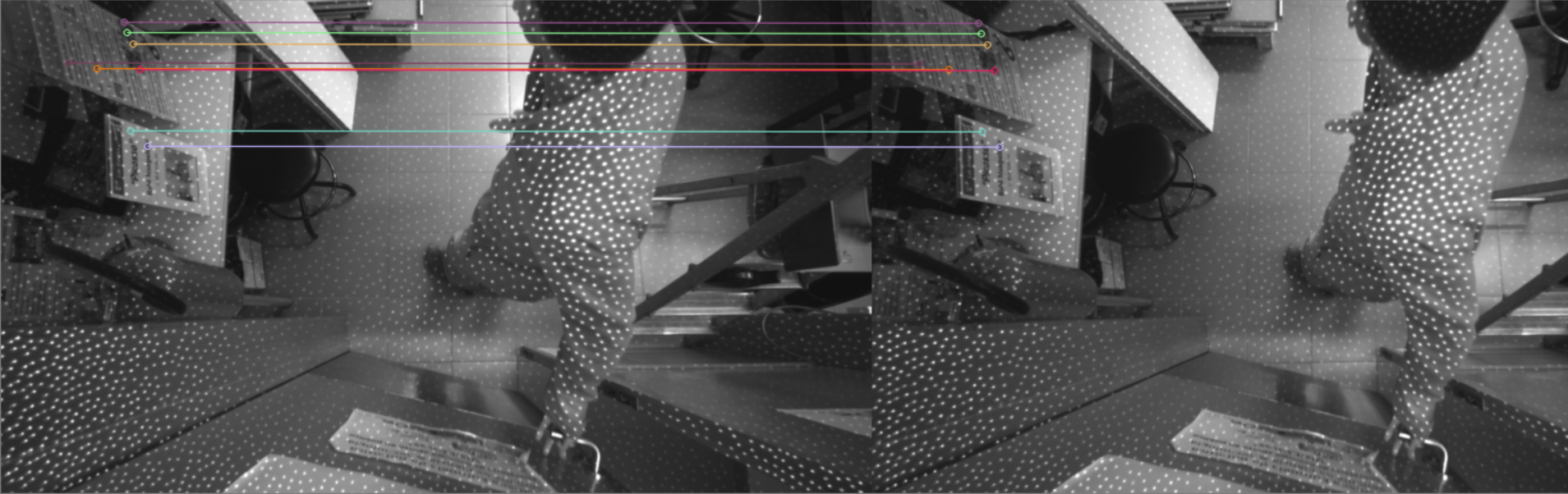


Retenção dos melhores N resultados/correspondencias

Estamos agora prontos para validar as correspondências obtidas usando a função *drawMatches* do *openCV* para visualizar as mesmas. Pode-se verificar que os pontos com melhores correspondência aparecem sempre associados a zonas de grande detalhe, neste caso, os cartazes afixados na parede esquerda[Figura X]. De notar que o uso do projetor IR para criar os pontos confere mais textura/detalhe à imagem o que melhora a deteção de *keypoints* e posteriormente de *matches*.



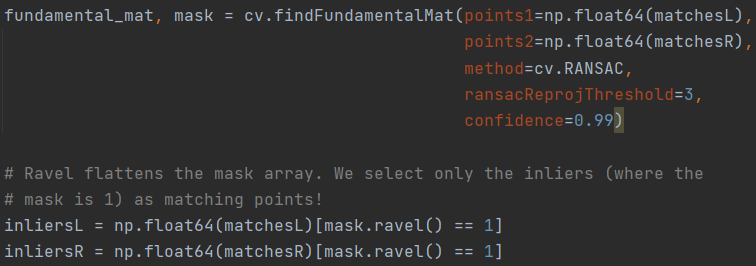




## Find Fundamental Matrix

A matriz fundamental estabelece a relação entre pontos correspondentes em duas imagens da mesma cena. Esta matriz é de extrema importância pois limita as correspondências entre pontos nas duas imagens à linha epipolar (*epipolar line*), podendo assim ser feita uma pesquisa por correspondências mais rápida e eliminando mais facilmente correspondências erradas (*outliers*).

O cálculo desta matriz foi feito com recurso à função *findFundamentalMat* do *openCV* com os parâmetros padrão. Esta função retorna-nos a matriz fundamental e uma máscara que usamos para excluir os *outliers* das correspondências.



Cálculo da matriz fundamental

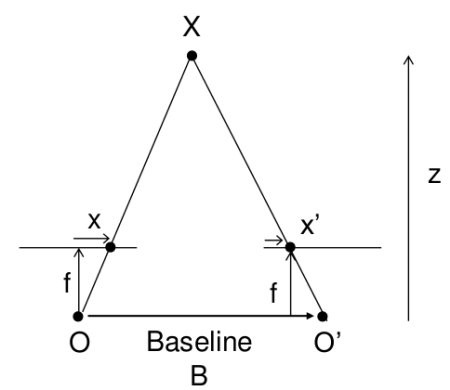
Após o cálculo da matriz fundamental e da exclusão dos *keypoints* falsos é calculado a matriz de transformação, ou homografia, que estabelece as transformações necessárias a efetuar numa imagem para que esta seja remapeada para um outro plano ou perspetiva.

TODO

## Rectify Images

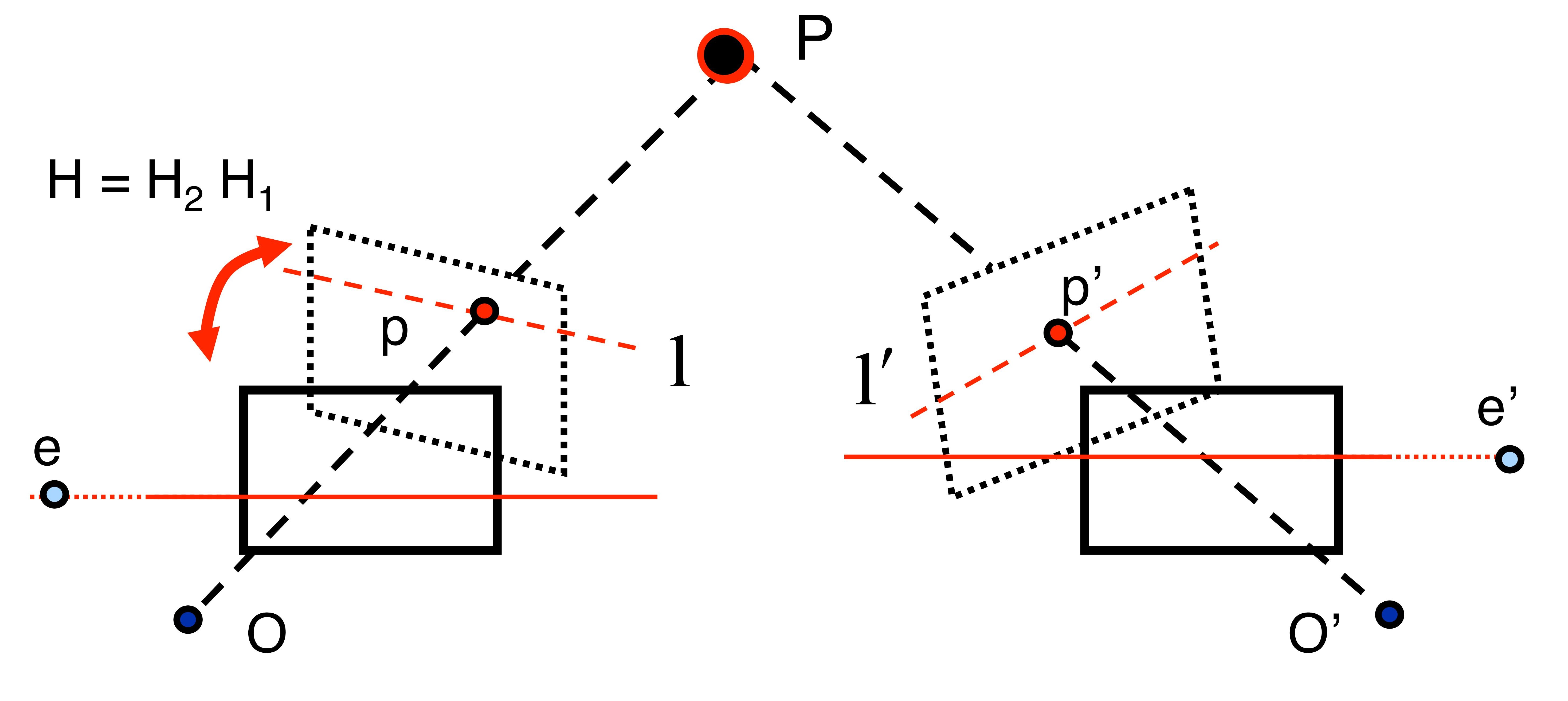
De modo a calcular a profundidade de duas imagens é necessário:

1. Conhecer a distância entre duas camaras paralelas (baseline);
2. Conhecer as propriedades das camaras, nomeadamente a distância focal (*focal length*);
3. Computar a disparidade entre as duas imagens (disparity).



Relações matemáticas da visão estereoscópica

Deste modo é pressuposto que as duas imagens sejam paralelas e por isso haja a necessidade da manipulação de ambas. Na realidade este processo é bastante difícil sendo que na prática é feita a projeção de ambas as imagens num plano virtual que é comum a ambas as imagens.



Retificação de imagens para um plano virtual comum

A retificação de imagem tem como objetivo fazer com que duas imagens com prespectivas diferentes sejam retificadas de modo a que a sua prespetiva seja a mesma.

TODO

## Find Disparity

<https://github.com/IntelRealSense/librealsense/blob/v2.29.0/doc/depth-from-stereo.md>

A disparidade é dada pela diferença de uma correspondência entre duas as imagens retificadas.

De modo a calcularmos a disparidade foi usada a função *compute* do *openCV* aplicada a um objeto do tipo *StereoMatcher*, que podem ser de dois tipos distintos:

1. *StereoBM*: Classe que computa correspondências *stereo* usando um algoritmo de *block matching*.
2. *StereoSGBM*: Classe que implementa o algoritmo de *H. Hirschmuller* modificado. Este algoritmo é também conhecido como *semi-global block matching*. Este algoritmo é adequado para implementações em tempo real.

A natureza do problema proposto é de execução em tempo real, deste modo foi usado um objeto do tipo *StereoSGBM* de modo a computar o mapa de disparidade.

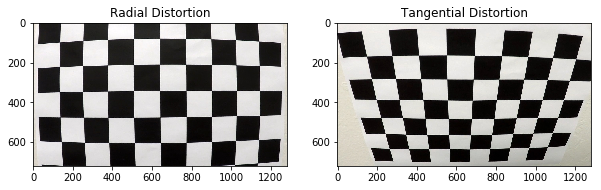
## Compute the Depth

# Tarefa 3 – Calibração da Camara

<https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html>

<https://en.wikipedia.org/wiki/Distortion_%28optics%29>

A distorção radial faz com que linhas retas na cena original sejam representadas como linhas curvas na imagem da cena, sendo que a distorção é maior quanto mais se afastamos do centro da lente. A distorção tangencial faz com que algumas zonas da imagem pareçam estar mais perto do que o esperado [Figura X].



A distorção pode ser caracterizada por um conjunto de equações [Eq. 1 e 2] em que os coeficientes de distorção , ,, e devem ser calculados.

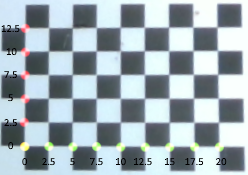
Além dos coeficientes de distorção é necessário conhecer os parâmetros intrínsecos e extrínsecos da camara.

Os parâmetros intrínsecos contêm características da camara como o comprimento/tamanho focal (e) e o centro ótico ( e ). Estes coeficientes são de extrema importância pois com eles podemos construir a *camera matrix* [Eq. 3] de modo a remover a distorção introduzida pelas lentes de uma camara. De referir que a *camera matrix* é única para todas as camaras sendo que podemos reutilizar a mesma sempre que capturamos novas imagens com a mesma camara.

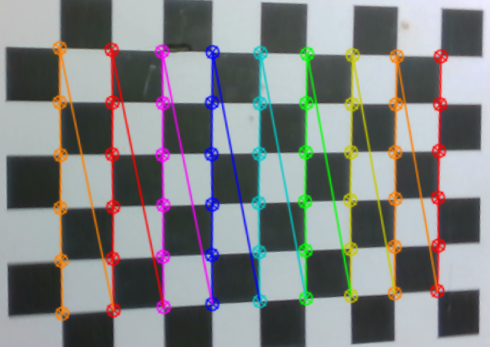
Os parâmetros extrínsecos descrevem a posição e direção da camara no mundo, sendo que estes são representados por uma matriz [Eq. 4] e compostos por uma matriz de rotação e um vetor de translação.

Com o intuito de resolver o problema de distorção é apresentado um padrão com geometria definida e conhecida à camara. Uma vez que a geometria do padrão é conhecida podemos relacionar os pontos reais com os pontos da imagem, usando as equações 1 e 2, de modo a obtermos os coeficientes de distorção. O mais comum é o uso de um padrão de xadrez para efetuar este procedimento.

A calibração da camara foi feita com recurso a um padrão de xadrez com 7 linhas e 10 colunas em que cada quadrado mede 2,5 cm de largura [Figura X]. A posição dos cantos internos do padrão de xadrez detetados pela função *findChessboardCorners* do *openCV*. Esta função, internamente, chama a função *cornerSubPix* de modo a melhorar a precisão da localização dos pontos, pelo que foi optou-se por não chamar a mesma explicitamente [Figura X].



Coordenadas reais dos cantos internos do padrão



Cantos internos encontrados pelo *openCV*

Finalmente é usada a função *calibrateCamera* do *openCV* que devolve, a matriz da camara, que descreve as suas propriedades intrínsecas, os coeficientes de distorção, e os vetores de rotação e translação que, em conjunto, associam as coordenadas de um ponto no mundo real para as coordenadas do ponto correspondente na imagem.

De modo a calibrar a camara foi então usado um padrão de xadrez em que cada quadrado mede 2,5 cm de comprimento. Deste modo, com o auxílio dos dados provenientes da camara de profundidade, é conhecida a posição real de cada ponto.