Extraction of 3D information from binocular images

Training course

G. Daubige – S. Thomas

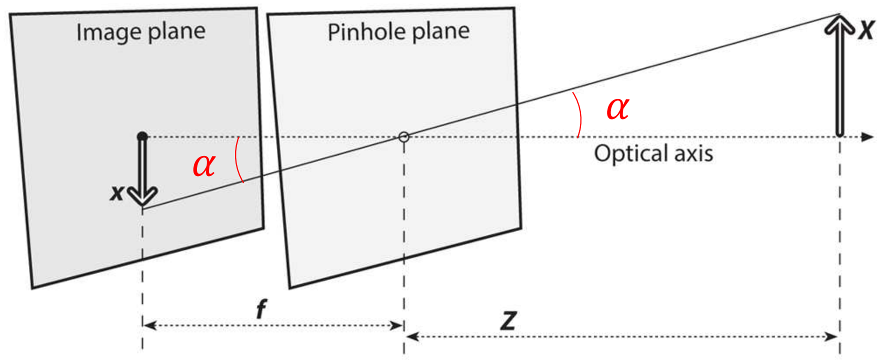
EPPS – 12/09/2025

Introduction

- Stereo imaging involves four steps, divided in to main 2 parts :
  - Part 1 : Calibration
    - Removing distorsions in the images from both cameras
    - Adjusting the positions and orientations between the two cameras, to obtain resulting coplanar image planes and collinear rows between the two images of the stereo pair
  - Part 2 : 3D reconstruction
    - Finding the correspondent pixels between the two images of the stereo pair, and building a disparity map
    - Reprojecting this map into a depth map by triangulation, knowing the geometry properties of the stereo camera (that come from the calibration).

Part1 – Step1 : Calibration of a single camera

- The pinhole camera model :



- Considering only absolute values, for the same angle α we have :

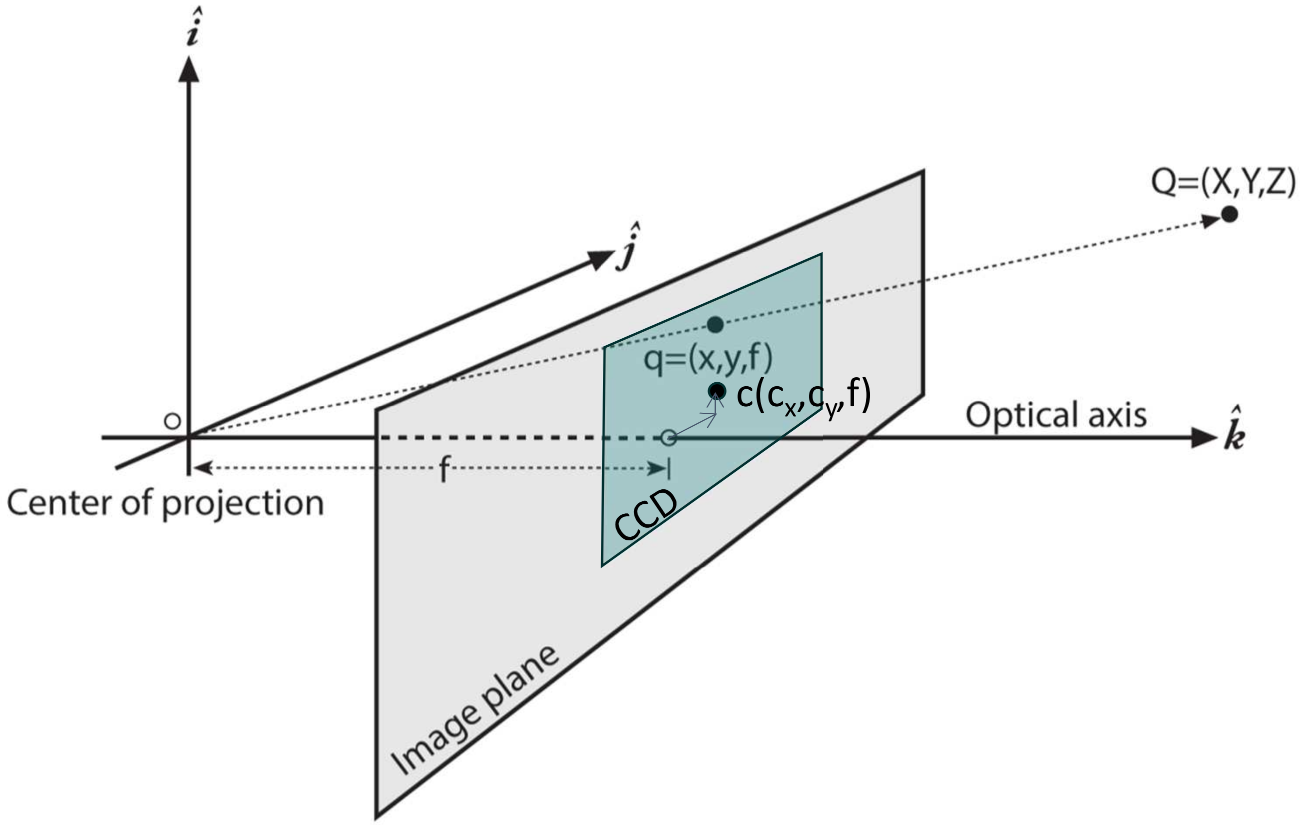
$$\tan\alpha = \frac{X}{Z} = \frac{x}{f}$$

- That leads to :

$$\boxed{x = f \times \frac{X}{Z}}$$

Where f is focal length of the camera

Intrinsics matrix



- Image plane brought in front of the pinhole
- Q(X,Y,Z) « real world » point projected on the imager
- q(x,y,f) projection of Q on the image plane
- The CCD imager is in the image plane, but its center c(c_x,c_y,f) is not exactly centered on the optical axis

- Coordinates of point q in the imager referential are :

$$\begin{cases} x_{CCD} = f_x \times \frac{X}{Z} + c_x \\ y_{CCD} = f_y \times \frac{Y}{Z} + c_y \end{cases}$$

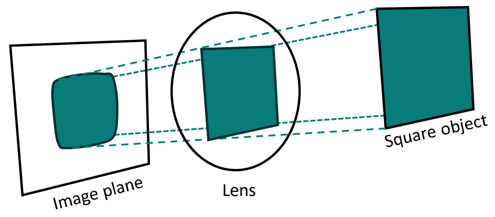
- Introducing a (3×3) matrix, this leads to :

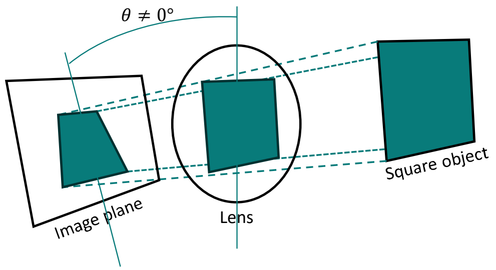
$$\underset{q}{\rightarrow}\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \times \underset{Q}{\rightarrow}\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

$M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ is the camera intrinsic matrix

Distorsion coefficients

Two main components :





- **Radial distorsion**, due to the lens : a square object in the real world appears to be rounded in the image. Correction can be « simply » modelled as the first terms of a Taylor serie around $r = 0$ (r : radial distance from the optical axis) :

$$\begin{cases} x_{corrected} = x \times (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{corrected} = y \times (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{cases}$$

- **Tangential distorsion**, due to manufacturing defaults (lens and CCD misalignment) :

$$\begin{cases} x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \\ y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \end{cases}$$

- This leads to a (5×1) distorsion vector :

$$\boxed{dist_{coefs} = [k_1, k_2, p_1, p_2, k_3]}$$

In practice

- Get a solid chessboard, with sufficient size and quality to make a correct calibration of a camera (you may found one on the Internet)

ex : ours is 14x9, with a cell size of 40 mm (/!\ : The OpenCV chessboard detection function works with internal corners (13x8 in our case)

- Acquire a sufficient set of chessboard images :

In the OpenCV documentation (OpenCV: Camera Calibration), a minimum set of 10 images is advised.

It's interesting to have more, (25 to 50 images), so that :

  - You can cover the entire range of distances to sensor you will need in your application
  - At a same distance, you may cover various orientations and positions of the chessboard in the image (especially the corners)

- Ensure the quality of the illumination and the good exposure of the images
    - Underexposed images lead to an increase in noise, whereas overexposed images lead to loss of the real information. In both cases, it may lead to poor calibration precision.

- Get the code at :

https://crop-phenotyping.labnotebook.ch/user/s.thomas@arvalis.fr/lab/tree/03_stereo_imaging

    - The code for performing your own calibration is contained in the notebook ‘calibration.ipynb’
    - You may execute the ‘First step – Part1’ and ‘First step – Part 2’ (for both left and right cameras) ; in this part, we call the ‘single_calibration’ function defined in ‘calibrationlib.py’, where states the implementation of the OpenCV functions for detecting the chessboard corners, and at the end, the ‘calibrateCamera’ function.

Part1 – Step2 : Calibration of a stereo camera pair

- Principle
  - General case

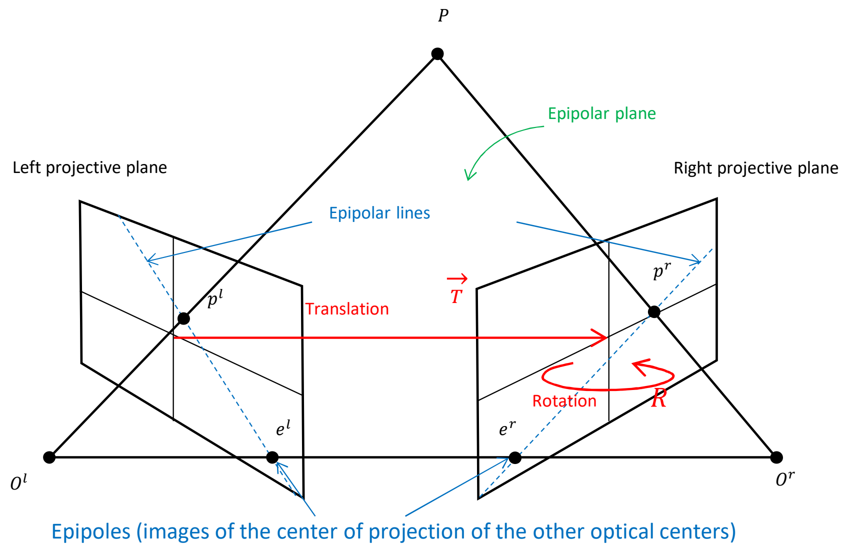
Considering an imaging system composed of two cameras, epipolar geometry tells us that :

- Every 3D point projections on the two cameras are contained in a same epipolar plane

- This plane intersects the image plane into an epipolar line

  => for a given p^l, p^r must lie along the corresponding epipolar line in the right imager : it's the epipolar constraint

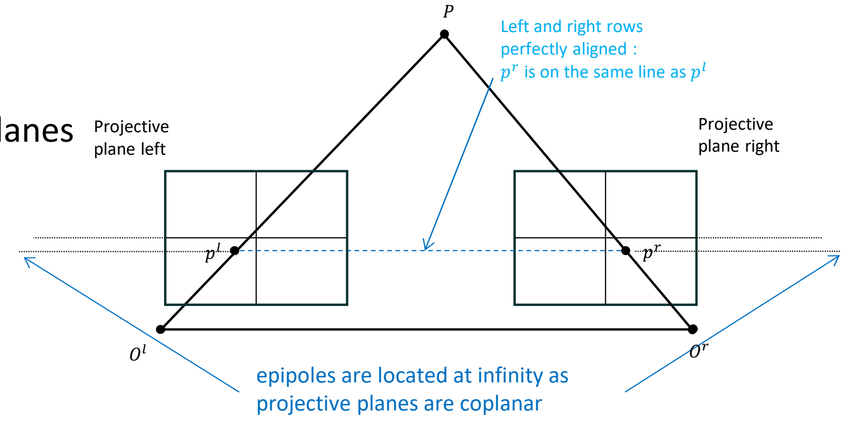
- Once the epipolar geometry of the stereo pair is known, the search of matching $p^l$and p^r in the two imagers becomes a 1D problem, along the epipolar lines



Ideal case

We consider two coplanar and row-aligned left and right image planes

- The projected pixels p^{left} and p^{right} on respectively the left and right image planes of the same point P in the real world are on the same line.



P
Left and right rows perfectly aligned : p^r is on the same line as p^l
Projective plane left
Projective plane right
p^l
p^r
O^l
O^r
epipoles are located at infinity as projective planes are coplanar

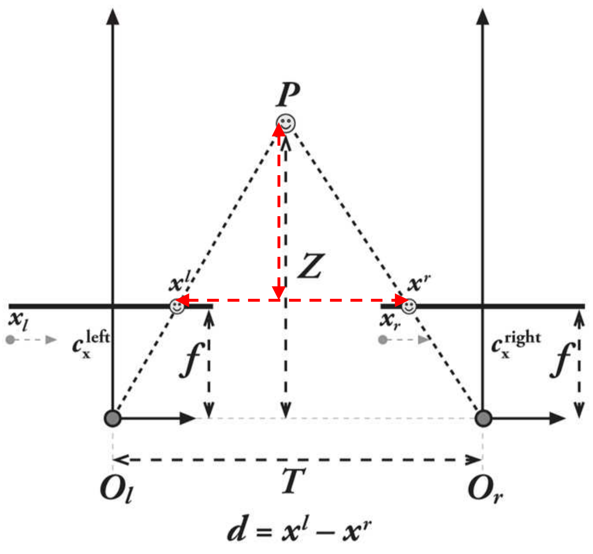
They have the following coordinates :

$$p^{left}\begin{cases}x^l\\y^l\end{cases}, \text{ and } \quad p^{right}\begin{cases}x^r = x^l - d\\ y^r = y^l\end{cases} \text{ where } d = x^l - x^r \text{ is called the 'disparity'}$$

- We can then retrieve the depth information, by the setting the following relation :

$$\boxed{\frac{T-(x^l-x^r)}{Z-f} = \frac{T}{Z} \Rightarrow Z = \frac{f \times T}{(x^l-x^r)}}$$

Where T is named the baseline of the stereo pair (the choice of T won't be discussed here but is an important point as it influences the range and resolution of the stereo imaging system)



How to proceed ?

As in real life, it's impossible that the two imaging systems of your stereo pair are perfectly aligned and naturally match the previous condition of :

- Coplanarity of image planes
- Collinearity of corresponding rows of the two sensors

- For coplanarity, we need to characterize the rotation matrix R and the translation vector T, which describe the location of the second camera (in OpenCV, it refers to the right one) relative to the first one (which refers to the left camera), in global coordinates.

This step is called 'stereo calibration'

- Once the image planes are coplanar (i.e. having found and applied R and T), it will then be possible to find the refined parameters that lead to the row-aligned images

This step is called 'stereo rectification'

At the end of this stage are generated some correction maps, that allow to map the coordinates of each pixel of the original image into its coordinates in the rectified image.

OpenCV proposes functions that implement both these operations (see in the 'documentation' directory for details)

In practice

- Get the code at https://crop-phenotyping.labnotebook.ch/user/s.thomas@arvalis.fr/lab/tree/03_stereo_imaging

  - The code for performing your own calibration is contained in the notebook 'calibration.ipynb'
  - You may execute the 'Second step : Calibration of the stereo pair' ; in this part, we call the 'stereo_calibration' function defined in 'calibrationlib.py', which implements the OpenCV function 'stereoCalibrate', from common image pairs selected during both left and right cameras.

  /!\ It may happen that for an image pair during the single camera calibration, one of the pair is successfully processed as the whole chessboard is seen on the image, while on the other camera, the chessboard is partially cropped and the imge finally not retained.

  In this case, for stereo calibration, it is important to reject the pair of corresponding image_points.

  It's mandatory that at the end, the list of the image_points for both left and right cameras are sorted so that they match the same source image pairs.

Part2 – Step1 : Generating and applying rectification maps to original images

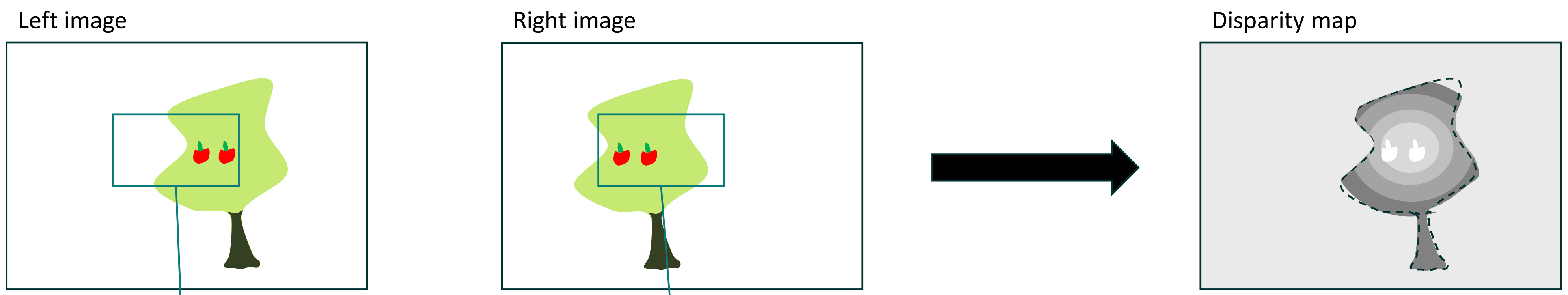
- Generating the rectification maps
  - This is a good idea to generate the rectification maps each time you want to make 3D reconstruction on a dataset, as these maps are heavy (2-channels floating point images, of the same size (height,width) as the original images : up to 60 Mb or more, depending on your camera). This avoids to store them on your disk, and is quite fast to be computed.
- Applying the rectification maps
  - This has to be done for each pair of images of your dataset, left images corrected from left maps, and right images from right maps.

Part2 – Step2 : Matching correspondence between left and right images

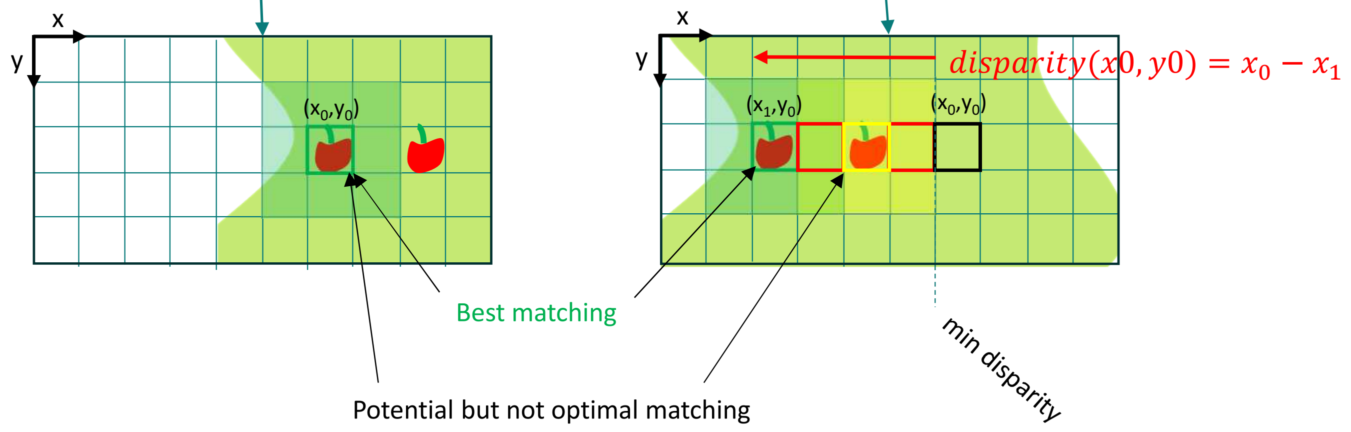
- Objective
  - At the end of this stage, the aim is to retrieve a disparity value for each pixel of the reference image (say the left one)
  - The expected disparity map is a 1-channel image of the same size as the left image, in which a pixel value encodes the distance in pixels of the counterpart right pixel on the same row
- Different approaches : geometric algorithms (BM, SGM, SGBM, …), deep learning models (RAFT, IGEV, FoundationStereo, S2M2, …)
  - Deep methods may give really good results where a correspondence can be found ; they also can give crazy values and tend to smooth the zones where big difference of disparities occur (still to be studied on our part)
  - OpenCV implements several algorithms with the geometric approach
- The OpenCV Semi Global Block Matching (SGBM) algorithm
  - Works on rectified left and right images, RGB or grayscale (RGB is more demanding on input ; quality but may be in this case more accurate)
  - Needs to predefine the minimal disparity and number of disparities we want to search for (i.e. the distance range of our objects in the image) => A good choice is important for relevance and computation time
  - Works by considering the neighborhood of each pixel in several directions (more accuracy/lower speed)

- (Highly) simplified operating principle

Iteration on each pixel of the left image



Search for the counterpart pixel in the right image


x
y
(x_0,y_0)
(x_1,y_0)
(x_0,y_0)
$disparity(x0,y0) = x_0 - x_1$
Best matching
Potential but not optimal matching
min disparity

- Possible issues :
  - Computation performance issues (image definition, too accurate parameters…)
  - Quality performance issues (bad disparity bounds, too poor parameters, too poor calibration…)
  - Occultations lead to no matching correspondence (due to the different point of view, the counterpart right pixel can be masked by an object)

    Good to know : in this case, the OpenCV SGBM algorithm sets the disparity value to min_disparity – 1

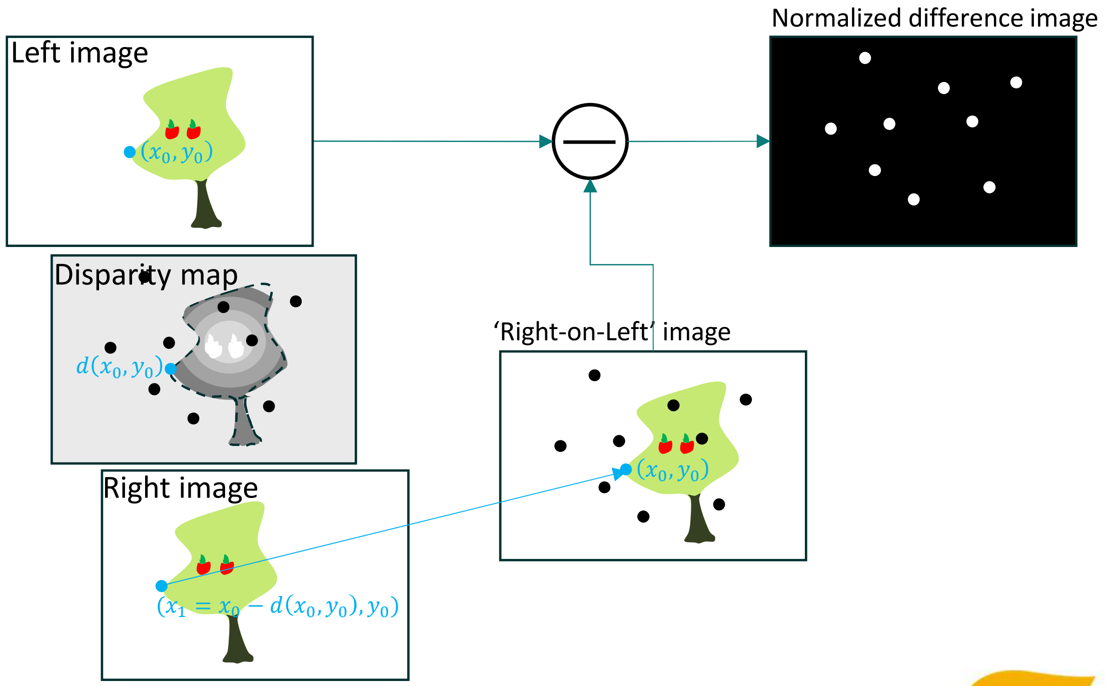
Part2 – Step3 : Evaluating the matching quality

- Normalized Total Gradient (NTG) : a good metric to evaluate the matching correspondence

The principle is to use the disparity map to pick pixels in the right image at the indices $(x_1 = x_0 - disparity(x_0, y_0), y_0)$ and to map them into a new image at the indices (x_0, y_0)

- This leads to a 'right-on-left' image, that we can compare (i.e. make the difference) with the original left image.

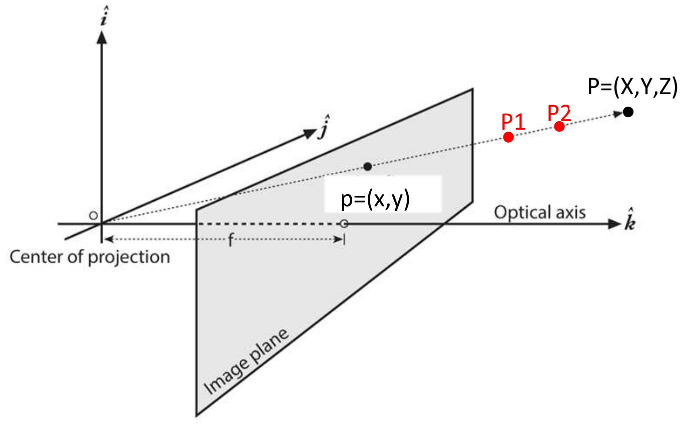
- The NTG is a unique normalized value that reports this difference : close to zero, it means that there are really few differences (= good matching quality); close to 1, it means that there are lots of differences (= bad matching quality)



Part2 - Step4 : Reprojecting the image pixels to 3D coordinates

- The knowledge of the disparity allows to project the pixels in the image on the 3D rectified camera coordinate system
  - We knew how to project a 3D point in the image 2D plane from the intrinsics matrix, but without the Z information, we were not able to perform the reverse operation :

    => The projected point might be P1, or P2, or anywhere else on the optical way
  - Associating the disparity(x,y) value (= the distance information) to a pixel p(x,y) allows to remove the ambiguity and to find its correct position P=(X,Y,Z)
  - OpenCV implements a single function, 'reprojectImageTo3D', which combines the Q reprojection matrix generated by 'stereoRectify', and the disparity map (see the OpenCV documentation for more details)
  - This leads to a 3-channel (X,Y,Z) image coregistered with the left rectified image, containing the coordinates of the 3D points relative to the RGB pixels, respectively along X, Y and Z (in mm, as during calibration we used this unit for scaling the chessboard cell size)



p(x,y)
+ *disparity*(x,y)

P(X,Y,Z)

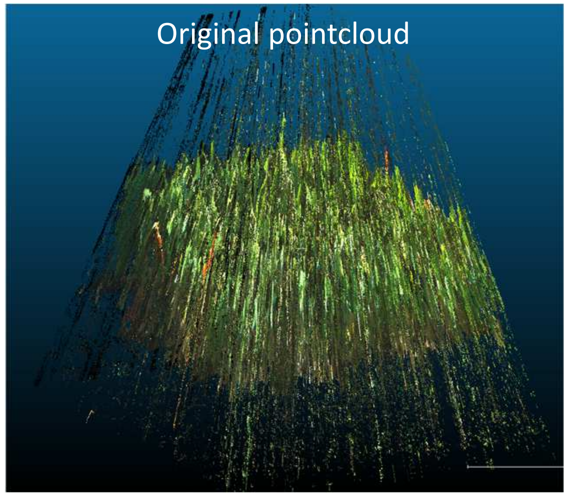
$$Q \cdot \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$$

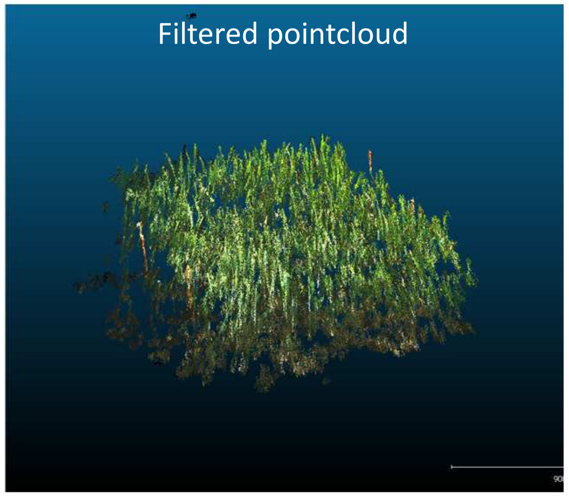
Reprojection matrix
from OpenCV 'stereoRectify'

Part2 - Step5 : Filtering the depth image by a 3D approach

- As disparity computation comes from 3D considerations, it does not make much sense to filter it as a standard image.
  - Using open3d, it's easy to put the (X,Y,Z) 3-channels image obtained at the previous step, together with the RGB pixel values, as they are all coregistered in the recitifed coordinate system of the left camera
  - Open3d gives access to several tools for filtering pointclouds, and the results are really satisfactory
  - Important : after such a filtering, the corresponding pixels positions of the remaining points in the rectified left coordinate system.

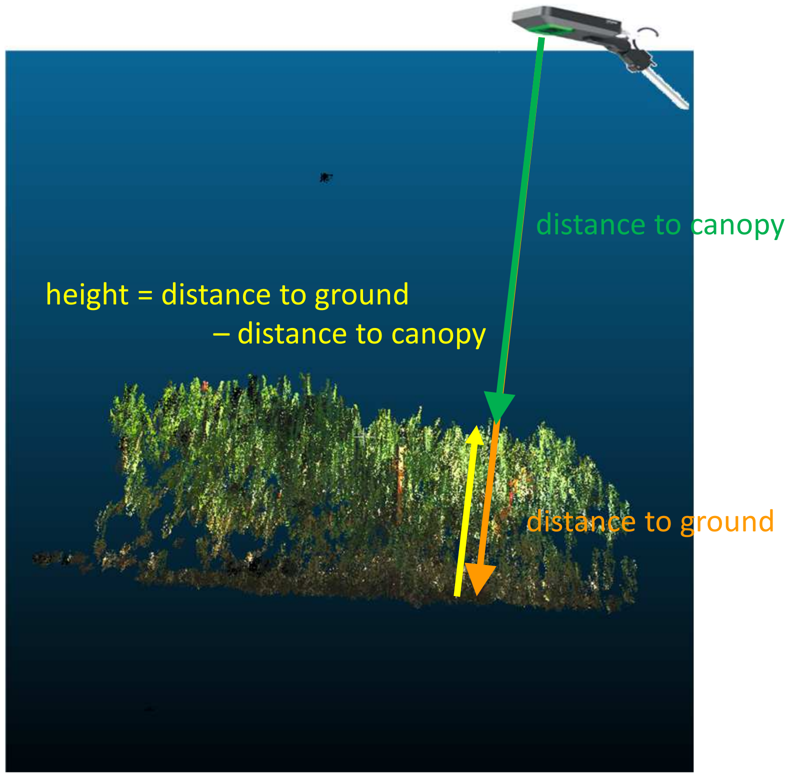
That's not an issue in our case, as further we will be only intersted in the Z values (= depth array from the stereo pair) of the filtered pointcloud





Part2 - Step6 : Estimating the mean height of the plants with a basic method

- The height of a plant is basically the distance between the ground and its top
  - By having access to the depth array, we can try to estimate the ground and the top of the canopy distances
  - By calculating the difference of these distances, we should retrieve the height of the plants



Discussion – important points

- Choice of an appropriate baseline between cameras
- Robustness of the cameras mounting system
- Perfect synchronization between the two cameras
- Importance of a reference image on the baresoil
- Working in patches rather than on the entire image
- Associating segmentation masks for instance, for refining the kind of objects searched

References

Open3D Paper (Zhou2018) : Qian-Yi Zhou and Jaesik Park and Vladlen Koltun. (2018)
Open3D : A Modern Library for 3D Data Processing
arXiv:1801.09847

NTG Paper : Chen, Shu-Jie & Shen, Hui-Liang & Li, Chunguang & Xin, John. (2017).
Normalized Total Gradient : A New Measure for Multispectral Image Registration.
IEEE Transactions on Image Processing. PP. 1-1. 10.1109/TIP.2017.2776753.

OpenCV documentation and tutorials :
https://docs.opencv.org/4.x/d4/d94/tutorial_camera_calibration.html

https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html

https://docs.opencv.org/4.x/d9/db7/tutorial_py_table_of_contents_calib3d.html

An interesting related topic :
https://learnopencv.com/depth-perception-using-stereo-camera-python-c/

Deep learning methods benchmark :
https://vision.middlebury.edu/stereo/eval3/