3D Vision and Extended Reality

Andrea Favero
andrea.favero96@gmail.com
Università degli Studi di Padova

Leonardo Monchieri
leonardo.monchieri@studenti.unipd.it
Università degli Studi di Padova

Contents

1. Perspective and cameras

Every image is created by the interaction of light, real objects and an optic device.

A camera acquires images from the real world following a precise pattern. A source of light emits rays that reach an object which, reflects them in many directions; a lens acquires these rays and focuses them on an array/plate of sensors. The output of the sensors is passed to some analogue electronics components that transform the light taken in input by the camera to a charge/tension and convert it into numerical digital values. These values are then stored into a memory after some processing.

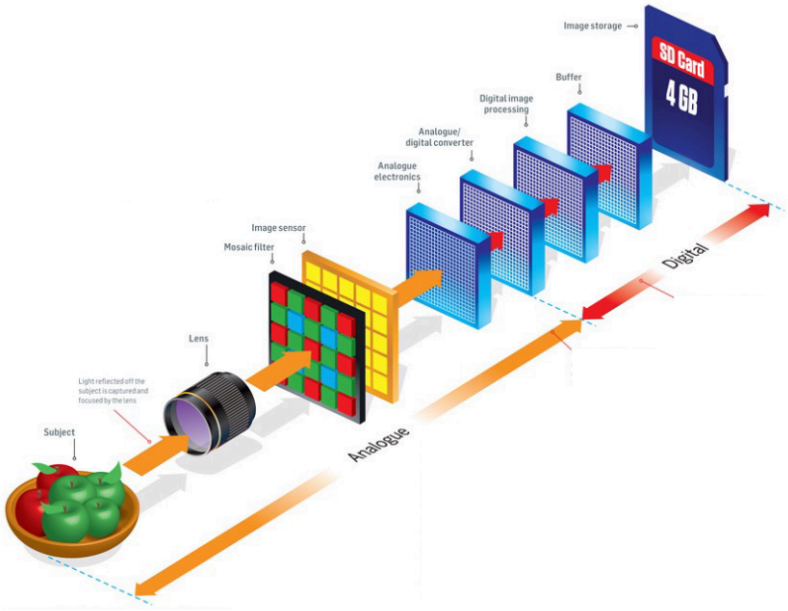


Figure 1: The camera's pipeline.

1.1. The simplified pinhole model

Rays travel from an illumination source to an object that reflects them, afterwards they are captured by an acquisition device (e.g.: our eye, an analog or a digital camera) that fucuses them on a photo responsive element (e.g.: the retina, a film, a matrix of sensors).

The *pinhole model* is a mathematically consistent model that is at the basis of computer vision and computer graphics.

The idea is that the acquisition element (e.g.: pupil, lens, ...) can be approximated to a *pinhole* or *center of projection* (COP) $\boldsymbol{C}$. The rays of light pass through the pinhole and are projected onto the *image plane* (also called the projection plane). The pinhole is so small that we assume that only a single ray of light for a specific direction goes into the hole.

The *optical axis* is the axis that passes through the pinhole and is orthogonal to the image plane. The *focal length* f is the distance from the pinhole to the image plane measured on the optical axis.

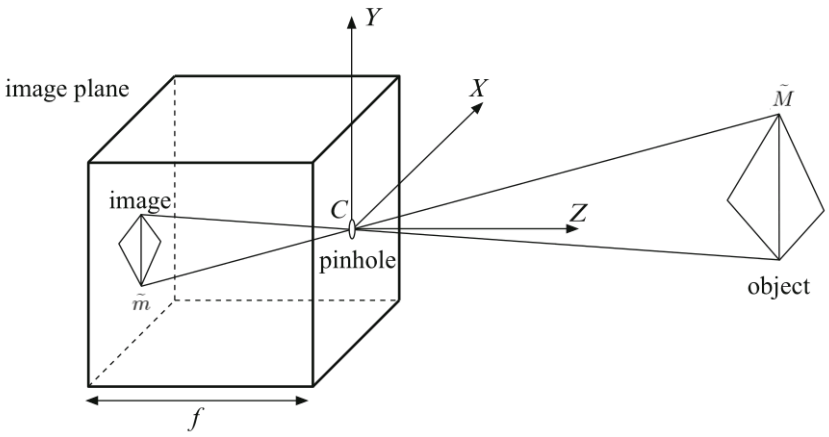


Figure 2: The pinhole camera model (from Fusiello's book).

Given a point $\widetilde{\boldsymbol{M}}$ in our 3D world, it is projected through the pinhole $\boldsymbol{C}$ onto the image plane. The intersection between the plane and the ray identifies the point $\widetilde{\boldsymbol{m}}$ in the plane. Capital bold letters are used for points in the 3D space and lower case bold letters are used for points in the 2D image plane.

To characterize the model from a geometric point of view, some references are required: we start with a 3D reference system $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ whose origin is located at the pinhole and a 2D reference system $\boldsymbol{u}, \boldsymbol{v}$ for the image plane, centered at the intersection between the image plane and the optical axis. The $\boldsymbol{Z}$'s direction is towards the 3D object (so it "points outside the camera"). This direction is just a convention and in some books you can find the opposite situation with $\boldsymbol{Z}$ that points towards the image plane.

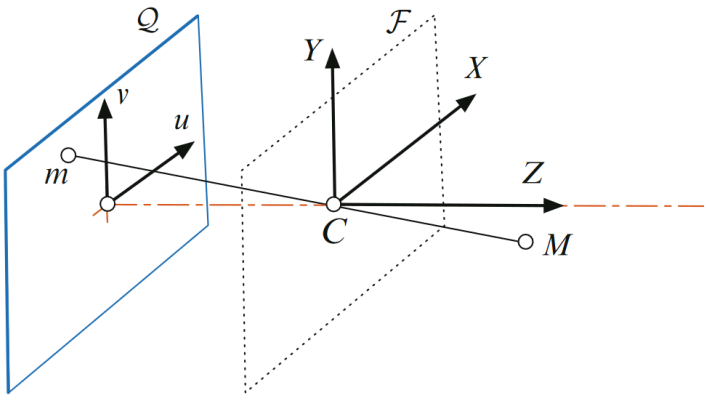


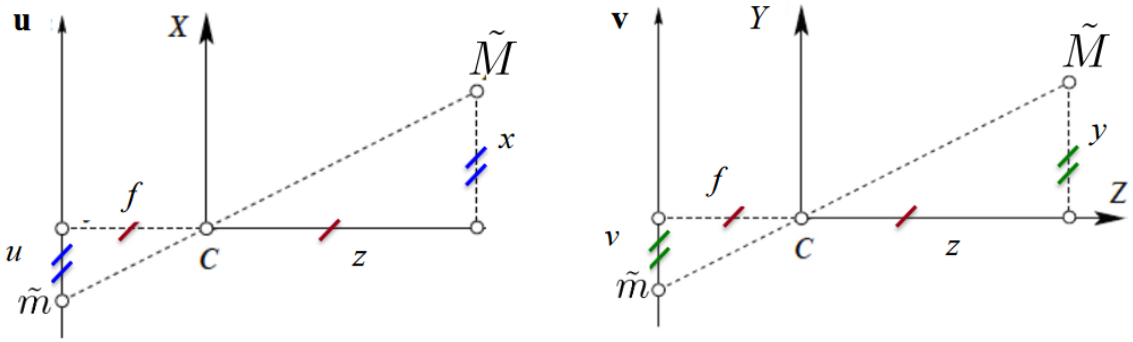
Figure 3: The pinhole camera model with reference systems from Fusiello's book. The orange dotted line is the optical axis. $\mathcal{Q}$ is the image plane, $\mathcal{F}$ is called the *focal plane* and it is a plane parallel to $\mathcal{Q}$ that contains the pinhole $\boldsymbol{C}$.

In real life, where we use digital cameras, an image is a matrix of discrete elements named *pixels*. Pixels information is obtained from a grid of CMOS sensors that measure the intensity of the light that hits them. The bigger is the grid, the bigger is the resolution of the camera and so of the image. The CMOS grid that we have in the real world has integer coordinates while, the pinhole camera model that exists in the ideal world has real coordinates. The coordinates of a point in a digital image are somehow quantized and can only be multiples of the sizes of the sensors.

When we refer to a digital picture we speak in terms of *pixels*, while when we refer to the pinhole model we speak in terms of *image points*.

1.2. Projective equations

For the sake of simplicity we consider two cases, one that involves the $\boldsymbol{X}$ and $\boldsymbol{Z}$ axes, and one that involves the $\boldsymbol{Y}$ and $\boldsymbol{Z}$ axes.



In both cases there is a ray that is projected from the 3D point $\widetilde{\boldsymbol{M}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$ onto the image plane at the point $\widetilde{\boldsymbol{m}} = \begin{bmatrix} u \\ v \end{bmatrix}$.

The triangle $\overline{\boldsymbol{C}\widetilde{\boldsymbol{M}}}, x, z$ is similar to the triangle $\overline{\boldsymbol{C}\widetilde{\boldsymbol{m}}}, u, f$ in the first case and to the triangle $\overline{\boldsymbol{C}\widetilde{\boldsymbol{m}}}, v, f$ in the second case.

From this fact it is clear that

$$\frac{f}{z} = \frac{-u}{x} = \frac{-v}{y}$$

where u and v are multiplied by -1 because they are located in the "negative side" of the $\boldsymbol{u}$ and $\boldsymbol{v}$ axes respectively, and we want them to be positive quantities.

From the equation above, the equations for the image points u and v can be derived.

$$\frac{f}{z} = \frac{-u}{x} \Rightarrow \frac{-f}{z} = \frac{u}{x} \Rightarrow u = -f\frac{x}{z}$$

$$\frac{f}{z} = \frac{-v}{y} \Rightarrow \frac{-f}{z} = \frac{v}{y} \Rightarrow v = -f\frac{y}{z}$$

and so the projection of a 3D point onto a point on the image plain is given by

$$\begin{cases} u = \frac{-f}{z}x \\ v = \frac{-f}{z}y \end{cases}$$

The fact that u and v are negative means that the object projected on the image plane is flipped both horizontaly and vertically.

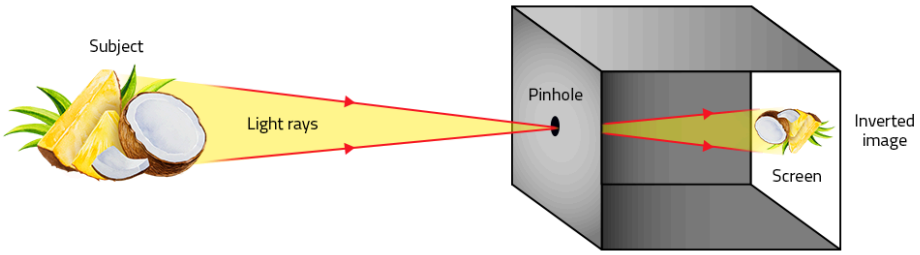


Figure 5: The image is flipped both horizontaly and vertically.

The derived equations tell us that the farther the point $\widetilde{\boldsymbol{M}}$ is, the smaller its projection will be on the image plane because, to find u and v we divide by z which is the distance of $\widetilde{\boldsymbol{M}}$ from the center of projection $\boldsymbol{C}$. The division by z accounts for the effect of *foreshortening* or *scorcio*.

Example: the spires of the Duomo di Milano have all the same height but, if we look at the ones in Figure 6, we notice that those which are far away appear smaller. This is the principle of *perspective*: the farther the objects are from the camera, the smaller they appear.

Figure 6: Duomo di Milano's spires.

1.3. Problems due to projection

It is impossible to measure the 3D world from a *single* image because the projection of a point from 3D to 2D is subject to a loss of information.

Projection means loss of:

- distances
- angles/vanishing points
- sizes

1.3.1. Loss of distances

Distances are lost due to projection. For example, if we have two coins of different size that lie on the same plane, we can notice they aren't the same.

Figure 7: Two coins of different size.

If you take the coins and place them properly in the 3D space, their projection on the 2D space will make them appear as they have the same size.

Figure 8: The same coins appear, as they have the same size.

There are also positions of the two coins whose projection makes the smaller one appear as the biggest.

1.3.2. Loss of angles

Example: rail tracks are parallel but, in Figure 9, they appear as converging to a point at the infinite.

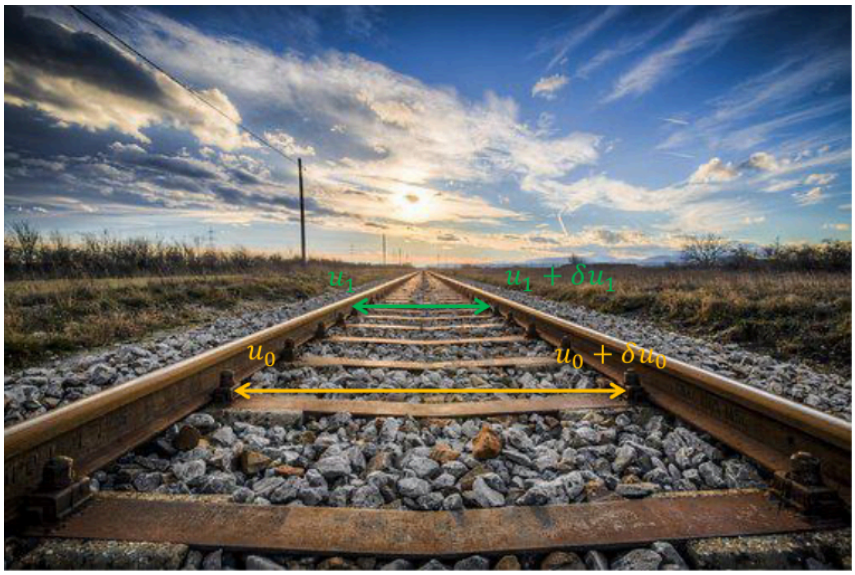


Figure 9: Loss of the angles.

Following the equations that we have obtained from the pinhole model, we can try to compute the coordinate in the $\boldsymbol{u}$ axis of the 2D points in Figure 9 (we can see the picture as the image plane whose projected objects are not flipped, so f is not multiplied by -1 in this case).

For the two points at the extrema of the yellow line, we have that

$$u_0 = f\frac{x_0}{z_0} \qquad u_0 + \delta u_0 = f\frac{x_0 + \Delta}{z_0}$$

while, for the two points at the extrema of the green line we have that

$$u_1 = f\frac{x_1}{z_1} \qquad u_1 + \delta u_1 = f\frac{x_1 + \Delta}{z_1}$$

The rails are parallel and so the distance between them is the same quantity Δ. The points on the yellow line are at distance z_0 from the center of projection, while the one on the green line, which are far away, are at distance z_1 where $z_0 < z_1$.

$$\delta u_0 = f\frac{x_0 + \Delta}{z_0} - u_0 = f\frac{x_0 + \Delta}{z_0} - f\frac{x_0}{z_0} = f\frac{\Delta}{z_0}$$

$$\delta u_1 = f\frac{x_1 + \Delta}{z_1} - u_1 = f\frac{x_1 + \Delta}{z_1} - f\frac{x_1}{z_1} = f\frac{\Delta}{z_1}$$

$z_0 < z_1 \Rightarrow \delta u_1 < \delta u_0$ which means that the projected rails are not parallel, otherwise we would have that $\delta u_0 = \delta u_1$.

1.3.3. Loss of sizes

As shown by the Figure 10, the projection is also responsible for the size loss of the objects.

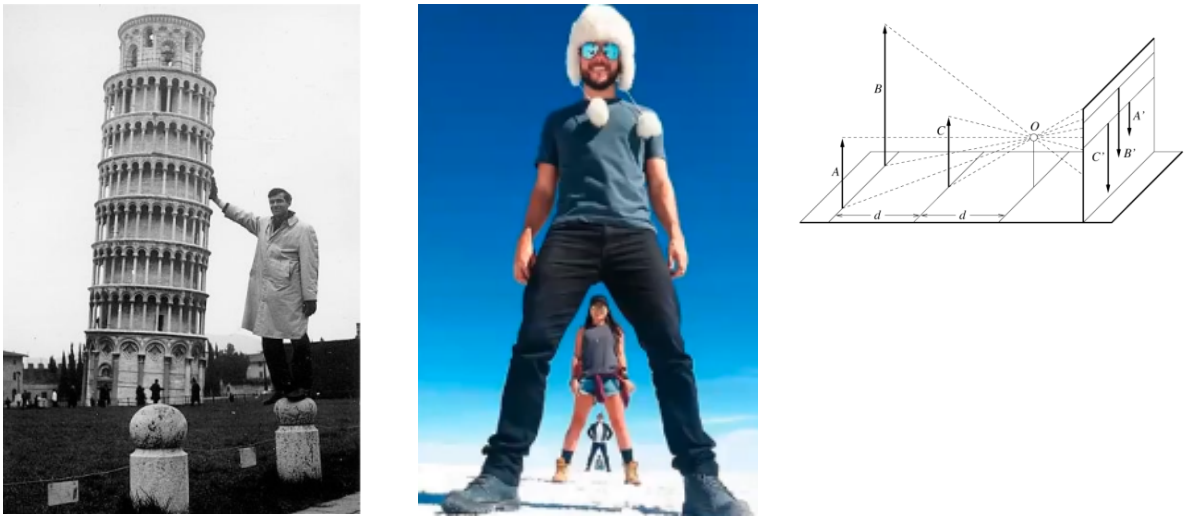


Figure 10: Loss of sizes.

1.4. Modify the simple pinhole model

Since the pinhole model is a mathematical model, it is possible to move the image plane between the center of projection and the world without loss of generality. In this way, the object's projection will not be flipped anymore.

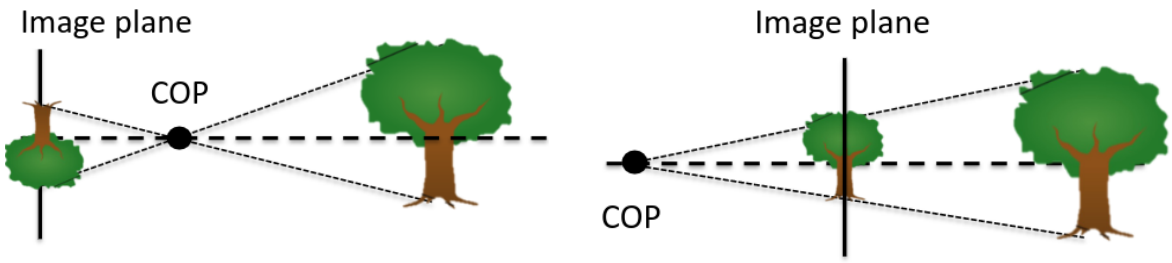


Figure 11: On the left: the version of the simple pinhole model with the image plane at the left of the COP. On the right: the version of the simple pinhole model with the image plane moved between the COP and the world.

1.5. Perspective projection of a plane and the projective space

A *perspective projection* is the mapping of 3D points $\boldsymbol{M}$ into 2D points $\boldsymbol{m}$ of the plane $\mathcal{Q}$, by intersecting the line passing from $\boldsymbol{M}$ and $\boldsymbol{C}$ with $\mathcal{Q}$. The points $\boldsymbol{M}$ are located on a ground plane $\mathcal{G}$ that is orthogonal to $\mathcal{Q}$.

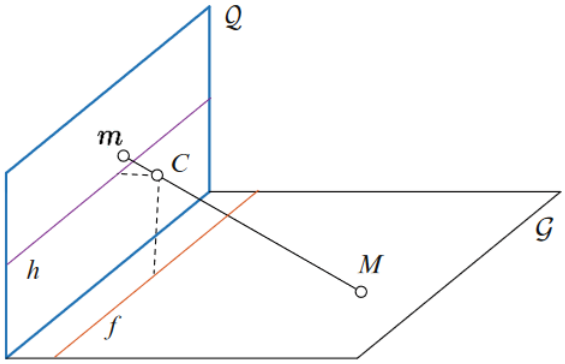


Figure 12: Perspective projection.

As Figure 12 suggest, the line f, determined by the intersection of the plane $\mathcal{G}$ with the plane parallel to $\mathcal{Q}$ and containing $\boldsymbol{C}$ does not project onto $\mathcal{Q}$ and, the line h, intersection of $\mathcal{Q}$ with the plane parallel to $\mathcal{G}$ and passing through $\boldsymbol{C}$ is not the projection of any line on the plane $\mathcal{G}$.

If we look at the perspective projection from the "side", we obtain a 2D representation like the one in Figure 13 where, the vertical line corresponds to $\mathcal{Q}$, and the horizontal line corresponds to $\mathcal{G}$.

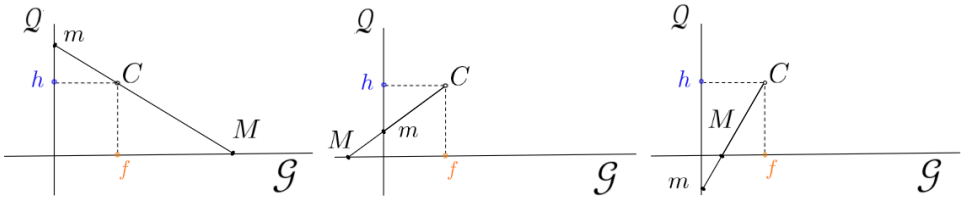


Figure 13: Perspective projection in 2D.

If $\boldsymbol{M}$ is moved towards $+\infty$ ($\boldsymbol{M} \to +\infty$) it is possible to notice that the point projected onto $\mathcal{Q}$ converges to h ($\boldsymbol{m} \to h$). Instead, if $\boldsymbol{M} \to f$ then $\boldsymbol{m} \to +\infty$.

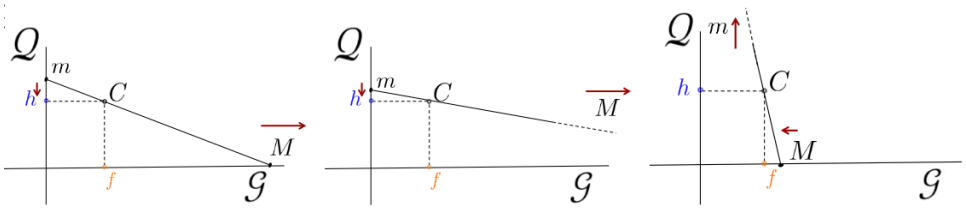


Figure 14: Perspective projection in 2D with points $\boldsymbol{M} \to \infty$ and $\boldsymbol{M} \to f$.

So, if we want our model to work correctly, we should add to the Euclidean planes, "ideal" lines lying at the infinite.

These new planes are called *projective planes*. The projective plane for the Figure 14 is

$$\mathbb{P}^2 \stackrel{\text{def}}{=} \mathbb{R}^2 \cup \{l_\infty\} \text{ where } l_\infty \text{ is the line at the infinite}$$

This concept is generalizable for $\mathbb{P}^n$ (e.g.: $\mathbb{P}^3 \stackrel{\text{def}}{=} \mathbb{R}^3 \cup \{p_\infty\}$ where p_∞ is a plane at the infinite).

In the Cartesian plane:

- Given two different points, there exists only one line that contains them both;
- There exists only one line with a given direction and containing a given point $\boldsymbol{P}$;
- Two different lines have either a common point (incident) **or** the same direction (parallel).

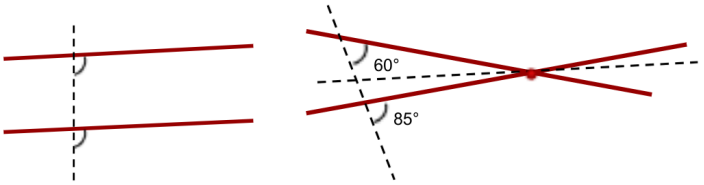


Figure 15: On the left: two parallel lines. On the right: two incident lines..

Let's consider two incident lines on a point $\boldsymbol{P}$. If $\boldsymbol{P}$ is drawn at the infinite the two lines become paralel.

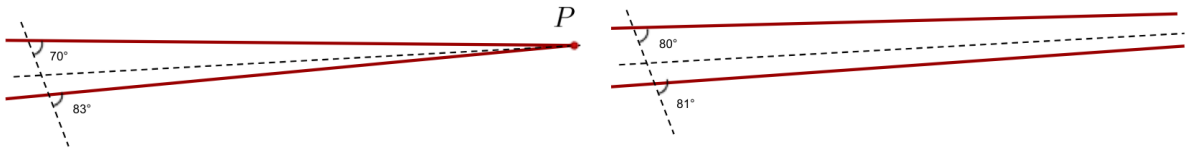


Figure 16: On the left: Two lines incident on a point P. On the right: The two lines becomes parallel when P is drawn at infinite.

In a projective space:

1. Given two points, there exists only one line that contains them both.
2. Two different lines **have** only **one common point**

These two rules identify a line point dualism and so, in projective spaces even parallel lines share a common point (which is at infinite).

To insert projective spaces in our model we need to model them analitically and we do so using homogeneus coordinates.

1.6. Homogeneus coordinates

If we are in the Cartesian plane $\mathbb{R}^2$ (or in the Euclidean space $\mathbb{R}^3$, ...), lines can be described by their standard equation form.

Two lines in $\mathbb{R}^2$ for example, are described by:

$$\begin{aligned} ax + by + c &= 0 \\ a'x + b'y + c' &= 0 \end{aligned}$$

To see if the two lines intersect, we can put them in a linear system. If the system has one unique solution the lines intersect, if it has infinite solutions the lines overlap, otherwise, if it has no solution at all, the lines are parallel.

The system

$$\begin{cases} ax + by + c = 0 \\ a'x + b'y + c' = 0 \end{cases} \iff \begin{cases} ax + by = -c \\ a'x + b'y = -c' \end{cases}$$

can be rewritten as

$$\begin{bmatrix} a & b \\ a' & b' \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -c \\ -c' \end{bmatrix}$$

Defining $A \stackrel{\text{def}}{=} \begin{bmatrix} a & b \\ a' & b' \end{bmatrix}$, it corresponds to:

$$A \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -c \\ -c' \end{bmatrix}$$

Cramer's theorem

Given a system of linear equations $A\boldsymbol{x} = \boldsymbol{b}$ (where A is an $n \times n$ matrix), if A is invertible ($\det(A) \neq 0$), then the system has a unique solution and $\forall x_i$ with $i = 1, ..., n$

$$x_i = \frac{1}{\det(A)} \Delta_i$$

where Δ_i is the determinant of the matrix that has the same columns of A except for the i-th one that, is equal to $\boldsymbol{b}$.

If the two lines intersect, the system has a unique solution $\Rightarrow$ the matrix A is invertible ($\det(A) \neq 0$) $\Rightarrow$ Cramer can be used to solve the system.

$$x = \frac{1}{\det(A)} \Delta_1 = \frac{\det\left(\begin{bmatrix} -c & b \\ -c' & b' \end{bmatrix}\right)}{\det\left(\begin{bmatrix} a & b \\ a' & b' \end{bmatrix}\right)} = \frac{bc' - b'c}{ab' - a'b} \stackrel{\text{def}}{=} \frac{u}{w}$$

$$y = \frac{1}{\det(A)} \Delta_2 = \frac{\det\left(\begin{bmatrix} a & -c \\ a' & -c' \end{bmatrix}\right)}{\det\left(\begin{bmatrix} a & b \\ a' & b' \end{bmatrix}\right)} = \frac{a'c - ac'}{ab' - a'b} \stackrel{\text{def}}{=} \frac{v}{w}$$

- If $\det(A) = w \neq 0$ then A is invertible and so the system has just one solution which is the point of intersection of the two lines $(x = u/w, y = v/w)$, and it belongs to $\mathbb{R}^2$;
- if $\det(A) = w = 0$ the matrix is not invertible, and so the system doesn't have a unique solution:
  - it could have infinite solutions ($u = v = 0$), which means that the lines overlap;
  - it could not have any solution at all in $\mathbb{R}^2$ ($u \neq 0$ or $v \neq 0$), which means that the lines are parallel. In this case, the intersection point **lies at the infinite** and its coordinates can be expressed in the projective space $\mathbb{P}^2$ with a triple $\begin{bmatrix} u \\ v \\ w \end{bmatrix}$ where $w = 0$.

The representation

$$\boldsymbol{m} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

where $\boldsymbol{m} \in \mathbb{P}^2$ is called *homogeneous coordinate*.

If $\gamma = 0$ then $\boldsymbol{m}$ lies at the infinite and is called an *ideal point*.

Points in cartesian coordinates are represented in bold with a tilde $\sim$ like $\widetilde{\boldsymbol{m}}$, while points in homogeneous coordinates are just in bold like $\boldsymbol{m}$.

1.7. Points in homogeneous coordinates and their relation with the Cartesian plane

As consequence of the defintion of homogeneous coordinates, we can represent points into the *projective space* $\mathbb{P}^2$ as a triplet:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} \neq \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The triplet $\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ corresponds to two lines that are coincident and so, it represents an infinite set of points and not a single point, therefore it doesn't belong to $\mathbb{P}^2$.

It is possible to convert homogeneous coordinates into Cartesian coordinates due to the fact that:

$$x = \frac{u}{w} \qquad y = \frac{v}{w}$$

and so:

$$\mathbb{P}^2 \ni \begin{bmatrix} u \\ v \\ w \end{bmatrix} \xrightarrow{w \neq 0} \begin{bmatrix} \frac{u}{w} \\ \frac{v}{w} \end{bmatrix} \in \mathbb{R}^2$$

where $w \neq 0$ because we can find a correspondent point in $\mathbb{R}^2$ only for real points, not the ones that lie at the infinite.

It is possible to do the inverse conversion from Cartesian coordinates to homogeneous coordinates and, it is even more simple:

$$\mathbb{P}^2 \ni \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \longleftarrow \begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{R}^2$$

It is interesting to notice that in projective spaces, the multiplication by a scalar constant $\lambda \neq 0$ in $\mathbb{P}^2$ doesn't change the corresponding point in $\mathbb{R}^2$. All the homogeneous coordinate vectors are defined with respect to a scale factor $\lambda \neq 0$. So $\boldsymbol{m}$ and $\lambda \boldsymbol{m}$ are the same point in $\mathbb{R}^2$

$\forall \lambda \neq 0$:

$$\lambda \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \lambda u \\ \lambda v \\ \lambda w \end{bmatrix} \in \mathbb{P}^2 \longrightarrow \begin{bmatrix} \frac{\lambda u}{\lambda w} \\ \frac{\lambda v}{\lambda w} \end{bmatrix} = \begin{bmatrix} \frac{u}{w} \\ \frac{v}{w} \end{bmatrix} \in \mathbb{R}^2$$

All these correspondences are generalizable to $\mathbb{P}^n$ and $\mathbb{R}^n$.

1.8. Lines in homogeneous coordinates

Similar to points, in $\mathbb{P}^2$ we can represent lines just using a vector of three real numbers. Like in $\mathbb{R}^2$ where a line $(ax + by + c = 0)$ has three real coefficients, we can represent a line in $\mathbb{P}^2$ with the equation:

$$a\frac{u}{w} + b\frac{v}{w} + c = 0$$

multiplying by w we obtain:

$$au + bv + cw = 0$$

using vector notation, it becomes:

$$[a \;\; b \;\; c] \begin{bmatrix} u \\ v \\ w \end{bmatrix} = 0$$

Posing $\boldsymbol{l}^T \stackrel{\text{def}}{=} [a \;\; b \;\; c]$ and $\boldsymbol{p} \stackrel{\text{def}}{=} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$ the equation can be rewritten as:

$$\boldsymbol{l}^T \boldsymbol{p} = 0$$

Recap with a visual scheme:

$$\begin{array}{ccc} a\frac{u}{w} + b\frac{v}{w} + c = 0 \text{ (in } \mathbb{P}^2) & \Longleftrightarrow & ax + by + c = 0 \text{ (in } \mathbb{R}^2) \\ \Updownarrow & & \\ au + bv + cw = 0 & & \\ \Updownarrow & & \\ [a \;\; b \;\; c] \begin{bmatrix} u \\ v \\ w \end{bmatrix} = 0 & & \\ \Updownarrow & & \\ \boldsymbol{l}^T \boldsymbol{p} = 0 & & \end{array}$$

1.9. Transformations

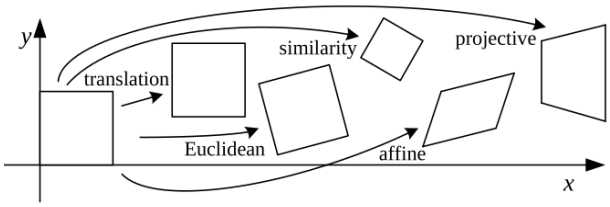


Figure 17: Transforms on $\mathbb{P}^2$ from Szeliski's book

1.9.1. Projective transformations (projectivities)

Projectivities are *linear functions* that map points in $\mathbb{P}^n$ to points in $\mathbb{P}^n$. They are defined by an *invertible* matrix H of order $(n+1) \times (n+1)$ called *homography*.

$$\begin{aligned} f : \mathbb{P}^n &\to \mathbb{P}^n \\ \boldsymbol{m} &\mapsto H\boldsymbol{m} \end{aligned}$$

- the fact that H is invertible implies that if the points $\boldsymbol{m}$ taken in input lies on the same plane, then also the points $H\boldsymbol{m}$ outputted by f lies on the same plane too (collinearity);
- projectivities form a group $f \in \mathcal{G}_P$;
- H and $\lambda H (\lambda \in \mathbb{R}, \lambda \neq 0)$ are the same (similarity).

1.9.2. Affine transformations

An affine transformation is a projectivity that maps real points into real points and, ideal points into ideal points (because of the last row $[0^T \;\; 1]$ of H that, when multiplied by $\boldsymbol{m}$, makes 1 the last element of the resulting vector $H\boldsymbol{m}$ if 1 is the last element of $\boldsymbol{m}$, 0 if 0 is the last element of $\boldsymbol{m}$).

$$H = \begin{bmatrix} A_{n \times n} & \boldsymbol{b} \\ \boldsymbol{0}^T & 1 \end{bmatrix}$$

An affine transform preserves the *parallelism* but doesn't preserve the angles.

In $\mathbb{P}^2$, H is 3×3 and has 6 degrees of freedom because the last row is fixed to $[0 \ 0 \ 1]$.

$$H = \begin{bmatrix} a_{1,1} & a_{1,2} & b_1 \\ a_{2,1} & a_{2,2} & b_2 \\ 0 & 0 & 1 \end{bmatrix}$$

1.9.3. Similarity

A similarity is a subclass of affinities that use an orthogonal rotation matrix R with a scale factor s and a traslation vector $\boldsymbol{t}$.

$$H = \begin{bmatrix} sR_{n\times n} & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

A similarity preserves the angles and, being an affine transform, preserves also the parallelism.

In the Euclidean space $\mathbb{R}^n$, similarity operates as

$$\widetilde{\boldsymbol{m}} \rightarrow sR_{n\times n}\widetilde{\boldsymbol{m}} + \boldsymbol{t}$$

In $\mathbb{R}^3$ for example:

$$H\boldsymbol{m} = \begin{bmatrix} sR_{3\times 3} & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} sR_{3\times 3} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \boldsymbol{t} \\ 1 \end{bmatrix} \Rightarrow sR_{3\times 3} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \boldsymbol{t} = sR_{3\times 3}\widetilde{m} + \boldsymbol{t}$$

In $\mathbb{P}^2$, H has 4 degrees of freedom.

1.9.4. Euclidean transformations

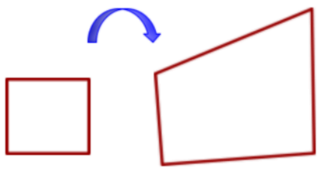
If in a similarity $s = 1$, the transform is called *rigid* transformation or *Euclidian* transformation.

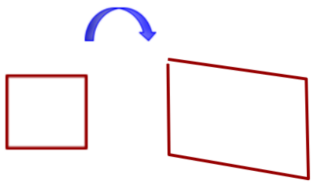
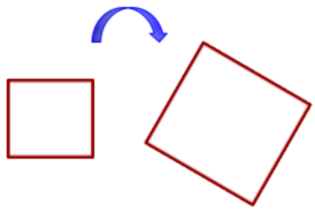
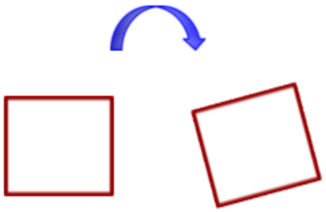
$$H = \begin{bmatrix} R_{n\times n} & t \\ 0^T & 1 \end{bmatrix}$$

A Euclidean transformation, preserves the distances, the lengths and, just like similarity, also the angles and the parallelism.

s is fixed, so in $\mathbb{P}^2$, H has 3 degrees of freedom.

1.9.5. Tranformations in $\mathbb{P}^2$, recap:

Transformation	D.o.f.	Matrix	Distortion	Preserves
Projectivity	8	$\begin{bmatrix} H_{1,1} & H_{1,2} & H_{1,3} \\ H_{2,1} & H_{2,2} & H_{2,3} \\ H_{3,1} & H_{3,2} & H_{3,3} \end{bmatrix}$		Collinearity

Transformation	D.o.f.	Matrix	Distortion	Preserves
Affinity	6	$\begin{bmatrix} H_{1,1} & H_{1,2} & H_{1,3} \\ H_{2,1} & H_{2,2} & H_{2,3} \\ 0 & 0 & 1 \end{bmatrix}$		Parallelism
Similarity	4	$\begin{bmatrix} sR_{n\times n} & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$		Angles
Euclidean	3	$\begin{bmatrix} R_{n\times n} & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$		Distances and lengths

1.10. Camera matrix

If we look at the projective equations in cartesian coordinates that we have derived before:

$$\begin{cases} u = \frac{-f}{z}x \\ v = \frac{-f}{z}y \end{cases}$$

we can see that they are *not linear* because x and y are multiplied by $1/z$.

The equations, can be rewritten *linearly* in matrix form, using homogeneous coordinates:

$$\begin{cases} zu = -fx \\ zv = -fy \\ z1 = z \end{cases} \iff z\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} -fx \\ -fy \\ z \end{bmatrix}$$

$$= \begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

If we define $\boldsymbol{m} \overset{\text{def}}{=} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \boldsymbol{M} \overset{\text{def}}{=} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, P \overset{\text{def}}{=} \begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ we can rewrite the equation as:

$$z\,\boldsymbol{m} = \begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{M}$$

$$\Updownarrow$$

$$z\,\boldsymbol{m} = P\boldsymbol{M}$$

Since homogeneous coordinates are not affected by the rescaling ($\forall z \neq 0 : \boldsymbol{m} = z\boldsymbol{m}$), $z\boldsymbol{m}$ and $\boldsymbol{m}$ represent the same point. So we write

$$\boldsymbol{m} \simeq P\boldsymbol{M}$$

to denote that $\boldsymbol{m}$ and $P\boldsymbol{M}$ are equal with *respect to a scale factor* z.

Notice that, even if the rescaling factor z is omitted in homogeneous coordinates, it plays a crucial role when referring to real measurements. The parameter z can be related to the distance of the points with respect to the reference system of the camera, and therefore, it links the size of real objects to the size of their projection in the image.

P is called *projection matrix* (or *camera matrix*) and defines the projection rules mapping 3D points into image points.

Using P and the two points in homogeneous coordinates we were able to transform a nonlinear operation into a linear one.

The simplest camera matrix P that we can obtain is the one were the focal length $f = -1$.

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = [I_3 \mid \underline{\mathbf{0}}]$$

It is an *ideal camera matrix* which is never met in general. Just by zooming, f changes. By the way, any camera matrix can be brought into this form.

1.11. General pinhole model

In our world, there are a lot of *non idealities* that affect the ideal camera matrix. When we are using our camera to take a picture of a subject, we change its focal lengths f just by zooming to put the target into focus. Furthermore, when we are dealing with digital images, their coordinates are in pixel not in meters; we consider them to have their origin $(0, 0)$ in the upper left corner, not at the center of the image plane anymore.

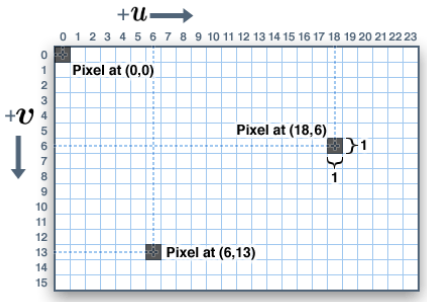


Figure 18: Coordinate system of a digital image.

We move from the simplified pinhole camera model to a more realistic model, called the *general pinhole model* that takes into account all these non idealities.

1.11.1. Intrinsic parameters

Every camera has some *intrinsic parameters* which are related to its manufacturing process. These parameters remain the same even if the camera changes its orientation because, they are related to its *configuration*, not the position.

The general equation of a camera matrix looks like:

$$P = \begin{bmatrix} -fk_u & -fk_u \cot\theta & u_0 & 0 \\ 0 & -\frac{fk_v}{\sin\theta} & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

and it takes into account three different non idealities.

The first non ideality, regards the fact that now, the point $(0,0)$ on the image plane isn't anymore at the center of the image plane but, it appears in the upper left corner. At the center now, there is the *principal point* (u_0, v_0). This adds two "shift" parameters u_0 and v_0 that define the position in pixel units of the center of the plane.

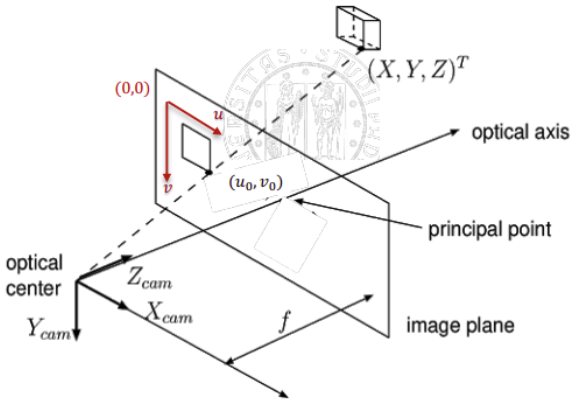


Figure 19: The image plane with its origin located in the upper left corner and, the principal point (u_0, v_0) at the center. (Notice that in this picture, the image plane has been moved to the right of the COP but, there would be no difference if it were at its left).

The second non ideality, regards the fact that, being the coordinates of the sensor grid expressed in pixel units, they represent integer values so, images coordinates must be quantized and discretized. Coordinates in the grid, can be obtained dividing the image coordinates by the width and height of a pixel. Pixels' shape could be rectangular instead of square, so we define p_u and p_v as the horizontal and vertical size of a sensor, measured in meters m, then, we define $k_u \stackrel{\text{def}}{=} 1/p_u$ and $k_v \stackrel{\text{def}}{=} 1/p_v$, the inverse of the effective pixel size along the direction $\boldsymbol{u}$ and $\boldsymbol{v}$ respectively, measured in m^{-1}. Finally, we need also to distinguish between the *focal length* f which is the physical distance between the image plane and COP, measured in meters, and the *focal* fk_u (or fk_v) used in the camera matrix of the general pinhole model, that is the product of the focal length f and k_u (or k_v) and is adimensional.

The third non ideality, regards the fact that, due to manufacturing errors, the grid of pixel isn't always perfectly squared and so, the camera coordinate system might be skewed. This means that the angle θ between the axes $\boldsymbol{u}$ and $\boldsymbol{u}$ could not be equal to $90°$. The difference, by the way, is usually not so much and, we will not consider this fact in the future (which means that for us $\theta = \pi/2$ so that $\cot(\pi/2) = 0$ and $\sin(\pi/2) = 1$).

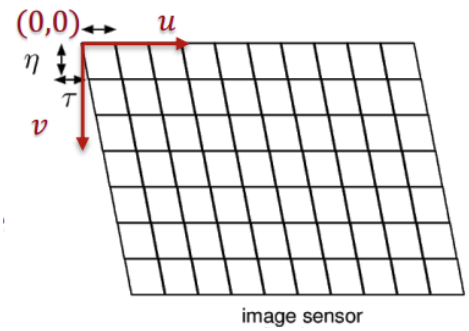


Figure 20: A skewed pixel grid of sensors

The equation of a pixel in the image plane can be derived in the following way:

$$z\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}-fk_u & -fk_u\cot\theta & u_0 & 0\\ 0 & -\frac{fk_v}{\sin\theta} & v_0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix} = \begin{bmatrix}-fk_ux & -fk_uy\cot\theta & zu_0 & 0\\ 0 & -\frac{fk_vy}{\sin\theta} & zv_0 & 0\\ 0 & 0 & z & 0\end{bmatrix}$$

Which in system form is equal to:

$$\begin{cases} zu = -fk_ux - fk_uy\cot\theta + zu_0\\ zv = -\frac{fk_vy}{\sin\theta} + zv_0\\ z1 = z\end{cases}$$

that, in Cartesian coordinates, assuming $\theta = \pi/2$ is:

$$\begin{cases} u = -fk_u\frac{x}{z} + u_0\\ v = -fk_v\frac{y}{z} + v_0\end{cases}$$

Intrinsic parameters can be represented using the *intrinsic matrix* K:

$$K = \begin{bmatrix}-fk_u & 0 & u_0\\ 0 & -fk_v & v_0\\ 0 & 0 & 1\end{bmatrix} \quad (\text{assuming } \theta = \pi/2)$$

and P can be written as:

$$P = \begin{bmatrix}-fk_u & 0 & u_0 & 0\\ 0 & -fk_v & v_0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix} = \begin{bmatrix}-fk_u & 0 & u_0\\ 0 & -fk_v & v_0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix} = K[I_3 \mid \mathbf{0}]$$

The coordinates of a point $\boldsymbol{m}$ can be normalized into a point $\boldsymbol{m}' = K^{-1}\boldsymbol{m}$, if we know the intrinsic parameters:

$$\begin{aligned}\boldsymbol{m} &\simeq P\boldsymbol{M}\\ \boldsymbol{m} &\simeq K[I_3 \mid 0]\boldsymbol{M}\\ K^{-1}\boldsymbol{m} &\simeq K^{-1}K[I_3 \mid 0]\boldsymbol{M}\\ \boldsymbol{m}' \overset{\text{def}}{=} K^{-1}\boldsymbol{m} &\simeq \underbrace{[I_3 \mid 0]}_{P}\boldsymbol{M}\end{aligned}$$

and it follows that in this case $P = [I_3 \mid 0]$, the ideal camera matrix. Normalized coordinates allows us to work with image points independently of the camera characteristics. Doing so, the obtained image points are *invariant* from the principal point (u_0, v_0) and the zoom factor.

1.11.2. Extrinsic parameters

Previously, we assumed that the world coordinates $(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z})$ were localized at the center of projection $\boldsymbol{C}$, now instead, the camera reference system and the world reference system are different.

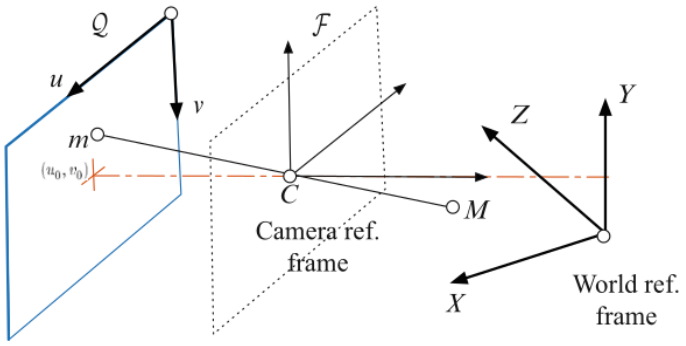


Figure 21: The general pinhole model.

It is possible that the camera is placed in a different position with respect to the world coordinates. We can use a rotation and translation to links the two axes systems. We do so, by using the matrix G that represent a Euclidean transformation.

$$G = \begin{bmatrix} R_{3\times3} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix}$$

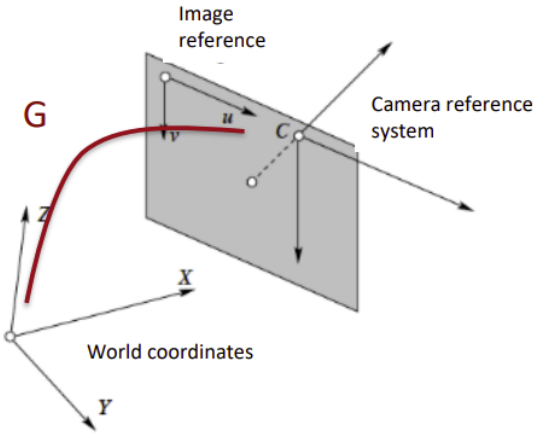


Figure 22: The world coordinates are mapped to the camera coordinates through a Euclidean transform.

We denote with $\boldsymbol{M_c}$ a point in camera reference coordinates and with $\boldsymbol{M}$ a point expressed in the world reference coordinates. The world coordinate $\boldsymbol{M}$, can be mapped into the camera coordinate $\boldsymbol{M_c}$ using G:

$$\boldsymbol{M}_c = G\boldsymbol{M}$$

G is called the matrix of the *extrinsic parameters*. It contains the rotation matrix R and the translation vector $\boldsymbol{t}$, so it has 6 parameters that encode the exterior orientation of the camera with respect to the world reference system. If the camera moves, only the extrinsic parameters change, not the instrinsic, provided that the acquiring configuration (focus, zooming, ...) doesn't change.

When the camera coordinates coincide with the world coordinates, we have that $R = I_3$ and $\boldsymbol{t} = \mathbf{0}$.

The equation of a pixel $\boldsymbol{m}$ in homogeneous coordinates is:

$$\begin{aligned}
\boldsymbol{m} \simeq P\boldsymbol{M_c} &= \begin{bmatrix} -f & 0 & u_0 & 0 \\ 0 & -f & u_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{M_c} \\
&= [K \mid \mathbf{0}]\, \boldsymbol{M_c} \\
&= K[I_3 \mid \mathbf{0}]\, \boldsymbol{M_c} \\
&= K[I_3 \mid \mathbf{0}]\, G\boldsymbol{M} \\
&= \underbrace{K}_{\substack{\text{intrinsic} \\ \text{parameters}}} \underbrace{[I_3 \mid \mathbf{0}]}_{\substack{\text{projective} \\ \text{normalized} \\ \text{coordinates}}} \underbrace{\begin{bmatrix} R_{3\times3} & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{bmatrix}}_{\substack{\text{coordinate} \\ \text{change}}} \boldsymbol{M} \\
&= K[R \mid \boldsymbol{t}]\boldsymbol{M}
\end{aligned}$$

and therefore, we can write the *general camera matrix* P as:

$$P = K[R \mid \boldsymbol{t}]$$

When the camera coordinates don't coincide with the world coordinates, it is possible to normalize the coordinates of a pixel $\boldsymbol{m}$ into a pixel $\boldsymbol{m}' = K^{-1}\boldsymbol{m}$, just by knowing K:

$$\begin{aligned}
\boldsymbol{m} &\simeq P\boldsymbol{M} \\
\boldsymbol{m} &\simeq K[R \mid \boldsymbol{t}]\boldsymbol{M} \\
K^{-1}\boldsymbol{m} &\simeq K^{-1}K[R \mid \boldsymbol{t}]\boldsymbol{M} \\
K^{-1}\boldsymbol{m} &\simeq K^{-1}K[R \mid \boldsymbol{t}]\boldsymbol{M} \\
\boldsymbol{m}' \overset{\text{def}}{=} K^{-1}\boldsymbol{m} &\simeq \underbrace{[R \mid \boldsymbol{t}]}_{P}\boldsymbol{M}
\end{aligned}$$

and it follows that in this case $P = [R \mid \boldsymbol{t}]$.

1.12. Center of projection coordinates

A generic camera matrix P can be rewritten according to its rows as:

$$P = \begin{bmatrix} \boldsymbol{p_1}^T \\ \boldsymbol{p_2}^T \\ \boldsymbol{p_3}^T \end{bmatrix}$$

and the prospective equation can be rewritten as:

$$\boldsymbol{m} \simeq P\boldsymbol{M} = \begin{bmatrix} \boldsymbol{p_1}^T \\ \boldsymbol{p_2}^T \\ \boldsymbol{p_3}^T \end{bmatrix} \boldsymbol{M} = \begin{bmatrix} \boldsymbol{p_1}^T\boldsymbol{M} \\ \boldsymbol{p_2}^T\boldsymbol{M} \\ \boldsymbol{p_3}^T\boldsymbol{M} \end{bmatrix}$$

so, the coordinates of a pixel in the plane becomes, in Cartesian coordinates:

$$\begin{cases} u = \frac{\boldsymbol{p_1}^T\boldsymbol{M}}{\boldsymbol{p_3}^T\boldsymbol{M}} \\ v = \frac{\boldsymbol{p_2}^T\boldsymbol{M}}{\boldsymbol{p_3}^T\boldsymbol{M}} \end{cases}$$

But what about $\boldsymbol{C}$? It is the 3D point that is at the origin of the camera coordinate system and lies on the focal plane $\mathcal{F}$. Homogeneous coordinates are nice to represent all the points made exception for $\boldsymbol{C}$ because, it represents a point in the Euclidean space with a specific location. Homogeneous coordinates, on the other hand, are used to represent points in the projective space, which extends Euclidean space to include points at infinity and facilitate projective transformations. It doesn't make sense to represent C as a point at infinity or as a point that can be scaled arbitrarily, therefore, it is represented using Cartesian coordinates.

$\mathcal{F}$ (made of all the points which are projected at the infinite, except for $\boldsymbol{C}$) is defined by the equation $\boldsymbol{p_3}^T\boldsymbol{M} = 0$, while the axes $u = 0$ and $v = 0$ correspond to the projection on the image plane $\mathcal{Q}$, of the planes $\boldsymbol{p_1}^T\boldsymbol{M} = 0$ and $\boldsymbol{p_2}^T\boldsymbol{M} = 0$ respectively.

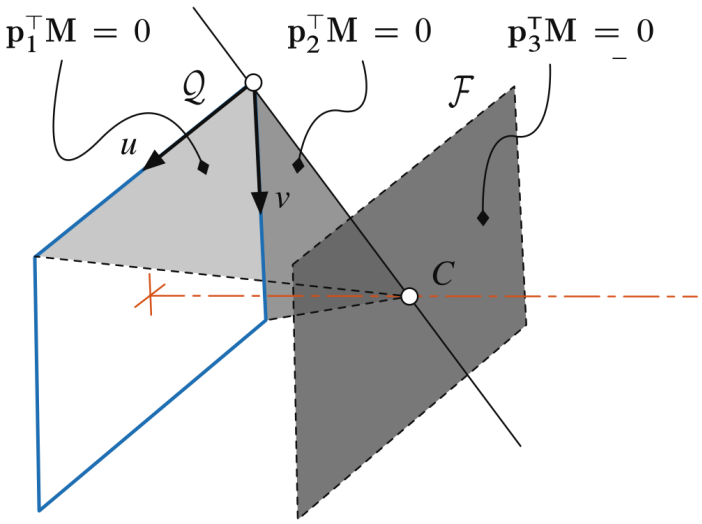


Figure 23: Geometric determination of the center of projection.

To find $\boldsymbol{C}$ we have to solve the system that express the intersection of the 3 planes at that point:

$$\begin{cases} \boldsymbol{p_1}^T\boldsymbol{C} = 0 \\ \boldsymbol{p_2}^T\boldsymbol{C} = 0 \\ \boldsymbol{p_3}^T\boldsymbol{C} = 0 \end{cases} \iff \quad P\boldsymbol{C} = \boldsymbol{0}$$

If we rewrite our generic P as

$$P = [Q_{3\times3} \mid \boldsymbol{q}]$$

and express $\boldsymbol{C}$ as

$$\boldsymbol{C} = \begin{bmatrix} \widetilde{\boldsymbol{C}} \\ 1 \end{bmatrix}$$

then

$$\begin{aligned} P\boldsymbol{C} &= \boldsymbol{0} \\ [Q \mid \boldsymbol{q}]\begin{bmatrix} \widetilde{\boldsymbol{C}} \\ 1 \end{bmatrix} &= \boldsymbol{0} \\ Q\widetilde{\boldsymbol{C}} + \boldsymbol{q} &= \boldsymbol{0} \end{aligned}$$

which leads to the Cartesian coordinates of the COP:

$$\widetilde{\boldsymbol{C}} = -Q^{-1}\boldsymbol{q}$$

1.13. Equation of a ray

Given two points $\boldsymbol{M_1}$ and $\boldsymbol{M_2}$, the convex combination of $\boldsymbol{M_1}$ and $\boldsymbol{M_2}$, gives you any point in the segment $\overline{\boldsymbol{M_1 M_2}}$:

$$\alpha \boldsymbol{M_1} + (1-\alpha)\boldsymbol{M_2} \quad \text{with } \alpha \in]0,1[$$

To represent any point on the line that passes through $\boldsymbol{M_1}$ and $\boldsymbol{M_2}$, a general linear combination can be used:

$$\alpha \boldsymbol{M_1} + \beta \boldsymbol{M_2} \quad \text{with } \alpha, \beta \in \mathbb{R}$$

The *optical ray* of a point $\boldsymbol{m}$ is the line that contains $\boldsymbol{C}$ and $\boldsymbol{m}$ itself. It corresponds to the infinite set of 3D points

$$\{\boldsymbol{M} : \boldsymbol{m} \simeq P\boldsymbol{M}\}$$

and it contains all the points $\boldsymbol{M}$ in the Euclidean space of which $\boldsymbol{m}$ is the projection onto the image plane.

By definition, $\boldsymbol{C}$ is in the set[1]. Another point contained in the set is the ideal point $\boldsymbol{M}_\infty$ defined as

$$M_\infty = \begin{bmatrix} Q^{-1}\boldsymbol{m} \\ 0 \end{bmatrix} \quad (\text{the last element is 0 because it lies at } \infty)$$

We can see that M_∞ is projected into $\boldsymbol{m}$ because given a generic $P = [Q \mid \boldsymbol{q}]$, $P\boldsymbol{M}_\infty$ is

$$P\boldsymbol{M}_\infty = \underbrace{QQ^{-1}}_{I}\boldsymbol{m} + 0\boldsymbol{q} = \boldsymbol{m}$$

In homogeneous coordinates, it is possible to write a parametric equation of the optical ray as a linear combination of $\boldsymbol{C}$ and $\boldsymbol{M}_\infty$:

$$\boldsymbol{M} = \boldsymbol{C} + \lambda \begin{bmatrix} Q^{-1}\boldsymbol{m} \\ 0 \end{bmatrix}, \qquad \lambda \in \mathbb{R} \cup \infty$$

Optical ray equations are the same for projectors and lasers as well. Given Q and $\boldsymbol{q}$ associated to the camera (projector) matrix P obtained from the process of calibration, it is possible to compute $\boldsymbol{M}$ in case the equation of the projection plane is known.

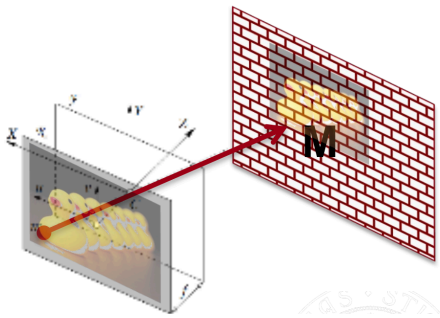


Figure 24: Projection of the point $\boldsymbol{M}$ on a plane.

[1] $\boldsymbol{m}$ is not an element of the set, it is a point in the image plane, not in the Euclidean space.

1.14. Camera calibration

There is a process called *camera calibration*, that allows us to estimate both the matrix of the intrinsic parameters K and the matrix of the extrinsic parameters $[R \mid \boldsymbol{t}]$.

We know the coordinates of n 3D points $\boldsymbol{M_i}$ (*calibration points*) and their projections $\boldsymbol{m_i} = P\boldsymbol{M_i}$ $(i = 1, ..., n)$ on the image plane. Knowing these points allows us to estimate the unknown parameters of P (assuming that we don't change the configuration or the position of the camera), which means knowing where every 3D point of the world will be projected in our camera.

Usually the $\boldsymbol{M_i}$ points are chosen from a 2D checkerboard (it is very easy to detect its corners). Multiple images of the checkerboard are taken in different positions, keeping the camera fixed and translating and rotating the checkerboard, simulating the motion of the camera (same K, different R and $\boldsymbol{t}$ for each picture).

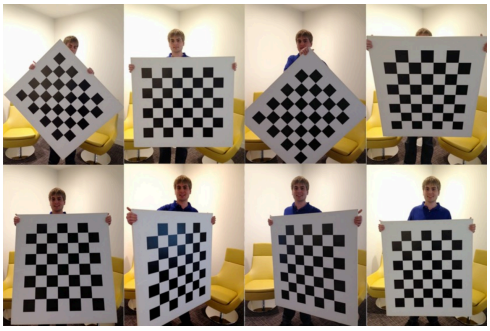


Figure 25: calibration through a checkerboad.

The calibration operation requires pixel precision. Usually, $\sim 30/40$ images are enough.

It is preferable to don't use JPEG images because, this format has compression and, compression destroys the smaller details. We need instead, to be as precise as possible!

We want to find a P such that the following quantity is *minimized*:

$$\sum_{i=1}^{n} \| \boldsymbol{m_i} - P\boldsymbol{M_i} \|$$

There exist different calibration methods, the most famous are:

- Direct calibration (camera parameters are estimated) [Caprile and Torre][2]
- Estimate perspective projection matrix [Faugeras 1993][3]
- Zhang's method[2000][4].

[2]https://link.springer.com/article/10.1007/BF00127813
[3]https://mitpress.mit.edu/9780262061582/three-dimensional-computer-vision/
[4]https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr98-71.pdf

2. Homography computation

2.1. Eigenvalues and eigenvectors recap

$$A_{n\times n}\boldsymbol{v} = \lambda\boldsymbol{v} \qquad \lambda \in \mathbb{R}$$

all the λ_i values that solve this equation are called *eigenvalues*, all the $\boldsymbol{v_i}$ that solve this equation are called *eigenvectors*.

A can be decomposed into

$$A = Q^T D Q$$

where

$$D = \begin{bmatrix} \lambda_1 & 0 & ... & ... & 0 \\ 0 & \lambda_2 & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \lambda_{n-1} & 0 \\ 0 & ... & .... & 0 & \lambda_n \end{bmatrix} \qquad \text{and} \qquad Q = [\boldsymbol{v_1}\ \boldsymbol{v_2}\ ...\ \boldsymbol{v_n}]$$

D is the diagonal matrix and Q is orthogonal (square matrix whose columns are orthogonal unit vectors)

If the matrix A is invertible ($\exists A^{-1}$), then every eigenvalue $\lambda_i \neq 0\ \ (i \in \{1, ..., n\})$. Otherwise, if A is not invertible ($\nexists A^{-1}$) it exists at least a λ_i such that $\lambda_i = 0$ with $i \in \{1, ..., n\}$.

If it exists, $A^{-1} = (Q^T D Q)^{-1} = Q^T D^{-1} Q$.

In case $\exists \lambda_i = 0$, we want to put on the right side of Q, all the columns $\boldsymbol{v_i}$ such that their $\boldsymbol{v_i} = \boldsymbol{0}$.

$$Q = \begin{bmatrix} \vdots & \vdots & ... & \vdots & 0 & ... & 0 \\ \vdots & \vdots & ... & \vdots & 0 & ... & 0 \\ \vdots & \vdots & ... & \vdots & 0 & ... & 0 \\ \vdots & \vdots & ... & \vdots & 0 & ... & 0 \end{bmatrix}$$

2.2. Plane to plane mapping

A first real world application of the homography is the **plane to plane mapping**, the task to map the *point of a plane* $\boldsymbol{\Pi}$ into the *image plane*. This operation can be done by a mapping function called **homography** (H_Π).
Starting from the equation $\boldsymbol{m} \simeq P\boldsymbol{M}$ we obatain that $\boldsymbol{m'} \simeq H_\Pi \boldsymbol{m}$:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \simeq \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,4} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Since we are working in 2D the z value is 0

Note that we can map any plane to any plane using $H_{\Pi(3\times3)}$ matrix. This means that we can compose homgraphy to map an image throught diffetent planes.

Since I got a 3×3 matrix I have 8 degrees of freedom. To compute H_Π we are going to use a set of i known points $(\boldsymbol{m'}, \boldsymbol{m})$:

$$
\begin{aligned}
&\boldsymbol{m_i}' \cong H_\Pi \boldsymbol{m_i} \\
&\qquad \Updownarrow \text{ since } \{(m, m') \mid \boldsymbol{m} \in \Pi \wedge \boldsymbol{m}' \in \text{image plane}\} \\
&\boldsymbol{m_i}' \times H_\Pi \boldsymbol{m_i} = 0 \\
&\qquad \Updownarrow \text{since } \boldsymbol{m_i}' \text{ is parallel to } H_\Pi \\
&\text{vec}\Big([\boldsymbol{m_i}']_\times H_\Pi \boldsymbol{m_i}\Big) = 0 \\
&\qquad \Updownarrow \text{vectorize the operation} \\
&\Big(\boldsymbol{m_i}^T \otimes [\boldsymbol{m_i}']_\times\Big) \text{ vec}(H_\Pi) = 0
\end{aligned}
$$

Let' s define $A = \boldsymbol{m_i}^T \otimes [\boldsymbol{m_i}']_\times$ which is a matrix of rank 2(only 2 equation are linearly indipendent) so given n points we are going to obtain $2n$ equations, for this reason we need at least **4** points to compute H_Π.

$$
A = \begin{bmatrix}
\boldsymbol{m_1}^T \otimes [\boldsymbol{m_1}']_\times \\
\boldsymbol{m_2}^T \otimes [\boldsymbol{m_2}']_\times \\
\boldsymbol{m_3}^T \otimes [\boldsymbol{m_3}']_\times
\end{bmatrix}
$$

Sometimes can be usefull to rescale the coordinates using a scale factor T/T'.

2.3. Features detection

We need at least 4 points to compute H_Π and so, we need to find to find a unequivocal correspondence between at least 4 points of Π and 4 points of the image plane.

This process can be thinked of as, finding the correspondences between the current image (from the camera) and some other visual information (from a database).

To find the correspodences between points, we compute *local descriptors* using *feature detection* algorithms.

Local Descriptors

Local descriptor are arrays that describe a specific point in an image so that, if an analogous array in another image is found, it may be assumed that, the two points described by the two arrays are exactly the same. This assumption by the way, may not always be correct.

More formally, in computer vision we need to distinguish between keypoints and descriptors:

A *key point* (also *local feature* or feature point) is a pixel coordinate (u, v) togheter with a *scale factor* s and an *orientation* o:

$$
\boldsymbol{K_i} = \begin{bmatrix}
u_i \\ v_i \\ s_i \\ o_i
\end{bmatrix}
$$

a *descriptor* is an array of values which is the signature of the key point, that is, a representation of the intensity/color function of the point. It is used to compare the similarity between features:

$$\boldsymbol{f}_i = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_t \end{bmatrix}$$

where α_i depends on the representation defined by a *feature detection algorithm*. By the way, according to the algotithm, there are various ways to represent a descriptor, such as:

- block of pixels;
- arrays of floats/bytes;
- binary arrays.

Feature detection algorithm

A feature detector (extractor) is an algorithm taking an image as input and outputting a set of regions: local features (key points).

Local features are *regions*, i.e. in principle arbitrary sets of pixels, not necessarily contiguous, which are at least:

- distinguishable in an image regardless of viewpoint/illumination, scale, rotation;
- robust to occlusion: must be local;
- must have a discriminative neighborhood: they are "features".

Example of detectors to extract local descriptors are: *SIFT, SURF, BRIEF, BRISK, FREAK*

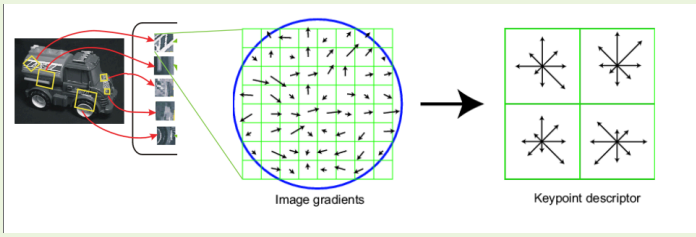


Figure 26: Desctiptor example using *Sift*

To identify the local descriptors, we proceed in two main steps:

1. *Feature detection*: we extract the features of interest (edges, corners, ...);
2. *Feature description*: we associate a *unique descriptor* to each feature (it characterizes the feature and allows to distinguish it from other features).

In this way we can identify the key points (feature points) that represent the set of points used to compute the *homography matrix* H_{Π}.

2.4. Feature matching

Once we have extracted the features $\boldsymbol{K_i}$ from the camera image and have computed their local descriptors $\boldsymbol{f_i}$, we do the same for the features $\boldsymbol{K_i'}$ and their local descriptors $\boldsymbol{f_i'}$ of the database image.

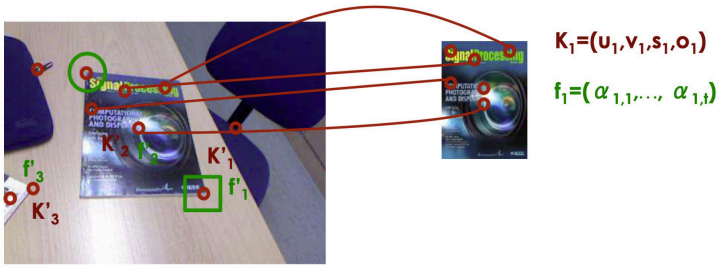


Figure 27: Key points (◯) and descriptors (□ and ◯)

If we take two descriptors, for example $\boldsymbol{f_1}$ and $\boldsymbol{f'_1}$ and compute their difference $\|\boldsymbol{f_1} - \boldsymbol{f'_1}\|$, we could argue that if the result is lower than a threshold ε, the two descriptors describe the same key point. In practice, we don't do this because it is very difficult to find a good ε; its value could be too high or too low and, taking into account also the presence of noise, there could be a lot of false positive or false negative matches.

Then instead, to find the key points who really match the local descriptors, we compute the distance $d_{i,j}$ between each pair $(\boldsymbol{f_i}, \boldsymbol{f'_j})$ of descriptors and, we order the results in ascending order; then we compute the ratio r between the best two matches and, compare it with a threshold τ that generally is 0.6. If $r < \tau$ then we can say that we have found a correct match (an *inlier*) otherwise we have a wrong match (an *outlier*).

The nitty-gritty is that, a key point should be uniquely distinguishable among all the other key points and, if $r > \tau$, this doesn't hold anymore. The value of τ determines if there is a match or not, in combination with the quality of the images and, it represents a crucial factor. The fact that we use a ratio between the best two matches excludes the possibility to have multiple matches (thing that could have happened instead if, we would have compared directly each descriptor $\boldsymbol{f_i}$ and $\boldsymbol{f'_j}$ with a threshold ε, that is: $\|\boldsymbol{f_i} - \boldsymbol{f'_j}\| < \varepsilon$).

This procedure is also sensible to noise and for this reason, in images with high levels of noise, there could be a lot of outliers.

Once we have determined the key points, we can finally compute the homography H_Π.

2.4.1. Key points computation example

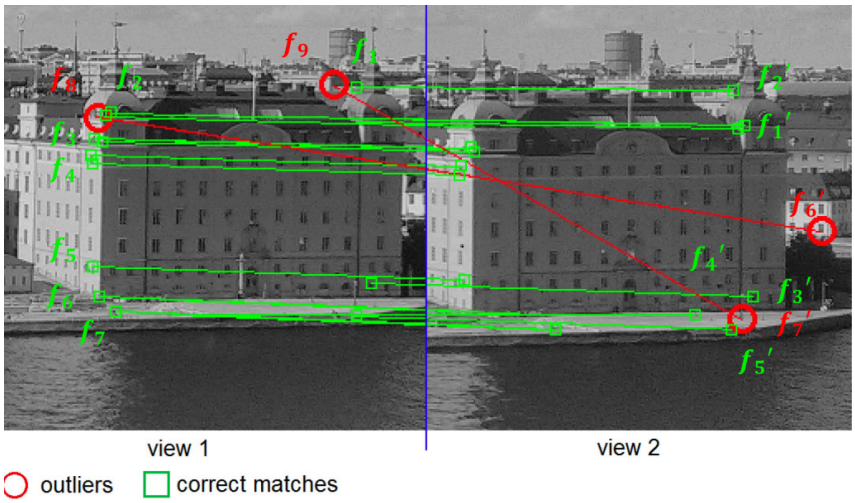


Figure 28: feature matching example
(notice that in this case the symbols ◯ and □ are used to represent outliers and inliers respectively)

Considering the Figure 28, for each descriptor $\boldsymbol{f_i}$ of the left picture, we compute the difference with all the descriptors $\boldsymbol{f'_j}$ of the right picture. For example, we consider the computations needed to find a correspondence for the points described by $\boldsymbol{f_1}$ and $\boldsymbol{f_8}$. Once the distances have been computed, we order them in ascending order and obtain:

$$
\begin{aligned}
d_{1,2} &= \|\boldsymbol{f_1} - \boldsymbol{f'_2}\| & d_{8,6} &= \|\boldsymbol{f_8} - \boldsymbol{f'_6}\| \\
&< & &< \\
d_{1,1} &= \|\boldsymbol{f_1} - \boldsymbol{f'_1}\| & d_{8,5} &= \|\boldsymbol{f_8} - \boldsymbol{f'_5}\| \\
&< & &< \\
&\vdots & &\vdots \\
&< & &< \\
d_{1,7} &= \|\boldsymbol{f_1} - \boldsymbol{f'_7}\| & d_{8,3} &= \|\boldsymbol{f_8} - \boldsymbol{f'_3}\| \\
&< & &< \\
d_{1,6} &= \|\boldsymbol{f_1} - \boldsymbol{f'_6}\| & d_{8,4} &= \|\boldsymbol{f_8} - \boldsymbol{f'_4}\|
\end{aligned}
$$

finally for each sequence, we compute the ratio between the best two matches and compare it with $\tau = 0.6$:

$$\frac{d_{1,2}}{d_{1,1}} = 0.35 < 0.6 \qquad \frac{d_{8,6}}{d_{8,5}} = 0.8 > 0.6$$

so, we can say that there is match between the key points represented by the descriptors $\boldsymbol{f_1}$ and $\boldsymbol{f'_2}$ and, that there are no matches for the point represented by the descriptor $\boldsymbol{f_8}$.

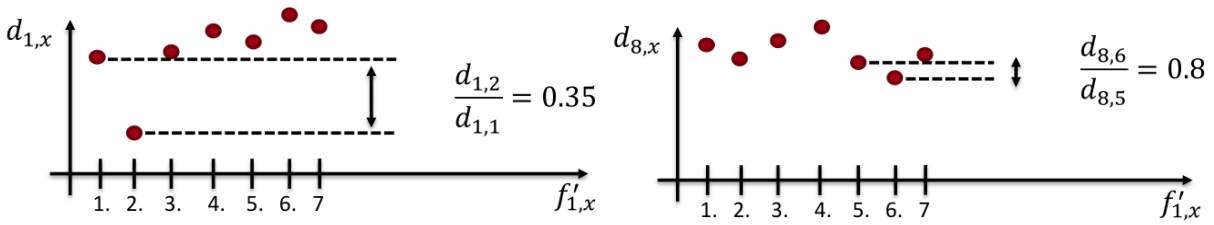


Figure 29:
On the left: the relation between the points $\boldsymbol{f'_x}$ and their distance from $\boldsymbol{f_1}$.
On the right: the relation between the points $\boldsymbol{f'_x}$ and their distance from $\boldsymbol{f_8}$.

2.5. Noise and compression

As mentioned above, the feature matching computation is sensible to strong *noise* levels that, generate false matches.

Unfortunately, noise and compression alter descriptors and key points (they change orientation, scale, locations); also, compression makes disappear the smallest local features and so, also the number of local descriptor is reduced.

The fact that compression also changes the position of the key points is a problem for 3D estimation because it reduces the level of precision when computing an homography.

Depending on the kind of descriptors that are used, there could be different problems. For example, SIFT is invariant to rotation but, only for angles of maximum $30°$, if the rotation introduced by the noise has a bigger angle, then features will not match anymore.

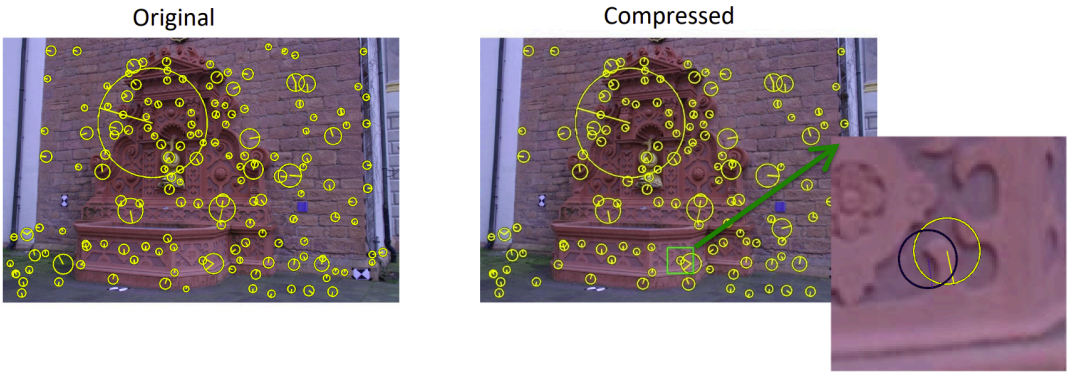


Figure 30: Effects of image compression.

We have seen that if we have an image I taken with our camera and a query image I_s, we can look for a common object between the two regardless of rescaling, occlusions, rotations, etc...

We can do so by computing (for example) SIFT key points and descriptors on I and I_s, obtaining the sets

$$\left\{(\boldsymbol{k_i}, \boldsymbol{f_i}) \mid \boldsymbol{K_i} = [u_i \;\; v_i \;\; s_i \;\; o_i]^T\right\} \qquad \text{and} \qquad \left\{(\boldsymbol{k_i^s}, \boldsymbol{f_i^s}) \mid \boldsymbol{K_i^s} = [u_i^s \;\; v_i^s \;\; s_i^s \;\; o_i^s]^T\right\}$$

respectively. Then we match the descriptors $\boldsymbol{f_i}$ to $\boldsymbol{f_j^s}$ so that it is possible to build a set of couples $\left(\boldsymbol{k_i}, \boldsymbol{k_j^s}\right)$. We assume without loss of generality that, the i-th key point in I is matched with the i-th key point in I_s so that the set is made of couples of the form $(\boldsymbol{k_i}, \boldsymbol{k_i^s})$.

Then we compute the affine transform (the homography) H that satisfies the relation

$$\begin{bmatrix} v_i \\ u_i \\ 1 \end{bmatrix} = H \begin{bmatrix} v_i^s \\ u_i^s \\ 1 \end{bmatrix}$$

and if

$$\left\| \begin{bmatrix} v_i \\ u_i \\ 1 \end{bmatrix} - H \begin{bmatrix} v_i^s \\ u_i^s \\ 1 \end{bmatrix} \right\| < \delta$$

for a certain threshold δ, the object is found.

The last step means that we are trying to match/make fit the object in I_s with the object in I through an affine transformation H that rotates and scales I_s. If the two roughly fit with respect to a delta δ, then we have found the object.

The problem is that the two images can be really noisy and the position of the key points could be moved. To solve this problem, we could adopt some more intelligent strategies, the one that we see are linear regression and the RANSAC algorithm.

2.5.1. Linear regression

Linear regression is a supervised learning problem that consists of fitting an $(n+1)$-dimensional hyperplane (in 2 dimensions a line) to a set of n points $\{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_N, y_N)\}$ where $\boldsymbol{x}_i = [x_{i,1} \;\; \cdots \;\; x_{i,n}]^T$. We want to find the set of parameters $\theta_1, ..., \theta_n$ that define the hyperplane which fits all these points. The plain can be defined as:

$$y_i = \theta_1 x_{i,1} + \theta_2 x_{i,2} + ... + \theta_n x_{i,n}$$

and it corresponds to the *linear model* of our data; it is corrupted by noise and in fact $y_i \stackrel{\text{def}}{=} \overline{y_i} + e_{y_i}$ and $x_{i,j} \stackrel{\text{def}}{=} \overline{x_{i,j}} + e_{x_{i,j}}$ where $\overline{y_i}$ and $\overline{x_{i,j}}$ are noise free and, e_{y_i} and $e_{x_{i,j}}$ are the noises added to them.

Considered:

$$\boldsymbol{x_i} = \underbrace{\begin{bmatrix} x_{i,1} \\ \vdots \\ x_{i,n} \end{bmatrix}}_{\substack{\text{independent} \\ \text{variables}}}, \quad \underbrace{y_i}_{\substack{\text{dependent} \\ \text{variable}}}, \quad \hat{\boldsymbol{\theta}} = \underbrace{\begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}}_{\substack{\text{Estimated} \\ \text{model} \\ \text{parameters}}}$$

it can be rewritten as

$$y_i \simeq \hat{\boldsymbol{\theta}}^T \boldsymbol{x}_i$$

and we're using the approximately equal sign "$\simeq$", because we can't be absolutely sure that all the data lie completely on a single line/plane.

In practice we have n weights $\theta_1, ..., \theta_n$ that must be tuned to minimize a loss function in a way such that:

$$y_i \simeq \theta_1 x_{i,1} + \theta_2 x_{i,2} + ... + \theta_n x_{i,n}$$

which corresponds to fitting the hyperplane to the set of data points.

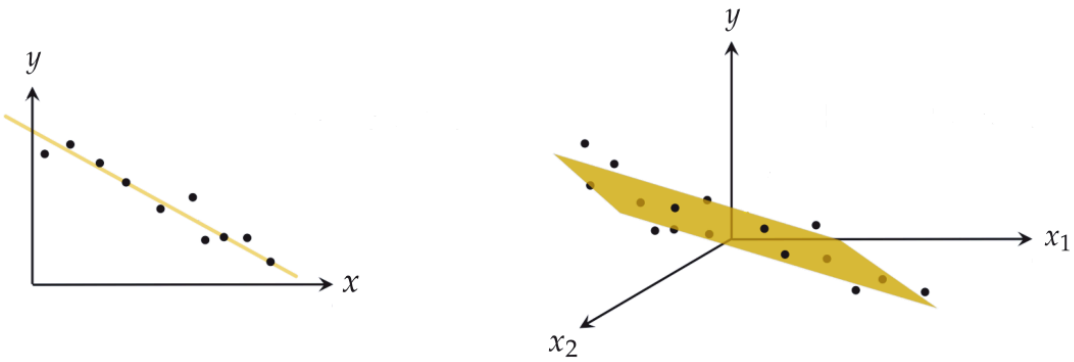


Figure 31:
On the left: linear regression in a 2D space.
On the right:linear regression in a 3D space.
(Picture taken from the book Machine Learning Refined by Watt, Borhani, Katsaggelos)

To fit the hyperplane we can use different approaches:

- *LS-estimator (Least Square-estimator)*: we want to find the parameter $\hat{\theta}$ such that, the *mean square error (MSE)* function

$$\sum_i \left\| y_i - \hat{\boldsymbol{\theta}}^T \boldsymbol{X_i} \right\|^2$$

is minimized.

$$y_i \simeq \hat{\boldsymbol{\theta}}^T \boldsymbol{x_i} \text{ is equivalent to } \hat{\boldsymbol{\theta}}^T \boldsymbol{x_i}^T \hat{\boldsymbol{\theta}} = y_i$$

which implies that the overdetermined system

$$\begin{bmatrix} \boldsymbol{x_1}^T \\ \vdots \\ \boldsymbol{x_n}^T \end{bmatrix} \hat{\boldsymbol{\theta}} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

is equivalent to

$$X\hat{\theta} = \boldsymbol{y}$$

We can multiply both parts of the equation for X^+, the pseudo inverse of X and get

$$\hat{\boldsymbol{\theta}} = X^+ \boldsymbol{y}$$

which is called the *least square solution*.

By the way, this approach is too simple and, the outliers still have an high impact on the computation of the paramter $\hat{\boldsymbol{\theta}}$.

In the LS estimate, we minimize the mean of of the square:

$$\min \left(\sum_i \left\| y_i - \hat{\boldsymbol{\theta}}^T \boldsymbol{x}_i \right\|^2 \right)$$

which is equivalent to:

$$\min \sum_i r_i^2 \equiv \min \mathbb{E}[r_i^2]$$

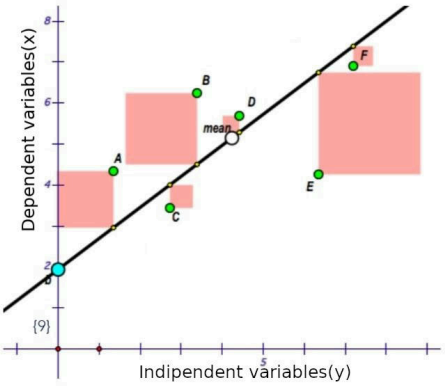


Figure 32: A representation of the mean square error approach, in red the *square error* (distance from the line) and in the center we can see the mean.

An approach to reduce the effect of the outliers on the estimation is the *LMedS (Least Median Square)* where the median of the residuals is computed instead of the mean:

$$\min\{\text{med}_i r_i^2\}$$

The LMedS algorithm is the following: given a set of N samples

1. take a random set of m samples ($m < N$);
2. compute the LS parameter estimate on the subset;
3. compute the median

$$\text{med}_i \left\{ \left(y_i - \hat{\boldsymbol{\theta}}^T \boldsymbol{x}_i \right)^2 \right\}$$

4. iterate stes 1.-3. t times and take the estimation with the minimum median.

- *M-estimator (Maximum likelihood-like estimator)*: An alternative is to modify the error function using a *loss/penalty function* ρ (which has to be symmetric, subquadratic and having minimum in $\mathbf{0}$) to obtain a more robust estimation:

$$\min \sum_i \rho(y_i - \theta^T \boldsymbol{x}_i)$$

An example of ρ is the Cauchy function

$$\rho(x) = \frac{\beta^2}{2} \log\left(1 + \frac{x^2}{\beta^2}\right)$$

- *R-estimator (Resistant estimator)*: suppose that we order residuals

$$r_i = y_i - \hat{\boldsymbol{\theta}}^T \boldsymbol{x}_i$$

such that the placement of residuals r_i is R_i.

Then we want to minimize:

$$\min\left(\sum_i a_n(R_i) r_i\right)$$

where a_n is a monotonic decreasing function such that

$$\sum_i a_n(R_i) = 0$$

2.5.2. RANSAC

RANSAC (Random Sample Consensus) is a widely-used iterative algorithm for robust model estimation that basically, subsample randomly couple of matches with the objective to leave out outliers. This process is repeated multiple times untill the desired result is obtained.

Given a set of N points:

1. Take a random subset of m samples ($m < N$);
2. Compute the least square parameter estimate on the subset using a robust estimation technique (e.g.: *LMedS*);
3. Compute the numbers of *inliers* over the entire set (with cardinality N), i.e.: the points that are distant at most ε from the regression line: $\|y_i - mx_i - q\|^2 < \varepsilon$;
4. Iterate step 1.-3. until the number of *inliers* is greater then a threshold T or, for a finite number of iterations;
5. Take the best least square estimation and recompute the parameters only on the set of inliers.

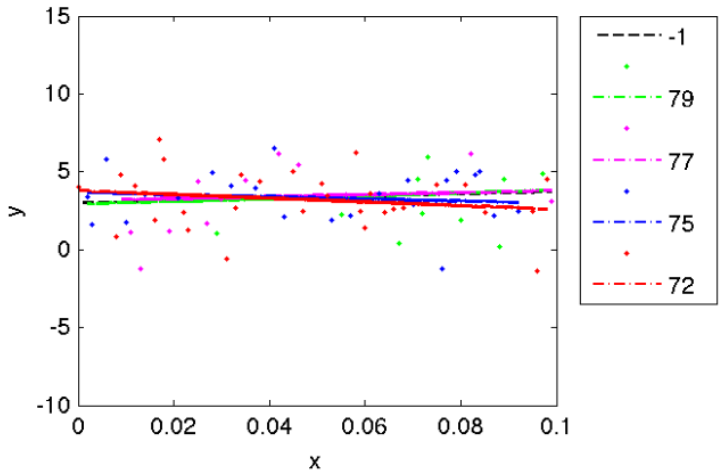


Figure 33: RANSAC example with various iterations. The best iteration is the one corresponding to the green line (79 inliers), which is the one that approximate better the ground truth line (represented for convention with -1 inliers)

2.5.2.1. RANSAC homography estimate

RANSAC can be used to estimate the homography H_{Π} and, the process is mostly the same as the one described above. In this case, the algorithm works on the couple of points from the two images.

1. Compute the set of possible matches (cardinality N) using any feature detection algorithm (i.e. SIFT);

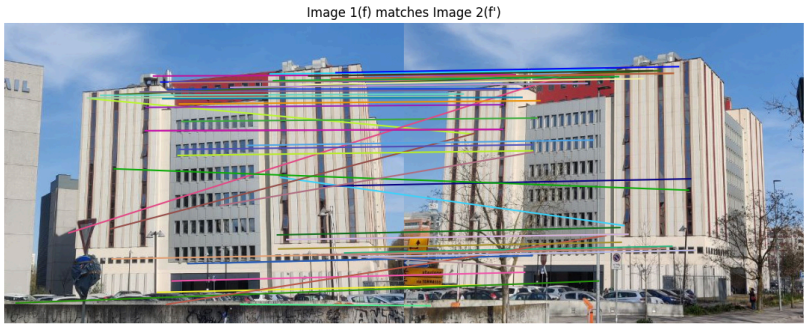


Figure 34: Match points obtained using SIFT.

- Select a random subset of points (cardinality $m < M$) from the set of all the possible matches computed in 1.;

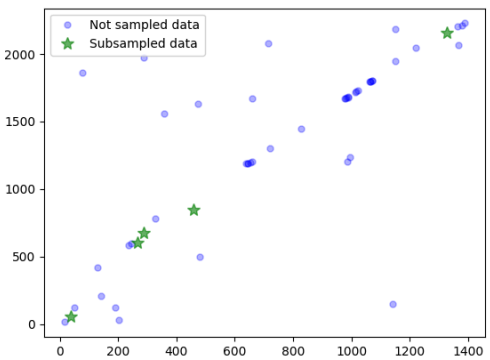


Figure 35: Subsample set of size 5 over a set of size 51.

- Compute the homography H over the subsampled data using linear regression;

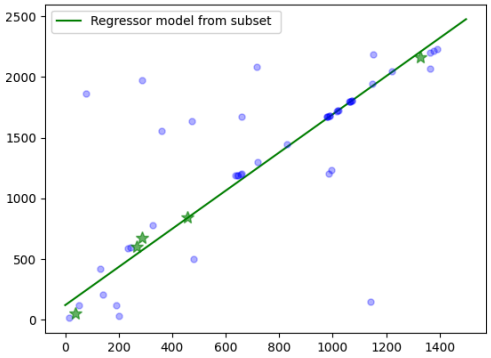


Figure 36: The regressor line computed over the subsamplded data

- Compute the number of inliers n (i.e.: the points such that $\| m_i' - Hm_i \|^2 < \varepsilon$);

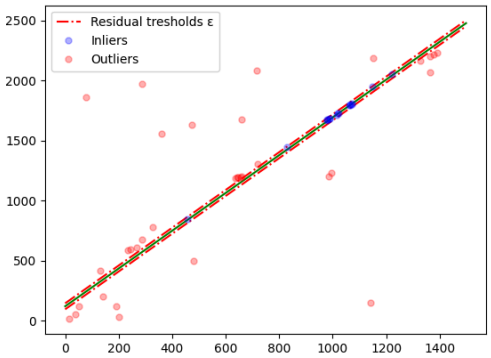


Figure 37: The identification of inliers (blue dots) and outliers (red dots) over the entire data using an $\varepsilon = 25$.

- Repeat until the max number of iterations is reached or, the inliers' threshold is reached. The number of iterations at the end is k.

$$\begin{aligned} &\text{it 1: } H_1 \longrightarrow \#\text{inliers } n_2 \\ &\text{it 2: } H_2 \longrightarrow \#\text{inliers } n_2 \\ &\qquad\vdots \\ &\text{it k: } H_k \longrightarrow \#\text{inliers } n_k \end{aligned}$$

2. Take H_k s.t. $k = \arg\max_i n_i$;
3. Finally recompute H_π on the inliers set.

At the end the object is found if $\#\text{inliers}$ is greater than a threshold T_{found}.

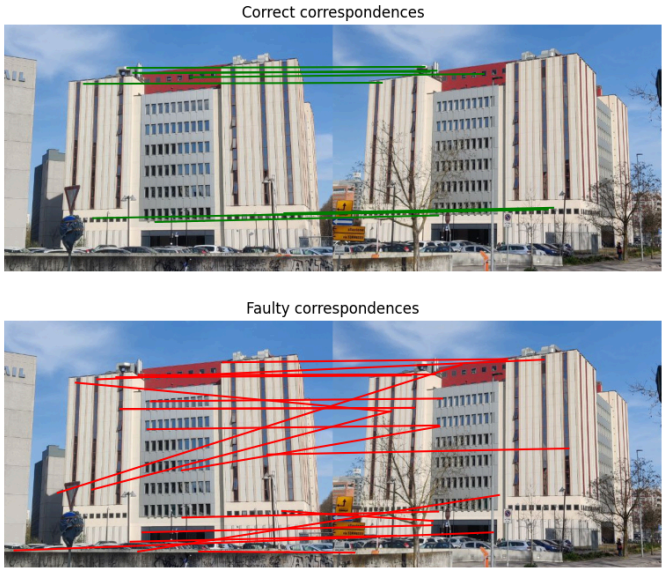


Figure 38: The inliers above and outliers below.

The accuracy of the homography estimation depends on:

- the tresholds:
  - ε: its accuracy tells us if a point is an inlier or an outlier;
    - low value $\Rightarrow$ more robust matches but less inliers;
    - high value $\Rightarrow$ less robust matches but more inliers. Notice that these matches are less robust, so in the end there is a risk of getting more false positive matches.
  - T_{found}: its accuracy tells us the number of inliers founded at the end of the algorithm;
    - low value $\Rightarrow$ faster but low confidence;
    - high value $\Rightarrow$ high computational cost with high confidence.
- the number of iterations and confidence:
  - more iterations $\Rightarrow$ more inliers but also more computational demanding;
  - more confidence $\Rightarrow$ more inliers but also more computational demanding;
  - the more keyponts the more matches have to be done, matches implies iterations and the more iterations the more computation has to be done.

Figure 39:
On the left: the number of inliers as a function of the number of iterations and the confidence.
On the right: the execution time as a function of the number of iterations and the confidence

So, in the end, a trade-off between the number of iterations and the confidence must be considered.

RANSAC improve the matching of local features despite variations in the viewpoint, illumination, and occlusions.

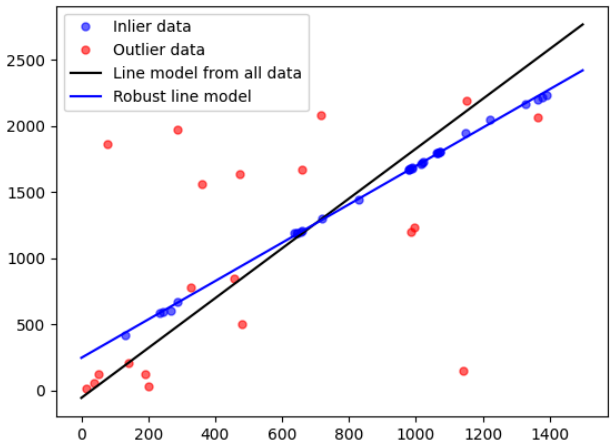


Figure 40: Comparison between the *linear* model and the the *RANSAC* model of Figure 34

2.5.2.2. The environment plays a significant role

Robust target recognition and tracking is challenging to obtain due to several factors:

- natural and artificial environmental occlusions;
- false matches as a consequence of projected shadows;
- insufficient features recognition because of brightness variation and lightning saturation.

Resulting on:

- Unstable 3D model registration and tracking;
- 3D model oscillation and wrong positioning when user's viewpoint is changed.

Possible solutions to this problem are:

- acquire a wide and diversified dataset;
- focus on multiple small and detailed targets;
- use 3D object targets.

Figure 41: Same picture taken under different enviromental conditions, during the day,

2.5.3. 3D object Target

It is important to notice that, in the end, when we are implementing a VR app, the homography estimation used to map the points will be unstable for tracking, as consequence of a change in the user's view point because, it moves the mobile phone and there can be model oscillations. A possible solution to this problem is the *3D object target* which, allows us to recognize and track particular objects in the real world based on their shape.

It extends the capabilities of recognition and tracking of complex 3D Objects and is recommended for monuments, statues, industrial objects, toys and tools.

It requires the 3D objects to be rigid, to present a sufficient number of stable surface features and to be fixed with respect to the environment.

It is implemented by using Object Recognition algorithms such as the Simultaneous Localization and Mapping (SLAM).

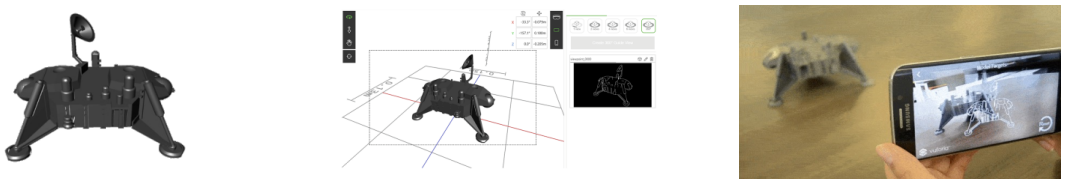


Figure 42: 3D object target example taken from vuforia.com