A Method for Measuring 3D Distances from a 2D Image Without Knowing Depth or Having a Known Pattern.

Abel Amado González Bernad

December 2023

Abstract

This paper presents a method to coordinate and measure elements captured by a 2D camera without any information about a measured pattern or knowledge of the depth of the elements in the image. It is necessary to constrain the presence of the photographed elements to a plane and to know the extrinsic position of the camera with respect to that plane. An example implementation is provided for positioning individuals using YOLOv8 as an instance detector and measuring the area of a sheet of paper in various positions.

Code available at: `https://github.com/abelBEDOYA/plane-measurement`

1 Introduction

Positioning and measuring elements in a 2D image is of interest. Capturing this image involves projecting 3D space onto a 2D sensor, resulting in irreversible information loss. Therefore, for precise measurements, the use of telecentric cameras is common, although they have limitations (workspace and lighting). To estimate positions with any 2D camera beyond pixel measurements, it is possible to rely on additional information that compensates for the loss in projection.

- Knowing the depth of the photographed element. Knowing the distance to the camera parallel to the optical axis allows coordinating its position.
- Having and recognizing a previously measured pattern. Knowing the dimensions of what is being photographed allows positioning and orienting that extended body in 3D space through algorithms based on *keypoints* and pose estimation [1] [2] [3].
- Restricting the elements in the image to a plane and knowing the extrinsic position of the camera with respect to it. This constraint helps overcome information loss due to projection. A method for coordinating and measuring under this condition is provided here.

2 Modeling

The pinhole camera model is considered, which proposes an ideal camera that assumes all rays forming the image pass through the same point. This models the camera optics with a matrix, Eq. 1, and a non-linear compensation for possible lens distortion, Eq. 2 and Eq. 3. These are the coefficients of the polynomials $(k_1, k_2, p_1, p_2, k_3)$, which perform radial and tangential pixel shifts in an image [4].

$$\mathbf{M} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$\begin{aligned} {}^{r}x_{\text{distorted}} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ {}^{r}y_{\text{distorted}} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{2}$$

$$\begin{aligned} {}^{t}x_{\text{distorted}} &= x + (2p_1 xy + p_2(r^2 + 2x^2)) \\ {}^{t}y_{\text{distorted}} &= y + (p_1(r^2 + 2y^2) + 2p_2 xy) \end{aligned} \tag{3}$$

2.1 Camera Calibration

Each camera has a different lens and sensor. This can be summarized using the pinhole camera model with the calibration matrix and distortion coefficients. These are parameters to be adjusted and allow the characterization of the camera. The `cv2.cameraCalibration()` method [4] can be used for this purpose. This method should receive the dimensions of a known pattern (e.g., a chessboard) and its corresponding pixel position in several sample images (calibration images) taken from different positions, as shown in Figure 1.

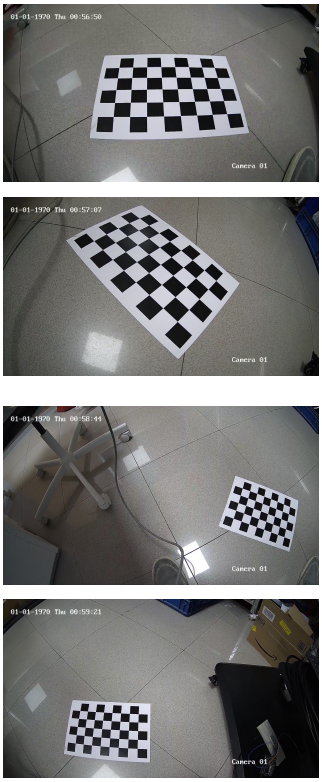


Figure 1: Example images of the calibration pattern for camera parameter adjustment.

2.2 Distortion Compensation

Calibration allows, among other things, the removal of lens distortion from the image, as shown in Figure 2.

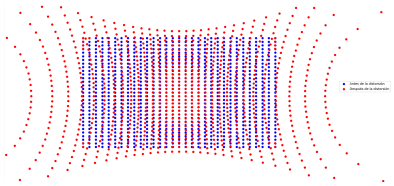


Figure 2: Pixel indices of the original image and transformed according to lens distortion. It accentuates as they move away from the center.

Removing distortion involves transforming the pixel positions, $\mathbf{r_p}$, to where they would have been without distortion, $\mathbf{r'_p}$, i.e., mapping the indices of the image matrix.

2.3 From Reality to Sensor

Camera calibration allows estimating to which pixel in the image, $\mathbf{r'_p} = (x'_p, y'_p)^{\mathrm{T}}$, a certain position in three-dimensional reality, $\mathbf{r} = (x, y, z)^{\mathrm{T}}$, will be mapped, Eq. 4.

$$\mathbf{r_p}' = M \cdot \mathbf{r} \div z$$

or,

$$\begin{bmatrix} x'_p \\ y'_p \\ 1 \end{bmatrix} = \left(\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \right) \div z \tag{4}$$

This is a linear transformation giving the indices of the pixel associated with that three-dimensional point. These pixels will later receive the compensation given by lens aberration, Eq. 2 and Eq. 3.

2.4 From Sensor to Reality, Given Depth

It is the inverse calculation of bringing reality to the sensor, i.e., starting from the index of a 2D pixel and estimating the 3D point is impossible due to information loss in the projection. However, knowing the depth, i.e., the value of the z component (parallel to the optical axis), allows coordinating the point in reality identified in the image. Thus, by inverting the transformation matrix M and solving for $\mathbf{r}$, we have,

$$\mathbf{r} = M^{-1} \cdot \mathbf{r'_p} \cdot z$$

or,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \left(\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} x'_p \\ y'_p \\ 1 \end{bmatrix} \right) \cdot z \tag{5}$$

where $\mathbf{r}'$ indicates the indices of the undistorted pixel. Note again the homogeneous coordinates $r' = (x_p, y_p, 1)^{\mathrm{T}}$.

3 Camera-Plane Calculation

Now, given the pinhole camera model and a method to coordinate 3D positions from 2D indices on the image, with known depth z, we propose a method to do the same without knowing this depth for the points.

To achieve this, it must be required that all elements to be located are contained in a plane, such as the ground, the sea, a lake, a field, ... For simplicity, this will be the plane defined by the equation $z = 0$. The extrinsic position of the camera relative to the plane must be known, i.e.,

- Its height, h.
- Incidence angle, β
- Camera roll angle, γ.

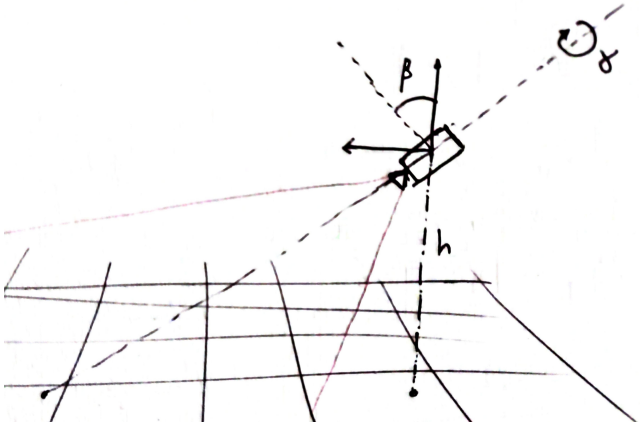


Figure 3: Scheme of the camera-plane problem. Note the three extrinsic parameters defining the camera's position and orientation: height, h, pitch, β, and roll, γ, relative to the plane.

3.1 Calculation of Parametric Equations

Each pixel in the photograph refers to a line in 3D space, and the photographed elements can be at any point on each line of each pixel. Determining the parametric equations of these lines is the first objective, as shown in Figure 4.

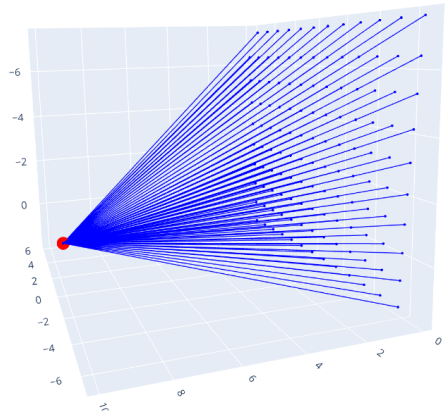


Figure 4: Lines emanating from the camera with directions to each pixel.

The fact that they are lines allows their equations to take the form,

$$\mathbf{r} = \begin{cases} x = x_0 + at, \\ y = y_0 + bt, \\ z = z_0 + ct. \end{cases} \tag{6}$$

where t is the parameter or degree of freedom that allows traversing a certain line.

It is possible to simplify by taking one of its components as a parameter to traverse the line. For convenience, y is taken, which is the choice of the camera axis that will coincide with its optical axis. In turn, we can assume that the camera is at the origin of the coordinate system and at a height h above the plane, such that $P = (0, 0, h)$, being its position the independent terms of the expression of each component. Rewriting results in,

$$\mathbf{r} = \begin{cases} x = pte_x \cdot y, \\ y = y, \\ z = pte_z \cdot y + h. \end{cases} \tag{7}$$

Now, two unknown values are apparent, pte_x and pte_z, and the freedom of the parameter y. This latter will be eliminated by imposing the restriction that the points must lie on the plane later on. However, the slopes of the x and z components depend intrinsically on the camera (and each pixel of the image). Remembering Eq. 5, the coordinate $\mathbf{r}$ linearly depends on the inverse of the camera matrix, being the proportionality factor particular for each pixel the slope (pte_x and pte_z) needed in the parametric equations of the lines.

$$\mathbf{r} = M^{-1} \cdot r'_p \cdot y \;\; \rightarrow$$
$$\frac{d\mathbf{r}}{dy} = M^{-1} \cdot r'_p = (pte_x, 1, pte_z)$$

It turns out that, as warned, the slopes of the parametric equations are constant, $pte_i(r'_p)$, that is, lines, with r'_p being the undistorted indices of a pixel in the image, such that,

$$pte_x(r'_p) = \frac{dx}{dy} = M_1^{-1} \cdot r'_p$$
$$pte_z(r'_p) = \frac{dz}{dy} = M_3^{-1} \cdot r'_p$$

These slopes have an immediate interpretation: they are the tangent of the field of view (FoV) that each of the pixels covers concerning the optical axis. The FoV in the vertical and horizontal axes is the arctangent of the slope of the extreme pixels in each axis. This is:

$$FoV_x = 2 \cdot \arctan\left(\text{pte}_x(w_p, h_p/2)\right) \tag{8}$$
$$FoV_y = 2 \cdot \arctan\left(\text{pte}_y(w_p/2, h_p)\right) \tag{9}$$

Where w_p and h_p are the width and height in pixels of the image respectively. Some examples of calibrated cameras can be seen in Figure 10.

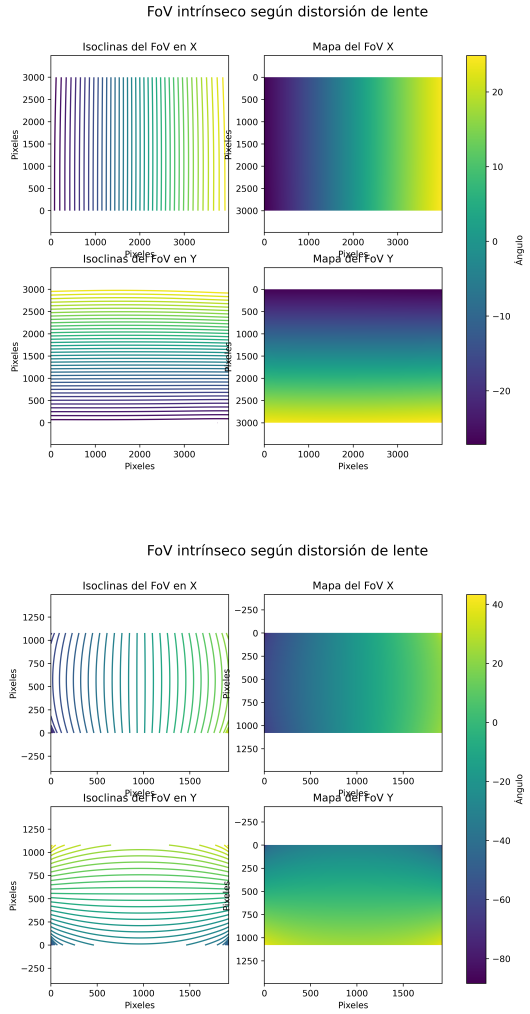


Figure 5: Angle covered by each pixel in the horizontal and vertical axes in the image concerning the optical axis. Note the lens distortion in the second camera.

3.2 Camera Rotations: Pitch, β, and Roll, γ

The equations of the form Eq. 7 assume a camera with an extrinsic position such that the optical axis is aligned with the y axis (parallel to the plane). However, there are 3 degrees of freedom in the orientation of a rigid body (a camera) that could be taken into account. One of them is dispensable for simplicity: rotations around the axis normal to the plane, in this case, the z axis. Therefore, it is only necessary to consider the extrinsic orientation through two angles: pitch, β, and roll, γ. These represent a rotation about the x axis, which controls the incidence of the optical axis to the plane, the height of the horizon in the image, zenith, etc., and the roll, γ, a rotation about the y axis that controls the camera's rotation about its own axis, i.e., how twisted the photo is taken. Mathematically, it is possible to apply a rotation about an axis in 3D dimensions using rotation matrices.

$$R_x(\beta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta) & -\sin(\beta) \\ 0 & \sin(\beta) & \cos(\beta) \end{bmatrix} \tag{10}$$

$$R_y(\gamma) = \begin{bmatrix} \cos(\gamma) & 0 & \sin(\gamma) \\ 0 & 1 & 0 \\ -\sin(\gamma) & 0 & \cos(\gamma) \end{bmatrix} \tag{11}$$

Taking the position vector, $\mathbf{r}$, defined by the parametric equations, Eq. 7, it is possible to apply the rotations one after another to mathematically "orient" the camera relative to the plane. Applying the rotation about the optical axis, i.e., the y axis, R_y,

$$\begin{bmatrix} x_\gamma \\ y_\gamma \\ z_\gamma \end{bmatrix} = \begin{bmatrix} \cos(\gamma) & 0 & \sin(\gamma) \\ 0 & 1 & 0 \\ -\sin(\gamma) & 0 & \cos(\gamma) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{12}$$

it yields,

$$\mathbf{r}_\gamma = \begin{cases} x_\gamma = (pte_x \cos\gamma + pte_z \sin\gamma) \cdot y, \\ y_\gamma = y, \\ z_\gamma = (-pte_x \sin\gamma + pte_z \cos\gamma) \cdot y. \end{cases} \tag{13}$$

Subsequently, the rotation about the x axis is applied, causing the pitch of the camera,

$$\begin{bmatrix} x_{\gamma\beta} \\ y_{\gamma\beta} \\ z_{\gamma\beta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta) & -\sin(\beta) \\ 0 & \sin(\beta) & \cos(\beta) \end{bmatrix} \cdot \begin{bmatrix} x_\gamma \\ y_\gamma \\ z_\gamma \end{bmatrix} \tag{14}$$

it yields,

$$\mathbf{r}_{\gamma\beta} = \begin{cases} x_{\gamma\beta} = (pte_x \cos\gamma + pte_z \sin\gamma) \cdot y, \\ y_{\gamma\beta} = [\cos\beta - \sin\beta(pte_z \cos\gamma - pte_x \sin\gamma)] \cdot y, \\ z_{\gamma\beta} = [\sin\beta + \cos\beta(pte_z \cos\gamma - pte_x \sin\gamma)] \cdot y. \end{cases} \tag{15}$$

3.3 Solving the Line-Plane Equations

Once the complete parametric equations for each pixel of the image are obtained, it is possible to impose the condition $z = 0$, i.e., that the solution points are contained in the plane. Taking into account an independent term in the z component due to the height, h, of the camera above the plane and solving the system of equations leads to,

$$\mathbf{r} = \begin{cases} x = \dfrac{-h(pte_x \cos\gamma + pte_z \sin\gamma)}{\sin\beta + \cos\beta(pte_z \cos\gamma - pte_x \sin\gamma)} \\ y = \dfrac{-h(\cos\beta + \sin\beta(pte_x \sin\gamma - pte_z \cos\gamma))}{\sin\beta + \cos\beta(pte_z \cos\gamma - pte_x \sin\gamma)}, \\ z = 0 \end{cases} \tag{16}$$

To visualize the results for each pixel of the image, it is possible to represent the line segments from the point $(0, 0, h)$, i.e., the camera position, to the found solution points, as shown in Figure 6.

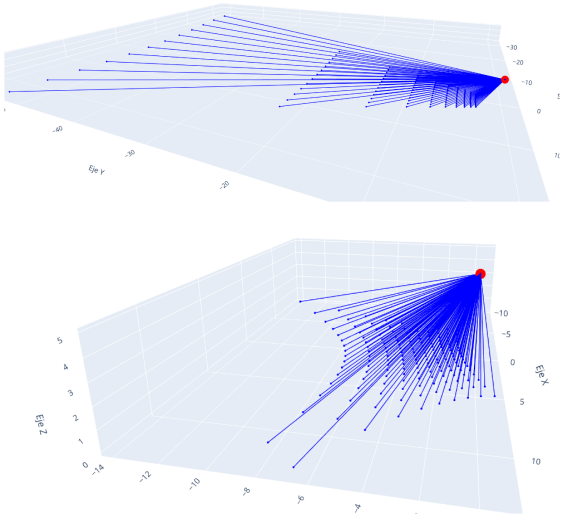


Figure 6: Solution of the parametric equations for a camera with and without lens distortion for given extrinsic configurations.

4 Interpretation and Validation of Results

The calculation aims to position and measure photographed elements that lie in a plane of reality (a floor, the sea, a plot of land, ...). Therefore, this section provides examples of use, validating its utility. These examples include positioning people walking on a street using an object detector based on deep learning and measuring the area of a sheet of paper.

4.1 Real Positioning of Each Pixel

Once the solutions are obtained, it is possible to represent the coordinate map (x, y) of the image for a given configuration, as shown in Figure 7.

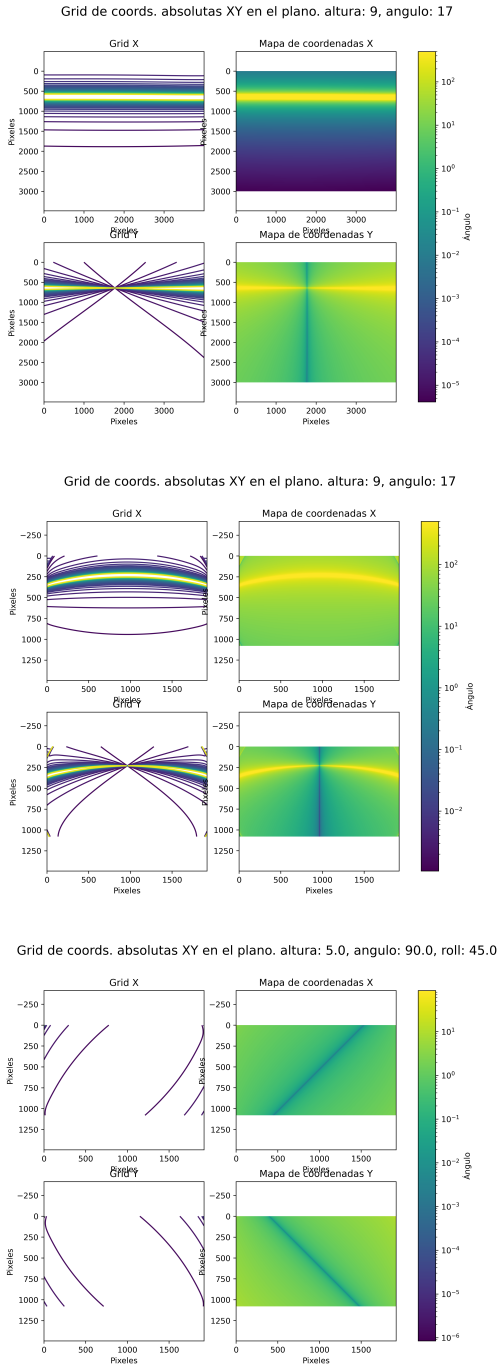


Figure 7: Grid of coordinates on the $z = 0$ plane for two cameras in different extrinsic configurations. Note the absolute value applied and the logarithmic scale of the colors.

With the correspondence of each pixel to a coordinate, $\mathbf{r} = (x, y)$, in reality on the plane, it is possible to integrate it with an object detector based on deep learning. The AI model provides the pixel in which a certain instance is located in the image, which corresponds to the indices, (i, j), of the position matrix, $\mathbf{r}$, which provides the real position.

An example of use has been implemented with YOLOv8, [5], detecting people. In Figure 8, the coordinates in meters with respect to the camera are shown above each detected person.

Figure 8: Localization of people implementing YOLOv8 along with the position matrix, **r**.

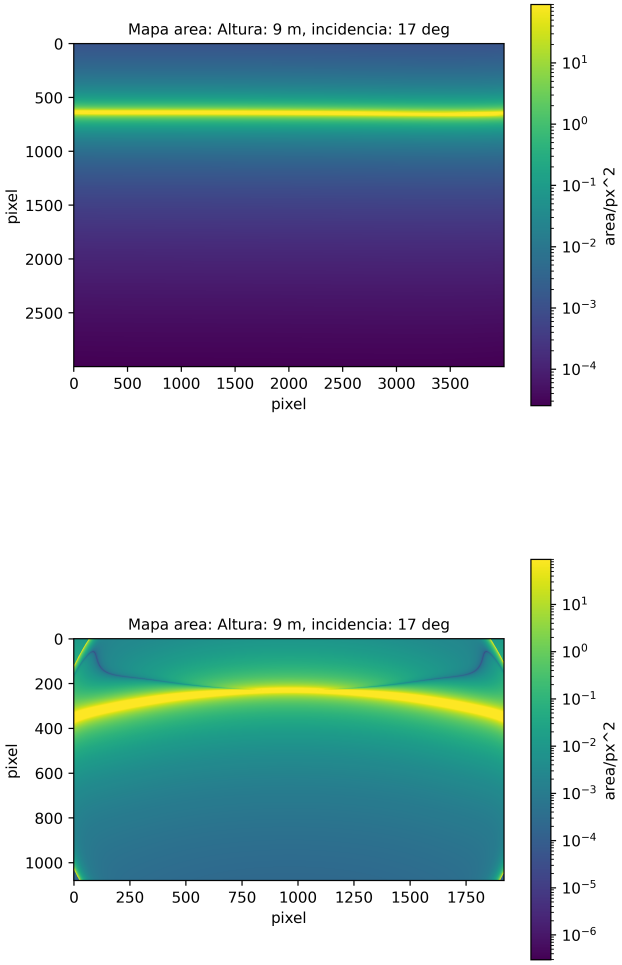


Figure 9: Weight map, A_{ij}, for converting pixel areas to real units, in this case: m^2, for two cameras and different arrangements.

4.2 Calculation of Real Areas

Once all the pixels of an image are located on the plane, it is possible to measure distances between them or areas. For the latter case, it is possible to calculate the rate of change of position or gradient from one pixel to another in the two axes of the image, obtaining Δx and Δy. This way, it is possible to calculate the area attributed to each pixel:

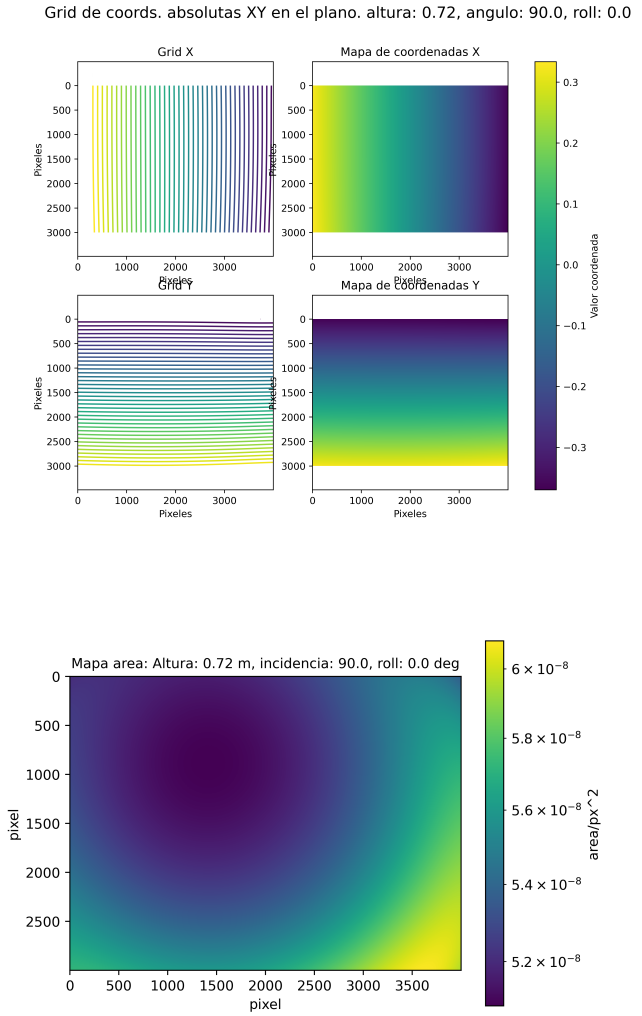
$$A_{ij} = \Delta x_{ij} \cdot \Delta y_{ij} \tag{17}$$

This results in a weight map, A_{ij}, with dimensions $[\mathrm{L}]^2[\mathrm{px}]^{-2}$, i.e., a pixel area to real area conversion.

This map greatly facilitates the calculation of areas, allowing the multiplication of each index of the weight matrix, A_{ij}, with a binary mask, M_{ij}, outlining the photographed elements, Eq. 18.

$$area = \sum_i \sum_j M_{ij} A_{ij} \tag{18}$$

Figure 10: Application of the weight matrix to calculate the area of a sheet of paper in two different configurations. The area of an A4 sheet is $0.6237\ \mathrm{m}^2$.



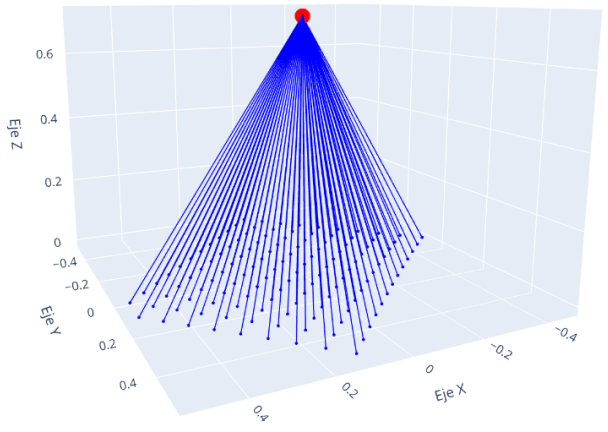


Figure 11: Solution for a top-down camera without distortion.

For this last case, where the photo is completely top-down, i.e., $\beta = 90º$, you would have coordinate and weight maps given by Figure 11.

As expected, the x axis grows from left to right, and the y axis from top to bottom. The weight map that allows calculating the area has radial symmetry, as expected.

5 Conclusion

A method has been obtained to locate and measure elements photographed with a 2D camera without knowing the depth of each point in the image. It is a necessary condition that the elements are contained in a plane and to know the camera's position relative to it. It is valid for any optics. Representations of the results have been provided, examples of use locating people, and validation estimating the area of an A4 sheet in two different camera configurations.

References

[1] X.S. Gao, X.-R. Hou, J. Tang, H.-F. Chang "Complete Solution Classification for the Perspective-Three-Point Problem" ([96]). In this case the function requires exactly four object and image points.

[2] F. Moreno-Noguer, V. Lepetit and P. Fua in the paper "EPnP: Efficient Perspective-n-Point Camera Pose Estimation"

[3] "A Consistently Fast and Globally Optimal Solution to the Perspective-n-Point Problem" by G. Terzakis and M.Lourakis.

[4] OpenCV, *Camera Calibration — OpenCV-Python Tutorials 1 documentation*, Available at: `https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html`.

[5] Ultralytics, YOLO (*You Only Look Once*. Available at: `https://github.com/ultralytics`.