Vislab, May 18
VISUAL ODOMETRY
Pure rotation correspondence between a camera and a inertial sensor

Coordinators: Alexandre Bernardino & John Van Opstal
Authors: Mariana Martins & Carlos Aleluia

1. INTRODUCTION

To achieve the goal of the main project, both a camera and an inertial sensor will be essential. This report covers what was needed to get a correct correspondence between the same rotation instances on the camera and the sensor, detailing all the prior knowledge needed to accomplish this.
The camera used is a uEye model UI-1220LE-C-HQ[1] and the sensor is a LPMS-CU which is a miniature inertial measurement unit (IMU) / attitude and heading reference system (AHRS)[2]. To implement this the Open Computer Vision (opencv) library, the ueye library and the LPMS sensor library were used along with C++ programming language.

2. CAMERA MODEL

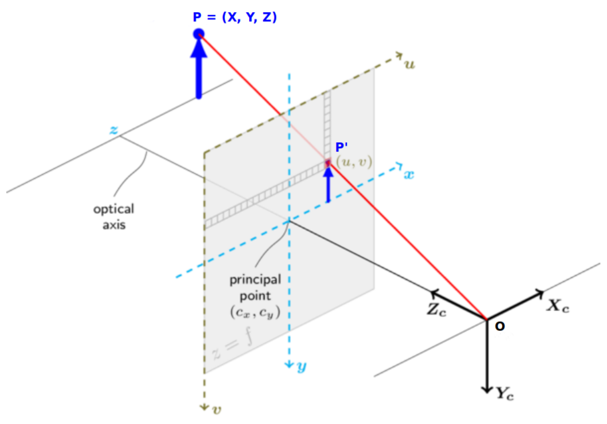

Fig. 1: Pinhole camera model with the coordinate system from opencv library

To compute the rotation between images, it is important to first understand the camera model used. Figure 1 illustrates this model using the coordinate system from the opencv library. Here, it's possible to establish a one-to-one mapping between points on the 3D object and the image, since the light rays of one point pass through the camera pinhole only. The principal point shown above is the center of the virtual image plane and the center of the camera or pinhole is represented by O. The distance between them is the focal length f. $P = [X, Y, Z]^T$ is a point of the 3D object and $P' = [u, v]^T$ is that point projected onto the virtual image plane. Through similar triangles, the following mapping on expression 1 can be obtained.

$$u = \frac{f \cdot X}{Z}, v = \frac{f \cdot Y}{Z} \tag{1}$$

Since the virtual image plane is expressed in pixels, the conversion from the metric system must also be accounted for. This ratio can be different for u and v since the camera might not have square pixels. The origin of the image coordinates O is set by the optical axis whereas the virtual image plane has its origin at the upper left corner. Thus, these 2D points are offset by $[c_x, c_y]^T$ (in pixels). Finally, there can also be a skew coefficient, which is non-zero if u and v axes are not orthogonal to each other. Having considered the ratio said above (f_x and f_y are in pixels), the so-called intrinsic parameters matrix is shown on expression 2.

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

If the camera is oriented arbitrarily, then its rotation matrix R [3 x 3] and translation vector T [3 x 1] must be considered to position it as is on figure 1. These are called the extrinsic parameters. Summing it all up on equations 3 and 4, the ideal pinhole camera model is defined,

$$P' = K \cdot [R|T] \cdot P_{arbitrary} \tag{3}$$

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \cdot [R|T] \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4}$$

where λ is a scale factor and the matrices are in homogeneous coordinates.

However, this ideal model does not have a lens and so what is above doesn't consider possible radial or tangential distortion which real cameras usually suffer from. Radial distortion can be modeled by the coefficients k_1, k_2 and k_3 or

[1] https://www.prolinx.co.jp/supplier/IDS/uEye_Manual_En/index.html?hw_modellbezeichnungen.html
[2] https://www.lp-research.com/lpms-cu/

more in equations 5 and 6,

$$u_{distorted} = u \cdot (1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6) \quad (5)$$

$$v_{distorted} = v \cdot (1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6) \quad (6)$$

where $r^2 = u^2 + v^2$.

As for tangential distortion, which occurs when the image plane and the lens are not parallel, the coefficients p_1 and p_2 account for it as seen on equations 7 and 8.

$$u_{distorted} = u + [2 \cdot p_1 * u * v + p_2 \cdot (r^2 + 2 * x^2)] \quad (7)$$

$$v_{distorted} = v + [p_1 \cdot (r^2 + 2 * y^2) + 2 \cdot p_2 * u * v] \quad (8)$$

3. CAMERA CALIBRATION

As matrix K remains the same as long as the focal length is not changed, it can be computed and used when necessary. When calibrating the camera, it is possible to estimate the extrinsic and intrinsic parameters by solving equation 3 using a calibration rig with a simple geometric pattern, for example a chessboard, with known dimensions.
By finding corresponding points between the 3D world chessboard (P_i) and its image (P_i'), it is possible to establish a linear system of equations. Since each point provides 2 equations and there are 11 parameters to discover from $K[R|T]$, only 6 points are needed at minimum. This system can be solved with Single Value Decomposition (SVD). Afterwards, it is necessary to recover each matrix, K, R and T, which can easily be done since the coefficients are all related.
Regarding, the distortion coefficients, it is necessary to solve a non-linear system of equations that can be simplified, but will not be covered here.

4. METHODOLOGY

The camera calibration was done using a method from opencv library - calibrateCamera[3] - to which 65 chessboard images were provided. The chessboard has 8x6 squares of 28.5 mm and the images are 640 by 480 pixels. From here the matrix K was obtained and used on the following calculations. The radial and tangential distortion coefficients were also obtained but here it is redundant to use them.

The goal was to determine the rotation between images and compare it to the rotation of the inertial sensor at those same instances. So, firstly for each image, the keypoints (interest points in the image that make it unique) were found through the Speeded Up Robust Features (SURF)[4] algorithm (provided by opencv library Features2D). The quatity of keypoints is determined by a hessian threshold. A bigger threshold removes low fidelity keypoints.
Afterwards, it was done the match between the keypoints of both images using the Fast Approximate Nearest Neighbor Search (FLANN)[5] algorithm (provided by the opencv library FLANN). The resulting matches were selected as good matches based on their keypoints pair distance. It was considered a good match if its keypoint distance was less than two times the minimum distance for all the matches, this helps at reducing the false positives.
To obtain a balance of a strong enough quantity of keypoints, yet that generate good matches, values for the hessian threshold in the range of 200 and 1000 are tested (incrementing 100) at each execution. The one used in the end, is the one that generated more good matches.

Now, it is possible to establish a relation between the images and from there determine the rotation. But because this camera does not provide depth information (there is no access to the 3D point coordinates), it is necessary to project the 2D points into a sphere to compute the pure rotation explained ahead. The points obtained from the captured image are $[u, v, 1]$, they are converted into raw/distorted points (equation 9) and projected in an unitary sphere using the scale factor λ as is expressed on equation 10. The relation R of the two images' points is in equation 11.

$$K^{-1} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{raw} \quad (9)$$

$$||(\lambda \cdot u_{raw}, \lambda \cdot v_{raw}, \lambda)|| = 1,\ \lambda = \frac{1}{\sqrt{u_{raw}^2 + v_{raw}^2 + 1}} \quad (10)$$

$$\lambda_2 P_2' = R_{camera} \cdot \lambda_1 P_1' \quad (11)$$

Having done this, it was now applied the solution to the orthogonal procrustes problem[6] to determine R. In greek mythology, this consisted on him making his victims fit his bed by either stretching their limbs or cutting them off. However, here for [3x3] matrices this just consists on finding the orthogonal matrix R which most closely maps P_1' to P_2' as shown on expression 12,

$$min||P_2' - R \cdot P_1'|| \quad (12)$$

where $R^T \cdot R = I$.
This is the same as finding the nearest orthogonal matrix to $M = P_2' \cdot P_1'^T$. Using SVD, $M = U \cdot \Sigma \cdot V^T$ and $R = U \cdot V^T$.

As for the sensor, the rotation matrices (in relation to the sensor base position) for the two instances were captured and

[3] https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html
[4] https://docs.opencv.org/3.4.0/df/dd2/tutorial_py_surf_intro.html
[5] https://docs.opencv.org/3.0-beta/doc/tutorials/features2d/feature_flann_matcher/feature_flann_matcher.html
[6] https://en.wikipedia.org/wiki/Orthogonal_Procrustes_problem

the rotation between them was calculated by equation 13.

$$R_{sensor} = R_2 R_1^{-1} \tag{13}$$

To more easily compare the rotation matrices between the camera R_{camera} and the inertial sensor R_{sensor}, this formalism was converted to Euler Axis and Angle. This parameterizes a rotation in a three-dimensional Euclidean space by two quantities, a unit vector ϵ indicating the direction of an axis of rotation, and an angle ϕ describing the magnitude of the rotation about the axis[7]. The angle's magnitude comes expressed in radians and can be obtained through equation 14.

$$Trace(R) = 1 + 2 \cdot cos(\phi) \tag{14}$$

The coordinate system of the sensor and the opencv library are different as shows on figure 2.

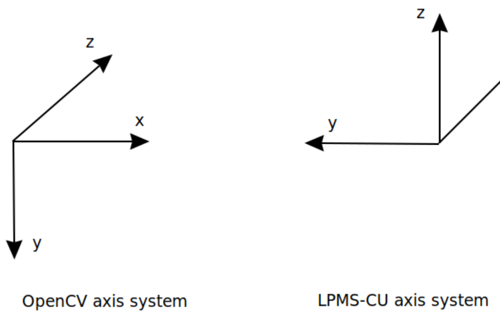

Fig. 2: The two different coordinate systems

The matrix that transforms the opencv coordinates axis into the sensor ones is expressed on 15. From now till the end of the report all coordinates are expressed in the LPMS-CU axis system.

$$\begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \tag{15}$$

5. TEST CASES

In relation to the physical setup used to test the accuracy of this rotations, a paper grid on natural lighting was used, and is drawn on figure 3, the measurements are expressed in centimeters.

[7] https://en.wikipedia.org/wiki/Axisangle_representation

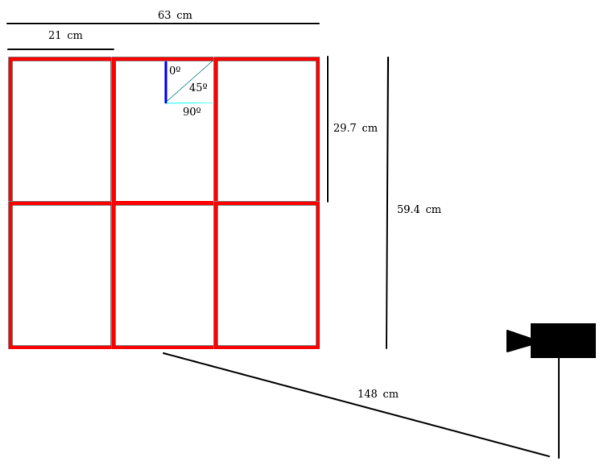

Fig. 3: Grid structure and measurements in centimeters

Having the distance from the camera to the grid (148 cm) and the distance from where the camera's center is pointing at to what is pointing at after rotating, by computing the inverse tangent, it was possible to make rotations in the z axis of 8.1°, 15.8° and 23.1° and in the y axis of 11.3° and 21.9°. The distance between the centers was taken by adjusting certain points to the camera's center at real-time. For the x axis, a vertical stripe was set in the grid (the blue stripe), and rotated 45° and 90°, adjusting the camera to always see it vertically. The sensor was glued and aligned to the camera as shows on figure 4, so it did the same rotations and at the same time.

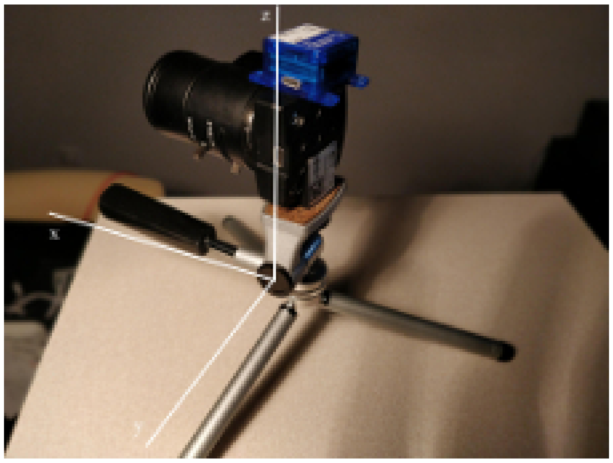

Fig. 4: Camera and sensor setup

The results of the experiences are presented on table 1, 2 and 3. In this three tables, H stands for Hessian threshold value and K for the number of keypoints matched. For each angle tested in each rotation axis, it's presented its the unit vector, the angle of rotation and the angle error (e_{angle}) between the angle meant to be and what actually turned out to be.

Angle: $11.3°$ **(0.19 rad)**

Camera (H=900, K=18)	Sensor
ϵ = (0.14, 0.99, 0.04)	ϵ = (0.37, 0.86, -0.34)
ϕ = 0.17 rad	ϕ = 0.21 rad
e_{angle} = 1.2° (0.02 rad)	e_{angle} = 1.2° (0.02 rad)

Angle: $21.9°$ **(0.38 rad)**

Camera (H=900, K=8)	Sensor
ϵ = (0.09, 0.99, 0.06)	ϵ = (0.45, 0.87, -0.20)
ϕ = 0.33 rad	ϕ = 0.40 rad
e_{angle} = 1.9° (0.05 rad)	e_{angle} = 1.2° (0.02 rad)

Table 1: Rotation in y

Table 1 corresponds to figures 5, 6 and 7. It's a positive rotation around the y-axis. The cross represents the center of the image (at where the camera was pointing at).

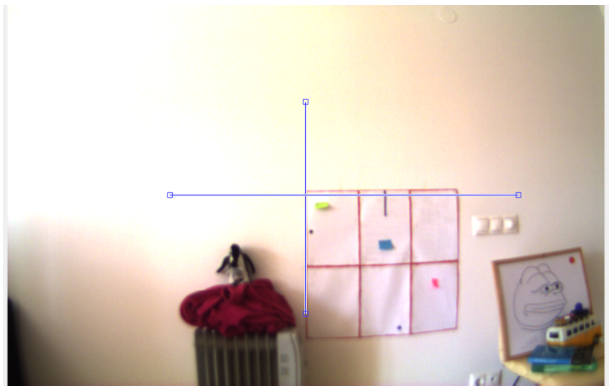
Fig. 5: Initial position ($0°$)

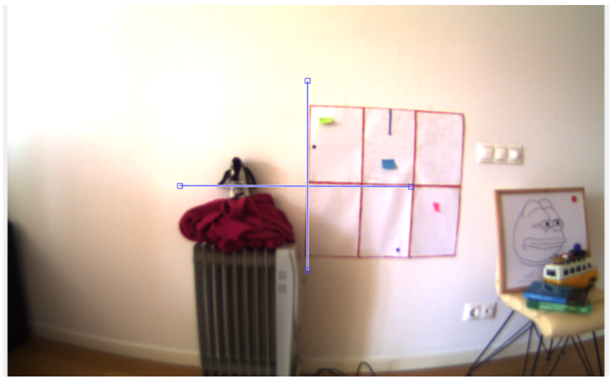
Fig. 6: Positive rotation in turn of y ($11.3°$ in relation to the initial position)

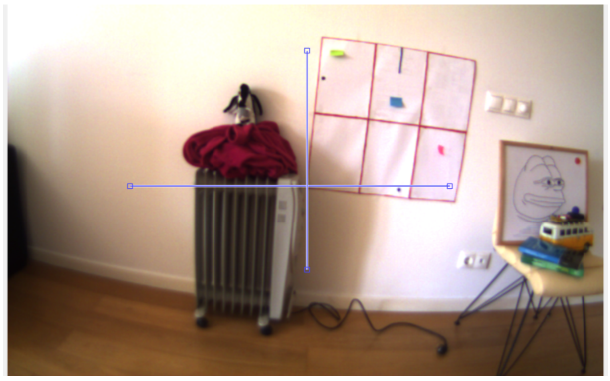
Fig. 7: Positive rotation in turn of y ($21.9°$ in relation to the initial position)

Angle: $8.1°$ **(0.14 rad)**

Camera (H=900, K=12	Sensor
ϵ = (-0.17, -0.08, -0.98)	(-0.003, -0.03, -0.99)
ϕ = 0.11 rad	ϕ = 0.17 rad
$e_{angle} \simeq 1.7°$ (0.03 rad)	$e_{angle} \simeq 1.7°$ (0.03 rad)

Angle: $15.8°$ **(0.28 rad)**

Camera (H=900, K=11)	Sensor
ϵ = (-0.19, -0.37, -0.91)	(0.003, -0.04, -0.99)
ϕ = 0.21 rad	ϕ = 0.31 rad
$e_{angle} \simeq 4.0°$ (0.07 rad)	$e_{angle} \simeq 1.7°$ (0.03 rad)

Angle: $23.1°$ **(0.40 rad)**

Camera (H=900, K=18)	Sensor
ϵ = (0.03, 0.04, -0.99)	(0.01, -0.02, -0.99)
ϕ = 0.33 rad	ϕ = 0.43 rad
$e_{angle} \simeq 4.0°$ (0.07 rad)	$e_{angle} \simeq 1.7°$ (0.03 rad)

Table 2: Rotation in z

Table 2 corresponds to figures 8, 9, 10 and 11. It's a negative rotation around z-axis.

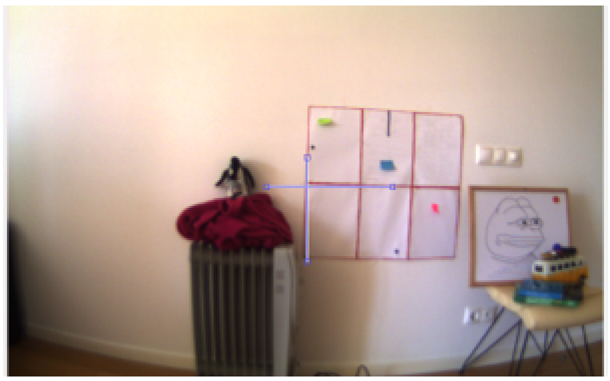
Fig. 8: Initial position ($0°$)

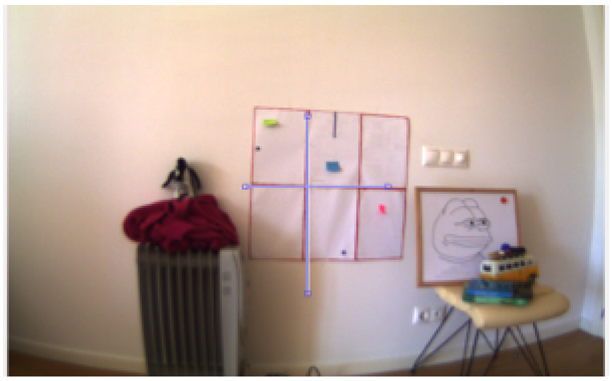

Fig. 9: Positive rotation in turn of z (8.1° in relation to the initial position)

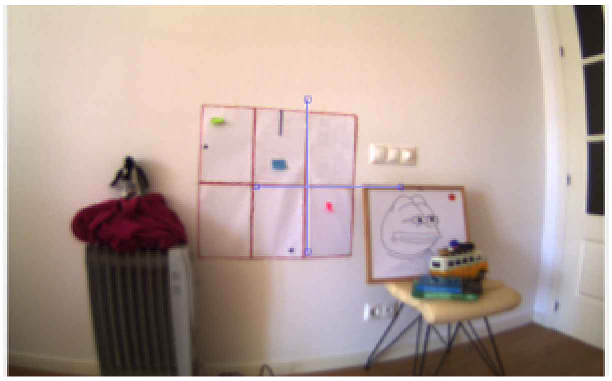

Fig. 10: Positive rotation in turn of z (15.8° in relation to the initial position)

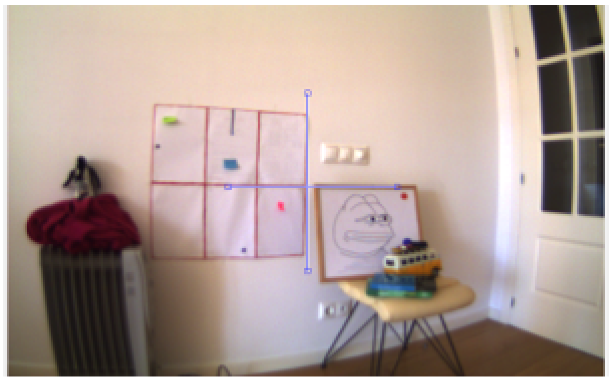

Fig. 11: Positive rotation in turn of z (23.1° in relation to the initial position)

Angle: 45° **(0.78 rad)**	
Camera (H=900, K=9)	Sensor
ϵ = (0.99, -0.05, -0.08)	ϵ = (0.76, 0.65, 0.001)
ϕ = 0.79 rad	ϕ = 0.77 rad
e_{angle} = 0.6° (0.01 rad)	e_{angle} = 0.6° (0.01 rad)

Angle: 90° **(1.57 rad)**	
Camera (H=900, K=28)	Sensor
ϵ = (0.99, -0.004, 0.02)	ϵ = (0.71, 0.70, 0.00)
ϕ = 1.66 rad	ϕ = 1.67 rad
e_{angle} = 5.2° (0.09 rad)	e_{angle} = 5.7° (0.1 rad)

Table 3: Rotation in x

Table 3 corresponds to figures 12, 13 and 14. It's a positive rotation around x-axis.

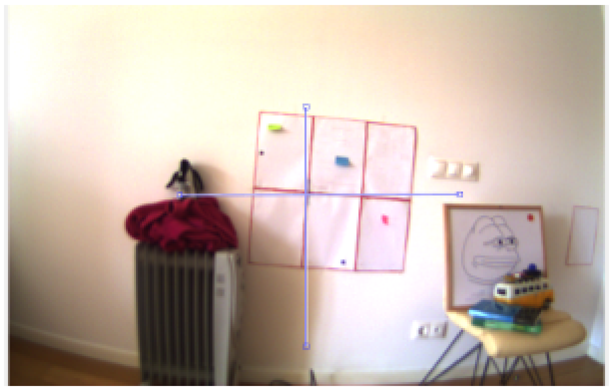

Fig. 12: Initial position (0°)

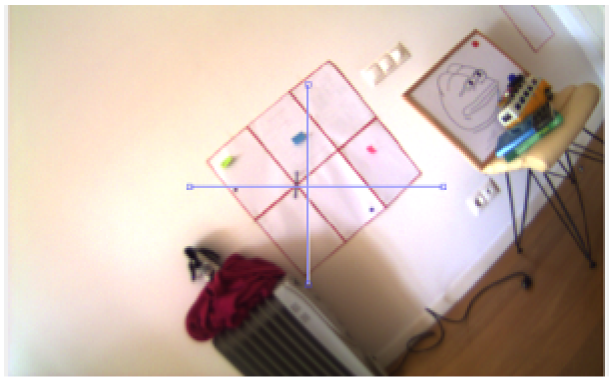

Fig. 13: Positive rotation in turn of x (45° in relation to the initial position)

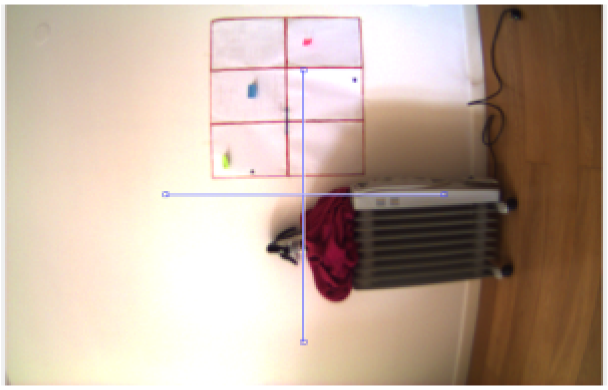

Fig. 14: Positive rotation in turn of x (90° in relation to the initial position)

6. RESULTS & DISCUSSION

Regarding the camera calibration, from the set of 65 images, 57 were successes, which means that in those the chessboard was found. From that, the intrinsic parameters obtained were the shown below on expression 16.

$$K = \begin{bmatrix} 489 & 0 & 324 \\ 0 & 489 & 213 \\ 0 & 0 & 1 \end{bmatrix} \tag{16}$$

In all the tables above, the value used for the hessian threshold was 900, meaning it was the value that allowed the biggest number of good matches in every experience. This is very dependent on the environment (lighting and objects), since some environments didn't have enough keypoints and others had a lot.

On table 1 and 3's experiences, the camera did no spin around itself because it was fixed on a support as shows on figure 4, this will be one of the next study motives - determining the parameters of the support that affect the rotations. When looking at the camera unit vectors on the respective tables this is not noticeable, since they only regard rotation, not translation, and when processing the images the supposed rotation did indeed happen. But looking at the figures 12, 13 and 14, the translation occurred is visible, since the camera center is not aligned with the center of the stripe no more. In figures, 5, 6 and 7, this is not so easy to see, but what happens is the camera becoming closer to the grid, meaning the distance of 148 cm is not so true. However, the sensor's unit vector, although having one coordinate that was clearly the axis (the closest to 1), the rest of the coordinates were messed up. This is because when translating the sensor, its rotation matrices change.
Table 2's experience was instead very accurate, since the support's axis of rotation is exactly the sames as the camera's.

As a final note, while doing this experiences, when trying to put objects at a close distance to the camera, a possible translation effect did not occur, since the Procrustes Analysis was computed as a pure rotation.