Camera calibration is an essential process in computer vision that aims to determine the intrinsic and extrinsic parameters of a camera.

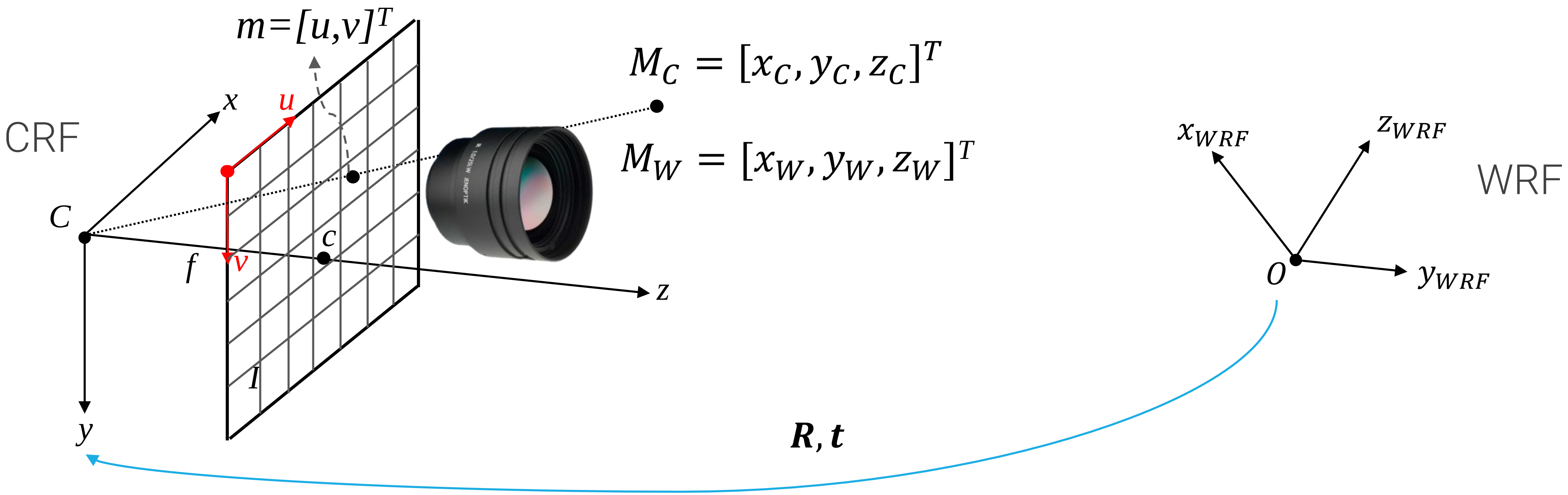
These parameters are crucial for accurately mapping 3D points in the scene to 2D points in the image

Lecture 2
Camera Calibration

IMAGE PROCESSING AND COMPUTER VISION – PART 2

SAMUELE SALTI

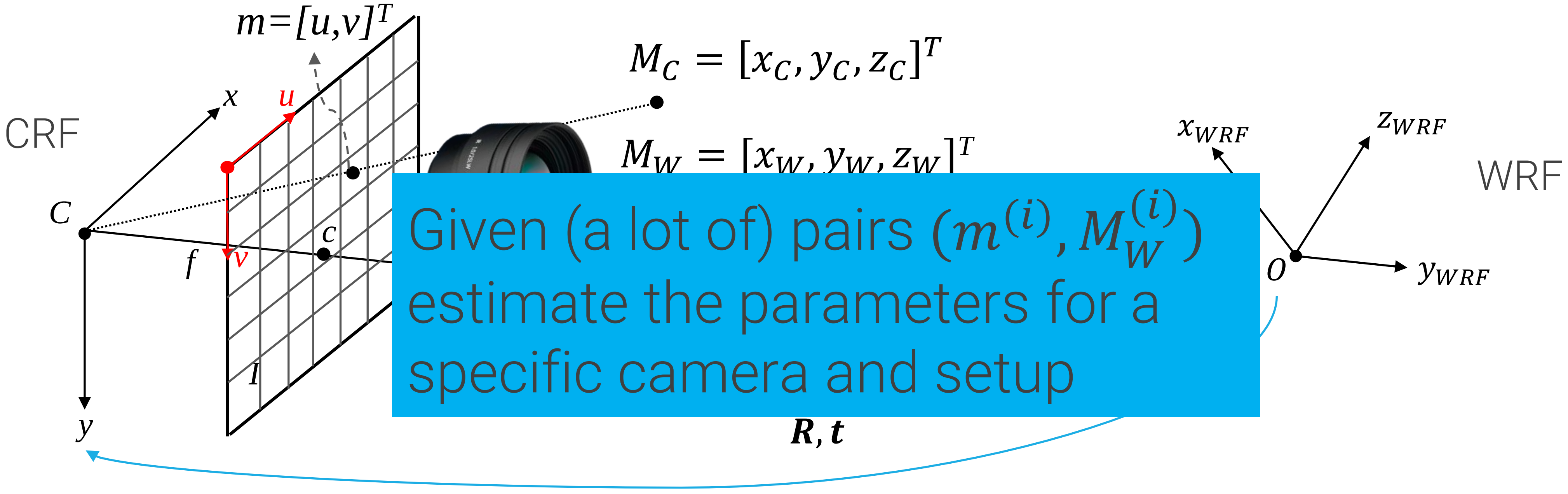
Complete camera model



2D point [pixels]		3D point in CRF [Metric, e.g. mm]		3D point in WRF [Metric, e.g. mm]
$m = \begin{bmatrix} u \\ v \end{bmatrix}$	← 4 Intrinsic parameters Distortion coefficients	$M_C = \begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix}$	← 6 Extrinsic parameters	$M_W = \begin{bmatrix} x_W \\ y_W \\ z_W \end{bmatrix}$

Camera calibration

given projected points and WRF points (their coordinates) estimate the parameters involved in the image formation process



Given (a lot of) pairs $(m^{(i)}, M_W^{(i)})$ estimate the parameters for a specific camera and setup

Known

2D point [pixels]

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

4 Intrinsic parameters
Distortion coefficients

3D point in CRF [Metric, e.g. mm]

$$M_C = \begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix}$$

Unknown

6 Extrinsic parameters

3D point in WRF [Metric, e.g. mm]

$$M_W = \begin{bmatrix} x_W \\ y_W \\ z_W \end{bmatrix}$$

Known

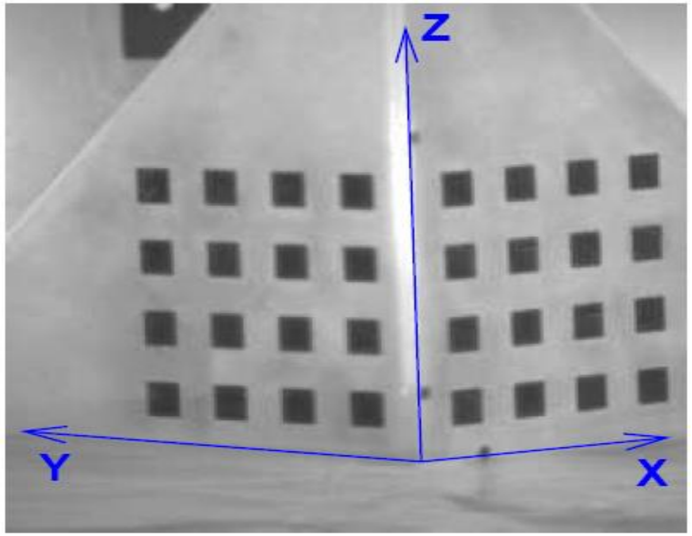
how retrieve correspondances between m and M?
- using calibration patterns

Calibration patterns

the calibration depends on the type of calibration objects and the calibration patterns used

Camera calibration approaches can be split into two main categories:

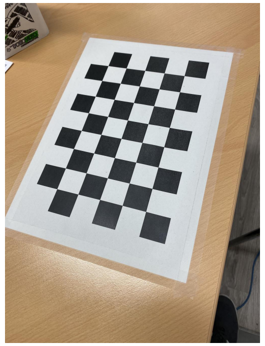
- Those relying on a single image of a 3D calibration object
  - featuring several (at least 2) planes containing a known pattern like a checkerboard which has a recognizable pattern
- Those relying on several (at least 3) different images of one given planar pattern from different positions and angles

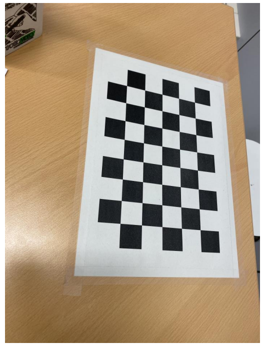


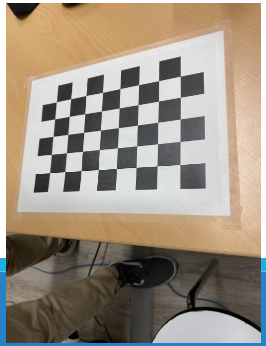
In practice, it is difficult to build accurate targets containing multiple planes, while an accurate planar target can be attained rather easily

Implementing a camera calibration software requires a significant effort

- the main Computer Vision toolboxes include specific functions (OpenCV, Matlab CC Toolbox)







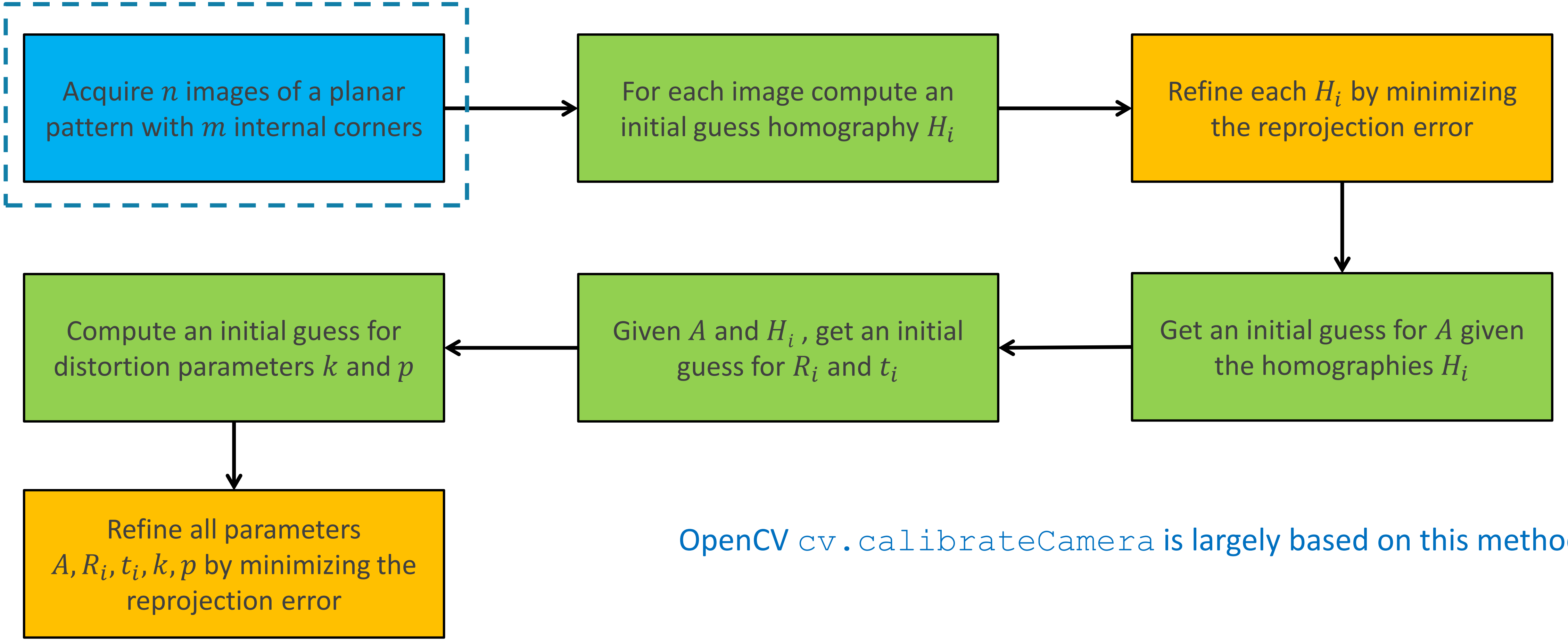
Zhang's Method

green: minimize algebraic error

solve a linear system with least square

yellow: minimize geometric error

solve for minimization of L2-norms



OpenCV `cv.calibrateCamera` is largely based on this method

Zhengyou Zhang. A flexible new technique for camera calibration. IEEE Trans. on PAMI, 2000.
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr98-71.pdf

1° step: acquire input images

Calibration pattern

ambiguous: if transformation applied, no clear distinction
e.g. chessboard if rotated we cannot distinguish differences wrt the two views

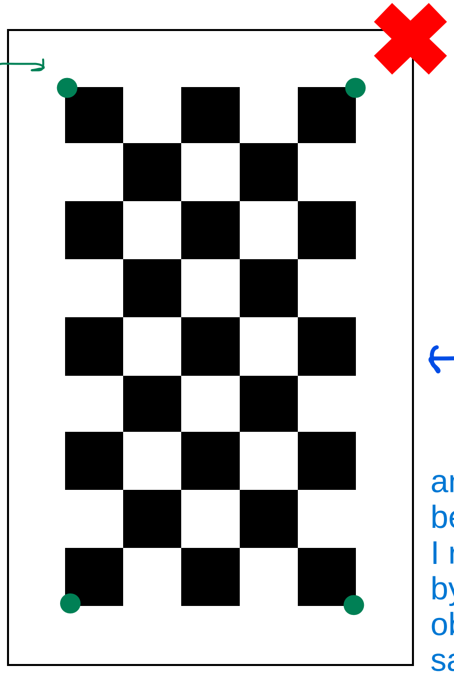
Given a chessboard pattern, we know:

- The number of internal corners of the pattern, usually **odd along one dimension and even along the other to remove rotation ambiguities**.
- The size of the squares that form the pattern (in mm, cm...)

doing this we retrieve the 2d coordinates we need (m)

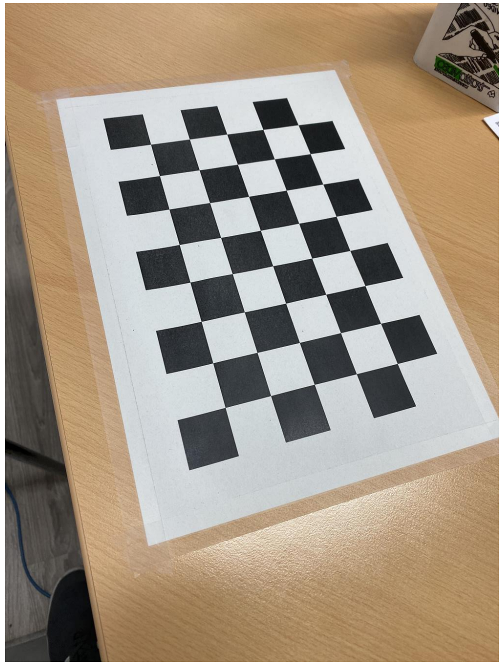
Internal corners can be detected easily by standard algorithms (e.g. the Harris corner detector)

internal corners

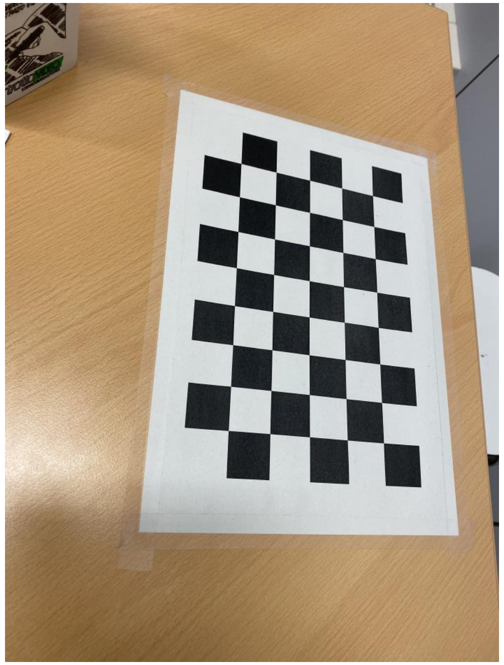


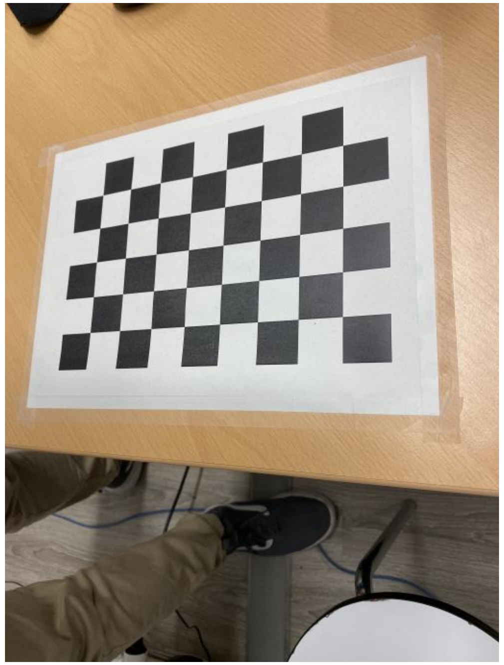
Ambiguous

ambiguous because if I rotate it by 180° I obtain the same picture



these images are easier to disambiguate after transformations like rotations





in the previous slide we have found 2D coordinates (m) now we want M_W, the 3D coords from WRF

3D coordinates

attach the WRF to the pattern itself in order to conveniently define the 3d points

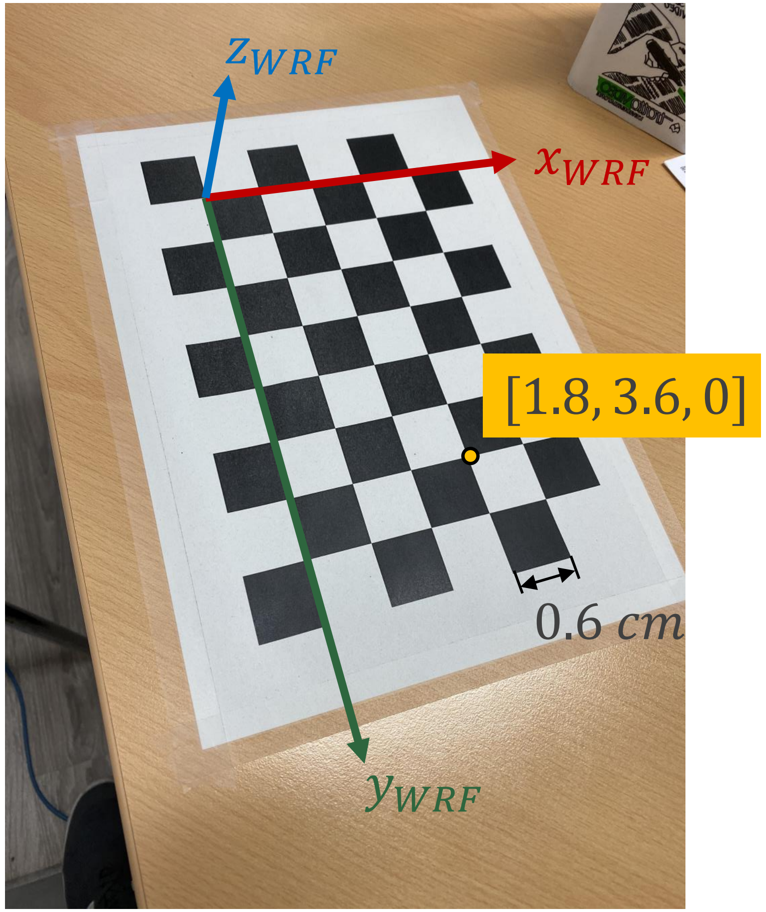
The World Reference Frame can be conveniently defined.

In an unambiguous pattern, the WRF can be defined so that

- it has its origin always in the same corner (e.g., the one next to the dark square on the right of the chessboard if both dark squares are on top);
- its plane $z = 0$ is the pattern itself => the third coordinate is always 0;
- the x, y axes are aligned to the chessboard (e.g., x along the short side and y along the long one).

Given such rules and the known square side, it is possible to define 3D coordinates for all corners in an image of the pattern.

This setup simplifies the problem because all the points on the pattern lie on a single plane, and we can set the z-coordinate of these points to 0. This assumption allows us to work with 2D coordinates on the plane, which can be extended to 3D homogeneous coordinates.



in order to estimate properly the coefficients of radial distortion, tha patter should be appear in the corners, because I can get correpsonadances only wher tha pattern is and I can estimate params

Extrinsic parameters

the WRF is always the same
- what is different is the position of the WRF wrt the Camera, according to different translations and rotations in each image, because we have to take different pictures where the pattern is located at differnt positions wrt the camera

since we collect several images

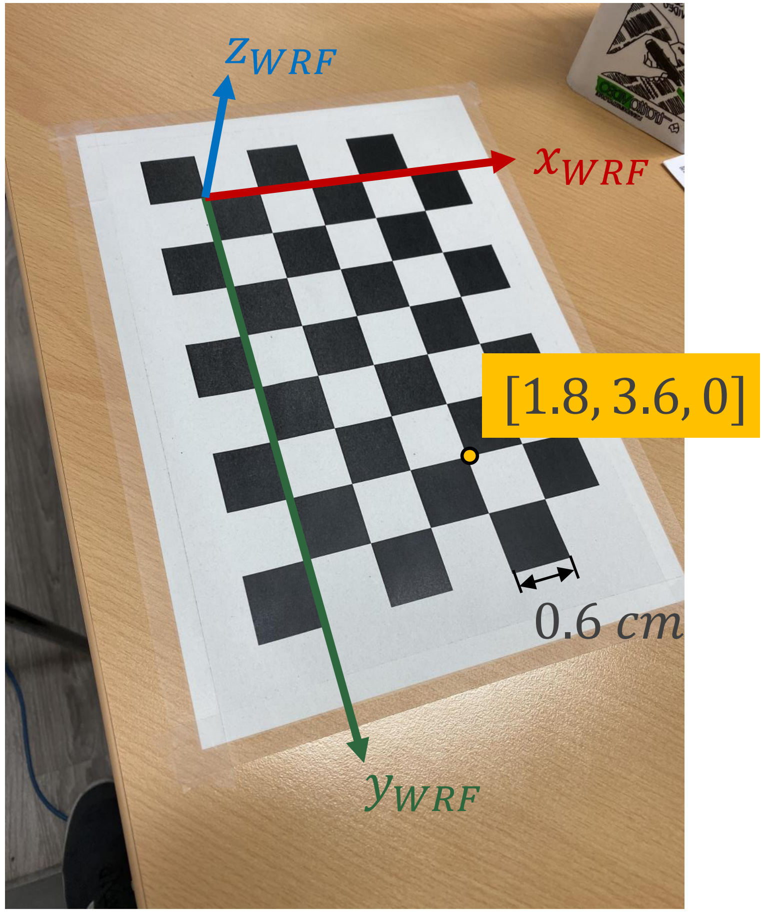
The World Reference Frame is different for each calibration image.

The $[\boldsymbol{R}\ \boldsymbol{t}]$ are estimated wrt the World Reference Frame attached to the target, which moves with the pattern. what it change is [R t] !

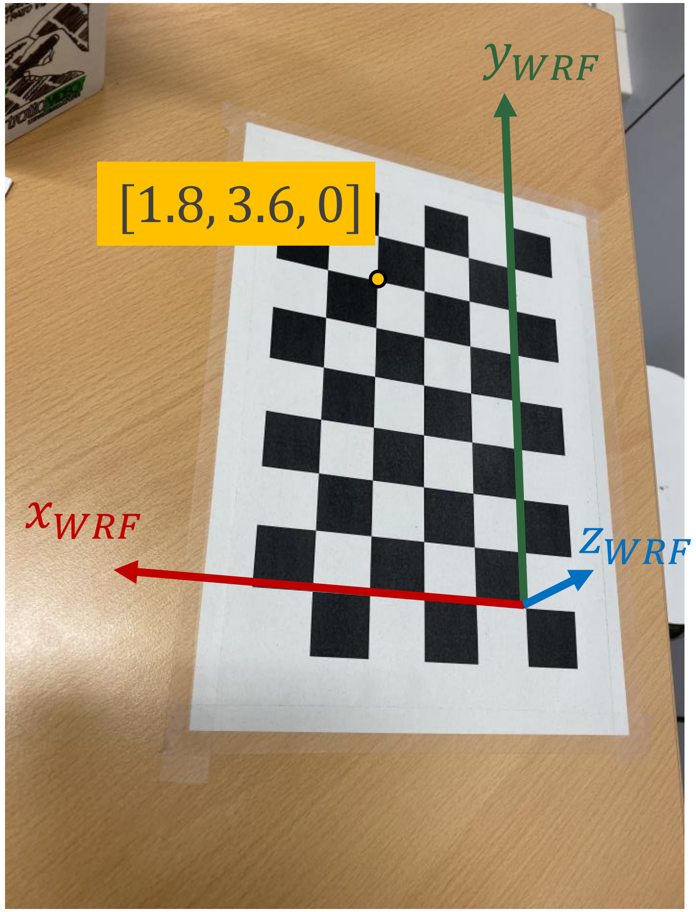
Therefore, we estimate as many extrinsic matrices as the number of images used for calibration (usually 10 to 20 images of the pattern)

we have to estimate, if we collect 'n' frames, '1' intrinsic matrix and 'n' extrinsic matrices

$\boldsymbol{R_1}, \boldsymbol{t_1}$

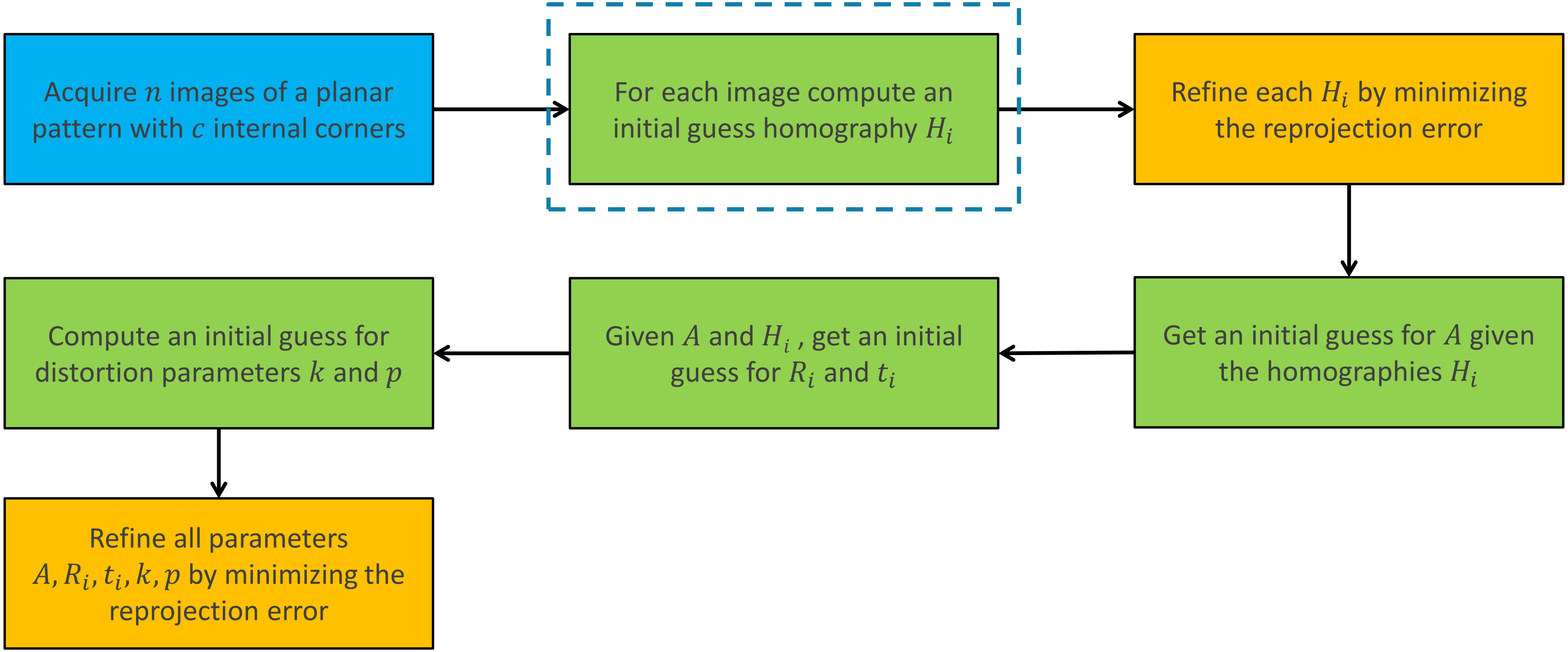


$\boldsymbol{R_2}, \boldsymbol{t_2}$



remember: PPMs are always the same in the image plane

Zhang's Method



Zhengyou Zhang. A flexible new technique for camera calibration. IEEE Trans. on PAMI, 2000.

convert both the 2d and 3d coords to homogeneous coordinates

P as a Homography

For each image, compute the homography matrix H that relates the points on the planar pattern in the WRF to the points in the image plane.

Due to the choice of the WRF associated with calibration images, in each of them we consider only 3D points with $z = 0$

Accordingly, the PPM for points on the pattern can be simplified to a 3x3 matrix:

We look at a planar scene

$$k\tilde{\boldsymbol{m}} = k\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{P}\tilde{M}_W = \begin{bmatrix} p_{1,1} & p_{1,2} & \cancel{p_{1,3}} & p_{1,4} \\ p_{2,1} & p_{2,2} & \cancel{p_{2,3}} & p_{2,4} \\ p_{3,1} & p_{3,2} & \cancel{p_{3,3}} & p_{3,4} \end{bmatrix}\begin{bmatrix} x \\ y \\ \boxed{0} \\ 1 \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,4} \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \boldsymbol{H}\tilde{w}$$

trans-
lation

the 3d coords of the point in the planar pattern

Such a transformation, denoted here as $\boldsymbol{H}$, is known as homography and represents a general transformation between projective planes the world plane and the image plane

$\boldsymbol{H}$ can be thought of as a simplification of $\boldsymbol{P}$ in case the imaged object is planar

The homography matrix H transforms points on the world plane (pattern plane) to the image plane. It can be viewed as a plane-to-plane projective transformation, similar to how the projection matrix (PPM) relates 3D points to 2D image points in general.

Estimating H_i (DLT algorithm)

Given the *i-th* image of a pattern with c corners, we can write 3 linear equations for each corner j where:

- 3D coordinates are known due to the WRF definition
- 2D coordinates are known due to corners having been detected in the i-*th* image
- the unknowns are the 9 elements in $\boldsymbol{H_i}$

Unknowns

break down H
into three rows

u in the frame i
corner j

$$\tilde{m}_{ij} \equiv \begin{bmatrix} u_{ij} \\ v_{ij} \\ 1 \end{bmatrix} \equiv \begin{bmatrix} p_{i,1,1} & p_{i,1,2} & p_{i,1,4} \\ p_{i,2,1} & p_{i,2,2} & p_{i,2,4} \\ p_{i,3,1} & p_{i,3,2} & p_{i,3,4} \end{bmatrix} \begin{bmatrix} x_j \\ y_j \\ 1 \end{bmatrix} \equiv \boldsymbol{H_i}\tilde{w}_j \equiv \begin{bmatrix} -\boldsymbol{h}_{i1}^T - \\ -\boldsymbol{h}_{i2}^T - \\ -\boldsymbol{h}_{i3}^T - \end{bmatrix} \tilde{w}_j \equiv \begin{bmatrix} \boldsymbol{h}_{i1}^T \tilde{w}_j \\ \boldsymbol{h}_{i2}^T \tilde{w}_j \\ \boldsymbol{h}_{i3}^T \tilde{w}_j \end{bmatrix}$$

dot product between
hi and wj

Coordinates of a
corner in the image

Coordinates of the same
corner in the WRF

what changes is [R t]

Stacking $3c$ such equations for the c corners we get a system of equations, but…

each corner will give me 3 equations

How do we solve a system of equations in a projective space?

then up to an unknown scale factor?

we basically have to solve a linear system

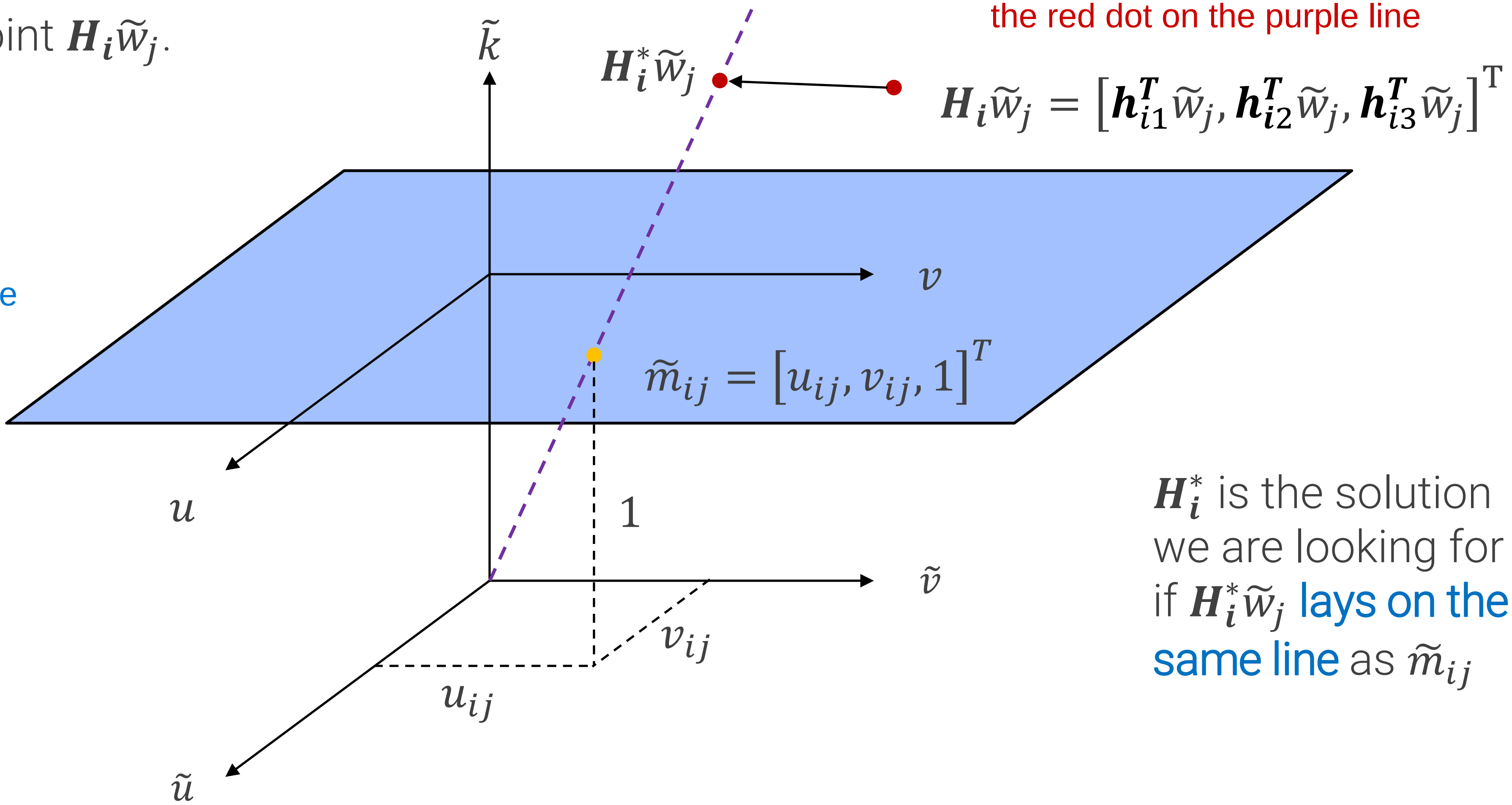
In projective geometry, two points are considered equivalent if they lie on the same line, meaning they are collinear

When are two 3D points equivalent in $\mathbb{P}^2$?

so that the transformation that maps the two dots up to an unknown scalar factor

we have to find the transformation H that put the red dot on the purple line

By deciding a value for the 9 entries of $\boldsymbol{H_i}$ we move the point $\boldsymbol{H_i}\tilde{w}_j$.



we want that red dot and the yellow one to be the same

$\boldsymbol{H_i^*}$ is the solution we are looking for if $\boldsymbol{H_i^*}\tilde{w}_j$ lays on the same line as $\tilde{m}_{ij}$

Estimating H_i (DLT algorithm)

the cross product measures the area of the parallelogram between two vectors

Two points lay on the same line if their cross product is the zero vector.

cross product

$$\tilde{m}_{ij} \equiv \boldsymbol{H}_i \tilde{w}_j \Rightarrow \tilde{m}_{ij} \times \boldsymbol{H}_i \tilde{w}_j = \boldsymbol{0} \Rightarrow \tilde{m}_{ij} \times \boldsymbol{H}_i \tilde{w}_j = \begin{bmatrix} u_{ij} \\ v_{ij} \\ 1 \end{bmatrix} \times \begin{bmatrix} \boldsymbol{h}_{i1}^T \tilde{w}_j \\ \boldsymbol{h}_{i2}^T \tilde{w}_j \\ \boldsymbol{h}_{i3}^T \tilde{w}_j \end{bmatrix} = \begin{bmatrix} v_{ij}\, \boldsymbol{h}_{i3}^T \tilde{w}_j - \boldsymbol{h}_{i2}^T \tilde{w}_j \\ \boldsymbol{h}_{i1}^T \tilde{w}_j - u_{ij}\, \boldsymbol{h}_{i3}^T \tilde{w}_j \\ u_{ij}\, \boldsymbol{h}_{i2}^T \tilde{w}_j - v_{ij}\, \boldsymbol{h}_{i1}^T \tilde{w}_j \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\boxed{\boldsymbol{h}_k^T \tilde{w}_j = \tilde{w}_j^T \boldsymbol{h}_k}$$

3x9 matrix

9x1 column vector

$$\begin{bmatrix} \boldsymbol{0}_{3\times1}^T & -\tilde{\boldsymbol{w}}_j{}^T & v_{ij}\, \tilde{\boldsymbol{w}}_j{}^T \\ \tilde{\boldsymbol{w}}_j{}^T & \boldsymbol{0}_{3\times1}^T & -u_{ij}\, \tilde{\boldsymbol{w}}_j{}^T \\ -v_{ij}\, \tilde{\boldsymbol{w}}_j{}^T & u_{ij}\, \tilde{\boldsymbol{w}}_j{}^T & \boldsymbol{0}_{3\times1}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{h}_{i1} \\ \boldsymbol{h}_{i2} \\ \boldsymbol{h}_{i3} \end{bmatrix} = \boldsymbol{0}_{3\times1}$$

because each h_i is a 3x1 vector

Only 2 equations are linearly independent (multiply first one by $-u$ and second one by $-v$, then sum them to get the third one)

$$\begin{bmatrix} \boldsymbol{0}_{3\times1}^T & -\tilde{\boldsymbol{w}}_j{}^T & v_{ij}\, \tilde{\boldsymbol{w}}_j{}^T \\ \tilde{\boldsymbol{w}}_j{}^T & \boldsymbol{0}_{3\times1}^T & -u_{ij}\, \tilde{\boldsymbol{w}}_j{}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{h}_{i1} \\ \boldsymbol{h}_{i2} \\ \boldsymbol{h}_{i3} \end{bmatrix} = \boldsymbol{0}_{2\times1}$$

2x9 matrix

Estimating H_i (DLT algorithm)

Ax = 0 form

more equations than unknowns

Given c corners, we can create a homogeneous, overdetermined linear system of equations

$$\begin{bmatrix} \mathbf{0}_{3\times1}^T & -\widetilde{\boldsymbol{w}}_1{}^T & v_{i1}\,\widetilde{\boldsymbol{w}}_1{}^T \\ \widetilde{\boldsymbol{w}}_1{}^T & \mathbf{0}_{3\times1}^T & -u_{i1}\,\widetilde{\boldsymbol{w}}_1{}^T \\ \vdots & \vdots & \vdots \\ \mathbf{0}_{3\times1}^T & -\widetilde{\boldsymbol{w}}_c{}^T & v_{ic}\,\widetilde{\boldsymbol{w}}_c{}^T \\ \widetilde{\boldsymbol{w}}_c{}^T & \mathbf{0}_{3\times1}^T & -u_{ic}\,\widetilde{\boldsymbol{w}}_c{}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{h}_{i1} \\ \boldsymbol{h}_{i2} \\ \boldsymbol{h}_{i3} \end{bmatrix} = \mathbf{0}_{2c\times1} \Rightarrow \boldsymbol{L}_i \boldsymbol{h}_i = \mathbf{0}$$

overdetermined systems have always one exact solution, that is 0

9x1 column vector

2cx9 matrix

To avoid the trivial solution $\boldsymbol{h} = \mathbf{0}$ we look for solutions with an additional constraint, e.g., $||\boldsymbol{h}|| = 1$.

the norm of h equal to one to avoid 0 solution trivial

generally, in strict math terms, this is an impossible system to solve
then we relax our constraints saying that H doesn't have to map perfectly the points onto the two different projective planes, but as close as possible, geometrically speaking.

algebrically, we solve this problem using Least Square, as we see in the following slide

Singular Value Decomposition

The solution $\boldsymbol{h}^*$ is found by minimizing the norm of the vector $\boldsymbol{L_i h_i}$

it becomes a Least Square problem

$$\boldsymbol{h}_i^* = \underset{\boldsymbol{h}_i \in \mathbb{R}^9}{\operatorname{argmin}} \|\boldsymbol{L}_i \boldsymbol{h}_i\| \;\; \boldsymbol{s.t.} \, \|\boldsymbol{h}_i\| = 1$$

It is known from linear algebra that the solution to such problem can be found via Singular Value Decomposition of $\boldsymbol{L_i}$. In particular, the solution is $\boldsymbol{h}_i^* = \boldsymbol{v_9}$, i.e., the last column of $\boldsymbol{V_i}$

every matrix, which transforms between spaces, can be decomposded with an SVD:

- rotate to a convenient ref. sys
- scale axis independently
- rotate to another position

2mx9 diagonal matrix

sigma

2mx9 matrix

$$\boldsymbol{L_i} = \boldsymbol{U_i} \; \boldsymbol{D_i} \; \boldsymbol{V_i^T}$$

2mx2m orthonormal matrix

9x9 orthonormal matrix

every image will have a different homography

Zhang's Method

so DLT gave an initial guess for Hi

the reprojection error is what we have interest into. It quantifies how accurately the estimated camera parameters and 3D world coordinates can reproduce the observed image points.

Acquire n images of a planar pattern with c internal corners → For each image compute an initial guess homography H_i → Refine each H_i by minimizing the reprojection error → Get an initial guess for A given the homographies H_i → Given A and H_i , get an initial guess for R_i and t_i → Compute an initial guess for distortion parameters k and p → Refine all parameters A, R_i, t_i, k, p by minimizing the reprojection error

Zhengyou Zhang. A flexible new technique for camera calibration. IEEE Trans. on PAMI, 2000.

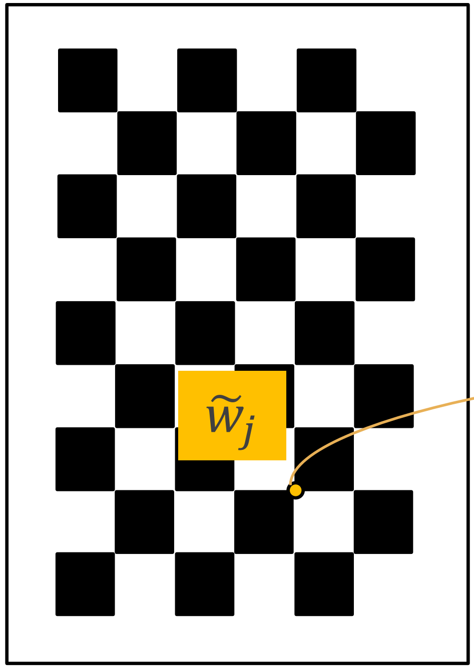
the mapping from wj' and mj cannot be linear, because we have consider also noise and lens distortions -> then the red dot cannot coincide with the blue dot -> then we approximate this mapping ignoring them just to have a starting point

H is a mapping between a pattern (and a corner) and the corner in another image

What error should we minimize?

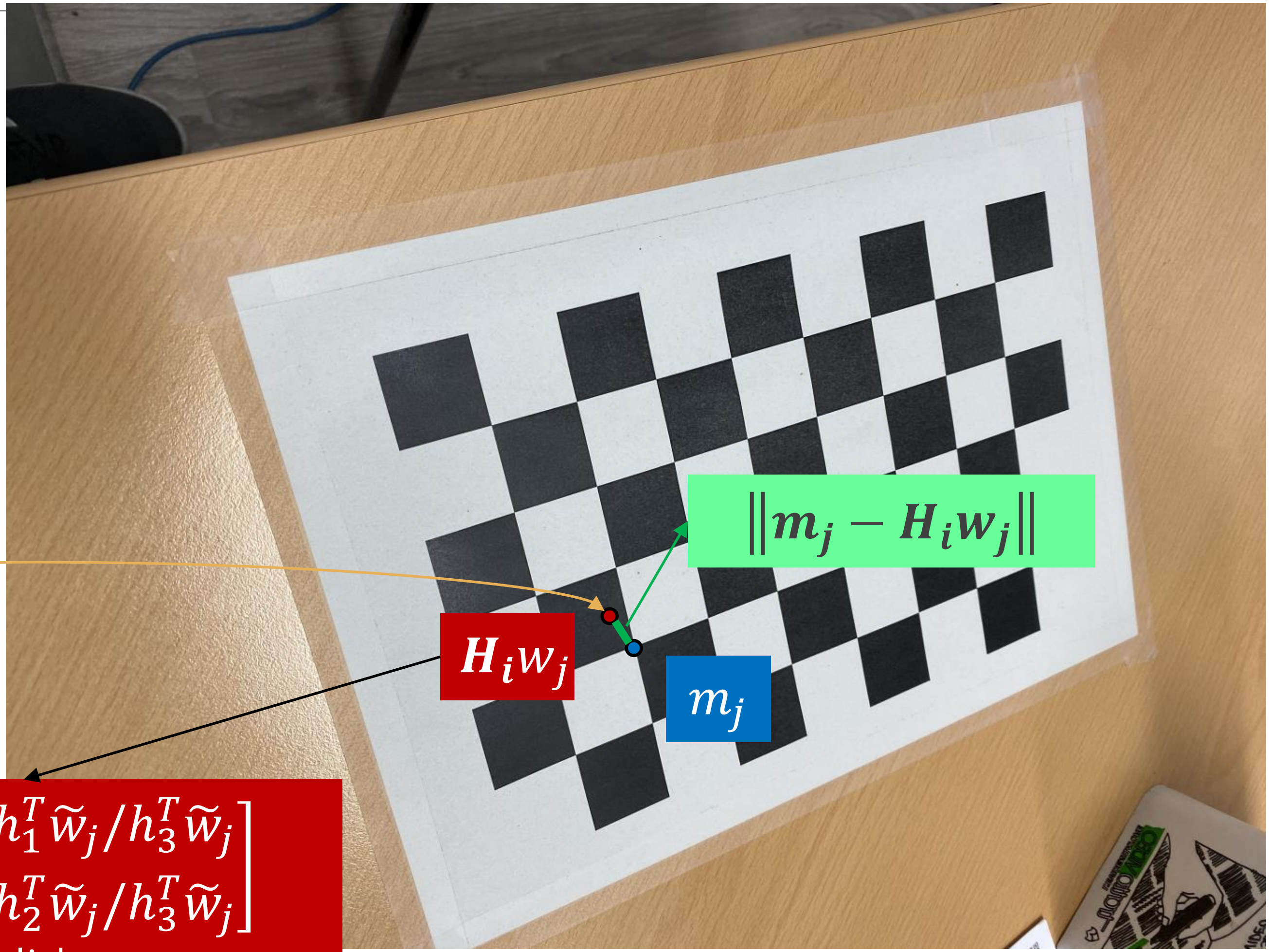
Hiwj is the projected corner and mj is the real corner

I am approximating the mapping as linear because atm I am not considering lenses, which inrtoduce non-lienar distortions



$\widetilde{w}_j$

$\boldsymbol{H_i}$



$\|\boldsymbol{m_j} - \boldsymbol{H_i w_j}\|$

$\boldsymbol{H_i} w_j$

m_j

divide by third-coord to get out the projective space and returning bak to the euclidean

$$\boldsymbol{H_i} w_j = \begin{bmatrix} h_1^T \widetilde{w}_j / h_3^T \widetilde{w}_j \\ h_2^T \widetilde{w}_j / h_3^T \widetilde{w}_j \end{bmatrix}$$

is a 2x1 Euclidean vector

the blue dot has two coordinates computed through harris corner detector

the ideal mapping maps as close as possible the red dot and the blue dot minimize a geometric error

Non-linear refinement of H_i

but it is non-linear, how to solve it?
- iterative algo, like GD, SGD etc.

Given the initial guess for $\boldsymbol{H_i}$, we can refine it by a non-linear minimization problem:

the main drawback that regards iterative algorithms is that they depend too much on the starting point, namely the initial guess

$$\boldsymbol{H_i^*} = \underset{H_i}{\operatorname{argmin}} \sum_{j=1}^{c} \left\| m_{ij} - \boldsymbol{H_i} w_j \right\|^2 \quad i = 1, \dots, n$$

which can be solved for by using an iterative algorithm, like the Levenberg-Marquardt algorithm.

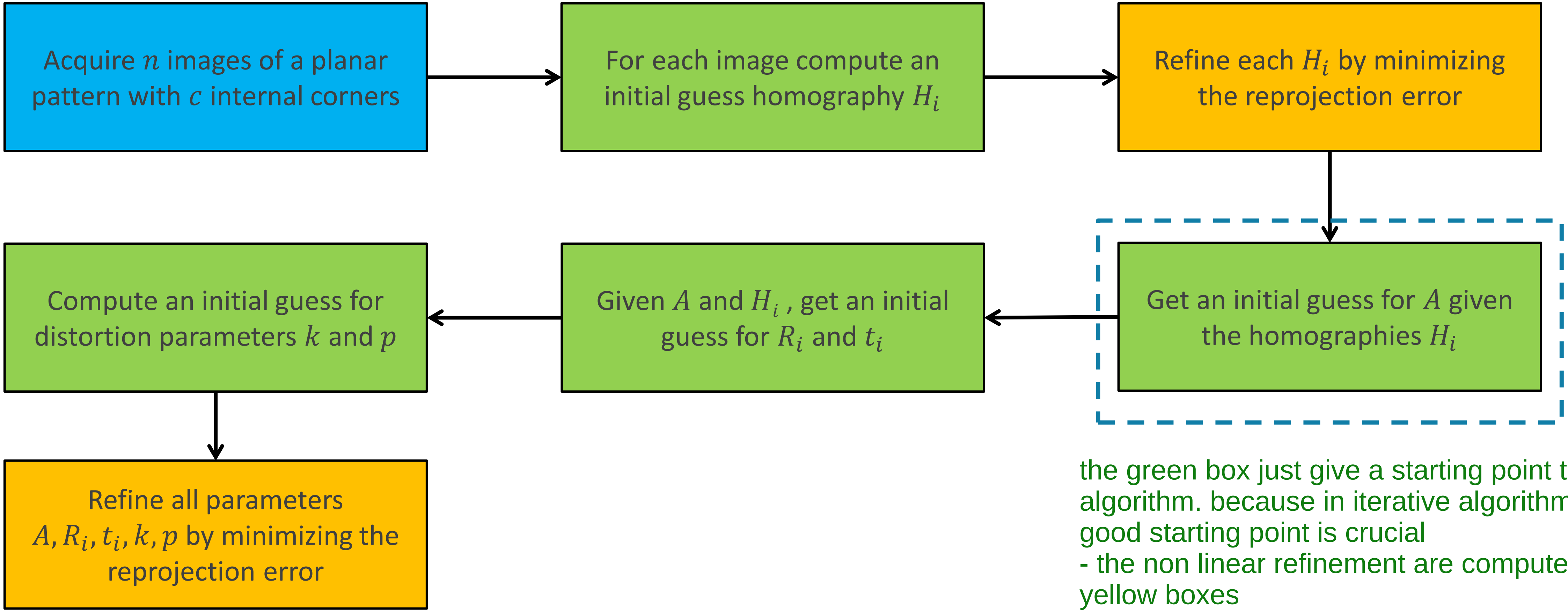
This additional optimization step corresponds to the minimization of the reprojection error (typically referred to as geometric error) measured for each of the 3D corners of the pattern by comparing the pixel coordinates predicted by the estimated homography to the pixel coordinates of the corresponding corner extracted in the image.

The error minimized to estimate the initial guess when solving the linear system is instead referred to as algebraic error or distance. Solutions based on minimization of the algebraic error may not be aligned with our intuition, yet there exist a unique solution, which is cheap to compute. Hence, they are a good starting point for a geometric, non-linear minimization, which effectively minimize the distance we care about.

Zhang’s Method

until now we have estimated 3 of the 4 columns of the PPM

in order to estimate the last one, we have to estimate the intrinsic and extrinsic parameters independently



the green box just give a starting point to the algorithm. because in iterative algorithms finding a good starting point is crucial
- the non linear refinement are computed into the yellow boxes

Zhengyou Zhang. A flexible new technique for camera calibration. IEEE Trans. on PAMI, 2000.

camera calibration wants to estimate all tha parameters. we already estimated the hmography H, then let's estimates the last column of P, that contains the intrinsic parameters of the lens, which we have neglected before.

Estimation of the intrinsic parameters

All the images acquired for calibration share the same intrinsic parameters parameters of the lenses

We can establish the following relations between them (the images) and the extrinsic and intrinsic parameters

because we said that H is P when we have planar objects

planar objects

the PPM of the i-th calibration frame $\boldsymbol{P_i} \equiv \boldsymbol{A}[\boldsymbol{R_i}|\boldsymbol{t_i}] = \boldsymbol{A}[\boldsymbol{r_{i1}} \quad \boldsymbol{r_{i2}} \quad \cancel{\boldsymbol{r_{i3}}} \quad \boldsymbol{t_i}] \Rightarrow \boldsymbol{H_i} = [\boldsymbol{h_{i1}} \quad \boldsymbol{h_{i2}} \quad \boldsymbol{h_{i3}}] = [k\boldsymbol{A}\boldsymbol{r_{i1}} \quad k\boldsymbol{A}\boldsymbol{r_{i2}} \quad k\boldsymbol{A}\boldsymbol{t_i}]$

remember: we have only one set of intrinsic params (A) and different extrinsic parameters for each frame

$$\Rightarrow k\,\boldsymbol{r_{i1}} = \boldsymbol{A}^{-1}\boxed{\boldsymbol{h_{i1}}}, \quad k\,\boldsymbol{r_{i2}} = \boldsymbol{A}^{-1}\boldsymbol{h_{i2}}$$

why do I need to put a scale factor k? because the equivalence between projective spaces holds by a scale factor k

k ri1 and k ri2 has to be orthogonal to each other becuase they are columns of an orthogonal rotation matrix Ri

at this step we dont know these

we have only estimated H up to now

Since the column vectors of each $\boldsymbol{R_i}$ are orthonormal, we get the following constraints

so if they are orthogonal also the RHS must be orthogonal

i'll have hi for each image

1) $\langle \boldsymbol{r_{i1}}, \boldsymbol{r_{i2}} \rangle = \boldsymbol{0} \quad \Rightarrow \quad \langle \boldsymbol{A}^{-1}\boldsymbol{h_{i1}}, \boldsymbol{A}^{-1}\boldsymbol{h_{i2}} \rangle = \boldsymbol{0} \quad \Rightarrow \quad \boldsymbol{h_{i1}^T}\boldsymbol{A}^{-T}\boldsymbol{A}^{-1}\boldsymbol{h_{i2}} = \boldsymbol{0}$

2) $\langle \boldsymbol{r_{i1}}, \boldsymbol{r_{i1}} \rangle = \langle \boldsymbol{r_{i2}}, \boldsymbol{r_{i2}} \rangle \Rightarrow \langle \boldsymbol{A}^{-1}\boldsymbol{h_{i1}}, \boldsymbol{A}^{-1}\boldsymbol{h_{i1}} \rangle = \langle \boldsymbol{A}^{-1}\boldsymbol{h_{i2}}, \boldsymbol{A}^{-1}\boldsymbol{h_{i2}} \rangle \Rightarrow \boldsymbol{h_{i1}^T}\boldsymbol{A}^{-T}\boldsymbol{A}^{-1}\boldsymbol{h_{i1}} = \boldsymbol{h_{i2}^T}\boldsymbol{A}^{-T}\boldsymbol{A}^{-1}\boldsymbol{h_{i2}}$

they also have unit length

the norm of these two vectors must be the same

therefore in order to calibrate i need at least three images

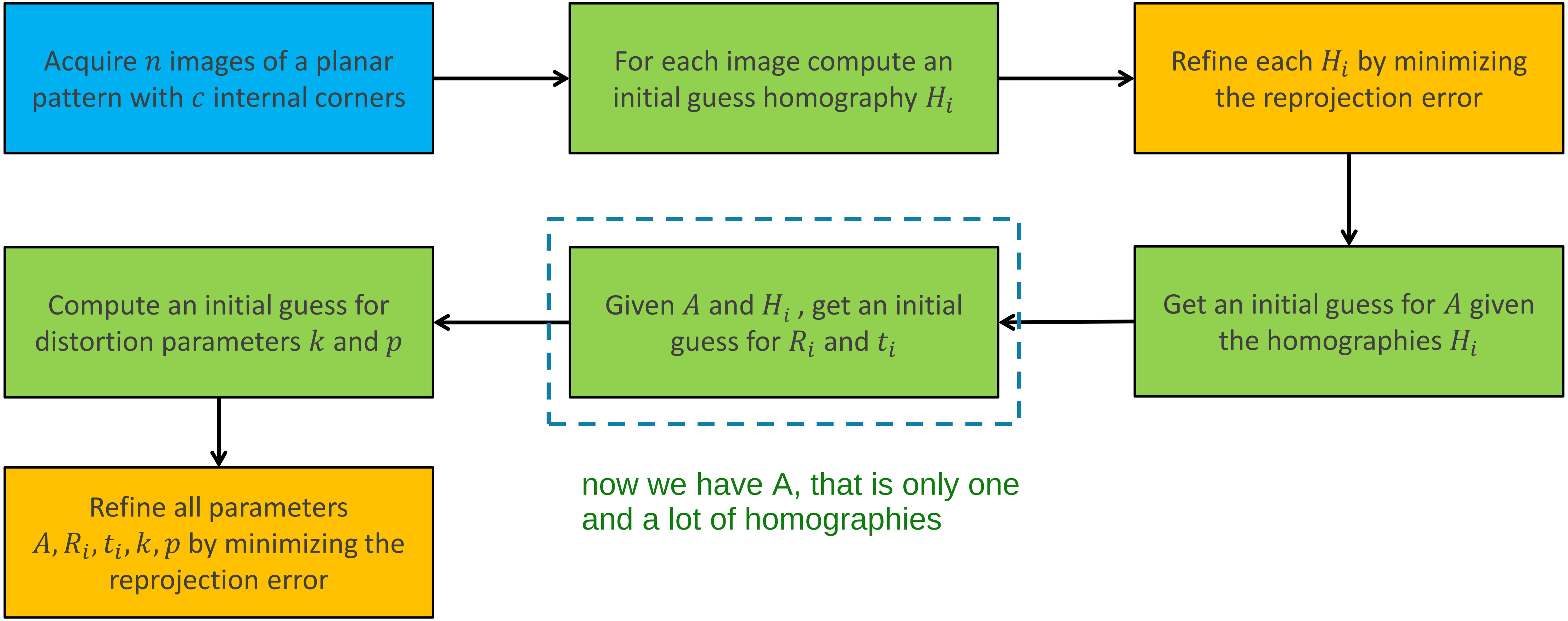
When you have multiple images of the same scene or object, each image provides additional constraints on the parameters to be estimated.

By stacking these two constraints for each image, we get a homogeneous system of equations which can be solved again by SVD if $n \geq 3$ images are collected (6 unknowns since $\boldsymbol{A}^{-T}\boldsymbol{A}^{-1}$ is symmetric).

I cannot say that the norm of A^-1 hi is 1 because i would neglecting k -> but i can say that the norms of k ri1 and k ri2 must be the same (obv not 1) because k is the same

I need atleast 3 images because the system of equations formed by stacking the constraints may not be overdetermined, meaning there may not be enough constraints to uniquely determine the unknown parameters. Adding more images allows for redundancy in the equations, making the system overdetermined and improving the stability and accuracy of the solution.

Zhang's Method



Zhengyou Zhang. A flexible new technique for camera calibration. IEEE Trans. on PAMI, 2000.

Estimation of the extrinsic parameters

let's retrieve an initial guess for rotation and translation parameters

Once $\boldsymbol{A}$ has been estimated, it is possible to compute $\boldsymbol{R_i}$ and $\boldsymbol{t_i}$ (for each image) given $\boldsymbol{A}$ and the previously computed homography $\boldsymbol{H_i}$:

$$\boldsymbol{H_i} = [\boldsymbol{h_{i1}} \quad \boldsymbol{h_{i2}} \quad \boldsymbol{h_{i3}}] = [k\boldsymbol{A}\boldsymbol{r_{i1}} \quad k\boldsymbol{A}\boldsymbol{r_{i2}} \quad k\boldsymbol{A}\boldsymbol{t_i}] \Rightarrow \boldsymbol{r_{i1}} = \frac{1}{k}\boldsymbol{A}^{-1}\boldsymbol{h_{i1}}$$

the only thing i miss to compute R is k

As $\boldsymbol{r_{i1}}$ is a unit vector, the normalization constant can be computed as $k = \|\boldsymbol{A}^{-1}\boldsymbol{h_{i1}}\|$

Then, the same constant can be used to compute

not perfectly unit vector because of noise etc.

$$\boldsymbol{r_{i2}} = \frac{1}{k}\boldsymbol{A}^{-1}\boldsymbol{h_{i2}} \text{ and } \boldsymbol{t_i} = \frac{1}{k}\boldsymbol{A}^{-1}\boldsymbol{h_{i3}}$$

Finally, enforcing again orthonormality of $\boldsymbol{R_i}$, $\boldsymbol{r_{i3}} = \boldsymbol{r_{i1}} \times \boldsymbol{r_{i2}}$

the cross product results in a vector orthogonal to the plane where the multiplying vectors lie

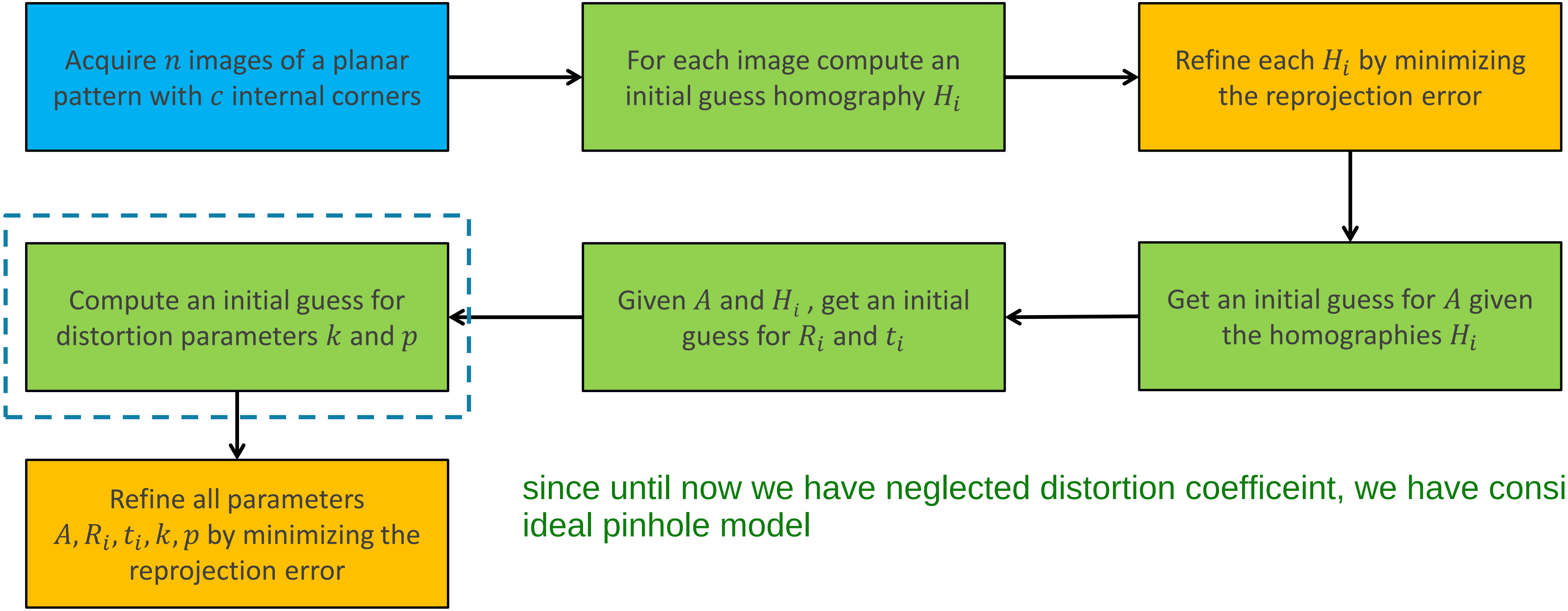
Yet, the resulting matrix $\boldsymbol{R_i}$ will not be exactly orthonormal since $\boldsymbol{r_{i1}}$ and $\boldsymbol{r_{i2}}$ are not necessarily orthogonal and $\boldsymbol{r_{i2}}$ does not necessarily have unit length since k was computed for $\boldsymbol{r_{i1}}$.

sigma

However, SVD of $\boldsymbol{R_i}$ allows to find the closest orthonormal matrix to it by substituting $\boldsymbol{D}$ with $\boldsymbol{I}$.

since we computed the norm of ri1 and applied to ri2, we are not sure it is exatcly a unit vector
therefore we computed an ALMOST orthonormal matrix -> we can however compute an orthonormal matrix through SVD, throwing away the sigma and multiplying U and V, namely substutuing it with the identiy I

Zhang's Method



since until now we have neglected distortion coefficeint, we have considered the ideal pinhole model

Zhengyou Zhang. A flexible new technique for camera calibration. IEEE Trans. on PAMI, 2000.

Lens distortion coefficients

strong assumption

So far, we have neglected lens distortion and calibrated a pure pinhole model. The coordinates predicted by the homographies starting from points in the WRFs correspond to the ideal (undistorted) pixel coordinates of the chessboard corners m_{undist}. The measured coordinates of the corners in the images are the real (distorted) coordinates m.

Original Zhang's method deploys such information to estimate coefficients k_1, k_2 of the radial distortion function:

the real coord we observe, computed with harris, which are "warped"

$$\begin{bmatrix} x \\ y \end{bmatrix} = \underbrace{L(r)}_{\text{radial function}} \begin{bmatrix} x_{undist} \\ y_{undist} \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4) \begin{bmatrix} x_{undist} \\ y_{undist} \end{bmatrix}$$

we take this from the homographies computed

LINEAR SYSTEM

OpenCV uses a different method for estimating the distortion parameters:

- 3 coefficients for radial distortion (k_1, k_2, k_3)
- 2 coefficients for tangential distortion (p_1, p_2)

we have to approximate x_undist, y_undist, doing a strong assumption: the homography H we have computed is a valid transformation, a good model

Metric image coordinates

we have to move into pixel space

Recall: lens distortion takes place before we change metric image coordinates to pixel coordinates. But we measure and predict pixel coordinates.

we want metric image coordinates, not the pixel coordinates u,v

We can transform back pixel coordinates $\begin{bmatrix} u \\ v \end{bmatrix}$ to metric image coordinates $\begin{bmatrix} x \\ y \end{bmatrix}$ thanks to the estimated intrinsic matrix $\boldsymbol{A}$

$$\underbrace{\begin{bmatrix} ku \\ kv \\ k \end{bmatrix}}_{\text{pixels}} \equiv \boldsymbol{A} \underbrace{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}_{\text{metric img coords}} \Rightarrow \begin{bmatrix} ku \\ kv \\ k \end{bmatrix} \equiv \begin{bmatrix} f_u x + u_0 \\ f_v y + v_0 \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{u-u_0}{f_u} \\ \frac{v-v_0}{f_v} \end{bmatrix}$$

The same transformation holds between u_{undist}, v_{undist} and x_{undist}, y_{undist}

Then, the distortion equation in pixel coordinates become

$$\begin{bmatrix} \frac{u-u_0}{f_u} \\ \frac{v-v_0}{f_v} \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4) \begin{bmatrix} \frac{u_{undist}-u_0}{f_u} \\ \frac{v_{undist}-v_0}{f_v} \end{bmatrix} \Rightarrow \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4) \begin{bmatrix} u_{undist} - u_0 \\ v_{undist} - v_0 \end{bmatrix}$$

u,v are real coords of our pixels warped by lens distortion, computed by harris

u/v undist are the coords computed by projecting the 3d points according to our homographies

Lens distortion coefficients

we know everything but k1 and k2,
the distortion coefficients

$$\begin{bmatrix} u-u_0 \\ v-v_0 \end{bmatrix} = (\boxed{1} + k_1 r^2 + k_2 r^4) \begin{bmatrix} u_{undist}-u_0 \\ v_{undist}-v_0 \end{bmatrix} \Rightarrow \begin{bmatrix} u-u_0 \\ v-v_0 \end{bmatrix} - \begin{bmatrix} u_{undist}-u_0 \\ v_{undist}-v_0 \end{bmatrix} = (\underline{k_1} r^2 + \underline{k_2} r^4) \begin{bmatrix} u_{undist}-u_0 \\ v_{undist}-v_0 \end{bmatrix}$$

all of this is known, except for k1,k2

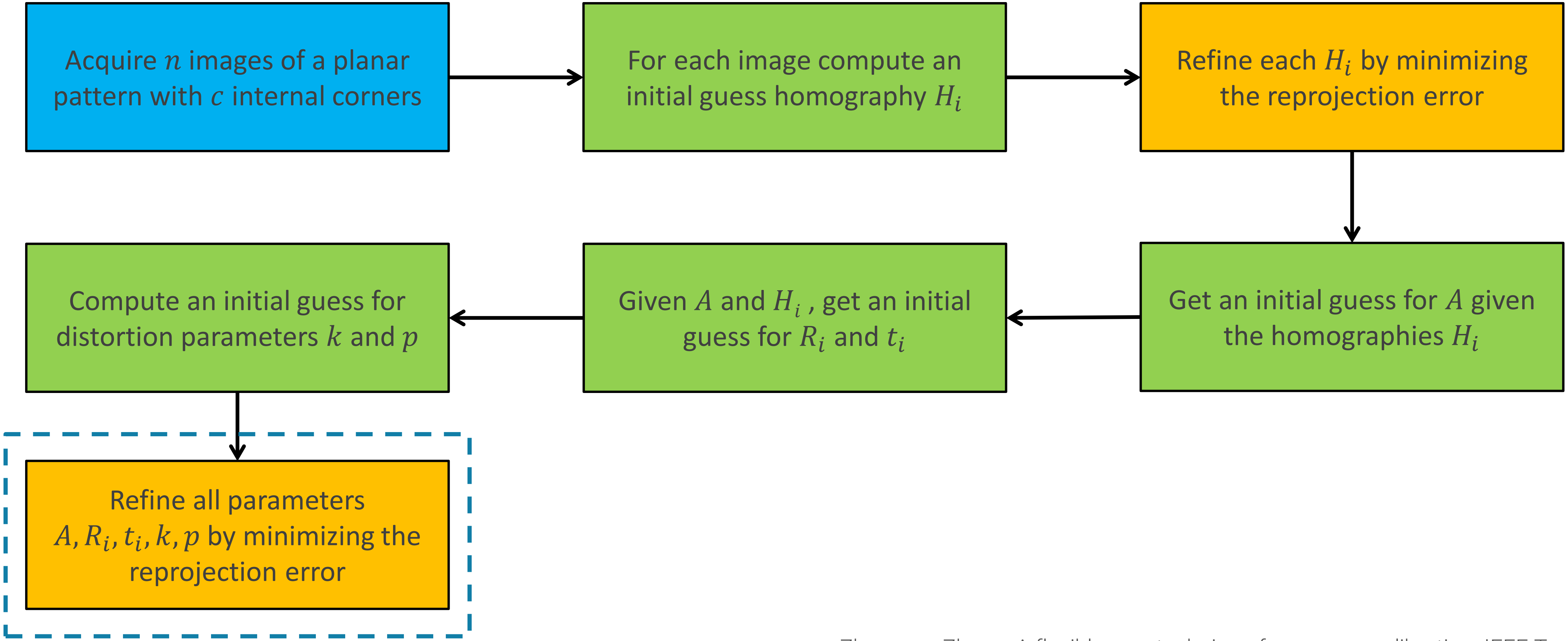
$$\begin{bmatrix} u-u_{undist} \\ v-v_{undist} \end{bmatrix} = \begin{bmatrix} (u_{undist}-u_0)r^2 & (u_{undist}-u_0)r^4 \\ (v_{undist}-v_0)r^2 & (v_{undist}-v_0)r^4 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix}$$

We get a linear, non-homogeneous system of linear equations $\boldsymbol{Dk} = \boldsymbol{d}$ in the unknowns $\boldsymbol{k} = [k_1 \quad k_2]^T$. With c corners in n images we get $2nc$ equations in 2 unknowns, which can be solved in a least square sense, i.e., minimizing $||\boldsymbol{Dk} - \boldsymbol{d}||_2$, by computing the pseudo-inverse matrix $\boldsymbol{D}^\dagger$ as

"D dagger"

$$\boldsymbol{k}^* = \min_{\boldsymbol{k}} ||\boldsymbol{Dk} - \boldsymbol{d}||_2 = \boldsymbol{D}^\dagger \boldsymbol{d} = \left(\boldsymbol{D}^T \boldsymbol{D}\right)^{-1} \boldsymbol{D}^T \boldsymbol{d}$$

Zhang's Method



Zhengyou Zhang. A flexible new technique for camera calibration. IEEE Trans. on PAMI, 2000.

Refinement by non-linear optimization

Final non-linear refinement of the estimated parameters. As for homographies, the procedure highlighted so far seeks to minimize an algebraic error, without any real physical meaning.

A more accurate solution can instead be found by a so called Maximum Likelihood Estimate (MLE) aimed at minimization of the geometric (i.e. reprojection) error.

We use all the values estimated so far as initial guesses.

Under the hypothesis of i.i.d. (independent identically distributed) noise, the MLE for our models is obtained by minimization of the error

real ones
computed
by Harris

$$\boldsymbol{A}^*, \boldsymbol{k}^*, \boldsymbol{R}_{\boldsymbol{i}}^*, \boldsymbol{t}_{\boldsymbol{i}}^* = \underset{\boldsymbol{A},\boldsymbol{k},\boldsymbol{R}_{\boldsymbol{i}},\boldsymbol{t}_{\boldsymbol{i}}}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{j=1}^{c} \left\| \tilde{\boldsymbol{m}}_{ij} - \hat{\mathbf{m}}\left(\mathbf{A}, \mathbf{k}, \mathbf{R}_i, \boldsymbol{t}_i, \tilde{\mathbf{w}}_j\right) \right\|^2$$

this time we weill optimize not for H, but for those paramters

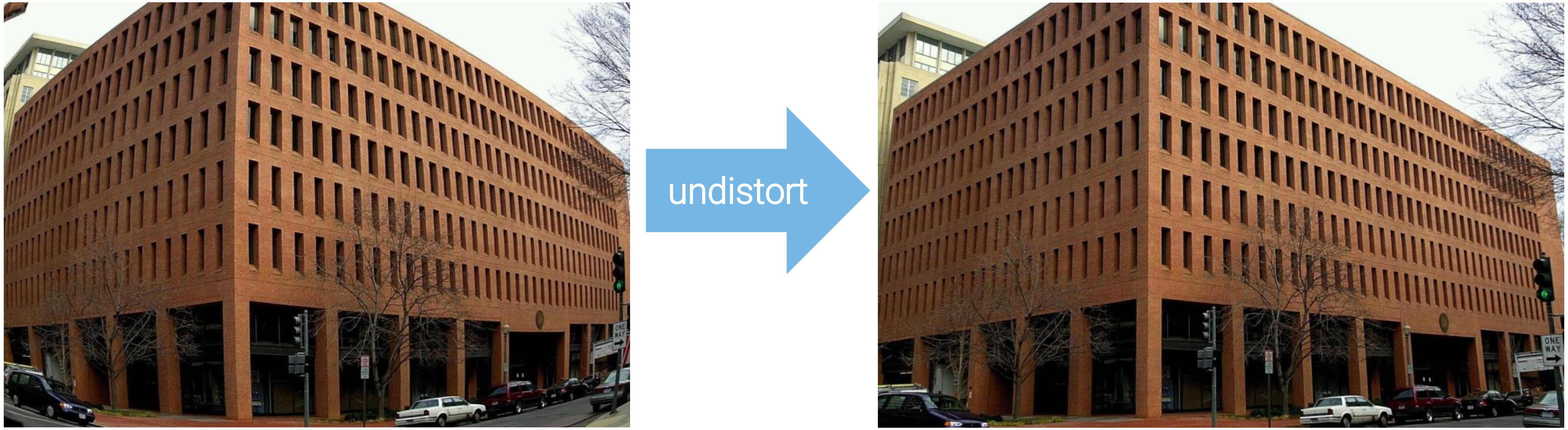
with respect to all the unknown camera parameters, which can be solved again by using an iterative algorithm, like the Levenberg-Marquardt algorithm.

get rid of

Compensate lens distortion

After calibration, we have a precise mathematical model that maps points in the 3D world to points in the image plane. Yet, lens distortion makes the system non-linear and therefore cumbersome to use.

Hence, it is common, once a camera has been calibrated, to warp the images it takes so to simulate a camera without lens distortion, whose camera formation model is linear, i.e. it is the estimated PPM.



https://docs.nvidia.com/isaac/archive/2020.2/packages/perception/doc/warp.html

Image Warping ≠ Image filtering

Warping refers to transformations of the spatial domain of images, while filtering refers to changes in the RGB values of images

more related to intensity, the content

Filtered (Gaussian blur)

Convolution.
Edge detection,
…

Original

?

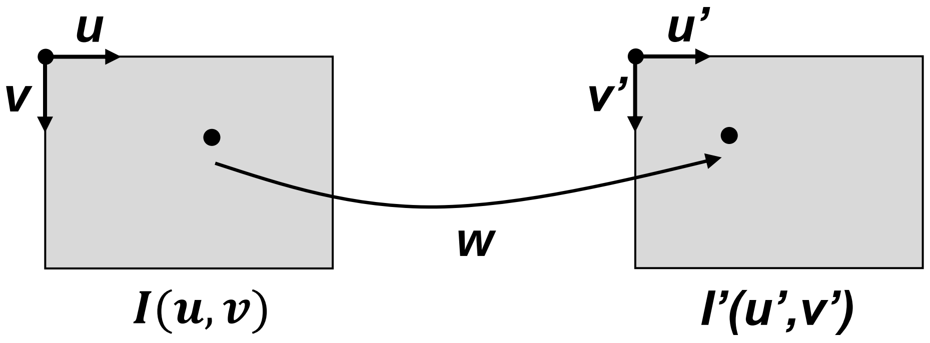
Warped (Rotation & scale)

Generated by Stable Diffusion (Prompt: a photograph of an astronaut riding a horse) by Asanagi - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=122422630

Image Warping

it is a relationship, a mapping, between coords
create a new image I'

If we have a function that computes point in image *I'* starting from point in image *I, we can copy the value*



warping function

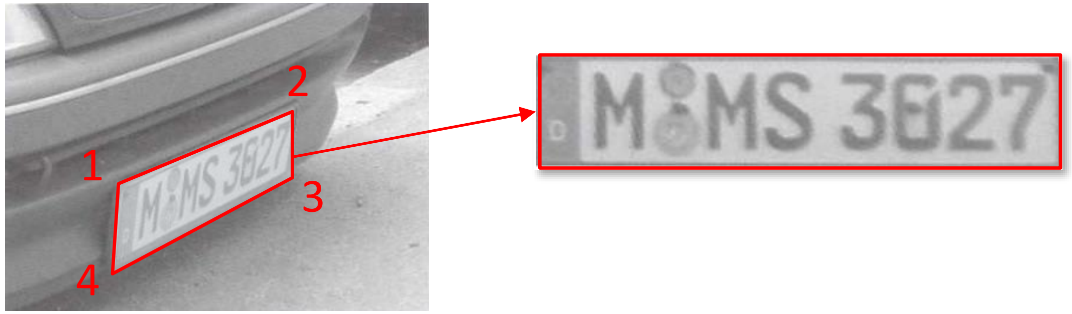
$$\begin{cases} u' = w_u(u,v) \\ v' = w_v(u,v) \end{cases}$$

$$I'\big(w_u(u,v), w_v(u,v)\big) = I(u,v)$$

Can be just a rotation $\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$

transf. coord

Or it can be a full homography $k\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{32} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$



Forward mapping

going from I to I' is called Forward mapping

If we start from the input image coordinates, after applying the warping function in general we get continuous coordinates in the output image, not discrete ones. This is called a forward mapping. we want discrete, not real continuous numbers -> therefore we do roundings

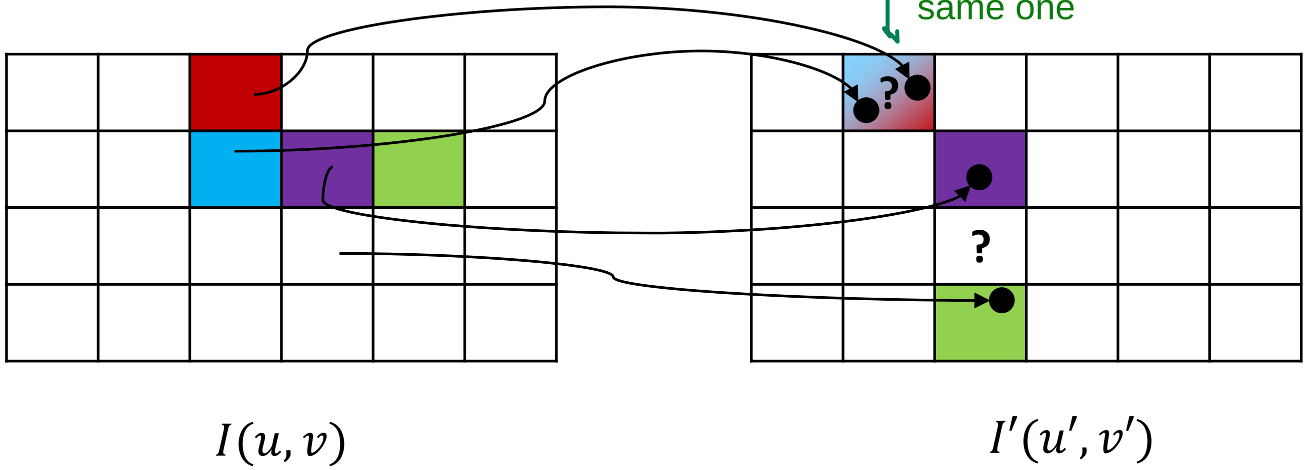
Possible choices to make coordinates integer numbers: truncate, neareast neighbor (i.e. rounding), … Regardless of the discretization function, due to rounding

- more than one pixel can go to one position (folds)
- some pixels of the destination image may not be hit (holes)

two main problems
of forw. mapping

but since we approximate we may end up in a situation like this one, where two pixels are mapped onto the same one

$$\begin{cases} u' = w_u(u,v) \\ v' = w_v(u,v) \end{cases}$$



we want an injective warping function

Backward mapping

we start from I' the target image using inverse mapping w^-1

doing this, we start for sure form a integer number, cause we start from a certain pixel and go back to the source image I, and then we'll have there a non-integer value, but it is not a problem because even if we discretize it with different techniques wel'll always end up with a value for the pixel in the target image

We can avoid these problems if we compute input coordinates corresponding to **each pair of integer coordinates in the output image,** by using the **inverse mapping** w^{-1}. This strategy is referred to as **backward mapping.**

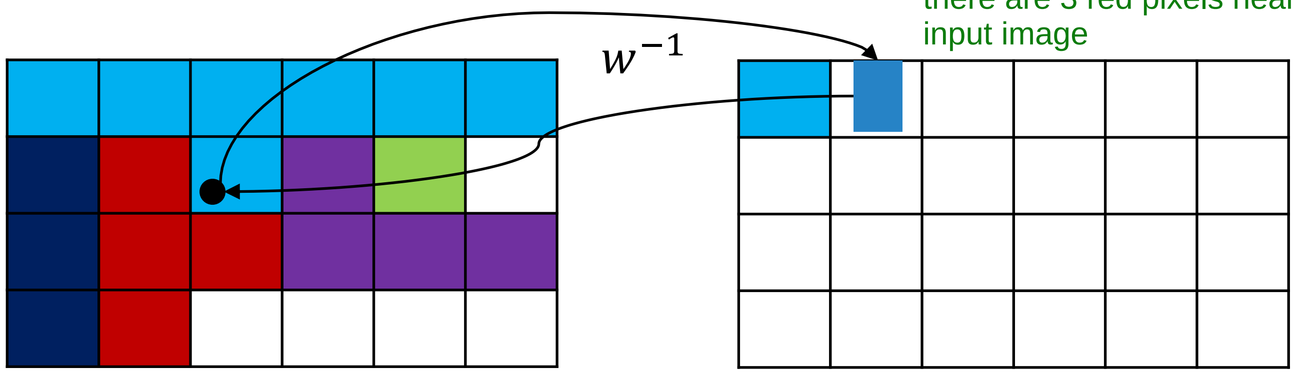
Yet, the input coordinates are still continuous values. Which discretization strategy are used?

- Truncate
- Nearest Neighbour   just choose the closest one
- **Interpolate between the 4 closest point** (**bilinear**, bicubic, etc…)

interpolation

it is reasonable that it will have some shade of red, since there are 3 red pixels near the correpsoinfing point in the input image

$$\begin{cases} u = w_u^{-1}(u', v') \\ v = w_v^{-1}(u', v') \end{cases}$$



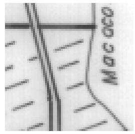
$I(u, v)$ input image, where I have RGB values

$I'(u', v')$

we compute a linear interpolation of intensities Ia and Ib, but I have to compute them doing another linear interpolation between I1 and I2

in order to get lin. inter. of Ia and Ib we compute first the horizontal interp. between I1 and I2 and I3 and I4

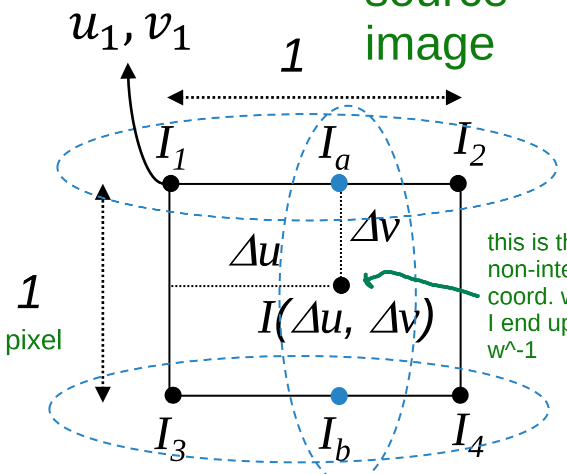
Bilinear Interpolation



Input

top-left

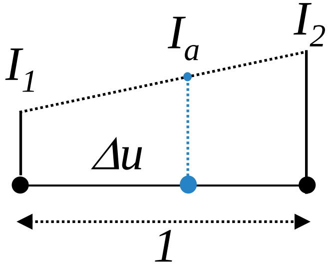
source image

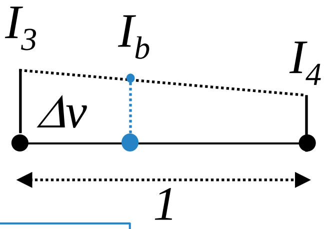


this is the non-integer coord. where I end up after w^-1

these 4 Ii are the nearest neighbors wrt the pixel of the landing position

the height is the intensity value: in this case I2 is brighter than I1





$$\Delta u = u - u_1$$
$$\Delta v = v - v_1$$

$$\frac{I_a - I_1}{\Delta u} = I_2 - I_1$$

$$I_a = (I_2 - I_1)\Delta u + I_1$$

$$I_b = (I_4 - I_3)\Delta u + I_3$$

The closer a point to a a pixel the smaller the other weights become

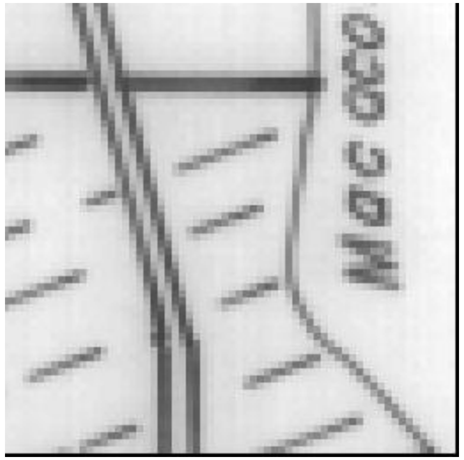
$$I(\Delta u, \Delta v) = (I_b - I_a)\Delta v + I_a$$

$$I(\Delta u, \Delta v) = ((I_4 - I_3)\Delta u + I_3 - ((I_2 - I_1)\Delta u + I_1))\Delta v + (I_2 - I_1)\Delta u + I_1$$

$$I'(u', v') = (1 - \Delta u)(1 - \Delta v)I_1 + \Delta u(1 - \Delta v)I_2 + (1 - \Delta u)\Delta v I_3 + \Delta u \Delta v I_4$$

it is a linear combination of the "corner" points: depending on the position I, e.g. if the point is close to I4, the resulting point will be really similar to I4 and the other points I1,I2, I3, will be negligible

Warp (Zoom) by nearest neighbor



sharper edges

use this in order to preserve e.g. binary masks/classes/categorical variables, etc.

Warp (Zoom) by Bilinear Interpolation

smoother image

use e.g. for natural images

«Undistort» warping

u,v is the point where the point should be landed without the distortion caused by lens, which bend the image

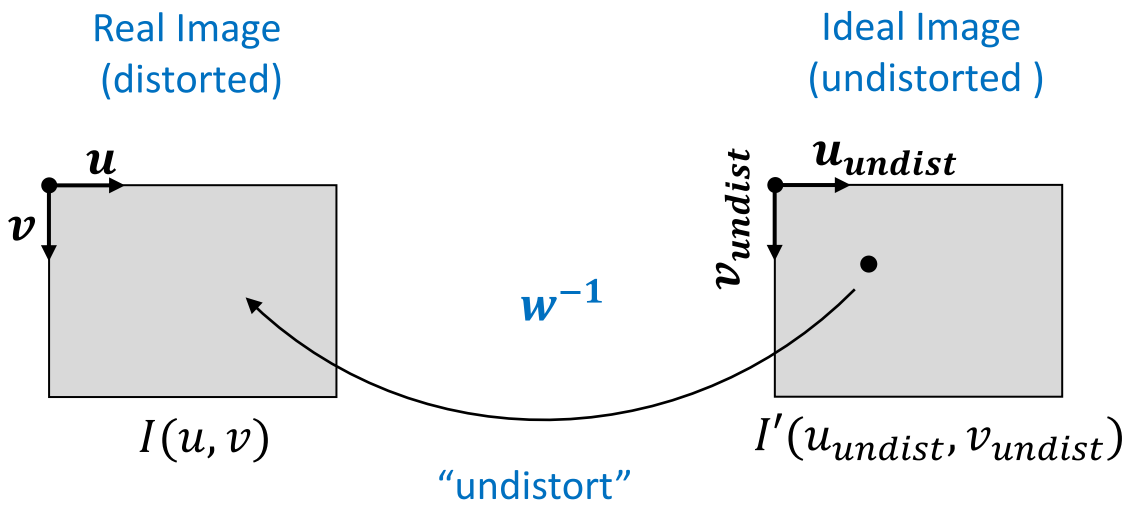
Once the lens distortion parameters have been computed by camera calibration, the image can be corrected by a backward warp from the undistorted to the distorted image based on the adopted lens distortion model. For this images, the image formation model is linear, i.e. the PPM.

$$\forall (u_{undist}, v_{undist}):\ \ I'(u_{undist}, v_{undist}) = I\left(w_u^{-1}(u_{undist}, v_{undist}), w_v^{-1}(u_{undist}, v_{undist})\right)$$

w^-1 is given by the zhang's radial distortion function that we use in camera calibration to find the real coordinates of pixels exposed to distortion effects

$$w = \begin{cases} u = u_{undist} + (k_1 r^2 + k_2 r^4)(u_{undist} - u_0) \\ v = v_{undist} + (k_1 r^2 + k_2 r^4)(v_{undist} - v_0) \end{cases}$$

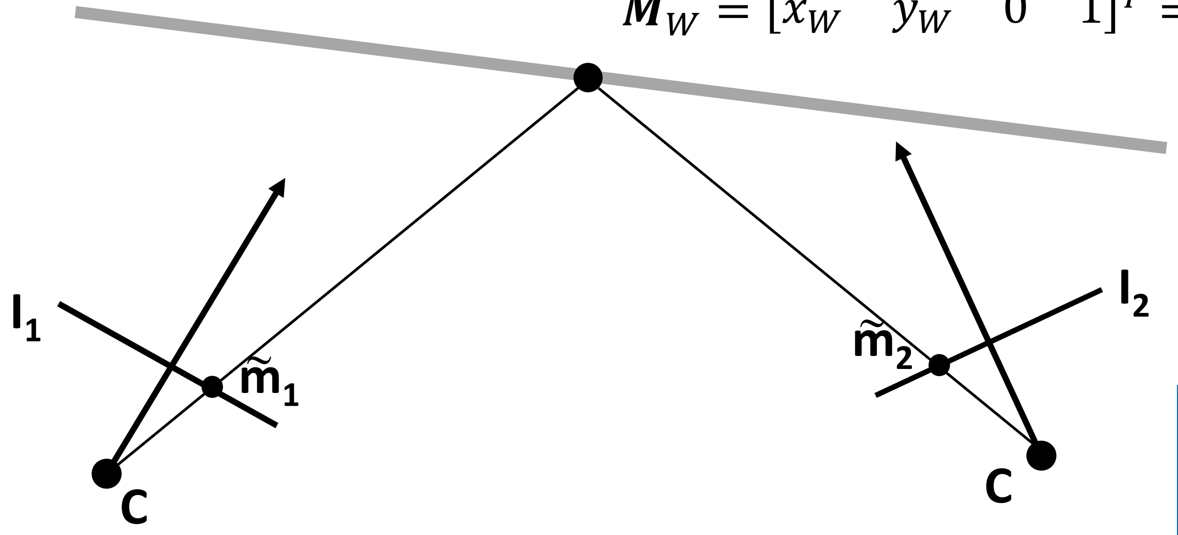
Zhang's
Radial distorsion



Warping with homographies

when we project M in WRF to camra 1 we obtain m1

$$\tilde{\boldsymbol{M}}_W = [x_W \quad y_W \quad 0 \quad 1]^T = [x_W \quad y_W \quad 1]^T$$



Any two images of a planar scene (without lens distortion) are related by a homography

the homography H maps Mw to m

$$\tilde{\boldsymbol{m}}_1 = \boldsymbol{H}_1 \tilde{\boldsymbol{M}}_W \qquad \tilde{\boldsymbol{m}}_2 = \boldsymbol{H}_2 \tilde{\boldsymbol{M}}_W$$

we are talking about linear camera model now, without distortion

which are the m1 coord in the other pov of m2?

$$\tilde{\boldsymbol{m}}_1 = \boldsymbol{H}_1 \boldsymbol{H}_2^{-1} \tilde{\boldsymbol{m}}_2 \qquad \tilde{\boldsymbol{m}}_2 = \boldsymbol{H}_2 \boldsymbol{H}_1^{-1} \tilde{\boldsymbol{m}}_1$$

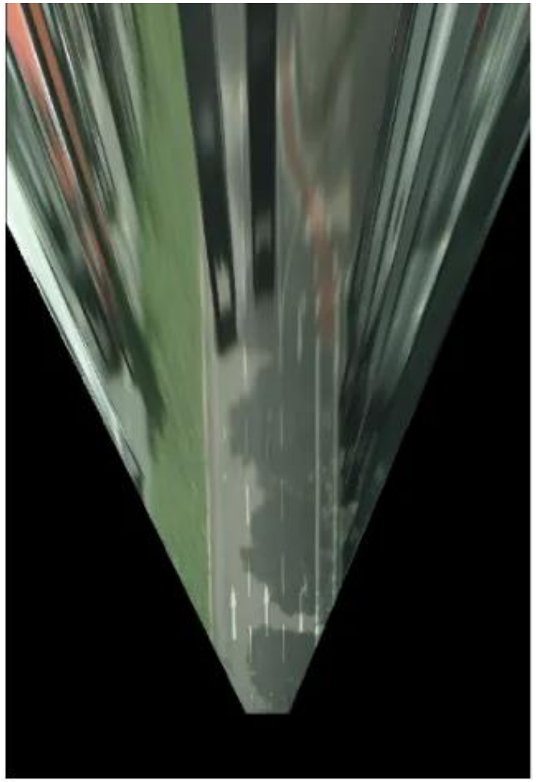
i can get a relationship between pixel coords, without considering WRF

$$\boldsymbol{H}_{21} \quad \text{Homography} \quad \boldsymbol{H}_{12} = \boldsymbol{H}_{21}^{-1}$$

this is the homopgraphy whihc goes from image plane 1 to image plane 2

Example: Inverse Perspective Mapping

in this case the secon image plane is avirtual sensor, it's like we simulate to havae another image e.g. taking interesting points on the street



https://towardsdatascience.com/a-hands-on-application-of-homography-ipm-18d9e47c152f another useful usa case: rotate camera around its optical center

I can compute images as we have one in a different orietnation -> really powerful operation

Warping with homographies

in this case the relationship between images are homographies and these homgraphies are valid for each Mw, for each 3D point in the WRF

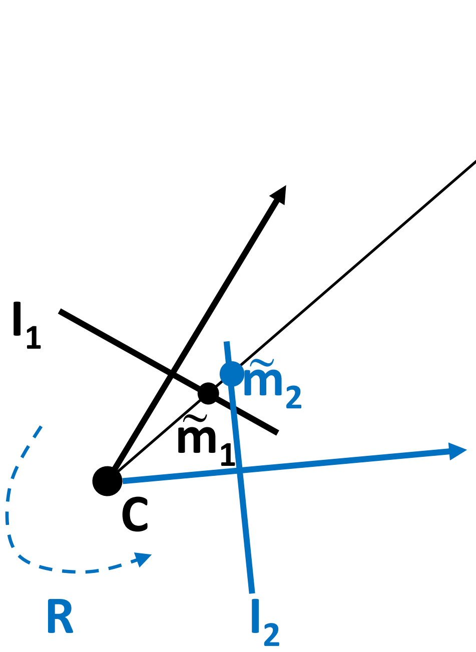
I can compute images as if I had a camera mounted in different orientation

Any two images taken by a camera rotating about its optical center are related by a homography (if lens distorsion has been removed)

same camera, just rotated around the center

generic world point

$$\tilde{\mathbf{M}}_W = \begin{bmatrix} x_W \\ y_W \\ z_W \\ 1 \end{bmatrix}$$



we get rid of the homog. part [I | 0]

what if I want to sim. a virtual sensor and project Mw onto I2? I have to apply a rotation to match the CRF from WRF

$$\tilde{\boldsymbol{m}}_1 = \underline{\boldsymbol{A}\,[\,\boldsymbol{I}\,|\boldsymbol{0}\,]\,\tilde{\boldsymbol{M}}_W} = \boldsymbol{A}\boldsymbol{M}_W \qquad \tilde{\boldsymbol{m}}_2 = \boldsymbol{A}\,[\,\boldsymbol{R}\,|\boldsymbol{0}\,]\,\tilde{\boldsymbol{M}}_W = \boldsymbol{A}\boldsymbol{R}\boldsymbol{M}_W$$

in this case there is no rotation or transl, no extrnsic params, the WRF and CRF coincides

$$\tilde{\boldsymbol{m}}_1 = \boldsymbol{A}\boldsymbol{R}^{-1}\boldsymbol{A}^{-1}\tilde{\boldsymbol{m}}_2 \qquad \tilde{\boldsymbol{m}}_2 = \boldsymbol{A}\boldsymbol{R}\boldsymbol{A}^{-1}\tilde{\boldsymbol{m}}_1$$

i can express pixel points as other pixel points, as before

$$\boldsymbol{H}_{21} \leftarrow \textbf{Homography} \rightarrow \boldsymbol{H}_{12} = \boldsymbol{H}_{21}^{-1}$$

the relation between m1 and m2 is a rotation

I can pass from image plane 1 to image plane 2 computing a rotation matrix pitch is the rotation around the horizontal x axis

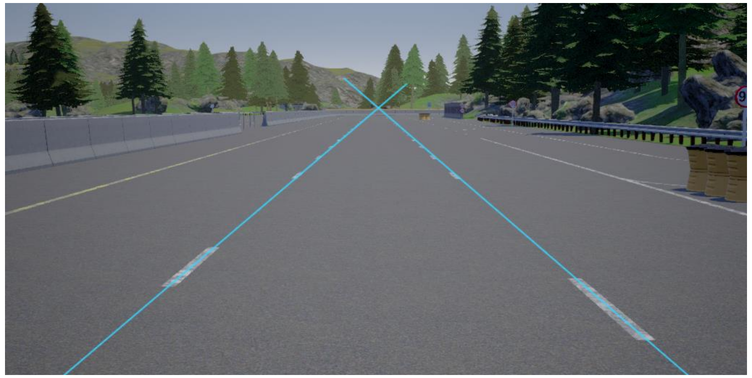
Example: compensate pitch or yaw

Autonomous driving cameras should be ideally mounted with the optical axis parallel to the road plane and aligned with the direction of motion. It is however difficult to obtain perfect alignment by mechanical mounting only.

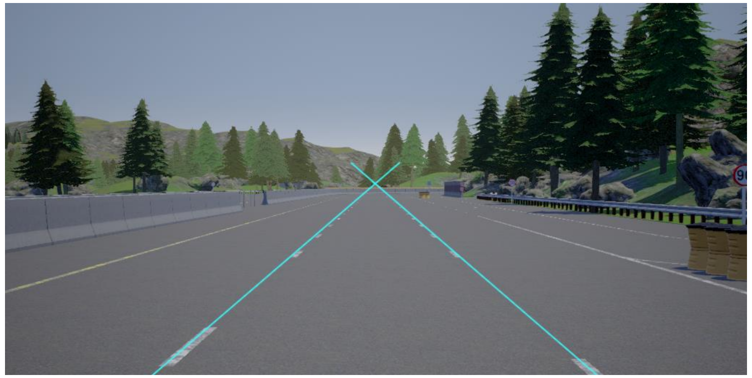
With a calibrated camera, however, it is possible to compensate pitch (rotation around the x axis) and yaw (rotation around the y axis) by estimating the vanishing point of the lane lines when the vehicle is travelling straight in a lane.

we can compute the straight lines with the Hough Transform and their intersection, namely the vanishing point, obtaining m_inf and its coordinates u,v

$Pitch = -5°$



$Pitch = 0°$



the vanishing point is perfectly at the center of the image

$Pitch = +5°$

https://thomasfermi.github.io/Algorithms-for-Automated-Driving/CameraCalibration/VanishingPointCameraCalibration.html

Example: compensate pitch or yaw

When the vehicle is driving straight with respect to the lines, their orientation in a world reference frame attached to the vehicle is parallel to the z axis, i.e. it is $[0 \quad 0 \quad 1]^T$ and their point at infinity in $\mathbb{P}^3$ is $[0 \quad 0 \quad 1 \quad 0]^T$.

The vanishing point is then $m_\infty \equiv A[R|0]\begin{bmatrix}0\\0\\1\\0\end{bmatrix} \equiv Ar_3 \equiv A\begin{bmatrix}0\\\sin\beta\\\cos\beta\end{bmatrix}$ since a rotation around the x axis of an

remember: is the projection of the point at infinity

angle β has expression $R_{pitch} = \begin{bmatrix}1 & 0 & 0\\0 & \cos\beta & \sin\beta\\0 & -\sin\beta & \cos\beta\end{bmatrix}$

our unknown

we want to estimate the pitch wrt the ideal image with 0 pitch

normalization

Then , if we estimate the coordinates of the vanishing point in the image , we can compute $r_3 = \frac{A^{-1}m_\infty}{\|A^{-1}m_\infty\|_2}$ and from it R_{pitch} and finally the warping homography from 0 pitch to input image as $AR_{pitch}A^{-1}$

With the same procedure it is possible to correct simultaneously yaw and pitch.

https://thomasfermi.github.io/Algorithms-for-Automated-Driving/CameraCalibration/VanishingPointCameraCalibration.html