G3DCV

RICCARDO ZULIANI

August 2023

1 Background Foreground Segmentation Part

- **Gain and Bias:** gain control contrast and bias control brightness
- **Log Transformation:** useful to compress the dynamic range for images with large variation in pixel values
- **Gamma Transformation:** same as lag but more flexible due to its parameter
- **Contrast Enchancement:** simpliest picewise-linear transformation (sigmoid shape)
- **Thresholding:** t is a constant defined for the whole image, if it depends on spartial coordinates, the process is refered as adaptive thresholding
- **Image Histogram:** allow us to analyse problems in the intensity distribution of an image. Is the empirical distribution of image intensities.
- **Histogram Equalization:** maximize the intensity range of the image to catch both dark and bright details. Mapping function to obtain a uniform distribution. approximation of PDF

WITHIN CLASS VARIANCE

$$\sigma_\omega^2(T) = P_1(T)\cdot\sigma_{C_1}^2(T) + P_1(T)\cdot\sigma_{C_2}^2(T)$$

BETWEEN CLASS VARIANCE

$$\sigma_B^2(T) = P_1(T)(m_1(T) - m_G)^2 + P_2(T)(m_2(T) - m_G)^2$$

VARIANCE OF EACH CLASS

- **Histogram for Thresholding:** a common problem is to automatically find good threshold t that separates well dark from well bright areas. Thresholding is essentially a clustering problem in which two clusters are sought. The idea is to find optimum threshold so that the variance of each class (*within-class-variance*) is minimized. Nobuyuki Otsu demonstrated that the optimal T that minimizes the *within-class-variance* also maximize the *between-class-variance*

- **Linear filter:** consist of a neighbourhood and a predefined operation that is performed on the image pixels. It can be described in terms of correlation and convolution
- **Correlation:** moving a filter mask over a signal and computing the sum of products at each location

- **Convolution**: like correlation but the filter mask is roteated by 180 degrees.
- **Noise Reduction:**
  - Gaussian-filter
  - Median-filter: median intensity value of that pixe for its neighborhood
  - Max-filter: brightest intensity value in the neighbourhood
  - Min-filter: darkest intensity value in the neighbourhood
- **Sharpening filters:** highlight transitions in intensity
  - Unsharp masking
  - First order derivative: thick edges because the derivative is non zero along a ramp
  - Second order derivative (better): double edge one pixel tick, separated by zeros
- **Laplacian Operators:** highlights intensity discontinuities in image and deemphasizes regions with slowly varying intensity levels

---

- **Color characteristics**
  - *Brightness:* how strong is the light regardless the color
  - *Hue:* dominant wavelength in a mixture of light waves
  - *Saturation:* the relative purity of the amount of white light mixed with a hue
- **Additive mixing:** from RGB color added together we obtain secondary color
- **Subtractive synthesis:** secondary color is defined as one that subtracts or absorbs a primary color
- **RGB:** 3D space, additive mixing, can't represent all colors
- **CMY (Cyan, Magenta, Yellow) and CMYK (Cyan, Magenta, Yellow, Black)** 3D space, subtractive mixing, can't represent all colors
- **HSI (Hue, Saturation, Intensity):** hue is an angle
- **YUV:** similar to HSI but the color vector is parametrized by u, v
- **Smoothing and Sharpening** convolutions can also be applied to color images for smoothing and/or sharpening

- **LAB color code**: LAB color space expresses color variations across three channels. One channel for brightness and two channels for color:
  - L-channel: representing lightness in the image
  - a-channel: representing change in color between red and green
  - b-channel: representing change in color between yellow and blue

SEVERAL HISTOGRAMS EACH CORRESPONDING TO A DISTINCT SECTION OF THE IMAGE
↳ USE THESE HISTOGRAMS TO REDISTRIBUTE THE LIGHTNESS OF THE IMAGE
↳ IT IMPROVE LOCAL CONTRAST AND ENHANCING THE DEFINITION OF EDGES IN EACH REGION OF A IMAGE
↳ IT HAS THE DEFECT THAT IT AMPLIFY THE NOISE
↳ IT OVERAMPLIFY THE CONTRAST IN NEAR-CONTRAST REGION

- **Adaptive histogram equalization:** is a computer image processing technique used to improve contrast in images. It differs from ordinary histogram equalization in the respect that the adaptive method computes several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values of the image. It is therefore suitable for improving the local contrast and enhancing the definitions of edges in each region of an image. However, AHE has a tendency to overamplify noise in relatively homogeneous regions of an image. A variant of adaptive histogram equalization called contrast limited adaptive histogram equalization (CLAHE) prevents this by limiting the amplification.

CLAHE → CONTRAST AMPLIFICATION IS LIMITED
↳ CONTRAST AMPLIFICATION IN THE VICINITY OF A GIVEN PIXEL VALUE IS GIVEN BY THE SLOPE OF THE TRANSFORMATION FUNCTION
↳ IS PROPORTIONAL TO THE SLOPE OF THE NEIGHBORHOOD CDF
↳ REDUCE THE AMPLIFICATION BY CLIPPING THE HISTOGRAM AT A PREDEFINED VALUE BEFORE COMPUTING THE CDF

- **CLAHE (Contrast Limited Adaptive Histogram Equalization):** Ordinary AHE tends to overamplify the contrast in near-constant regions of the image, since the histogram in such regions is highly concentrated. As a result, AHE may cause noise to be amplified in near-constant regions. Contrast Limited AHE (CLAHE) is a variant of adaptive histogram equalization in which the contrast amplification is limited, so as to reduce this problem of noise amplification. In CLAHE, the contrast amplification in the vicinity of a given pixel value is given by the slope of the transformation function. This is proportional to the slope of the neighbourhood cumulative distribution function (CDF) and therefore to the value of the histogram at that pixel value. CLAHE limits the amplification by clipping the histogram at a predefined value before computing the CDF. This limits the slope of the CDF and therefore of the transformation function. The value at which the histogram is clipped, the so-called clip limit, depends on the normalization of the histogram and thereby on the size of the neighbourhood region. Common values limit the resulting amplification to between 3 and 4.

IT IS THE BEST CHOICHE REGARDING SEGMENTATION SINCE A VARIATION OF ONE OF THE 3 CHANNEL RESULTS IN A SLIGTHLY DIFFERENT COLOR
↳ USING RGB FORMAT, IF WE CHANGE A CHANNEL OF A BIT WE END UP IN A COMPLETLY DIFFERENT COLOR

- **HSV** (hue, saturation, value) colorspace is a model to represent the colorspace similar to the RGB color model. Since the hue channel models the color type, it is very useful in image processing tasks that need to segment objects based on its color. Variation of the saturation goes from unsaturated to represent shades of gray and fully saturated (no white component). Value channel describes the brightness or the intensity of the color.

- **Preliminaries:** considering thresholded images containing only black and white pixels
- **Set translation:** creates a new set by translating every point of the initial set by the same amount $z(z_1, z_2)$
- **Set reflection** creates a new set whose coordinates are replaced with $(-x, -y)$
- **Structuring elements:** small set of sub-images used to probe an image under study for properties of interest
- **Erosion:** it erodes away the boundaries of foreground object. A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero). All the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.
- **Dilatation:** It is just opposite of erosion. Here, a pixel element is '1' if at least one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

COMBINATIONS OF THE TWO OPERATIONS IN ORDER TO CANCEL OUT NOISE

- **Idea:** by combining operations, somehow the effect cancel out noise in region where I'm not interested to change the shape of the object, out some pixel will change
- **Opening (erosion followed by a dilatation):** smoothens the contour of an object, eliminates thin protrusions
- **Closing (dilatation followed by a erosion):** smoothens sections of contours, fuses narrow breaks and long thin gulfs, eliminates small holes, fills gaps in the contour
- **Grayscale morphology:** erosion, dilatation, opening and closing can also be defined for grayscale images

MINIMUM INTENSITY VALUE IMAGE DARKER BRIGHTNESS IS REDUCED

- **Erosion:** computes the minimum intensity values over every neighborhood of (x,y), the resulting image is darker and the size of bright features is in general reduced

MAXIMUM INTENSITY VALUE IMAGE BRIGHTER, DARKNESS IS REDUCED

- **Dilatation:** gives opposite result wrt erosion, The resulting image is brighter, bright features are thickened and the intensities of the dark features is reduced

COMPUTE THE MAXIMUM INTENSITY VALUE OVER EVERY NEIGHBORHOOD OF (X,Y)

THE MEANING OF OPENING AND CLOSING IS ASSOCIATED TO THE BACKGROUND

REMOVE NOISE

CLOSE HOLES INSIDE THE FOREGROUND IMAGE

- **Opening:** *attenuate bright features* and has negligible effect on the dark features and the background of the image
- **Closing:** *attenuate dark features* and as negligible effect on the bright features and the background of the image
- **Morphological Gradient:** is the difference between dilation and erosion of an image
- **Top-hat:** is the difference between input image and Opening of the image
- **Bottom-hat or Black-hat:** is the difference between the closing of the input image and input image

2 Marker Detector Part

- **Features:** location of sudden change in intensity. Useful since information content is high, invariant to change of viewpoint or illumination, reduces computational burden
- **Good Feature** is invariant to:
  - Viewpoint
  - Lighting conditions
  - Object deformations
  - Partial occlusions

  And should be unique and easy to be found and extracted
- **Edges:** edges are pixels at which the intensity of an image function changes abruptly
- **Edge detection:** find image pixels that present abrupt changes in intensity
- **Edge models:**
  - *Step edge:* from 0 to 1 in a single pixel
  - *Ramp edge:* blurring while changing intensity
  - *Roof edge:* rise and goes down within a bit pixels

MAGNITUDE OF THE FIRST DERIVATE CAN BE USED TO DETECT PRESENCE OF AN EDGE AT A POINT IN AN IMAGE

- **Derivatives:**
  - The magnitude of the first derivative can be used to detect the presence of an edge at a point in an image
  - *Gradient* finds the strength and direction at point(x, y)
  - *Magnitude* is the amount of change in that direction
  - *Computing the gradient:* approximation of the partial derivatives over a neighborhood about a point is required.
  - *Prewitt operators*
  - *Sobel operators* is a slight variation of Prewitt since it uses a weight of 2 in the center coefficient, better noise-suppression
- **Marr-Hildrreth Edge Detection** Laplacian of Gaussian instead of Sobel mask
  - *The Gaussian* blur the image reducing the intensity of structures thus reducing the artefacts
  - *The Laplacian* responds equally in intensity in any direction

  - *Zero-Crossing* a zero crossing in a 3x3 neighbourhood centered at p implies that the signs of at least two of uts opposing neighbouring pixels must differ

- **Marr-Hildrreth Edge Detection Algorithm**
  - Blur the image with Gaussian filter → BLUR THE IMAGE REDUCING THE INTENSITY OF STRUCTURES
  - Compute the Laplacina of the blurred image → RESPOND EQUALLY IN INTENSITY IN ANY DIRECTION
  - Find the zero crossing: very sensible to noise, in addition to the previous requirements of zero crossing we require also the absolute value of their numerical difference must exceed a threshold T

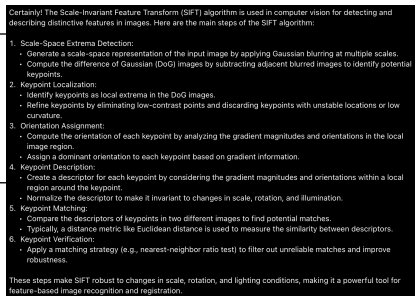
- **Canny Edge Detector**
  - Smooth the input image with a 2D Gaussian filter
  - Compute the gradient magnitude and angle of the smoothed image reduce the noise
  - Non-maxima suppression to the gradient magnitude: for each pixel check the neighbours along the direction of the gradient, if the intensity of the gradient is less than at least one of the two neigh, it must be suppressed

FOR EACH PIXELS IT CHECK THE NEIGHBORHOOD PIXELS ALONG THE GRADIENT DIRECTION (MAGNITUDE)

IF THE INTENSITY OF THE GRADIENT IS LESS THAN AT LEAST ONE OF THE TWO NEIGHBOURS IT MUST BE REMOVED

  - Use double thresholding and connectivity analysis to detect and link edges (Hysteresis thresholding): This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded.

- **Harris Corner Detector** see on the slides
- **SIFT** see on the slides


Certainly! The Scale-Invariant Feature Transform (SIFT) algorithm is used in computer vision for detecting and describing distinctive features in images. Here are the main steps of the SIFT algorithm:

1. Scale-Space Extrema Detection:
   - Generate a scale-space representation of the input image by applying Gaussian blurring at multiple scales.
   - Compute the difference of Gaussian (DoG) images by subtracting adjacent blurred images to identify potential keypoints.
2. Keypoint Localization:
   - Identify keypoints as local extrema in the DoG images.
   - Refine keypoints by eliminating low-contrast points and discarding keypoints with unstable locations or low curvature.
3. Orientation Assignment:
   - Compute the orientation of each keypoint by analyzing the gradient magnitudes and orientations in the local image region.
   - Assign a dominant orientation to each keypoint based on gradient information.
4. Keypoint Description:
   - Create a descriptor for each keypoint by considering the gradient magnitudes and orientations within a local region around the keypoint.
   - Normalize the descriptor to make it invariant to changes in scale, rotation, and illumination.
5. Keypoint Matching:
   - Compare the descriptors of keypoints in two different images to find potential matches.
   - Typically, a distance metric like Euclidean distance is used to measure the similarity between descriptors.
6. Keypoint Verification:
   - Apply a matching strategy (e.g., nearest-neighbor ratio test) to filter out unreliable matches and improve robustness.

These steps make SIFT robust to changes in scale, rotation, and lighting conditions, making it a powerful tool for feature-based image recognition and registration.

- **cv.findContours(mask_thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)**:
  - *cv.RETR_TREE*: retrieves all of the contours and reconstructs a full hierarchy of nested contours
  - *cv.CHAIN_APPROX_SIMPLE*: compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points

THE BORDER FOLLOWING ALGORITHM IS USED IN THE FINDCONTOURS METHOD IN ORDER TO EXTRACT THE TOPOLOGICAL INFORMATION OF THE CONTOURS IN THE GRAY IMAGE THAT WE PASS TO THE ALGORITHM

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

The "Border Following Algorithm" implemented in the paper "Topological Structural Analysis of Digitized Binary Images by Border Following" by Suzuki and Abe is a fundamental technique for tracing the boundary of connected components in binary images. Here are more details about how this algorithm works:

1. **Initialization**: The algorithm starts by identifying an unvisited pixel (usually on the outer boundary of an object) in the binary image. This pixel is chosen as the starting point for tracing the boundary.
2. **Contour Tracing**: From the starting pixel, the algorithm moves in a clockwise or counterclockwise direction along the boundary of the connected component. It continuously follows the edges of the object, marking visited pixels along the way.
3. **Vertex Detection**: The algorithm detects and records vertices (junction points) along the boundary. These vertices represent locations where the boundary direction changes. They are crucial for topological analysis and are typically classified into three types: "Start of a boundary," "End of a boundary," and "Regular vertex."
4. **Loop Handling**: When the algorithm encounters loops or holes within the object, it ensures that these regions are completely traced by following the boundary around them. This process continues until the algorithm returns to the starting pixel, signifying the completion of tracing for that connected component.
5. **Data Structure**: The algorithm maintains data structures to keep track of visited pixels, record the boundary points, and store information about detected vertices. This data is used for subsequent analysis.
6. **Topological Analysis**: Once the algorithm completes tracing for all connected components, the extracted boundary information and vertex data can be used for various topological analyses, such as computing Euler numbers, distinguishing objects and holes, and characterizing the shape of the objects in the image.

The Border Following Algorithm is essential in image processing and computer vision for tasks such as object recognition, shape analysis, and topological feature extraction. It provides a systematic way to extract the structure and topology of binary objects in digital images, making it a valuable tool in the field of image analysis.

GREEN'S FORMULA ESTABLISH A RELATIONSHIP BETWEEN:
- LINE INTEGRALS AROUND A CLOSED CURVE
- DOUBLE INTEGRALS OVER THE REGION ENCLOSED BY THAT CURVE

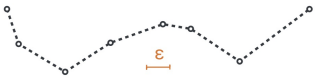
- **cv.contourArea()**: The function computes a contour area. Similarly to moments , the area is computed using the Green formula. Green's Theorem is a fundamental result in vector calculus that establishes a relationship between line integrals around a closed curve and double integrals over the region enclosed by that curve
- **cv.approxPolyDP(cnt, 0.015 * cv.arcLength(cnt, closed=True), closed=True)**:
  - Approximates a curve or a polygon with another curve/polygon with less vertices so that the distance between them is less or equal to the specified precision. → ε

Given is the starting curve C as an ordered set of n points

$$C = (P_1, P_2, P_3, \cdots, P_n)$$

and the distance dimension ε with

$$\varepsilon > 0.$$



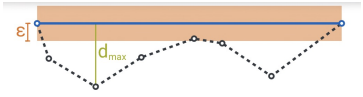
The first approximation of C is the line segment $\overline{P_1P_n}$. The algorithm marks the first and last point to be kept.
Now all $n-2$ inner points are looked at, to find the point P_m that is furthest from that line segment with

$$d_{max} = \max_{i=2\cdots n-1} d(P_i, \overline{P_1P_n}).$$

If d_{max} is within the tolerance

$$d_{max} \leq \varepsilon$$

then the simplification is finished and all inner points can be discarded without the simplified curve being worse than ε.

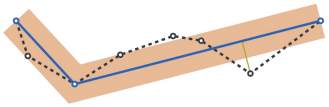


Else, P_m must be kept and the algorithm recursivly processes the new parts

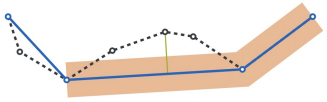
$$C_1 = (P_1, \cdots, P_m)$$

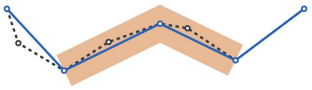
and

$$C_2 = (P_m, \cdots, P_n).$$

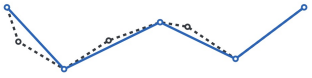


Those parts are then again processed recursivly.





When the recursion is complete, the output curve is the set of all kept points.



- Douglas-Peucker algorithm is an algorithm that decimates a curve composed of line segments to a similar curve with fewer points
  - The starting curve is an ordered set of points or lines and the distance dimension $\epsilon > 0$.
  - The algorithm recursively divides the line. Initially it is given all the points between the first and last point. It automatically marks the first and last point to be kept. It then finds the point that is farthest from the line segment with the first and last points as end points; this point is obviously farthest on the curve from the approximating line segment between the end points. If the point is closer than ϵ to the line segment, then any points not currently marked to be kept can be discarded without the simplified curve being worse than ϵ.
  - If the point farthest from the line segment is greater than ϵ from the approximation then that point must be kept. The algorithm recursively calls itself with the first point and the farthest point and then with the farthest point and the last point, which includes the farthest point being marked as kept.
  - When the recursion is completed a new output curve can be generated consisting of all and only those points that have been marked as kept.

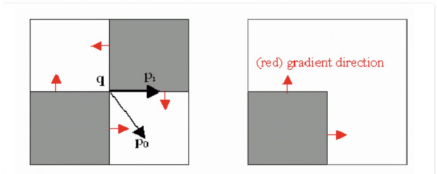
THIS IS OUR ε FOR THE APPROXPOLYDP

CALCULATE THE CONTOUR PERIMETER

- **cv.arcLength(cnt, closed=True)**: Calculates a contour perimeter or a curve length. This step is very important, since it is the epsilon for the cv.approxPolyDP. The correct approximation of the contour depends on selection of epsilon.

- **cv.cornerSubPix(imgray, np.float32(approx_cnt), winSize_sub, zeroZone_sub, criteria_sub):** refines the corner locations.

Consider the image shown below. Suppose, q is the starting corner location and p is the point located within the neighborhood of q.



Source

Clearly, the dot product between the gradient at p and the vector q-p is 0. For instance, for the first case because p_0 lies in a flat region, so the gradient is 0 and hence the dot product. For the second case, the vector q-p_1 lies on the edge and we know that the gradient is perpendicular to the edge so the dot product is 0.

Similarly, we take other points in the neighborhood of q (defined by the winSize parameter) and set the dot product of gradient at that point and the vector to 0 as we did above. Doing so we will get a system of equations. These equations form a linear system that can be solved by the inversion of a single autocorrelation matrix. But this matrix is not always invertible owing to small eigenvalues arising from the pixels very close to q. So, we simply reject the pixels in the immediate neighborhood of q (defined by the zeroZone parameter).

This will give us the new location for q. Now, this will become our starting corner location. Keep iterating until the user-specified termination criterion is reached. I hope you understood this.

- image
- corners
- winSize: Half of the side length of the search window. For example, if winSize=Size(5,5) , then a $(5*2+1) \times (5*2+1) = 11 \times 11$ search window is used.
- zeroZone: Half of the size of the dead region in the middle of the search zone over which the summation in the formula below is not done. It is used sometimes to avoid possible singularities of the autocorrelation matrix. The value of (-1,-1) indicates that there is no such a size.
- criteria: Criteria for termination of the iterative process of corner refinement. That is, the process of corner position refinement stops either after criteria.maxCount iterations or when the corner position moves by less than criteria.epsilon on some iteration.

- WE CONTINUE TO TAKE POINTS IN THE NEIGHBORHOOD OF q
- SET THE DOT PRODUCT AT THAT POINT AND THE VECTOR q-p' TO ZERO
- GET A SYSTEM OF EQUATIONS
- SOLVED BY THE INVERSION OF A SINGLE AUTOCORRELATION MATRIX
- NOT ALWAYS INVERTIBLE
- REJECT THE PIXELS IN THE IMMEDIATE NEIGHBORHOOD OF q

Sub-pixel accurate corner locator is based on the observation that every vector from the center q to a point p located within a neighborhood of q is orthogonal to the image gradient at p subject to image and measurement noise.

- **cv.convexHull():** finds the convex hull of a 2D point set using the Sklansky's algorithm, which then was found to be incorrect
  - A Convex object is one with no interior angles greater than 180 degrees. A shape that is not convex is called Non-Convex or Concave.
  - Hull means the exterior or the shape of the object.
  - Convex Hull of a shape or a group of points is a tight fitting convex boundary around the points or the shape.

- **Flow and Tracking:** track position and movement of an object across a video sequence
- **Motion Field:** would like to estimate the2D velocities of the image points induced by the relative motion between the viewing camera and the observed objects. Projection of the 3D velocity field onto the image plane

IT IS THE APPROXIMATION OF THE MOTION FIELD

- **Optical Flow:** observed 2D displacements of brightness patterns in the image $E(x, y, t)$. (x, y) set of pixels, t time
- **Brightness consistency equation:** the apparent brightness will remain constant during the movement

$$\frac{\partial E}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial E}{\partial y}\frac{\partial y}{\partial t} + \frac{\partial E}{\partial t} = 0$$

Where:

  - $\frac{\partial E}{\partial x}$ and $\frac{\partial E}{\partial y}$ horizontal and vertical gradient (spartial gradients computed via convolution Sobel)
  - $\frac{\partial x}{\partial t}$ and $\frac{\partial y}{\partial t}$ optical flow: $\vec{v} + (\frac{\partial x}{\partial t}, \frac{\partial y}{\partial t}) = (u, v)$
  - $\frac{\partial E}{\partial t}$ image derivative across frames

$$(\Delta E)^T \vec{v} + E_t = 0 \quad \text{IN VECTORIAL FORM}$$

SPATIAL DERIVATIVE IN X DIRECTION — SPATIAL DERIVATIVE IN Y DIRECTION — DIFFERENCE BETWEEN CURRENT IMAGE AND PREVIOUS IMAGE

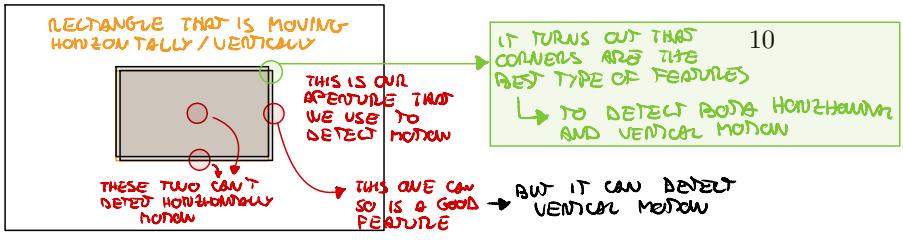
$$E_x u + E_y v + E_t = 0$$

We have 3 know coefficients (x, y, t) and 2 unknowns (u, v). This is like the equation of a line

EXPLAINATION

- **The aperture problem:** the component of the optical flow orthogonal to the spartial image gradient is not constrained by the image brightness constancy equation. Thus only the flow in the gradient direction can be determined NOT AS MUCH CLEAR

THIS IS OUR SCREEN



- **Computing OF with Lukas Kanade flow:**
  - Based on the spartial and temporal variations of the image brightness at all pixels
  - Used to compute dense flow
  - One equation per pixel is not enough so we make an additional assumption: the optical flow is well approximated by a constant vector within any small patch of the image plane
  - Suppose that we use a patch Q with size $N \times N$ (like 5×5), supposing that (u, v) is constant for every pixel in Q we obtain a system of 25 equations in 2 unknowns, which is over-determined. A better solution is obtained with least square fit method thst is suitable for over-estimated system.

$$\underset{u,v}{\operatorname{argmin}} \sum_{p_i \in Q} \left(E_x(p_i)u + E_y(p_i)v + E_t(p_i) \right)^2$$

∀ ELEMENT OF THE PATCH

Or, in matrix form: $\underset{\beta}{\operatorname{argmin}} \|\boldsymbol{y} - \boldsymbol{X}\beta\|^2$

Where:

$$\mathbf{X} = \begin{pmatrix} E_x(\mathbf{p_1}) & E_y(\mathbf{p_1}) \\ E_x(\mathbf{p_2}) & E_y(\mathbf{p_2}) \\ \vdots & \vdots \\ E_x(\mathbf{p_{N^2}}) & E_y(\mathbf{p_{N^2}}) \end{pmatrix} \qquad \mathbf{y} = - \begin{pmatrix} E_t(\mathbf{p_1}) \\ E_t(\mathbf{p_2}) \\ \vdots \\ E_t(\mathbf{p_{N^2}}) \end{pmatrix}$$

$$\beta = (u, v)^T = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

  - Solving this least square problem depends by inverting the matrix $A = X^T X$ where:
    - A should be invertible, thus both eigenvalues must be not zero
    - A should be well conditioned, large λ_1 and $\frac{\lambda_1}{\lambda_2}$ not too large
  - A is exactly the structure tensor used in the Harris corner detector
  - Corners are places in which bot λ_1, λ_2 are big and this also is the case in which the LK flow works best
  - Aperture problem disappear at corners. **Corners are a good place to compute optical flow**

AS PREVIOUSLY MENTIONED

$$\mathbf{X} = \begin{pmatrix} E_x(\mathbf{p_1}) & E_y(\mathbf{p_1}) \\ E_x(\mathbf{p_2}) & E_y(\mathbf{p_2}) \\ \vdots & \vdots \\ E_x(\mathbf{p_{N^2}}) & E_y(\mathbf{p_{N^2}}) \end{pmatrix}$$

SECOND MOMENTUM MATRIX

$$A = X^T X = \begin{pmatrix} \sum E_x E_x & \sum E_x E_y \\ \sum E_y E_x & \sum E_y E_y \end{pmatrix}$$

LIKE IN SIFT

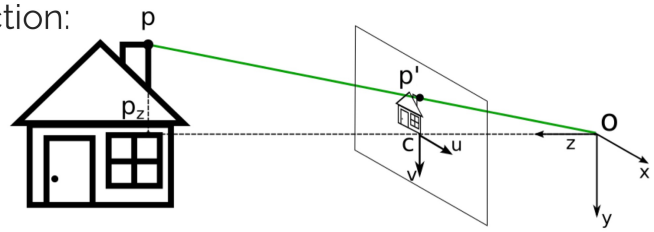
UP IN THE PYRAMID SMALL MOTION IS REMOVED AND LARGE MOTION BECOMES SMALL MOTIONS

– Until now, we were dealing with small motions, so it fails when there is a large motion. To deal with this we use pyramids. When we go up in the pyramid, small motions are removed and large motions become small motions. So by applying Lucas-Kanade there, we get optical flow along with the scale.

3 Pose Estimation Part

- Points
- 2D Projective Space
- Homogeneous coordinates
- Ideal Points
- 2D lines
- Ideal points and the line at infinity
- 2D Projectivities
  - 2D Translation
  - 2D Rotation
  - 2D Rigid Motion: rotation + translation
  - 2D Similarity: rotation + isotopic scale + translation
  - 2D Affine Transformation: linear transformation followed by a translation
  - 2D Projective Transformation: homography since we work with P^2, is a general non singular transformation of homogeneous coordinates
  - Projective VS Affine: main difference is v^T this vector is responsible for the non-linear effects of the projectivity and allows such transformation to model vanishing points
- Projective 3-Space: homogeneous coordinates as a 4-dimensional vector
- Projective transformations: is a linear transformation in P^3 that can be represented by any non-singular 4×4 matrix
- Rigid Motion: the euclidean transformation is a projective transformation composed by a rotation R around an axis and a translation t
- **Pinhole camera:** how small must be the pinhole be?
  - *Large pinhole:* blurring
  - *Small pinhole:* gain focus but less light passes through, diffraction effect
- **Solution:** uses lenses to focuses light onto the sensor
- **Camera and lenses:**
  - All parallel rays converge to one point on a plane located at the focal length f
  - There is a specific distance at which the objects are in focus
  - Rays a re reflected when passing through the lens according to Snell's law

USUALLY IT IS COMMON TO CONSIDER A VIRTUAL IMAGE PLANE IN FRONT OF THE CENTER OF PROJECTION



AND TANGENTIAL

- **Distortion:** imperfect lenses may cause radial distortion
- **Pinhole camera model**

TWO TRIANGLES ARE SIMILAR IFF THE CORRESPONDING ANGLE HAVE THE SAME MEASURE

$$p = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad p' = \begin{pmatrix} u \\ v \end{pmatrix} \quad u = \frac{x}{z} f \quad v = \frac{y}{z} f$$

- **Projection:** when we capture a scene with a pinhole camera, we are mapping a 3D points to a 2D points according the the function $E : R^3 \to R^2$

$$(x, y, z) \to \left(\frac{x}{z} f, \frac{y}{z} f\right)$$

This function is not linear due to the division by z By using homogeneous coordinates , we can express the projection with a linear mapping $E : P^3 \to P^2$

$$P' = \begin{pmatrix} fx \\ fy \\ z \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

The division by z occurs only when we transform P' back to inhomogeneous coordinates

- **Principal point**
  - The image reference system is usually placed at the top-left corner of the image
  - After the projection the coordinates of p' are expressed with respect to the principal point C

$$P' = \begin{pmatrix} fx + c_x z \\ fy + c_y z \\ z \end{pmatrix} = \begin{pmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- **Lens Distortion:** lens distortion produces a non linear displacement of points after their projection
- **Modeling the lens distortion:** as a polynomial function of the radial distance form the lens center
- **Image undistort:** once the lens distortion parameters are known, an image can be "undistorted" as it would be taken with a perfect pinhole camera. Compute the inverse wrap from the ideal undistorted image to the original distorted image according to the distortion function For each pixel u, v in ther target image
  - Move backward in the projection chain to the retinal plane

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = K^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

Some pinhole cameras introduce significant distortion to images. Two major kinds of distortion are radial distortion and tangential distortion.

Radial distortion causes straight lines to appear curved. Radial distortion becomes larger the farther points are from the center of the image. For example, one image is shown below in which two edges of a chess board are marked with red lines. But, you can see that the border of the chess board is not a straight line and doesn't match with the red line. All the expected straight lines are bulged out.

- RADIAL DISTORTION
  - STRAIGHT LINES TO APPEAR CURVED, IT BECOMES LARGER THE FARTHER THE POINTS ARE FROM THE CENTER OF THE IMAGE
- TANGENTIAL DISTORTION
  - IMAGE TAKING LENSE IS NOT ALIGNED PERFECTLY PARALLEL TO THE IMAGE PLANE

Radial distortion can be represented as follows:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Similarly, tangential distortion occurs because the image-taking lense is not aligned perfectly parallel to the imaging plane. So, some areas in the image may look nearer than expected. The amount of tangential distortion can be represented as below:

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

In short, we need to find five parameters, known as distortion coefficients given by:

$$Distortion\ coefficients = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

In addition to this, we need to some other information, like the intrinsic and extrinsic parameters of the camera. Intrinsic parameters are specific to a camera. They include information like focal length (f_x, f_y) and optical centers (c_x, c_y). The focal length and optical centers can be used to create a camera matrix, which can be used to remove distortion due to the lenses of a specific camera. The camera matrix is unique to a specific camera, so once calculated, it can be reused on other images taken by the same camera. It is expressed as a 3x3 matrix:

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

Extrinsic parameters corresponds to rotation and translation vectors which translates a coordinates of a 3D point to a coordinate system.

For stereo applications, these distortions need to be corrected first. To find these parameters, we must provide some sample images of a well defined pattern (e.g. a chess board). We find some specific points of which we already know the relative positions (e.g. square corners in the chess board). We know the coordinates of these points in real world space and we know the coordinates in the image, so we can solve for the distortion coefficients. For better results, we need at least 10 test patterns.

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = (1 + \mathbf{k_1} r^2 + \mathbf{k_2} r^4 + \mathbf{k_3} r^6) \begin{bmatrix} x_d \\ y_d \end{bmatrix} + \begin{bmatrix} 2\mathbf{p_1} x_d y_d + \mathbf{p_2}(r^2 + 2x_d^2) \\ \mathbf{p_1}(r^2 + 2y_d^2) + 2p_2 x_d y_d \end{bmatrix}$$

Distorted location in retinal plane

point location in retinal plane if the pinhole camera were perfect

* Apply the radial distortion function to obtain $(\widetilde{x}', \widetilde{y}', 1)^T$
* Move forward in the projection chain to get the coordinates in the distorted image

$$\begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} = K \begin{pmatrix} \widetilde{x}' \\ \widetilde{y}' \\ 1 \end{pmatrix}$$

– **Camera pose:**
  * All of this is under the assumption that object points were expressed in the camera reference system
  * When dealing with multiple cameras it is common to represent points in a common world reference system
  * A rotation matrix R and a translation vector T express the rigid motion from a world reference system to the camera reference system

– **World to Camera** to be projected a point $p_w \in P^3$ must be first transformed into the camera coordinate system, in 4D homogeneous coordinates we got:

$$p = \begin{pmatrix} R & T \\ 0 & 1 \end{pmatrix} p_w$$

– R **and** T are called the **extrinsic parameter**

– **Complete projection**

$$p' = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R & T \end{pmatrix} \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix}$$

$$p' = K \begin{pmatrix} R & T \end{pmatrix} \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix}$$

– **Finite projective camera**
  * Pixel may not be squared
  * The produced image is not rectangular

$$P = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R & T \end{pmatrix}$$

4 Camera Calibration Part

see notes.

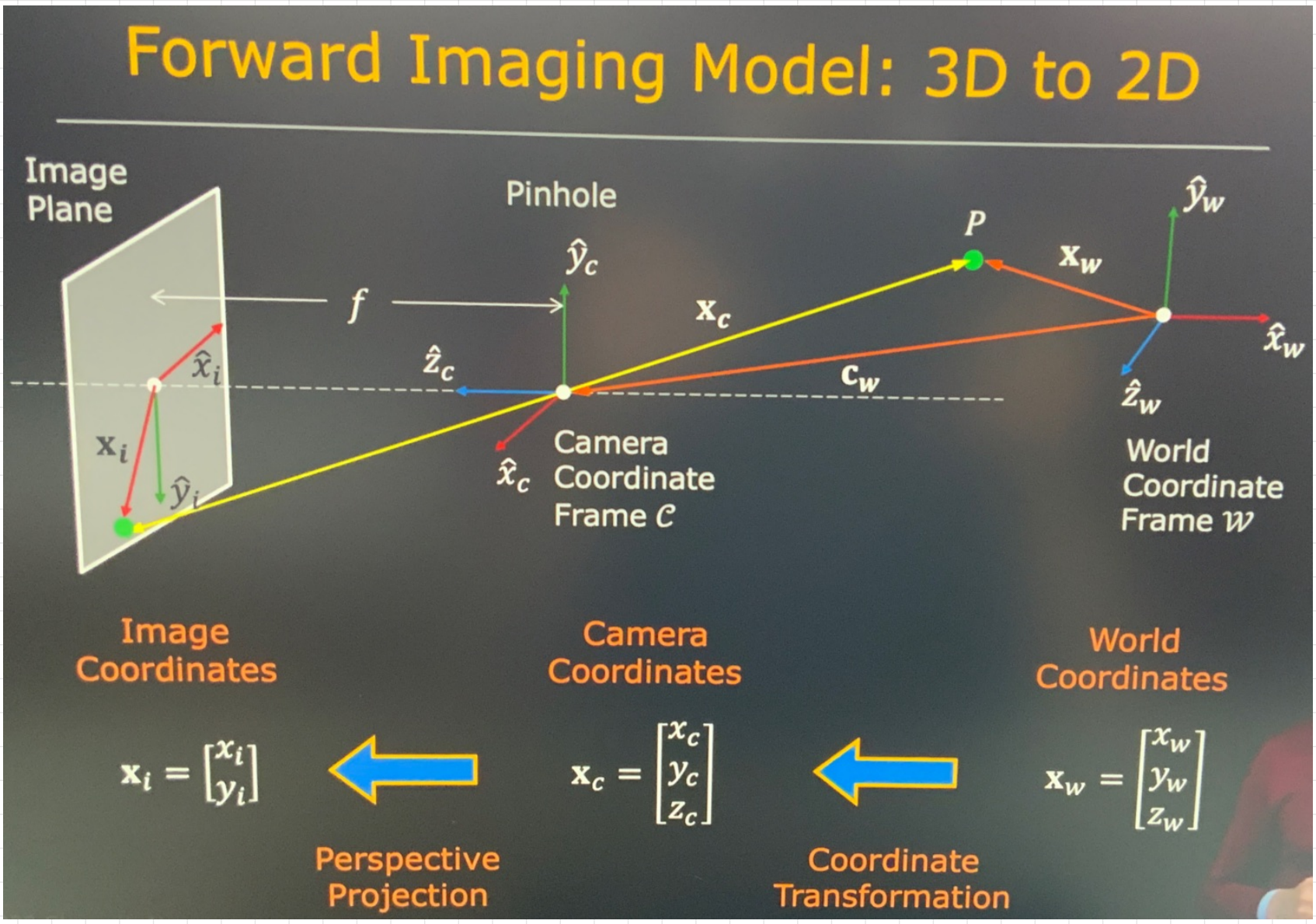
In addition to this, we need to some other information, like the intrinsic and extrinsic parameters of the camera. Intrinsic parameters are specific to a camera. They include information like focal length (f_x, f_y) and optical centers (c_x, c_y). The focal length and optical centers can be used to create a camera matrix, which can be used to remove distortion due to the lenses of a specific camera. The camera matrix is unique to a specific camera, so once calculated, it can be reused on other images taken by the same camera.

SO IT FINDS THE CHESS BOARD CORNER IN THE IMAGE (2D) THE 3D POINTS ARE GIVEN BY THE USER THEN IN THE CALIBRATE CAMERA FUNCTION

4.1 Used Methods

- **findChessboardCorners():** Finds the positions of internal corners of the chessboard.
- **calibrateCamera()** GIVEN THE 2D IMAGE POINTS AND THE 3D WORLD COORDINATE FRAME POINTS — RETURN CAMERA INTRINSIC AND EXTRINSIC PARAMETERS AND THE DISTORTION COEFFICIENTS
- **getOptimalNewCameraMatrix():** The function computes and returns the optimal new camera intrinsic matrix based on the free scaling parameter. By varying this parameter, you may retrieve only sensible pixels alpha=0 , keep all the original image pixels if there is valuable information in the corners alpha=1 , or get something in between. When alpha¿0 , the undistorted result is likely to have some black pixels corresponding to "virtual" pixels outside of the captured distorted image. — ROI
- **undistort():** Transforms an image to compensate for lens distortion. The function transforms an image to compensate radial and tangential lens distortion.
- **solvePnP():** returns the rotation and the translation vectors that transform a 3D point expressed in the object coordinate frame to the camera coordinate frame — RETURNS ROTATION AND TRANSLATION VECTOR USED TO GET THE PROJECTION OF POINTS FROM WORLD COORDINATE FRAME TO CAMERA COORDINATE FRAME
- **projectPoints():** The function computes the 2D projections of 3D points to the image plane, given intrinsic and extrinsic camera parameters. — 3D TO 2D → GIVEN THE INTRINSIC AND EXTRINSIC PARAMETERS
- **Alternative methods to projectPoints() is:**
  - Run calibrateCamera to obtain the 3×3 matrix as cameraMatrix, a 4×1 matrix as distCoeffs, and rvecs and tvecs that are vectors of 3×1 rotation(R) and 3×1 transformation(t) matrices.
  - Run Rodriguez with the 3×1 rotation(R) matrix to obtain the 3×3 rotation(R) matrix
  - Concatenate the rotation matrix with the transformation column vector
    ADDING A new 0 0 0 1 SO WE EXPRESS IN HOMOGENEOUS COORDINATES

* Multiply the $[cameraMatrix]$ with the $[R|t]$ to obtain the 4×3 projection matrix
* To obtain the projection points multiply the projection matrix with the $4 \times N$ matrix containing N 3D points in homogeneous coordinates



PROSPECTIVE PROJECTION → ASSUMING THE POINTS ARE DEFINED IN THE CAMERA COORDINATE FRAME

$\frac{x_i}{f} = \frac{x_c}{z_c}$ AND $\frac{y_i}{f} = \frac{y_c}{z_c}$ ⇨ $x_i = f\frac{x_c}{z_c}$ AND $y_i = f\frac{y_c}{z_c}$

BY THE SIMILAR TRIANGULAR LEMMA ↑

COORDINATES OF THE PROJECTION OF THE POINT i IN THE IMAGE PLANE

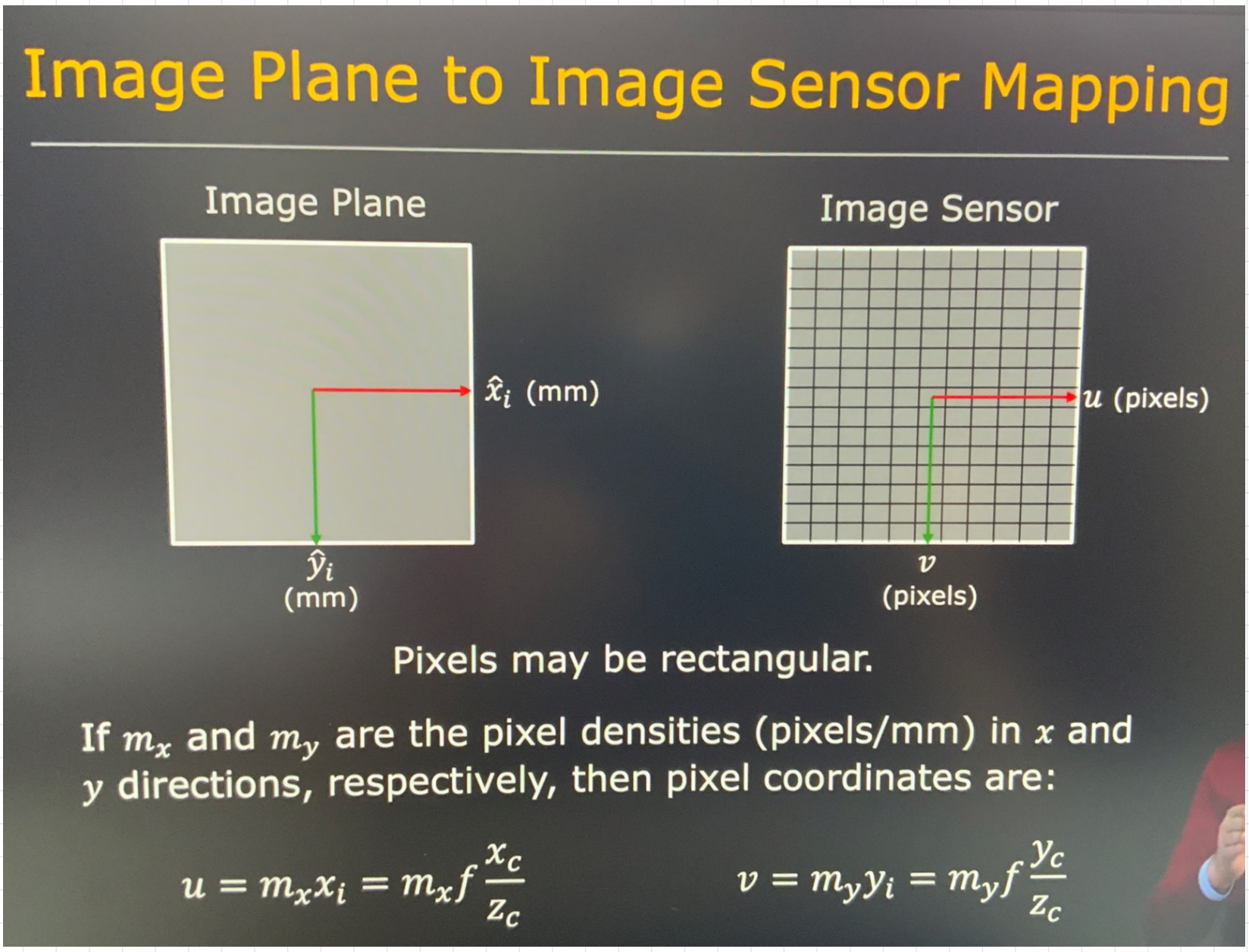
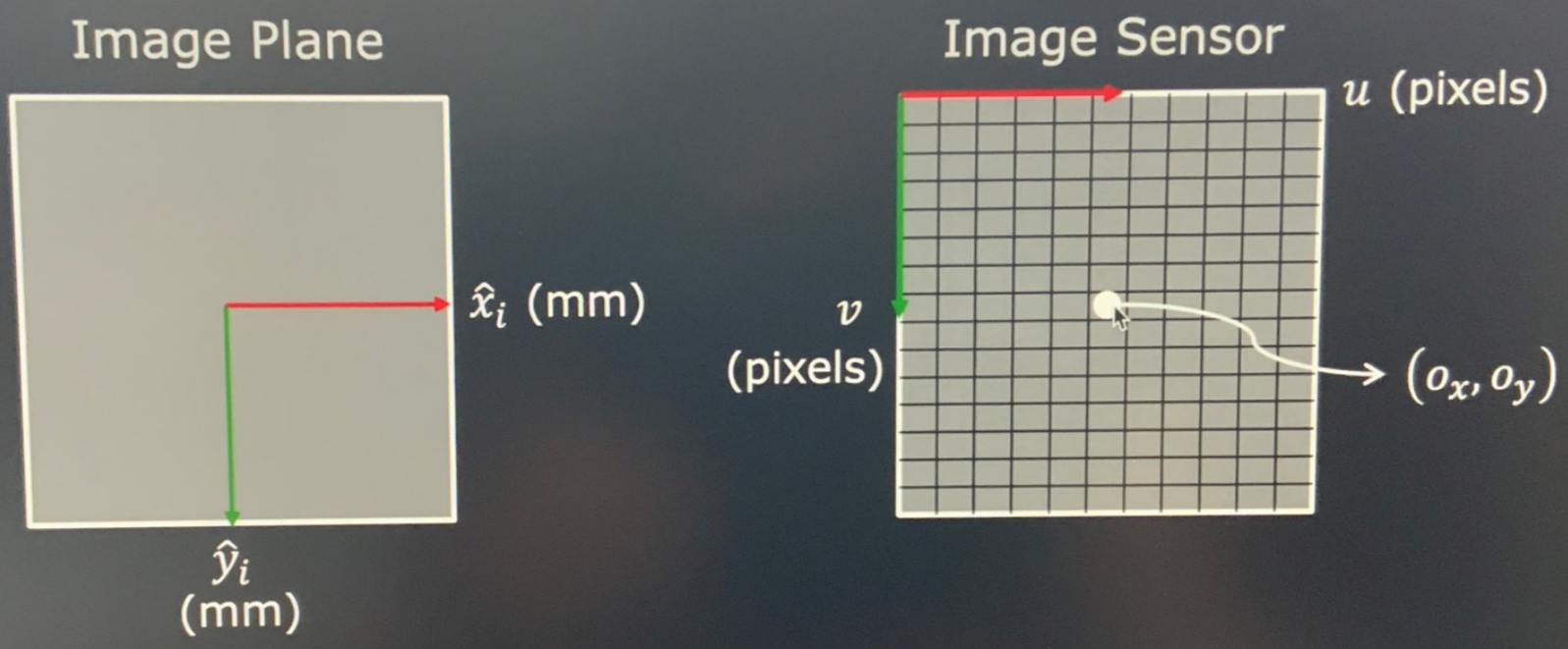


Image Plane to Image Sensor Mapping



We usually treat the top-left corner of the image sensor as its origin (easier for indexing). If pixel (o_x, o_y) is the Principle Point where the optical axis pierces the sensor, then:

$$u = m_x f \frac{x_c}{z_c} + o_x \qquad v = m_y f \frac{y_c}{z_c} + o_y$$

Perspective Projection

$$u = m_x f \frac{x_c}{z_c} + o_x \qquad v = m_y f \frac{y_c}{z_c} + o_y$$

$$u = f_x \frac{x_c}{z_c} + o_x \qquad v = f_y \frac{y_c}{z_c} + o_y$$

where: $(f_x, f_y) = (m_x f, m_y f)$ are the focal lengths in pixels in the x and y directions.

(f_x, f_y, o_x, o_y): Intrinsic parameters of the camera.
They represent the camera's internal geometry.

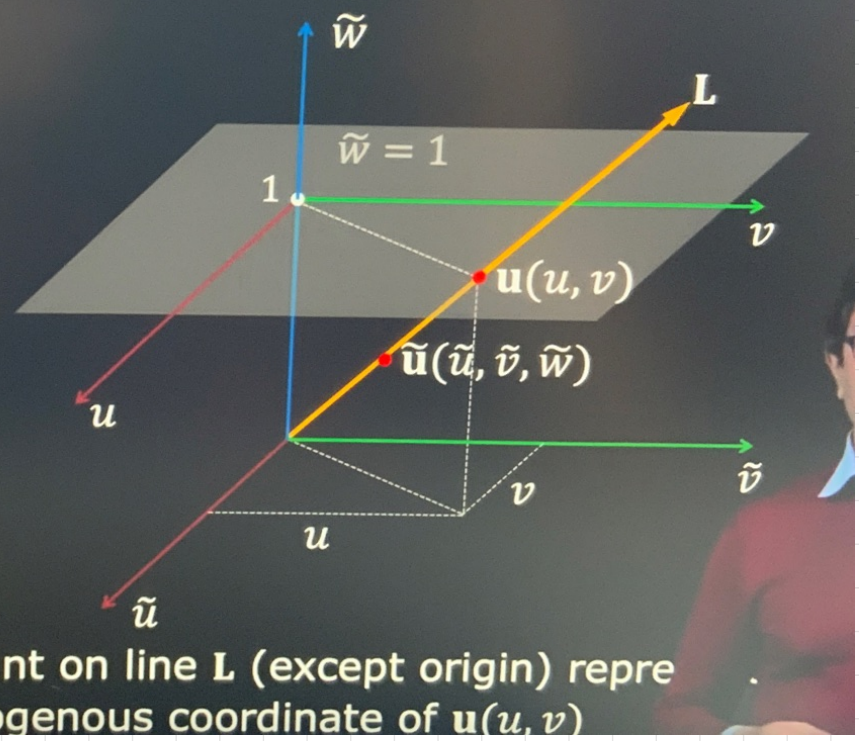
THESE ARE NOW LINEAR, IT IS CONVINIENT TO EXPRESS THEM AS LINEAR EQUATION

Homogenous Coordinates

The homogenous representation of a 2D point $\mathbf{u} = (u, v)$ is a 3D point $\tilde{\mathbf{u}} = (\tilde{u}, \tilde{v}, \tilde{w})$. The third coordinate $\tilde{w} \neq 0$ is fictitious such that:

$$u = \frac{\tilde{u}}{\tilde{w}} \qquad v = \frac{\tilde{v}}{\tilde{w}}$$

$$\mathbf{u} \equiv \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{w}u \\ \tilde{w}v \\ \tilde{w} \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \tilde{\mathbf{u}}$$



Every point on line $\mathbf{L}$ (except origin) repre
the homogenous coordinate of $\mathbf{u}(u, v)$

REVIEW

Homogenous Coordinates

The homogenous representation of a 3D point $\mathbf{x} = (x, y, z) \in \mathcal{R}^3$ is a 4D point $\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w}) \in \mathcal{R}^4$.
The fourth coordinate $\tilde{w} \neq 0$ is fictitious such that:

$$x = \frac{\tilde{x}}{\tilde{w}} \qquad y = \frac{\tilde{y}}{\tilde{w}} \qquad z = \frac{\tilde{z}}{\tilde{w}}$$

$$\mathbf{x} \equiv \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{w}x \\ \tilde{w}y \\ \tilde{w}z \\ \tilde{w} \end{bmatrix} \equiv \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{bmatrix} = \tilde{\mathbf{x}}$$

Perspective Projection

Perspective projection equations:

$$u = f_x \frac{x_c}{z_c} + o_x \qquad v = f_y \frac{y_c}{z_c} + o_y$$

Homogenous coordinates of (u, v):

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} \equiv \begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix} = \begin{bmatrix} f_x x_c + z_c o_x \\ f_y y_c + z_c o_y \\ z_c \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

where: $(u, v) = (\tilde{u}/\tilde{w}, \tilde{v}/\tilde{w})$

Linear Model for Perspective Projection

Intrinsic Matrix

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

Calibration Matrix:

$$K = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$$

Upper Right Triangular Matrix

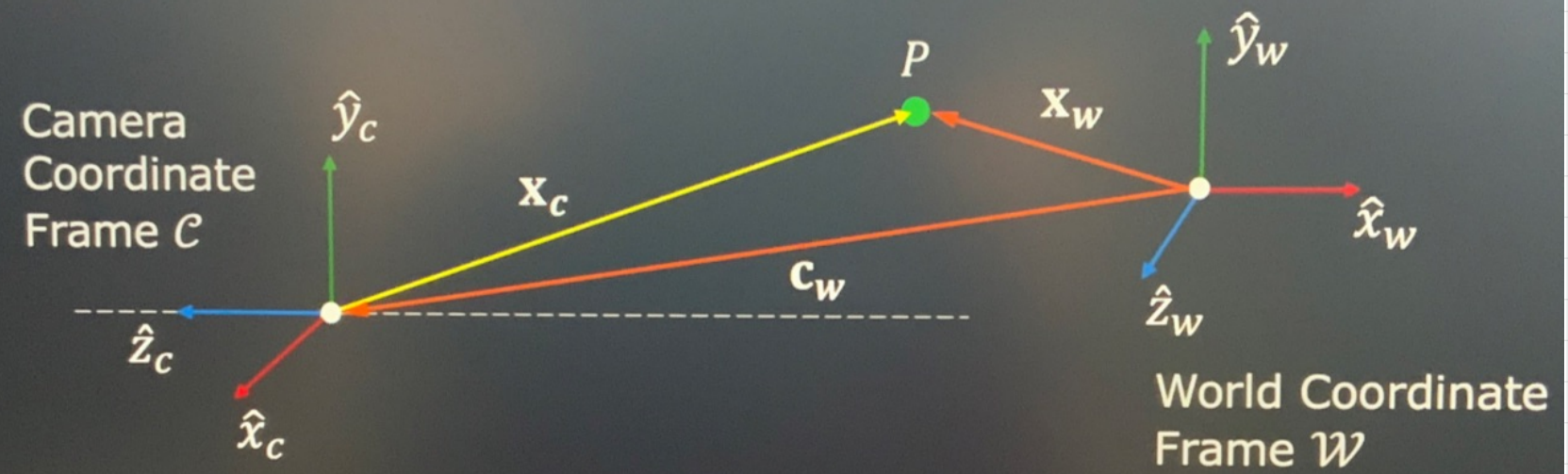
Intrinsic Matrix:

$$M_{int} = [K|0] = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\tilde{\mathbf{u}} = [K|0]\,\tilde{\mathbf{x}}_c = M_{int}\,\tilde{\mathbf{x}}_c$$

COORDINATE TRANSFORMATION → WORLD COORDINATES - CAMERA FRAME

Extrinsic Parameters



Position $\mathbf{c}_w$ and Orientation R of the camera in the world coordinate frame $\mathcal{W}$ are the camera's Extrinsic Parameters.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

→ Row 1: Direction of $\hat{x}_c$ in world coordinate frame
→ Row 2: Direction of $\hat{y}_c$ in world coordinate frame
→ Row 3: Direction of $\hat{z}_c$ in world coordinate frame

Orientation/Rotation Matrix R is Orthonormal

Orthonormal Vectors and Matrices

Orthonormal Vectors: Two vectors $\mathbf{u}$ and $\mathbf{v}$ are orthonormal if and only if:

$$dot(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T\mathbf{v} = 0 \quad \text{and} \quad \mathbf{u}^T\mathbf{u} = \mathbf{v}^T\mathbf{v} = 1$$

(Orthogonality) (Unit length)

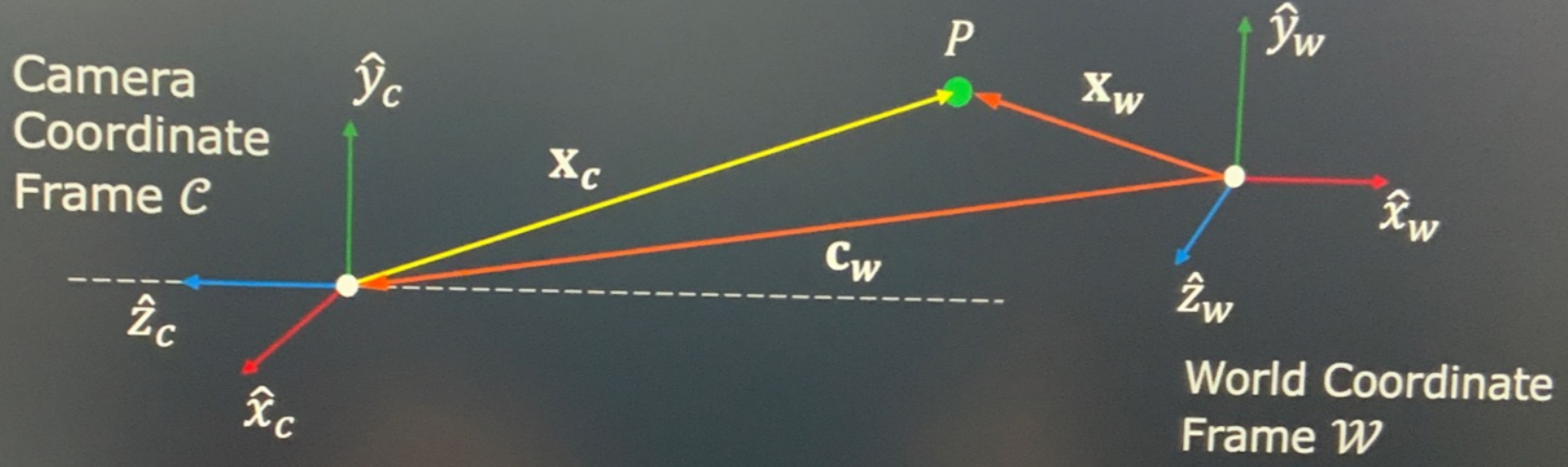
Example: The x-, y- and z-axes of $\mathbb{R}^3$ Euclidean space

Orthonormal Matrix: A square matrix R whose row (or column) vectors are orthonormal. For such a matrix:

$$R^{-1} = R^T \qquad R^TR = RR^T = I$$

A Rotation Matrix is an Orthonormal Matrix

World-to-Camera Transformation



Given the extrinsic parameters $(R, \mathbf{c}_w)$ of the camera, the camera-centric location of the point P in the world coordinate frame is:

$$\mathbf{x}_c = R(\mathbf{x}_w - \mathbf{c}_w) = R\mathbf{x}_w - R\mathbf{c}_w = R\mathbf{x}_w + \mathbf{t}$$

$$\mathbf{t} = -R\mathbf{c}_w$$

$$\mathbf{x}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Extrinsic Matrix

Rewriting using homogenous coordinates:

$$\tilde{\mathbf{x}}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Extrinsic Matrix: $M_{ext} = \begin{bmatrix} R_{3\times3} & \mathbf{t} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$$\tilde{\mathbf{x}}_c = M_{ext}\tilde{\mathbf{x}}_w$$

Projection Matrix P

Camera to Pixel

$$\begin{bmatrix}\tilde{u}\\ \tilde{v}\\ \tilde{w}\end{bmatrix} = \begin{bmatrix}f_x & 0 & o_x & 0\\ 0 & f_y & o_y & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}x_c\\ y_c\\ z_c\\ 1\end{bmatrix}$$

$$\tilde{\mathbf{u}} = M_{int}\,\tilde{\mathbf{x}}_c$$

World to Camera

$$\begin{bmatrix}x_c\\ y_c\\ z_c\\ 1\end{bmatrix} = \begin{bmatrix}r_{11} & r_{12} & r_{13} & t_x\\ r_{21} & r_{22} & r_{23} & t_y\\ r_{31} & r_{32} & r_{33} & t_z\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_w\\ y_w\\ z_w\\ 1\end{bmatrix}$$

$$\tilde{\mathbf{x}}_c = M_{ext}\tilde{\mathbf{x}}_w$$

Combining the above two equations, we get the full projection matrix P:

$$\tilde{\mathbf{u}} = M_{int}M_{ext}\,\tilde{\mathbf{x}}_w = P\,\tilde{\mathbf{x}}_w$$

$$\begin{bmatrix}\tilde{u}\\ \tilde{v}\\ \tilde{w}\end{bmatrix} = \begin{bmatrix}p_{11} & p_{12} & p_{13} & p_{14}\\ p_{21} & p_{22} & p_{23} & p_{24}\\ p_{31} & p_{32} & p_{33} & p_{34}\end{bmatrix}\begin{bmatrix}x_w\\ y_w\\ z_w\\ 1\end{bmatrix}$$

CAMERA CALIBRATION

Camera Calibration Procedure

Step 3: For each corresponding point i in scene and image:

$$\underbrace{\begin{bmatrix} u^{(i)} \\ v^{(i)} \\ 1 \end{bmatrix}}_{\text{Known}} \equiv \underbrace{\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}}_{\text{Unknown}} \underbrace{\begin{bmatrix} x_w^{(i)} \\ y_w^{(i)} \\ z_w^{(i)} \\ 1 \end{bmatrix}}_{\text{Known}}$$

Expanding the matrix as linear equations:

$$u^{(i)} = \frac{p_{11}x_w^{(i)} + p_{12}y_w^{(i)} + p_{13}z_w^{(i)} + p_{14}}{p_{31}x_w^{(i)} + p_{32}y_w^{(i)} + p_{33}z_w^{(i)} + p_{34}}$$

$$v^{(i)} = \frac{p_{21}x_w^{(i)} + p_{22}y_w^{(i)} + p_{23}z_w^{(i)} + p_{24}}{p_{31}x_w^{(i)} + p_{32}y_w^{(i)} + p_{33}z_w^{(i)} + p_{34}}$$

Camera Calibration Procedure

Step 4: Rearranging the terms

$$\underbrace{\begin{bmatrix} x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & 0 & 0 & 0 & 0 & -u_1x_w^{(1)} & -u_1y_w^{(1)} & -u_1z_w^{(1)} & -u_1 \\ 0 & 0 & 0 & 0 & x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & -v_1x_w^{(1)} & -v_1y_w^{(1)} & -v_1z_w^{(1)} & -v_1 \\ \vdots & & & & & & & & & & & \\ x_w^{(i)} & y_w^{(i)} & z_w^{(i)} & 1 & 0 & 0 & 0 & 0 & -u_ix_w^{(i)} & -u_iy_w^{(i)} & -u_iz_w^{(i)} & -u_i \\ 0 & 0 & 0 & 0 & x_w^{(i)} & y_w^{(i)} & z_w^{(i)} & 1 & -v_ix_w^{(i)} & -v_iy_w^{(i)} & -v_iz_w^{(i)} & -v_i \\ \vdots & & & & & & & & & & & \\ x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & 0 & 0 & 0 & 0 & -u_nx_w^{(n)} & -u_ny_w^{(n)} & -u_nz_w^{(n)} & -u_n \\ 0 & 0 & 0 & 0 & x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & -v_nx_w^{(n)} & -v_ny_w^{(n)} & -v_nz_w^{(n)} & -v_n \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix}}_{\mathbf{p}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Known — Unknown

Step 5: Solve for $\mathbf{p}$

$$A\,\mathbf{p} = \mathbf{0}$$

Scale of Projection Matrix

Projection matrix acts on homogenous coordinates.

We know that:

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} \equiv k \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} \quad (k \neq 0 \text{ is any constant})$$

That is:

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \equiv k \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Therefore, Projection Matrices P and kP produce the same homogenous pixel coordinates.

Projection Matrix P is defined only up to a scale.

Least Squares Solution for P

Option 1: Set scale so that: $p_{34} = 1$

Option 2: Set scale so that: $\|\mathbf{p}\|^2 = 1$

We want $A\mathbf{p}$ as close to 0 as possible and $\|\mathbf{p}\|^2 = 1$:

$$\min_{\mathbf{p}} \|A\mathbf{p}\|^2 \text{ such that } \|\mathbf{p}\|^2 = 1$$

$$\min_{\mathbf{p}} (\mathbf{p}^T A^T A \mathbf{p}) \text{ such that } \mathbf{p}^T \mathbf{p} = 1$$

Define Loss function $L(\mathbf{p}, \lambda)$:

$$L(\mathbf{p}, \lambda) = \mathbf{p}^T A^T A \mathbf{p} - \lambda(\mathbf{p}^T \mathbf{p} - 1)$$

(Similar to Solving Homography in Image Stitching)

Constrained Least Squares Solution

Taking derivatives of $L(\mathbf{p}, \lambda)$ w.r.t $\mathbf{p}$: $2A^TA\mathbf{p} - 2\lambda\mathbf{p} = \mathbf{0}$

$$\boxed{A^TA\mathbf{p} = \lambda\mathbf{p}}$$ Eigenvalue Problem

Eigenvector $\mathbf{p}$ with smallest eigenvalue λ of matrix A^TA minimizes the loss function $L(\mathbf{p})$.

Rearrange solution $\mathbf{p}$ to form the projection matrix P.

Extracting Intrinsic/Extrinsic Parameters

We know that:

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

M_{int} M_{ext}

That is:

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = KR$$

Given that K is an Upper Right Triangular matrix and R is an Orthonormal matrix, it is possible to uniquely "decouple" K and R from their product using "QR factorization".

Extracting Intrinsic/Extrinsic Parameters

We know that:

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

That is:

$$\begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = K\mathbf{t}$$

Therefore:

$$\mathbf{t} = K^{-1} \begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix}$$