Camera Models and Calibration



Antonio Carta

2026-10-08

University of Pisa

antonio.carta@unipi.it

Motivation

Recap: Perspective Projection and Homogeneous Coordinates

- we defined rigid transformations in the homogeneous coordinate system
- in the end, every transformation becomes a matrix
- we can easily compose and invert transformations

Recap: camera matrix P = K[R| -Rt]

- the transformation from 3D coordinates to the pixels grid is a 3x4 matrix P that we need to estimate
- intrinsics: parameters controlling the relationship between image and sensor plane (i.e. camera hardware)
- extrinsics: camera view (rotation and translation of the camera view w.r.t. world coordinates)

$$P = K[R \mid -RT]$$

Camera Calibration

- setup: take a set of photos that respect sufficient conditions for a good estimate
- camera calibration: estimate the matrix P
- profit: most 3D tasks require knowing P

Outline

- examples of camera matrices
- direct linear transform (DLT)
- practical aspects of camera calibration

Examples of Camera Views

Extrinsics Parameters

Converting from 3D world coordinates to 2D image coordinates (in meters)

- first, translate the frame s.t. the origin of the world and camera frame matched
- the rotate the frame appropriately so that they match

$$P = K[R| - RT]$$

Examples of Camera Views

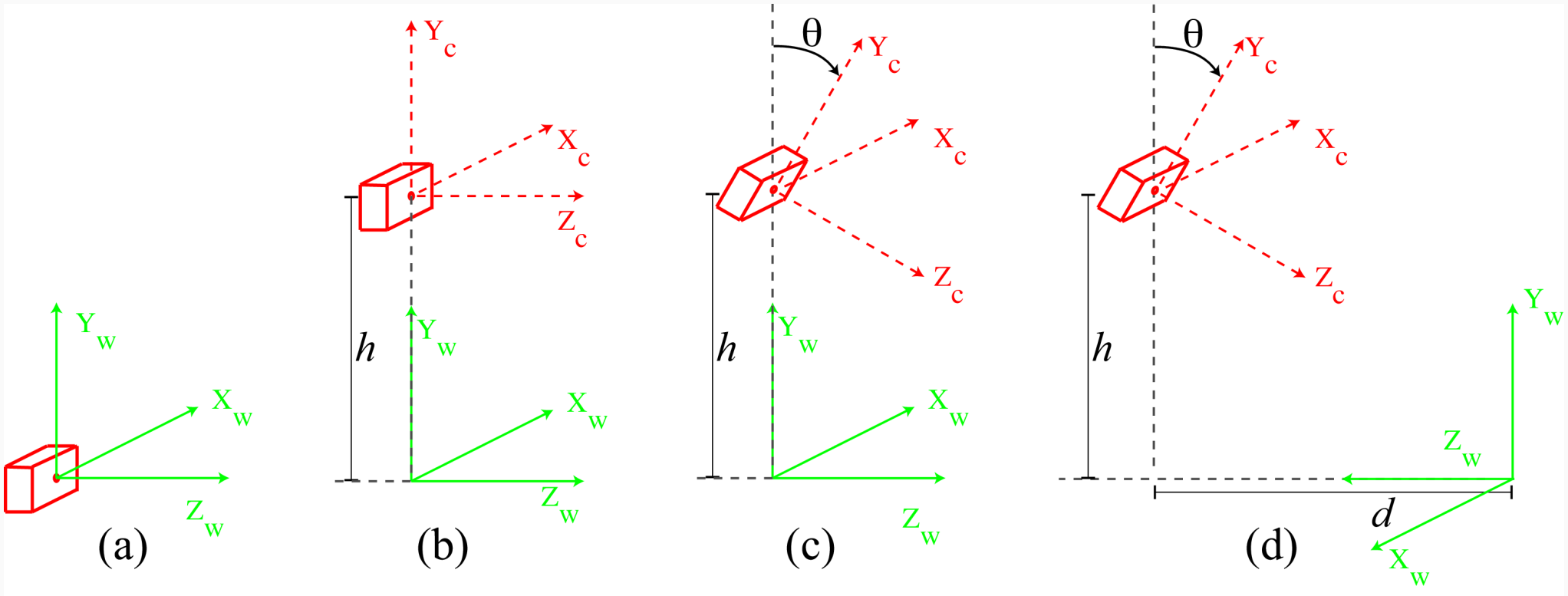


Figure 1: source: visionbook

Example (a)

Simple case where we only need the projection equations:

- camera located at the origin of world coordinates
- axis aligned
- we set the center of the image $(c_x, c_y) = (0, 0)$

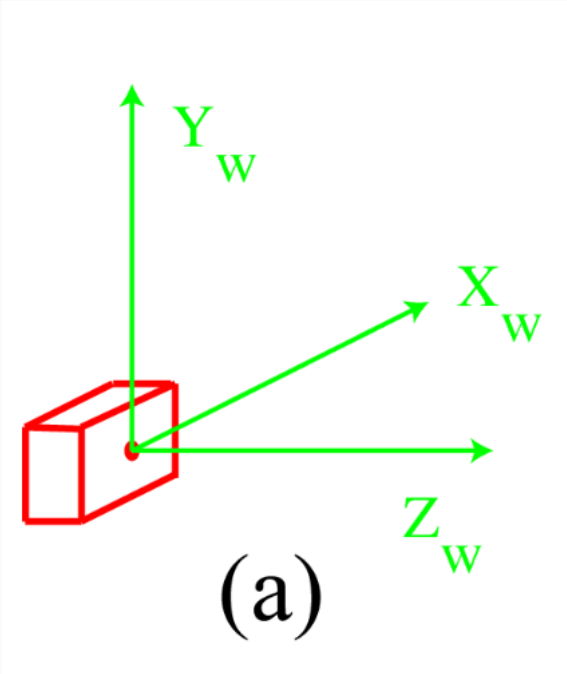


Figure 2: source: visionbook

Example (a)

$$M = K \begin{bmatrix} \boldsymbol{R} & -\boldsymbol{RT} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

reminder:

- $a = f\frac{N}{w}$, with f focal length, N number of pixels w pixel width

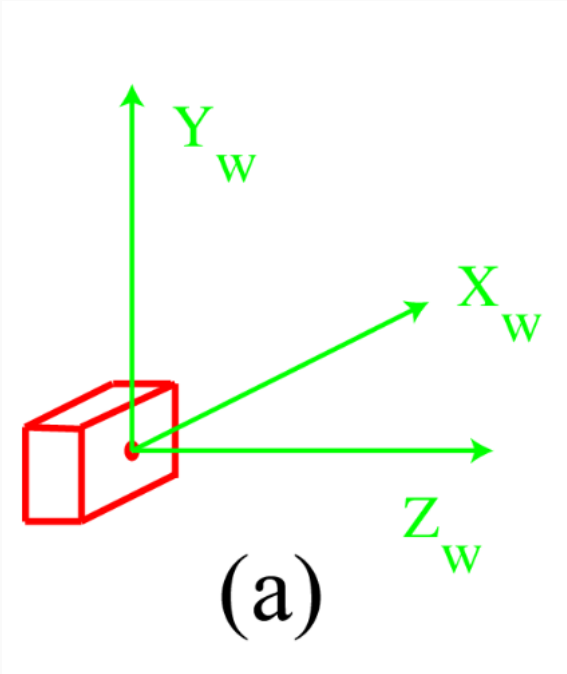


Figure 3: source: visionbook

Example (b)

second scenario:

- person holding a camera toward the horizon
- optical axis still parallel to the ground
- axis are aligned but there is a vertical displacement

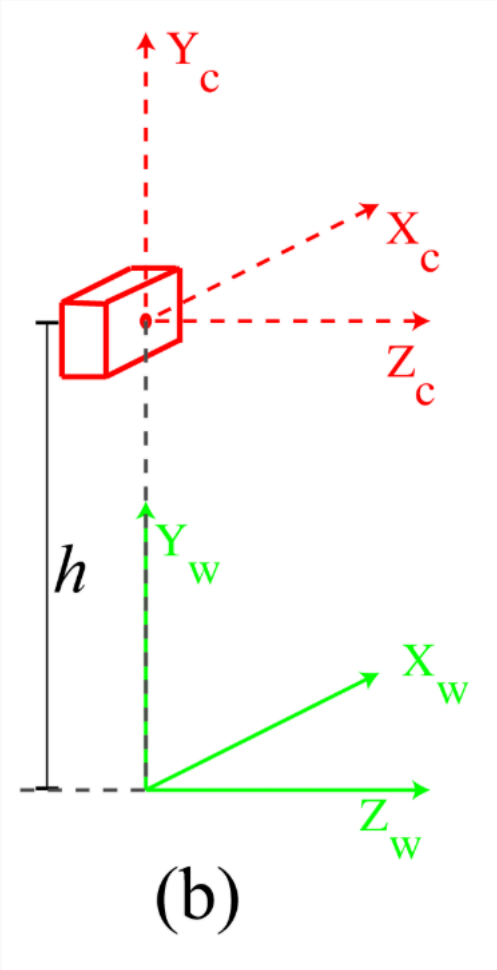


Figure 4: source: visionbook

Example (b)

$$M = K \begin{bmatrix} \boldsymbol{R} & -\boldsymbol{RT} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

scenario (a):

$$M = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

scenario (b): The displacement is a vector $\boldsymbol{T} = (0, h)$

$$M = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -h \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & -ah \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

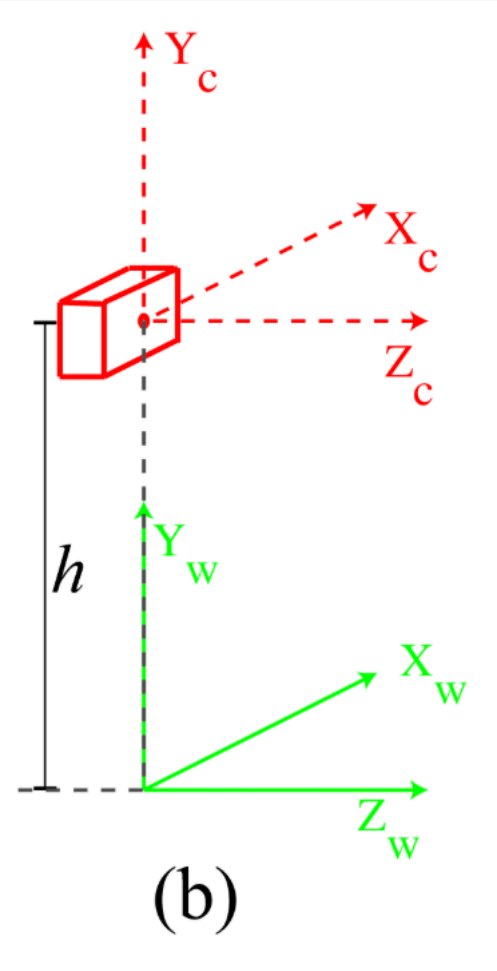


Figure 5: source: visionbook

Example (b)

scenario (b): The displacement is a vector $\boldsymbol{T} = (0, h)$

$$M = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -h \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & -ah \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

in heterogeneous coordinates:

$x = a\frac{X}{Z}, y = a\frac{Y-h}{Z}$

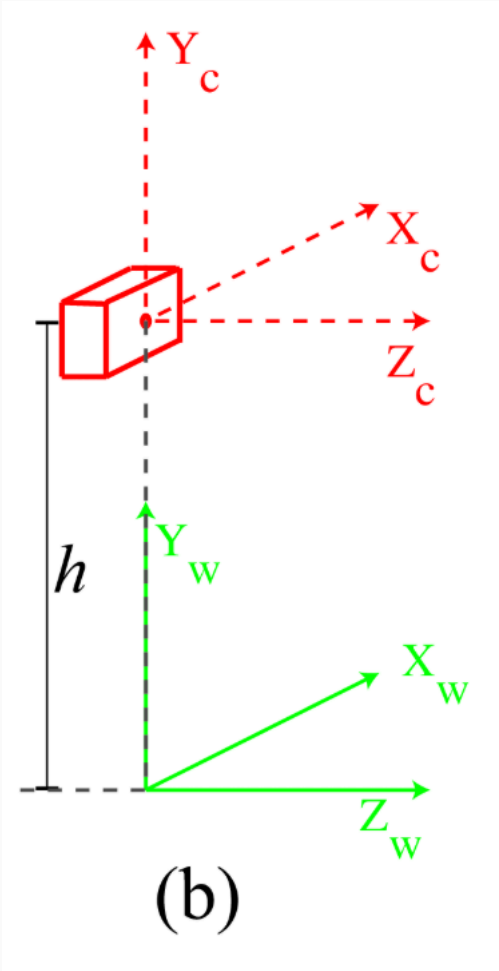


Figure 6: source: visionbook

Example (b): Horizon Line

in heterogeneous coordinates: $x = a\frac{X}{Z}, y = a\frac{Y-h}{Z}$

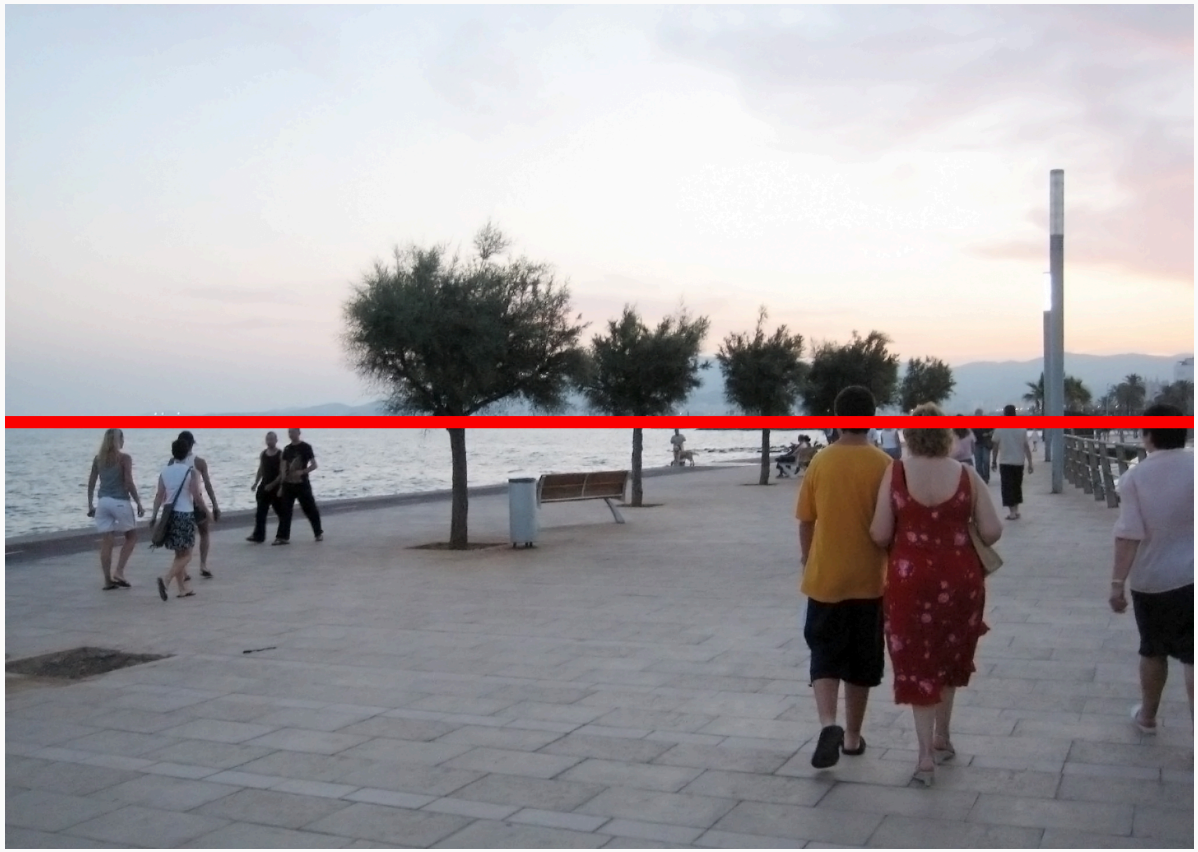


Figure 7: source: visionbook

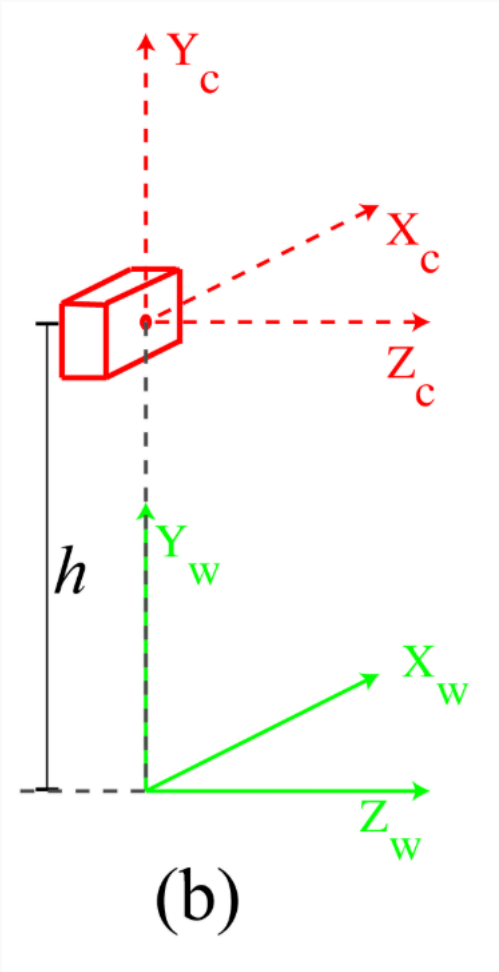


Figure 8: source: visionbook

Example (c)

- now the camera is also tilted, pointing to the ground by a rotation $\theta > 0$
- the rotation is around the X_c axis (X_c stays fixed, the other two move)

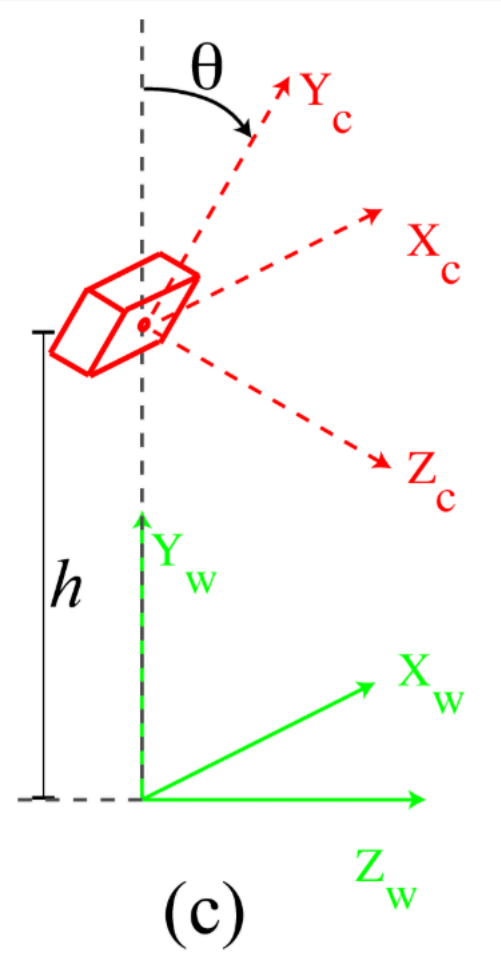


Figure 9: source: visionbook

Example (c)

$$M = K \begin{bmatrix} \boldsymbol{R} & -\boldsymbol{RT} \\ \boldsymbol{0}^T & 1 \end{bmatrix}$$

scenario (c), matrix M_2:

$$\begin{bmatrix} \boldsymbol{R} & -\boldsymbol{RT} \\ \boldsymbol{0}^T & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -h \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & -h\cos(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) & h\sin(\theta) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

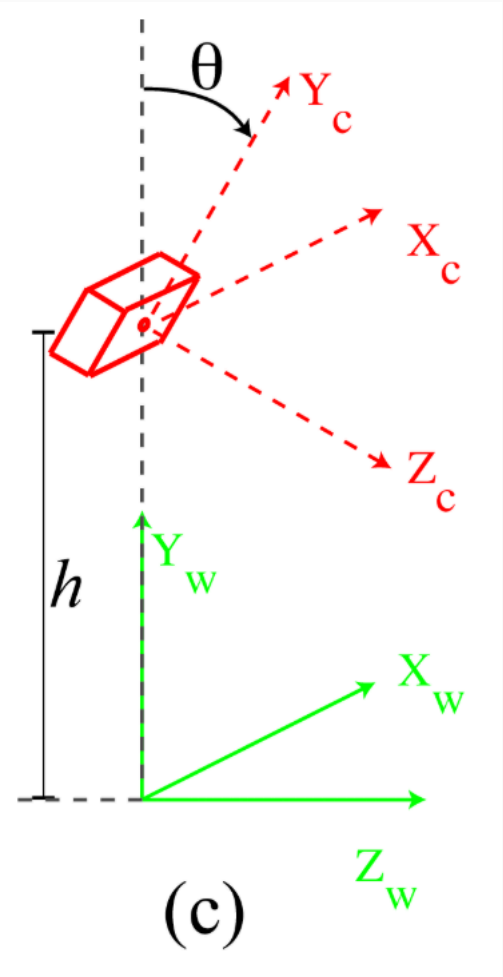


Figure 10: source: visionbook

Example (c)

combining it with the extrinsic parameters we computed before for scenario (b): $M =$

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & -h\cos(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) & h\sin(\theta) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

final result:

$$M = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a\cos(\theta) & a\sin(\theta) & -ah\cos(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) & h\sin(\theta) \end{bmatrix}$$

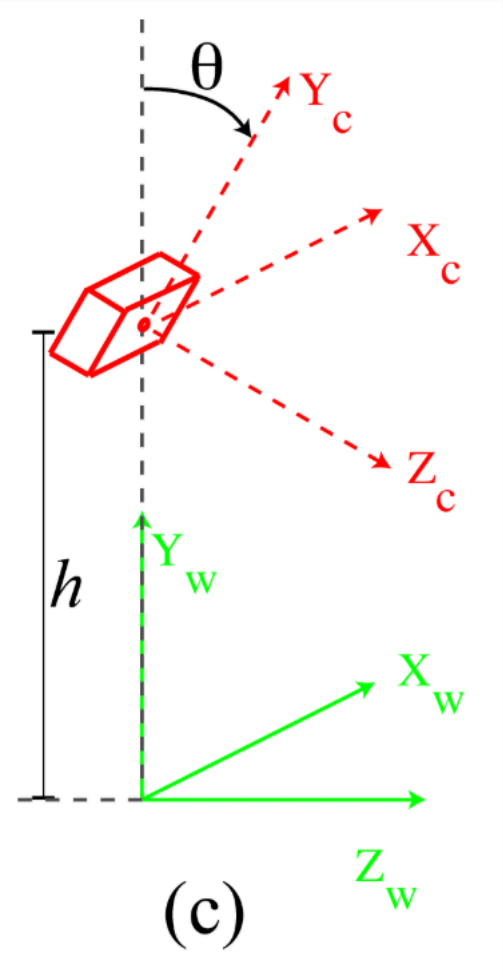


Figure 11: source: visionbook

Example (c)

final result:

$$MP = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a\cos(\theta) & a\sin(\theta) & -ah\cos(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) & h\sin(\theta) \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

in heterogeneous coordinates:

$$x = a\frac{X}{\sin(\theta)(h-Y)+\cos(\theta)Z}$$

$$y = a\frac{\cos(\theta)(Y-h)+\sin(\theta)Z}{\sin(\theta)(h-Y)+\cos(\theta)Z}$$

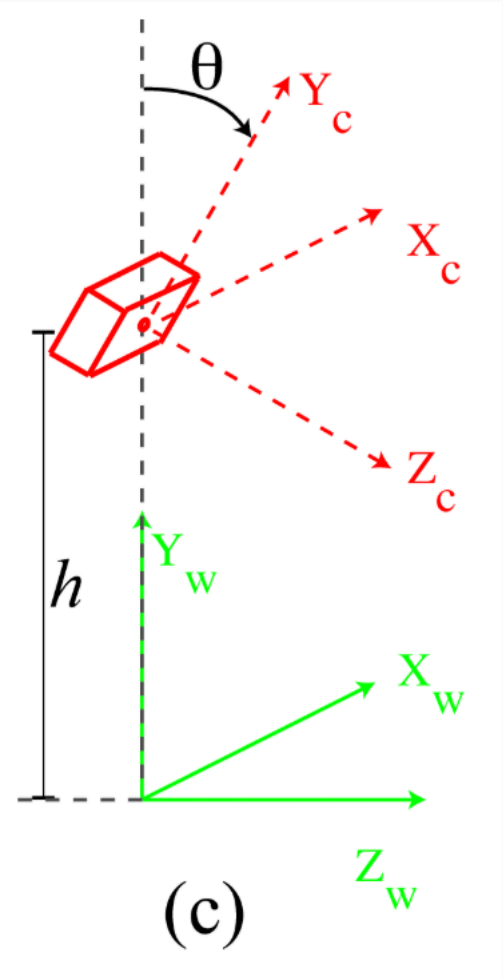


Figure 12: source: visionbook

Example (c)

in heterogeneous coordinates:

$$x = a\frac{X}{\sin(\theta)(h-Y)+\cos(\theta)Z}$$

$$y = a\frac{\cos(\theta)(Y-h)+\sin(\theta)Z}{\sin(\theta)(h-Y)+\cos(\theta)Z}$$

- if $\theta = 0$ we have scenario (b)
- if $\theta = 90$ degree (camera looking down) $x = a\frac{X}{h-Y}, a\frac{Z}{h-Y}$
  - similar to the standard projection equations $x = a\frac{X}{Z}$
  - $h - Y$ (distance from the camera) acts as Z
  - Z acts as Y

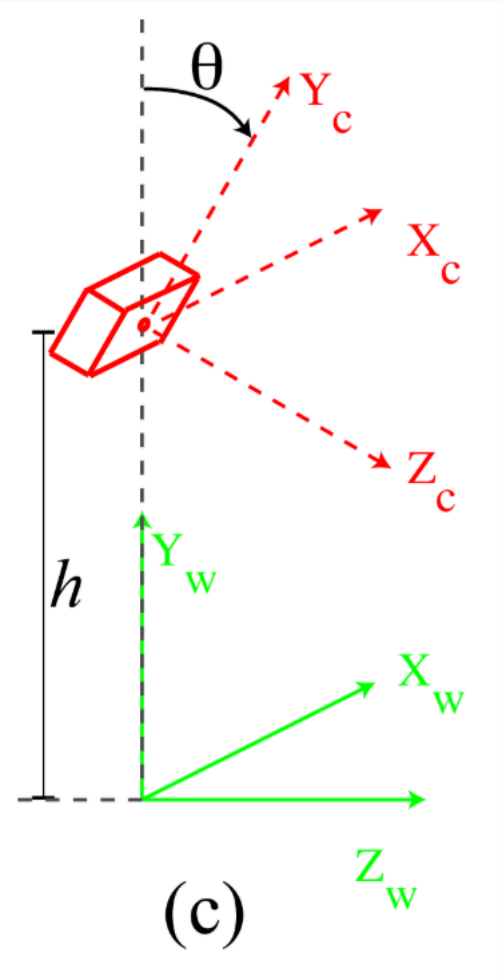


Figure 13: source: visionbook

Example (c)

in heterogeneous coordinates:

$$x = a\frac{X}{\sin(\theta)(h-Y)+\cos(\theta)Z}$$

$$y = a\frac{\cos(\theta)(Y-h)+\sin(\theta)Z}{\sin(\theta)(h-Y)+\cos(\theta)Z}$$

for arbitrary θ, we can compute the horizon line by letting $Z \to +\inf$

- $y \to a\tan(\theta)$

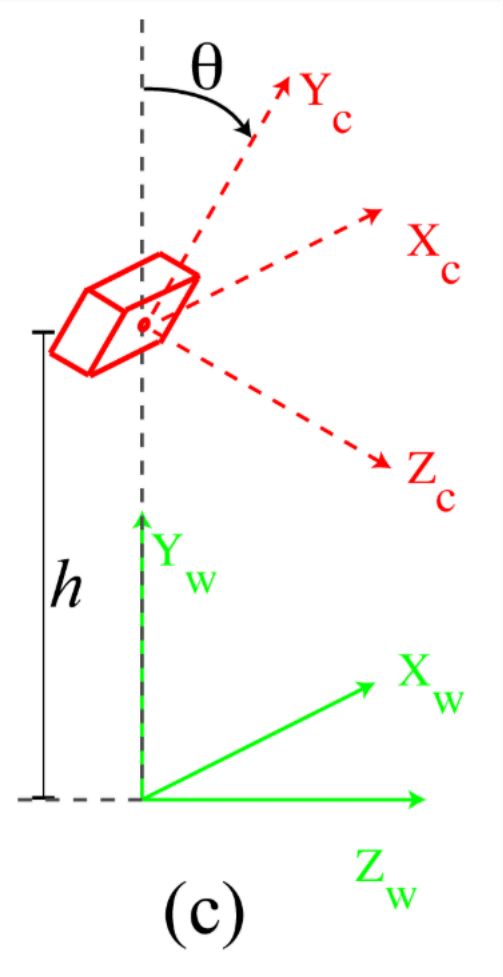


Figure 14: source: visionbook

Example (c)

for arbitrary θ, we can compute the horizon line by letting $Z \to +\inf$

- $y \to a\tan(\theta)$
- the eyes of the people with your same height will sit on the horizon line $a\tan(\theta)$ regardless of their distance

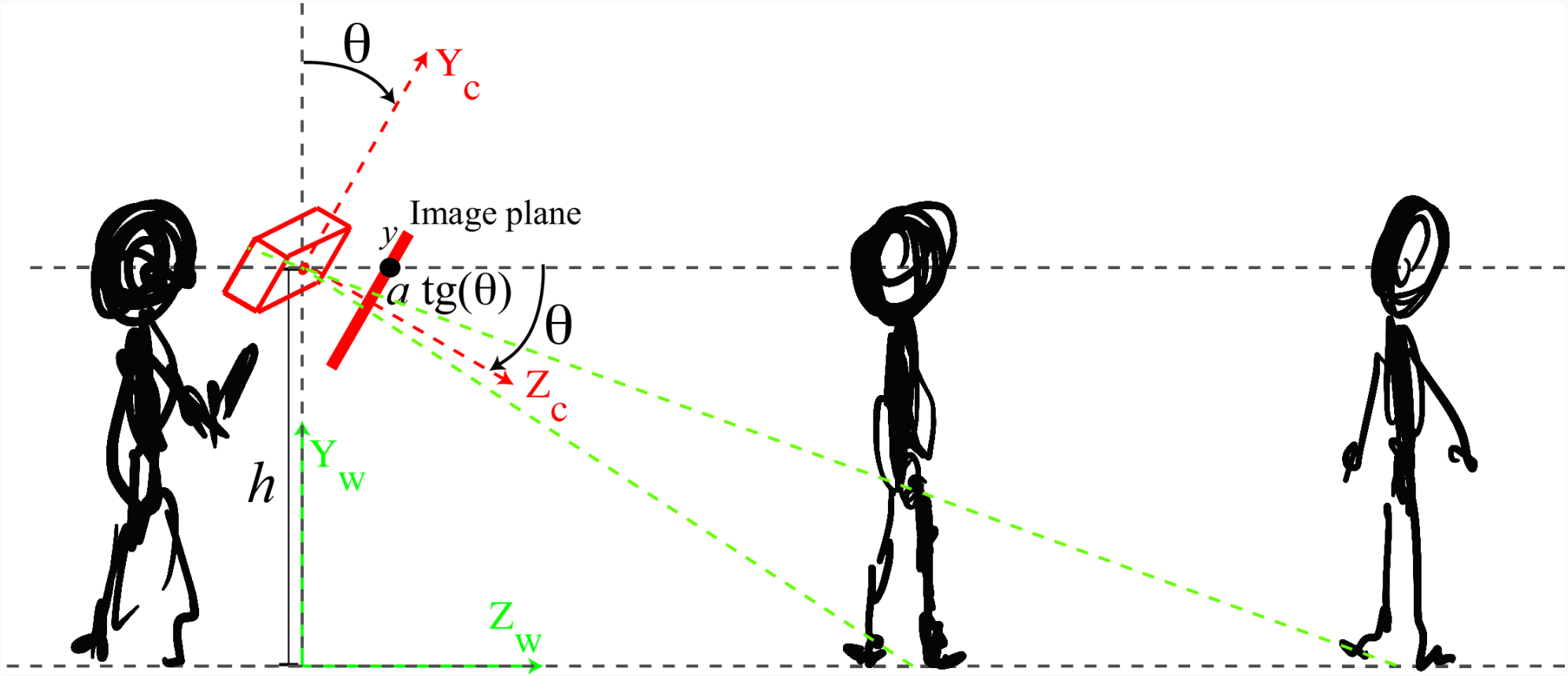


Figure 15: source: visionbook

Example (c)

- since you know the horizon line is $y \rightarrow a\tan(\theta)$
- given the horizon line and a (i.e. you calibrated your camera before), you can find θ
- given the horizon line and θ, you can find a



Figure 39.13: The two pictures taken with two different camera angles. The red line indicates the position of the horizon line estimated as the horizontal location in which the two vanishing lines (blue) intersect.

Figure 16: source: visionbook

Example (d)

Note that while the camera frame in (c) is the same of (d), the required rotation is different!
We have to rotate the **world coordinates (green frame)** into the **camera coordinates (red frame)**

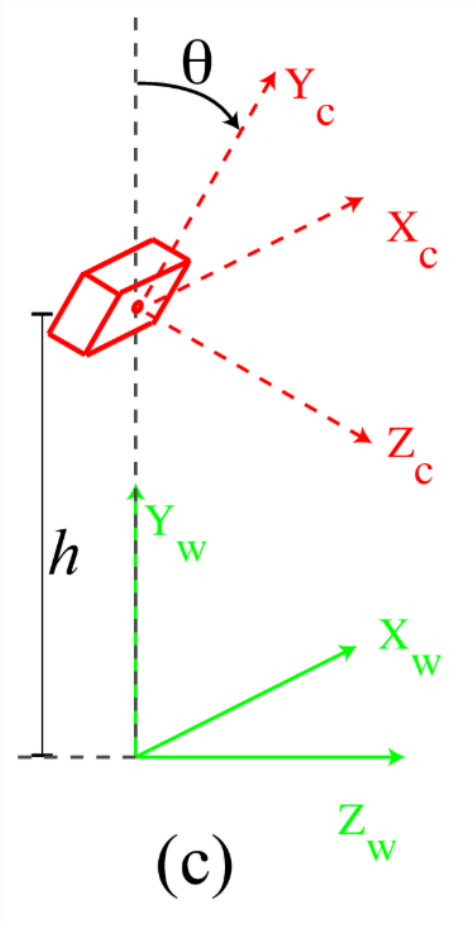


Figure 17: source: visionbook

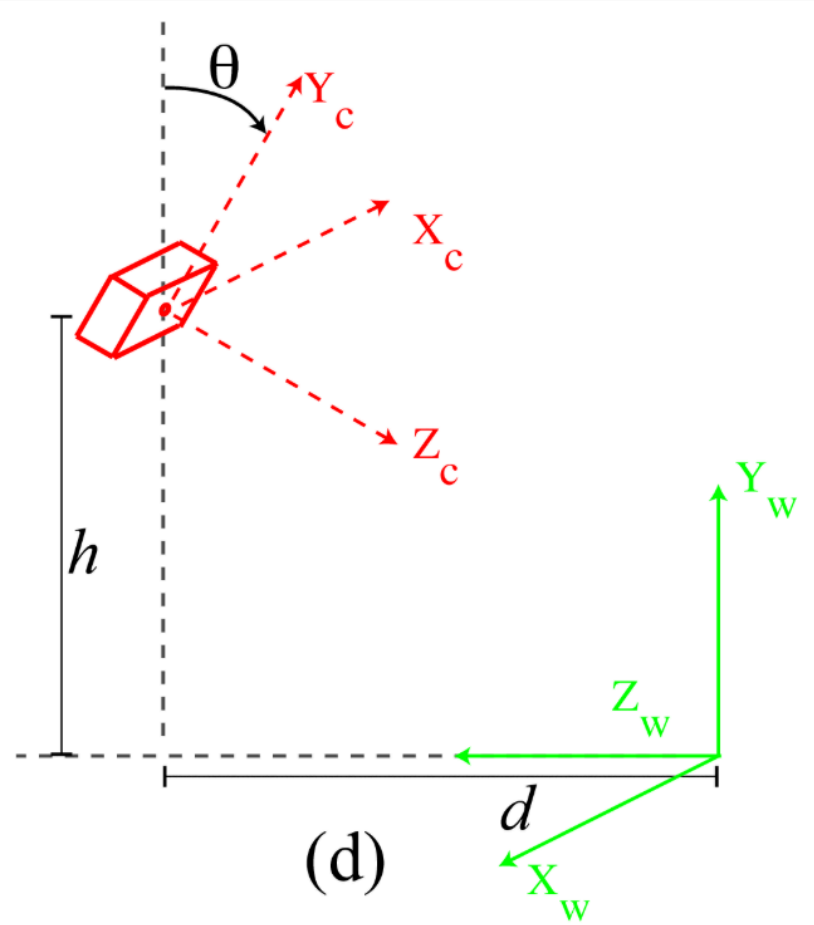


Figure 18: source: visionbook

Example (d)

Note that while the rotation in (c) looks the same as in (d), it's not! We have to rotate the **world coordinates (green frame)** into the **camera coordinates (red frame)**

- we rotate around the X_w axis by $\theta_x = \theta$
- we rotate around Y_w by $\theta_y = 180$ degrees

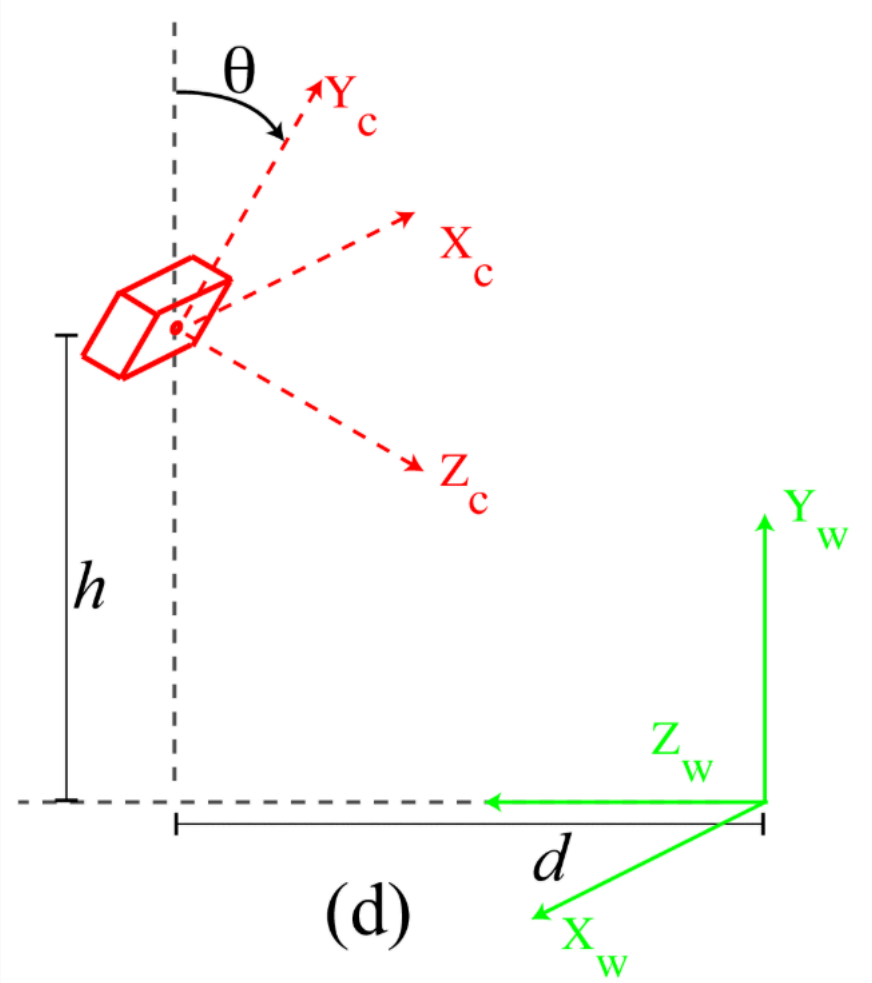


Figure 19: source: visionbook

Example (d)

- we rotate around Y_w by $\theta_y = 180$ degrees
- then, we rotate around the X_w axis by $\theta_x = \theta$

$$\boldsymbol{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & \sin(\theta_x) \\ 0 & -\sin(\theta_x) & \cos(\theta_x) \end{bmatrix} \begin{bmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) \\ 0 & 1 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) \\ 0 & -\sin(\theta_x) & -\cos(\theta_x) \end{bmatrix}$$

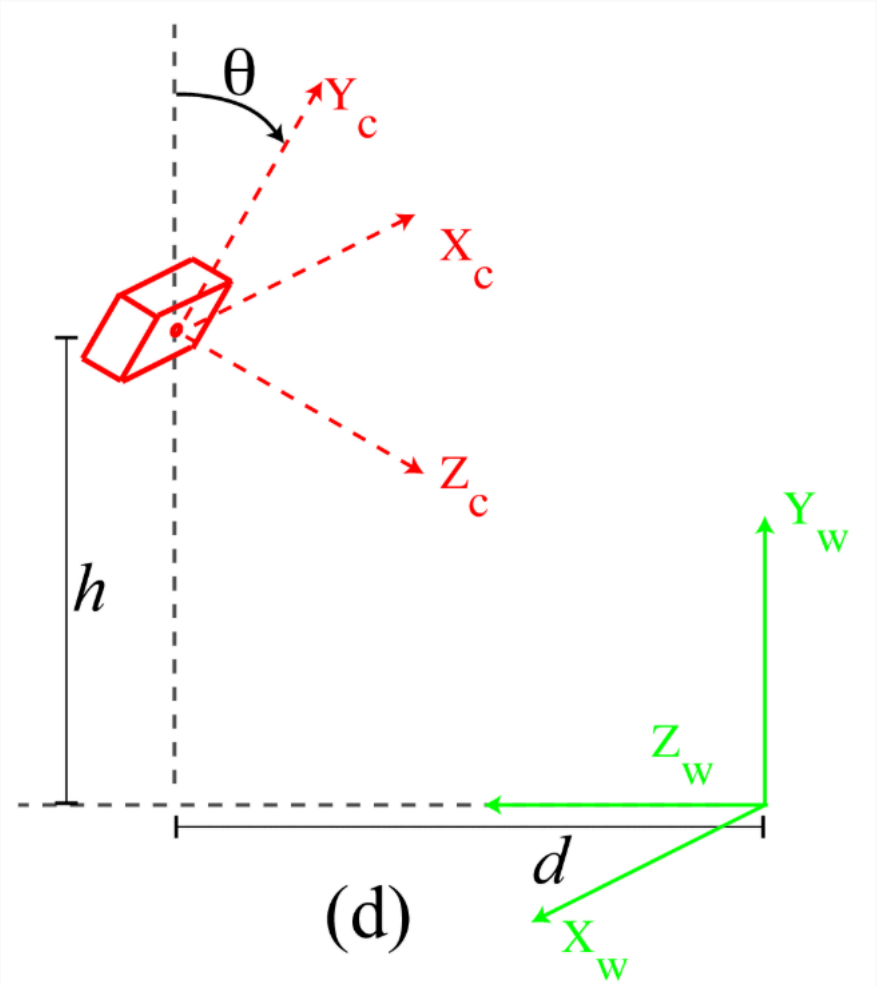


Figure 20: source: visionbook

remember: matrix multiplication does not commute. Rotations generally do not commute.

$\theta_y = \pi$

$$\boldsymbol{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & \sin(\theta_x) \\ 0 & -\sin(\theta_x) & \cos(\theta_x) \end{bmatrix}, \quad \boldsymbol{R}_y = \begin{bmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) \\ 0 & 1 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$\boldsymbol{R} = \boldsymbol{R}_x \boldsymbol{R}_y = \begin{bmatrix} -1 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) \\ 0 & -\sin(\theta_x) & -\cos(\theta_x) \end{bmatrix}$$

$$\boldsymbol{R} = \boldsymbol{R}_y \boldsymbol{R}_x = \begin{bmatrix} -1 & 0 & 0 \\ 0 & \cos(\theta_x) & \sin(\theta_x) \\ 0 & \sin(\theta_x) & -\cos(\theta_x) \end{bmatrix}$$

Example (d)

$$M = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) & 0 \\ 0 & -\sin(\theta_x) & -\cos(\theta_x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -h \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -a & 0 & 0 & 0 \\ 0 & a\cos(\theta) & -a\sin(\theta) & -ah\cos(\theta) + ad\sin(\theta) \\ 0 & -\sin(\theta) & -\cos(\theta) & h\sin(\theta) + d\cos(\theta) \end{bmatrix}$$

Camera Calibration

Camera Calibration

- a camera is **calibrated** if we know the world → pixel coordinate transformation
- **intrinsic**:
  - in theory, we could compute this from the technical specs of the camera
  - practically, it's quicker and more robust to estimate them (e.g. wear and post-processing can change the intrinsic parameters)
- **extrinsic**: we don't have full control of the scene and camera setup
- we have to estimate the parameters

Direct Linear Transform

DLT: Setup

note: it's the same method we used to estimate the homography H for image panoramas.

- we have a set of matches between 3D world coordinates and 2D pixel coordinates
- the transformation matrix $\boldsymbol{M}$ has 12 elements and 11 degrees of freedom
  - we lose one due to scale invariance
- $\boldsymbol{p}_i$ point in pixel coordinates
- $\boldsymbol{P}_i$ its corresponding point in world coordinates

Each correspondence gives us the following constraints on $\boldsymbol{M}$

$$\boldsymbol{p}_i = \boldsymbol{M}\boldsymbol{P}_i = \begin{bmatrix} \boldsymbol{m}_0^T \\ \boldsymbol{m}_1^T \\ \boldsymbol{m}_2^T \end{bmatrix} \boldsymbol{P}_i$$

$$\boldsymbol{p}_i = \boldsymbol{M}\boldsymbol{P}_i = \begin{bmatrix} \boldsymbol{m}_0^T \\ \boldsymbol{m}_1^T \\ \boldsymbol{m}_2^T \end{bmatrix} \boldsymbol{P}_i$$

element-wise in heterogeneous coordinates:

$$x_i = \frac{\boldsymbol{m}_0^T \boldsymbol{P}_i}{\boldsymbol{m}_2^T \boldsymbol{P}_i}$$

$$y_i = \frac{\boldsymbol{m}_1^T \boldsymbol{P}_i}{\boldsymbol{m}_2^T \boldsymbol{P}_i}$$

which gives the following linear constraints on $\boldsymbol{M}$:

$$\boldsymbol{P}_i^T \boldsymbol{m}_0 - x_i \boldsymbol{P}_i^T \boldsymbol{m}_2 = 0$$

$$\boldsymbol{P}_i^T \boldsymbol{m}_1 - y_i \boldsymbol{P}_i^T \boldsymbol{m}_2 = 0$$

linear constraints on M:

$$\boldsymbol{P}_i^T \boldsymbol{m}_0 - x_i \boldsymbol{P}_i^T \boldsymbol{m}_2 = 0$$

$$\boldsymbol{P}_i^T \boldsymbol{m}_1 - y_i \boldsymbol{P}_i^T \boldsymbol{m}_2 = 0$$

in matrix form:

$$\boldsymbol{A}\boldsymbol{m} = \begin{bmatrix} -\boldsymbol{P}_i^T & \boldsymbol{0} & x_i \boldsymbol{P}_i^T \\ \boldsymbol{0} & -\boldsymbol{P}_i^T & y_i \boldsymbol{P}_i^T \end{bmatrix} \begin{bmatrix} \boldsymbol{m}_0 \\ \boldsymbol{m}_1 \\ \boldsymbol{m}_2 \end{bmatrix} = \boldsymbol{0}$$

- $\boldsymbol{m}$ is the vector obtained by flattening $\boldsymbol{M}$
- we need to solve $\boldsymbol{A}\boldsymbol{m} = 0$
- $\boldsymbol{A}$ is a 2×12 matrix

each correspondence gives a 2×12 matrix:

$$\boldsymbol{A}\boldsymbol{m} = \begin{bmatrix} -\boldsymbol{P}_i^T & \mathbf{0} & x_i \boldsymbol{P}_i^T \\ \mathbf{0} & -\boldsymbol{P}_i^T & y_i \boldsymbol{P}_i^T \end{bmatrix} \begin{bmatrix} \boldsymbol{m}_0 \\ \boldsymbol{m}_1 \\ \boldsymbol{m}_2 \end{bmatrix} = \mathbf{0}$$

- since $\boldsymbol{M}$ has 11 degrees of freedom, we only need 6 correspondences
- in practice, due to noise, it's better to use more correspondences

DLT: Solution

After concatenating all the constraints for each correpondence in the matrix $\boldsymbol{A}$, we solve for $\boldsymbol{m}$ by minimizing

$\arg\min_m \|\boldsymbol{A}m\|$ s.t. $\|\boldsymbol{m}\| = 1$

- $\boldsymbol{m}$ is scale-invariant, so adding $\|\boldsymbol{m}\| = 1$ doesn't change the solution
  - if we don't add this constraint, we get $\boldsymbol{m} = 0$
- the solution is the minimum eigenvector of $\boldsymbol{A}^T\boldsymbol{A}$

Recovering Intrinsic and Extrinsic Camera Parameters

we have $M = \begin{bmatrix} KR & -KRT \end{bmatrix} = \begin{bmatrix} B & b \end{bmatrix}$

- B is 3×3, b is 3×1
- $T = B^{-1}b$
- KR can also be recovered by QR decomposition

Nonlinearities in the Calibration Objective

- We minimized the error in the **homogeneous coordinates**, but homogeneous coordinates are scale-invariant.
- each correspondence has a **different scale** in the constraints

homogeneous-to-hetererogeneous transformation:

$$\pi \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} \frac{u}{w} \\ \frac{v}{w} \end{pmatrix}$$

Notice the division by w

our true objective is in heterogeneous coordinates:

$$\min_{M} \sum_{i=1}^{N} \|p_i - \pi(MP_i)\|^2$$

Nonlinear Optimization by Minimizing Reprojection Error

reprojection error: difference between estimated image coordinates and it's ground-truth

$$\sum_{i=1}^{N} \|\boldsymbol{p}_i - \boldsymbol{p}_i'\|^2 = \sum_{i=1}^{N} \|\boldsymbol{p}_i - \pi(K[R \ -RT]\boldsymbol{P}_i)\|^2$$

- π is the homogeneous-to-heterogeneous transformation
- K, R, T are minimized by SGD
- initialized by DLT

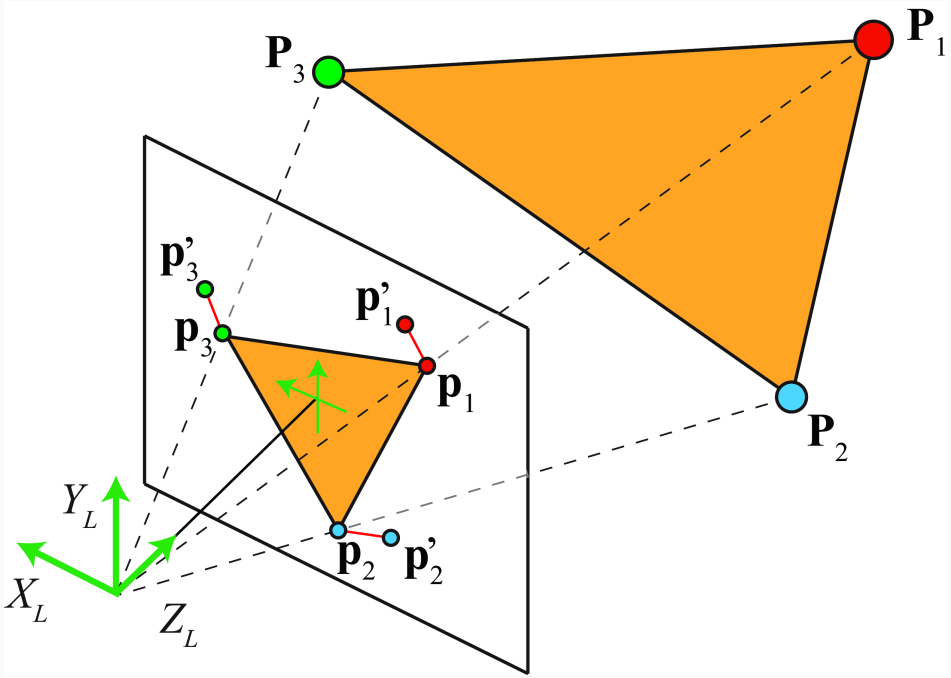


Figure 21: source: visionbook

Calibration Targets

Multiplane Calibration Method

- to apply DLT we need a set of images and correspondences between points
  - the set of images needs to be easy to take in any situation, quick, and robust when looking for the matches
- Zhang's method is one of the standard calibration methods
- standard planar calibration charts are used
- we take multiple pictures of the chart in different locations and orientations relative to a fixed camera

Advantages of Standardized Calibration Targets

- calibration needs **known** 3D world points $\boldsymbol{P}_i$ matched to their **observed** 2D image points $\boldsymbol{p}_i$
- in principle any object with precisely measured 3D points would work, but:
  - measuring 3D points by hand is slow and error-prone
  - we want **many** correspondences, spread across the image, with sub-pixel accuracy

solution: print a pattern with a known, regular geometry

- **world coordinates** are known by construction (we designed the pattern)
- **image coordinates** can be detected automatically and precisely

Checkerboard Patterns

- a checkerboard has $n \times m$ **internal corners**
- these corners are the calibration points: locally well-localized, with strong contrast in two directions
- corners are found as local maxima of a **corner response** (e.g. Harris), then ordered into a grid using the known checkerboard topology
- **world coordinates** of corner (r, c) are trivial: $(r \cdot s, c \cdot s, 0)$ for square size s

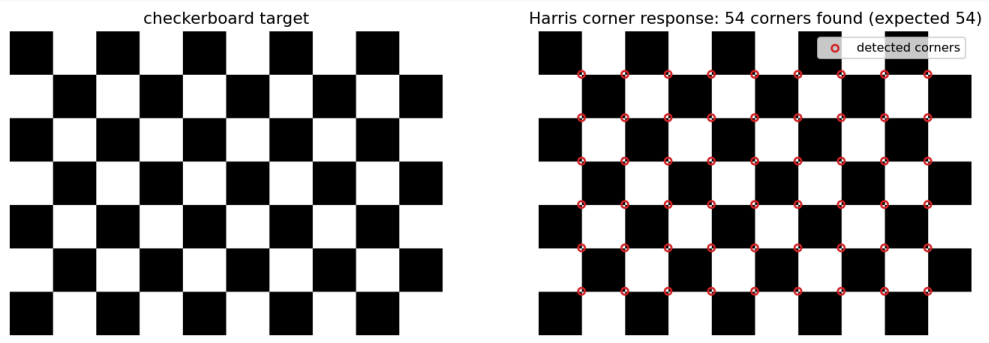


Figure 22: Checkerboard and corner detection.

Practical Considerations

- **rigidity**: the planarity assumption must hold. A warped or bent piece of paper will not work
- **size**: large enough that corners are detected accurately, small enough to fit fully in the image
- **coverage**: take images where the target covers
  - the center and the corners/edges of the image (distortion is strongest at the edges)
  - a range of distances and tilts

Convergence considerations

We use SGD, which is a **local** optimizer: it converges to the nearest stationary point

- a good closed-form initialization (Zhang's method) is helpful
- poor corner detection (outliers) can happen. Check visually that corner detection is working on every view that you are using
- degenerate view sets (e.g. all near-fronto-parallel) leave some parameters (e.g. principal point, or distortion) weakly constrained, so the optimizer may not move them meaningfully from their initial value

Camera Calibration with OpenCV

Dataset: a set of checkerboard images from one camera

- typical setup: a rigid, planar (internal corners) checkerboard, known square size
- capture 15-25 images, varying:
  - distance to the camera (near and far)
  - tilt in both directions (not just rotation around one axis)
  - position in the frame (center, corners, edges)
- keep the board **rigid** and **in focus**; avoid motion blur (use a tripod or short exposure)

Overview of the OpenCV calibration workflow

```
for each calibration image:
    findChessboardCorners  # detect corners
    cornerSubPix  # refine corners
    (object_points, image_points)  # collect

K, dist, rvecs, tvecs = calibrateCamera(object_points, image_points,
image_size)

for each new image:
    undistort(image, K, dist)
```

- `object_points`: the same known board coordinates $(X, Y, 0)$ for every view (board geometry is fixed)
- `image_points`: detected corner pixel locations, one set per calibration image
- everything from the previous sections (direct linear estimation + nonlinear refinement + distortion) happens inside `calibrateCamera`

`findChessboardCorners`: detecting calibration pattern corners

- `cv2.findChessboardCorners(image, pattern_size)` searches for the $n \times m$ grid of internal corners and returns them in a consistent row-major order (needed so the i-th detected corner matches the i-th `object_point`)
- returns a boolean (pattern found or not) and the corner locations if found; not all images succeed (occlusion, poor contrast, motion blur), discard those
- similar to Harris-based detection with extra logic to group corners into the correct grid topology and order

Discard images where the pattern was not found; spot-check a few detections (e.g. a wrong corner order is a also potential failure)

Sub-pixel corner refinement

- `findChessboardCorners` gives integer-pixel estimates; `cv2.cornerSubPix(image, corners, win_size, ...)` refines each corner to sub-pixel accuracy
- idea: around the initial guess, the true corner is where the image gradient is **orthogonal** to the vector from a candidate point to neighboring pixels (a corner is the point that locally fits this constraint best across a small window); this is solved with a few Gauss-Newton-like iterations
- without refinement integer-pixel corners would cap the calibration accuracy at roughly 0.5 px regardless of how many images you use

`calibrateCamera`: running the full calibration

```
ret, K, dist, rvecs, tvecs = cv2.calibrateCamera(
    object_points,  # list of (n_points, 3) arrays, one per view
    image_points,   # list of (n_points, 2) arrays, one per view
    image_size,     # (width, height) in pixels
)
```

- runs Zhang's closed-form initialization, then nonlinear refinement of intrinsic, distortion, and per-view (rvecs, tvecs) parameters
- `ret` is the overall RMS reprojection error (in pixels) of the final fit

Some Sanity Checks

some sanity checks:

- **image center**: u_0, v_0 close to image center (unless the lens is known to be strongly off-axis)
- **non-square-pixels**: $\alpha \approx \beta$ unless pixels are known to be non-square
- **lens distortion**: $\gamma \approx 0$ for most consumer cameras.

Evaluating calibration quality: reprojection errors

- per-image RMS reprojection error (not the global one returned by `calibrateCamera`): a good calibration has low and uniform error across all images
- high error (right panel) indicate a detection problem (e.g. motion blur, partial occlusion) or poor views (e.g. extreme angle)
  - remove them and recalibrate
- RMS reprojection error should be under 1 px (or even 0.5 px) with proper views: perfectly planar board, good corner detection, no poor view.



Good vs. Bad calibration

good calibration

- sub-pixel per-image RMS error (e.g. all $<$ 0.3-0.5 px)
- $\boldsymbol{K}$ and distortion stable when recomputed from a random subset of the images

poor calibration

- a few images with much higher error (possible bad detections or extreme views)
- u_0, v_0 far from the image center with no physical reason, or implausible focal length
- residual curvature near image edges after undistortion (distortion under-corrected)

Conclusions

Possible Mistakes in Calibration

- **poor image coverage**: if all calibration images avoid the corners/edges of the frame, distortion (strongest at the edge) is poorly estimated, even if the average reprojection error looks fine
- **planar degeneracy**: views that are all (near) fronto-parallel, or all at very similar tilts, leave $\boldsymbol{K}$ weakly constrained (recall: why multiple, **varied**, views are needed)
- **motion blur or poor corner detection**: blurry or low-contrast images give noisy/biased corner locations. Inspect detected corners visually, and check per-image reprojection error
- **overfitting distortion coefficients**: adding k_3 (or more exotic terms) when k_1, k_2 already fit well mostly fits noise and can make the model worse on images outside the calibration set; only add higher-order terms when the residual clearly needs them (e.g. a fisheye lens)
  - in general, most consumer cameras should have very small distortion effects

Take-Home Messages

- camera calibration estimates intrinsic parameters, distortion, and per-view $\boldsymbol{R}, \boldsymbol{t}$ from images of a known planar target
- the standard pipeline has two steps:
  - **DLT**: error in homogeneous coordinates ignoring distortion
  - **nonlinear refinement**: error in heterogeneous coordinates. Includes distortion
- **sanity checks**: per-image reprojection error and visual inspection of undistorted images
- good calibration data (varied tilts, distances, and image coverage) is fundamental

Further Reading

- visionbook Ch. 39
- OpenCV documentation: Camera Calibration and 3D Reconstruction “https://docs.opencv.org/4.13.0/d9/d0c/group__calib3d.html”

Next Lecture

- Stereo vision and depth estimation