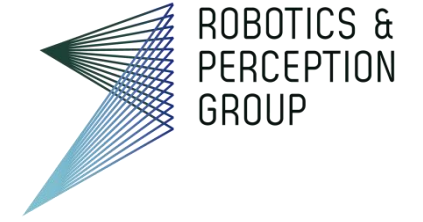




**University of  
Zurich** <sup>UZH</sup>



# Vision Algorithms for Mobile Robotics

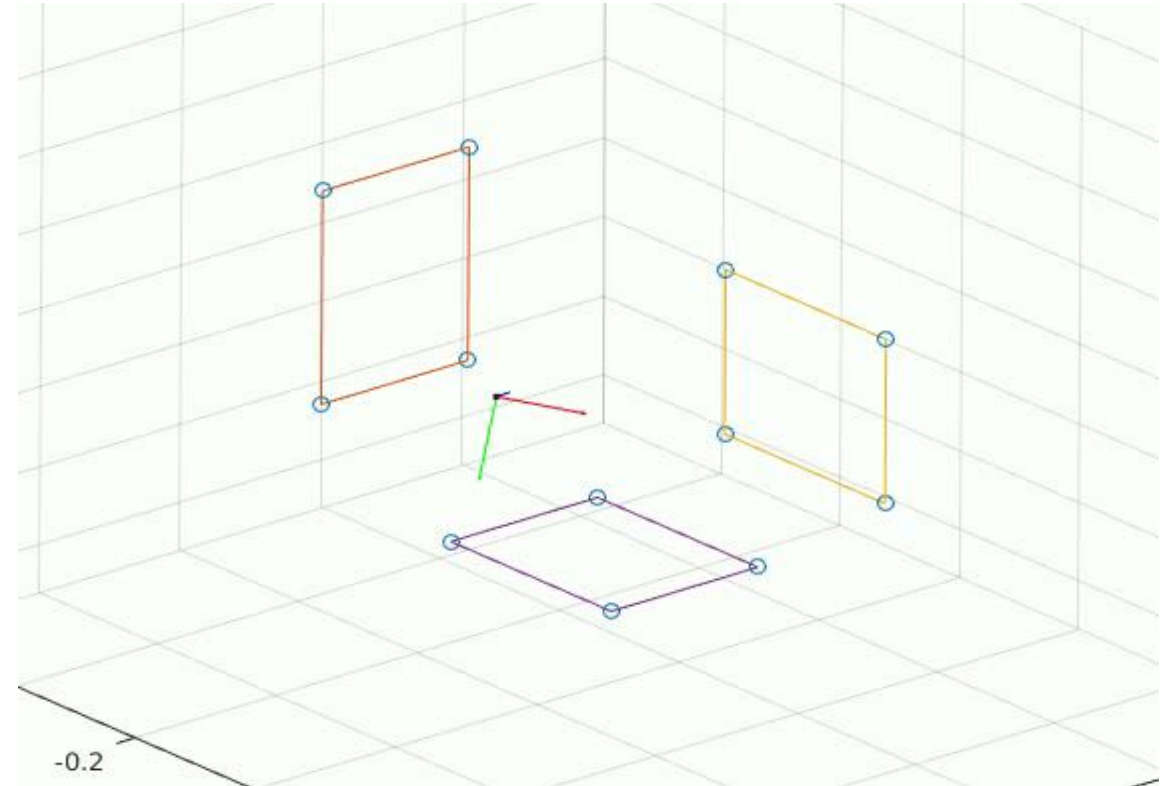
## Lecture 03 Camera Calibration

Davide Scaramuzza

<https://rpg.ifi.uzh.ch>

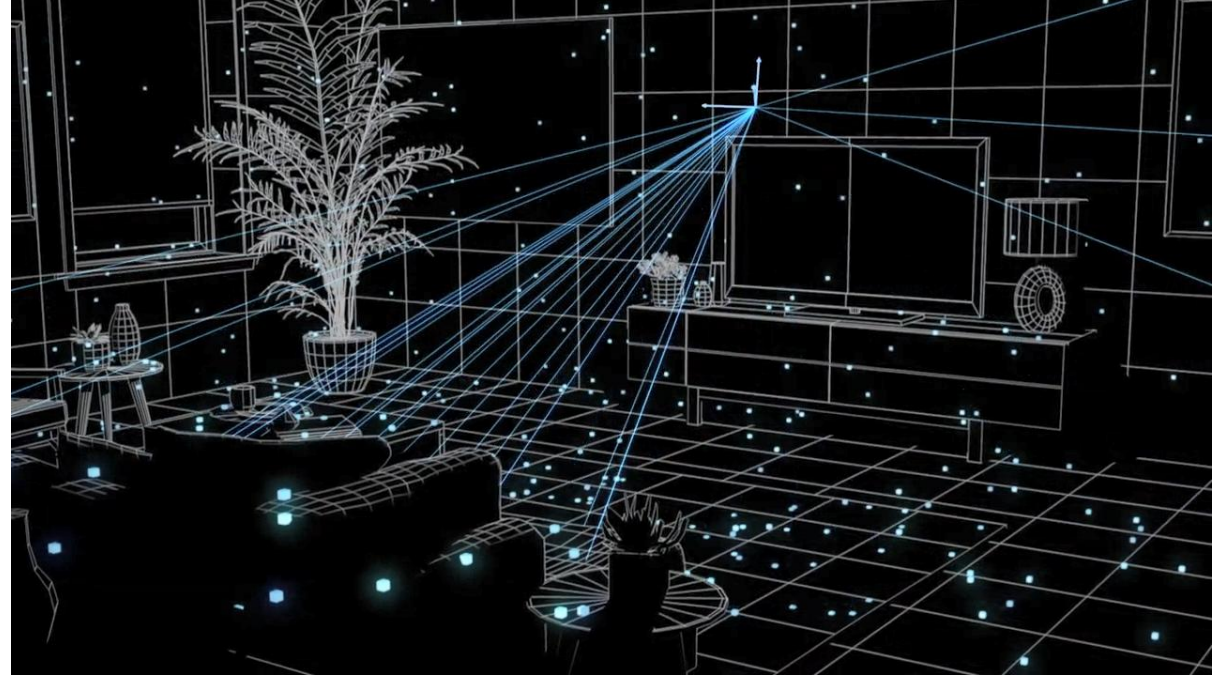
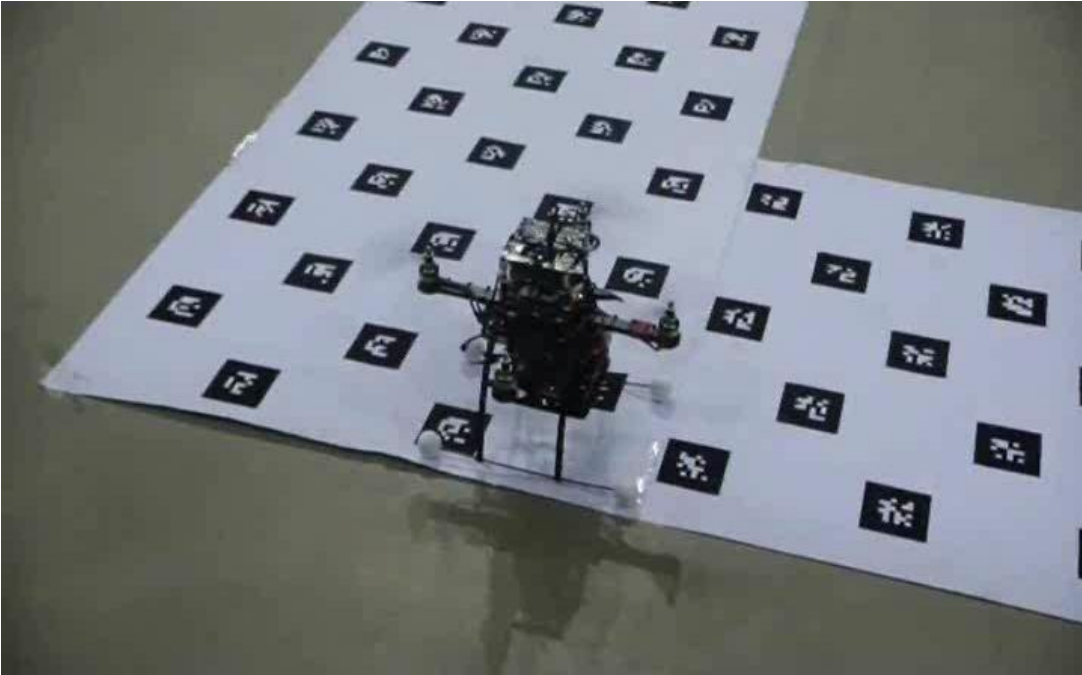
# Lab Exercise 2 – Today

Implement your first camera motion estimator using the DLT algorithm



# Goal of today's lecture

- Learn how to calibrate a camera
- Study the foundational algorithms for camera localization



Two applications of the camera localization algorithms covered in this lecture: drone navigation & Meta Quest

# Today's Outline

- Camera calibration
- Camera localization
- Non conventional camera models: fisheye and catadioptric cameras

# Camera Calibration

- Calibration is the process to determine the **intrinsic parameters** ( $K$  plus lens distortion) **and extrinsic parameters** ( $R, T$ ) of a camera. For now, we will **neglect the lens distortion** and see later how it can be determined.
- $K, R, T$  can be **determined by applying the perspective projection** equation to known 3D-2D point correspondences:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R & | & T \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

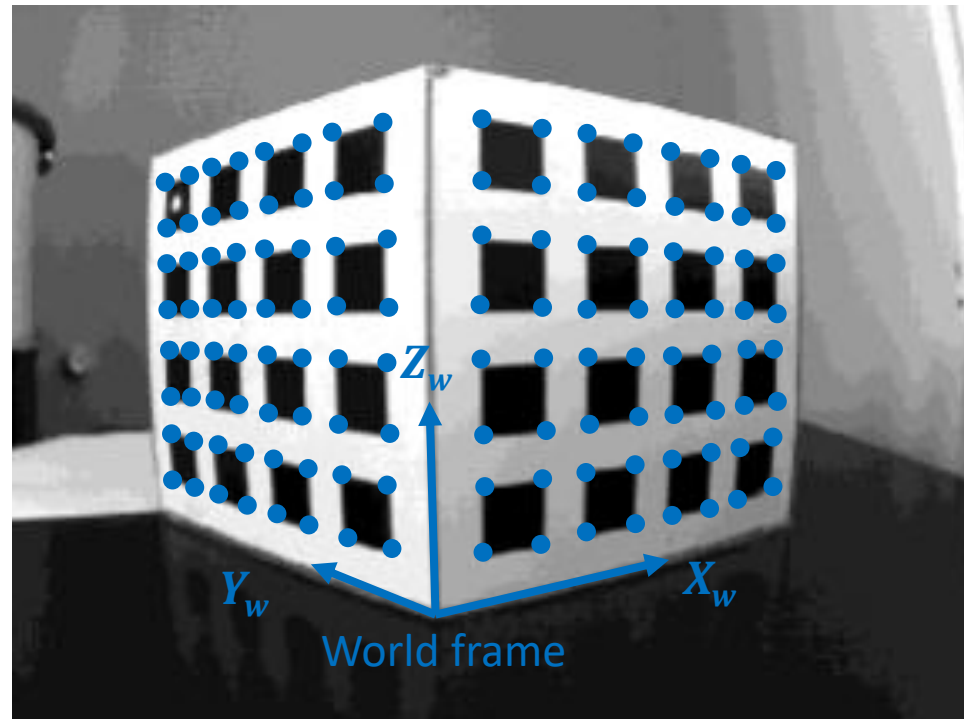
- There are two popular methods:
  - **Tsai's method**: uses 3D objects
  - **Zhang's method**: uses planar grids

# Today's Outline

- Camera calibration
  - Tsai's method: From 3D objects
  - Zhang's method: from planar grids
- Camera localization
- Non conventional camera models: fisheye and catadioptric cameras

# Tsai's Method: Calibration from 3D Objects

- This method was proposed in 1987 by Tsai and consists of measuring the 3D position of  $n \geq 6$  **control points** on a 3D calibration target and the **2D coordinates of their projection** in the image.



Tsai, Roger Y. (1987) "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses," *IEEE Journal of Robotics and Automation*, 1987. [PDF](#).

# Applying the Direct Linear Transform (DLT) algorithm

The idea of the DLT is to rewrite the perspective projection equation as a **homogeneous linear equation** and solve it by standard methods. Let's write the perspective equation for a generic 3D-2D point correspondence:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R | T] \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \Rightarrow$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_u r_{11} + u_0 r_{31} & \alpha_u r_{12} + u_0 r_{32} & \alpha_u r_{13} + u_0 r_{33} & \alpha_u t_1 + u_0 t_3 \\ \alpha_v r_{21} + v_0 r_{31} & \alpha_v r_{22} + v_0 r_{32} & \alpha_v r_{23} + v_0 r_{33} & \alpha_v t_2 + v_0 t_3 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

# Applying the Direct Linear Transform (DLT) algorithm

The idea of the DLT is to rewrite the perspective projection equation as a **homogeneous linear equation** and solve it by standard methods. Let's write the perspective equation for a generic 3D-2D point correspondence:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R | T] \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \Rightarrow$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

What are the assumptions  
behind this this  
substitution?

# Applying the Direct Linear Transform (DLT) algorithm

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = M \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

where  $m_i^T$  is the  $i$ -th row of  $M$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_1^T \\ m_2^T \\ m_3^T \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

# Applying the Direct Linear Transform (DLT) algorithm

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_1^T \\ m_2^T \\ m_3^T \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \rightarrow P$$

Conversion back from homogeneous coordinates to pixel coordinates leads to:

$$\begin{aligned} u &= \frac{\lambda u}{\lambda} = \frac{m_1^T \cdot P}{m_3^T \cdot P} \\ v &= \frac{\lambda v}{\lambda} = \frac{m_2^T \cdot P}{m_3^T \cdot P} \end{aligned} \Rightarrow \begin{aligned} (m_1^T - u_i m_3^T) \cdot P &= 0 \\ (m_2^T - v_i m_3^T) \cdot P &= 0 \end{aligned}$$

# Applying the Direct Linear Transform (DLT) algorithm

- By re-arranging the terms, we obtain

$$\begin{aligned} (m_1^T - u_i m_3^T) \cdot P &= 0 \\ (m_2^T - v_i m_3^T) \cdot P &= 0 \end{aligned} \Rightarrow \begin{pmatrix} P_1^T & 0^T & -u_1 P_1^T \\ 0^T & P_1^T & -v_1 P_1^T \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ m_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

- For  $n$  points, we can stack all these equations into a big matrix:

$$\begin{pmatrix} P_1^T & 0^T & -u_1 P_1^T \\ 0^T & P_1^T & -v_1 P_1^T \\ & & \vdots \\ P_n^T & 0^T & -u_n P_n^T \\ 0^T & P_n^T & -v_n P_n^T \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ m_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

# Applying the Direct Linear Transform (DLT) algorithm

$$\underbrace{\begin{pmatrix} X_w^1 & Y_w^1 & Z_w^1 & 1 & 0 & 0 & 0 & 0 & -u_1 X_w^1 & -u_1 Y_w^1 & -u_1 Z_w^1 & -u_1 \\ 0 & 0 & 0 & 0 & X_w^1 & Y_w^1 & Z_w^1 & 1 & -v_1 X_w^1 & -v_1 Y_w^1 & -v_1 Z_w^1 & -v_1 \\ & & & & & & & \vdots & & & & \\ X_w^n & Y_w^n & Z_w^n & 1 & 0 & 0 & 0 & 0 & -u_n X_w^n & -u_n Y_w^n & -u_n Z_w^n & -u_n \\ 0 & 0 & 0 & 0 & X_w^n & Y_w^n & Z_w^n & 1 & -v_n X_w^n & -v_n Y_w^n & -v_n Z_w^n & -v_n \end{pmatrix}}_{\text{Q (this matrix is \textbf{known})}} \underbrace{\begin{pmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{pmatrix}}_{\text{M (this matrix is \textbf{unknown})}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \Rightarrow \mathbf{Q} \cdot \mathbf{M} = \mathbf{0}$$

# Applying the Direct Linear Transform (DLT) algorithm

$$Q \cdot M = 0$$

## Minimal solution

- $Q_{(2n \times 12)}$  should have rank 11 to have a unique (up to a scale) *non-zero* solution  $M$
- Because each 3D-to-2D point correspondence provides 2 independent equations, then  $5 + \frac{1}{2}$  point correspondences are needed (in practice **6 point** correspondences!)

## Over-determined solution

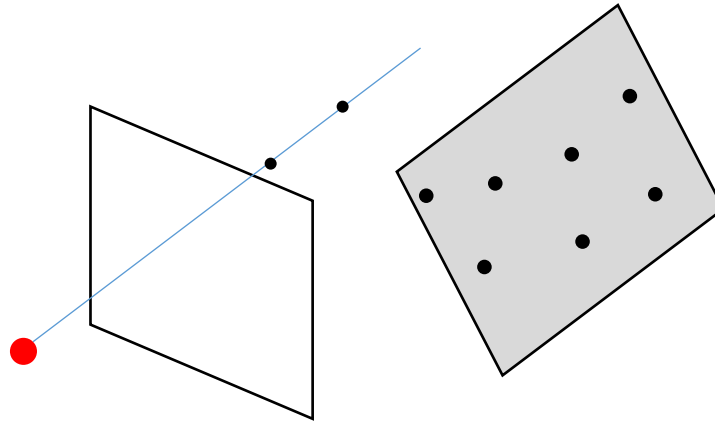
- For  $n \geq 6$  points, a solution is the **Least Square solution**, which minimizes the sum of squared residuals,  $\|QM\|^2$ , subject to the constraint  $\|M\|^2 = 1$ . It can be solved through Singular Value Decomposition (SVD). The solution is the eigenvector corresponding to the smallest eigenvalue of the matrix  $Q^T Q$  (because it is the unit vector  $x$  that minimizes  $\|Qx\|^2 = x^T Q^T Q x$ ).
- Matlab instructions:
  - `[U, S, V] = SVD(Q);`
  - `M = V(:, 12);`

# Applying the Direct Linear Transform (DLT) algorithm

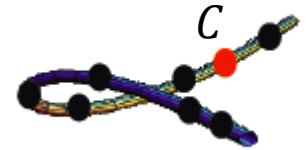
## Degenerate configurations

$$Q \cdot M = 0$$

1. Points lying on a **plane** and/or along a single **line** passing through the **center of projection**



2. Camera and points on a **twisted cubic** (i.e., smooth curve in 3D space of degree 3)



# Applying the Direct Linear Transform (DLT) algorithm

- Once we have determined  $M$ , we can recover the intrinsic and extrinsic parameters by remembering that:

$$M = K(R \mid T)$$

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}$$

# Applying the Direct Linear Transform (DLT) algorithm

- Once we have determined  $M$ , we can recover the intrinsic and extrinsic parameters by remembering that:

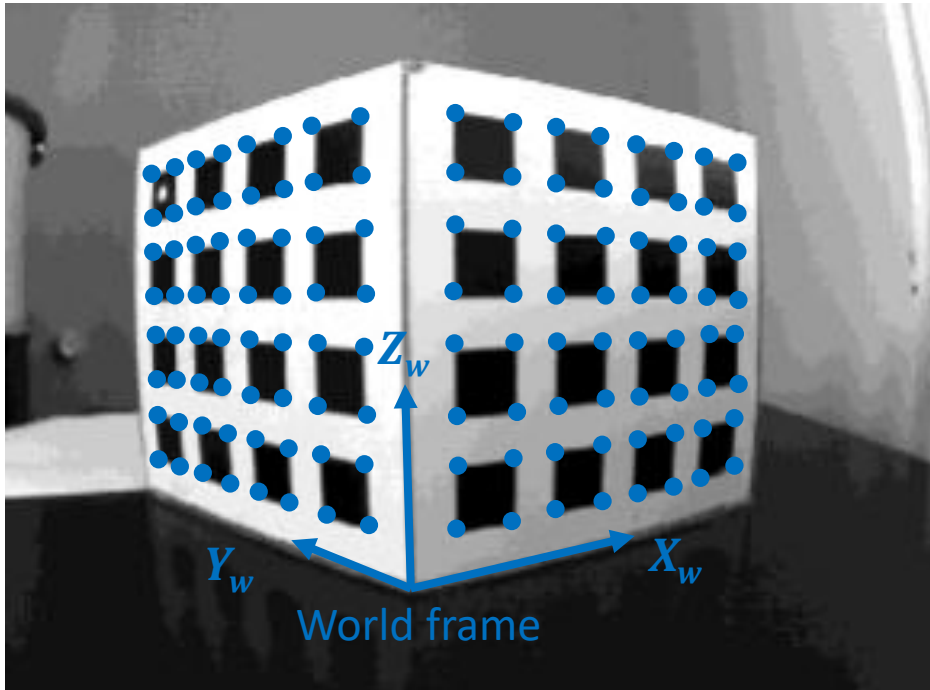
$$M = K(R \mid T)$$

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} = \begin{bmatrix} \alpha r_{11} + u_0 r_{31} & \alpha r_{12} + u_0 r_{32} & \alpha r_{13} + u_0 r_{33} & \alpha t_1 + u_0 t_3 \\ \alpha r_{21} + v_0 r_{31} & \alpha r_{22} + v_0 r_{32} & \alpha r_{23} + v_0 r_{33} & \alpha t_2 + v_0 t_3 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}$$

- However, notice that we are not enforcing the constraint that  $R$  is **orthogonal**, i.e.,  $R \cdot R^T = I$
- To do this, we can use the so-called **QR factorization of  $M$** , which decomposes  $M$  into a  $R$  (orthogonal),  $T$ , and an upper triangular matrix (i.e.,  $K$ )
- What if  $K$  is known (calibrated camera)?

# Example of Tsai's Calibration Results

**Recommendation:** use many more than 6 points (ideally more than 20) and non coplanar



Corners can be detected with accuracy  $< 0.1$  pixels  
(see Lecture 5)

| $\alpha_u$ | $\alpha_u/\alpha_v$ | $K_{12}$ | $u_0$  | $v_0$  | Average Reprojection error |
|------------|---------------------|----------|--------|--------|----------------------------|
| 1673.3     | 1.0063              | 1.39     | 379.96 | 305.78 | 0.365                      |

Why is this ratio not 1?

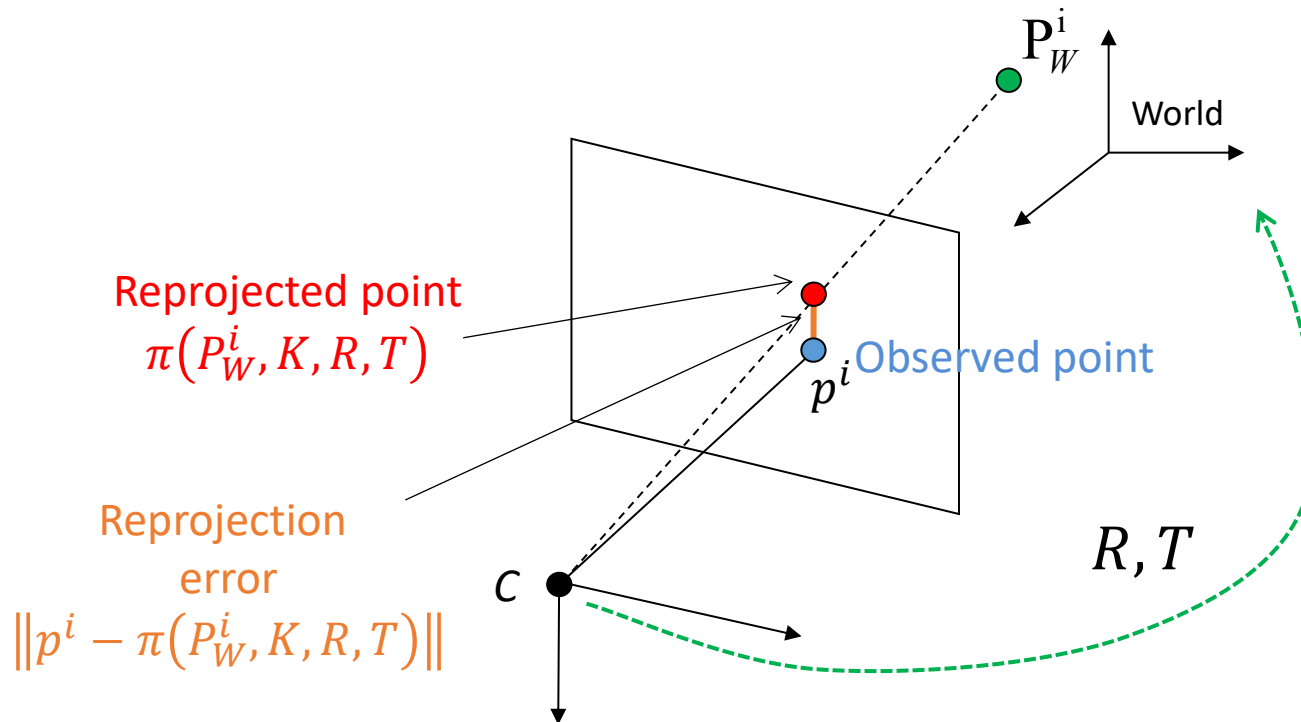
What is this?

What is this?

How can we estimate the lens distortion parameters?  
How can we enforce  $\alpha_u = \alpha_v$  and  $K_{12} = 0$ ?

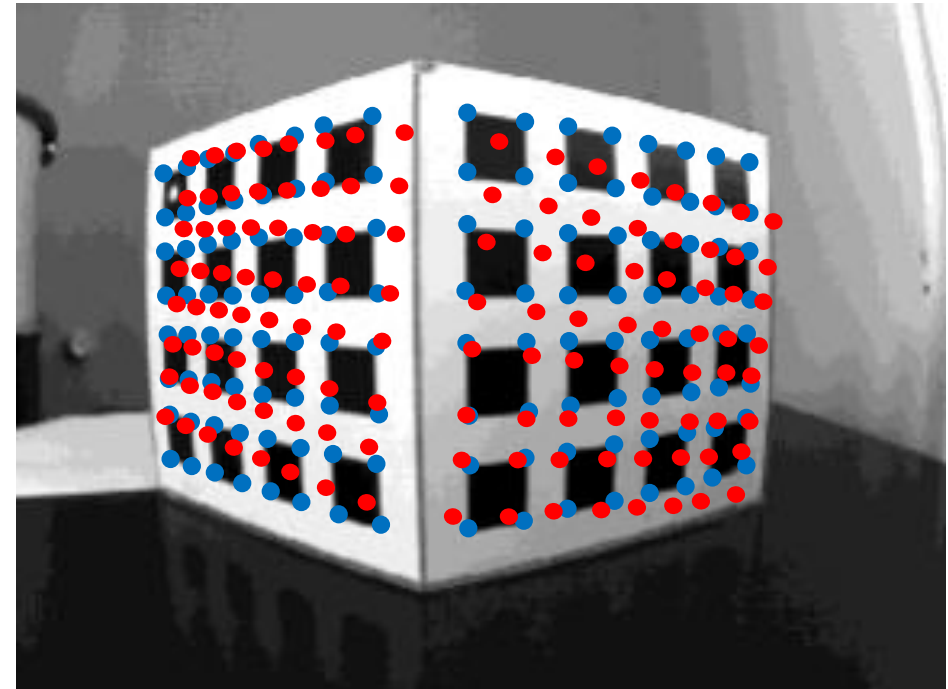
# Reprojection Error

- The reprojection error is the **Euclidean distance** (in pixels) between an **observed image point** and the **corresponding 3D point reprojected** onto the camera frame **according to the estimated K, R, T**.
- The reprojection error gives us a **quantitative measure of the accuracy** of the calibration (**ideally it should be zero**).



# Reprojection Error

- The reprojection error can be used to **assess the quality of the camera calibration**
- What reprojection error is acceptable?
- What are the sources of the reprojection error?
- Can the reprojection error ever be perfectly 0?
- How can we further improve the calibration parameters?



● Control points  
(observed points)

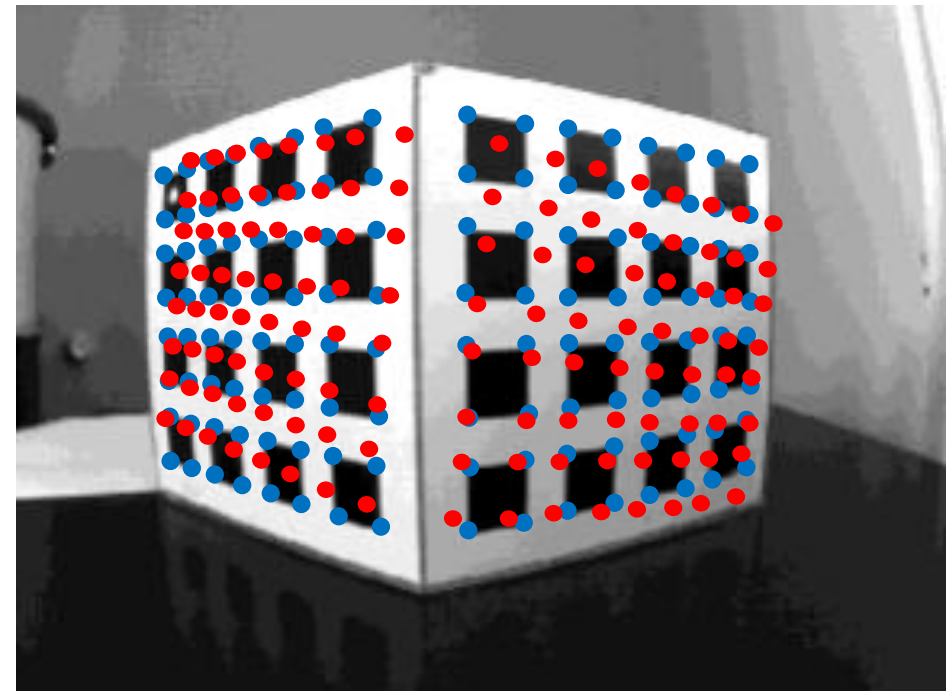
● Reprojected points  
 $\pi(P_W^i, K, R, T)$

# Non-Linear Calibration Refinement

- The calibration parameters  $K, R, T$  determined by the DLT can be refined by minimizing the following cost:

$$K, R, T, \text{ lens distortion} = \underset{K, k_1, R, T}{\operatorname{argmin}} \sum_{i=1}^n \|p^i - \pi(P_W^i, K, k_1, R, T)\|^2$$

- This time we also include the **lens distortion**  $k_1$  parameter (can be set to 0 for initialization)
- Can be minimized using **Levenberg–Marquardt** (more robust than Gauss-Newton to local minima)



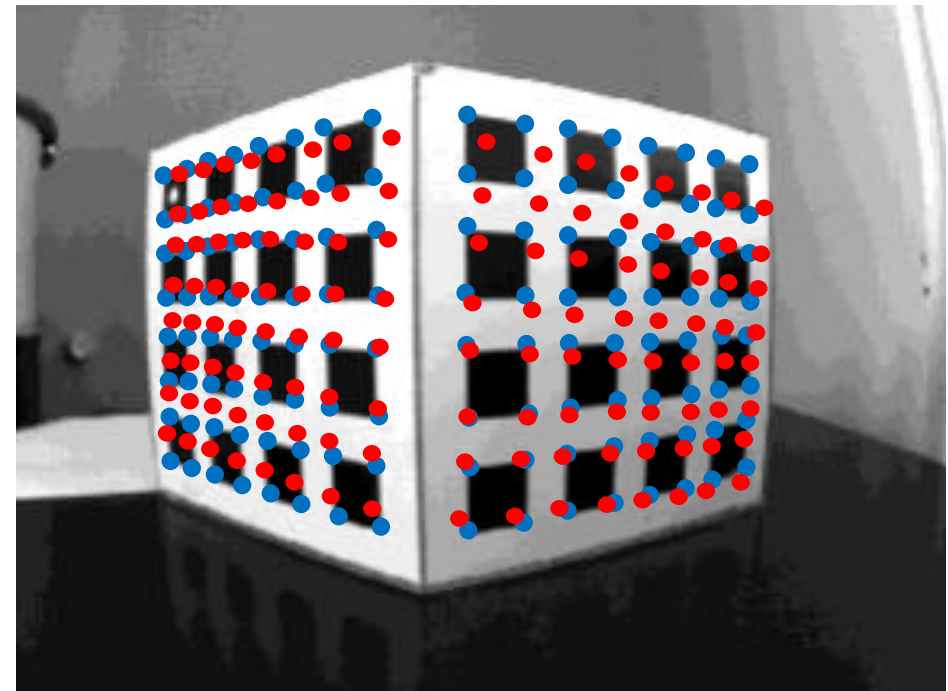
- Control points  
(observed points)
- Reprojected points  
 $\pi(P_W^i, K, R, T)$

# Non-Linear Calibration Refinement

- The calibration parameters  $K, R, T$  determined by the DLT can be refined by minimizing the following cost:

$$K, R, T, \text{ lens distortion} = \underset{K, k_1, R, T}{\operatorname{argmin}} \sum_{i=1}^n \|p^i - \pi(P_W^i, K, k_1, R, T)\|^2$$

- This time we also include the **lens distortion**  $k_1$  parameter (can be set to 0 for initialization)
- Can be minimized using **Levenberg–Marquardt** (more robust than Gauss-Newton to local minima)



● Control points  
(observed points)

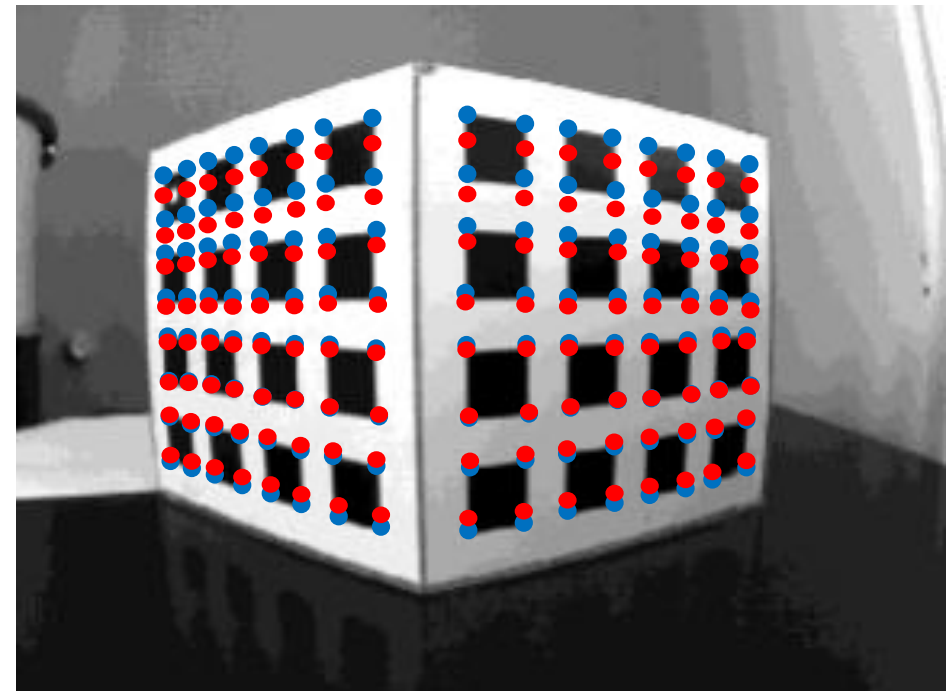
● Reprojected points  
 $\pi(P_W^i, K, R, T)$

# Non-Linear Calibration Refinement

- The calibration parameters  $K, R, T$  determined by the DLT can be refined by minimizing the following cost:

$$K, R, T, \text{ lens distortion} = \underset{K, k_1, R, T}{\operatorname{argmin}} \sum_{i=1}^n \|p^i - \pi(P_W^i, K, k_1, R, T)\|^2$$

- This time we also include the **lens distortion**  $k_1$  parameter (can be set to 0 for initialization)
- Can be minimized using **Levenberg–Marquardt** (more robust than Gauss-Newton to local minima)



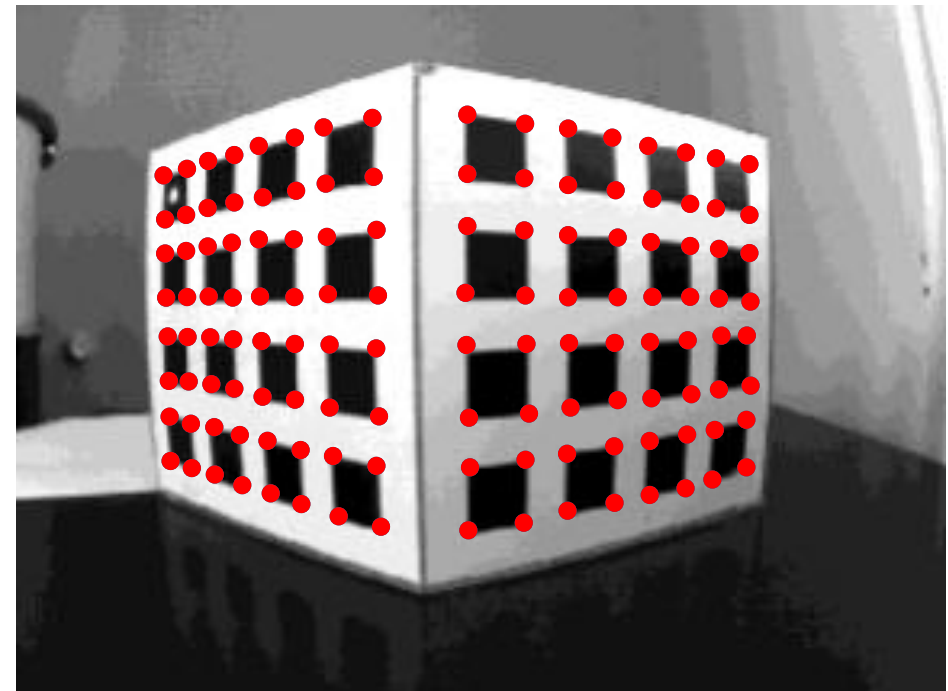
- Control points  
(observed points)
- Reprojected points  
 $\pi(P_W^i, K, R, T)$

# Non-Linear Calibration Refinement

- The calibration parameters  $K, R, T$  determined by the DLT can be refined by minimizing the following cost:

$$K, R, T, \text{ lens distortion} = \underset{K, k_1, R, T}{\operatorname{argmin}} \sum_{i=1}^n \|p^i - \pi(P_W^i, K, k_1, R, T)\|^2$$

- This time we also include the **lens distortion**  $k_1$  parameter (can be set to 0 for initialization)
- Can be minimized using **Levenberg–Marquardt** (more robust than Gauss-Newton to local minima)



● Control points  
(observed points)

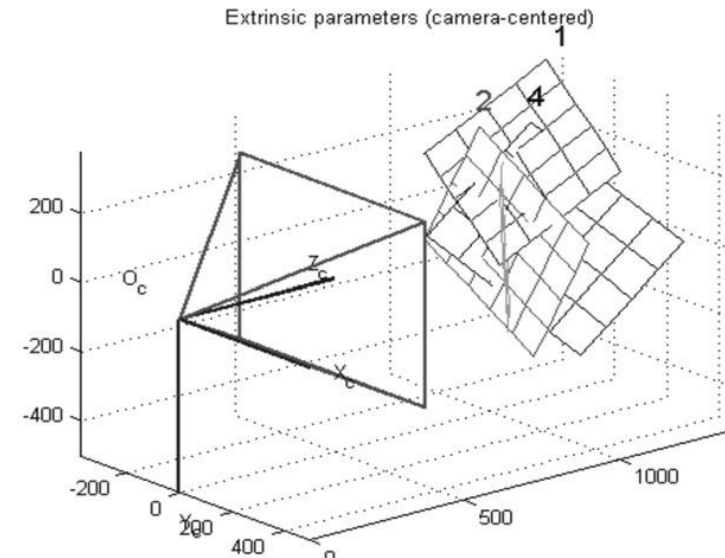
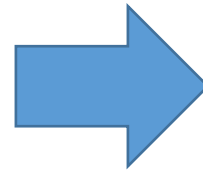
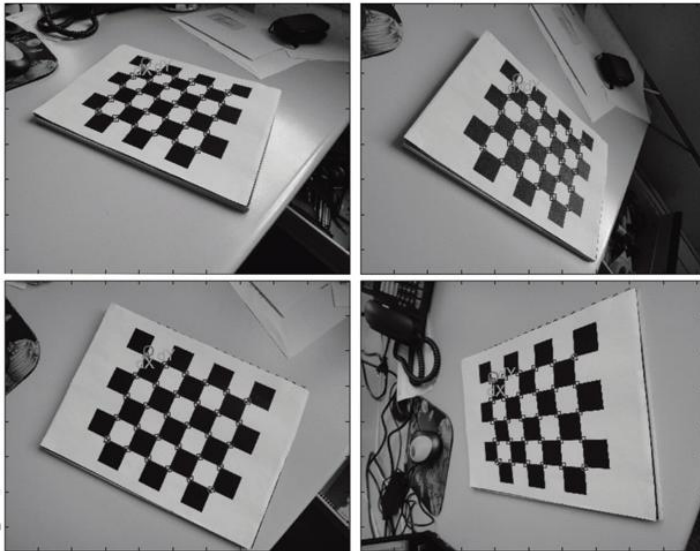
● Reprojected points  
 $\pi(P_W^i, K, R, T)$

# Today's Outline

- Camera calibration
  - Tsai's method: From 3D objects
  - Zhang's method: from planar grids
- Camera localization
- Non conventional camera models: fisheye and catadioptric cameras

# Zhang's Algorithm: Calibration from Planar Grids

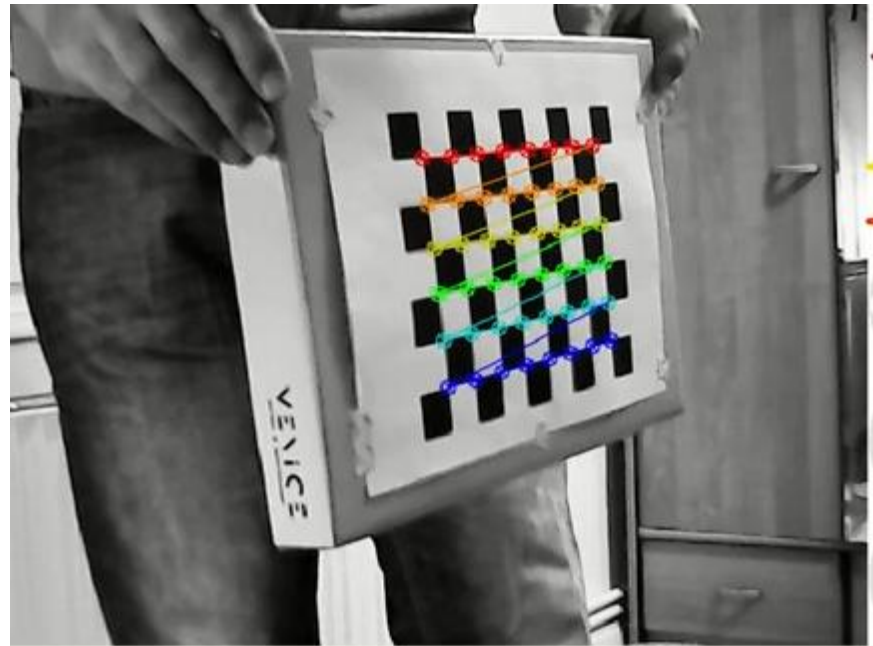
- **Tsai's calibration** requires that the world's 3D points are non-coplanar, which is **not very practical**
- **Today's camera calibration toolboxes** ([Matlab](#), [OpenCV](#)) use **multiple views** of a **planar grid** (e.g., a checker board)
- They are based on a method developed in 2000 by Zhang (Microsoft Research)



Zhang, A flexible new technique for camera calibration, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000. [PDF](#).

# Zhang's Algorithm: Calibration from Planar Grids

- **Tsai's calibration** requires that the world's 3D points are non-coplanar, which is **not very practical**
- **Today's camera calibration toolboxes** ([Matlab](#), [OpenCV](#)) use **multiple views** of a **planar grid** (e.g., a checker board)
- They are based on a method developed in 2000 by Zhang (Microsoft Research)



Zhang, A flexible new technique for camera calibration, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000. [PDF](#).

# Applying the Direct Linear Transform (DLT) algorithm

As in Tsai's method, we start by writing the perspective projection equation (again, we neglect the radial distortion). However, in **Zhang's method the points are all coplanar**, i.e.,  $\mathbf{Z}_w = \mathbf{0}$ , and thus we can write:

$$\begin{aligned} \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= K \begin{bmatrix} R & | & T \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 0 \\ 1 \end{bmatrix} \Rightarrow \\ \Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 0 \\ 1 \end{bmatrix} \\ \Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} \end{aligned}$$

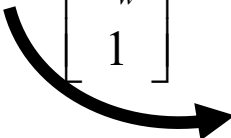
# Applying the Direct Linear Transform (DLT) algorithm

As in Tsai's method, we start by writing the perspective projection equation (again, we neglect the radial distortion). However, in **Zhang's method the points are all coplanar**, i.e.,  $\mathbf{Z}_w = \mathbf{0}$ , and thus we can write:

$$\begin{aligned} \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= K \begin{bmatrix} R & | & T \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 0 \\ 1 \end{bmatrix} \Rightarrow \\ \Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 0 \\ 1 \end{bmatrix} \\ \Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} \end{aligned}$$

# Applying the Direct Linear Transform (DLT) algorithm

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = H \cdot \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$


This matrix is called Homography

where  $h_i^T$  is the  $i$ -th row of  $H$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} h_1^T \\ h_2^T \\ h_3^T \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

# Applying the Direct Linear Transform (DLT) algorithm

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} h_1^T \\ h_2^T \\ h_3^T \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} \rightarrow P$$

Conversion back from homogeneous coordinates to pixel coordinates leads to:

$$\begin{aligned} u &= \frac{\lambda u}{\lambda} = \frac{h_1^T \cdot P}{h_3^T \cdot P} \\ v &= \frac{\lambda v}{\lambda} = \frac{h_2^T \cdot P}{h_3^T \cdot P} \end{aligned} \Rightarrow \begin{aligned} (h_1^T - u_i h_3^T) \cdot P_i &= 0 \\ (h_2^T - v_i h_3^T) \cdot P_i &= 0 \end{aligned}$$

# Applying the Direct Linear Transform (DLT) algorithm

- By re-arranging the terms, we obtain:

$$\begin{aligned} (h_1^T - u_i h_3^T) \cdot P_i &= 0 \\ (h_2^T - v_i h_3^T) \cdot P_i &= 0 \end{aligned} \Rightarrow \begin{aligned} P_i^T \cdot h_1 + 0 \cdot h_2^T - u_i P_i^T \cdot h_3^T &= 0 \\ 0 \cdot h_1^T + P_i^T \cdot h_2 - v_i P_i^T \cdot h_3^T &= 0 \end{aligned} \Rightarrow \begin{pmatrix} P_i^T & 0^T & -u_i P_i^T \\ 0^T & P_i^T & -v_i P_i^T \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

- For  $n$  points (from a **single view**), we can stack all these equations into a big matrix:

$$\underbrace{\begin{pmatrix} P_1^T & 0^T & -u_1 P_1^T \\ 0^T & P_1^T & -v_1 P_1^T \\ \dots & \dots & \dots \\ P_n^T & 0^T & -u_n P_n^T \\ 0^T & P_n^T & -v_n P_n^T \end{pmatrix}}_Q \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} \Rightarrow Q \cdot H = 0$$

Q (this matrix is **known**) H (this matrix is **unknown**)

# Applying the Direct Linear Transform (DLT) algorithm

$$Q \cdot H = 0$$

## Minimal solution

- $Q_{(2n \times 9)}$  should have rank 8 to have a unique (up to a scale) non-trivial solution  $H$
- Each point correspondence provides 2 independent equations
- Thus, a minimum of **4 non-collinear points** is required (otherwise it is a degenerate configuration)

## Solution for $n \geq 4$ points

- It can be solved through Singular Value Decomposition (SVD) (same considerations as before)

# How to recover $K, R, T$

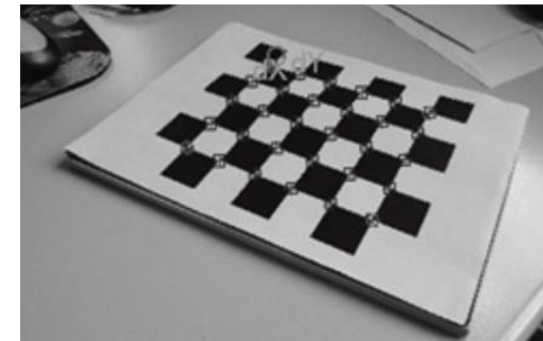
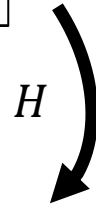
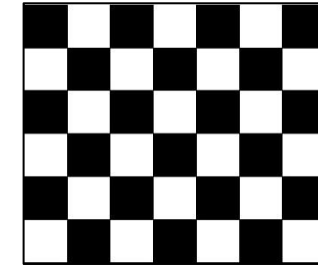
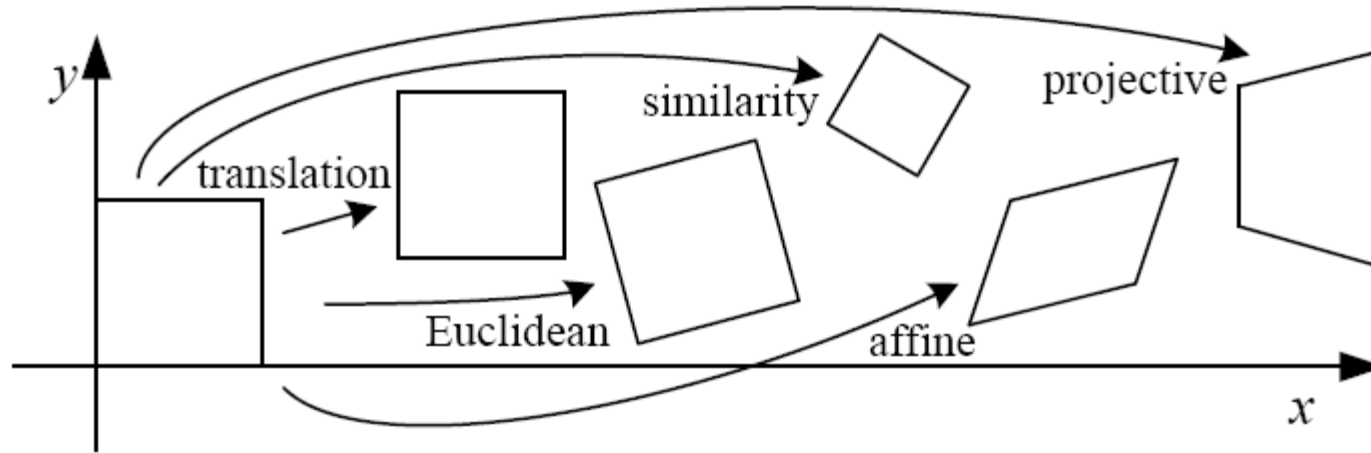
- $H$  can be decomposed by recalling that:
- Differently from Tsai's, the decomposition of  $H$  into  $K, R, T$  requires **at least two views if we assume**  $\alpha_u \neq \alpha_v$ , or 1 view if  $\alpha_u = \alpha_v$

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix}$$

- In practice **the more views the better**, e.g., 20-50 views **spanning the entire field of view** of the camera for the **best calibration results!**
- Notice that now **each view  $j$  has a different homography  $H^j$**  (and so a different  $R^j$  and  $T^j$ ). However,  **$K$  is the same for all views:**

$$\begin{bmatrix} h_{11}^j & h_{12}^j & h_{13}^j \\ h_{21}^j & h_{22}^j & h_{23}^j \\ h_{31}^j & h_{32}^j & h_{33}^j \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11}^j & r_{12}^j & t_1^j \\ r_{21}^j & r_{22}^j & t_2^j \\ r_{31}^j & r_{32}^j & t_3^j \end{bmatrix}$$

# Types of 2D Transformations



| Name              | Matrix                                                 | # D.O.F. | Preserves:        | Icon |
|-------------------|--------------------------------------------------------|----------|-------------------|------|
| translation       | $\begin{bmatrix} I & t \end{bmatrix}_{2 \times 3}$     | 2        | orientation + ... |      |
| rigid (Euclidean) | $\begin{bmatrix} R & t \end{bmatrix}_{2 \times 3}$     | 3        | lengths + ...     |      |
| similarity        | $\begin{bmatrix} sR & t \end{bmatrix}_{2 \times 3}$    | 4        | angles + ...      |      |
| affine            | $\begin{bmatrix} A \end{bmatrix}_{2 \times 3}$         | 6        | parallelism + ... |      |
| projective        | $\begin{bmatrix} \tilde{H} \end{bmatrix}_{3 \times 3}$ | 8        | straight lines    |      |

This matrix is called **Homography**

# Projective Transformation (Homography)

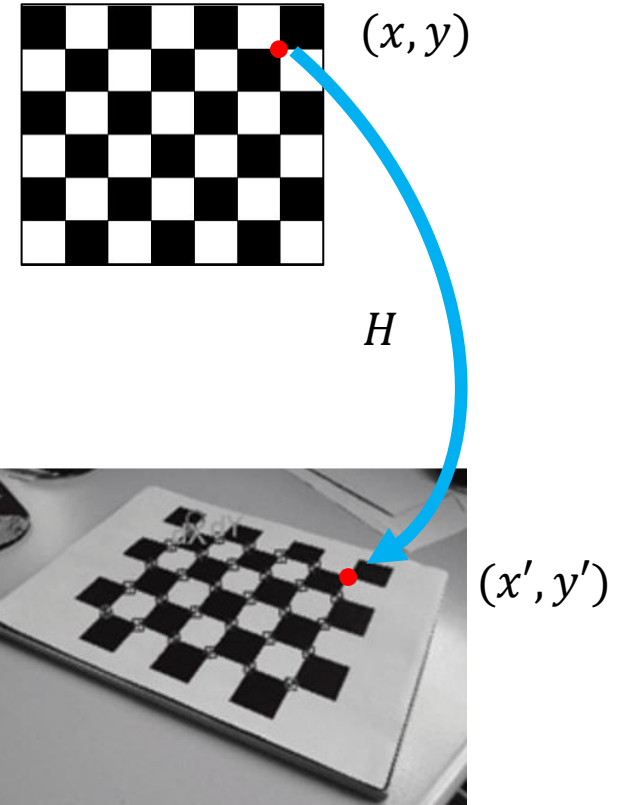
- A point  $(x, y)$  is transformed into  $(x', y')$  via:

$$x' = \frac{a_1x + a_2y + a_3}{a_7x + a_8y + 1}$$

$$y' = \frac{a_4x + a_5y + a_6}{a_7x + a_8y + 1}$$

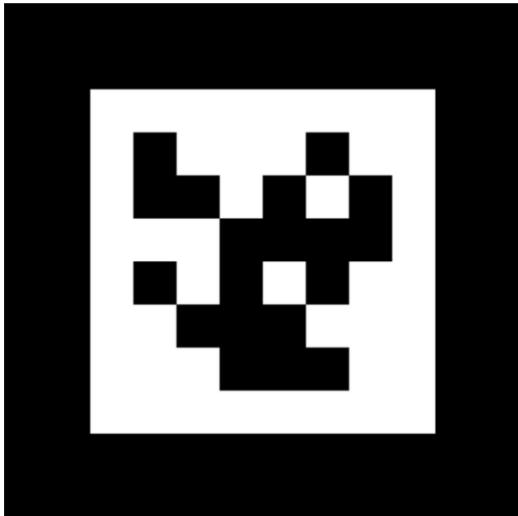
- Homogeneous coordinates:

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



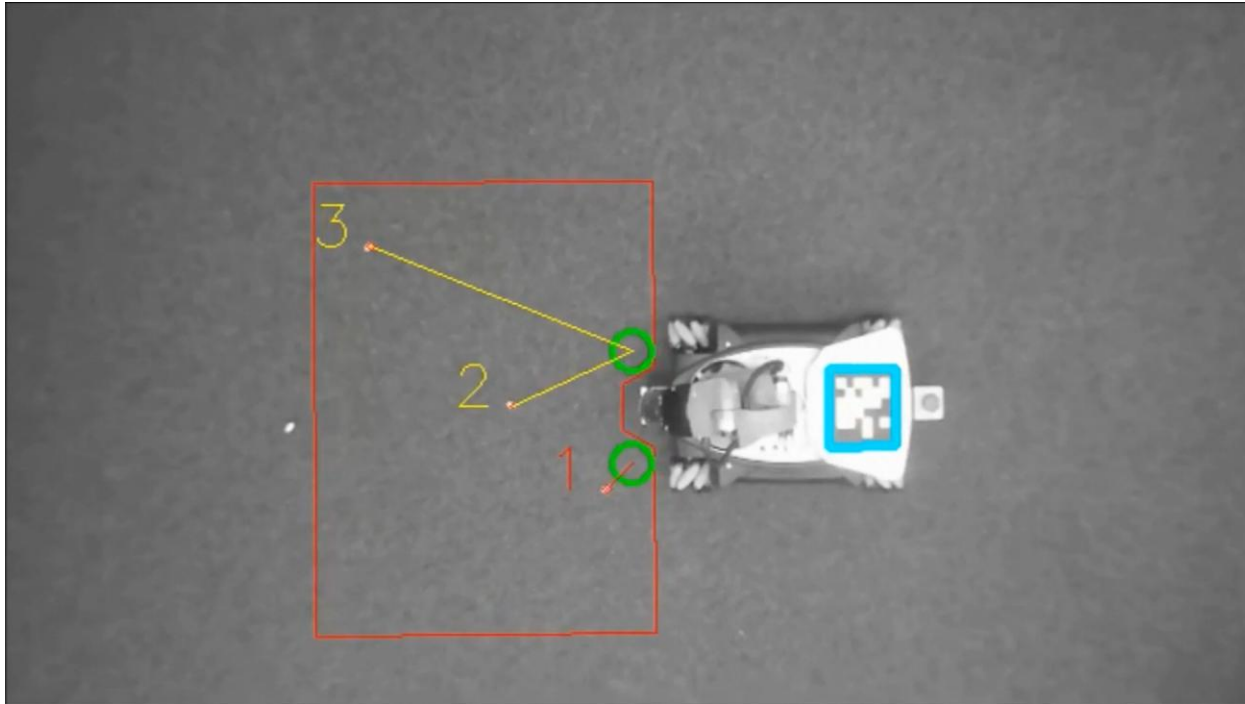
# Application to Augmented Reality

- Today, there are thousands of application of Zhang's algorithm, e.g. Augmented Reality (AR)
- See [AprilTag](#) or [ARuco Markers](#)

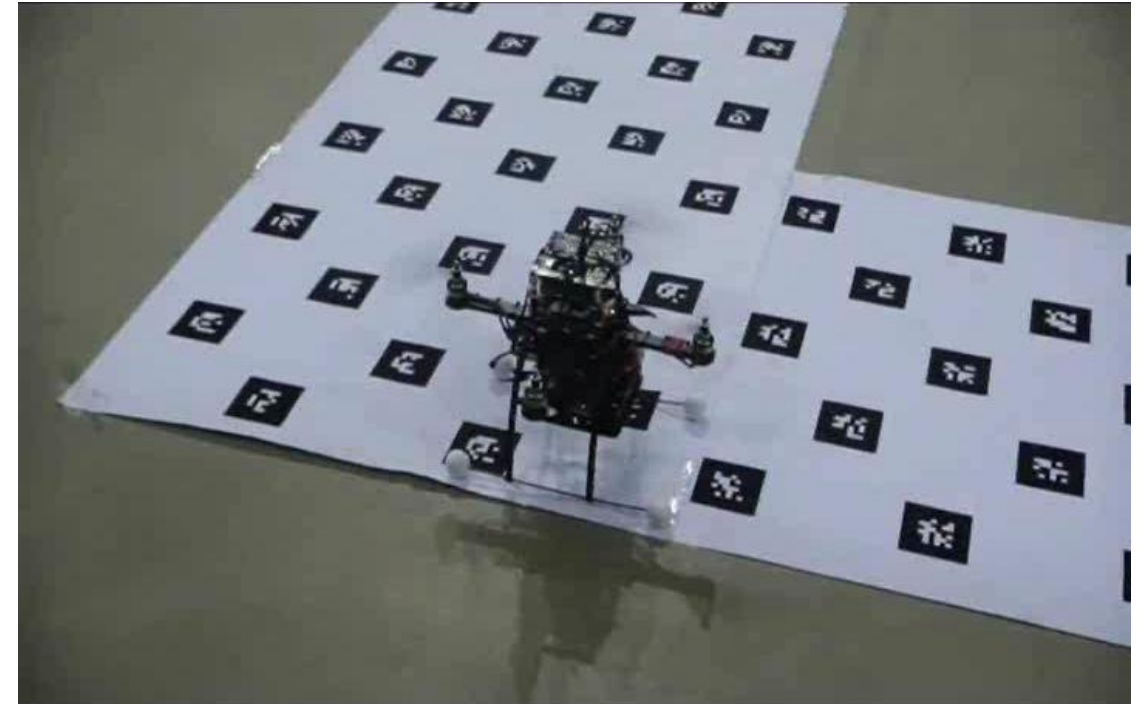


# Application to Robotics

- Do we need to know the size of the tag?
  - For Augmented Reality?
  - For Control?



My lab. [Video](#).



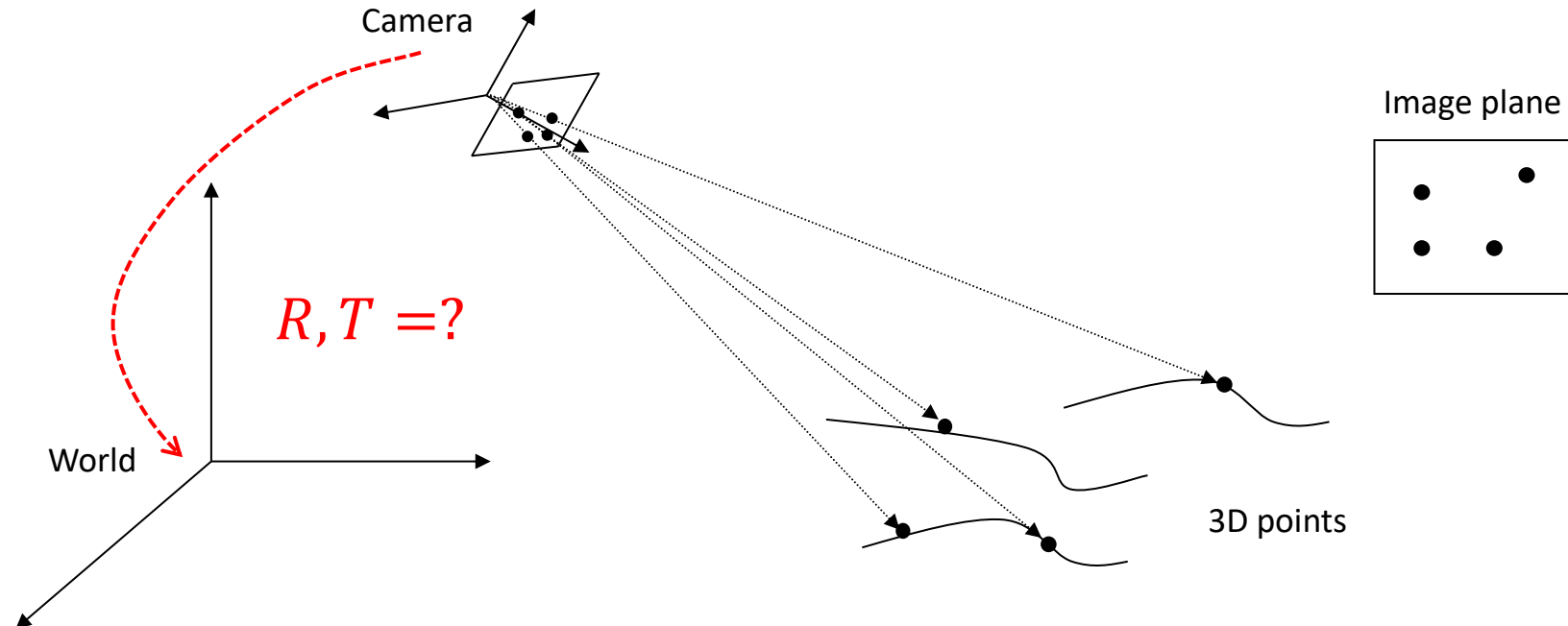
Marc Pollefeys' lab. [Video](#).

# Today's Outline

- Camera calibration
- Camera localization
- Non conventional camera models: fisheye and catadioptric cameras

# Camera Localization (or Perspective from $n$ Points: PnP)

- This is the problem of determining the **6DoF pose of a calibrated camera** (position and orientation) with respect to the world frame **from a set of 3D-2D point correspondences**.
- It assumes that the **camera is already calibrated** (i.e., we know its intrinsic parameters)
- The **DLT can be used** to solve this problem **but is suboptimal**. We want to study **algebraic solutions** to the problem.

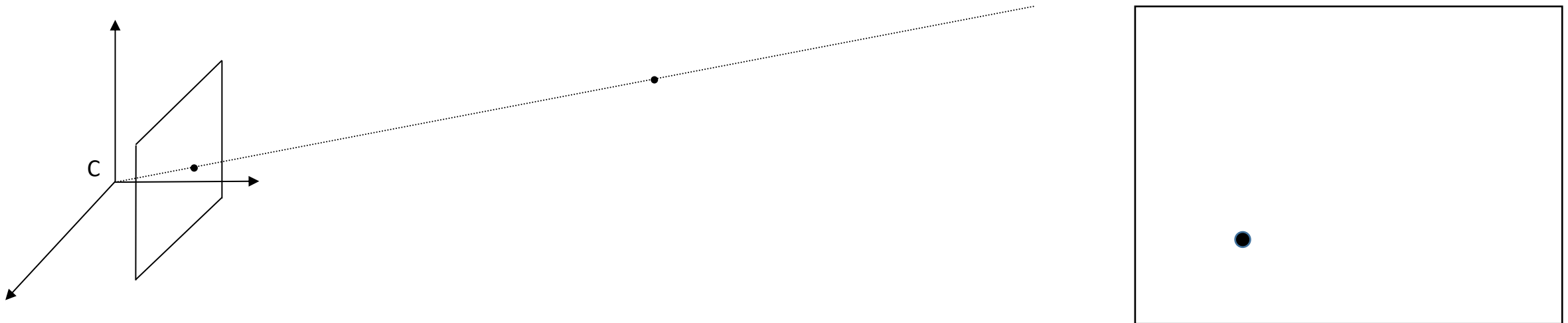


# How Many Points are Enough?

- **1 Point:**
  - infinite solutions
- **2 Points:**
  - infinitely many solutions, but bounded
- **3 Points (non collinear):**
  - up to 4 solution
- **4 Points:**
  - Unique solution

# 1 Point

- **1 Point:**
  - infinite solutions



# 2 Points

- **2 Points:**
  - infinite solutions, but bounded

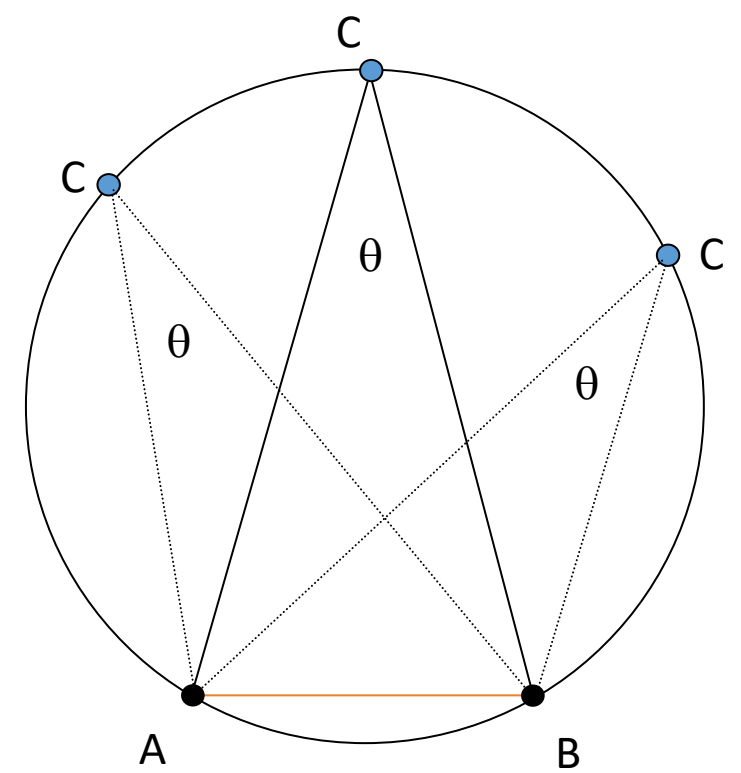
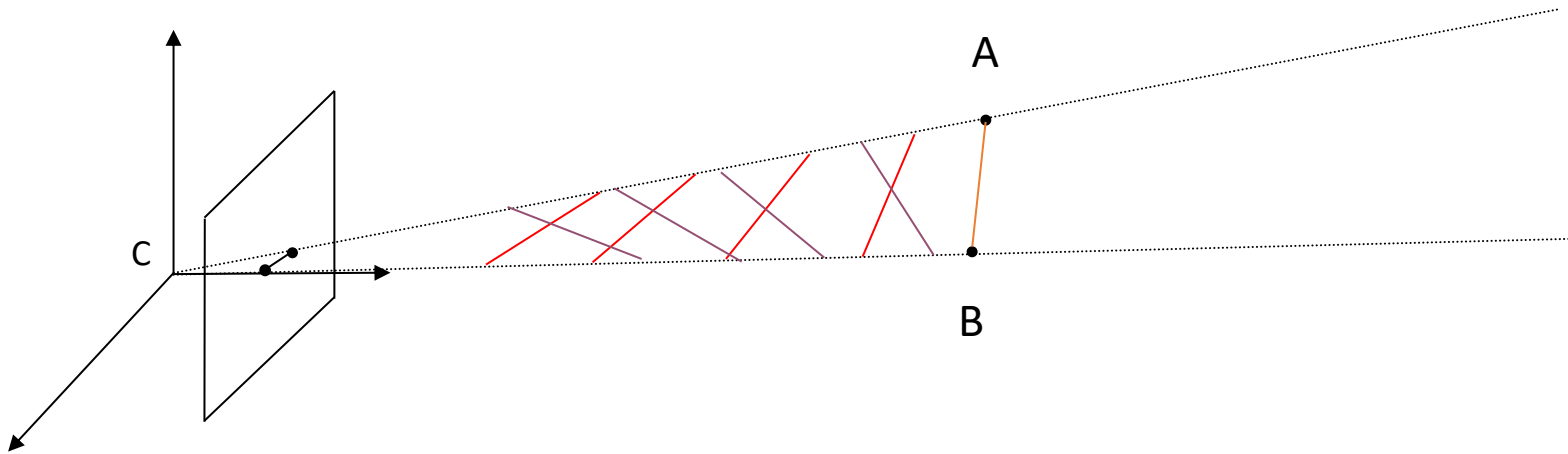
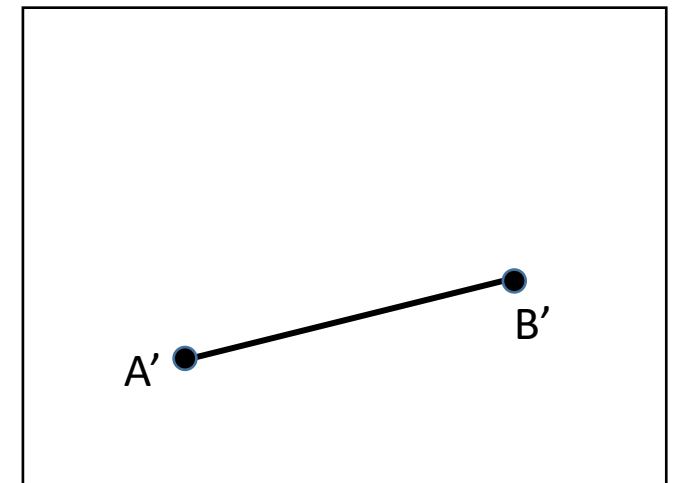


Image plane



# 3 Points (P3P problem)

- **3 Points (non collinear):**
  - up to 4 solution

From the [law of cosines](#):

$$s_1^2 = L_B^2 + L_A^2 - 2L_B L_A \cos \theta_{AB}$$

$$s_2^2 = L_A^2 + L_C^2 - 2L_A L_C \cos \theta_{AC}$$

$$s_3^2 = L_B^2 + L_C^2 - 2L_B L_C \cos \theta_{BC}$$

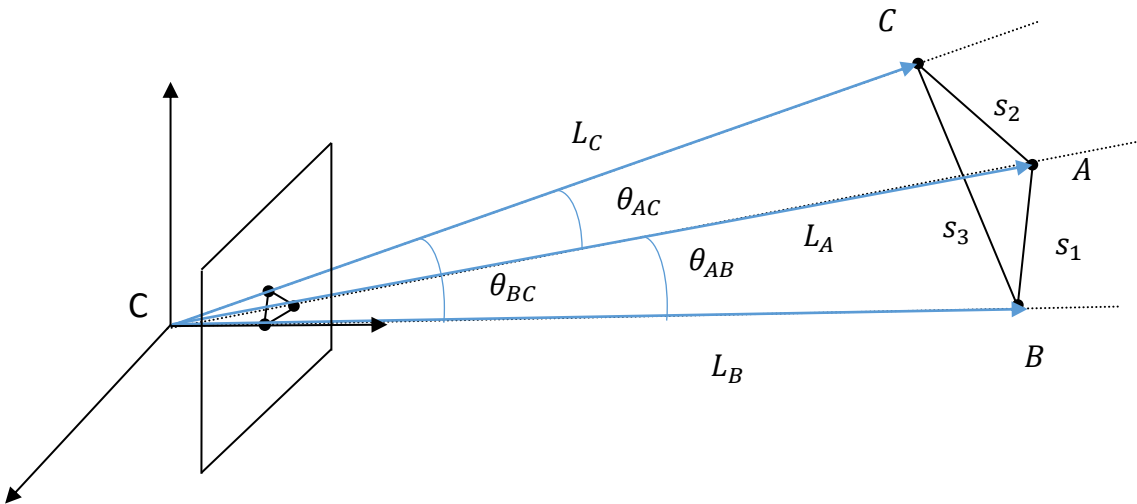
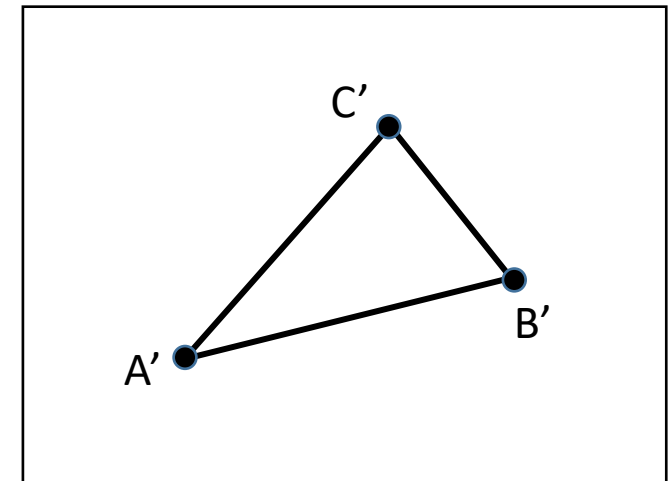


Image plane



# Algebraic Approach: reduce to 4<sup>th</sup> order equation

$$s_1^2 = L_B^2 + L_A^2 - 2L_B L_A \cos \theta_{AB}$$

$$s_2^2 = L_A^2 + L_C^2 - 2L_A L_C \cos \theta_{AC}$$

$$s_3^2 = L_B^2 + L_C^2 - 2L_B L_C \cos \theta_{BC}$$

- It is known that  **$n$  independent polynomial equations**, in  **$n$  unknowns**, can have no more **solutions** than the **product of their respective degrees**. Thus, the system can have a maximum of **8** solutions. However, because every term in the system is either a constant or of second degree, for every real positive solution **there is a negative solution**.
- Thus, with 3 points, there are at most **4 valid (positive) solutions**.

This algorithm was published in:

- M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Graphics and Image Processing, 1981. [PDF](#).

However the first solution to the P3P problem was published in 1841:

- J. A. Grunert. Das pothenotische problem in erweiterter gestalt nebst über seine anwendungen in der geodesie. *Grunerts archiv für mathematik und physik*, **1841**.

# Algebraic Approach: reduce to 4<sup>th</sup> order equation

$$s_1^2 = L_B^2 + L_A^2 - 2L_B L_A \cos \theta_{AB}$$

$$s_2^2 = L_A^2 + L_C^2 - 2L_A L_C \cos \theta_{AC}$$

$$s_3^2 = L_B^2 + L_C^2 - 2L_B L_C \cos \theta_{BC}$$

- By defining  $x = L_B/L_A$ , it can be shown that the system can be reduced to a 4<sup>th</sup> order equation:

$$G_0 + G_1 x + G_2 x^2 + G_3 x^3 + G_4 x^4 = 0$$

How can we disambiguate the 4 solutions? How do we determine  $R$  and  $T$ ?

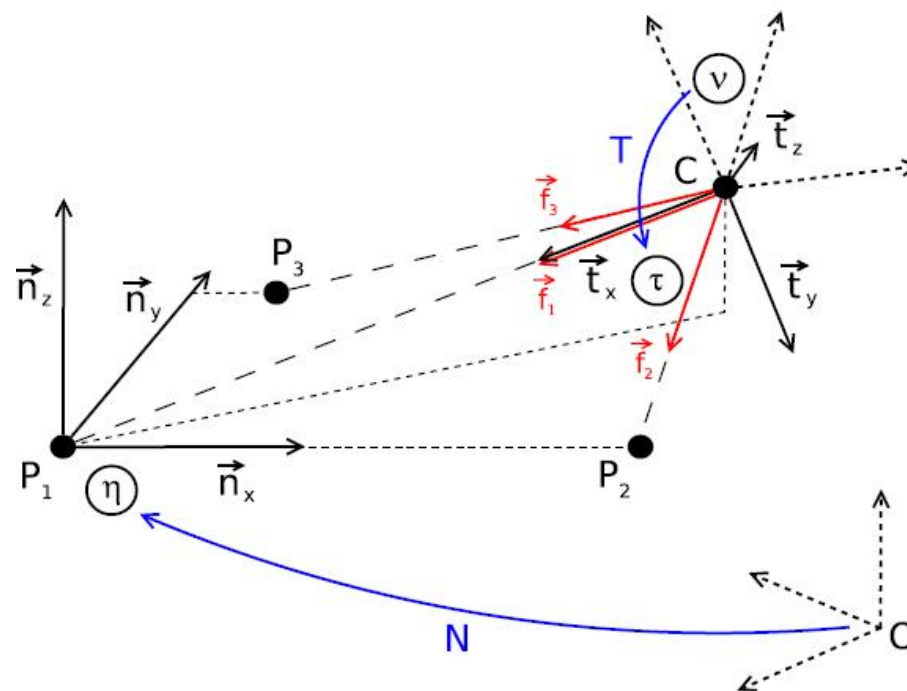
- A 4<sup>th</sup> point can be used to disambiguate the solutions. A classification of the four solutions and the determination of  $R$  and  $T$  from the point distances was given by Gao's algorithm, implemented in OpenCV ([solvePnP P3P](#))

# Modern Solution to P3P

Won't be asked  
at the exam  
😊

A more **modern version of P3P** was developed by Kneip in 2011 and **directly solves for the camera's pose** (not distances from the points). This solution inspired the algorithm currently used in OpenCV ([solvePnP AP3P](#)), by Ke'17, which consists of two steps:

1. Eliminate the camera's position and the features' distances to yield a system of 3 equations in *the camera's orientation alone*.
  2. Successively eliminate two of the unknown 3-DOFs (angles) algebraically and arrive at a **quartic polynomial equation**.
- Outperforms previous methods in terms of speed, accuracy, and robustness to close-to-singular cases.



Kneip, Scaramuzza, Siegwart. A Novel Parameterization of the Perspective-Three-Point Problem for a Direct Computation of Absolute Camera Position and Orientation. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2011. [PDF](#).

Ke, Roumeliotis. An Efficient Algebraic Solution to the Perspective-Three-Point Problem. CVPR'17. [PDF](#).

# Solution to PnP for $n \geq 4$

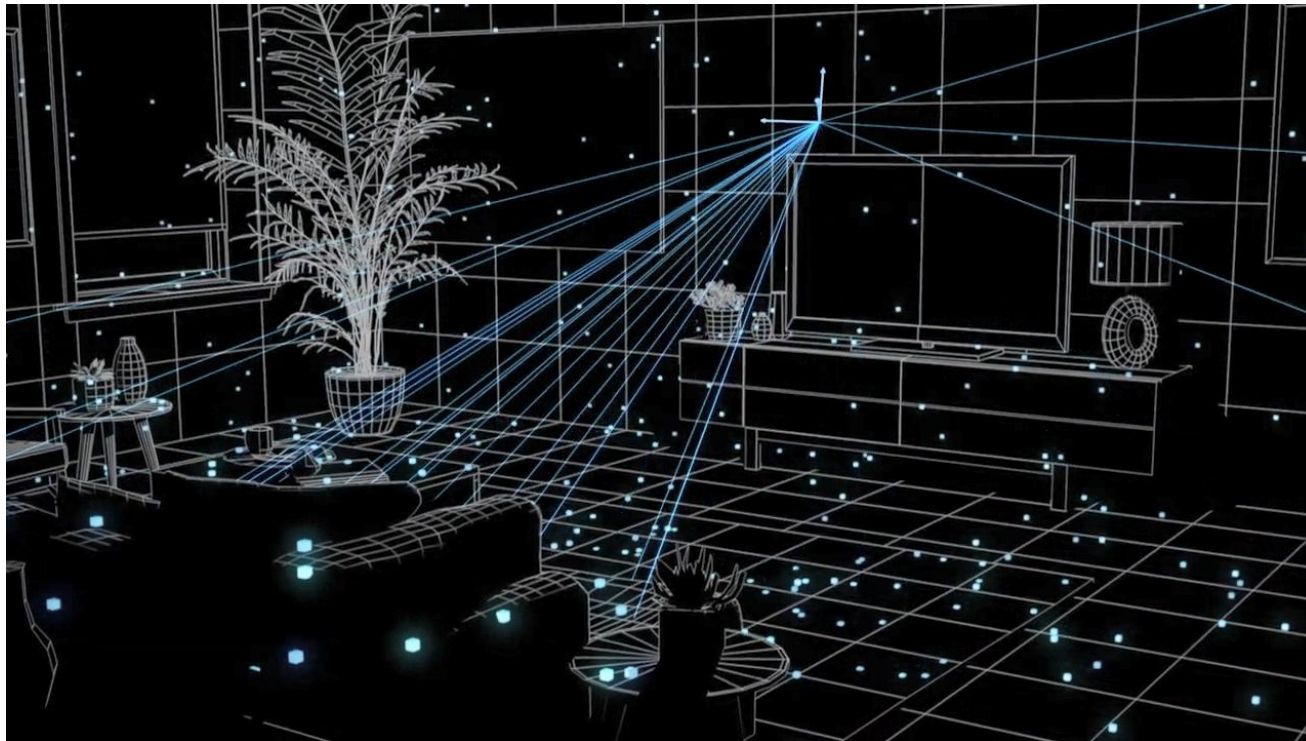
An efficient algebraic solution to the PnP problem for  $n \geq 4$  was developed by Lepetit in 2009 and was named **EPnP** (Efficient PnP) and can be found in OpenCV ([solvePnP EPnP](#))

- EPnP expresses the  $n$  world's points as a weighted sum of **four virtual control points**
- The coordinates of these virtual control points become the **unknowns of the problem**, which can be solved in  $O(n)$  time by solving a **constant number of quartic polynomial equations**
- The final pose of the camera is then solved from the control points



# Application to Monocular Localization

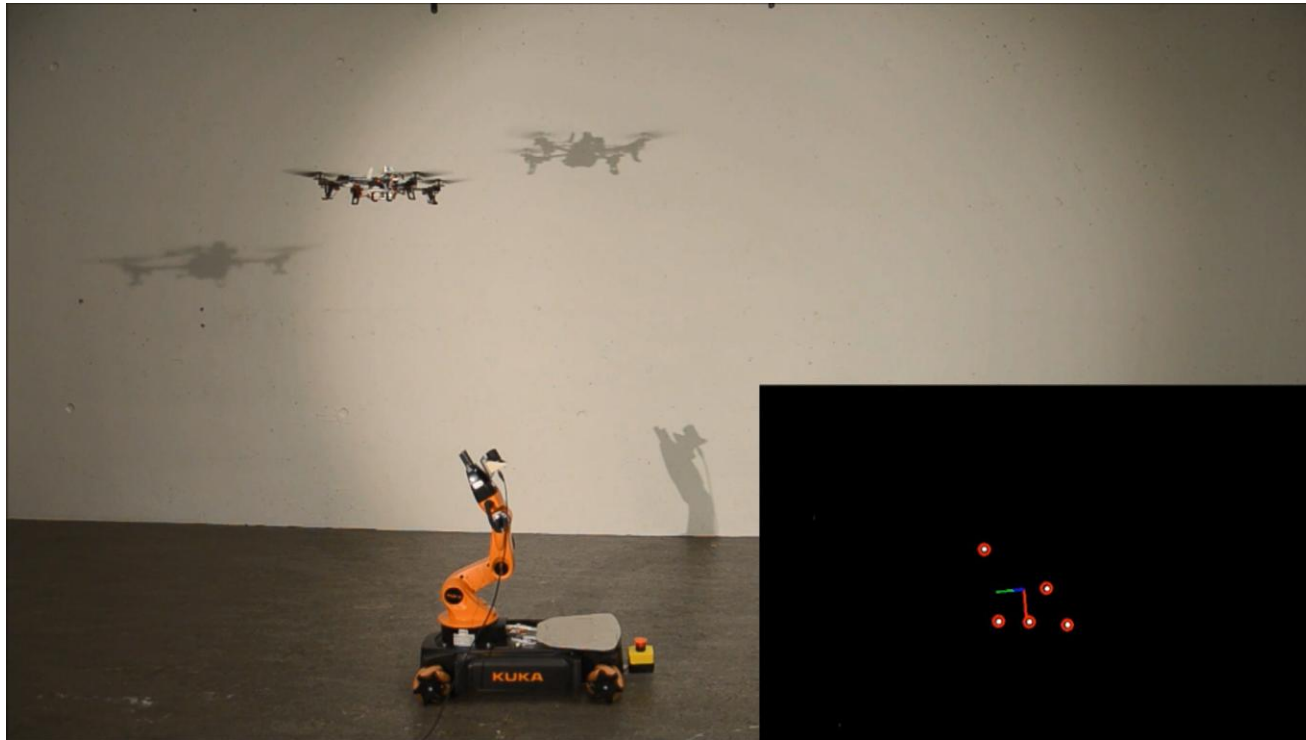
**Localization:** Given a 3D point cloud (map), determine the pose of the camera



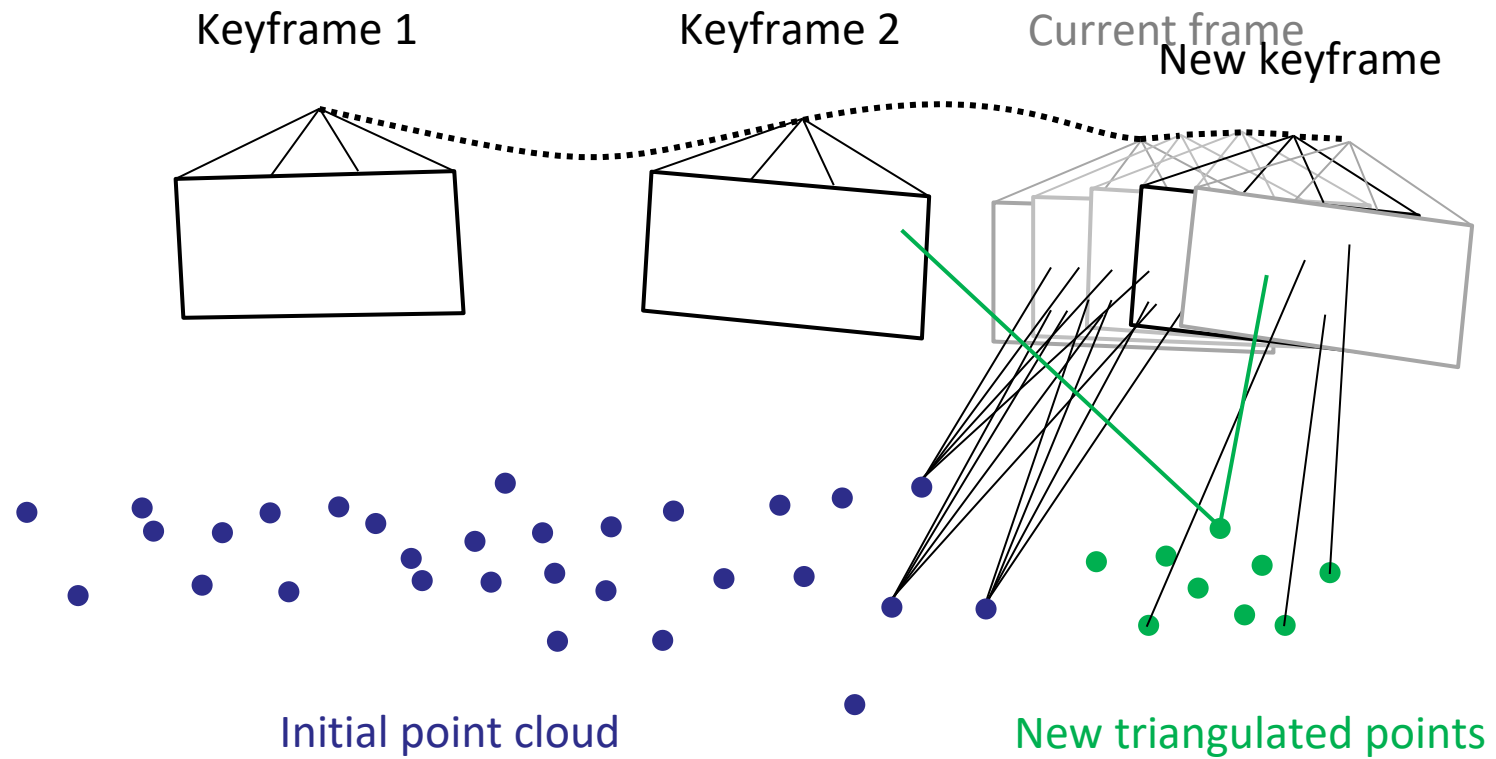
[Video](#) of Oculus Insight (the VIO used in Meta Quest) built by former [Zurich-Eye team](#), today Meta Zurich. The [story](#) from Zurich-Eye to Meta Quest.

# Application to Multi-Robot mutual Localization

- Here, the drone carries 5 LEDs that are used by the ground robot to control the drone's position relative to it
- 4 is the minimum but here 5 points are used for robustness to occlusions. Also observe that there is a distractor (an outlier LED moved by the person). We use RANSAC (Lecture 9) to remove it



# Application to Monocular Visual Odometry



# Robust Estimation in Presence of Outliers

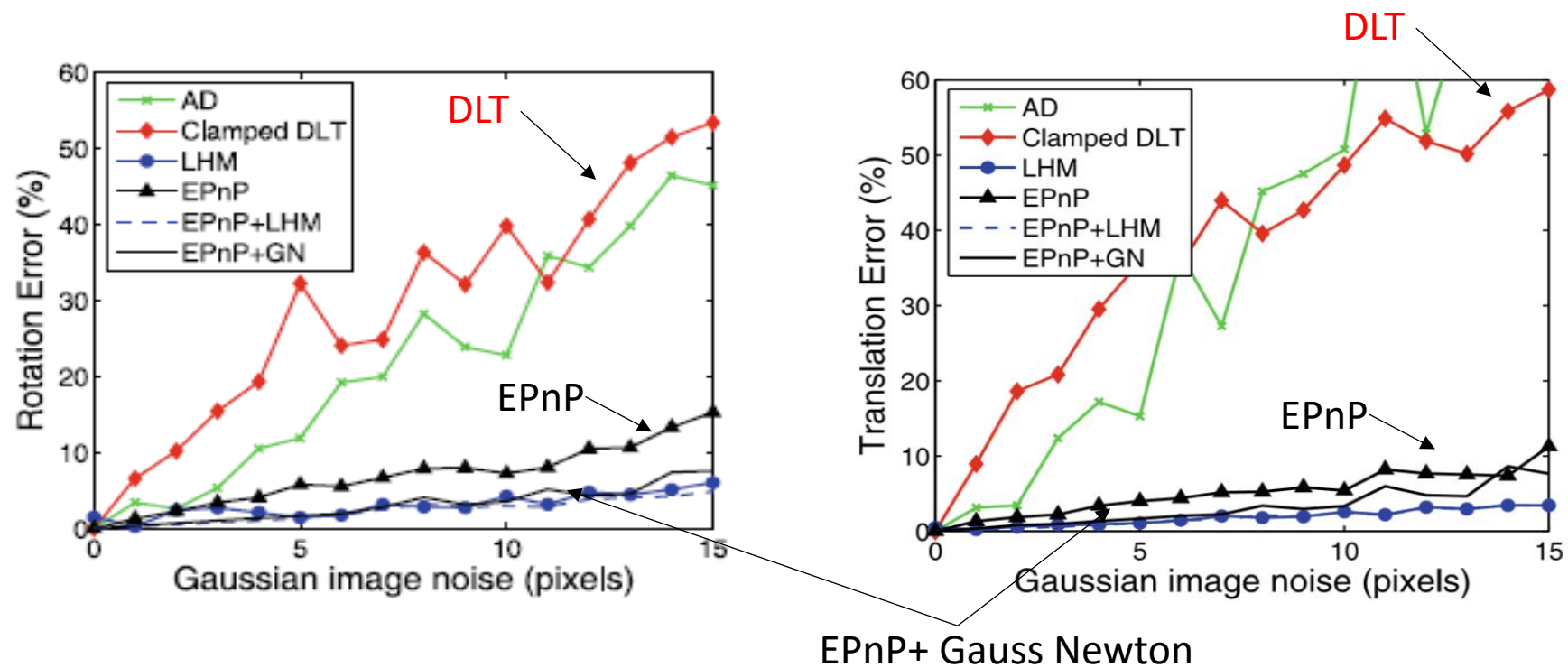
- **All PnP problems** (solved by DLT, EPnP, or P3P algorithms) assume that the 3D-2D point correspondences are correct. If the correspondences are incorrect (i.e., they contain outliers), then the output of these algorithms may be incorrect.
- The **RANSAC** algorithm (**Lecture 09**) can be used, in conjunction with the PnP algorithm, to **remove the outliers** (we will do this in **Exercise 07**).
- PnP with RANSAC can be found in OpenCV's ([solvePnP Ransac](#))

# EPnP vs. DLT

**If a camera is calibrated**, only  $R$  and  $T$  need to be determined. In this case, **should we use DLT or EPnP** for localization?

# EPnP vs. DLT: Accuracy vs. noise

EPnP is up to **10 times more robust to noise** than DLT

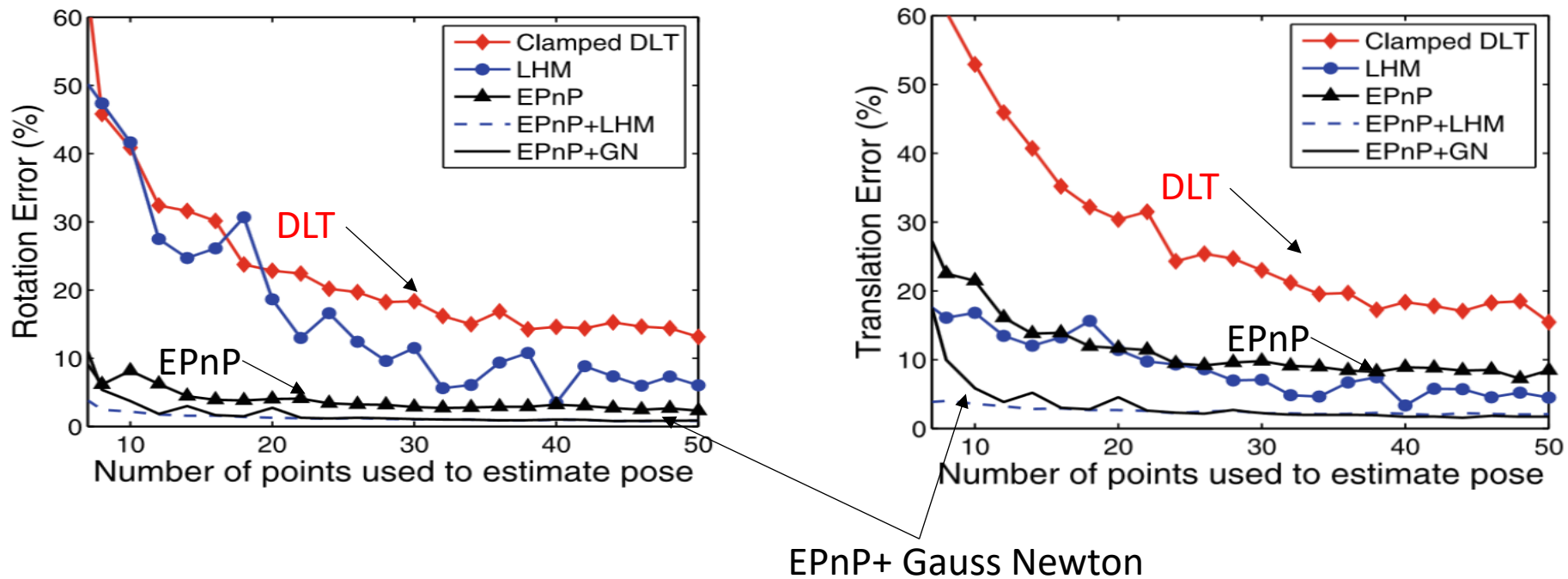


Plots from

Lepetit, Moreno Noguer, Fua, EPnP: An Accurate  $O(n)$  Solution to the PnP Problem, International Journal of Computer Vision. [PDF](#).

# EPnP vs. DLT: Accuracy vs. number of points

EPnP is up to **10 times more accurate** than DLT

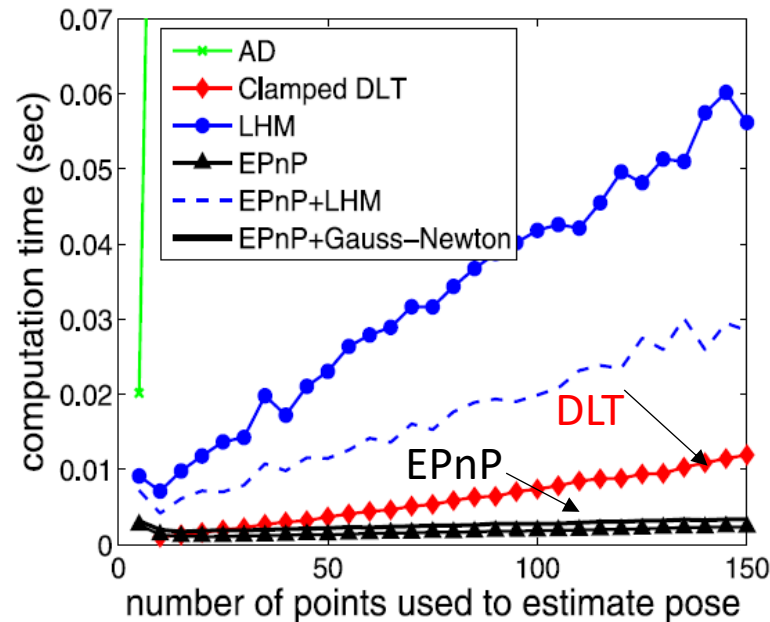


Plots from

Lepetit, Moreno Noguer, Fua, EPnP: An Accurate  $O(n)$  Solution to the PnP Problem, International Journal of Computer Vision. [PDF](#).

# EPnP vs. DLT: Timing

EPnP is up to **10 times more efficient** than DLT



Plots from

Lepetit, Moreno Nogue, Fua, EPnP: An Accurate  $O(n)$  Solution to the PnP Problem, International Journal of Computer Vision. [PDF](#).

# PnP problem: Recap

| <b>Uncalibrated camera</b><br>(i.e., intrinsic parameters are NOT known) | <b>Calibrated camera</b><br>(i.e., instrinc parameters are known) |
|--------------------------------------------------------------------------|-------------------------------------------------------------------|
| <b>Only DLT</b> can be used                                              | <b>Either DLT or EPnP</b> can be used                             |

**EPnP**: minimum number of points: **3 (P3P) +1** for disambiguation

**DLT**: Minimum number of points: **4 if coplanar, 6 if non-coplanar**

---

The output of both DLT and EPnP can be refined via **non-linear optimization** by minimizing the sum of squared reprojection errors.

NB: Neither EPnP or DLT compute the **lens distortion parameters**, these can be **estimate via reprojection error minimization**

# Today's Outline

- Camera calibration
- Camera localization
  - Case study: Vision-based Autonomous Drone Racing
- Non conventional camera models: fisheye and catadioptric cameras

The international journal of science / 31 August 2023

# nature

## DRONE RACER

AI pilot beats  
human champions  
in aerial contest



**Offset agreement**  
Overhaul pricing of  
carbon credits to help  
fund climate projects

**Dining companions**  
Corals devour algal  
partners when food  
supplies run low

**Sentence synthesis**  
Brain implants show  
promise in rendering  
speech from thoughts

Vol. 620, No. 7976  
nature.com

# Science Robotics

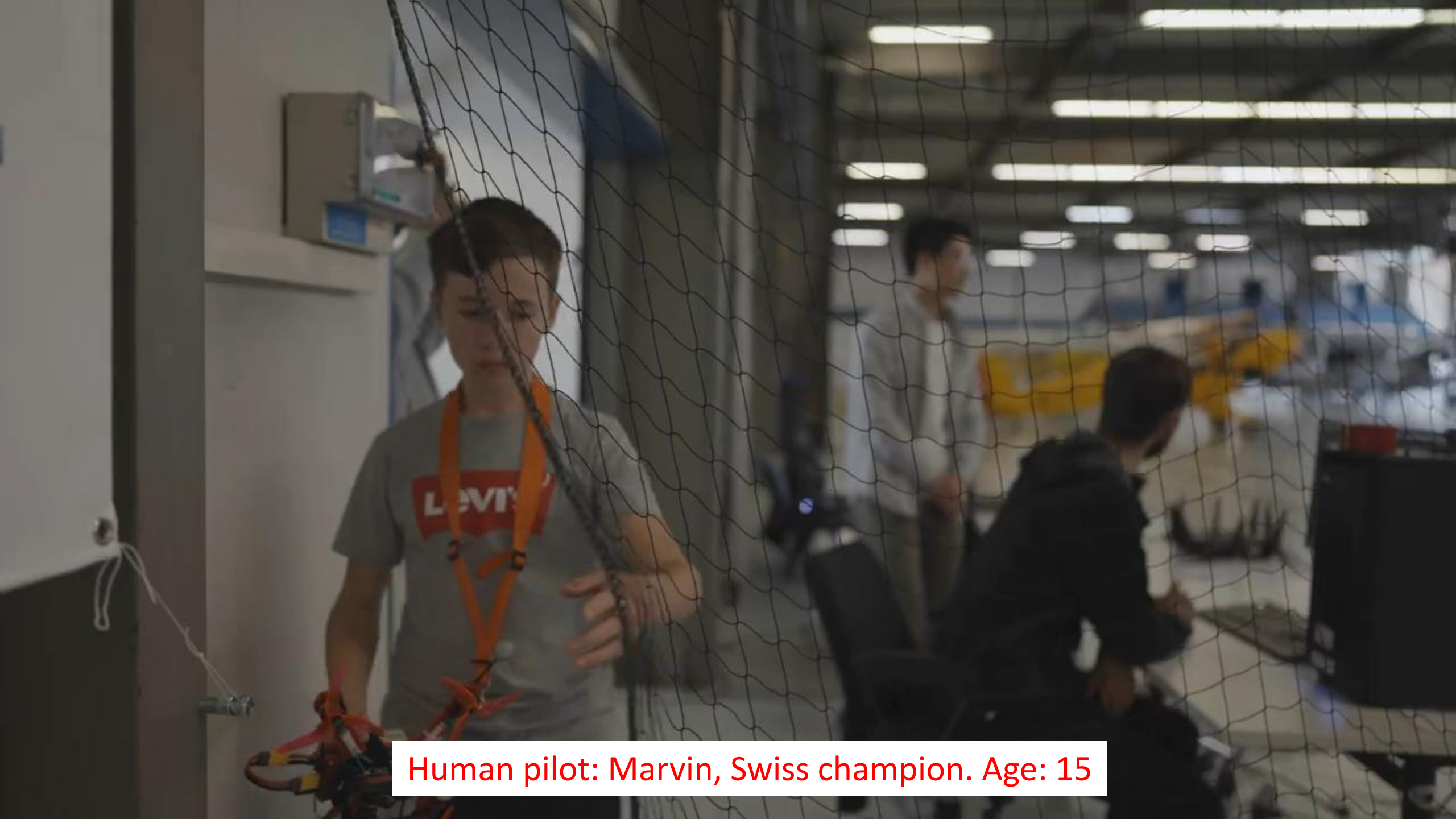
SEPTEMBER 2023



AAAS

Kaufmann et al., *Champion-Level Drone Racing using Deep Reinforcement Learning*, **Nature**, 2023

Song et al., *Reaching the Limit in Autonomous Racing: Optimal Control versus Reinforcement Learning*, **Science Robotics**, 2023

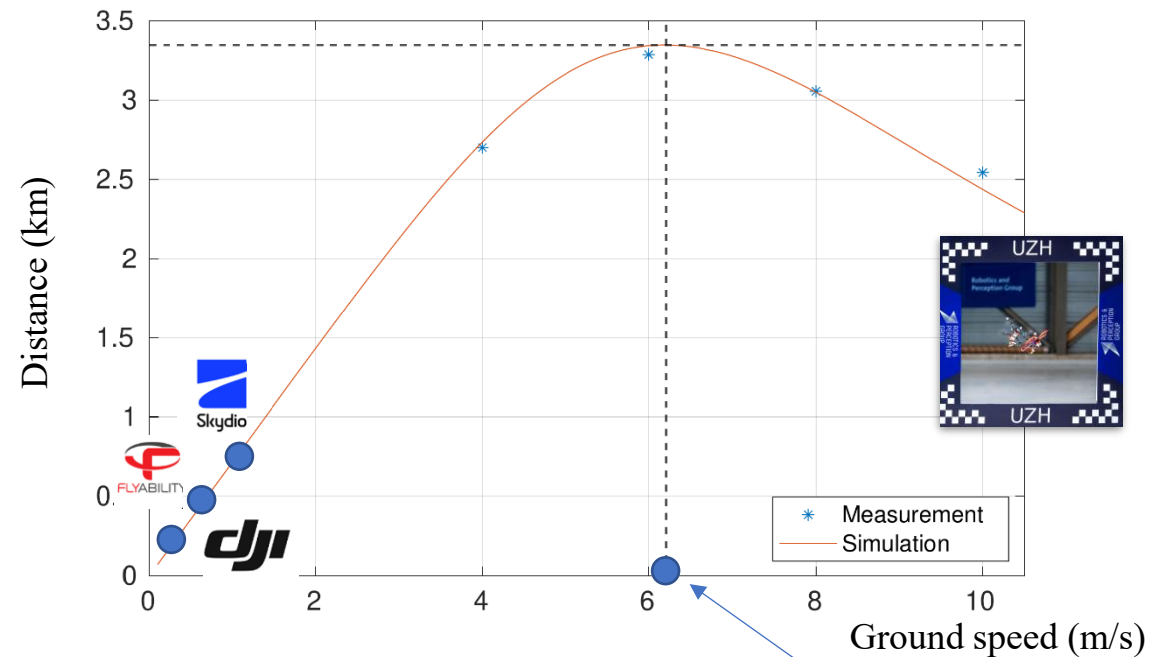


Human pilot: Marvin, Swiss champion. Age: 15

# Why Agile? To increase productivity!

- Multi-rotor drone's **battery** limited to **30 minutes**
- By flying **faster**, drones can be more **productive** [1,2]

Drone racing does not increase the productivity but it was important to study it as a benchmark to push the limits of vision-based navigation, very much like parkour is a benchmark for humanoid robots

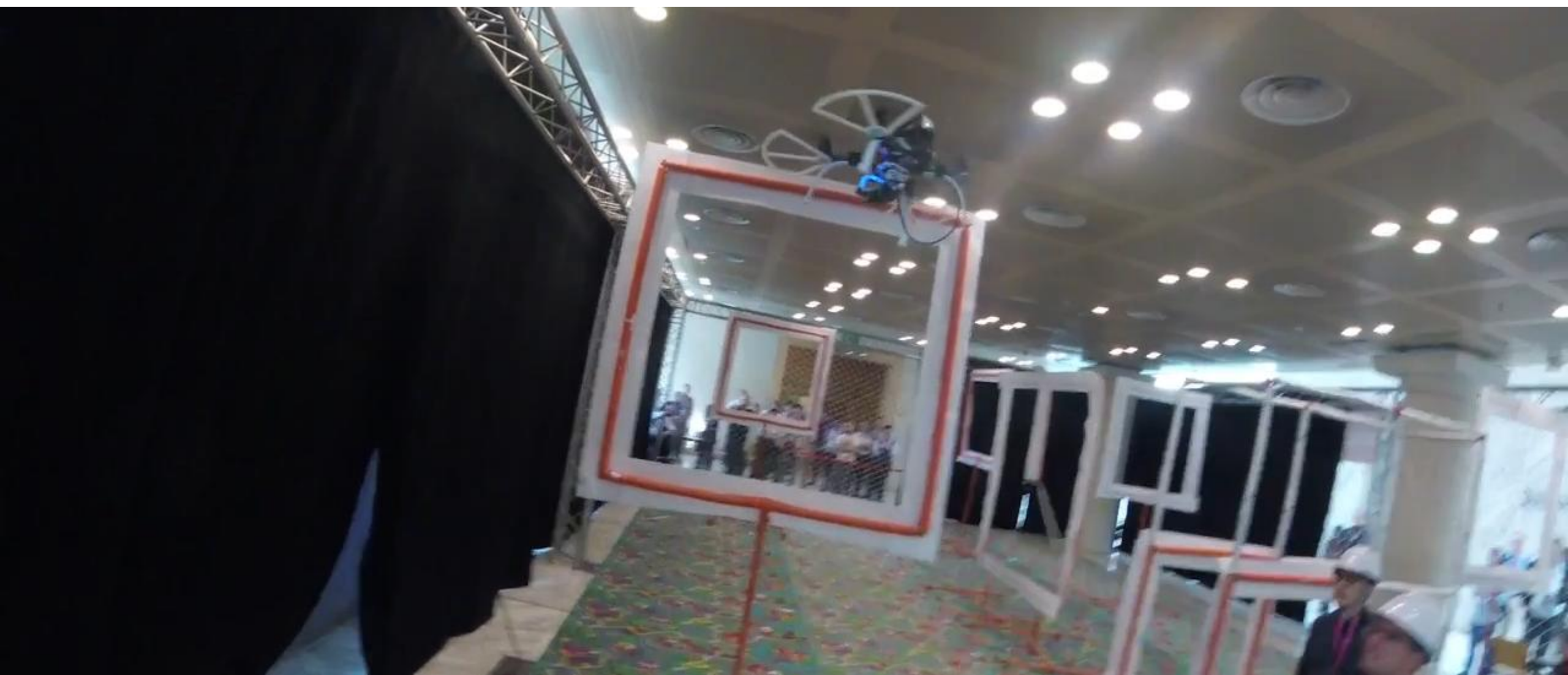


Optimal speed

[1] Bauersfeld, Scaramuzza, *Range, Endurance, and Optimal Speed Estimates for Multicopters*, RAL'22

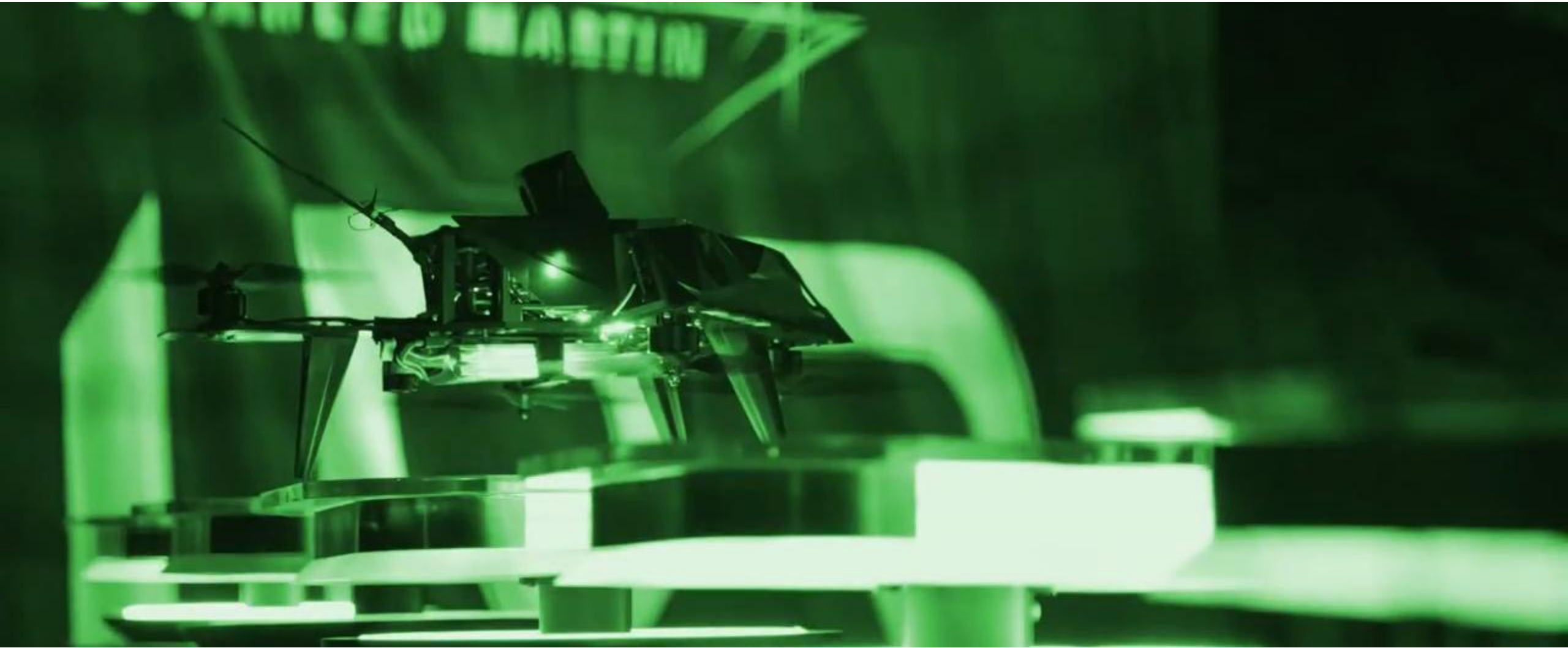
[2] Karydis, Kumar, *Energetics in robotic flight at small scales*, Interface Focus, 2017

# 2018: autonomous drone race in Madrid



Maximum speed 3 km/h

# 2019: autonomous drone race in Austin



Maximum speed 30km/h

# Our AI Drone, named SWIFT

**The brain**  
NVIDIA Jetson TX2

**The eyes**  
Intel RealSense camera

From 0 to 100 km/h in 1 second  
Weight: 0.8kg  
Max acceleration: 5 G



# Neural-Network controller trained with Reinforcement Learning



# Can we outrace the best human pilot?

**After 7 years of work**, in June 2022, we invited the world champions of drone racing



**Alex Vanover**

DRL World  
Champion



**Thomas  
Bitmatta**

MultiGP  
International  
World Champion



**Marvin  
Schaepper**

Swiss Drone  
League  
Champion



**AI Drone**

**Both human and AI drones  
were identical**

Kaufmann, Bauersfeld, Loquercio, Mueller, Koltun, Scaramuzza,  
*Champion-Level Drone Racing using Deep Reinforcement Learning*, **Nature**, 2023. [PDF](#). [Datasets](#). [Our video](#). [Video by Nature](#)



Kaufmann, Bauersfeld, Loquercio, Mueller, Koltun, Scaramuzza,  
*Champion-Level Drone Racing using Deep Reinforcement Learning*, **Nature**, 2023. [PDF](#). [Datasets](#). [Our video](#). [Video by Nature](#)

# Drone Racing

Autonomous  
Drone

"Swift"

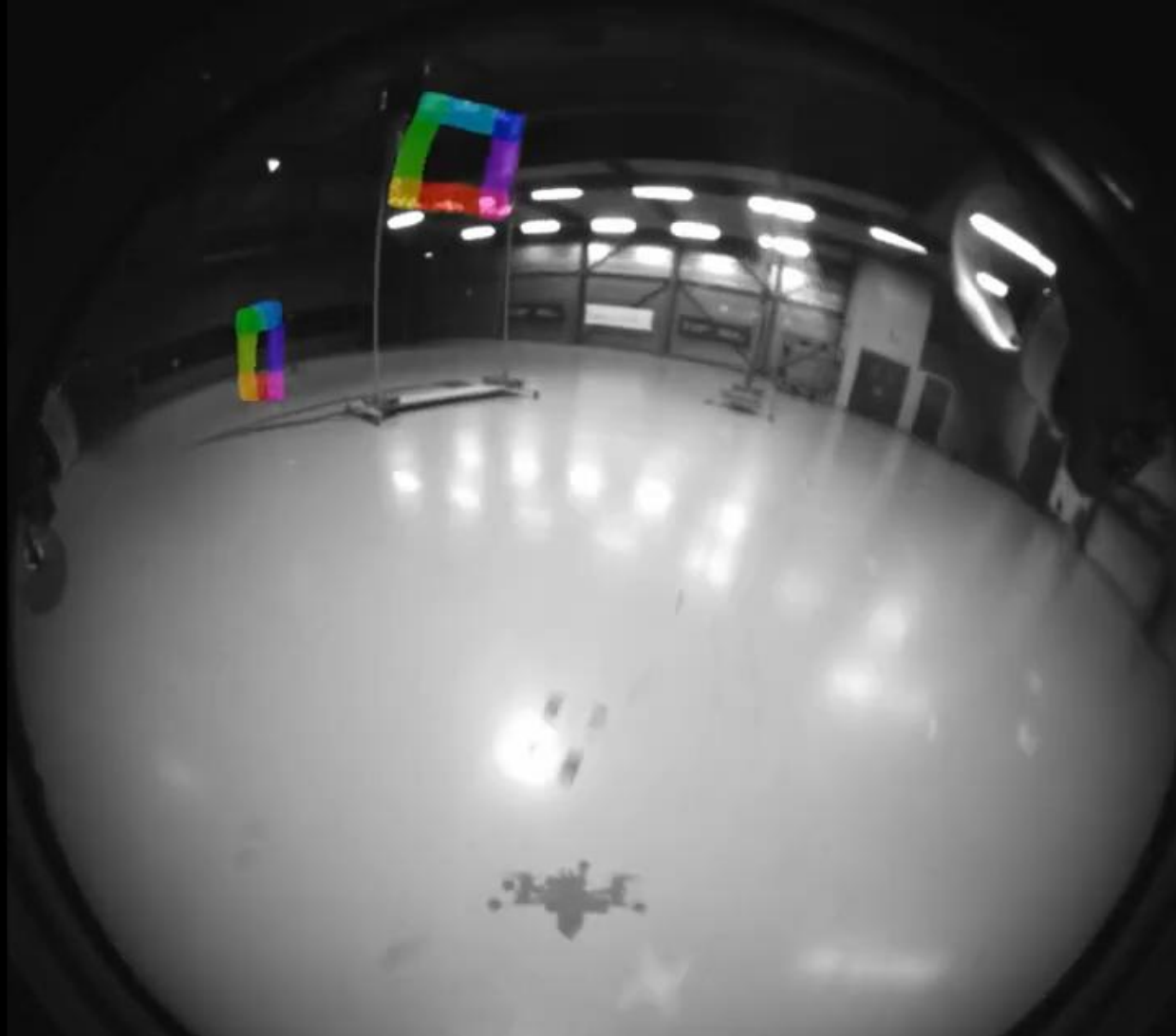
World's Best  
Human Pilots

A. Vanover,  
T. Bitmatta,  
M. Schaepper

# Current documentary screening at Arthouse Piccadilly Zurich

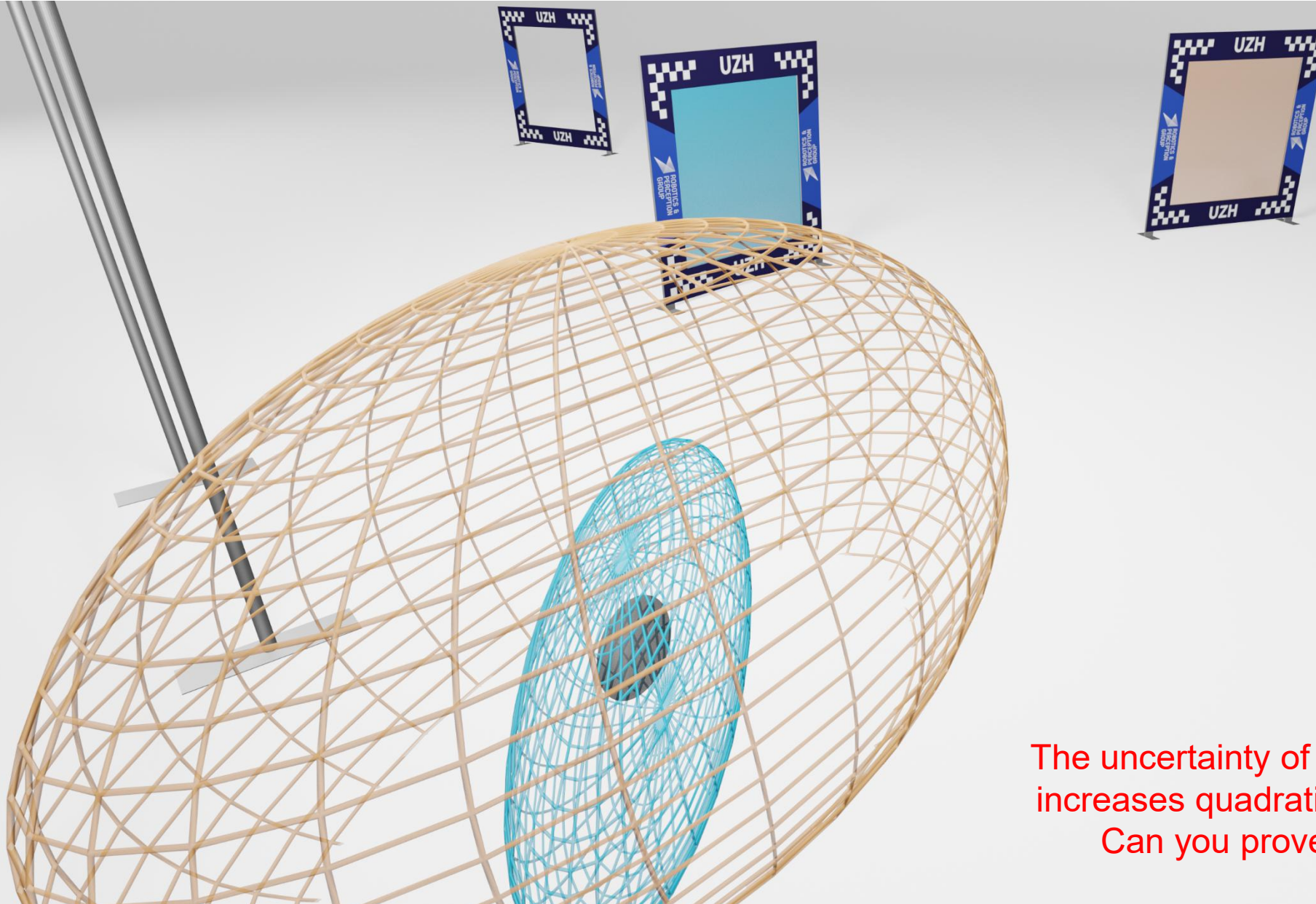


# Compute Drone Pose via Gate Detection and P3P



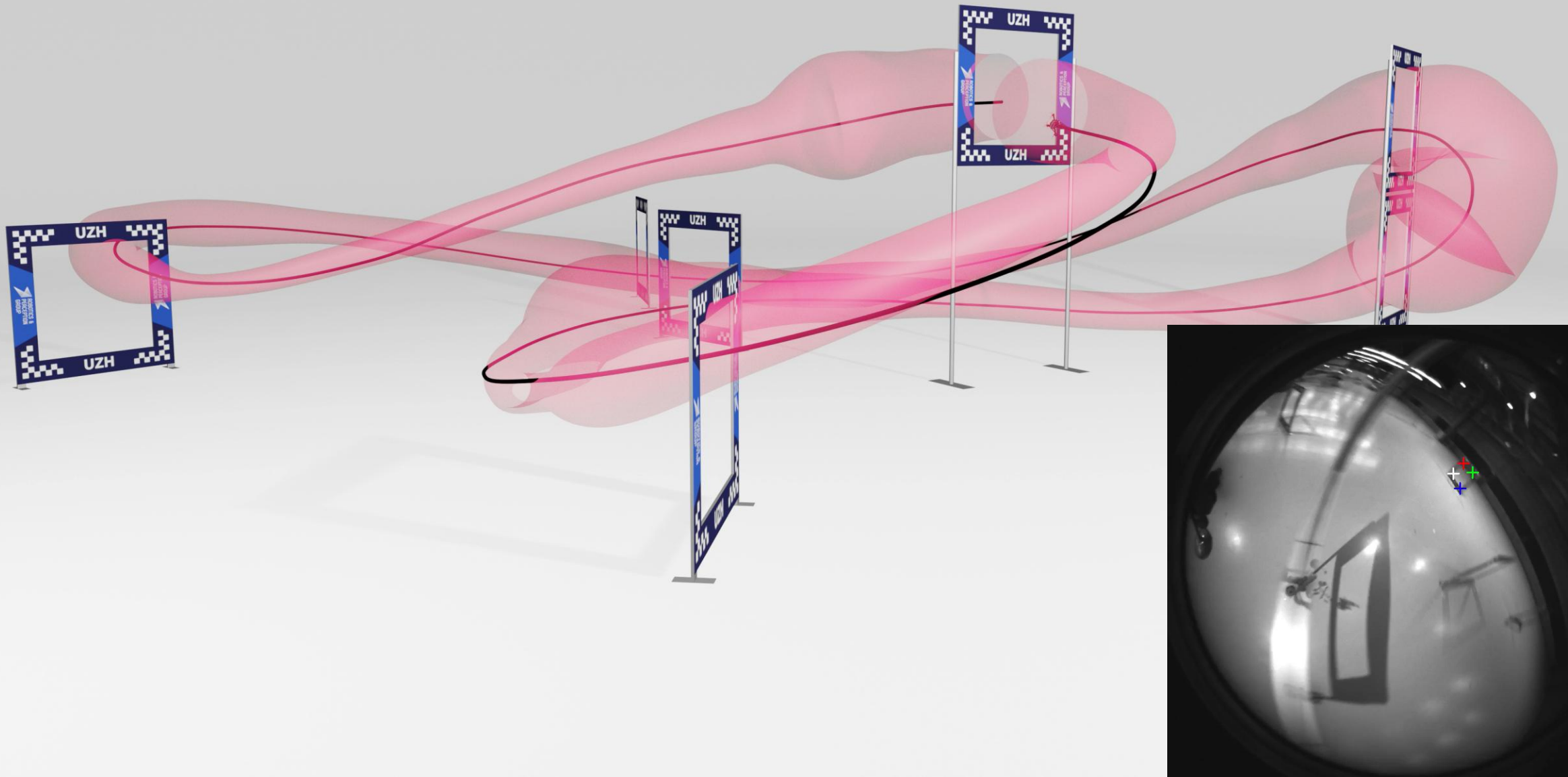
Kaufmann, Bauersfeld, Loquercio, Mueller, Koltun, Scaramuzza,  
*Champion-Level Drone Racing using Deep Reinforcement Learning*, **Nature**, 2023. [PDF](#). [Datasets](#). [Our video](#). [Video by Nature](#)

# Calculate Uncertainty of Drone Position



The uncertainty of the estimation distance increases quadratically with the distance.  
Can you prove it mathematically?

# Calculate Uncertainty of Drone Position



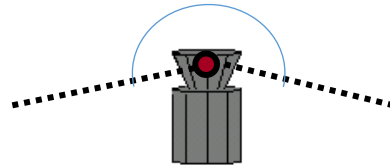
# Today's Outline

- Camera calibration
- Camera localization
- Non conventional camera models: fisheye and catadioptric cameras

# Overview on Omnidirectional Cameras

Fisheye

FOV > 130°



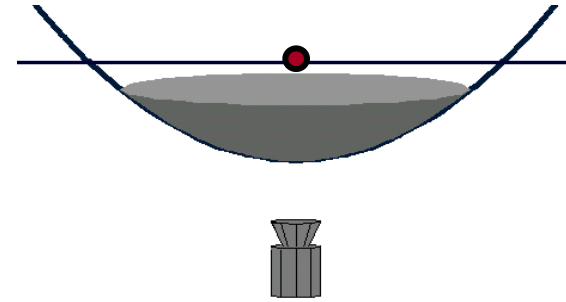
Wide FOV dioptric cameras (e.g. fisheye)



FOV = Field of View

Catadioptric

360° all around



Catadioptric cameras (e.g. cameras and mirror systems)



# Camera View Comparison



Perspective



Fisheye  
(highest resolution is achieved around the  
optical axis)



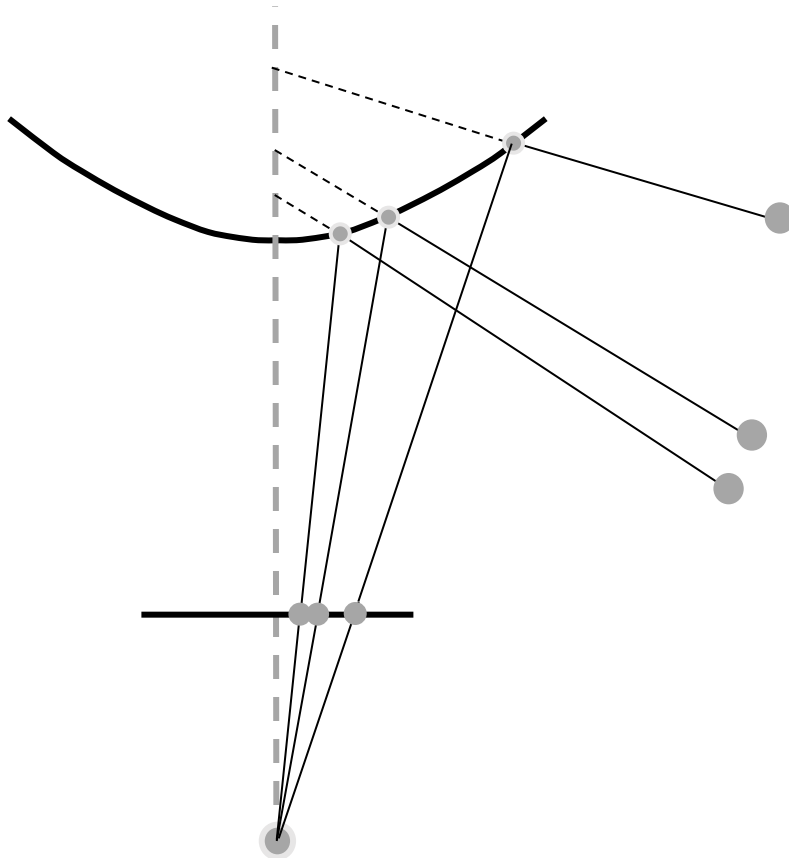
Catadioptric  
(highest resolution is achieved  
perpendicularly to the  
optical axis)

Zhang, Rebecq, Forster, Scaramuzza. Benefit of Large Field-of-View Cameras for Visual Odometry.  
IEEE International Conference on Robotics and Automation (ICRA), 2016. [PDF](#).

# Central vs Noncentral Omnidirectional Cameras

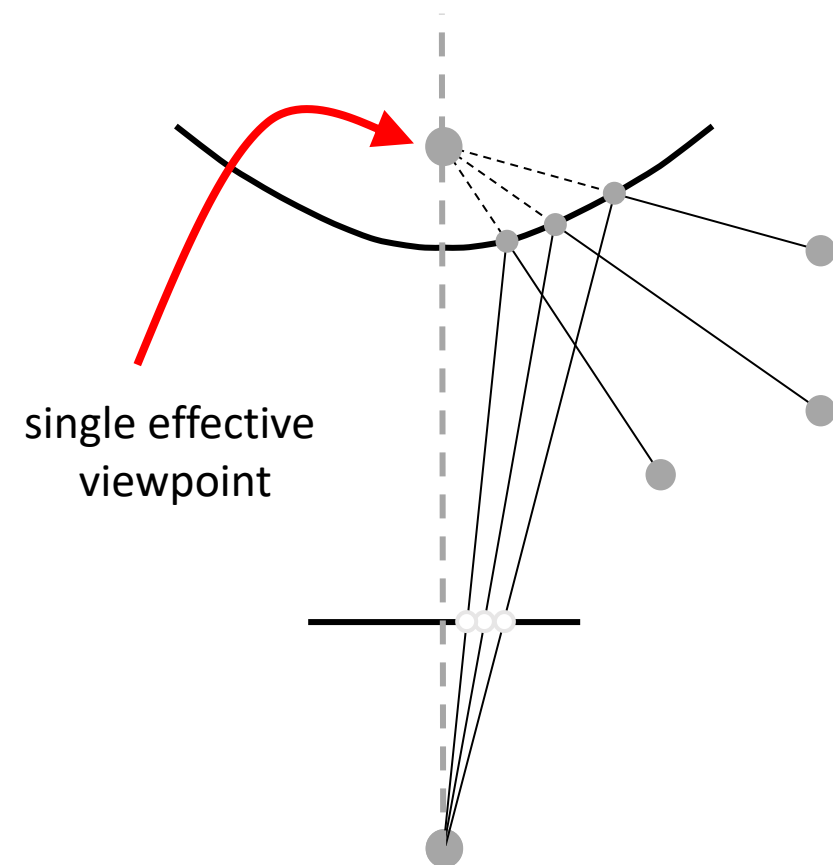
## Non-Central projection system

Rays do not intersect in a single point



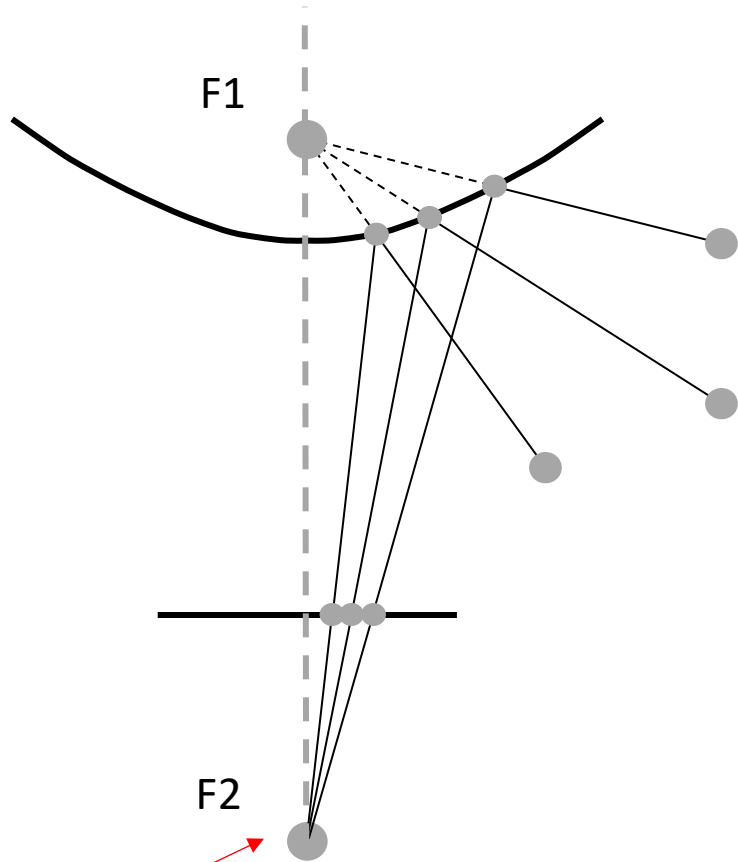
## Central projection system

Rays intersect in a single point



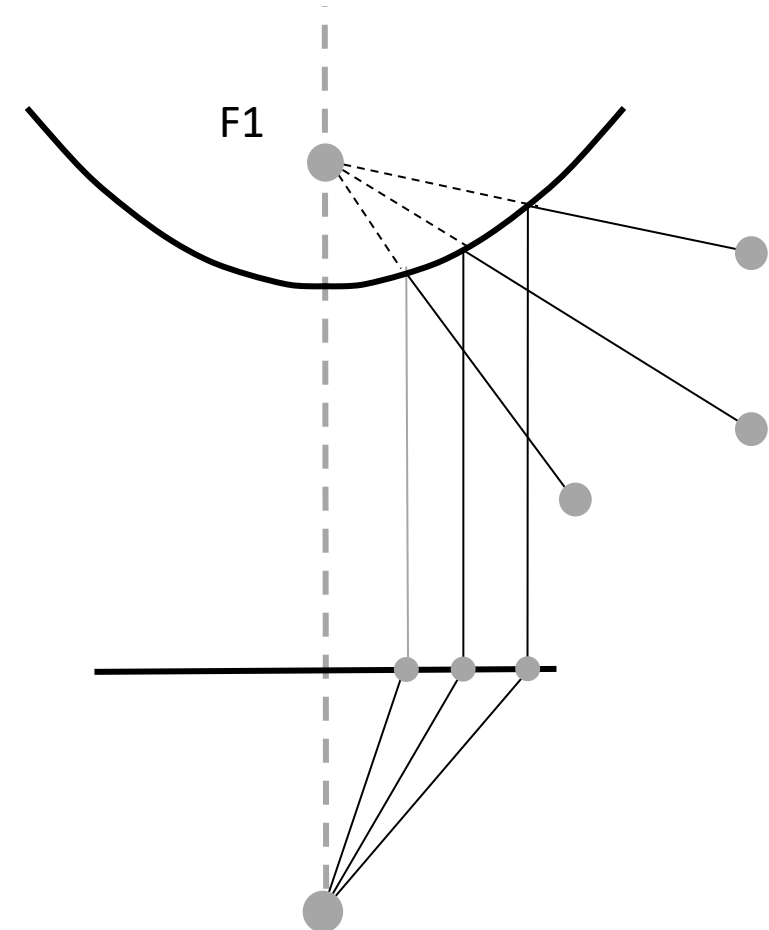
# Central Omnidirectional Cameras

Hyperbola + Perspective camera



NB: one of the foci of the hyperbola must lie in the camera's center of projection

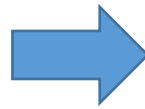
Parabola + Orthographic lens



# Why do we prefer central cameras?

Because we can:

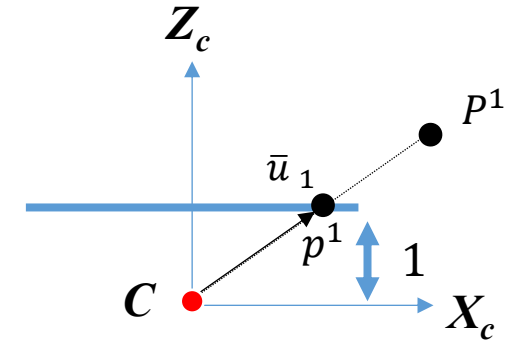
- Apply standard algorithms valid for perspective geometry.
- Unwarp parts of an image into a perspective one
- Transform image points into normalized vectors on the unit sphere



# Recall: Normalized Image Coordinates (Lecture 2, slide 62)

If we pre-multiply both terms of the perspective projection equation in camera frame by  $K^{-1}$ , we get:

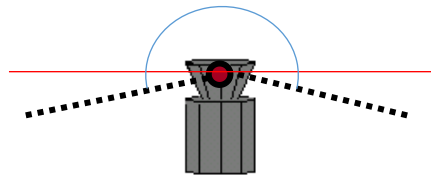
$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \Rightarrow \lambda K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \Rightarrow \lambda \begin{bmatrix} \frac{u - u_0}{\alpha} \\ \frac{v - v_0}{\alpha} \\ 1 \end{bmatrix} = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$



$$\begin{bmatrix} \bar{u} \\ \bar{v} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{u - u_0}{\alpha} \\ \frac{v - v_0}{\alpha} \\ 1 \end{bmatrix}$$

# How do we model 3D points that are behind the camera?

The standard pinhole model is not enough. We need a distortion model.



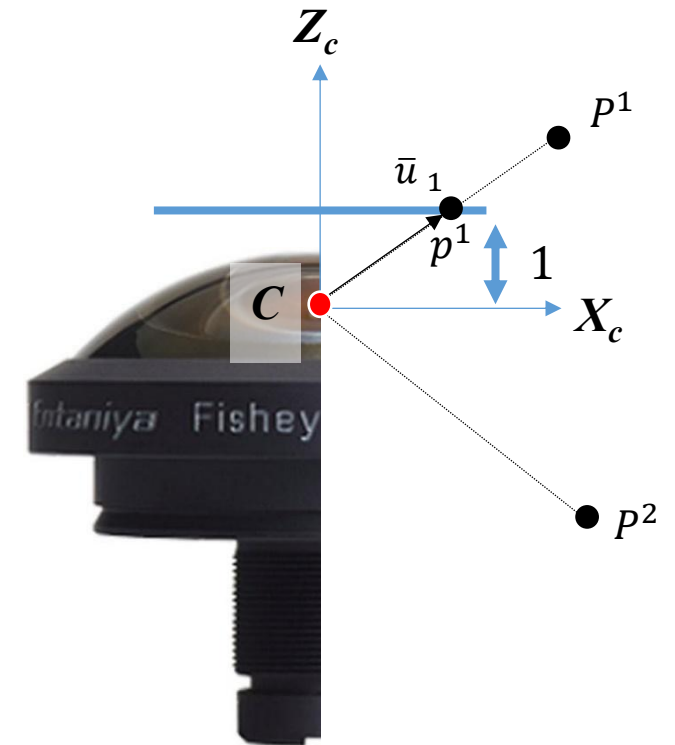
Wide FOV dioptric cameras (e.g. fisheye)



The red line is the x-y plane passing through C



$$\lambda \begin{bmatrix} \frac{u - u_0}{\alpha} \\ \frac{v - v_0}{\alpha} \\ 1 \end{bmatrix} = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$



# Unified Omnidirectional Camera Model (for Fisheye and Catadioptric cameras)

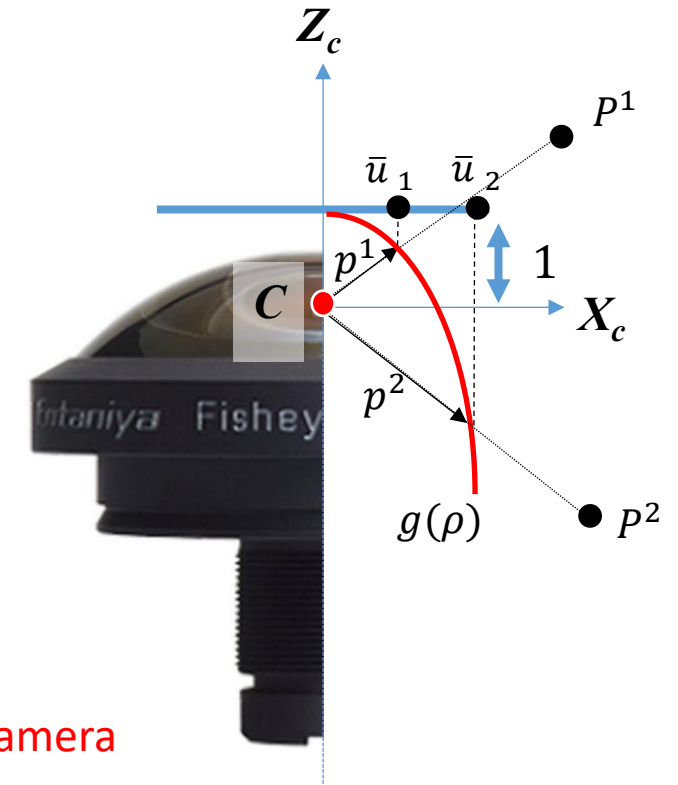
- We model the focal length as polynomial function, whose coefficients are the parameters to be estimated
- The coefficients of the polynomial, the intrinsic parameters, and extrinsics are then found via DLT

$$\lambda \begin{bmatrix} \frac{u - u_0}{\alpha} \\ \frac{v - v_0}{\alpha} \\ g(\rho) \end{bmatrix} = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

$$g(\rho) = 1 + a_1\rho + a_2\rho^2 + \dots + a_N\rho^N$$

$$\rho = \sqrt{(u - u_0)^2 + (v - v_0)^2}$$

When  $a_1, a_2, \dots, a_N = 0$  then we get a pinhole camera

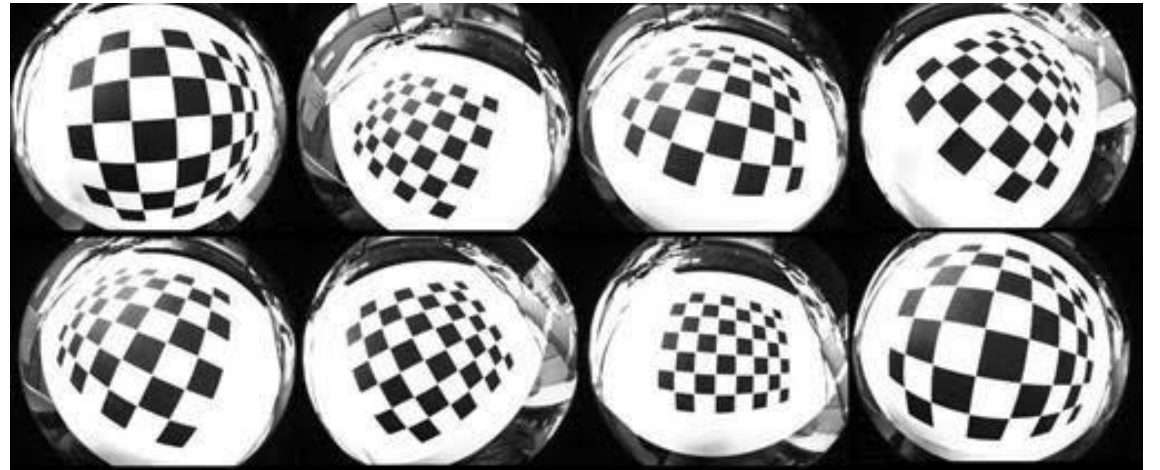


# OCamCalib: Omnidirectional Camera Calibration Toolbox

- Released in 2006, [OCamCalib](#) is the standard toolbox for calibrating wide angle cameras (fisheye and catadioptric)
- Since 2015, included in the [Matlab Computer Vision Toolbox](#)



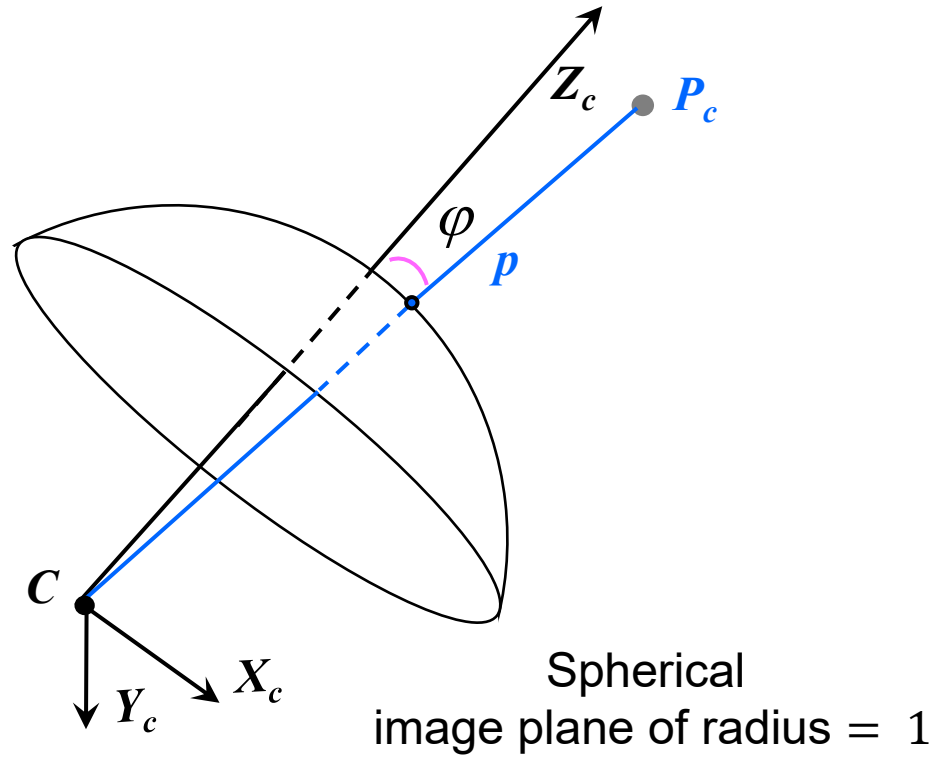
Example calibration images of a catadioptric camera



Example calibration images of a fisheye camera

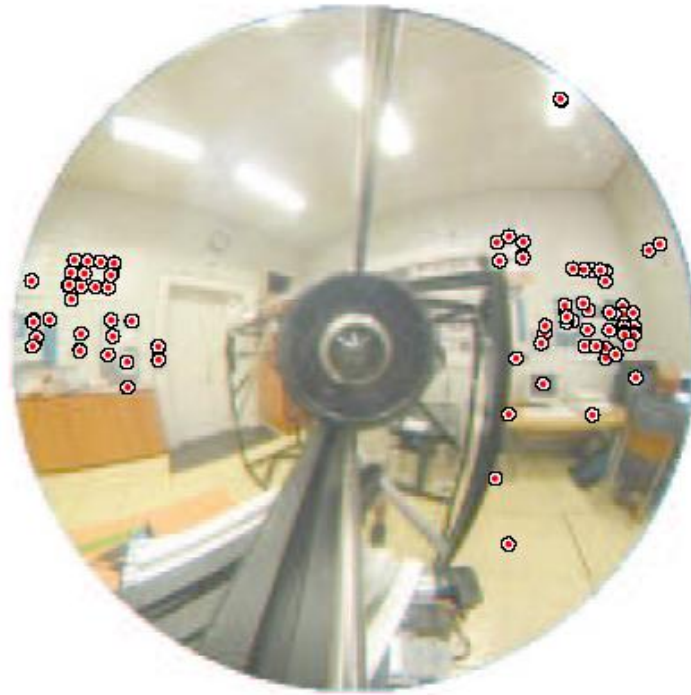
# Projection of Image Points on the Unit Sphere

- Always possible after the camera has been calibrated

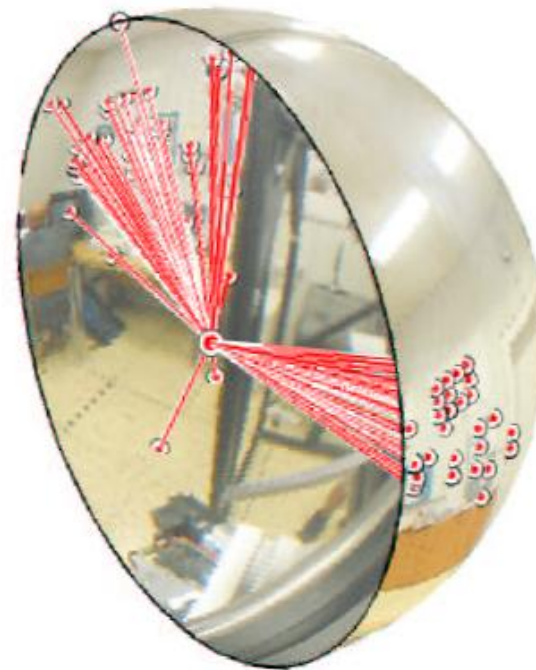


# Projection of Image Points on the Unit Sphere

- Always possible after the camera has been calibrated



Points



Rays

# Summary (things to remember)

- Calibration from 3D objects: DLT algorithm
- Calibration from planar grids: DLT algorithm using homography projection
- Reprojection Error and non linear optimization
- P3P algorithm
- DLT vs EPNP comparison
- Omnidirectional cameras
  - Central vs non central projection
  - Unified (spherical) model for perspective and omnidirectional cameras

# Readings

- Ch. 2.1 of Szeliski book, 2<sup>nd</sup> Edition
- Chapter 4 of Autonomous Mobile Robots book: [link](#)

# Understanding Check

Are you able to:

- Describe the differences between Tsai's and Zhang's calibration methods
- Explain and derive the DLT in both Tsai's and Zhang's methods? What is the minimum number of point correspondences they require?
- Describe the general PnP problem and derive the behavior of its solutions?
- Explain the working principle of the P3P algorithm? Why do we need 4 points? What's the key difference between P3P and EPnP?
- What is the reprojection error and how is it used for refining the calibration?
- What are the key technical differences between DLT and EPnP their differences in terms of robustness to noise, number of points, and computational efficiency?
- Prove mathematically that the uncertainty of the distance estimated by PnP increases quadratically with the distance.
- Define central and noncentral omnidirectional cameras?
- What kind of mirrors ensure central projection?

# Appendix: How to recover $K, R, T$ from $H$ and from multiple views?

Won't be asked  
at the exam  
😊

1. Estimate the homography  $H_i$  for each  $i$ -th view using the DLT algorithm.
2. Determine the intrinsics  $K$  of the camera from a set of homographies:
  1. Each homography  $H_i \sim K(\mathbf{r}_1, \mathbf{r}_2, \mathbf{t})$  provides two *linear* equations in the 6 entries of the matrix  $B := K^{-\top} K^{-1}$ . Letting  $\mathbf{w}_1 := K\mathbf{r}_1$ ,  $\mathbf{w}_2 := K\mathbf{r}_2$ , the rotation constraints  $\mathbf{r}_1^\top \mathbf{r}_1 = \mathbf{r}_2^\top \mathbf{r}_2 = 1$  and  $\mathbf{r}_1^\top \mathbf{r}_2 = 0$  become  $\mathbf{w}_1^\top B \mathbf{w}_1 - \mathbf{w}_2^\top B \mathbf{w}_2 = 0$  and  $\mathbf{w}_1^\top B \mathbf{w}_2 = 0$ .
  2. Stack  $2N$  equations from  $N$  views, to yield a linear system  $A\mathbf{b} = \mathbf{0}$ . Solve for  $\mathbf{b}$  (i.e.,  $B$ ) using the Singular Value Decomposition (SVD).
  3. Use Cholesky decomposition to obtain  $K$  from  $B$ .
3. The extrinsic parameters for each view can be computed using  $K$ :  
 $\mathbf{r}_1 \sim \lambda K^{-1} H_i(:, 1)$ ,  $\mathbf{r}_2 \sim \lambda K^{-1} H_i(:, 2)$ ,  $\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$  and  $T_i = \lambda K^{-1} H_i(:, 3)$ , with  $\lambda = 1/K^{-1} H_i(:, 1)$ .  
Finally, build  $R_i = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)$  and enforce rotation matrix constraints.