

# Practical Course: Vision-based Navigation

## Winter Semester 2019

### Lecture 2. Camera Models and Optimization

Vladyslav Usenko, Nikolaus Demmel,  
Prof. Dr. Daniel Cremers

# Contents

- Camera Intrinsic and Extrinsic
- From State Estimation to Least Squares
- Batch Least Square
- Application: Camera Calibration

# Contents

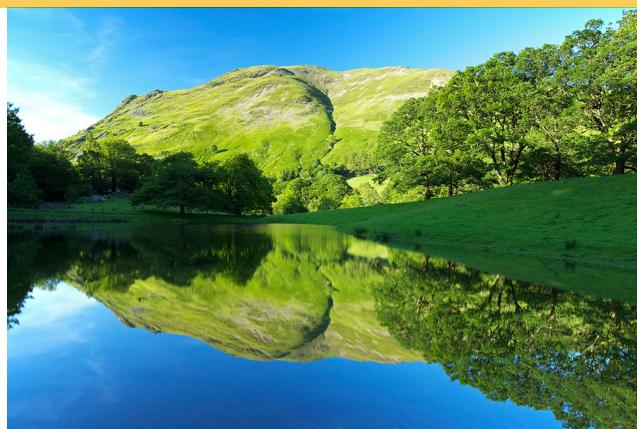
- Camera Intrinsic and Extrinsic
- From State Estimation to Least Squares
- Batch Least Square
- Application: Camera Calibration

# 1. Camera intrinsic and extrinsic

- Go back to the first page:

$$\begin{cases} \boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{w}_k) & \text{Motion model} \\ \boldsymbol{z}_{k,j} = h(\boldsymbol{y}_j, \boldsymbol{x}_k, \boldsymbol{v}_{k,j}) & \text{Observation model} \end{cases} .$$

- Cameras give you the images of the world
- How are these pixels projected from the 3D environment?



# 1. Camera intrinsic and extrinsic

## ■ Pinhole camera

By similar triangles:

$$\frac{Z}{f} = -\frac{X}{X'} = -\frac{Y}{Y'}.$$

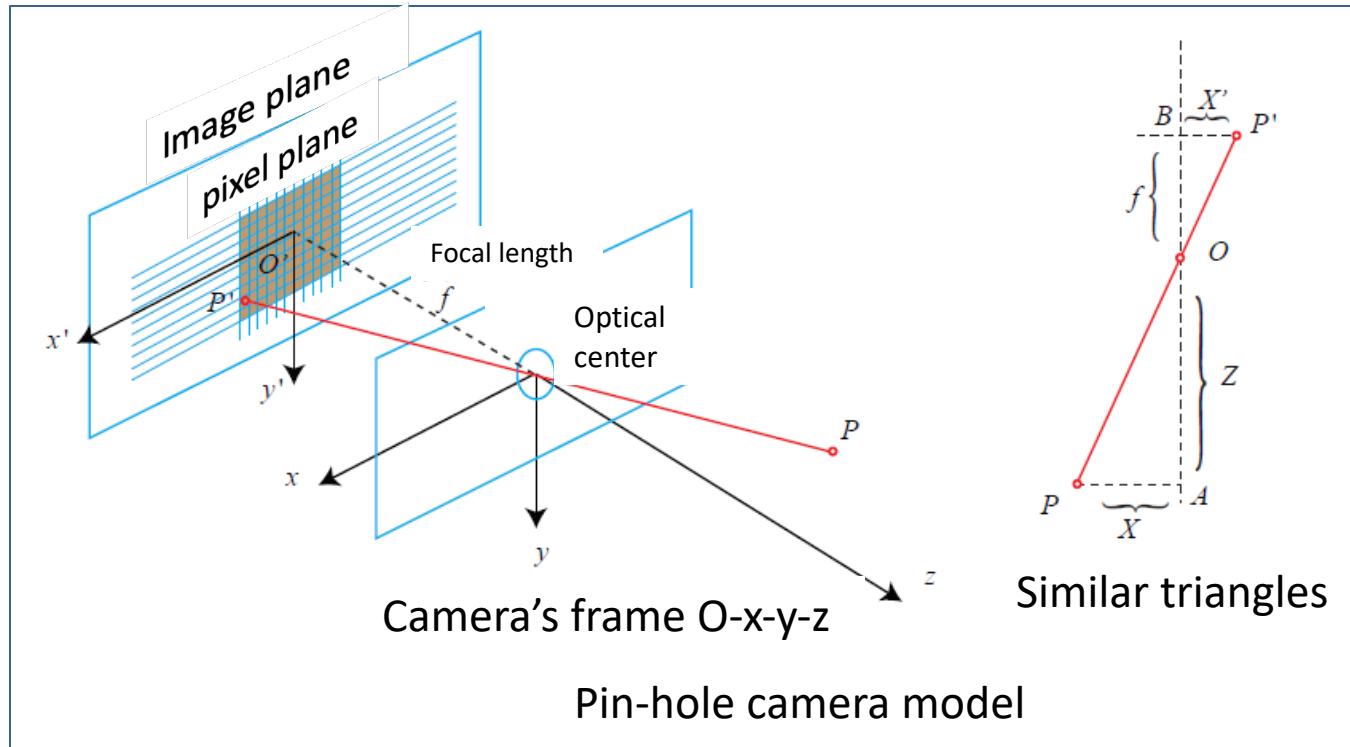
Flip to the front:

$$\frac{Z}{f} = \frac{X}{X'} = \frac{Y}{Y'}.$$

Rearrange it:

$$X' = f \frac{X}{Z}$$

$$Y' = f \frac{Y}{Z}$$



# 1. Camera intrinsic and extrinsic

## ■ Pinhole cameras

From image plane  
to pixels:

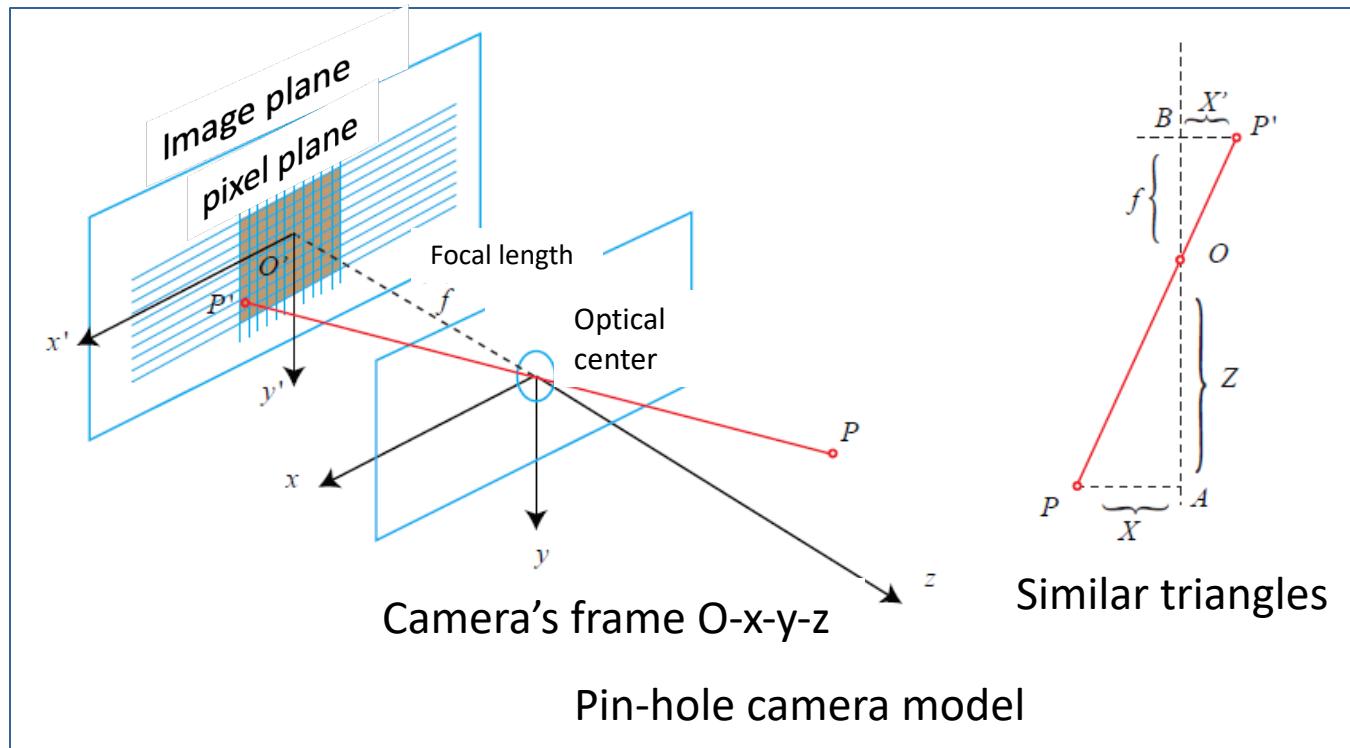
$$\begin{cases} u = \alpha X' + c_x \\ v = \beta Y' + c_y \end{cases}$$

Take into:

$$\begin{aligned} X' &= f \frac{X}{Z} \\ Y' &= f \frac{Y}{Z} \end{aligned}$$

Then we get:

$$\begin{cases} u = f_x \frac{X}{Z} + c_x \\ v = f_y \frac{Y}{Z} + c_y \end{cases}$$



# 1. Camera intrinsic and extrinsic

- Pinhole models:

$$\begin{cases} u = f_x \frac{X}{Z} + c_x \\ v = f_y \frac{Y}{Z} + c_y \end{cases}.$$

- Matrix form:

Put Z to left:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \frac{1}{Z} \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \triangleq \frac{1}{Z} \mathbf{K} \mathbf{P}.$$

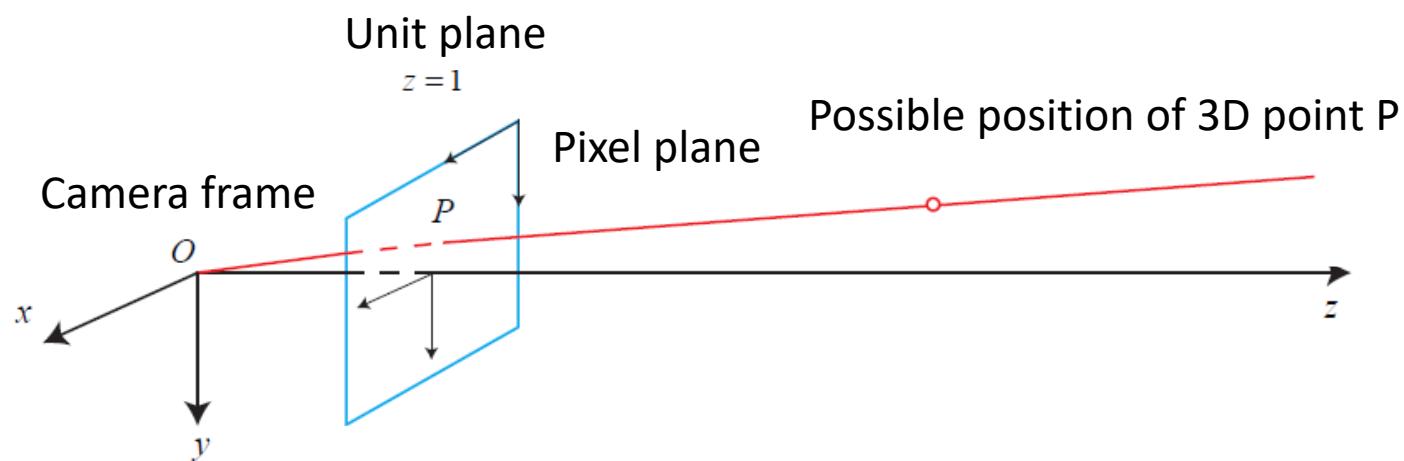
$$Z \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \triangleq \mathbf{K} \mathbf{P}.$$

- K is called as **intrinsic camera matrix**
  - Which is **fixed** for each real camera
  - And can be **calibrated before running slam.**

# 1. Camera intrinsic and extrinsic

- Distance is lost during the projection

$$Z \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \triangleq \mathbf{K}\mathbf{P}.$$



# 1. Camera intrinsic and extrinsic

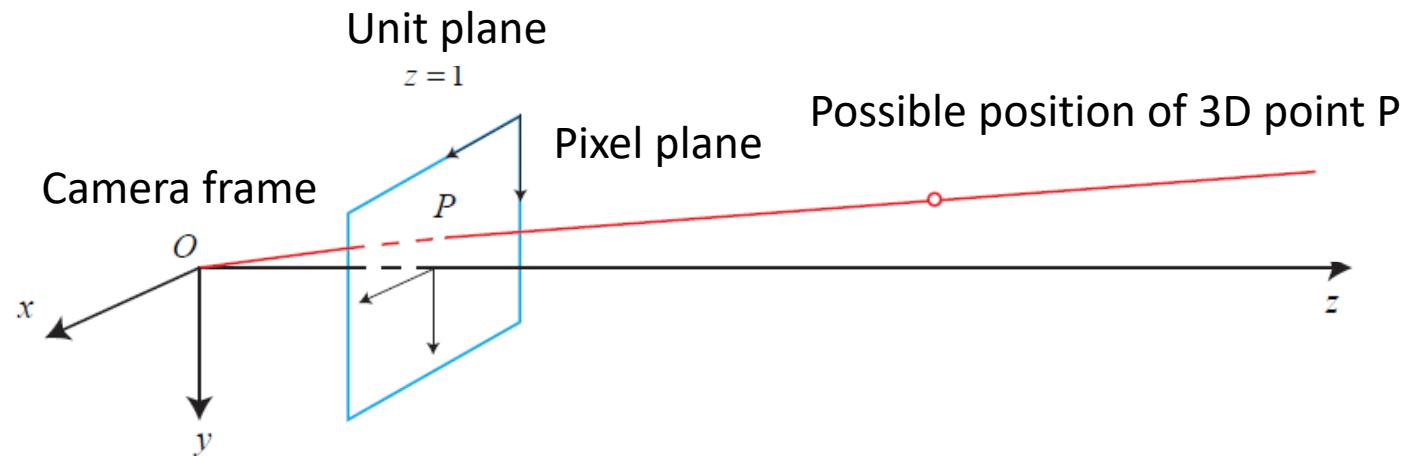
- There's another rotation and translation from the world to the camera

$$ZP_{uv} = Z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K(RP_w + t) = KTP_w.$$

- Here R,t or T is called as extrinsic
  - Note we assume the homogeneous coordinates are cast to non-homogeneous coordinates automatically
  - In SLAM, the extrinsic R,t is our estimate purpose

# 1. Camera intrinsic and extrinsic

- Summary
  - Projection orders: world->camera->unit plane->pixels



# 1. Camera intrinsic and extrinsic

- Distortion
  - Lens will cause distortion when you have a wide range lens



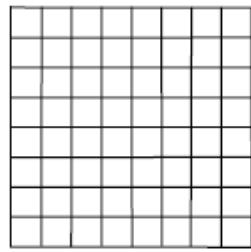
Wide range lens



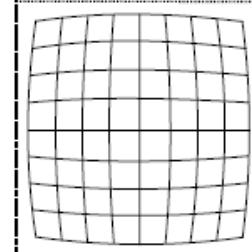
Fisheye cameras

# 1. Camera intrinsic and extrinsic

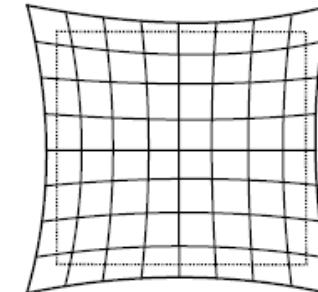
- Distortion types: radial distortion and tangential distortion



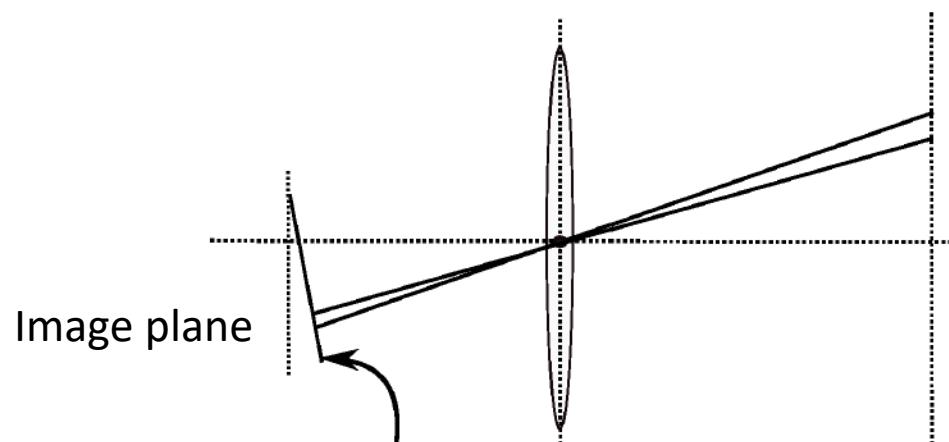
Original image



Barrel distortion



Pincushion distortion



# 1. Camera intrinsic and extrinsic Distortion

- Mathematic form

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

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

Radial distortion

$$x_{\text{distorted}} = x + 2p_1 xy + p_2(r^2 + 2x^2)$$

$$y_{\text{distorted}} = y + p_1(r^2 + 2y^2) + 2p_2 xy$$

tangential distortion

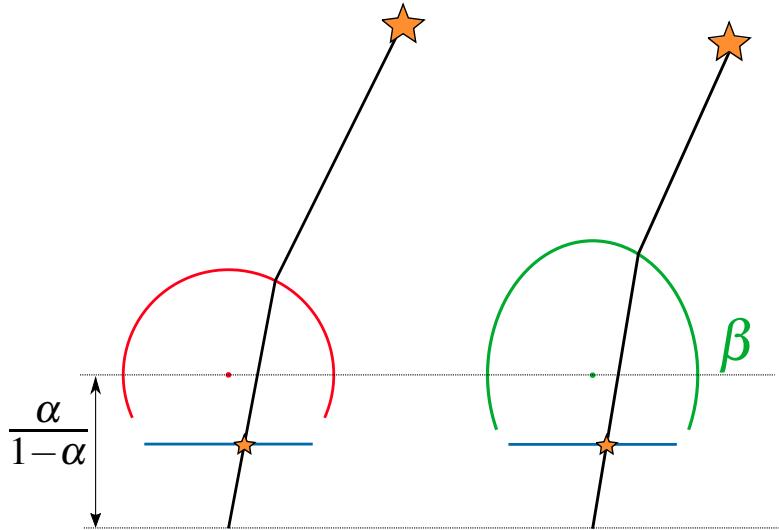
- Put them together

$$x_{\text{distorted}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 xy + p_2(r^2 + 2x^2)$$

$$y_{\text{distorted}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y^2) + 2p_2 xy$$

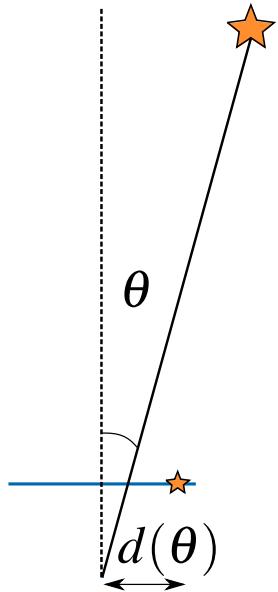
- In practice, you can choose the order of distortion params

# 1. Camera intrinsic and extrinsic: (Extended) Unified Camera Models



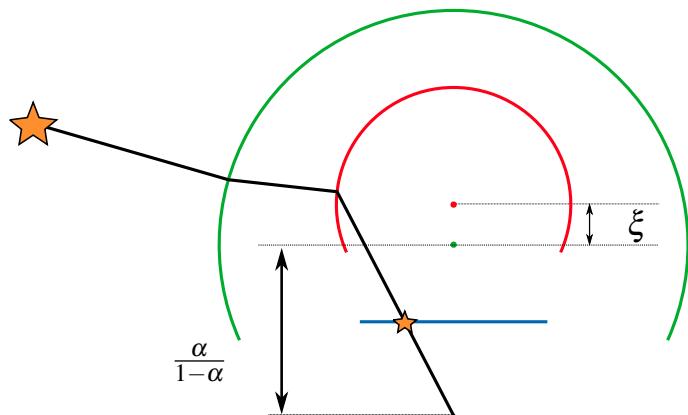
$$\mathbf{i} = [f_x, f_y, c_x, c_y, \alpha, \beta]^T, \alpha \in [0, 1], \beta > 0$$
$$\pi(\mathbf{x}, \mathbf{i}) = \begin{bmatrix} f_x \frac{x}{\alpha d + (1-\alpha)z} \\ f_y \frac{y}{\alpha d + (1-\alpha)z} \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix},$$
$$d = \sqrt{\beta(x^2 + y^2) + z^2}.$$

# 1. Camera intrinsic and extrinsic: Kannala-Brandt Model



$$\mathbf{i} = [f_x, f_y, c_x, c_y, k_1, k_2, k_3, k_4]^T$$
$$\pi(\mathbf{x}, \mathbf{i}) = \begin{bmatrix} f_x & d(\theta) & \frac{x}{r} \\ f_y & d(\theta) & \frac{y}{r} \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix},$$
$$r = \sqrt{x^2 + y^2}, \theta = \text{atan2}(r, z),$$
$$d(\theta) = \theta + k_1\theta^3 + k_2\theta^5 + k_3\theta^7 + k_4\theta^9$$

# 1. Camera intrinsic and extrinsic: Double Sphere Camera Model



$$\pi(\mathbf{x}, \mathbf{i}) = \begin{bmatrix} f_x \frac{x}{\alpha d_2 + (1-\alpha)(\xi d_1 + z)} \\ f_y \frac{y}{\alpha d_2 + (1-\alpha)(\xi d_1 + z)} \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix},$$
$$d_1 = \sqrt{x^2 + y^2 + z^2},$$
$$d_2 = \sqrt{x^2 + y^2 + (\xi d_1 + z)^2},$$

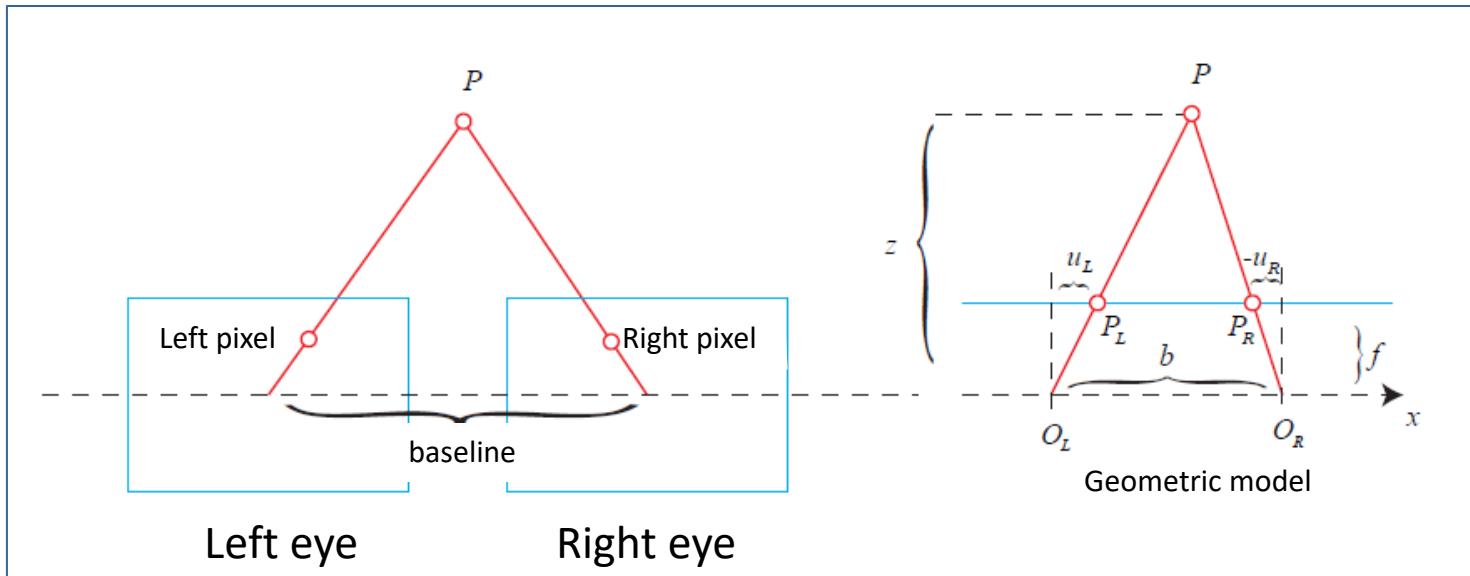
More info:

Vladyslav Usenko, Nikolaus Demmel, and Daniel Cremers. “The Double Sphere Camera Model”. In: *Proc. of the Int. Conference on 3D Vision (3DV)*. Sept. 2018. eprint: <http://arxiv.org/abs/1807.08957>.

# 1. Camera intrinsic and extrinsic

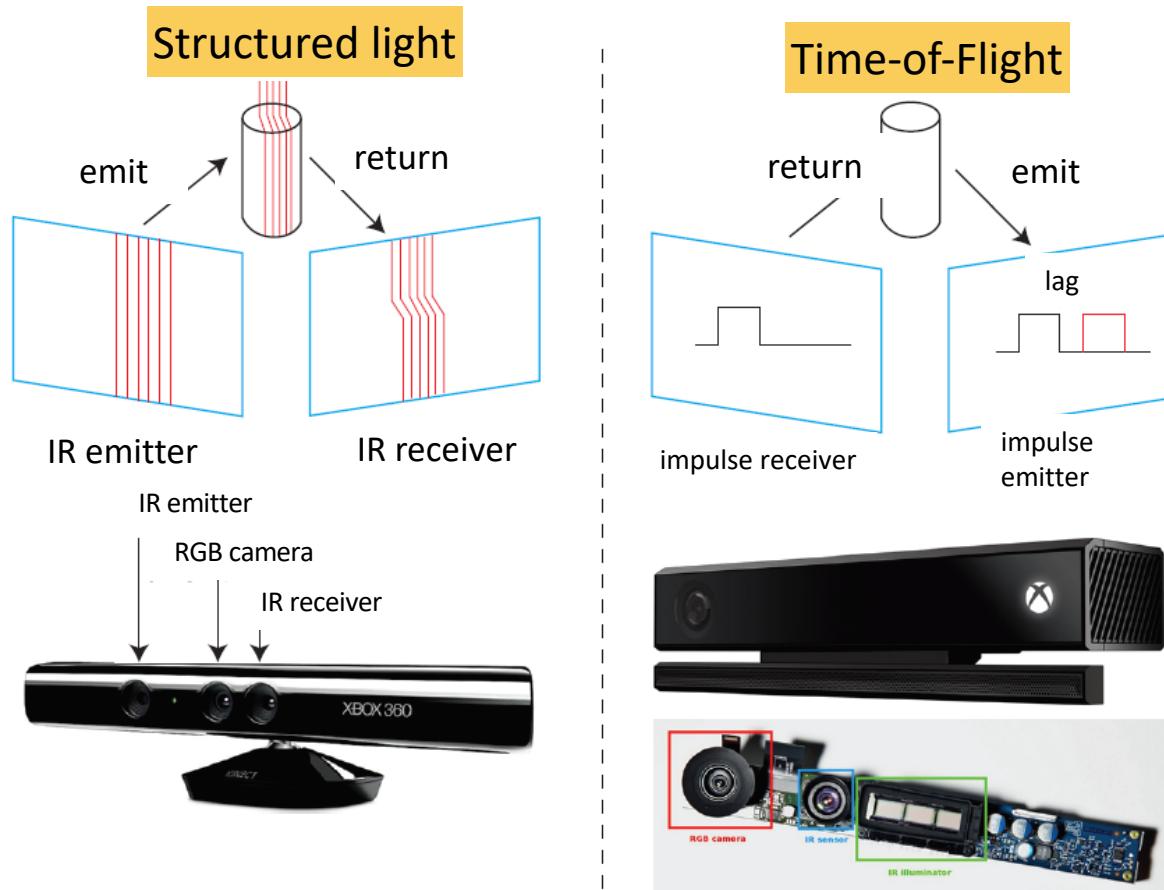
## Stereo camera

- Two cameras (usually) placed horizontally



- The distance between left camera center to the right is called as baseline
- From geometric model:  $\frac{z - f}{z} = \frac{b - u_L + u_R}{b} \Rightarrow z = \frac{fb}{d}, \quad d = u_L - u_R.$

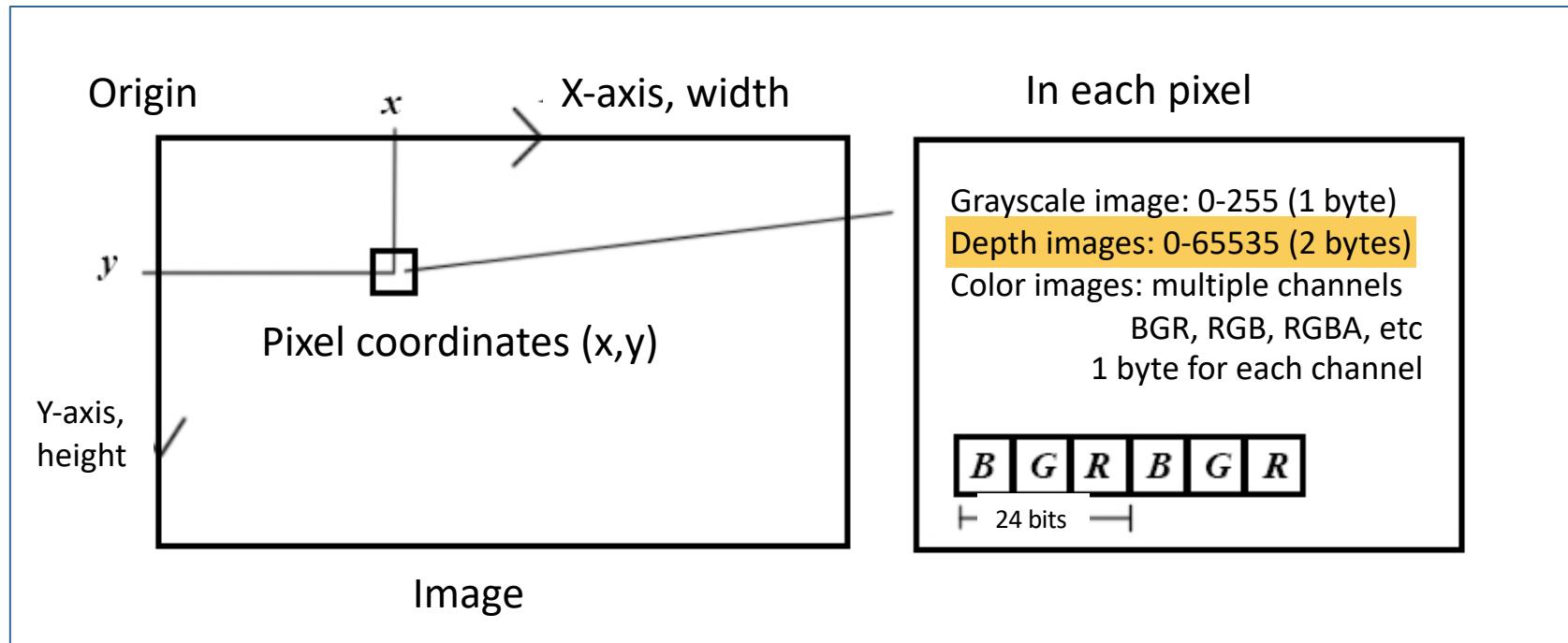
# 1. Camera intrinsic and extrinsic



RGB-D cameras

# 1. Camera intrinsic and extrinsic

- Images
- 2D arrays stored in computer
- Usually 0-255 (1 byte) grayscale values after quantification



# Contents

- Camera Intrinsic and Extrinsic
- From State Estimation to Least Squares
- Batch Least Square
- Application: Camera Calibration

## 2. From state estimation to least square

- Recall the motion model and observation model

$$\begin{cases} \boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{w}_k) \\ \boldsymbol{z}_{k,j} = h(\boldsymbol{y}_j, \boldsymbol{x}_k, \boldsymbol{v}_{k,j}) \end{cases}.$$

- How to estimate the unknown variables given the observation data?

## 2. Batch state estimation

- Batch approach

- Give all the measurements
- To estimate all the state variables

- State variables:

$$x = \{x_1, \dots, x_N, y_1, \dots, y_M\}.$$

Observation and input:

$$u = \{u_1, u_2, \dots\}, z = \{z_{k,j}\}$$

- Our purpose:

$$P(x|z, u).$$

- Bayes' Rule:

$$p(x|u, z) = \frac{P(z|x, u)p(x|u)}{P(z|u)}$$

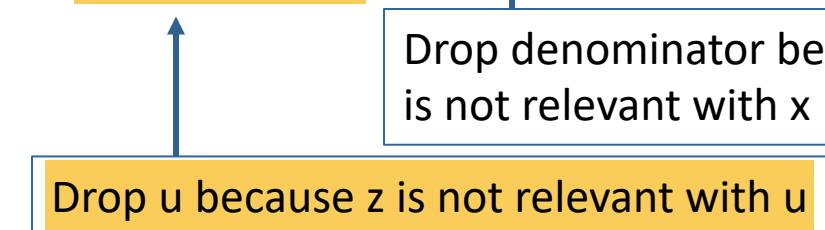
Posteriori

## 2. From state estimation to least square

- It is usually hard to write out the full distribution of Bayes' formula, but we can:
- MAP: Maximum A Posteriori

$$x_{MAP} = \operatorname{argmax}_x P(x|u, z) = \operatorname{argmax} \frac{P(z|x, u)P(x|u)}{P(z|u)}$$

$$= \operatorname{argmax} P(z|x)P(x|u)$$



- “In which state it is most likely to produce such measurements”

## 2. From state estimation to least square

- From MAP to batch least square
- We assume the noise variables are independent, so that the joint pdf can be factorized:

$$P(z|x) = \prod_{k=0}^K P(z_k|x_k)$$

- Let's consider a single observation:
  - Affected by white Gaussian noise:
- The observation model gives us a conditional pdf:
$$P(z_{j,k}|x_k, y_j) = N(h(y_j, x_k), Q_{k,j})$$
- Then how to compute the MAP of x,y given z?

## 2. From state estimation to least square

- Gaussian distribution (matrix form)

$$P(x) = \frac{1}{\sqrt{(2\pi)^N \det(\Sigma)}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right).$$

- Take minus logarithm at both sides:

$$-\ln(P(x)) = \frac{1}{2} \ln((2\pi)^N \det(\Sigma)) + \frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu).$$

Constant w.r.t x

Mahalanobis distance (sigma-norm)

- Maximum of  $P(x)$  is equivalent to minimum of  $-\ln(P(x))$

## 2. From state estimation to least square

- Take this into the MAP:

$$\text{Max: } P(z_{j,k}|x_k, y_j) = N(h(y_j, x_k), Q_{k,j}).$$

$$\rightarrow x_k, y_j = \operatorname{argmin} \left( (z_{k,j} - h(y_j, x_k))^T Q_{j,k}^{-1} (z_{k,j} - h(y_j, x_k)) \right)$$

Error or residual of single observation

Information matrix

- We turn a MAP problem into a least square problem

## 2. From state estimation to least square

- Batch least square
- Original problem

Least square

Define the errors(residuals)

$$\begin{cases} \mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k, \mathbf{w}_k) \\ \mathbf{z}_{k,j} = h(\mathbf{y}_j, \mathbf{x}_k, \mathbf{v}_{k,j}) \end{cases}.$$

$$\begin{aligned} \mathbf{e}_{v,k} &= \mathbf{x}_k - f(\mathbf{x}_{k-1}, \mathbf{u}_k) \\ \mathbf{e}_{y,j,k} &= \mathbf{z}_{k,j} - h(\mathbf{x}_k, \mathbf{y}_j), \end{aligned}$$

$$x_{MAP} = \text{argmax} P(z|x)P(x|u)$$

- Sum of the squared residuals:

min

$$J(\mathbf{x}) = \sum_k \mathbf{e}_{v,k}^T \mathbf{R}_k^{-1} \mathbf{e}_{v,k} + \sum_k \sum_j \mathbf{e}_{y,j,k}^T \mathbf{Q}_{k,j}^{-1} \mathbf{e}_{y,j,k}.$$

## 2. From state estimation to least square

$$J(x) = \sum_k e_{v,k}^T R_k^{-1} e_{v,k} + \sum_k \sum_j e_{y,k,j}^T Q_{k,j}^{-1} e_{y,k,j}$$

- Some notes:

- Because of noise, when we take the estimated trajectory and map into the models, they won't fit perfectly
- Then we adjust our estimation to get a better estimation (minimize the error)
- The error distribution is affected by noise distribution (information matrix)

- Structure of the least square problem

- Sum of many squared errors
- The dimension of total state variable maybe high
- But single error item is easy (only related to two states in our case)
- If we use Lie group and Lie algebra, then it's a non-constrained least square

# Contents

- Camera Intrinsic and Extrinsic
- From State Estimation to Least Squares
- Batch Least Square
- Application: Camera Calibration

### 3. Batch least square

- How to solve a least square problem?
  - Non-linear, discrete time, non-constrained
- Let's start from a simple example

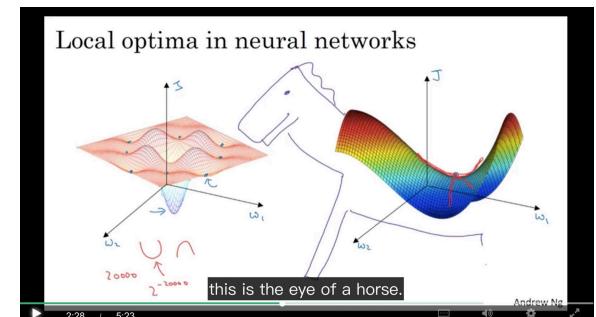
- Consider minimizing a squared error:
- When J is simple, just solve:

$$\frac{dJ}{dx} = 0$$

$$\min J(x) = \min \frac{1}{2} \|f(x)\|_2^2$$

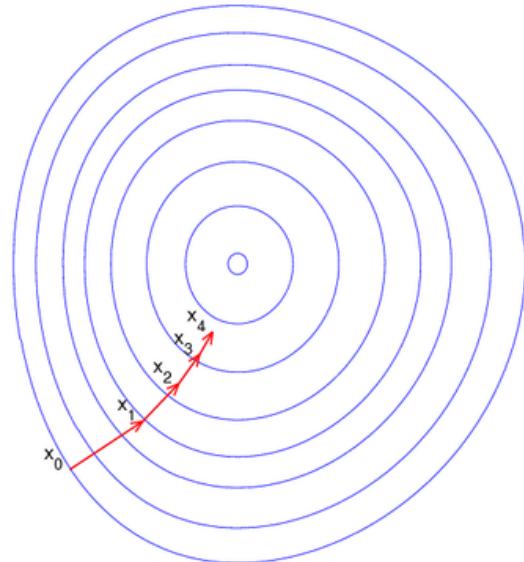
$$x \in \mathbb{R}^n$$

- And we will find the maxima/minima/saddle points



### 3. Batch least square

- When  $J$  is a complicated function:
  - $dJ/dx=0$  is hard to solve
  - We use **iterative methods**
- Iterative methods
  1. Start from a initial estimation  $x_0$
  2. At iteration  $k$ , we find a incremental  $\Delta x_k$  to make  $\|f(x_k + \Delta x_k)\|_2^2$  become smaller
  3. If  $\Delta x_k$  is small enough, stop (converged)
  4. If not, set  $x_{k+1} = x_k + \Delta x_k$  and return to step 2.



### 3. Batch least square

- How to find the incremental part?
- By the gradient
- Taylor expansion of the object function:

$$\|f(x + \Delta x)\|_2^2 \approx \|f(x)\|_2^2 + J(x) \Delta x + \frac{1}{2} \Delta x^T H \Delta x.$$

Jacobian

Hessian

- First order methods and second order methods
- First order: (Steepest descent)

$$\min_{\Delta x} \|f(x)\|_2^2 + J \Delta x$$

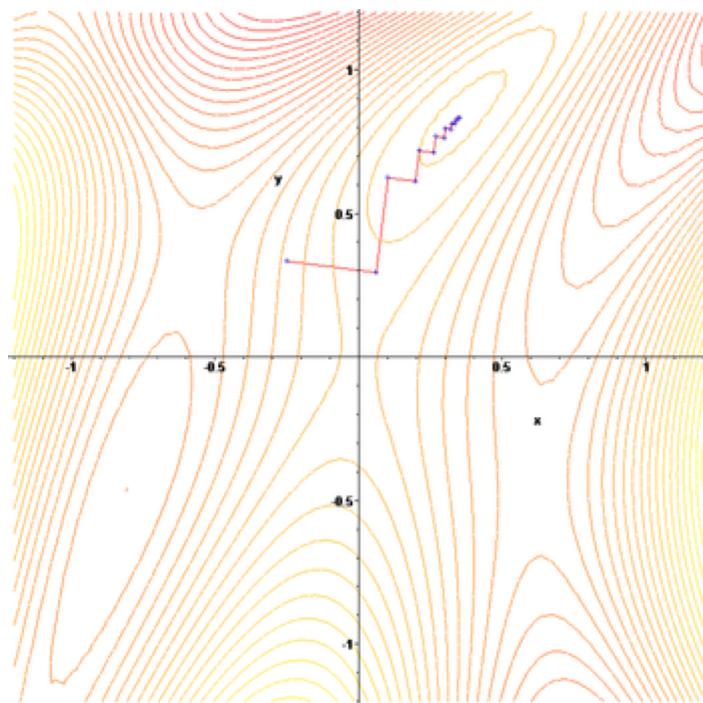
Incremental will be:

$$\Delta x^* = -J^T(x).$$

Usually we need a step size

### 3. Batch least square

- Zig-zag in steepest descent



Other shortcomings

- Slow convergence speed
- Slow when close to the minimum

### 3. Batch least square

- Second order methods

$$\|f(x + \Delta x)\|_2^2 \approx \|f(x)\|_2^2 + J(x) \Delta x + \frac{1}{2} \Delta x^T H \Delta x.$$

- Solve an increment to minimize it:

$$\Delta x^* = \arg \min \|f(x)\|_2^2 + J(x) \Delta x + \frac{1}{2} \Delta x^T H \Delta x.$$

- Let the derivative to  $\Delta x$  be zero, then we get:  $H \Delta x = -J^T$ .
- This is called Newton's method

### 3. Batch least square

- Second order method converges more quickly than first order methods
- But the Hessian matrix maybe hard to compute:  $H\Delta x = -J^T$ .
- Can we avoid the Hessian matrix and also keep second order's convergence speed?
  - Gauss-Newton
  - Levenberg-Marquardt

### 3. Batch least square

#### ■ Gauss-Newton

- Taylor expansion of  $f(x)$ : 
$$f(x + \Delta x) \approx f(x) + J(x) \Delta x.$$
- Then the squared error becomes:

$$\begin{aligned}\frac{1}{2} \|f(x) + J(x) \Delta x\|^2 &= \frac{1}{2} (f(x) + J(x) \Delta x)^T (f(x) + J(x) \Delta x) \\ &= \frac{1}{2} \left( \|f(x)\|_2^2 + 2f(x)^T J(x) \Delta x + \Delta x^T J(x)^T J(x) \Delta x \right).\end{aligned}$$

- Also let its derivative with  $\Delta x$  be zero:

$$2J(x)^T f(x) + 2J(x)^T J(x) \Delta x = 0.$$

$$\underline{J(x)^T J(x) \Delta x} = -\underline{J(x)^T f(x)}.$$

$H$

$g$

$\underline{H \Delta x = g.}$

### 3. Batch least square

$$\mathbf{J}(x)^T \mathbf{J}(x) \Delta x = -\mathbf{J}(x)^T f(x).$$

- Gauss-Newton use  $\mathbf{J}(x)^T \mathbf{J}(x)$  as an approximation of the Hessian
  - Therefore avoiding the computation of H in the Newton's method
- But  $\mathbf{J}(x)^T \mathbf{J}(x)$  is only semi-positive definite
  - H maybe singular when  $\mathbf{J}^T \mathbf{J}$  has null space

### 3. Batch least square

- Levenberg-Marquardt method

- Trust region approach: approximation is only valid in a region
  - Evaluate if the approximation is good:

$$\rho = \frac{f(x + \Delta x) - f(x)}{J(x) \Delta x}.$$

Real descent/approx. descent

- If rho is large, increase the region
  - If rho is small, decrease the region

- LM optimization:  $\min_{\Delta x_k} \frac{1}{2} \|f(x_k) + J(x_k)\Delta x_k\|^2, s.t. \|\Delta x_k\|^2 \leq \mu$ 
  - Assume the approximation is only good within a ball

### 3. Batch least square

- Trust region problem:

$$\min_{\Delta x_k} \frac{1}{2} \|f(x_k) + J(x_k)\Delta x_k\|^2, \text{ s.t. } \|\Delta x_k\|^2 \leq \mu$$

- Expand it just like in G-N's case, the incremental will be:

$$(J(x_k)^T J(x_k) + \lambda I) \Delta x_k = g \quad \lambda(\|\Delta x_k\|^2 - \mu) = 0$$

- This  $\lambda I$  increase the semi-positive definite property of the Hessian
  - Also balancing the first-order and second-order items

### 3. Batch least square

- Other methods
  - Dog-leg method
  - Conjugate gradient method
  - Quasi-Newton's method
  - Pseudo-Newton's method
  - ...
- You can find more in optimization books if you are interested
- In SLAM, we use G-N or L-M to solve camera's motion, pixel's movement, optical-flow, etc.

### 3. Batch least square

- Problem in the Practical Assignment
- Curve fitting: find best parameters  $a, b, c$  from the observation data:

Curve function:  $y = \exp(ax^2 + bx + c) + w,$

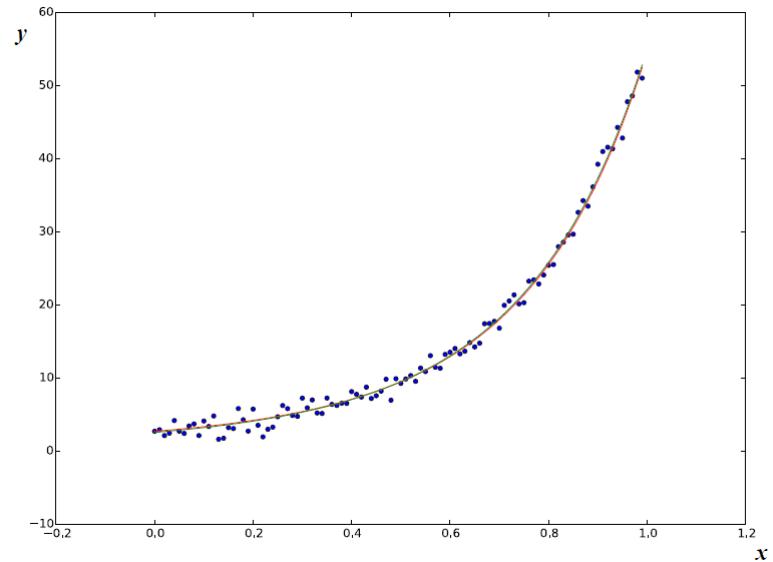
- Error:

$$e_i = y_i - \exp(ax_i^2 + bx_i + c)$$

- Least square problem:

$$a, b, c$$

$$= \operatorname{argmin} \sum_{i=1}^N \|y_i - \exp(ax_i^2 + bx_i + c)\|^2$$



### 3. Batch least square

- You are asked to solve this problem with a **ceres solver** (tutorial)
  - Google Ceres Solver <http://ceres-solver.org/>

### 3. Batch least square

- Google Ceres
  - An optimization library for solving least square problems
  - Tutorial: <http://ceres-solver.org/tutorial.html>
  - Define your residual class as a **functor** (overload the () operator)

```
struct ExponentialResidual {
    ExponentialResidual(double x, double y)
        : x_(x), y_(y) {}

    template <typename T>
    bool operator()(const T* const m, const T* const c, T* residual) const {
        residual[0] = T(y_) - exp(m[0] * T(x_) + c[0]);
        return true;
    }

private:
    // Observations for a sample.
    const double x_;
    const double y_;
};
```

### 3. Batch least square

- Build the optimization problem:

```
double m = 0.0;
double c = 0.0;

Problem problem;
for (int i = 0; i < kNumObservations; ++i) {
    CostFunction* cost_function =
        new AutoDiffCostFunction<ExponentialResidual, 1, 1, 1>(
            new ExponentialResidual(data[2 * i], data[2 * i + 1]));
    problem.AddResidualBlock(cost_function, NULL, &m, &c);
}
```

- With auto-diff, Ceres will compute the Jacobians for you

### 3. Batch least square

- Finally solve it by calling the Solve() function and get the result summary
- You can set some parameters like number of iterations, stop conditions or the linear solver type.

```
Solver::Options options;  
options.max_num_iterations = 25;  
options.linear_solver_type = ceres::DENSE_QR;  
options.minimizer_progress_to_stdout = true;
```

```
Solver::Summary summary;  
Solve(options, &problem, &summary);
```

### 3. Batch least square

#### ■ Summary

- In the batch estimation, we estimate all the status variable given all the measurements and input
- The batch estimation problem can be formulated into a least square problem, after solving it we get a MAP estimation
- The least square problem can be solved by iterative methods like gradient descent, Newton's method, Gauss-Newton or Levenberg-Marquardt method
- The least square problem can also be represented by a graph and forms a (factor) graph optimization problem

# Contents

- Camera Intrinsic and Extrinsic
- From State Estimation to Least Squares
- Batch Least Square
- Application: Camera Calibration

## 4. Application: Camera Calibration

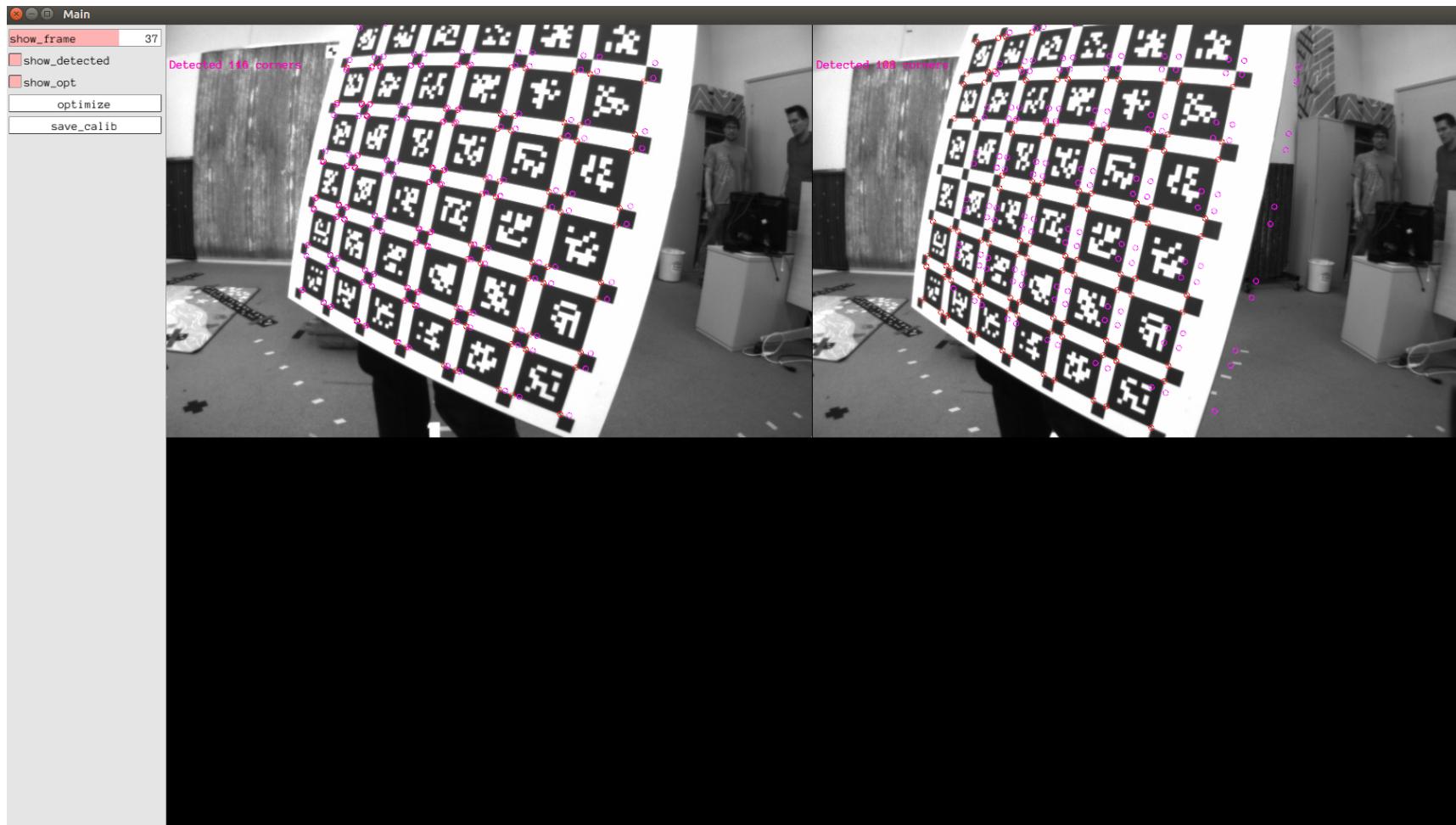
- Suppose we want to estimate the camera pose
- We have several observations from the projection function
- Minimizing the reprojection error:

$$(R, t)^* = T^* = \operatorname{argmin} \frac{1}{2} \sum_{i=1}^N \|u_i - \pi(RP_i + t)\|_2^2$$

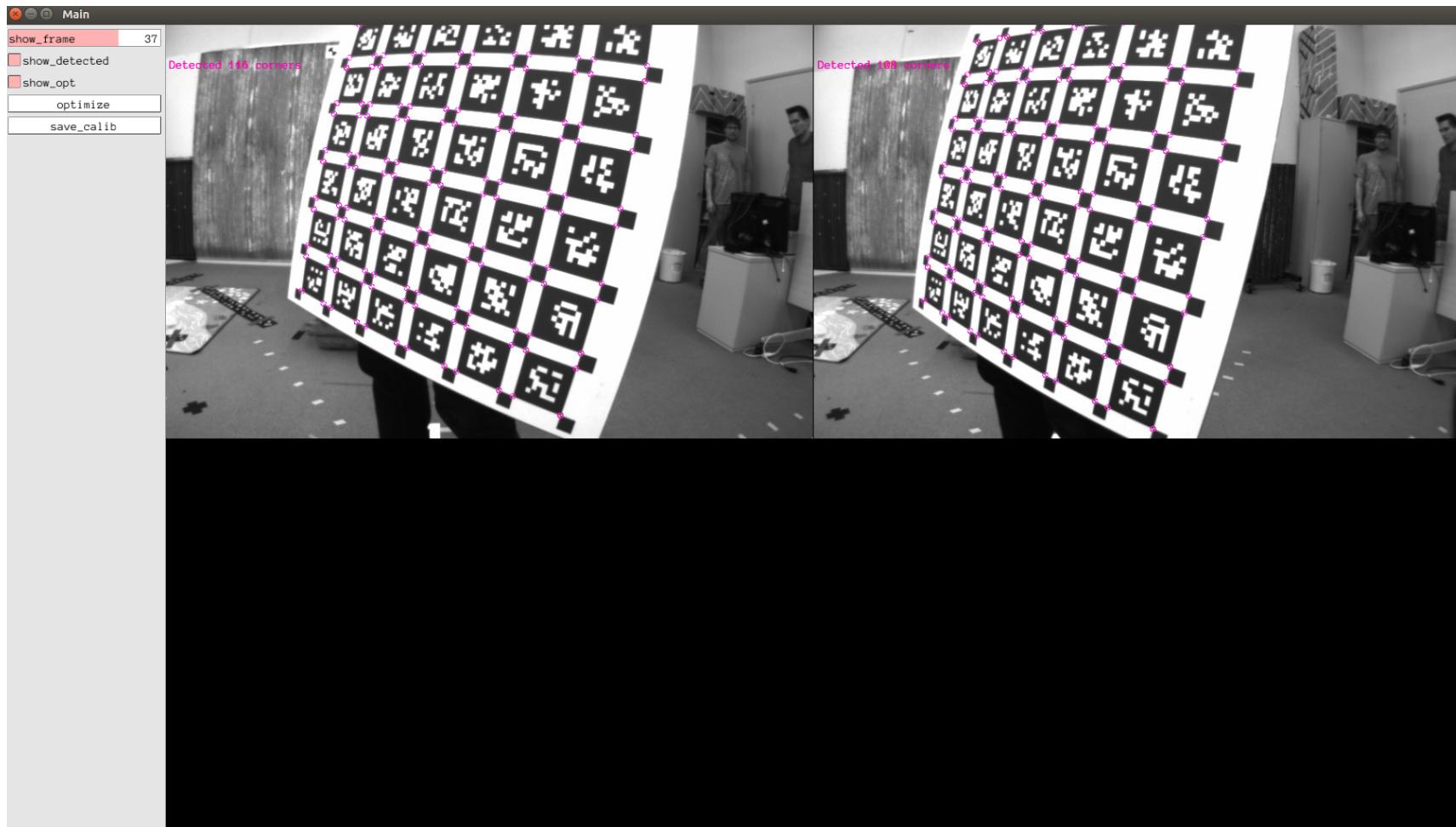
- Where  $\pi(\cdot)$  is the projection equation (observation model)
- Corner points are detected using Apriltags

E. Olson. AprilTag: A robust and flexible visual fiducial system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3400–3407. IEEE, May 2011.

# 4. Application: Camera Calibration



# 4. Application: Camera Calibration



## 4. Application: Camera Calibration

- Use camera models presented here to get initial projections
- Use optimization method to find the camera poses and intrinsic parameters
- Test different models. How well do they fit the lens?

# Questions?