Practical Course: Vision-based Navigation WS 2018/2019

Lecture 2. Camera Models and Optimization

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

Contents

Contents

1. Camera intrinsic and extrinsic

- Go back to the first page:

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

- 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:

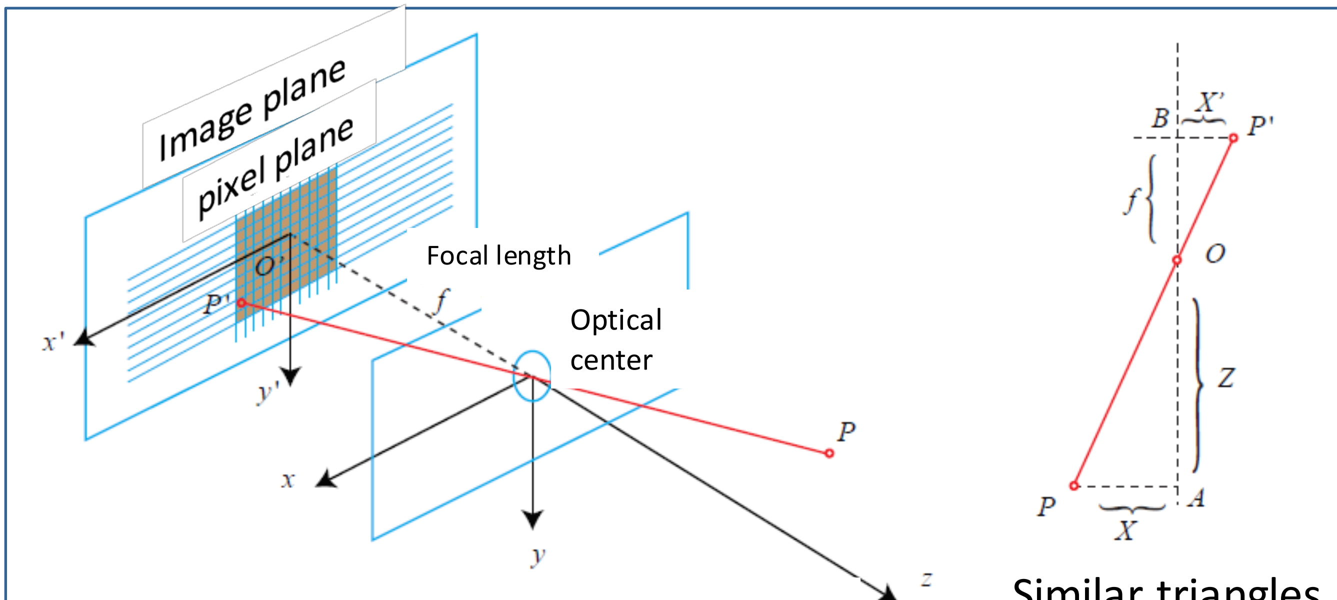
$$\frac{Z}{f} = -\frac{X}{X'} = -\frac{Y}{Y'}.$$

Flip to the front:

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

Rearrange it:

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



Camera's frame O-x-y-z

Similar triangles

Pin-hole camera model

1. Camera intrinsic and extrinsic

- Pinhole cameras

From image plane to pixels:

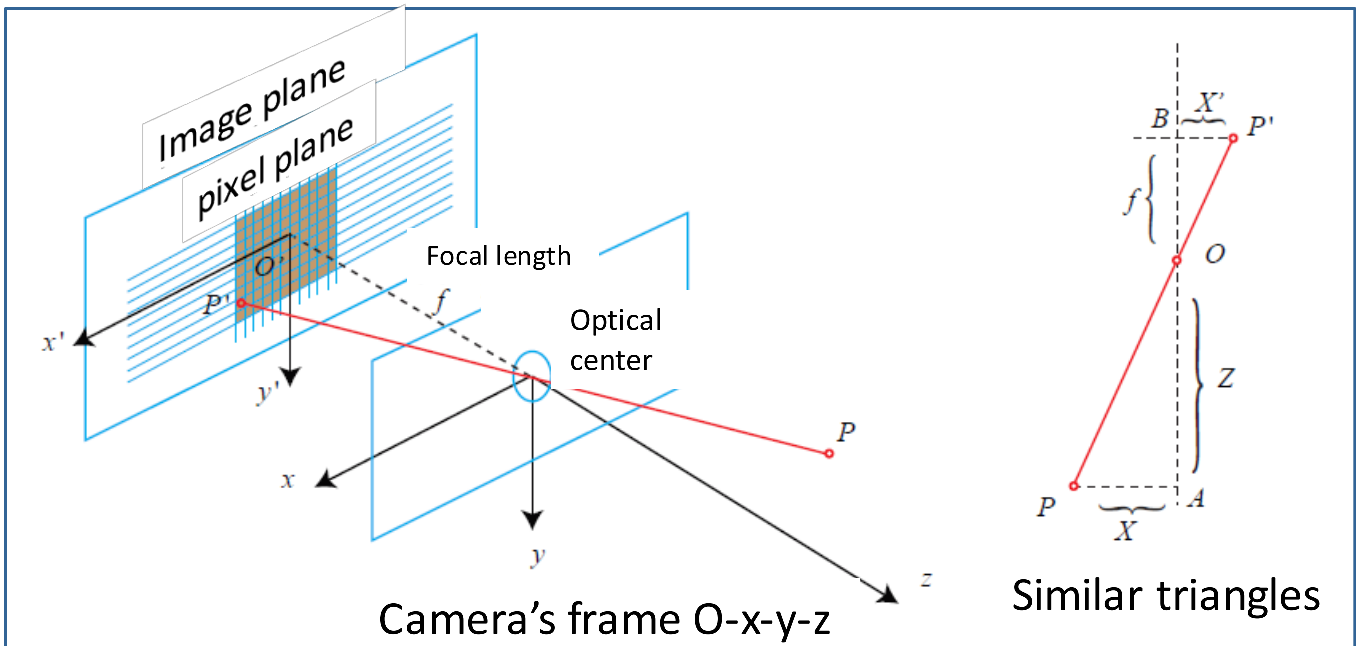
$$\begin{cases} u = \alpha X' + c_x \\ v = \beta Y' + c_y \end{cases}.$$

Take into:

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

Then we get:

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



Pin-hole camera model

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:

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

Put Z to left:

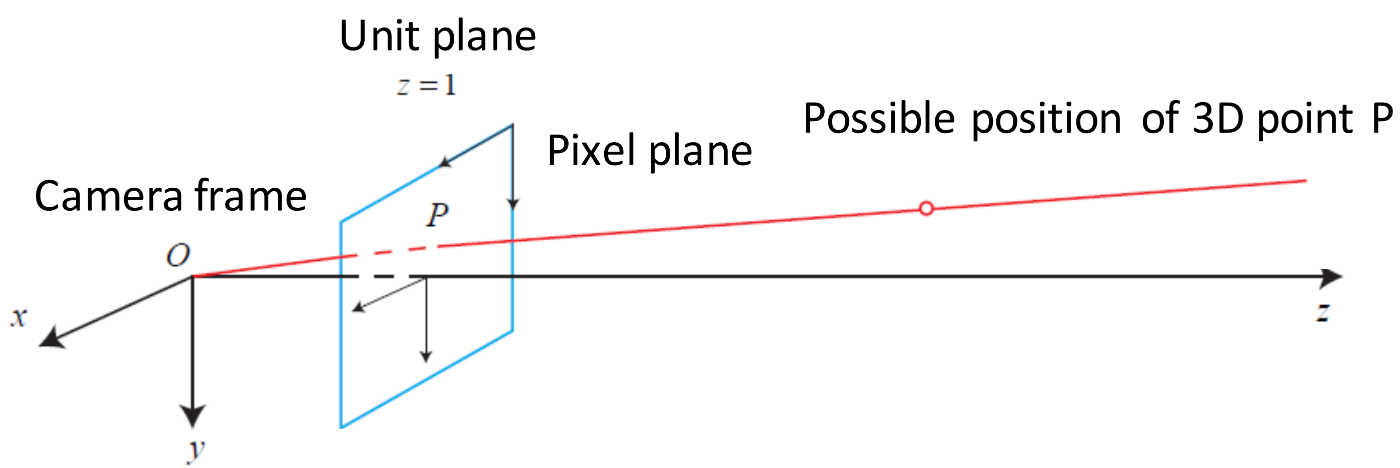
$$Z \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \triangleq \boldsymbol{KP}.$$

- 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

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



1. Camera intrinsic and extrinsic

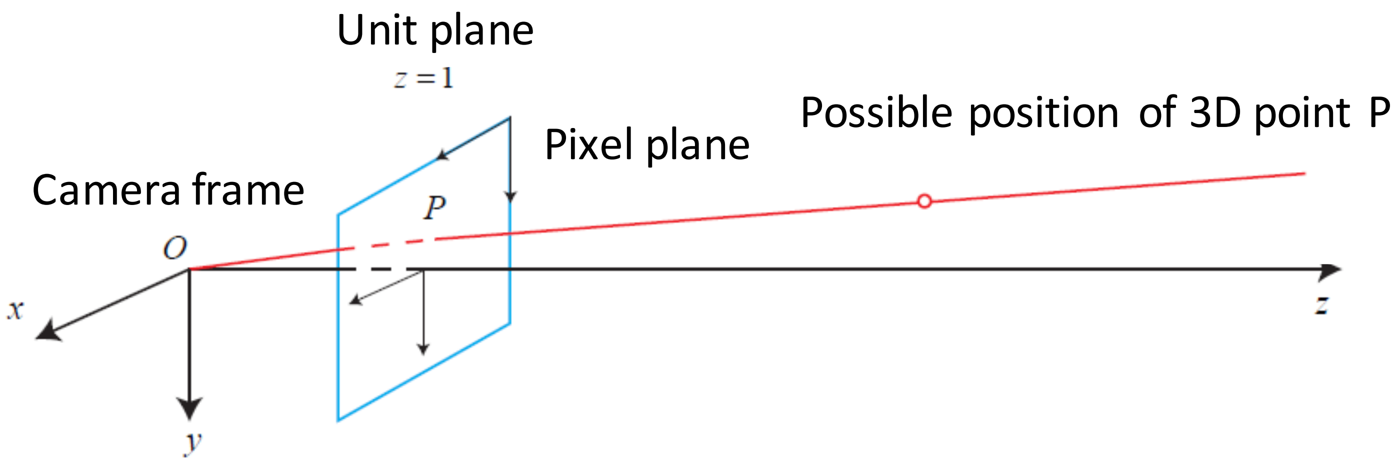
- There's another rotation and translation from the world to the camera

$$Z P_{uv} = Z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \left(R P_w + t \right) = K T P_w .$$

- Here R,t or T is called as extrinsic
  - Note we assume the homogeneous coordinates are cast to non-homogenous 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

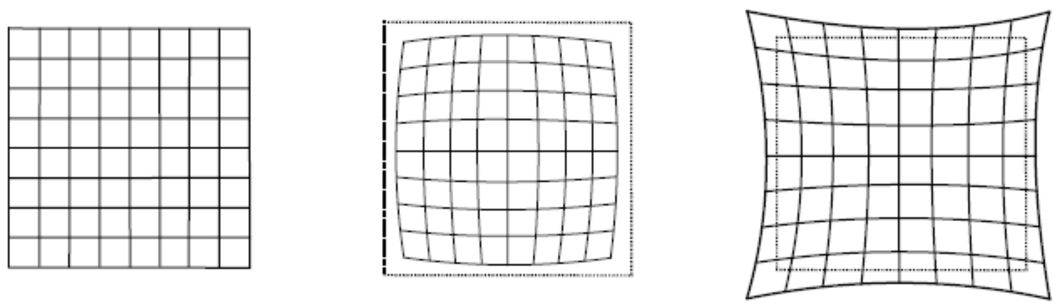
- Distortion
  - Lens will cause distortion when you have a wide range lens

Wide range lens

Fisheye cameras

1. Camera intrinsic and extrinsic

- Distortion types: radial distortion and tangential distortion



Original image   Barrel distortion   Pincushion distortion

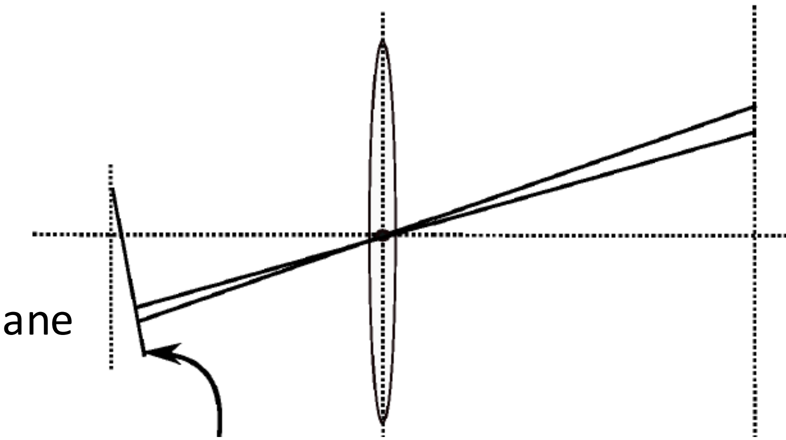


Image plane

1. Camera intrinsic and extrinsic Distortion

- Mathematic form

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

Radial distortion

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

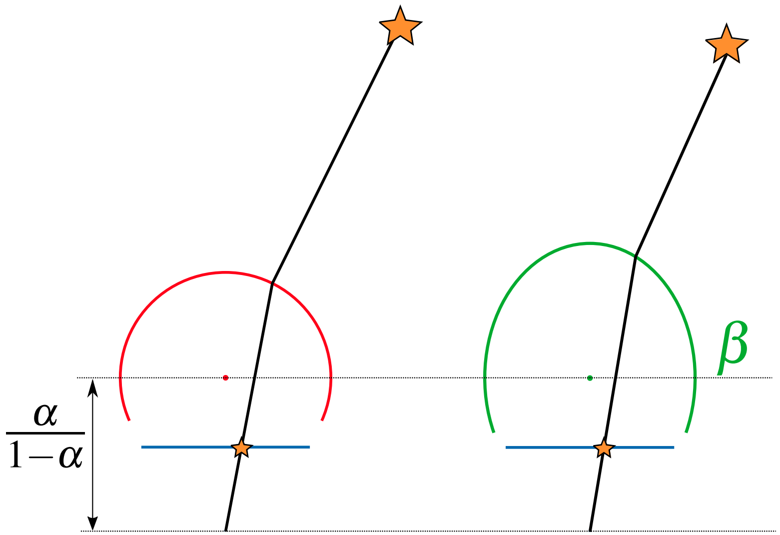
tangential distortion

- Put them together

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

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

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

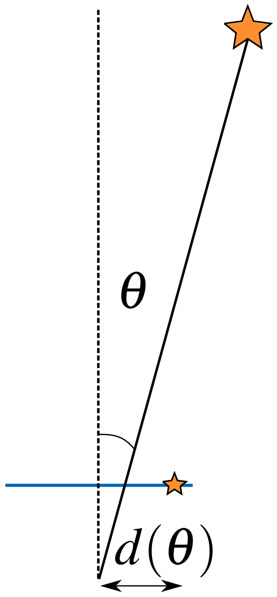


$$\mathbf{i} = [f_x, f_y, c_x, c_y, \alpha, \beta]^T, \alpha \in [0,1], \beta > 0$$

$$\pi(\mathbf{x}, \mathbf{i}) = \begin{bmatrix} f_x \frac{x}{\alpha d + (1-\alpha) z} \\ f_y \frac{y}{\alpha d + (1-\alpha) z} \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix},$$

$$d = \sqrt{\beta(x^2 + y^2) + z^2}.$$

1. Camera intrinsic and extrinsic: Kannala-Brandt Model



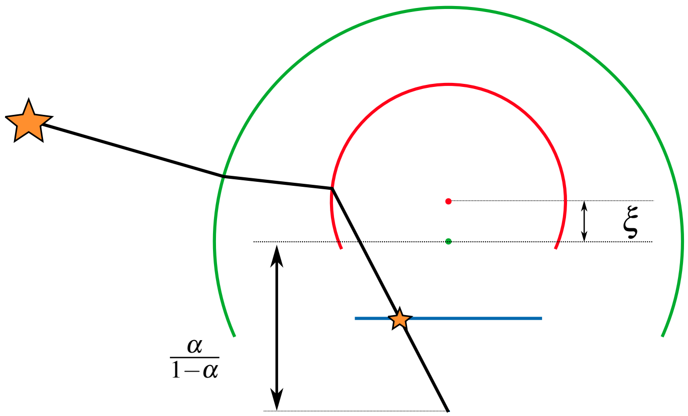
$$\mathbf{i} = [f_x, f_y, c_x, c_y, k_1, k_2, k_3, k_4]^T$$

$$\pi(\mathbf{x}, \mathbf{i}) = \begin{bmatrix} f_x\, d(\theta)\, \frac{x}{r} \\ f_y\, d(\theta)\, \frac{y}{r} \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix},$$

$$r = \sqrt{x^2 + y^2}, \theta = \text{atan2}(r, z),$$

$$d(\theta) = \theta + k_1\theta^3 + k_2\theta^5 + k_3\theta^7 + k_4\theta^9$$

1. Camera intrinsic and extrinsic: Double Sphere Camera Model



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

$$d_1 = \sqrt{x^2 + y^2 + z^2},$$

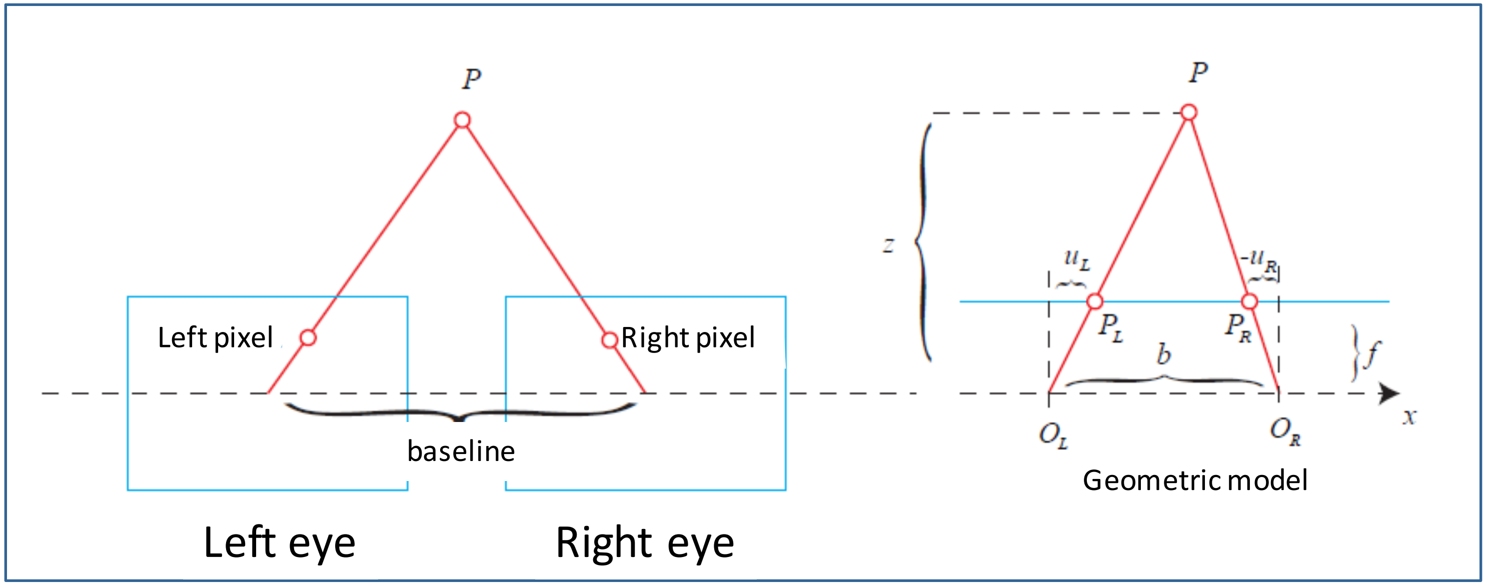
$$d_2 = \sqrt{x^2 + y^2 + (\xi d_1 + z)^2},$$

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

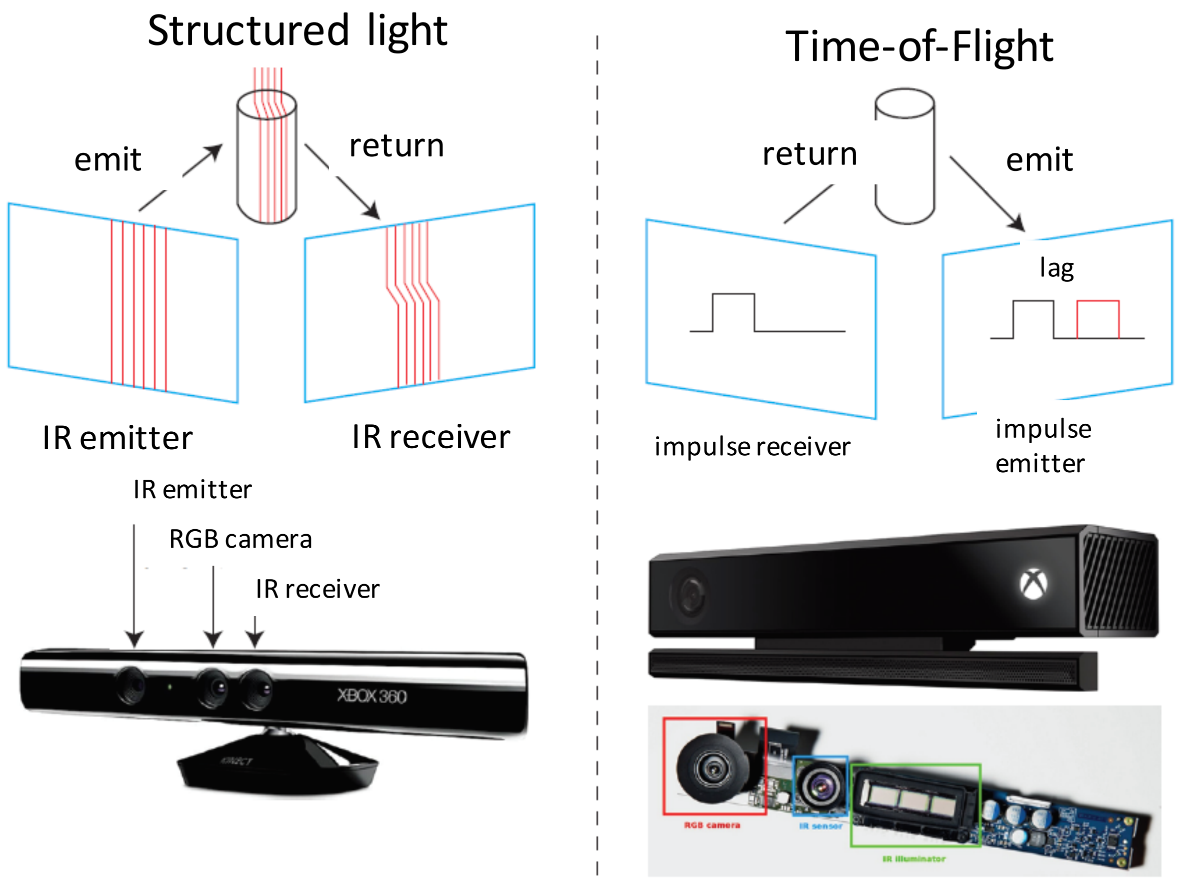
- Stereo camera
  - Two cameras (usually) placed horizontally



  - The distance between left camera center to the right is called as baseline
  - From geometric model:

$$\frac{z-f}{z} = \frac{b-u_L+u_R}{b}. \Rightarrow \quad z = \frac{fb}{d}, \quad d = u_L - u_R.$$

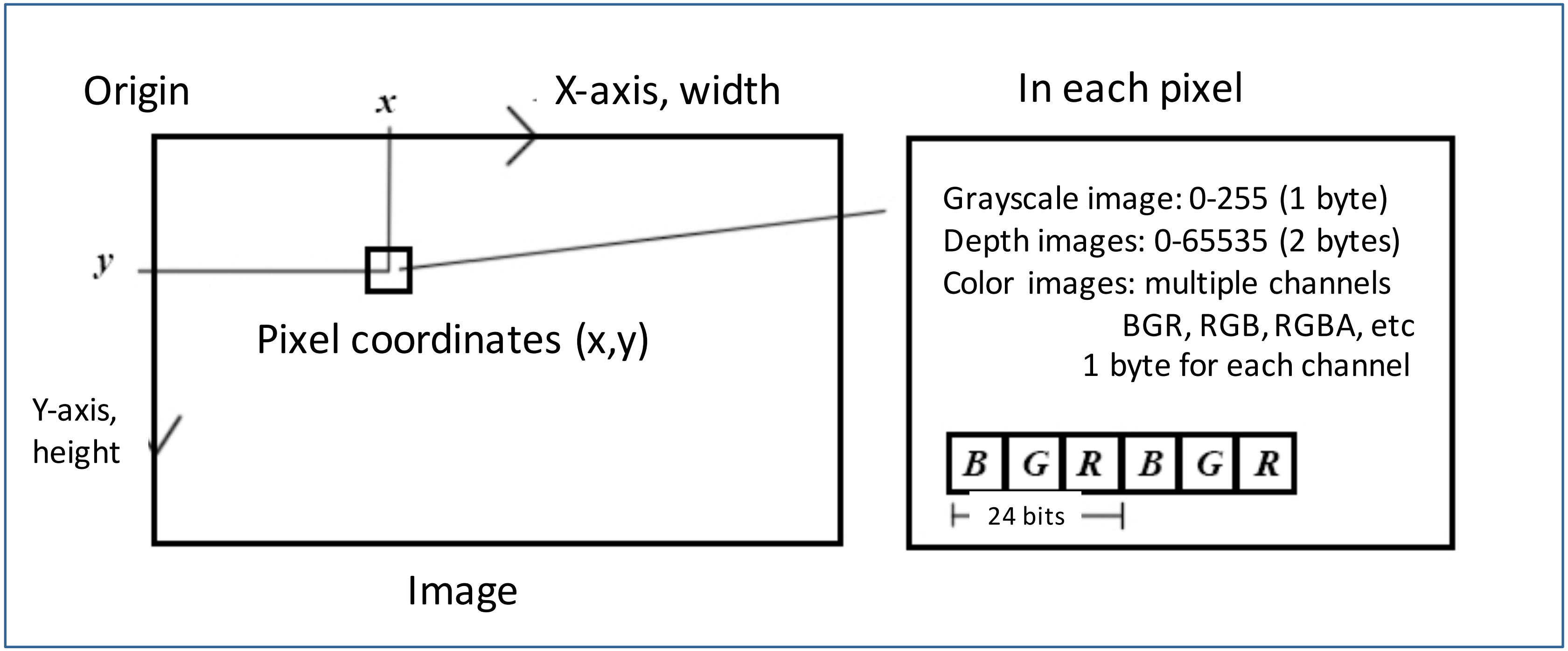
1. Camera intrinsic and extrinsic



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

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

- 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:

$$\boldsymbol{x} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_M\}.$$

Observation and input:

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

- Our purpose:

$$P(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{u}).$$

- Bayes' Rule:

Likehood Priori

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

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

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

Drop denominator because it is not relevant with x

Drop u because z is not relevant with 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: $\boldsymbol{z}_{k,j} = h\left(\boldsymbol{y}_j, \boldsymbol{x}_k\right) + \boldsymbol{v}_{k,j},$
  - Affected by white Gaussian noise: $v_{k,j} \sim N\left(0, Q_{k,j}\right)$

- The observation model gives us a conditional pdf:

$$P(\boldsymbol{z}_{j,k}|\boldsymbol{x}_k, \boldsymbol{y}_j) = N\left(h(\boldsymbol{y}_j, \boldsymbol{x}_k), \boldsymbol{Q}_{k,j}\right).$$

- Then how to compute the MAP of x,y given z?

2. From state estimation to least square

- Gaussian distribution (matrix form)

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

- Take minus logarithm at both sides:

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

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:

Max: $P(\boldsymbol{z}_{j,k}|\boldsymbol{x}_k, \boldsymbol{y}_j) = N\left(h(\boldsymbol{y}_j, \boldsymbol{x}_k), \boldsymbol{Q}_{k,j}\right).$

Information matrix

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

Error or residual of single observation

- We turn a MAP problem into a least square problem

2. From state estimation to least square

- Batch least square
- Original problem

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

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

Least square
Define the errors(residuals)

$$\boldsymbol{e}_{v,k} = \boldsymbol{x}_k - f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k\right)$$

$$\boldsymbol{e}_{y,j,k} = \boldsymbol{z}_{k,j} - h\left(\boldsymbol{x}_k, \boldsymbol{y}_j\right),$$

- Sum of the squared residuals:

min

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

2. From state estimation to least square

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

- 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

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:

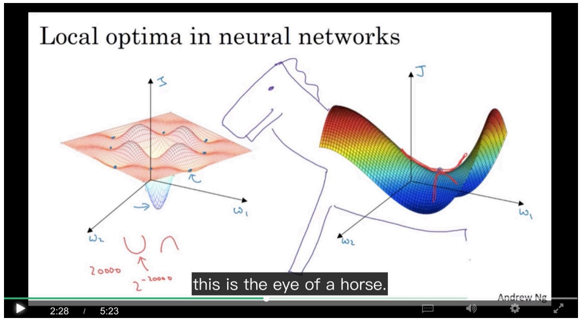
$$\min J(x) = \min \frac{1}{2}\|f(x)\|_2^2$$

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

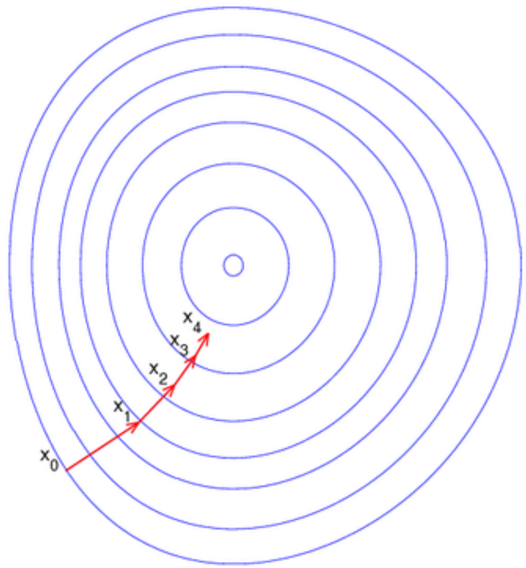
- When J is simple, just solve:

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

- 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 Δx_k to make $\|f(x_k + \Delta x_k)\|_2^2$ become smaller
  3. If Δx_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:

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

Jacobian Hessian

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

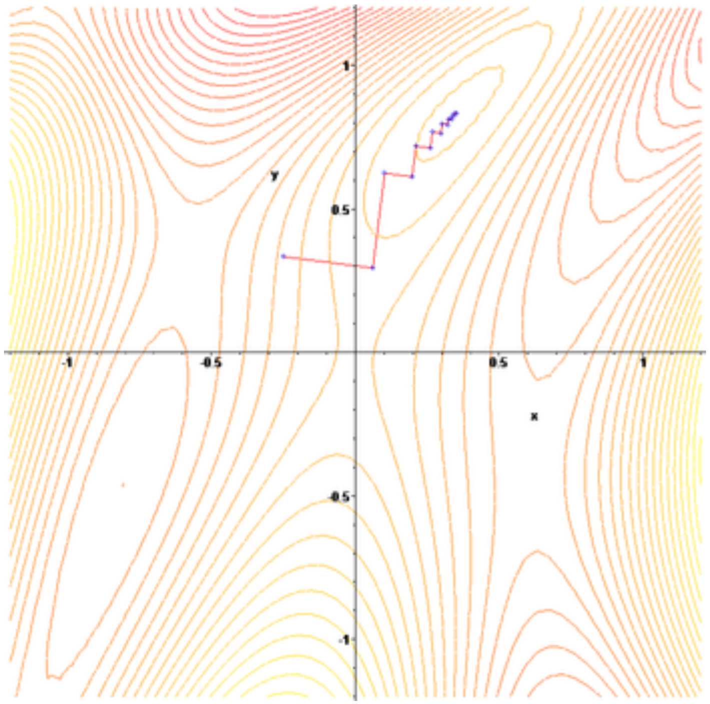
$$\min_{\Delta x}\|f(x)\|_2^2 + J\Delta x$$

Incremental will be: $\Delta\boldsymbol{x}^* = -\boldsymbol{J}^T(\boldsymbol{x}).$

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

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

- Solve an increment to minimize it:

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

- Let the derivative to Δx be zero, then we get: $\boldsymbol{H}\Delta\boldsymbol{x} = -\boldsymbol{J}^T.$

- 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): $f(\boldsymbol{x}+\Delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x}.$
  - Then the squared error becomes:

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

  - Also let its derivative with Δx be zero:

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

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

↓ H ↓ g

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

3. Batch least square

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

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

3. Batch least square

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

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

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:

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

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

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

- This λI 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:

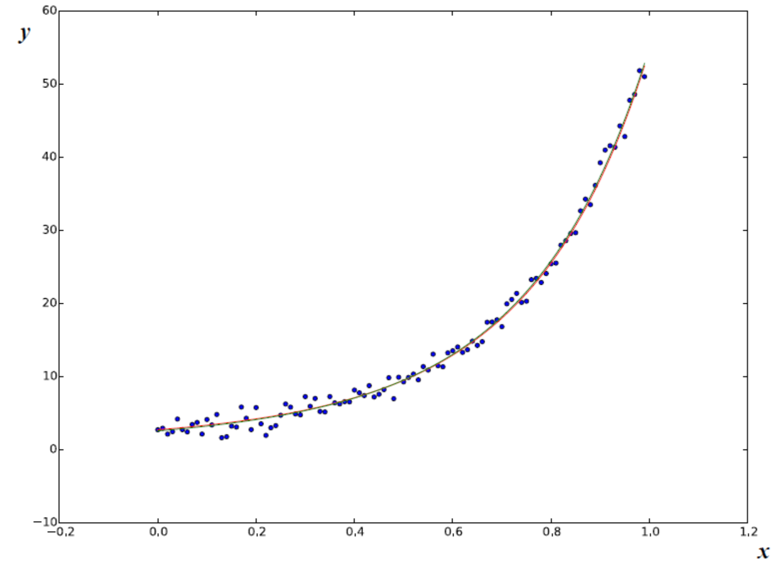
Curve function: $y = \exp(ax^2 + bx + c) + w,$

- Error:

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

- Least square problem:

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



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 Levernberg-Marquardt method
  - The least square problem can also be represented by a graph and forms a (factor) graph optimization problem

Contents

4. Application: Camera Calibration

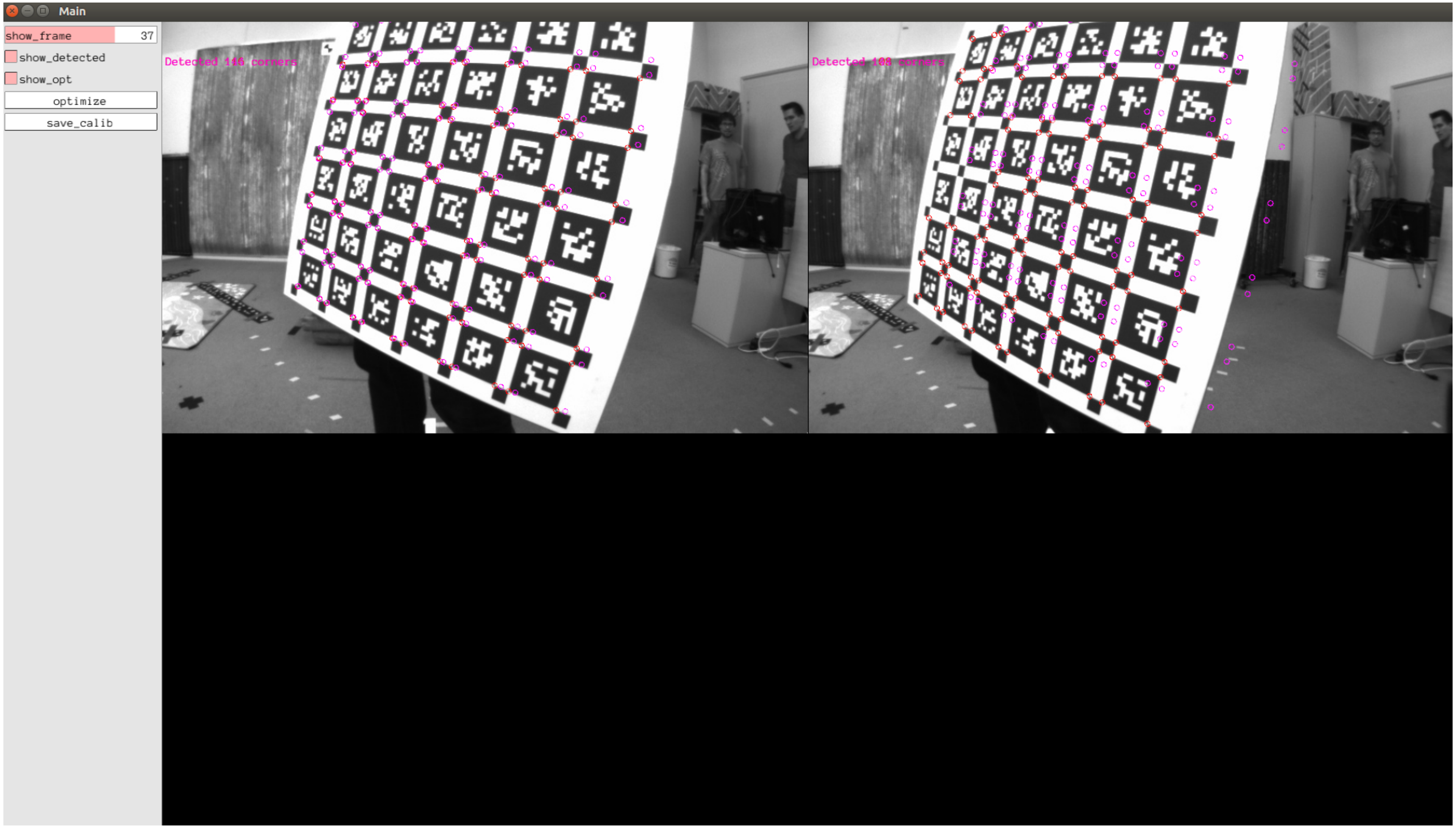
- Suppose we want to estimate the camera pose
- We have several observations from the projection function
- Minimizing the reprojection error:

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

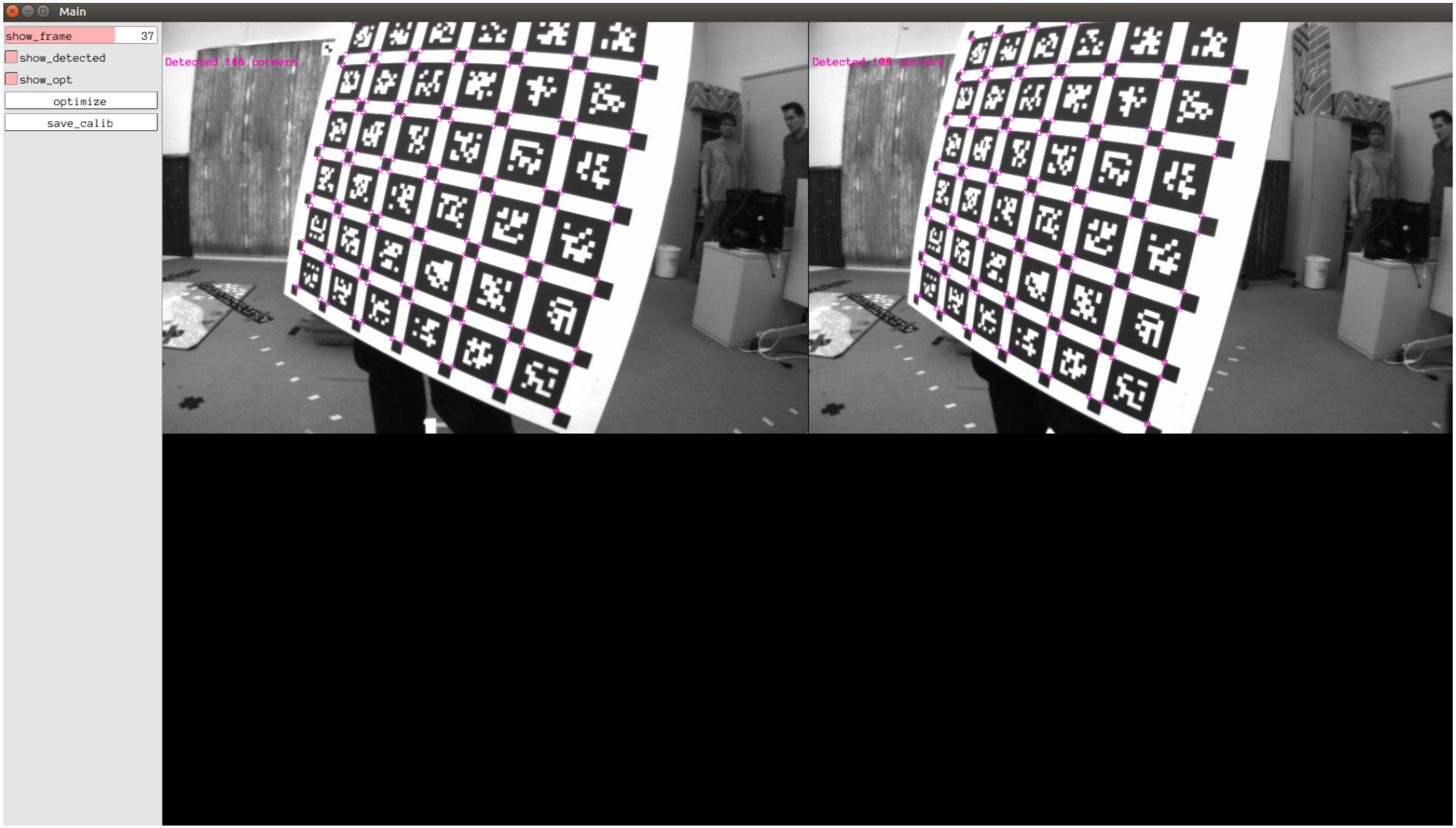
  - Where $\pi(\cdot)$ is the projection equation (observation model)
- Corner points are detected using Apriltags

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

- Linearize the error: $e_i(x \oplus \Delta x) \approx e_i(x) + J(x)\Delta x$
- Derivative is defined by SE(3) disturb model:

$$\frac{\partial e}{\partial T} = \lim_{\delta\xi \to 0} \frac{e(\delta\xi \oplus T) - e(T)}{\delta\xi}$$

$$= \lim_{\delta\xi \to 0} \frac{\frac{1}{Z} K(\delta\xi \oplus T)P - \frac{1}{Z} KTP}{\delta\xi}$$

- Let $P' = TP$ then use chain rule: $\frac{\partial e}{\partial T} = \frac{\partial e}{\partial P'} \frac{\partial P'}{\partial T}$
- For P' we have:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix}.$$

$$u = f_x \frac{X'}{Z'} + c_x, \quad v = f_y \frac{Y'}{Z'} + c_y.$$

$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{P}'} = - \begin{bmatrix} \frac{\partial u}{\partial X'} & \frac{\partial u}{\partial Y'} & \frac{\partial u}{\partial Z'} \\ \frac{\partial v}{\partial X'} & \frac{\partial v}{\partial Y'} & \frac{\partial v}{\partial Z'} \end{bmatrix} = - \begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} \end{bmatrix}.$$

4. Application: Camera Calibration

- The second item: $\frac{\partial (TP')}{\partial T} = \begin{bmatrix} I & -P'^{\wedge} \\ 0^T & 0^T \end{bmatrix}$ See Lecture 2.
- Remove the homogeneous part:

$$\frac{\partial (TP')}{\partial T} = \begin{bmatrix} I & -P'^{\wedge} \end{bmatrix}$$

- Put them together:

$$\frac{\partial e}{\partial T} = -\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} & -\frac{f_x X' Y'}{Z'^2} & f_x + \frac{f_x X^2}{Z'^2} & -\frac{f_x Y'}{Z'} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} & -f_y - \frac{f_y Y'^2}{Z'^2} & \frac{f_y X' Y'}{Z'^2} & \frac{f_y X'}{Z'} \end{bmatrix}.$$

4. Application: Camera Calibration

- If we want to take the derivative of Point P

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_i = K(RP_i + t) = KTP_i$$

$$\frac{\partial e}{\partial P} = \frac{\partial e}{\partial P'}\frac{\partial P'}{\partial P} = -\begin{bmatrix} f_x/Z' & 0 & -f_x X'/Z'^2 \\ 0 & f_y/Z' & -f_y Y'/Z'^2 \end{bmatrix} R$$

- P is not relevant to translation t

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?