

Robot Vision

TTK4255

Lecture 05 – Iterative Pose Estimation and Non-linear Optimization

Annette Stahl

(Annette.Stahl@ntnu.no)

Department of Engineering Cybernetics – ITK

NTNU, Trondheim

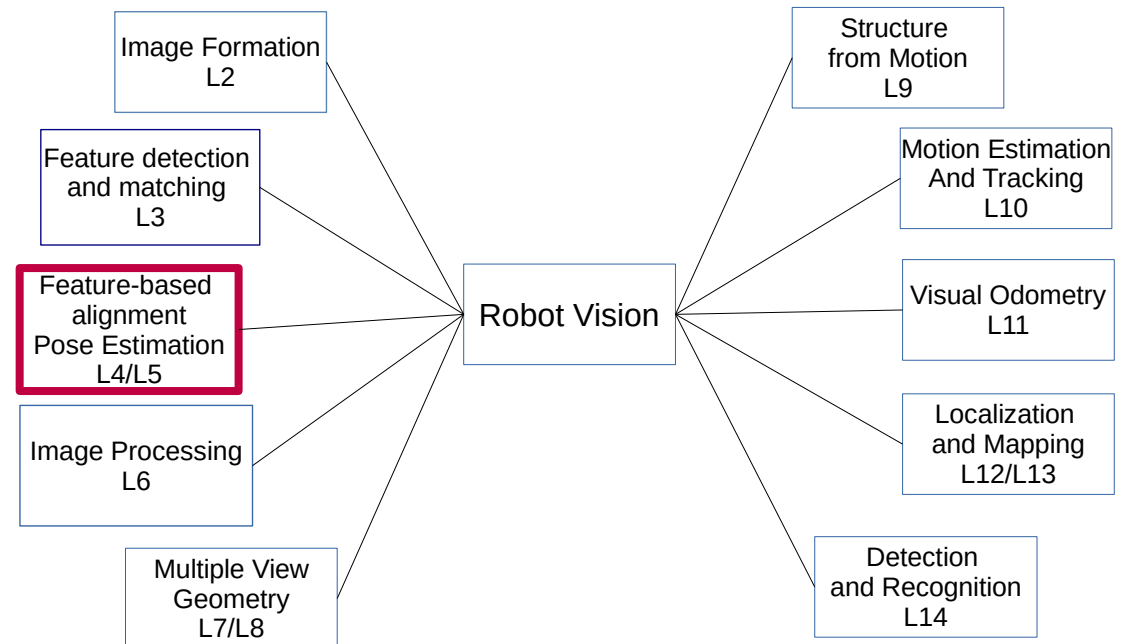
Spring Semester

03. February 2020

Lecture 05 — Iterative Pose Estimation and Non-linear Optimization

Annette Stahl (Annette.Stahl@ntnu.no)

Simen Haugo (Simen.Haugo@ntnu.no)



Outline of the fifth lecture:

- Orientation and Pose
- Non-linear Least-Squares
- Iterative Pose Estimation
- Gauss Newton
- Levenberg-Marquardt

Recap L04

2D-2D Feature Alignment: Homography

- Automatic point-correspondences
- RANSAC estimation
 - Basic DLT (Direct Linear Transform) on 4 random correspondences
 - Inliers determined from the re-projection
 - Improve estimate by normalized DLT

3D-2D Feature Alignment: Pose estimation

- RANSAC and Perspective n Points - PnP Problems
- Alternative methods

Iterative Pose Estimation



Extract Local Features

Establish 2D-3D Matches

Non-linear
Least Squares Problem
Iterative Pose Estimation

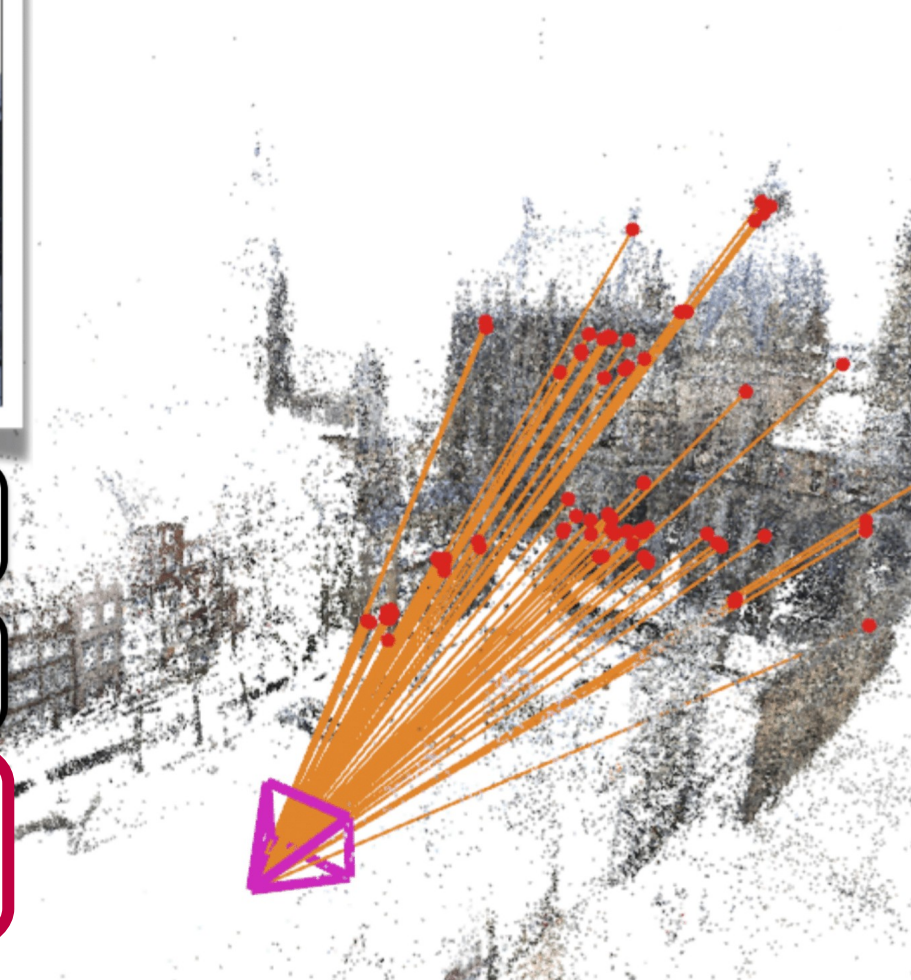


Image Courtesy: Torstein Sattler, CVPR 2015 Tutorial on Large-Scale Visual Place Recognition and Image-Based Localization

Orientation

- **Orientation** describes the relationship between coordinate systems
- **Orientation** is influenced by **Rotation**

In order to describe **the orientation** of a **camera coordinate system** relative to
the **world coordinate system**

We have to know how the **world coordinate frame** should **rotate** in order to align with
the **camera frame**

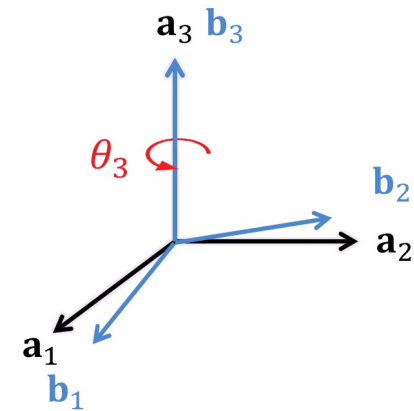
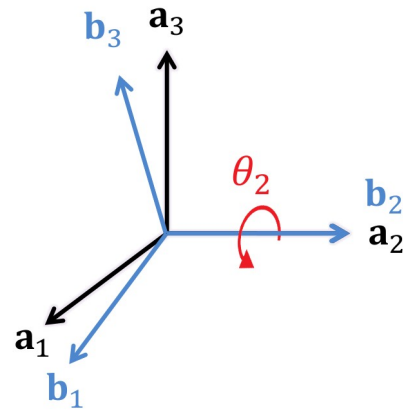
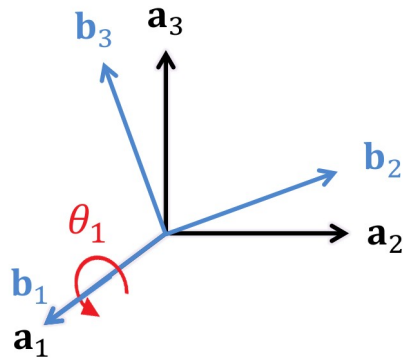
Rotations

$$R(\theta_1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_1 & -\sin \theta_1 \\ 0 & \sin \theta_1 & \cos \theta_1 \end{bmatrix}$$

$$R(\theta_2) = \begin{bmatrix} \cos \theta_2 & 0 & \sin \theta_2 \\ 0 & 1 & 0 \\ -\sin \theta_2 & 0 & \cos \theta_2 \end{bmatrix}$$

$$R(\theta_3) = \begin{bmatrix} \cos \theta_3 & -\sin \theta_3 & 0 \\ \sin \theta_3 & \cos \theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

R_{ab}



Rotation in Euler Angles

- The **roll-pitch-yaw** sequence $\mathbf{R}_z(\theta_3)\mathbf{R}_y(\theta_2)\mathbf{R}_x(\theta_1)$ is singular when $\theta_2 = \frac{\pi}{2}$

$$\mathbf{R}(\theta_1, \theta_2, \theta_3) = \begin{bmatrix} \cos \theta_3 & -\sin \theta_3 & 0 \\ \sin \theta_3 & \cos \theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta_2 & 0 & \sin \theta_2 \\ 0 & 1 & 0 \\ -\sin \theta_2 & 0 & \cos \theta_2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_1 & -\sin \theta_1 \\ 0 & \sin \theta_1 & \cos \theta_1 \end{bmatrix}$$

$\theta_3 = \text{yaw}$

$\theta_2 = \text{pitch}$

$\theta_1 = \text{roll}$

- If we use the notation $c_i = \cos(\theta_i)$ and $s_i = \sin(\theta_i)$ then we can write

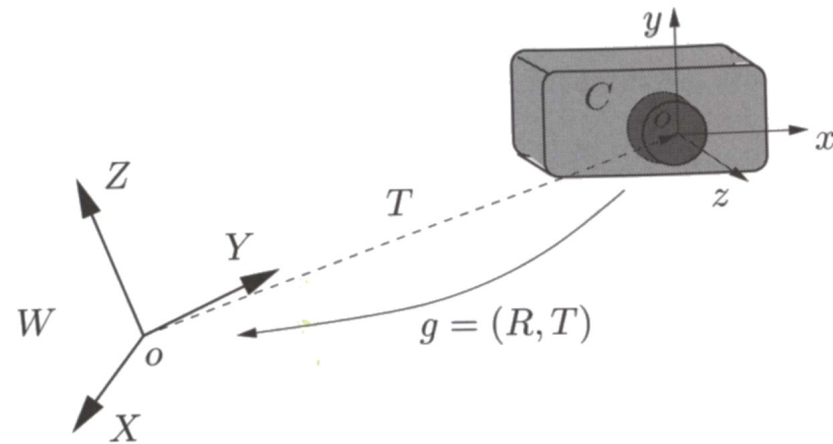
$$\mathbf{R}(\theta_1, \theta_2, \theta_3) = \begin{bmatrix} c_2 c_3 & s_1 s_2 c_3 - c_1 s_3 & c_1 s_2 c_3 + s_1 s_3 \\ c_2 s_3 & s_1 s_2 s_3 + c_1 c_3 & c_1 s_2 s_3 - s_1 c_3 \\ -s_2 & s_1 c_2 & c_1 c_2 \end{bmatrix}$$

Pose

- **Pose** describes the relationship between coordinate systems
- **Pose = [Position, Orientation]**

In order to describe the pose of a **camera coordinate system** relative to the **world coordinate system**

we have to know how the **world coordinate system** should **rotate and translate** in order to coincide with the **camera coordinate system**



Pose – (Recap L01)

The **pose** of the **camera coordinate system** with respect to the **world coordinate system** can be expressed by an Euclidean transformation matrix $[R/T]$:

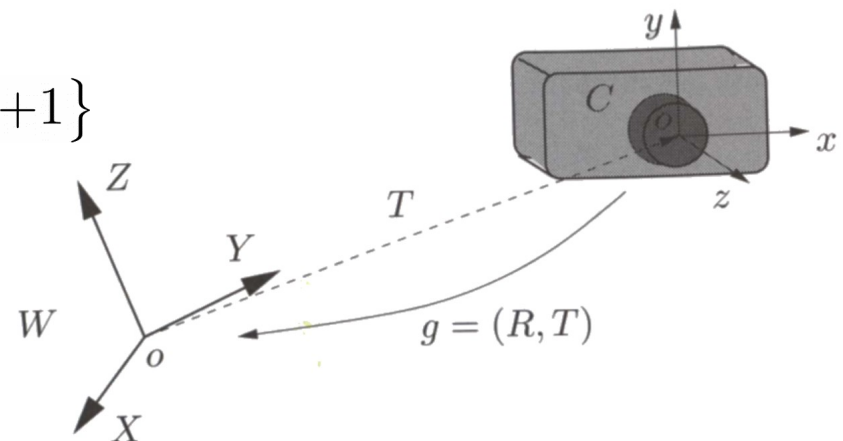
$$\tilde{g} = \begin{bmatrix} R(t) & T(t) \\ 0 & 1 \end{bmatrix} \in SE(3) \qquad g(t) = (R(t), T(t)) \in SE(3)$$

Where the $R \in \mathbb{R}^{3 \times 3}$ is a rotation matrix and $T \in \mathbb{R}^3$ is a translation vector given in world coordinates. The special Euclidean group is then described by

$$SE(3) = \left\{ \tilde{g} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \mid R \in SO(3), T \in \mathbb{R}^3 \right\} \subset \mathbb{R}^{4 \times 4}$$

with the special orthogonal group

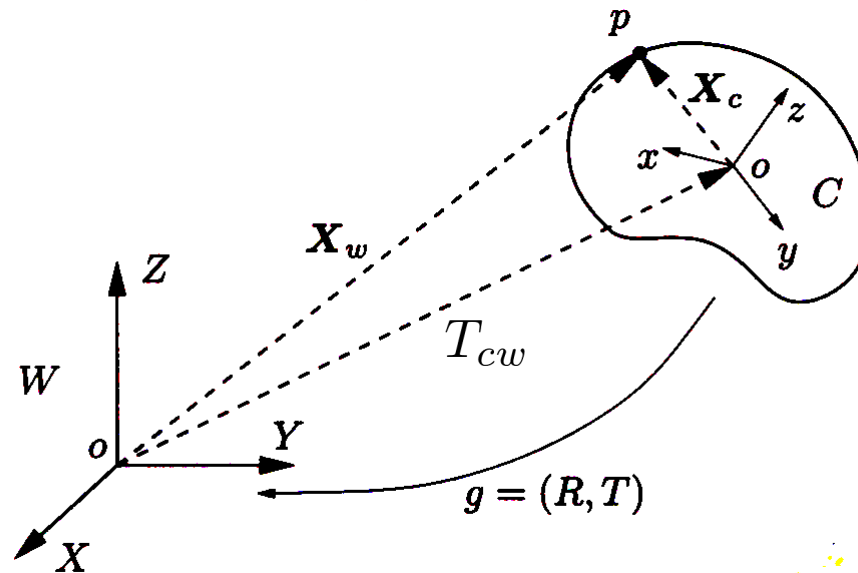
$$SO(3) = \{ R \in \mathbb{R}^{3 \times 3} \mid R^T R = I, \det R = +1 \}$$



Pose – (Recap L01)

- The matrix $[R/T]$ represents the **pose** of the world coordinate system relative to the camera coordinate system. In addition it is also a point transformation from the world to the camera coordinate system
- A point in world coordinates can be transformed to the camera coordinates by the **rigid body motion** $g = (R, T)$:

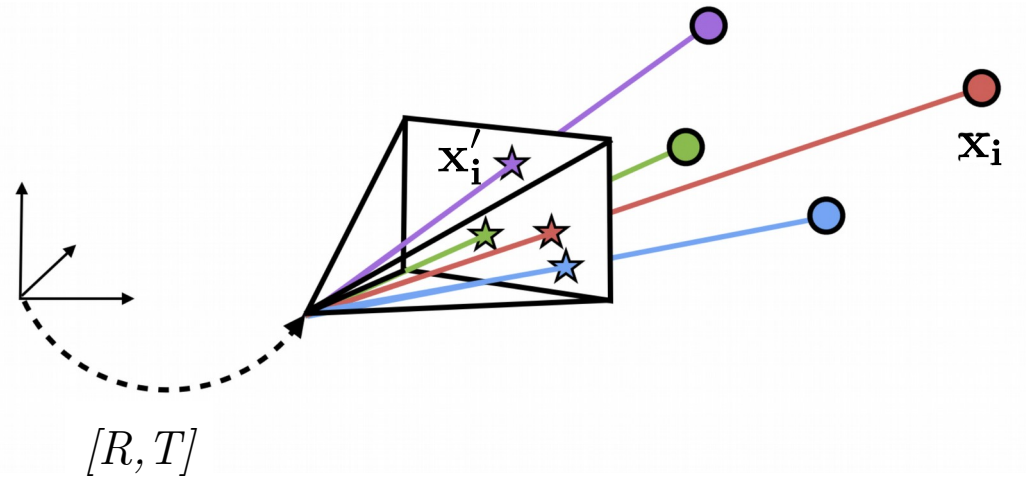
$$x_c = R_{cw}X_w + T_{cw}$$



Other Representations - Pose

- **Representations for rotation in 3D**
 - Orthonormal rotation matrix R
 - Euler angles $(\theta_1, \theta_2, \theta_3)$
 - Axis angle (\mathbf{a}, ϕ)
 - Unit quaternions $q = q_0 + q_1i + q_2j + q_3k$
- **Representations for pose in 3D**
 - Transformation matrix $T \in SE(3)$
 - Pair of rotation matrix and translation vector (R, T)
 - Euler angles and translation vector $(\theta_1, \theta_2, \theta_3, T)$
 - Axis angle and translation vector (\mathbf{a}, ϕ, T)
 - Unit quaternion and translation vector (q, T)

Recap L04 Perspective from n Point (PnP) Problem



- Given: n 2D-3D correspondences $\mathbf{x}_i \leftrightarrow \mathbf{x}_i', i = 1, 2, \dots, n$
- Compute pose $[R/T]$ s.t. $K[R/T] \mathbf{x}_i' = H\mathbf{x}_i$
- Optionally: Also estimate internal camera matrix K
 - In form of individual parameters
 - In form of projection matrix $P = K[R/T]$

Image Courtesy: Torstein Sattler, CVPR 2015 Tutorial on Large-Scale Visual Place Recognition and Image-Based Localization

Recap L04: Pose estimation - P6P

Problem: Given at least 6 known 3D landmark positions \mathbf{x}_i in the world frame and
 Given their 2D image correspondences \mathbf{x}'_i in the camera frame, $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i, i = 1, 2, \dots, n$
 Estimate the 6Dof 3D pose of the camera in the world frame including intrinsic parameters.

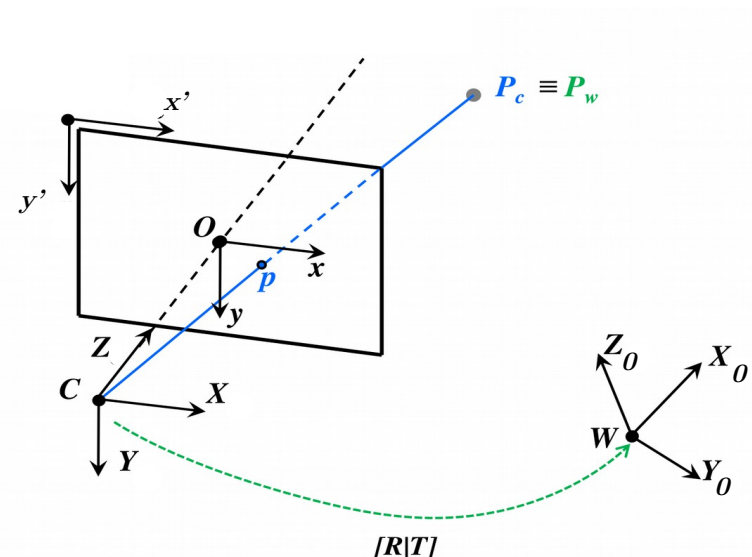
Solution: We can estimate a general **projection matrix** P that satisfies the perspective projection model

$$\lambda \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} = \begin{bmatrix} fs_x & fs_\theta & o_x \\ 0 & fs_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}$$

$$\tilde{\mathbf{x}}'_i = K[R|T]\tilde{\mathbf{x}}_i$$

$$\tilde{\mathbf{x}}'_i = P\tilde{\mathbf{x}}_i$$

where $P = K[R|T]$



Direct Linear Transformation - DLT

- **DLT** provides a solution relying on the **solution of a linear system**
(Solution of a homogeneous linear system is the eigenvector of A corresponding to its minimal eigenvalue (computed via the SVD of A))
 - The solution is very sensitive to noise
 - use a method that is more accurate and more robust against noise
 - A solution that **considers the non-linear constraints of the system**

Iterative Pose Estimation



Extract Local Features

Establish 2D-3D Matches

Non-linear
Least Squares Problem
Iterative Pose Estimation

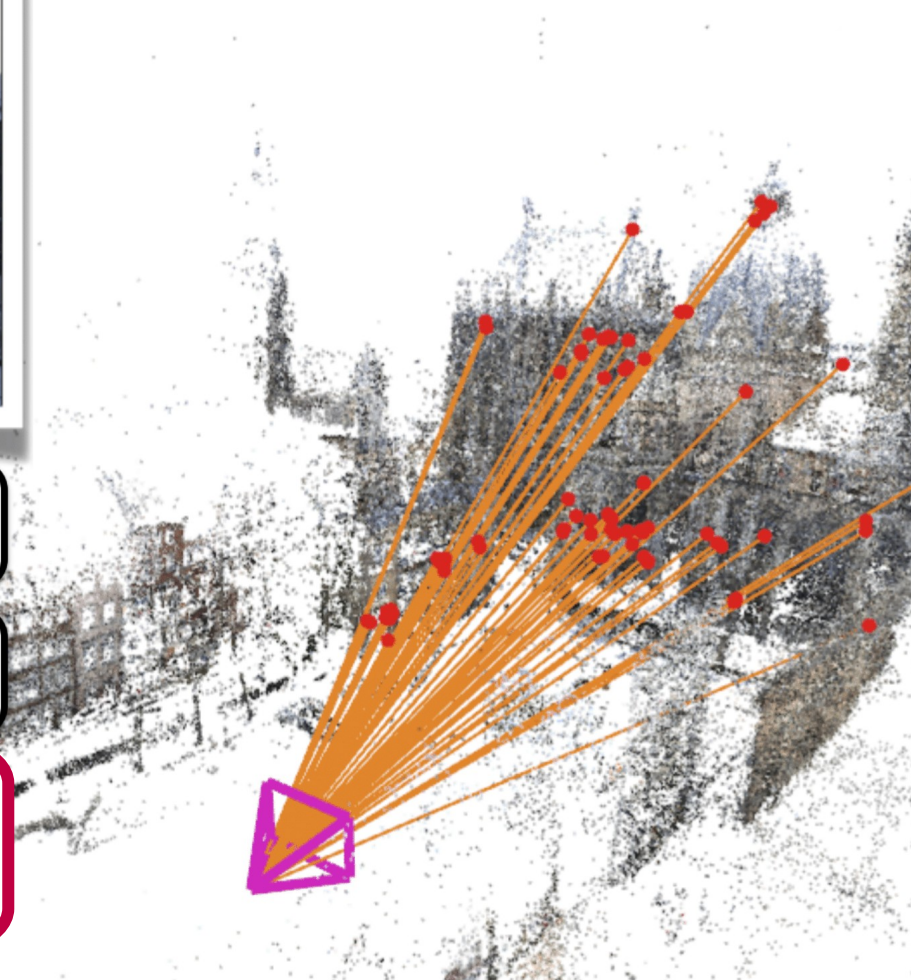


Image Courtesy: Torstein Sattler, CVPR 2015 Tutorial on Large-Scale Visual Place Recognition and Image-Based Localization

Non-linear Pose Estimation

Goal: Solve the **non-linear Least-Squares Problem**

$$\min_{\theta} f(\theta) = \min_{\theta} \frac{1}{2} \|r(\theta)\|^2 = \min_{\theta} \frac{1}{2} \sum_{i=1}^m r_i^2(\theta)$$

by finding a minimizer $\theta^* = \arg \min_{\theta} f(\theta)$ of the **nonlinear objective function** $f(\theta)$.

The **residual** is given with respect to all parameters of θ we wish to estimate regarding m measurements

$$r(\theta) = [r_1(\theta), \dots, r_m(\theta)]^\top, \quad \theta = [f, o_x, o_y, \theta_x, \theta_y, \theta_z, t_x, t_y, t_z]^\top$$

The objective function is given by the **re-projection error** as a function of the unknown pose parameters in (R, T) and optionally K (if the camera is uncalibrated).

Note: Many pose estimation problems consider the camera as calibrated and the pose parameters are given by a 6-element vector $\theta = [\theta_x, \theta_y, \theta_z, t_x, t_y, t_z]^\top$ with 3 Euler angles and 3 translation parameters.

Overview

- Preliminaries (Reminder)
 - Gradients,
 - Jacobian Matrix,
 - Chain Rule,
 - Hessian
 - Iterative Optimization
- Non-linear optimization problems
 - Gauss-Newton
 - Trust Region Method: Levenberg-Marquardt
- Pose estimation by iterative algorithm
 - Levenberg-Marquardt

Gradients – Jacobian (First Derivatives)

Gradient of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that depends on multiple variables:

$$\frac{\partial}{\partial x} f(x) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right) = \nabla f(x)$$

Gradient of a vector-valued function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ that depends on multiple variables defines the Jacobian Matrix $J_f \in \mathbb{R}^{m \times n}$:

$$\frac{\partial}{\partial x} f(x) = J_f = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

Chain Rule

If $f : \mathbb{R} \rightarrow \mathbb{R}$ and $g : \mathbb{R} \rightarrow \mathbb{R}$ the chain rule for gradients applies as

$$\frac{\partial}{\partial x} f(g(x)) = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x}$$

If $f : \mathbb{R}^o \rightarrow \mathbb{R}^m$ and $g : \mathbb{R}^n \rightarrow \mathbb{R}^o$ with $J_f \in \mathbb{R}^{m \times o}$ and $J_g \in \mathbb{R}^{o \times n}$ we obtain

$$\frac{\partial}{\partial x} f(g(x)) = J_f \cdot J_g = \begin{pmatrix} \frac{\partial f_1}{\partial g_1} & \dots & \frac{\partial f_1}{\partial g_o} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial g_1} & \dots & \frac{\partial f_m}{\partial g_o} \end{pmatrix} \cdot \begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \dots & \frac{\partial g_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_o}{\partial x_1} & \dots & \frac{\partial g_o}{\partial x_n} \end{pmatrix}$$

Hessian (Second Derivative)

The **first partial derivatives** of the residuals and hence the **Jacobian matrix** J are relatively inexpensive to calculate. We can obtain the gradient of f as

$$\nabla f(x) = \sum_{i=1}^m r_i(x) \nabla r_i(x) = J^\top r(x)$$

$$f(\theta) = \frac{1}{2} \sum_{i=1}^m r_i^2(\theta), \quad \theta = x$$

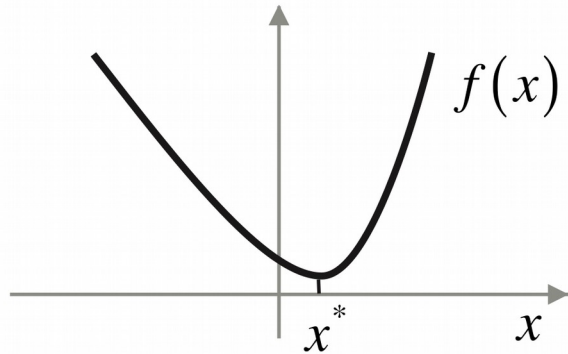
The **Hessian** of f can then be expressed as follows:

$$\begin{aligned} \nabla^2 f(x) &= \sum_{i=1}^m \nabla r_i(x) \nabla r_i(x)^\top + \sum_{i=1}^m r_i(x) \nabla^2 r_i(x) \\ &= J^\top J + \sum_{i=1}^m r_i(x) \nabla^2 r_i(x) \end{aligned}$$

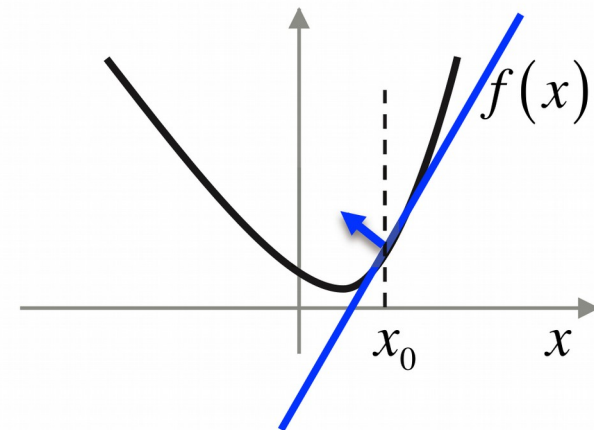
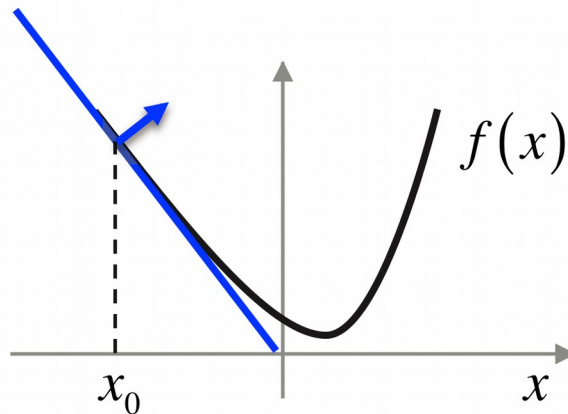
Note: Using the Jacobian we can also calculate the first term of the last expression of the Hessian without evaluating any second derivatives of the functions. This term is often more important than the second summation term, either because the residuals are close to affine near the solution or because of small residuals.

Minimizing a Non-linear Function

Find a x^* that minimizes a non-linear function $f(x)$.

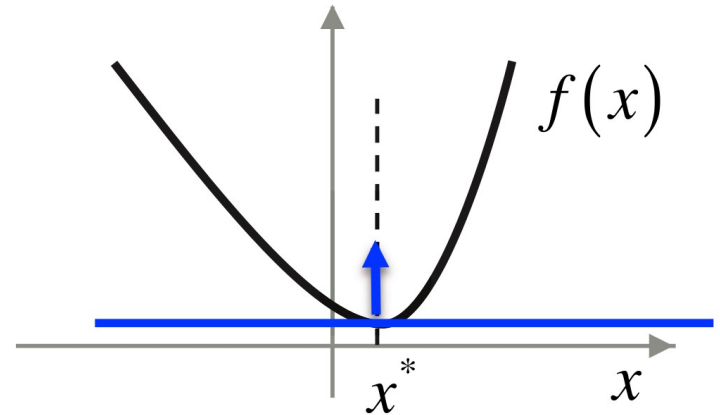


→ compute gradient (first derivative) of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ at some point x_0



Minimizing a Non-linear Function

Extrema: Necessary conditions for optimality are derived by assuming that x^* is a local minimizer and then proving facts about $\nabla f(x^*)$ and $\nabla^2 f(x^*)$



First-order Necessary Conditions:

If f is continuously differentiable in an open neighborhood of x^* then

$$\nabla f(x^*) = 0$$

Second-order necessary Conditions:

If $\nabla^2 f$ exists and is continuous in an open neighborhood of x^* then

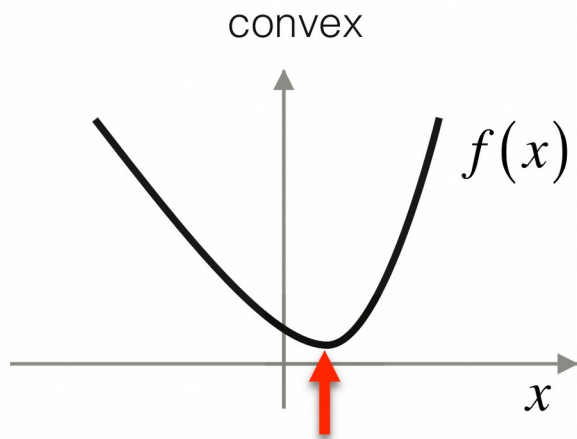
$$\nabla f(x^*) = 0$$

and $\nabla^2 f(x^*)$ is positive semidefinite.

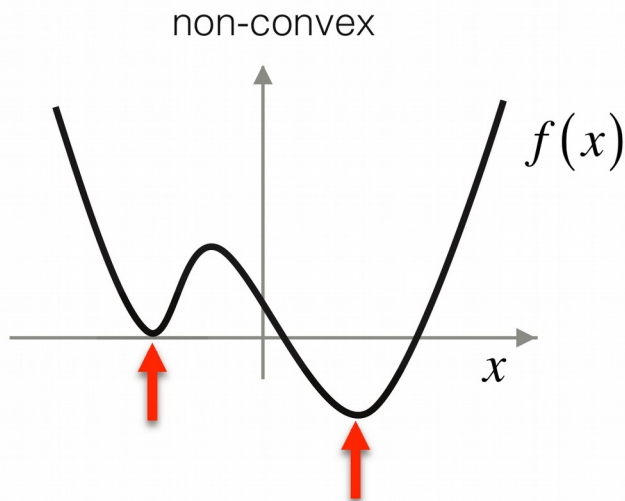
(Note: second derivative used to verify the maximum or minimum)

Optimization

Convex Optimization – only a single **global** minimum exists



Non-convex optimization – multiple **local** minima exist



Overview

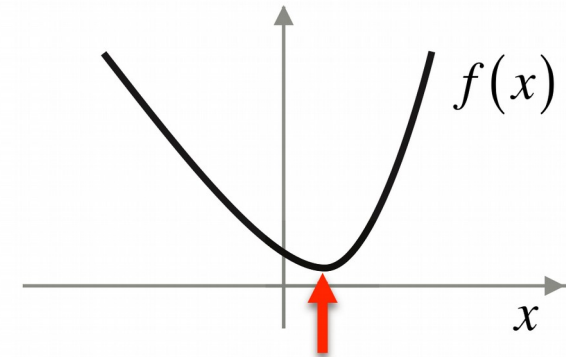
- Preliminaries (Reminder)
 - Gradients,
 - Jacobian Matrix,
 - Chain Rule,
 - Hessian
 - Iterative Optimization
- Non-linear optimization problems
 - Gauss-Newton
 - Trust Region Method: Levenberg-Marquardt
- Pose estimation by iterative algorithms
 - Levenberg-Marquardt

Non-linear Least-Squares Problem

How to find the (local) minimum of a non-linear function f ?

Solve the non-linear **Least-Squares Problem**

$$\begin{aligned}\min_x f(x) &= \min_x \frac{1}{2} \sum_{i=1}^m r_i^2(x) \\ &= \min_x \frac{1}{2} \|r(x)\|^2\end{aligned}$$

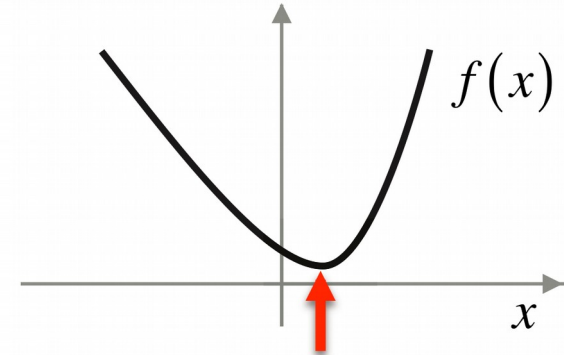


where $r(x) = [r_1(x), \dots, r_m(x)]^\top$ is the residual vector. The derivatives of $f(x)$ can be expressed in terms of the Jacobian J , which is the $m \times n$ matrix of first partial derivatives of the residuals defined by

$$J_r = J = \begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \cdots & \frac{\partial r_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_m}{\partial x_1} & \cdots & \frac{\partial r_m}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla r_1(x) \\ \vdots \\ \nabla r_m(x) \end{bmatrix}$$

Iterative Optimization – Gauss Newton

$$\min_x f(x) = \min_x \frac{1}{2} \sum_{i=1}^m r_i^2(x)$$



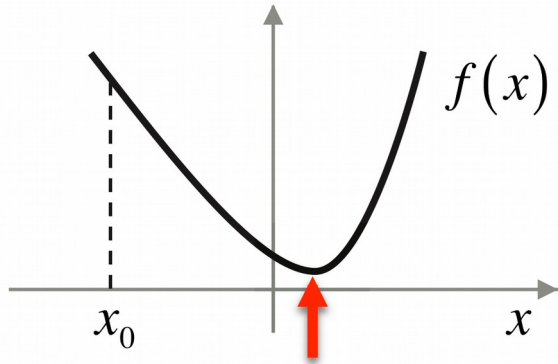
Non-linear Least-Squares Problems cannot be solved directly.

→ it requires an iterative optimization procedure like the **Gauss-Newton Method** to solve it:

1. Start at an initial estimate
2. Linearize the problem by using Taylor Series Expansion
3. Solve the linearized problem by using a linear Least-Square Problem
4. Update the estimate, return to step 2.

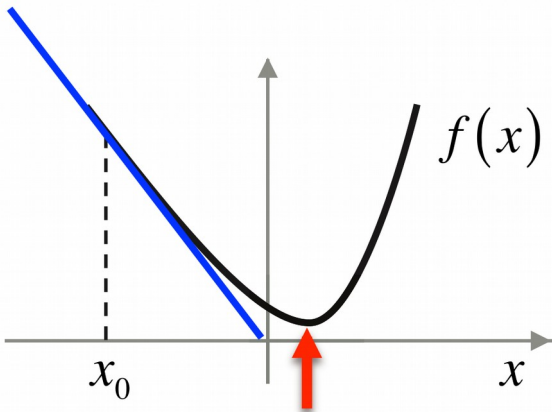
Gauss Newton

1. Start with an initial estimate x_0



$$\min_x f(x) = \min_x \frac{1}{2} \|r(x)\|^2$$

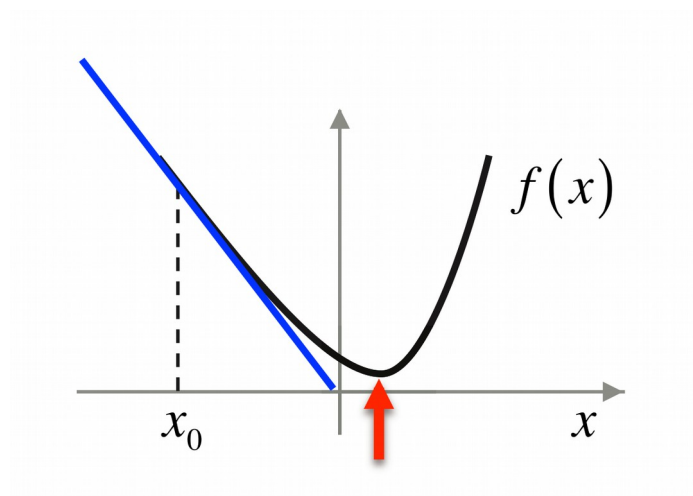
2. We can linearize the problem by using a first order Taylor expansion of f at the initial estimate x_0



$$\begin{aligned} \min_{\Delta x} f(x_0 + \Delta x) &= \min_{\Delta x} \frac{1}{2} \|r(x_0 + \Delta x)\|^2 \\ &= \min_{\Delta x} \frac{1}{2} \left\| r(x_0) + \frac{\partial}{\partial x} r(x_0) \Delta x + \epsilon \right\|^2 \\ &\approx \min_{\Delta x} \frac{1}{2} \left\| r(x_0) + \frac{\partial}{\partial x} r(x_0) \Delta x \right\|^2 \\ &= \min_{\Delta x} \frac{1}{2} \|r(x_0) + \nabla r(x_0) \Delta x\|^2 \\ &= \min_{\Delta x} \frac{1}{2} \|r(x_0) + J_r \Delta x\|^2 \end{aligned}$$

Gauss Newton

3. Solve **Linear Least-Squares Problem** with the new linear objective function $l(\Delta x)$



$$\min_{\Delta x} l(\Delta x) = \min_{\Delta x} \frac{1}{2} \|r(x_0) + J_r \Delta x\|^2$$

The solution Δx^* that minimizes l is the solution to the following linear system of equations

$$\begin{aligned} 2 \frac{1}{2} (r(x_0) + J_r \Delta x) J_r &= 0 \\ J_r^\top r(x_0) + J_r^\top J_r \Delta x &= 0 \end{aligned}$$

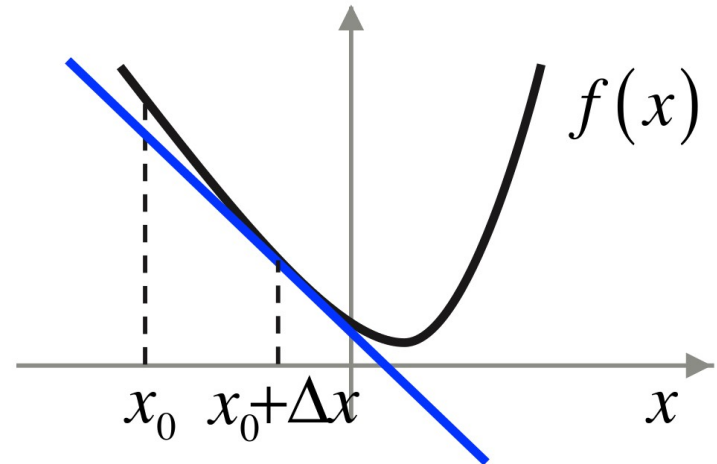
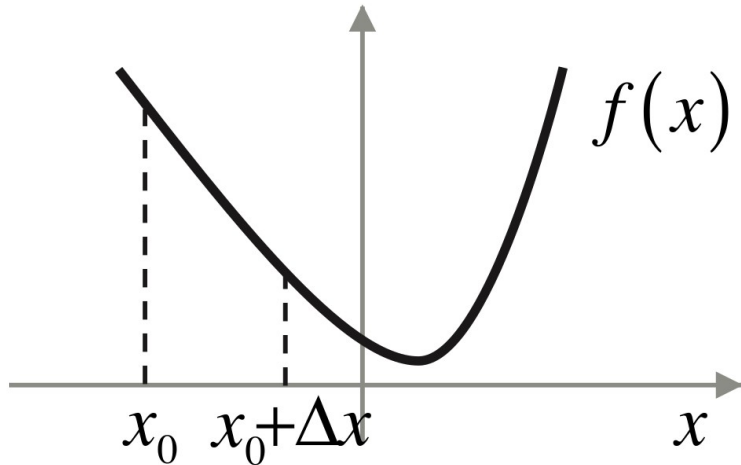
$$\Rightarrow \Delta x = (J_r^\top J_r)^{-1} (-J_r^\top r(x_0))$$

$$\Delta x = (J^\top J)^{-1} (-J^\top r(x_0))$$

$$J_r = J = \begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \cdots & \frac{\partial r_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_m}{\partial x_1} & \cdots & \frac{\partial r_m}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla r_1(x) \\ \vdots \\ \nabla r_m(x) \end{bmatrix}$$

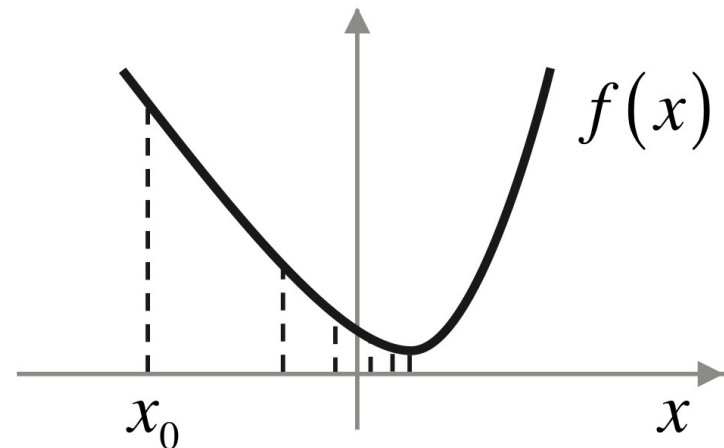
Gauss Newton

4. Take step and update $x_1 = x_0 + \alpha \Delta x$, where α is the step size; return to step 2.



$$\Delta x = (J^\top J)^{-1}(-J^\top r(x_k))$$

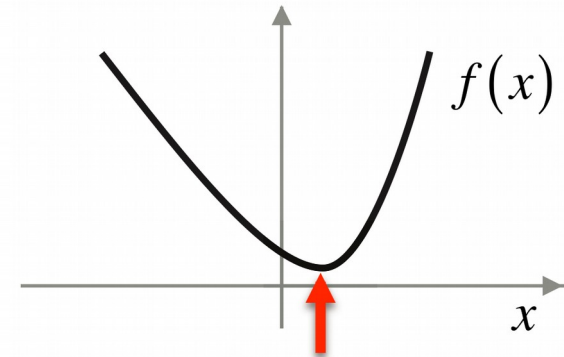
$$\Rightarrow x_{k+1} = x_k + \alpha \Delta x, \quad k = 0, 1, \dots$$



Iterative Optimization – Gauss Newton

Solution of the non-linear Least-Squares Problems

$$\min_x f(x) = \min_x \frac{1}{2} \sum_{i=1}^m r_i^2(x)$$



Iterative optimization - **Gauss-Newton Method:**

1. Start at an initial estimate
2. For $k = 0, 1, \dots$ Linearize the problem by using Taylor Series Expansion

$$\min_{\Delta x} f(x_k + \Delta x) = \min_{\Delta x} \frac{1}{2} \|r(x_k + \Delta x)\|^2$$

3. Solve the linearized problem by using a linear Least-Square Problem

$$\min_{\Delta x} l(\Delta x) = \min_{\Delta x} \frac{1}{2} \|r(x_k) + J_r \Delta x\|^2$$

4. Update the estimate, return to step 2.

$$J^\top J \Delta x = -J^\top r(x_k) \quad x_{k+1} = x_k + \alpha \Delta x, \quad k = 0, 1, \dots$$

Iterative Optimization – Gauss Newton

The Gauss Newton Method approximates the Hessian Matrix of the objective function f

$$\begin{aligned}\nabla f(x) &= \sum_i^m r_i(x) \nabla r_i(x) = J^\top r(x) \\ \nabla^2 f(x) &= \sum_i^m \nabla r_i(x) \nabla r_i(x)^\top + \sum_i^m r_i(x) \nabla^2 r_i(x) \\ &= \boxed{J^\top J} + \sum_i^m r_i(x) \nabla^2 r_i(x)\end{aligned}$$

If We assume that J has full rank, then the Hessian approximation

$$\nabla^2 f(x) \approx J^\top J$$

is positive definite and the Gauss Newton search direction has a descent direction.

→ The approximation is well conditioned if we start near the solution and

- The updated direction is good
- The updated step length is good
- We obtain nearly a quadratic convergence towards the (local) minimum

Iterative Optimization – Gauss Newton

We receive a poor approximation if the matrix $J^\top J$ is **ill-conditioned** (i.e. not invertible) and the equation

$$J^\top J \Delta x = -J^\top r$$

does not have an unique solution. In this case the problem is said to be **under-determined** or **over-parameterized**.

If that is the case than

- The updated direction may still be decent
- The updated step length might be overshooting or undershooting
- We may diverge or the we converge slower

Trust Region Method

In general:

- The Gauss Newton method is **not guaranteed to converge** because of the approximate Hessian matrix
- Update directions typically are decent, therefore a **limitation in step size leads to convergence**
- Such methods are often called **Trust Region Methods**
- One example is the **Levenberg-Marquardt Method**

Levenberg-Marquardt Method

Given an objective function $f(x)$ and a good **initial estimate** x_0

Set $\lambda = 10^{-3}$

For $k = 0, 1, \dots$

Linearize the objective function at $x_k \rightarrow J, r$

Solve the linearized problem with $(J^\top J + \lambda \text{diag}(J^\top J))\Delta x = -J^\top r(x_k)$

if $f(x_k + \Delta x) < f(x_k)$

$$x_{k+1} = x_k + \alpha \Delta x$$

$$\lambda = \lambda/10$$

else

$$x_{k+1} = x_k$$

$$\lambda = \lambda * 10$$

Terminate early if $f(x)$ is very small or $\|x_{k+1} - x_k\| < \text{xtol}$

Levenberg Marquardt

The **Levenberg Marquardt** method is a small modification of the Gauss Newton Method

Differences:

- Set step size
- Update the step size with

$$(J^{\top} J + \lambda \text{diag}(J^{\top} J)) \Delta x = -J^{\top} r$$

where $\text{diag}()$ is a positive definite matrix and λ is a scalar determined by the following rules:

1. At the start of optimization, λ is initialized to a relatively small value. A commonly used heuristic is 10^{-3} times the average of $\text{diag}()$.
2. In each iteration, if the update in step size leads to a reduced error, then the step is accepted and λ is decreased (e.g. divided by 10) before the next iteration.
3. Otherwise, if the update in step size leads to an increased error, then λ is increased (e.g. multiplied by 10) and the normal equations are solved again. This is repeated until an update in step size is found that leads to a reduced error.

Add a termination condition to prevent unnecessary iterations. A common choice is to stop when the change in the estimates between two successive iterations is less than the desired precision:

$$\|x_{k+1} - x_k\| < \text{xtol}$$

Overview

- Preliminaries (Reminder)
 - Gradients,
 - Jacobian Matrix,
 - Chain Rule,
 - Hessian
 - Iterative Optimization
 - Non-linear optimization problems
 - Gauss-Newton
 - Trust Region Method: Levenberg-Marquardt
- Pose estimation by iterative algorithms
 - Levenberg-Marquardt

Pose Estimation using Levenberg-Marquardt

Goal: Solve the **non-linear Least-Squares Problem**

$$\min_{\theta} f(\theta) = \min_{\theta} \frac{1}{2} \|r(\theta)\|^2 = \min_{\theta} \frac{1}{2} \sum_{i=1}^m r_i^2(\theta)$$

by finding a minimizer $\theta^* = \arg \min_{\theta} f(\theta)$ of the **nonlinear objective function** $f(\theta)$.

The **residual**

$$r(\theta) = [r_1(\theta), \dots, r_m(\theta)]^T, \quad \theta = [f, o_x, o_y, \theta_x, \theta_y, \theta_z, t_x, t_y, t_z]^T$$

in the objective function is given by the **re-projection error** as a function of the unknown pose parameters in (R, T) and optionally K (if the camera is uncalibrated).

Note: General pose estimation problems consider the camera as calibrated and the pose parameters are given by a 6-element vector $\theta = [\theta_x, \theta_y, \theta_z, t_x, t_y, t_z]^T$ with 3 Euler angles and translation parameters.

Non-linear Pose Estimation

The residuals $r_i(\theta)$ of the objective function $f(\theta)$ are given by the re-projection error

$$r_i(\theta) = (\pi_i(g(\theta)) - \tilde{\mathbf{x}}'_i) = (\mathbf{u}'_i - \mathbf{x}'_i)$$

with respect to the 3D-2D corresponding points $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i, i = 1, 2, \dots, m$

Where the re-projection error can also be expressed using inhomogeneous coordinates

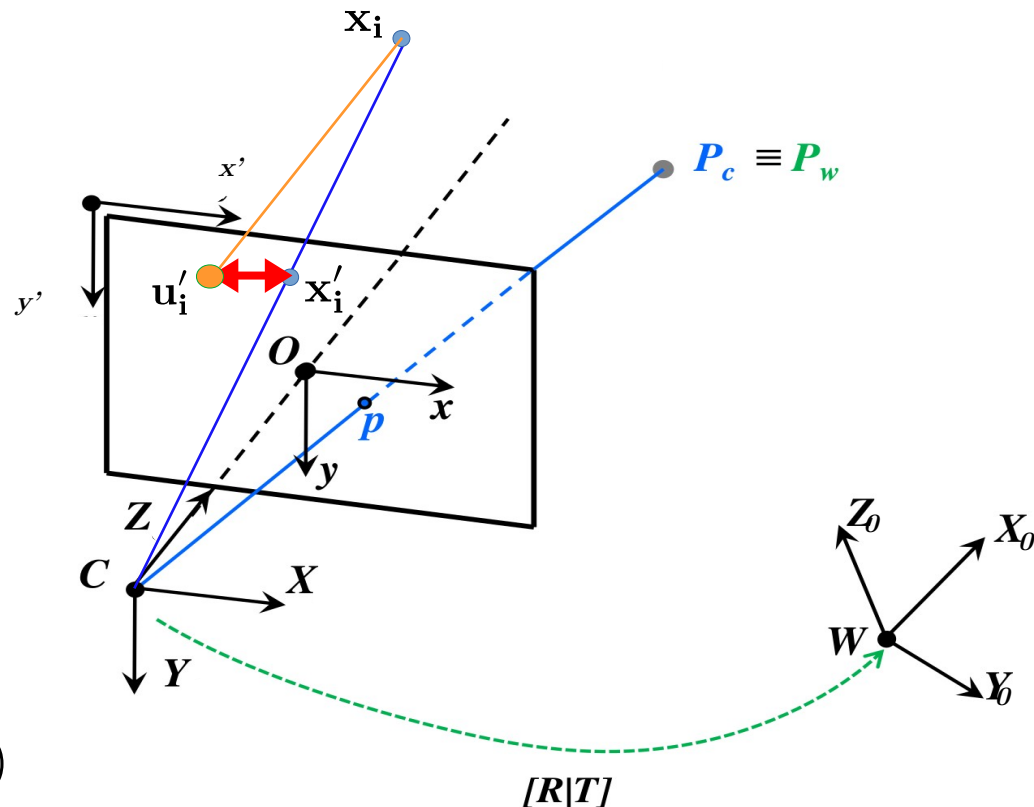
$$\begin{aligned} (\pi_i(g(\theta)) - \tilde{\mathbf{x}}'_i) &= (P\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}'_i) \\ &= (\mathbf{u}'_i - \mathbf{x}'_i) \end{aligned}$$

by using $[\tilde{x}, \tilde{y}, \tilde{w}]^\top \longrightarrow \left[\frac{\tilde{x}}{\tilde{w}}, \frac{\tilde{y}}{\tilde{w}}, 1 \right]^\top = [x, y]$

The projection matrix $P = K[R|T]$ can be written as

$$P(\theta) = K(f, o_x, o_y)[R(\theta_x, \theta_y, \theta_z)|T(t_x, t_y, t_z)]$$

Note: \mathbf{x}'_i is the measured data. The nonlinearity arises only from the observation function $\pi_i(g(\theta))$



Pose Estimation using Levenberg-Marquard

Solve the non-linear Least-Squares Problem:

$$\min_{\theta} f(\theta) = \min_{\theta} \frac{1}{2} \sum_{i=1}^m r_i^2(\theta) = \min_{\theta} \frac{1}{2} \sum_{i=1}^m (\pi_i(g(\theta)) - \tilde{\mathbf{x}}'_i)^2$$

Apply **Levenberg-Marquard algorithm**:

0. $\lambda = 10^{-3}$ Start with an good initial estimate

For $k = 0, 1, \dots$ **Linearize the non-linear objective function** $f(\theta)$ using the first order Taylor expansion:

$$\begin{aligned} f(\theta_k + \delta) &\approx \frac{1}{2} \sum_{i=1}^m (r_i(\theta_k) + \nabla r_i(\theta) \delta)^2 \\ &= \frac{1}{2} \sum_{i=1}^m ((\pi_i(g(\theta_k)) - \tilde{\mathbf{x}}'_i) + \nabla(\pi_i(g(\theta)) - \tilde{\mathbf{x}}'_i) \delta)^2 \end{aligned}$$

Pose Estimation using Levenberg-Marquard

Solve linear Least-Square Problem:

$$\begin{aligned}\min_{\delta} l(\delta) &= \min_{\delta} \frac{1}{2} \sum_{i=1}^m ((\pi_i(g(\theta_k)) - \tilde{\mathbf{x}}'_i) + \nabla(\pi_i(g(\theta)) - \tilde{\mathbf{x}}'_i)\delta)^2 \\ &= \min_{\delta} \frac{1}{2} \|(\pi(g(\theta_k)) - \tilde{\mathbf{x}}') + \nabla(\pi(g(\theta)) - \tilde{\mathbf{x}}')\delta\|^2\end{aligned}$$

$$\Rightarrow 2 \cdot \frac{1}{2} ((\pi(g(\theta_k)) - \tilde{\mathbf{x}}') + \nabla(\pi(g(\theta)) - \tilde{\mathbf{x}}')\delta) \cdot \nabla(\pi(g(\theta)) - \tilde{\mathbf{x}}') = 0$$

$$(r(\theta_k) + J\delta) \cdot J = 0$$

$$J^{\top} r(\theta_k) + J^{\top} J\delta = 0$$

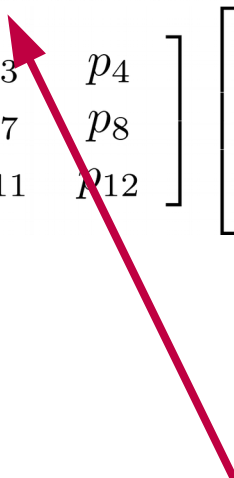
Pose Estimation using Levenberg-Marquardt

Compute Jacobians:

Transform 3D point into 2D image space using homogeneous coordinates:

$$\lambda \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & o_x \\ 0 & f & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}$$

$$= P \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}$$


Where

$$p = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{12} \end{bmatrix}$$

Note:

If K is already given in a **calibrated camera** setting then use given parameters.

Pose Estimation using Levenberg-Marquardt

Compute Jacobians:

Split up the image formation into two functions

$$\pi(p) = \pi(g(\theta))$$

We have to apply the Chain Rule!

Compute $\pi(p)$ which uses the elements of the projection matrix P to compute projected 2D image coordinates.

$$\pi(p) = \begin{bmatrix} \pi_{1x}(p) \\ \pi_{1y}(p) \\ \vdots \\ \pi_{mx}(p) \\ \pi_{my}(p) \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ \vdots \\ x'_m \\ y'_m \end{bmatrix} = \begin{bmatrix} \frac{p_1 x_1 + p_2 y_1 + p_3 z_1 + p_4}{p_9 x_1 + p_{10} y_1 + p_{11} z_1 + p_{12}} \\ \frac{p_5 x_1 + p_6 y_1 + p_7 z_1 + p_8}{p_9 x_1 + p_{10} y_1 + p_{11} z_1 + p_{12}} \\ \vdots \\ \frac{p_1 x_m + p_2 y_m + p_3 z_m + p_4}{p_9 x_m + p_{10} y_m + p_{11} z_m + p_{12}} \\ \frac{p_5 x_m + p_6 y_m + p_7 z_m + p_8}{p_9 x_m + p_{10} y_m + p_{11} z_m + p_{12}} \end{bmatrix}$$

The Jacobian of $\pi(p)$ can be computed as

$$J_\pi = \begin{bmatrix} \frac{\partial \pi_{1x}}{\partial p_1} & \dots & \frac{\partial \pi_{1x}}{\partial p_{12}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \pi_{my}}{\partial p_1} & \dots & \frac{\partial \pi_{my}}{\partial p_{12}} \end{bmatrix}$$

Pose Estimation using Levenberg-Marquardt

Compute Jacobians:

Split up the image formation into two functions

$$\pi(p) = \pi(g(\theta))$$

We have to apply the Chain Rule!

Compute $g(\theta)$ which uses the 3 camera matrix and 6 pose parameters to compute the projection matrix in vector form as

$$g(\theta) = \begin{bmatrix} g_1(\theta) \\ g_2(\theta) \\ \vdots \\ g_{12}(\theta) \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{12} \end{bmatrix}, \quad \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} = \begin{bmatrix} f & 0 & o_x \\ 0 & f & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}$$

The Jacobian of $g(\theta)$ can be computed as

$$J_g = \begin{bmatrix} \frac{\partial g_1}{\partial \theta_1} & \cdots & \frac{\partial g_1}{\partial \theta_9} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_{12}}{\partial \theta_1} & \cdots & \frac{\partial g_{12}}{\partial \theta_9} \end{bmatrix} \quad \begin{aligned} \theta &= [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7, \theta_8, \theta_9]^\top \\ &= [f, o_x, o_y, \theta_x, \theta_y, \theta_z, t_x, t_y, t_z]^\top \end{aligned}$$

Pose Estimation using Levenberg-Marquardt

In order to compute the $2m \times 9$ Jacobian of $\pi(g(\theta))$ we apply the chain rule and receive

$$J = J_\pi \cdot J_g = \begin{bmatrix} \frac{\partial \pi_{1x}}{\partial p_1} & \dots & \frac{\partial \pi_{1x}}{\partial p_{12}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \pi_{my}}{\partial p_1} & \dots & \frac{\partial \pi_{my}}{\partial p_{12}} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial g_1}{\partial \theta_1} & \dots & \frac{\partial g_1}{\partial \theta_9} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_{12}}{\partial \theta_1} & \dots & \frac{\partial g_{12}}{\partial \theta_9} \end{bmatrix}$$

Solve the linearized problem by solving

$$\theta_{k+1} = \theta_k + (J^\top J + \lambda \text{diag}(J^\top J))^{-1} (-J^\top (\pi(g(\theta_k)) - \tilde{\mathbf{x}}'))$$

$$\theta_{k+1} = \theta_k + (J^\top J + \lambda \text{diag}(J^\top J))^{-1} (-J^\top r(\theta_k))$$

if

$$f(x_k + \Delta x) < f(x_k)$$

$$x_{k+1} = x_k + \alpha \Delta x$$

else

$$\lambda = \lambda / 10$$

$$x_{k+1} = x_k$$

$$\lambda = \lambda * 10$$

Terminate if

$$\|x_{k+1} - x_k\| < \text{xtol}$$

Summary L05

- Orientation and Pose
- Non-linear Least-Squares for Pose Estimation
- Iterative Pose Estimation
 - Gauss Newton
 - Thrust Region Method: Levenberg-Marquardt

Literature

Nocedal and Wright, Numerical Optimization

Ch. 2: Fundamentals of Unconstrained Optimization

Ch. 10: Least-Squares Problems (p. 245-256)