

Positional Tracking II

Gordon Wetzstein
Stanford University

EE 267 Virtual Reality

Lecture 12

stanford.edu/class/ee267/



WARNING

- this class will be dense!
- will learn how to use nonlinear optimization (Levenberg-Marquardt algorithm) for pose estimation
- why ???
 - more accurate than homography method
 - can dial in lens distortion estimation, and estimation of intrinsic parameters (beyond this lecture, see lecture notes)
 - LM is very common in 3D computer vision → camera-based tracking

Pose Estimation - Overview

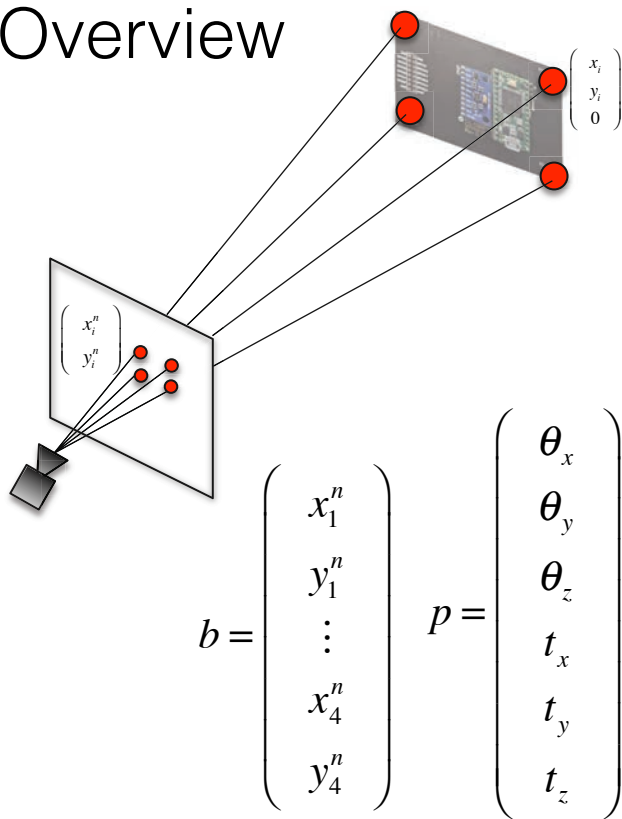
- goal: estimate pose via nonlinear least squares optimization

$$\underset{\{p\}}{\text{minimize}} \left\| b - f(g(p)) \right\|_2^2$$



image formation

- minimize reprojection error
- pose p is 6-element vector with 3 Euler angles and translation of VRduino w.r.t. base station



Overview

- review: gradients, Jacobian matrix, chain rule, iterative optimization
- nonlinear optimization: Gauss-Newton, Levenberg-Marquardt
- pose estimation using LM
- pose estimation with VRduino using nonlinear optimization

Review

Review: Gradients

- gradient of a function that depends on multiple variables:

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

$$f : \Re^n \rightarrow \Re$$

Review: The Jacobian Matrix

- gradient of a vector-valued function that depends on multiple variables:

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

$$f : \mathfrak{R}^n \rightarrow \mathfrak{R}^m, \quad J_f \in \mathfrak{R}^{m \times n}$$

Review: The Chain Rule

- here's how you've probably been using it so far:

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

- this rule applies when $f : \mathfrak{K} \rightarrow \mathfrak{K}$
 $g : \mathfrak{K} \rightarrow \mathfrak{K}$

Review: The Chain Rule

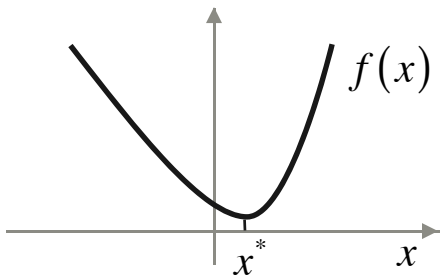
- here's how it is applied in general:

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

$$f : \mathfrak{R}^o \rightarrow \mathfrak{R}^m, \quad g : \mathfrak{R}^n \rightarrow \mathfrak{R}^o, \quad J_f \in \mathfrak{R}^{m \times o}, \quad J_g \in \mathfrak{R}^{o \times n}$$

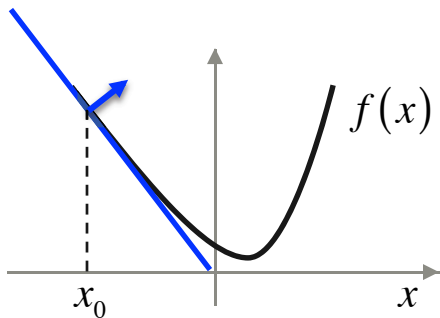
Review: Minimizing a Function

- goal: find point x^* that minimizes a nonlinear function $f(x)$



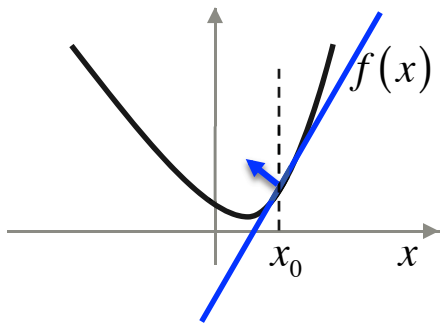
Review: What is a Gradient?

- gradient of f at some point x_0 is the slope at that point



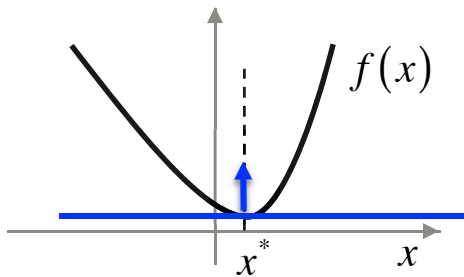
Review: What is a Gradient?

- gradient of f at some point x_0 is the slope at that point



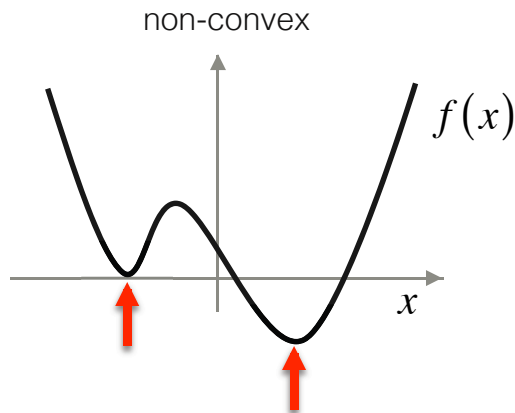
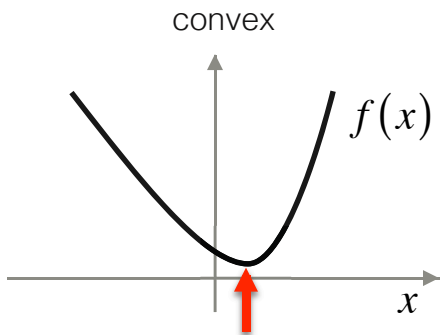
Review: What is a Gradient?

- extremum is where gradient is 0! (sometimes have to check 2nd derivative to see if it's a minimum and not a maximum or saddle point)



Review: Optimization

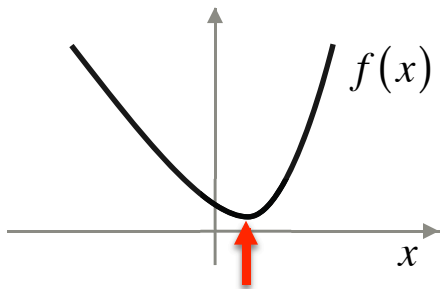
- extremum is where gradient is 0! (sometimes have to check 2nd derivative to see if it's a minimum and not a maximum or saddle point)



- convex optimization: there is only a single *global* minimum
- non-convex optimization: multiple *local* minima

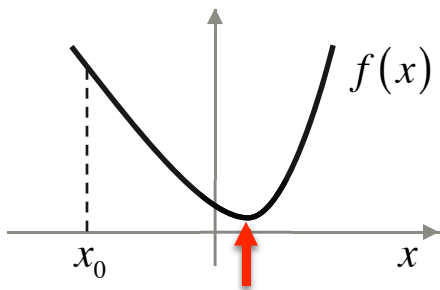
Review: Optimization

- how to find where gradient is 0?



Review: Optimization

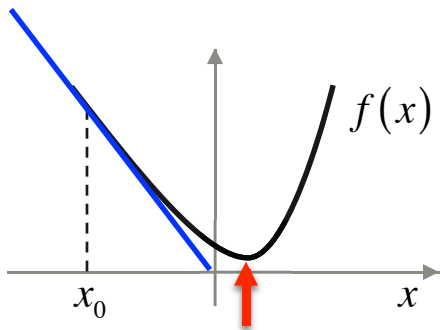
- how to find where gradient is 0?



1. start with some initial guess x_0 , e.g. a random value

Review: Optimization

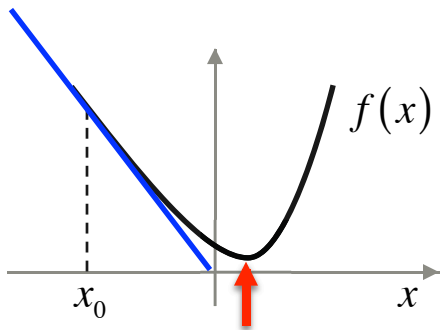
- how to find where gradient is 0?



1. start with some initial guess x_0 , e.g. a random value
2. update guess by linearizing function and minimizing that

Review: Optimization

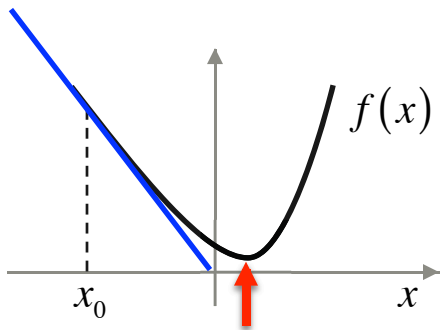
- how to linearize a function? → using Taylor expansion!



$$f(x_0 + \Delta x) \approx f(x_0) + J_f \Delta x + \varepsilon$$

Review: Optimization

- find minimum of linear function approximation

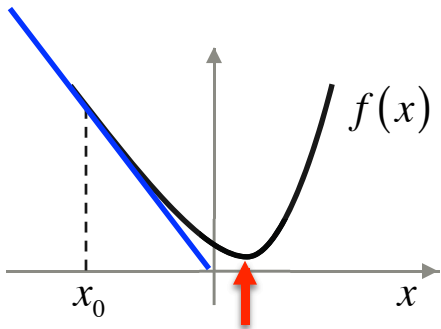


$$f(x_0 + \Delta x) \approx f(x_0) + J_f \Delta x + \varepsilon$$

$$\begin{aligned} \underset{\Delta x}{\text{minimize}} \quad & \|b - f(x_0 + \Delta x)\|_2^2 \\ & \approx \|b - (f(x_0) + J_f \Delta x)\|_2^2 \end{aligned}$$

Review: Optimization

- find minimum of linear function approximation (gradient=0)



$$f(x_0 + \Delta x) \approx f(x_0) + J_f \Delta x + \varepsilon$$

$$\begin{aligned} \underset{\Delta x}{\text{minimize}} \quad & \|b - f(x_0 + \Delta x)\|_2^2 \\ & \approx \|b - (f(x_0) + J_f \Delta x)\|_2^2 \end{aligned}$$

equate gradient to zero:

$$0 = J_f^T J_f \Delta x - J_f^T (b - f(x))$$

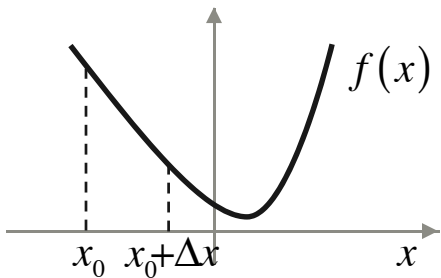


$$\Delta x = (J_f^T J_f)^{-1} J_f^T (b - f(x))$$

normal equations

Review: Optimization

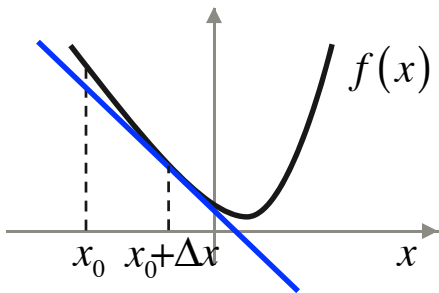
- take step and repeat procedure



$$\Delta x = \left(J_f^T J_f \right)^{-1} J_f^T (b - f(x))$$

Review: Optimization

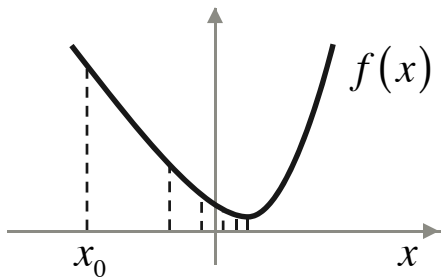
- take step and repeat procedure, will get there eventually



$$\Delta x = \left(J_f^T J_f \right)^{-1} J_f^T (b - f(x))$$

Review: Optimization – Gauss-Newton

- results in an iterative algorithm

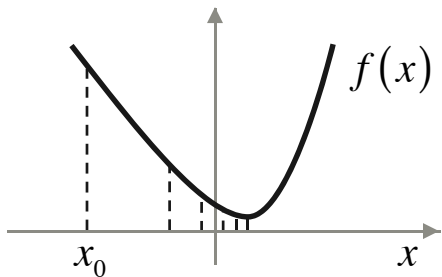


$$x_{k+1} = x_k + \Delta x$$

$$\Delta x = \left(J_f^T J_f \right)^{-1} J_f^T (b - f(x))$$

Review: Optimization – Gauss-Newton

- results in an iterative algorithm

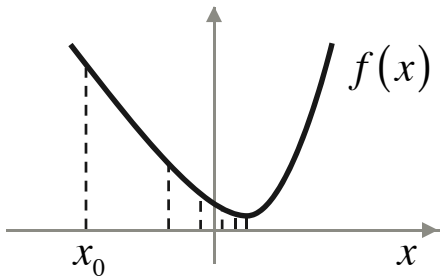


```
1. x = rand() // initialize x0
2. for k=1 to max_iter
3.   f = eval_objective(x)
4.   J = eval_jacobian(x)
5.   x = x + inv(J'*J)*J'*(b-f) // update x
```

$$\Delta x = \left(J_f^T J_f \right)^{-1} J_f^T (b - f(x))$$

Review: Optimization – Gauss-Newton

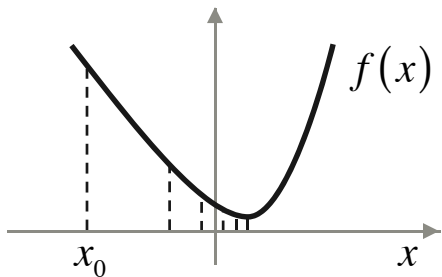
- matrix $J^T J$ can be ill-conditioned (i.e. not invertible)



$$\Delta x = \left(J_f^T J_f \right)^{-1} J_f^T (b - f(x))$$

Review: Optimization – Levenberg

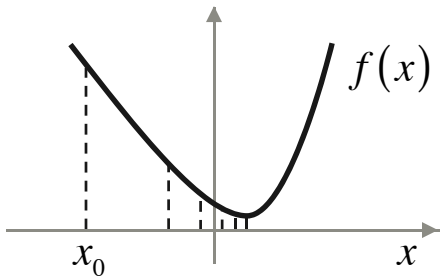
- matrix $J^T J$ can be ill-conditioned (i.e. not invertible)
- add a diagonal matrix to make invertible – acts as damping



$$\Delta x = \left(J_f^T J_f + \lambda I \right)^{-1} J_f^T (b - f(x))$$

Review: Optimization – Levenberg-Marquardt

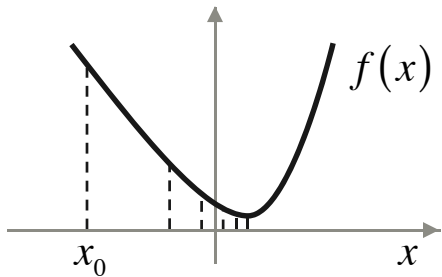
- matrix $J^T J$ can be ill-conditioned (i.e. not invertible)
- better: use $J^T J$ instead of I as damping. This is LM!



$$\Delta x = \left(J_f^T J_f + \lambda \operatorname{diag}(J_f^T J_f) \right)^{-1} J_f^T (b - f(x))$$

Review: Optimization – Levenberg-Marquardt

- matrix $J^T J$ can be ill-conditioned (i.e. not invertible)
- better: use $J^T J$ instead of I as damping. This is LM!



```
1. x = rand() // initialize x0
2. for k=1 to max_iter
3.   f = eval_objective(x)
4.   J = eval_jacobian(x)
5.   x = x+inv(J'*J+lambda*diag(J'*J))*J'*(b-f)
```

$$\Delta x = \left(J_f^T J_f + \lambda \text{diag}(J_f^T J_f) \right)^{-1} J_f^T (b - f(x))$$

Pose Estimation via Levenberg-Marquardt

Pose Estimation - Overview

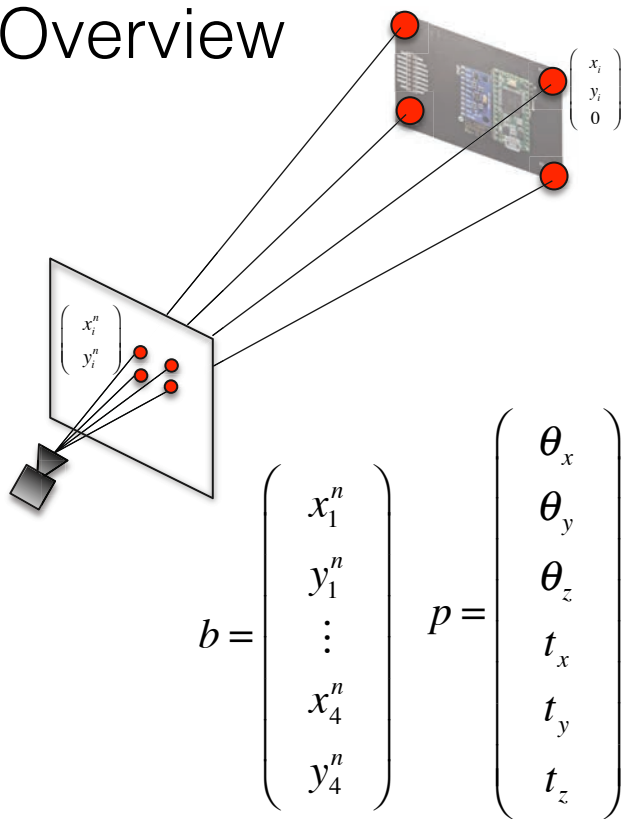
- goal: estimate pose via nonlinear least squares optimization

$$\underset{\{p\}}{\text{minimize}} \left\| b - f(g(p)) \right\|_2^2$$



image formation

- minimize reprojection error
- pose p is 6-element vector with 3 Euler angles and translation of VRduino w.r.t. base station



Pose Estimation - Objective Function

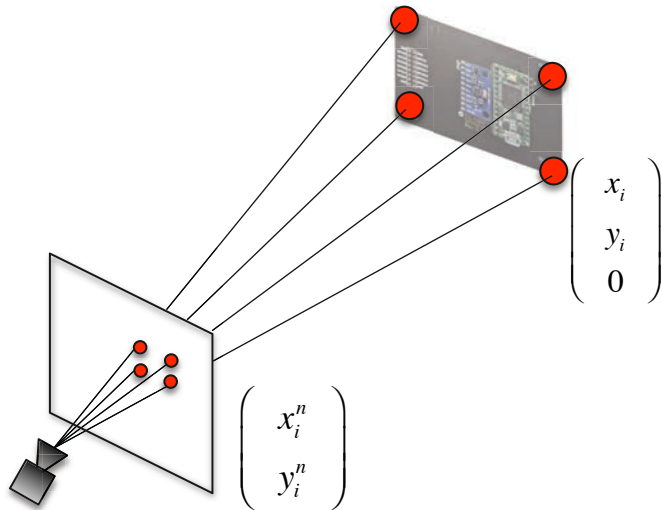
- goal: estimate pose via nonlinear least squares optimization

$$\underset{\{p\}}{\text{minimize}} \left\| b - f(g(p)) \right\|_2^2$$



image formation

- objective function is sum of squares of reprojection error



$$\left\| b - f(g(p)) \right\|_2^2 = \left(x_1^n - f_1(g(p)) \right)^2 + \left(y_1^n - f_2(g(p)) \right)^2 + \dots + \left(x_4^n - f_7(g(p)) \right)^2 + \left(y_4^n - f_8(g(p)) \right)^2$$

Image Formation

1. transform 3D point into view space:

$$\begin{pmatrix} x_i^c \\ y_i^c \\ z_i^c \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{pmatrix} \cdot \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$$
$$= \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix} \cdot \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$$

2. perspective divide:

$$\begin{pmatrix} x_i^n \\ y_i^n \end{pmatrix} = \begin{pmatrix} \frac{x_i^c}{z_i^c} \\ \frac{y_i^c}{z_i^c} \end{pmatrix}$$

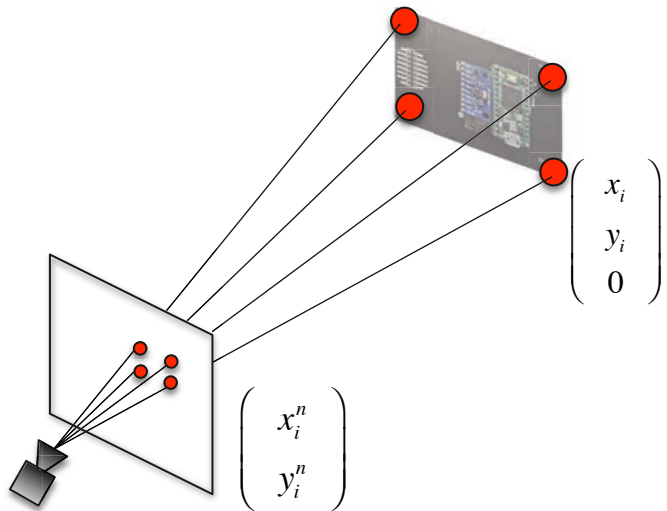


Image Formation: $g(p)$ and $f(h)$

- split up image formation into two functions

$$f(h) = f(g(p))$$

$$g : \mathfrak{R}^6 \rightarrow \mathfrak{R}^9, \quad f : \mathfrak{R}^9 \rightarrow \mathfrak{R}^8$$

Image Formation: $f(h)$

- $f(h)$ uses elements of homography matrix h to compute projected 2D coordinates as

$$f(h) = \begin{pmatrix} f_1(h) \\ f_2(h) \\ \vdots \\ f_7(h) \\ f_8(h) \end{pmatrix} = \begin{pmatrix} x_1^n \\ y_1^n \\ \vdots \\ x_4^n \\ y_4^n \end{pmatrix} = \begin{pmatrix} \frac{h_1x_1 + h_2y_1 + h_3}{h_7x_1 + h_8y_1 + h_9} \\ \frac{h_4x_1 + h_5y_1 + h_6}{h_7x_1 + h_8y_1 + h_9} \\ \vdots \\ \frac{h_1x_4 + h_2y_4 + h_3}{h_7x_4 + h_8y_4 + h_9} \\ \frac{h_4x_4 + h_5y_4 + h_6}{h_7x_4 + h_8y_4 + h_9} \end{pmatrix}$$

Jacobian Matrix of $f(h)$

$$f_1(h) = \frac{h_1 x_1 + h_2 y_1 + h_3}{h_7 x_1 + h_8 y_1 + h_9}$$

$$J_f = \begin{pmatrix} \frac{\partial f_1}{\partial h_1} & \dots & \frac{\partial f_1}{\partial h_9} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_8}{\partial h_1} & \dots & \frac{\partial f_8}{\partial h_9} \end{pmatrix}$$

- first row of Jacobian matrix



$$\frac{\partial f_1}{\partial h_1} = \frac{x_1}{h_7 x_1 + h_8 y_1 + h_9}$$

$$\frac{\partial f_1}{\partial h_2} = \frac{y_1}{h_7 x_1 + h_8 y_1 + h_9}$$

$$\frac{\partial f_1}{\partial h_3} = \frac{1}{h_7 x_1 + h_8 y_1 + h_9}$$

$$\frac{\partial f_1}{\partial h_4} = 0, \quad \frac{\partial f_1}{\partial h_5} = 0, \quad \frac{\partial f_1}{\partial h_6} = 0$$

$$\frac{\partial f_1}{\partial h_7} = - \left(\frac{h_1 x_1 + h_2 y_1 + h_3}{(h_7 x_1 + h_8 y_1 + h_9)^2} \right) x_1$$

$$\frac{\partial f_1}{\partial h_8} = - \left(\frac{h_1 x_1 + h_2 y_1 + h_3}{(h_7 x_1 + h_8 y_1 + h_9)^2} \right) y_1$$

$$\frac{\partial f_1}{\partial h_9} = - \frac{h_1 x_1 + h_2 y_1 + h_3}{(h_7 x_1 + h_8 y_1 + h_9)^2}$$

Jacobian Matrix of $f(h)$

$$J_f = \begin{pmatrix} \frac{\partial f_1}{\partial h_1} & \dots & \frac{\partial f_1}{\partial h_9} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_8}{\partial h_1} & \dots & \frac{\partial f_8}{\partial h_9} \end{pmatrix}$$

- the remaining rows of the Jacobian can be derived with a similar pattern
- see *course notes* for a detailed derivation of the elements of this Jacobian matrix

Image Formation: $g(p)$

- $g(p)$ uses 6 pose parameters to compute elements of homography matrix h as

$$g(p) = \begin{pmatrix} g_1(p) \\ \vdots \\ g_9(p) \end{pmatrix} = \begin{pmatrix} h_1 \\ \vdots \\ h_9 \end{pmatrix}$$

definition of homography matrix:

$$\begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{pmatrix}$$

rotation matrix from Euler angles:

$$R = R_z(\theta_z) \cdot R_x(\theta_x) \cdot R_y(\theta_y)$$
$$\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} = \begin{pmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) \\ 0 & \sin(\theta_x) & \cos(\theta_x) \end{pmatrix} \begin{pmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) \\ 0 & 1 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) \end{pmatrix}$$

Image Formation: $g(p)$

- write as

$$h_1 = g_1(p) = \cos(\theta_y)\cos(\theta_z) - \sin(\theta_x)\sin(\theta_y)\sin(\theta_z)$$

$$h_2 = g_2(p) = -\cos(\theta_x)\sin(\theta_z)$$

$$h_3 = g_3(p) = t_x$$

$$h_4 = g_4(p) = \cos(\theta_y)\sin(\theta_z) + \sin(\theta_x)\sin(\theta_y)\cos(\theta_z)$$

$$h_5 = g_5(p) = \cos(\theta_x)\cos(\theta_z)$$

$$h_6 = g_6(p) = t_y$$

$$h_7 = g_7(p) = \cos(\theta_x)\sin(\theta_y)$$

$$h_8 = g_8(p) = -\sin(\theta_x)$$

$$h_9 = g_9(p) = -t_z$$

Jacobian Matrix of $g(p)$

$$p = (p_1, p_2, p_3, p_4, p_5, p_6) = (\theta_x, \theta_y, \theta_z, t_x, t_y, t_z)$$

$$J_g = \begin{pmatrix} \frac{\partial g_1}{\partial p_1} & \dots & \frac{\partial g_1}{\partial p_6} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_9}{\partial p_1} & \dots & \frac{\partial g_9}{\partial p_6} \end{pmatrix}$$

$$h_1 = g_1(p) = \cos(\theta_y)\cos(\theta_z) - \sin(\theta_x)\sin(\theta_y)\sin(\theta_z)$$

$$h_2 = g_2(p) = -\cos(\theta_x)\sin(\theta_z)$$

$$h_3 = g_3(p) = t_x$$

$$h_4 = g_4(p) = \cos(\theta_y)\sin(\theta_z) + \sin(\theta_x)\sin(\theta_y)\cos(\theta_z)$$

$$h_5 = g_5(p) = \cos(\theta_x)\cos(\theta_z)$$

$$h_6 = g_6(p) = t_y$$

$$h_7 = g_7(p) = \cos(\theta_x)\sin(\theta_y)$$

$$h_8 = g_8(p) = -\sin(\theta_x)$$

$$h_9 = g_9(p) = -t_z$$

Jacobian Matrix of $g(p)$

$$p = (p_1, p_2, p_3, p_4, p_5, p_6) = (\theta_x, \theta_y, \theta_z, t_x, t_y, t_z)$$

$$J_g = \begin{pmatrix} \frac{\partial g_1}{\partial p_1} & \dots & \frac{\partial g_1}{\partial p_6} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_9}{\partial p_1} & \dots & \frac{\partial g_9}{\partial p_6} \end{pmatrix}$$

$$h_1 = g_1(p) = \cos(\theta_y)\cos(\theta_z) - \sin(\theta_x)\sin(\theta_y)\sin(\theta_z)$$

- first row of Jacobian matrix



$$\frac{\partial g_1}{\partial p_1} = -\cos(\theta_x)\sin(\theta_y)\sin(\theta_z)$$

$$\frac{\partial g_1}{\partial p_2} = -\sin(\theta_y)\cos(\theta_z) - \sin(\theta_x)\cos(\theta_y)\sin(\theta_z)$$

$$\frac{\partial g_1}{\partial p_3} = -\cos(\theta_y)\sin(\theta_z) - \sin(\theta_x)\sin(\theta_y)\cos(\theta_z)$$

$$\frac{\partial g_1}{\partial p_4} = 0, \quad \frac{\partial g_1}{\partial p_5} = 0, \quad \frac{\partial g_1}{\partial p_6} = 0$$

Jacobian Matrix of $g(p)$

$$J_g = \begin{pmatrix} \frac{\partial g_1}{\partial p_1} & \dots & \frac{\partial g_1}{\partial p_6} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_9}{\partial p_1} & \dots & \frac{\partial g_9}{\partial p_6} \end{pmatrix}$$

- the remaining rows of the Jacobian can be derived with a similar pattern
- see *course notes* for a detailed derivation of the elements of this Jacobian matrix

Jacobian Matrices of f and g

- to get the Jacobian of $f(g(p))$, compute the two Jacobian matrices and multiply them

$$J = J_f \cdot J_g = \begin{pmatrix} \frac{\partial f_1}{\partial h_1} & \dots & \frac{\partial f_1}{\partial h_9} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_8}{\partial h_1} & \dots & \frac{\partial f_8}{\partial h_9} \end{pmatrix} \cdot \begin{pmatrix} \frac{\partial g_1}{\partial p_1} & \dots & \frac{\partial g_1}{\partial p_6} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_9}{\partial p_1} & \dots & \frac{\partial g_9}{\partial p_6} \end{pmatrix}$$

Pose Tracking with LM

- LM then iteratively updates pose as


$$p^{(k+1)} = p^{(k)} + \left(J^T J + \lambda \operatorname{diag}(J^T J) \right)^{-1} J^T \left(b - f(p^{(k)}) \right)$$

- pseudo-code
 1. `p = ... // initialize p0`
 2. `for k=1 to max_iter`
 3. `f = eval_objective(p)`
 4. `J = get_jacobian(p)`
 5. `p = p + inv(J'*J+lambda*diag(J'*J))*J'*(b-f)`


Pose Tracking with LM

1. value = function eval_objective(p)

2. for i=1:4

3. value(2*(i-1)) = ... 

$$\frac{h_1x_i + h_2y_i + h_3}{h_7x_i + h_8y_i + h_9}$$

4. value(2*(i-1)+1) = ... 

$$\frac{h_4x_i + h_5y_i + h_6}{h_7x_i + h_8y_i + h_9}$$

$$\frac{h_4x_i + h_5y_i + h_6}{h_7x_i + h_8y_i + h_9}$$


Pose Tracking with LM


1. `J = function get_jacobian(p)`

2. `Jf = get_jacobian_f(p)`

3. `Jg = get_jacobian_g(p)`

4. `J = Jf*Jg`


$$J_f = \begin{pmatrix} \frac{\partial f_1}{\partial h_1} & \dots & \frac{\partial f_1}{\partial h_9} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_8}{\partial h_1} & \dots & \frac{\partial f_8}{\partial h_9} \end{pmatrix}$$


$$J_g = \begin{pmatrix} \frac{\partial g_1}{\partial p_1} & \dots & \frac{\partial g_1}{\partial p_6} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_9}{\partial p_1} & \dots & \frac{\partial g_9}{\partial p_6} \end{pmatrix}$$

Pose Tracking with LM on VRduino

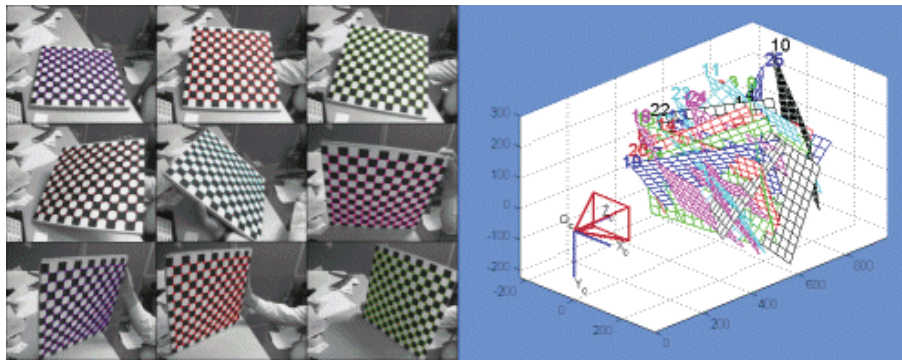
- some more hints for implementation:
 - let Arduino Matrix library compute matrix-matrix multiplications and also matrix inverses for you!
 - run homography method and use that to initialize p for LM
 - use something like 5-25 iterations of LM per frame for real-time performance
 - user-defined parameter λ
 - good luck!

Pose Tracking with LM on VRduino

live demo

Outlook: Camera Calibration

- camera calibration is one of the most fundamental problems in computer vision and imaging
- task: estimate intrinsic (lens distortion, focal length, principle point) & extrinsic (translation, rotation) camera parameters given images of planar checkerboards
- uses similar procedure as discussed today



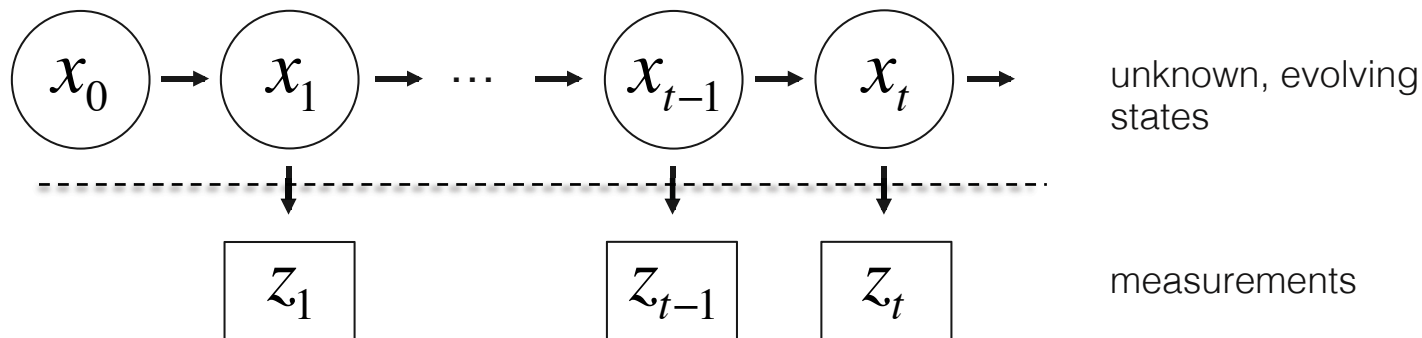
Outlook: Sensor Fusion with Extended Kalman Filter

- also desirable: estimate bias of each of all IMU sensors
- also desirable: joint pose estimation from all IMU + photodiode measurements
- can do all of that with an Extended Kalman Filter - slightly too advanced for this class, but you can find a lot of literature in the robotic vision community

Outlook: Sensor Fusion with Extended Kalman Filter

- Extended Kalman filter: can be interpreted as a Bayesian framework for sensor fusion
- Hidden Markov Model (HMM)

known initial
state



Must read: course notes on tracking!