UNIVERSITEIT
VAN
AMSTERDAM

Part I: Hartley and Zisserman Appendix 6:

Iterative estimation methods

Part II: Zhengyou Zhang:

A Flexible New Technique for Camera Calibration

Presented by Daniel Fontijne

UNIVERSITEIT VAN AMSTERDAM

HZ Appendix 6: Iterative estimation methods

Topics:

- Basic methods: Newton, Gauss-Newton, gradient descent.
- Levenberg-Marquardt.
- Sparse Levenberg-Marquardt.
- Applications to homography, fundamental matrix, bundle adjustment.
- Sparse methods for equations solving.
- Robust cost functions.
- Parameterization.

Lecture notes which I found useful

(methods for non-linear least squares problems):

http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3215/pdf/imm3215.pdf

UNIVERSITEIT VAN AMSTERDAM

Iterative estimation methods

Problem: how to find minimum of non-linear functions?

Iterative estimation methods

Problem: how to find minimum of non-linear functions?

Examples of HZ problems:
-homography estimation.
-fundamental matrix estimation.
-multiple image bundle adjustment.
-camera calibration (Zhang paper).

Iterative estimation methods

Problem: how to find minimum of non-linear functions?

Examples of HZ problems:
-homography estimation.
-fundamental matrix estimation.
-multiple image bundle adjustment.
-camera calibration (Zhang paper).

Examples of my recent problems:
-optimization of skeleton geometry given marker data.
-optimization of skeleton pose given marker data.

Iterative estimation methods

Problem: how to find minimum of non-linear functions?

Examples of HZ problems:
-homography estimation.
-fundamental matrix estimation.
-multiple image bundle adjustment.
-camera calibration (Zhang paper).

Examples of my recent problems:
-optimization of skeleton geometry given marker data.
-optimization of skeleton pose given marker data.

Central approach of Appendix 6: Levenberg-Marquardt.

Questions: Pronunciation? Why LM?

Universiteit van Amsterdam

Newton iteration

Goal: minimize $\mathbf{X} = \mathbf{f}(\mathbf{P})$ for $\mathbf{P}$.

$\mathbf{X}$ is the measurement vector.

$\mathbf{P}$ is the parameter vector.

$\mathbf{f}$ is some non-linear function.

UNIVERSITEIT
VAN
AMSTERDAM

Newton iteration

Goal: minimize $\mathbf{X} = \mathbf{f}(\mathbf{P})$ for $\mathbf{P}$.
$\mathbf{X}$ is the measurement vector.
$\mathbf{P}$ is the parameter vector.
$\mathbf{f}$ is some non-linear function.

In other words:
Minimize $\boldsymbol{\epsilon} = \mathbf{X} - \mathbf{f}(\mathbf{P})$.

Newton iteration

Goal: minimize $\mathbf{X} = \mathbf{f}(\mathbf{P})$ for $\mathbf{P}$.
$\mathbf{X}$ is the measurement vector.
$\mathbf{P}$ is the parameter vector.
$\mathbf{f}$ is some non-linear function.

In other words:
Minimize $\boldsymbol{\epsilon} = \mathbf{X} - \mathbf{f}(\mathbf{P})$.

We assume $\mathbf{f}$ is locally linear at each $\mathbf{P}_i$, then
$\mathbf{f}(\mathbf{P}_i + \Delta_i) = \mathbf{f}(\mathbf{P}_i) + \mathsf{J}_i\Delta_i$,
where matrix J_i is the Jacobian $\partial\mathbf{f}/\partial\mathbf{P}$ at $\mathbf{P}_i$.

Newton iteration

Goal: minimize $\mathbf{X} = \mathbf{f}(\mathbf{P})$ for $\mathbf{P}$.
$\mathbf{X}$ is the measurement vector.
$\mathbf{P}$ is the parameter vector.
$\mathbf{f}$ is some non-linear function.

In other words:
Minimize $\boldsymbol{\epsilon} = \mathbf{X} - \mathbf{f}(\mathbf{P})$.

We assume $\mathbf{f}$ is locally linear at each $\mathbf{P}_i$, then
$\mathbf{f}(\mathbf{P}_i + \Delta_i) = \mathbf{f}(\mathbf{P}_i) + \mathsf{J}_i\Delta_i$,
where matrix J_i is the Jacobian $\partial\mathbf{f}/\partial\mathbf{P}$ at $\mathbf{P}_i$.

So we want to minimize $\|\boldsymbol{\epsilon}_i + \mathsf{J}_i\Delta_i\|$ for some vector Δ_i.

Newton iteration

Goal: minimize $\mathbf{X} = \mathbf{f}(\mathbf{P})$ for $\mathbf{P}$.
$\mathbf{X}$ is the measurement vector.
$\mathbf{P}$ is the parameter vector.
$\mathbf{f}$ is some non-linear function.

In other words:
Minimize $\boldsymbol{\epsilon} = \mathbf{X} - \mathbf{f}(\mathbf{P})$.

We assume $\mathbf{f}$ is locally linear at each $\mathbf{P}_i$, then
$\mathbf{f}(\mathbf{P}_i + \Delta_i) = \mathbf{f}(\mathbf{P}_i) + \mathsf{J}_i\Delta_i$,
where matrix J_i is the Jacobian $\partial\mathbf{f}/\partial\mathbf{P}$ at $\mathbf{P}_i$.

So we want to minimize $\|\boldsymbol{\epsilon}_i + \mathsf{J}_i\Delta_i\|$ for some vector Δ_i.

Find Δ_i either using normal equations: $\mathsf{J}_i^\mathsf{T}\mathsf{J}_i\Delta = -\mathsf{J}_i^\mathsf{T}\boldsymbol{\epsilon}_i$
or using pseudo-inverse: $\Delta_i = -\mathsf{J}_i^+\boldsymbol{\epsilon}_i$.

Iterate until convergence . . .

UNIVERSITEIT
VAN
AMSTERDAM

Gauss-Newton method

Suppose we want to minimize some cost function
$g(\mathbf{P}) = \frac{1}{2}\|\boldsymbol{\epsilon}(\mathbf{P})\|^2 = \frac{1}{2}\boldsymbol{\epsilon}(\mathbf{P})^{\mathtt{T}}\boldsymbol{\epsilon}(\mathbf{P})$.

UNIVERSITEIT VAN AMSTERDAM

Gauss-Newton method

Suppose we want to minimize some cost function
$g(\mathbf{P}) = \frac{1}{2}\|\boldsymbol{\epsilon}(\mathbf{P})\|^2 = \frac{1}{2}\boldsymbol{\epsilon}(\mathbf{P})^{\mathtt{T}}\boldsymbol{\epsilon}(\mathbf{P})$.

We may expand in a Taylor series up to second degree
$g(\mathbf{P}+\Delta) = g + g_{\mathbf{P}}\Delta + \Delta^{\mathtt{T}} g_{\mathbf{PP}}\Delta/2$,
where subscript $\mathbf{P}$ denotes differentiation.

UNIVERSITEIT VAN AMSTERDAM

Gauss-Newton method

Suppose we want to minimize some cost function
$g(\mathbf{P}) = \frac{1}{2}\|\boldsymbol{\epsilon}(\mathbf{P})\|^2 = \frac{1}{2}\boldsymbol{\epsilon}(\mathbf{P})^{\mathtt{T}}\boldsymbol{\epsilon}(\mathbf{P})$.

We may expand in a Taylor series up to second degree
$g(\mathbf{P} + \Delta) = g + g_{\mathbf{P}}\Delta + \Delta^{\mathtt{T}} g_{\mathbf{PP}}\Delta/2$,
where subscript $\mathbf{P}$ denotes differentiation.

Differentiating w.r.t. Δ, setting to zero results in $g_{\mathbf{PP}}\Delta = -g_{\mathbf{P}}$.
Using this equation we could compute Δ if we knew $g_{\mathbf{PP}}$ and $g_{\mathbf{P}}$.

Gauss-Newton method

Suppose we want to minimize some cost function
$g(\mathbf{P}) = \frac{1}{2}\|\boldsymbol{\epsilon}(\mathbf{P})\|^2 = \frac{1}{2}\boldsymbol{\epsilon}(\mathbf{P})^\mathsf{T}\boldsymbol{\epsilon}(\mathbf{P})$.

We may expand in a Taylor series up to second degree
$g(\mathbf{P}+\Delta) = g + g_\mathbf{P}\Delta + \Delta^\mathsf{T} g_\mathbf{PP}\Delta/2$,
where subscript $\mathbf{P}$ denotes differentiation.

Differentiating w.r.t. Δ, setting to zero results in $g_\mathbf{PP}\Delta = -g_\mathbf{P}$.
Using this equation we could compute Δ if we knew $g_\mathbf{PP}$ and $g_\mathbf{P}$.

Gradient vector: $g_\mathbf{P} = \boldsymbol{\epsilon}_\mathbf{P}^\mathsf{T}\boldsymbol{\epsilon} = \mathtt{J}^\mathsf{T}\boldsymbol{\epsilon}$. Intuition?

Hessian: $g_\mathbf{PP} = \boldsymbol{\epsilon}_\mathbf{P}^\mathsf{T}\boldsymbol{\epsilon}_\mathbf{P} + \boldsymbol{\epsilon}_\mathbf{PP}^\mathsf{T}\boldsymbol{\epsilon} \approx \mathtt{J}^\mathsf{T}\mathtt{J}$. Assume linear again . . .

Gauss-Newton method

Suppose we want to minimize some cost function
$g(\mathbf{P}) = \frac{1}{2}\|\boldsymbol{\epsilon}(\mathbf{P})\|^2 = \frac{1}{2}\boldsymbol{\epsilon}(\mathbf{P})^\mathtt{T}\boldsymbol{\epsilon}(\mathbf{P})$.

We may expand in a Taylor series up to second degree
$g(\mathbf{P}+\Delta) = g + g_\mathbf{P}\Delta + \Delta^\mathtt{T} g_{\mathbf{PP}}\Delta/2$,
where subscript $\mathbf{P}$ denotes differentiation.

Differentiating w.r.t. Δ, setting to zero results in $g_{\mathbf{PP}}\Delta = -g_\mathbf{P}$.
Using this equation we could compute Δ if we knew $g_{\mathbf{PP}}$ and $g_\mathbf{P}$.

Gradient vector: $g_\mathbf{P} = \boldsymbol{\epsilon}_\mathbf{P}^\mathtt{T}\boldsymbol{\epsilon} = \mathtt{J}^\mathtt{T}\boldsymbol{\epsilon}$. Intuition?

Hessian: $g_{\mathbf{PP}} = \boldsymbol{\epsilon}_\mathbf{P}^\mathtt{T}\boldsymbol{\epsilon}_\mathbf{P} + \boldsymbol{\epsilon}_{\mathbf{PP}}^\mathtt{T}\boldsymbol{\epsilon} \approx \mathtt{J}^\mathtt{T}\mathtt{J}$. Assume linear again . . .

Putting it all together we get $\mathtt{J}^\mathtt{T}\mathtt{J}\Delta = -\mathtt{J}^\mathtt{T}\boldsymbol{\epsilon}$.
So we arrive at the normal equations again.
(So what was the point?)

Gradient descent

Gradient descent or steepest descent searches in the direction of most rapid decrease $-g_{\mathbf{P}} = -\boldsymbol{\epsilon}_{\mathbf{P}}^{\mathsf{T}}\boldsymbol{\epsilon}$.

Gradient descent

Gradient descent or steepest descent searches in the direction of most rapid decrease $-g_{\mathbf{P}} = -\boldsymbol{\epsilon}_{\mathbf{P}}^{\mathsf{T}}\boldsymbol{\epsilon}$.

So we take steps $\lambda\Delta = -g_{\mathbf{P}}$ where λ controls the step size and is found through line search.

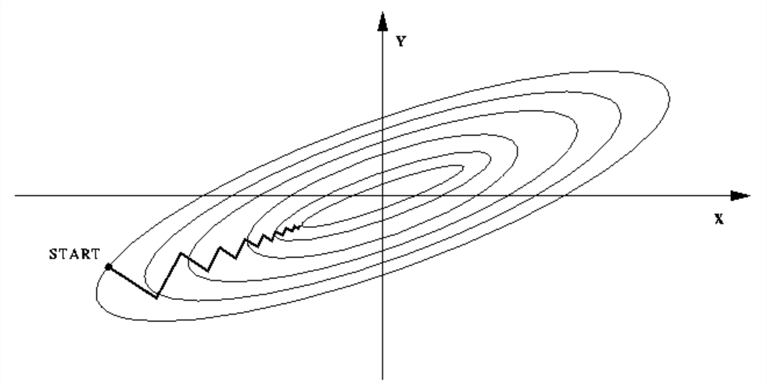
Universiteit van Amsterdam

Gradient descent

Gradient descent or steepest descent searches in the direction of most rapid decrease $-g_{\mathbf{P}} = -\boldsymbol{\epsilon}_{\mathbf{P}}^{\mathsf{T}}\boldsymbol{\epsilon}$.

So we take steps $\lambda\Delta = -g_{\mathbf{P}}$ where λ controls the step size and is found through line search.

A problem is zig-zagging which can cause slow convergence:



Levenberg-Marquardt

Levenberg-Marquardt is a blend of Gauss-Newton and gradient descent. Update equation:

$$(\mathtt{J}^{\mathtt{T}}\mathtt{J} + \lambda \mathtt{I})\Delta = -\mathtt{J}^{\mathtt{T}}\boldsymbol{\epsilon}.$$

UNIVERSITEIT VAN AMSTERDAM

Levenberg-Marquardt

Levenberg-Marquardt is a blend of Gauss-Newton and gradient descent. Update equation:

$(\mathtt{J}^{\mathtt{T}}\mathtt{J} + \lambda \mathtt{I})\Delta = -\mathtt{J}^{\mathtt{T}}\boldsymbol{\epsilon}.$

Algorithm:

- Initially set $\lambda = 10^{-3}$.
- Try update equation.
- If improvement: divide λ by 10. I.e., shift towards Gauss-Newton.
- Else: multiply λ by 10. I.e., shift towards gradient descent.

UNIVERSITEIT VAN AMSTERDAM

Levenberg-Marquardt

Levenberg-Marquardt is a blend of Gauss-Newton and gradient descent. Update equation:

$(\mathtt{J}^{\mathtt{T}}\mathtt{J} + \lambda \mathtt{I})\Delta = -\mathtt{J}^{\mathtt{T}}\boldsymbol{\epsilon}.$

Algorithm:

- Initially set $\lambda = 10^{-3}$.
- Try update equation.
- If improvement: divide λ by 10. I.e., shift towards Gauss-Newton.
- Else: multiply λ by 10. I.e., shift towards gradient descent.

The idea is (?):
-take big gradient descent steps far away from minimum.
-take Gauss-Newton steps near (hopefully quadratic) minimum.

UNIVERSITEIT VAN AMSTERDAM

Sparse Levenberg-Marquardt 1/2

In many estimation problems, the Jacobian is sparse.
One should this to lower the time complexity (sometimes even from $O(n^3)$ to $O(n)$).

UNIVERSITEIT VAN AMSTERDAM

Sparse Levenberg-Marquardt 1/2

In many estimation problems, the Jacobian is sparse.
One should this to lower the time complexity (sometimes even from $O(n^3)$ to $O(n)$).

In the example, the parameters are partitioned into two blocks:
$\mathbf{P} = (\mathbf{a}^T, \mathbf{b}^T)^T$

The Jacobian then has the form $\mathtt{J} = [A|B]$, with
$A = [\partial\hat{\mathbf{X}}/\partial\mathbf{a}]$, $\quad B = [\partial\hat{\mathbf{X}}/\partial\mathbf{b}]$.

UNIVERSITEIT VAN AMSTERDAM

Sparse Levenberg-Marquardt 1/2

In many estimation problems, the Jacobian is sparse.
One should this to lower the time complexity (sometimes even from $O(n^3)$ to $O(n)$).

In the example, the parameters are partitioned into two blocks: $\mathbf{P} = (\mathbf{a}^T, \mathbf{b}^T)^T$

The Jacobian then has the form $\mathtt{J} = [A|B]$, with
$A = [\partial\hat{\mathbf{X}}/\partial\mathbf{a}], \qquad B = [\partial\hat{\mathbf{X}}/\partial\mathbf{b}]$.

Using A and B, the normal equations $(\mathtt{J}^\mathtt{T}\mathtt{J})\Delta = -\mathtt{J}^\mathtt{T}\boldsymbol{\epsilon}$ take on the the form

$$\left[\begin{array}{c|c} A^T A & A^T B \\ \hline B^T A & B^T B \end{array}\right] \left(\begin{array}{c} \boldsymbol{\delta}_\mathbf{a} \\ \hline \boldsymbol{\delta}_\mathbf{b} \end{array}\right) = \left(\begin{array}{c} A^T \boldsymbol{\epsilon} \\ \hline B^T \boldsymbol{\epsilon} \end{array}\right).$$

Sparse Levenberg-Marquardt 2/2

If the normal equations are written as (what's with the *?)

$$\begin{bmatrix} \mathtt{U}^* & \mathtt{W} \\ \mathtt{W}^T & \mathtt{V}^* \end{bmatrix} \begin{pmatrix} \boldsymbol{\delta}_{\mathbf{a}} \\ \boldsymbol{\delta}_{\mathbf{b}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\epsilon}_A \\ \boldsymbol{\epsilon}_B \end{pmatrix},$$

we can rewrite this to

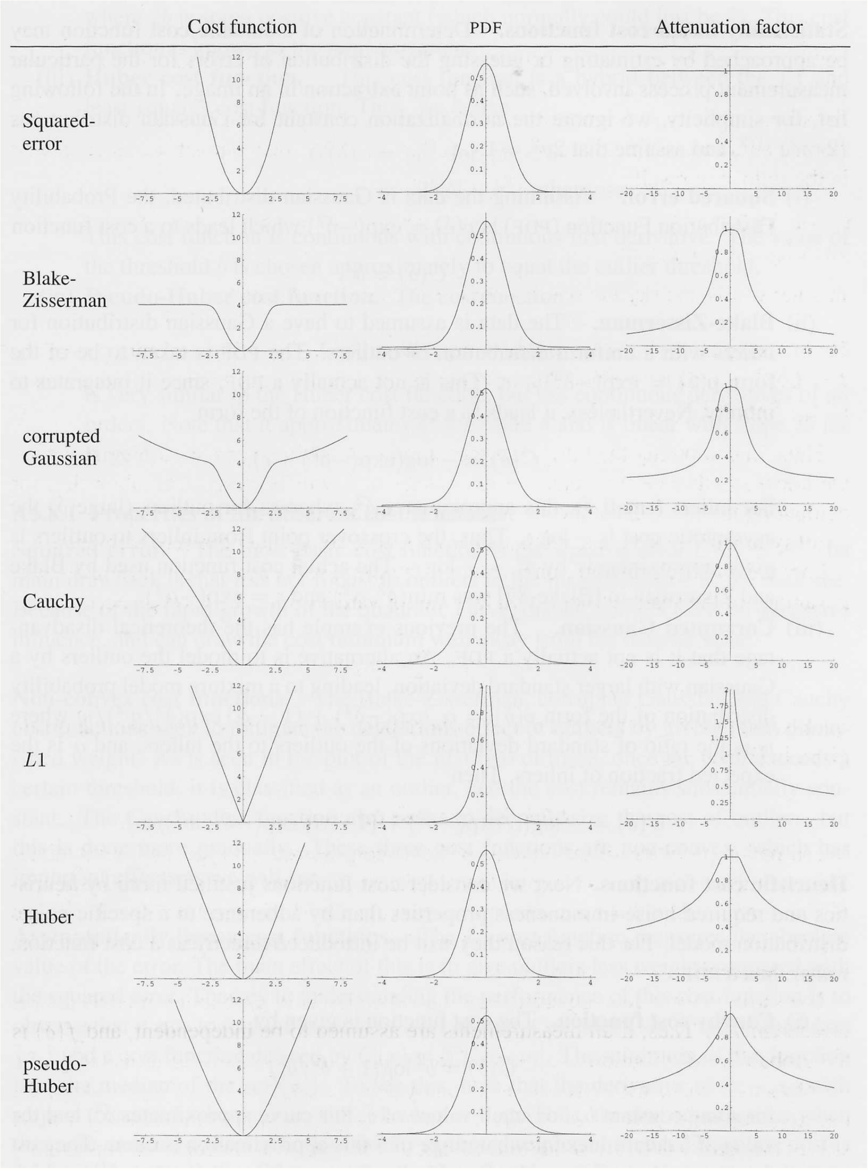
$$\begin{bmatrix} \mathtt{U}^* - \mathtt{W}\mathtt{V}^{*-1}\mathtt{W}^T & 0 \\ \mathtt{W}^T & \mathtt{V}^* \end{bmatrix} \begin{pmatrix} \boldsymbol{\delta}_{\mathbf{a}} \\ \boldsymbol{\delta}_{\mathbf{b}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\epsilon}_A - \mathtt{W}\mathtt{V}^{*-1}\boldsymbol{\epsilon}_B \\ \boldsymbol{\epsilon}_B \end{pmatrix}$$

by multiplying on the left by $\begin{bmatrix} \mathtt{I} & \mathtt{W}\mathtt{V}^{*-1} \\ 0 & \mathtt{I} \end{bmatrix}$.

Now first solve the top half, then the lower half using back-substitution.

UNIVERSITEIT
VAN
AMSTERDAM

Robust cost functions 1/5



Universiteit van Amsterdam

Robust cost functions 2/5

Squared-error is not usable unless outliers are filtered out.

UNIVERSITEIT VAN AMSTERDAM

Robust cost functions 2/5

Squared-error is not usable unless outliers are filtered out.

Alternatives:

- Blake-Zisserman: outliers are given a constant cost.
  Disadvantages: not a PDF, not convex.

Robust cost functions 2/5

Squared-error is not usable unless outliers are filtered out.

Alternatives:

- Blake-Zisserman: outliers are given a constant cost.
  Disadvantages: not a PDF, not convex.
- Corrupted Gaussian: blend two Gaussians, one for inliers and one for outliers.
  Disadvantages: not convex.

UNIVERSITEIT VAN AMSTERDAM

Robust cost functions 2/5

Squared-error is not usable unless outliers are filtered out.

Alternatives:

- Blake-Zisserman: outliers are given a constant cost.
  Disadvantages: not a PDF, not convex.
- Corrupted Gaussian: blend two Gaussians, one for inliers and one for outliers.
  Disadvantages: not convex.
- Cauchy: (?).
  disadvantages: not convex.

Robust cost functions 2/5

Squared-error is not usable unless outliers are filtered out.

Alternatives:

- Blake-Zisserman: outliers are given a constant cost.
  Disadvantages: not a PDF, not convex.
- Corrupted Gaussian: blend two Gaussians, one for inliers and one for outliers.
  Disadvantages: not convex.
- Cauchy: (?).
  disadvantages: not convex.
- L1: absolute error (not squared).
  Disadvantages: not differentiable at 0, minimum is not at a single point when summed.

Robust cost functions 2/5

Squared-error is not usable unless outliers are filtered out.

Alternatives:

- Blake-Zisserman: outliers are given a constant cost.
  Disadvantages: not a PDF, not convex.
- Corrupted Gaussian: blend two Gaussians, one for inliers and one for outliers.
  Disadvantages: not convex.
- Cauchy: (?).
  disadvantages: not convex.
- L1: absolute error (not squared).
  Disadvantages: not differentiable at 0, minimum is not at a single point when summed.
- Huber cost function: like L1, but 'rounded'.
  Disadvantages: non-continuous derivative from 2^{nd} order and up.

Robust cost functions 2/5

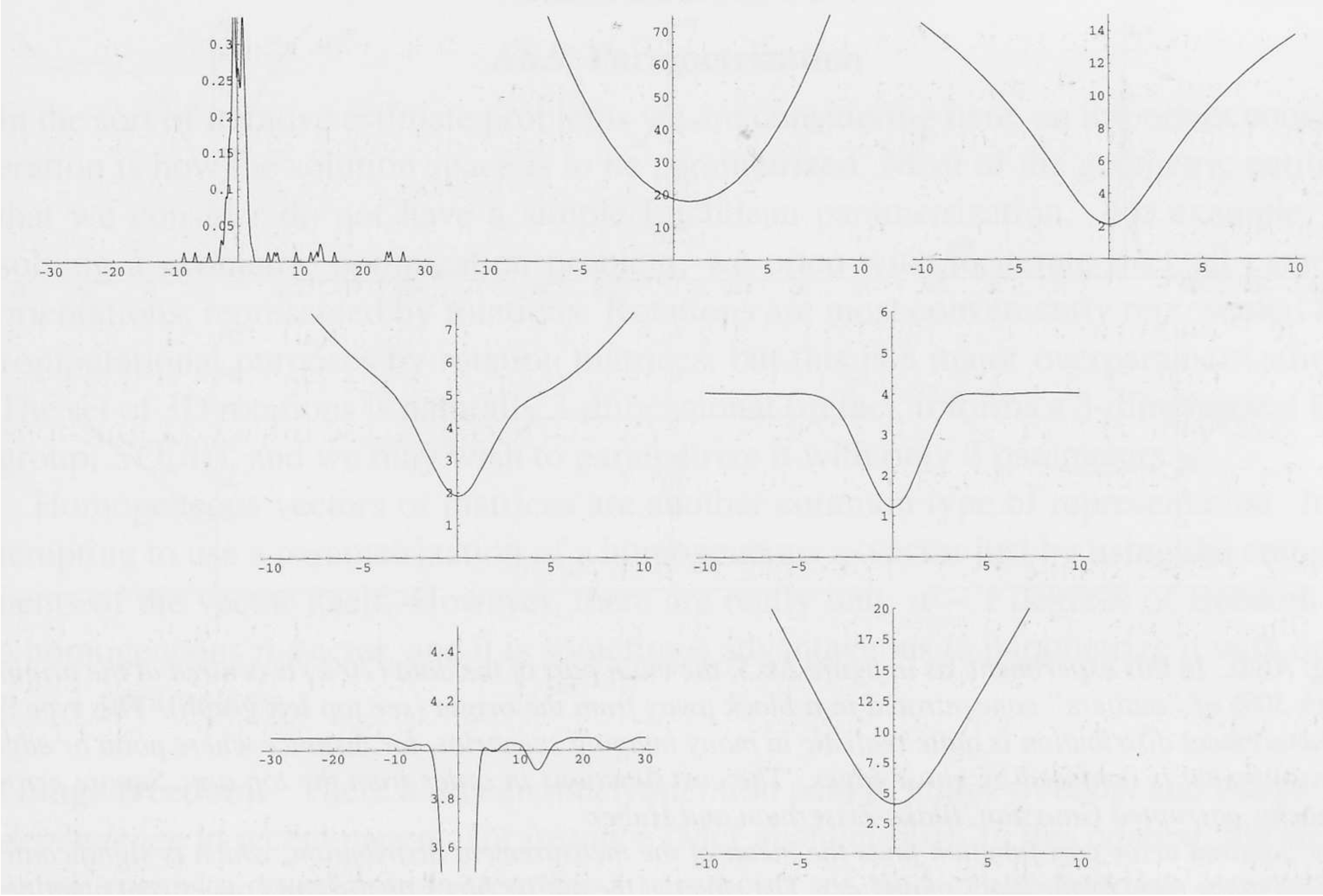
Squared-error is not usable unless outliers are filtered out.

Alternatives:

- Blake-Zisserman: outliers are given a constant cost.
  Disadvantages: not a PDF, not convex.
- Corrupted Gaussian: blend two Gaussians, one for inliers and one for outliers.
  Disadvantages: not convex.
- Cauchy: (?).
  disadvantages: not convex.
- L1: absolute error (not squared).
  Disadvantages: not differentiable at 0, minimum is not at a single point when summed.
- Huber cost function: like L1, but 'rounded'.
  Disadvantages: non-continuous derivative from 2^{nd} order and up.
- Pseudo Huber: like Huber, but with continuous derivatives.

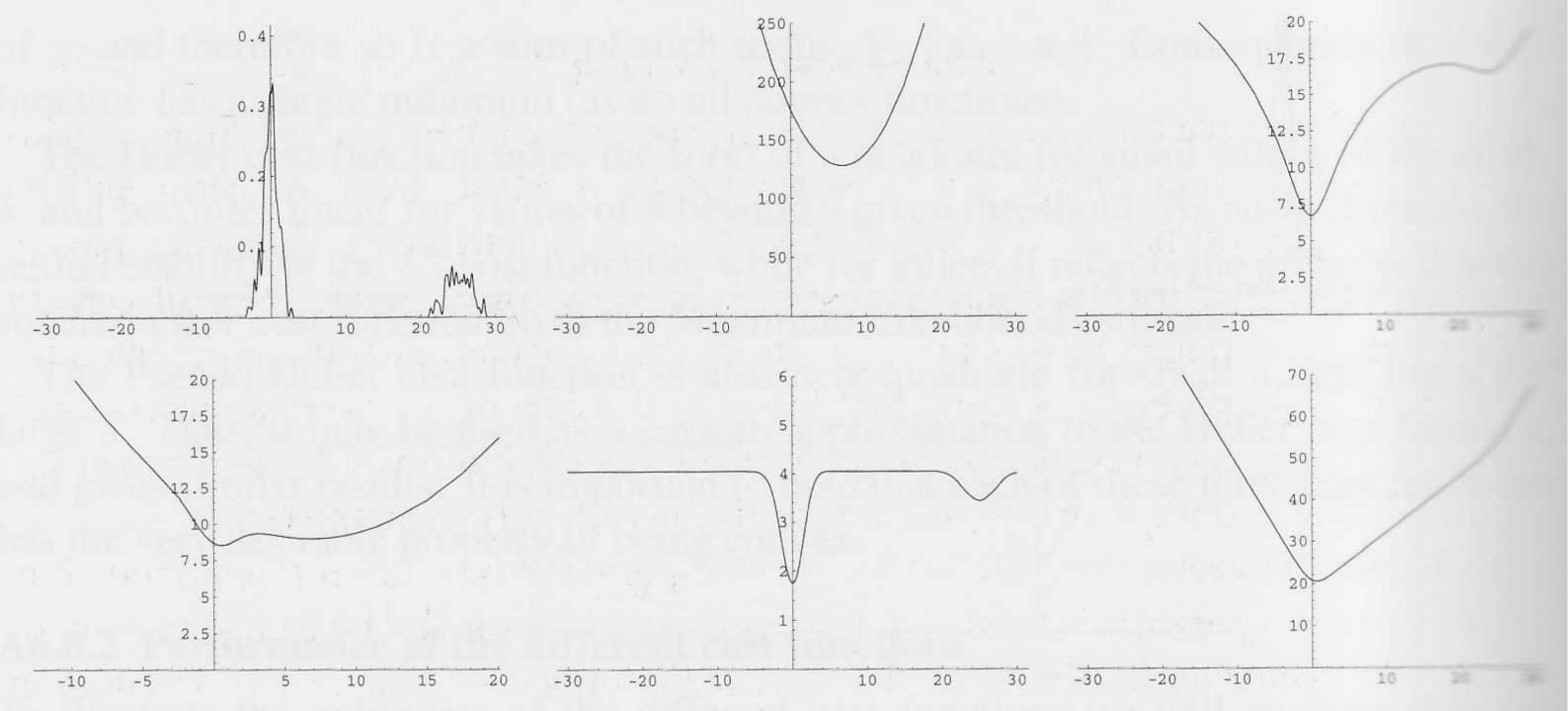
Robust cost functions 3/5

Figure A.6.5



Robust cost functions 4/5

Figure A.6.6



Universiteit van Amsterdam

Robust cost functions 5/5

Summary:

- Squared-error cost function is very susceptible to outliers.

Robust cost functions 5/5

Summary:

- Squared-error cost function is very susceptible to outliers.
- The non-convex functions (like L1 and corrupted Gaussian) may be good, but they have local minima. So do not use them unless already close to true minimum.

Robust cost functions 5/5

Summary:

- Squared-error cost function is very susceptible to outliers.
- The non-convex functions (like L1 and corrupted Gaussian) may be good, but they have local minima. So do not use them unless already close to true minimum.
- Best: Huber and Pseudo-Huber.

Universiteit van Amsterdam

Fooling LM implementations

Most implementations of Levenberg-Marquardt use the squared error cost function. What if you want a different cost function C instead?

UNIVERSITEIT VAN AMSTERDAM

Fooling LM implementations

Most implementations of Levenberg-Marquardt use the squared error cost function. What if you want a different cost function C instead?

Replace the each difference δ_i with a weighted version

$$\delta_i' = w_i \delta_i$$

such that

$$\|\delta_i\|^2 = w_i^2 \|\delta_i\|^2 = C(\|\delta_i\|).$$

UNIVERSITEIT VAN AMSTERDAM

Fooling LM implementations

Most implementations of Levenberg-Marquardt use the squared error cost function. What if you want a different cost function C instead?

Replace the each difference δ_i with a weighted version

$$\delta_i' = w_i \delta_i$$

such that

$$\|\delta_i\|^2 = w_i^2 \|\delta_i\|^2 = C(\|\delta_i\|).$$

Thus

$$w_i = \frac{\sqrt{C(\|\delta_i\|)}}{\|\delta_i\|}.$$

(confusion about δ being a vector? why not scalar?)

Parameterization for Levenberg-Marquardt

A good parameterization for use with LM is singularity free (at least in area visited during optimization). This means:

- continuous,
- differentiable,
- one-to-one.

Universiteit van Amsterdam

Parameterization for Levenberg-Marquardt

A good parameterization for use with LM is singularity free (at least in area visited during optimization). This means:

- continuous,
- differentiable,
- one-to-one.

So latitude-longitude is not suitable to parameterize sphere.

Parameterization for Levenberg-Marquardt

A good parameterization for use with LM is singularity free (at least in area visited during optimization). This means:

- continuous,
- differentiable,
- one-to-one.

So latitude-longitude is not suitable to parameterize sphere.

And Euler angles are not suitable to parameterize rotations.

UNIVERSITEIT VAN AMSTERDAM

Parameterization for Levenberg-Marquardt

A good parameterization for use with LM is singularity free (at least in area visited during optimization). This means:

- continuous,
- differentiable,
- one-to-one.

So latitude-longitude is not suitable to parameterize sphere.

And Euler angles are not suitable to parameterize rotations.

Gauge freedom?

UNIVERSITEIT VAN AMSTERDAM

Parameterization for Levenberg-Marquardt

A good parameterization for use with LM is singularity free (at least in area visited during optimization). This means:

- continuous,
- differentiable,
- one-to-one.

So latitude-longitude is not suitable to parameterize sphere.

And Euler angles are not suitable to parameterize rotations.

Gauge freedom?

Variance?

Parameterization of 3-D rotations

3-D rotation matrix: 9 elements, only 3 degrees of freedom.

Angle-axis (3-vector) representation: 3 elements, 3 d.o.f.

UNIVERSITEIT
VAN
AMSTERDAM

Parameterization of 3-D rotations

3-D rotation matrix: 9 elements, only 3 degrees of freedom.

Angle-axis (3-vector) representation: 3 elements, 3 d.o.f.

Observations: (these are mostly just general observations about log(rotation))

- Identity rotation: $\mathbf{t} = \mathbf{0}$.

UNIVERSITEIT
VAN
AMSTERDAM

Parameterization of 3-D rotations

3-D rotation matrix: 9 elements, only 3 degrees of freedom.

Angle-axis (3-vector) representation: 3 elements, 3 d.o.f.

Observations: (these are mostly just general observations about log(rotation))

- Identity rotation: $\mathbf{t} = \mathbf{0}$.
- Inverse rotation: $-\mathbf{t}$.

Universiteit van Amsterdam

Parameterization of 3-D rotations

3-D rotation matrix: 9 elements, only 3 degrees of freedom.

Angle-axis (3-vector) representation: 3 elements, 3 d.o.f.

Observations: (these are mostly just general observations about log(rotation))

- Identity rotation: $\mathbf{t} = \mathbf{0}$.
- Inverse rotation: $-\mathbf{t}$.
- Small rotation: the rotation matrix is $\mathtt{I} + [\mathbf{t}]_\times$.

Parameterization of 3-D rotations

3-D rotation matrix: 9 elements, only 3 degrees of freedom.

Angle-axis (3-vector) representation: 3 elements, 3 d.o.f.

Observations: (these are mostly just general observations about log(rotation))

- Identity rotation: $\mathbf{t} = \mathbf{0}$.
- Inverse rotation: $-\mathbf{t}$.
- Small rotation: the rotation matrix is $\mathtt{I} + [\mathbf{t}]_\times$.
- For small rotations: $\mathtt{R}(\mathbf{t}_1)\mathtt{R}(\mathbf{t}_2) \approx \mathtt{R}(\mathbf{t}_1 + \mathbf{t}_2)$.

Universiteit
van
Amsterdam

Parameterization of 3-D rotations

3-D rotation matrix: 9 elements, only 3 degrees of freedom.

Angle-axis (3-vector) representation: 3 elements, 3 d.o.f.

Observations: (these are mostly just general observations about log(rotation))

- Identity rotation: $\mathbf{t} = \mathbf{0}$.
- Inverse rotation: $-\mathbf{t}$.
- Small rotation: the rotation matrix is $\mathtt{I} + [\mathbf{t}]_\times$.
- For small rotations: $\mathtt{R}(\mathbf{t}_1)\mathtt{R}(\mathbf{t}_2) \approx \mathtt{R}(\mathbf{t}_1 + \mathbf{t}_2)$.
- All rotations can be represented by $\mathbf{t}$ with $\|\mathbf{t}\| \leq \pi$. When $\|\mathbf{t}\| = n2\pi$, (n positive integer) you get identity rotation again (singularity).

UNIVERSITEIT
VAN
AMSTERDAM

Parameterization of 3-D rotations

3-D rotation matrix: 9 elements, only 3 degrees of freedom.

Angle-axis (3-vector) representation: 3 elements, 3 d.o.f.

Observations: (these are mostly just general observations about log(rotation))

- Identity rotation: $\mathbf{t} = \mathbf{0}$.
- Inverse rotation: $-\mathbf{t}$.
- Small rotation: the rotation matrix is $\mathtt{I} + [\mathbf{t}]_\times$.
- For small rotations: $\mathtt{R}(\mathbf{t}_1)\mathtt{R}(\mathbf{t}_2) \approx \mathtt{R}(\mathbf{t}_1 + \mathbf{t}_2)$.
- All rotations can be represented by $\mathbf{t}$ with $\|\mathbf{t}\| \leq \pi$. When $\|\mathbf{t}\| = n2\pi$, (n positive integer) you get identity rotation again (singularity).
- Normalization: stay away from $\|\mathbf{t}\| = 2\pi$.

UNIVERSITEIT
VAN
AMSTERDAM

Parameterization of homogeneous vectors

Let $\mathbf{v}$ be a n-D-vector (already stripped of ‘extra’ homogeneous coordinate?).

Then parameterize it as $n+1$ vector:
$\bar{v} = (\text{sinc}(\|\mathbf{v}\|/2)\mathbf{v}^T, \cos(\|\mathbf{v}\|/2))^T$.

Parameterization of the n-sphere

How to parameterize unit vectors $\mathbf{x}$?

Compute Householder matrix (reflection) such that $\mathtt{H}_{\mathbf{v}(\mathbf{x})}\mathbf{x} = (0, \ldots, 0, 1)^{\mathtt{T}}$.

UNIVERSITEIT VAN AMSTERDAM

Parameterization of the n-sphere

How to parameterize unit vectors $\mathbf{x}$?

Compute Householder matrix (reflection) such that $\mathtt{H}_{\mathbf{v}(\mathbf{x})}\mathbf{x} = (0, \ldots, 0, 1)^{\mathtt{T}}$.

(i) $f(\mathbf{y}) = \hat{\mathbf{y}}/\|\hat{\mathbf{y}}\|$ where $\hat{\mathbf{y}} = (\mathbf{y}^T, 1)^T$, (?)
(ii) $f(\mathbf{y}) = (\mathrm{sinc}(\|\mathbf{y}\|/2)\mathbf{y}^T, \cos(\|\mathbf{y}\|/2))^T$ (?).
both have a Jacobian $\partial f/\partial \mathbf{y} = [\mathtt{I}|\mathbf{0}]^T$.

UNIVERSITEIT VAN AMSTERDAM

Parameterization of the n-sphere

How to parameterize unit vectors $\mathbf{x}$?

Compute Householder matrix (reflection) such that $\mathtt{H}_{\mathbf{v}(\mathbf{x})}\mathbf{x} = (0, \ldots, 0, 1)^{\mathtt{T}}$.

(i) $f(\mathbf{y}) = \hat{\mathbf{y}}/\|\hat{\mathbf{y}}\|$ where $\hat{\mathbf{y}} = (\mathbf{y}^T, 1)^T$, (?)
(ii) $f(\mathbf{y}) = (\text{sinc}(\|\mathbf{y}\|/2)\mathbf{y}^T, \cos(\|\mathbf{y}\|/2))^T$ (?).
both have a Jacobian $\partial f/\partial \mathbf{y} = [\mathtt{I}|\mathbf{0}]^T$.

So ‘constrained’ Jacobian can be computed

$$\mathtt{J} = \frac{\partial C}{\partial \mathbf{y}} = \frac{\partial C}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{y}} = \frac{\partial C}{\partial \mathbf{x}}\mathtt{H}_{\mathbf{v}(\mathbf{x})}\mathbf{x}[\mathtt{I}|\mathbf{0}]^T.$$

Zhang Paper

Zhengyou Zhang

A Flexible New Technique for Camera Calibration

(1998)

Zhang Paper

Zhengyou Zhang

A Flexible New Technique for Camera Calibration

(1998)

As implemented for:

Matlab The Camera Calibration Toolbox for Matlab

C++ Intel OpenCV

Internal Camera Calibration 1/4

Primary use of the Zhang algorithm is internal camera calibration. It computes:

- focal center c_x and c_y.
- focal length f_x and f_y.
- skew s (optional).

Internal Camera Calibration 1/4

Primary use of the Zhang algorithm is internal camera calibration. It computes:

- focal center c_x and c_y.
- focal length f_x and f_y.
- skew s (optional).

In short, the camera intrinsic matrix:

$$\mathbf{A} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Internal Camera Calibration 2/4

The Zhang algorithm also computes radial lens distortion parameters $[k_1, k_2, k_3, k_4]$.

The original paper uses
$x_d = x + x\,(k_1\,(x^2 + y^2) + k_2\,(x^2 + y^2)^2)$,
$y_d = y + y\,(k_1\,(x^2 + y^2) + k_2\,(x^2 + y^2)^2)$,
where x and y are normalized image coordinates and x_d and y_d are the distorted coordinates.

UNIVERSITEIT
VAN
AMSTERDAM

Internal Camera Calibration 2/4

The Zhang algorithm also computes radial lens distortion parameters $[k_1, k_2, k_3, k_4]$.

The original paper uses
$x_d = x + x\,(k_1\,(x^2 + y^2) + k_2\,(x^2 + y^2)^2)$,
$y_d = y + y\,(k_1\,(x^2 + y^2) + k_2\,(x^2 + y^2)^2)$,
where x and y are normalized image coordinates and x_d and y_d are the distorted coordinates.

But the implementations use a more complex model
$x_d = x + x\,(k_1(x^2 + y^2) + k_2\,(x^2 + y^2)^2) + x_{td}$,
$y_d = y + y\,(k_1(x^2 + y^2) + k_2\,(x^2 + y^2)^2) + y_{td}$,
where
$x_{td} = 2k_3\,x\,y + k_4\,(3\,x^2 + y^2)$,
$y_{td} = 2k_4\,x\,y + k_3\,(x^2 + 3\,y^2)$.

UNIVERSITEIT
VAN
AMSTERDAM

Internal Camera Calibration 3/4

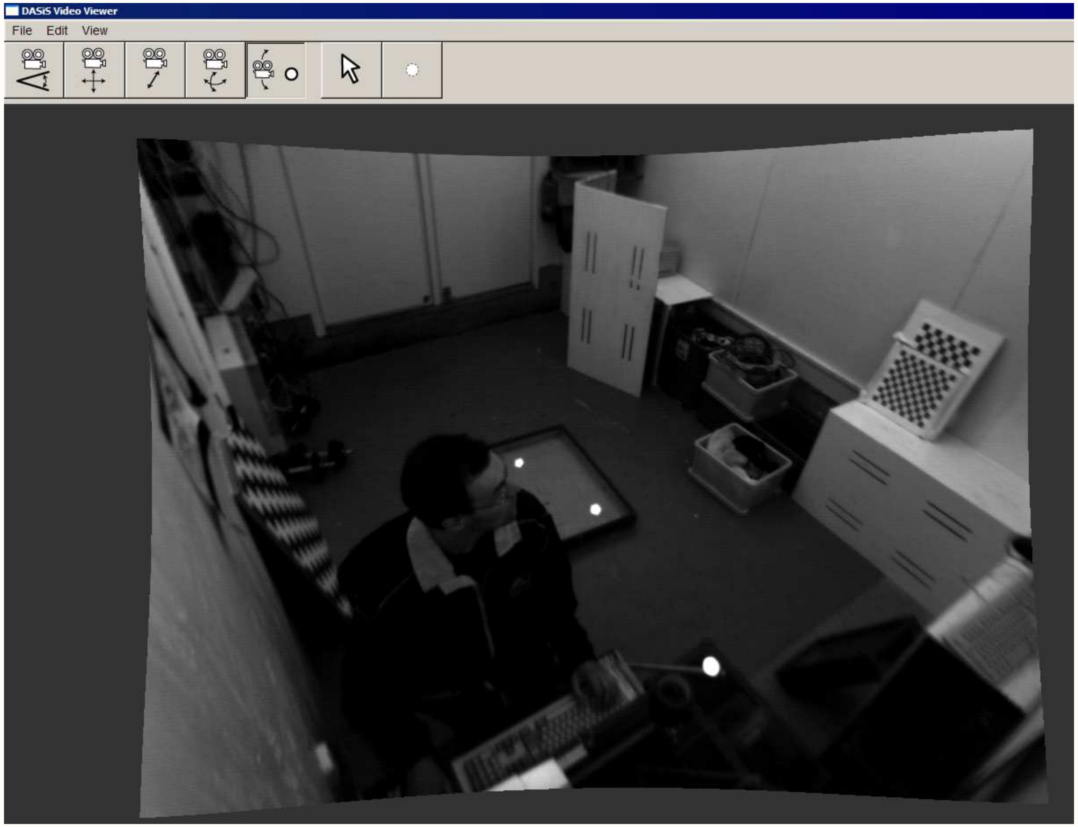
Example of internal camera calibration parameters.

Camera: PixeLINK A741, 2/3 inch CMOS sensor, 1280x1024.
Lens: Cosmicar 8.5mm fixed focal length.

$$f_x = 1272.872 \,\text{pixels} = 8.528\text{mm}$$
$$f_y = 1272.988 \,\text{pixels} = 8.529\text{mm}$$
$$c_x = 632.740$$
$$c_y = 507.648$$
$$k_1 = -0.204$$
$$k_2 = 0.171$$
$$k_3 = -0.00074896$$
$$k_4 = 0.00008878$$

Internal Camera Calibration 4/4

Show lens distortion in DASiS video viewer. . .



External Camera Calibration

The Zhang algorithm may also be used for external camera calibration.

Camera rotation and translation are computed as side-product of internal calibration.

If two cameras see the same calibration pattern at the same time, their relative position and orientation may be computed.

UNIVERSITEIT VAN AMSTERDAM

Overall approach

- Measure projected position of points in a plane (e.g., checkerboard).

Overall approach

- Measure projected position of points in a plane (e.g., checkerboard).
- Do so for at least two different camera orientations.

Overall approach

- Measure projected position of points in a plane (e.g., checkerboard).
- Do so for at least two different camera orientations.
- Setup equations in order to estimate camera intrinsics.

UNIVERSITEIT VAN AMSTERDAM

Overall approach

- Measure projected position of points in a plane (e.g., checkerboard).
- Do so for at least two different camera orientations.
- Setup equations in order to estimate camera intrinsics.
- Given camera intrinsics, estimate extrinsics.

Overall approach

- Measure projected position of points in a plane (e.g., checkerboard).
- Do so for at least two different camera orientations.
- Setup equations in order to estimate camera intrinsics.
- Given camera intrinsics, estimate extrinsics.
- Estimate radial distortion.

Overall approach

- Measure projected position of points in a plane (e.g., checkerboard).
- Do so for at least two different camera orientations.
- Setup equations in order to estimate camera intrinsics.
- Given camera intrinsics, estimate extrinsics.
- Estimate radial distortion.
- Use Levenberg-Marquardt to optimize initial estimates.

Basic Equations

Plane (‘checkerboard’) is at $Z = 0$.

Basic Equations

Plane ('checkerboard') is at $Z = 0$.

Homogeneous 2-D image point: $\widetilde{\mathbf{m}}$.

Homogeneous 3-D world point: $\widetilde{\mathtt{M}} = [X \;\; Y \;\; 0 \;\; 1]^T$.

Basic Equations

Plane ('checkerboard') is at $Z = 0$.

Homogeneous 2-D image point: $\widetilde{\mathbf{m}}$.
Homogeneous 3-D world point: $\widetilde{\mathtt{M}} = [X \;\; Y \;\; 0 \;\; 1]^T$.

Projection:

$$s\widetilde{\mathbf{m}} = \mathbf{A}[\mathbf{R} \;\; \mathbf{t}]\widetilde{\mathtt{M}} =$$

$$\begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} [\mathbf{r}_1 \;\; \mathbf{r}_2 \;\; \mathbf{r}_3 \;\; \mathbf{t}] \, [X \;\; Y \;\; 0 \;\; 1]^T =$$

$$\mathbf{A} \, [\mathbf{r}_1 \;\; \mathbf{r}_2 \;\; \mathbf{t}] \, [X \;\; Y \;\; 1]^T$$

Homography, constraints

An homography $\mathbf{H}$ can be estimated between known points on the calibration object and the measured world points.

$$\mathbf{H} = [\mathbf{h}_1 \ \ \mathbf{h}_2 \ \ \mathbf{h}_3] = \lambda \mathbf{A} [\mathbf{r}_1 \ \ \mathbf{r}_2 \ \ \mathbf{t}]$$

Homography, constraints

An homography $\mathbf{H}$ can be estimated between known points on the calibration object and the measured world points.

$$\mathbf{H} = [\mathbf{h}_1 \;\; \mathbf{h}_2 \;\; \mathbf{h}_3] = \lambda \mathbf{A} [\mathbf{r}_1 \;\; \mathbf{r}_2 \;\; \mathbf{t}]$$

We demand:

C1: $\mathbf{r}_1^T \mathbf{r}_2 = 0$ ($\mathbf{r}_1$, $\mathbf{r}_2$ orthogonal),

C2: $\mathbf{r}_1^T \mathbf{r}_1 = \mathbf{r}_2^T \mathbf{r}_2$ ($\mathbf{r}_1$, $\mathbf{r}_2$ have same length).

UNIVERSITEIT
VAN
AMSTERDAM

Homography, constraints

An homography $\mathbf{H}$ can be estimated between known points on the calibration object and the measured world points.

$$\mathbf{H} = [\mathbf{h}_1 \;\; \mathbf{h}_2 \;\; \mathbf{h}_3] = \lambda \mathbf{A} [\mathbf{r}_1 \;\; \mathbf{r}_2 \;\; \mathbf{t}]$$

We demand:
C1: $\mathbf{r}_1^T \mathbf{r}_2 = 0$ ($\mathbf{r}_1$, $\mathbf{r}_2$ orthogonal),
C2: $\mathbf{r}_1^T \mathbf{r}_1 = \mathbf{r}_2^T \mathbf{r}_2$ ($\mathbf{r}_1$, $\mathbf{r}_2$ have same length).

We know:
$$\mathbf{h}_1 = \lambda \mathbf{A}\, \mathbf{r}_1 \quad \rightarrow \quad \mathbf{r}_1 = \lambda^{-1} \mathbf{A}^{-1} \mathbf{h}_1$$
$$\mathbf{h}_2 = \lambda \mathbf{A}\, \mathbf{r}_2 \quad \rightarrow \quad \mathbf{r}_2 = \lambda^{-1} \mathbf{A}^{-1} \mathbf{h}_2$$

UNIVERSITEIT
VAN
AMSTERDAM

Homography, constraints

An homography $\mathbf{H}$ can be estimated between known points on the calibration object and the measured world points.

$$\mathbf{H} = [\mathbf{h}_1 \;\; \mathbf{h}_2 \;\; \mathbf{h}_3] = \lambda \mathbf{A} [\mathbf{r}_1 \;\; \mathbf{r}_2 \;\; \mathbf{t}]$$

We demand:

C1: $\mathbf{r}_1^T \mathbf{r}_2 = 0$ ($\mathbf{r}_1$, $\mathbf{r}_2$ orthogonal),

C2: $\mathbf{r}_1^T \mathbf{r}_1 = \mathbf{r}_2^T \mathbf{r}_2$ ($\mathbf{r}_1$, $\mathbf{r}_2$ have same length).

We know:

$$\mathbf{h}_1 = \lambda \mathbf{A}\, \mathbf{r}_1 \quad \rightarrow \quad \mathbf{r}_1 = \lambda^{-1} \mathbf{A}^{-1} \mathbf{h}_1$$
$$\mathbf{h}_2 = \lambda \mathbf{A}\, \mathbf{r}_2 \quad \rightarrow \quad \mathbf{r}_2 = \lambda^{-1} \mathbf{A}^{-1} \mathbf{h}_2$$

So the constraints are:

C1: $\mathbf{h}_1^T \, \mathbf{A}^{-T} \, \mathbf{A}^{-1} \, \mathbf{h}_2 = 0$,

C2: $\mathbf{h}_1^T \, \mathbf{A}^{-T} \, \mathbf{A}^{-1} \, \mathbf{h}_1 = \mathbf{h}_2^T \, \mathbf{A}^{-T} \, \mathbf{A}^{-1} \, \mathbf{h}_2$.

UNIVERSITEIT VAN AMSTERDAM

Closed-form solution using constraints 1/4

Using the constraints, we can first find $\mathbf{A}$, followed by $\mathbf{R}$ and $\mathbf{t}$. Let

$$\mathbf{B} = \mathbf{A}^{-T}\mathbf{A}^{-1} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{\alpha^2} & -\frac{\gamma}{\alpha^2\beta} & \frac{v_0\gamma - u_0\beta}{\alpha^2\beta} \\ -\frac{\gamma}{\alpha^2\beta} & \frac{\gamma^2}{\alpha^2\beta^2} + \frac{1}{\beta^2} & -\frac{\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} \\ \frac{v_0\gamma - u_0\beta}{\alpha^2\beta} & -\frac{\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} & \frac{(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2} + \frac{v_0^2}{\beta^2} + 1 \end{bmatrix}.$$

This allows to solve for α, β, etc.

Closed-form solution using constraints 2/4

If we reshuffle the six unique elements of $\mathbf{B}$ into a vector $\mathbf{b} = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]$,

we can rewrite both constraints as $\mathbf{h}_i^T \mathbf{B} \mathbf{h}_j = \mathbf{v}_{ij}^T \mathbf{b}$,

where

$$\mathbf{v}_{ij} = [h_{i1}h_{j1}, h_{i1}h_{j2} + h_{i2}h_{j1}, h_{i2}h_{j2}, \\ h_{i3}h_{j1} + h_{i1}h_{j3}, h_{i3}h_{j2} + h_{i2}h_{j3}, h_{i3}h_{j3}]^T,$$

ultimately resulting in

$$\begin{bmatrix} \mathbf{v}_{12}^T \\ (\mathbf{v}_{11} - \mathbf{v}_{22})^T \end{bmatrix} \mathbf{b} = 0.$$

Closed-form solution using constraints 3/4

Next, stack all the equations from n measurements (estimated homographies) of the plane ('checkerboard'):

$$\mathbf{V}\mathbf{b} = 0,$$

where $\mathbf{V}$ is a $2n \times 6$ matrix. Solve as usual using the SVD.

Closed-form solution using constraints 4/4

Once $\mathbf{A}$ is known, we can obtain $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{t}$:

$$\mathbf{r}_1 = \lambda^{-1}\mathbf{A}^{-1}\mathbf{h}_1,$$
$$\mathbf{r}_2 = \lambda^{-1}\mathbf{A}^{-1}\mathbf{h}_2,$$
$$\mathbf{t} = \lambda^{-1}\mathbf{A}^{-1}\mathbf{h}_3.$$

UNIVERSITEIT
VAN
AMSTERDAM

Closed-form solution using constraints 4/4

Once $\mathbf{A}$ is known, we can obtain $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{t}$:

$$\mathbf{r}_1 = \lambda^{-1}\mathbf{A}^{-1}\mathbf{h}_1,$$
$$\mathbf{r}_2 = \lambda^{-1}\mathbf{A}^{-1}\mathbf{h}_2,$$
$$\mathbf{t} = \lambda^{-1}\mathbf{A}^{-1}\mathbf{h}_3.$$

Now Zhang says
$\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$,
and use SVD to make matrix $\mathbf{R}$ orthogonal, i.e.,
$\mathbf{R} = \mathbf{U}\mathbf{V}^T$.

UNIVERSITEIT VAN AMSTERDAM

Closed-form solution using constraints 4/4

Once $\mathbf{A}$ is known, we can obtain $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{t}$:

$$\mathbf{r}_1 = \lambda^{-1}\mathbf{A}^{-1}\mathbf{h}_1,$$
$$\mathbf{r}_2 = \lambda^{-1}\mathbf{A}^{-1}\mathbf{h}_2,$$
$$\mathbf{t} = \lambda^{-1}\mathbf{A}^{-1}\mathbf{h}_3.$$

Now Zhang says

$\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$,

and use SVD to make matrix $\mathbf{R}$ orthogonal, i.e.,

$\mathbf{R} = \mathbf{U}\mathbf{V}^T$.

I say:

Make $\mathbf{r}_1$, $\mathbf{r}_2$ orthogonal in least-squares sense.

The compute $\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$.

Is simpler and boils down to the same thing.

Radial distortion

Using the camera intrinsics and extrinsics undistorted coordinates of points (corners on the checkerboard) can be approximated. These is used to solve for k_1, k_2:

$$\begin{bmatrix} (u-u_0)(x^2+y^2) & (u-u_0)(x^2+y^2)^2 \\ (v-v_0)(x^2+y^2) & (v-v_0)(x^2+y^2)^2 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} \breve{u}-u \\ \breve{v}-v \end{bmatrix}.$$

Radial distortion

Using the camera intrinsics and extrinsics undistorted coordinates of points (corners on the checkerboard) can be approximated. These is used to solve for k_1, k_2:

$$\begin{bmatrix} (u-u_0)(x^2+y^2) & (u-u_0)(x^2+y^2)^2 \\ (v-v_0)(x^2+y^2) & (v-v_0)(x^2+y^2)^2 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} \breve{u}-u \\ \breve{v}-v \end{bmatrix}.$$

These equations are stacked ($\mathbf{D}[k_1 \;\; k_2]^T = \mathbf{d}$) and we solve least squares $[k_1 \;\; k_2]^T = (\mathbf{D}^T\mathbf{D})^{-1}\mathbf{D}^T\mathbf{d}$.

Then iterate both algorithm (internal+external, radial) until convergence.

Maximum likelihood estimation

Optimize: use Levenberg-Marquardt to find minimum of

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \|\mathbf{m}_{ij} - \breve{\mathbf{m}}(\mathbf{A}, k_1, k_2, \mathbf{R}_i, \mathbf{t}_i, \mathtt{M}_i)\|^2$$

(n images, m points per image)

All done . . .

Notable experimental results

- Using three different images, the results are pretty good. Results keep getting better with more images.

Notable experimental results

- Using three different images, the results are pretty good. Results keep getting better with more images.
- 45 degree angle between image plane and checkerboard seems to give best result. Loss of precision in corner detection was not taken into account (simulated data).

UNIVERSITEIT VAN AMSTERDAM

Notable experimental results

- Using three different images, the results are pretty good. Results keep getting better with more images.
- 45 degree angle between image plane and checkerboard seems to give best result. Loss of precision in corner detection was not taken into account (simulated data).
- Systematic non-planarity of checkerboard has more effect than random noise (duh).

UNIVERSITEIT
VAN
AMSTERDAM

Notable experimental results

- Using three different images, the results are pretty good. Results keep getting better with more images.
- 45 degree angle between image plane and checkerboard seems to give best result. Loss of precision in corner detection was not taken into account (simulated data).
- Systematic non-planarity of checkerboard has more effect than random noise (duh).
- Cylindrical non-planarity is worse than spherical non-planarity (cylindrical more common in practice?).

Notable experimental results

- Using three different images, the results are pretty good. Results keep getting better with more images.
- 45 degree angle between image plane and checkerboard seems to give best result. Loss of precision in corner detection was not taken into account (simulated data).
- Systematic non-planarity of checkerboard has more effect than random noise (duh).
- Cylindrical non-planarity is worse than spherical non-planarity (cylindrical more common in practice?).
- Even with systematic non-planarity, results still usable.

Universiteit van Amsterdam

Notable experimental results

- Using three different images, the results are pretty good. Results keep getting better with more images.
- 45 degree angle between image plane and checkerboard seems to give best result. Loss of precision in corner detection was not taken into account (simulated data).
- Systematic non-planarity of checkerboard has more effect than random noise (duh).
- Cylindrical non-planarity is worse than spherical non-planarity (cylindrical more common in practice?).
- Even with systematic non-planarity, results still usable.
- Error in compute sensor center seems not to have too much effect in 3-D reconstruction.