All you need is love (of data)

Bootstrapping-inspired methods for intrinsic and extrinsic camera calibration

Andrea Censi

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology

slides available at censi.mit.edu/slides
“Calibration by correlation”: you can calibrate an arbitrary camera just by waving it around.

Censi, Scaramuzza. Calibration by correlation using metric embedding from non-metric similarities. PAMI 2013
“Calibration by correlation”: you can calibrate an arbitrary camera just by waving it around.

Censi, Scaramuzza. *Calibration by correlation using metric embedding from non-metric similarities*. PAMI 2013
“Calibration by correlation”: you can calibrate an arbitrary camera just by waving it around.

Censi, Scaramuzza. *Calibration by correlation using metric embedding from non-metric similarities.* PAMI 2013
“Calibration by correlation”: you can calibrate an arbitrary camera just by waving it around.

Censi, Scaramuzza. Calibration by correlation using metric embedding from non-metric similarities. PAMI 2013
“Calibration by correlation”: you can calibrate an arbitrary camera just by waving it around.

Censi, Scaramuzza. *Calibration by correlation using metric embedding from non-metric similarities*. PAMI 2013
everything is known

nothing is known
everything is known

adaptive control

self-calibration

nothing is known
everything is known

nothing is known
Let's not assume anything, and see what we can do.

**everything is known**

**most research in robotic learning**

**nothing is known**

Muelling, Peters (MP.Tuebingen)
Let's not assume anything, and see what we can do.

---

everything is known

most research in robotic learning

nothing is known

Muelling, Peters (MP.Tuebingen)
everything is known  

nothing is known
everything is known

nothing is known
How to quantify the value of learning, as an alternative to traditional design?
Everything is known

How to quantify the value of learning, as an alternative to traditional design?

What is the least amount of prior knowledge that is needed?
• How to quantify the value of learning, as an alternative to traditional design?

• What is the least amount of prior knowledge that is needed?
bootstrapping agent
bootstrapping agent
bootstrapping agent

any robot
commands any robot

observations

bootstrapping agent

any robot
bootstrapping agent

observations

any robot

commands
sensorimotor dynamics?

bootstrapping agent

observations

any robot

commands
observations

bootstrapping agent

sensor

any robot
bootstrapping agent

observations

sensor

any robot
intrinsinc calibration?

bootstrapping agent

observations

sensor

any robot
Image space

(u, v)
Image space

\(i, j: \) indices of a pixel
$i, j$: indices of a pixel
Visual sphere

$S^2 = \{ s \in \mathbb{R}^3 \mid \|s\| = 1 \}$

Image space

$i, j$: indices of a pixel
Image space

$i, j$: indices of a pixel

Visual sphere

$S^2 = \{ s \in \mathbb{R}^3 \mid \| s \| = 1 \}$

$s_i$: direction on the visual sphere
$s_i$: direction on the visual sphere
Intrinsic calibration (for central cameras):
find the direction of each pixel on the visual sphere.

\( s_i \): direction on the visual sphere

\( s_j \): direction on the visual sphere
Intrinsic calibration (for central cameras):
find the direction of each pixel on the visual sphere.

\[ s_i, s_j \] direction on the visual sphere
Intrinsic calibration (for central cameras):
find the direction of each pixel on the visual sphere.

$s_i$: direction on the visual sphere

$s_j$: direction on the visual sphere
Intrinsic calibration (for central cameras):
find the direction of each pixel on the visual sphere.
Intrinsic calibration (for central cameras): find the direction of each pixel on the visual sphere.
Intrinsic calibration (for central cameras):
find the direction of each pixel on the visual sphere.
Intrinsic calibration (for central cameras):
find the direction of each pixel on the visual sphere.
Intrinsic calibration (for central cameras):
find the direction of each pixel on the visual sphere.
Intrinsic calibration (for central cameras): find the direction of each pixel on the visual sphere.
- **Intrinsic calibration (for central cameras):** find the direction of each pixel on the visual sphere.
Traditional intrinsic calibration techniques:
Traditional intrinsic calibration techniques:
- rely on known special patterns
Traditional intrinsic calibration techniques:
- rely on known special patterns
Traditional intrinsic calibration techniques:
- rely on known special patterns

omnidirectional camera

fisheye camera
Traditional intrinsic calibration techniques:
- rely on known special patterns

omnidirectional camera

fisheye camera

pinhole camera
Traditional intrinsic calibration techniques:
- rely on known special patterns
- are tailored to specific classes of optics
Traditional intrinsic calibration techniques:
- rely on known special patterns
- are tailored to specific classes of optics

omnidirectional camera
fisheye camera
pinhole camera

everything is known
nothing is known
- Traditional intrinsic calibration techniques:
  - rely on known special patterns
  - are tailored to specific classes of optics

omnidirectional camera

fisheye camera

pinhole camera
“Calibration by correlation”: you can calibrate an arbitrary camera just by waving it around.
“Calibration by correlation”:
 you can calibrate
 an arbitrary camera
 just by waving it around.

- no assumption about camera optics
  (works with all central cameras)
“Calibration by correlation”: you can calibrate an arbitrary camera just by waving it around.

- no assumption about camera optics (works with all central cameras)
- no assumptions about environment (no features needed)
“Calibration by correlation”: you can calibrate an arbitrary camera just by waving it around.

- no assumption about camera optics (works with all central cameras)
- no assumptions about environment (no features needed)
- works with other sensors (e.g., range-finders)
demo
“Calibration by correlation”: you can calibrate an arbitrary camera just by waving it around.
“Calibration by correlation”: you can calibrate an arbitrary camera just by waving it around.

(input data)

(magic happens)

(calibration results)

Pixel directions \( \{s_i\} \subseteq S^2 \)

-30° 0° 30°

-15° 0° 15°

azimuth
elevation
“Calibration by correlation”: you can calibrate an arbitrary camera just by waving it around.
“Calibration by correlation”: you can calibrate an arbitrary camera just by waving it around.
input data

correlation statistics

calibration results

omnidirectional

fisheye (GoPro)

pinhole camera

metric embedding from nonmetric similarities

metric embedding from nonmetric similarities

metric embedding from nonmetric similarities

Pixel directions \( \{s_i\} \subset S^2 \)

Pixel directions \( \{s_i\} \subset S^2 \)

Pixel directions \( \{s_i\} \subset S^2 \)
**input data**

**omnidirectional**

**fisheye (GoPro)**

**pinhole camera**

**correlation statistics**

**metric embedding from nonmetric similarities**

**corr**

**corr**

**corr**

**calibration results**

**Pixel directions \( \{ s_i \} \subset S^2 \)**

**Pixel directions \( \{ s_i \} \subset S^2 \)**

**Pixel directions \( \{ s_i \} \subset S^2 \)**

**Pixel directions \( \{ s_i \} \subset S^2 \)**

**Pixel directions \( \{ s_i \} \subset S^2 \)**

**Pixel directions \( \{ s_i \} \subset S^2 \)**
The signals from two pixels are “similar” if they are physically close.
The signals from two pixels are “similar” if they are physically close.
The signals from two pixels are “similar” if they are physically close.
The signals from two pixels are “similar” if they are physically close.
The signals from two pixels are “similar” if they are physically close.
The signals from two pixels are "similar" if they are physically close.
The signals from two pixels are "similar" if they are physically close.

\[ R_{12} = \text{corr}(y_1, y_2) \]
The signals from two pixels are “similar” if they are physically close.

\[ R_{12} = \text{corr}(y_1, y_2) \]
\[ R_{23} = \text{corr}(y_2, y_3) \]
The signals from two pixels are “similar” if they are physically close.

⇒ You can easily reconstruct the **sensor topology**.
The signals from two pixels are “similar” if they are physically close.

⇒ You can easily reconstruct the sensor topology.

But topology is not enough for calibration!
The signals from two pixels are “similar” if they are physically close.

⇒ You can easily reconstruct the sensor topology.

But topology is not enough for calibration!
The signals from two pixels are “similar” if they are physically close.

⇒ You can easily reconstruct the sensor topology.

But **topology is not enough** for calibration!

*we want correct shape*
The signals from two pixels are “similar” if they are physically close.

⇒ You can easily reconstruct the **sensor topology**.

But **topology is not enough** for calibration!

- we want **correct shape**
- we want **correct scale**
The trick:
The trick: random motion makes the problem easier.
The trick: random motion makes the problem easier.

Bruce Springsteen
*Streets of Philadelphia*

Davide Scaramuzza
*Calibration in the Streets of Philadelphia*
Assuming **uniformly random motion**, the **correlation** between pixel values is an **unknown function** $f$ of the pixel **distances on the visual sphere**.

\[ R_{ij} = f(d(s_i, s_j)) \]
Assuming **uniformly random motion**, the **correlation** between pixel values is an **unknown function** $f$ of the pixel **distances on the visual sphere**.

$$R_{ij} = f(d(s_i, s_j))$$
Assuming uniformly random motion, the correlation between pixel values is an unknown function $f$ of the pixel distances on the visual sphere.

$$R_{ij} = f(d(s_i, s_j))$$

attitude R is uniform:

$$\{R_t \mid t \in \text{dataset} \} = \text{Uniform}(\text{SO}(3))$$
Assuming *uniformly random motion*, the *correlation* between pixel values is an *unknown function* $f$ of the pixel *distances on the visual sphere*.

$$R_{ij} = f(d(s_i, s_j))$$

attitude $R$ is uniform:

$$\{R_t \mid t \in \text{dataset}\} = \text{Uniform}(\text{SO}(3))$$
Assuming uniformly random motion, the correlation between pixel values is an unknown function $f$ of the pixel distances on the visual sphere.

$$R_{ij} = f(d(s_i, s_j))$$
Assuming uniformly random motion, the correlation between pixel values is an unknown function $f$ of the pixel distances on the visual sphere.

$$R_{ij} = f(d(s_i, s_j))$$
Assuming uniformly random motion, the correlation between pixel values is an unknown function $f$ of the pixel distances on the visual sphere.

$$R_{ij} = f(d(s_i, s_j))$$
Assuming uniformly random motion, the correlation between pixel values is an unknown function $f$ of the pixel distances on the visual sphere.

$$ R_{ij} = f(d(s_i, s_j)) $$
Assuming uniformly random motion, the correlation between pixel values is an unknown function $f$ of the pixel distances on the visual sphere.

$$R_{ij} = f(d(s_i, s_j))$$
Assuming uniformly random motion, the correlation between pixel values is an unknown function $f$ of the pixel distances on the visual sphere.

$$R_{ij} = f(d(s_i, s_j))$$

$$R_{12} = \text{corr}(y_1(t), y_2(t))$$
Assuming uniformly random motion, the correlation between pixel values is an unknown function $f$ of the pixel distances on the visual sphere.

$$R_{ij} = f(d(s_i, s_j))$$
Assuming uniformly random motion, the correlation between pixel values is an unknown function $f$ of the pixel distances on the visual sphere.

$$R_{ij} = f(d(s_i, s_j))$$

$$R_{12} = \text{corr}(y_1(t), y_2(t))$$
Assuming **uniformly random motion**, the correlation between pixel values is an **unknown function** $f$ of the pixel **distances on the visual sphere**.

$$R_{ij} = f(d(s_i, s_j))$$

$R_{12} = \text{corr}(y_1(t), y_2(t))$

$$R_{12} = f(d(s_1, s_2))$$
Assuming uniformly random motion, the correlation between pixel values is an unknown function $f$ of the pixel distances on the visual sphere.

$$R_{ij} = f(d(s_i, s_j))$$

$$R_{12} = f(d(s_1, s_2))$$
Assuming uniformly random motion, the correlation between pixel values is an unknown function $f$ of the pixel distances on the visual sphere.

$$R_{ij} = f(d(s_i, s_j))$$

$$R_{12} = \text{corr}(y_1(t), y_2(t))$$

$$R_{12} = f(d(s_1, s_2))$$
If the observable similarities $R_{ij}$ are an unknown \textbf{monotonic} function $f$ of the pixel distances $d(s_i, s_j)$:

$$R_{ij} = f(d(s_i, s_j))$$

Then*, from $R_{ij}$ we can recover the directions \{s_i\} (up to isometries).
If the observable similarities $R_{ij}$ are an unknown **monotonic** function $f$ of the pixel distances $d(s_i, s_j)$:

$$R_{ij} = f(d(s_i, s_j))$$

Then*, from $R_{ij}$ we can recover the directions $\{s_i\}$ (up to isometries).

*other technical conditions apply (e.g., connectedness of domain)
If the observable similarities $R_{ij}$ are an unknown \textbf{monotonic} function $f$ of the pixel distances $d(s_i, s_j)$:

$$R_{ij} = f(d(s_i, s_j))$$

Then*, from $R_{ij}$ we can recover the directions $\{s_i\}$ (up to isometries).

* other technical conditions apply (e.g., connectedness of domain)

(!) The \textbf{scale} (field of view) is \textbf{observable}
If the observable similarities $R_{ij}$ are an unknown **monotonic** function $f$ of the pixel distances $d(s_i, s_j)$:

$$R_{ij} = f(d(s_i, s_j))$$

Then*, from $R_{ij}$ we can recover the directions $\{s_i\}$ (up to isometries).

*other technical conditions apply (e.g., connectedness of domain)

(\!) The **scale** (field of view) is **observable**

sphere $\mathbb{S}^2$
If the observable similarities $R_{ij}$ are an unknown monotonic function $f$ of the pixel distances $d(s_i, s_j)$:

$$R_{ij} = f(d(s_i, s_j))$$

Then*, from $R_{ij}$ we can recover the directions $\{s_i\}$ (up to isometries).

* Other technical conditions apply (e.g., connectedness of domain)

(!) The scale (field of view) is observable

scale not observable

$\mathbb{S}^2$ sphere

$\mathbb{R}^n$ circle $\mathbb{S}^1$
If the observable similarities $R_{ij}$ are an unknown monotonic function $f$ of the pixel distances $d(s_i, s_j)$:

$$R_{ij} = f(d(s_i, s_j))$$

Then*, from $R_{ij}$ we can recover the directions $\{s_i\}$ (up to isometries).

* other technical conditions apply (e.g., connectedness of domain)

(!) The scale (field of view) is observable

scale not observable

sphere $S^2$

$\mathbb{R}^n$

circle $S^1$
If the observable similarities $R_{ij}$ are an unknown **monotonic** function $f$ of the pixel distances $d(s_i, s_j)$:

$$R_{ij} = f(d(s_i, s_j))$$

Then*, from $R_{ij}$ we can recover the directions $\{s_i\}$ (up to isometries).

* other technical conditions apply (e.g., connectedness of domain)

(!) The **scale** (field of view) is **observable**

$$\rho > 0$$

sphere $\mathbb{S}^2$

scale not observable

curvature $\rho = 0$

$\mathbb{R}^n$

circle $\mathbb{S}^1$
If the observable similarities $R_{ij}$ are an unknown **monotonic** function $f$ of the pixel distances $d(s_i, s_j)$:

$$R_{ij} = f(d(s_i, s_j))$$

Then*, from $R_{ij}$ we can recover the directions $\{s_i\}$ (up to isometries).

*other technical conditions apply (e.g., connectedness of domain)

(!) The **scale** (field of view) is observable

- **scale observable**
  - $\rho > 0$
  - sphere $S^2$

- **scale not observable**
  - curvature $\rho = 0$
  - $\mathbb{R}^n$
  - circle $S^1$
If the observable similarities $R_{ij}$ are an unknown **monotonic** function $f$ of the pixel distances $d(s_i, s_j)$:

$$R_{ij} = f(d(s_i, s_j))$$

Then*, from $R_{ij}$ we can recover the directions $\{s_i\}$ (up to isometries).

* other technical conditions apply (e.g., connectedness of domain)

(!) **The scale** (field of view) **is observable**

<table>
<thead>
<tr>
<th>scale observable</th>
<th>scale not observable</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho &gt; 0$</td>
<td>curvature $\rho = 0$</td>
</tr>
<tr>
<td>sphere $S^2$</td>
<td>$\mathbb{H}^n$</td>
</tr>
<tr>
<td>$\rho &lt; 0$</td>
<td>$\mathbb{R}^n$</td>
</tr>
<tr>
<td>hyperbolic space</td>
<td>circle $S^1$</td>
</tr>
</tbody>
</table>
**metric embedding from nonmetric similarities**

*Find node positions given an unknown function of inter-point distances.*

\[ R_{ij} = f(d(s_i, s_j)) \]
**metric embedding from nonmetric similarities**

Find node positions given an unknown function of inter-point distances.

\[ R_{ij} = f(d(s_i, s_j)) \]

**metric embedding**

Find node positions given inter-point distances.

\[ \tilde{d}_{ij} = d(s_i, s_j) \]
**metric embedding from nonmetric similarities**

Find node positions given an unknown function of inter-point distances.

$$R_{ij} = f(d(s_i, s_j))$$

**metric embedding**

Find node positions given inter-point distances.

$$\tilde{d}_{ij} = d(s_i, s_j)$$

Classic algorithm: MDS ("multidimensional scaling")

$$\{d_{ij}\} \xrightarrow{\text{MDS}} \{s_i\} \subset \mathbb{S}^2$$
**metric embedding from nonmetric similarities**

*Find node positions given an unknown function of inter-point distances.*

\[ R_{ij} = f(d(s_i, s_j)) \]

---

**metric embedding**

*Find node positions given inter-point distances.*

\[ \tilde{d}_{ij} = d(s_i, s_j) \]

---

Classic algorithm: MDS (*“multidimensional scaling”*)

\[ \{d_{ij}\} \xrightarrow{\text{MDS}} \{s_i\} \subset S^2 \]
metric embedding from nonmetric similarities

Find node positions given an unknown function of inter-point distances.

\[ R_{ij} = f(d(s_i, s_j)) \]
metric embedding from nonmetric similarities

Find node positions given an unknown function of inter-point distances.

\[ R_{ij} = f(d(s_i, s_j)) \]

- Chicken-and egg problem!
**metric embedding from nonmetric similarities**

*Find node positions given an unknown function of inter-point distances.*

\[ R_{ij} = f(d(s_i, s_j)) \]

- **Chicken-and egg problem!**
  - Two variables:


**metric embedding from nonmetric similarities**

*Find node positions given an unknown function of inter-point distances.*

\[ R_{ij} = f(d(s_i, s_j)) \]

- **Chicken-and egg problem!**
  - Two variables:
    - unknown pixel directions \( \{s_i\} \)
**metric embedding from nonmetric similarities**

*Find node positions given an unknown function of inter-point distances.*

\[ R_{ij} = f(d(s_i, s_j)) \]

- **Chicken-and egg problem!**
  - Two variables:
    - unknown pixel directions \( \{s_i\} \)
    - unknown function \( f \)
**metric embedding from nonmetric similarities**

Find node positions given an unknown function of inter-point distances.

\[ R_{ij} = f(d(s_i, s_j)) \]

- **Chicken-and egg problem!**
  - Two variables:
    - unknown pixel directions \( \{s_i\} \)
    - unknown function \( f \)
  - If you know \( f \), you can find \( \{s_i\} \) using MDS.
**metric embedding from nonmetric similarities**

Find node positions given an unknown function of inter-point distances.

\[ R_{ij} = f(d(s_i, s_j)) \]

- **Chicken-and egg problem!**
  - Two variables:
    - unknown pixel directions \( \{s_i\} \)
    - unknown function \( f \)
  - If you know \( f \), you can find \( \{s_i\} \) using MDS.
  - If you know \( \{s_i\} \), you can find \( f \) by fitting.
**metric embedding from nonmetric similarities**

*Find node positions given an unknown function of inter-point distances.*

\[
R_{ij} = f(d(s_i, s_j))
\]

- **Chicken-and egg problem!**
  - Two variables:
    - unknown pixel directions \(\{s_i\}\)
    - unknown function \(f\)
  - If you know \(f\), you can find \(\{s_i\}\) using MDS.
  - If you know \(\{s_i\}\), you can find \(f\) by fitting.
For a chicken-and-egg problem, try something like EM and cross your fingers.

Chicken-and egg problem!

- Two variables:
  - unknown pixel directions \( \{s_i\} \)
  - unknown function \( f \)

- If you know \( f \), you can find \( \{s_i\} \) using MDS.
- If you know \( \{s_i\} \), you can find \( f \) by fitting.

Metric embedding from nonmetric similarities
Find node positions given an unknown function of inter-point distances.

\[
R_{ij} = f(d(s_i, s_j))
\]
**metric embedding from nonmetric similarities**

*Find node positions given an unknown function of inter-point distances.*

\[ R_{ij} = f(d(s_i, s_j)) \]
metric embedding from nonmetric similarities

Find node positions given an unknown function of inter-point distances.

\[ R_{ij} = f(d(s_i, s_j)) \]
metric embedding from nonmetric similarities
Find node positions given an unknown function of inter-point distances.

Algorithm sketch

- Start with a guess: \( f_0(d) = 1 - d \)
metric embedding from nonmetric similarities

Find node positions given an unknown function of inter-point distances.

\[ R_{ij} = f(d(s_i, s_j)) \]

**Algorithm sketch**

- Start with a guess: \( f_0(d) = 1 - d \)

1) Guess distances as \( f_k^{-1}(R_{ij}) \) then use MDS to estimate directions.
metric embedding from nonmetric similarities

Find node positions given an unknown function of inter-point distances.

\[ R_{ij} = f(d(s_i, s_j)) \]

Algorithm sketch

- Start with a guess: \( f_0(d) = 1 - d \)

  1) Guess distances as \( f_k^{-1}(R_{ij}) \) then use MDS to estimate directions.

  2) Use current directions to re-estimate \( f_{k+1} \).
metric embedding from nonmetric similarities
Find node positions given an unknown function of inter-point distances.

\[ R_{ij} = f(d(s_i, s_j)) \]

Algorithm sketch

- Start with a guess: \( f_0(d) = 1 - d \)

- Repeat until convergence:
  1) Guess distances as \( f_k^{-1}(R_{ij}) \) then use MDS to estimate directions.
  2) Use current directions to re-estimate \( f_{k+1} \).
metric embedding from nonmetric similarities

Find node positions given an unknown function of inter-point distances.

\[ R_{ij} = f(d(s_i, s_j)) \]

**Algorithm sketch**

- Start with a guess: \( f_0(d) = 1 - d \)
  - Use multiple initializations.
- Repeat until convergence:
  1) Guess distances as \( f_k^{-1}(R_{ij}) \) then use MDS to estimate directions.
  2) Use current directions to re-estimate \( f_{k+1} \).
metric embedding from nonmetric similarities
Find node positions given an unknown function of inter-point distances.

\[ R_{ij} = f(d(s_i, s_j)) \]

Algorithm sketch

- Start with a guess: \( f_0(d) = 1 - d \)
- Use multiple initializations.
- Repeat until convergence:
  1) Guess distances as \( f_k^{-1}(R_{ij}) \) then use MDS to estimate directions.
  2) Use current directions to re-estimate \( f_{k+1} \).
- Use non-parametric fitting of \( f \).
**Algorithm sketch**

- Start with a guess: \( f_0(d) = 1 - d \)
  
  **Use multiple initializations.**

- Repeat until convergence:
  1) Guess distances as \( f_k^{-1}(R_{ij}) \) then use MDS to estimate directions.
  2) Use current directions to re-estimate \( f_{k+1} \).

  **Use non-parametric fitting of \( f \).**

- Separate phase for recovering scale.
Algorithm 1 The SK\textsuperscript{v+w} embedding algorithm for a generic manifold $\mathcal{M}$.

**Input:** similarities $Y \in \mathbb{R}^{n \times n}$; manifold-specific functions: $\text{MDS}_\mathcal{M}$, $\text{distances}_\mathcal{M}$, init\(_\mathcal{M}\). **Output:** $S \in \mathcal{M}^n$.

```
for $D^0$ in init\(_\mathcal{M}\)(order($Y$)): # Some manifolds need multiple starting points
    $S^0 = \text{MDS}_\mathcal{M}(D^0)$ # Compute first guess by MDS
for $k = 1, 2, \ldots$ until $s^k$ converged:
    $D^k = \text{distances}_\mathcal{M}(S^{k-1})$ # Compute current distances
    $D_* = \text{vec}^{-1}(\text{sorted}(\text{vec}(D))[\text{order}(\text{vec}(Y))])$ # Nonparametric fitting and inversion of $f$.
    $S^k = \text{MDS}_\mathcal{M}(D_*)$ # Embed according to the modified distances.
    $s^k = \text{spearman\_score}(S^k, Y)$
$S^* = S^{k^*}$, where $k^* = \arg\max_k s^k$ # Find best iteration according to the score.
if $\mathcal{M}$ is $S^m$, $m \geq 2$: # Find optimal warping factor to embed in the sphere.
    $D^* = \text{distances}_\mathcal{M}(S^*)$
    $\alpha^* = \arg\min_\alpha \sigma_{m+1}^\alpha / \sigma_{m+2}^\alpha$ for $\{\sigma_i^\alpha\} = \text{singular\_values}(\cos(\alpha D^*))$
return $\text{MDS}_\mathcal{M}(\alpha^* D^*)$
return $S^*$
```

$\mathcal{M}$-specific initializations: \init_{\mathbb{R}^m}(Y) \triangleq Y; \init_{S^m}(Y) \triangleq \{\pi Y / n^2, 2\pi Y / n^2\}.$
input data

omnidirectional

fisheye (GoPro)

pinhole camera

metric embedding from nonmetric similarities

Pixel directions \( \{s_i\} \subset S^2 \)

Pixel directions \( \{s_i\} \subset S^2 \)

Pixel directions \( \{s_i\} \subset S^2 \)

calibration results
input data

omnidirectional

fisheye (GoPro)

pinhole camera

metric embedding from nonmetric similarities

Pixel directions $\{s_i\} \subset S^2$

Pixel directions $\{s_i\} \subset S^2$

Pixel directions $\{s_i\} \subset S^2$

calibration results
- Using 20-25 minutes of data.
- Mismatch with traditional techniques: $1-3^\circ$. 

\[ \text{Pixel directions } \{s_i\} \subset S^2 \]
“Calibration by correlation”: you can calibrate a camera just by waving it around.
“Calibration by correlation”: you can calibrate a camera just by waving it around.

- no assumption about camera optics (works with all central cameras)
“Calibration by correlation”:
you can calibrate a camera just by waving it around.

- no assumption about camera optics
  (works with all central cameras)

- no assumptions about environment
  (no features needed)
“Calibration by correlation”: you can calibrate a camera just by waving it around.

- no assumption about camera optics (works with all central cameras)
- no assumptions about environment (no features needed)
- works with other sensors (e.g., range-finders)
All you need is love (of data)

Bootstrapping-inspired methods for intrinsic and extrinsic camera calibration
"SALLIE GARDNER," owned by LELAND STANFORD; running at a 1.40 gait over the Palo Alto track, 19th June, 1878.

The negatives of these photographs were made at intervals of twenty-seven inches of distance, and about the twenty-fifth part of a second of time; they illustrate consecutive positions assumed in each twenty-seven inches of progress during a single stride of the mare. The vertical lines were twenty-seven inches apart; the horizontal lines represent elevations of four inches each. The exposure of each negative was less than the two-thousandth part of a second.
DVS (Dynamic Vision Sensor): events generated any time a single pixel sees a change in brightness.
DVS (Dynamic Vision Sensor): 
- events generated 
any time a single pixel 
sees a change in brightness.
DVS (Dynamic Vision Sensor): events generated any time a single pixel sees a change in brightness.
DVS (Dynamic Vision Sensor): 
**events** generated 
any time a **single pixel** 
sees a **change in brightness**.

\[ |y_t - y_0| > \epsilon \]
DVS (Dynamic Vision Sensor): events generated any time a single pixel sees a change in brightness.
DVS (Dynamic Vision Sensor): **events** generated any time a **single pixel** sees a **change in brightness**.

\[ |y_t - y_0| > \epsilon \]
DVS (Dynamic Vision Sensor): events generated any time a single pixel sees a change in brightness.

\[ |y_t - y_0| > \epsilon \]
DVS (Dynamic Vision Sensor): events generated any time a single pixel sees a change in brightness.

\[ |y_t - y_0| > \epsilon \]
DVS (Dynamic Vision Sensor): events generated any time a single pixel sees a change in brightness.

\[ |y_t - y_0| > \epsilon \]
DVS    CMOS camera
DVS    CMOS camera
DVS  CMOS camera
Need **spatio-temporal calibration:**
- Temporal calibration: two different clocks.
- Spatial calibration = extrinsic calibration.
Need **spatio-temporal calibration**:
- Temporal calibration: two different clocks.
- Spatial calibration = extrinsic calibration.

*All you need is love (of data)*
Temporal calibration can be done by looking at the event rate.
Temporal calibration can be done by looking at the event rate.

\[
|\omega_t| \quad (\text{IMU})
\]

30°/s
Temporal calibration can be done by looking at the event rate.

\[ \frac{d}{dt} \]

\[ |\omega_t| \]

(IMU)
Temporal calibration can be done by looking at the event rate.

- Event rate (log scale)
- $\frac{d}{dt}$
- $|\omega_t|$ (IMU)
- 100k events/s
- 30°/s
- **Extrinsic calibration** can be done easily assuming **small baseline**.
Extrinsic calibration can be done easily assuming small baseline.

\[ \approx \text{two central cameras with the same center} \]
Extrinsic calibration can be done easily assuming small baseline.

≈ two central cameras with the same center
Extrinsic calibration can be done easily assuming small baseline.

≃ two central cameras with the same center
Extrinsic calibration can be done easily assuming small baseline.

Æ two central cameras with the same center
How to map one image space to the other?
How to map one image space to the other?
How to map one image space to the other?
How to map one image space to the other?
Idea: find a reasonable **similarity function**, match the most similar pixel.
Idea: find a reasonable similarity function, match the most similar pixel.

\( \text{similarity}(i, j) \)
Idea: find a reasonable **similarity function**, match the most similar pixel.

\[ \text{similarity}(i, j) \]
Idea: find a reasonable **similarity function**, match the most similar pixel.

\[
similarity(i, j) = \frac{1}{\text{similarity}(i, j)}
\]
Idea: find a reasonable **similarity function**, match the most similar pixel.

\[
\text{similarity}(i, j) = \text{corr}(\quad , \quad )
\]
Idea: find a reasonable **similarity function**, match the most similar pixel.

\[
similarity(i, j) = corr( \text{brightness changes at } j )
\]
Idea: find a reasonable similarity function, match the most similar pixel.

\[
similarity(i, j) = \text{corr}(\text{brightness changes at } j, \frac{d}{dt}\mid_{\text{image}})
\]
Idea: find a reasonable similarity function, match the most similar pixel.

\[
similarity(i, j) = \text{corr}(\text{brightness changes at } j)
\]
Idea: find a reasonable **similarity function**, match the most similar pixel.

\[
similarity(i, j) = \text{corr}(\text{brightness changes at } j, \text{ event rate at } i)
\]
Idea: find a reasonable similarity function, match the most similar pixel.

\[
similarity(i, j) = \text{corr}(\text{brightness changes at } j, \text{ event rate at } i)
\]
Idea: find a reasonable similarity function, match the most similar pixel.

\[ \text{similarity}(i, j) = \text{corr}(\text{brightness changes at } j, \text{ event rate at } i) \]
Idea: find a reasonable \textit{similarity function}, match the most similar pixel.

\[
similarity(i, j) = \text{corr}(\text{brightness changes at } j, \text{ event rate at } i)
\]
Idea: find a reasonable similarity function, match the most similar pixel.

\[ \text{similarity}(i, j) = \text{corr}(\text{brightness changes at } j, \text{ event rate at } i) \]
Idea: find a reasonable similarity function, match the most similar pixel.

\[ \text{similarity}(i, j) = \text{corr( brightness changes at } j , \text{ event rate at } i ) \]
Idea: find a reasonable *similarity function*, match the most similar pixel.

\[
\text{similarity}(i, j) = \text{corr( brightness changes at } j , \text{ event rate at } i )
\]
Idea: find a reasonable similarity function, match the most similar pixel.

\[
similarity(i, j) = \text{corr( brightness changes at } j, \text{ event rate at } i )
\]
Idea: find a reasonable **similarity function**, match the most similar pixel.

\[
similarity(i, j) = \text{corr( brightness changes at } j \text{, event rate at } i )
\]
Idea: find a reasonable *similarity function*, match the most similar pixel.

\[
similarity(i, j) = \text{corr( brightness changes at } j \text{, event rate at } i )
\]
Idea: find a reasonable similarity function, match the most similar pixel.

\[ \text{similarity}(i, j) = \text{corr}(\text{brightness changes at } j, \text{ event rate at } i) \]
Idea: find a reasonable **similarity function**, match the most similar pixel.

\[
similarity(i, j) = \text{corr}(\text{brightness changes at } j, \text{ event rate at } i)
\]
Idea: find a reasonable similarity function, match the most similar pixel.

\[ \text{similarity}(i, j) = \text{corr}(\text{brightness changes at } j, \text{ event rate at } i) \]
Idea: find a reasonable similarity function, match the most similar pixel.

\[
similarity(i, j) = \text{corr( brightness changes at } j \text{ , event rate at } i )
\]
Idea: find a reasonable similarity function, match the most similar pixel.

\[
similarity(i, j) = \text{corr}(\text{brightness changes at } j, \text{ event rate at } i)
\]
Idea: find a reasonable similarity function, match the most similar pixel.

\[
similarity(i, j) = \text{corr( brightness changes at } j \text{, event rate at } i )
\]
Idea: find a reasonable similarity function, match the most similar pixel.
Idea: find a reasonable **similarity function**, match the most similar pixel.
- You can make results more precise by using **model priors**.
You can make results more precise by using **model priors**.
You can make results more precise by using **model priors**.
You can make results more precise by using **model priors**.
CMOS

DVS
CMOS

DVS
CMOS

transfer luminance signal

DVS
All you need is love (of data)
All you need is love (of data)
All you need is love (of data)

everything is known

nothing is known
All you need is love (of data)

everything is known

higher performance

nothing is known

lower performance

All you need is love (of data)
All you need is love (of data)
All you need is love (of data)