Fast and Robust Registration of Multiple Depth-Sensors and Virtual Worlds

Andre Mühlenbrock, Roland Fischer, René Weller, Gabriel Zachmann

University of Bremen, Germany

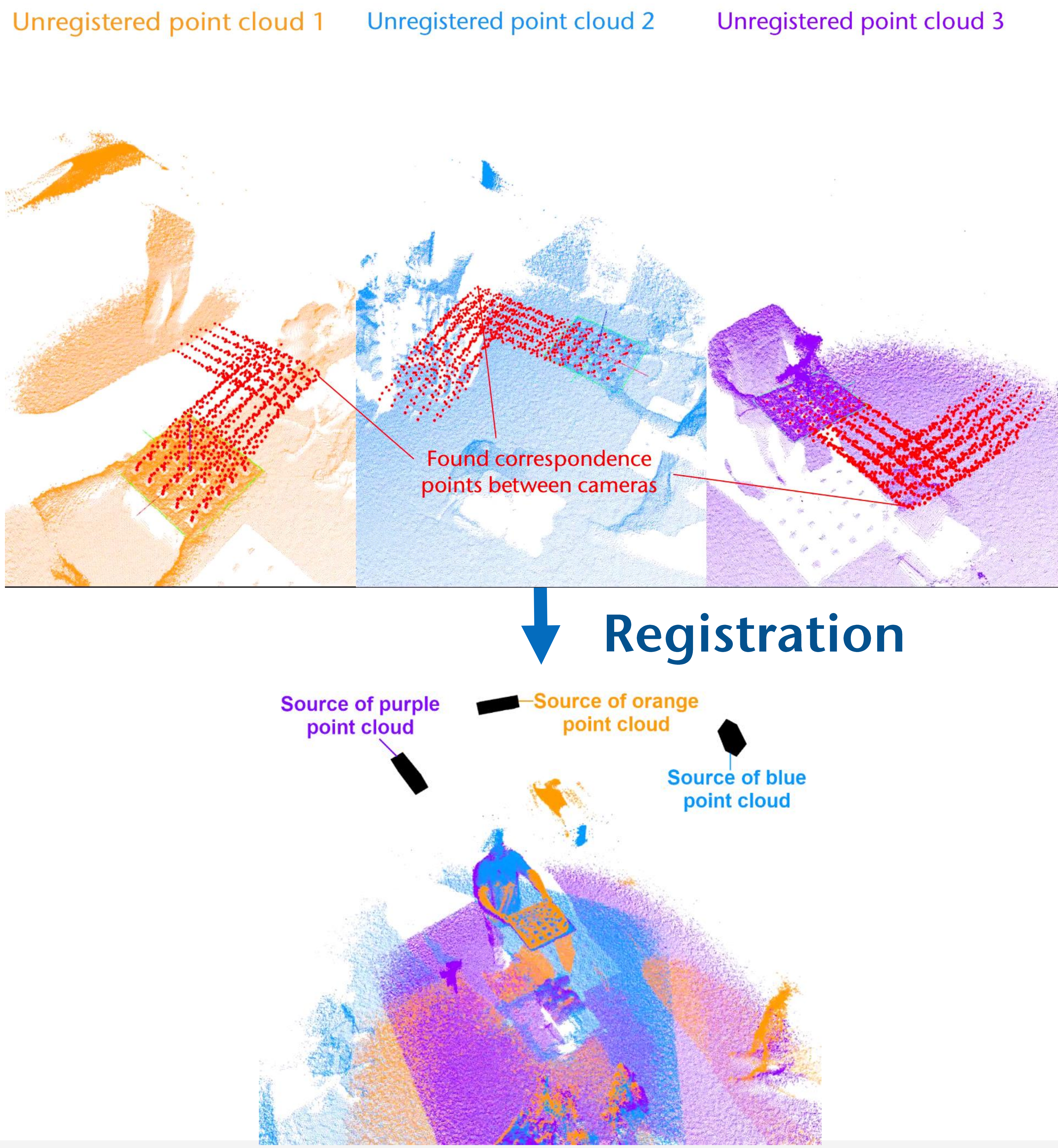
muehlenb@uni-bremen.de

Cyberworlds 2021

28-30 November, Caen, France

Motivation

- Capturing a point cloud of a scene might require multiple depth sensors
  - Due to occlusions, capture volume, etc.
- Results in unregistered point clouds
- To merge the point clouds, a registration (= extrinsic calibration) is required

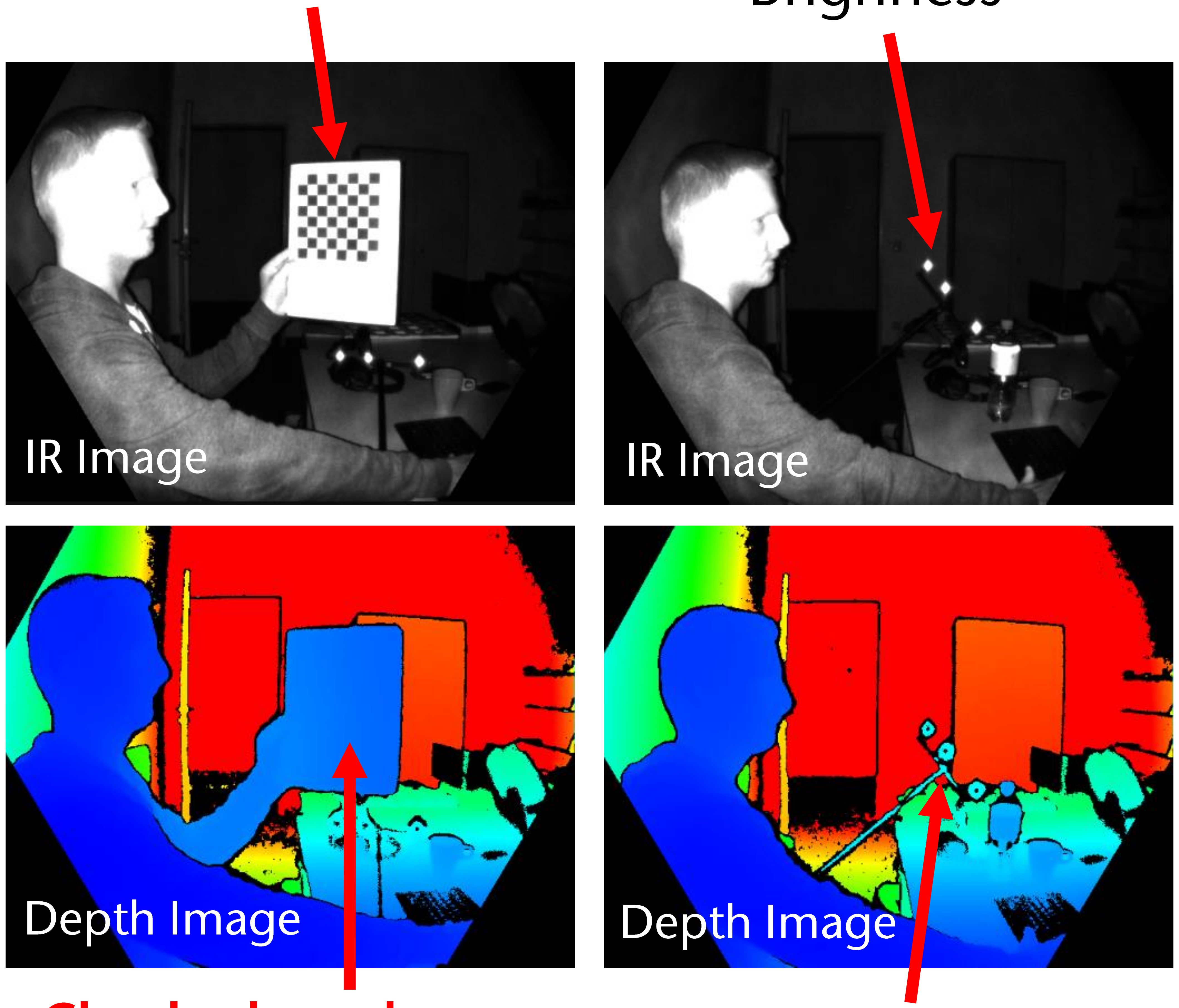

Unregistered point cloud 1
Unregistered point cloud 2
Unregistered point cloud 3
Found correspondence points between cameras
Registration
Source of purple point cloud
Source of orange point cloud
Source of blue point cloud

Previous Work

- Usually color- or IR-images are used to detect a registration target.
- Not applicable, if:
  - Color or IR image not available.
  - Different projections are used.
    - E.g. Orbbec sensors [Reyes-Aviles 20]
  - ➢ Therefore, our approach **only** relies on the depth image.

Corner detection

Segmentation by Brighness



Checkerboard gone

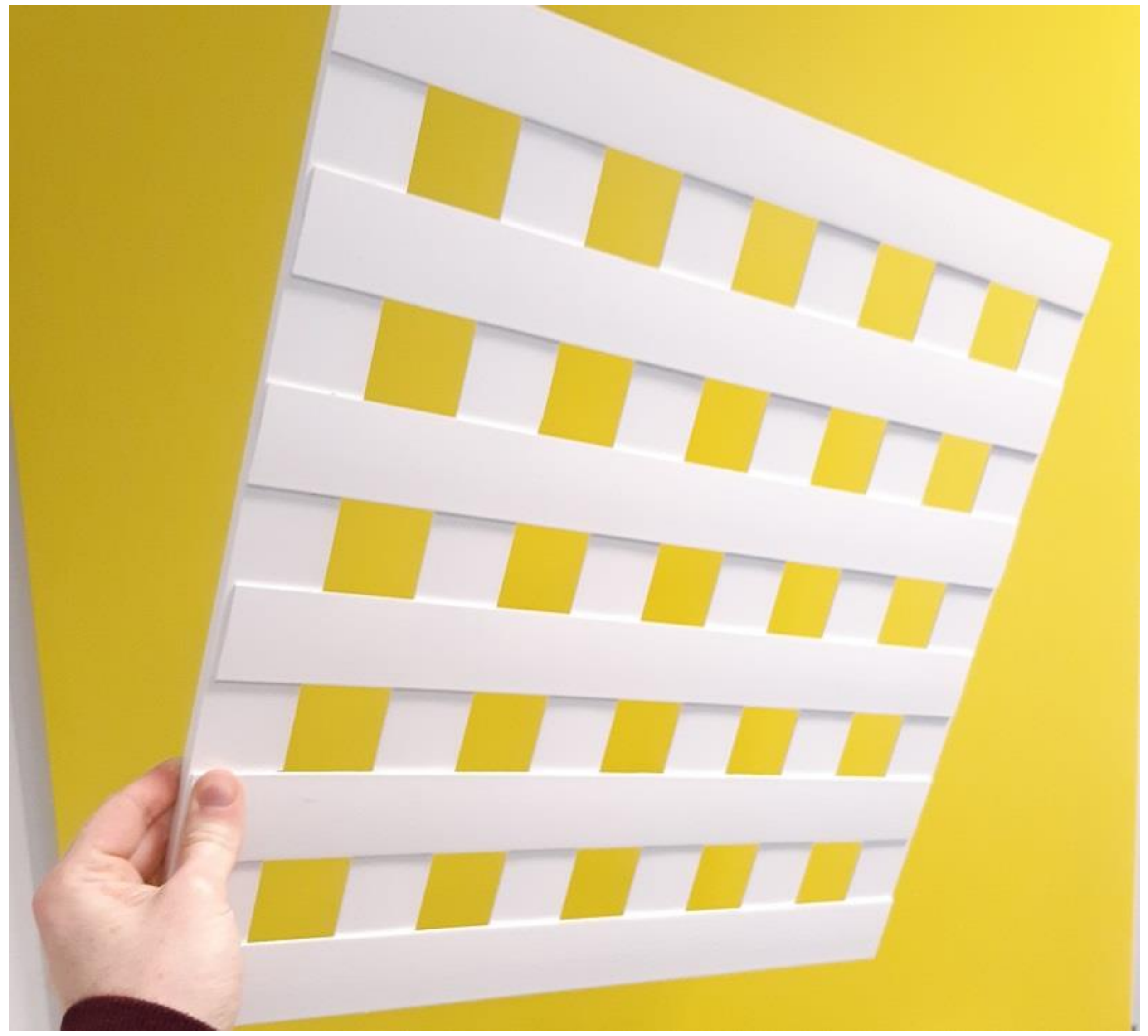
No salient color available

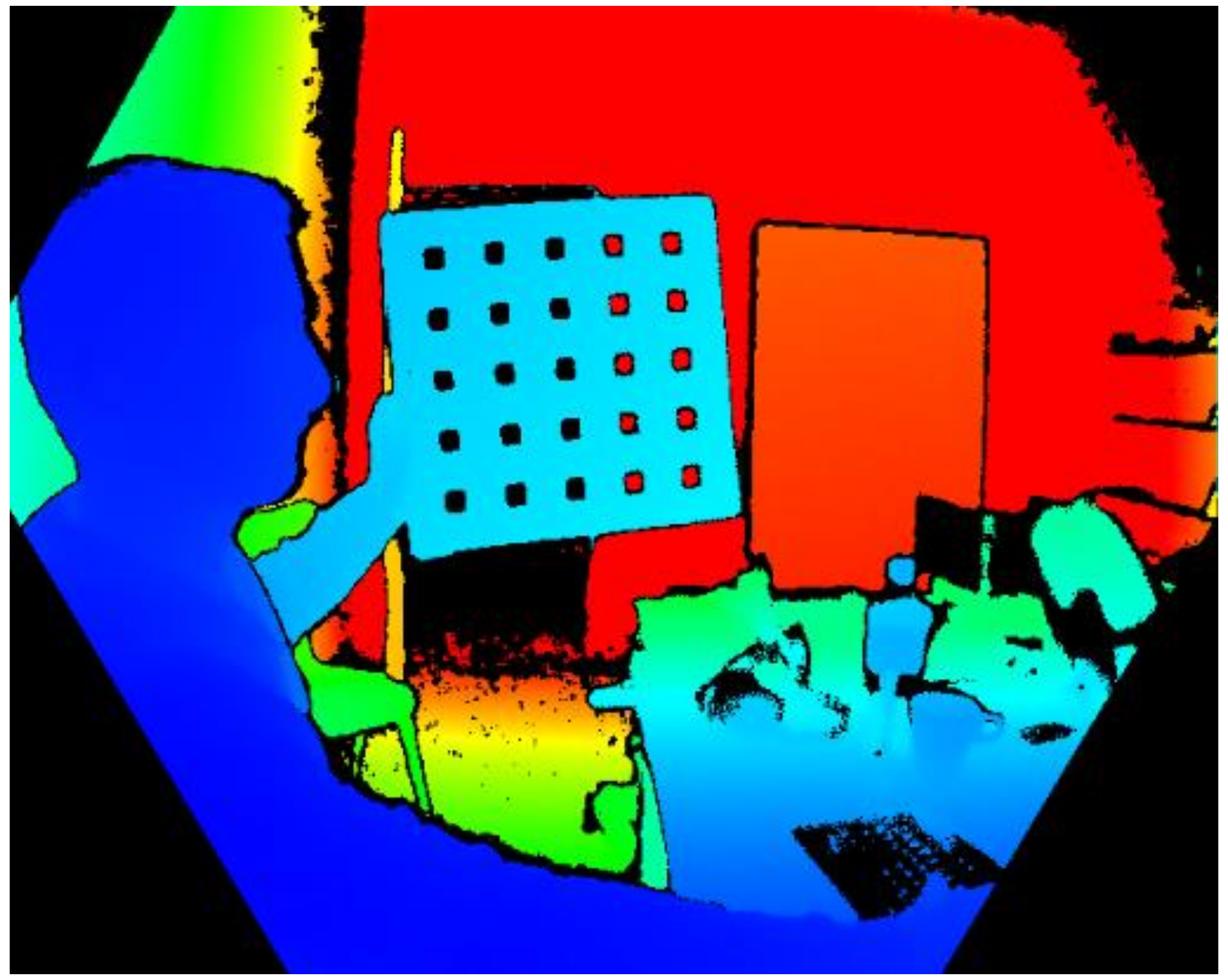
Our Approach

- We designed:

1. **A lattice** as registration target which is visible in the depth image.
2. **An algorithm** which robustly detects this lattice and it‘s unique feature points in the depth image only.

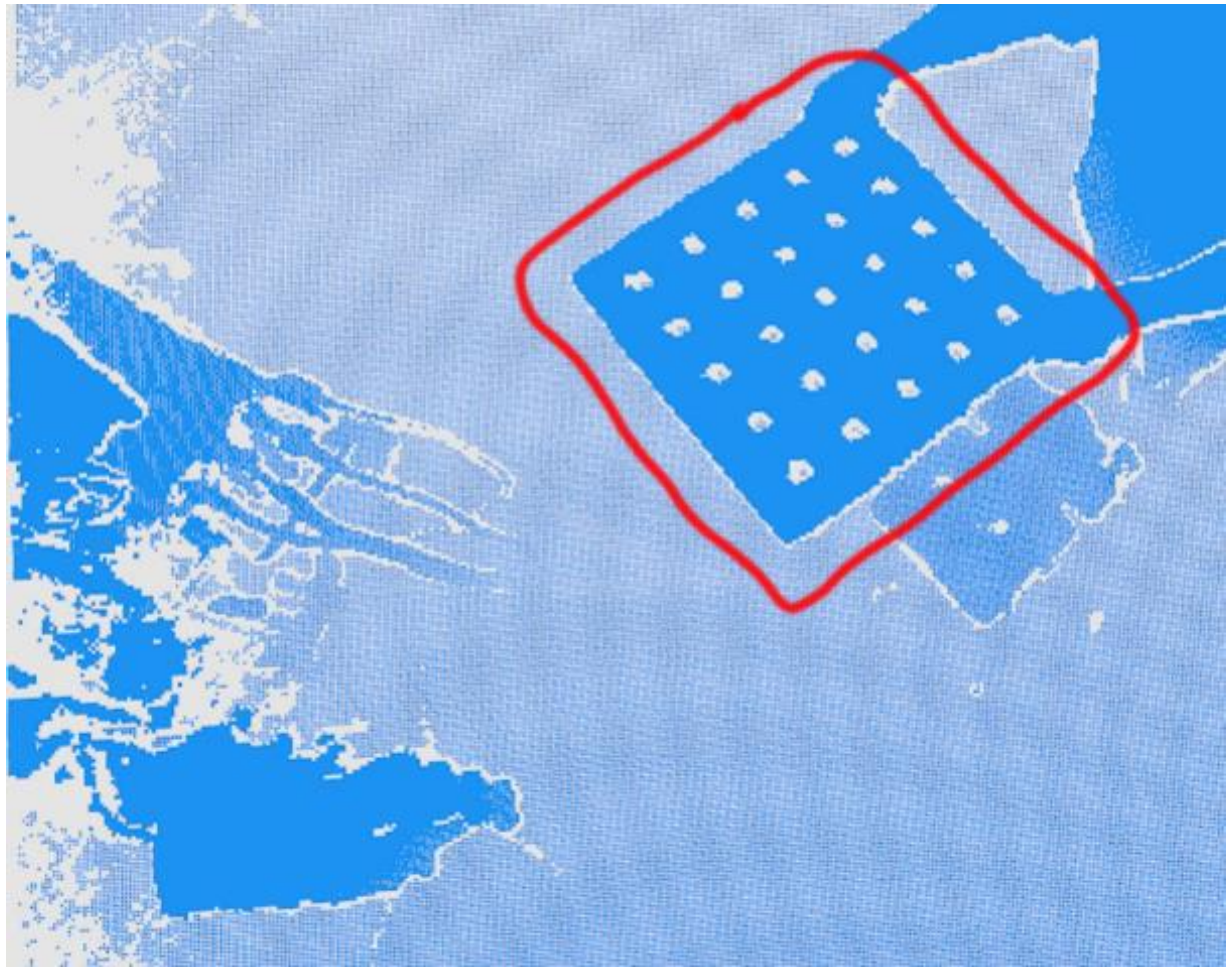
➢ We use hole centers as feature points





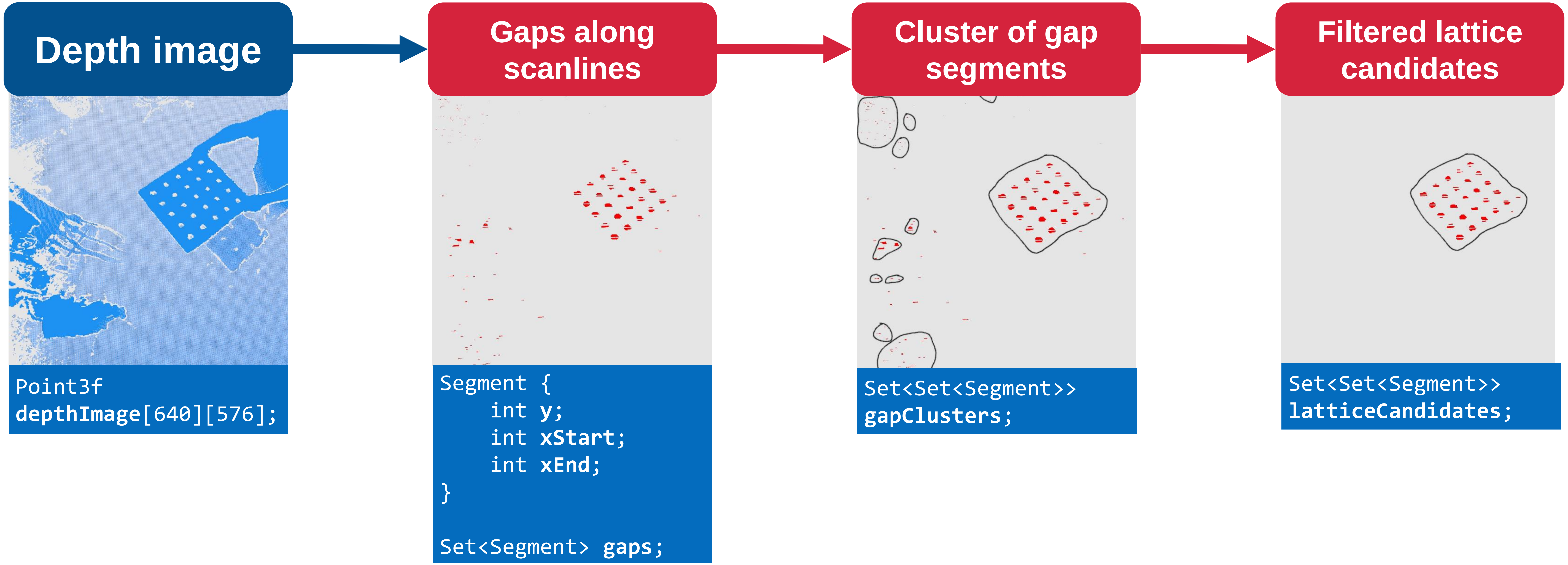
Algorithm Overview

1. **Identify the region of the lattice**
2. Feature point recognition and lattice pose estimation
3. Unique hole center identification
4. Extrinsic calibration



Algorithm Overview

1. Identify the region of the lattice



Depth image

```
Point3f
depthImage[640][576];
```

Gaps along scanlines

```
Segment {
    int y;
    int xStart;
    int xEnd;
}

Set<Segment> gaps;
```

Cluster of gap segments

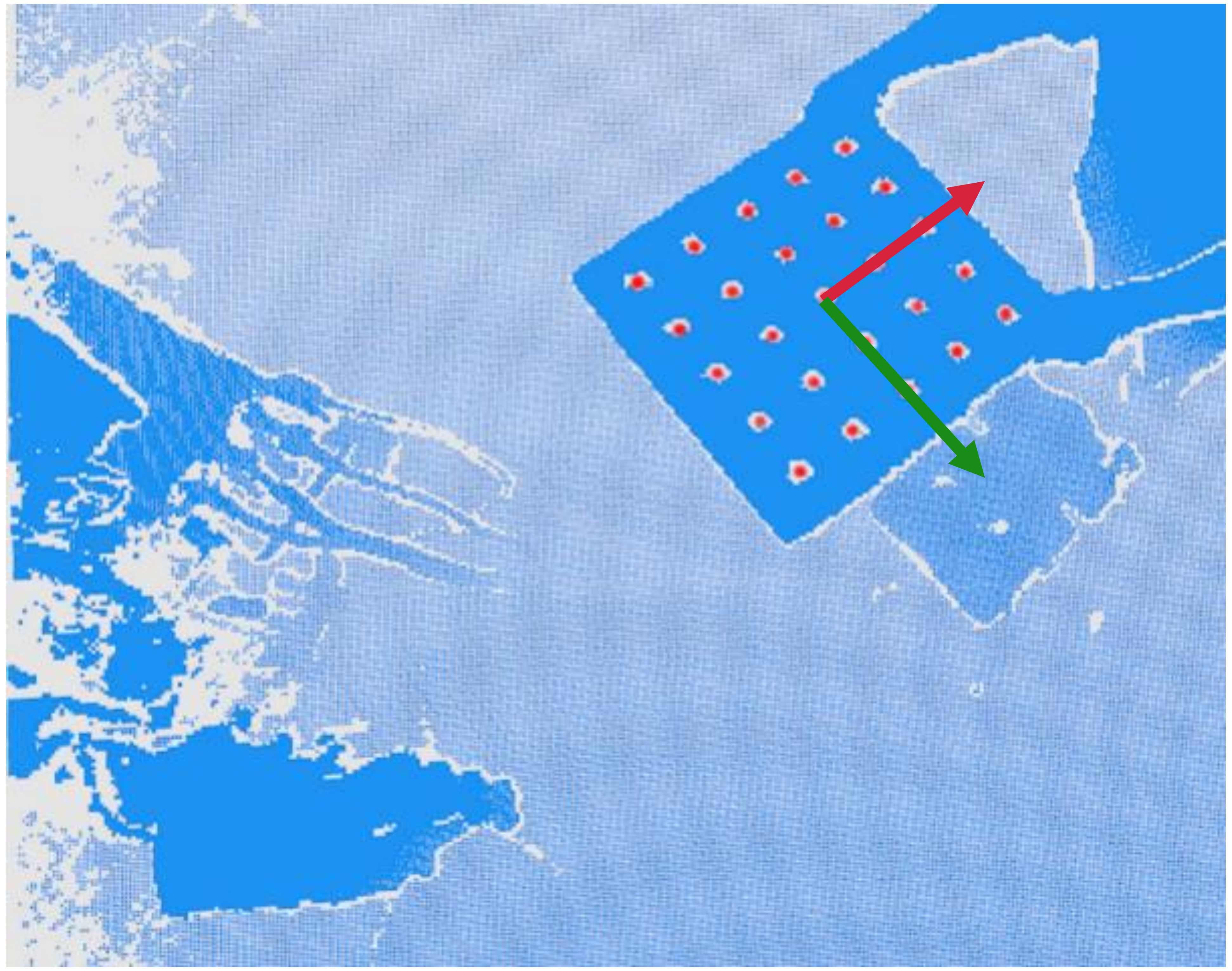
```
Set<Set<Segment>>
gapClusters;
```

Filtered lattice candidates

```
Set<Set<Segment>>
latticeCandidates;
```

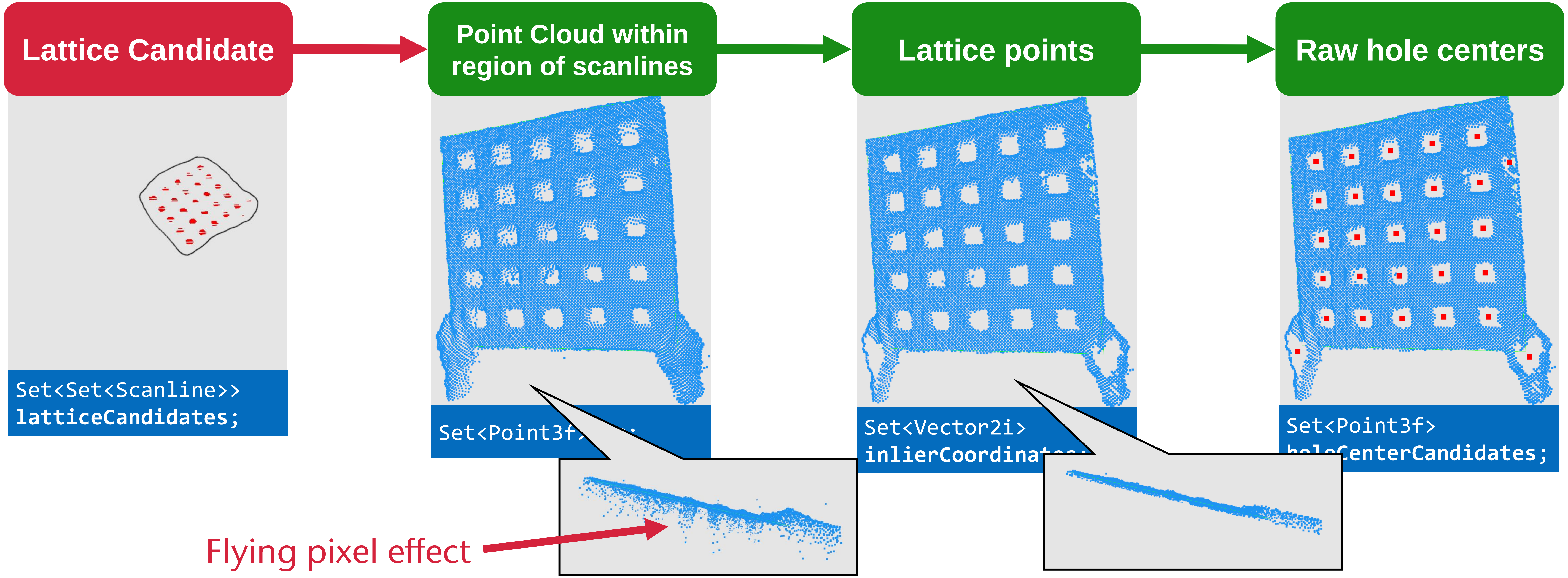
Algorithm Overview

1. Identify the region of the lattice
2. **Feature point recognition and lattice pose estimation**
3. Unique hole center identification
4. Extrinsic calibration



Algorithm Overview

2. Feature point recognition and lattice pose estimation



Lattice Candidate → Point Cloud within region of scanlines → Lattice points → Raw hole centers

```
Set<Set<Scanline>>
latticeCandidates;
```

```
Set<Point3f>
```

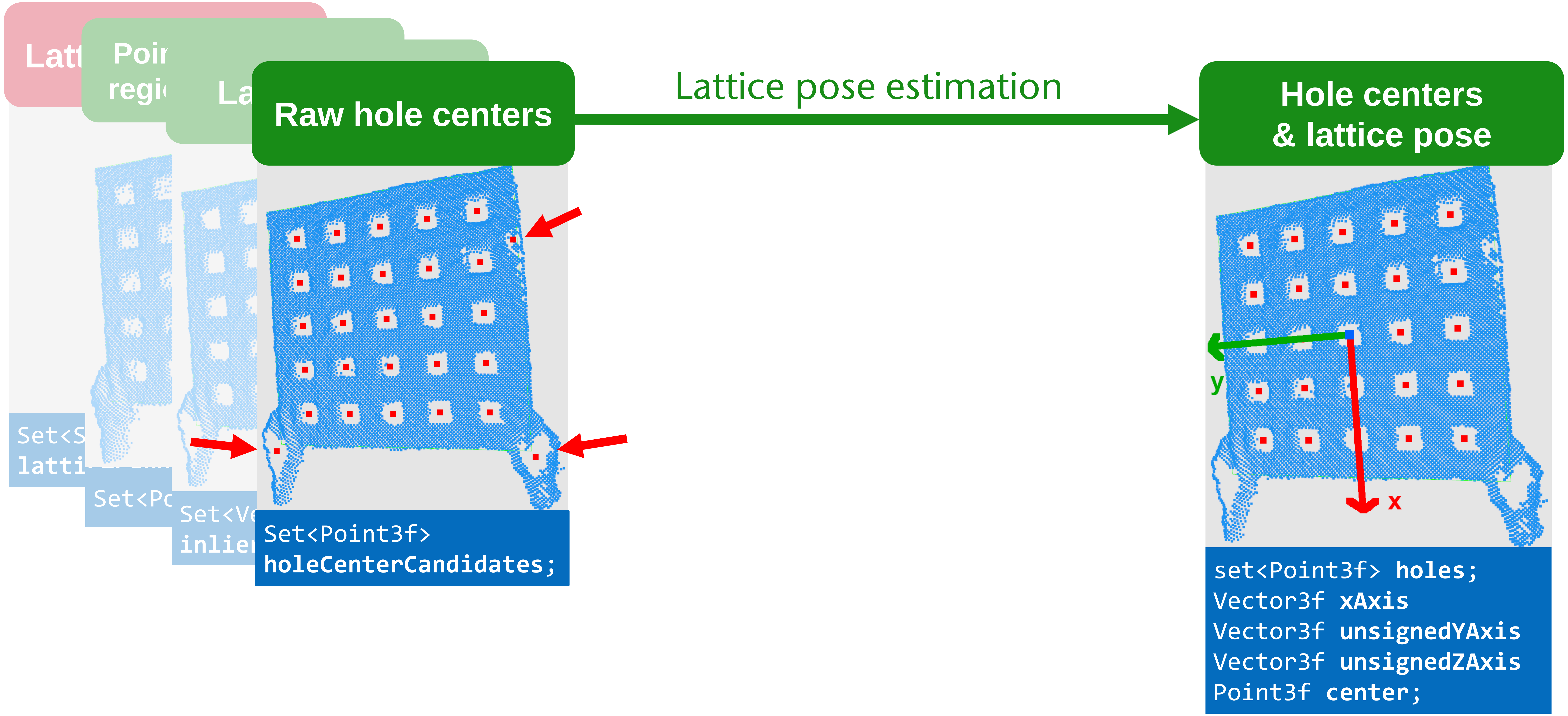
```
Set<Vector2i>
inlierCoordinates;
```

```
Set<Point3f>
holeCenterCandidates;
```

Flying pixel effect

Algorithm Overview

2. Feature point recognition and lattice pose estimation



Raw hole centers

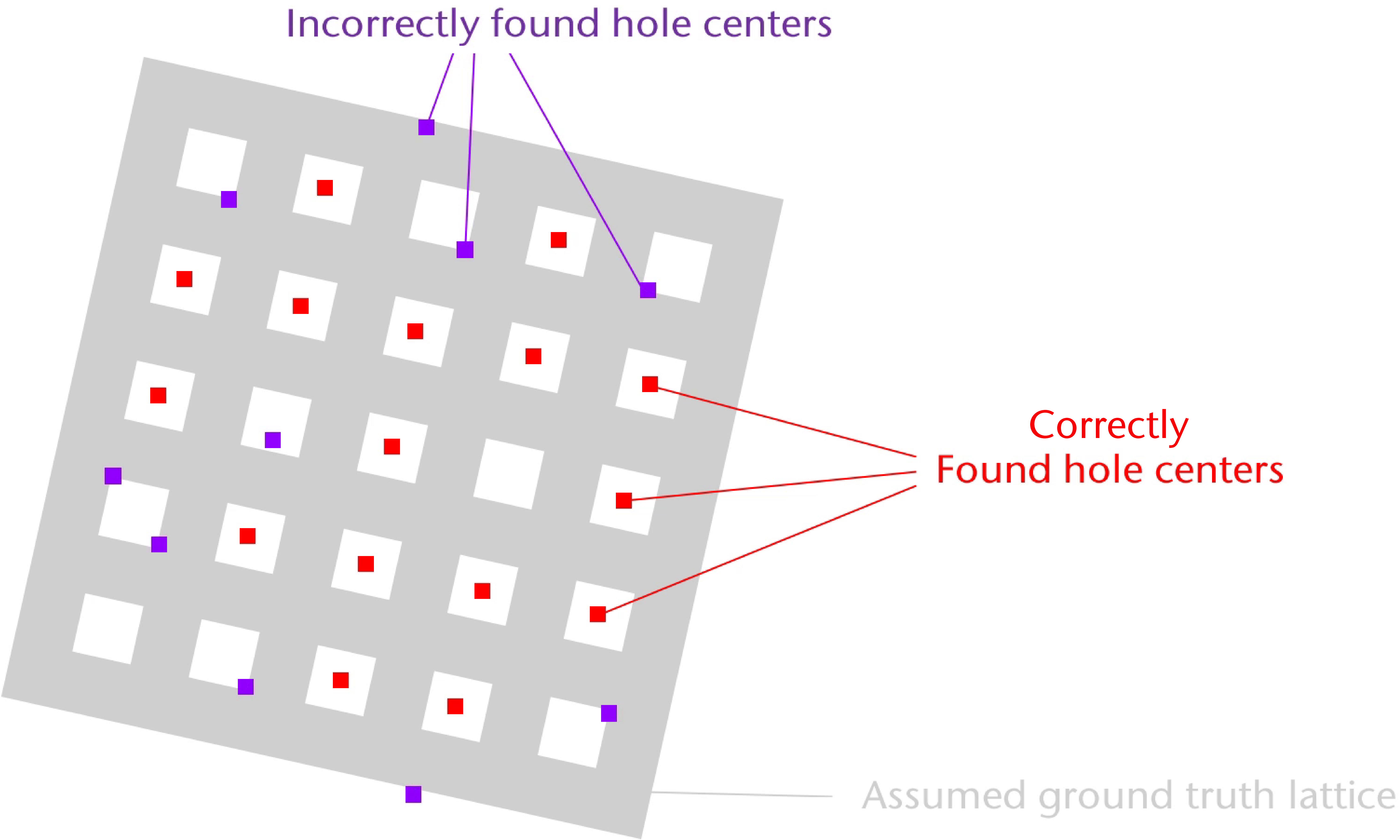
```
Set<Point3f>
holeCenterCandidates;
```

Lattice pose estimation

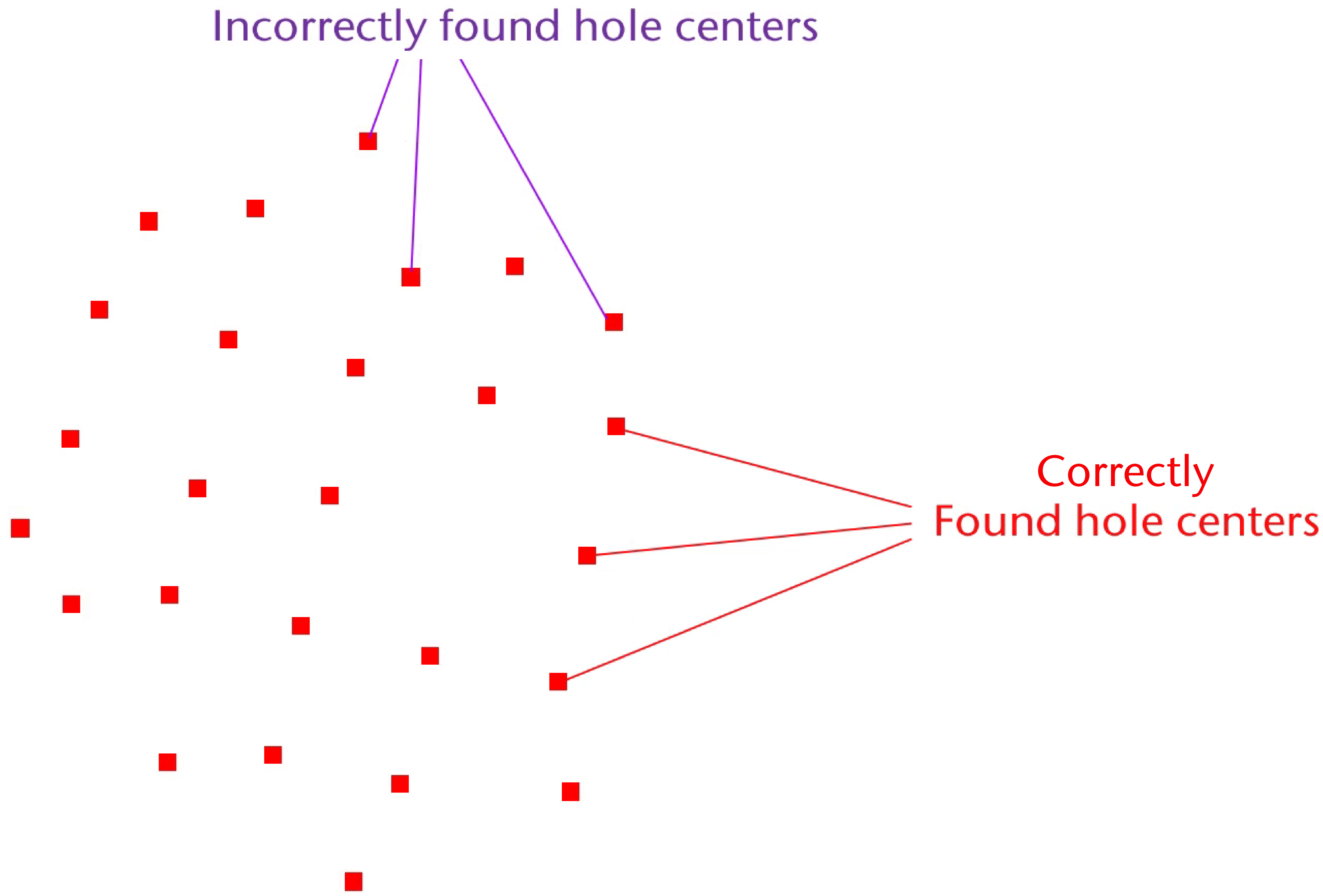
Hole centers
& lattice pose

```
set<Point3f> holes;
Vector3f xAxis
Vector3f unsignedYAxis
Vector3f unsignedZAxis
Point3f center;
```

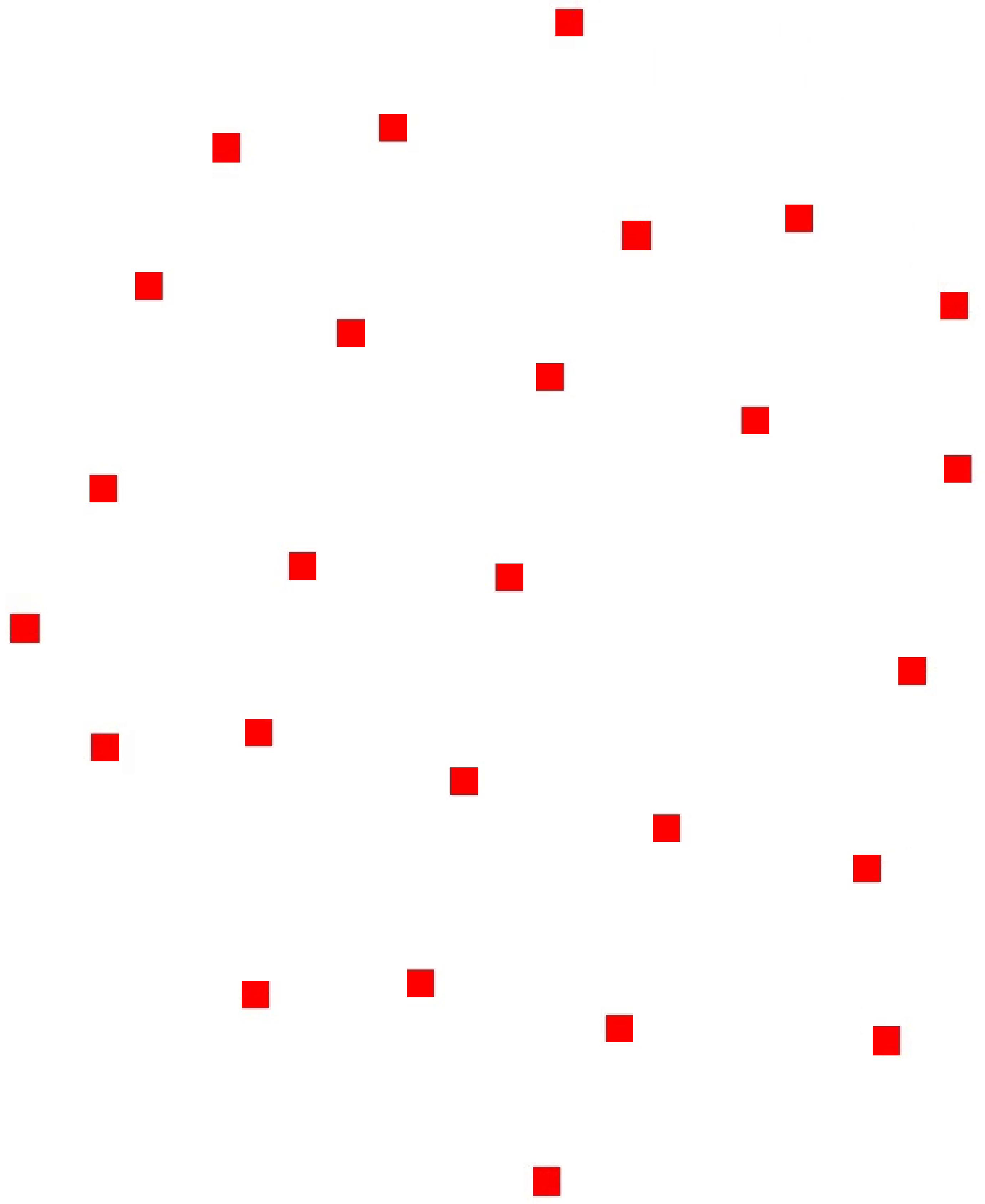
Algorithm Overview



Algorithm Overview

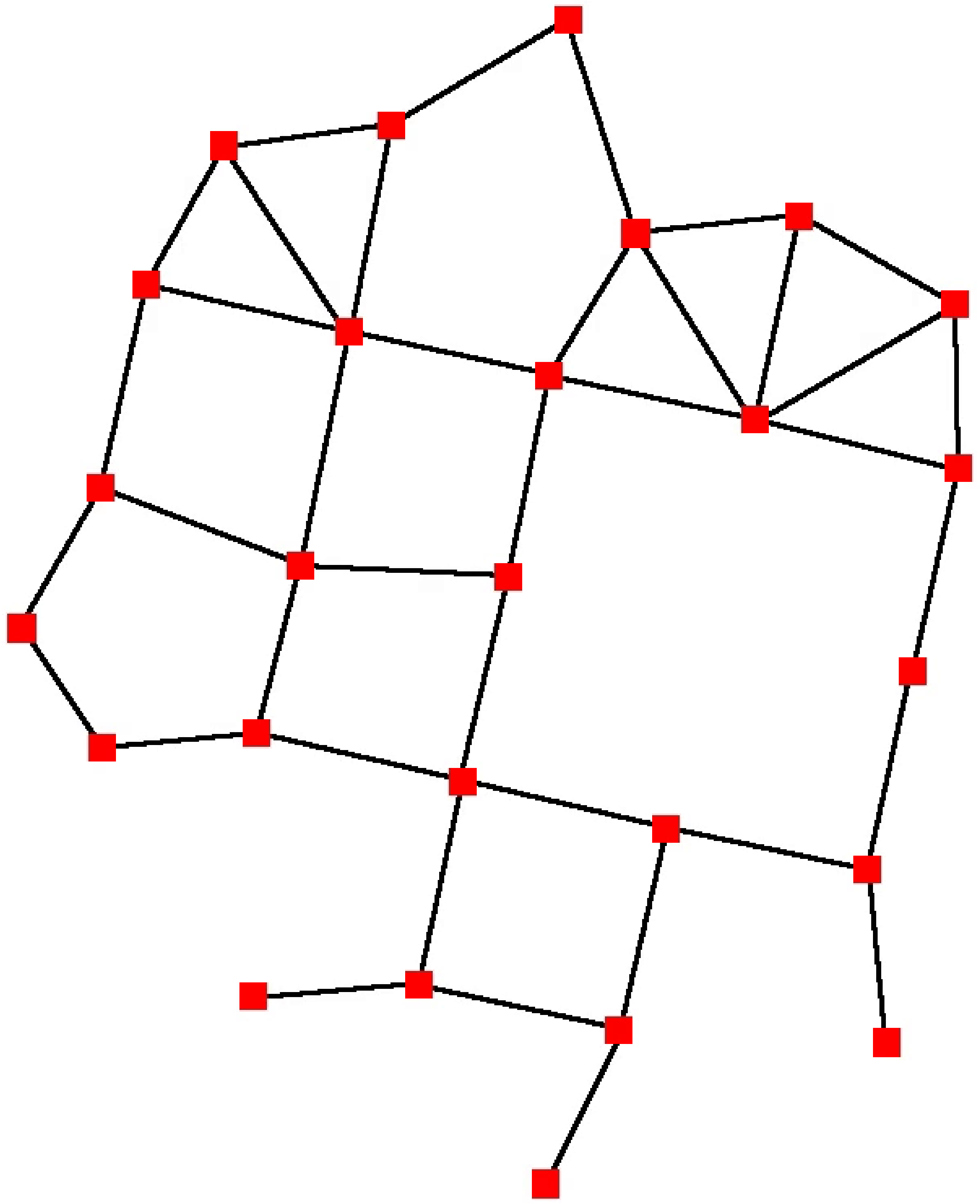


Algorithm Overview

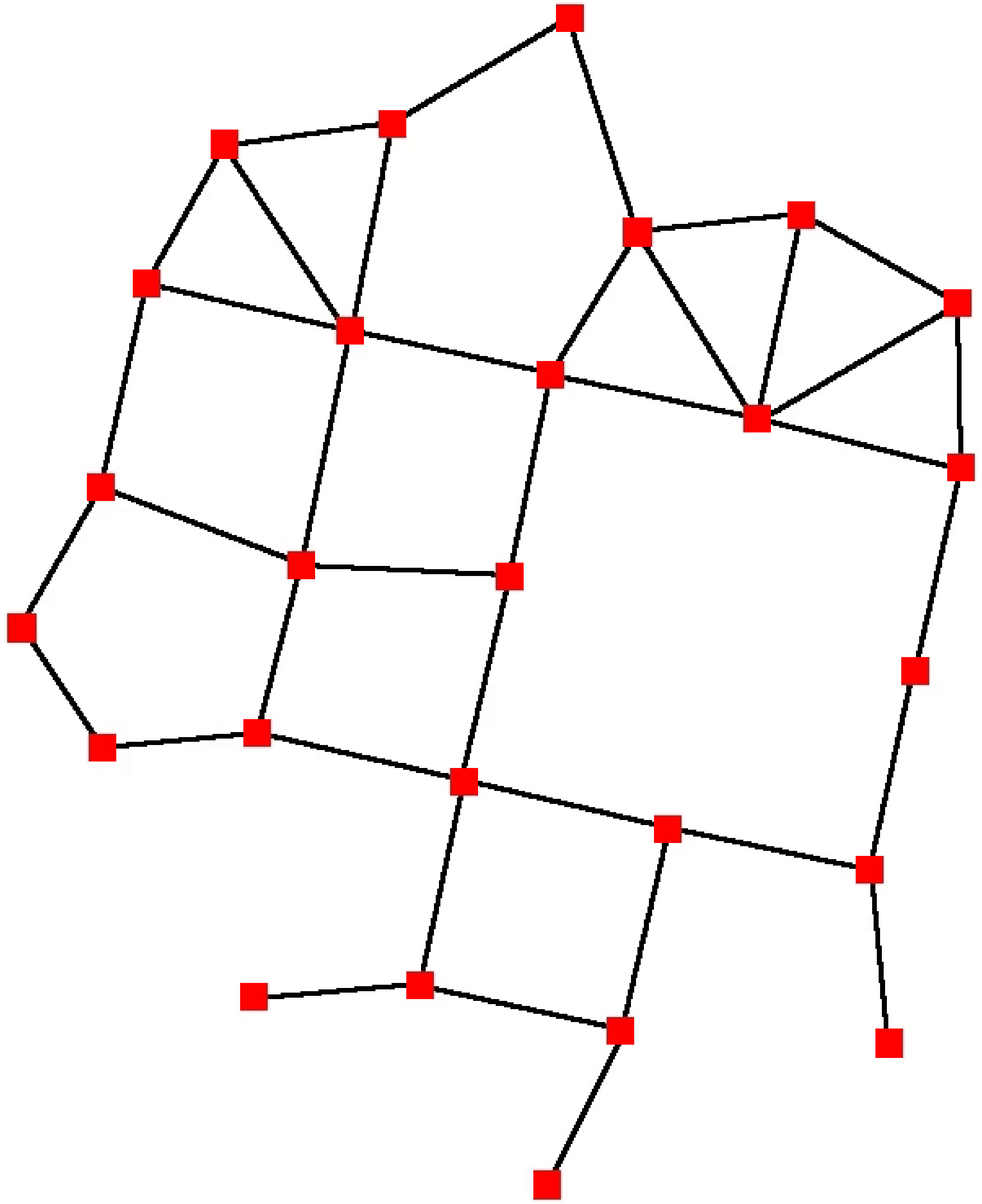


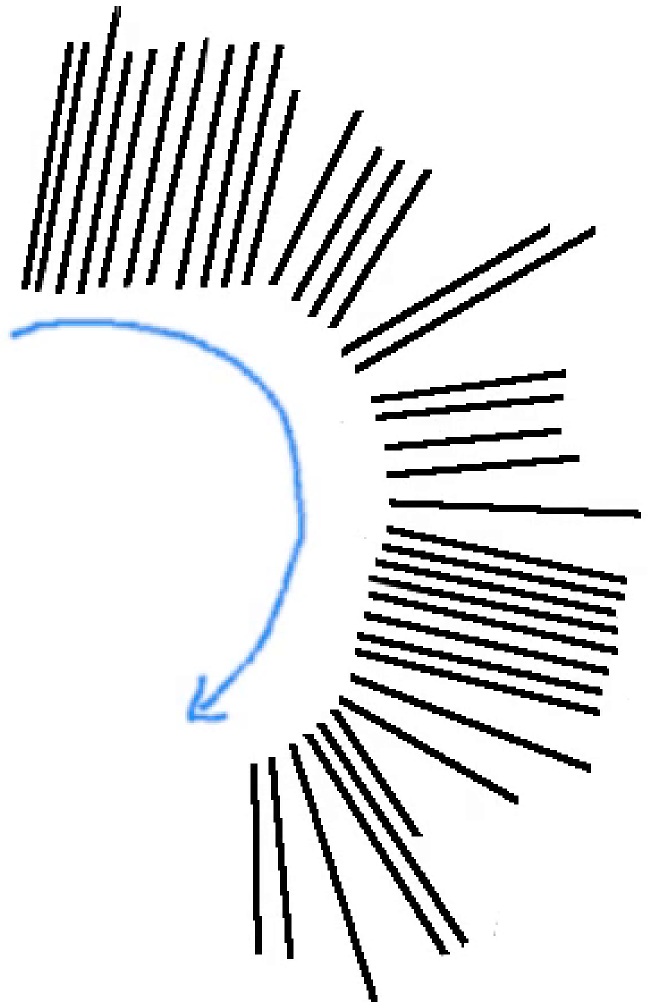
Algorithm Overview

Create proximity graph on hole centers



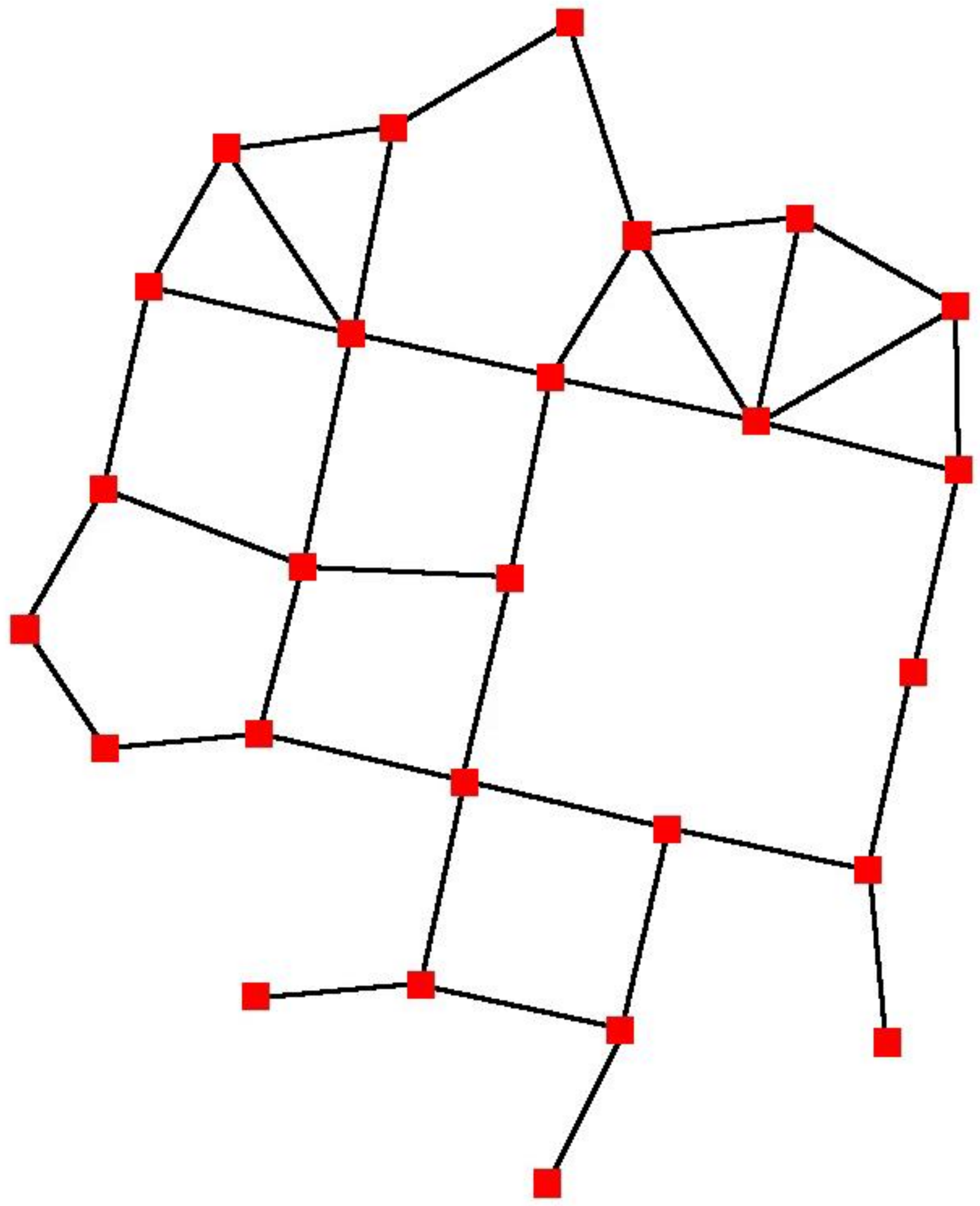
Algorithm Overview

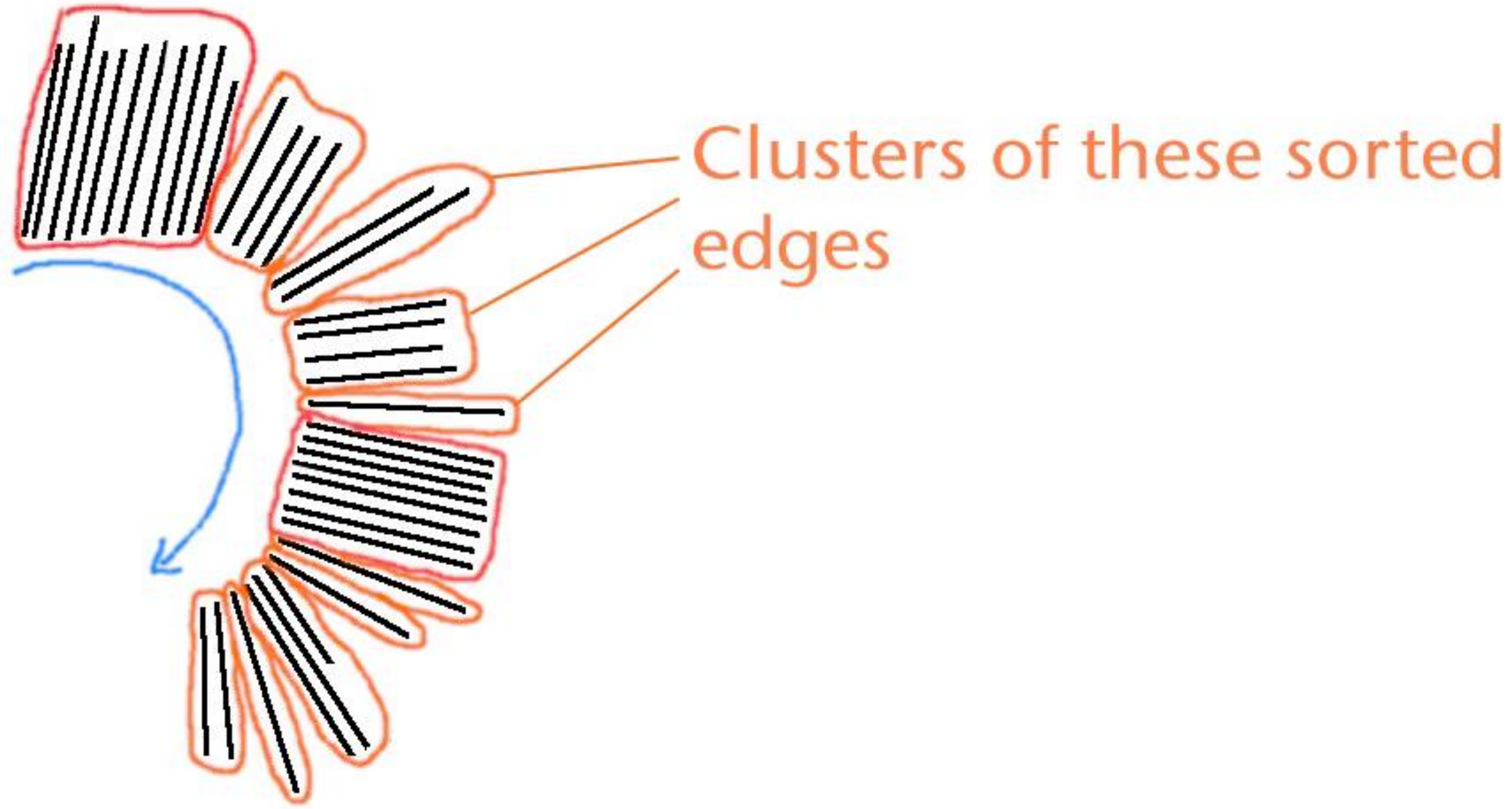




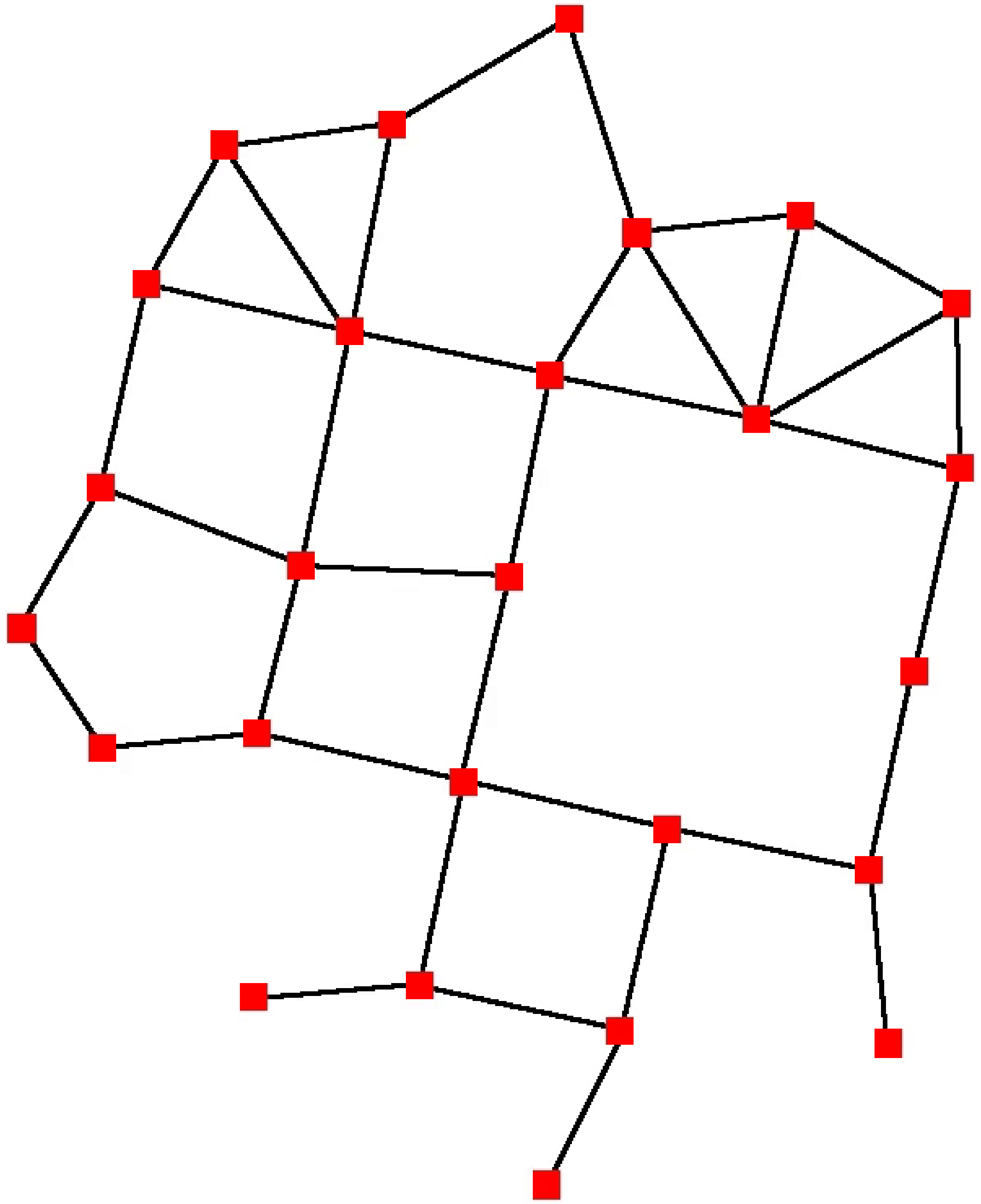
Edges sorted by their angle they subtend with the x-axis

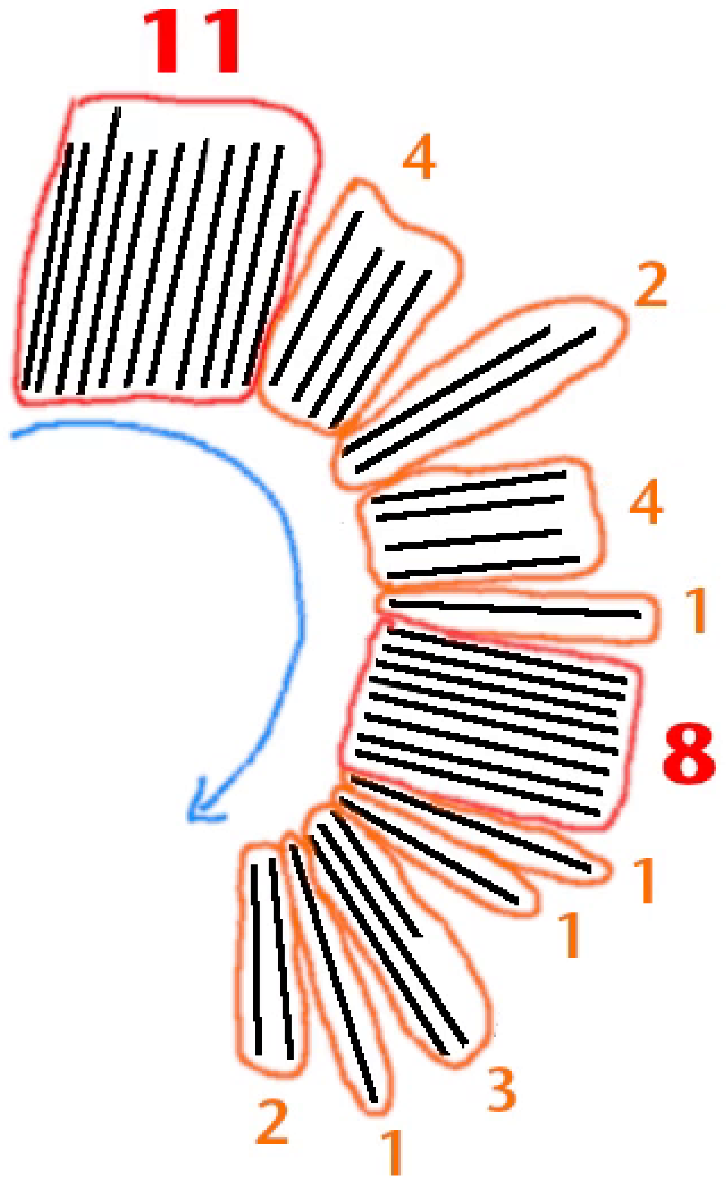
Algorithm Overview



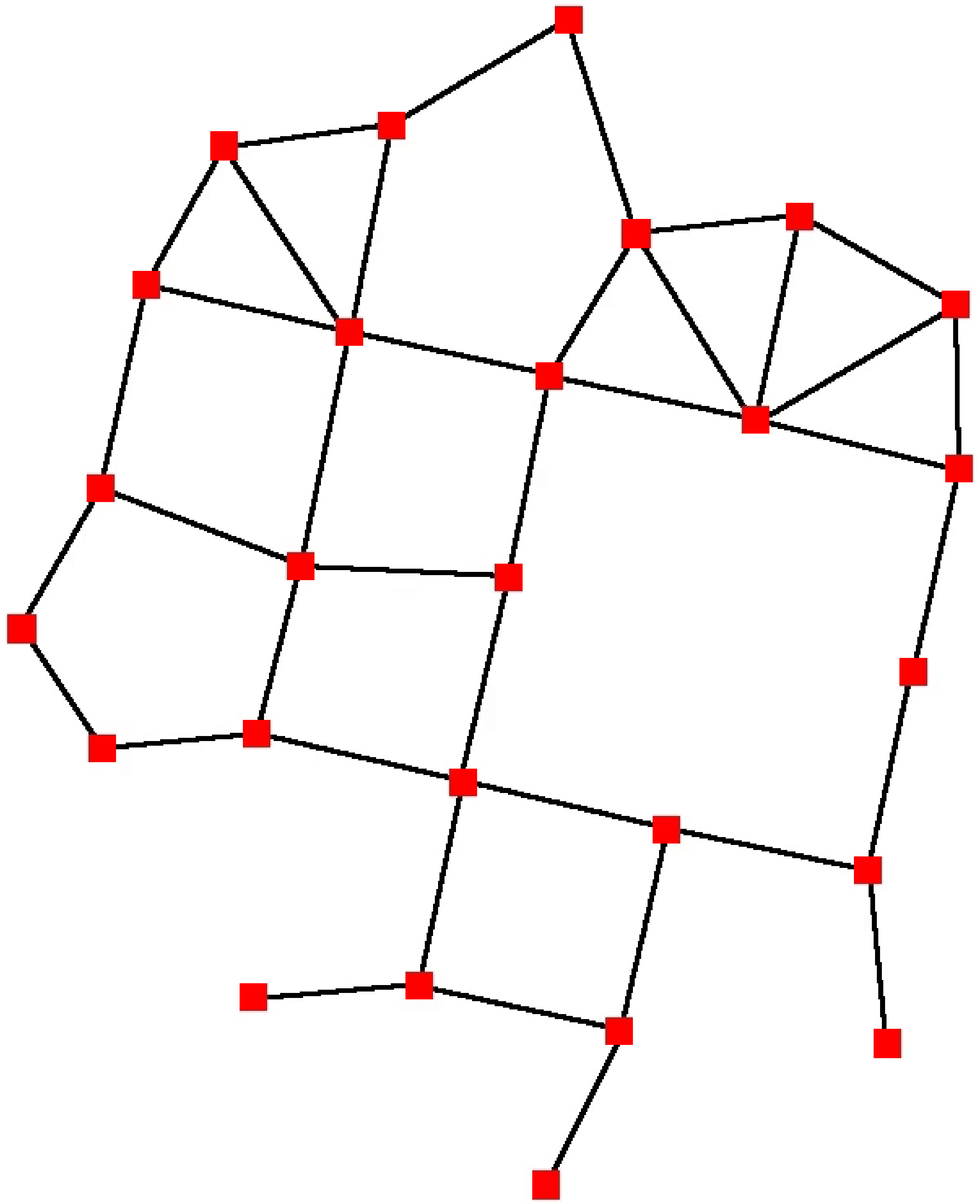


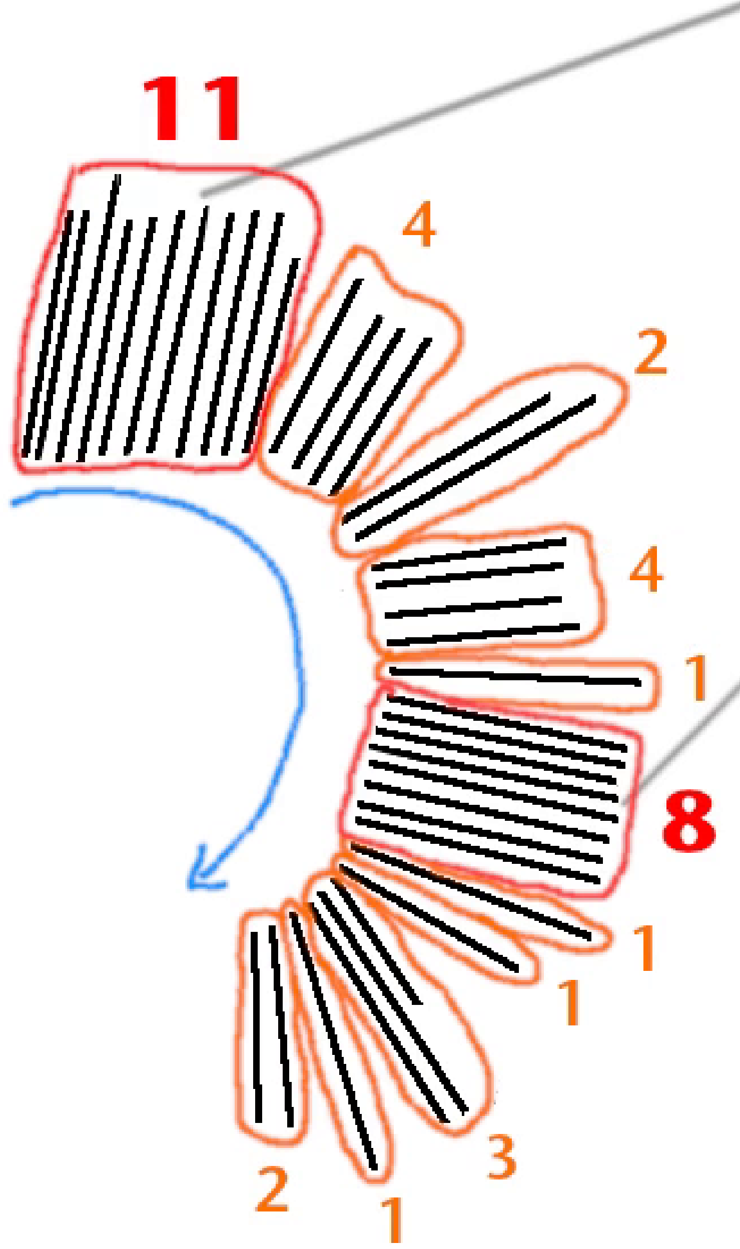
Algorithm Overview





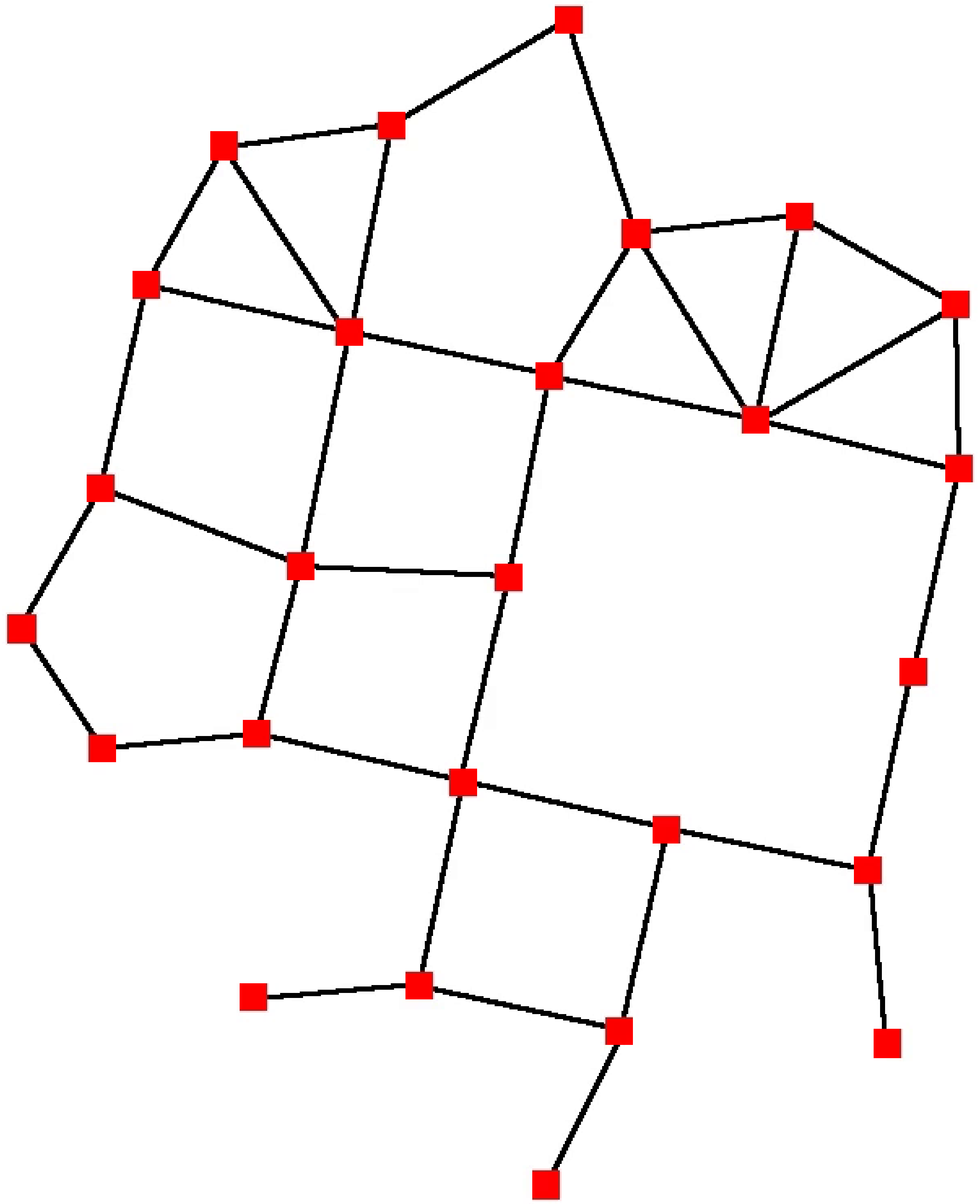
Algorithm Overview

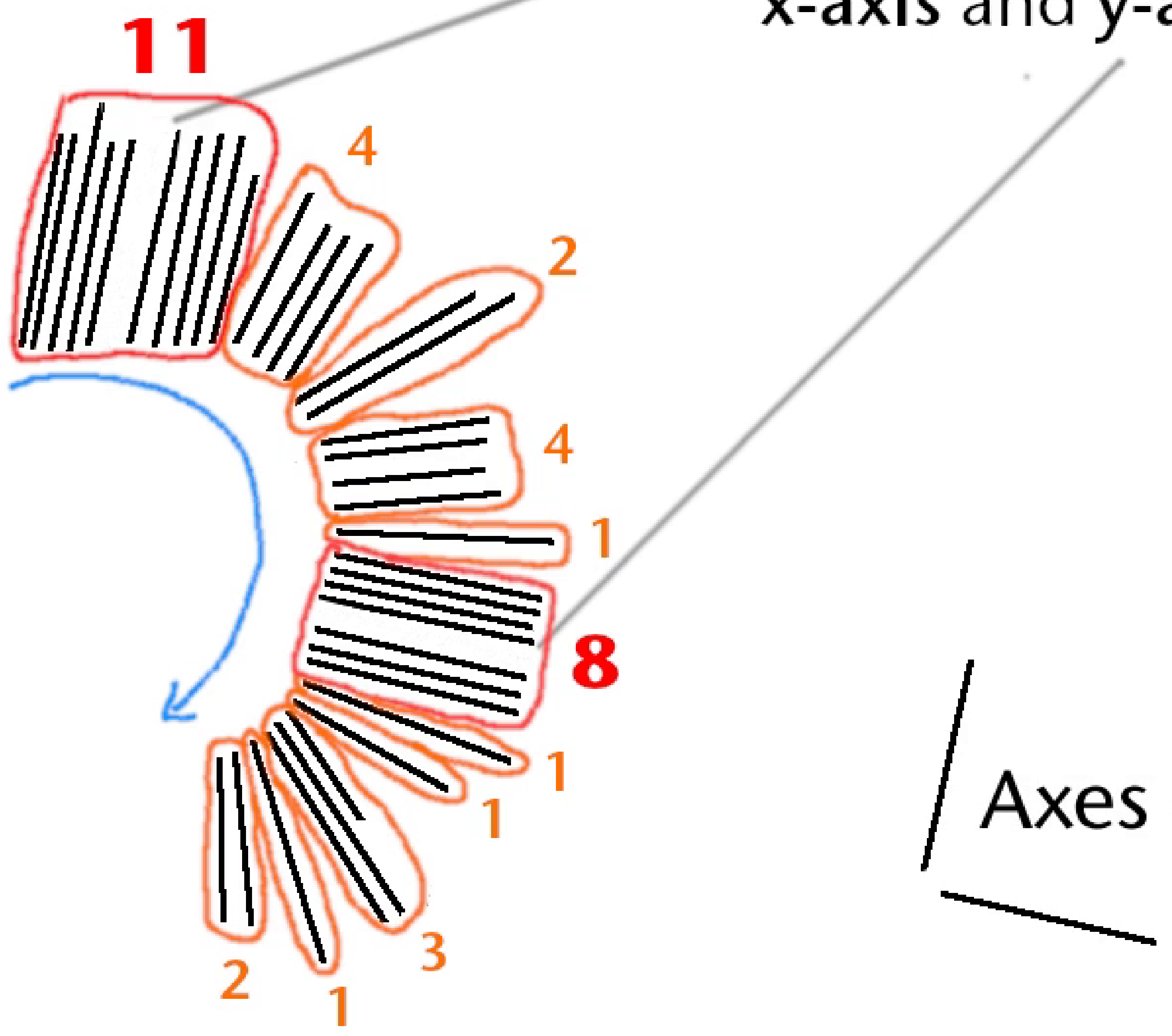




Select the **median** of each of the **two largest clusters** as the **x-axis** and **y-axis directions**

Algorithm Overview

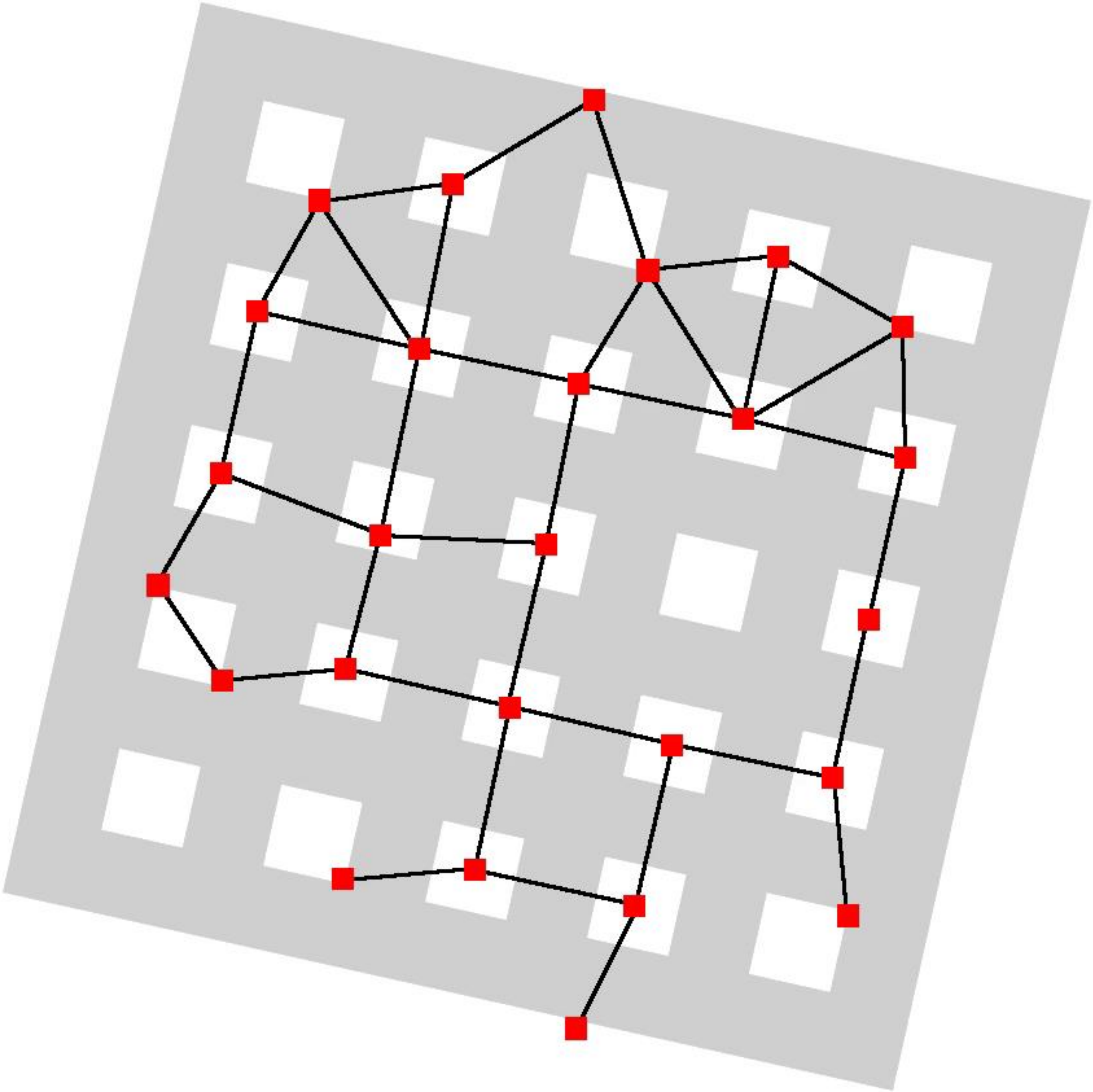


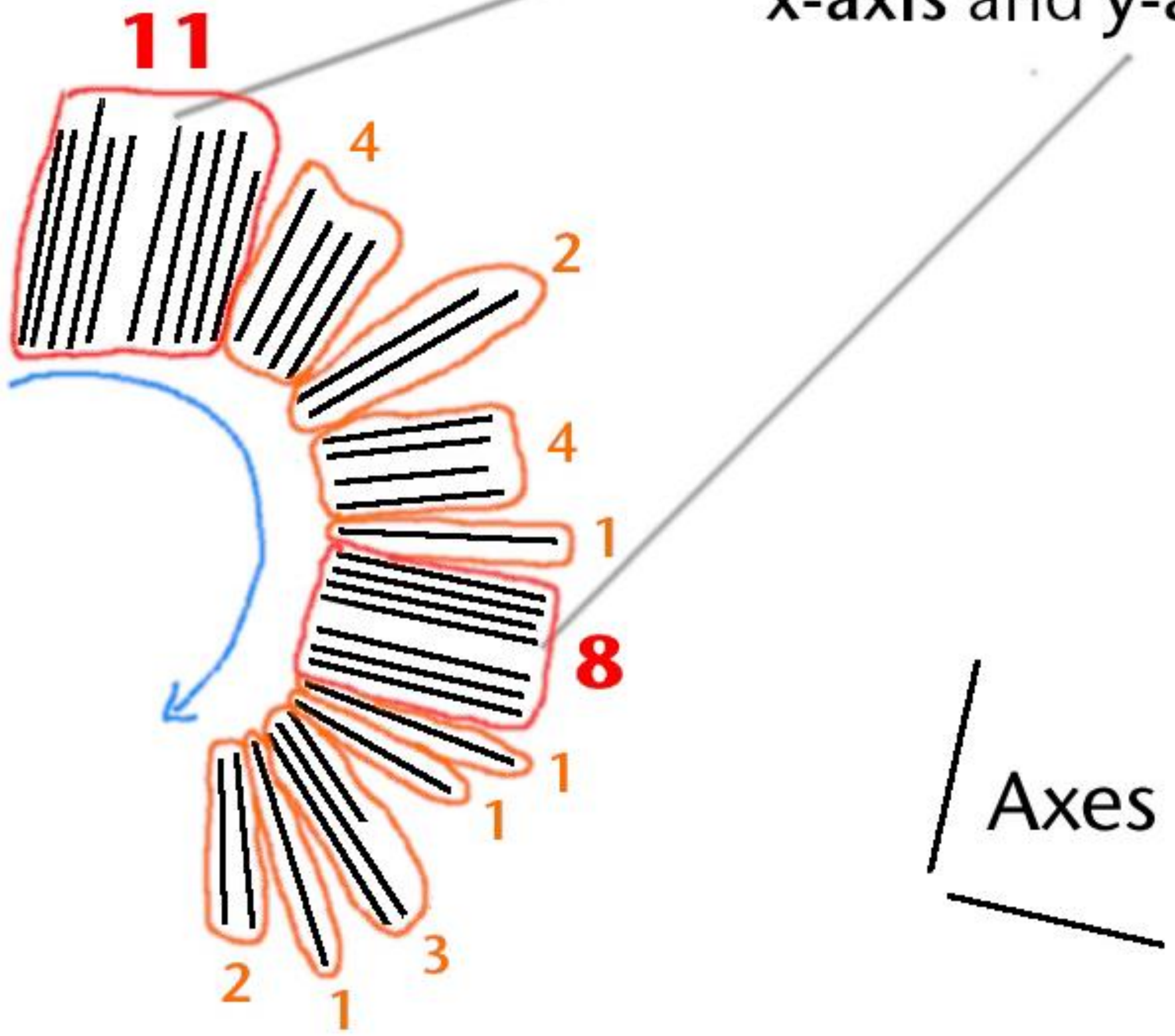


Select the **median** of each of the **two largest clusters** as the **x-axis** and **y-axis directions**

Axes directions

Algorithm Overview

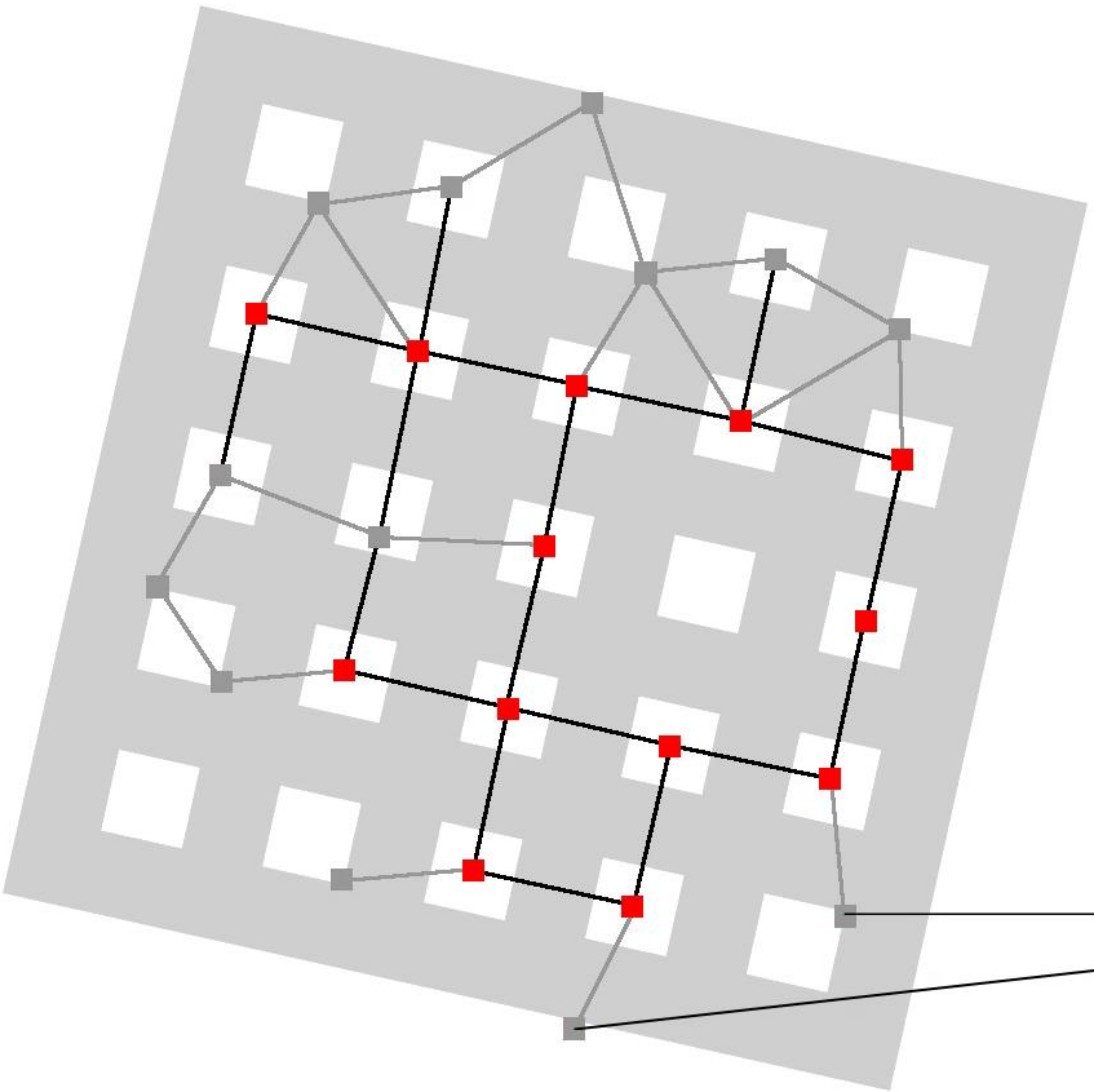




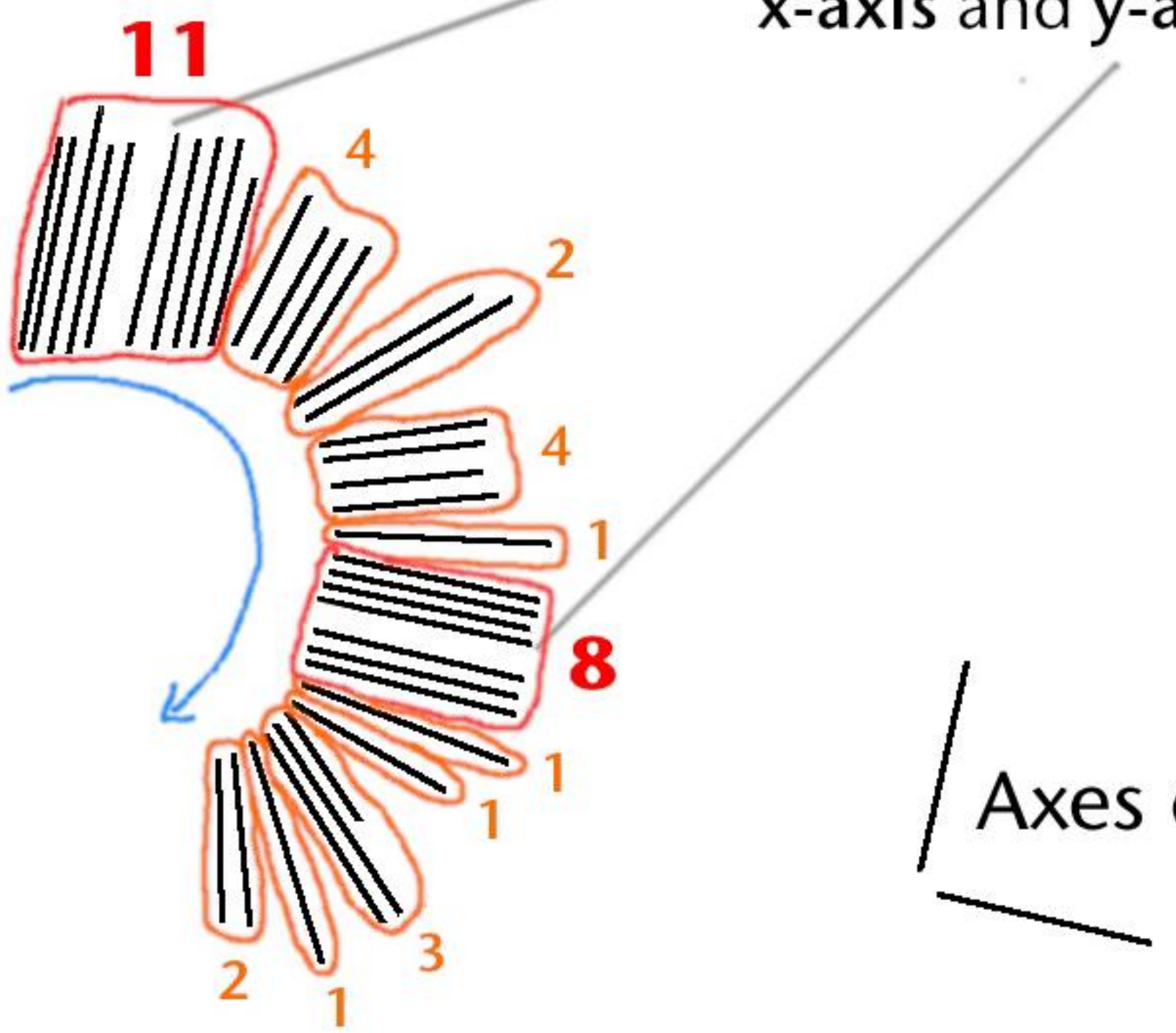
Select the **median** of each of the **two largest clusters** as the **x-axis** and **y-axis directions**

Axes directions

Algorithm Overview



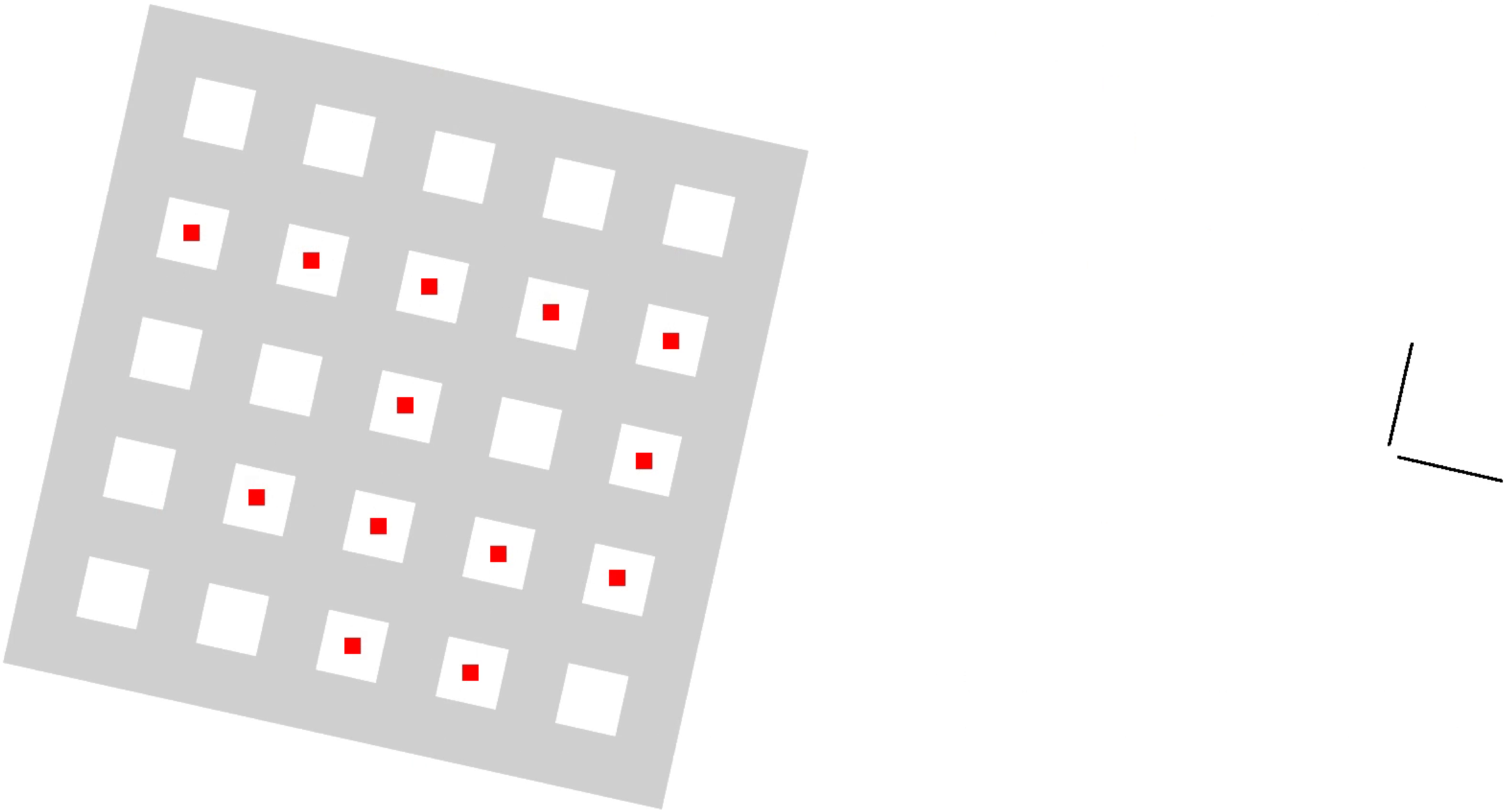
Outliers removed based on cluster membership of neighboring edges



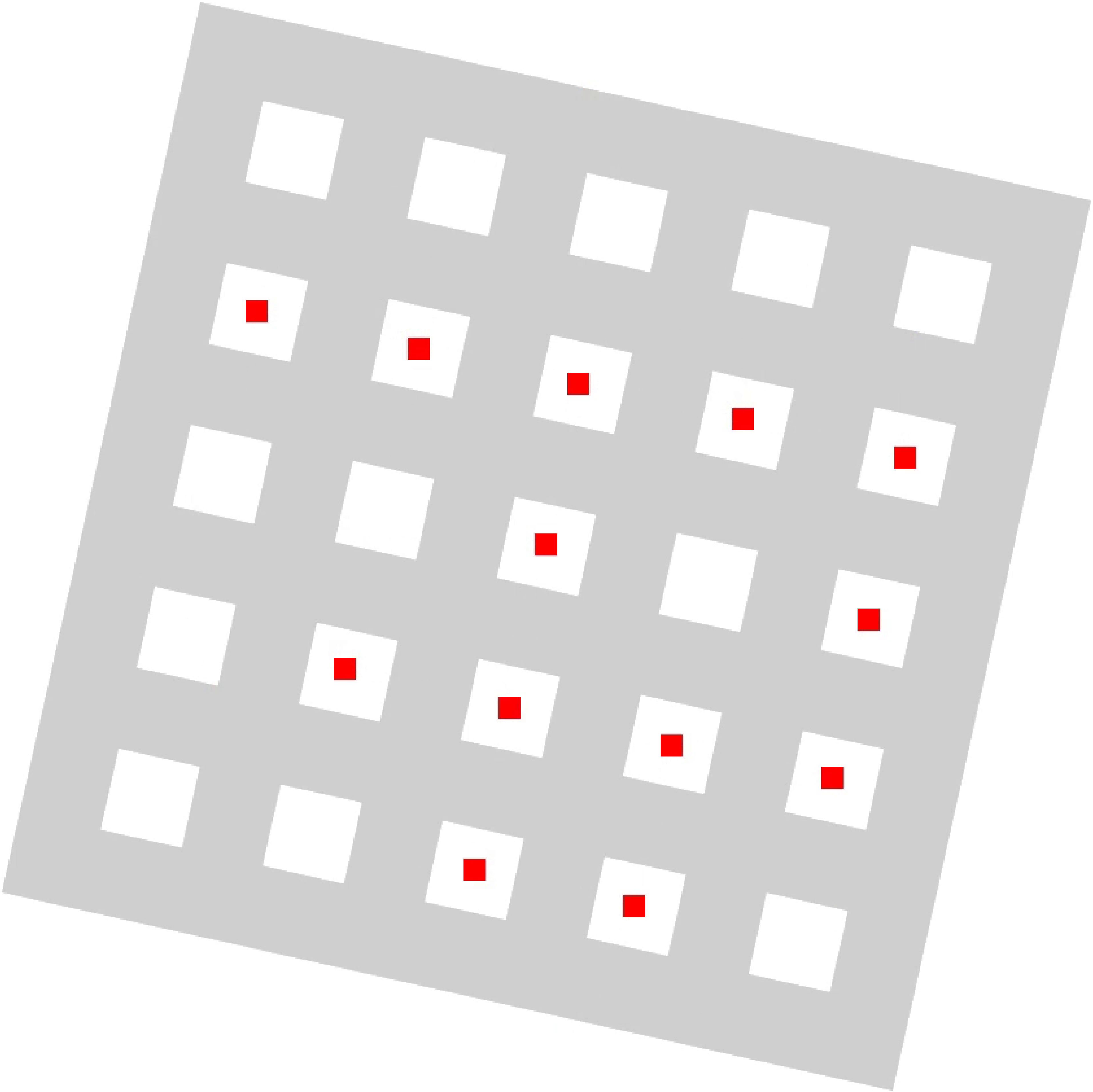
Select the **median** of each of the **two largest clusters** as the **x-axis** and **y-axis directions**

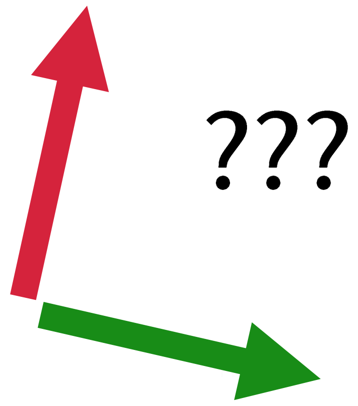
Axes directions

Algorithm Overview

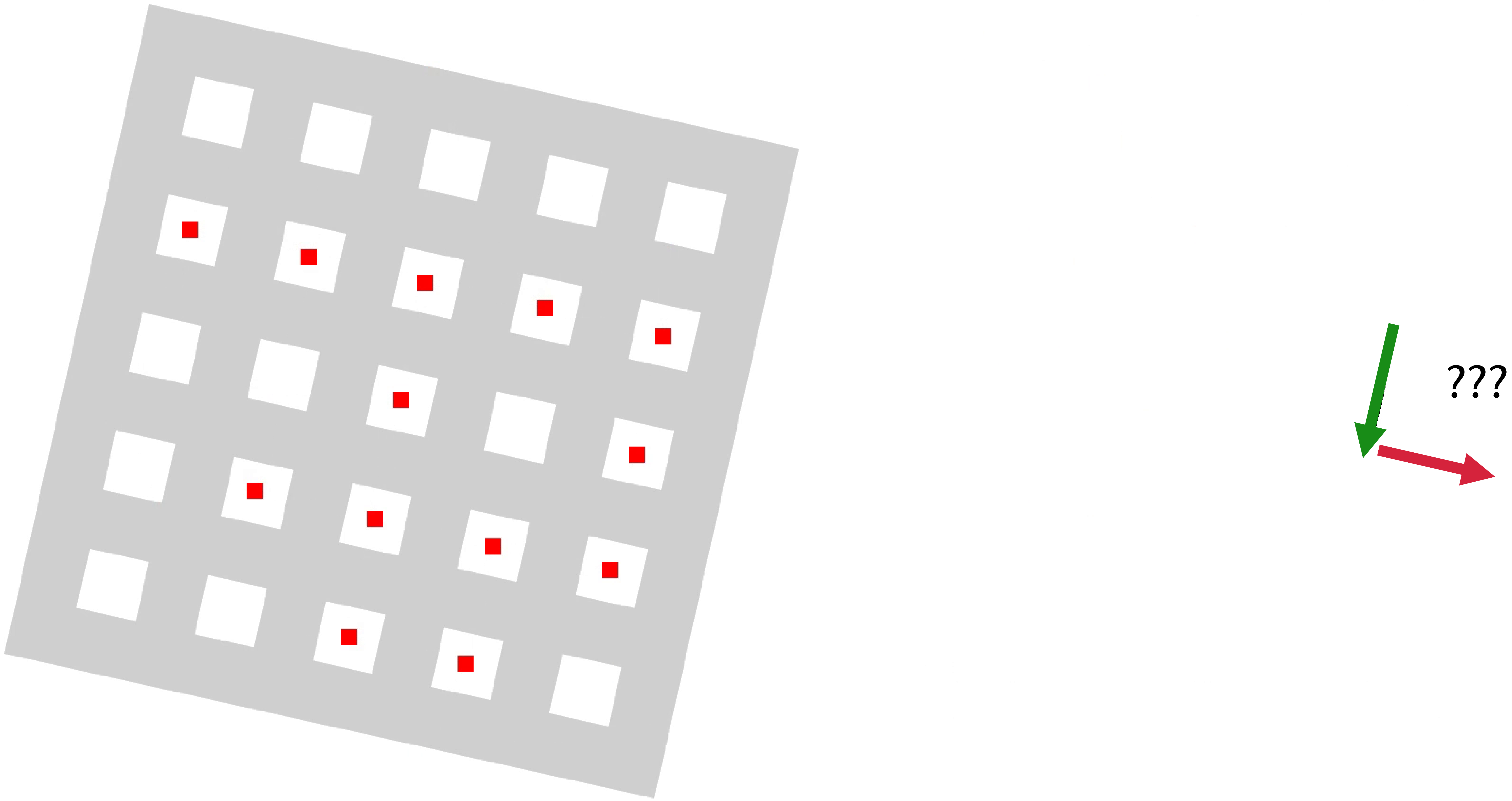


Algorithm Overview

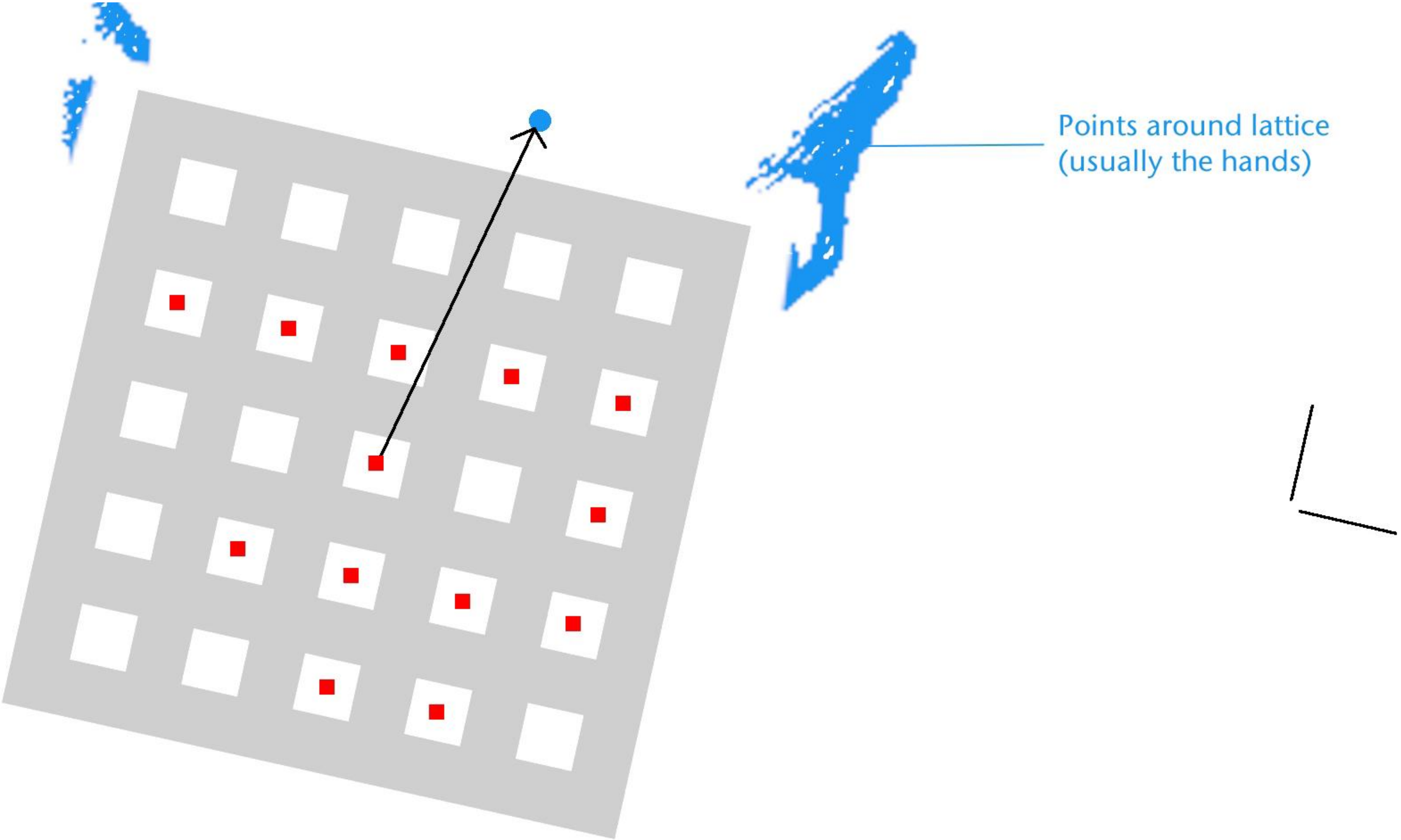




Algorithm Overview

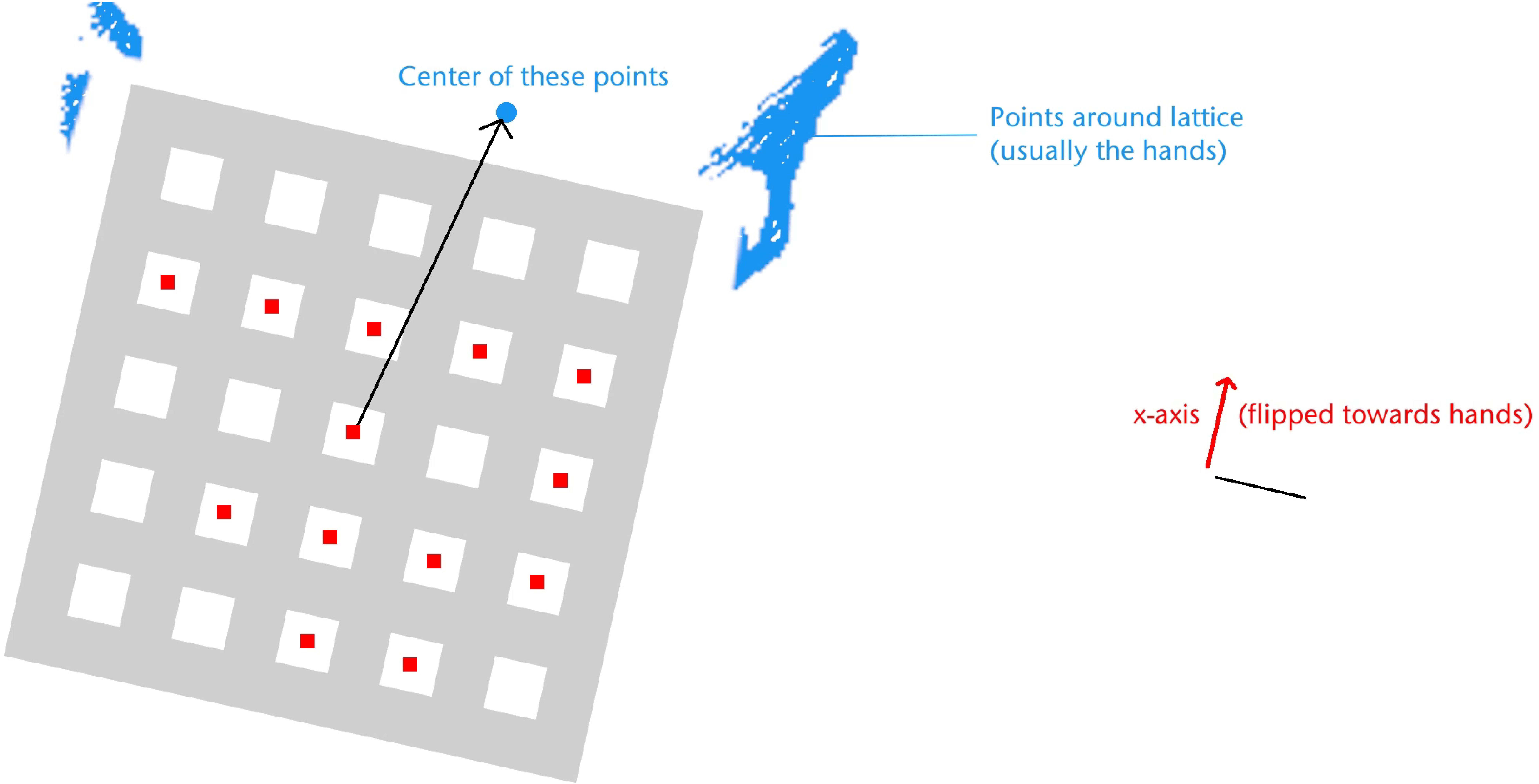


Algorithm Overview

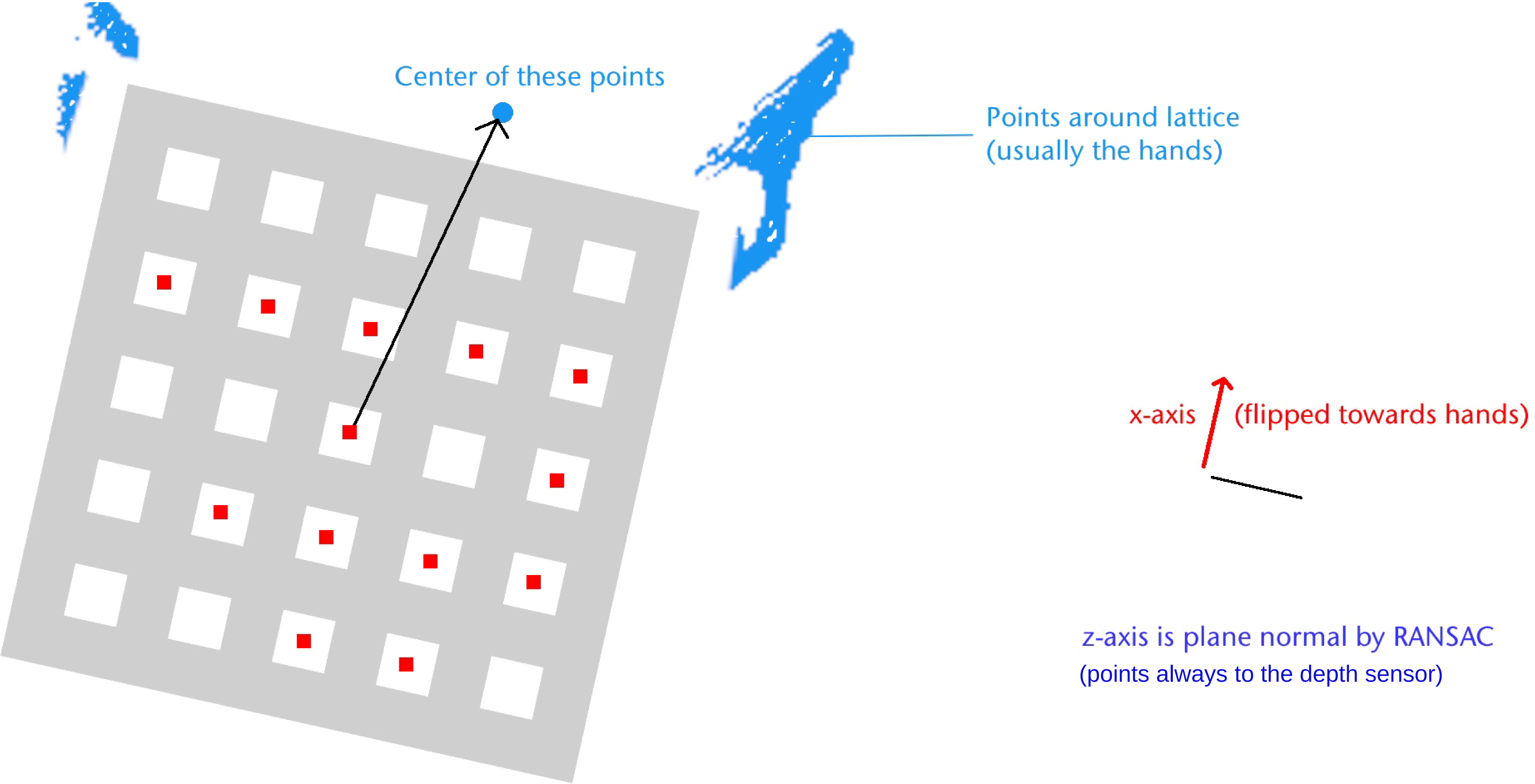


Points around lattice
(usually the hands)

Algorithm Overview



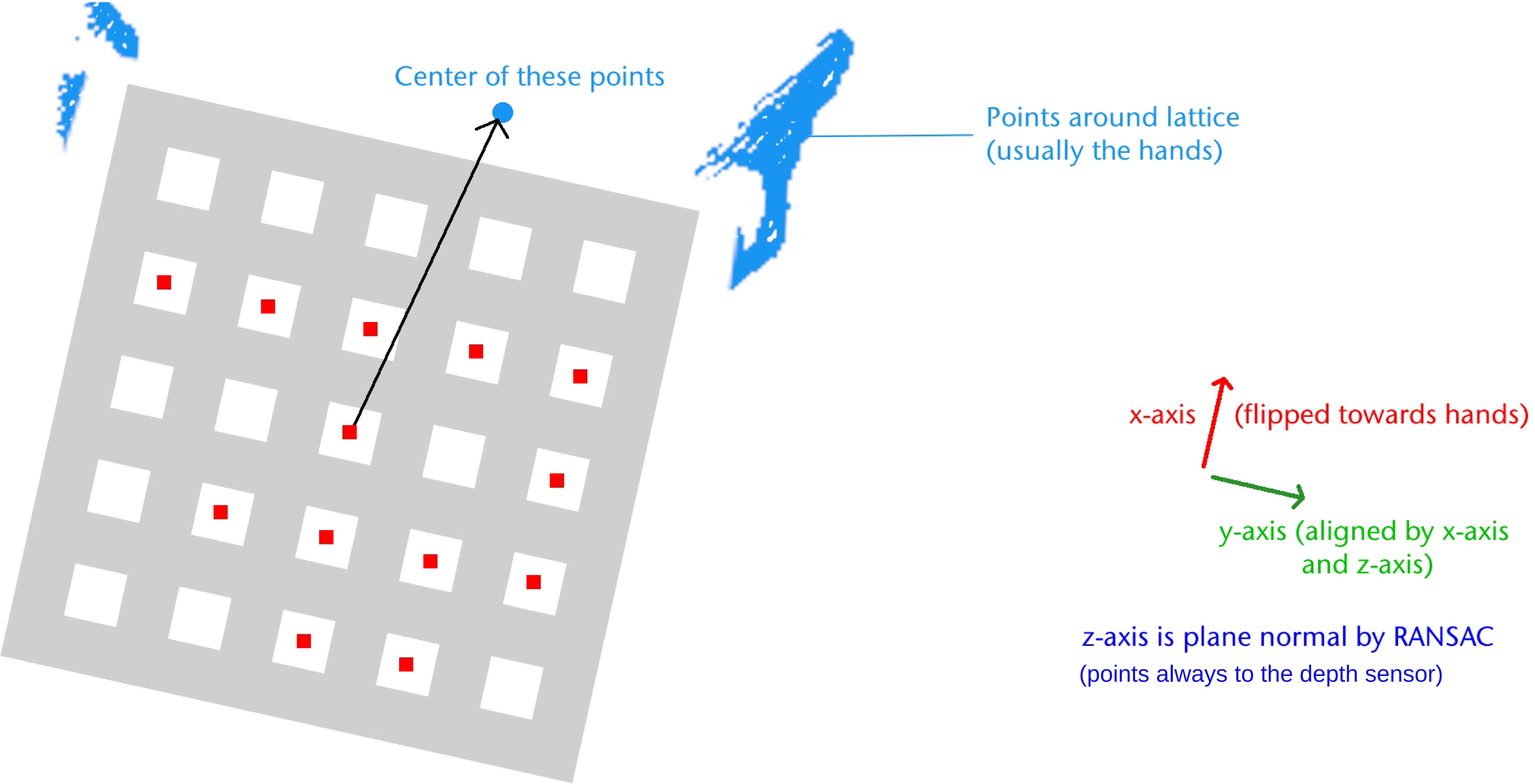
Algorithm Overview



z-axis is plane normal by RANSAC

(points always to the depth sensor)

Algorithm Overview

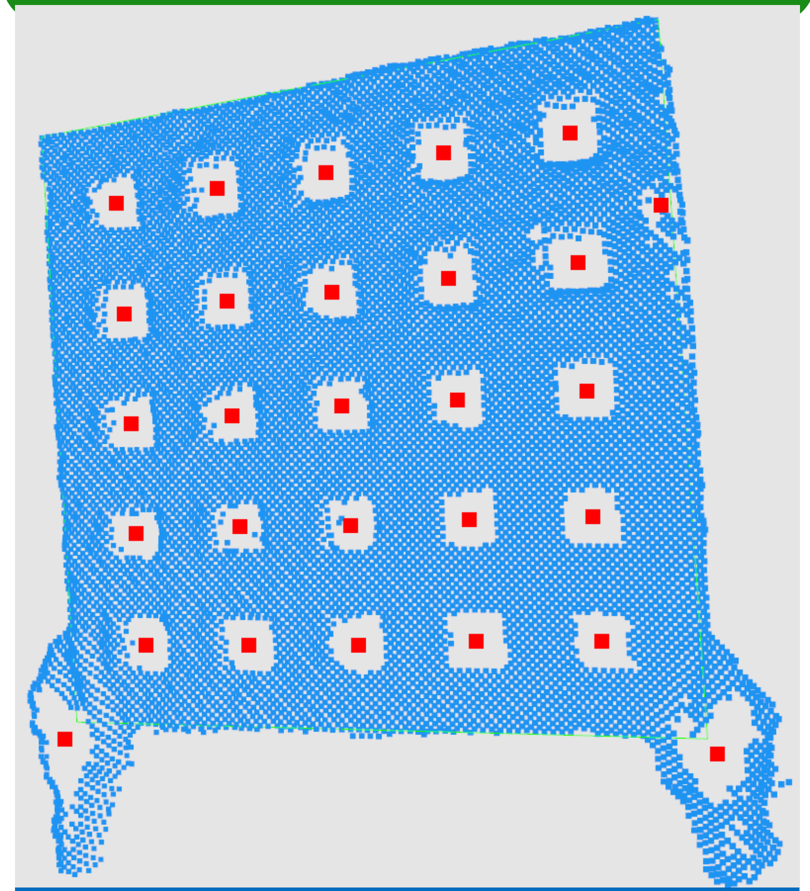


z-axis is plane normal by RANSAC
(points always to the depth sensor)

Algorithm Overview

2. Feature point recognition and lattice pose estimation

Raw hole centers

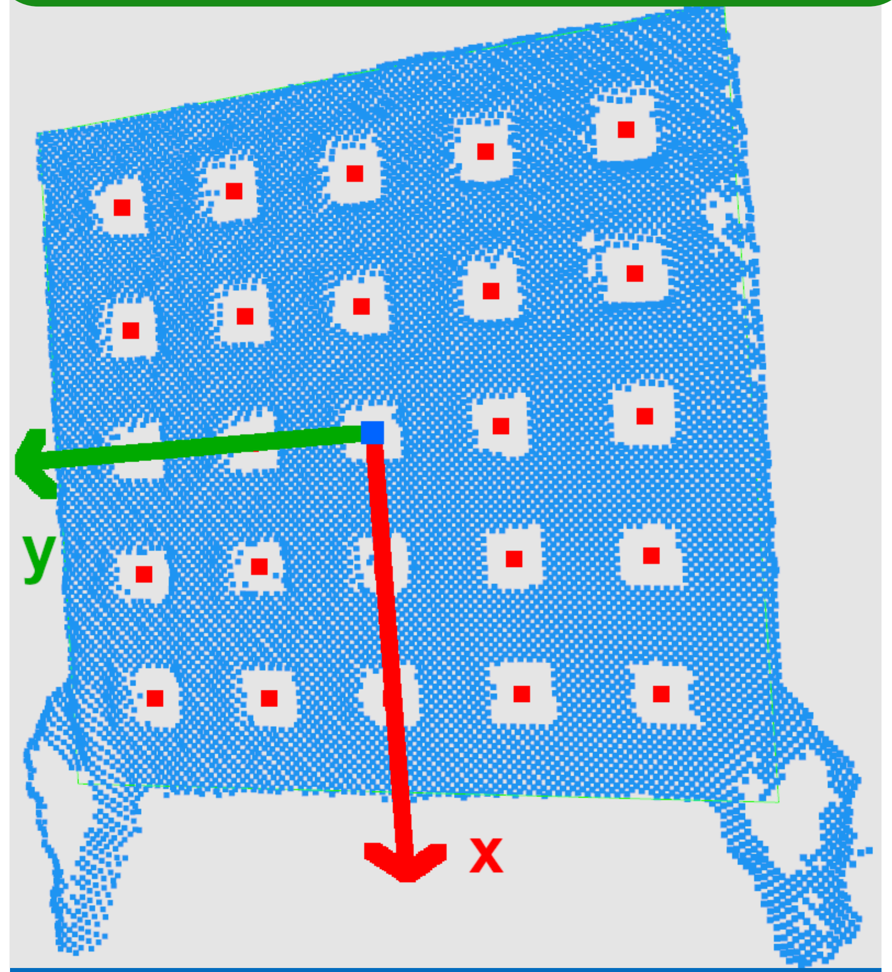


```
Set<Point3f>
holeCenterCandidates;
```

Pose estimation

heuristic

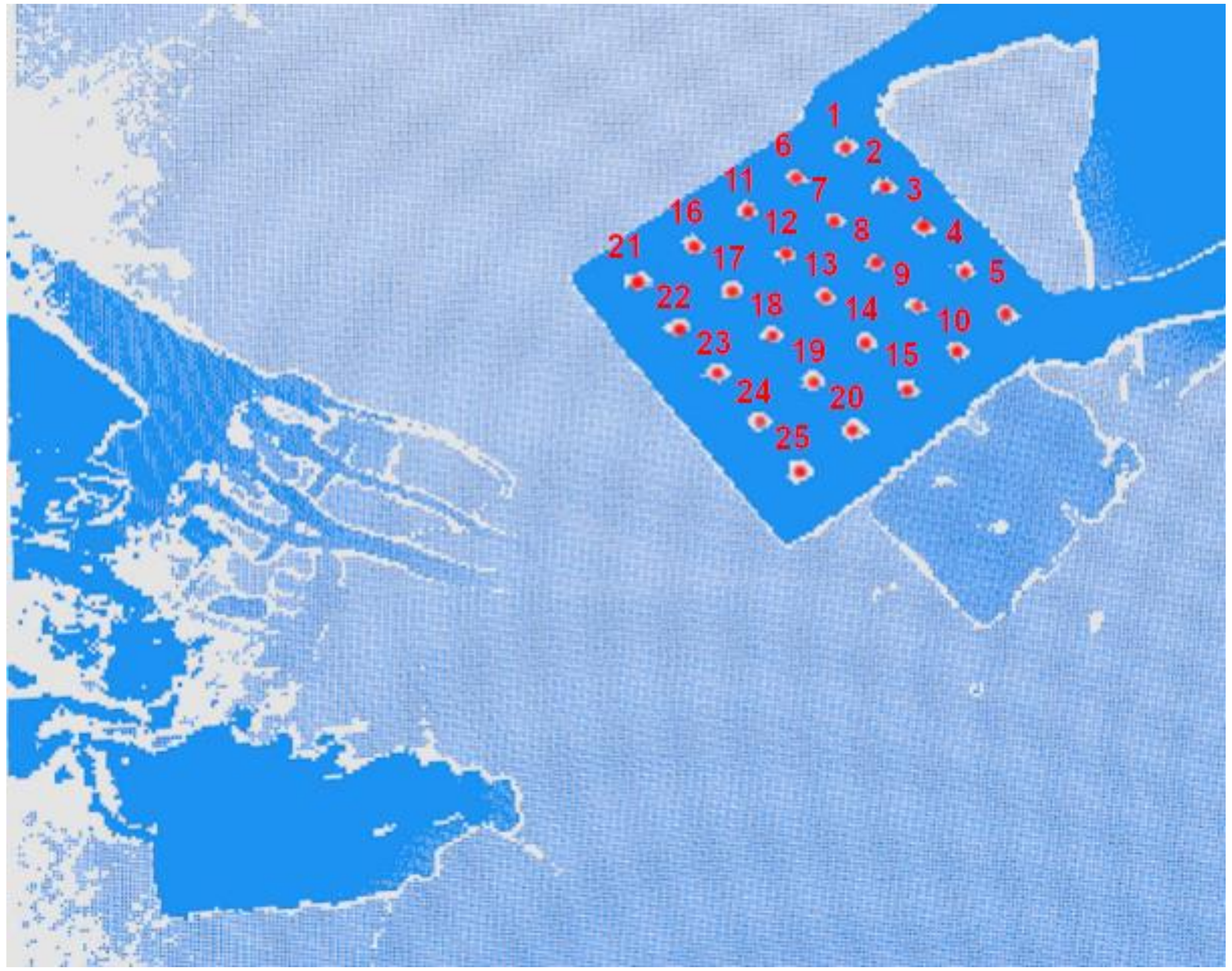
Hole centers & lattice pose



```
set<Point3f> holes;
Vector3f xAxis
Vector3f unsignedYAxis
Vector3f unsignedZAxis
Point3f center;
```

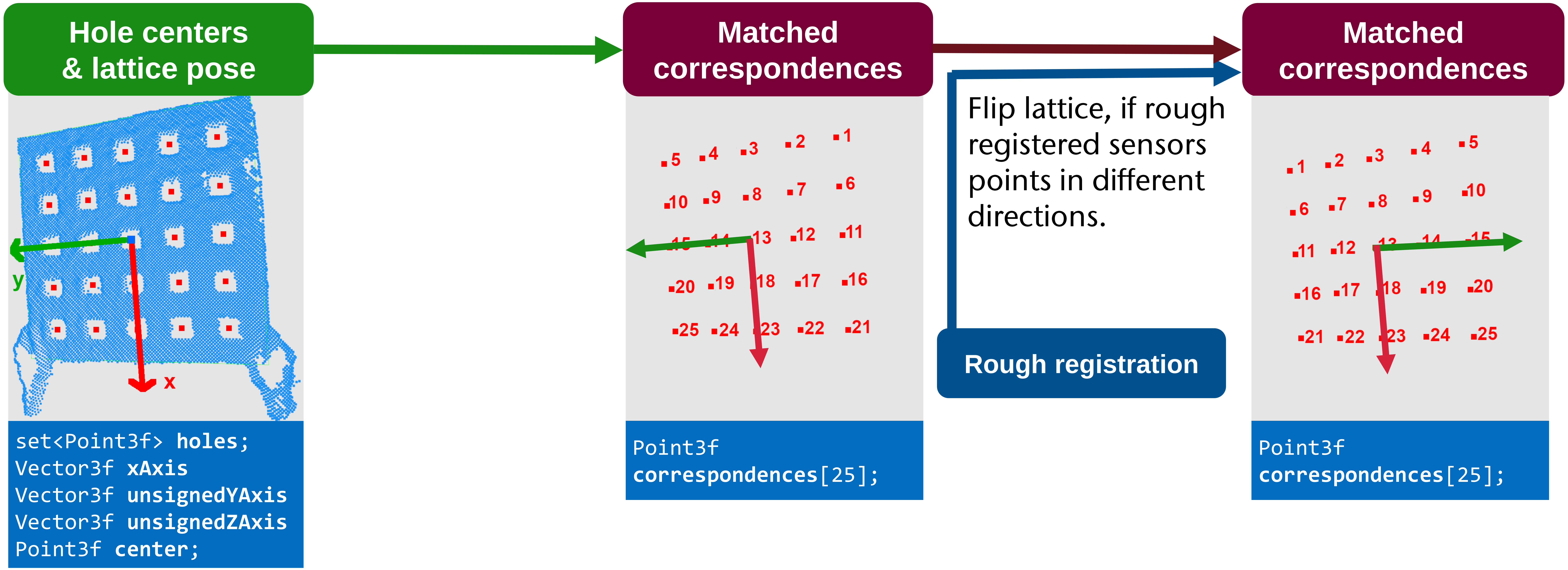
Algorithm Overview

1. Identify the region of the lattice
2. Feature point recognition and lattice pose estimation
3. **Unique hole center identification**
4. Extrinsic calibration



Algorithm Overview

3. Unique hole center identification



Hole centers & lattice pose

```
set<Point3f> holes;
Vector3f xAxis
Vector3f unsignedYAxis
Vector3f unsignedZAxis
Point3f center;
```

Matched correspondences

```
Point3f
correspondences[25];
```

Flip lattice, if rough registered sensors points in different directions.

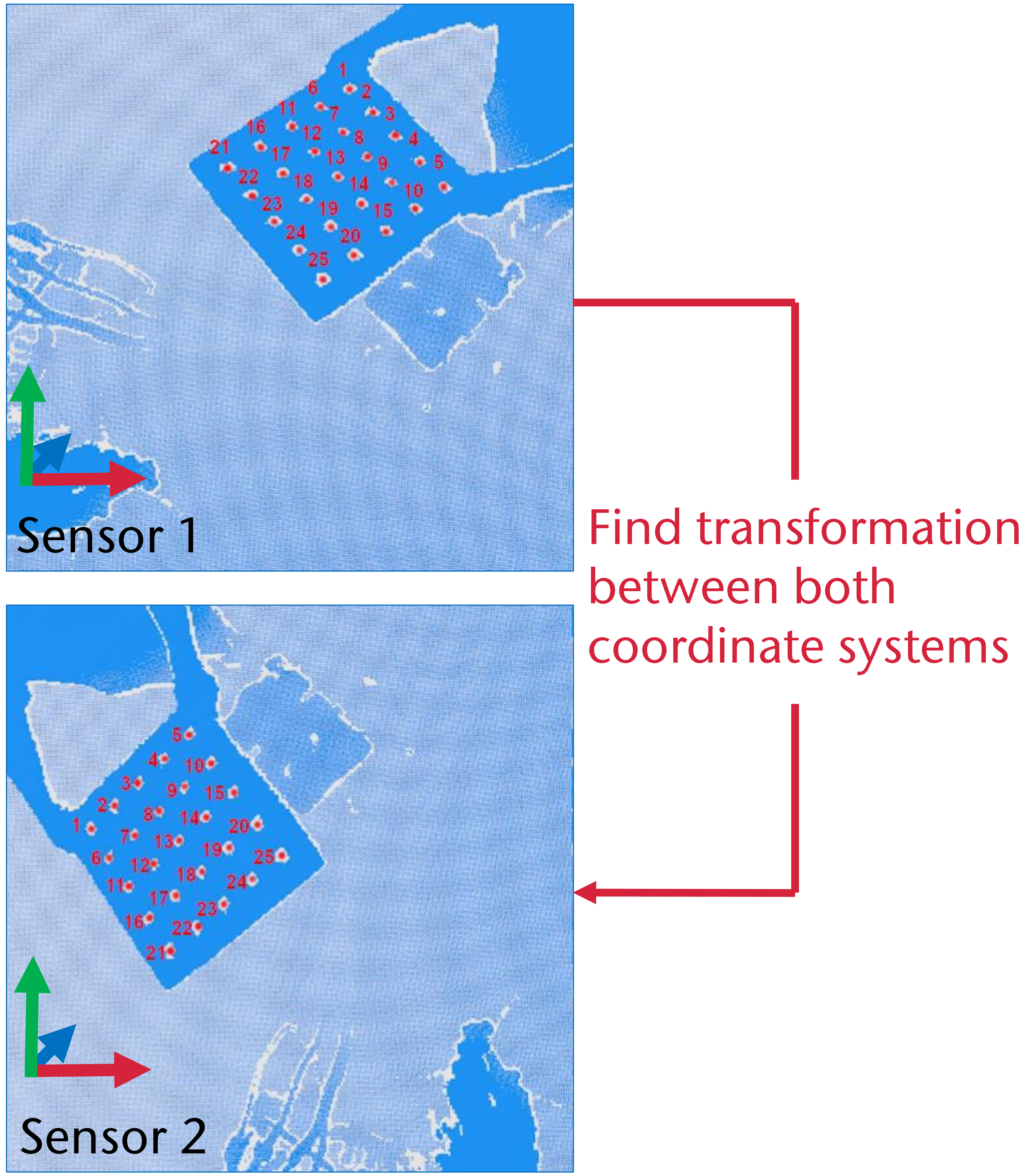
Rough registration

Matched correspondences

```
Point3f
correspondences[25];
```

Algorithm Overview

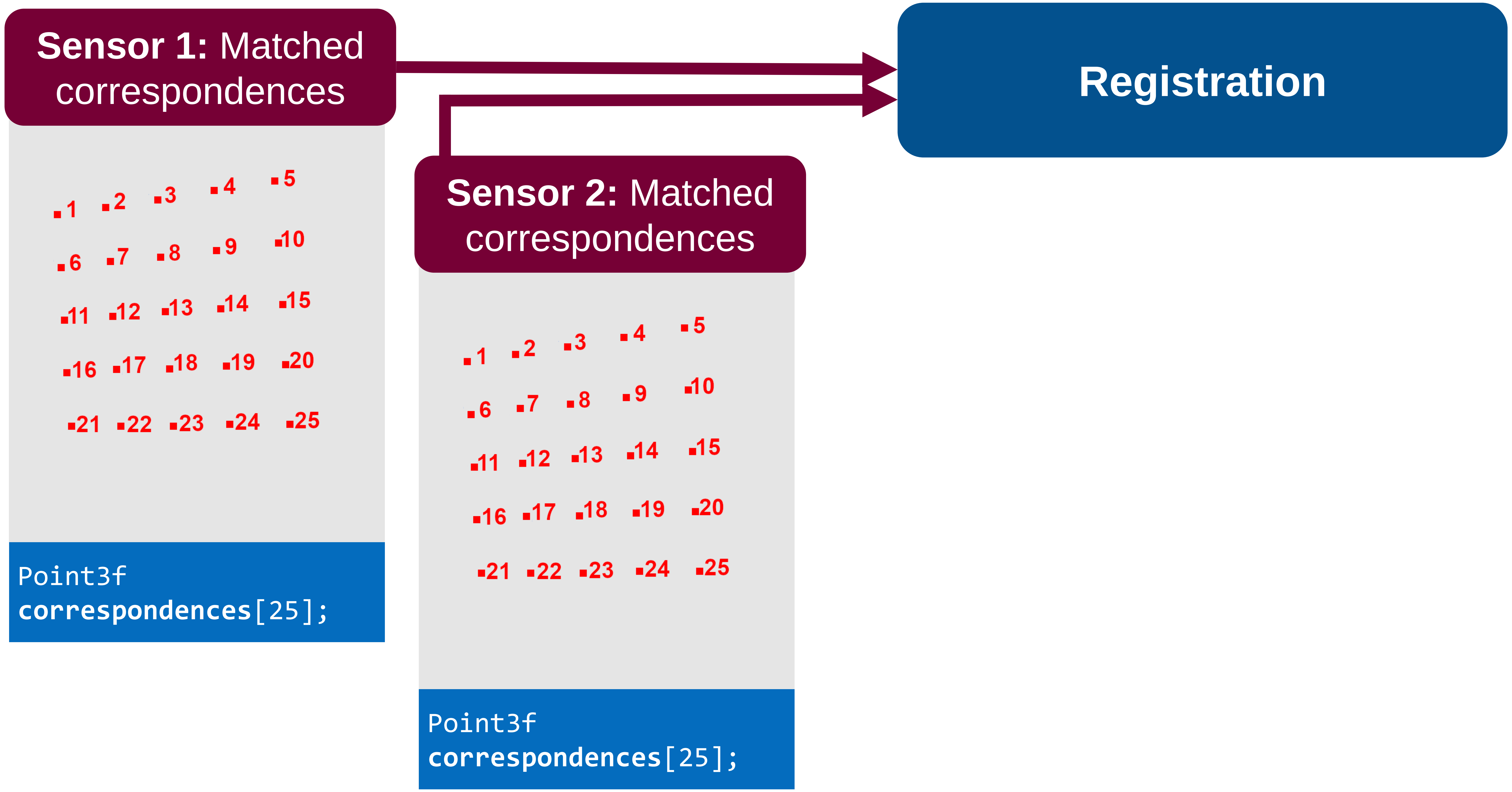
1. Identify the region of the lattice
2. Feature point recognition and lattice pose estimation
3. Unique hole center identification
4. **Extrinsic calibration**



Find transformation between both coordinate systems

Algorithm Overview

4. Extrinsic estimation



Sensor 1: Matched correspondences

```
Point3f
correspondences[25];
```

Sensor 2: Matched correspondences

```
Point3f
correspondences[25];
```

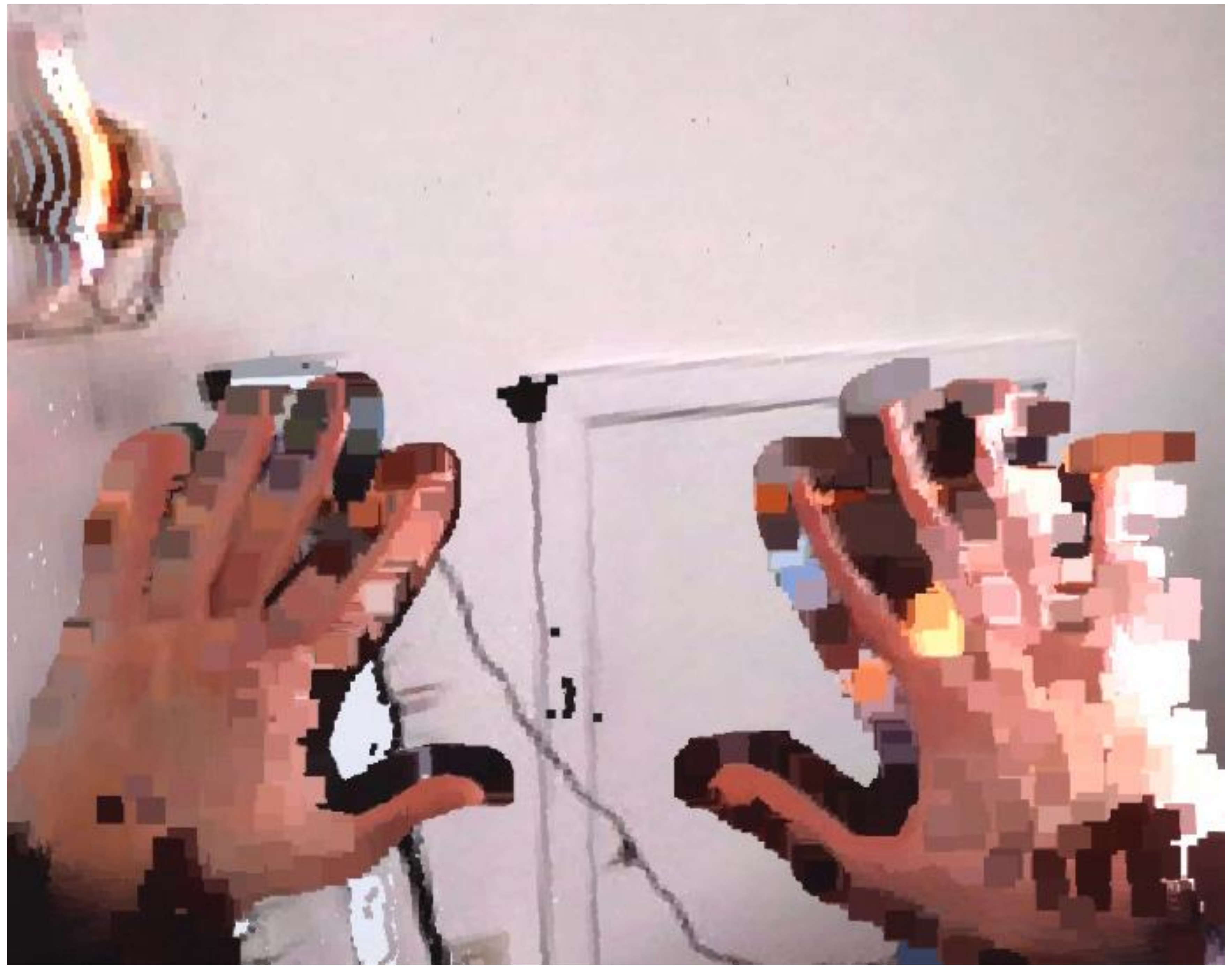
Registration

Video

Registered Pount Clouds

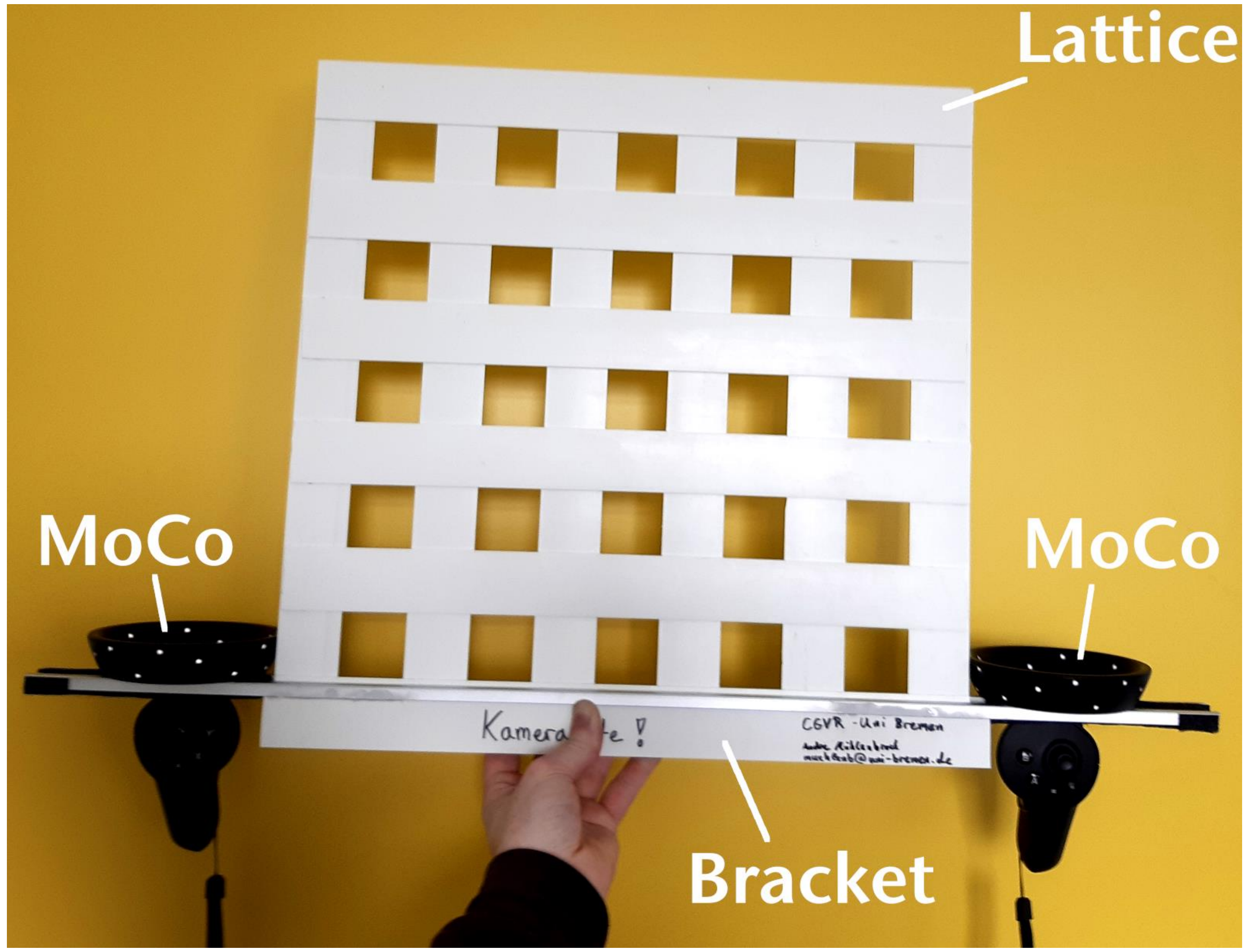
Virtual World Registration

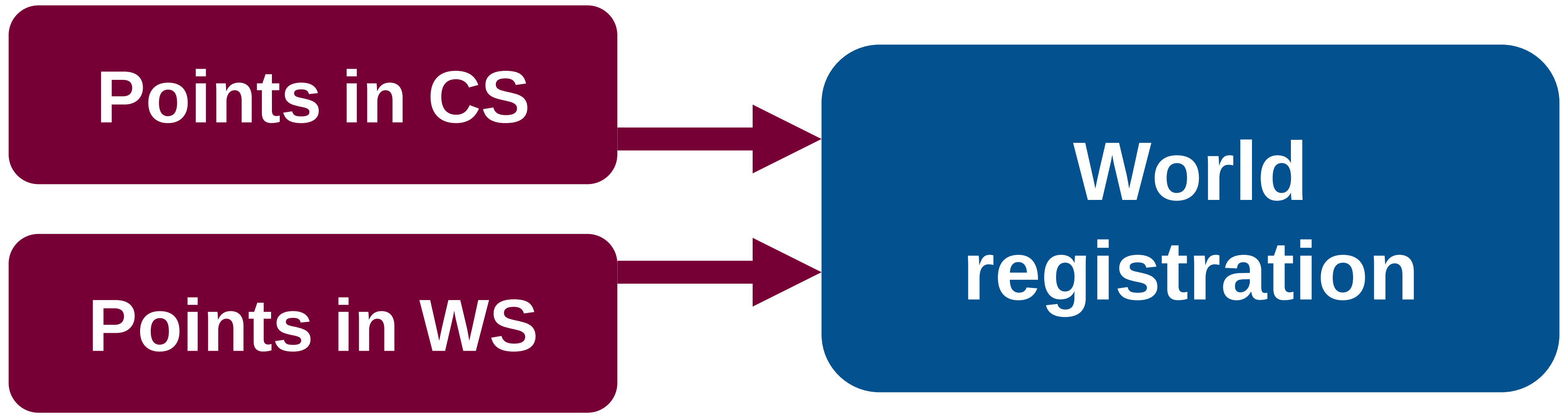
- Register point cloud into virtual world (for use with HMD)
  - Cast shadow by real geometry in VR.
  - Point cloud avatars.



Virtual World Registration

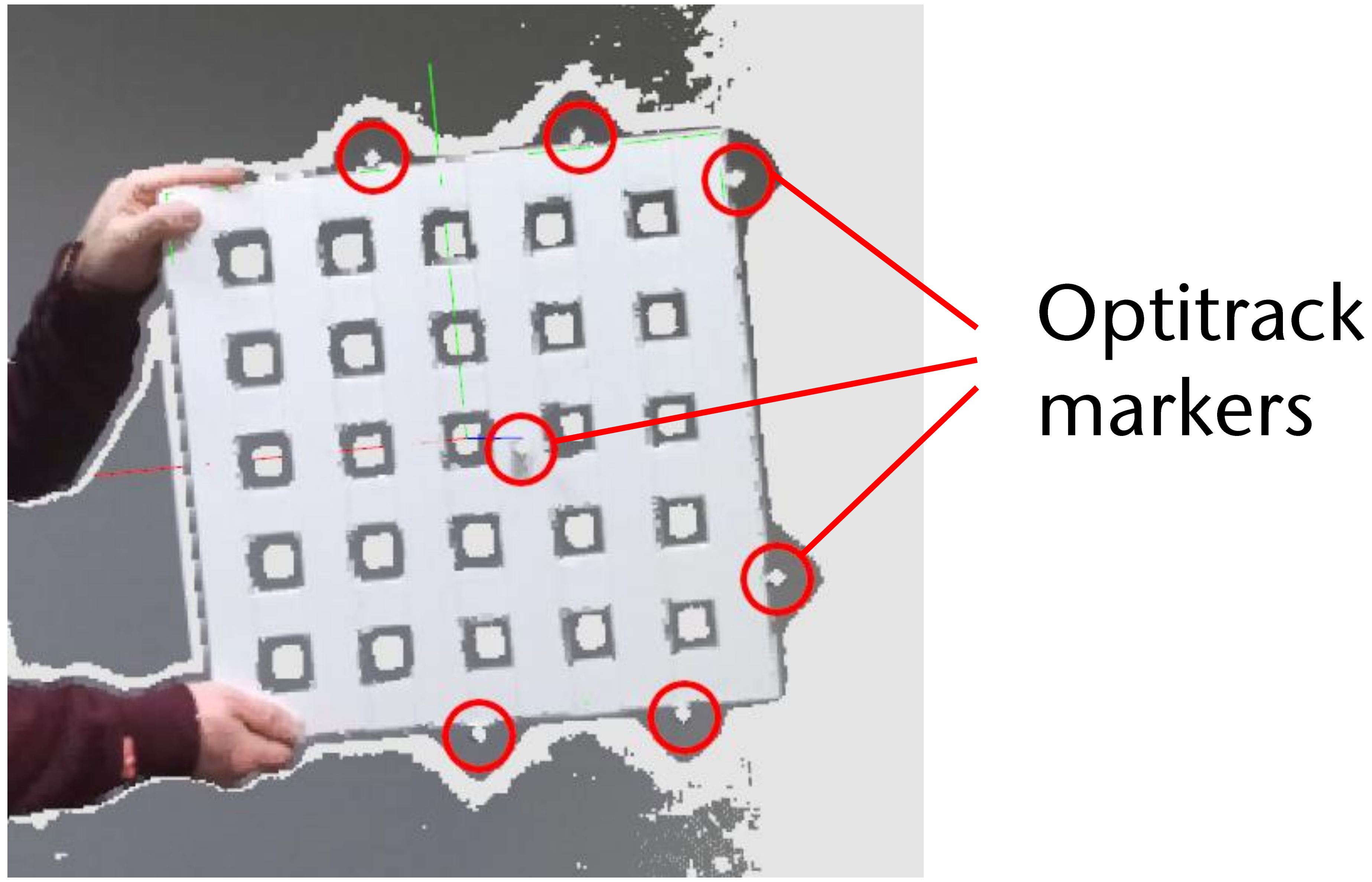
- Requires correspondence points both between World Space and Camera Space:
  - Use **M**otion **Co**ntrollers to estimate lattice pose in WS
  - Use lattice for **CS**





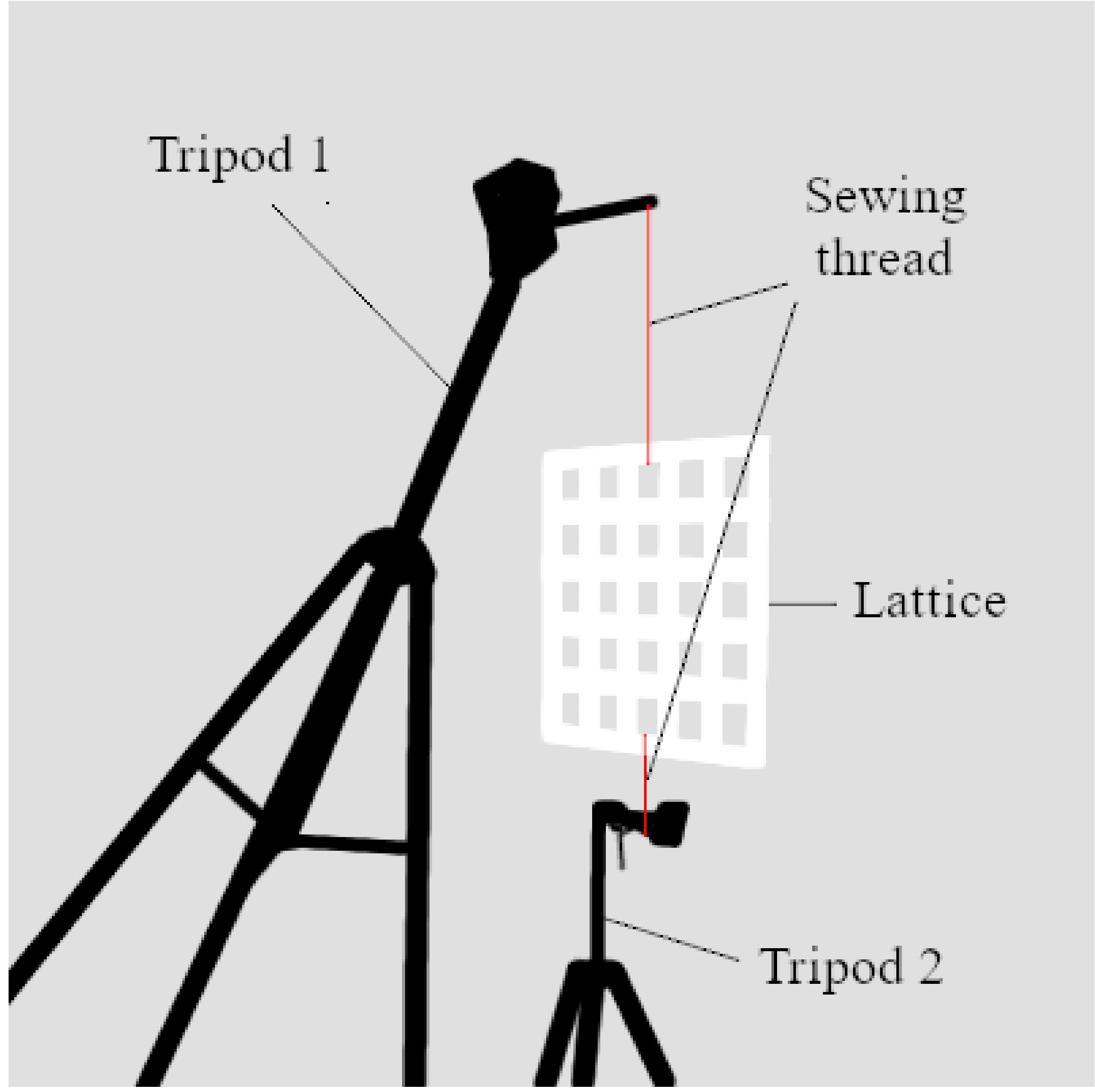
Results

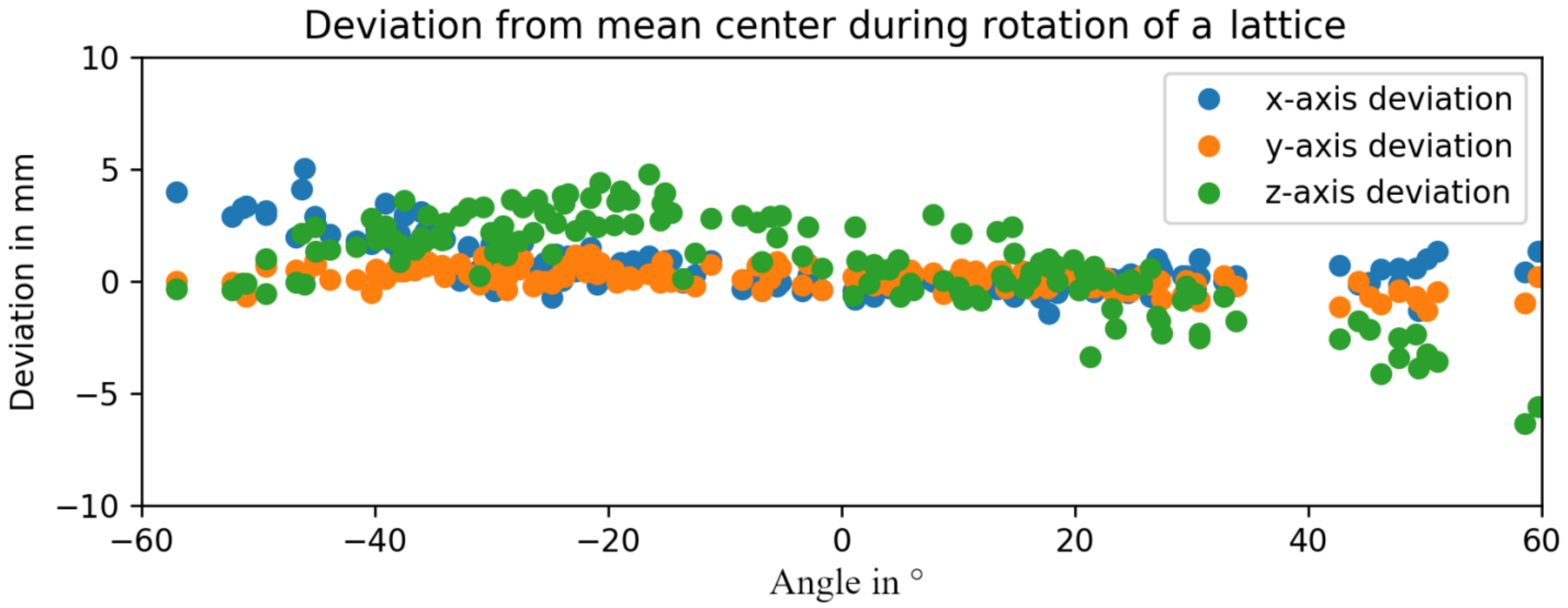
- Method:
  - **Azure Kinect** as depth sensor.
  - **Optitrack** for ground truth measure.
    - High precision optical tracking (accuracy of ≈0.7mm in our case)
  - Details: See paper.



Results

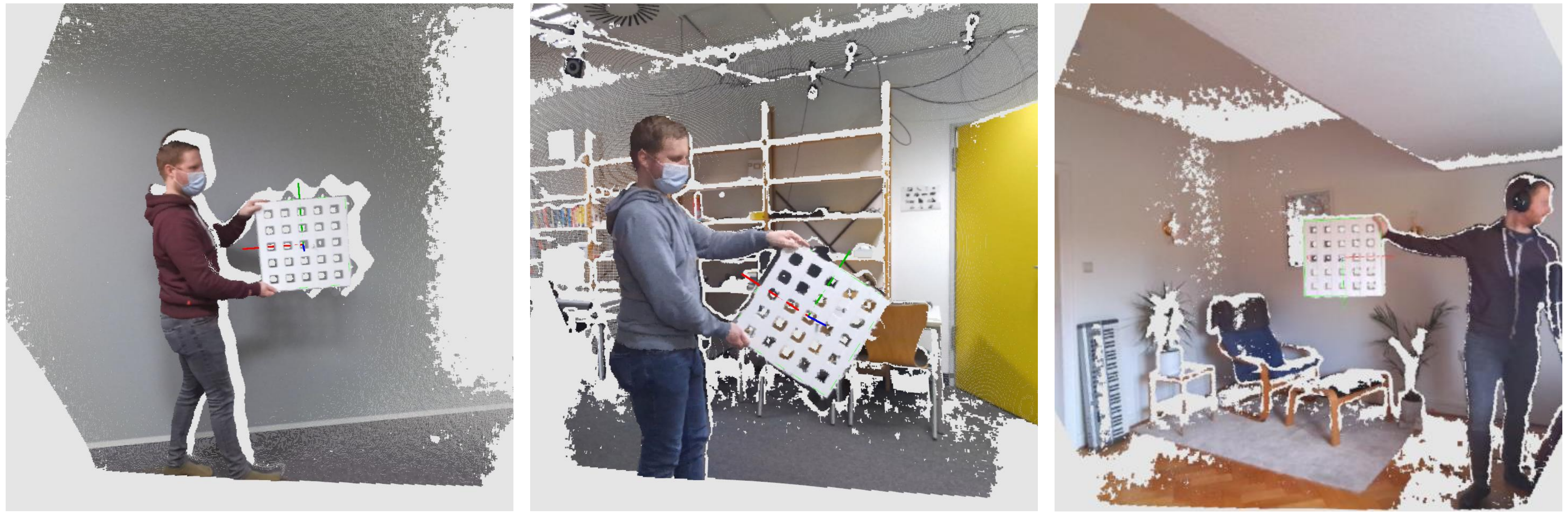
- Registration into Ground Truth **CS:**
  - Mean registration deviation about **3.8 mm** to **4.4 mm**
  - Below expected error: > **10 mm**
- Rotational stability:
  - Lattice was detected in a range of approx **[-55°, 55°]**

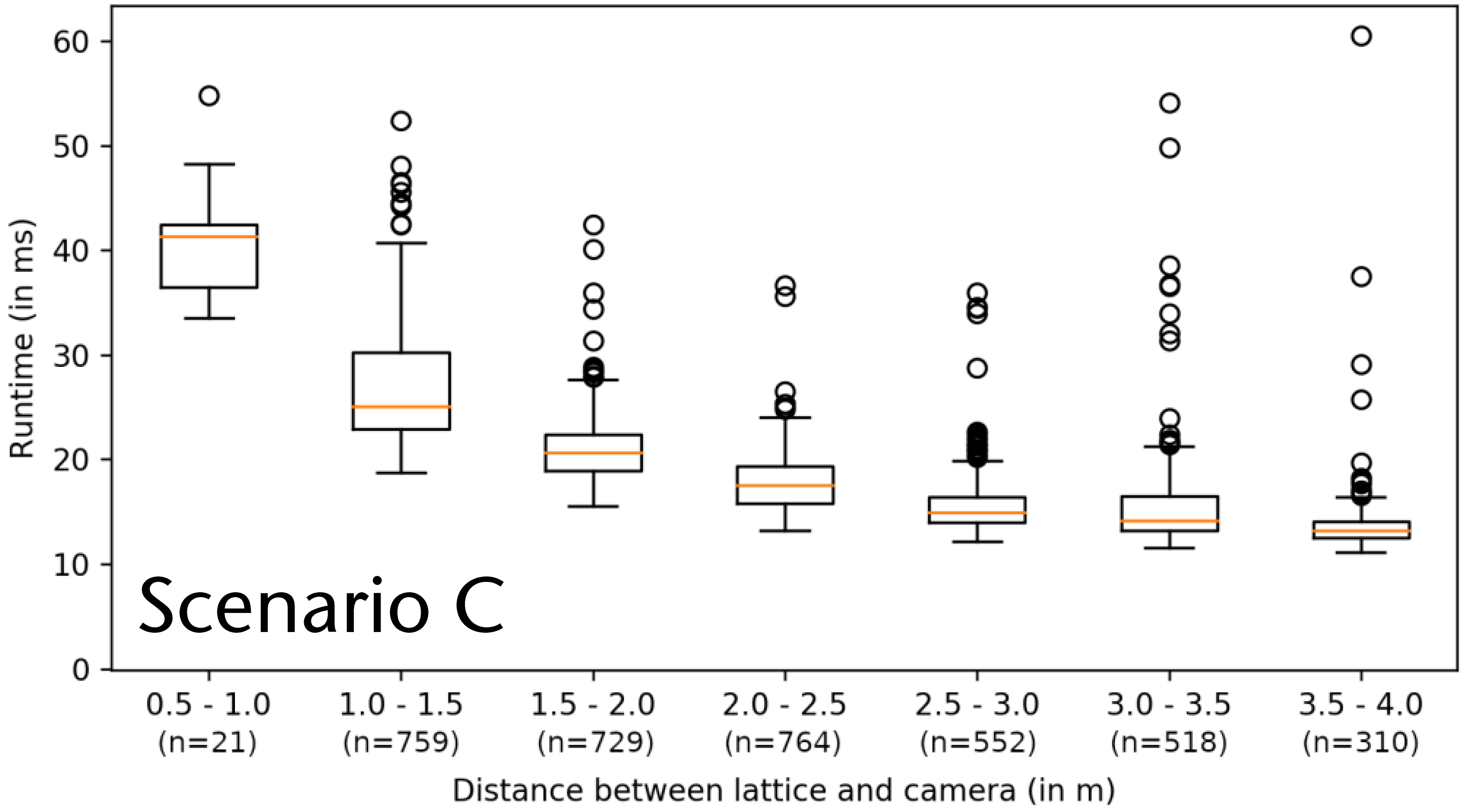




Results

- Robustness:
  - Precision: > **0.99**
- Runtime:
  - AVG: ≈ **21 ms** per sensor (Single core)
  - Depends on distance between **lattice** and **sensor**.
- See paper for more informations

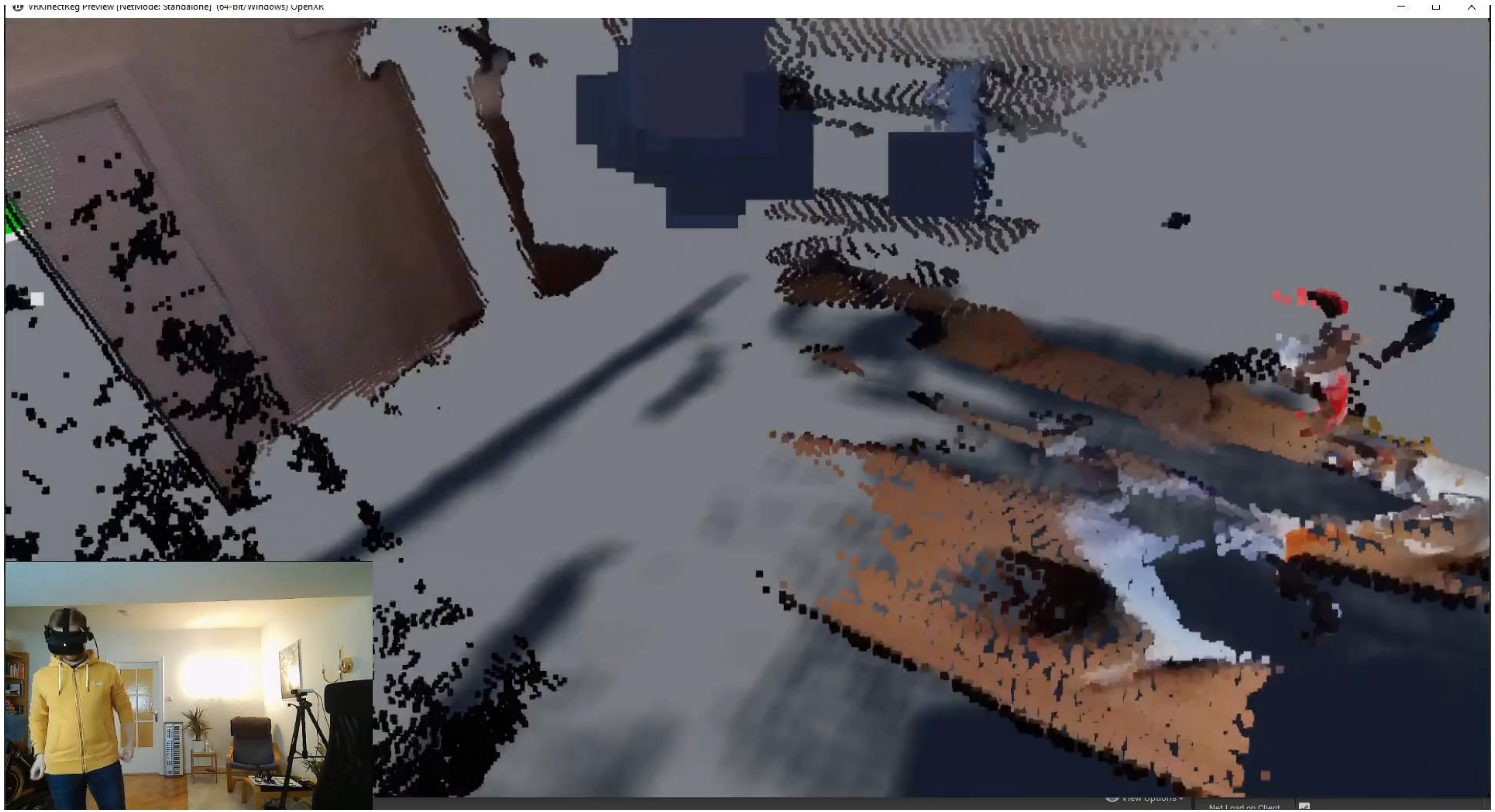




Conclusion

- Registration procedure requiring only depth images
- Fast (avg. **21 ms** per sensor)
- Very robust and precise (≈ **4 mm**)
- Further work:
  - Registration between RGB- and Depth-Sensor
  - Integrate with time synchronization [Beck et al. 2017]

Thank you for your attention!


VRKinectReg Preview [NetMode: Standalone] (64-bit/Windows) OpenXR
View Options
Net Load on Client