Addressing Challenges with Augmented Reality Applications on Smartphones

Krzysztof Zienkiewicz

krzysztof.k.zienkiewicz@vanderbilt.edu

Vanderbilt University
Nashville, Tennessee

Presented at Mobilware 2010

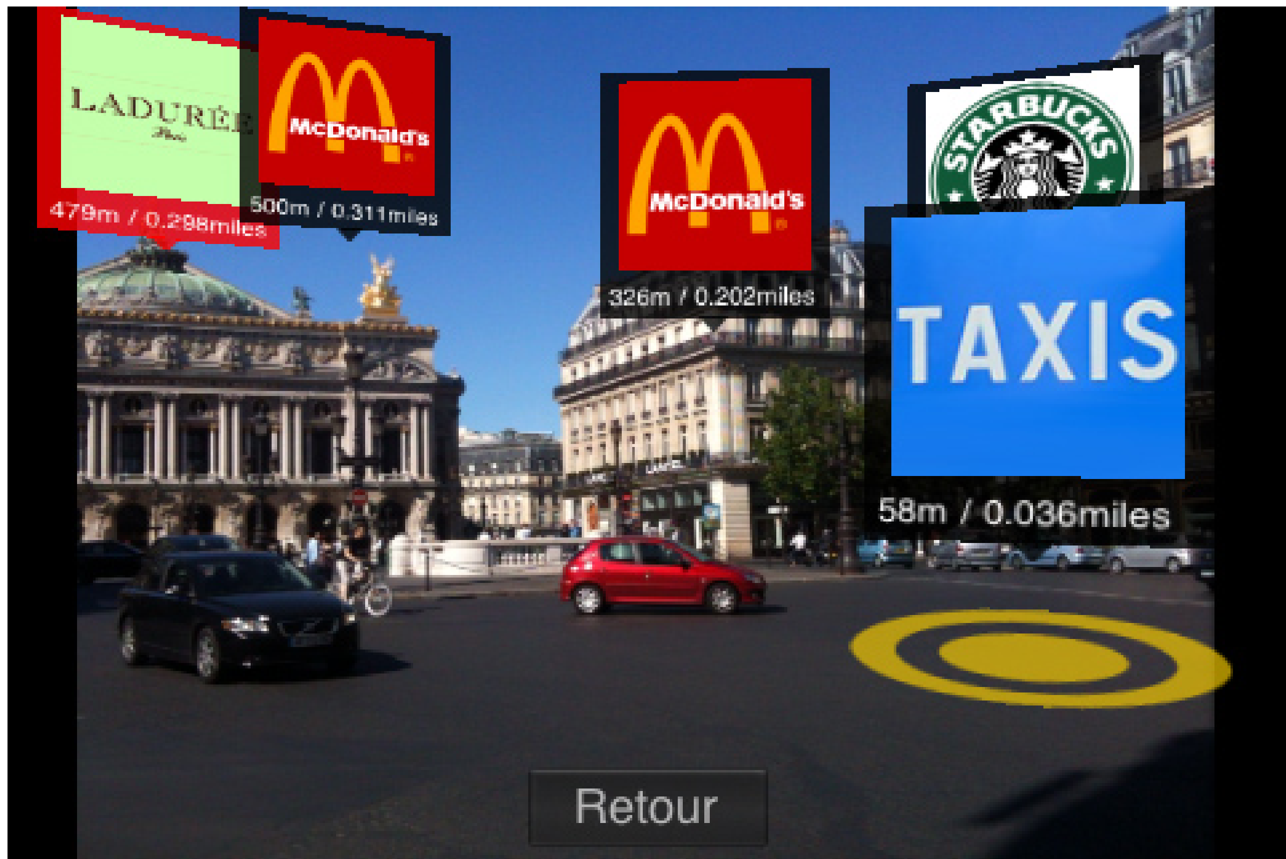
Augmented Reality (AR)

- Azuma's definition: AR is a system that has the following characteristics
  - Combines real and virtual
  - Interactive in real time
  - Registered in 3D

- Milgram's Continuum

Components of an AR application

- Rendering 3D context-specific data
- Retrieving geotagged Points Of Interest (POIs)
- Frame of Reference estimation



Challenges

- Mobile 3D solutions are not optimal and hard to mesh with camera imagery
- Filtering geotagged POIs by proximity is computationally intensive
- Real-time estimation of frame of reference is computationally demanding
- Geomagnetic sensor noise makes orientation estimation hard

Mobile 3D solutions are not optimal

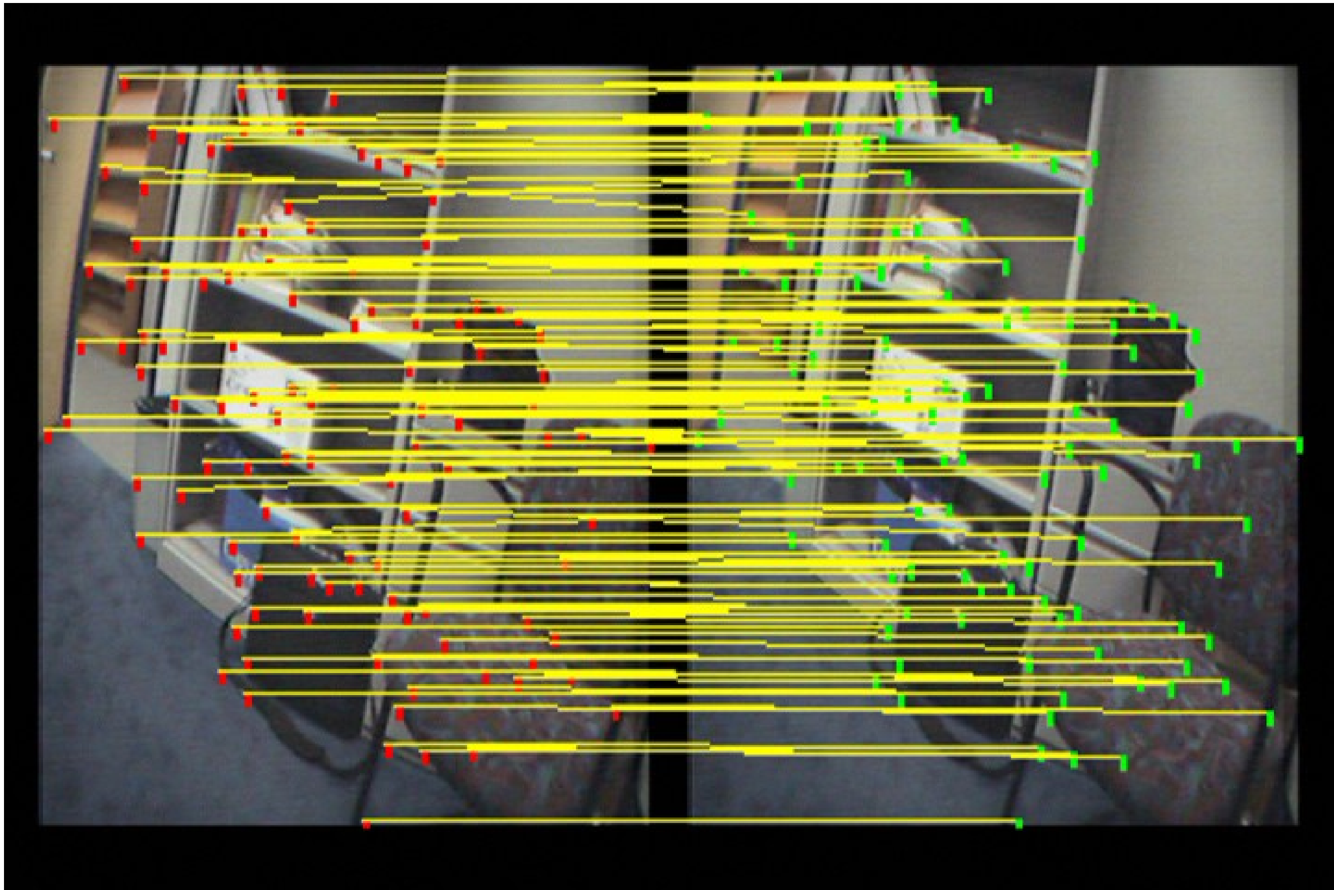
- AR needs precise 3D overlays to provide an intuitive interface
- Without accuracy, user experience deteriorates
- Meshing 3D data with camera preview is highly platform dependent
- Mobile GLs don't offer a "pick" mode

Retrieving POIs by proximity is expensive

- Areas of interest may have a high density of POIs
- Retrieval by location requires many comparisons
- Bandwidth limitations

Estimation of frame of reference

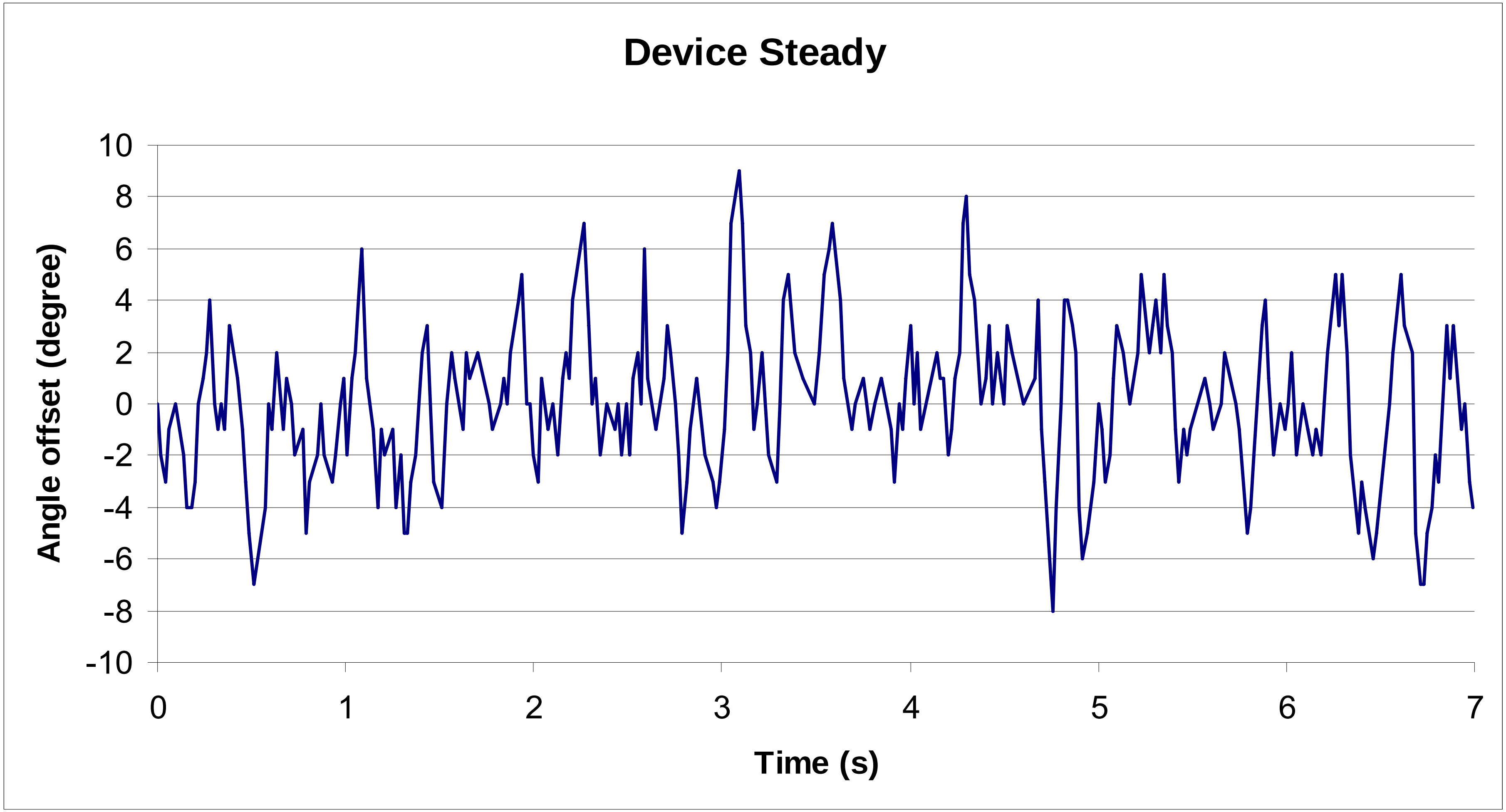
- Fiducial markers
- Feature extraction



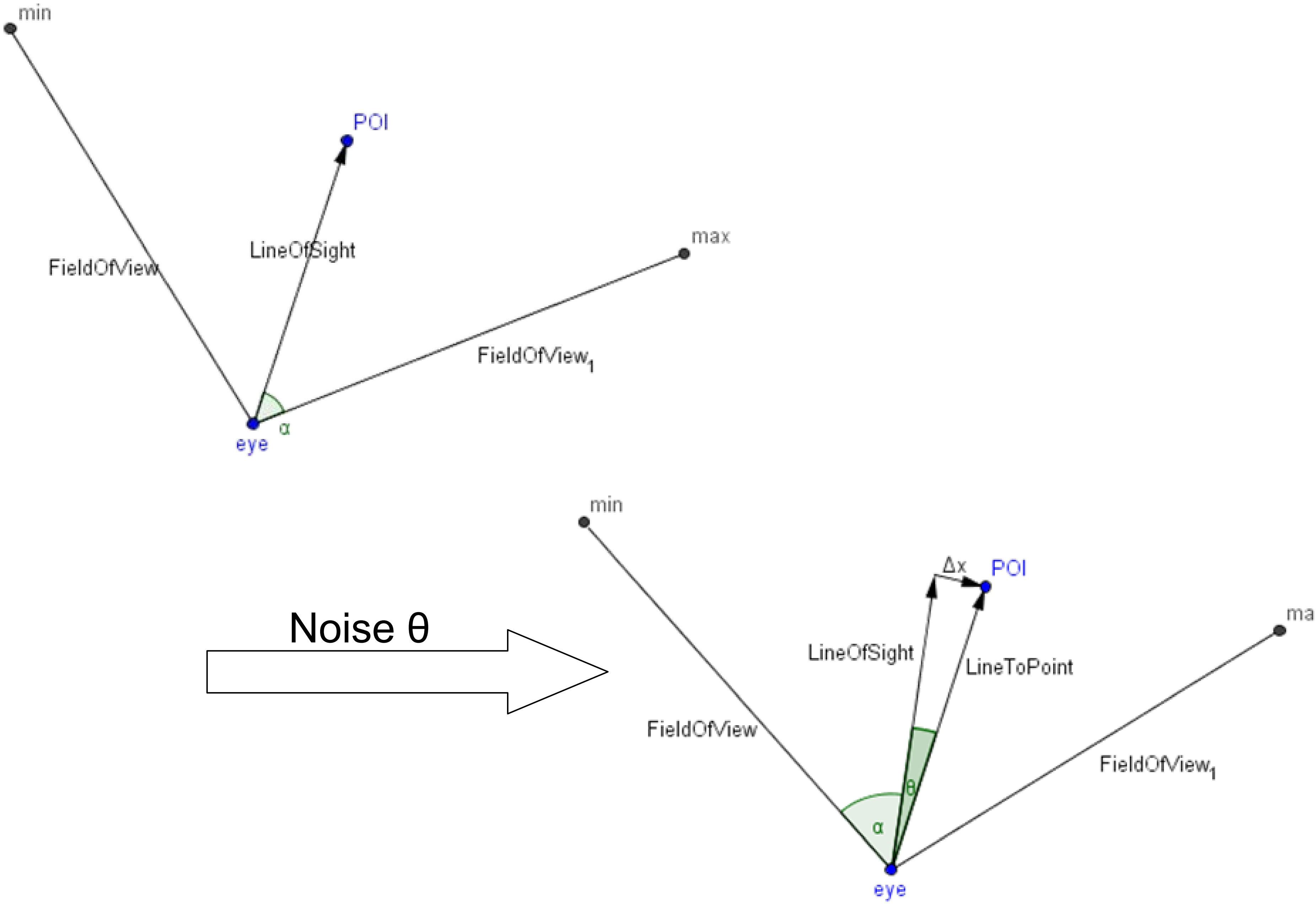
Geomagnetic sensor noise

- Compass used to estimate the field of vision
- Compass noise introduces jitter in the rendered overlays
- Savitzky-Golay smoothing filter is too slow
- Noise +/- 9 degrees

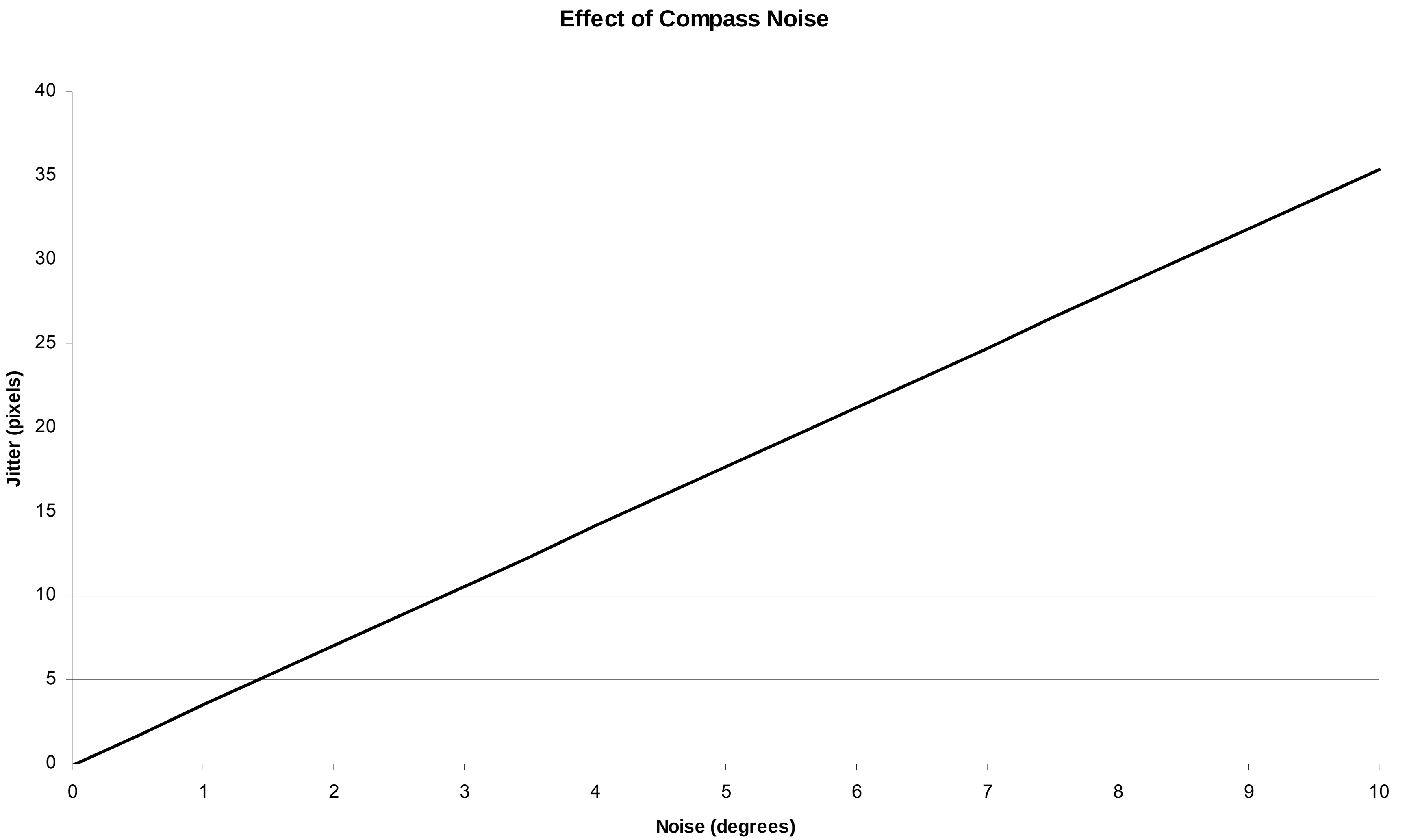
Geomagnetic sensor noise



Effects of geomagnetic sensor noise



Effects of geomagnetic sensor noise



$$jitter = \frac{width}{2\tan\alpha}\tan\theta$$

Solutions

- Using hardware accelerated 3D APIs to display rendered content
- A grid-based approach to data storage and retrieval
- Using GPS and geomagnetic sensors to estimate device position and orientation
- Statistical analysis filter

Hardware accelerated 3D APIs

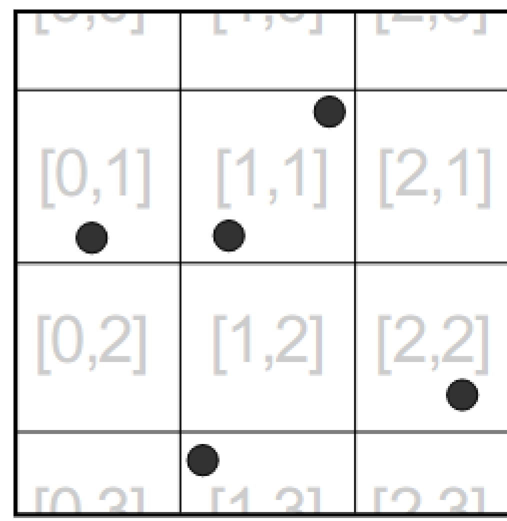
- Apple's UIKit allows a transformation matrix to be applied to views
  - Allows for hit testing
- Other platforms will need to do hit testing by hand
  - But get to use GLs for rendering
- Meshing achieved via layering

$$\begin{bmatrix} \frac{F}{aspect} & 0 & 0 & 0 \\ 0 & F & 0 & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2fn}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Grid-based data storage and retrieval

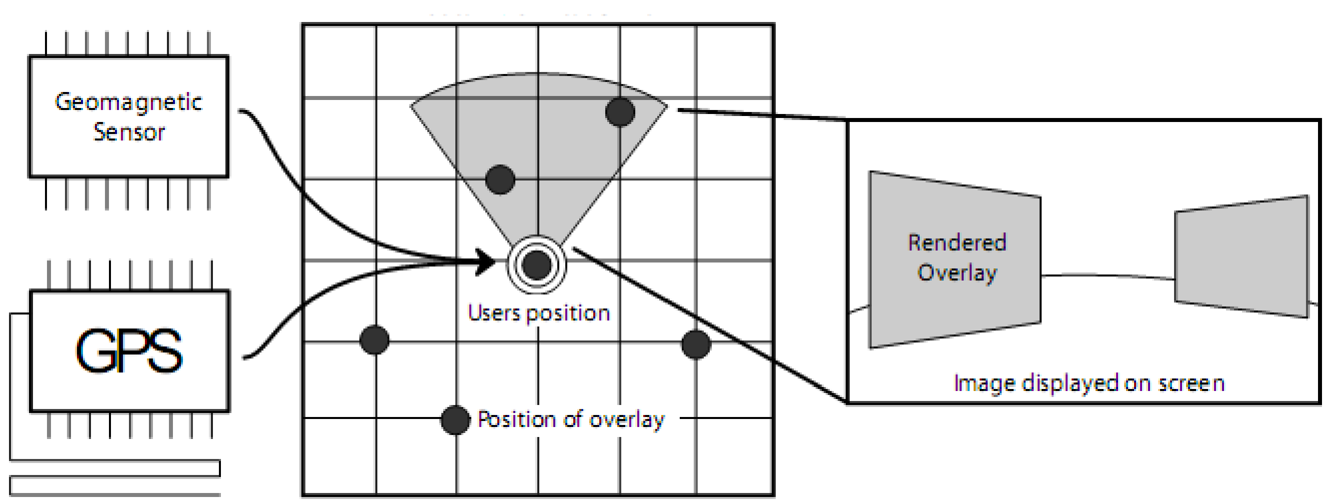
- Mapping function takes location to grid indices
- Each square contains all POIs within the region
- No numeric comparisons are performed by the server
- POIs are downloaded in bulk and need to be filtered on the phone

Query Type	Response Time
Latitude range (latitude column indexed)	581700 µs
Latitude range (latitude column not indexed)	284600 µs
Specific latitude bucket	5209 µs

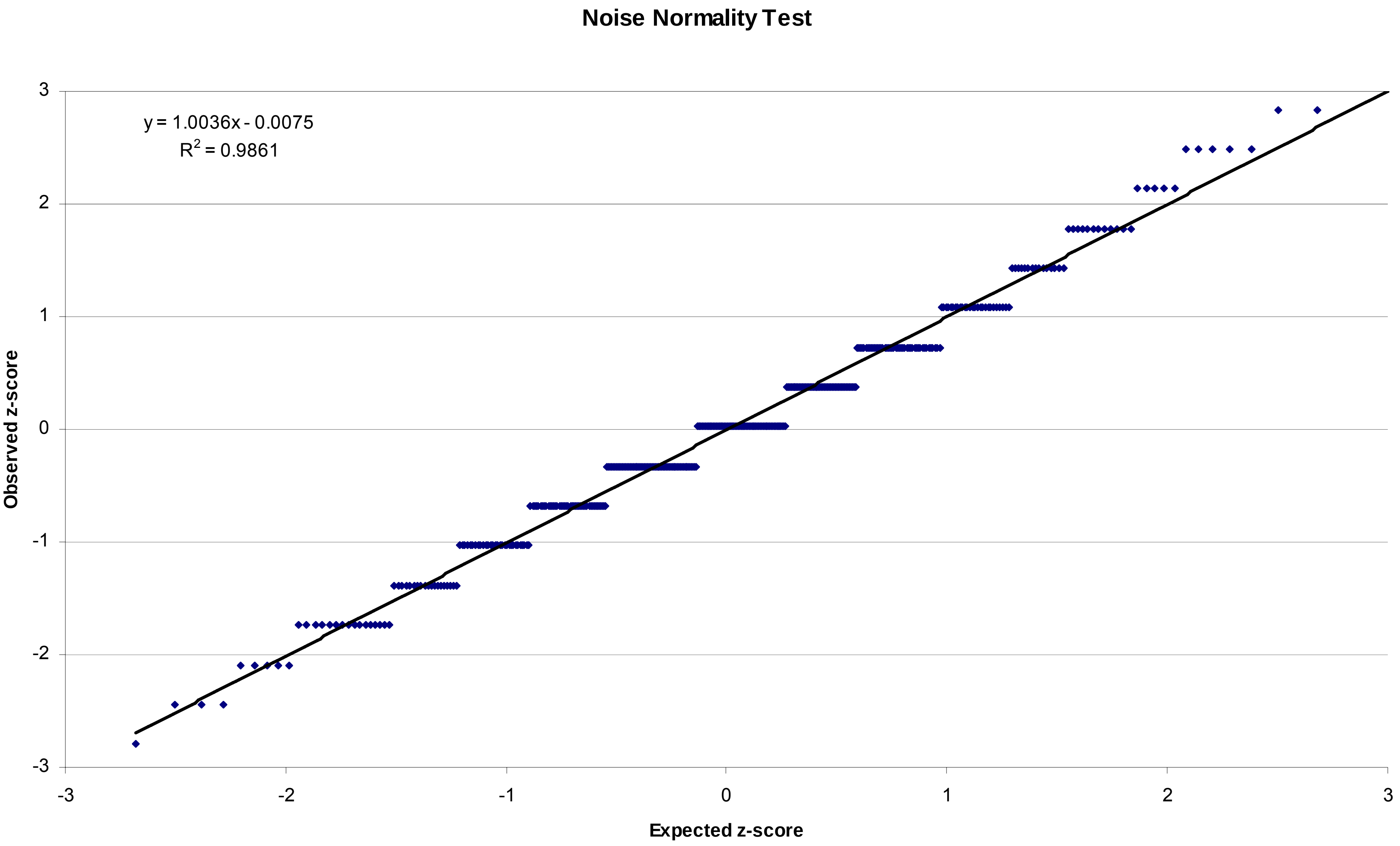


Using sensors to estimate field of view

- The computationally cheap solution
- Only need to update the transformation matrix



Nature of the noise



Filtering algorithm

Variables/Functions:

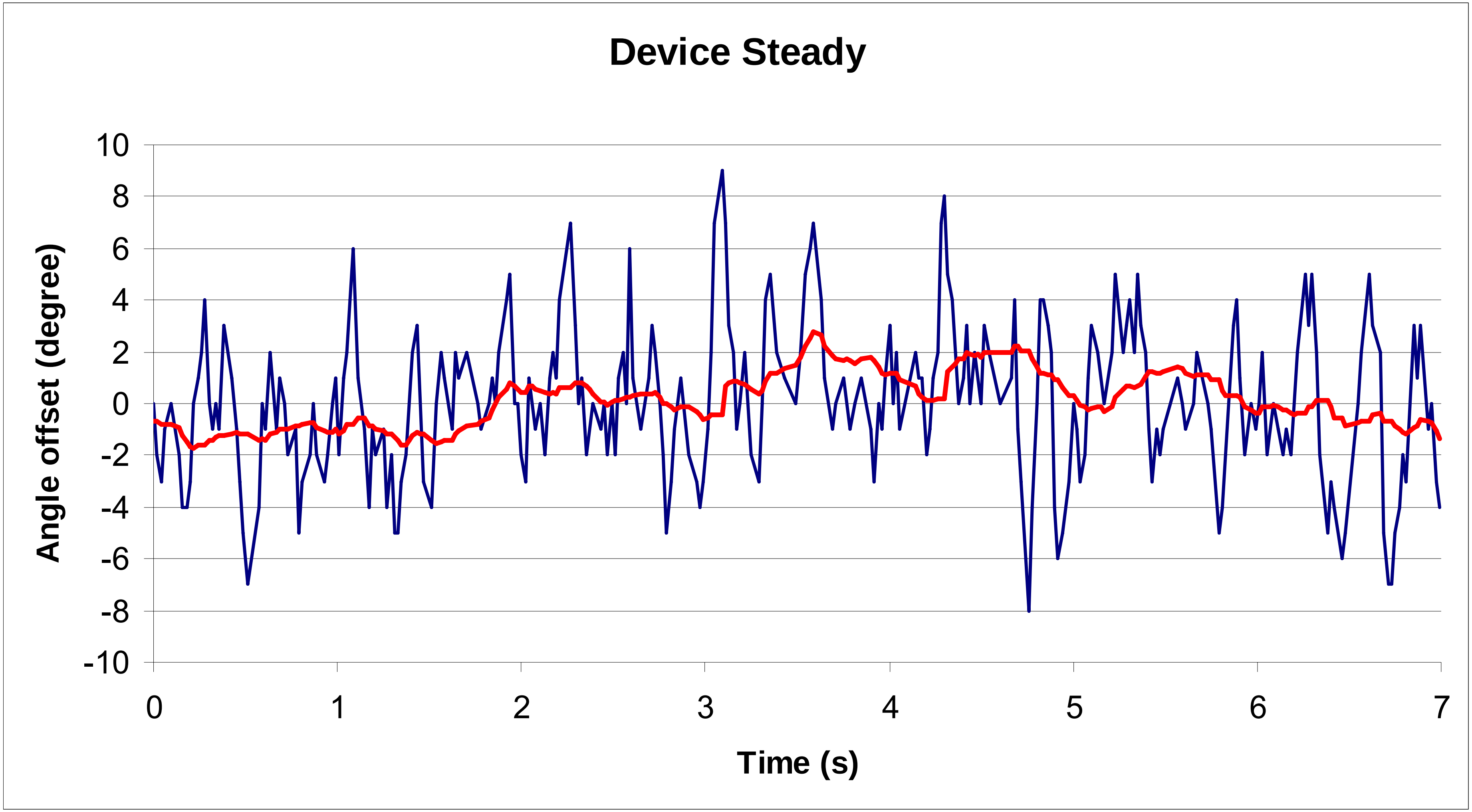
$$
\begin{aligned}
R &= Ring\ Buffer\ of\ Received\ Data \\
O &= Ring\ Buffer\ of\ Outlier\ Data \\
|R| &= |O| = Maximum\ Allowable\ Size\ of\ Buffer \\
size(buffer) &= ReturnsCurrentSizeofBuffer \\
p_i &= A\ compass\ reading\ as\ a\ Single\ Precision\ Float \\
Z(p_i) &= (p_i - mean(R))/stdDev(R) \\
Z_{range} &= Maximum\ Allowable\ Deviation \\
outlierDirection(p_i) &= pi > mean(R)?1 : -1 \\
enqueue(buffer, p_i) &= Adds\ p_i\ to\ the\ Buffer
\end{aligned}
$$

Algorithm:

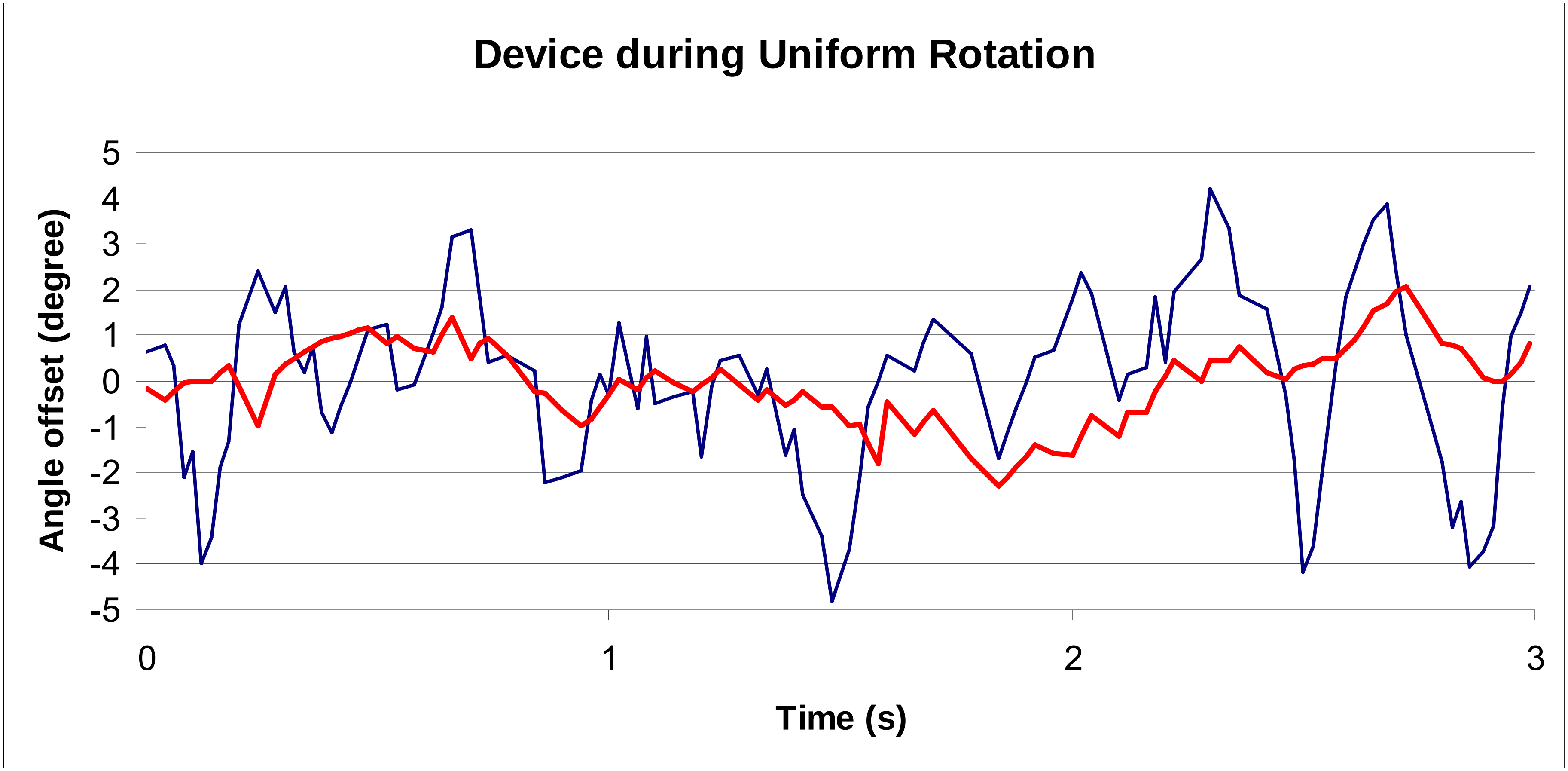
```
filtered(p_i) =
        if  size(R) < |R|: enqueue(R, p_i)
        else:
            z_i = Z(p_i)
            if  abs(z_i) ≤ Z_range:
                enqueue(R, p_i)
                clear(O)
            else: enqueue(O, p_i)
        if size(O) = |O| :
            side = outlierCluster()
            ∀p_j ∈ O
                if  outlierDirection(p_j) = side:
                    enqueue(R, p_j)
                    clear(O)
        return mean(R)
outlierCluster() =
        int sum = 0
        ∀p_j ∈ O
            sum+ = p_j − mean(R)
        return signum(sum)
```

- Small number of parameters
- Extendable for dynamic parameterization
- Good results: 60% noise reduction

Filtering algorithm



Filtering algorithm



Thanks

Questions?