CVPR2012 Providence, RI, USA

PCL :: Object Detection – LINEMOD

Stefan Holzer, TU Munich (TUM)
Stefan Hinterstoißer, TU Munich (TUM)

June 17, 2012

pointcloudlibrary

Overview

Goal for today: Object Detection using PCL

- Introduction to Template-Matching : ***LINEMOD***

  Multimodal Templates for Real-Time Detection of Texture-less Objects in Heavily Clutered Scenes, S. Hinterstoißer, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, V. Lepetit, ICCV 2011

- How to: Learn Objects using PCL

- How to: Detect Objects using PCL

Motivation

Aims for Texture-less Objects:

Daily objects often do not show much Texture.

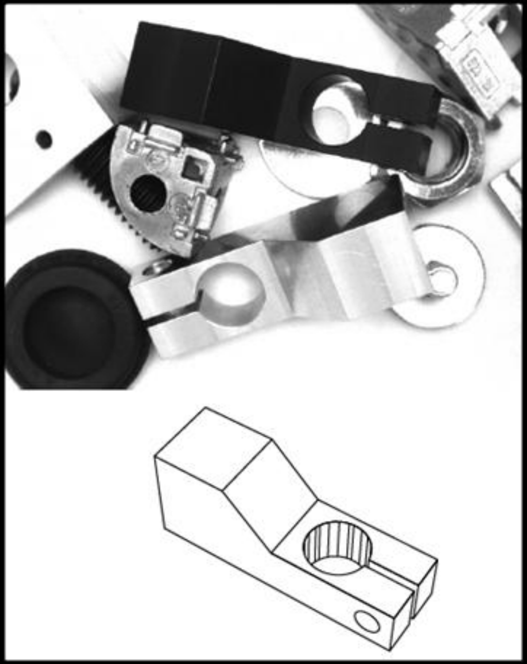
[Courtesy of M. Bollini and D. Rus, MIT]

Some more examples...

Motivation

Aims for Texture-less Objects:

Objects used in Industry are often Texture-less.



[Courtesy of M. Ulrich, C. Wiedemann and C. Steger, ICRA 2011]

Some more examples…

Motivation

Texture-less objects:

- **Lack of Interest Points** – prevents search space reduction influencing ***efficiency***
- **Lack of Texture** – makes it difficult to build discriminative descriptors influencing ***robustness/reliability***

Motivation

Aims for Cluttered Scenes:

Industry

Household

Conventions

pointcloudlibrary

Motivation

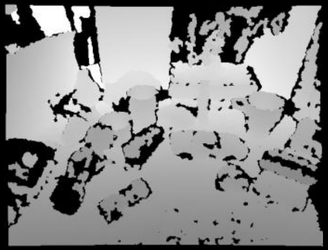
Clutter causes:

- **Many false positives** - require post-processing
- **Need for exhaustive search** in the full image
- **Contamination of grid-like descriptors** because of changing background

object
background

pointcloudlibrary

LINEMOD



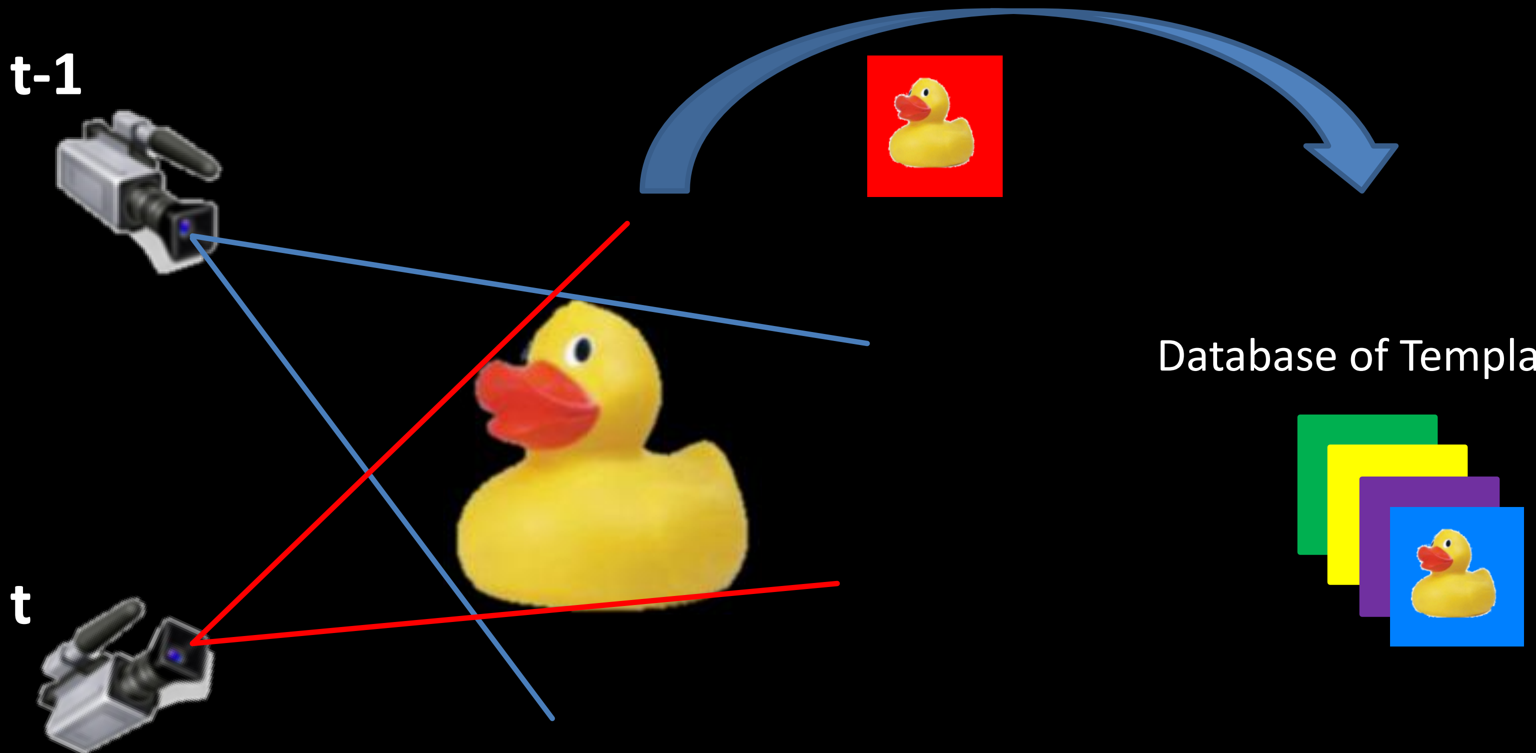
Using Multimodal Templates

- Combining **color** and **depth** information
  → Improves detection of Texture-less Objects
  → Improved handling of Cluttered Background

Efficient Implementation enables Real-Time Performance

- Quantizing and spreading the feature values
- Precomputing response maps
- Linearizing the memory

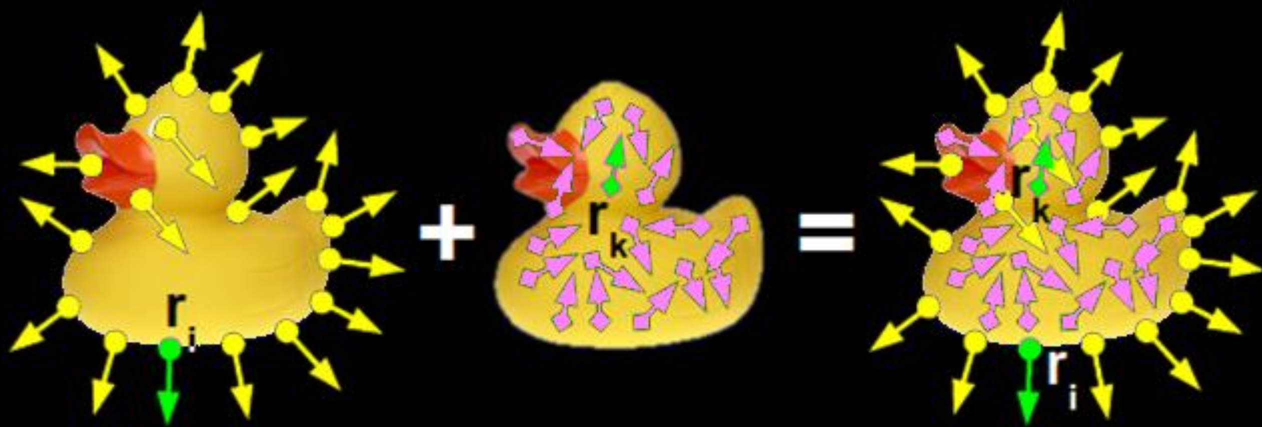
Training Stage



- Learning Objects simply means adding Templates to the Database.
  - ➡ few milliseconds per template
- View-point dependent templates keep information about their approximate pose

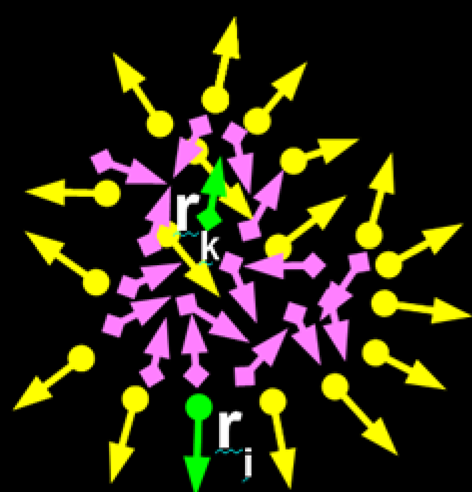
Modalities

Color Gradients and Surface Normals are Complementary!



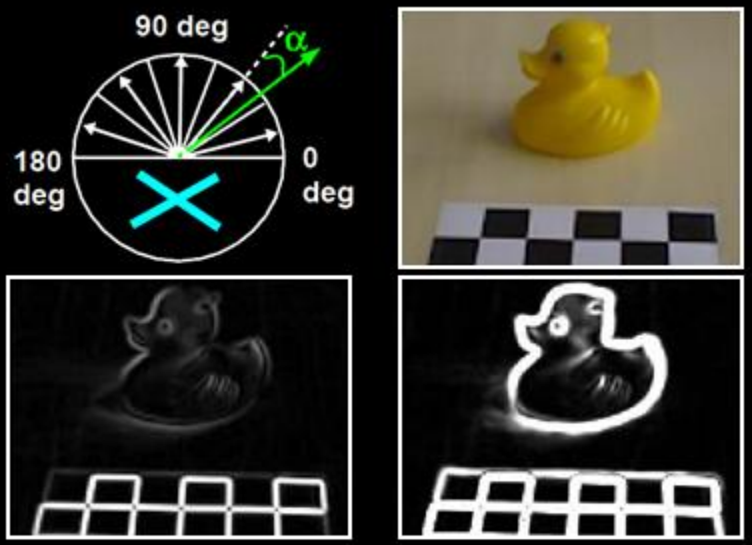
2D Color Gradients 3D Surface Normals Multiple Modalities

Template is not constrained by the use of a regular grid!



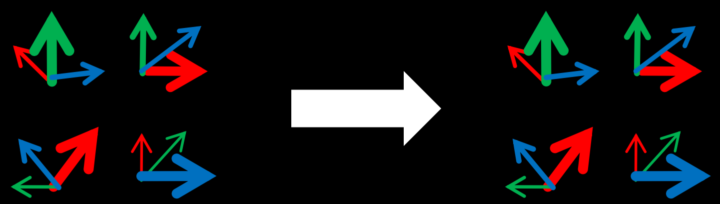
Modalities

2D Color Gradients



(Gray Value Gradients) **(Color Gradients)**

Computation:



[Dalal et al. CVPR05]

Similarity Measure:

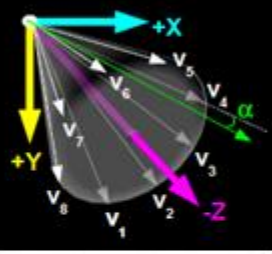
$$f_{\mathcal{G}}(\mathcal{O}_{\mathcal{G}}(r), \mathcal{I}_{\mathcal{G}}(t)) = \left| \mathcal{O}_{\mathcal{G}}(r)^{\top} \mathcal{I}_{\mathcal{G}}(t) \right|$$

$\mathcal{O}_{\mathcal{G}}(r)$: Color gradient on the template

$\mathcal{I}_{\mathcal{G}}(t)$: Color gradient on the input image

Modalities

3D Surface Normals



(Depth Image)

(Normal Image)

Similarity Measure:

$$f_{\mathcal{D}}(\mathcal{O}_{\mathcal{D}}(r), \mathcal{I}_{\mathcal{D}}(t)) = \mathcal{O}_{\mathcal{D}}(r)^{\top} \mathcal{I}_{\mathcal{D}}(t)$$

$\mathcal{O}_{\mathcal{D}}(r)$: Surface Normal on the template

$\mathcal{I}_{\mathcal{D}}(t)$: Surface Normal in the input image

pointcloudlibrary

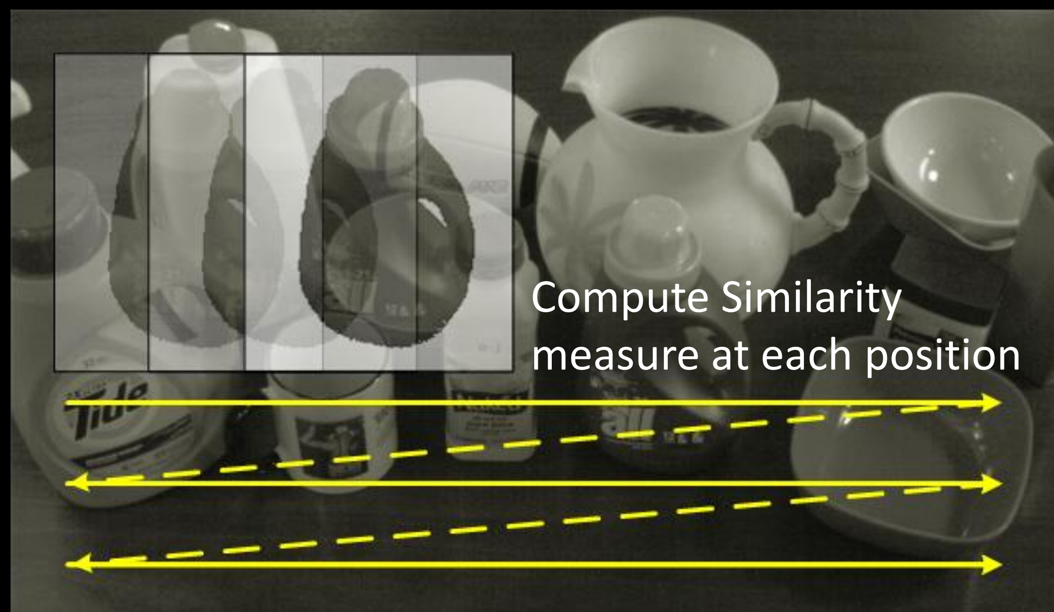
Detection Stage

We can’t use Interest Points for efficient Object Localization!

Therefore, we have to use a Sliding Window Approach:

Template

Current Scene



Naïve Sliding Window is inefficient: how can we make it fast?

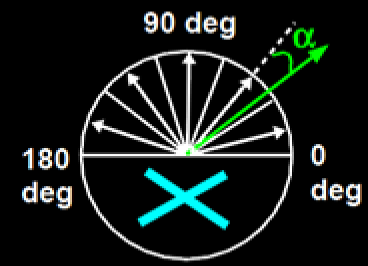
Detection Stage

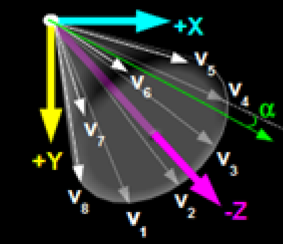
Efficient Implementation of our Similarity Measure:

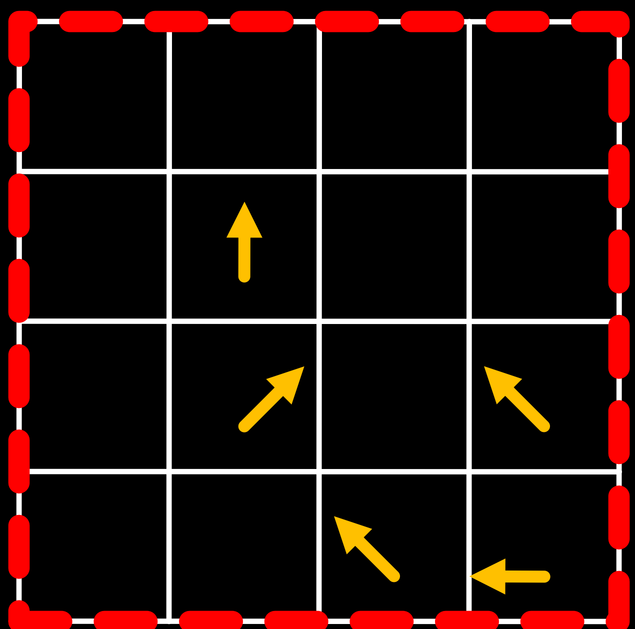
1. **Spreading the features**
2. Precompute Response Maps
3. Use Look-Up Tables
4. Linearize the Memory

Spreading Features

We first ***quantize*** the features and ***spread*** them around their initial position.







Data of every modality is quantized in 8 bins, e. g.:

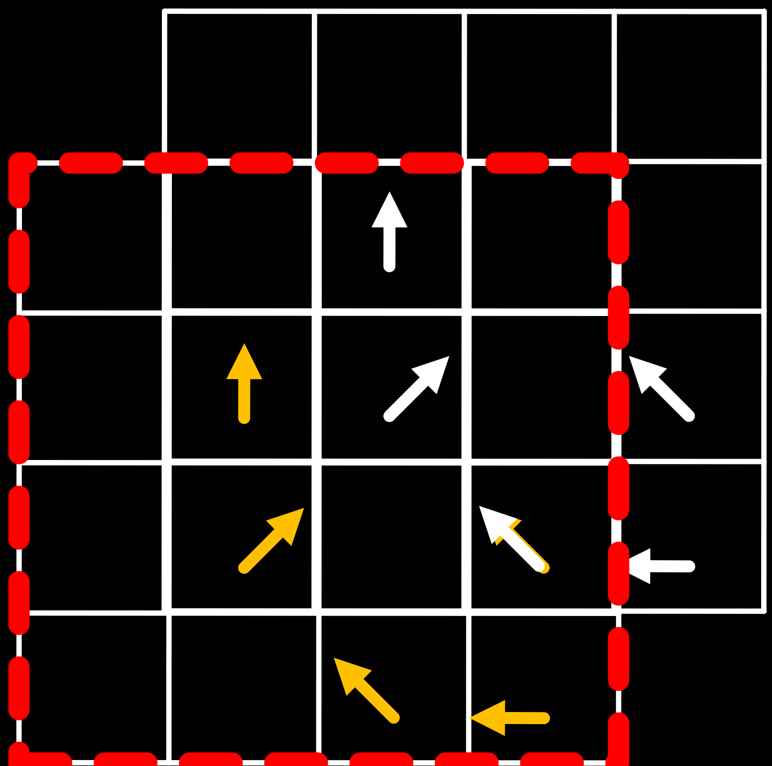
↑ = 00 00 00 01

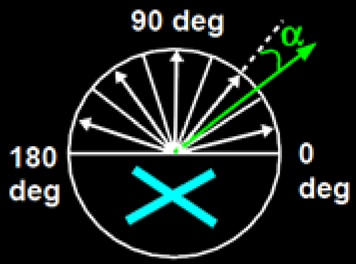
↗ = 00 00 00 10

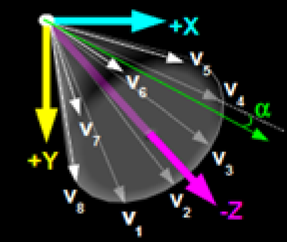
→ = 00 00 01 00

Spreading Features

We first ***quantize*** the features and ***spread*** them around their initial position.

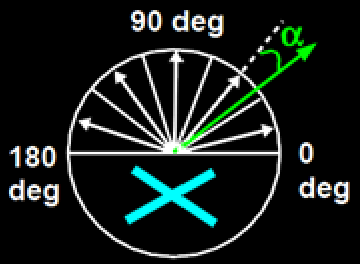


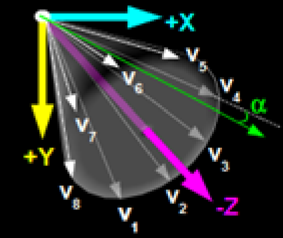


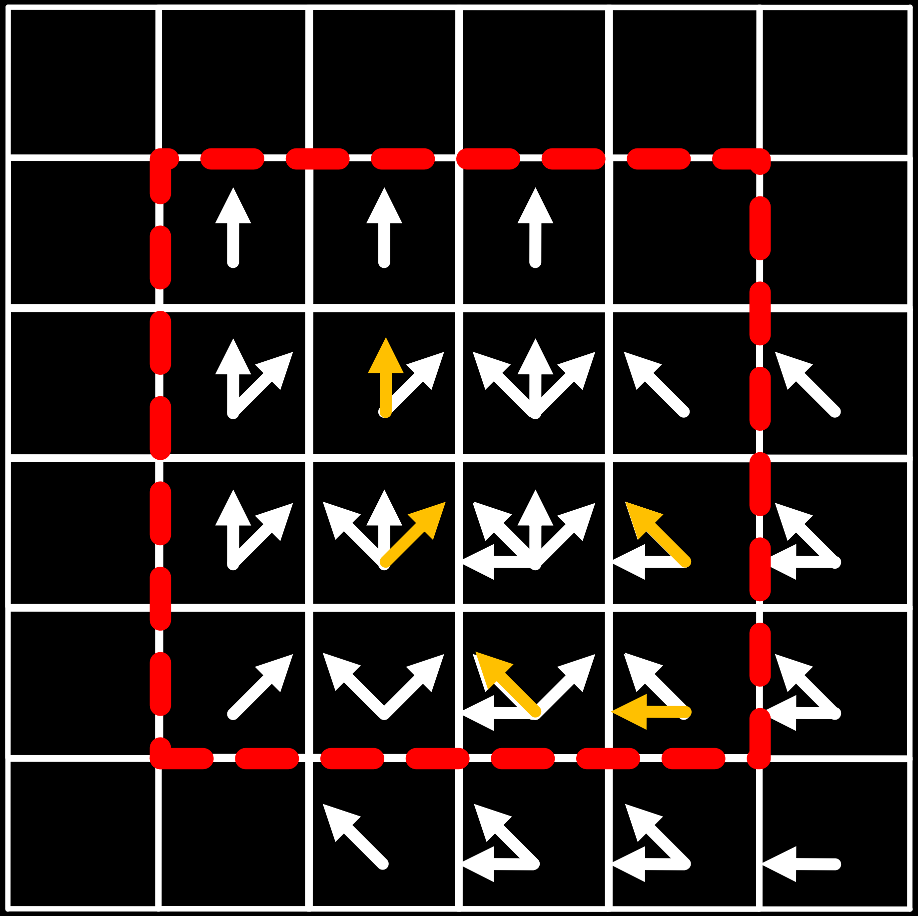


Spreading Features

We first ***quantize*** the features and ***spread*** them around their initial position.

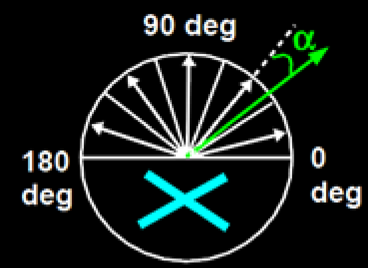


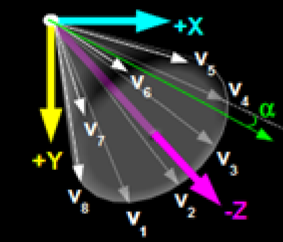


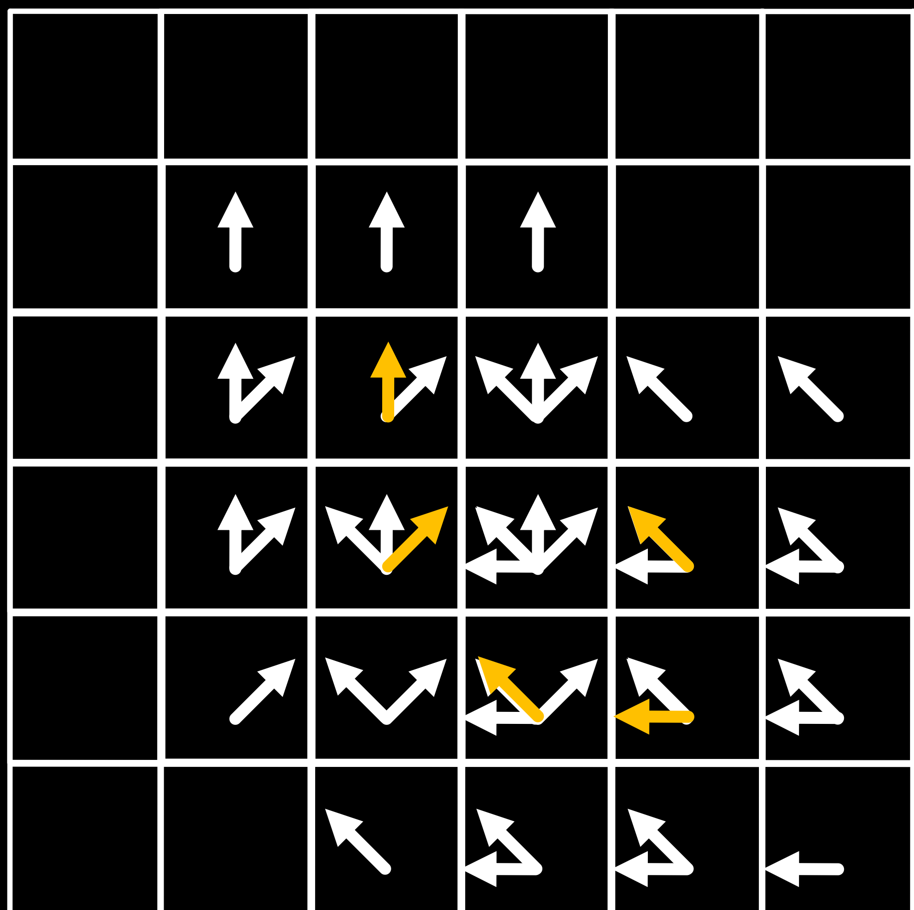


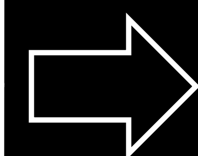
Spreading Features

We first ***quantize*** the features and ***spread*** them around their initial position.









Binarized Image

00100	00100	00100	00000
00110	00110	01110	01000
00110	01110	11110	11000
00010	01010	11010	11000

Feature spreading can be efficiently implemented using the OR operator

Detection Stage

Efficient Implementation of our Similarity Measure:

1. Spreading the features
2. **Precompute Response Maps**
3. Use Look-Up Tables
4. Linearize the Memory

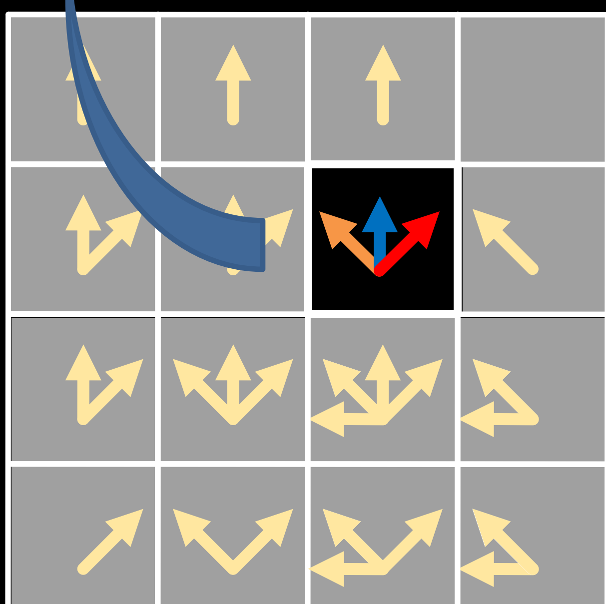
Response Maps

Computation of Response Map for Feature :

$$\max(\,|\cos(\uparrow, \nwarrow)|,\ |\cos(\uparrow, \uparrow)|,\ |\cos(\uparrow, \nearrow)|\,)$$

$$= |\cos(\uparrow, \uparrow)|$$

$$= \mathbf{1}$$



Response Map for Feature $\uparrow$:

1	1	1	0
1	1	1	0
1	1	1	0.7
0.7	0.7	0.7	0.7

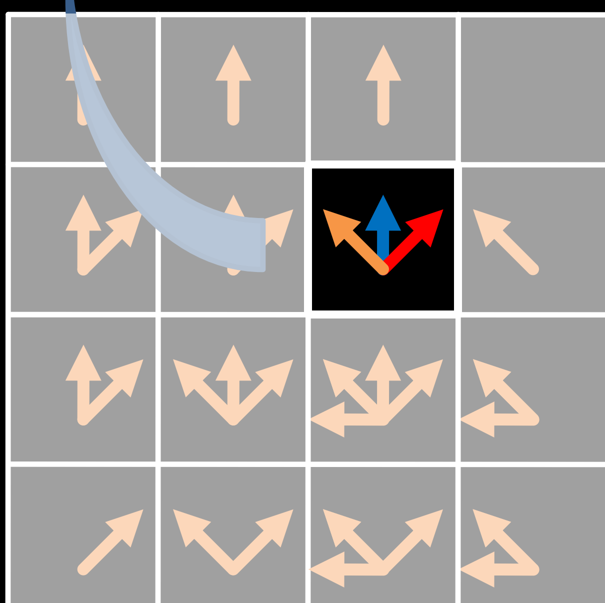
Response Maps

Computation of Response Map for Feature $\rightarrow$:

$$\max(\ |\cos(\rightarrow, \nwarrow)|,\ |\cos(\rightarrow, \uparrow)|,\ |\cos(\rightarrow, \nearrow)|)$$

$$= |\cos(\rightarrow, \nearrow)|$$

$$= \mathbf{0.7}$$



Response Map for Feature $\rightarrow$:

0	0	0	0
0.7	0.7	0.7	0.7
0.7	0.7	1	1
0.7	0.7	1	1

Detection Stage

Efficient Implementation of our Similarity Measure:

1. Spreading the features
2. Precompute Response Maps
3. **Use Look-Up Tables**
4. Linearize the Memory

Response Maps

Computation of Response Map for Feature : Efficient Implementation using Look Up Tables

Look-Up Table for	
-	-
-	-
01110	1
-	-

00100	00100	00100	00000
00110	[illegible]	01110	01000
00110	01110	11110	11000
00010	01010	11010	11000

Response Map for Feature :

1	1	1	0
1	1	1	0.7
1	1	1	0.7
0.7	0.7	0.7	0.7

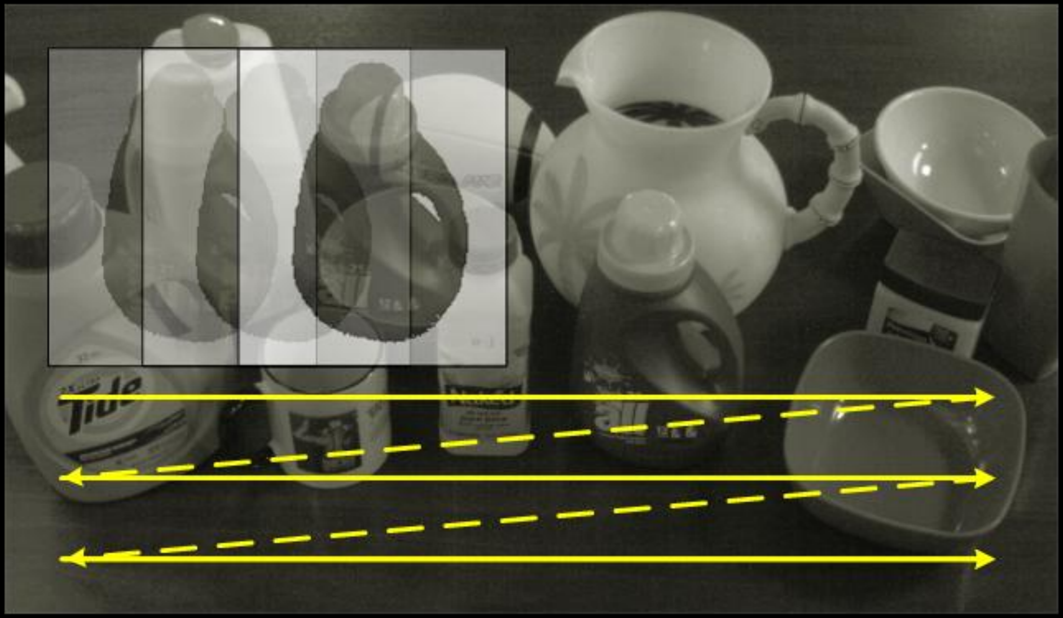
Detection Stage

Efficient Implementation of our Similarity Measure:

1. Spreading the features
2. Precompute Response Maps
3. Use Look-Up Tables
4. **Linearize the Memory**

Linearize Memory

Due to the feature ***spreading*** we have ***invariance to small translations***. Therefore we only have to consider each **i'th pixel**…



Linearize the gradient response maps: **- SSE instructions**

- Avoid cache misses

pointcloudlibrary

Linearize Memory

Due to the feature ***spreading*** we have ***invariance to small translations***. Therefore we only have to consider each **i'th pixel**…

Precomputed Response Map for :

T=2

0.7	0	1	0	…	T=2
0.7	1	1	0.7	…	
0.7	1	0	0.7	…	
0	0	0.7	0	…	
⋮	⋮	⋮	⋮	⋱	

Linearize the gradient response maps:

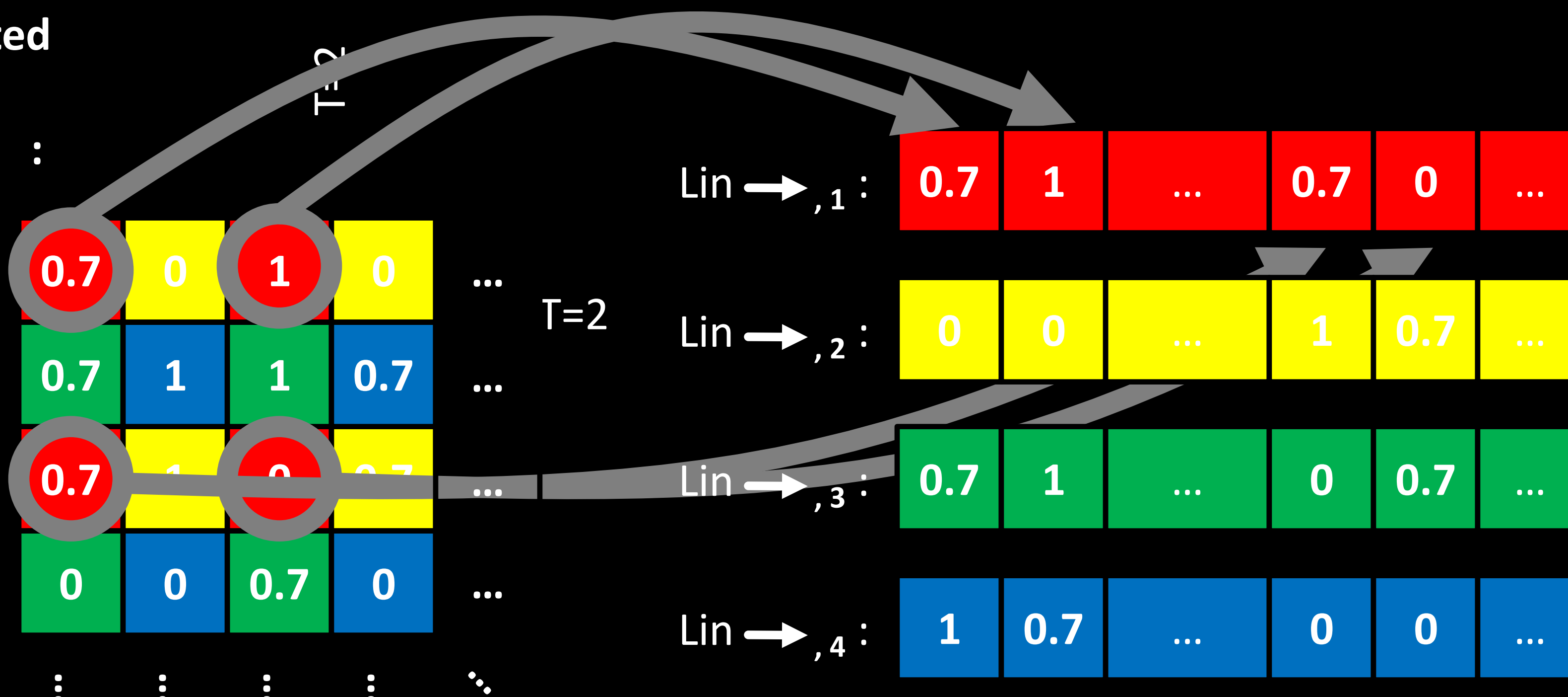
- **SSE instructions**
- **Avoid cache misses**

pointcloudlibrary

Linearize Memory

Due to the feature ***spreading*** we have ***invariance to small translations***. Therefore we only have to consider each **i'th pixel**…

Precomputed Response Map for :



Linearize the gradient response maps: - **SSE instructions**

- **Avoid cache misses**

Code

We provide two classes:

- pcl::LINEMOD
  - Contains actual implementation of LINEMOD
  - Independent of specific modalities
    → allows implementation of new types of modalities

- pcl::LineRGBD<PointXYZT, RGBT>
  - Simplyfied interface for special case
  - Modalities: Surface Normals, Max Color Gradients

pointcloudlibrary

Code

Training and Detection are very simple with LineRGBD

setup

```
// setup LINEMOD for surface normals and max color gradients
LineRGBD<PointXYZRGBA> line_rgbd;

// setup data
PointCloud<PointXYZRGBA>::Ptr cloud (new PointCloud<PointXYZRGBA> ());
// fill data...

// provide data to LINEMOD
line_rgbd.setInputCloud (cloud);
line_rgbd.setInputColors (cloud);
```

training

```
// setup mask of the desired object to train
MaskMap mask_map (width, height);
// fill mask ...

line_rgbd.createAndAddTemplate (cloud, object_id, mask_map, mask_map, region);
```

detection

```
// detect
vector<LineRGBD<PointXYZRGBA>::Detection> detections;
line_rgbd.detect (detections);

// do something with the detections
const LineRGBD<PointXYZRGBA>::Detection & detection = detections[i];
```

Questions?

Questions?