CVPR 2012 Providence, RI, USA

PCL :: Object Detection – LINEMOD
Stefan Holzer, TU Munich (TUM)
Stefan Hinterstoßer, TU Munich (TUM)
June 17, 2012
Goal for today: **Object Detection using PCL**

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

- 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...
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**
Aims for **Cluttered Scenes**:  

- **Industry**  
- **Household**  
- **Conventions**
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
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
Color Gradients and Surface Normals are Complementary!

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

2D Color Gradients

Computation:

\[
\begin{align*}
\left[ \frac{\text{Dalal et al. CVPR05}}{\text{Gray Value Gradients}} \right] & \quad \left[ \frac{\text{Color Gradients}}{\text{Color Gradients}} \right] \\
\end{align*}
\]

Similarity Measure:

\[
f_g(O_g(r), I_g(t)) = |O_g(r)^{\top}I_g(t)|
\]

- \(O_g(r)\): Color gradient on the template
- \(I_g(t)\): Color gradient on the input image
**Modalities**

**3D Surface Normals**

- **Depth Image**
- **Normal Image**

**Similarity Measure:**

\[
 f_D(O_D(r), I_D(t)) = O_D(r)^T I_D(t)
\]

- \(O_D(r)\): Surface Normal on the template
- \(I_D(t)\): Surface Normal in the input image
We can’t use Interest Points for efficient Object Localization!
Therefore, we have to use a Sliding Window Approach:

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

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

- $\uparrow$ = 00 00 00 01
- $\uparrow$ = 00 00 00 10
- $\rightarrow$ = 00 00 01 00
We first **quantize** the features and **spread** them around their initial position.
We first **quantize** the features and **spread** them around their initial position.
We first **quantize** the features and **spread** them around their initial position.

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:

\[
\text{max} (|\cos(\uparrow, \rightarrow)|, |\cos(\uparrow, \leftarrow)|, |\cos(\uparrow, \downarrow)|)
\]

\[
= |\cos(\uparrow, \leftarrow)|
\]

\[
= 1
\]
Response Maps

Computation of Response Map for Feature \( \rightarrow \) :

\[
\begin{align*}
\max & ( |\cos(\longrightarrow, \textcolor{red}{\uparrow})|, |\cos(\longrightarrow, \textcolor{blue}{\uparrow})|, |\cos(\longrightarrow, \textcolor{red}{\downarrow})| ) \\
& = |\cos(\longrightarrow, \textcolor{red}{\uparrow})| \\
& = 0.7
\end{align*}
\]

Response Map for Feature \( \rightarrow \) :

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0.7 & 0.7 & 0.7 & 0.7 \\
0.7 & 0.7 & 1 & 1 \\
0.7 & 0.7 & 1 & 1 \\
\end{array}
\]
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
Computation of Response Map for Feature

Look-Up Table for Efficient Implementation using Look Up Tables

Response Map for Feature
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**
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
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 0 0 0 0 0 0 0 0 0 0 0 0
0.7 1 1 0.7 0 0 0 0 0 0 0 0 0 0 0
0.7 1 0 0.7 0 0 0 0 0 0 0 0 0 0 0
0 0 0.7 0 0 0 0 0 0 0 0 0 0 0 0
```

**Linearize** the gradient response maps:
- SSE instructions
- Avoid cache misses
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 $T=2$

Lin$_{1,1}$: 0.7 1 ... 0.7 0 ...

Lin$_{1,2}$: 0 0 ... 1 0.7 ...

Lin$_{1,3}$: 0.7 1 ... 0 0.7 ...

Lin$_{1,4}$: 1 0.7 ... 0 0 ...

**Linearize** the gradient response maps: - **SSE instructions**  
- **Avoid cache misses**
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>**
  - Simplified interface for special case
  - Modalities: Surface Normals, Max Color Gradients
Training and Detection are very simple with **LineRGBD**

```cpp
// 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);

// 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, mask_map, region);

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

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