Visual Computing in OpenCV
Lecture 5: Building your own Representation

Michael Felsberg
michael.felsberg@liu.se

Johan Wiklund
johan.wiklund@liu.se
Channel Toolbox

• Tutorial
• Based on Mat/SparseMat
• Makes use of separate channel basis class

```
cvl::ChannelBasis
  cvl::BsplineChannelBasis
  cvl::CombinedChannelBasis
  cvl::Cos2ChannelBasis
```

• cvl::ChannelVector inherits from Mat_<float>
• cvl::ChannelSVector inherits from SparseMat_<float>
Tutorial on Channel Representations

Michael Felsberg

Computer Vision Laboratory
Linköping University, Sweden
What are channels?
Channel Representation

- distribution functions (Granlund, 1973)
- channel coding (Snippe/Koenderink, 1992)
- bandpass channels (Howard/Rogers, 1995)
- population coding (Zemel et al., 1998)
- channel representation (Granlund, 2000)
- channel filtering (Felsberg/Granlund, 2003)
- channel smoothing (Felsberg et al., 2006)
- “bilateral filtering” (Paris/Durand, 2006)
- orientation scores (Duits et al., 2007)
- channel coded feature maps (Jonsson/Felsberg, 2007)
- distribution fields (Sevilla-Lara/Learned-Miller, 2012)
Channel Representation

- *distribution functions* (Granlund, 1973)
- *channel coding* (Snippe/Koenderink, 1992)
- *bandpass channels* (Howard/Rogers, 1995)
- *population coding* (Zemel et al., 1998)
- *channel representation* (Granlund, 2000)
- *channel filtering* (Felsberg/Granlund, 2003)
- *channel smoothing* (Felsberg et al., 2006)
- “*bilateral filtering*” (Paris/Durand, 2006)
- *orientation scores* (Duits et al., 2007)
- *channel coded feature maps* (Jonsson/Felsberg, 2007)
- *distribution fields* (Sevilla-Lara/Learned-Miller, 2012)
Channel Representation

$$c_n(f) = \langle \delta_f | k_n \rangle = \int \delta_f(z) k_n(z) \, dz$$

$$\delta_f(z) = \delta(z - f)$$

$$k_n(z) = k(z - n)$$
Channel Representation

$$c_n(f) = \langle \delta_f | k_n \rangle = \int \delta_f(z) k_n(z) \, dz$$

$$k_n(z) = k(z - n)$$

$$\delta_f(z) = \delta(z - f)$$
Channel Representation

Encode values in $K$-D channel vector

$$x_k = F_k(f) \quad k = 1, \ldots, K$$

Motivated from population coding, sparse coding
Channel Representation

\[ x = \begin{bmatrix} 0 & 0.25 & 1.0 & 0.25 & 0 & 0 & 0 & 0.25 & 1.0 & 0.25 & 0 & 0 & 0 \end{bmatrix}^T \]
B-Spline Encoding

The value of the $k$-th channel is obtained by

$$x_k(f) = B_2(f - k) \quad k = 1 \ldots K$$

($f$ is shifted and rescaled such that the channels are at integer positions)

$$k = \text{round}(f)$$

$$x[k-1] = (f-k-0.5)^2/2$$

$$x[k] = 0.75 - (f-k)^2$$

$$x[k+1] = (k-f-0.5)^2/2$$

$$x[1 \ldots k-2] = x[k+2 \ldots K] = 0$$
Channels are ...

- soft histograms
- frame vector projections
- different from Parzen window/kernel density estimators (not located at samples)
Kernel Density Estimation

- Estimate pdf from samples by convolving with a kernel function

\[ \tilde{p}(f) = \frac{1}{N} \sum_{n=1}^{N} k(f - f_n) \]

- Expectation of estimate:

\[ \mathbb{E}\{\tilde{p}(f)\} = \int k(f - f') p(f') \, df' = (k * p)(f) \]
Relation to Channels

• Adding channel representation of samples = sampled kernel density estimation

\[
E\left\{ \frac{1}{N} \sum_{n=1}^{N} u_k(f_n) \right\} = E\{\tilde{p}(f)\}_{f=k} \\
= (B_2 \ast p)(f)_{f=k} \quad k = 1 \ldots K
\]
Problem: Image Denoising

- Real data is noisy and discontinuous
  - Inlier noise
  - Salt&Pepper noise
  - Image discontinuities
Channel Smoothing

Diagram showing the process of channel smoothing, including encoding and decoding steps, with a focus on local averaging of channels and metamery region.
Decoding

- Normalized convolution of the channel vector

\[ f_{k_0} = \frac{u_{k_0+1}(f) - u_{k_0-1}(f)}{u_{k_0-1}(f) + u_{k_0}(f) + u_{k_0+1}(f)} + k_0 \]

- Choice of \( k_0 \):
  - Largest denominator (3-box filter)
  - Additional: local maximum
LS & Robust Optimization

• Minimize error functional:

\[ E(f_0) = \int (f - f_0)^2 p(f) \, df \]

\[ f_0 = \arg \min E(f_0) \]

• Idea of robust error norm:
  – saturated for outliers
  – quadratic near the origin

• in Bayesian sense
LS & Robust Optimization

- Minimize error functional:
  \[ E(f_0) = \int \rho(f - f_0)p(f) \, df \]
  \[ f_0 = \arg \min E(f_0) \]

- Idea of robust error norm:
  - saturated for outliers
  - quadratic near the origin

- in Bayesian sense
• Necessary condition:

\[ 0 = \int (f - f_0) \, p(f) \, df \]

\[ f_0 = \int f \, p(f) \, df \]

• Robust influence:
  – zero for outliers
  – no direct solution
LS & Robust Optimization

• Necessary condition:

\[ 0 = \int \psi(f - f_0) p(f) \, df \]

\[ \psi = \rho' \]

• Robust influence:
  – zero for outliers
  – no direct solution

Efficient methods required!
Influence Function of C.R.

 Obtained from linear decoding:

$$\psi(f) = B_2(f - 1) - B_2(f + 1)$$

$$f_{k_0} = \frac{u_{k_0+1}(f) - u_{k_0-1}(f)}{u_{k_0-1}(f) + u_{k_0}(f) + u_{k_0+1}(f)} + k_0$$
Error Norm of C.R.

Obtained by integrating the influence function:

\[ \rho(f) = 2B_3 \left( \frac{1}{2} \right) - B_3 \left( f + \frac{1}{2} \right) - B_3 \left( f - \frac{1}{2} \right) \]
Quantization Effect

- Noisy signal
- Standard decoding, $\sigma=5$
- Standard decoding, $\sigma=50$
- Optimized decoding, $\sigma=50$
Algorithm 1 Channel smoothing algorithm.

Require: $f \in [1.5; N - 0.5]$

1: for $n = 1$ to $N$ do
2: \quad $c_n \leftarrow B_2(f - n)$
3: \quad $c_n \leftarrow g_\sigma * c_n$
4: end for
5: $[\hat{f}, \hat{p}] \leftarrow \text{decode}(c)$
### Algorithm 1 Channel smoothing algorithm.

**Require:** \( f \in [1.5; N - 0.5] \)

1: \textbf{for} \( n = 1 \) to \( N \) \textbf{do}
2: \hspace{1em} \( c_n \leftarrow B_2(f - n) \)
3: \hspace{1em} \( c_n \leftarrow g_\sigma \ast c_n \)
4: \textbf{end for}
5: \( [\hat{f}, \hat{\rho}] \leftarrow \text{decode}(c) \)
Algorithm 1 Channel smoothing algorithm.

Require: $f \in [1.5; N - 0.5]$

1: for $n = 1$ to $N$ do
2: \hspace{1em} $c_n \leftarrow B_2(f - n)$
3: \hspace{1em} $c_n \leftarrow g_\sigma * c_n$
4: end for
5: $[\hat{f}, \hat{\rho}] \leftarrow \text{decode}(c)$
## Channel Smoothing

**Algorithm 1** Channel smoothing algorithm.

**Require:** $f \in [1.5; N - 0.5]$

1. for $n = 1$ to $N$ do
2. $c_n \leftarrow B_2(f - n)$
3. $c_n \leftarrow g_\sigma \ast c_n$
4. end for
5. $[\hat{f}, \hat{\rho}] \leftarrow \text{decode}(c)$
Image Denoising
Random Sample

• Real data is incomplete
Orientation Estimation

- orientation estimation using the Riesz transform
  - Gaussian noise variance vs. mean orientation error
  - Salt & Pepper noise vs. mean orientation error
  - Line styles and labels:
    - raw estimates (35.6°)
    - bilateral filtering (15.9°)
    - diffusion (16.0°)
    - channel smoothing (13.6°)
    - mean shift filtering (17.5°)
Disparity Estimation
Disparity Estimation
Drawback

- no coherence enhancing filtering possible
Channel Matrix
Channel Matrix
Experiment

original image

coherence enhancing diffusion

anisotropic channel smoothing
Experiment

original image  coherence enhancing diffusion  anisotropic channel smoothing
Corner Detection
Motivation CCFM

frame#: 103
resolution: 78 x 78
channels: 19
• point-wise encoding

\[ c_{l,m,n}(f(x, y)) = k_f(f(x, y) - n)k_x(x - l)k_y(y - m) \]
Object Recognition
COIL-100 Objects

- All 100 objects
- 12 / 60 view for training / evaluation
New Linear Scale-Space

• simultaneously increasing scale in spatial domain and feature domain is obviously wrong

• from a statistical point of view it makes sense to increase feature resolution with decreasing spatial resolution
Algorithm 7 CCFM smoothing algorithm.

Require: $f \in [1.5; N - 0.5]$

Require: $x = (x, y)^T \in [1.5; X - 0.5] \times [1.5; Y - 0.5]$

1: $C \leftarrow \text{CCFM}(x, y, f)$
2: for all $x$ do
3: $c_f \leftarrow \text{interpolate}(C, x)$
4: $[f(x) \ E(x)] \leftarrow \text{decode}(c_f)$
5: $i(x) \leftarrow \text{arg max}_n E_n(x)$
6: $[\hat{f}(x) \ \hat{E}(x)] \leftarrow [f_{i(x)}(x) \ E_{i(x)}(x)]$
7: end for
ChannelVector

- ChannelVector () *Standard constructor.*
- ChannelVector (ChannelBasis *chBasis) *Constructor with basis initialization.*
- ChannelVector (ChannelBasis *chBasis, const cv::Mat_<float> coeffs) *Constructor with basis initialization and copying coefficient matrix.*
- void addSample (const cv::Mat &vals) *Add sample(s) functionality.*
- void decode (cv::Mat &res, const int nrModes=1) *const Decoding functionality.*
ChannelVector

- void normalize () *Normalize the vector.*
- void setChannelBasis (ChannelBasis *chBasis) *Set or change the channel basis to be used.*
- void channelImage (cv::Mat &res) const *Generate different layout, suitable for channel smoothing.*
- void histogramMatrix (cv::Mat &res) const *Generate different layout, compatible with histograms (N-D matrices); last dimension corresponds to the previous rows.*
cvl::ChannelVector::ChannelVector(
    ChannelBasis *chBasis):
    cv::Mat_<float>(1,chBasis->getNrChannels(),0.0f){
        m_chBasis = chBasis;
        m_support.resize(2);
        m_support[0] = 1;
        m_support[1] = 1;
    }
void cvl::ChannelVector::setChannelBasis(ChannelBasis *chBasis) {
    m_chBasis = chBasis;
    this->create(1,
        m_chBasis->getNrChannels());
    this->setTo(0.0f);
    return;
}
Design Questions

void cvl::ChannelVector::normalize()
{
    float nrm = norm(*this, cv::NORM_L1);
    if (nrm > 0)
        (*this) *= (m_chBasis->getNorm() / nrm);
    else
        this->setTo(m_chBasis->getNorm() / this->cols);
    return;
}
void cvl::ChannelVector::channelImage(
    cv::Mat &res) const {
    res = reshape(cols,m_support[0]);
}

(the usual design is 1-channel, cols = number of channels, rows = number of samples)
void cvl::ChannelVector::histogramMatrix(cv::Mat &res) const {
    std::vector<int> sizes(1);
    m_chBasis->getNrChannelsVec(sizes);
    int numDims = (int)sizes.size()+1;
    int sizeIn[numDims];
    for (int i=0; i<numDims-1; ++i)
        sizeIn[i] = sizes[i];
    sizeIn[numDims-1] = rows;
    cv::Mat resTmp(numDims,sizeIn,CV_32F,this->data);
    res = resTmp;
}
ChannelSVector

• ChannelSVector () *Standard constructor.*
• ChannelSVector (ChannelBasis *chBasis) *Constructor with basis initialization.*
• void addSample (const cv::Mat &vals) *Add sample(s) functionality.*
• void decode (cv::Mat_<float> &res, const int nrModes=1) const *Decoding functionality.*
• void normalize () *Normalize the vector.*
• void setChannelBasis (ChannelBasis *chBasis) *Set or change the channel basis to be used.*
cvl::ChannelSVector::ChannelSVector(ChannelBasis *chBasis):cv::SparseMat_<float>() {
    m_chBasis = chBasis;
    int sizes[] = {chBasis->getNrChannels()};
    create(1, sizes);
    m_support.resize(2);
    m_support[0] = 1;
    m_support[1] = 1;
}
void cvl::..::decode(cv::Mat_<float> &res, const int nrModes) const {
    std::vector<int> sizes(1);
    m_chBasis->getNrChannelsVec(sizes);
    int nrChannels = nrModes*(sizes.size()+1);
    CV_Assert(nrChannels<=CV_CN_MAX);
    int nrEls = m_support[0]*m_support[1];
    res.create(nrEls,nrChannels);
    cv::Mat_<float> res_col(1,nrChannels);
    cv::Mat_<float> tmpM(size(1),size(0),0.0f);
    for (cv::SparseMatConstIterator_<float> it = begin(); it != end(); ++it)
        tmpM(it.node()->idx[1],it.node()->idx[0])=*it;
    for (int cdx = 0; cdx<size(1); cdx++) {
        m_chBasis->decode(res_col,tmpM.row(cdx),nrModes);
        res_col.copyTo(res(cv::Range(cdx,cdx+1),cv::Range::all()));
    }
    res.reshape(nrChannels,m_support[0]);
}