**COMPUTER VISION AND IMAGE PROCESSING**

**Percobaan 8: Filter Bank**



**Oleh:**

**Luthfi Aminulloh 1020181013**

**TEKNIK ELEKTRO**

**PROGRAM PASCASARJANA S2 TERAPAN**

**POLITEKNIK ELEKTRONIKA NEGERI SURABAYA**

**2019**

# Tujuan Percobaan

1. Mahasiswa mengetahui transformasi citra dari domain waktu diskrit menjadi domain frekuensi diskrit.
2. Mahasiswa memahami bahwa transformasi dari waktu ke frekuensi dapat menunjukkan detail informasi dari sisi lain.

# Persiapan

1. Praktikum ini dapat dikerjakan dengan pra-syarat bahwa mahasiswa:

* Telah mendapatkan matematika lanjut
* Memiliki konsep dasar sinyal dan sistem
* Memiliki konsep dasar pemrograman dan pengoperasian MS Visual C++

1. Software yang diperlukan:

* Microsoft Visual C++ 2010 express atau lebih tinggi.
* OpenCV Library 2.4.9 atau lebih tinggi.

1. Sarana penunjang praktikum:

* File gambar (.bmp, .jpg)
* File video (.avi)

# Pendahuluan

# Prosedur percobaan

## Wavelet

|  |
| --- |
| List Program: |
| class image  {  public:  Mat im, im1, im2, im3, im4, im5, im6, temp, im11, im12, im13, im14, imi, imd, imr;  float a, b, c, d; int getim();  };  int image::getim()  {  im = imread("../data/OrangeCat.jpg", 0); //Load image in Gray Scale  cv::Mat im\_res(1024,1024,CV\_8UC1);  cv::resize(im,im\_res,cv::Size(1024,1024),0,0,1);  imi=Mat::zeros(im.rows,im.cols,CV\_8U);  im.copyTo(imi);  int rcnt, ccnt;  im.convertTo(im, CV\_32F, 1.0, 0.0);  im1 = Mat::zeros(im.rows / 2, im.cols, CV\_32F);  im2 = Mat::zeros(im.rows / 2, im.cols, CV\_32F);  im3 = Mat::zeros(im.rows / 2, im.cols / 2, CV\_32F);  im4 = Mat::zeros(im.rows / 2, im.cols / 2, CV\_32F);  im5 = Mat::zeros(im.rows / 2, im.cols / 2, CV\_32F);  im6 = Mat::zeros(im.rows / 2, im.cols / 2, CV\_32F);  //--------------Decomposition-------------------  for (rcnt = 0;rcnt<im.rows;rcnt += 2)  {  for (ccnt = 0;ccnt<im.cols;ccnt++)  {  a = im.at<float>(rcnt, ccnt);  b = im.at<float>(rcnt + 1, ccnt);  c = (a + b)\*0.707;  d = (a - b)\*0.707;  int \_rcnt = rcnt / 2;  im1.at<float>(\_rcnt, ccnt) = c;  im2.at<float>(\_rcnt, ccnt) = d;  }  }  for (rcnt = 0;rcnt<im.rows / 2;rcnt++)  {  for (ccnt = 0;ccnt<im.cols;ccnt += 2)  {  a = im1.at<float>(rcnt, ccnt);  b = im1.at<float>(rcnt, ccnt + 1);  c = (a + b)\*0.707;  d = (a - b)\*0.707;  int \_ccnt = ccnt / 2;  im3.at<float>(rcnt, \_ccnt) = c;  im4.at<float>(rcnt, \_ccnt) = d;  }  }  for (rcnt = 0;rcnt<im.rows / 2;rcnt++)  {  for (ccnt = 0;ccnt<im.cols;ccnt += 2)  {  a = im2.at<float>(rcnt, ccnt);  b = im2.at<float>(rcnt, ccnt + 1);  c = (a + b)\*0.707;  d = (a - b)\*0.707;  int \_ccnt = ccnt / 2; im5.at<float>(rcnt, \_ccnt) = c;  im6.at<float>(rcnt, \_ccnt) = d;  }  }  imr = Mat::zeros(im.rows, im.cols, CV\_32F);  imd = Mat::zeros(im.rows, im.cols, CV\_32F);  im3.copyTo(imd(Rect(0, 0, im.cols\*0.5, im.rows\*0.5)));  im4.copyTo(imd(Rect(0, im.rows\*0.5 - 1, im.cols\*0.5, im.rows\*0.5)));  im5.copyTo(imd(Rect(im.cols\*0.5 - 1, 0, im.cols\*0.5, im.rows\*0.5)));  im6.copyTo(imd(Rect(im.cols\*0.5 - 1, im.rows\*0.5 -1, im.cols\*0.5, im.rows\*0.5)));  //---------------------Reconstruction---------------  im11 = Mat::zeros(im.rows / 2, im.cols, CV\_32F);  im12 = Mat::zeros(im.rows / 2, im.cols, CV\_32F);  im13 = Mat::zeros(im.rows / 2, im.cols, CV\_32F);  im14 = Mat::zeros(im.rows / 2, im.cols, CV\_32F);  for (rcnt = 0;rcnt<im.rows / 2;rcnt++)  {  for (ccnt = 0;ccnt<im.cols / 2;ccnt++)  {  int \_ccnt = ccnt \* 2;  im11.at<float>(rcnt, \_ccnt) = im3.at<float>(rcnt, ccnt);  //Upsampling of stage I  im12.at<float>(rcnt, \_ccnt) = im4.at<float>(rcnt, ccnt);  im13.at<float>(rcnt, \_ccnt) = im5.at<float>(rcnt, ccnt);  im14.at<float>(rcnt, \_ccnt) = im6.at<float>(rcnt, ccnt);  }  }  for (rcnt = 0;rcnt<im.rows / 2;rcnt++)  {  for (ccnt = 0;ccnt<im.cols;ccnt += 2)  {  a = im11.at<float>(rcnt, ccnt);  b = im12.at<float>(rcnt, ccnt);  c = (a + b)\*0.707;  im11.at<float>(rcnt, ccnt) = c;  d = (a - b)\*0.707;  im11.at<float>(rcnt, ccnt + 1) = d;  a = im13.at<float>(rcnt, ccnt);  b = im14.at<float>(rcnt, ccnt);  c = (a + b)\*0.707;  im13.at<float>(rcnt, ccnt) = c;  d = (a - b)\*0.707;  im13.at<float>(rcnt, ccnt + 1) = d;  }  }  temp = Mat::zeros(im.rows, im.cols, CV\_32F);  for (rcnt = 0;rcnt<im.rows / 2;rcnt++)  {  for (ccnt = 0;ccnt<im.cols;ccnt++)  {  int \_rcnt = rcnt \* 2; imr.at<float>(\_rcnt, ccnt) = im11.at<float>(rcnt, ccnt);  //Upsampling at stage II  temp.at<float>(\_rcnt, ccnt) = im13.at<float>(rcnt, ccnt);  }  }  for (rcnt = 0;rcnt<im.rows;rcnt += 2)  {  for (ccnt = 0;ccnt<im.cols;ccnt++)  {  a = imr.at<float>(rcnt, ccnt);  b = temp.at<float>(rcnt, ccnt);  c = (a + b)\*0.707;  imr.at<float>(rcnt, ccnt) = c;  d = (a - b)\*0.707;  imr.at<float>(rcnt + 1, ccnt) = d;  }  }  imd.convertTo(imd, CV\_8U);  namedWindow("Input Image", 1);  imshow("Input Image", imi);  namedWindow("Wavelet Decomposition", 1);  imshow("Wavelet Decomposition", imd);  cv::imwrite("hasil.bmp", imd);  imr.convertTo(imr, CV\_8U);  namedWindow("Wavelet Reconstruction", 1);  imshow("Wavelet Reconstruction", imr);  waitKey(0);  return 0;  }  int main()  {  image my;  my.getim();  return 0;  } |

Hasil:

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

## Gabor Filter

|  |
| --- |
| List Program: |
| cv::Mat mkKernel(int ks, double sig, double th, double lm, double ps)  {  int hks = (ks - 1) / 2;  double theta = th\*CV\_PI / 180;  double psi = ps\*CV\_PI / 180;  double del = 2.0 / (ks - 1);  double lmbd = lm;  double sigma = sig / ks;  double x\_theta;  double y\_theta;  cv::Mat kernel(ks, ks, CV\_32F);  for (int y = -hks; y <= hks; y++)  {  for (int x = -hks; x <= hks; x++)  {  x\_theta = x\*del\*cos(theta) + y\*del\*sin(theta); y\_theta = -x\*del\*sin(theta) + y\*del\*cos(theta); kernel.at<float>(hks + y, hks + x) = (float)exp(-0.5\*(pow(x\_theta, 2) + pow(y\_theta, 2)) / pow(sigma, 2))\* cos(2 \* CV\_PI\*x\_theta / lmbd + psi);  }  }  return kernel;  }  int kernel\_size = 21;  int pos\_sigma = 5;  int pos\_lm = 50;  int pos\_th = 0;  int pos\_psi = 90;  cv::Mat src\_f;  cv::Mat dest;  void Process(int, void \*)  {  double sig = pos\_sigma;  double lm = 0.5 + pos\_lm / 100.0;  double th = pos\_th;  double ps = pos\_psi;  cv::Mat kernel = mkKernel(kernel\_size, sig, th, lm, ps);  cv::filter2D(src\_f, dest, CV\_32F, kernel);  cv::imshow("Process window", dest);  cv::Mat Lkernel(kernel\_size \* 20, kernel\_size \* 20, CV\_32F);  cv::resize(kernel, Lkernel, Lkernel.size());  Lkernel /= 2.;  Lkernel += 0.5;  cv::imshow("Kernel", Lkernel);  cv::Mat mag;  cv::pow(dest, 2.0, mag);  cv::imshow("Mag", mag);  }  int main(int argc, char\*\* argv)  {  cv::Mat image = cv::imread("../data/OrangeCat.jpg", 1);  cv::imshow("Src", image);  cv::Mat src;  cv::cvtColor(image, src, CV\_BGR2GRAY);  src.convertTo(src\_f, CV\_32F, 1.0 / 255, 0);  if (!kernel\_size % 2)  {  kernel\_size += 1;  }  cv::namedWindow("Process window", 1);  cv::createTrackbar("Sigma", "Process window", &pos\_sigma,kernel\_size, Process);  cv::createTrackbar("Lambda", "Process window", &pos\_lm, 100, Process);  cv::createTrackbar("Theta", "Process window", &pos\_th, 180, Process);  cv::createTrackbar("Psi", "Process window", &pos\_psi, 360, Process);  Process(0, 0);  cv::waitKey(0);  return 0;  } |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |