**COMPUTER VISION AND IMAGE PROCESSING**

**Percobaan 8: Template Matching**



**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: |
| // Percobaan 1 Template matching  /// Global Variables  Mat img;  Mat templ;  Mat result;  char\* image\_window = "Source Image";  char\* result\_window = "Result window";  int match\_method;  int max\_Trackbar = 5;  /// Function Headers  void MatchingMethod(int, void\*);  /\*\* @function main \*/  int main(int argc, char\*\* argv)  {  /// Load image and template  img = imread("1.png", 1);  templ = imread("1a.png", 1);  /// Create windows  namedWindow(image\_window, CV\_WINDOW\_AUTOSIZE);  namedWindow(result\_window, CV\_WINDOW\_AUTOSIZE);  /// Create Trackbar  char\* trackbar\_label = "Method: \n 0: SQDIFF \n 1: SQDIFF NORMED \n 2: TM CCORR \n 3: TM CCORR NORMED \n 4: TM COEFF \n 5: TM COEFF NORMED";  createTrackbar(trackbar\_label, image\_window, &match\_method, max\_Trackbar, MatchingMethod);  MatchingMethod(0, 0);  waitKey(0);  return 0;  }  /\*\*  \* @function MatchingMethod  \* @brief Trackbar callback  \*/  void MatchingMethod(int, void\*)  {  /// Source image to display  Mat img\_display;  img.copyTo(img\_display);  /// Create the result matrix  int result\_cols = img.cols - templ.cols + 1;  int result\_rows = img.rows - templ.rows + 1;  result.create(result\_rows, result\_cols, CV\_32FC1);  /// Do the Matching and Normalize  matchTemplate(img, templ, result, match\_method);  normalize(result, result, 0, 1, NORM\_MINMAX, -1, Mat());  /// Localizing the best match with minMaxLoc  double minVal;  double maxVal;  Point minLoc;  Point maxLoc;  Point matchLoc;  minMaxLoc(result, &minVal, &maxVal, &minLoc, &maxLoc, Mat());  /// For SQDIFF and SQDIFF\_NORMED, the best matches are lower values. For all the other methods, the higher the better  if (match\_method == CV\_TM\_SQDIFF || match\_method == CV\_TM\_SQDIFF\_NORMED)  {  matchLoc = minLoc;  }  else  {  matchLoc = maxLoc;  }  /// Show me what you got  rectangle(img\_display, matchLoc, Point(matchLoc.x + templ.cols, matchLoc.y + templ.rows), Scalar::all(0), 2, 8, 0);  rectangle(result, matchLoc, Point(matchLoc.x + templ.cols, matchLoc.y + templ.rows), Scalar::all(0), 2, 8, 0);  imshow("image\_window", img\_display);  imshow("result\_window", result);  return;  } |

Hasil:

|  |
| --- |
|  |
|  |

## Gabor Filter

|  |
| --- |
| List Program: |
| #include<stdlib.h>  #include<math.h>  #include<string.h>  #include<iostream>  #include<opencv2\opencv.hpp>  #include<opencv2\core\core.hpp>  #include<opencv2\highgui\highgui.hpp>  #include<opencv2\imgproc\imgproc.hpp>  #include<opencv2\ml\ml.hpp>  #include<opencv2\objdetect\objdetect.hpp>  //using namespace std;  //using namespace cv;  /\*\*  \* Function to perform fast template matching with image pyramid  \*/  void fastMatchTemplate(cv::Mat& srca, // The reference image  cv::Mat& srcb, // The template image  cv::Mat& dst, // Template matching result  int maxlevel) // Number of levels  {  std::vector<cv::Mat> refs, tpls, results;  // Build Gaussian pyramid  cv::buildPyramid(srca, refs, maxlevel);  cv::buildPyramid(srcb, tpls, maxlevel);  cv::Mat ref, tpl, res;  // Process each level  for (int level = maxlevel; level >= 0; level--)  {  ref = refs[level];  tpl = tpls[level];  res = cv::Mat::zeros(ref.size() + cv::Size(1, 1) - tpl.size(), CV\_32FC1);  if (level == maxlevel)  {  // On the smallest level, just perform regular template matching  cv::matchTemplate(ref, tpl, res, CV\_TM\_CCORR\_NORMED);  }  else  {  // On the next layers, template matching is performed on pre-defined  // ROI areas. We define the ROI using the template matching result  // from the previous layer.  cv::Mat mask;  cv::pyrUp(results.back(), mask);  cv::Mat mask8u;  mask.convertTo(mask8u, CV\_8U);  // Find matches from previous layer  std::vector<std::vector<cv::Point> > contours;  cv::findContours(mask8u, contours, CV\_RETR\_EXTERNAL, CV\_CHAIN\_APPROX\_NONE);  // Use the contours to define region of interest and  // perform template matching on the areas  for (int i = 0; i < contours.size(); i++)  {  cv::Rect r = cv::boundingRect(contours[i]);  cv::matchTemplate(  ref(r + (tpl.size() - cv::Size(1, 1))),  tpl,  res(r),  CV\_TM\_CCORR\_NORMED  );  }  }  // Only keep good matches  cv::threshold(res, res, 0.94, 1., CV\_THRESH\_TOZERO);  results.push\_back(res);  }  res.copyTo(dst);  }  int main()  {  cv::Mat ref = cv::imread("1.png");  cv::Mat tpl = cv::imread("1a.png");  if (ref.empty() || tpl.empty())  return -1;  cv::Mat ref\_gray, tpl\_gray;  cv::cvtColor(ref, ref\_gray, CV\_BGR2GRAY);  cv::cvtColor(tpl, tpl\_gray, CV\_BGR2GRAY);  cv::Mat dst;  fastMatchTemplate(ref\_gray, tpl\_gray, dst, 2);  while (true)  {  double minval, maxval;  cv::Point minloc, maxloc;  cv::minMaxLoc(dst, &minval, &maxval, &minloc, &maxloc);  if (maxval >= 0.9)  {  cv::rectangle(  ref, maxloc,  cv::Point(maxloc.x + tpl.cols, maxloc.y + tpl.rows),  CV\_RGB(0, 255, 0), 2  );  cv::floodFill(  dst, maxloc,  cv::Scalar(0), 0,  cv::Scalar(.1),  cv::Scalar(1.)  );  }  else  break;  }  cv::imshow("result", ref);  cv::waitKey();  return 0;  } |

Hasil:

|  |
| --- |
|  |
|  |