# LAMPIRAN

**LAMPIRAN 1 – Source Code**

**main.cpp**

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv2/contrib/contrib.hpp>
#include <iostream>

using namespace cv;
using namespace std;

Mat templ, img, sample_img, sample_img_8bit, lab_img, thresh_img;
int counter;

Mat gaussianPyramid(Mat,int);
Mat laplacianPyramid(Mat,int);
Mat kmeansClustering(Mat);
void rangeScalar(Scalar, Scalar, Scalar);
Mat preprocessingTemplate(Mat);
void subsamplingVideo(Mat,Mat,Mat);
void histogramEqualize(Mat,Mat,int);
void colorQuantization(Mat&);
Mat denoising(Mat,int);
Mat segmentasiWarna(Mat,Scalar,Scalar,Scalar,Scalar,int);
Point templateMatching(Mat,Mat,double);

int main (int argc, char** argv)
{
   int timer = 0;
   counter = 0;
   Point garis;
   VideoCapture cap( argv[1] );
   if ( !cap.isOpened() ) return -1;
   templ = imread( argv[2] );

   VideoWriter
output("output.avi",CV_FOURCC('M','J','P','G'),cap.get(CV_CAP_PROP_FPS),Size(cap.
get(CV_CAP_PROP_FRAME_WIDTH),cap.get(CV_CAP_PROP_FRAME_HEIGHT)),
true);

   Mat temp(templ.rows/2,templ.cols/2,CV_8UC3);
   Mat ROItemp((temp.rows+20)*2,(temp.cols+20)*2,CV_8UC3, Scalar(0));
//    temp = denoising(templ,3);
   temp = gaussianPyramid(templ,2);
   Mat warna = preprocessingTemplate( templ );
   Scalar warna1, warna2, warna3, warna4;
   for(int wr=0; wr<3; wr++)
   {
      warna1[wr] = warna.at<float>(0,wr);
```

```
        warna2[wr] = warna.at<float>(1,wr);
        warna3[wr] = warna.at<float>(2,wr);
        warna4[wr] = warna.at<float>(3,wr);
    }
    Mat img1(Size(100,100),CV_8UC3,warna1);
    Mat img2(Size(100,100),CV_8UC3,warna2);
    Mat img3(Size(100,100),CV_8UC3,warna3);
    Mat img4(Size(100,100),CV_8UC3,warna4);
    cvtColor(img1,img1,CV_BGR2HSV);
    cvtColor(img2,img2,CV_BGR2HSV);
    cvtColor(img3,img3,CV_BGR2HSV);
    cvtColor(img4,img4,CV_BGR2HSV);
    for(int wr=0; wr<3; wr++)
    {
        warna1[wr] = img1.at<uchar>(0,wr);
        warna2[wr] = img2.at<uchar>(0,wr);
        warna3[wr] = img3.at<uchar>(0,wr);
        warna4[wr] = img4.at<uchar>(0,wr);
    }

    for (;;)    {
        cap >> img;
        if ( img.empty() )
            break;

        Point matchLocA, matchLobB;
        if( counter%6 == 0 )
        {
            /* subsampling video */
            Mat panelKiri( img.rows/3*2, img.cols/2, CV_8UC3 );
            Mat panelKanan( img.rows/3*2, img.cols/2, CV_8UC3 );
            subsamplingVideo( img, panelKiri, panelKanan );
            /* histogram equalize */
            histogramEqualize(panelKiri,panelKiri,0);
//            histogramEqualize(panelKanan,panelKanan,0);
            /* color quantization dan denoising */
            colorQuantization( panelKiri );
//            colorQuantization( panelKanan );
            panelKiri = denoising( panelKiri, 3 );
//            panelKanan = denoising( panelKanan, 3 );
            /* segmentasi warna */
            panelKiri = segmentasiWarna(panelKiri,warna1,warna2,warna3,warna4,11);
//            panelKanan = segmentasiWarna(panelKanan,
warna1,warna2,warna3,warna4,11);
            /* template matching */
            matchLocA = templateMatching(panelKiri,temp,0.926);
//            matchLobB = templateMatching(panelKanan,temp,0.92);
        }
        if(matchLocA!=Point(0,0))
        {
```

```
            stringstream filename;
            std::string name ("ROI");
            filename << name << counter << "_" << matchLocA << ".jpeg";
            string filenames = filename.str();
            Mat ROI( img, Rect( Point( matchLocA.x -10, matchLocA.y -10), Point(
matchLocA.x + temp.cols +10, matchLocA.y + temp.rows +10 )));
            imwrite( filenames, ROI);
            ROItemp = laplacianPyramid(ROI,2);
            garis = matchLocA;
            rectangle( img, matchLocA, Point( matchLocA.x + temp.cols, matchLocA.y +
temp.rows ), Scalar(255,0,0), 2, 8, 0 );
            timer = 1;
         }
         if (timer != 0 && timer < 30){
            if (timer != 0 && timer < 20){
               line(img, Point(garis.x-timer*5.5, garis.y-timer*2), Point(img.cols/2-
ROItemp.cols/2,img.rows/3), Scalar(255,0,0), 2, 8, 0 );
            }
            ROItemp.copyTo(img.rowRange(img.rows/3-ROItemp.rows/2,img.rows/3-
ROItemp.rows/2+ROItemp.rows).colRange(img.cols/2-ROItemp.cols/2,img.cols/2-
ROItemp.cols/2+ROItemp.cols));
            rectangle(img,Point(img.cols/2-ROItemp.cols/2,img.rows/3-
ROItemp.rows/2),Point(img.cols/2-ROItemp.cols/2+ROItemp.cols,img.rows/3-
ROItemp.rows/2+ROItemp.rows), Scalar(255,0,0), 2, 8, 0 );
            timer++;
         }else{
            timer = 0;
         }
         output.write(img);
         imshow("source",img);
         counter++;

         if ( waitKey(30) >= 0 ) break;
      }

}

Mat gaussianPyramid(Mat src, int scale)
{
   Mat dst;
   pyrDown( src, dst, Size( src.cols/scale, src.rows/scale ));
   return dst;
}

Mat laplacianPyramid(Mat src, int scale)
{
   Mat dst;
   pyrUp( src, dst, Size( src.cols*scale, src.rows*scale ));
   return dst;
}
```

```
Mat kmeansClustering( Mat src )
{
   int K = 4;
   int n = src.rows * src.cols;
   Mat data = src.reshape(1, n);
   data.convertTo(data, CV_32F);

   vector<int> labels;
   Mat colors;
   kmeans(data, K, labels, cv::TermCriteria(), 1, cv::KMEANS_PP_CENTERS, colors);

   return colors;
}

Mat preprocessingTemplate(Mat src)
{
   Mat klaster;
   klaster = kmeansClustering( src );
   return klaster;
}

void subsamplingVideo(Mat src,Mat dstKiri,Mat dstKanan)
{
   for( int y=0; y<dstKiri.rows; y++ )
   {
      for( int x=0; x<dstKiri.cols; x++ )
      {
         for( int ch=0; ch<3; ch++ )
         {
            dstKiri.at<Vec3b>(y,x)[ch] = src.at<Vec3b>(y,x)[ch];
         }
      }
   }

   for( int y=0; y<dstKanan.rows; y++ )
   {
      for( int x=0; x<dstKanan.cols; x++ )
      {
         for( int ch=0; ch<3; ch++ )
         {
            dstKanan.at<Vec3b>(y,x)[ch] = src.at<Vec3b>(y,(src.cols-
dstKanan.cols+x))[ch];
         }
      }
   }
}

void histogramEqualize(Mat src,Mat dst,int ch)
{
   vector<Mat> kanal;
```

```
    cvtColor( src, dst, CV_BGR2YCrCb );
    split( dst, kanal );
    equalizeHist( kanal[ch], kanal[ch] );
    merge( kanal, dst );
    cvtColor( dst, dst, CV_YCrCb2BGR );
}

Mat denoising(Mat src,int uk)
{
    Mat dst;
    GaussianBlur(src, dst, Size(uk,uk),uk,uk);
    return dst;
}

void colorQuantization(Mat& src)
{
    uchar* pixelPtr = src.data;
    for (int i = 0; i < src.rows; i++)
    {
        for (int j = 0; j < src.cols; j++)
        {
            const int pi = i*src.cols*3 + j*3;
            for(int ch=0; ch<3; ch++)
            {
                if(pixelPtr[pi + ch] < 240)
                    pixelPtr[pi + ch] = uchar(pixelPtr[pi + ch] / 16.0 + 0.5) * 16;
                else
                    pixelPtr[pi + ch] = 255;
            }
        }
    }
}

Mat segmentasiWarna(Mat src,Scalar warna1,Scalar warna2,Scalar warna3,Scalar
warna4,int br)
{
    Mat imgHSV,dst;
    cvtColor(src,imgHSV,CV_BGR2HSV);
    imgHSV = gaussianPyramid(imgHSV,2);
    Mat mask1, mask2, mask3, mask4, allMask;
    int r = 30;

    if(warna1[0]<r)
    {
        Mat mask11, mask12;
        inRange(imgHSV, Scalar(180+(warna1[0]-r),50,50), Scalar(180,255,255), mask11);
        inRange(imgHSV, Scalar(0,50,50), Scalar(r-warna1[0],255,255), mask12);
        mask1 = mask11 + mask12;
    }
    else if(warna1[0]>(180-r))
```

```
{
   Mat mask11, mask12;
   inRange(imgHSV, Scalar(warna1[0]-r,50,50), Scalar(180,255,255), mask11);
   inRange(imgHSV, Scalar(0,50,50), Scalar(0+(180-warna1[0]),255,255), mask12);
   mask1 = mask11 + mask12;
}
else
   inRange(imgHSV, Scalar(warna1[0]-r,50,50), Scalar(warna1[0]+r,255,255), mask1);
if(warna2[0]<r)
{
   Mat mask21, mask22;
   inRange(imgHSV, Scalar(180+(warna2[0]-r),50,50), Scalar(180,255,255), mask21);
   inRange(imgHSV, Scalar(0,50,50), Scalar(r-warna2[0],255,255), mask22);
   mask2 = mask21 + mask22;
}
else if(warna2[0]>(180-r))
{
   Mat mask21, mask22;
   inRange(imgHSV, Scalar(warna2[0]-r,50,50), Scalar(180,255,255), mask21);
   inRange(imgHSV, Scalar(0,50,50), Scalar(0+(180-warna2[0]),255,255), mask22);
   mask2 = mask21 + mask22;
}
else
   inRange(imgHSV, Scalar(warna2[0]-r,50,50), Scalar(warna2[0]+r,255,255), mask2);
if(warna3[0]<r)
{
   Mat mask31, mask32;
   inRange(imgHSV, Scalar(180+(warna3[0]-r),50,50), Scalar(180,255,255), mask31);
   inRange(imgHSV, Scalar(0,50,50), Scalar(r-warna3[0],255,255), mask32);
   mask3 = mask31 + mask32;
}
else if(warna3[0]>(180-r))
{
   Mat mask31, mask32;
   inRange(imgHSV, Scalar(warna3[0]-r,50,50), Scalar(180,255,255), mask31);
   inRange(imgHSV, Scalar(0,50,50), Scalar(0+(180-warna3[0]),255,255), mask32);
   mask3 = mask31 + mask32;
}
else
   inRange(imgHSV, Scalar(warna3[0]-r,50,50), Scalar(warna3[0]+r,255,255), mask3);
if(warna4[0]<r)
{
   Mat mask41, mask42;
   inRange(imgHSV, Scalar(180+(warna4[0]-r),50,50), Scalar(180,255,255), mask41);
   inRange(imgHSV, Scalar(0,50,50), Scalar(r-warna4[0],255,255), mask42);
   mask4 = mask41 + mask42;
}
else if(warna4[0]>(180-r))
{
   Mat mask41, mask42;
```

```
        inRange(imgHSV, Scalar(warna4[0]-r,50,50), Scalar(180,255,255), mask41);
        inRange(imgHSV, Scalar(0,50,50), Scalar(0+(180-warna4[0]),255,255), mask42);
        mask4 = mask41 + mask42;
    }
    else
        inRange(imgHSV, Scalar(warna4[0]-r,50,50), Scalar(warna4[0]+r,255,255), mask4);

    allMask = mask1 + mask2 + mask3 + mask4;
    Mat dilatedMask;
    dilate(allMask,dilatedMask,Mat());
    Mat mask,pyramid;
    mask = denoising( dilatedMask, br );
    pyramid = laplacianPyramid( mask, 2 );
    src.copyTo(dst,pyramid);
    return dst;
}

Point templateMatching(Mat src,Mat temp,double thresholdnum)
{
    Point matchLoc;
    Mat result;
    int baris_result = src.rows - temp.rows + 1;
    int kolom_result = src.cols - temp.cols + 1;
    result.create( baris_result, kolom_result, CV_32FC1 );
    matchTemplate( src, temp, result, CV_TM_CCORR_NORMED );

    Mat dilated, thresholed_matching_space, local_maxima, thresholded_8bit;
    dilate( result, dilated, Mat() );
    compare( result, dilated, local_maxima, CMP_EQ );
    threshold( result, thresholed_matching_space, thresholdnum, 255, THRESH_BINARY
);
    thresholed_matching_space.convertTo( thresholded_8bit, CV_8U );
    bitwise_and( local_maxima, thresholded_8bit, local_maxima );

    double minVal, maxVal;
    Point minLoc, maxLoc;

    minMaxLoc( local_maxima, &minVal, &maxVal, &minLoc, &maxLoc, Mat() );

    if( maxVal > 0 )
    {
        matchLoc = maxLoc;
        return matchLoc;
    }

}
```