headdetection.h

```
#ifndef HEADDETECTION_H
#define HEADDETECTION_H
#define PI 3.14

#include <QWidget>
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <QLabel>
#include <QPushButton>
#include <QGridLayout>
#include <QTimer>
#include <QSpinBox>
#include <QSlider>
#include <QLineEdit>


#include <stdio.h>
#include <qmath.h>
#include <qqueue.h>
#include "opencv/cv.h"
#include "opencv/highgui.h"
#include "opencv/cxcore.h"


using namespace std;
using namespace cv;

typedef unsigned short uint16_t;

class HeadDetection : public QWidget
{
    Q_OBJECT
public:
    explicit HeadDetection(QWidget *parent = 0);
    ~HeadDetection();
    QImage IplImage2QImage(const IplImage *iplImage);
    float r;
    float d;
    float k;
    QTimer *tmrTimer;
    QLabel *InputLabel;
    QPushButton *PushButton;
    QPushButton *ResetPushbutton;
    QPushButton *ClosePushbutton;
    QLabel *Keliling;
    QGridLayout *Layout;
    QLineEdit *Nama;
    QLineEdit *Umur;

    deque<CvSeq*> samples;
    QImage qt_img_;QImage img_;
    CvMemStorage* storage = cvCreateMemStorage(0);
```

```
    IplImage* grayscaleImg = cvCreateImage(cvSize(540, 480), 8,
1);
    IplImage* frame; //The images you bring out of the camera
    CvCapture *capture_ = 0; //The camera
    IplImage *img;
    CvSeq *circles;
    bool test;

    CvSeq* getCirclesInImage(IplImage*, CvMemStorage*, IplImage*);
    float eucdist(CvPoint, CvPoint);
    void drawCircleAndLabel(IplImage*, float*, const char*);
    bool circlesBeHomies(float*, float*);
    void KelilingLing(float x);

signals:

public slots:
    void Camera();
    void Reset();
    void on_btnPauseResume_clicked();
    void Close();
};

#endif // HEADDETECTION_H
```

Headdetection.cpp

```
#include "headdetection.h"

const int MIN_IDENT = 50;
const int MAX_RAD_DIFF = 10;
const int HISTORY_SIZE = 5;
const int X_THRESH = 15;
const int Y_THRESH = 15;
const int R_THRESH = 20;
const int MATCHES_THRESH = 3;
const int HUE_BINS = 32;

HeadDetection::HeadDetection(QWidget *parent) : QWidget(parent)
{
    InputLabel = new QLabel();
    Keliling = new QLabel();
    PushButton = new QPushButton("Pause");
    PushButton->setFixedSize(80,30);
    ResetPushbutton = new QPushButton("Reset");
    ResetPushbutton->setFixedSize(80,30);
    ClosePushbutton = new QPushButton("Close");
    ClosePushbutton->setFixedSize(80,30);
    Nama = new QLineEdit();
    Nama->setStyleSheet("*{color:yellow;background-color:
rgba(0,0,0,0);border:1px transparant;}");
    Nama->setPlaceholderText("Enter Your Name");
    Umur = new QLineEdit();
    Umur->setStyleSheet("*{color:yellow;background-color:
rgba(0,0,0,0);border:1px transparant;}");
    Umur->setPlaceholderText("Enter Your Age");

    Layout = new QGridLayout(this);
    Layout->addWidget(InputLabel,0,0,50,50);
    Layout->addWidget(Keliling,0,0);
    Layout->addWidget(PushButton, 3,0);
    Layout->addWidget(Nama,1,0);
    Layout->addWidget(Umur,2,0);
    Layout->addWidget(ResetPushbutton,4,0);
    Layout->addWidget(ClosePushbutton,5,0);
    tmrTimer = new QTimer();

    capture_=0;

    frame=0;
    capture_    = cvCaptureFromCAM(0);
    if (!capture_ )
    {
      printf("Could not connect to camera\n" );
      return ;
    }

    frame = cvQueryFrame(capture_ );
    storage = cvCreateMemStorage(0);
```

```
    // Grayscale image
    grayscaleImg = cvCreateImage(cvSize(640, 480), 8/*depth*/,
1/*channels*/);


    connect(tmrTimer,SIGNAL(timeout()),this,SLOT(Camera()));
    connect(ResetPushbutton,SIGNAL(clicked()),this,SLOT(Reset()));
    connect(ClosePushbutton,SIGNAL(clicked()),this,SLOT(close()));
    connect(PushButton,SIGNAL(clicked()),this,
SLOT(on_btnPauseResume_clicked()));
    if (test == true)
        tmrTimer->start(0);
    else
    {
        tmrTimer->stop();
    }
}

HeadDetection::~HeadDetection()
{
}

void HeadDetection::Camera()
{
    //CvCapture *capture_ = cvCaptureFromCAM( 0 );
    frame = cvQueryFrame( capture_ );
    //deque<CvSeq*> stableCircles;

    circles = getCirclesInImage(frame, storage, grayscaleImg);
      for(int i = 0; i < circles->total; i++ )
      {

        int matches = 0;
        float *p = (float*)cvGetSeqElem( circles, i );
        float x = p[0];
        float y = p[1];
        r = p[2];
        d=r*0.102;
        k=d*3.14;

        if (x-r < 0 || y-r < 0 || x+r >= frame->width || y+r >=
frame->height)
        {
          continue;
        }

        for (int j = 0; j < samples.size(); j++)
        {
          CvSeq *oldSample = samples[j];
          for (int k = 0; k < oldSample->total; k++)
          {
            float* p2 = (float*)cvGetSeqElem( oldSample, k );
            if (circlesBeHomies(p, p2))
            {
              matches++;
              break;
```

```
            }
          }
        }

        if (matches > MATCHES_THRESH)
        {
          cvSetImageROI(frame, cvRect(x-r, y-r, 2*r, 2*r));
          IplImage* copy = cvCreateImage(cvSize(2*r, 2*r), frame-
>depth, 3);
          cvCvtColor(frame, copy, CV_BGR2HSV);
          IplImage* hue = cvCreateImage(cvGetSize(copy), copy-
>depth, 1);
          cvSplit(copy, hue, 0, 0, 0);
          int hsize[] = {HUE_BINS};
          float hrange[] = {0,180};
          float* range[] = {hrange};
          IplImage* hueArray[] = {hue};
          //int channel[] = {0};
          CvHistogram* hist = cvCreateHist(1, hsize,
CV_HIST_ARRAY, range, 1);
          cvCalcHist(hueArray, hist, 0, 0);
          cvNormalizeHist(hist, 1.0);

          cvResetImageROI(frame);
          char label[64];
          sprintf(label, "color: %s", "red");
          drawCircleAndLabel(frame, p, label);
        }
      }

      samples.push_back(circles);
      img = cvCloneImage(frame);
      if (img->origin)
      {
          cvFlip(img);
          img->origin= 0;
      }
      if (samples.size() > HISTORY_SIZE)
      {
        samples.pop_front();
      }
      qt_img_=IplImage2QImage(img);
      InputLabel->setPixmap(QPixmap::fromImage(qt_img_));
      cvReleaseImage(&img);
      KelilingLing(k);
}

CvSeq* HeadDetection::getCirclesInImage(IplImage* frame,
CvMemStorage* storage, IplImage* grayscaleImg)
{
  // houghification
  // Convert to a single-channel, grayspace image
  cvCvtColor(frame, grayscaleImg, CV_BGR2GRAY);

  // Gaussian filter for less noise
  cvSmooth(grayscaleImg, grayscaleImg, CV_GAUSSIAN, 7, 9 );
```

```
  return cvHoughCircles(grayscaleImg,
                        storage,
                        CV_HOUGH_GRADIENT,
                        2,
                        grayscaleImg->height/4,
                        200,
                        100 );
}

float HeadDetection::eucdist(CvPoint c1, CvPoint c2)
{
    float d = sqrt(pow((float)c1.x - c2.x, 2) + pow((float)c1.y -
c2.y, 2));
    return d;
}

void HeadDetection::drawCircleAndLabel(IplImage* frame, float* p,
const char* label)
{
      //Draw the circle on the original image
      //There's lots of drawing commands you can use!
      CvFont font;
      cvInitFont(&font, CV_FONT_HERSHEY_SIMPLEX, 1, 1, 0.0, 1, 8);
      //void cvCircle(CvArr* img, CvPoint center, int radius,
CvScalar color, int thickness=1, int line_type=8, int shift=0 )
      cvCircle( frame, cvPoint(cvRound(p[0]),cvRound(p[1])),
                cvRound(p[2]), CV_RGB(255,0,0), 3, 8, 0 );

      cvPutText( frame, label,
cvPoint(cvRound(p[0]),cvRound(p[1])), &font, CV_RGB(255,0,0) );
}

bool HeadDetection::circlesBeHomies(float* c1, float* c2)
{
      return (abs(c1[0]-c2[0]) < X_THRESH) && (abs(c1[1]-c2[1]) <
Y_THRESH) &&
      (abs(c1[2]-c2[2]) < R_THRESH);
}

void HeadDetection::KelilingLing(float x)
{
    QFont fontcm = Keliling->font();
    fontcm.setPointSize(50);
    Keliling->setFont(fontcm);

    QFont font = Keliling->font();
    font.setPointSize(50);
    font.setBold(true);
    Keliling->setFont(font);
    Keliling->setText("<font color='red'>"+
                      QString::number(x,'f',1)
                      +"</font>"+
                      "<font color='yellow' size=1>"+
                        "cm"
```

```
                        +"</font>"
                    );

}

void HeadDetection::on_btnPauseResume_clicked()
{

    if(test==true) test = false;
    else test = true;
    if (test == true)
    {
        tmrTimer->start(0);
    }
    else
    {
        tmrTimer->stop();
    }
    int Result=1;
}

void HeadDetection::Close()
{
    ClosePushbutton->close();
}

void HeadDetection::Reset()
{
    Umur->clear();
    Nama->clear();
}


QImage HeadDetection::IplImage2QImage(const IplImage *iplImage)
{
    int height = iplImage->height;
    int width = iplImage->width;

    const uchar *qImageBuffer =(const uchar*)iplImage->imageData;
    QImage img_(qImageBuffer, width, height,
QImage::Format_RGB888);

    return img_.rgbSwapped();
}
```