ДОДАТОК А

ЛІСТИНГИ ПРОГРАМНИХ КОМПОНЕНТІВ

MainWindow.cpp:

```
#include <iostream>

#include "mainwindow.h"
#include "./ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    connect(ui->scanButton, &QPushButton::pressed, this,
&MainWindow::OnScanPressed);
    connect(ui->closeVideoOutputButton, &QPushButton::pressed, this,
&MainWindow::OnCloseVideoPressed);
    connect(ui->saveButton, &QPushButton::pressed, this,
&MainWindow::OnSaveUserPressed);
    connect(ui->recogniseButton, &QPushButton::pressed, this,
&MainWindow::OnRecognisePressed);

    controller.reset(new ApplicationController(ui->labelVideo, ui->logLabel));
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::OnScanPressed()
{
    ui->saveButton->setEnabled(true);
    controller->addUser();
}

void MainWindow::OnCloseVideoPressed()
{
```

```
        ui->saveButton->setEnabled(false);
        controller->StopDisplayingImage();
        ui->labelVideo->hide();
    }

    void MainWindow::OnSaveUserPressed()
    {
        controller->makePhoto();
    }

    void MainWindow::OnRecognisePressed()
    {
        ui->resLabel->setText("");
        bool res;

        std::thread authThread([&res, this]()
        {
            res = controller->authenticate();

            if(res)
                ui->resLabel->setText("success");
            else
                ui->resLabel->setText( "you fucking donkey");
        });

        authThread.detach();


    }
```

Applicationcontroller.cpp:

```
#include "applicationcontroller.h"

#include <fstream>
#include <QFileSystemModel>

constexpr size_t dummyFaceCount = 40;
constexpr std::string_view fn_haar =
"/usr/local/share/opencv4/haarcascades/haarcascade_frontalface_default.xml";
constexpr std::string_view fn_csv = "/home/pi/facesQT/faces/dummyFaces.csv";
```

```
constexpr std::string_view fn_csv_usr = "/home/pi/facesQT/faces/users.csv";

static const QString FingerprintSerial = "/dev/serial0";
static const QString FingerPrinttBaudrate = "57600";

ApplicationController::ApplicationController(QLabel* videoOutput, QLabel* logLabel) :
userProfcontroller(new UserProfileController())
{
    vidOut = videoOutput;
    logOut = logLabel;
    model = cv::face::LBPHFaceRecognizer::create();
    TrainFaceRecogniserOnDummy();
    UpdateFaceRecogniser(fn_csv_usr.data());
}

void ApplicationController::retrainRecogniser()
{
    model->clear();
    TrainFaceRecogniserOnDummy();
    UpdateFaceRecogniser(fn_csv_usr.data());
}

void ApplicationController::ProcessVideoOutput(EVidOutMode mode,
std::vector<std::string>& paths, int userID, bool& auth)
{
    if(is_displaying)
        return;

        auto deviceId = 0;

        cv::VideoCapture cap(deviceId);
        cap.set(cv::CAP_PROP_FOURCC, cv::VideoWriter::fourcc('M', 'J', 'P', 'G'));

        if(!cap.isOpened())
            std::cerr << "Capture Device ID " << deviceId << "cannot be opened." <<
std::endl;

        haarCascade.load(fn_haar.data());

        cv::Mat frame;
```

```
        Timer timer;
        int labell=-1;
        double accur=0.0;
        int countRecognised = 0;

        timer.start();
        is_displaying = true;
        while(is_displaying)
        {
            cap.read(frame);
            // Clone the cursolutionrent frame:
            cv::Mat original = frame.clone();
            // Convert the current frame to grayscale:
            cv::Mat gray;
            cvtColor(original, gray, cv::COLOR_BGRA2GRAY);
            // Find the faces in the frame:
            std::vector< cv::Rect_<int> > faces;
            haarCascade.detectMultiScale(gray, faces);

            for(size_t i = 0; i < faces.size(); i++)
            {
                cv::Rect face_i = faces[i];
                cv::Mat face = gray(face_i);
                cv::Mat face_resized;
                cv::resize(face, face_resized, cv::Size(100, 100), 1.0, 1.0,
cv::INTER_CUBIC);
                rectangle(original, face_i, CV_RGB(0, 255,0), 1);

                if(mode == EVidOutMode::REGISTER_USER && toMakePhoto)
                    paths.push_back(savePhoto(face_resized, userID));

                if(mode == EVidOutMode::RECOGNISE_USER)
                {
                    int pos_x = std::max(face_i.tl().x - 10, 0);
                    int pos_y = std::max(face_i.tl().y - 10, 0);
                    std::string box_text = cv::format("Prediction = %d Confidence=%f",
labell, accur);
                    putText(original, box_text, cv::Point(pos_x, pos_y),
cv::FONT_HERSHEY_PLAIN, 1.0, CV_RGB(255,0,0), 2.0);
                    model->predict(face_resized, labell, accur);
```

```
                    if(labell == userID && accur > 60)
                        countRecognised++;

                    if(countRecognised > 20)
                    {
                        auth = true;
                        is_displaying = false;
                        break;
                    }
                    else if(timer.elapsedSeconds() > 20)
                    {
                        auth = false;
                        is_displaying = false;
                        break;
                    }
                }
            }

            if(is_displaying)
            {
                cvtColor(original, original, cv::COLOR_BGR2RGB);
                QImage image1= QImage((uchar*) original.data, original.cols,
original.rows, original.step, QImage::Format_RGB888);

                vidOut->setPixmap(QPixmap::fromImage(image1));
                vidOut->show();
            }
        }
}

void ApplicationController::StopDisplayingImage()
{
    if(is_displaying)
    {
        is_displaying = false;
        controllerThread->join();
    }
}

void ApplicationController::UpdateFaceRecogniser(const std::string& path)
```

```
{
    std::vector<cv::Mat> images;
    std::vector<int> labels;
    try {
        readCsv(path, images, labels);
    } catch (cv::Exception& e) {
        std::cerr << "Error opening file \"" << path << "\". Reason: " << e.msg <<
std::endl;
        exit(1);
    }

    if(images.size() == 0)
        return;

    imWidth = images[0].cols;
    imHeight = images[0].rows;

    model->update(images, labels);
}

void ApplicationController::TrainFaceRecogniserOnDummy()
{
    std::vector<cv::Mat> images;
    std::vector<int> labels;
    try {
        readCsv(fn_csv.data(), images, labels);
    } catch (cv::Exception& e) {
        std::cerr << "Error opening file \"" << fn_csv << "\". Reason: " << e.msg <<
std::endl;
        exit(1);
    }
    imWidth = images[0].cols;
    imHeight = images[0].rows;

    model->train(images, labels);
}

void ApplicationController::readCsv(const std::string& filename
                                    , std::vector<cv::Mat>& images
                                    , std::vector<int>& labels
                                    , char separator)
```

```
{
    std::ifstream file(filename.c_str(), std::ifstream::in);

    if (!file)
    {
        std::string error_message = "No valid input file was given, please check the
given filename.";
        std::cerr << error_message << std::endl;
        return;
    }

    std::string line, path, classlabel;
    while (getline(file, line))
    {
        std::stringstream liness(line);
        getline(liness, path, separator);
        getline(liness, classlabel);
        if(!path.empty() && !classlabel.empty())
        {
            const auto& im = cv::imread(path, 0);
            cv::Mat resized;
            if(!im.empty())
            {
                resize(im, resized, cv::Size(92, 112), 1.0, 1.0, cv::INTER_CUBIC);
                images.push_back(im);
                labels.push_back(atoi(classlabel.c_str()));
            }
        }
    }
}

void ApplicationController::addUser()
{

    controllerThread = std::make_unique<std::thread>([this]()
    {
        auto userID = userProfcontroller->getUserDBSize() + dummyFaceCount + 1;

        logOut->setText(logOut->text() + QString("Adding user under ID %d
\n").arg(userID));
```

```
        logOut->setText(logOut->text() + QString("Adding fingerprint\n"));

        finger.reset(new Adafruit_Fingerprint{FingerprintSerial,
FingerPrinttBaudrate});
        finger->begin(FingerPrinttBaudrate);
        EnrollExample::runEnroll(*finger.get(), userID);
        finger->close();

        std::vector<std::string> paths;

        bool dummy;
        ProcessVideoOutput(EVidOutMode::REGISTER_USER, paths, userID, dummy);

        UserProfile prof (userID, paths);

        userProfcontroller->addUserProfile(prof);
        vidOut->hide();
        retrainRecogniser();
    });

}

bool ApplicationController::authenticate()
{

    bool faceRecognised;

    controllerThread = std::make_unique<std::thread>([this, &faceRecognised]()
    {
        finger.reset(new Adafruit_Fingerprint{FingerprintSerial,
FingerPrinttBaudrate});
        finger->begin(FingerPrinttBaudrate);
        int userID = FingerprintExample::runFind(*finger.get());
        finger->close();

        std::vector<std::string> dummy;
        ProcessVideoOutput(EVidOutMode::RECOGNISE_USER, dummy, userID, faceRecognised);
    });

    while(!controllerThread->joinable()){}
```

```
    controllerThread->join();
    vidOut->hide();

    return faceRecognised;
}

void ApplicationController::makePhoto()
{
    if(is_displaying)
        toMakePhoto = true;
}

std::string ApplicationController::savePhoto(cv::Mat& pic, int userID)
{
    assert(is_displaying);
    assert(toMakePhoto);

    toMakePhoto = false;

    auto saveDirPath = "/home/pi/facesQT/faces/g" + QVariant(userID).toString();

    if(!QDir(saveDirPath).exists())
        QDir().mkdir(saveDirPath);

    int fileIndex = 0;

    QString saveFilePath = saveDirPath + "/p" + QVariant(fileIndex).toString() +
".bmp";

    while(QFile(saveFilePath).exists())
    {
        fileIndex++;
        saveFilePath = saveDirPath + "/p" + QVariant(fileIndex).toString() + ".bmp";
    }

    std::cout << saveFilePath.toStdString() << std::endl;

    cv::imwrite(saveFilePath.toStdString(), pic);

    return saveFilePath.toStdString();
}
```

FingerPrintExample.cpp:

```
#include "fingerprintExample.hpp"
#include "Adafruit_Fingerprint.h"

#include "Logger.hpp"
#include <thread>

namespace
{

uint8_t getFingerprintID( Adafruit_Fingerprint& _fingerprintSensor )
{
  uint8_t p = _fingerprintSensor.getImage();
  switch (p) {
    case FINGERPRINT_OK:
      Logger::Instance().logDebugEndl("Image taken");
      break;
    case FINGERPRINT_NOFINGER:
      Logger::Instance().logDebugEndl("No finger detected");
      return p;
    case FINGERPRINT_PACKETRECIEVEERR:
      Logger::Instance().logDebugEndl("Communication error");
      return p;
    case FINGERPRINT_IMAGEFAIL:
      Logger::Instance().logDebugEndl("Imaging error");
      return p;
    default:
      Logger::Instance().logDebugEndl("Unknown error");
      return p;
  }

  // OK success!

  p = _fingerprintSensor.image2Tz();
  switch (p) {
    case FINGERPRINT_OK:
      Logger::Instance().logDebugEndl("Image converted");
      break;
```

```
    case FINGERPRINT_IMAGEMESS:
      Logger::Instance().logDebugEndl("Image too messy");
      return p;
    case FINGERPRINT_PACKETRECIEVEERR:
      Logger::Instance().logDebugEndl("Communication error");
      return p;
    case FINGERPRINT_FEATUREFAIL:
      Logger::Instance().logDebugEndl("Could not find fingerprint features");
      return p;
    case FINGERPRINT_INVALIDIMAGE:
      Logger::Instance().logDebugEndl("Could not find fingerprint features");
      return p;
    default:
      Logger::Instance().logDebugEndl("Unknown error");
      return p;
  }

  // OK converted!
  p = _fingerprintSensor.fingerFastSearch();
  if (p == FINGERPRINT_OK) {
    Logger::Instance().logDebugEndl("Found a print match!");
  } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
    Logger::Instance().logDebugEndl("Communication error");
    return p;
  } else if (p == FINGERPRINT_NOTFOUND) {
    Logger::Instance().logDebugEndl("Did not find a match");
    return p;
  } else {
    Logger::Instance().logDebugEndl("Unknown error");
    return p;

  // found a match!
  Logger::Instance().logDebug("Found ID #");
Logger::Instance().logDebug(_fingerprintSensor.fingerID);
  Logger::Instance().logDebug(" with confidence of ");
Logger::Instance().logDebugEndl(_fingerprintSensor.confidence);
  return _fingerprintSensor.fingerID;
}

// returns -1 if failed, otherwise returns ID #
int getFingerprintIDez( Adafruit_Fingerprint& _fingerprintSensor ) {
```

```
  uint8_t p = _fingerprintSensor.getImage();
  if (p != FINGERPRINT_OK)  return -1;

  p = _fingerprintSensor.image2Tz();
  if (p != FINGERPRINT_OK)  return -1;

  p = _fingerprintSensor.fingerFastSearch();
  if (p != FINGERPRINT_OK)  return -1;

  // found a match!
  Logger::Instance().logDebug("Found ID #");
Logger::Instance().logDebug(_fingerprintSensor.fingerID);
  Logger::Instance().logDebug(" with confidence of ");
Logger::Instance().logDebugEndl(_fingerprintSensor.confidence);
  return _fingerprintSensor.fingerID;
}
}
namespace FingerprintExample
{

int runFind( Adafruit_Fingerprint& _fingerprintSensor )
{
    _fingerprintSensor.getTemplateCount();
    Logger::Instance().logDebugEndl("Sensor contains ");
    Logger::Instance().logDebug( _fingerprintSensor.templateCount );
    Logger::Instance().logDebug(" templates");
    Logger::Instance().logDebugEndl("Waiting for valid _fingerprintSensor...");

    int res = -1;
    using namespace  std::chrono_literals;
    while(res == -1)
    {
        res = getFingerprintIDez( _fingerprintSensor );
        std::this_thread::sleep_for(50ms);
    }

    return res;
}
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
```

```
#include <string.h>
#include <fstream>
#include <sstream>
#include <vector>
#include <math.h>
#include <png++/png.hpp>
#include "Feature.h"
#include "WeakClassifier.h"
#include "StrongClassifier.h"
#include "CascadeClassifier.h"

using namespace std;

void add_vertical_mirror(vector<float*> &set, int width, int height) {
  float *vmirror;
  int i, j, x, y;
  int size = set.size();
  for (i=0; i<size; i++) {
    vmirror = new float[width*height];
    for (y=0; y<height; y++) {
      for (x=0; x<width; x++) {
        vmirror[(y*width)+width-1-x] = set[i][(y*width)+x];
      }
    }
    set.push_back(vmirror);
  }
}

float* matrix_to_vector(float **img, int width, int height) {
  float *v = new float[width*height];
  for (int j=0; j<height; j++) {
    for (int i=0; i<width; i++) {
      v[(width*j)+i]=img[i][j];
    }
  }
  return v;
}

float* integral_image(float *img, int width, int height) {
  float* ii = new float[width*height];
  float* s = new float[width*height];
```

```
  int x, y;

  for (y = 0; y < height; y++) {
    for (x = 0; x < width; x++) {
      if (x == 0) s[(y*width)+x] = img[(y*width)+x];
      else s[(y*width)+x] = s[(y*width)+x-1] + img[(y*width)+x];
      if (y == 0) ii[(y*width)+x] = s[(y*width)+x];
      else ii[(y*width)+x] = ii[((y-1)*width)+x] + s[(y*width)+x];
    }
  }
  return ii;
}

float* squared_integral_image(float *img, int width, int height) {
  float* ii = new float[width*height];
  float* s = new float[width*height];
  int x, y;

  for (y = 0; y < height; y++) {
    for (x = 0; x < width; x++) {
      if (x == 0) s[(y*width)+x] = pow(img[(y*width)+x], 2);
      else s[(y*width)+x] = s[(y*width)+x-1] + pow(img[(y*width)+x], 2);
      if (y == 0) ii[(y*width)+x] = s[(y*width)+x];
      else ii[(y*width)+x] = ii[((y-1)*width)+x] + s[(y*width)+x];
    }
  }
  return ii;
}

float evaluate_integral_rectangle(float *ii, int iiwidth, int x, int y, int w, int h) {
  float value = ii[((y+h-1)*iiwidth)+(x+w-1)];
  if (x > 0) value -= ii[((y+h-1)*iiwidth)+(x-1)];
  if (y > 0) value -= ii[(y-1)*iiwidth+(x+w-1)];
  if (x > 0 && y > 0) value += ii[(y-1)*iiwidth+(x-1)];
  return value;
}

float* normalize(float *img, int width, int height) {
  float mean = 0;
  float stdev = 0;
  int i;
```

```
  for (i = 0; i<width*height; i++) mean += img[i];
  mean = mean/float(width*height);

  for (i = 0; i<width*height; i++) stdev += pow(img[i]-mean, 2);
  stdev = stdev/(width*height);
  stdev = sqrt(stdev);
  if (stdev == 0) stdev = 1;

  for (i = 0; i<width*height; i++) img[i] = (img[i]-mean)/stdev;

  return img;
}

vector<Feature*> generate_feature_set(int base_resolution) {
  vector<Feature*> feature_set;
  Feature *f;
  int minwidth, minheight, width, height, x, y;
  int i = 0;

  // removed 5th type (4 sub-squares)
  //for (int type = 0; type<5; type++) {
  for (int type = 0; type<4; type++) {
    if (type != 2) {
      minwidth = 4;
      width = 4;
    }
    else {
      width = 3;
      minwidth = 3;
    }
    if (type != 3) height = 4;
    else height = 3;
    x = 0;
    y = 0;
    while (height <= base_resolution) {
      while (width <= base_resolution) {
        while (y+height <= base_resolution) {
          while (x+width <= base_resolution) {
            f = new Feature(type, x, y, width, height);
            feature_set.push_back(f);
```

```
            x++;
          }
          x = 0;
          y++;
        }
        y = 0;
        if (type == 0) width+=2;
        else if (type == 2) width+=3;
        else width++;
      }
      width = minwidth;
      if (type == 1) height+=2;
      else if (type == 3) height+=3;
      else height++;
    }
  }
  return feature_set;
}

void print_cclassifier_best_features(CascadeClassifier *cc, int n) {
  vector<StrongClassifier*> sc = cc->getStrongClassifiers();
  vector<WeakClassifier*> wc;
  Feature *f;
  int i = 0;
  for (vector<StrongClassifier*>::iterator it = sc.begin(); it != sc.end(); ++it) {
    wc = (*it)->getWeakClassifiers();
    for (vector<WeakClassifier*>::iterator it2 = wc.begin(); it2 != wc.end(); ++it2) {
      f = (*it2)->getFeature();
      if (i < n) {
        cout << "  #" << i+1 << " (Type Width Height X Y) = (" << f->toString() << ")"
<< endl;
      }
      else break;
      i++;
    }
    if (i > n) break;
  }
}

void print_cclassifier_performance(CascadeClassifier *cc, vector<float*> &positive_set,
vector<float*> &negative_set, int base_resolution)
```

```
{
  int ferror = 0;
  int nferror = 0;
  for (int i=0; i<positive_set.size(); i++)
  {
    if (cc->classify(positive_set[i], base_resolution, 0, 0, 0, 1) == false)
    {
      ferror++;
    }
  }

  for (int j=0; j<negative_set.size(); j++)
  {
    if (cc->classify(negative_set[j], base_resolution, 0, 0, 0, 1) == true)
    {
      nferror++;
    }
  }
  cout << "Cascade classifier performance:" << endl << "  FN = " << ferror << "/" <<
positive_set.size() << ", FP = " << nferror << "/" << negative_set.size() << endl;
}

StrongClassifier* AdaBoostLearning(CascadeClassifier *cc, vector<Feature*>
&feature_set, vector<float*> &positive_set, vector<float*> &negative_set,
vector<float*> &validation_set, float minfpr, float maxfnr, int base_resolution, bool
verbose) {
  WeakClassifier *bestwc, *wc;
  StrongClassifier *sc = new StrongClassifier();
  float wsum, minerror, wcerror, betat, cfpr;
  int i, j;
  vector<Feature*>::iterator it;

  float *weights = new float[positive_set.size()+negative_set.size()];

  for (i=0; i<positive_set.size(); i++) weights[i] = 1/float(2*positive_set.size());
  for (i=0; i<negative_set.size(); i++) weights[positive_set.size()+i] =
1/float(2*negative_set.size());

  cfpr = 1.0;
  float *fvalues = new float[positive_set.size()+negative_set.size()];
  while (cfpr > minfpr) {
```

```
    // Stop adding new weak classifiers if all negative samples are correctly
classified
    if (sc->fpr(negative_set, base_resolution) == 0) {
      if (verbose) cout << "  All training negative samples classified correctly. Could
not achieve validation target FPR (" << cfpr << " > " << minfpr << ") for this stage."
<< endl;
      break;
    }

    // Normalize weights[i]
    wsum = 0;
    for (i=0; i<(positive_set.size()+negative_set.size()); i++) wsum += weights[i];
    for (i=0; i<(positive_set.size()+negative_set.size()); i++) weights[i] =
weights[i]/wsum;

    // Select best weak classifier
    minerror = 1;
    bestwc = NULL;
    for (it = feature_set.begin(); it != feature_set.end(); ++it) {
      wc = new WeakClassifier(*it);
      for (i=0; i<positive_set.size(); i++) fvalues[i] = (*it)-
>getValue(positive_set[i], base_resolution, 0, 0);
      for (i=0; i<negative_set.size(); i++) fvalues[positive_set.size()+i] = (*it)-
>getValue(negative_set[i], base_resolution, 0, 0);
      wcerror = wc->find_optimum_threshold(fvalues, positive_set.size(),
negative_set.size(), weights);
      if (wcerror < minerror) {
        delete bestwc;
        bestwc = wc;
        minerror = wcerror;
      }
      else delete wc;
    }

    // Update sample weights
    betat = minerror/(1-minerror);
    for (i=0; i<positive_set.size(); i++) if (bestwc->classify(positive_set[i],
base_resolution, 0, 0, 0, 1) == 1) weights[i] = weights[i]*betat;
    for (i=0; i<negative_set.size(); i++) if (bestwc->classify(negative_set[i],
base_resolution, 0, 0, 0, 1) == -1) weights[positive_set.size()+i] =
weights[positive_set.size()+i]*betat;
```

```
    // Update current false positive ratio
    sc->add(bestwc, log(1/betat));
    sc->optimise_threshold(positive_set, base_resolution, maxfnr);
    if (validation_set.size() > 0) {
      cc->push_back(sc);
      cfpr = cc->fpr(validation_set);
      cc->pop_back();
    }
    else cfpr = sc->fpr(negative_set, base_resolution);

    if (verbose) {
      if (validation_set.size() > 0) cout << "    -> Added new Haar feature (Validation
set FPR=" << cfpr << ")" << endl;
      else cout << "    -> Added new Haar feature (FPR=" << cfpr << ")" << endl;
    }
  }
  delete[] weights;
  delete[] fvalues;

  return sc;
}

float* parse_float_string(string image, int width, int height) {
  float *img = new float[width*height];
  int x=0, y=0;
  float value;
  istringstream iss(image);

  while (iss >> value) {
    if (y == height) {
      cout << endl << "WARNING: image size differs from " << width << "x" << height <<
" (ignoring sample)" << endl;
      return NULL;
    }
    img[(y*width)+(x++)] = value;
    if (x == width) {
      x = 0;
      y++;
    }
  }
```

```
  if (x != 0 || y != height) {
    cout << endl << "WARNING: image size differs from " << width << "x" << height << "
(ignoring sample)" << endl;
    return NULL;
  }
  return img;
}

int* parse_int_string(string image, int width, int height) {
  int *img = new int[width*height];
  int x=0, y=0;
  int value;
  istringstream iss(image);

  while (iss >> value) {
    if (y == height) {
      cout << endl << "WARNING: image size differs from " << width << "x" << height <<
" (ignoring sample)" << endl;
      return NULL;
    }
    img[(y*width)+(x++)] = value;
    if (x == width) {
      x = 0;
      y++;
    }
  }
  if (x != 0 || y != height) {
    cout << endl << "WARNING: image size differs from " << width << "x" << height << "
(ignoring sample)" << endl;
    return NULL;
  }
  return img;
}

void merge_detections(vector<int*> detections) {
  int x1, y1, x2, y2, s1, s2;
  int minx, miny, maxx, maxy;
  for (int i=0; i<detections.size(); i++) {
    x1 = detections[i][0]; y1 = detections[i][1]; s1 = detections[i][2];
    for (int j=i+1; j<detections.size(); j++) {
      x2 = detections[j][0]; y2 = detections[j][1]; s2 = detections[j][2];
```

```
      if (j!=i && ((x1 < x2+s2) && (x2 < x1+s1) && (y1 < y2+s2) && (y2 < y1+s1))) {
        // There's overlapping between detections
        if (x1 > x2) {
          minx = x2;
          maxx = x1;
        }
        else {
          minx = x1;
          maxx = x2;
        }
        if (y1 > y2) {
          miny = y2;
          maxy= y1;
        }
        else {
          miny = y1;
          maxy = y2;
        }
        detections[i][0]=minx; detections[i][1]=miny; detections[i][2]=max(maxx-minx,
maxy-miny);
        detections.erase(detections.begin()+j);
        j=-1;
      }
    }
  }
}

void draw_square(png::image<png::rgb_pixel> *img, int x, int y, int size) {
  int thickness = 1+(size/100);
  png::rgb_pixel red;
  red.red = 255;
  red.green = 0;
  red.blue = 0;

  for (size_t i = x; i < x+size; ++i)
  {
    for (size_t j = y; j < y+thickness; ++j)
      img->set_pixel(i, j, red);

    for (size_t k = y+size-1; k > (y+size-1)-thickness; --k)
      img->set_pixel(i, k, red);
```

```
  }
  for (size_t l = y; l < y+size; ++l) {
    for (size_t m = x; m < x+thickness; ++m)
      img->set_pixel(m, l, red);

    for (size_t n = x+size-1; n > (x+size-1)-thickness; --n)
      img->set_pixel(n, l, red);
  }
}

png::image<png::rgb_pixel> detect_objects(png::image<png::rgb_pixel> img,
CascadeClassifier *cc, float fscale, float fincrement) {
  png::rgb_pixel p;
  float *gsimg, *iimg, *siimg;
  int i, j, a, b, increment;
  size_t x, y;
  int fnotfound = 0;
  float mean, stdev;
  int* detection;
  vector<int*> detections;

  // Convert RGB image to grayscale
  gsimg = new float[img.get_width()*img.get_height()];
  for (y = 0; y < img.get_height(); ++y) {
    for (x = 0; x < img.get_width(); ++x) {
      p = img.get_pixel(x, y);
      gsimg[(y*img.get_width())+x] = ((0.21*p.red)+(0.71*p.green)+(0.07*p.blue));
    }
  }

  // Calculate integral image and squared integral image
  iimg = integral_image(gsimg, img.get_width(), img.get_height());
  siimg = squared_integral_image(gsimg, img.get_width(), img.get_height());
  delete[] gsimg;

  // Run face detection on multiple scales
  int base_resolution = cc->getBaseResolution();
  while (base_resolution <= img.get_width() && base_resolution <= img.get_height()) {
    increment = base_resolution*fincrement;
    if (increment < 1) increment = 1;
```

```
    // Slide window over image
    for (i=0; (i+base_resolution)<=img.get_width(); i+=increment) {
      for (j=0; (j+base_resolution)<=img.get_height(); j+=increment) {
        // Calculate mean and std. deviation for current window
        mean=evaluate_integral_rectangle(iimg, img.get_width(), i, j, base_resolution,
base_resolution)/pow(base_resolution, 2);
        stdev = sqrt((evaluate_integral_rectangle(siimg, img.get_width(), i, j,
base_resolution, base_resolution)/pow(base_resolution, 2))-pow(mean, 2));

        // Classify window (post-normalization of feature values using mean and stdev)
        if (cc->classify(iimg, img.get_width(), i, j, mean, stdev) == true) {
          detection = new int[3];
          detection[0]=i; detection[1]=j; detection[2]=base_resolution;
          detections.push_back(detection);
        }
        else fnotfound++;
      }
    }

    cc->scale(fscale);
    base_resolution = cc->getBaseResolution();
  }

  // Merge overlapping detections
  merge_detections(detections);

  cout << detections.size() << " objects found (" << detections.size()+fnotfound << "
total subwindows checked)" << endl;
  for (std::vector<int*>::iterator it = detections.begin(); it != detections.end(); +
+it) {
    draw_square(&img, (*it)[0], (*it)[1], (*it)[2]);
  }

  return img;
}

int main(int argc, char** argv) {
  bool train = false;
  bool test = false;
  int base_resolution = 21;
  int csteps = 8;
```

```
  float scalefstep = 1.25;
  float slidefstep = 0.1;
  float strictness = 1;
  float maxfnr = 0.01;
  float Ftarget = 0.000001; // target overall false positive rate
  float *Fi; // target false positive rate for current classifier
  string positive_samples_url = "";
  string negative_samples_url = "";
  string validation_samples_url = "";
  string modelfile = "";
  vector<float*> positive_set, negative_set, validation_set;
  bool verbose = true;
  bool rotatenegative = false;
  bool mirrorpositive = false;
  bool use_validation = false;
  bool disable_normalization = false;
  int neg_samples_per_step = 0;
  int validation_samples = 0;
  float *auximg;
  float *auxiimg;

  if (argc < 3) {
    printUsage(argv[0]);
    return -1;
  }

  // Read input arguments
  for (int i=1; i<argc; i++) {
    if (strcmp(argv[i], "-t") == 0 || strcmp(argv[i], "--train") == 0) train = true;
    else if (strcmp(argv[i], "-m") == 0 || strcmp(argv[i], "--model") == 0) {
      modelfile = argv[i+1];
      i++;
    }
    else if (strcmp(argv[i], "--base-resolution") == 0) {
      base_resolution = stoi(argv[i+1]);
      i++;
    }
    else if (strcmp(argv[i], "--cascade-steps") == 0 || strcmp(argv[i], "-c") == 0) {
      csteps = stoi(argv[i+1]);
      i++;
    }
```

```
    else if (strcmp(argv[i], "--test") == 0) test = true;
    else if (strcmp(argv[i], "-o") == 0 || strcmp(argv[i], "--output") == 0) {
      modelfile = argv[i+1];
      i++;
    }
    else if (strcmp(argv[i], "--enable-rotation") == 0) {
      rotatenegative = true;
    }
    else if (strcmp(argv[i], "--enable-mirroring") == 0) {
      mirrorpositive = true;
    }
    else if (strcmp(argv[i], "--maxfnr-per-step") == 0) {
      maxfnr = stof(argv[i+1]);
      i++;
    }
    else if (strcmp(argv[i], "--negative-samples-per-step") == 0) {
      neg_samples_per_step = stoi(argv[i+1]);
      i++;
    }
    else if (strcmp(argv[i], "--validation-samples") == 0) {
      use_validation = true;
      validation_samples = stoi(argv[i+1]);
      i++;
    }
    else if (strcmp(argv[i], "--target-fpr") == 0) {
      Ftarget = stof(argv[i+1]);
      i++;
    }
    else if (strcmp(argv[i], "--disable-norm") == 0) {
      disable_normalization = true;
    }
    else if (strcmp(argv[i], "--scale-step") == 0) {
      scalefstep = stof(argv[i+1]);
      i++;
    }
    else if (strcmp(argv[i], "--slide-step") == 0) {
      slidefstep = stof(argv[i+1]);
      i++;
    }
    else if (strcmp(argv[i], "-s") == 0 || strcmp(argv[i], "--strictness") == 0) {
      strictness = stof(argv[i+1]);
```

```
      i++;
    }
    else if (positive_samples_url.compare("") == 0) positive_samples_url = argv[i];
    else if (negative_samples_url.compare("") == 0) negative_samples_url = argv[i];
    else if (validation_samples_url.compare("") == 0) {
      use_validation = true;
      validation_samples_url = argv[i];
    }
    else {
      printUsage(argv[0]);
      return -1;
    }
  }

  // Training mode
  if (train) {
    if (modelfile.compare("") == 0 || positive_samples_url.compare("") == 0 ||
negative_samples_url.compare("") == 0) {
      cout << "usage for train mode: " << argv[0] << " --train --output MODELFILE [--
maxfnr-per-step F --cascade-steps N --target-fpr T --negative-samples-per-step N
--validation-samples V --enable-mirroring --enable-rotation] --base-resolution X
POS_SAMPLES_FILE NEG_SAMPLES_FILE [VALIDATION_NEG_SAMPLES_FILE]" << endl;
      return -1;
    }

    if (csteps < 1) {
      cout << "Error: the number of cascade steps must be greater than 0 (" << csteps
<< " selected)" << endl;
      return -1;
    }
    // Calculate target FPR per step according to csteps and Ftarget
    Fi = new float[csteps];
    if (csteps == 1) Fi[0]=Ftarget;
    else {
      Fi[0]=0.5;
      if (csteps == 2) Fi[1]=Ftarget/Fi[0];
      else {
        Fi[1]=0.25;
        float initialfprperstep = Fi[0]*Fi[1];
        float fprperstep = pow((Ftarget/initialfprperstep), 1/float(csteps-2));
        for (int cs = 2; cs < csteps; cs++) Fi[cs] = fprperstep;
```

```
      }
    }

    // Print cascade classifier training details
    if (verbose) {
      cout << "Training cascade classifier with the following parameters:" << endl <<
endl << "  Maximum number of cascade steps: " << csteps << endl << "  Target FPR: " <<
Ftarget << endl << "  Maximum FPR per cascade step: ";
      for (int k=0; k<csteps; k++) cout << Fi[k] << " ";
      cout << endl << "  Maximum FNR per cascade step: " << maxfnr << endl;
      if (neg_samples_per_step != 0) cout << "  Negative training samples per step: "
<< neg_samples_per_step << endl;
      cout << "  Cascade model file generated: " << modelfile << endl << endl;
    }

    // Load faces file
    ifstream pfile(positive_samples_url.c_str());
    string sline = "";
    if (verbose) {
      if (mirrorpositive) cout << "Loading positive samples file (+ vertical mirror)
from \"" << positive_samples_url << "\" ... " << flush;
      else cout << "Loading positive samples from \"" << positive_samples_url <<
"\" ... " << flush;
    }
    while (getline(pfile, sline)) {
      if (disable_normalization) positive_set.push_back(parse_float_string(sline,
base_resolution, base_resolution));
      else positive_set.push_back(normalize(parse_float_string(sline, base_resolution,
base_resolution), base_resolution, base_resolution));
    }
    pfile.close();
    // Duplicate positive samples set by adding the vertical mirror
    if (mirrorpositive) add_vertical_mirror(positive_set, base_resolution,
base_resolution);
    if (verbose) cout << "Done (" << positive_set.size() << ")" << endl;

    // Set number of negative samples for training per cascade step to faces.size() if
not specified
    if (neg_samples_per_step == 0) neg_samples_per_step = positive_set.size();

    // Load negative samples file (read done incrementally step by step)
```

```
    ifstream nffile(negative_samples_url.c_str());
    sline = "";

    // Load validation file if available, otherwise take first
validation_samples/neg_samples_per_step negative samples from negative samples set
    if (use_validation && validation_samples_url.compare("") == 0) {
      if (verbose) {
        if (validation_samples == 0) validation_samples = neg_samples_per_step;
        cout << "Loading " << validation_samples << " validation samples from \"" <<
negative_samples_url << "\" ... " << flush;
      }
      while (getline(nffile, sline)) {
        auximg = parse_float_string(sline, base_resolution, base_resolution);
        if (auximg != NULL) {
          if (disable_normalization) validation_set.push_back(auximg);
          else validation_set.push_back(normalize(auximg, base_resolution,
base_resolution));
          if (validation_set.size() == validation_samples) break;
        }
      }
      if (verbose) cout << "Done (" << validation_set.size() << ")" << endl;
    }
    else if (use_validation) {
      ifstream vfile(validation_samples_url.c_str());
      if (verbose) {
        if (validation_samples == 0) cout << "Loading validation samples from \"" <<
validation_samples_url << "\" ... " << flush;
        else cout << "Loading " << validation_samples << " validation samples from \""
<< validation_samples_url << "\" ... " << flush;
      }
      while (getline(vfile, sline)) {
        auximg = parse_float_string(sline, base_resolution, base_resolution);
        if (auximg != NULL) {
          if (disable_normalization) validation_set.push_back(auximg);
          else validation_set.push_back(normalize(auximg, base_resolution,
base_resolution));
        }
        if (validation_samples != 0 && validation_set.size() == validation_samples)
break;
      }
      vfile.close();
```

```
      if (verbose) cout << "Done (" << validation_set.size() << ")" << endl;
    }

    // Compute integral image for all images in positive and validation sets
    if (verbose) cout << "Computing integral image for all images ... " << flush;
    int psize = positive_set.size();
    for (int im=0; im<psize; im++) {
      positive_set.push_back(integral_image(positive_set[im], base_resolution,
base_resolution));
      positive_set.erase(positive_set.begin());
    }
    int vsize = validation_set.size();
    for (int vim=0; vim<vsize; vim++) {
      validation_set.push_back(integral_image(validation_set[vim], base_resolution,
base_resolution));
      validation_set.erase(validation_set.begin());
    }
    if (verbose) cout << "Done" << endl;

    // Generate complete feature set for BASERESxBASERES resolution
    vector<Feature*> feature_set = generate_feature_set(base_resolution);
    if (verbose) cout << endl << "Total number of features: " << feature_set.size() <<
endl;

    int n, fdel, nfdel, invalidsamples, validsamples, rotation;
    float maxfpr = 1.0;
    StrongClassifier *sc;

    // Build cascade classifier
    CascadeClassifier *cc = new CascadeClassifier(base_resolution);
    for (int k=0; k<csteps; k++) {
      if (verbose) cout << endl << "Building cascade classifier (step " << (k+1) << "
of " << csteps << "):" << endl << "  Loading missclassified negative samples ... " <<
flush;

      // Add negative samples to set until negative_set.size()=neg_samples_per_step
      if (negative_set.size() < neg_samples_per_step) {
        invalidsamples = 0;
        validsamples = 0;
        while (getline(nffile, sline)) {
          auximg = parse_float_string(sline, base_resolution, base_resolution);
```

```
          if (auximg != NULL) {
            validsamples++;
            if (!disable_normalization) auximg = normalize(auximg, base_resolution,
base_resolution);
            if (rotatenegative) {
              for (rotation = 0; rotation < 4; rotation++) {
                auxiimg = integral_image(auximg, base_resolution, base_resolution);
                if (cc->classify(auxiimg, base_resolution, 0, 0, 0, 1) == true) {
                  negative_set.push_back(auxiimg);
                  if (negative_set.size() == neg_samples_per_step) break;
                }
                if (rotation < 3) auximg = rotate_90deg(auximg, base_resolution,
base_resolution);
              }
              if (negative_set.size() == neg_samples_per_step) break;
            }
            else {
              auximg = integral_image(auximg, base_resolution, base_resolution);
              if (cc->classify(auximg, base_resolution, 0, 0, 0, 1) == true) {
                negative_set.push_back(auximg);
                if (negative_set.size() == neg_samples_per_step) break;
              }
            }
          }
          else invalidsamples++;
        }
      }

      if (negative_set.size() > 0) {
        // Run AdaBoost algorithm to select best Haar features until Fi[0]*...*Fi[k]
(FPR for current step) is met on validation set or Fi[k] is met on negative set
        if (use_validation) maxfpr = maxfpr*Fi[k];
        else maxfpr = Fi[k];
        if (verbose) cout << "Done (" << validsamples+invalidsamples << " negative
samples consumed)" << endl << "  Running AdaBoost algorithm (target FPR = " << maxfpr
<< "):" << endl;
        sc = AdaBoostLearning(cc, feature_set, positive_set, negative_set,
validation_set, maxfpr, maxfnr, base_resolution, verbose);
        cc->push_back(sc);
        if (verbose) {
```

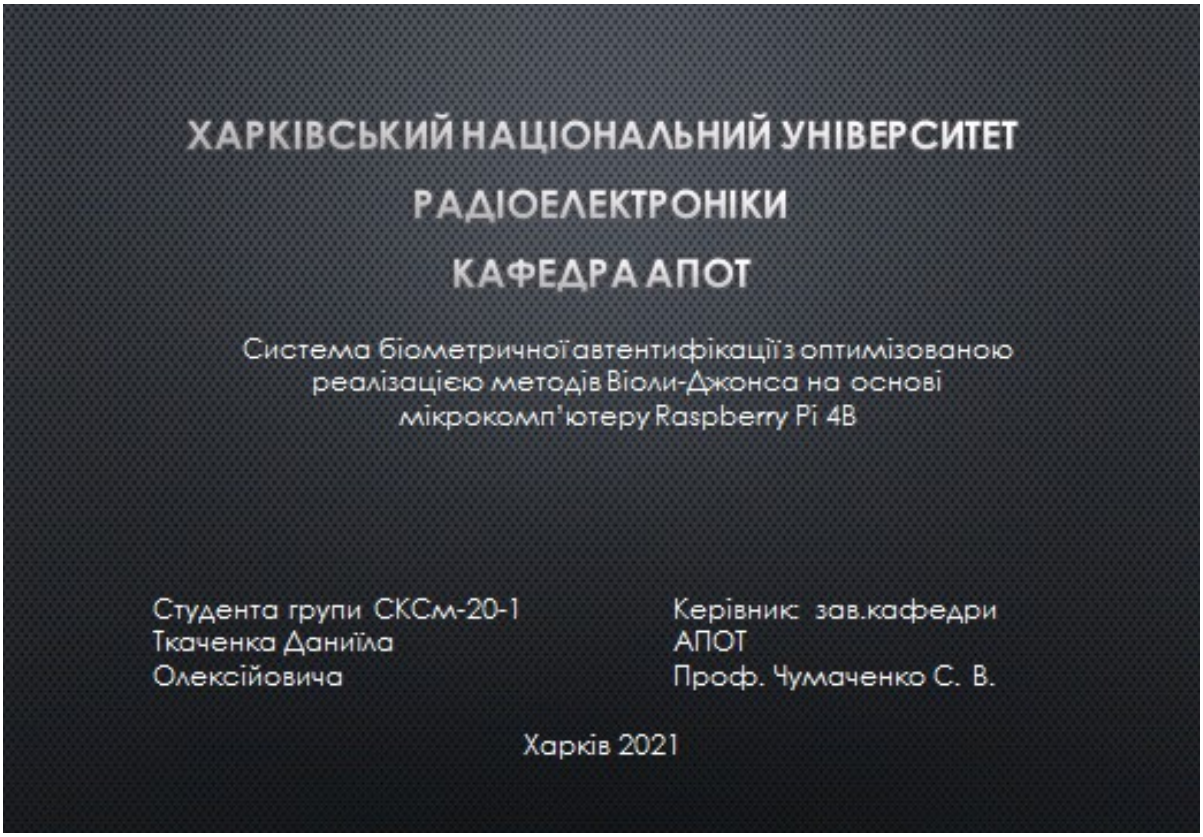
```
          if (use_validation) print_cclassifier_performance(cc, positive_set,
validation_set, base_resolution);
          else print_cclassifier_performance(cc, positive_set, negative_set,
base_resolution);
          cout << "  Removing false detections from training set ... " << flush;
        }
        // Remove false detections from training set
        fdel = 0; nfdel=0;
        for (n=0; n<negative_set.size(); n++) {
          if (sc->classify(negative_set[n], base_resolution, 0, 0, 0, 1) == false) {
            negative_set.erase(negative_set.begin()+n);
            n--;
            nfdel++;
          }
        }
        for (n=0; n<positive_set.size(); n++) {
          if (sc->classify(positive_set[n], base_resolution, 0, 0, 0, 1) == false) {
            positive_set.erase(positive_set.begin()+n);
            n--;
            fdel++;
          }
        }
        if (verbose) cout << "Done (" << nfdel << " negative samples removed)" << endl;
      }
      else {
        if (verbose) cout << endl << "Not enough negative samples missclassified by the
current cascade classifier for training a new step" << endl;
        break;
      }
    }
    nffile.close();
    if (use_validation) print_cclassifier_performance(cc, positive_set, validation_set,
base_resolution);
    if (verbose) cout << "Saving cascade classifier model file to \"" << modelfile <<
"\" ... " << flush;
    cc->save(modelfile);
    if (verbose) cout << "Done" << endl;
  }

  // Test cascade classifier model
  else if (test) {
```

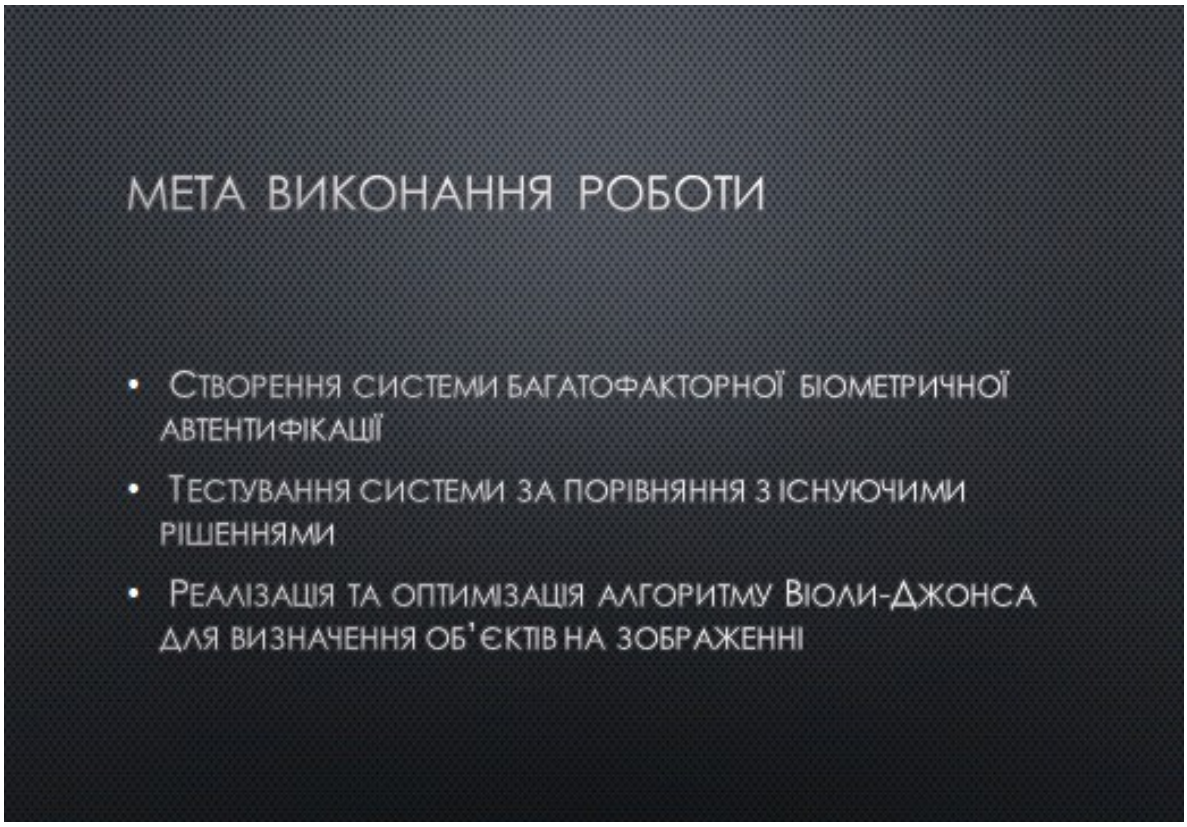
```
    if (modelfile.compare("") == 0 || positive_samples_url.compare("") == 0 ||
negative_samples_url.compare("") == 0) {
      cout << "usage for test mode: " << argv[0] << " --test -m MODELFILE [--enable-
rotation --enable-mirroring] --base-resolution X POS_SAMPLES_FILE NEG_SAMPLES_FILE" <<
endl;
      return -1;
    }

    CascadeClassifier *c = new CascadeClassifier(modelfile);
    c->strictness(strictness);
    // Load faces file
    ifstream pfile(positive_samples_url.c_str());
    string sline = "";
    if (verbose) {
      if (mirrorpositive) cout << "Loading positive samples file (+ vertical
mirror) ... " << flush;
      else cout << "Loading positive samples file ... " << flush;
    }

    while (getline(pfile, sline)) {
      if (disable_normalization) positive_set.push_back(parse_float_string(sline,
base_resolution, base_resolution));
      else positive_set.push_back(normalize(parse_float_string(sline, base_resolution,
base_resolution), base_resolution, base_resolution));
    }

    pfile.close();
    // Duplicate faces set by adding the vertical mirror
    if (mirrorpositive) add_vertical_mirror(positive_set, base_resolution,
base_resolution);
    if (verbose) cout << "OK (" << positive_set.size() << ")" << endl;

    // Load no-faces file
    ifstream nffile(negative_samples_url.c_str());
    sline = "";
    if (verbose) {
      if (rotatenegative) cout << "Loading negative samples file (+ 90, 180 and 270
deg.) ... " << flush;
      else cout << "Loading negative samples file ... " << flush;
    }
    while (getline(nffile, sline))
```

```
    {
      auximg = parse_float_string(sline, base_resolution, base_resolution);
      if (auximg != NULL) {
        if (disable_normalization) negative_set.push_back(auximg);
        else negative_set.push_back(normalize(auximg, base_resolution,
base_resolution));
      }
    }

    nffile.close();
    // Enlarge non-faces set by rotating non-face images by 90, 180 and 270 degrees
    if (rotatenegative) add_rotated_images(negative_set, base_resolution,
base_resolution);
    if (verbose) cout << "OK (" << negative_set.size() << ")" << endl << endl;

    // Compute integral image for all images
    int psize = positive_set.size();
    for (int im=0; im<psize; im++) {
      positive_set.push_back(integral_image(positive_set[0], base_resolution,
base_resolution));
      positive_set.erase(positive_set.begin());
    }
    int nsize = negative_set.size();
    for (int nim=0; nim<nsize; nim++) {
      negative_set.push_back(integral_image(negative_set[0], base_resolution,
base_resolution));
      negative_set.erase(negative_set.begin());
    }

    cout << "Cascade classifier total steps: " << c->getStrongClassifiers().size() <<
endl << endl;
    cout << "Cascade classifier best 3 features:" << endl;
    print_cclassifier_best_features(c, 3);
    cout << endl;
    print_cclassifier_performance(c, positive_set, negative_set, base_resolution);
  }

  // Face detection over image
  else {
    if (modelfile.compare("") == 0 || positive_samples_url.compare("") == 0)
    {
```

```
      cout << "usage for detection mode: " << argv[0] << " -m MODELFILE [--scale-step X
--slide-step Y --strictness S] IMAGE.PNG" << endl;
      return -1;
    }

    // Load cascade classifier model
    CascadeClassifier *c = new CascadeClassifier(modelfile);
    c->strictness(strictness);
    // Load PNG image

    png::image<png::rgb_pixel> image(positive_samples_url);

    cout << "Detecting objects in image \"" << positive_samples_url << "\" ... " <<
flush;

    png::image<png::rgb_pixel> res = detect_objects(image, c, scalefstep, slidefstep);
    res.write(positive_samples_url.substr(0, positive_samples_url.size()-
4)+".lfobject.png");
  }

  return 0;
                                          }
```

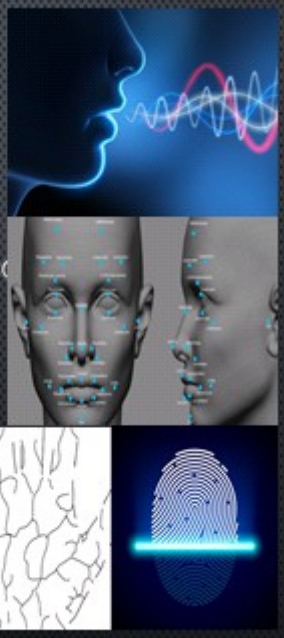
ДОДАТОК Б


ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ
КАФЕДРА АПОТ
Система біометричної автентифікації з оптимізованою реалізацією методів Віоли-Джонса на основі мікрокомп'ютеру Raspberry Pi 4B
Студента групи СКСм-20-1
Ткаченка Даниїла
Олексійовича
Керівник зав.кафедри
АПОТ
Проф. Чумаченко С. В.
Харків 2021


МЕТА ВИКОНАННЯ РОБОТИ
• Створення системи багатофакторної біометричної автентифікації
• Тестування системи за порівняння з існуючими рішеннями
• Реалізація та оптимізація алгоритму Віоли-Джонса для визначення об'єктів на зображенні

АКТУАЛЬНІСТЬ

- Біометричні ознаки які використовуються
- у сучасних системах:
- - Геометрія обличчя
- - Відбитки пальців
- - Голос
- - Схема вен
- - Радужна оболонка очей



МЕТА ВИКОНАННЯ РОБОТИ

- Створення системи багатофакторної біометричної автентифікації
- Тестування системи за порівняння з існуючими рішеннями
- Реалізація та оптимізація алгоритму Віоли-Джонса для визначення об'єктів на зображенні

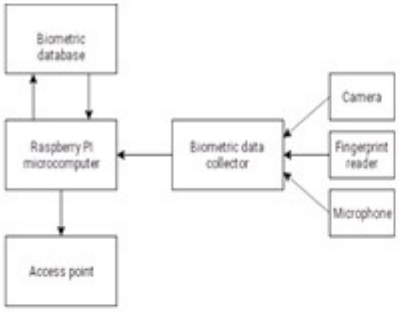
ОГЛЯД ВИКОРИСТАНИХ АПАРАТНИХ КОМПОНЕНТІВ

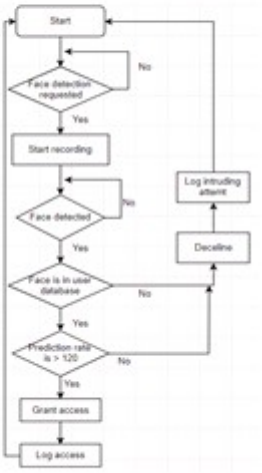
- Зовнішній вигляд пристрою
- В даний набір входить:
- - Мікрокомп'ютер Raspberry Pi;
- - Зчитувач відбитків пальців;
- - Мікрофон;
- - Камера;
- - Елемент живлення;
- - Блок живлення.

ОГЛЯД ВИКОРИСТАНИХ ПРОГРАМНИХ КОМПОНЕНТІВ

- - Як мову для розробки була використана мова C++
- - Бібліотека OpenCV для розпізнавання обличчя
- - Бібліотека Adafruit fingerprint для роботи зі сканером
- - Бібліотека CMU Sphinx для роботи з голосом

БЛОК СХЕМА СИСТЕМИ ТА АЛГОРИТМ РОБОТИ



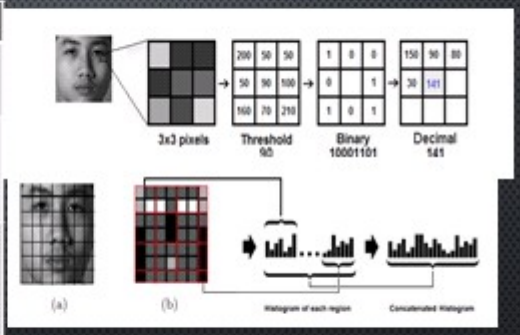


ЗОВНІШНІЙ ВИГЛЯД ПРИСТРОЮ

- В РАМКАХ ПРОЕКТУ ДЛЯ ТЕСТУВАННЯ БУВ ЗІБРАНИЙ МАКЕТ СИСТЕМИ З ВИКОРИСТАННЯМ СКАНЕРУ ВІДБИТКІВ ТА КАМЕРОЮ LOGITECH ЯКА МІСТИТЬ У СОБІ МІКРОФОН

АЛГОРИТМ LBPH ДЛЯ РОЗПІЗНАВАННЯ ОБЛИЧЧЯ

Переваги	Недоліки
Безкоштовний	Невисока точність
Відкритий	Критично реагує на світло
Висока швидкість	Висока кількість фото для навчання
Використання кольорових фото	Відносно застарілий

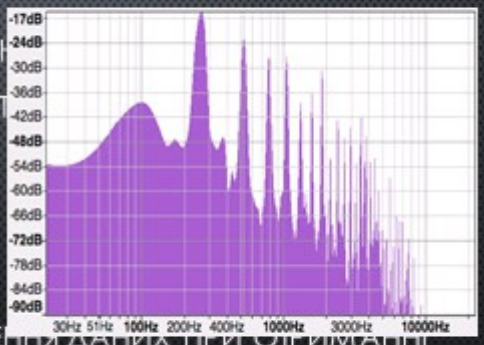


MINUTIAE POINTS ДЛЯ РОЗПІЗНАВАННЯ ВІДБИТКІВ ПАЛЬЦІВ

- Переваги:
- - Висока точність
  - Висока швидкість
- - Мінімальні ресурсні затра
- Недоліки:
- - Простота підроблення вхідних даних при отриманні негативів відбитків клієнта

РОЗПІЗНАВАННЯ ГОЛОСУ

- Переваги:
- - Варіативність опорі
- - Простота використ
- - Швидкість роботи
- Недоліки:
- - Низька точність
- - Простота підроблення даних при отриманні запису голоса



РЕАЛІЗАЦІЯ ТА ОПТИМІЗАЦІЯ АЛГОРИТМУ ВІОЛИ-ДЖОНСА

В процесі оптимізації були використані такі методи як:

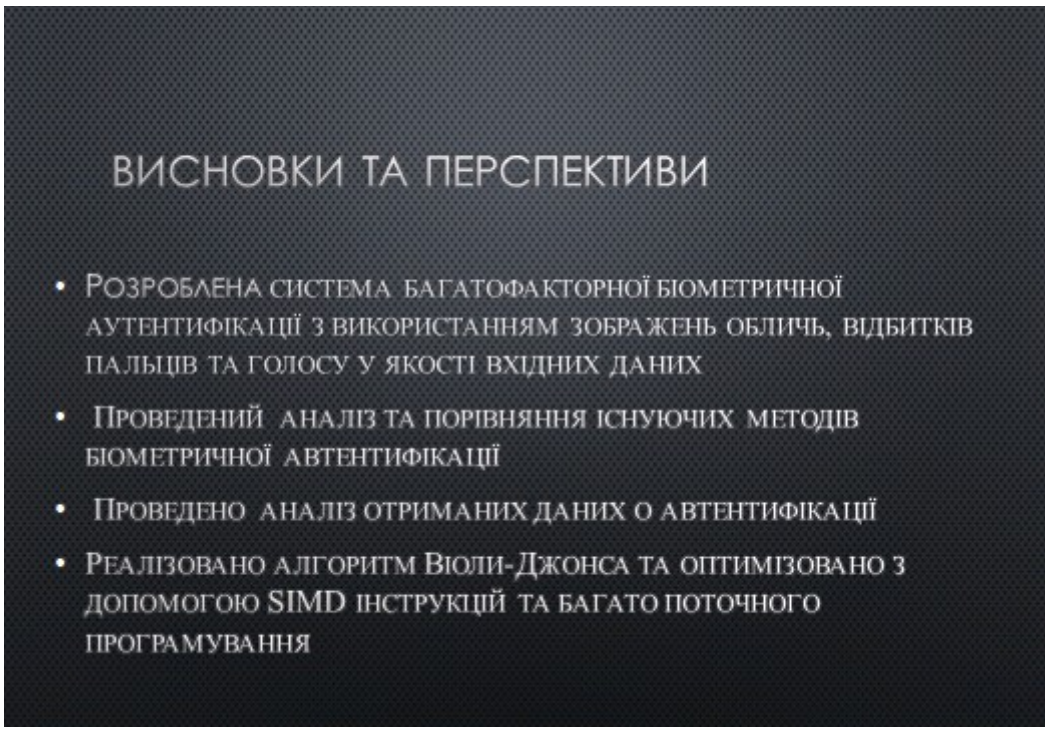
- Використання SIMD інструкцій для оптимізації роботи з кешем процесору
- Використання багато поточності для оптимізації паралельних обчислювань

РЕЗУЛЬТАТИ ВИМІРЮВАНЬ

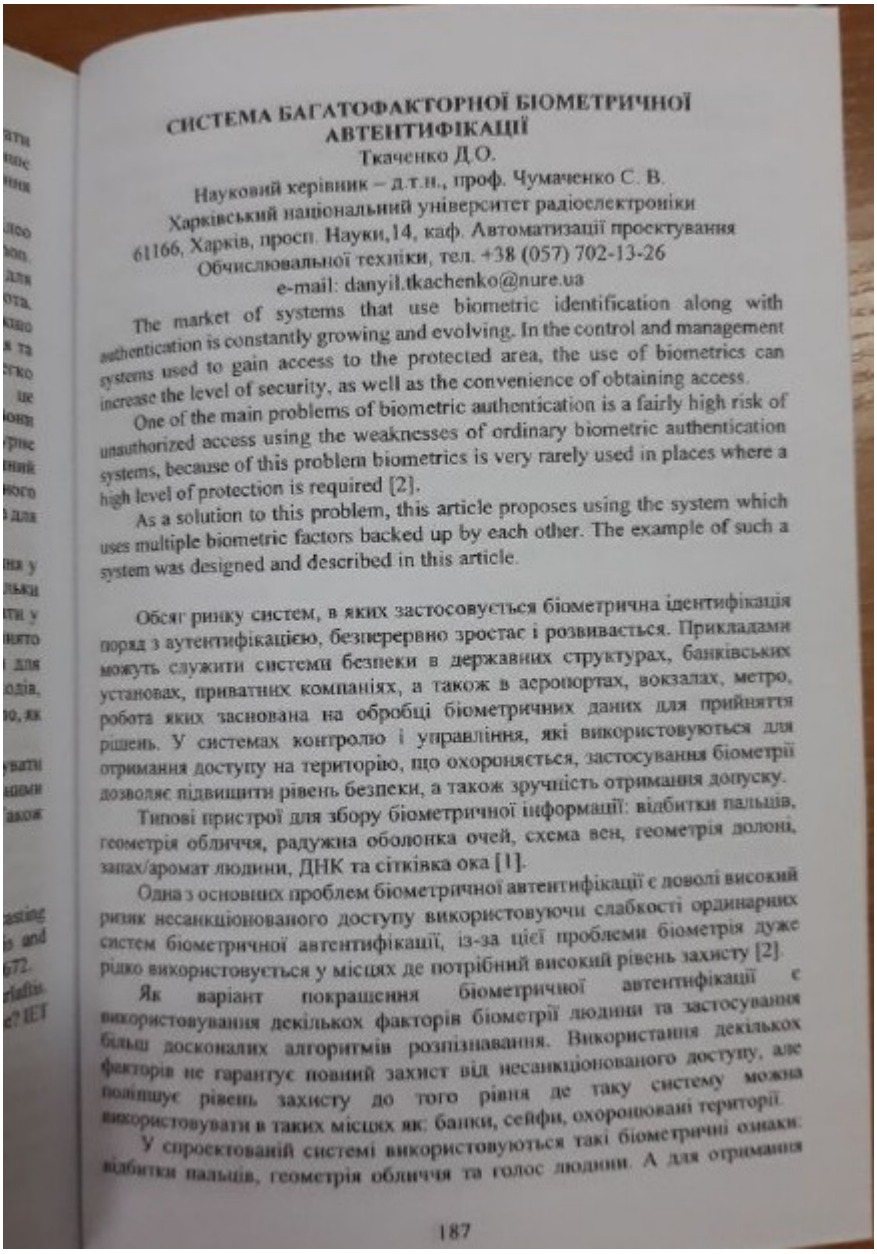
Біометрична ознака	Точність		Час обробки	Простота використання
	FRR	FAR	мс	
Геометрія обличчя	~ 6%	~ 0,1%	~750мс	Висока
Відбитки пальців	~ 1%	~ 0,00002 %	~400мс	Середня
Голос	~ 3%	~ 0,1%	~1500мс	Висока
Система ББІ	~ 1%	~ 0,00001 %	~2600мс	Середня

РЕЗУЛЬТАТИ ВИМІРЮВАНЬ

Реалізація	Точність		Середній час обробки
	FRR	FAR	мс
Проектна реалізація	~ 15%	~ 5%	~500мс
OpenCV	~ 10%	~ 3%	~750мс


ВИСНОВКИ ТА ПЕРСПЕКТИВИ
• Розроблена система багатофакторної біометричної аутентифікації з використанням зображень обличь, відбитків пальців та голосу у якості вхідних даних
• Проведений аналіз та порівняння існуючих методів біометричної автентифікації
• Проведено аналіз отриманих даних о автентифікації
• Реалізовано алгоритм Віоли-Джонса та оптимізовано з допомогою SIMD інструкцій та багато поточного програмування

ДОДАТОК В



СИСТЕМА БАГАТОФАКТОРНОЇ БІОМЕТРИЧНОЇ АВТЕНТИФІКАЦІЇ

Ткаченко Д.О.
Науковий керівник – д.т.н., проф. Чумаченко С. В.
Харківський національний університет радіоелектроніки
61166, Харків, просп. Науки,14, каф. Автоматизації проектування
Обчислювальної техніки, тел. +38 (057) 702-13-26
e-mail: danyil.tkachenko@nure.ua

The market of systems that use biometric identification along with authentication is constantly growing and evolving. In the control and management systems used to gain access to the protected area, the use of biometrics can increase the level of security, as well as the convenience of obtaining access.

One of the main problems of biometric authentication is a fairly high risk of unauthorized access using the weaknesses of ordinary biometric authentication systems, because of this problem biometrics is very rarely used in places where a high level of protection is required [2].

As a solution to this problem, this article proposes using the system which uses multiple biometric factors backed up by each other. The example of such a system was designed and described in this article.

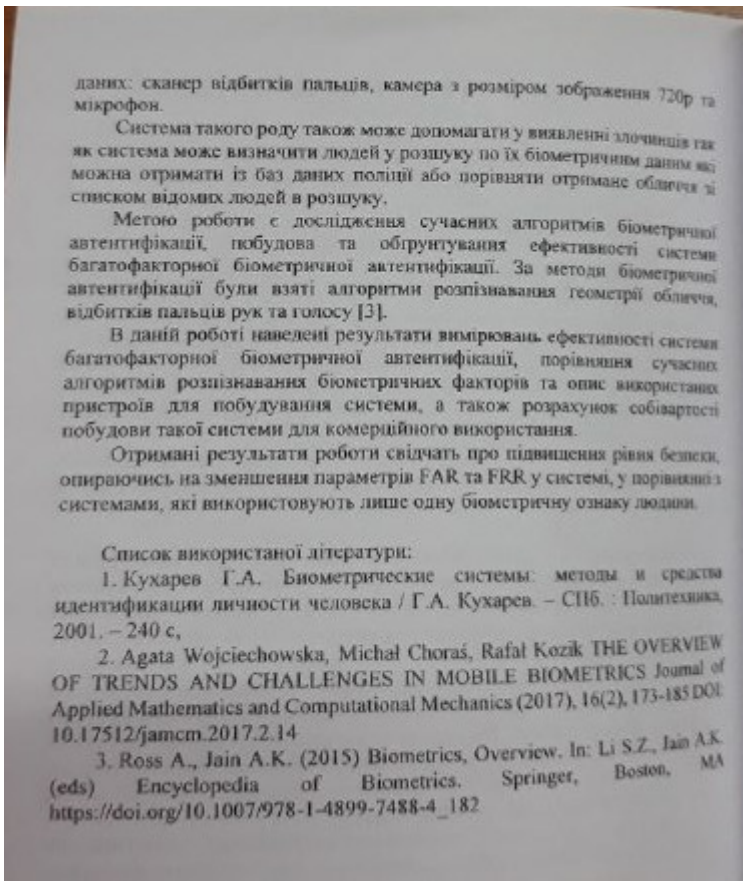
Обсяг ринку систем, в яких застосовується біометрична ідентифікація поряд з аутентифікацією, безперервно зростає і розвивається. Прикладами можуть служити системи безпеки в державних структурах, банківських установах, приватних компаніях, а також в аеропортах, вокзалах, метро, робота яких заснована на обробці біометричних даних для прийняття рішень. У системах контролю і управління, які використовуються для отримання доступу на територію, що охороняється, застосування біометрії дозволяє підвищити рівень безпеки, а також зручність отримання допуску.

Типові пристрої для збору біометричної інформації: відбитки пальців, геометрія обличчя, радужна оболонка очей, схема вен, геометрія долоні, запах/аромат людини, ДНК та сітківка ока [1].

Одна з основних проблем біометричної автентифікації є доволі високий ризик несанкціонованого доступу використовуючи слабкості ординарних систем біометричної автентифікації, із-за цієї проблеми біометрія дуже рідко використовується у місцях де потрібний високий рівень захисту [2].

Як варіант покращення біометричної автентифікації є використовування декількох факторів біометрії людини та застосування більш досконалих алгоритмів розпізнавання. Використання декількох факторів не гарантує повний захист від несанкціонованого доступу, але поліпшує рівень захисту до того рівня де таку систему можна використовувати в таких місцях як: банки, сейфи, охоронювані території.

У спроектованій системі використовуються такі біометричні ознаки: відбитки пальців, геометрія обличчя та голос людини. А для отримання



даних: сканер відбитків пальців, камера з розміром зображення 720p та мікрофон.

Система такого роду також може допомагати у виявленні злочинців так як система може визначити людей у розшуку по їх біометричним даним які можна отримати із баз даних поліції або порівняти отримане обличчя зі списком відомих людей в розшуку.

Метою роботи є дослідження сучасних алгоритмів біометричної автентифікації, побудова та обґрунтування ефективності системи багатофакторної біометричної автентифікації. За методи біометричної автентифікації були взяті алгоритми розпізнавання геометрії обличчя, відбитків пальців рук та голосу [3].

В даній роботі наведені результати вимірювань ефективності системи багатофакторної біометричної автентифікації, порівняння сучасних алгоритмів розпізнавання біометричних факторів та опис використаних пристроїв для побудування системи, а також розрахунок собівартості побудови такої системи для комерційного використання.

Отримані результати роботи свідчать про підвищення рівня безпеки, опираючись на зменшення параметрів FAR та FRR у системі, у порівнянні з системами, які використовують лише одну біомстричну ознаку людини.

Список використаної літератури:

1. Кухарев Г.А. Биометрические системы: методы и средства идентификации личности человека / Г.А. Кухарев. – СПб. : Политехника, 2001. – 240 с,

2. Agata Wojciechowska, Michał Choraś, Rafał Kozik THE OVERVIEW OF TRENDS AND CHALLENGES IN MOBILE BIOMETRICS Journal of Applied Mathematics and Computational Mechanics (2017), 16(2), 173-185 DOI: 10.17512/jamcm.2017.2.14

3. Ross A., Jain A.K. (2015) Biometrics, Overview. In: Li S.Z., Jain A.K. (eds) Encyclopedia of Biometrics. Springer, Boston, MA https://doi.org/10.1007/978-1-4899-7488-4_182

Відомість кваліфікаційної роботи

№	Позначення	Найменування	Дод. відомості
		Текстові документи	
1	ГЮІК. 46741Х.020 ПЗ	Пояснювальна записка	62с.
		Графічні документи	
2		Слайд-презентація	15 слайдів
		Носії інформації	
3		Компакт-диск (CD)	1 шт.

Змін.	Арк.	Номер докум.	Підп.	Дата	ГЮІК. 46741Х.020 ВД			
Розроб.		Ткаченко Д.О.			Система біометричної автентифікації з оптимізованою реалізацією методів Віоли-Джонса на основі мікрокомп'ютеру Raspberry Pi 4B	Літ	Аркуш	Аркушів
Перевір.		Шкіль О.С.				У		
Н.контр.		Шкіль О.С.			Відомість кваліфікаційної роботи	ХНУРЕ Кафедра АПОТ		
Затв.		Чумаченко С.В.						