**Assignment 1:**

**Gesture Recognition**

**Jason Mann**

**2/18/2013**

**COMS 4735: Visual Interfaces for Computers**

Table of Contents

**Code And Documentation 3**

**Readme.Txt 3**

**Main.cpp 6**

**MaskSelection.h 7**

**MaskSelection.cpp 9**

**SkinDetectController.h 15**

**SkinDetectController.cpp 17**

**SkinDetector.h 18**

**SkinDetector.cpp 19**

**ColorHistogram.h 20**

**GestureDetector.h 24**

**GestureDetector.cpp 26**

**Hand.h 31**

**Password.h 34**

**Application Views 36**

**Possible Hand Gestures 37**

**10 Example Sequences 38**

**Sequence 1 38**

**Sequence 2 39**

**Sequence 3 40**

**Sequence 4 41**

**Sequence 5 43**

**Sequence 6 44**

**Sequence 7 45**

**Sequence 8\*\*\* 46**

**Sequence 9\*\*\* 47**

**Sequence 10\*\*\* 48**

# Code and Documentation

Readme.txt

Creator: Jason Carlisle Mann (on2valhalla | jcm2207@columbia.edu)

Updated: 2013.02.18

------------

INTRODUCTION

------------

This project was designed for an assignment in COMS 4735 Visual Interfaces for

Computers under the supervision of Prof. John Kender at Columbia University in

Spring 2013.

The goal of this project was to filter out skin regions in imagery (static or

motion), detect regions belonging to hands, and then interpret specific pre-

defined gestures from the captures, as part of a password grammar.

The project is coded in C++, relying on OpenCV 2.4.3 for computer vision

algorithms, and QT 4.8 (and QT Creator) for the design of the GUI forms. Some

code and inspiration was found in the OpenCV2 Cookbook by Robert Laganiere,

specifically the design of the SkinDetector and Controller classes (though not

necessarily the implementation).

The skin regions are differentiated from the background by manual configuration

of an HSV thresholding mask (setting min and max for each value) through the use

of sliders on an introductory form. While photos can be thresholded, they are

not implemented for use in the password grammar. The masks are applied using

cv::inRange() to provide a binary image of skin regions, and that image is then

blurred and closed (eroded()/dilate()). Once the correct mask is found, you

proceed to the detection form, which begins looking for hand gestures.

Immediately, any faces in the frame are identified and eliminated from

consideration, using OpenCV's built-in Haar cascade classifier

(frontalface\_alt\_tree). Any other skin regions in the frame are considered to be

hands, so a long sleeve shirt is required. These regions are captured as

contours using cv::findContours. The gestures used for the passwords are

captured at regular intervals (hard coded to 3 seconds) to allow the user time to

configure the next gesture.

I chose five gestures (or absence of) for my grammar, a closed fist (FIST),

single finger vertical point (POINT), and closed finger extended palm (PALM), no

detected gestures (NONE), and a skin region that was unidentified (UNK). These

are differentiated based on ratios (width/height) of their respective upright

and minimum fit bounding rectangles, as well as the general mass of the moment

of the contour points (see cv::Moment::m00). Any number of hand gestures can be

captured in the frame, although a maximum of 2 is used for the password grammar.

If there are two gestures in the capture, there is no differentiation between

them, they are viewed as a non-ordered set. The passwords are hard coded into the

PasswordCheck class as a tree of conditional statements, though this could

easily be expanded to a search tree. The password sequences are all 3 sequences

long (although they can have up to two gestures per sequence). There are two

special sequences different from the passwords, two PALMs in one frame, and a

sequence of 3 captures with no gesture detected. Two PALMs signifies that the

password attempt is complete, and to check against the grammar. This check also

occurs automatically if a sequence of 6 uninterrupted captures is encountered. If

the 3 first captures are all NONE or UNK, the grammar restarts with the 4th

capture.

The hard coded accepted password sequences are as follows:

(Capture 1 | Capture 2 | Capture 3)

1. FIST | FIST | FIST

2. POINT | FIST | PALM

3. PALM | (NONE OR UNK) | PALM

4. (PALM & POINT) | (FIST & FIST) | (POINT & FIST)

Two special sequences:

1. (PALM & PALM)

>immediately check the recorded sequence preceding

2. (NONE OR UNK) | (NONE OR UNK) |(NONE OR UNK)

>reset, and check for one of the passwords listed

>above starting with the 4th capture

Upon entering a successful password, the system announces in the text box

PASSWORD ACCEPTED, and on failure: INTRUDER.... INTRUDER....

-----------

ENVIRONMENT

-----------

Due to creating a mask selection form, I tested in multiple locations and was

able to achieve ~70% accuracy in most locations. Those that fared worse were

generally sunlit areas, areas with uni-directional lighting, and areas that

have flesh colored tones, or light birch wood everywhere (like the Science

and Engineering Library). In a controlled environment, I have seen up to ~90%

success, though with a system trained towards my hands. All in all, I attempted

to eliminate the environment concerns from this project.

However, because of the specific geometric data used, a long sleeve shirt is

required for accurate captures. Also, the recommended distance for the built-

in MacBook Air iSight webcam (late 2011) is between 2.5 and 4 feet. This is due

mostly to the mass measurement differing a palm from other gestures.

------

ISSUES

------

1. The masks are configurable with the first form, but if

there are skin-similar (in HSV) regions in the background, it can be hard to

find the right masks. I have added a hardcoded inverse option that allows for a

bit more configuration (specifically if the sought after color bridges the max

and min of the hue spectrum) but there are environments that can still cause

malfunction. This could be avoided by using either background substitution

(Codebook method), or histogram training.

2. The gesture recognition relies on ranges probably specific to my hands,

and will have to be customized to each user in the code. This can be solved by

an introductory training/calibration session for each user, but is beyond the

scope of this project. A more fitting calibration would also be to separately

label the left and right hands.

3. Auto-focus, and auto-exposure on modern web-cameras can cause a problem

with the mask no longer being a valid fit, so if possible disable these

options on your camera. On OSX there is no direct option, though through the

use of iGlasses(Shareware/$19.95) you can disable auto-exposure on the built-

in MacBook web cameras.

4. Currently only supports hardcoded camera number (set at 1 for iGlasses).

to be fixed soon....

------------

ORGANIZATION

------------

Source code specific to this program is contained in src/, extendable classes

are contained in include/ as follows:

GesturePasswords/

GesturePassword.pro -QT Creator project file

README.txt -this file

bin/

|GesturePasswords.tar.gz -binary package compiled 02.18.2013 (OSX)

doc/

|gesture data.xlsx (recorded data for my hand gestures)

include/

|colorhistogram.h -a class for displaying a histogram

| of the image (modified for HSV)

|hand.h

|skindetectcontroller.cpp -a singleton wrapper with

| ease of use interfaces for

|skindetectcontroller.h the SkinDetector class

|skindetector.cpp

|skindetector.h -the main processing class

| for detecting skin regions

src/

|gesturedetector.cpp -detector form functions

|gesturedetector.h -detector form header

|gesturedetector.u -detector form design (QDialog QT 4.8)

|main.cpp -project main method

|maskselection.cpp -mask selection form functions

|maskselection.h -mask selection form header

|maskselection.ui -mask selection form design (QWindow QT 4.8)

|passwordcheck.h -class for storing and processing

| hand gesture password sequences

-----

USAGE

-----

A binary is provided in bin/ that was compiled on 02.18.2013 under

OSX 10.8 with all required dependencies, no knowledge of whether this works

on other systems.

Otherwise, if you have both OpenCV2 and QT Creator, simply import into Creator

and run from there. If you have QT 4.8 you can also use the .pro file to

compile from the command line.

# **Main.cpp**

/\*

Created by: Jason Carlisle Mann (on2valhalla | jcm2207@columbia.edu)

Main method, and start of all the fun

\*/

#include "../src/maskselection.h"

#include <QApplication>

#include <iostream>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MaskSelection w;

w.show();

return a.exec();

}

## MaskSelection.h

/\*------------------------------------------------------------------------------------------\*\

Created by: Jason Carlisle Mann (on2valhalla | jcm2207@columbia.edu)

Some inspiration for design and organization from Robert Laganiere (OpenCV2 Cookbook)

This class is a QT based form, with sliders and a image display area

so that you can actively select a mask to create a binary skin image from

a standard RGB. The primary purpose of the form is to serve video, but

it also has an option to display photos. You select a mask by moving

sliders (in HSV values) for the min and max thresholds sent to the

SkinDetector class

\\*------------------------------------------------------------------------------------------\*/

#ifndef MASKSELECTION\_H

#define MASKSELECTION\_H

#include <QMainWindow>

#include <QFileDialog>

#include <QColorDialog>

#include <QTimer>

//sys

#include <iostream>

#include <string>

//OpenCV

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

//color detector, controller, histogram

#include "../include/skindetectcontroller.h"

#include "../include/colorhistogram.h"

#include "../src/gesturedetector.h"

namespace Ui {

class MaskSelection;

}

class MaskSelection : public QMainWindow{

Q\_OBJECT

public:

MaskSelection(QWidget \*parent = 0);

~MaskSelection();

protected:

void changeEvent(QEvent \*e);

void displayMat(const cv::Mat& img);

// camera vars

cv::VideoCapture cap;

static const int CAMERA = 1;

// timer vars

QTimer\* timer;

bool process;

bool histEnable;

private:

Ui::MaskSelection \*ui;

GestureDetector \*gd;

// the thresholding masks

cv::Scalar min, max;

cv::Mat histogram;

ColorHistogram cHist;

private slots:

void processColorDetection();

void showHistogram();

void setImage();

void toggleCamera();

void updateTimer();

void setThreshold();

void beginDetection();

void setMinHue(int value);

void setMinSat(int value);

void setMinValue(int value);

void setMaxHue(int value);

void setMaxSat(int value);

void setMaxValue(int value);

};

#endif // MASKSELECTION\_H

MaskSelection.cpp

/\*------------------------------------------------------------------------------------------\*\

Created by: Jason Carlisle Mann (on2valhalla | jcm2207@columbia.edu)

Some inspiration for design and organization from Robert Laganiere (OpenCV2 Cookbook)

This class is a QT based form, with sliders and a image display area

so that you can actively select a mask to create a binary skin image from

a standard RGB. The primary purpose of the form is to serve video, but

it also has an option to display photos. You select a mask by moving

sliders (in HSV values) for the min and max thresholds sent to the

SkinDetector class

\\*------------------------------------------------------------------------------------------\*/

#include "../src/maskselection.h"

#include "../src/ui\_maskselection.h"

// Constructor

MaskSelection::MaskSelection(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MaskSelection)

{

//setup forms

ui->setupUi(this);

//set up video ------------------

//get camera

cap.open(CAMERA);

// check if we succeeded, if not do not enable camera toggle

if(!cap.isOpened())

ui->pushButton\_Camera->setEnabled(false);

//set up timer for camera display

timer = new QTimer(this);

//end setup video ---------------------

// slot action methods -------------

// select color

connect(timer, SIGNAL(timeout()), this, SLOT(updateTimer()));

connect(ui->pushButton\_OpenImage, SIGNAL(clicked()), this, SLOT(setImage()));

connect(ui->pushButton\_Camera, SIGNAL(clicked()), this, SLOT(toggleCamera()));

connect(ui->pushButton\_Process, SIGNAL(clicked()), this, SLOT(processColorDetection()));

connect(ui->pushButton\_Histogram, SIGNAL(clicked()), this, SLOT(showHistogram()));

connect(ui->pushButton\_Detect, SIGNAL(clicked()), this, SLOT(beginDetection()));

connect(ui->verticalSlider\_MinHue, SIGNAL(valueChanged(int)),

this, SLOT(setMinHue(int)));

connect(ui->verticalSlider\_MinSat, SIGNAL(valueChanged(int)),

this, SLOT(setMinSat(int)));

connect(ui->verticalSlider\_MinValue, SIGNAL(valueChanged(int)),

this, SLOT(setMinValue(int)));

connect(ui->verticalSlider\_MaxHue, SIGNAL(valueChanged(int)),

this, SLOT(setMaxHue(int)));

connect(ui->verticalSlider\_MaxSat, SIGNAL(valueChanged(int)),

this, SLOT(setMaxSat(int)));

connect(ui->verticalSlider\_MaxValue, SIGNAL(valueChanged(int)),

this, SLOT(setMaxValue(int)));

//end slots ----------------

//default settings

process = false;

histEnable = false;

cHist = ColorHistogram();

// Hsv default filter (for home environment)

min[0] = 0;

min[1] = 40;

min[2] = 93;

max[0] = 23;

max[1] = 255;

max[2] = 255;

setThreshold();

// set defaults for sliders

ui->verticalSlider\_MinHue->setSliderPosition(min[0]);

ui->verticalSlider\_MinSat->setSliderPosition(min[1]);

ui->verticalSlider\_MinValue->setSliderPosition(min[2]);

ui->verticalSlider\_MaxHue->setSliderPosition(max[0]);

ui->verticalSlider\_MaxSat->setSliderPosition(max[1]);

ui->verticalSlider\_MaxValue->setSliderPosition(max[2]);

}

// Destructor

MaskSelection::~MaskSelection()

{

delete ui;

delete timer;

delete gd;

}

void MaskSelection::changeEvent(QEvent \*e)

{

QMainWindow::changeEvent(e);

switch (e->type()) {

case QEvent::LanguageChange:

ui->retranslateUi(this);

break;

default:

break;

}

}

/\*

This method is called by a button on the form

and in turn pulls up an open file dialog, where

the user can select a photo for thresholding

\*/

void MaskSelection::setImage()

{

timer->stop();

QFileDialog::Options options;

QString selectedFilter;

QString fileName = QFileDialog::getOpenFileName(this,

tr("Open Image Files"),

"",

tr("Image files (\*.jpg \*.jpeg \*.png \*.gif \*.bmp)"),

&selectedFilter,

options);

if (!fileName.isEmpty()){

cv::Mat image = cv::imread(fileName.toStdString(),1); //0 for grayscale

displayMat(image);

//Send Filename to the skin detector

SkinDetectController::getInstance()->setInputImage(fileName.toStdString());

}

}

/\*

Called by the Camera button, toggles the video feed

on and off by toggling the timer. It will only attempt to

turn on the timer if the cv::VideoCapture is opened

\*/

void MaskSelection::toggleCamera()

{

if(!cap.isOpened())

return;

if(timer->isActive())

timer->stop();

else

{

timer->start(25);

}

}

/\*

A simple conversion method to display a cv::Mat image (BGR or binary)

in a QLabel, that only takes a QImage as a pixelmap.

\*/

void MaskSelection::displayMat(const cv::Mat& image)

{

//BGR openCV Mat to QImage

QImage img\_qt = QImage((const unsigned char\*)image.data,image.cols, image.rows, image.step, QImage::Format\_RGB888);

//For Binary Images

if (img\_qt.isNull()){

//ColorTable for Binary Images

QVector<QRgb> colorTable;

for (int i = 0; i < 256; i++)

colorTable.push\_back(qRgb(i, i, i));

img\_qt = QImage((const unsigned char\*)image.data,image.cols, image.rows, image.step, QImage::Format\_Indexed8);

img\_qt.setColorTable(colorTable);

}

//Display the QImage in the Label

QPixmap img\_pix = QPixmap::fromImage(img\_qt.rgbSwapped()); //BGR to RGB

this->ui->label->setPixmap(img\_pix.scaled(ui->label->size(), Qt::KeepAspectRatio));

}

/\*

Called by the process button, this method toggles the processing

and the display of the processed images/video. This method calls

the process function from the SkinDetector class

\*/

void MaskSelection::processColorDetection()

{

if(timer->isActive())

process = !process;

else

{

SkinDetectController::getInstance()->process();

cv::Mat resulting = SkinDetectController::getInstance()->getLastResult();

if (!resulting.empty())

displayMat(resulting);

}

}

/\*

Controls the display of the video, on the interval set in

toggleCamera above. It also regulates the display of the

histogram, and processing of the video.

\*/

void MaskSelection::updateTimer()

{

cv::Mat img;

cap >> img; //capture a frame

//send SkinDetector the frame

SkinDetectController::getInstance()->setInputImage(img);

if(histEnable)

{ // update the histogram

histogram = cHist.getHistogramImage(img);

cv::imshow("Histogram", histogram);

}

if(process)

{ //process the frame

SkinDetectController::getInstance()->process();

//retrieve the processed frame

cv::Mat result = SkinDetectController::getInstance()->getLastResult();

if (!result.empty())

img = result;

}

displayMat(img);

}

/\*

Toggles the histogram window on and off

\*/

void MaskSelection::showHistogram()

{

if(timer->isActive() && !histEnable)

{ //create histogram window for video display

cv::namedWindow("Histogram", cv::WINDOW\_AUTOSIZE);

histEnable = true;

}

else if (!timer->isActive())

{ //create histogram for image display

histogram = cHist.getHistogramImage(SkinDetectController::getInstance()->getHSVImage());

cv::imshow("Histogram", histogram);

histEnable = false;

}

else

{ //destroy histogram window

cv::destroyWindow("Histogram");

histEnable = false;

}

}

/\*

Sets the skin detector's masks min and max from

their cached values

\*/

void MaskSelection::setThreshold()

{

SkinDetectController::getInstance()->setThreshold(min, max);

if(!timer->isActive() && process)

processColorDetection();

}

/\*

Changes minHue from slider

\*/

void MaskSelection::setMinHue(int value)

{

min[0] = value;

this->ui->label\_MinHue->setText(QString::number(value));

setThreshold();

}

/\*

Changes minimum saturation from slider

\*/

void MaskSelection::setMinSat(int value)

{

min[1] = value;

this->ui->label\_MinSat->setText(QString::number(value));

setThreshold();

}

/\*

Changes minimum value from slider

\*/

void MaskSelection::setMinValue(int value)

{

min[2] = value;

this->ui->label\_MinValue->setText(QString::number(value));

setThreshold();

}

/\*

Changes maximum hue from slider

\*/

void MaskSelection::setMaxHue(int value)

{

max[0] = value;

this->ui->label\_MaxHue->setText(QString::number(value));

setThreshold();

}

/\*

Changes maximum saturation from slider

\*/

void MaskSelection::setMaxSat(int value)

{

max[1] = value;

this->ui->label\_MaxSat->setText(QString::number(value));

setThreshold();

}

/\*

Changes maximum value from slider

\*/

void MaskSelection::setMaxValue(int value)

{

max[2] = value;

this->ui->label\_MaxValue->setText(QString::number(value));

setThreshold();

}

/\*

Creates a new detection form if necessary,

and sets up the form to begin recording hand

gestures

\*/

void MaskSelection::beginDetection()

{

if(!gd)

gd = new GestureDetector(this);

gd->show();

if(cap.isOpened())

{

toggleCamera();

gd->setCap(cap);

gd->start();

}

}

SkinDetectController.h

/\*

Modified from ColorDetector from chapter 3 of the cookbook:

Computer Vision Programming using the OpenCV Library.

by Robert Laganiere, Packt Publishing, 2011.

Created by: Jason Carlisle Mann (on2valhalla | jcm2207@columbia.edu)

A singleton wrapper class for SkinDetector, to gain continuity

throughout the program, and also to provide more simple access

to its methods, and a local cache of the most recent images.

\*/

#if !defined SKN\_CNTRLLR

#define SKN\_CNTRLLR

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <iostream>

#include "../include/skindetector.h"

class SkinDetectController

{

private:

static SkinDetectController \*singleton;

SkinDetector \*sknDetect;

// hsvImage storage

cv::Mat hsvImage;

cv::Mat resultImg;

public:

SkinDetectController()

{

sknDetect = new SkinDetector();

}

// Deletes all processor objects created by the controller.

~SkinDetectController()

{

delete sknDetect;

}

// Singleton static members

static SkinDetectController \*getInstance()

{

if (singleton == 0)

singleton= new SkinDetectController;

return singleton;

}

// Releases the singleton instance of this controller.

static void destroy() {

if (singleton != 0)

{

delete singleton;

singleton= 0;

}

}

// Sets the input hsvImage. Reads it from file.

bool setInputImage(std::string filename)

{

cv::Mat imgIn = cv::imread(filename);

// convert color space

cv::cvtColor(imgIn, hsvImage, CV\_BGR2HSV);

if (!hsvImage.data)

return false;

else

return true;

}

// Sets the input hsvImage.

bool setInputImage(cv::Mat imgIn) {

// convert color space

cv::cvtColor(imgIn, hsvImage, CV\_BGR2HSV);

if (!hsvImage.data)

return false;

else

return true;

}

cv::Mat getInputImage(){

cv::Mat imgOut;

cv::cvtColor(hsvImage, imgOut, CV\_HSV2BGR);

return imgOut;

}

// Returns the current input hsvImage.

// NOTE: this returns HSV!!!!!

const cv::Mat getHSVImage() const {

return hsvImage;

}

const cv::Mat getLastResult() const{

return resultImg;

}

void setThreshold(cv::Scalar min, cv::Scalar max){

sknDetect->setThreshold(min, max);

}

void process(){

resultImg = sknDetect->processHSV(hsvImage);

}

};

#endif

SkinDetectController.cpp

/\*

Modified from ColorDetector from chapter 3 of the cookbook:

Computer Vision Programming using the OpenCV Library.

by Robert Laganiere, Packt Publishing, 2011.

Created by: Jason Carlisle Mann (on2valhalla | jcm2207@columbia.edu)

A singleton wrapper class for SkinDetector, to gain continuity

throughout the program, and also to provide more simple access

to its methods, and a local cache of the most recent images.

\*/

#include "../include/skindetectcontroller.h"

SkinDetectController \*SkinDetectController::singleton=0;

SkinDetector.h

/\*

Modified from ColorDetector from chapter 3 of the cookbook:

Computer Vision Programming using the OpenCV Library.

by Robert Laganiere, Packt Publishing, 2011.

Created by: Jason Carlisle Mann (on2valhalla | jcm2207@columbia.edu)

This class uses holds input threshold min and max

masks to process an image for skin blobs in HSV colorspace

\*/

#if !defined SKINDETECT

#define SKINDETECT

#include <opencv2/core/core.hpp>

#include <opencv2/imgproc/imgproc.hpp>

#include <iostream>

class SkinDetector

{

private:

// HSV min and max limits as array of Scalars

cv::Scalar hsvThreshold[2];

// image containing result of processing

cv::Mat resultImg;

// image containing converted color space

cv::Mat converted;

public:

//empty Constructor

SkinDetector()

{

// initialize default parameters

hsvThreshold[0][0] = 0;

hsvThreshold[0][1] = 0;

hsvThreshold[0][2] = 0;

hsvThreshold[1][0] = 180;

hsvThreshold[1][1] = 255;

hsvThreshold[1][2] = 255;

}

void setThreshold(cv::Scalar min, cv::Scalar max)

{

hsvThreshold[0] = min;

hsvThreshold[1] = max;

//std::cout << "min: " << min << "\t" << "max" << max << "\n";

}

void getThreshold(cv::Scalar &min, cv::Scalar &max)

{

min = hsvThreshold[0];

max = hsvThreshold[1];

}

// Processes an already HSV image. Returns a 1-channel binary image.

cv::Mat processHSV(const cv::Mat &image);

};

#endif

SkinDetector.cpp

/\*

Modified from ColorDetector from chapter 3 of the cookbook:

Computer Vision Programming using the OpenCV Library.

by Robert Laganiere, Packt Publishing, 2011.

Created by: Jason Carlisle Mann (on2valhalla | jcm2207@columbia.edu)

This class uses holds input threshold min and max

masks to process an image for skin blobs in HSV colorspace

\*/

#include "../include/skindetector.h"

#include "../include/colorhistogram.h"

/\*

Processes an HSV image and returns a binary image

containing blobs of skin regions.

@hsvImg input HSV colorspace OpenCV2 image

@return processed binary blob image

\*/

cv::Mat SkinDetector::processHSV(const cv::Mat &hsvImg)

{

//re-allocate binary map if necessary

//if so create one channel image with

//same cols and rows as original

resultImg.create(hsvImg.rows, hsvImg.cols, CV\_8U);

//reduce the colors for faster processing

ColorHistogram h;

h.colorReduce(hsvImg, 12);

//threshold the image with the stored masks

cv::inRange(hsvImg, hsvThreshold[0], hsvThreshold[1], resultImg);

//filtering parameter, increase size for greater effect

cv::Mat morpElement(5,5,CV\_8U,cv::Scalar(1));

//current morphological processing functions to

//close the skin blobs

cv::erode(resultImg, resultImg, morpElement);

cv::blur(resultImg, resultImg, morpElement.size());

cv::dilate(resultImg, resultImg, morpElement);

//optional morphological processing is helpful

//depending on the background environment

// cv::bitwise\_not(resultImg, resultImg);

// cv::erode(resultImg, resultImg, morpElement);

// cv::dilate(resultImg, resultImg, morpElement);

// cv::morphologyEx(resultImg,resultImg,cv::MORPH\_CLOSE, morpElement);

return resultImg;

}

ColorHistogram.h

/\*------------------------------------------------------------------------------------------\*\

This file contains material supporting chapter 4 of the cookbook:

Computer Vision Programming using the OpenCV Library.

by Robert Laganiere, Packt Publishing, 2011.

This program is free software; permission is hereby granted to use, copy, modify,

and distribute this source code, or portions thereof, for any purpose, without fee,

subject to the restriction that the copyright notice may not be removed

or altered from any source or altered source distribution.

The software is released on an as-is basis and without any warranties of any kind.

In particular, the software is not guaranteed to be fault-tolerant or free from failure.

The author disclaims all warranties with regard to this software, any use,

and any consequent failure, is purely the responsibility of the user.

Copyright (C) 2010-2011 Robert Laganiere, www.laganiere.name

\\*------------------------------------------------------------------------------------------\*/

/\*

This class wraps easy methods for dealing with histograms around OpenCV2 functions

It also provides a colorReduce function for use before sending an image for

processing. This code has been modified, but retains the original copyright above.

Modified by: Jason Carlisle Mann (on2valhalla | jcm2207@columbia.edu)

\*/

#if !defined COLHISTOGRAM

#define COLHISTOGRAM

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

#include <iostream>

#include <stdio.h>

class ColorHistogram {

private:

int histSize[3];

float hranges[2];

const float\* ranges[3];

int channels[3];

public:

ColorHistogram() {

// Prepare arguments for a color histogram

histSize[0]= histSize[1]= histSize[2]= 256;

hranges[0]= 0.0; // BRG range

hranges[1]= 255.0;

ranges[0]= hranges; // all channels have the same range

ranges[1]= hranges;

ranges[2]= hranges;

channels[0]= 0; // the three channels

channels[1]= 1;

channels[2]= 2;

}

// Computes the histogram.

cv::MatND getHistogram(const cv::Mat &image)

{

cv::MatND hist;

// BGR color histogram

hranges[0]= 0.0; // BRG range

hranges[1]= 255.0;

channels[0]= 0; // the three channels

channels[1]= 1;

channels[2]= 2;

// Compute histogram

cv::calcHist(&image,

1, // histogram of 1 image only

channels, // the channel used

cv::Mat(), // no mask is used

hist, // the resulting histogram

3, // it is a 3D histogram

histSize, // number of bins

ranges // pixel value range

);

return hist;

}

// Computes the histogram.

cv::SparseMat getSparseHistogram(const cv::Mat &image)

{

cv::SparseMat hist(3,histSize,CV\_32F);

// BGR color histogram

hranges[0]= 0.0; // BRG range

hranges[1]= 255.0;

channels[0]= 0; // the three channels

channels[1]= 1;

channels[2]= 2;

// Compute histogram

cv::calcHist(&image,

1, // histogram of 1 image only

channels, // the channel used

cv::Mat(), // no mask is used

hist, // the resulting histogram

3, // it is a 3D histogram

histSize, // number of bins

ranges // pixel value range

);

return hist;

}

// Computes the 2D ab histogram.

// BGR source image is converted to Lab

cv::MatND getabHistogram(const cv::Mat &image)

{

cv::MatND hist;

// Convert to Lab color space

cv::Mat lab;

cv::cvtColor(image, lab, CV\_BGR2Lab);

// Prepare arguments for a 2D color histogram

hranges[0]= -128.0;

hranges[1]= 127.0;

channels[0]= 1; // the two channels used are ab

channels[1]= 2;

// Compute histogram

cv::calcHist(&lab,

1, // histogram of 1 image only

channels, // the channel used

cv::Mat(), // no mask is used

hist, // the resulting histogram

2, // it is a 2D histogram

histSize, // number of bins

ranges // pixel value range

);

return hist;

}

// Computes the 1D Hue histogram with a mask.

// BGR source image is converted to HSV

cv::MatND getHueHistogram(const cv::Mat &image)

{

cv::MatND hist;

// Convert to Lab color space

cv::Mat hue;

cv::cvtColor(image, hue, CV\_BGR2HSV);

// Prepare arguments for a 1D hue histogram

hranges[0]= 0.0;

hranges[1]= 180.0;

channels[0]= 0; // the hue channel

// Compute histogram

cv::calcHist(&hue,

1, // histogram of 1 image only

channels, // the channel used

cv::Mat(), // no mask is used

hist, // the resulting histogram

1, // it is a 1D histogram

histSize, // number of bins

ranges // pixel value range

);

return hist;

}

cv::Mat colorReduce(const cv::Mat &image, int div=64)

{

int n= static\_cast<int>(log(static\_cast<double>(div))/log(2.0));

// mask used to round the pixel value

uchar mask= 0xFF<<n; // e.g. for div=16, mask= 0xF0

cv::Mat\_<cv::Vec3b>::const\_iterator it= image.begin<cv::Vec3b>();

cv::Mat\_<cv::Vec3b>::const\_iterator itend= image.end<cv::Vec3b>();

// Set output image (always 1-channel)

cv::Mat result(image.rows,image.cols,image.type());

cv::Mat\_<cv::Vec3b>::iterator itr= result.begin<cv::Vec3b>();

for ( ; it!= itend; ++it, ++itr) {

(\*itr)[0]= ((\*it)[0]&mask) + div/2;

(\*itr)[1]= ((\*it)[1]&mask) + div/2;

(\*itr)[2]= ((\*it)[2]&mask) + div/2;

}

return result;

}

// Returns an image of a histogram of the image in BGR

cv::Mat getHistogramImage(const cv::Mat &image)

{

/// Separate the image in 3 places ( B, G and R )

std::vector<cv::Mat> bgr\_planes;

cv::split( image, bgr\_planes );

/// Establish the number of bins

int histSize = 256;

/// Set the ranges ( for B,G,R) )

float range[] = { 0, 256 } ;

const float\* histRange = { range };

bool uniform = true;

bool accumulate = false;

cv::Mat b\_hist, g\_hist, r\_hist;

/// Compute the histograms:

cv::calcHist( &bgr\_planes[0], 1, 0, cv::Mat(),

b\_hist, 1, &histSize, &histRange, uniform, accumulate );

cv::calcHist( &bgr\_planes[1], 1, 0, cv::Mat(),

g\_hist, 1, &histSize, &histRange, uniform, accumulate );

cv::calcHist( &bgr\_planes[2], 1, 0, cv::Mat(),

r\_hist, 1, &histSize, &histRange, uniform, accumulate );

// Draw the histograms for B, G and R

int hist\_w = 512; int hist\_h = 400;

int bin\_w = cvRound( (double) hist\_w/histSize );

cv::Mat histImage( hist\_h, hist\_w, CV\_8UC3, cv::Scalar( 0,0,0) );

/// Normalize the result to [ 0, histImage.rows ]

cv::normalize(b\_hist, b\_hist, 0, histImage.rows, cv::NORM\_MINMAX, -1, cv::Mat() );

cv::normalize(g\_hist, g\_hist, 0, histImage.rows, cv::NORM\_MINMAX, -1, cv::Mat() );

cv::normalize(r\_hist, r\_hist, 0, histImage.rows, cv::NORM\_MINMAX, -1, cv::Mat() );

/// Draw for each channel

for( int i = 1; i < histSize; i++ )

{

cv::line( histImage, cv::Point( bin\_w\*(i-1),

hist\_h - cvRound(b\_hist.at<float>(i-1)) ) ,

cv::Point( bin\_w\*(i), hist\_h - cvRound(b\_hist.at<float>(i)) ),

cv::Scalar( 255, 0, 0), 2, 8, 0 );

cv::line( histImage, cv::Point( bin\_w\*(i-1),

hist\_h - cvRound(g\_hist.at<float>(i-1)) ) ,

cv::Point( bin\_w\*(i), hist\_h - cvRound(g\_hist.at<float>(i)) ),

cv::Scalar( 0, 255, 0), 2, 8, 0 );

cv::line( histImage, cv::Point( bin\_w\*(i-1),

hist\_h - cvRound(r\_hist.at<float>(i-1)) ) ,

cv::Point( bin\_w\*(i), hist\_h - cvRound(r\_hist.at<float>(i)) ),

cv::Scalar( 0, 0, 255), 2, 8, 0 );

}

return histImage;

}

};

#endif

GestureDetector.h

/\*

Created by: Jason Carlisle Mann (on2valhalla | jcm2207@columbia.edu)

This class is a QT based form (QDialog), with a display area for images

or video, and a text pane for output

Using the PasswordCheck class, it samples a video feed provided by

cv::VideoCapture and analyzes it for skin blobs based on the SkinDetector

class. It identifies faces first and labels them with a rectangle.

The remaining skin blobs are analyzed as hand gestures, for the purpose

of completing a password.

\*/

#ifndef GESTUREDETECTION\_H

#define GESTUREDETECTION\_H

#include <QDialog>

#include <QTimer>

#include <QDebug>

//sys

#include <iostream>

#include <string>

#include <cmath>

#include <stdio.h>

//OpenCV

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv/highgui.h>

#include <opencv2/objdetect/objdetect.hpp>

//locals

#include "../src/ui\_gesturedetector.h"

#include "../include/colorhistogram.h"

#include "../include/skindetectcontroller.h"

#include "../src/passwordcheck.h"

#include "../include/hand.h"

// Haar Cascade Classifier face file location

static std::string FACEFILE = "/opt/local/share/OpenCV/haarcascades/haarcascade\_frontalface\_alt\_tree.xml";

namespace Ui {

class GestureDetector;

}

class GestureDetector : public QDialog

{

Q\_OBJECT

public:

explicit GestureDetector(QWidget \*parent = 0);

~GestureDetector();

void start();

void setCap(cv::VideoCapture &cap);

private:

Ui::GestureDetector \*ui;

QTimer \*timer;

cv::VideoCapture cap;

cv::CascadeClassifier cascadeFace; //HAAR Cascade for detecting faces

PasswordCheck pw; //Class containing the passwords

std::string intro;

// TIMER VARS

static const int DELAY = 30; //Timer delay, determines how fast video plays

static const int RECINT = 3000; //Interval between gesture captures

static const int WARNINT = 1000; //Interval to print warnings to user

static const int WARNMAX = RECINT / WARNINT; //Maximum warning num

int timeCount;

int warnCount;

//for recording and documentation

int setCount;

std::vector<cv::Mat> imageCache;

//display an image in the dialog

void displayMat(const cv::Mat &image);

// Detect and Identify the hand gesture present in the image

std::vector<Hand> detect(cv::Mat &image, cv::Mat &filtered);

private slots:

void updateTimer();

void reject();

void pause();

};

#endif // GESTUREDETECTION\_H

GestureDetector.cpp

/\*

Created by: Jason Carlisle Mann (on2valhalla | jcm2207@columbia.edu)

This class is a QT based form (QDialog), with a display area for images

or video, and a text pane for output

Using the PasswordCheck class, it samples a video feed provided by

cv::VideoCapture and analyzes it for skin blobs based on the SkinDetector

class. It identifies faces first and labels them with a rectangle.

The remaining skin blobs are analyzed as hand gestures, for the purpose

of completing a password.

\*/

#include "../src/gesturedetector.h"

// for recording and documentation purposes

static const char \*DESKTOP = "/Users/on2valhalla/Desktop/pass";

GestureDetector::GestureDetector(QWidget \*parent) :

QDialog(parent),

ui(new Ui::GestureDetector)

{

ui->setupUi(this);

timer = new QTimer();

cascadeFace = cv::CascadeClassifier(FACEFILE);

connect(timer, SIGNAL(timeout()), this, SLOT(updateTimer()));

connect(ui->btnPause, SIGNAL(clicked()), this, SLOT(pause()));

setCount = 0;

intro = "Welcome!\n"

"You can display any combination of the following "

"to attempt to unlock the password (1 or 2 at a time):\n"

"{PALM, FIST, POINT UP, NONE, OTHER}\n\n"

"To check your entry hold 2 palms up.";

}

GestureDetector::~GestureDetector()

{

delete timer;

delete ui;

}

//On form close button, clean up

void GestureDetector::reject()

{

timer->stop();

ui->textEdit->setText("");

QDialog::reject();

}

//Set the video capture device

void GestureDetector::setCap(cv::VideoCapture &cap)

{

this->cap = cap;

}

//Start the timer

void GestureDetector::start()

{

if(cap.isOpened())

timer->start(DELAY);

timeCount = 0;

warnCount = 0;

ui->textEdit->setText(QString::fromStdString(intro));

imageCache.clear();

pw.reset();

ui->textEdit->append("\n\nPREPARE FOR CAPTURE IN....\n3....");

}

// Stop and Start the video

void GestureDetector::pause()

{

if(timer->isActive())

{

timer->stop();

ui->btnPause->setText("Play");

}

else

{

start();

ui->btnPause->setText("Pause");

}

}

//Convert cv::Mat to QImage and display

void GestureDetector::displayMat(const cv::Mat& image)

{

//BGR openCV Mat to QImage

QImage img\_qt = QImage((const unsigned char\*)image.data,image.cols, image.rows, image.step, QImage::Format\_RGB888);

//For Binary Images

if (img\_qt.isNull()){

//ColorTable for Binary Images

QVector<QRgb> colorTable;

for (int i = 0; i < 256; i++)

colorTable.push\_back(qRgb(i, i, i));

img\_qt = QImage((const unsigned char\*)image.data,image.cols, image.rows, image.step, QImage::Format\_Indexed8);

img\_qt.setColorTable(colorTable);

}

//Display the QImage in the Label

QPixmap img\_pix = QPixmap::fromImage(img\_qt.rgbSwapped()); //BGR to RGB

this->ui->lblImage->setPixmap(img\_pix.scaled(ui->lblImage->size(), Qt::KeepAspectRatio));

}

void GestureDetector::updateTimer()

{

timeCount = (timeCount + DELAY) % RECINT; //increment counter on a cycle

warnCount = (warnCount + DELAY) % WARNINT; //do same with the warning count

cv::Mat image, filtered;

cap >> image;

//set the image from the camera, process it and get filtered

SkinDetectController::getInstance()->setInputImage(image);

SkinDetectController::getInstance()->process();

filtered = SkinDetectController::getInstance()->getLastResult(); //binary image of blobs

//retrieve the hand gestures from the image from the image

std::vector<Hand> hands = detect(image, filtered);

if(timeCount < DELAY)

{ // Only store the hands when the timeCount

//rolls over after iterating to the record

//interval (RECINT)

pw.addHandSet(hands);

imageCache.push\_back(image.clone());

//print out the captured hands

if(hands.size() > 1)

ui->textEdit->append(hands[0].toQString()

+ "\n and: " + hands[1].toQString());

else

ui->textEdit->append(hands[0].toQString());

// Determine if it is time to check the password

if(pw.doCheck(hands))

{

if(pw.checkPassword())

ui->textEdit->append("----------------------\n"

"PASSWORD ACCEPTED!!!!!!\n"

"----------------------");

else

ui->textEdit->append("----------------------\n"

"INTRUDER... INTRUDER....\n"

"----------------------");

char filename[200];

//record pictures for documentation

for(int i = 0; i < imageCache.size(); i++)

{

sprintf(filename, "%s\_%d\_%d.jpg", DESKTOP, setCount, i);

cvSaveImage(filename, &(IplImage(imageCache[i])));

}

pw.reset();

setCount++;

imageCache.clear();

}

}

// warn the user of the time to capture

// ticks down from WARNMAX to capture

if(warnCount < DELAY)

{

int timeLeft = WARNMAX - timeCount/WARNINT;

if(timeLeft == WARNMAX)

ui->textEdit->append("\n\nPREPARE FOR CAPTURE IN....");

ui->textEdit->append(QString("%1...........").arg(timeLeft));

}

// display the processed image

displayMat(image);

}

// Detect and Identify the hand gesture present in the image

std::vector<Hand> GestureDetector::detect(cv::Mat &image, cv::Mat &filtered)

{

std::vector<Hand> hands;

// find contours in the filtered image

std::vector< std::vector<cv::Point> > contours;

std::vector<cv::Vec4i> hierarchy;

cv::findContours(filtered,

contours, // a vector of contours

hierarchy, // a hierarchy of contours if there are parent

//child relations in the image

CV\_RETR\_EXTERNAL, // retrieve the external contours

CV\_CHAIN\_APPROX\_TC89\_L1); // an approximation algorithm

//------------------Find Face----------------

//preprocess for face recognition

std::vector< cv::Rect > faces;

cv::Mat gray;

//shrink the image for speed

cv::resize(image, gray, cv::Size2i(image.cols/4, image.rows/4));

cv::cvtColor(gray, gray, CV\_BGR2GRAY);

cv::equalizeHist(gray, gray);

cascadeFace.detectMultiScale(gray, faces);

// draw bounds for faces

for (unsigned int i = 0; i < faces.size(); i++ )

{

//resize the rectangle to match the image

faces[i] += cv::Point(faces[i].x \* 3,faces[i].y \* 3);

faces[i] += faces[i].size() +faces[i].size() +faces[i].size();

rectangle(image, faces[i], cv::Scalar(0,0,204), 3);

}

//----------------End Face--------------------

//------------Find all skin regions-----------

QString join("");

int massMin = 5000, massMax = 24000; // minimum contour mass

cv::Scalar color( 100, 150, 255 ); //random color

int idx = 0;

// iterate through all the top-level contours

for( ; idx >= 0; idx = hierarchy[idx][0] )

{

// compute all moments and mass

cv::Moments mom = cv::moments(cv::Mat(contours[idx]));

cv::Point center = cv::Point(mom.m10/mom.m00,mom.m01/mom.m00);

int cMass = mom.m00;

//calculate the rotated and bounding rectangles of the blobs

cv::RotatedRect rotRect = cv::minAreaRect( cv::Mat(contours[idx]) );

cv::Rect bRect = rotRect.boundingRect();

// find intersections of detected faces and blobs

bool faceOverlap = false;

for(unsigned int i = 0; i < faces.size(); i++)

{

cv::Rect intersect = faces[i] & bRect;

if(intersect.width > 30 || intersect.height > 30)

faceOverlap = true;

}

// Eliminate contours that are too small,

// and ones that coincide with one of the found faces

if(cMass > massMin && cMass < massMax && !faceOverlap)

{

Hand curHand(rotRect, mom);

hands.push\_back(curHand); //push the detected hand onto the list

// draw each connected component

cv::Mat contourImg(image.size(), image.type(), cv::Scalar(0));

cv::drawContours( contourImg, contours, idx, color, CV\_FILLED, 8, hierarchy );

cv::GaussianBlur(contourImg, contourImg, cv::Size(5,5), 0);

image += contourImg;

// draw bounding rotated rectangles

for( int j = 0; j < 4; j++ )

line( image, curHand.rotPoints[j],

curHand.rotPoints[(j+1)%4], cv::Scalar(0,102,204), 3, 8 );

//draw bounding upright rectangle

rectangle(image, curHand.boxRect, cv::Scalar(0,204,102), 3);

}

}

if(hands.size() == 0)

{ //if no hands were found create a 'NONE' hand

hands.push\_back(Hand());

}

return hands;

}

Hand.h

/\*

Created by: Jason Carlisle Mann (on2valhalla | jcm2207@columbia.edu)

Creates a Hand, stores information on its geometry, and classifies

it as a certain type of gesture based on hardcoded geometric data

\*/

#ifndef HAND\_H

#define HAND\_H

#include "../src/gesturedetector.h"

#include <opencv2/core/core.hpp>

enum HandType{

FIST,

PALM,

POINT,

UNK,

NONE

};

class Hand

{

public:

// VARIABLES

HandType type;

cv::RotatedRect rotRect;

cv::Point2f rotPoints[4];

cv::Rect boxRect;

cv::Moments mom;

double rRatio;

double bRatio;

double mRatio; //mass ratio of bounding versus moments

// Easy Calculation of Euclidean Distance

double pointDist(cv::Point2f &p1, cv::Point2f &p2)

{

double dx = p1.x - p2.x;

double dy = p1.y - p2.y;

return sqrt(dx \* dx + dy \* dy);

}

void setType()

{

// based on collected data

// of my own gestures, see doc/gesture data.xlsx

//decision for palm gesture

if (mom.m00 > 8500 &&

bRatio > .5 && bRatio < .95 &&

((rRatio > .5 && rRatio < .6) ||

(rRatio > 1.5 && rRatio < 2)))

type = PALM;

//fist

else if(mom.m00 < 10000 &&

bRatio > .80 && rRatio >.7 &&

mRatio < 2)

type = FIST;

//decision for point gesture

else if(mom.m00 < 10000 &&

((rRatio > 1.4 && rRatio < 2.2) ||

(rRatio > .35 && rRatio < .75)) &&

bRatio > .4 && bRatio < 1.3)

type = POINT;

else //unknown gesture

type = UNK;

}

QString toQString()

{

QString typeStr("");

switch (type)

{

case FIST:

typeStr = "FIST";

break;

case PALM:

typeStr = "PALM";

break;

case POINT:

typeStr = "POINT";

break;

case UNK:

typeStr = "UNK";

break;

default:

typeStr = "NONE";

break;

}

return QString("%1 area: %2 bratio: %3 rratio: %4 mratio: %5 ")

.arg(typeStr)

.arg(mom.m00)

.arg(bRatio,0,'f',3)

.arg(rRatio,0,'f',3)

.arg(mRatio,0,'f',3);

}

//Default constructor

Hand()

{

type = NONE;

}

//Constructor

Hand(cv::RotatedRect r, cv::Moments m)

{

rotRect = r;

mom = m;

rotRect.points(rotPoints);

boxRect = rotRect.boundingRect();

rRatio = pointDist(rotPoints[2],rotPoints[1]) / pointDist(rotPoints[2], rotPoints[3]);

bRatio = static\_cast<double>(boxRect.width)/boxRect.height;

mRatio = (static\_cast<double>(boxRect.width)\*boxRect.height)/mom.m00;

setType();

}

//copy constructor

Hand(const Hand& h)

{

type = h.type;

rotRect = h.rotRect;

for(unsigned int i = 0; i < sizeof(rotPoints); i++)

rotPoints[i] = h.rotPoints[i];

boxRect = h.boxRect;

mom = h.mom;

bRatio = h.bRatio;

rRatio = h.rRatio;

mRatio = h.mRatio;

}

//assignment operator

Hand& operator=(const Hand& rhs)

{

if(this == &rhs)

return \*this;

type = rhs.type;

rotRect = rhs.rotRect;

for(unsigned int i = 0; i < sizeof(rotPoints); i++)

rotPoints[i] = rhs.rotPoints[i];

boxRect = rhs.boxRect;

mom = rhs.mom;

bRatio = rhs.bRatio;

rRatio = rhs.rRatio;

mRatio = rhs.mRatio;

return \*this;

}

//destructor

~Hand()

{

}

};

#endif

PasswordCheck.h

/\*------------------------------------------------------------------------------------------\*\

Created by: Jason Carlisle Mann (on2valhalla | jcm2207@columbia.edu)

This class supports the GesturePassword program, by analyzing sequences of

hand gestures provided one set at a time, and checking them against predetermined

passwords. Gestures are defined by the Hand class

The currently valid sequences are:

1. FIST | FIST | FIST

2. POINT | FIST | PALM

3. PALM | (NONE OR UNK) | PALM

4. (PALM & POINT) | (FIST & FIST) | (POINT & FIST)

Also if the first three gestures are NONE or UNKOWN, the checking continues on the above,

ignoring those inputs.

\\*------------------------------------------------------------------------------------------\*/

#ifndef PASSCHECK\_H

#define PASSCHECK\_H

#include "../include/hand.h"

#include "../src/gesturedetector.h"

#include <iostream>

class PasswordCheck

{

public:

//default constructor

PasswordCheck() { }

/\*

add a set of hands to the sequence (currently the

grammar only supports up to two hands in one capture)

@set the parameter is the vector of captures.

\*/

void addHandSet(const std::vector<Hand> &set)

{

for(unsigned int i = 0; i < set.size() ; i++)

std::cout << "adding: " << set[i].type << "\t";

input.push\_back(set);

}

/\*

clears the current sequence

\*/

void reset()

{

input.clear();

}

/\*

This function determines whether or not it is time to

check the stored sequence against the password grammar

currently 2 palms, or 6 items begins the check

\*/

bool doCheck(std::vector<Hand> &hands)

{

if(input.size() >= 6 ||

(hands.size() == 2 && hands[0].type == PALM

&& hands[1].type == PALM))

return true;

else

return false;

}

/\*

This method checks the current sequence stored

against a tree of conditional statements that represent

the password grammar hardcoded into the Class.

This method could be expanded into using decision trees,

but with only these few sequences, that is unnecessary.

@return returns true if the sequence matched a password,

else false

\*/

bool checkPassword()

{

if(input.back()[0].type == PALM && input.back()[1].type == PALM)

input.pop\_back();

int i = -1; //start one low to allow addition early in loop

Hand curHand;

while (true) //loop to allow reset sequence

{

if(input.size() - ++i < 3) //check for too few items

return false;

if(input[0].size() == 1) {

if(input[i][0].type == FIST){

if(input[++i][0].type == FIST){

if(input[++i][0].type == FIST){

if(input.size() - i == 1){

return true; // FIST / FIST / FIST

}}}}

else if(input[i][0].type == POINT){

if(input[++i][0].type == FIST){

if(input[++i][0].type == PALM){

if(input.size() - i == 1){

return true; // POINT / FIST / PALM

}}}}

else if(input[i][0].type == PALM){

if(input[++i][0].type == NONE ||

input[i][0].type == UNK){

if(input[++i][0].type == PALM){

if(input.size() - i == 1){

return true; //PALM / (NONE|UNKOWN) / PALM

}}}}

else {

if(input[++i][0].type == NONE ||

input[i][0].type == UNK){

if(input[++i][0].type == NONE ||

input[i][0].type == UNK){

continue; //begin the loop again, continue

}}} //searching with the next element

}

else if(input[0].size() == 2){

if((input[i][0].type == POINT && input[i][1].type == PALM) ||

(input[i][1].type == POINT && input[i][0].type == PALM)){

if((input[++i][0].type == FIST && input[i][1].type == FIST)){

if((input[++i][0].type == POINT &&

input[i][1].type == FIST) ||

(input[i][1].type == POINT &&

input[i][0].type == FIST)){

if(input.size() - i == 1){

return true;

//PALM & POINT / FIST & FIST / FIST & POINT

}}}}}

return false; // incorrect passphrase sequence

}}

private:

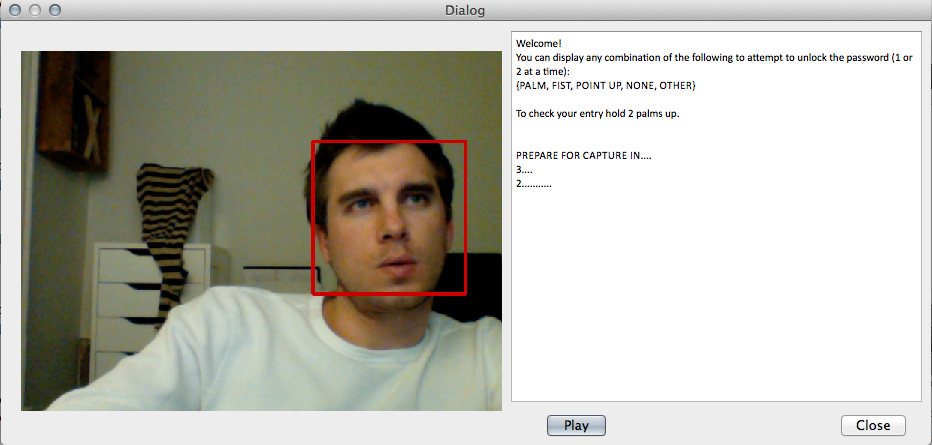
std::vector< std::vector < Hand > > input; //the sequence of hand gestures (up to 2 in one capture)

};

#endif

Application Views (3 forms: Mask Selection, Histogram, Gesture Detection)

|  |
| --- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:Screen Shot 2013-02-19 at 2.25.47 AM.png |
| The mask selection form allowsEnki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:Screen Shot 2013-02-19 at 2.22.17 AM.png you to see your mask settings immediately take effect |
|  |



|  |  |
| --- | --- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:2:pass_2_2.jpg | Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:2:pass_2_1.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:2:pass_2_0.jpg | Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:4:pass_4_1.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:2:pass_2_3.jpg | Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:5:pass_0_0.jpg |

# Possible Hand Gestures

**-Closed fist, closed palm, vertical single finger point, or none**

**-One or Two gestures at a time**

# 10 Example Sequences:

**-5 correctly read sequences with the right password, accepted: PASS**

**-2 correctly read, wrong password, denied: PASS**

**-2 incorrectly read, right password, denied: FAIL**

**-1 incorrectly read, wrong password, accepted: FAIL**

|  |  |
| --- | --- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:1:pass_0_0.jpg | Sequence 1**-Correct read****-Correct password** PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  FIST area: 5727 bratio: 1.361 rratio: 0.707 mratio: 1.638  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  FIST area: 5823.5 bratio: 1.354 rratio: 1.359 mratio: 1.452  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  FIST area: 5861 bratio: 1.333 rratio: 1.337 mratio: 1.493  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  PALM area: 11449 bratio: 0.656 rratio: 0.567 mratio: 1.918  and: PALM area: 13087.5 bratio: 0.791 rratio: 1.587 mratio: 2.442  ----------------------  PASSWORD ACCEPTED!!!!!!  ---------------------- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:1:pass_0_1.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:1:pass_0_2.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:1:pass_0_3.jpg |

|  |  |
| --- | --- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:2:pass_2_0.jpg | Sequence 2**-Correct read****-Correct password** PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  POINT area: 6920 bratio: 0.638 rratio: 0.636 mratio: 2.131  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  FIST area: 7303.5 bratio: 1.089 rratio: 1.109 mratio: 1.871  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  PALM area: 14128 bratio: 0.760 rratio: 1.752 mratio: 2.628  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  PALM area: 11085.5 bratio: 0.638 rratio: 0.562 mratio: 1.804  and: PALM area: 14062 bratio: 0.750 rratio: 1.767 mratio: 2.581  ----------------------  PASSWORD ACCEPTED!!!!!!  ---------------------- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:2:pass_2_1.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:2:pass_2_2.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:2:pass_2_3.jpg |

|  |  |
| --- | --- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:3:pass_3_0.jpg | Sequence 3**-Correct read****-Correct password** PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  PALM area: 12934.5 bratio: 0.727 rratio: 1.843 mratio: 2.622  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  NONE area: 0 bratio: 0.000 rratio: 0.000 mratio: 0.000  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  PALM area: 12499 bratio: 0.703 rratio: 1.696 mratio: 2.137  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  PALM area: 11306 bratio: 0.585 rratio: 0.583 mratio: 1.393  and: PALM area: 12570 bratio: 0.722 rratio: 1.688 mratio: 2.253  ----------------------  PASSWORD ACCEPTED!!!!!!  ---------------------- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:3:pass_3_1.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:3:pass_3_2.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:3:pass_3_3.jpg |

|  |  |
| --- | --- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:4:pass_4_0.jpg | Sequence 4**-Correct read****-Correct password** PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  NONE area: 0 bratio: 0.000 rratio: 0.000 mratio: 0.000  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  NONE area: 0 bratio: 0.000 rratio: 0.000 mratio: 0.000  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  NONE area: 0 bratio: 0.000 rratio: 0.000 mratio: 0.000 |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:4:pass_4_1.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:4:pass_4_2.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:4:pass_4_3.jpg | Sequence 4 (cont.) PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  FIST area: 6602.5 bratio: 1.261 rratio: 1.264 mratio: 1.479  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  FIST area: 6659.5 bratio: 1.264 rratio: 1.267 mratio: 1.437  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  FIST area: 6650.5 bratio: 1.208 rratio: 0.803 mratio: 1.674  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  PALM area: 11708.5 bratio: 0.657 rratio: 0.577 mratio: 1.779  and: PALM area: 13538.5 bratio: 0.777 rratio: 1.676 mratio: 2.556  ----------------------  PASSWORD ACCEPTED!!!!!!  ---------------------- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:4:pass_4_4.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:4:pass_4_5.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:4:pass_4_6.jpg |

|  |  |
| --- | --- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:5:pass_0_0.jpg | Sequence 5****-Correct read********-Correct password**** PREPARE FOR CAPTURE IN....  3....  2...........  1...........  POINT area: 6152.5 bratio: 0.660 rratio: 0.657 mratio: 2.131  and: PALM area: 11655 bratio: 0.695 rratio: 0.563 mratio: 2.086  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  FIST area: 6540.5 bratio: 1.077 rratio: 1.078 mratio: 1.364  and: FIST area: 5740 bratio: 1.310 rratio: 1.347 mratio: 1.610  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  FIST area: 6479.5 bratio: 1.100 rratio: 1.101 mratio: 1.375  and: POINT area: 5740 bratio: 0.651 rratio: 0.615 mratio: 2.416  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  PALM area: 11484.5 bratio: 0.672 rratio: 0.571 mratio: 1.896  and: PALM area: 11762.5 bratio: 0.785 rratio: 1.566 mratio: 2.309  ----------------------  PASSWORD ACCEPTED!!!!!!  ---------------------- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:5:pass_0_1.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:5:pass_0_2.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:5:pass_0_3.jpg |

|  |  |
| --- | --- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:6:pass_1_0.jpg | Sequence 6 **-Correct read**  **-Incorrect password**  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  POINT area: 6333 bratio: 0.711 rratio: 0.672 mratio: 2.592  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  POINT area: 6303 bratio: 0.631 rratio: 0.628 mratio: 2.222  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  POINT area: 6443 bratio: 0.707 rratio: 0.665 mratio: 2.705  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  PALM area: 11158.5 bratio: 0.596 rratio: 0.594 mratio: 1.385  and: PALM area: 12773.5 bratio: 0.780 rratio: 1.652 mratio: 2.443  ----------------------  INTRUDER... INTRUDER....  ---------------------- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:6:pass_1_1.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:6:pass_1_2.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:6:pass_1_3.jpg |

|  |  |
| --- | --- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:7:pass_5_0.jpg | Sequence 7 **-Correct read**  **-Incorrect password**  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  POINT area: 7796 bratio: 0.651 rratio: 0.620 mratio: 2.300  and: PALM area: 14043 bratio: 0.542 rratio: 0.539 mratio: 1.422  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  FIST area: 8491.5 bratio: 0.921 rratio: 1.089 mratio: 1.721  and: FIST area: 8296.5 bratio: 0.969 rratio: 0.963 mratio: 1.913  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  PALM area: 13342.5 bratio: 0.716 rratio: 0.588 mratio: 2.082  and: POINT area: 7518.5 bratio: 0.654 rratio: 0.652 mratio: 2.116  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  PALM area: 12941 bratio: 0.596 rratio: 0.594 mratio: 1.348  and: PALM area: 15713 bratio: 0.734 rratio: 1.934 mratio: 2.780  ----------------------  INTRUDER... INTRUDER....  ---------------------- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:7:pass_5_1.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:7:pass_5_2.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:7:pass_5_3.jpg |

|  |  |
| --- | --- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:false positive:pass_1_0.jpg | Sequence 8\*\*\* **-Incorrect read**  **-False Positive Password**  **-Because of the metrics used, similar proportioned gestures will register a false positive.**  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  FIST area: 5569 bratio: 1.067 rratio: 1.067 mratio: 1.551  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  UNK area: 5392.5 bratio: 1.050 rratio: 1.065 mratio: 1.985  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  POINT area: 9024.5 bratio: 0.627 rratio: 0.624 mratio: 1.400  and: PALM area: 10638 bratio: 0.766 rratio: 1.843 mratio: 2.910  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  POINT area: 9174.5 bratio: 0.634 rratio: 0.631 mratio: 1.393  and: PALM area: 10668 bratio: 0.783 rratio: 1.863 mratio: 3.026  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  PALM area: 9004 bratio: 0.660 rratio: 0.574 mratio: 1.854  and: PALM area: 10273.5 bratio: 0.757 rratio: 1.676 mratio: 2.414  ----------------------  PASSWORD ACCEPTED!!!!!!  ---------------------- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:false positive:pass_1_1.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:false positive:pass_1_2.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:false positive:pass_1_4.jpg |
|  |

|  |  |
| --- | --- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:false negative 1:pass_1_1.jpg | Sequence 9\*\*\* **-Incorrect read**  **-False Negative Password**  **-Too much skin from the forearm corrupted the read of the fist, causing it to not accept the correct password.**  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  UNK area: 11727.5 bratio: 0.846 rratio: 0.845 mratio: 1.601  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  UNK area: 10081 bratio: 0.863 rratio: 0.862 mratio: 1.825  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  UNK area: 10019 bratio: 0.879 rratio: 0.869 mratio: 1.948  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  PALM area: 13215 bratio: 0.686 rratio: 0.597 mratio: 1.893  and: PALM area: 15177.5 bratio: 0.774 rratio: 1.697 mratio: 2.490  ----------------------  INTRUDER... INTRUDER....  ---------------------- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:false negative 1:pass_1_2.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:false negative 1:pass_1_3.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:false negative 1:pass_1_4.jpg |

|  |  |
| --- | --- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:false negative 2:pass_2_0.jpg | Sequence 10\*\*\* **-Incorrect read**  **-False Negative Password**  **-The program incorrectly judged a fist, palm, and then a set of two palms, until it ran up to the 6 sequence limit.**  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  POINT area: 5140.5 bratio: 0.836 rratio: 1.969 mratio: 5.809  and: UNK area: 13140 bratio: 0.621 rratio: 0.618 mratio: 1.430  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  FIST area: 6459.5 bratio: 1.208 rratio: 1.271 mratio: 1.908  and: FIST area: 6783 bratio: 1.151 rratio: 0.806 mratio: 1.907  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  UNK area: 6809 bratio: 1.173 rratio: 0.775 mratio: 2.084  and: POINT area: 6623 bratio: 0.645 rratio: 0.643 mratio: 2.340  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  UNK area: 12254 bratio: 0.655 rratio: 0.614 mratio: 1.619  and: PALM area: 13339.5 bratio: 0.768 rratio: 1.644 mratio: 2.374  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  UNK area: 12214 bratio: 0.642 rratio: 0.640 mratio: 1.379  and: PALM area: 13356.5 bratio: 0.771 rratio: 1.609 mratio: 2.333  PREPARE FOR CAPTURE IN....  3...........  2...........  1...........  PALM area: 12345 bratio: 0.667 rratio: 1.673 mratio: 1.750  and: UNK area: 13357.5 bratio: 0.644 rratio: 0.642 mratio: 1.459  ----------------------  INTRUDER... INTRUDER....  ---------------------- |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:false negative 2:pass_2_1.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:false negative 2:pass_2_2.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:false negative 2:pass_2_3.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:false negative 2:pass_2_4.jpg |
| Enki_Files:Documents:school:Visual Interfaces:projects:GesturePasswords:doc:img:false negative 2:pass_2_5.jpg |