**Main.cpp**

1. #define \_CRT\_SECURE\_NO\_WARNINGS
2. #include <iostream>
3. #include <algorithm>
4. #include <sstream>
5. #include <fstream>
6. #include <string>
7. #include <vector>
8. #include <iomanip>
10. #include <opencv2/opencv.hpp>
11. #include <opencv2/core/core.hpp>
12. #include <opencv2/highgui/highgui.hpp>
13. #include <opencv2/objdetect/objdetect.hpp>
14. #include <opencv2/imgproc/imgproc.hpp>
15. #include <opencv2/video/background\_segm.hpp>
16. #include <opencv2/imgcodecs.hpp>
17. #include <opencv2/videoio.hpp>
18. #include <opencv2/video.hpp>
20. //Programmer's created header files
21. #include "FaceSwapping.hpp"
22. #include "FaceDetect.hpp"
23. #include "FaceFilter.hpp"
24. #include "SaveFrame.hpp"
25. #include "ColorMapping.hpp"
27. using namespace std;
28. using namespace cv;

31. enum Mode { normal , swap , mask , save, colormap };
33. int main()
34. {
35. const unsigned int faceinframe = 2;
36. string text = "Enter ESC to exit";
37. FaceDetector facedetect("C:**\\**Users**\\**TCCOM**\\**Documents**\\**opencv**\\**sources**\\**data**\\**haarcascades**\\**haarcascade\_frontalface\_alt.xml", 0, faceinframe);
38. FaceFilter filter("Mask1.jpg");
39. FaceSwapper face\_swapper("C:**\\**Users**\\**TCCOM**\\**Documents**\\**dlib-19.4**\\**shape\_predictor\_68\_face\_landmarks.dat");
40. Mode mode = normal;
42. ColorMapping colormap(BONE);
44. facedetect.instruction();
45. try {
46. while (true)
47. {
48. Mat ColorMapFrame;
49. Mat Frame;
50. facedetect >> Frame;
51. SaveImage save(Frame, "Swapper.png");
53. putText(Frame, text, cvPoint(20, 20), FONT\_HERSHEY\_TRIPLEX, 0.8, cvScalar(0, 255, 0), 1, CV\_AA);
55. vector<Rect> cv\_faces = facedetect.faces();
57. switch (mode)
58. {
59. case normal:
60. break;
62. case Mode::swap:
64. if (cv\_faces.size() == faceinframe)
65. face\_swapper.swapFaces(Frame, cv\_faces[0], cv\_faces[1]);
66. break;
68. case Mode::mask:
69. for (int i = 0; i < cv\_faces.size(); i++)
70. {
71. Point center(cv\_faces[i].x + cv\_faces[i].width\*0.5, cv\_faces[i].y + cv\_faces[i].height\*0.5);
72. filter.CreateMask(Frame, center, Size(cv\_faces[i].width, cv\_faces[i].height));
73. }
74. break;
76. case Mode::colormap:
77. colormap.ConvertColor(Frame, Frame);
78. break;
79. }
81. imshow("Face Swap", Frame);
83. switch (cv::waitKey(1))
84. {
85. //Mode !
86. case 'a':
87. mode = Mode::swap;
88. break;
89. case 'b':
90. mode = Mode::mask;
91. break;
92. case 'c':
93. mode = Mode::colormap;
94. break;
96. //Action !
97. case '1':
98. filter.setMask("Mask1.jpg");
99. break;
100. case '2':
101. filter.setMask("Mask2.jpg");
102. break;
103. case 'q':
104. colormap.setColorTone(BONE);
105. break;
106. case 'w':
107. colormap.setColorTone(WINTER);
108. break;
109. case 'e':
110. colormap.setColorTone(OCEAN);
111. break;
112. case 'r':
113. colormap.setColorTone(SUMMER);
114. break;
115. case 't':
116. colormap.setColorTone(COOL);
117. break;
118. case 'y':
119. colormap.setColorTone(HSV);
120. break;
121. case 'u':
122. colormap.setColorTone(HOT);
123. break;
124. case 's':
125. save.setSaveFile();
126. save.SaveImagetoFile(Frame);
127. break;
128. case 27:
129. return 0;
130. break;
132. default:
133. break;
134. }
135. }
136. }
137. catch (exception& e)
138. {
139. cout << e.what() << endl;
140. }
141. }

**FaceDetect.hpp**

1. #pragma once
2. #define \_CRT\_SECURE\_NO\_WARNINGS
3. #ifndef FACEDETECT\_HPP
4. #define FACEDETECT\_HPP
5. #include "FaceFilter.hpp"
7. #include <opencv2/core/core.hpp>
8. #include <vector>
9. #include <string>
10. #include <memory>
12. namespace cv
13. {
14. class VideoCapture;
15. class CascadeClassifier;
16. }
18. class FaceDetector
19. {
20. public:
22. //Initializes detector with cascade file, initializes camera with camera index and sets number of faces to track
24. FaceDetector(const std::string cascadeFilePath, const int cameraIndex, size\_t numFaces);
25. ~FaceDetector();
27. //Return next frame and operate face detection
28. void operator >> (cv::Mat &frame);
30. // Return all face detected in the form of vector
32. std:: vector<cv::Rect> faces(); //
33. static cv::Rect doubleRectSize(const cv::Rect &rect, const cv::Size &frameSize);
35. //Face Detection and Tracking
36. void detect();
37. void track();
39. //Program's instruction
40. virtual void instruction();
42. private:
44. //VideoCapture for camera
45. std::unique\_ptr<cv::VideoCapture> CamFrame;
47. //Create Cascade Classifier used for facial detection
48. std::unique\_ptr<cv::CascadeClassifier> Face\_Cascade;
50. //Adjust the frame to smaller size in order to speed up the swap operation
51. cv::Mat AdjustedFrame;
53. //Fixed Width of the downscaled frame
54. const int FixedWidth = 256;
56. //Vector used to cover the face in a camera frame
57. std::vector<cv::Rect> FaceRects;
59. //Vector of the face vector used for tracking a face in the camera frame (Note : 1 vector per 1 detected face)
60. std::vector<cv::Rect> TempFacesRect;  //Stored a deteced face as the temp face rect
62. //Used to determin if the pixels are in region of interest
63. std::vector<bool>                       INRoi;
65. //Store the face as face template
66. std::vector<cv::Mat>                    FaceTemp;
68. //Rectangular area covering facial region of interest
69. std::vector<cv::Rect>                   FaceRegionOfInterest;
71. //Matching the output of face detected
72. cv::Mat                                 MatchingResult;
74. //Adjust framesize
75. cv::Size                                AdjustedSize;
77. //Original Frame Size
78. cv::Size                                OriginalFrameSize;
79. cv::Point2f                             RatioForAdjustment;
81. //Used to determine the status of face detection
82. bool                                    Istracking = false;
84. //Number of face being found in a frame
85. unsigned int FaceFound = 0;
87. //Control the time for face detection
88. //Time being measured in millisecond Thus, long long must be used
90. std::vector<long long>                  BeginAddface;
91. std::vector<long long>                  StopAddFace;
92. const double                            MaxTimeOnFrame = 2.0;
94. };
96. #endif

**FaceDetect.cpp**

1. #include "FaceDetect.hpp"
3. #include <opencv2/core/core.hpp>
4. #include <opencv2/video/video.hpp>
5. #include <opencv2/imgproc/imgproc.hpp>
6. #include <opencv2/objdetect/objdetect.hpp>
7. #include <opencv2/highgui/highgui.hpp>
9. #include <iostream>
11. FaceDetector::FaceDetector(const std::string cascadeFilePath, const int cameraIndex, size\_t numFaces)
12. {
13. try {
14. CamFrame = std::make\_unique<cv::VideoCapture>(cameraIndex);
15. }
17. catch (exception &e)
18. {
19. if (CamFrame->isOpened() == false)
20. {
21. std::cerr << "Failed opening camera" << std::endl;
22. cout << "Error: " << e.what();
23. exit(-1);
24. }
25. }
27. try
28. {
30. Face\_Cascade = std::make\_unique<cv::CascadeClassifier>(cascadeFilePath);
31. }
33. catch (exception &e) {
34. if (Face\_Cascade->empty())
35. {
36. std::cerr << "Error loading cascade file " << cascadeFilePath << endl <<
37. "Make sure the file exists" << endl;
38. cout << "Error: " << e.what() << endl;
39. system("pause");
40. //exit(-1);
41. }
42. }
44. OriginalFrameSize.width = (int)CamFrame->get(cv::CAP\_PROP\_FRAME\_WIDTH);
45. OriginalFrameSize.height = (int)CamFrame->get(cv::CAP\_PROP\_FRAME\_HEIGHT);

48. AdjustedSize.width = FixedWidth;
49. AdjustedSize.height = (AdjustedSize.width \* OriginalFrameSize.height) / OriginalFrameSize.width;
51. RatioForAdjustment.x = (double)OriginalFrameSize.width / AdjustedSize.width;
52. RatioForAdjustment.y = (double)OriginalFrameSize.height / AdjustedSize.height;
54. FaceFound = numFaces;
55. }
57. FaceDetector::~FaceDetector()
58. {
59. }
61. void FaceDetector:: instruction()
62. {
63. cout << "////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////" << endl;
64. cout << "////////////////////                   WELCOME TO PROJECT SNOOP PHOTO                            ///////////////////////" << endl;
65. cout << "////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////" << endl << endl <<endl;
66. cout << "**\t\t** This Project consist of 3 main features : " << endl << "**\t\t** 1 ) Face Swapper(Press A)" << endl << "**\t\t** 2 ) Mask (Press B)" << endl << "**\t\t** 3 ) Change Color (Press C)" << endl;
67. cout << endl << endl << endl << "**\t\t** There are some option to play with!!" << endl << "**\t\t** [[[[[[[[[   Mask Option   ]]]]]]]]]]" << endl << endl << "**\t\t** Press 1 : Toxic Mask" << "**\t\t** Press 2 : Wooden Mask" << endl << endl;
68. cout << endl << endl << "**\t\t**              Window Color Option              " << endl << endl << "**\t\t** Press q  : Bone " << endl << "**\t\t** Press w  : Winter" << endl;
69. cout << "**\t\t** Press e  : Ocean" << endl << "**\t\t** Press r  : Summer" << endl << "**\t\t** Press t  : COOL" << endl << "**\t\t** Press y  : HSV " << endl << "**\t\t** Press u  : Hot" << endl;
71. }
72. void FaceDetector::operator >> (cv::Mat &frame)
73. {
74. if (CamFrame->isOpened() == false)
75. {
76. frame.release();
77. return;
78. }
79. \*CamFrame >> frame;
81. cv::resize(frame, AdjustedFrame, AdjustedSize);
83. if (!Istracking) // Search for faces on whole frame until 2 faces are found
84. {
85. detect();
86. return;
87. }
88. else // if (m\_tracking)
89. {
90. track();
91. }
92. }
94. std::vector<cv::Rect> FaceDetector::faces()
95. {
96. vector<cv::Rect> faces;
98. faces.clear();
99. for (const Rect& face : FaceRects)
100. {
101. int AdjustX = face.x \* RatioForAdjustment.x;
102. int AdjustY = face.y \* RatioForAdjustment.y;
103. int AdjustW = face.width \* RatioForAdjustment.x;
104. int AdjustH = face.height \* RatioForAdjustment.y;
106. faces.push\_back(cv::Rect(AdjustX, AdjustY, AdjustW, AdjustH));
107. }
108. return faces;
109. }
111. void FaceDetector :: detect()
112. {
113. //Min face size is 1/5 of screen height
114. int getMinFaceWidth = AdjustedFrame.rows / 5;
115. int getMinFaceHeight = AdjustedFrame.rows / 5;
116. //Max face size must be 2/3 of screen height
117. int getMaxFaceWidth = AdjustedFrame.rows \* 2 / 3;
118. int getMaxFaceHeight = AdjustedFrame.rows \* 2 / 3;
120. Face\_Cascade ->detectMultiScale(AdjustedFrame, FaceRects, 1.1, 3, 0,
121. cv::Size(getMinFaceWidth, getMinFaceHeight), cv::Size(getMaxFaceWidth, getMaxFaceHeight));
123. if (FaceRects.size() < FaceFound)
124. {
125. return;
126. }
127. else if (FaceRects.size() >= FaceFound)
128. {
129. FaceRects.resize(FaceFound);
130. }
132. // Get face templates
133. FaceTemp.clear();
134. for (cv::Rect face : FaceRects)
135. {
136. face.width /= 2;
137. face.height /= 2;
138. face.x += face.width / 2;
139. face.y += face.height / 2;
141. //Duplicate the downscaledframe and put to face\_template
142. FaceTemp.push\_back(AdjustedFrame(face).clone());
143. }
145. // Get face ROIs
146. FaceRegionOfInterest.clear();
147. for (const cv::Rect& face : FaceRects)
148. {
149. FaceRegionOfInterest.push\_back(doubleRectSize(face, AdjustedSize));
150. }
152. // Initialize template matching timers
153. INRoi.clear();
154. BeginAddface.clear();
155. StopAddFace.clear();
157. INRoi.resize(FaceRects.size(), false);
158. BeginAddface.resize(FaceRects.size());
159. StopAddFace.resize(FaceRects.size());
161. // Turn on tracking
162. Istracking = true;
163. }
165. void FaceDetector :: track()
166. {
167. for (int i = 0; i < FaceRegionOfInterest.size(); i++)
168. {
169. const cv::Rect &roi = FaceRegionOfInterest[i]; // roi
171. //Min FaceWidthROI 4/10 of detected face & Min FaceHeightROI 4/10 of the detected face
172. int getMinROIW = roi.width \* 4 / 10;
173. int getMinROIH = roi.height \* 4 / 10;
174. int getMaxROIW = roi.width \* 6 / 10;
175. int getMaxROIH = roi.width \* 6 / 10;
176. // Detect faces sized +/-20% off biggest face in previous search
177. const cv::Mat &faceRoi = AdjustedFrame(roi);
178. Face\_Cascade ->detectMultiScale(faceRoi, TempFacesRect, 1.1, 3, 0,
179. cv::Size(getMinROIW,getMinROIH), cv::Size(getMaxROIW, getMaxROIH));
181. if (TempFacesRect.empty()) //In case that no face can be detected
182. {
183. if (BeginAddface[i] == 0) // Start the time for adding face
184. {
185. BeginAddface[i] = cv::getCPUTickCount();  // Use as timer
186. }
188. if (FaceTemp[i].cols <= 1 || FaceTemp[i].rows <= 1)
189. {
190. FaceRects.clear();
191. Istracking = false;
192. return;
193. }
195. // Template matching
197. cv::matchTemplate(faceRoi, FaceTemp[i], MatchingResult, CV\_TM\_SQDIFF\_NORMED);
198. cv::normalize(MatchingResult, MatchingResult, 0, 1, cv::NORM\_MINMAX, -1, cv::Mat());
199. double min, max;
200. cv::Point minLoc, maxLoc;
201. cv::minMaxLoc(MatchingResult, &min, &max, &minLoc, &maxLoc);
203. // Add roi offset to face position
204. FaceRects[i].x = minLoc.x + roi.x - FaceTemp[i].cols / 2;
205. FaceRects[i].y = minLoc.y + roi.y - FaceTemp[i].rows / 2;
206. FaceRects[i].width = FaceTemp[i].cols \* 2;
207. FaceRects[i].height = FaceTemp[i].rows \* 2;
209. StopAddFace[i] = cv::getCPUTickCount();
211. double duration = (double)(StopAddFace[i] - BeginAddface[i]) / cv::getTickFrequency();
212. if (duration > MaxTimeOnFrame)
213. {
214. FaceRects.clear();
215. Istracking = false;
216. return; // Stop tracking faces
217. }
218. }
219. else
220. {
221. INRoi[i] = false;
223. BeginAddface[i] = 0;
224. StopAddFace[i] = 0;
226. FaceRects[i] = TempFacesRect[0];
228. FaceRects[i].x += roi.x;
229. FaceRects[i].y += roi.y;
230. }
231. }
233. for (int i = 0; i < FaceRects.size(); i++)
234. {
235. for (int j = i + 1; j < FaceRects.size(); j++)
236. {
237. if ((FaceRects[i] & FaceRects[j]).area() > 0)
238. {
239. FaceRects.clear();
240. Istracking = false;
241. return;
242. }
243. }
244. }
245. }
247. cv::Rect FaceDetector :: doubleRectSize(const cv::Rect &inputRect, const cv::Size &frameSize)
248. {
249. cv::Rect outputRect;
250. // Double rect size
251. outputRect.width = inputRect.width \* 2;
252. outputRect.height = inputRect.height \* 2;
254. // Center rect around original center
255. outputRect.x = inputRect.x - inputRect.width / 2;
256. outputRect.y = inputRect.y - inputRect.height / 2;
258. // Handle edge cases
259. if (outputRect.x < 0) {
260. outputRect.width += outputRect.x;
261. outputRect.x = 0;
262. }
264. if (outputRect.y < 0) {
265. outputRect.height += outputRect.y;
266. outputRect.y = 0;
267. }
269. if (outputRect.x + outputRect.width > frameSize.width) {
270. outputRect.width = frameSize.width - outputRect.x;
271. }
272. if (outputRect.y + outputRect.height > frameSize.height) {
273. outputRect.height = frameSize.height - outputRect.y;
274. }
276. return outputRect;
277. }

**FaceFilter.hpp**

1. #pragma once
2. #ifndef FACEFILTER\_HPP
3. #define FACEFILTER\_HPP
4. #include <iostream>
5. #include <string>
6. #include <vector>
7. #include "FaceDetect.hpp"
8. #include <opencv2/objdetect/objdetect.hpp>
9. #include <opencv2/highgui/highgui.hpp>
10. #include <opencv2/imgproc/imgproc.hpp>
12. using namespace cv;
13. using namespace std;
14. class FaceFilter
15. {
16. private:
17. double MAX\_FACE\_SIZE = 200;
18. double MIN\_FACE\_SIZE = 20;
19. string facefile;
20. vector<Rect> faces;
21. Mat Mask;
22. Mat ResizedMask;
23. Mat TempSourceFrame;
24. Mat TransparentWhite;
25. Mat TransparentedMask;
26. Mat MaskAlpha;
27. public:
28. string getFaceFile() { return facefile; }
29. void setMask(string f) { facefile = f; }
30. void ReadMaskFile();
31. FaceFilter(const string facefile);
32. cv::Mat LocateTheMask(Mat frame);
33. ~FaceFilter();
34. cv::Mat CreateMask(Mat& source\_frame, Point face\_centre, Size face\_size);
35. };
36. #endif

**FaceFilter.cpp**

1. #include "FaceFilter.hpp"
3. FaceFilter:: FaceFilter(const string mask)
4. {
5. facefile = "C:**\\**Users**\\**TCCOM**\\**Desktop**\\**YEAR1 TERM2**\\**C++ Slide**\\**OPENCVTESTER**\\**OPENCVTESTER**\\**" + mask;
6. }
8. FaceFilter::~FaceFilter()
9. {
10. }
11. void FaceFilter::ReadMaskFile()
12. {
13. Mask = imread(facefile.c\_str());
14. }
15. cv::Mat FaceFilter:: LocateTheMask(Mat frame)
16. {
17. CascadeClassifier face\_cascade("C:**\\**Users**\\**TCCOM**\\**Documents**\\**opencv**\\**sources**\\**data**\\**haarcascades**\\**haarcascade\_frontalface\_alt.xml");
18. face\_cascade.detectMultiScale(frame, faces, 1.2, 2, 0 | CV\_HAAR\_SCALE\_IMAGE, Size(MIN\_FACE\_SIZE, MIN\_FACE\_SIZE), Size(MAX\_FACE\_SIZE, MAX\_FACE\_SIZE));
20. // Draw circles on the detected faces
21. for (int i = 0; i < faces.size(); i++)
22. {
23. MIN\_FACE\_SIZE = faces[i].width\*0.7;
24. MAX\_FACE\_SIZE = faces[i].width\*1.5;
25. Point center(faces[i].x + faces[i].width\*0.5, faces[i].y + faces[i].height\*0.5);
26. frame = CreateMask(frame, center, Size(faces[i].width, faces[i].height));
27. }
28. return frame;
29. }
31. cv::Mat FaceFilter::CreateMask(Mat& source\_frame, Point centre, Size face\_size)
32. {
33. ReadMaskFile();
34. cv::resize(Mask, ResizedMask, face\_size);
35. // ROI selection
36. Rect RegionOfInterest(centre.x - face\_size.width / 2, centre.y - face\_size.width / 2, face\_size.width, face\_size.width);
38. source\_frame(RegionOfInterest).copyTo(TempSourceFrame);
39. //Make white region transparent
40. cv::cvtColor(ResizedMask, TransparentWhite, CV\_BGR2GRAY);
41. cv::threshold(TransparentWhite, TransparentWhite, 230, 255, CV\_THRESH\_BINARY\_INV);
43. vector<Mat> maskChannels(3), result\_mask(3);
44. //Separate the resized mask from mask channel
45. cv::split(ResizedMask, maskChannels);
47. cv::bitwise\_and(maskChannels[0], TransparentWhite, result\_mask[0]);  // Get transparent mask
48. cv::bitwise\_and(maskChannels[1], TransparentWhite, result\_mask[1]);
49. cv::bitwise\_and(maskChannels[2], TransparentWhite, result\_mask[2]);
50. cv::merge(result\_mask, TransparentedMask);
52. TransparentWhite = 255 - TransparentWhite; // White color - mask itself
54. vector<Mat> srcChannels(3);
55. cv::split(TempSourceFrame, srcChannels);
56. cv::bitwise\_and(srcChannels[0], TransparentWhite, result\_mask[0]);  //Get black mask region
57. cv::bitwise\_and(srcChannels[1], TransparentWhite, result\_mask[1]);
58. cv::bitwise\_and(srcChannels[2], TransparentWhite, result\_mask[2]);
59. cv::merge(result\_mask, MaskAlpha);
60. cv::addWeighted(TransparentedMask, 1, MaskAlpha, 1, 0, MaskAlpha);  //(Source, alpha , maskalpha(src 2), beta, gamma, outputmask)
62. MaskAlpha.copyTo(source\_frame(RegionOfInterest));
63. return source\_frame;
64. }

**ColorMapping.hpp**

1. #pragma once
2. #define \_CRT\_SECURE\_NO\_WARNINGS
3. #ifndef COLORMAPPING\_HPP
4. #define COLORMAPPING\_HPP
5. #include <iostream>
6. #include <string>
7. #include <vector>
8. #include <iomanip>
9. #include <opencv2/opencv.hpp>
10. #include <opencv2/core/core.hpp>
11. #include <opencv2/highgui/highgui.hpp>
12. #include <opencv2/objdetect/objdetect.hpp>
13. #include <opencv2/imgproc/imgproc.hpp>
14. #include <opencv2/video/background\_segm.hpp>
15. #include <opencv2/imgcodecs.hpp>
16. #include <opencv2/videoio.hpp>
17. #include <opencv2/highgui.hpp>
18. #include <opencv2/video.hpp>
19. using namespace std;
20. using namespace cv;
22. enum ListColor
23. {
24. BONE = 1,
25. COOL = 2,
26. HOT = 3,
27. HSV = 4,
28. OCEAN = 6,
29. SUMMER = 10,
30. WINTER = 11
31. };
33. class ColorMapping
34. {
35. public:
36. ColorMapping(ListColor cmap);
37. void ConvertColor(Mat inputframe,Mat outputFrame);
38. void setColorTone(ListColor color);
40. private:
41. int colorframe;
42. Mat inputFrame;
43. Mat OutputFrmae;
44. ListColor colorlist;
45. };
46. #endif

**ColorMapping.cpp**

1. #include "ColorMapping.hpp"
2. ColorMapping::ColorMapping(ListColor cmap)
3. {
4. colorlist = cmap;
5. }
6. void ColorMapping :: setColorTone(ListColor color)
7. {
8. switch (color)
9. {
10. case BONE:
11. colorframe = 1;
12. break;
13. case WINTER:
14. colorframe = 2;
15. break;
16. case OCEAN:
17. colorframe = 3;
18. break;
19. case SUMMER:
20. colorframe = 4;
21. break;
22. case COOL:
23. colorframe = 6;
24. break;
25. case HSV :
26. colorframe = 10;
27. break;
29. case HOT :
30. colorframe = 11;
31. break;
32. }
33. }
35. void ColorMapping::ConvertColor(Mat inf, Mat outF)
36. {
37. try {
38. inputFrame = inf;
39. applyColorMap(inputFrame, outF, colorframe);
40. }
42. catch (runtime\_error &error)
43. {
44. std::cerr << "Error : " << error.what();
45. }
47. catch (out\_of\_range &another\_error)
48. {
49. std::cerr << "Out Of Range Error : " << another\_error.what();
50. }
51. }

**SaveImage.hpp**

KB

1. #pragma once
3. #ifndef SAVEFRAME\_HPP
4. #define SAVEFRAME\_HPP
6. #include <iostream>
8. #include <sstream>
9. #include <fstream>
10. #include <vector>
11. #include <string>
13. #include <opencv2/core/core.hpp>
14. #include <opencv2/video/video.hpp>
15. #include <opencv2/imgproc/imgproc.hpp>
16. #include <opencv2/objdetect/objdetect.hpp>
17. #include <opencv2/highgui/highgui.hpp>
19. using namespace std;
20. using namespace cv;
22. class SaveImage
23. {
25. public:
26. SaveImage(Mat& frame,string imagename);
28. SaveImage(string Path,string imagename, Mat& frame);
30. void setFolderPath(string Path);
32. void printFilename();
34. string getFolderPath();
36. Mat getFrame();
38. vector<int> getImageQuality();
40. void setImageName(string imagename) { filename = imagename; }
42. void SaveImagetoFile(Mat Frame);
44. string intToString(int number);
46. void setSaveFile();
48. private:
49. int imagecount;
50. int filenum;
51. vector<int> imagequality;
52. Mat Frame;
53. string folderpath;
54. string filename;
55. };
56. #endif

**SaveImage.cpp**

1. #include "SaveFrame.hpp";
2. #include <fstream>
4. SaveImage::SaveImage(Mat& frame,string imagename)
5. {
6. ifstream in\_file;
7. in\_file.open("C:**\\**Users**\\**TCCOM**\\**Desktop**\\**YEAR1 TERM2**\\**C++ Slide**\\**OPENCVTESTER**\\**OPENCVTESTER**\\**ProjectPic.txt");
8. if (!in\_file.is\_open())
9. {
10. throw exception();
11. }
12. in\_file >> imagecount;
13. in\_file.close();
14. folderpath = "C:**\\**Users**\\**TCCOM**\\**Desktop**\\**YEAR1 TERM2**\\**C++ Slide**\\**OPENCVTESTER**\\**OPENCVTESTER**\\**SavedImage**\\**" + imagename;
15. Frame = frame;
16. imagequality.push\_back(CV\_IMWRITE\_PNG\_COMPRESSION);
17. imagequality.push\_back(98);
19. }
21. SaveImage::SaveImage(string path,string imagename, Mat& frame)
22. {
23. folderpath = path + imagename;
24. Frame = frame;
25. imagequality.push\_back(CV\_IMWRITE\_PNG\_COMPRESSION);
26. imagequality.push\_back(98);
27. }

30. string SaveImage::getFolderPath()
31. {
32. return folderpath;
33. }
35. vector<int> SaveImage:: getImageQuality()
36. {
37. return imagequality;
38. }
40. Mat SaveImage :: getFrame()
41. {
42. return Frame;
43. }
45. string SaveImage :: intToString(int number)
46. {
48. std::stringstream ss;
49. ss << number;
50. return ss.str();
51. }
53. void SaveImage::setSaveFile()
54. {
55. string outnum = intToString(imagecount);
56. string filenum = outnum.c\_str();
57. folderpath = "C:**\\**Users**\\**TCCOM**\\**Desktop**\\**YEAR1 TERM2**\\**C++ Slide**\\**OPENCVTESTER**\\**OPENCVTESTER**\\**SavedImage**\\**Swapper" + filenum + ".png";
58. imagecount++;
59. ofstream out\_file;
60. out\_file.open("C:**\\**Users**\\**TCCOM**\\**Desktop**\\**YEAR1 TERM2**\\**C++ Slide**\\**OPENCVTESTER**\\**OPENCVTESTER**\\**ProjectPic.txt");
61. out\_file << imagecount;
62. out\_file.close();
63. }

66. //void SaveImage :: setSaveFile(int fnum)
67. //{
68. //  string outnum = intToString(fnum);
69. //  string filenum = outnum.c\_str();
70. //  folderpath = "C:\\Users\\TCCOM\\Desktop\\YEAR1 TERM2\\C++ Slide\\OPENCVTESTER\\OPENCVTESTER\\SavedImage\\Swapper" + filenum + ".png";
71. //
72. //}
74. void SaveImage :: SaveImagetoFile(Mat frame)
75. {
76. try {
77. bool SaveToFile = imwrite(folderpath, frame, imagequality);
78. }
79. catch (exception& e)
80. {
81. std::cerr << "Fail to Save Image because " << e.what();
82. }
84. }
86. void SaveImage::setFolderPath(string Path)
87. {
88. folderpath = Path;
89. }
91. void SaveImage::printFilename()
92. {
93. cout << folderpath;
94. }

**FaceSwapping.hpp**

**FaceSwapping.cpp**