2014

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

MIDO©, FACEMARK©, AND AGEME©: A SUITE OF TOOLS FOR FACIAL ANALYSIS AND PROCESSING DEVELOPED FOR THE FBI

Benjamin E. Barbour

A Capstone Project Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems & Operations Management

University of North Carolina Wilmington

2014

Approved by

Advisory Committee

Devon Simmonds

Elizabeth Baker

Karl Ricanek, Jr.

Chair

Accepted by

Dean, Graduate School

TABLE OF CONTENTS

Abstract

MIDO©, FaceMark©, and AgeMe©: A suite of tools for facial analysis and processing developed for the FBI. Benjamin E. Barbour, 2014. Capstone Project, University of North Carolina Wilmington.

The Multiple Image Dataset Organizer (MIDO) is an application designed to organize image datasets into a single uniform database format enabling robust search no matter the distribution method and source metadata format. It enables researchers and forensic analysis teams to quickly query for and analyze a subset of images across any dataset.

FaceMark allows users to annotate images by defining specific coordinates on a 2D image that represent areas of interest. This data can then be exported into other tools for use in computer vision related algorithms and data analysis tools.

AgeMe represents the culmination of 11 years of research in face analysis and processing. It implements age estimation, gender and race classification, and age progression algorithms in an easy to use application. A user simply supplies an image of a face and receives results automatically and in real-time.

Dedication

This work is dedicated to my wife, Megan, and son, Paul, who have put up with me working 25 hours a day 8 days a week for the past several years.

To my parents, Dr. Kimberly Howe & Dr. Carlie Barbour who have always instilled the value of education while letting me find my own path.

To my grandfather, Paul Howe (1923 - 2013), who showed me that it is never too late to go for it.

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

dataset. A set of data collected under similar conditions and for a specific purpose

metadata. Labels given to an image to describe the contents of the image

BCOE. FBI's Biometric Center of Excellence

DOB. Date of birth

DOA. Date of acquisition

EXIF. Exchangeable Image Format

FACS. Facial Action Coding System

MBDB. Multi-biometric Database

MIDO. Multiple Image Dataset Organizer

UNCW. University of North Carolina Wilmington

Chapter 1: Introduction

After using technology developed by UNCW's Face Aging Group to assist the FBI in their search for various suspects, it became necessary for the FBI to have access to these tools in-house. A project was developed to transition the applications from research-quality to commercial-quality implementations. With the help of West Virginia University's Office of Research Program Management and under the FBI's Biometric Center of Excellence (BCOE), UNCW delivered high quality software that could be used both in the field and the lab.

The three applications, Multiple Image Dataset Organizer (MIDO©), FaceMark©, and AgeMe© comprise a suite of tools developed for the FBI to address a growing need for sophisticated facial image management and automated analysis. MIDO provides a means to store and track images of faces and the associated metadata such as age, gender, race, expression, occlusions, etc. FaceMark allows a trained individual to annotate key landmarks (points of interest) for use in human analysis or automated machine algorithms. Lastly, AgeMe is a fully automated implementation of the facial analytics and processing tools developed over the last 11 years at UNCW's Face Aging Group. It can accept mugshot-like[1] images of faces and predict the age, gender, and race of the individual. It can also render an image that illustrates what the individual may look like as they get older or what they may have looked like when they were younger.

Two events in recent history have underlined the need for image management and automated analysis. August 6th, 2011, thousands of people across England rioted in response to the shooting death of Mark Duggan [1]. The riots lasted 5 days during which 5 people died, at least 213 were injured, including 5 police dogs. Looting generated an estimate of 200 million in property damage and loss. April 15th, 2013, 2:49pm two explosions rocked the site of the 117th Boston Marathon killing three and injuring 264 [2]. Two men left unattended backpacks on the ground containing home-made bombs inside pressure cookers and blended into the crowd to make their escapes.

[1] neutral expression, forward facing

These events have one thing in common. They were both heavily documented by video and still image capture. Great Britain has employed more CCTV cameras throughout its cities than any other country. At over 10,000 cameras in London, England alone, the sheer amount of data that must be processed is tremendous. Similarly, the Boston Marathon bombing was an event that was well documented by the participants. Spectators and journalists used a wide variety of devices ranging from professional high definition cameras to consumer hand held cameras, cell phones, and other capture devices. As we become an increasingly real-time, self-documented society, the need to quickly extract meaningful information from the sea of data is ever more important. In the case of the Boston Bombing, the FBI had all the information they required to find the terrorists on the first day, but it still took 3 days to identify the bomber.

This is the ”technology gap” MIDO is intended to fill. It differs from traditional consumer photo management tools in that it places the metadata as the priority and the image as a secondary attribute. This is quite similar to existing tools in law enforcement such as IAFIS which catalogs fingerprints of criminals or CODIS which catalogs DNA of criminals. The latest technology employed by the FBI is called NGI for Next Generation Identification System and it catalogs a large variety of biometric modalities including the face. MIDO differs from these tools in that it is not designed to track or store the identity of any particular person (though it does retain that capability). Its purpose is to catalog the metadata associated with the people in a particular group to help identify trends or anomalies that could provide answers faster and with less invasion of privacy.

Once a subset of images has been identified, they may be processed using FaceMark to identify key points on the face for facial analysis in AgeMe. FaceMark may be used in two modes. For scenarios where a face will be processed by a particular system, such as through a machine learning algorithm to predict or classify some attribute, it may be used in guided mode. In this mode, the points are predetermined and consistently placed across all images. FaceMark will guide the user through the process by indicating where the next point belongs. The second mode is called ”free” mode. As the name implies, it allows any

number of annotation points to be placed on an image, in any order or position. This could be as simple as identifying eye coordinates or more complex such as outlining an irregular object in an image such as a scar or tattoo. It can also be used for counting or identifying the positions of objects in the image such as counting the number of people in a crowd from an aerial photograph.

AgeMe is a prime example of a system that uses a specific landmark annotation scheme to perform classification and regression algorithms. Once an image has been annotated, it may be fed into AgeMe to produce a prediction of the persons age, gender, and race. From there, a user may adjust a slider to render representations of the person at any age across a lifetime. A new image may then be saved of the altered appearance and run through an existing facial recognition platform. Since, at this time, the vast majority of existing facial recognition algorithms are highly sensitive to the effects of aging, this ability is a distinct advantage. In the case where only an older image of an individual exists, an agent could progress the age of the face in the image forward and apply that new image to current imagery.

The suite of applications assists in solving the problem of managing large amounts of data and easily and quickly performing useful analysis to solve real problems. In the scenario of the Boston bombing or the riots across England, images can be cataloged by MIDO and analyzed by AgeMe to get a quick overview of the demographics specific to that group of individuals. From there, agents may narrow down the search to a specific subset of images rather than manually scanning the entire set. This not only improves efficiency but also increases privacy by automatically eliminating images that are not related to the search parameters.

In this paper I discuss the history, development, and futures of MIDO, FaceMark, and AgeMe. To avoid redundancy, the paper will focus primarily on MIDO while addressing the unique aspects of FaceMark and AgeMe. Chapter 2 discusses the development history of the suite and the problems they are designed to solve. In chapter 3 I discuss the requirements and features of each application. Chapter 4 outlines the design patterns used

to develop the applications and discusses thier implementation and development. Finally, chapter 5 concludes the work with a discussion about the features to be implemented in version 3.0 and into what domains this technology could be extended.

Chapter 2: Background

2.1 MIDO

MIDO (Multiple Image Dataset Organizer) began as a research tool in 2010 called MBDB (Multi-Biometric Database) and was used to bring together datasets gathered by a wide variety of sources into one unified, indexed, and searchable database. Each entry in a database may take the form of 2D images, 3D models, or 1D feature vectors. An additional time dimension may be also be included (for example, 2D or 3D video). Datasets are collected around the world by a number of different groups for various reasons. They range in size from just a few to several million or more entries and each contain a subset of metadata to describe and categorize the entry.

MBDB serves two purposes. Its primary purpose is to archive biometric data in a unified structure and enable searching across any dataset regardless of the modality or provided metadata. In addition, metadata can easily be updated. For example, in the case of 2D faces, annotating the presence and type of occlusion or providing the correct gender can be done with a single keystroke across one or more selected entries. Secondarily it enables a user to retrieve entries that fit a given criteria and then export the images along with a CSV file containing the associated metadata.

There are many public datasets available either for immediate download or by request/fee. The public version of each dataset contains the metadata collected and approved for distribution by that entity. For example, in the BioID [3] dataset, the only metadata provided are eye coordinates in a separate file per-image. The Bosphorus [4] dataset provides gender and some occlusion details in a separate file per-subject as well as a file naming system that encodes subject ID, expression, and pose. In addition, even though the file name may include FACS[2] encoding, there is a single file available separately that supersedes the filename with a certified FACS specialist's interpretations for all images in the dataset. FGNet [7], another popular dataset among researchers, provides subject ID and gender

[2]FACS is a research tool useful for measuring any facial expression a human being can make. It is an anatomically based system for comprehensively describing all observable facial movement [5, 6].

encoded in a custom filename format. FRGC [8] provides its metadata across several tab delimited files and a PostgreSQL dump. In just these four examples, it can be clearly seen that each dataset differs wildly in what meta data it chooses to provide and how it chooses to encode it making evaluating across datasets a challenging and time consuming task.

In 2013, the FBI contracted UNCW's I3S lab to develop tools to be used for analyzing faces in images and videos. As a part of project, MBDB was redesigned to focus on 2D images of faces and renamed MIDO. The purpose of MIDO was also changed from archiving and indexing public image datasets to collecting and analyzing image data across events and cases. The transition from an internal research tool to a professional software package included the addition of features to ease the management and configuration for a variety of platforms and configurations. In particular, its features were rewritten to support an older operating environment as was required for incorporation into the FBI's systems.

MBDB was originally designed as a 64-bit application for use on computers with at least 8GB of memory. As such, it preferred to download the entire database on startup and allowed very fast offline use. This became a significant bottleneck when the FBI required a 32-bit application. A limitation on 32-bit software is a maximum addressable memory space of 2GB. This address space is further limited by application and OS overhead leaving a maximum typical usable memory of 1.7GB. Under these restrictions, a maximum of approximately 400,000 results can be retrieved and stored in memory. This would place an artificial limit on the maximum number of images that MIDO can handle. As a result, version 2.0 moved away from allowing a user to browse the entire database and instead required the user to first search for a subset of records before being able to browse. The 32-bit limit still caps the browsable search results to approximately 400k however the database itself no longer has such restrictions. The number of records that MIDO can store in the database is limited only by the server architecture. This enables MIDO to scale extremely well based on standard Microsoft SQL Server design practices. When MIDO transitions to a 64-bit application, it will immediately benefit in speed and the capability to handle much larger results. In the mean time, in order to retrieve more than 400k results, it is

recommended to develop a custom application using the C++ Library, detailed in section 4.1.3, which is 32-bit and 64-bit compatible.

Existing photo management applications take a photo-centric approach where metadata is generally used to categorize images rather than describe the contents. Consumer software such as Apple's iPhoto, Google's Picasa, or Windows Live Photo Gallery provide easy access to images and even provide some limited ability to associate metadata. These applications are designed to index photos taken by cell phones and point & shoot cameras where images are organized by EXIF metadata such as date and time, location, or other camera settings. They also include features to automatically detect and recognize faces, group photos by location and time, and other functions that appeal to personal photo collections. They are not suited to indexing anything besides 2D images and do not provide the ability to customize or search by custom metadata. Other professional software such as Apple's Aperture or Adobe's Lightroom also provide advanced cataloging of extensive photo inventories but still fall short of the needs addressed by MIDO. For example, Apple's Aperture is does not allow more than 100,000 images. These tools seek to address the needs of a professional photographer and thus provide excellent tools to index photos based on photo shoots. Finally, many of these tools are migrating to a cloud solution where all your images are stored online. Giving companies access to your images and metadata may be against your personal or corporate policies.

2.2 Landmarks

Many popular computer vision algorithms require landmarks to identify features in a 2D image. Active Shape Models [9, 10, 11] (ASM), as well as those based upon this approach require quality ground truth landmarks to obtain accurate results. The simplest method for generating a ground truth is for a human expert to annotate each of a series of images with a set of corresponding points. In practice, however, this can be very time consuming [12, 13].

A landmark is a point of correspondence on an object that matches between and within populations [14]. They are represented as a set of pixel coordinates corresponding

to the location of the feature in a given image. A "good" landmark is one that can be consistently located from one image to another. For example, in an image of a face, the corners of the eyes or mouth would be considered good landmark locations as they are constant in every face. There are three classifications of landmarks, mathematical, anatomical, and pseudo [14].

2.2.1 Mathematical Landmarks

Mathematical landmarks correspond to locations on an object that may be defined by some mathematical property. These are often areas with strong edges such as areas of high gradience or curvature.

2.2.2 Anatomical Landmarks

Anatomical landmarks correspond to specific features on the face that represent an underlying biological feature. For example, cheekbones indirectly contribute to the variations between faces and may be modeled by an anatomical landmark. Some landmarks may be considered both anatomical and mathematical such as corners of the eyes and lips.

2.2.3 Pseudo/Intermediate Landmarks

Pseudo, or intermediate, landmarks are used to help define the boundaries between mathematical and anatomical landmarks. These landmarks are typically evenly spaced between mathematical or anatomical landmarks.

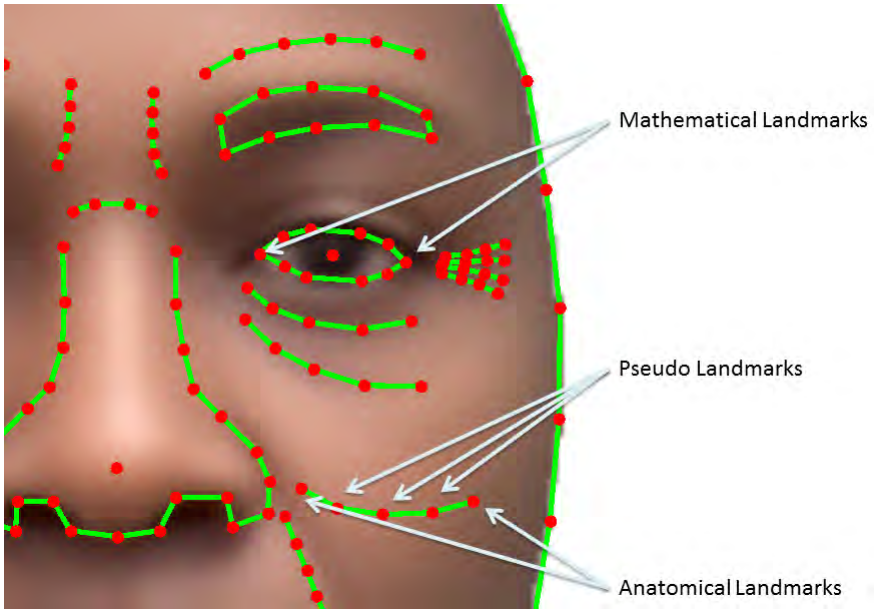

Figure 2.1: Good mathematical landmarks are areas of high curvature or junctions. Pseudo-points are equally spaced between landmarks [12]

.

2.2.4 Part

A part is a label given to a subset of landmarks corresponding to a sub-feature of the overall object being defined. In the case of a face, this may denote the eye, mouth, nose, or other such sub-component of the face. Parts form a polygon that follows a feature and may have a closed or open boundary. Parts are not typically used as components of an algorithm. They are instead used as visual cues for ground truthing landmarks to assist in correct landmark placement as seen in Figure 2.2 and above, denoted by the solid green lines, in Figure 2.1.

2.2.5 Shape

A shape is the quality of a configuration of points which is invariant under some transformation [12]. In many algorithms, the trained shape is the result of averaging each landmark across the training set following Procrustes Analysis. Procrustes Analysis is an algorithm that aligns a set of points by performing a series of rotations, scales, and translations on each shape with the objective of minimizing the total sum squared error

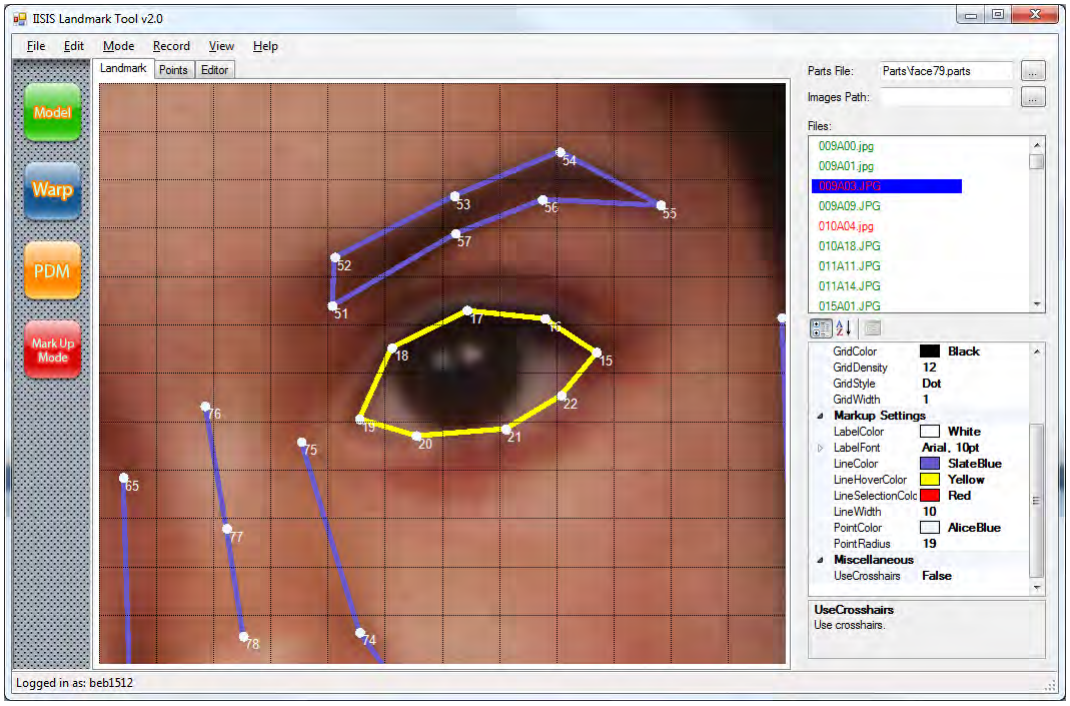

Figure 2.2: Example of a part, in this case the right eye (from the perspective of the viewer).

(TSSE) for the entire system where the error is the distance between each point. The result is a new representation of each shape under some transformation that represents the underlying shape in relation to the training set.

The shape X in k dimensions is often mathematically represented as each dimension concatenated to form a kn length vector where n is the number of points used to define the shape as shown in Equation 2.1 [15].

$$X = [x_1, ..., x_n, y_1, ..., y_n]^T \tag{2.1}$$

2.3 *FaceMark*

Tim Cootes, as part of his work on Active Appearance Models (AAM) [16], developed and released to the public a series of closed source tools to demonstrate the technology. One of these tools, called *am_markup* facilitates defining key landmarks on images. The idea behind *am_markup* is to manually annotate a few images, build a model, and use that model to automatically annotate subsequent images. If necessary, automatically anno-

tated images could be tweaked and a new model built in an iterative procedure to ideally produce better and better results. To accomplish this procedure there are 26 steps to follow, not including iterations. Some of these steps include manually editing text files that describe the files used in the process.

2.4 AgeMe

In 2003, Dr. Karl Ricanek Jr., Dr. Eric Patterson, and Dr. Midori Albert, identified, in a white paper [17], a challenging area of biometrics wherein current facial recognition technology performance degrades quickly when the age of the subject's enrolled image differs from the age of the probe image. They formed the Face Aging Group [18] at the University of North Carolina Wilmington and began research efforts to identify solutions to the problem. Prior to this, there was no publicly available research quantifying the performance characteristics of modern face recognition algorithms with respect to age variances between an enrolled template and an incoming probe.

2.4.1 Forensic Sketch Matching

Figure 2.3: Example of forensic age progression. Illustration by Teri Blythe[3].

The notion of rendering an illustration of what an individual might look like at an older age given their younger state is not new. Forensic artists have studied the aging process and evolved their craft to produce hand drawn representations of the individual at

[3]http://www.teriblythe.co.uk

various ages in order to assist in the search for an individual. The basic premise of face recognition is that, given two images of a person's face, can we say with confidence that the images are of the same person. To do this, there must be a way to measure the qualities and features of both images in such a way that those measurements may be accurately compared. To leverage the history of forensic face sketch artistry, algorithms have been developed to attempt to match camera photography to artist renditions. To solve the age problem in face recognition, an artist could sketch a face at the appropriate age for comparison to a photo and, if the matching algorithm were sufficient, produce a better match than two camera photos from different ages.

Figure 2.4: Forensic sketch of suspect (left) and photo (right). The artist, Paul Moody[5], assisted Palm Beach Sheriff's Office in capturing the suspect from a witness' description.

A variety of techniques have been employed such as attempting to extract features from both camera and sketch such that those features can be compared [19]. Another approach attempts to automatically create a version of the camera image that mimics a hand sketch [20] such that the computer generated sketch and the hand sketch may be compared. One issue with these approaches are that the most common dataset, Chinese University of Hong Kong Face Sketch (CUFS) [21], Figure 2.5, is that the artists drew the sketches using the source photo as reference. As a result, the camera photo and the sketch are very similar leading to artificially inflated accuracy results [22] that don't generalize to practices in the real world where the sketch is more likely to be based on a recalled verbal description or a collection of reference templates. The variations in a real world sketch are based on a large number of factors such as artist skill, emotional state of the witness, how

[5]https://www.linkedin.com/pub/paul-moody/10/8b0/387

long the witness was exposed to the face, the quality of that exposure (lighting, occlusion, duration, etc.) [23]. In the experiments performed on CUFS in [24], the authors show that the similarities between photo and sketch were so remarkable that one could achieve a high rate of match accuracy by comparing just the subject's hair line and showing an 85.22% success. Using such arbitrary regions and yielding high accuracy disproves the quality of the source dataset used in these experimentations.

Figure 2.5: Example forensic sketch from CUFS [21] dataset. Left column is original photo, right column is artist's rendition.

The lack of viable source data for forensic sketch to photo face matching, in addition to the amount of time it takes to produce a sketch, leads to the need for a different approach. Since 2003, research around the world has focused on the notion of synthesizing a photo-realistic render of a face at a different age from an exemplar photo using auto-

mated techniques. A texture based method such as this allows existing face recognition algorithms that are susceptible to the age problem a means to leverage existing technology by manipulating the inputs rather than the algorithms.

2.4.2 Synthetic Age Progression

There are two primary approaches to synthesizing an aged face. The first is a template-based, or texture replacement, approach where prototypical wrinkles are placed on the face to simulate the effects of aging. While these approaches may be seen in numerous mobile and web based apps, their effectiveness in producing accurate synthesized age progressions that improve the quality of face recognition has not yet been evaluated. The second is an appearance based approach where the shape and texture of a large set of faces is subjected to a statistical learning algorithm to build a model of the face age space.

The Face Aging Group has done extensive work in appearance based age progression using a unique approach based on Active Appearance Models (AAMs) developed by Tim Cootes et al [25]. In AAM, both shape and texture are vectorized and encoded using Principle Component Analysis (PCA). By manipulating the resulting vector, a new face may be synthesized that incorporates characteristics of aging. A key benefit to this approach is that it can adjust the face to a specific age making it very useful in forensic face recognition scenarios. The final result is a photo-realistic representation of the face as it might appear years later. The new face largely retains the characteristics of the original and thus is suitable for use in existing face recognition algorithms. In [26], it was shown that this approach, when used in conjunction with existing commercial algorithms improves the rank 1 accuracy by over 12% from 18.75% to 31.25%. Figure 2.6 shows the latest results in synthetic aging where two subjects, aged 21, were rendered at approximately 70 years of age.

Figure 2.6: Synthetic aging from approximately 18 to 70

The feature vectors extracted from the AAM process can also be used for a wide variety of facial analysis techniques when coupled with a classification or regression algorithm. In addition to synthetic age progression, the Face Aging Group has also explored algorithms such as age estimation as well as gender and race classification using the popular support vector machine (SVM) approach.

2.4.3 Age Estimation

The research community has proposed many different approaches to determining the age of an individual from an image of their face. Age estimation from facial imagery research began in 1994 with the work of Young Ho Kwon and Niels da Vitoria Lobo [27] who proposed that faces could be grouped into age categories based on ratios of facial geometry and the location and intensity of wrinkles. Despite advances in technology this geometric approach continues to be favored by some as in [28, 29] where ratios of facial

geometry are classified using an Artificial Neural Network (ANN) into age and gender categories. While these approaches are reminiscent of Bertillon's identification through body measurements system from the mid 1800's, other approaches take advantage of texture as well as shape. Fusing both global and local features, [30] reduces the high dimensional representation via Linear Discriminant Analysis (LDA) and classifies the face into age groups using K-nearest neighbor regression (KNN-R) to achieve an MAE as low as 5.8 on the FG-NET [7] database.

2.4.4 Race and Gender Classification

Automatic gender classification from facial images dates back to 1991 with the work of Golomb et al. [31] which employed a neural network approach to classify 90 photos of young adults (45 male, 45 female). Results showed that the neural network performed similarly to humans in that that when humans guessed the sex of an unseen face correctly, it likewise guessed the sex of the face. When humans struggled however, it also failed to perform adequately.

Gender classification is usually approached as a two-class (male and female) or a three-class (male, female, and unknown) classification problem. Race classification is often approached as an N-class classification problem where N represents the number of races in the system. Approaches to these two problems are often similar and represent a prototypical computer vision classification problem. Frequently this involves an algorithm structured as face detection, registration/alignment, feature extraction, and classification. The difference in approach is commonly a choice in which algorithms to use to solve each of the components. In [32], they used a boosting nested cascade detector for face detection, local texture classifiers for alignment, and two methods for feature extraction: AAM and affine transforms using coplanar facial points, and three classification methods: SVM, FLD, and Real Adaboost. They concluded that performance was best using appearance based features with SVM.

Age, gender, and race prediction are of great interest to those who are less interested in the identity of an individual and more concerned with the demographics of a group.

Automated approaches allow the collection and analysis of such data without intruding into the privacy of the individual.

Chapter 3: Features

The development of the facial processing suite did not go through a proper requirements phase as it was originally an in-house tool to address specific problems. The requirements therefore have been established through an iterative revision process between the stakeholders (I3S, WVU, FBI). It has gone through three phases of iterative feature revisions each followed by one month of ”feature lock” where a version is considered feature complete and further revisions are postponed to the next version. This document details the latest revisions, currently version 2.6.

3.1 MIDO

The major features of MIDO are as follows:

- Microsoft Windows client w/MS SQL database
- Search
- Browse & edit
- Export images & data
- Single & bulk import
- Robust automated installer
- C++ Library
- Plugin Architecture

MIDO is a 32-bit Microsoft Windows application developed using C# 4.5. It is restricted to the Windows platform as it cannot be compiled using the Mono [33] project for Apple’s OSX or Linux due to its use of the System.Drawing and System.Windows.Forms namespaces which, at the time of writing, are not properly emulated to maintain cross-platform compatibility. MIDO also makes use of propriety, closed-source, 3rd party GUI components developed by Infragistics [34] which are likewise, not cross-platform compatible. The system is composed of the MIDO client and a Microsoft SQL Server 2008

database on the backend. To adhere to the FBI's requirements, the application is designed to maintain compatibility from Windows XP through Windows 10. Likewise, the database interop is tested compatible from Microsoft SQL Server 2008 R2 to 2014.

Search functionality in the client is implemented by providing a simple graphical interface allowing the user to search by dataset, age, race, and gender. Based upon user selection, the client identifies whether there is an optimized stored procedure available for use. If one exists, it uses the stored procedure and returns the results. Should no stored procedure fit the search criteria, a custom SQL query will be generated and used instead.

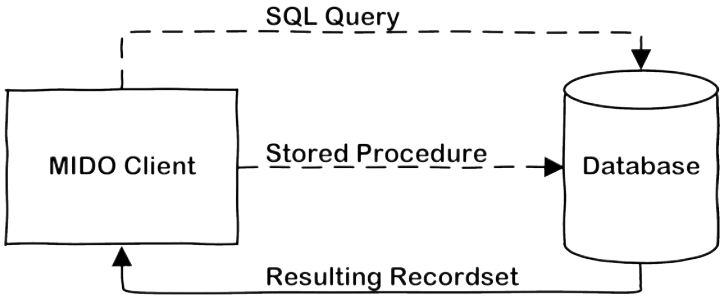

Figure 3.7: MIDO SQL Interop

Once a subset of images have been returned, the user may then browse the search results and mark images for export. In addition, the images can be annotated using the metadata toolbar tab and specifying gender, race, occlusions, and facial expressions. Once updated with the correct metadata, the images will show up using the appropriate refined criteria in the search process.

Another useful feature of MIDO is the statistics quick-look. Interactive charts and graphs illustrate the demographic distribution of the search results showing the age, gender, and race distribution across the results.

Users may import data in two ways. Images may be imported one at a time or in bulk. Individual images are useful when inputting images acquired in the field or when there are very few images in a dataset. Most of the time however, it is more practical to build a CSV describing the entire dataset and import the data in bulk. Bulk import is achieved by creating a CSV file with all necessary fields and then linking the field in a GUI

to the corresponding fields in the database.

Data is exported from MIDO by marking the desired images and choosing Export. When exporting, a target directory for images is specified along with a choice of up to three database fields to categorize the data in sub-directories. In addition to images, a CSV or tab delimited file may be generated by choosing the desired metadata fields. Since MIDO attempts to maintain the original file integrity, exporting images across datasets may mean that images are stored in a variety of formats across the marked images. Users may choose to convert the images to Windows BMP, PNG, JPG, or TIF upon export. Likewise, images may be automatically resized to a specified size. If maintaining proportions while resizing the images, MIDO will attempt to fit the image inside the specified dimensions by scaling along the dimension that maintains the largest image area.

Since MIDO is a professionally delivered software application it must be trivial to install and configure on a wide variety of unknown and unpredictable environments. In addition, the FBI computers operate in a restricted, secured, and most importantly , offline environment. As a result, any prerequisites for any valid configuration must be identified and included during installation. MIDO uses the Nullsoft Scripted Installation System (NSIS) version 2.46 [35] to accomplish this. The installer detects the operating system, checks for the existence of prerequisites, and installs all necessary components of MIDO with as little input from the user as possible. Advanced users however, maintain the option to fully configure their environment.

MIDO supports two installation types. The simplest is the offline stand-alone installation which installs and configures MIDO automatically for an environment that includes the client and the database server on the same machine. By default this is a secured private installation that does not rely on any outside network communication. For the advanced user or in a multi-user environment, the installer may be set to instead target a remote MS SQL server. The installer in this case requires administrative rights to the database server in order to create and populate the MIDO database.

The MIDO SQL server database should never be modified directly as this could

risk breaking the client. However, should developers wish to write their own utilities, a C++ library is provided that allows querying the database and returning results as a custom class. The library also returns the image as an OpenCV [36] matrix and uses Boost [37] for file IO and multi-threaded operation.

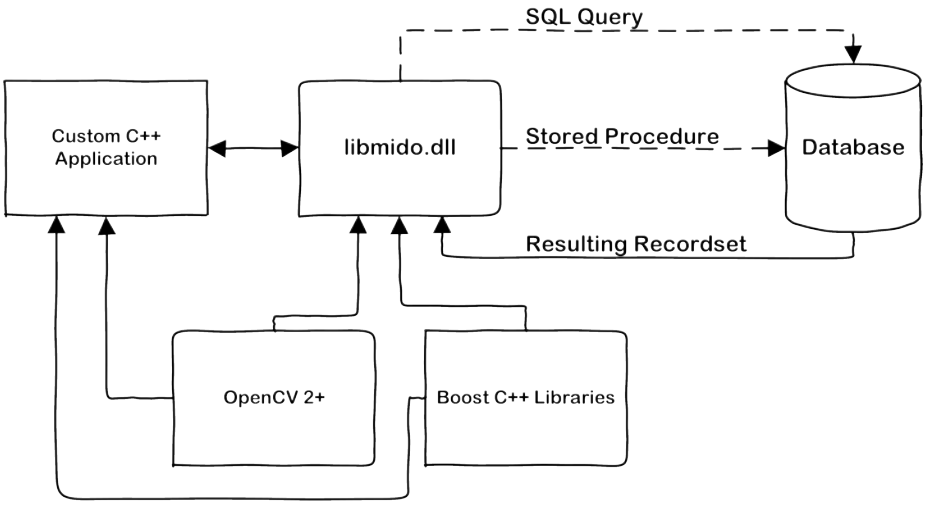

Figure 3.8: C++ Library Overview

MIDO 2.6 introduces a plugin system allowing end users to develop custom client plugins with the full power of the .NET framework. Any .NET language may be used to develop plugins that may interact with the search results, marked images, and even directly modify the currently displayed image. Following a few simple guidelines, users may develop powerful plugins to perform virtually any task imaginable directly within the MIDO client. For example, a plugin could be written to call a 3rd party face recognition algorithm directly on the search results, collect the scores, and return a similarity matrix without the need to first export the data and run a 3rd party application.

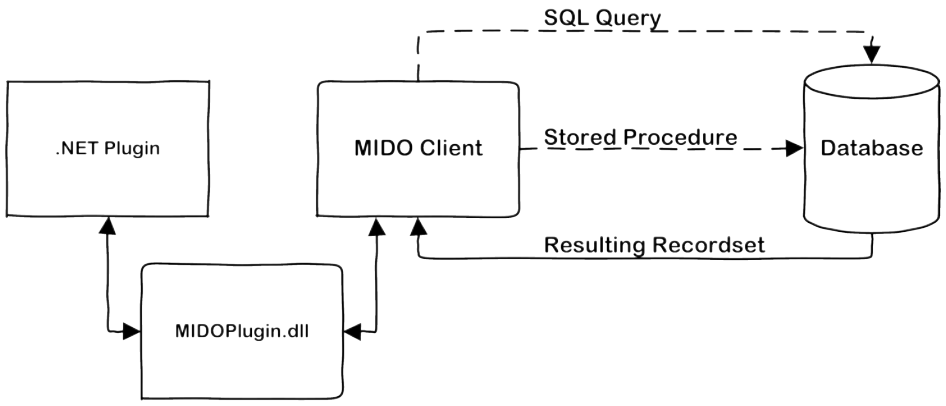

Figure 3.9: .NET Plugin Library Overview

Plugins have direct access to the images and utilize a custom image class called *Picture*. The .NET framework does not have a simplified, native way to directly access the pixels of an image in memory for large scale manipulation. The existing implementation is extremely slow. The *Picture* class handles a wide variety of direct access methods with realtime direct access to pixel bytes or as a .NET Color class. Images may be interchangeably accessed as flattened 1D arrays, 2D row/column matrices, or even 3D color matrices. Color may be addressed directly as grayscale, RGB, or HSV. In addition to image access methods, the plugin system provides access to a multi-threaded optimized 2D FFT implementation that can be referenced using the provided Complex number class or conveniently as interpolated integers. As a result, images may be analyzed in the frequency domain just as easily as the spacial domain.

Not only can a MIDO plugin directly manipulate both metadata and the currently displayed image, but a plugin can also manipulate the MIDO client itself. This can be used to apply custom themes or extend the functionality of the client. For example, a custom import or export routine could be implemented to handle a proprietary data format not currently supported by MIDO.

Finally, a dataset conversion utility is provided that converts between 24 different public and private research datasets. Since most of these datasets cannot be distributed with MIDO, a tool to convert their natively distributed formats into MIDO's import format

is provided. This tool is as simple as providing the folder for the dataset and the name of the CSV file to output. Once generated, the CSV file may be imported using MIDO's bulk import utility.

3.2 FaceMark

FaceMark began as an answer to some of the challenges and limitations presented in *am_markup*, described in Chapter 2.3. Firstly, the process of annotating images needed to be simplified but maintain backwards compatibility with the industry standard tools as established by Cootes. Secondly, the connection between annotating images and using an automated annotation feature needed to be separated. Finally, the actual process of using the tool should be easy and familiar.

The requirements for FaceMark include features to facilitate establishing quality ground truth landmarks for facial images to be processed by AgeMe. In addition, FaceMark needs to retain the ability to be used in the future under unknown circumstances on unknown images for unknown annotation schemes. As a result, it needs the flexibility to both operate in a constrained environment where the annotation scheme must be strictly adhered as well as the scenario where landmarks may need to be established for some other purpose. These alternate uses may include placing individual landmarks on points of interest, drawing bounding boxes around a region of interest, or drawing complex contours around a particular object.

The major features of FaceMark are, therefore:

- Microsoft Windows compatible
- Annotate an image with no existing landmarks
- Edit existing landmarks
- Freely place landmarks on an image
- Annotate images with strict guidelines
- Generate new guideline files

- Customize the colors and attributes of the points and lines on the image
- Save results in the file format defined by Tim Cootes' tools
- Copy/Paste list of coordinates to/from a spreadsheet or text editor

Like MIDO, FaceMark is also a 32-bit Microsoft Windows application developed using C# 4.5. It maintains compatibility from Windows XP to Windows 10.

The desired workflow for annotating images is to establish an annotation scheme and guide the user through the annotation process using that scheme. An annotation scheme is defined as a set of labelled points grouped by their relationship to a specific feature to be described. For example, four points could be used to define an enclosed rectangle that bounds the mouth while a single point would describe the tip of the nose as in Figure 3.10.

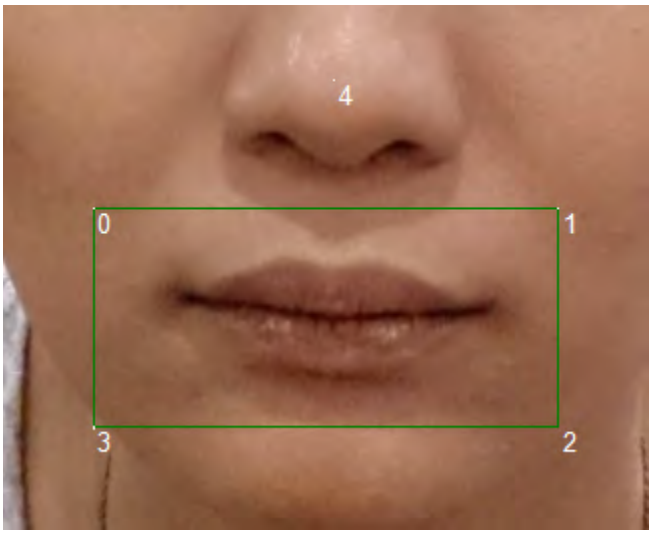

Figure 3.10: Example free mode annotation

3.3 AgeMe

AgeMe is required to implement advanced machine learning algorithms for face processing. It needs to do so in as simple a manner as possible. Users should be able to produce reliable results by simply loading an image. Should the automated process not produce acceptable results, a means of manually adjusting the landmarks should be built into the AgeMe application.

The major features of AgeMe are:

- Microsoft Windows compatible
- Age Estimation (between the ages of 18 and 60)
- Gender Classification (between Male and Female)
- Race Classification (between Caucasian and African-Descent)
- Age Progression/Regression (between 18 and 60)
- Automatically annotate images and allow manual editing
- Export the annotations
- Export the age progressed image
- Automatically batch process a directory of images

AgeMe is a 64-bit Microsoft Windows application developed using both C# 4.5 and Microsoft C++, both managed and unmanaged. It maintains compatibility from Windows 7 to Windows 10. The choice to use 64-bit architecture over 32-bit is due to the memory requirements and speed benefits.

4.1 MIDO

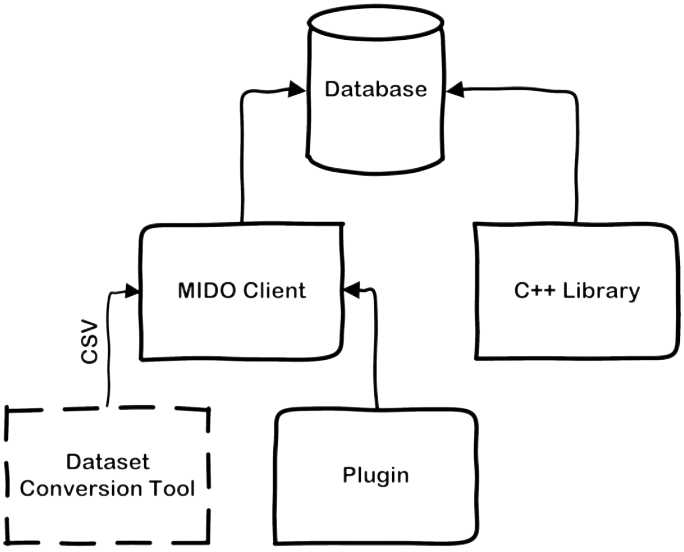

Figure 4.11: MIDO Solution Diagram

4.1.1 Database

An interesting challenge with database design is that each dataset contains a variety of types of metadata. For example, some include just the images, while others include metadata such as race or gender. A select few include detailed information such as azimuth, elevation, and lumens for the lightning used in the image acquisition. The database must be able to handle this metadata efficiently. A trade-off decision was made between complete user-customization and robust, compact data storage. As such, when a new type of metadata is encountered, special allocation is made in the database. To facilitate easy expansion of the database, it is divided into two logical groups, ”Subjects” and ”Image Metadata.” Figure 4.12 provides an overview of MIDO v2.5’s database. Each type of image metadata is a table associated with the primary Images table.

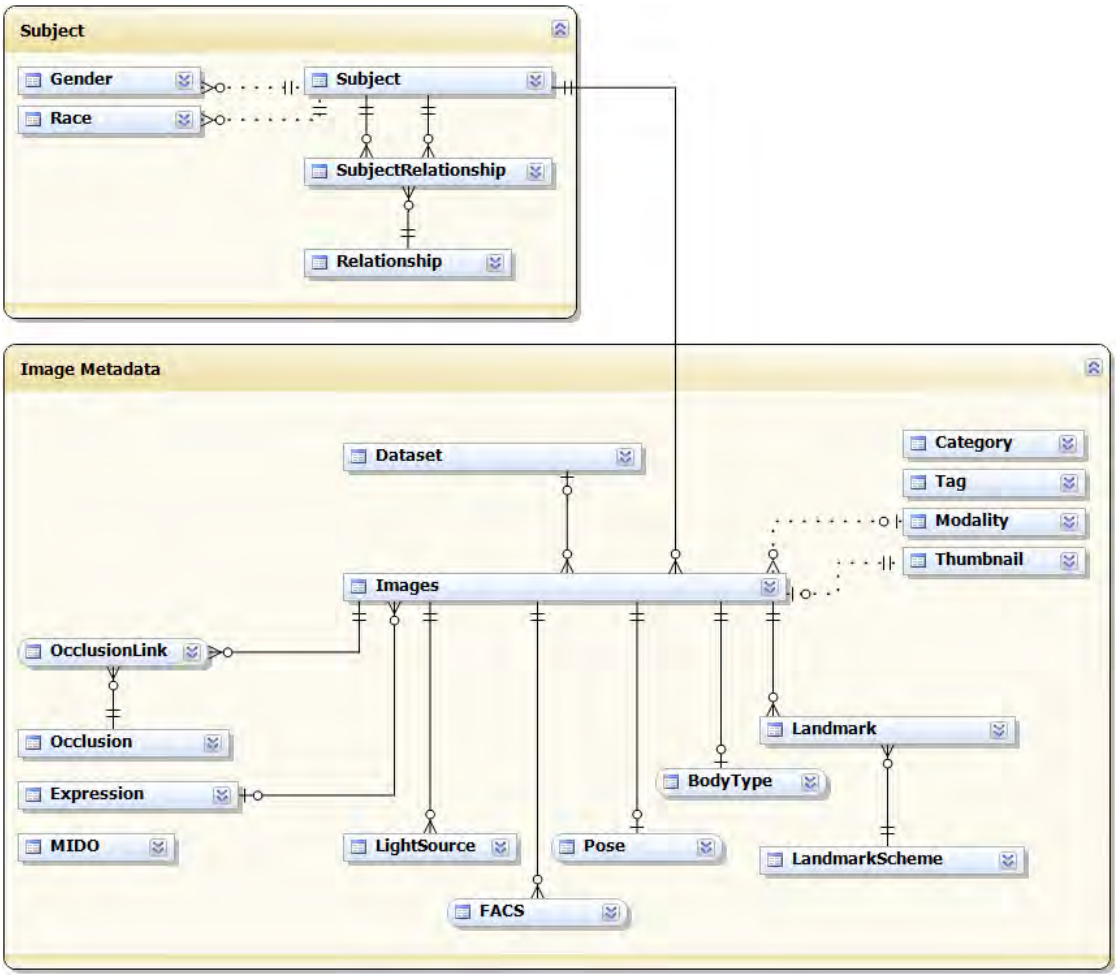

Figure 4.12: MIDO Database Diagram Overview

A subject describes the unchanging metadata for a single person. This includes things like gender, race, date of birth, and contact information. In addition, subjects may be related to eachother via the SubjectRelationship table which connects two subjects by a relationship type.

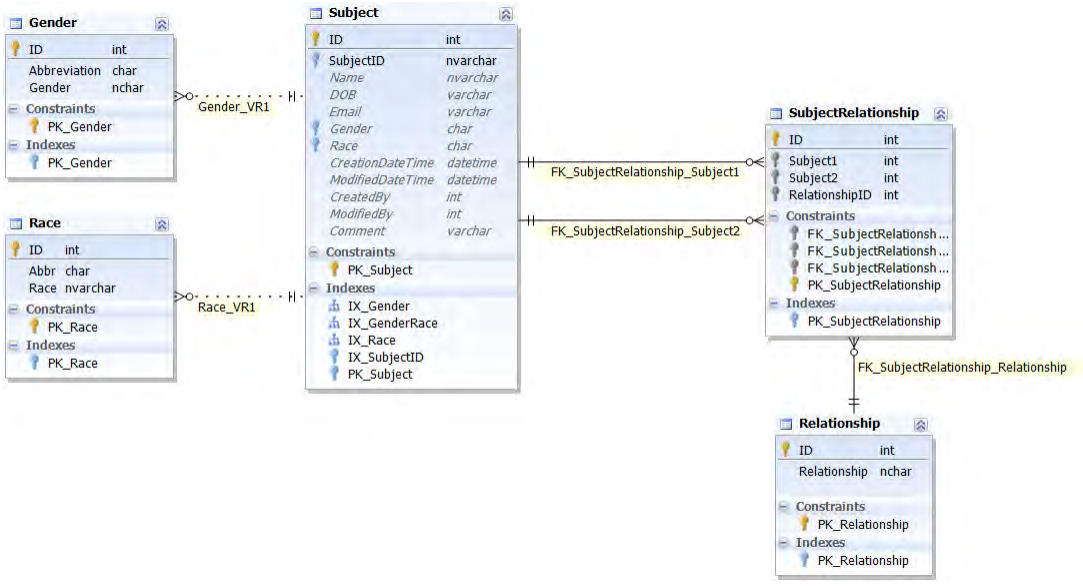

Figure 4.13: MIDO Database Subject Diagram Detail

An Image describes a single image in the dataset and all associated metadata. Each dataset provider chooses to what level of detail an image is described within their dataset. As a result, to conserve storage space in the database, less frequently used metadata is placed in optional tables. Should a dataset not provide a particular piece of information, it need not be stored as blank in the database. In addition, the images themselves are not stored in the database. Instead, a relative URI is stored that references the image from a given root. Since images may be stored on a networked server, the path to that server may be custom to each client machine. As a result, the MIDO client uses a user configurable ”base path” which it then concatenates with the path stored in the database to form an absolute reference to the image location.

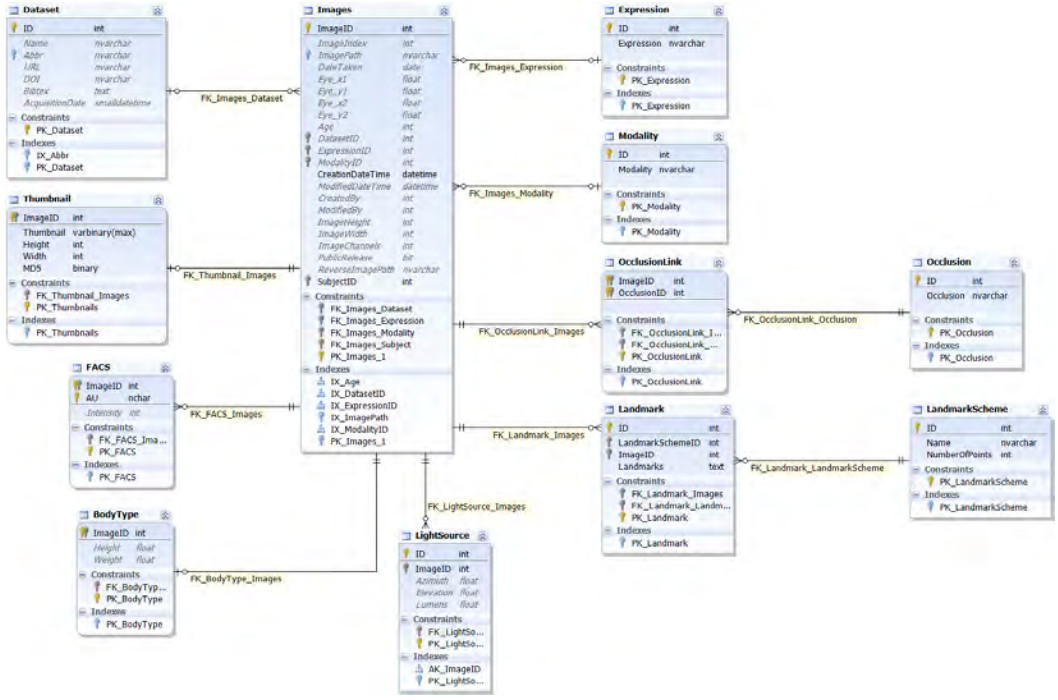

Figure 4.14: MIDO Database Images Diagram Detail

4.1.2 Client

The MIDO client application is the primary interface to the MIDO Database. It is developed using the Windows Forms architecture and implements visual controls supplied by Infragistics [34] in the NetAdvantage Suite. These 3rd party controls include the ribbon menu bar which emulates the look and feel of the menuing system introduced in Microsoft Office 2007, a robust grid layout tool that displays database search results, and a consistently themed status bar along the bottom to provide feedback to the user. In addition, a variety of other tools were used including charting, hierarchical tree views, and drop down lists. An early design mockup of MIDO featured a simple user experience where images were grouped by person, Figure 4.15 however, as it evolved, it quickly outgrew this and became a robust database front end that could facilitate collecting and referencing images for experimentation, shown in Figure 4.16.

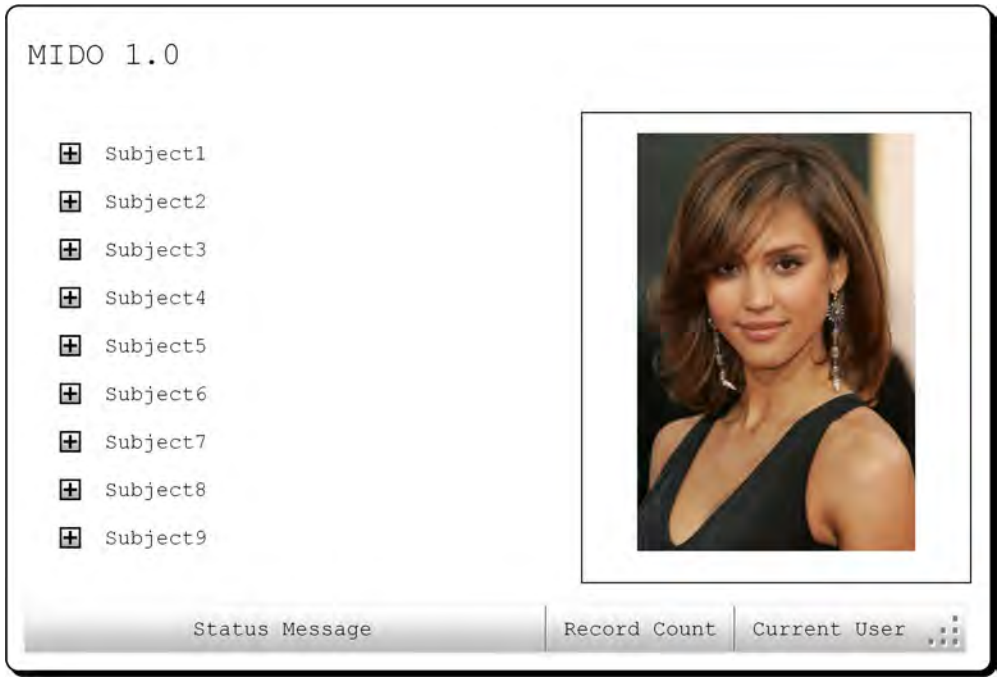

Figure 4.15: MIDO v1.0 Mockup

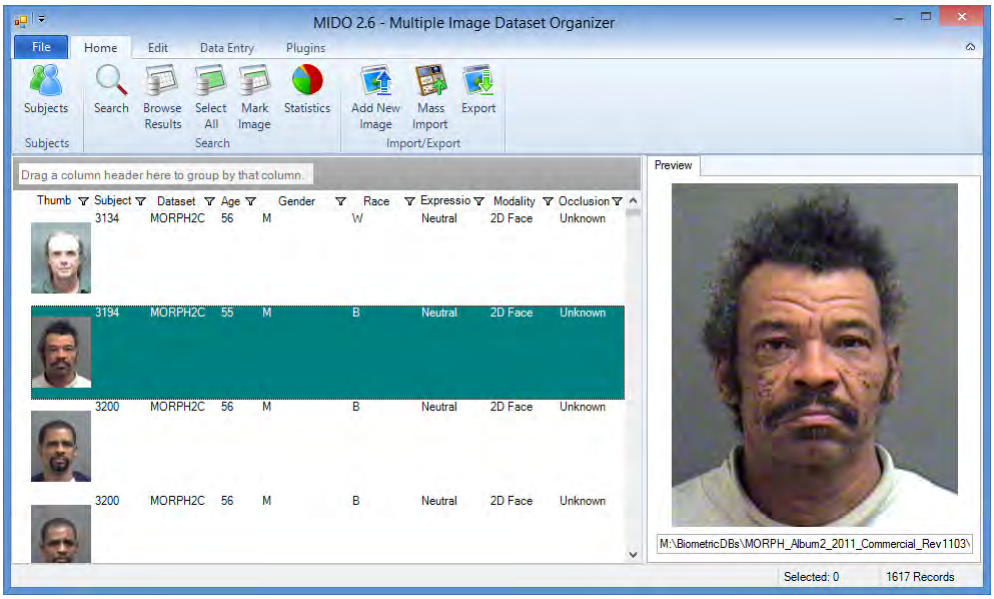

Figure 4.16: MIDO v2.5

The toolbar in MIDO provides navigation and editing functionality. The ”Home” bar, Figure 4.17, has buttons to navigate between different views as well as import and export images and metadata. The ”Edit” bar, Figure 4.18, contains quick edit buttons to

update the metadata of the currently selected image. Another bar, the ”Data Entry” bar, Figure 4.19, contains buttons to configure the various pieces of global metadata available in MIDO such as a list of genders, races, datasets, categories, and tags. A fourth menu, not pictured, called ”Plugins”, appears only when a plugin has been loaded into MIDO. Custom plugins, discussed in Chapter 4.1.7, may specify a custom menu option type for each function it implements.

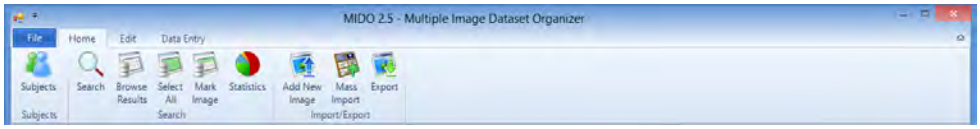

Figure 4.17: MIDO Ribbon Toolbar - Home

Figure 4.18: MIDO Ribbon Toolbar - Edit

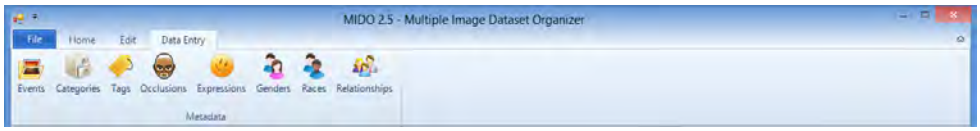

Figure 4.19: MIDO Ribbon Toolbar - Data Entry

When the client is first launched, it presents the user with a simplified search interface, Figure 4.20. This interface allows the user to specify which dataset to draw images from and a quick filter for age, race, and gender. When the user has specified their choices, the database is queried and any found records are returned.

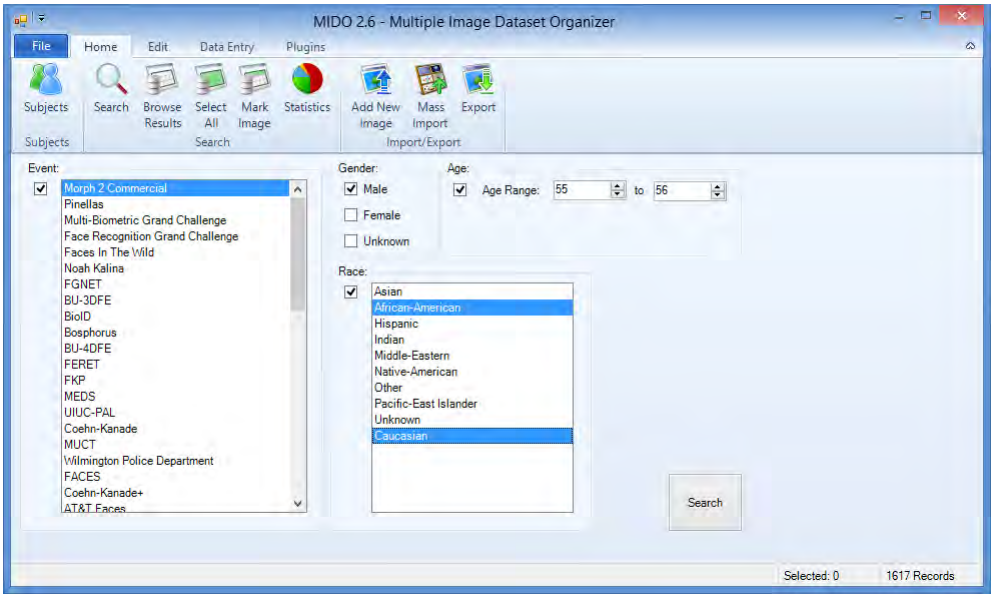

Figure 4.20: MIDO Simple Search Interface

At the heart of the client is the search results which are laid out using a grid view as shown in Figure 4.21. A grid view is similar to a table wherein records are laid out in row order. A thumbnail is shown on the left and metadata is distributed across the row. Inside the grid interface are several helpful sorting, filtering, and categorization tools to help a user narrow down on a particular subset of images. Sorting is accomplished by simply clicking on the header of the row you wish to sort by. For multiple rows, hold down shift and click the header in the order of nested sorts. For example, to sort by age and then by race, first click age and, while holding shift, click race. Any number of nested sort orders may be specified this way. To filter, click the image of a funnel to the right of the column you wish to filter, shown in Figure 4.22. A drop down menu will appear listing options including ”All” to remove any filtering, ”Custom” to create a robust filtering criteria based on a large collection of logical conditions, ”Blanks” to show only rows that do not contain metadata in this column and conversely, ”Non Blanks” to show only rows that do have values in this column. The remainder of the list of choices are the distinct values currently present in the column. In the case of gender for example, this could be an ”M” and an ”F” and perhaps a ”U”. By choosing one of these options, you can filter to rows that match a specific value.

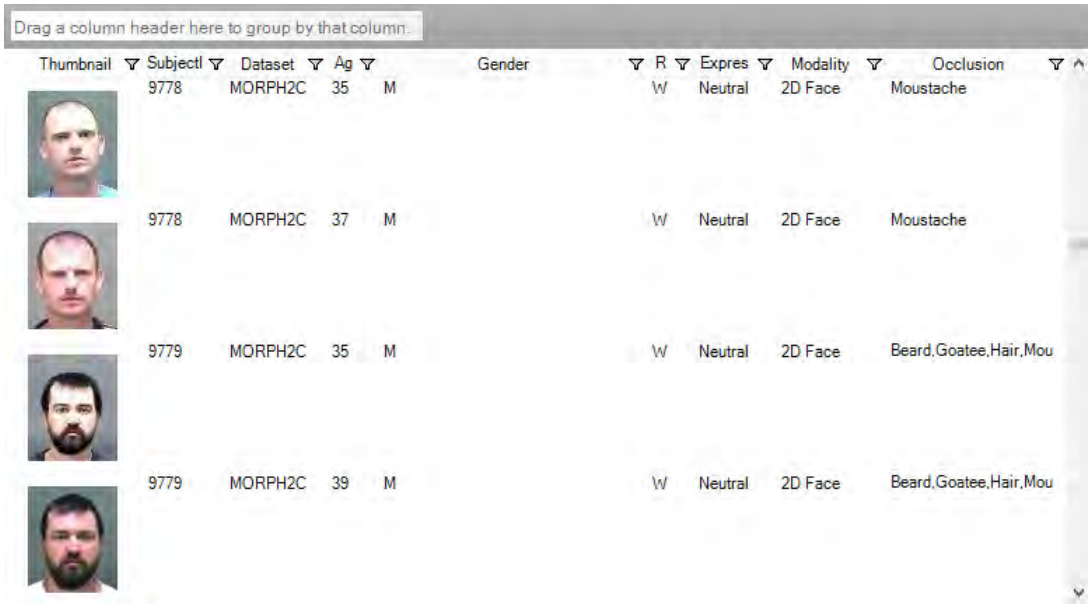

Figure 4.21: MIDO Search Results Grid View

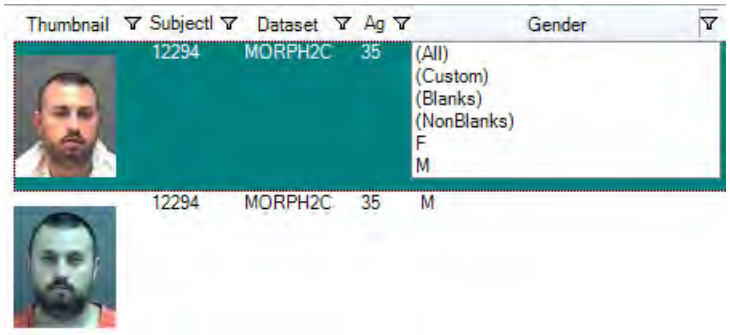

Figure 4.22: MIDO Grid Filter Example

The final method of customizing the view of the grid is to group rows together based on a value in a column, shown in Figure 4.23. For example, you can group records by gender which would create two groups of rows. Unlike filtering, this allow selective hiding and unhiding of any combination of grouped rows. Like sorting, grouping by column allows nested grouping. MIDO provides an easy drag and drop interface to visually control the grouping.

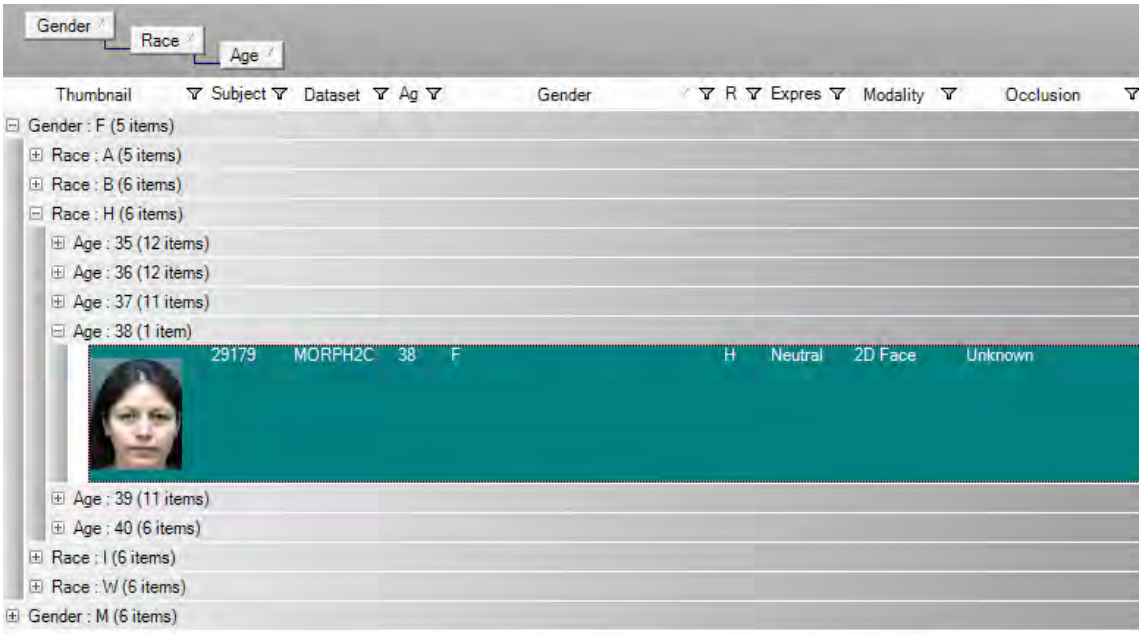

Figure 4.23: MIDO Hierarchical Group by Column

While MIDO does not actually store the source images in the database, it does store a thumbnail of each image. To reduce import time and to minimize bandwidth in network environments, thumbnails are only generated and stored the first time the thumbnail is requested. To facilitate this, a thumbnail manager class checks to see that a thumbnail doesn't already exist in the database. A thumbnail is a 64x64 pixel version of the source image and is indexed and compressed meaning it is generally less than 1kb of storage. Source images on the other hand can be several orders of magnitude larger. If an thumbnail is not found in the database, the source image is asynchronously retrieved, resized, and submitted back to the database. As the images are processed into thumbnails, they appear in the client. On slower networks, this may be a noticeable delay as the thumbnails are generated.

4.1.3 C++ Library

A C++ Library was developed the allows a 3rd party application to query the database and return the image metadata and the associated image as an OpenCV Mat. It also contains a C++ class version of the MIDO SQL Database. In this way, interacting with the database in C++ is intuitive for developers.

In the following example, Listing 1, MIDO is queried to find all males under the age of 21. It then loops through the results and displays the image using the dataset name as the title of the window.

```
#include <mido.hpp>

using namespace std;
using namespace cv;
using namespace I3S::Acquisition;

MIDO mido("localhost\\instance",
          "MIDO",
          "mido",
          "mido",
          path("N:\\images"));
mido.Open();
mido.Select("Gender = 'M' AND Age < 21");
cout << mido.GetCount() << " records found." << endl;
while (mido.HasNext())
{
    Mat img = mido.GetNext();
    auto metaData = mido.GetImageRecord();
    imshow(metaData.dataset->Name, img);
    waitKey(0);
}
mido.Close();
```

Listing 1: MIDO C++ Library Example

The constructor consists of 5 required parameters.

- Server and instance name
- Database name
- Username

- Password
- Base Path to images

Once an instance of the MIDO class has been created, you must open the connection with the *Open()* method. Next, using the *Select()* method, provide a list of criteria to limit the search. The string passed into the *Select()* method is the *WHERE* clause of an SQL statement, without the *WHERE* keyword. This string may be customized with any legal SQL query and is simply appended to a predefined *SELECT* statement. Calling *Select()* is a synchronous blocking call that will send the request to the server and download the records including all associated metadata. For obvious memory and speed reasons, the images themselves are not downloaded until *GetNext()* is called. To retrieve the path to the current image, use *GetImagePath()*. Using *GetImageRecord()*, a copy of the metadata for the current image may be retrieved. This is a custom class that mirrors the MIDO database structure.

4.1.4 Picture Library

The .NET platform provides an image class that is very useful for reading, viewing, and writing images but manipulating images on the pixel level is incredibly slow. The built in *SetPixel()* method of the *Bitmap* class in .NET performs three expensive functions. First, it copies the pixels from managed memory into a temporary, mutable, storage location. It then changes the requested pixel value. Finally, it copies the entire image with the single changed pixel back to the managed memory location. To do this across more than a few pixels is impractical, much less the several million pixels that are common from today's cameras.

To allow for fast image manipulation in the MIDO client and plugins, the internal image format is a custom *Picture* class. This is the fundamental component of the plugin architecture allowing a plugin to manipulate the image on a per-pixel level for any sort of algorithm the plugin needs to implement: object detection/recognition/classification, automated annotations, image processing. The *Picture* class solves the speed dilemma by

maintaining a mutable version of the raw image bytes and updating the managed version only when explicitly told to do so in code. The full details of the *Picture* class are beyond the scope of this document. The complete manual may be found online[6] as part of the Image Processing Sandbox documentation.

The *Picture* class batches writes between *Suspend()* and *Resume()*. When an image is suspended, changes to the image are performed in high speed, unmanaged memory. Upon resuming, those changes are written back into managed memory and can be used as a native *Image* object. Listings [2, 3] illustrate how to change one pixel to red, using row-column indexing, for C# and Managed C++ respectively.

```
void OneRedPixel()
{
    Picture pic = new Picture(@"C:\\MyImage.png");
    pic.Suspend();
    pic[0, 0] = Colors.Red;
    pic.Resume();
    // Use updated image here ...
}
```

Listing 2: Example using the *Picture* class in C#

```
void OneRedPixel()
{
    Picture^ pic = gcnew Picture("C:\\MyImage.png");
    pic->Suspend();
    pic[0, 0] = Colors::Red;
    pic->Resume();
    // Use updated image here ...
}
```

Listing 3: Example using the *Picture* class in managed C++

[6]http://www.benbarbour.com/ImageProcessingSandbox

Documentation for the Picture class is generated dynamically using Sandcastle Help File Builder (SHFB) which reads XML documentation code that is generated by Visual Studio. XML documentation is written directly inside the source code as a comment and with the compiler command, */doc*, an XML file is created that can be used to generate documentation in many different formats.

```
1  /// <summary>
2  /// Saves the image in the default format using the FileName.
3  /// </summary>
4  /// <returns>True if successfully saved</returns>
5  public bool Save()
6  {
7      if (FileName != "")
8          return Save(FileName);
9      return false;
10 }
```

Listing 4: Example XML documentation

In the example in Listing 4, the *Save()* function has a summary and a return value. This is converted into an MS Help document as well as a fully functional website that mimics the Microsoft Developer Network style.

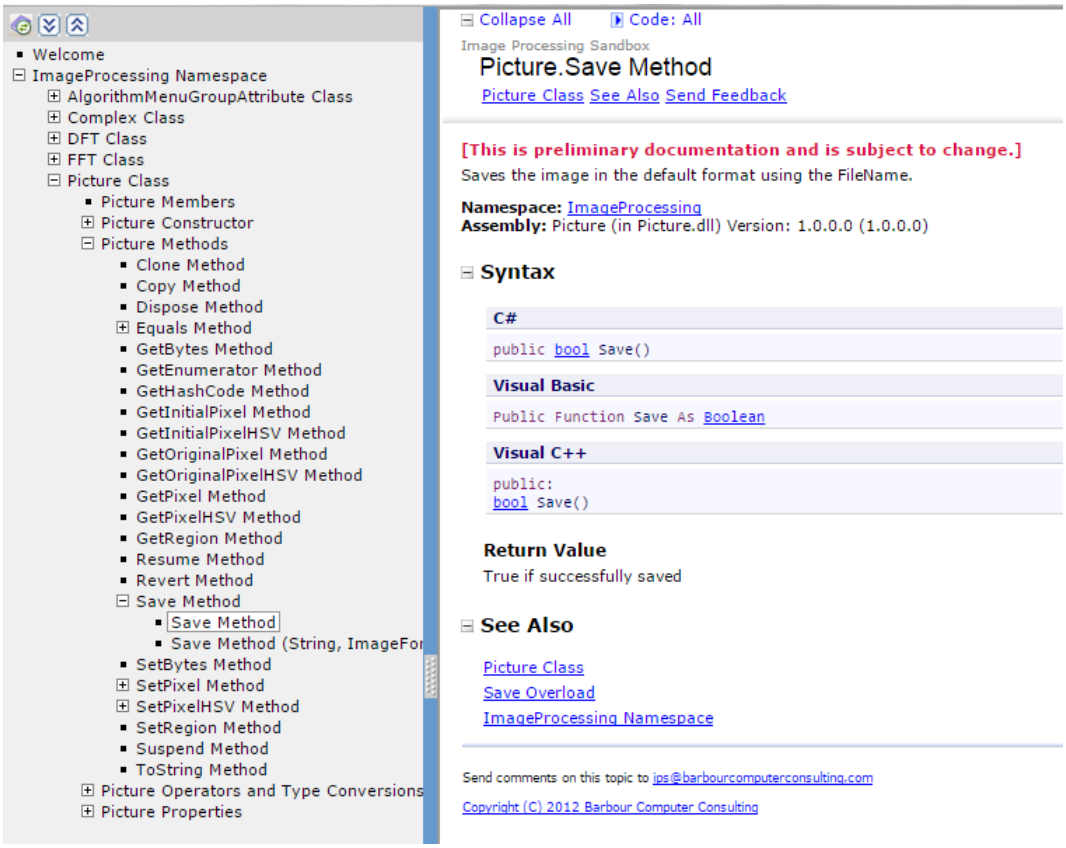

Figure 4.24: Website version of XML documentation shown in Listing 4

More details on the Picture class including class diagrams, examples, and documentation, may be found in Appendix 2.

4.1.5 Automated Installer

The automated installer is based on Nullsoft Scripted Installer System (NSIS). It can automatically install all necessary prerequisites for Microsoft SQL Server and MIDO as well as fully automating the Microsoft SQL Server installation process, create and import SQL data, copy public datasets, and of course install MIDO. The machines that will install the suite at the FBI are not connected to the Internet and therefore the installation system must handle the local installation of all prerequisites across any version of Windows.

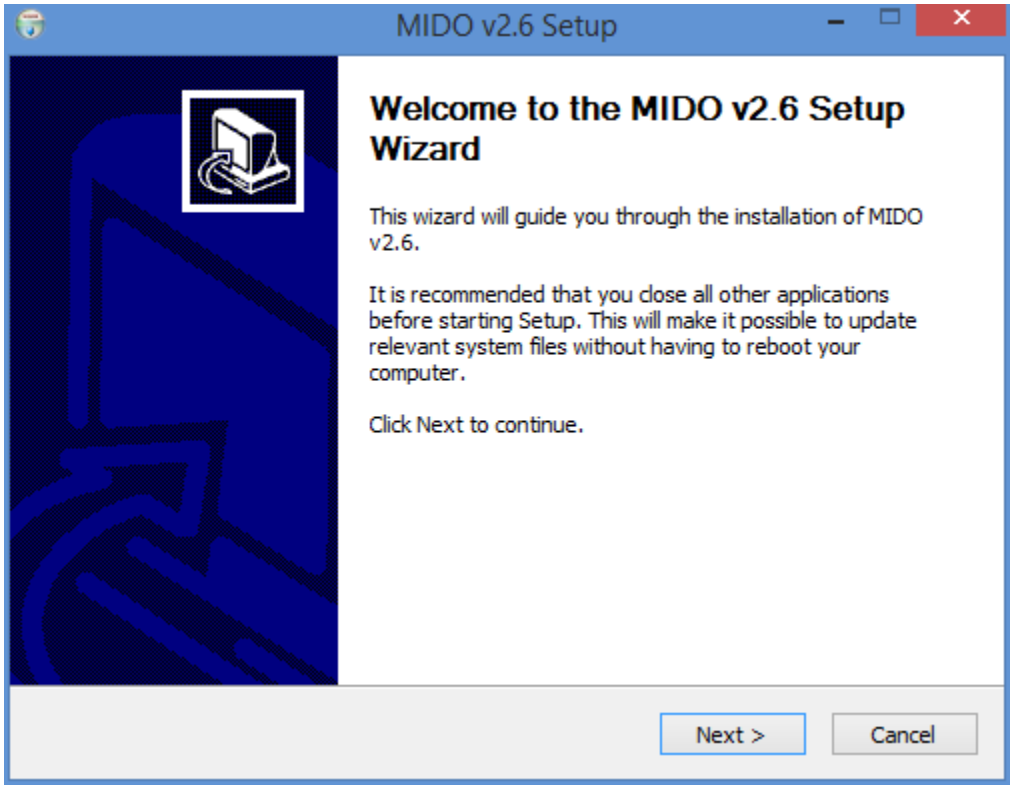

Figure 4.25: MIDO Installer

The MIDO installer is in two stages. The first is a prerequisites installer that checks the version of the Windows Installer framework and installs an updated version if necessary. It then checks for the presence of the Microsoft .NET framework and installs versions 3.5 and 4.0 if needed. Finally, it attempts to install the Microsoft PowerShell which is a prerequisite for Microsoft SQL Server. After a reboot, the second stage begins the actual MIDO installation process.

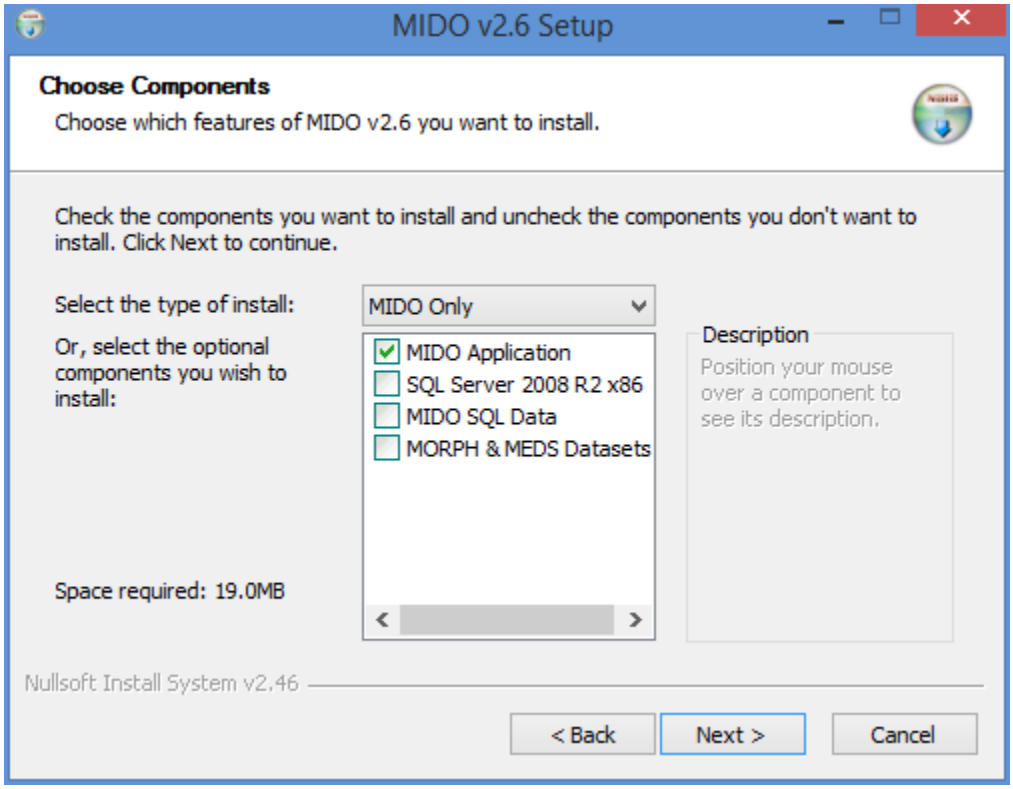

Figure 4.26: MIDO Installer - Component Selection

MIDO may be installed in two types of environments. A standalone installation where the local computer runs both the database server and the MIDO client and a networked installation where the SQL server may be hosted remotely while MIDO is installed locally. In the case of the former, the installation and configuration process is completely automated. Figure 4.26 shows the component selection options where installation may be customized for the user's environment. The latter requires an existing remote SQL server with the MIDO database already installed and appropriate access rights configured. Installation of the database itself is also automated, the user simply needs to connect to a database using credentials with *CREATE DATABASE* rights. The simplicity of this portion of the installer is showin in Figure 4.27.

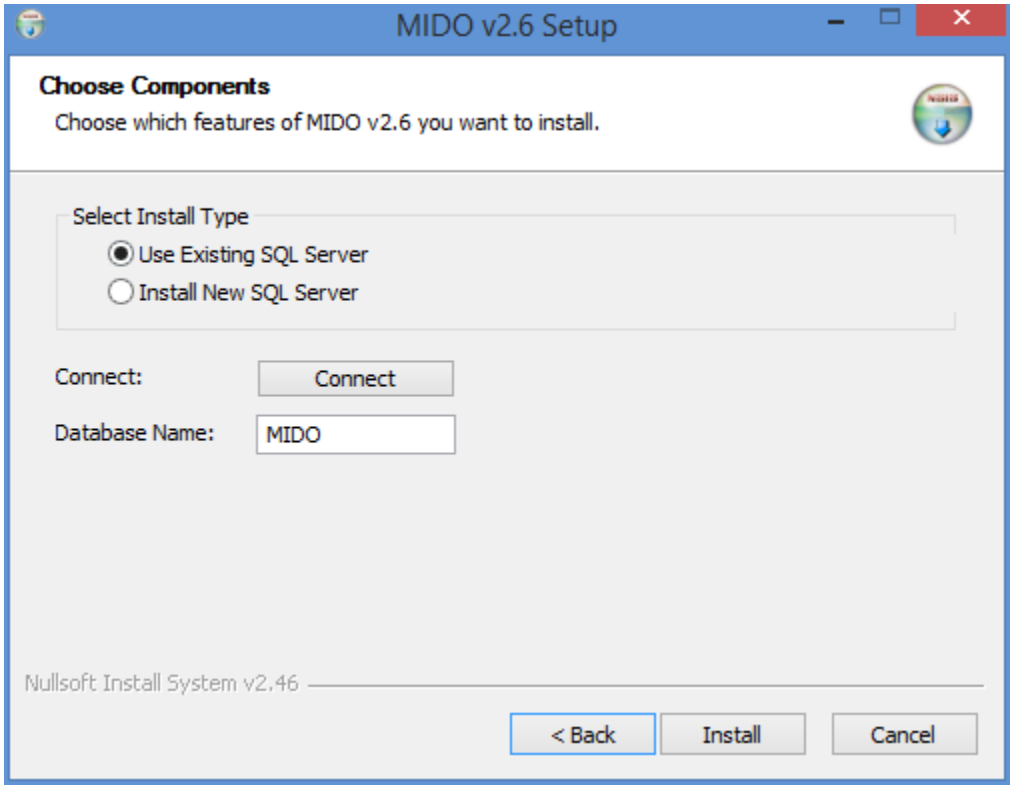

Figure 4.27: MIDO Installer - SQL Server Connection

4.1.6 Dataset Import Utility

The dataset import utility is project that converts an existing dataset in its native form into a CSV file useable by MIDO's import utility. Since private datasets, including their image metadata, cannot be distributed with MIDO, a conversion utility enables easy importing of known datasets for end users.

This utility is a command line tool, shown in Figure 4.28 that accepts a path to a dataset and produces a flat CSV file suitable to import into MIDO or consume in any other application such as a spreadsheet or data analysis tool such as Matlab or R.

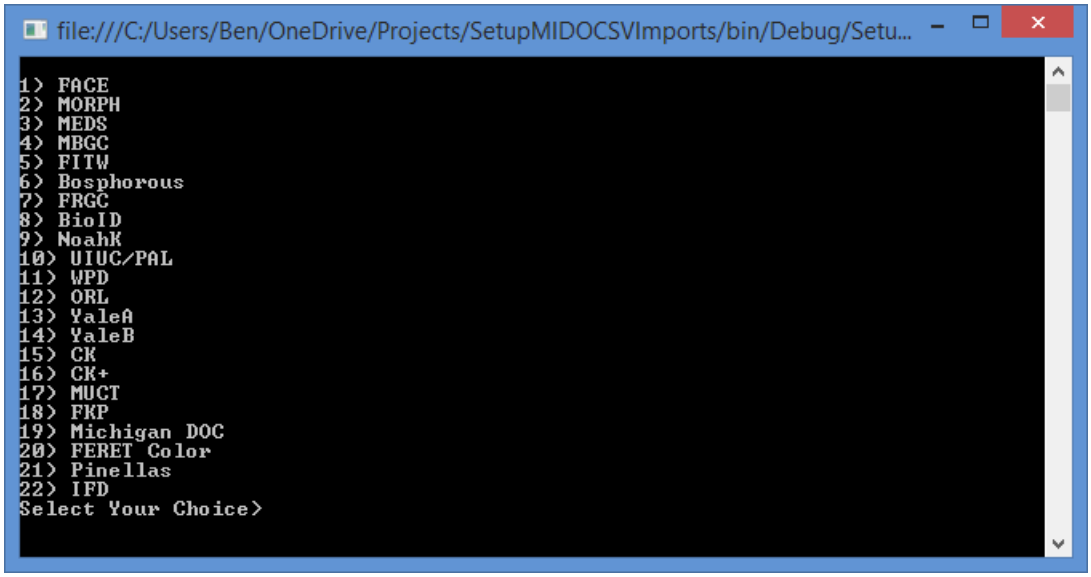

Figure 4.28: MIDO Dataset Import Utility

4.1.7 Plugin Architecture

The MIDO plugin architecture scans for a directory named ”Plugins” underneath the current working directory. If it finds a managed .NET assembly it will load that assembly and process any plugins found. A plugin is implemented as a static method inside an assembly. The assembly should wrap the method inside a public class named *UserPlugin* inside a namespace named *MIDO*. The method should have one of three parameter lists: none, Picture, or MIDO.Data.Image[]. If no parameters are specified, the plugin cannot interact directly with the MIDO client but it can still provide any sort of independent functionality. If the plugin’s method accepts a *Picture* object, it will be provided access to the currently displayed image in the client. Changes to this object will be reflected in the client. In the final constructor, an array of Image[7] objects represent the current recordset in the client.

When a plugin method is found, MIDO looks for an annotation called *PluginMenuGroup* which provides the group and title for the button to be placed in the plugins submenu of the client. In Listing 5, a very minimalist plugin is shown that creates a group

[7]The ambiguity of the name ”Image” will be resolved in a later version. In this case *Image* represents the corresponding database table name, not the *System.Drawing.Image* class.

called ”Test”, a plugin menu button called ”Hello World”, and displays ”Hello World” in a message box.

```
using System;
using MIDO.Plugin;

namespace MIDO
{
    public class UserPlugin
    {
        [PluginMenuGroup("Test", "Hello World")]
        public static void Test()
        {
            MessageBox.Show("Hello World");
        }
    }
}
```

Listing 5: Example MIDO plugin

Listing 6, takes the records from a search and generates a text file listing the absolute path for each image. If no search has been executed, *images* will be null.

```
[PluginMenuGroup("Data Processing", "Log File Paths")]
public static void LogFilePaths(MIDO.Data.Image[] images)
{
    StringBuilder txt = new StringBuilder();
    foreach (var img in images)
        txt.AppendLine(img.ImagePath);
    using (TextWriter tw = new StreamWriter("images.txt"))
        tw.Write(txt.ToString());
}
```

Listing 6: Example MIDO plugin using database

4.2 FaceMark

FaceMark, by comparison to MIDO, is significantly less complex. Its architecture is comprised of three key components, the client, the Picture library detailed in Chapter 4.1.4, and the core annotation logic which is contained in it's own assembly called *Landmark*. Figure 4.29 illustrates how these three components interact.

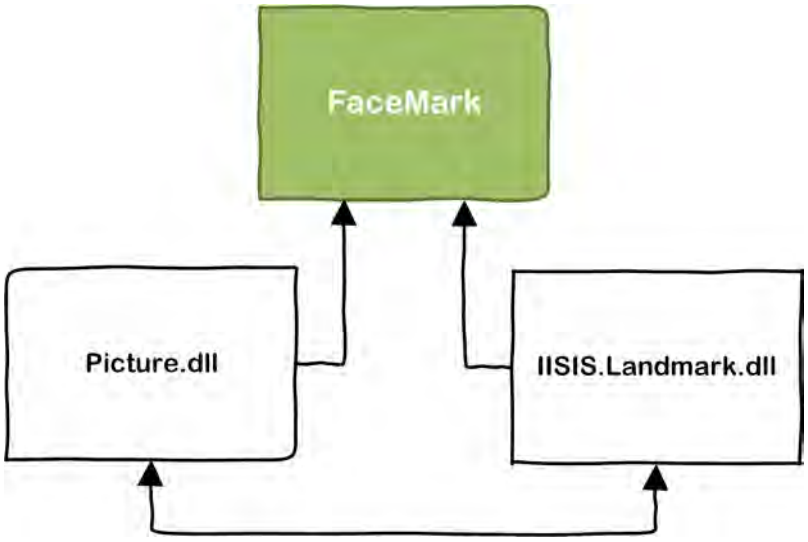

Figure 4.29: FaceMark Architecture

4.2.1 Landmark Control Library

The *Landmark* library is a .NET Windows Forms user control. A user control inherits from the *UserControl* base class which implements the basic functionality for a graphical element that can be used on a Windows Form [38]. In this way, the *Landmark* assembly may be used to provide annotation functionality to any .NET application. Figure 4.30 shows the graphical component of the library.

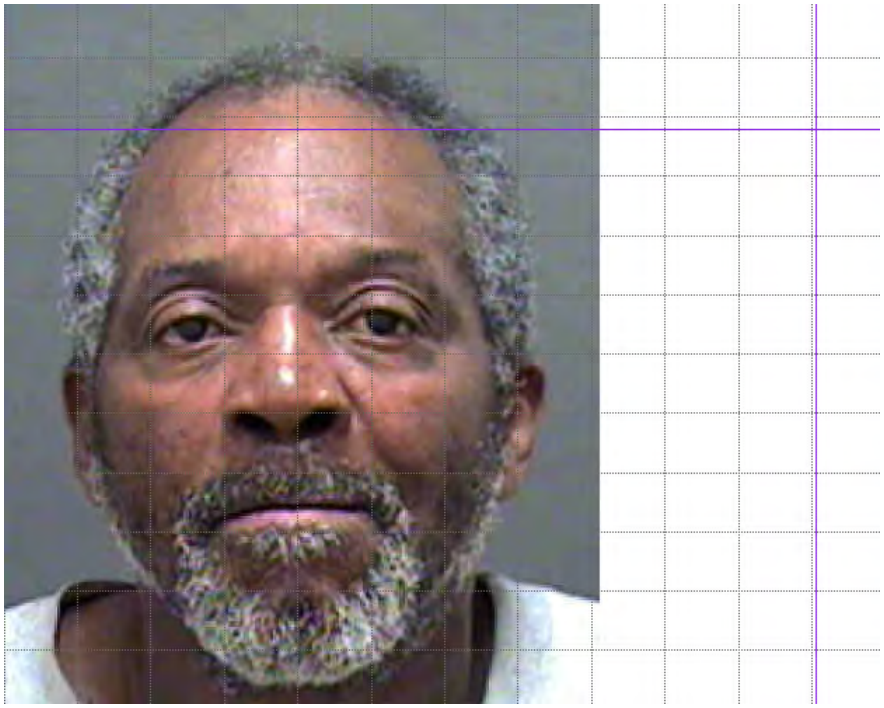

Figure 4.30: IISIS.Landmark Control

The control provides the following functionality:

- Load, display, and save images
- Load, display, and save annotation points
- Mouse interaction
- Keyboard interaction
- Visualization elements: grid, crosshairs
- Annotation manipulation: select, drag, copy/paste, rotate, scale

4.2.2 Client

The FaceMark client, Figure 4.31, implements the *Landmark* control library and provides an interface to manage the annotation process. It exposes an extensive array of customization choices, Figure 4.32, as well as guided annotation visualization.

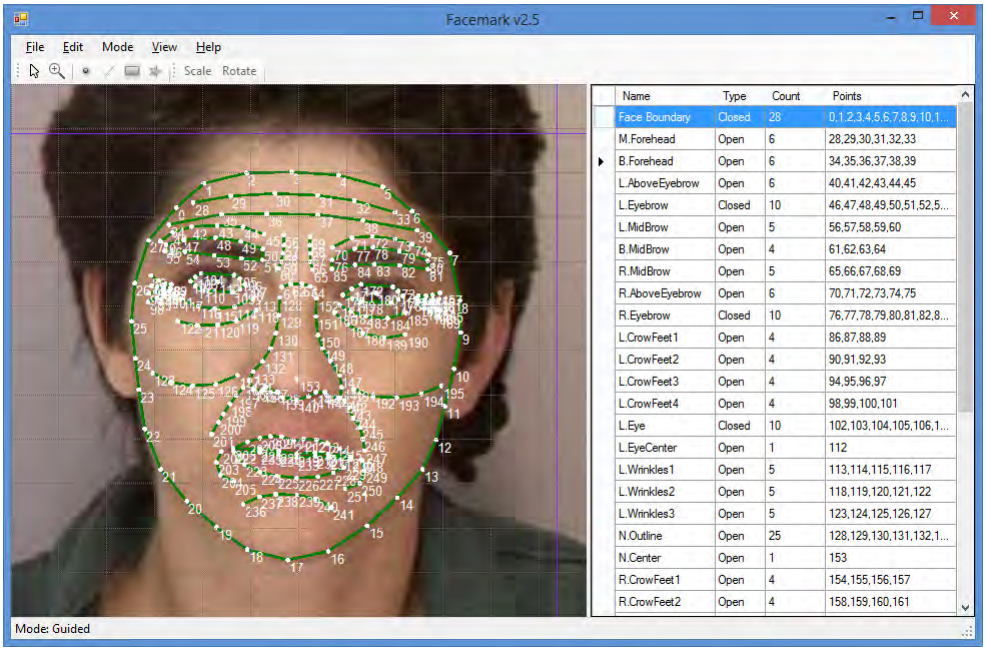

Figure 4.31: FaceMark

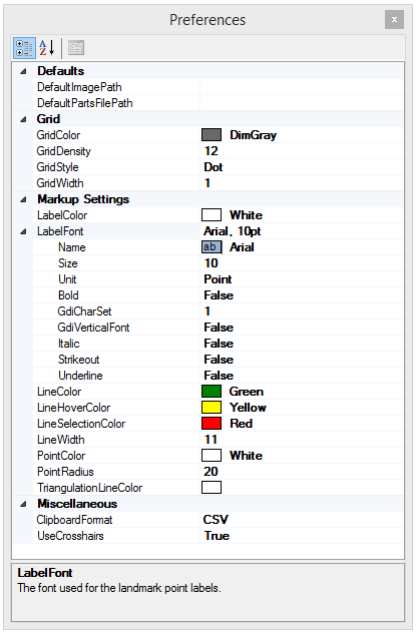

Figure 4.32: FaceMark Options Dialog

4.2.3 *Installer*

Figure 4.33: FaceMark Installer

The installation process for FaceMark is relatively simple. An option is presented to install sample images and points files in addition to the executable. If the .NET Framework 4.0 is not present on the system, it will be installed.

4.3 AgeMe

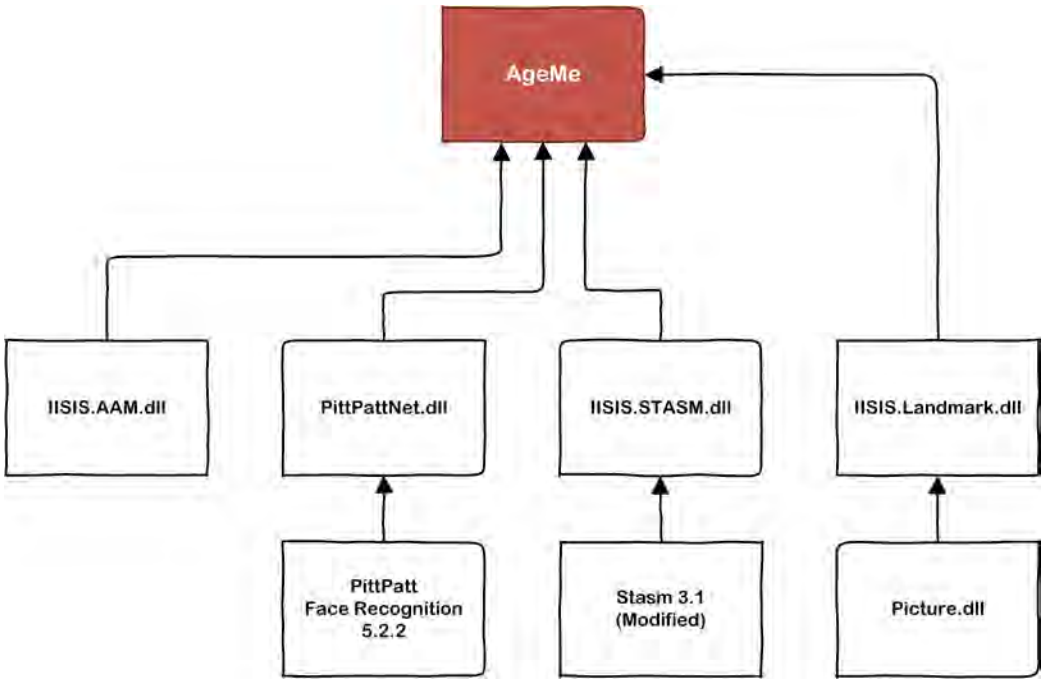

Figure 4.34: AgeMe Architecture

4.3.1 PittPatt

Pittsburg Pattern Recognition (PittPatt) was launched in 2004 as a Carnegie Mellon University spin-off founded by Dr. Henry Schneiderman, Dr. Michael Nechyba, and Dr. Michael Sipe. PittPatt developed a high quality face detection, face recognition, automatic landmark detection, and video tracking SDK that caught the attention of Google in 2012. Google purchased PittPatt [39, 40] for use in their Android "Face Unlock" as well as other areas in the Google catalog such as Picasa, Google+, and YouTube. Unfortunately the acquisition by Google closed the doors on PittPatt and the SDK is no longer available to the public.

The final version released to the public is version 5.2.2. This version is used in AgeMe as the face recognition and initial landmark detection for eye and nose coordinates. Since PittPatt is closed source and written in pure C and compiled using the MSVC 2008 compiler, a .NET wrapper was written in managed C++ to simplify its use in modern graphical applications. The wrapper handles the in-memory conversion from a *Sys-*

tem.Drawing.Image object to a *pittpatt_image* format. A single call to *GetLargestFace()* will return the bounds, eye coordinates, nose coordinate, yaw and roll of the largest face, and a confidence measure describing the quality of the results as per an internal PittPatt metric.

```
1  using IISIS.FaceDetector;
2
3  void PittPattExample()
4  {
5      PittPatt pp = new PittPatt();
6      Image img = Image.FromFile("face.png");
7      Face face = pp.GetLargestFace(img);
8      double confidence = pp.GetConfidence();
9      double yaw = pp.GetYaw();
10     double roll = pp.GetRoll();
11     Console.WriteLine("Face found at {0} with size {1}",
12                       face.FaceBounds.Location,
13                       face.FaceBounds.Size);
14     Console.WriteLine("Eyes found at {0} and {1}",
15                       face.LeftEye,
16                       face.RightEye);
17     Console.WriteLine("Nose found at {0}",
18                       face.Nose);
19 }
```

Listing 7: Example of using the PittPatt wrapper

4.3.2 Stasm

In his 2007 master's thesis [41], Steven Milborrow and his advisor Dr. Fred Nicolls at the University of Cape Town, South Africa, developed a method of automatic face annotation building on the work of Tim Cootes' Active Shape Models [9, 11]. The method proved to be a very robust technique for landmark localization [42, 43]. Stasm is currently in its fourth revision [44] at the time of this writing however, AgeMe uses a modified ver-

sion of the 3.1 [45] release.

Stasm was modified to address several problems. The first was to update old libraries, remove unnecessary libraries, and to convert the application to 64-bit allowing access to significantly more memory and thus the ability to build larger models with higher point density. The second major change was to correct its sensitivity to initial positioning. The bounding box returned by face detectors is not predictable and its accuracy varies greatly with pose and face shape. In addition, a model built with a certain face detector is required to use the exact same detector with the exact same configuration to produce usable results. The most common face detector, and the one coupled to Stasm is Viola Jones which returns a square encompassing the lower lip to the eyebrows. This placement causes poor results when using extremely dense annotation schemes. However, given the eyes and nose positions, the initial placement can be warped to a much more accurate initial position yielding significantly better results. Thirdly, the loading of configuration files was decoupled from the search process to enable thread-safe searching. Finally, the search call was changed to accept an image already in memory rather than require it to load an image from disk.

Once coordinates for the eyes and nose have been obtained using PittPatt, Stasm will automatically attempt to find 249 more points of interest on the face. These include locating a tight boundary around the face, outlining the mouth, nose, eyes, eyebrows, and locating the cheekbone, forehead, nasolabial folds, and many other features. The result is a detailed analysis of the shape of the given face as shown in Figure 4.35.

Figure 4.35: Left to right: original face, landmarks aligned to face, visualizing landmarks returned from stasm

```
1  using IISIS;
2  using IISIS.FaceDetector;
3
4  void StasmExample()
5  {
6      Image img = Image.FromFile("face.png");
7
8      PittPatt pp = new PittPatt();
9      Face face = pp.GetLargestFace(img);
10
11     STASM stasm = new STASM();
12     stasm.LoadModels("stasm.conf0", "stasm.conf1");
13     PointF[] landmarks = stasm.GetLandmarks(img, face);
14 }
```

Listing 8: Example of using the Stasm library

After instantiating the *STASM* object, line 11 loads the model files. Stasm can operate in traditional ASM mode or in stacked ASM mode. For the former, the second parameter may be an empty string.

Line 12 creates an array of *PointF* objects that represent the landmarks.

4.3.3 Active Appearance Model Library

The AAM library is a mixed mode assembly with a managed C++ interface and unmanaged C/C++ core. Unlike the Stasm and PittPatt assemblies which required .NET wrappers around 3rd party code to interact with the client, the AAM library was written from the ground up using modern portable standards. While it is called the AAM library, it incorporates both Appearance Model functions as well as implementation of age estimation, gender and race classification, and age progression.

The majority of the AAM library is written in native C++ using OpenCV 2.4.8 [36], Boost 1.55 [37], and libsvm 3.18 [46]. OpenCV is an industry standard computer vision and machine learning API used in a wide variety of applications from hobbyist to professional systems. Boost, likewise, provides a wide array of portable helper functions that eliminate the need to write custom code for distinct platforms. In particular, file system access and threading each have unique implementations between the Windows and *nix-based platforms. Libsvm is a peer reviewed implementation of support vector machines and is used by the AAM library for classification and regression.

To use the library in an application, you must first load pre-built models for each function to be used. There is a single *LoadModel()* call which accepts a list of file paths for the various models that can be loaded. If a particular function is not needed, its model file path can be set to an empty string in order to bypass that functionality.

```
1  using IISIS;
2
3  void AAMExample()
4  {
5      FacialAnalytics aam = new FacialAnalytics();
6      aam.LoadModel("aam.model",
7                    "age.model",
8                    "gender.model",
9                    "race.model",
10                   "age.table");
11     Image img = Image.FromFile("face.png");
12     PointF[] landmarks = aam.LoadPoints("face.pts");
13     double age = aam.PredictAge(img, landmarks);
14     int race = aam.PredictRace(img, landmarks);
15     int gender = aam.PredictRace(img, landmarks);
16
17     Image ageProgressed = aam.AgeProgress(img,
18                                           landmarks,
19                                           age,
20                                           age + 10.0);
21 }
```

Listing 9: Example of using the AAM library

Line 4 shows the *LoadModel()* function. The first parameter is an active appearance model file built with *Build()*. The next three parameters are support vector machine (SVM) models built using the C++ version of libsvm. The final parameter is a proprietary file used for age progression built using internal I3S tools.

Line 10 loads annotation points in the Cootes ”pts” format. In this case, these are usually generated by Stasm, DASM, or by using FaceMark.

Lines 11-13 perform the classifications for age, gender, and race. Age will return a double precision value between 18 and 80 years while the race and gender return an

integer that represents a specific classification. For race, a 0 represents Caucasian and a 1 represents African-descent. For gender, a 0 represents male and a 1 represents female. The advantage to using an integer return value is that the classifiers can be easily extended to support other classifications without breaking existing code.

Line 15 demonstrates producing a new image of the given face, aged by 10 years. The third parameter is the start age for age progression. In most cases this should be the predicted age. The last parameter is the target age, in this case computed by adding 10 years to the predicted age.

4.3.4 Client

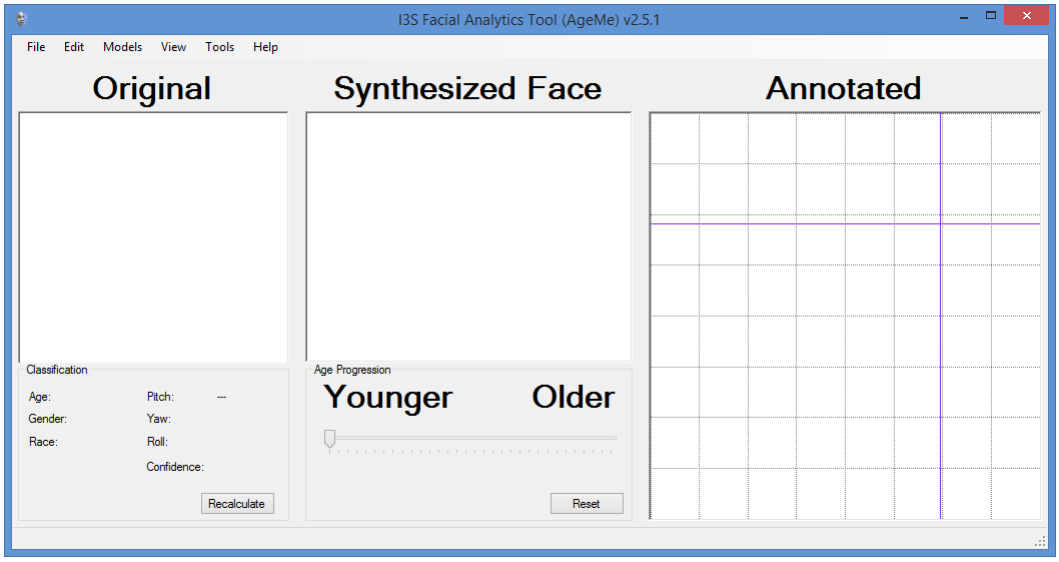

Figure 4.36: AgeMe User Interface

The AgeMe client provides an all-in-one application to evaluate the age, gender, race, and age progressions of an individual. By using the *File* menu and selecting *Load Image*, the user will be presented with three images. On the left, the original image is visible for reference. The middle image is, at first, an image that represents how the system sees the face. On the right is the *Landmark* control that can be used to adjust the annotations for the face. As soon as an image is loaded, it is evaluated by the face detector, followed by the automatic annotator, and finally it is run through the classifiers and the results are shown to the user. If the automatic annotations are not accurate, the user may adjust the

landmarks and re-run the classifiers using the hand tuned annotations. Figure 4.37 shows the application with a sample image loaded and processed.

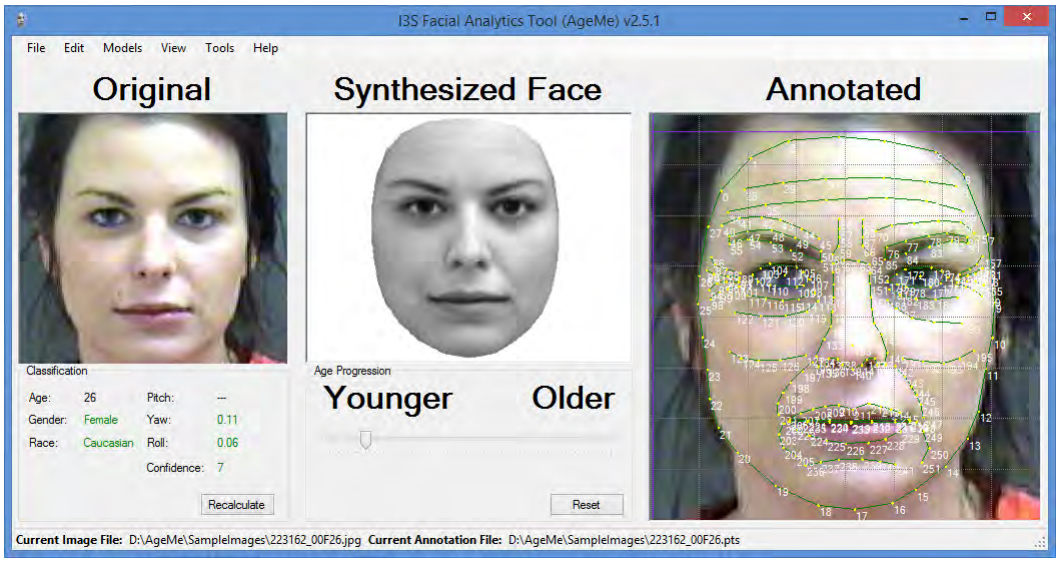

Figure 4.37: AgeMe GUI with image loaded and processed

Underneath the image on the left are the results of face detection and classification. The colors of the numbers represent the confidence in those classifications. For the gender and race classification results, green indicates that the algorithm predicted with high confidence while yellow and red indicate poor and unreliable confidence respectively. In the case of the face detector's yaw, roll, and confidence measures, a yellow or red indicates that the face may not be in a suitable position for quality classification and the results might not be meaningful[8]. Figure 4.38 demonstrates the effect of pose on the quality of results. With a yaw rotation of that extreme, the confidence in the gender detection was quite poor. The current approach to age estimation does not generate a confidence measure so its color does not change however, its result likewise cannot be trusted due to the poor pose.

[8]There is also a label for pitch in the GUI however, the algorithm for determining the pitch of a face was never completed prior to Google's acquisition of PittPatt and therefore the label always reads "–".

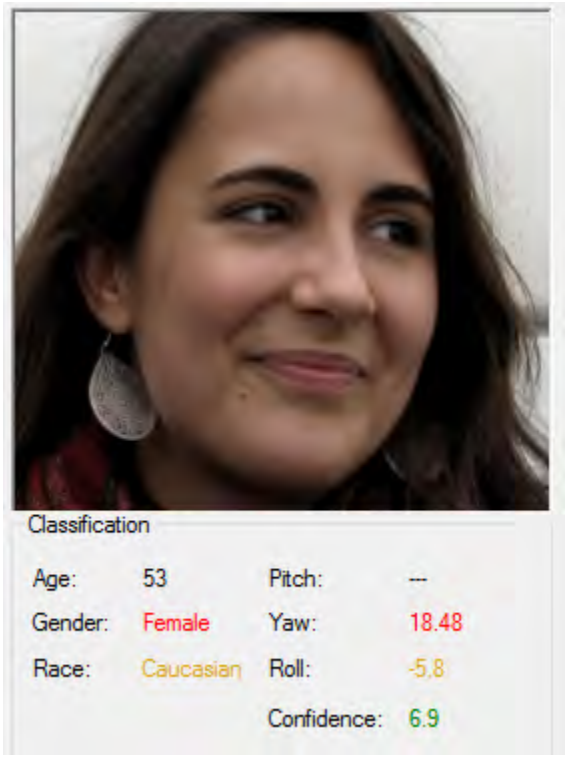

Figure 4.38: AgeMe results affected by poor pose angle

In the middle section, a slider may be scaled from left to right which updates the face in the center with an age progressed (or regressed) version. By pressing ”Reset”, the face will revert to the closest representation at the current estimated age of the source image. Figure 4.39 shows the effect of adjusting the slider while Figure 4.40 illustrates the aging process from source, reconstruction, and age progression. The second and third images were produced using the ”Save Transformed” option from the file menu which allows users to save either the composite reconstruction which overlays the synthetic face on the original image or just the synthetic face.

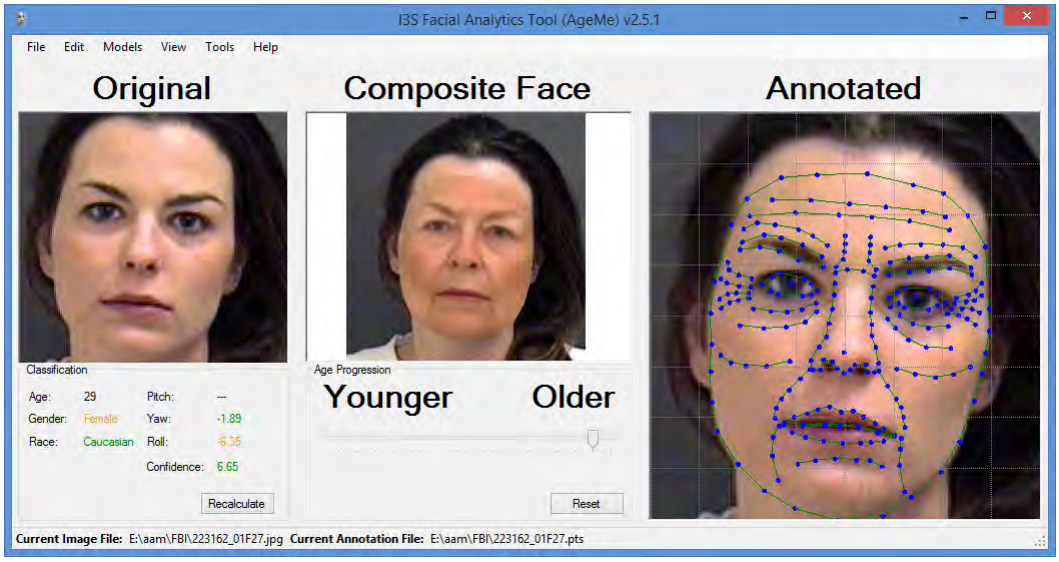

Figure 4.39: Age progression from around 26 years to approximately 70 years

Figure 4.40: Detail of age progression results. Left to right: original face, synthesized reconstruction, aged approximately 40 years

The age progression results are dependent upon the quality of the data used to create the model. That is to say, a model built to represent a specific ethnogender group is more likely to produce results faithful to that group than one built on a diverse set of individuals. Since visual fidelity is a primary concern, several models have been provided in the AgeMe application: Generalized Grayscale, Caucasian Female, Caucasian Male, African-descent Female, African-descent Male. The generalized grayscale model incorporates thousands of faces across genders and races and produces results in grayscale while the ethnogender

models are restricted to their respective classes. Figure 4.41 shows the effect of different models across the same face.

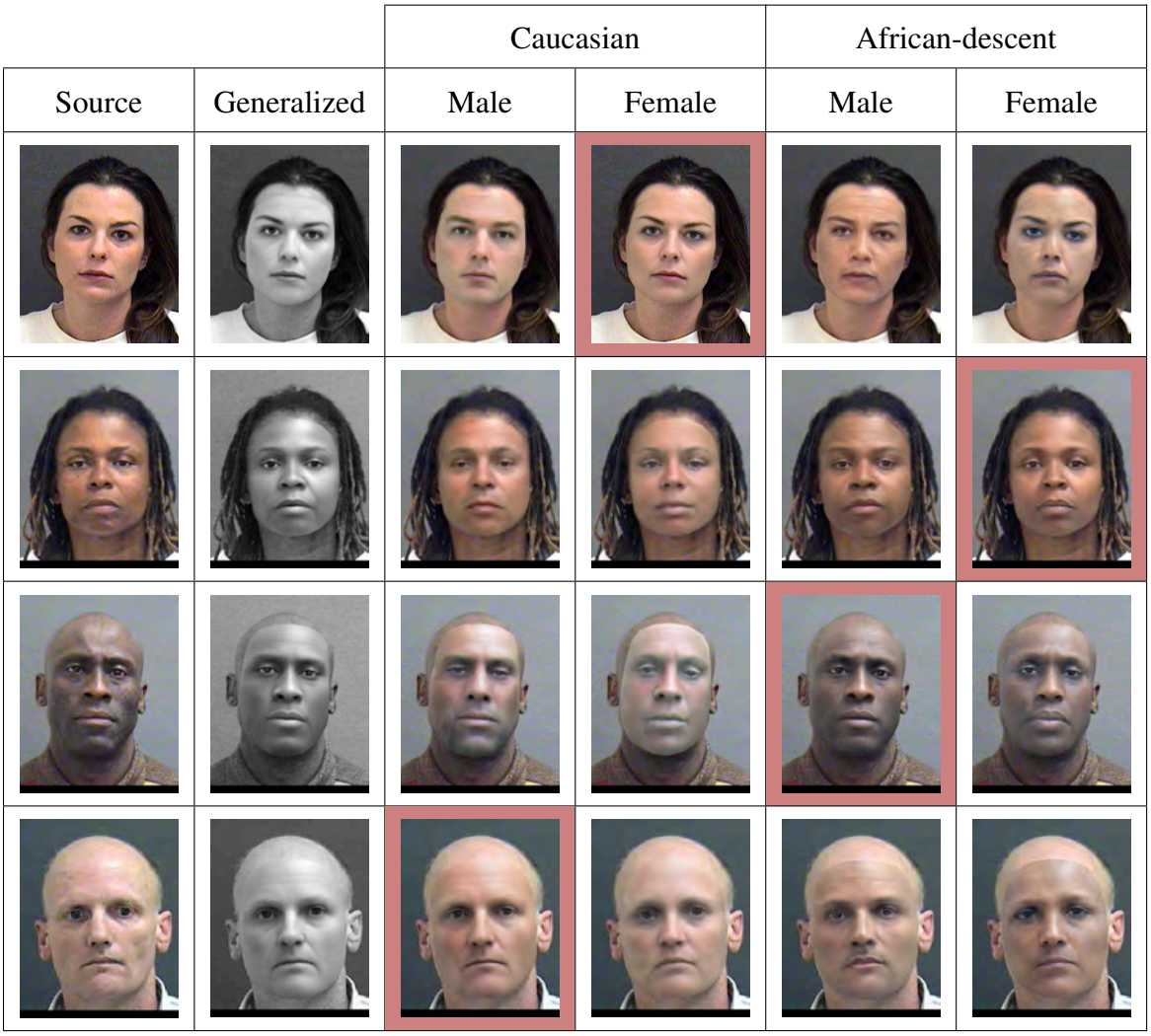

Figure 4.41: Effects of various age models on face reconstruction. Outlined in red are the corresponding race/gender reconstructions for the source.

The AgeMe tool's annotation and analysis process is great for producing quality hand tuned results on a single image a time but not practical for large scale analysis. To process an entire directory at once, there is an option in the "Tools" menu called "Process Directory" which accepts the path to a directory and the name of a CSV file to generate. The result is a CSV file that contains the fields listed in Table 4.1.

FileName	string
Age	double
Gender	string
Race	string
Yaw	double
Roll	double
Confidence	double

Table 4.1: AgeMe batch processing CSV file format

4.3.5 Installer

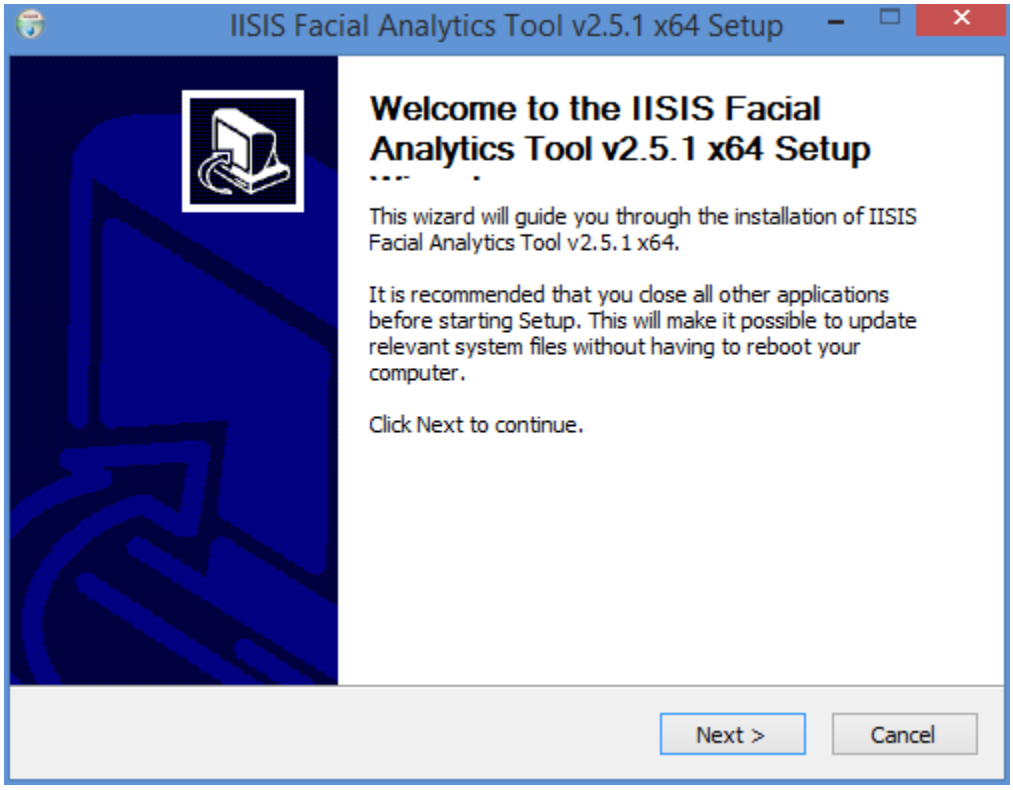

Figure 4.42: AgeMe Installer

The AgeMe installer, Figure 4.42 takes care of installing the prerequisites: MSVC 2013 x64 Redistributable and Microsoft .NET 4.0/4.5 Redistributables. In addition to installing the AgeMe application, it also configures the user interface with default values. The configuration for installing x64 bit application is slightly different than the default x86 architecture. In a 64-bit version of Windows, x86 applications are installed into their own ”Program Files” folder called ”Program Files (x86)” whereas x64 applications are

installed in the standard ”Program Files” folder. This means that the default ”Program Files” folder corresponds to the default system architecture. The NSIS variable *$PROGRAMFILES* however, contains ”Program Files” on an x86 machine and ”Program Files (x86)” on a x64 machine. An addon was written for NSIS to make up for this issue called ”x64.nsh” which creates 64-bit versions where problems exist. This makes correcting the error as simple as using *$PROGRAMFILES64* instead.

Chapter 5: Conclusion

In this paper I have covered the design, development, and implementation of three applications created for the FBI to address the need to manage large numbers of facial images and perform automated analysis on those images. These tools transformed research performed at UNCW into professional quality applications that could be managed with minimal training and experience. Final versions were shipped on October 24th, 2014.

Figure 5.43: Final deliverable DVD label

This technology, though developed specifically for facial processing, can easily be extended into a wide range of other domains. In the healthcare industry, MIDO could be used to catalog imagery related to ailments and conditions while an automated analysis tool could classify new images, rank severity, or predict outcomes. Ferry et al. [47] use craniofacial characteristics, analyzed using AAMs and classified using SVMs to cluster faces into specific phenotypes in order to diagnose known genetic disorders.

An age progression app could provide a glimpse into the future of a child. Due to the nature of human aging, most parents will never get to see their children reach an old age. This is never more true than when a parent loses a child early in life. Many apps of this nature exist for a variety of platforms however none of them work on children. Rather

than produce a reasonable representation of a face at a given age, they place wrinkles on the face to make you appear old. This is insufficient for anything more than a fun joke.

Facial analysis, including texture and shape representations, also have implications in healthcare. Expression analysis for pain measurement [48] has been used for many years for determining the severity of pain. Typically patients are asked to rate their pain on an arbitrary scale of 1 to 10. Unfortunately, ranking pain is extremely difficult to do and thus it is difficult to use a patient's response to determine the correct dose of medication to combat the pain effectively. It often becomes an iterative process of adjusting the levels to find the appropriate amount. In addition, this approach is rendered useless when the patient is unable to communicate due to a physical impediment or language barrier. Loresh et al's face scale could be automated, using similar classification and analysis approaches as in AgeMe, to monitor patients by video and alert healthcare professionals when pain levels have exceeded a threshold.

The approach used to render an aged face can also be used to render other representations of a face. For example, the algorithm could be modified to predict the effects of smoking, drinking, sun exposure, and drugs over time and used to educate people about the harmful effects of exposure to these types of things. Effects of medically necessary or elective plastic surgery could be modeled and demonstrated to the recipient prior to undergoing surgery.

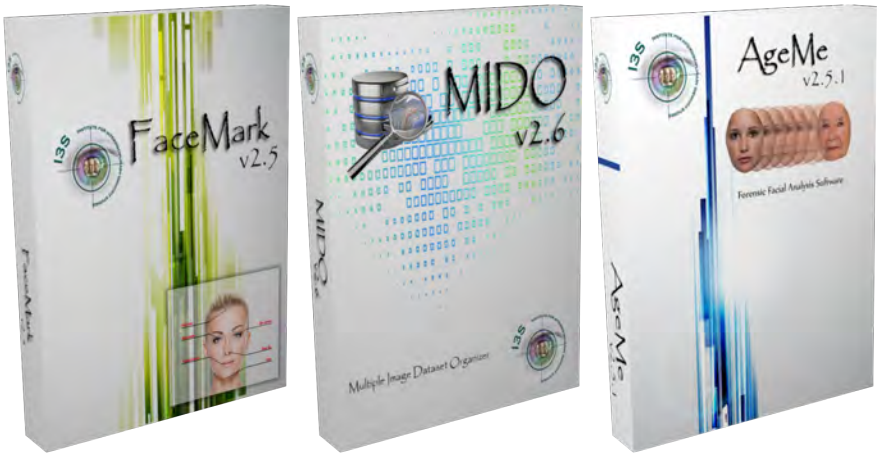

Figure 5.44: Final deliverable box art

As demonstrated, these tools open the door to a wide variety of uses. They establish a platform on which many real world problems may be examined and addressed. By producing user-friendly, professional quality software that applies scientifically sound algorithms this work has moved the bar for what can be produced from a research environment.

Acknowledgments

I would like to acknowledge the following people for their contributions to this work, quality assurance and beta testing, suggestions and critiques, and most importantly supporting me over the past several years.

Megan Barbour

Dr. Karl Ricanek Jr.

Dr. Devon Simmonds

Dr. Elizabeth Baker

Jason Vandeventer

Amrutha Sethuram

Brandon & Erin Hilton

Jeff Raynor

Dave Macurak

Michael Sodomsky

Harry Atwal

References

[1] Wikipedia, "2011 england riots." http://en.wikipedia.org/wiki/2011_England_riots, May 2014.

[2] Wikipedia, "2013 boston marathon." http://en.wikipedia.org/wiki/2013_Boston_Marathon, May 2014.

[3] "Bioid face database." https://www.bioid.com/About/BioID-Face-Database, Nov. 2014.

[4] A. Savran, N. Alyz, H. Dibekliolu, O. eliktutan, B. Gkberk, B. Sankur, and L. Akarun, "Bosphorus database for 3d face analysis," in Biometrics and Identity Management (B. Schouten, N. Juul, A. Drygajlo, and M. Tistarelli, eds.), vol. 5372 of Lecture Notes in Computer Science, pp. 47–56, Springer Berlin Heidelberg, 2008.

[5] "Facs - paul ekman group." http://www.paulekman.com/facs/, Nov. 2014.

[6] P. Ekman and W. V. Friesen, "Measuring facial movement," Environmental psychology and nonverbal behavior, vol. 1, no. 1, pp. 56–75, 1976.

[7] "Fg-net aging database." http://www-prima.inrialpes.fr/FGnet/html/benchmarks.html.

[8] P. Phillips, P. Flynn, T. Scruggs, K. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min, and W. Worek, "Overview of the face recognition grand challenge," in Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol. 1, pp. 947–954 vol. 1, June 2005.

[9] T. F. Cootes, A. Hill, C. J. Taylor, and J. Haslam, "The use of active shape models for locating structures in medical images," in Proceedings of the 13th International Conference on Information Processing in Medical Imaging, IPMI '93, (London, UK, UK), pp. 33–47, Springer-Verlag, 1993.

[10] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham, "Active shape models – their training and application," Comput. Vis. Image Underst., vol. 61, pp. 38–59, jan 1995.

[11] T. F. Cootes, C. J. Taylor, and A. Lanitis, "Active shape models: Evaluation of a multi-resolution method for improving image search," 1994.

[12] T. Cootes, C. Taylor, and M. M. Pt, "Statistical models of appearance for computer vision," 2000.

[13] B. Barbour and K. Ricanek, Jr., "An interactive tool for extremely dense landmarking of faces," in Proceedings of the 1st International Workshop on Visual Interfaces for Ground Truth Collection in Computer Vision Applications, VIGTA '12, (New York, NY, USA), pp. 13:1–13:5, ACM, 2012.

[14] I. L. Dryden and K. V. Mardia, Statistical Shape Analysis. John Wiley, Oct 1998.

[15] M. B. Stegmann and D. D. Gomez, "A brief introduction to statistical shape analysis," Mar 2002.

[16] T. Cootes, G. Edwards, and C. Taylor, "Active appearance models," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 23, pp. 681–685, Jun 2001.

[17] M. A. Karl Ricanek, Eric Patterson, "Age related morphological changes: Effects on facial recognition technologies," July 2003.

[18] "Face aging group." `http://www.faceaginggroup.com`, Nov. 2014.

[19] H. Galoogahi and T. Sim, "Face sketch recognition by local radon binary pattern: Lrbp," in Image Processing (ICIP), 2012 19th IEEE International Conference on, pp. 1837–1840, Sept 2012.

[20] X. Wang and X. Tang, "Face photo-sketch synthesis and recognition," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 31, pp. 1955–1967, Nov 2009.

[21] X. Wang and X. Tang, "Face photo-sketch synthesis and recognition," 2009.

[22] Z. Khan, Y. Hu, and A. Mian, "Facial self similarity for sketch to photo matching," in Digital Image Computing Techniques and Applications (DICTA), 2012 International Conference on, pp. 1–7, Dec 2012.

[23] Y. Zhang, C. McCullough, J. Sullins, and C. Ross, "Hand-drawn face sketch recognition by humans and a pca-based algorithm for forensic applications," Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, vol. 40, pp. 475–485, May 2010.

[24] J. Choi, A. Sharma, D. Jacobs, and L. Davis, "Data insufficiency in sketch versus photo face recognition," in Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on, pp. 1–8, June 2012.

[25] T. Cootes, G. Edwards, and C. Taylor, "Active appearance models," Proc. European Conf. Computer Vision, vol. 2, pp. 484–498, 1998.

[26] K. R. A. Sethuram, E. Patterson and A. Rawls, "Improvements and performance evaluation concerning synthetic age progression and face recognition affected by adult aging," June 2009.

[27] Y. H. Kwon and N. da Vitoria Lobo, "Age classification from facial images," in Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on, pp. 762–767, Jun 1994.

[28] S. Izadpanahi and O. Toygar, "Geometric feature based age classification using facial images," in Image Processing (IPR 2012), IET Conference on, pp. 1–5, July 2012.

[29] T. Kalansuriya and A. Dharmaratne, "Facial image classification based on age and gender," in Advances in ICT for Emerging Regions (ICTer), 2013 International Conference on, pp. 44–50, Dec 2013.

[30] A. Tharwat, A. Ghanem, and A. Hassanien, "Three different classifiers for facial age estimation based on k-nearest neighbor," in Computer Engineering Conference (ICENCO), 2013 9th International, pp. 55–60, Dec 2013.

[31] D. T. S. T. J. Golomb, B. A. Lawrence, "Sexnet: A neural network identifies sex from human faces," in Advances in Neural Information Processing Systems, pp. 572–577, 1991.

[32] Z. Yang, M. Li, and H. Ai, "An experimental study on automatic face gender classification," in Pattern Recognition, 2006. ICPR 2006. 18th International Conference on, vol. 3, pp. 1099–1102, 2006.

[33] "The mono project." http://www.mono-project.com/Main_Page, May 2014.

[34] "Infragistics .net controls." http://www.infragistics.com, May 2014.

[35] "Nullsoft scriptable install system." http://sourceforge.net/projects/nsis/, May 2014.

[36] "Open computer vision library." http://www.opencv.org, May 2014.

[37] "Boost c++ libraries." http://www.boost.org, May 2014.

[38] "Creating a windows form user control." http://msdn.microsoft.com/en-us/library/aa302342.aspx, Nov. 2014.

[39] "Google picks up pittpatt." http://www.cmu.edu/homepage/computing/2011/summer/google-acquires-pittpatt.shtml, Nov. 2014.

[40] L. Rao, "Google acquires facial recognition software company pittpatt." http://techcrunch.com/2011/07/22/google-acquires-facial-recognition-software-company-pittpatt/, July 2011.

[41] S. Milborrow, "Locating facial features with active shape models," 2007.

[42] A. Sethuram, J. Saragih, K. Ricanek, and B. Barbour, "Extremely dense face registration: Comparing automatic landmarking algorithms for general and ethno-gender models," in Biometrics: Theory, Applications and Systems (BTAS), 2012 IEEE Fifth International Conference on, pp. 135–142, Sept 2012.

[43] S. U. Oya eliktutan and B. Sankur, "A comparative study of face landmarking techniques," Mar. 2013.

[44] S. Milborrow and F. Nicolls, "Active Shape Models with SIFT Descriptors and MARS," VISAPP, 2014.

[45] S. Milborrow and F. Nicolls, "Locating facial features with an extended active shape model," ECCV, 2008.

[46] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," ACM Transactions on Intelligent Systems and Technology, vol. 2, pp. 27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[47] Q. Ferry, J. Steinberg, C. Webber, D. R. FitzPatrick, C. P. Ponting, A. Zisserman, and C. Nellåker, "Diagnostically relevant facial gestalt information from ordinary photos," eLife, vol. 3, 2014.

[48] C. D. Lorish and R. Maisiak, "The face scale: A brief, nonverbal method for assessing patient mood," Arthritis Rheumatism, vol. 29, no. 7, pp. 906–909, 1986.

Appendices A

MIDO CSV Import File Format

The CSV file format for MIDO must contain a header and at least the fields specified in Table 5.2. Optionally, the CSV file may contain the additional fields shown in Table 5.4.

SubjectID	nvarchar[50]
ImagePath	nvarchar[255]
DatasetID	int[4]

Table 5.2: Minimum MIDO CSV import requirements

For example, a simple CSV file with the minimum requirements might look like the following:

```
SubjectID,ImagePath,DatasetID
MORPH001,path/to/image/morph0001.png,1
MORPH002,path/to/image/morph0002.png,1
MORPH003,path/to/image/morph0003.png,1
```

The CSV file may also specify an Update column that contains a 1 or a 0. This allows MIDO, if it finds a duplicate entry for a record with the bit set, to update the entry with the imported data. If Update is set to 0 or omitted, the duplicate record will not be imported. This is different from a typical ”overwrite” flag as MIDO will attempt to merge the two records rather than replace the original record with the new record.

OcclusionID is special because you may specify more than one occlusion for an image. For example, an image may contain both a beard, moustache, and glasses. To specify several occlusions at once, you may list out each occlusion ID in a string separated by a semi-colon. For example, ”1;4;6” would cause MIDO to add 3 occlusions to this record. If only one occlusion is present, you may specify the single integer ID value. MIDO will automatically detect the difference and choose the correct parsing method.

DOB or Date of Birth, is also a somewhat unique field. Normally dates are specified in standard day/month/year or month/day/year formats however, many datsets were collected with month/year date of births. In this case, when MIDO detects two integers

ImageIndex	int[4]
DateTaken	date
Eye_x1	float
Eye_y1	float
Eye_x2	float
Eye_y2	float
Age	int[4]
Weight	int[4]
Height	int[4]
ExpressionID	int[4]
ModalityID	int[4]
OcclusionID	int[4] or char[255] (see below)
ImageHeight	int[4]
ImageWidth	int[4]
ImageChannels	int[4]
Name	nvarchar[100]
DOB	varchar[50] (see below)
Email	varchar[100]
Gender	char[1]
Race	char[1]
PublicRelease	bit
Pitch	float
Yaw	float
Roll	float
Lumens	float
Azimuth	float
Elevation	float
Comment	varchar[255]

Table 5.4: MIDO CSV File Format

separated by forward slash, it will convert that to a date using 1 for the day of the month for use in age calculations.

If Age is omitted but DOB and DateTaken are provided, Age will be automatically calculated as DateTaken - DOB.

Appendices B

The Picture Class

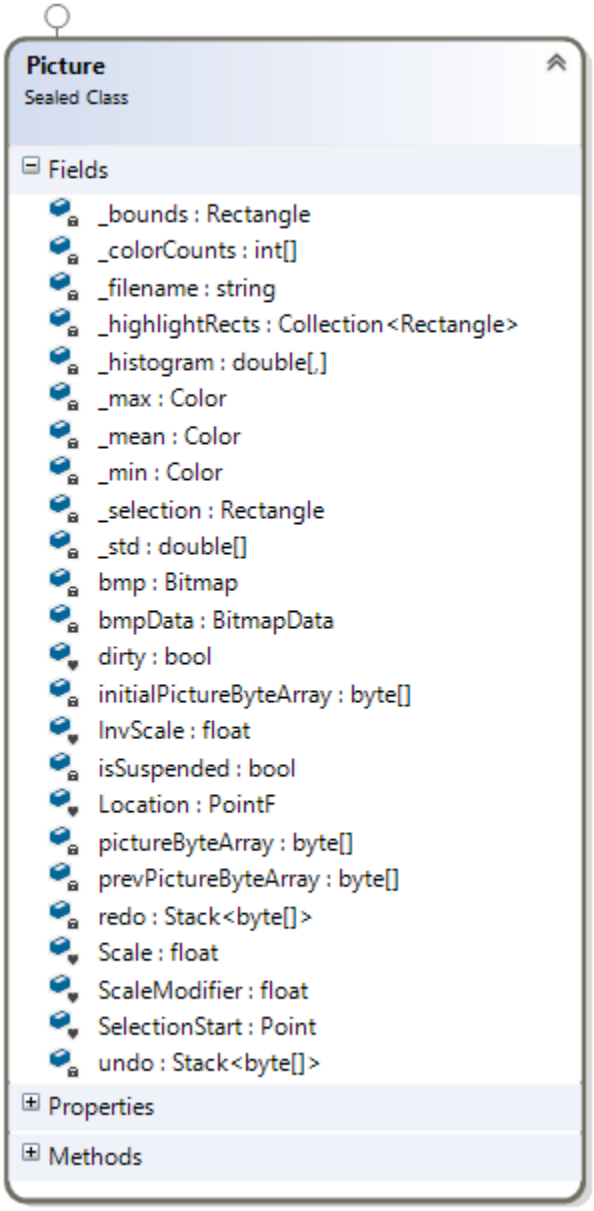

Figure 5.45: Picture Class Fields

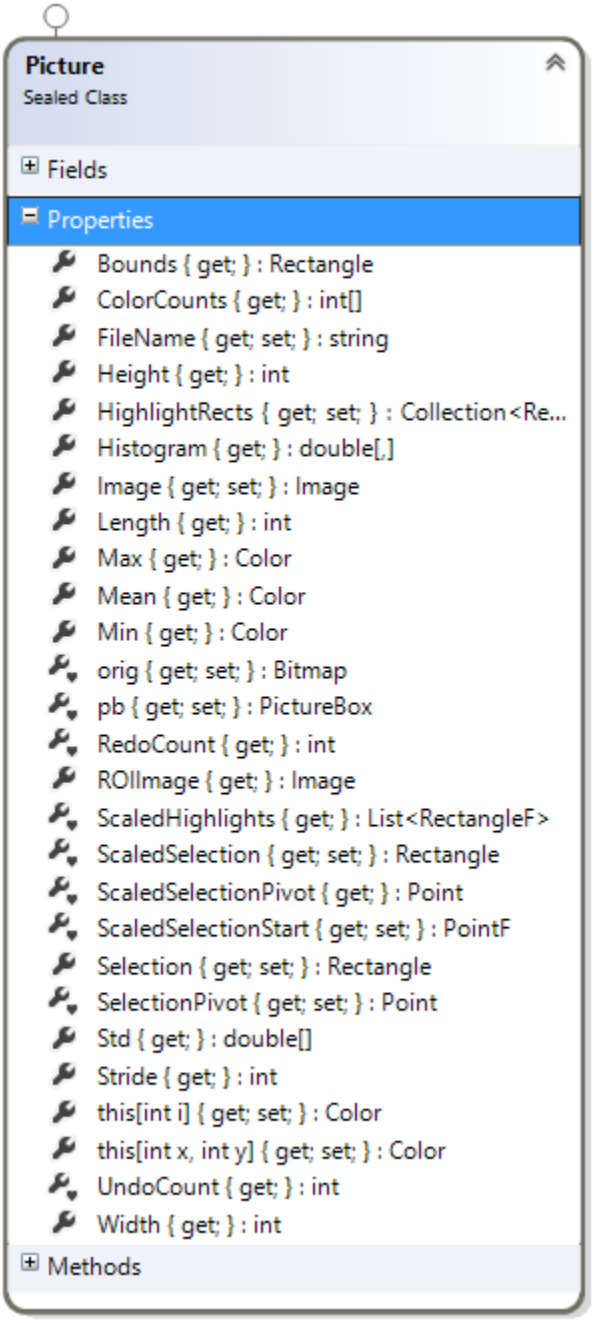

Figure 5.46: Picture Class Properties

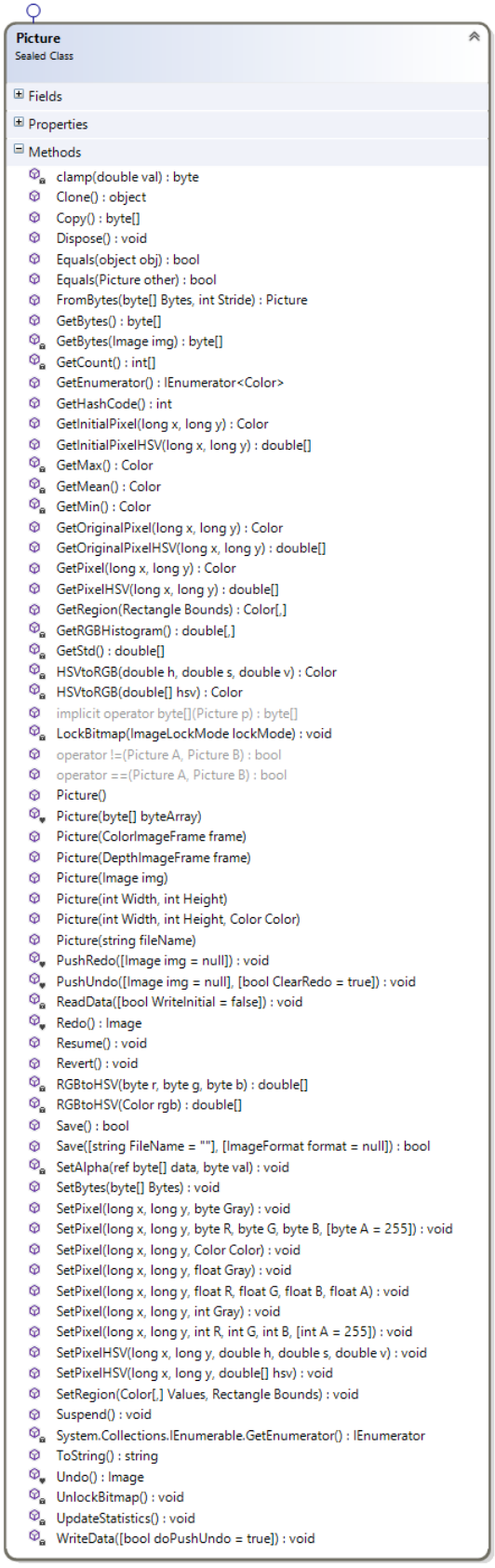

Figure 5.47: Picture Class Methods

```
[PluginMenuGroup("Image Processing", "Grayscale")]
public static void Test(Picture picture)
{
  picture.Suspend();
  Parallel.For(0, picture.Height, r =>
  {
    Parallel.For(0, picture.Width, c =>
    {
      var pixel = picture.GetPixel(r, c);
      var gray = (byte)((pixel.R + pixel.G + pixel.B) / 3);
      picture.SetPixel(r, c,
                       Color.FromArgb(gray, gray, gray));
    });
  });
  picture.Resume();
}
```

Listing 10: Using the Picture class to convert to grayscale in parallel