MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

CENG 491 – DETAILED DESIGN REPORT

MİLSOFT - PHOTOGRAMMETRY LAB PROJECT [PHOTOLAB]

Hanife ÜNAL

Meryem AYAS

Serap ATILGAN

Serra Sinem TEKİROĞLU

18 January 2008

TABLE OF CONTENTS

1 INTRODUCTION

While writing Initial Design Report, the general design of PHOTOLAB was started to be constructed with the drawing diagrams and specification of requirements.Now, as a group 'Bad Sector', we prepare this report in order to demonstrate the last and final design of our product PHOTOLAB . As a result of our researches, we become aware of what basic functionalities we are required to implement and what features can we add our product. So, at this report we aim to let you know Photolab's design issues, its features and how we implement these features etc.While preparing initial design report, initial design report will be a main guide for us.In order to clarify every single detail of the reader's mind ,we explained our design process via various diagrams. We have used Data Flow Diagrams, Class Diagrams,Sequence Diagrams, State-Transition etc.According to the feedback of our project assistant,we corrected our ER diagrams in initial design report.

1.1 Project Definition and Scope

Photogrammetry is to get reliable information about physical objects of photographic images. For instance , photogrammetry is used while we need to convert two dimensional image into a three dimensional one. Therefore, we can find the correct location and elevation of a point in earth using photogrammetry in multiple images taken from different positions. When we narrow down the scope of photogrammetry to our project, we face with the images collected by Unmanned Air Vehicles(UAVs). Nowadays, in all over the world, new projects have been launched for the UAV systems whose role in defense concepts is becoming more important, and the expenses for UAV systems are increasing above the expectations.
UAVs can do exploration, observation and target detection with the help of a camera that is mounted on them. The images collected by UAVs are just a collection of photos without processing. To make use of these photos, they have to be processed and so we get meaningful outputs. This process is called Image Exploitation. Image exploitation, by using some techniques, makes use of image processing algorithms for information extraction. This project is about Image Exploitation Systems. The data collected from the Unmanned Air Vehicle is not just a collection of 2D pixels, it is a 3D view of the Earth's surface; therefore the image has a depth. In addition, the images may not be taken at the same time or at the same attitude, so there are some difficulties in this project and we need to follow a proper order during this project. The expected four methods and their proper order are as follows:

- ✓ **Digital Elevation Model (DEM):** A three-dimensional surface map of an area is usually stored as a grid of elevation points called a Digital Elevation Model (DEM). DEM simply brings depth as third dimension to a two dimensional aerial image. Photolab takes one or more images as input and estimates the height of each point in these images. As an output we obtain exact location of the image in earth surface. So, using a DEM, a user can view an area in three dimensions, giving a clearer understanding of the problem.
- ✓ **Orthophoto:** An orthophoto is an aerial photograph that has been geometrically corrected to remove geometric distortion caused by camera tilt and differences in elevation. The digital orthophoto is a photographic image of the terrain - but more importantly, it is true to scale and therefore accurate distances and areas can be measured. These

orthophoto images are well suited for detail planning and analysis of what exists on the ground.

- ✓ **Mosaic:** The main goal of mosaicing is to combine the images to create an image with a wide perspective. To generate mosaic, we have to determine the matching points in different but related images. By applying the appropriate transformations and merging the overlapping regions of warped images, it is possible to construct a single image covering the entire visible area of the scene. This merged single image is the motivation for the term ``mosaic''.
- ✓ **Super Resolution:** Super-resolution is a term for a set of methods of increasing image or video resolution. All these methods are based on same idea: using information from a set of low-resolution images to create one or a set of high-resolution images. These methods try to extract details from frames to reconstruct other frames.

1.2 Application Areas

The images collected from UAVs' cameras have wide range usage areas. UAVs are currently used in a number of military roles such as gaining military or medical information and also they are also used in growing number of civil applications. After processing these images to get meaningful images, we can make use of these images in many areas. For example, we can make use of these images in searching and rescuing humans trapped in collapsed buildings, in security issues like scenes of murders, burglars or smugglings, or in damage detection issues, in industrial applications ranging from traffic engineering to mining engineering; anywhere geometric properties of objects are useful, in computer applications, such as games, atlases, and encyclopedias and also in realistic display of landforms for such diverse areas as pilot training, weapons guidance, and landscape architecture.

1.3 System Requirements

We examined the system requirements in two parts: hardware requirements and software requirements.

1.3.1 Hardware Requirements

While determining the hardware requirements, we look after the similar software programs that have been developed till now. The software products that we mentioned at market research part generally had similar hardware requirements. The following requirements for hardware seem to be minimal and enough for our project development:

- P4 or Equivalent Processor
- Minimum 512 MB RAM
- 3D Graphics Card
- Minimum 40 GB HDD

1.3.2 Software Requirements

We will use several tools for different phases during development of the project. The following requirements during documentation and development phases are necessary for project.

Documentation Phase

- **Microsoft Office 2007**
- **Milestone Professional 2008**
  For drawing Gantt Charts for the reports. It is more functional than SmartDraw while drawing Gantt Charts, so we preferred Milestone Professional 2008.
- **Diagram Studio 5.3**
  For drawing the diagrams and charts for the reports we were using SmartDraw. However, from now on we draw our diagrams in Diagram Studio since it is more flexible.
- **Adobe Acrobat Reader**
  For writing and submitting the reports. We write our reports on Microsoft Office 2007 and submit them after we convert it to pdf format.

Development Phase

- **Windows XP and Linux**
  PHOTOLAB must be running operating system independent, so we will work on both Windows and Linux.
- **Microsoft Visual Studio .NET 2003**
  From our previous experiences, we are used to work on Visual Studio .NET. Because the language we will implement PHOTOLAB is C++, and .NET provides easy usage, we preferred it.
- **wxWidgets GUI Toolkit**
  We will use wxWidgets in GUI design. Software is advised by Milsoft and because it is working with C++, we preferred it. The initial phase of implementation has started with wxWidgets. It has reliable and easy to implement event handlers. In addition, GUI design fragments are user-friendly and clear to understand.
  The class architecture which will be clearly described in this documentation, partially contains wxWidgets related structures, especially data types. The reason why Detailed Design covers those structures is current state of Photolab exactly uses them.

- **Microsoft Visual Assist**
  It is an add-in for Visual Studio .NET and provides generally more accurate and complete code suggestions. It also has the syntax highlighting feature.
- **OpenGL Library**
  Since it can be run under both Windows and Linux platforms and it is compatible with C++, we preferred OpenGL libraries for 2D and 3D image visualization.
- **OSSIM (Open Source Software Image Map)**
  We preferred it since OSSIM is a high performance software system for image processing, geographical information systems and photogrammetry.
- **OpenCV (Open Source Computer Vision Library)**
  We preferred to use OpenCV for its supporting calibration techniques, feature detection techniques and many other image processing functions we are going to use for this project. OpenCV is almost the most important part of Photolab Photogrammetry algorithms. In addition to Photogrammetric facilities, we take

advantage of mathematical and geometrical features of OpenCV, like matrix multiplications. Besides WxWidgets, OpenCV related data types are also used in detailed design. Function parameters, return types and Photolab's own data structures have OpenCV specific types by relying on the current implementations.

2 DESIGN CONSTRAINTS & CONSIDERATIONS

In the light of the fact that good limitation leads good design, Bad Sector team has formed the constraints that put necessary limits to the design of the project. The constraints can be divided into categories as:

- Timing Constraints
- Programming Language & Software Constraints
- Performance Constraints
- Quality Constraints
- Legal/Ethical Constraints
- Group Members Related Constraints

Below are the explanations which assert how each constraint affects the system and design of the software.

2.1 Timing Constraints

Timing constraints play an important role for the successful completion of the project by the end of the year. Bad Sector team has scheduled PHOTOLAB according the milestones in the syllabus. In the light of the schedule, the team will represent a prototype demo by the end of each semester. With the help of the assignments given by Milsoft the team gains experience on managing the time. Since it is a hard and long-term project, in order to minimize the potential risks of not completing the project on time, Bad Sector team moves on with the high conception of its duties. As a result of this conception, the team updates detailed time schedule regularly, and the team members pay attention to follow the schedule on the Gantt chart in order to complete the project successfully on time.

2.2 Programming Language & Software Constraints

Bad Sector team has decided to implement the PHOTOLAB with C++ which is one of the two options (JAVA or C++) that Milsoft had suggested for the implementation of the project. That C++ supports best open source libraries on Geographical Information Systems and that C++ is the team members' and Milsoft' s primary choice for the implementation of the project are two main reasons for the selection of C++ as the programming language for the project. Bad Sector team uses wxWidgets GUI Toolkit for the design of Graphical User Interface and OpenCV as the main library for the development of Photogrammetry and File System Modules.

2.3 Performance Constraints

One of the important constraints of PHOTOLAB project is a satisfying performance. In order to accomplish a good performance (i.e. a fast and efficient execution), Bad Sector team has decided to search and inspect all algorithms, methods and approaches carefully before deciding on the method which will be used. Bearing mind that PHOTOLAB will be fast, efficient and easy to use, the team compares the algorithms and chooses the one which is faster, easy to implement and more accurate. In cases such that these three considerations are not satisfied at the same time, the team gives top priority firstly to accuracy, then to fastness and finally to easiness of the implementation.

2.4 Quality Constraints

Bad Sector team has decided to design PHOTOLAB with a good quality planning. Therefore, the team specifies quality related constraints in order to satisfy the quality plan according to below quality criteria specified by Bad Sector team:

- Completion on deadline:
  PHOTOLAB is expected to be completed at the end of the 2007-2008 spring semester of METU.
- Functionality:
  PHOTOLAB is expected to be functional in terms of suitability, accuracy, and interoperability.
- Reliability:
  PHOTOLAB is expected to be mature and error-free when the reliability is in consideration.
- Efficiency:
  PHOTOLAB is expected to be efficient on time behavior and resource utilization issues.
- Maintenance:
  PHOTOLAB is expected to be analyzable, testable, stable and upgradeable when the maintenance is in question.
- Comply with the naming, documentation and user interface standards:
  PHOTOLAB is expected to be in compliance with the standards put by the team.
- Portability :
  PHOTOLAB is expected to be portable in terms of installability, replaceability, and adaptability.

In the light of the thought that the way things are built affects how they can be built better, Bad Sector team develops PHOTOLAB in spiral process model. This model will provide compensation of the mistakes made during the implementation of the project with the help of the quality constraints specified above.

2.5 Legal/Ethical Constraints

One other important consideration that Bad Sector team pays attention is legal/ethical constraints. Recognizing that the patent and copyright issues play an important role in an intellectual software

developer's life, the team develops PHOTOLAB taking patent and copyright issues into account. These previously stated constraints are also important since confidentiality agreement signed with MILSOFT requires not using a software component or even a piece of code which is not open-source. Therefore, Bad Sector team develops PHOTOLAB using only open-source libraries and information.

2.6 Group Members Related Constraints

Since the developers' programming and design skills and experiences play an important role during the design and implementation of the project, Bad Sector team takes the group member related risks into account. Gaining experience in every stage, the team has overcome the difficulties encountered during the design of the project. From now on, only the risk which is not being technically qualified on Wx Widgets, Libraries, Geographical Information Systems and the risk which is the existence of a bad situation in one or more team members' physical and psychological health are decided to be paid attention by the team. In order to decrease the effects of these potentials stated above, Bad Sector team plans not only giving equal work to all of the members (to give every member the chance to develop herself technically) but also meeting at least twice a week. The team also plans to consult the project supervisor and the authorized person of Milsoft on PHOTOLAB in case of a need.

3 ARCHITECTURAL and COMPONENT LEVEL DESIGNS

3.1 Photolab Modules

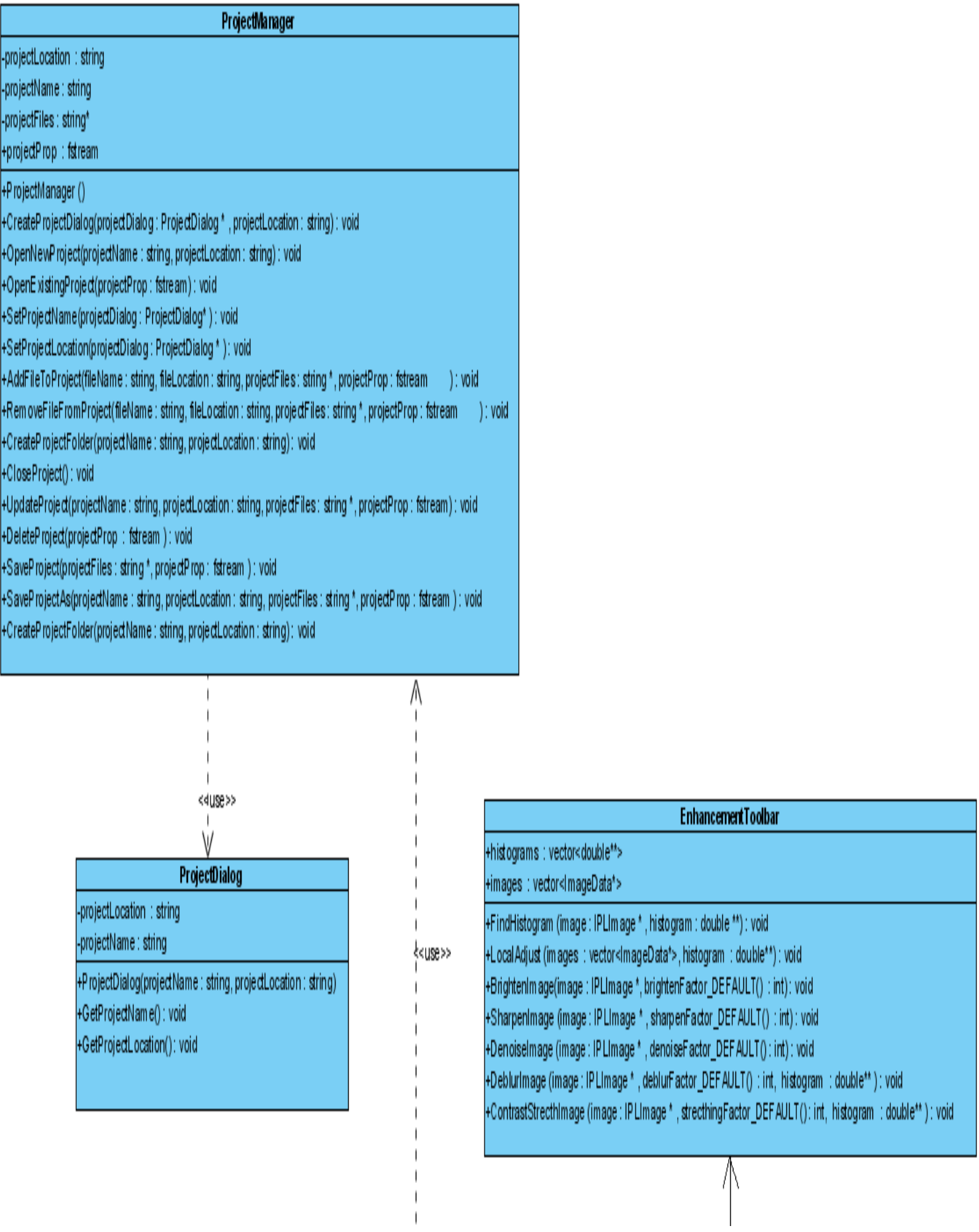
Photolab is designed on modular base for a systematic architecture. There are three modules in Photolab; namely, GUI, Photogrammetry, and File System Module. These modules are connected to each other according to data traffic.

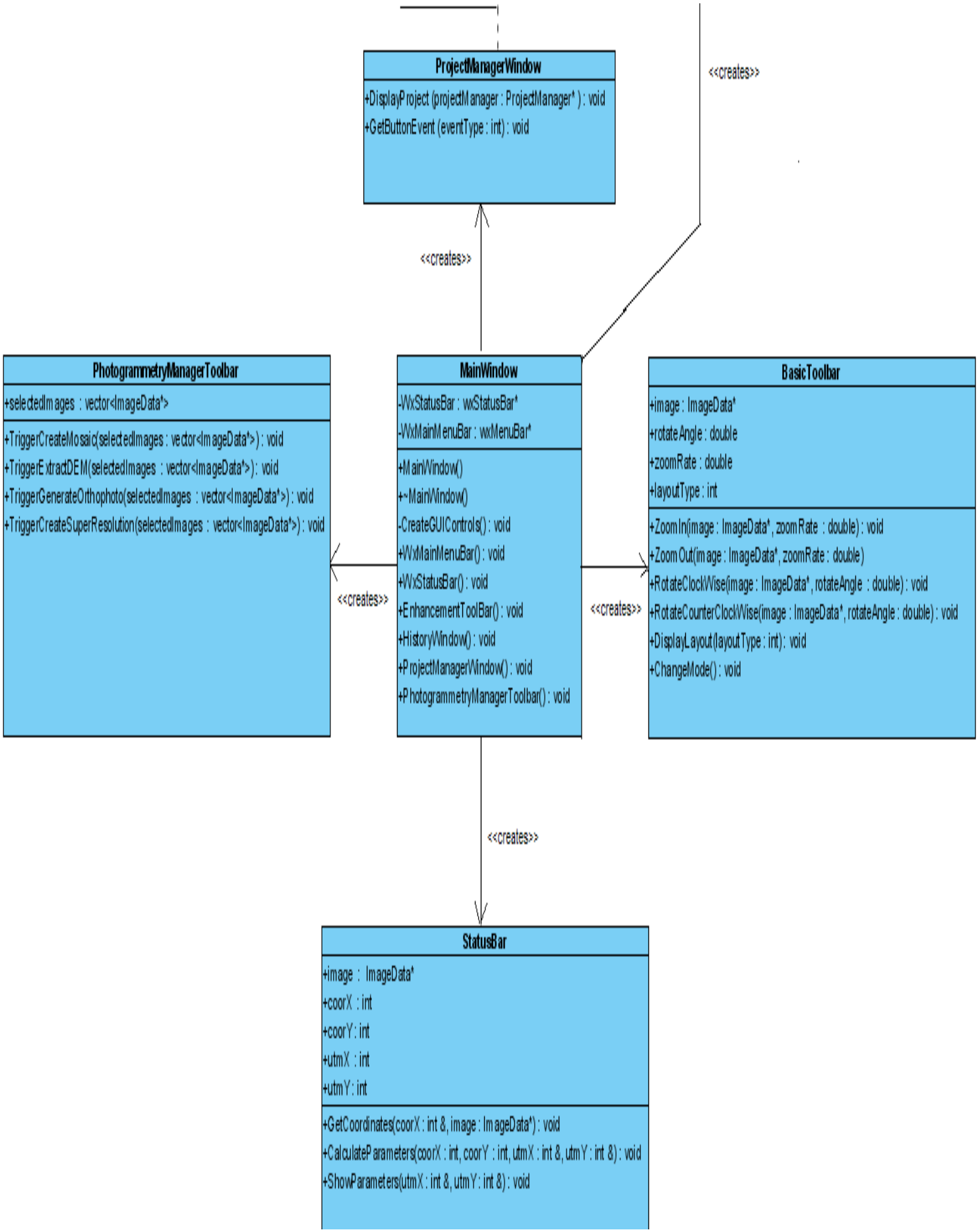
3.1.1 GUI Module

GUI module is central part of the modular architecture because it provides communication between the modules with the help of windows and toolbars. This module contains 8 classes which are BasicToolbar, MainWindow, ProjectManagerWindow, ProjectManager, ProjectDialog, EnhancementToolbar, StatusBar and PhotogrammetryToolbar classes. User sends operation requests by the help of events and these events trigger FileSystem module and Photogrammetry module classes. GUI distributes requests and delivers their responces to the user. The methods of the classes are triggered by the events from GUI module.

GUI provides user manual and automatic modes. In manual mode user selects tie points manually, or the other option is automatic mode in which the program finds tie points automatically. In addition GUI supports multiple image handling and visual layout options. User can either see the images on the same grid or as cascading windows.GUI module handles all the operations done on the project. Saving, opening, deleting, updating and such other basic project operations can be done by ProjectManager class. EnhancementToolbar class of GUI module has enhancement functions for better displaying purposes. It also has basic functions in BasicToolbar class such as

rotate and zoom in and out functions. StatusBar class methods only calculates the UTM coordinates and shows them on the status bar.The general class diagrams of the GUI module is as follows:

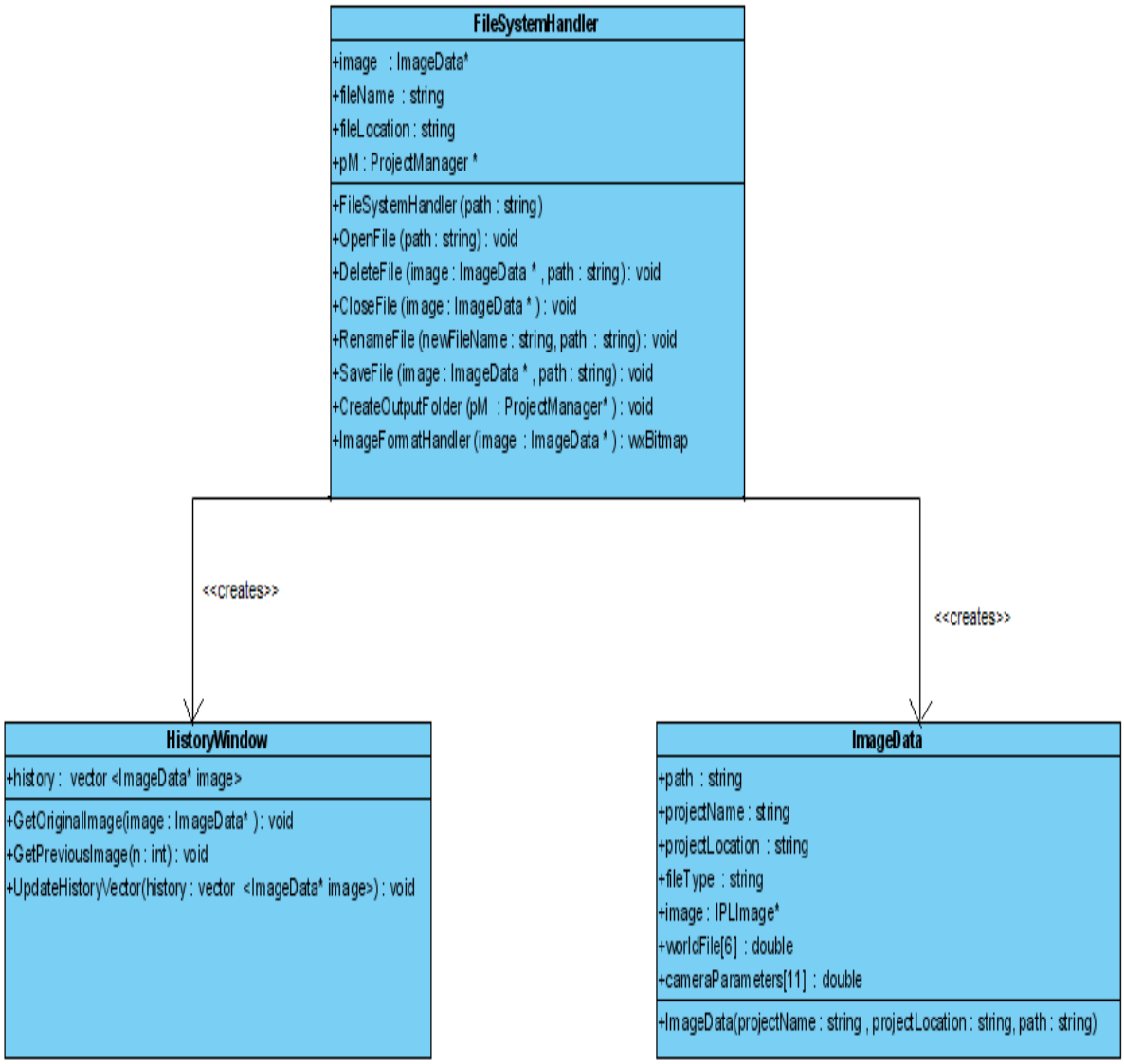

ProjectManager
-projectLocation : string
-projectName : string
-projectFiles : string*
+projectProp : fstream
+ProjectManager ()
+CreateProjectDialog(projectDialog : ProjectDialog * , projectLocation : string) : void
+OpenNewProject(projectName : string, projectLocation : string) : void
+OpenExistingProject(projectProp : fstream) : void
+SetProjectName(projectDialog : ProjectDialog*) : void
+SetProjectLocation(projectDialog : ProjectDialog *) : void
+AddFileToProject(fileName : string, fileLocation : string, projectFiles : string * , projectProp : fstream) : void
+RemoveFileFromProject(fileName : string, fileLocation : string, projectFiles : string * , projectProp : fstream) : void
+CreateProjectFolder(projectName : string, projectLocation : string) : void
+CloseProject() : void
+UpdateProject(projectName : string, projectLocation : string, projectFiles : string * , projectProp : fstream) : void
+DeleteProject(projectProp : fstream) : void
+SaveProject(projectFiles : string * , projectProp : fstream) : void
+SaveProjectAs(projectName : string, projectLocation : string, projectFiles : string * , projectProp : fstream) : void
+CreateProjectFolder(projectName : string, projectLocation : string) : void
<<use>>
ProjectDialog
-projectLocation : string
-projectName : string
+ProjectDialog(projectName : string, projectLocation : string)
+GetProjectName() : void
+GetProjectLocation() : void
<<use>>
EnhancementToolbar
+histograms : vector<double**>
+images : vector<ImageData*>
+FindHistogram (image : IPLImage * , histogram : double **) : void
+LocalAdjust (images : vector<ImageData*>, histogram : double**) : void
+BrightenImage(image : IPLImage * , brightenFactor_DEFAULT() : int) : void
+SharpenImage (image : IPLImage * , sharpenFactor_DEFAULT() : int) : void
+DenoiseImage (image : IPLImage * , denoiseFactor_DEFAULT() : int) : void
+DeblurImage (image : IPLImage * , deblurFactor_DEFAULT() : int, histogram : double**) : void
+ContrastStrecthImage (image : IPLImage * , strecthingFactor_DEFAULT() : int, histogram : double**) : void


ProjectManagerWindow
+DisplayProject (projectManager : ProjectManager*) : void
+GetButtonEvent (eventType : int) : void
<<creates>>
<<creates>>
PhotogrammetryManagerToolbar
+selectedImages : vector<ImageData*>
+TriggerCreateMosaic(selectedImages : vector<ImageData*>) : void
+TriggerExtractDEM(selectedImages : vector<ImageData*>) : void
+TriggerGenerateOrthophoto(selectedImages : vector<ImageData*>) : void
+TriggerCreateSuperResolution(selectedImages : vector<ImageData*>) : void
MainWindow
-WxStatusBar : wxStatusBar*
-WxMainMenuBar : wxMenuBar*
+MainWindow()
+~MainWindow()
-CreateGUIControls() : void
+WxMainMenuBar() : void
+WxStatusBar() : void
+EnhancementToolBar() : void
+HistoryWindow() : void
+ProjectManagerWindow() : void
+PhotogrammetryManagerToolbar() : void
BasicToolbar
+image : ImageData*
+rotateAngle : double
+zoomRate : double
+layoutType : int
+ZoomIn(image : ImageData*, zoomRate : double) : void
+ZoomOut(image : ImageData*, zoomRate : double)
+RotateClockWise(image : ImageData*, rotateAngle : double) : void
+RotateCounterClockWise(image : ImageData*, rotateAngle : double) : void
+DisplayLayout(layoutType : int) : void
+ChangeMode() : void
<<creates>>
<<creates>>
<<creates>>
StatusBar
+image : ImageData*
+coorX : int
+coorY : int
+utmX : int
+utmY : int
+GetCoordinates(coorX : int &, image : ImageData*) : void
+CalculateParameters(coorX : int, coorY : int, utmX : int &, utmY : int &) : void
+ShowParameters(utmX : int &, utmY : int &) : void

3.1.2 File System Module

This module is responsible of handling basic file operations correctly and it also provides working on various image formats by converting them into bitmap files. According to process on multiple and large images, file operations must be managed in most efficient way. To succeed in efficiency, this module contains 3 classes which are FileSystemHandler, HistoryWindow and ImageData classes.

FileSystemHandler class handles file operations on files that are in project folder. These files can also be video files besides image files. By the help of our library , openCV, we capture frames from video and process them as basic image files. Another class in this module is HistoryWindow class that is just for making the user's job easy, for example, the user may not want to lose the original image after he/she does any process on it, HistoryWindow class makes it possible to undo up to 3 previous processes done on image. ImageData class stores all the necessary information about the image such as world file information, camera parameters etc. The class diagram of file system module is below:

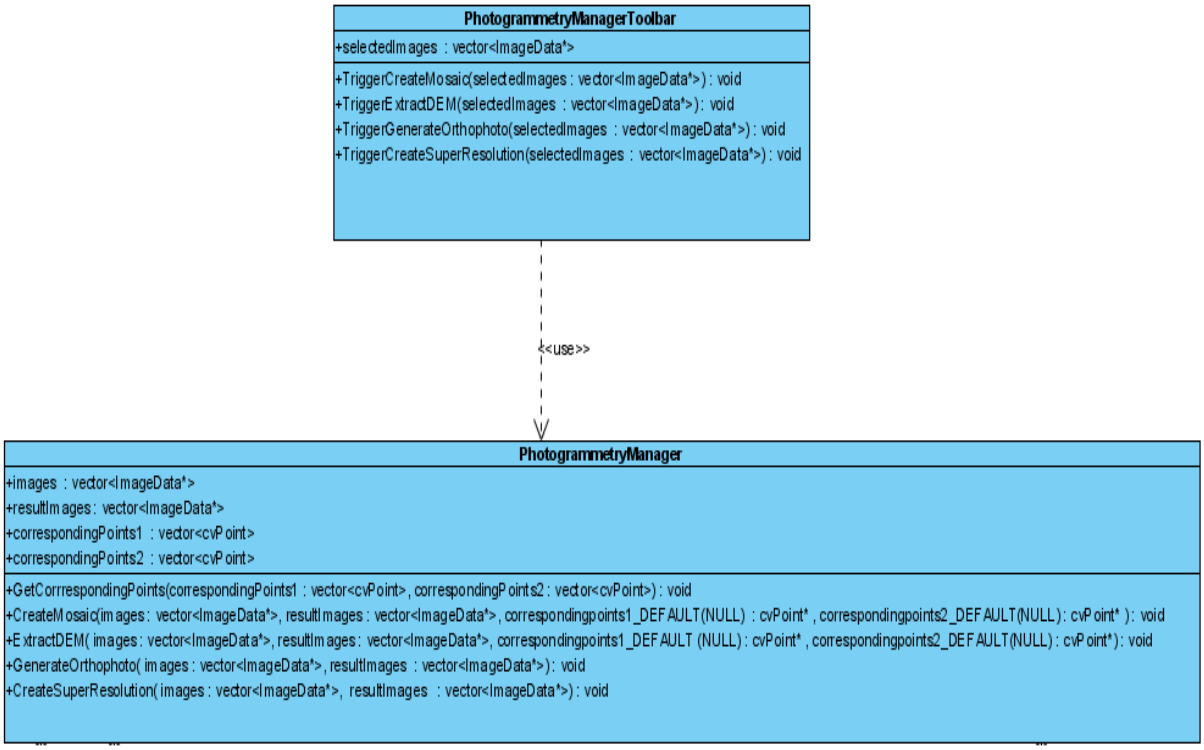


3.1.3 Photogrammetry Module

Photogrammetry Module is designed on three stages. In the base stage, module has four classes. These classes are DEM, Mosaic, Orthophoto and SuperResolution. Each of them uses Photolab's basic image object; ImageData class. OpenCV is the main library which has helper methods for these four classes, especially data structures that are already presented in this library. The basic algorithmic operations are performed on these base classes. The class members and methods as steps on Photogrammetry algorithms are described in detail at class diagrams part of the detailed design report.

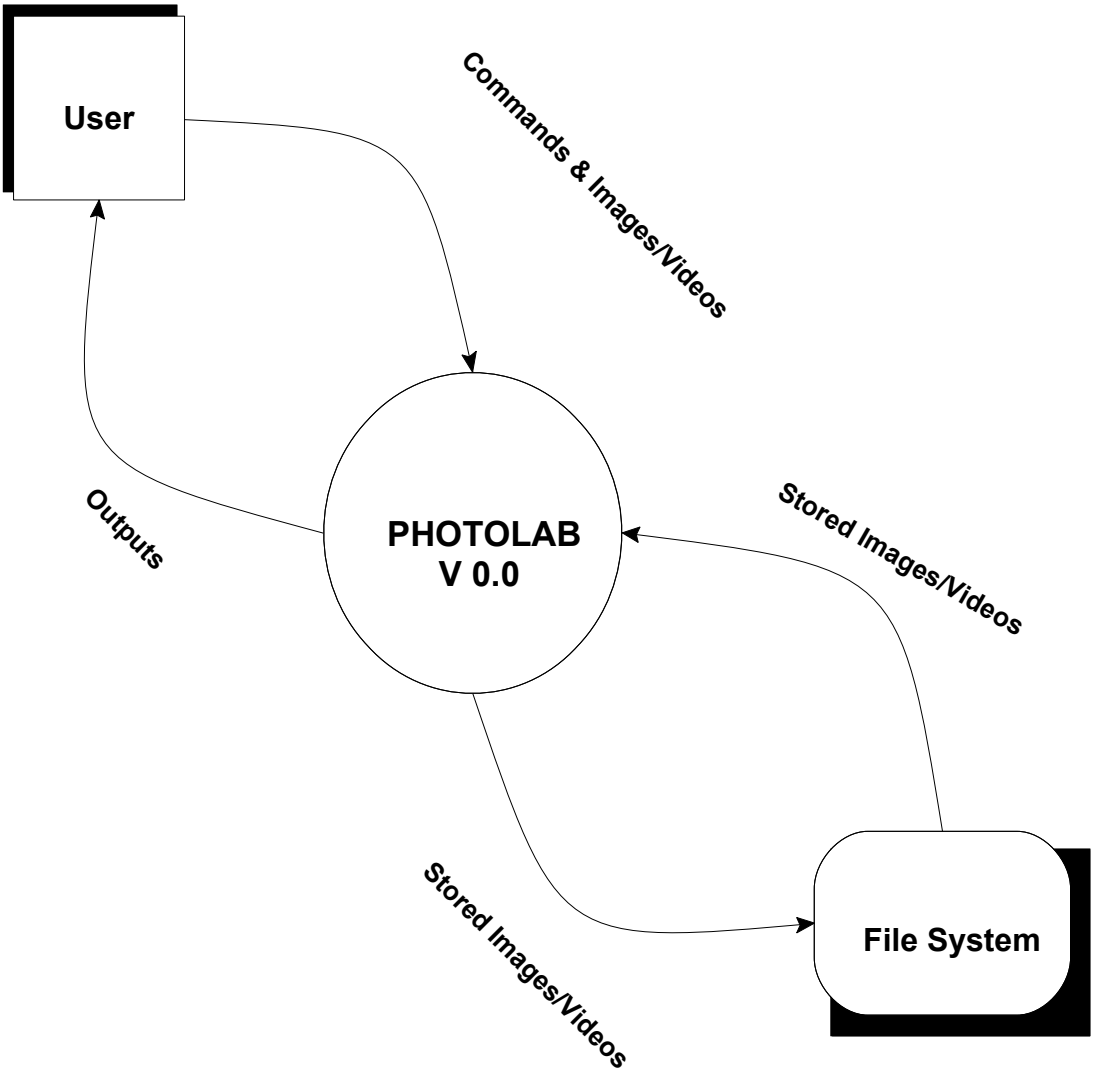
The second stage is PhotogrammetryManager Class. This class has methods that combines the algorithmic steps and creates objects of DEM, Mosaic, Orthophoto, or SuperResolution by assigning the object members. Except Orthophoto, all Photogrammetry operations will work with two input images. However, users usually will want to work on more than two images. To handle this situation we store input and output images in the vectors in PhotogrammetryManager Class and use simple loops to generate outputs. Orthophoto operations use image arrays as parameter to methods. More input images for Orthophoto, more correct results we can obtain.

The top stage is the event base stage. PhotogrammetryManagerToolbar Class basically triggers the PhotogrammetryManager Class methods depending on the users button clicks. This buttons are fixed on the PhotogrammetryManager toolbar. When main window is created, all toolbars included PhotogrammetryManager toolbar is activated and becomes ready to get events from user. This class has also store selected images that are selected for the DEM, Mosaic, Orthophoto, SuperResolution. The selected images can be determined by the user among the images that are belonged to the current Project. The class diagram of the top two stages of photogrammetry module is below:



3.2 Data Flow Diagram

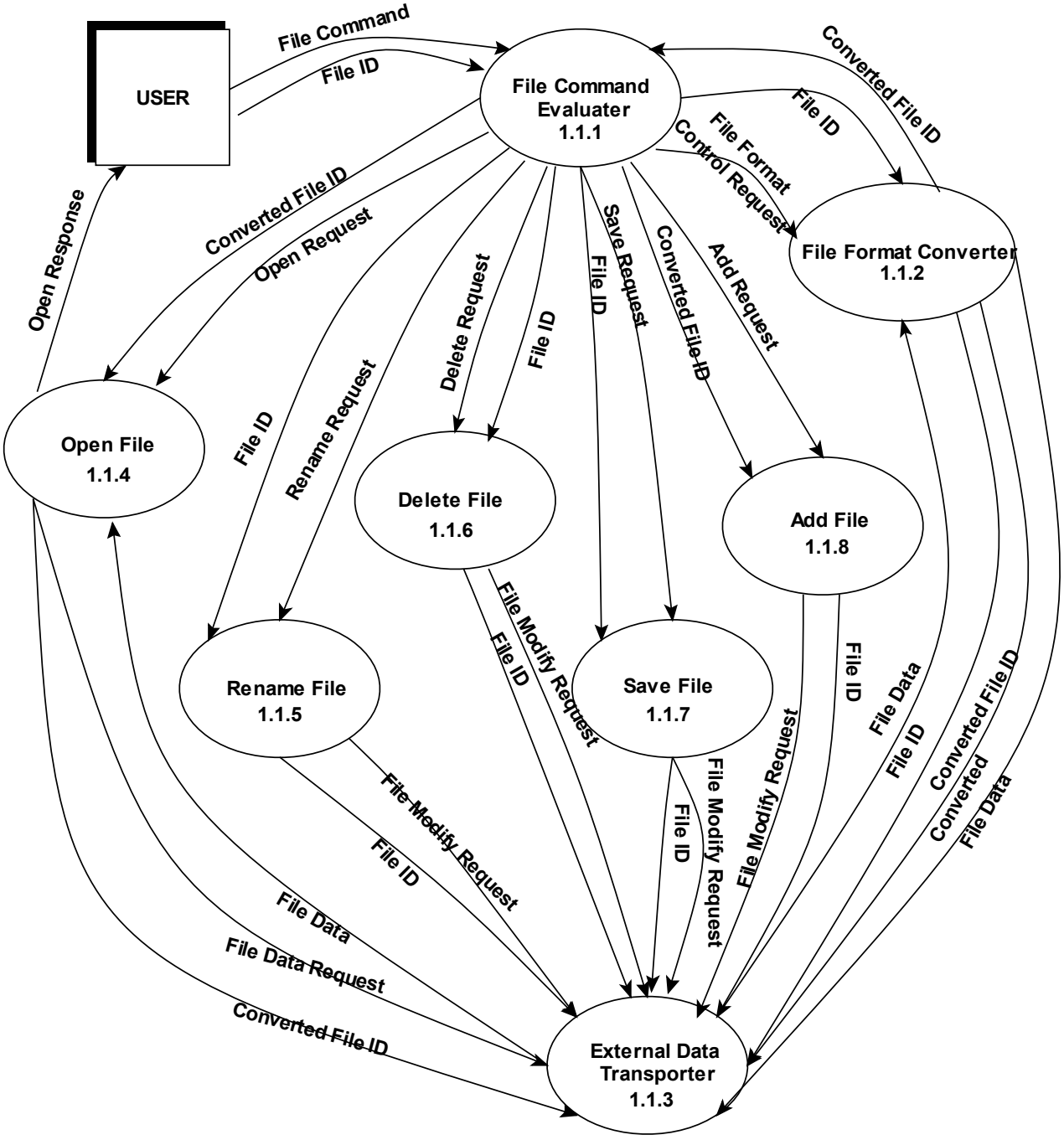
3.2.1 Level 0 DFD



3.2.2 Level 1 DFD

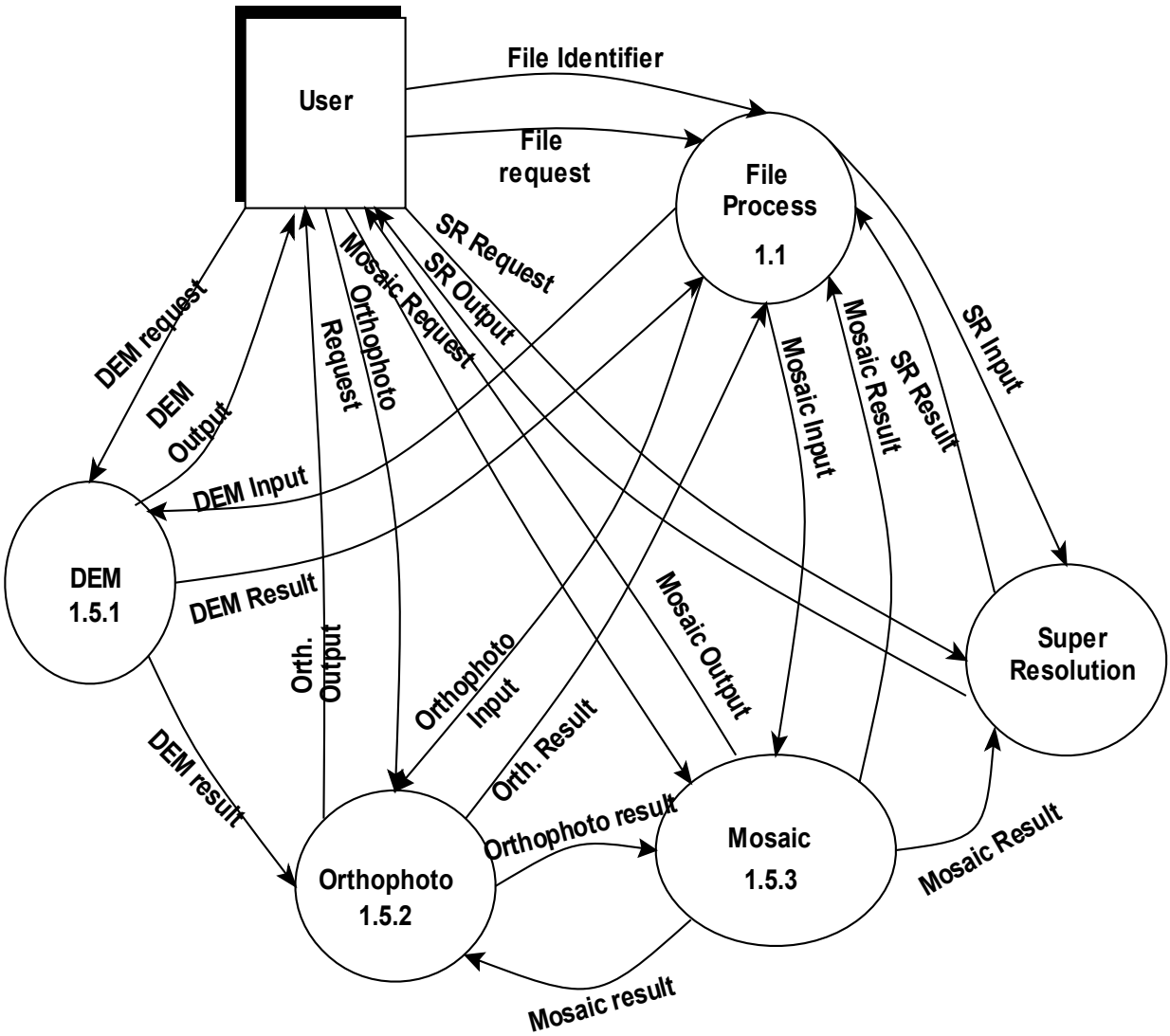
3.2.3 Level 2 DFD

File System Process



3.2.4 Level 2 DFD

Photogrammetry Process



3.3 Data Dictionary

Name	File Identifier
Where & How Used	File Process(1.1) INPUT Project Folder(D2) INPUT History Process(1.3) OUTPUT
Description	Identifier for the file consists of name of file and name of Project it belongs to.

Name	File Command
Where & How Used	File Process(1.1) INPUT USER OUTPUT
Description	Commands for file operations of adding, opening, deleting, saving and renaming.

Name	File Request
Where & How Used	File Process(1.1) INPUT Project Folder (D2) INPUT History Process(1.3) OUTPUT
Description	Request for the file from the file process with file identifier.

Name	File Response
Where & How Used	File Process(1.1) INPUT Project Folder (D2) OUTPUT
Description	Response of the file system to the file process.

Name	Project Identifier
Where & How Used	Project Folder (D2) INPUT Project Process(1.2) OUTPUT File Process(1.1) INPUT & OUTPUT
Description	Identifier of the project for the files it includes.

Name	Project Commands
Where & How Used	File Process(1.1) INPUT Project Process(1.2) OUTPUT
Description	Commands for opening, closing, updating, deleting, saving, displaying projects.

Name	Project Request

Where & How Used	Project Folder (D2) INPUT File Process(1.1) OUTPUT
Description	Request for the project from the file process with project identifier.

Name	Project Response
Where & How Used	File Process(1.1) INPUT Project Folder (D2) OUTPUT
Description	Response of the file system to the file process.

Name	User Command
Where & How Used	Project Process(1.3) INPUT USER OUTPUT
Description	Commands of the user about project.

Name	History Request
Where & How Used	History Process(1.2) INPUT USER OUTPUT
Description	User request to undo the processes done on images.

Name	Data Identifier
Where & How Used	Workspace(D1) INPUT File Process(1.1) OUTPUT
Description	Identifier for data that is a combination of file identifier and a flag of file that defines whether it is an image file, video file or world file.

Name	Data Request
Where & How Used	Workspace(D1) INPUT File Process(1.1) OUTPUT
Description	Request for data with data identifier from workspace.

Name	Data Response
Where & How Used	File Process(1.1) INPUT Workspace(D1) OUTPUT
Description	Response of the workspace to the data request of the file process.

Name	Photogrammetry Request
Where & How Used	Photogrammetry(1.5) INPUT USER OUTPUT
Description	Request of the user for a photogrammetry process on images.

Name	Photogrammetry Result
Where & How Used	USER INPUT Photogrammetry (1.5) OUTPUT
Description	Result of the photogrammetry process on images.

Name	Photogrammetry Input
Where & How Used	Photogrammetry(1.5) INPUT File Process(1.1) OUTPUT
Description	Input files for photogrammetry process.

Name	Photogrammetry Output
Where & How Used	File Process(1.1) INPUT Enhancement(1.4) INPUT Photogrammetry(1.5) OUTPUT
Description	Output files of photogrammetry process.

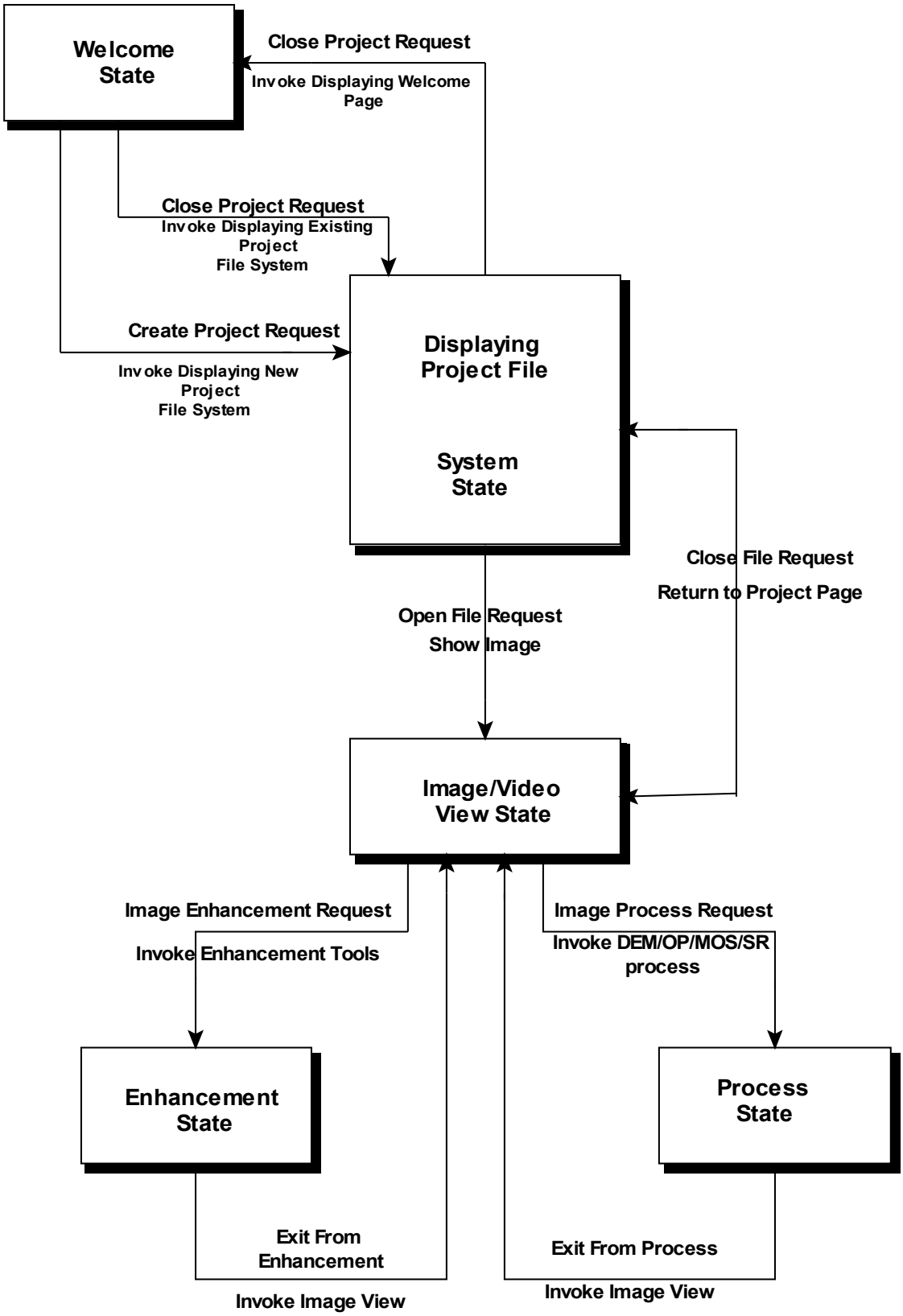
Name	Enhancement Input
Where & How Used	Enhancement(1.4) INPUT File Process(1.1) OUTPUT
Description	Input files for enhancement process.

Name	Enhancement Output
Where & How Used	File Process(1.1) INPUT Photogrammetry(1.5) INPUT Enhancement(1.4) OUTPUT
Description	Output files of enhancement process.

Name	Enhancement Request
Where & How Used	Enhancement(1.4) INPUT USER OUTPUT
Description	Request of the user for an enhancement process on images.

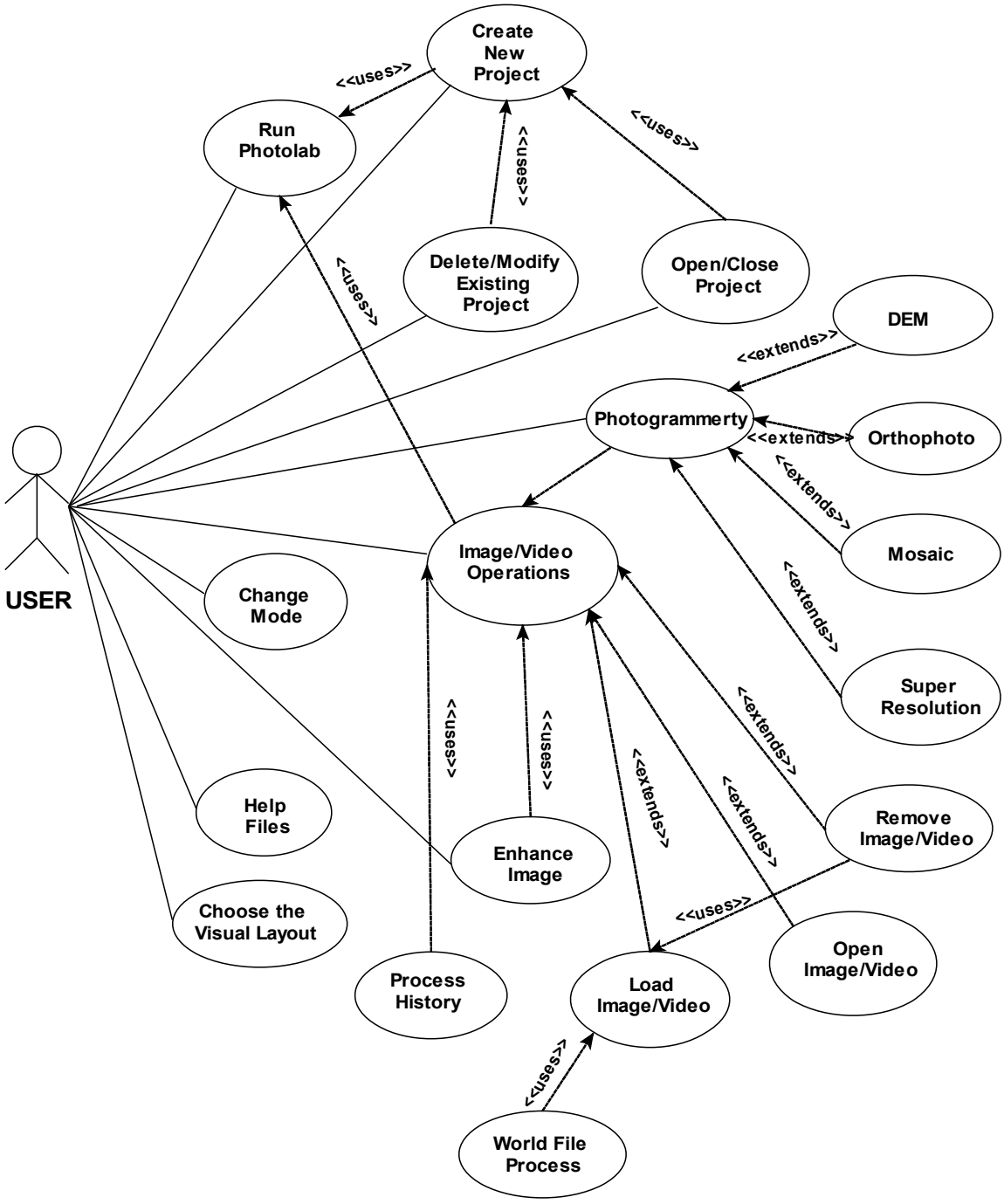
Name	Enhancement Response
Where & How Used	USER INPUT Enhancement(1.4) OUTPUT
Description	Response of the enhancement to the enhancement request.

3.4 State Transition Diagram


Welcome State
Close Project Request
Invoke Displaying Welcome Page
Close Project Request
Invoke Displaying Existing Project File System
Create Project Request
Invoke Displaying New Project File System
Displaying Project File
System State
Close File Request
Return to Project Page
Open File Request
Show Image
Image/Video View State
Image Enhancement Request
Invoke Enhancement Tools
Image Process Request
Invoke DEM/OP/MOS/SR process
Enhancement State
Process State
Exit From Enhancement
Invoke Image View
Exit From Process
Invoke Image View

4 OBJECT ORIENTED DIAGRAMS

4.1 Use Case Diagrams

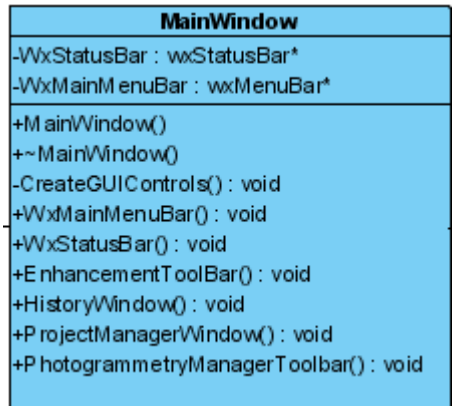


4.2 Class Diagrams

4.2.1 GUI Module Class Diagrams

The class diagrams of the GUI module are positioned below GUI module explanation.

4.2.1.1 MainWindow Class



- **wxMenuBar *WxMainMenuBar :** This menubar stores all basic file operation trigger handling buttons.
- **wxStatusBar *WxStatusBar :** This is needed to show mouse coordinate or world coordinate on the image.

- **MainWindow( )**

It is the constructor of the MainWindow class. The initializer function gets the wxFrame parameters from the constructor. In addition it uses CreateGUIControls to show each toolbar on the main window. This simple activate event handlers.

- **(void) CreateGUIControls( )**

All events handlers are created and showed on this function for the MainWindow constructor. Toolbar methods in the following will be called in this method.

- **(void) WxMainMenuBar( )**

This method creates and activates the MainMenuBar.

- **(void) WxStatusBar( )**

This method creates and activates StatusBar class on main window by using StatusBar Class.

- **(void)EnhancementToolBar( )**

This method creates and activates the Enhancement Toolbar on main window by using EnhancementToolbar Class.

- **(void)HistoryWindow( )**

This method creates an object of HistoryWindow class for main window.

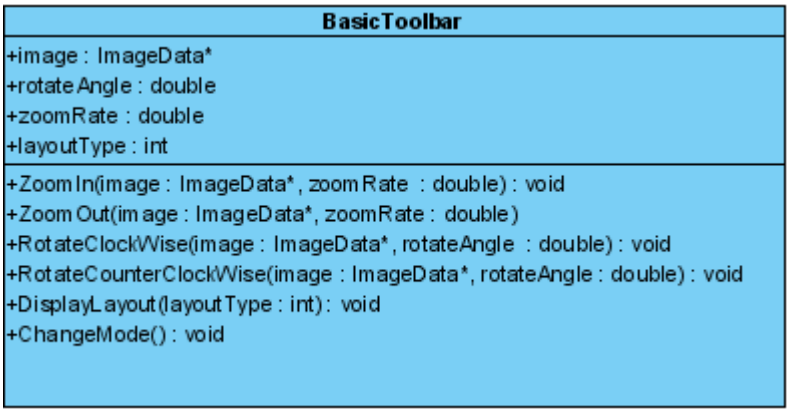
- **(void)ProjectManagerWindow( )**

This method creates an object of ProjectManagerWindow class for main window.

- **(void)PhotogrammetryManagerToolbar( )**

This method creates and activates the PhotogrammetryManager Toolbar on main window by using PhotogrammetryManager Toolbar Class.

4.2.1.2 BasicToolbar Class



- **ImageData* image:** It is the instance of the ImageData class.
- **double rotateAngle:** It is used for specifying the rotation angle.
- **double zoomRate:** It is used for specifying the zoom rate.
- **int layoutType:** It is used for the type of display layout.

- **(void) ZoomIn(ImageData* image, double zoomRate)**

This method does a zoom in process on image with a specified zoom rate.

- **(void) ZoomOut(ImageData* image, double zoomRate)**

This method does a zoom out process on image with a specified zoom rate.

- **(void) RotateClockWise(ImageData* image, double rotateAngle)**

This method rotates the image clockwise with a specified angle.

- **(void) RotateCounterClockWise(ImageData* image, double rotateAngle)**

This method rotates the image counter clockwise with a specified angle.

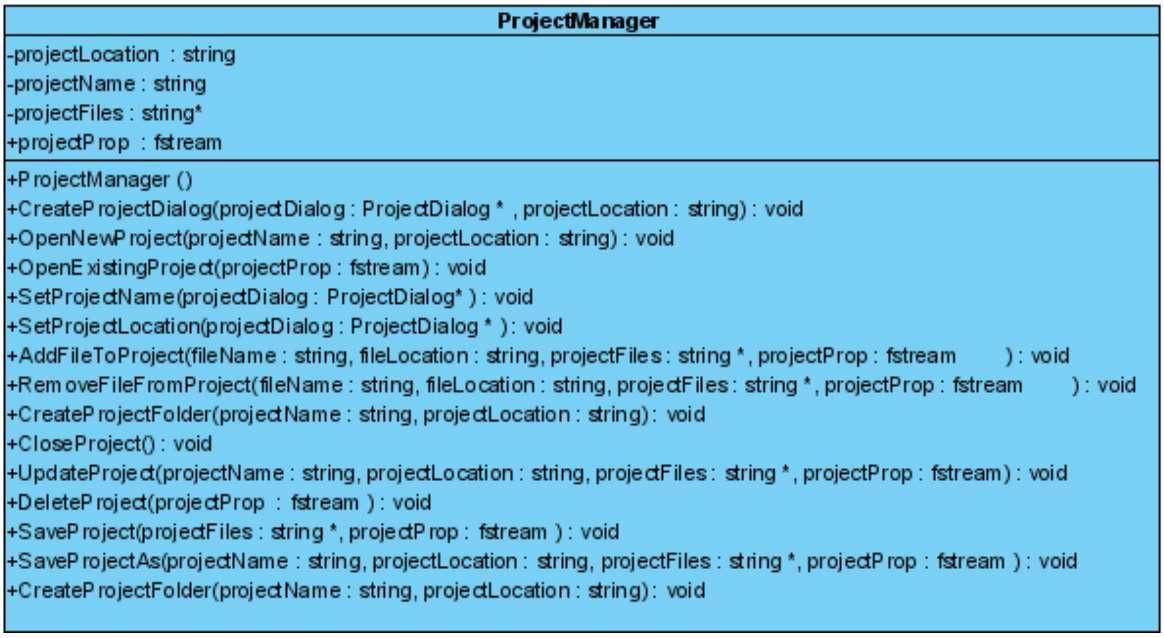
- **(void) DisplayLayout(int layoutType)**

The layout style is changed according to the layoutType parameter.

- **(void) ChangeMode( )**

Change the mode from automatic to manual or manual to automatic for detecting tie points. It uses a global flag value to change the mode.

4.2.1.3 ProjectManager Class



- **string projectLocation :** Stores the location of the project.
- **string projectName:** Stores the name of the project.
- **string *projectFiles:** It stores the locations of the files that belong to the project.
- **fstream projectProp :** When user is creating a new project this file will be created and all changes will be inserted at the time when there is a change. To open an existing project this file will be used.

- **ProjectManager ( )**

This is constructor for the ProjectManager Class. It assigns NULL initially for projectLocation, projectName and projectFiles.

- **CreateProjectDialog(string projectLocation, ProjectDialog * projectDialog)**

This method is triggered by OpenNewProject( ) method. It opens a dialog window on the screen to take project name and project location from the user.

- **(void) OpenNewProject(string projectName, string projectLocation)**

This method creates a new project with the name projectName and with the location projectLocation. The projectName and projectLocation is set from the newProjectDialog window. It also creates projectProp file.

- **(void) OpenExistingProject(fstream projectProp)**

This method opens an existing project by using the file projectProp.

- **(void) SetProjectName(ProjectDialog* projectDialog)**

This method sets the project name as projectName. It is triggered by CreateProjectDialog( ) method.

- **(void) SetProjectLocation(ProjectDialog * projectDialog)**

This method simply sets the location of the project as projectLocation. It is triggered by CreateProjectDialog() method.

- **(void) AddFileToProject(string fileName, string fileLocation,string *projectFiles, fstream projectProp)**

This method adds the file with name fileName and location fileLocation to the projectFiles of the project with the properties given in projectProp.

- **(void) RemoveFileFromProject(string fileName, string fileLocation,string *projectFiles, fstream projectProp)**

This method removes the file with name fileName and location fileLocation from the projectFiles of the project with the properties given in projectProp.

- **(void) CreateProjectFolder(string projectName, string projectLocation)**

This method creates the folder in location projectLocation with the name projectName for storing project files.

- **(void) CloseProject( )**

This method closes the project that is open in main window without exiting the program.

- **(void) UpdateProject(string projectName, string projectLocation, string *projectFiles, fstream projectProp)**

This method updates the project if there are modifications in it and it saves these modifications in projectProp file.

- **(void) DeleteProject(fstream projectProp)**

This method deletes the project with the properties in projectProp.

- **(void) SaveProject(string *projectFiles, fstream projectProp)**

This method saves the project with the given project properties in projectProp file and with the files in projectFiles.

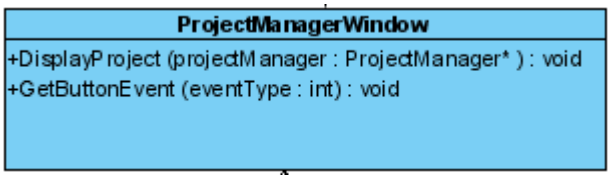
- **(void) SaveProjectAs(string projectName, string projectLocation, string *projectFiles, fstream projectProp)**

This method saves the project with the given project name as projectName, project location as projectLocation, project files as projectFiles and project properties as projectProp.

- **(void) CreateProjectFolder(string projectName, string projectLocation)**

This method creates a folder in specified location set by projectLocation, and necessary files are saved under that folder. It is triggered by OpenNewProject( ) method.

4.2.1.4 ProjectManagerWindow Class



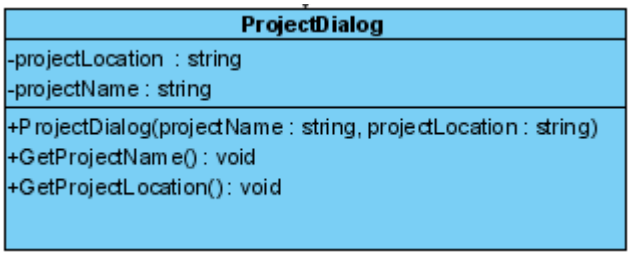
- **(void)DisplayProject (ProjectManager* projectManager)**

This method activates the ProjectManager class constructor and shows the existing project on the ProjectManagerWindow.

- **(void)GetButtonEvent (int eventType)**

This method triggers the methods of the ProjectManager class according to the event type.

4.2.1.5 ProjectDialog Class



- **string projectLocation :** Stores the location of the project.
- **string projectName:** Stores the name of the project.

- **ProjectDialog(string projectName, string projectLocation)**

The constructor of the class.It gets the project name and project location from the user.

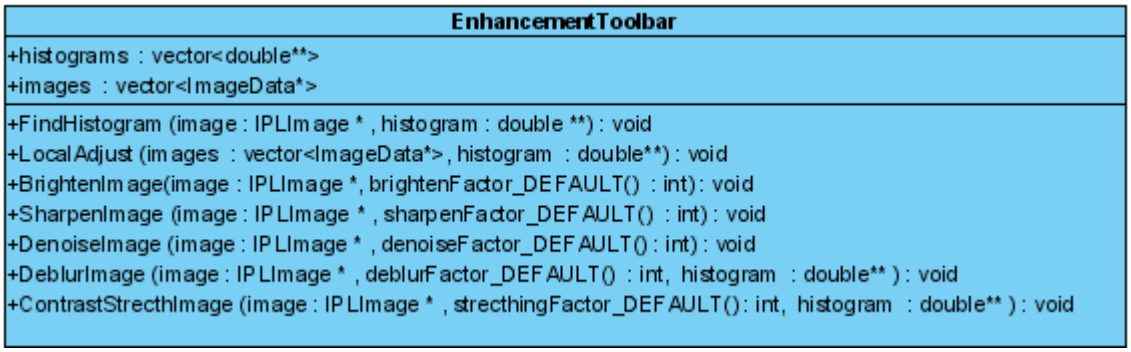
- **GetProjectName()**

This method returns the project name.

- **GetProjectLocation()**

This method returns the project location.

4.2.1.6 EnhancementToolbar Class



- **vector< double**> histograms :** It stores pointers pointing the histograms of the images.
- **vector<ImageData*> images :** It stores the pointers pointing the images.

- **(void) FindHistogram (IPLImage * image, double ** histogram)**

It calculates the histogram of ‘ image ’ and stores it in the ‘ histogram[][] ’.

- **(void) LocalAdjust (vector<ImageData*> images, double** histogram)**

It makes local adjustment by examining the image in terms of the local intensities of the need areas in order to avoid ghosting and blur in the image.

- **(void) BrightenImage (IPLImage * image, int brightenFactor_DEFAULT())**

It brightens the image according to a brightenFactor which is either taken from Enhancement ToolBar on the MainWindow or a defult value.

- **(void) SharpenImage (IPLImage * image, int sharpenFactor_DEFAULT())**

It sharpens the image according to a sharpenFactor which is either taken from Enhancement ToolBar on the MainWindow or a defult value.

- **(void) DenoiseImage (IPLImage * image, int denoiseFactor_DEFAULT())**

It removes noises on the image according to a denoiseFactor which is either taken from Enhancement ToolBar on the MainWindow or a defult value.

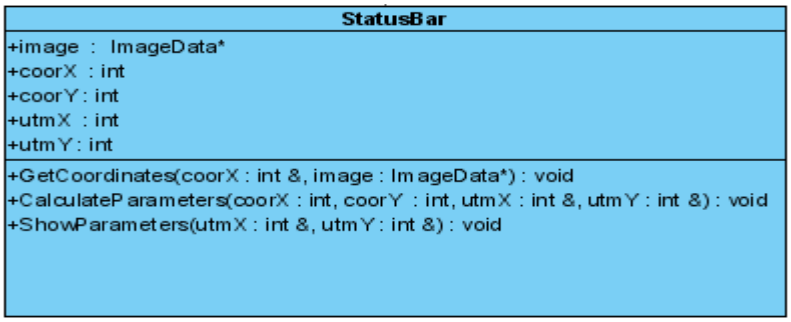
- **(void) DeblurImage (IPLImage * image, int deblurFactor_DEFAULT(),double** histogram)**

It blurs the image according to a deblurFactor which is either taken from Enhancement ToolBar on the MainWindow or a defult value.

- **(void) ContrastStrecthImage (IPLImage * image, int strecthingFactor_DEFAULT(), double** histogram)**

It strech the contrast of image according to a stretchingFactor which is either taken from Enhancement ToolBar on the MainWindow or a defult value.

4.2.1.7 StatusBar Class



- **ImageData* image :** It is the instance of the ImageData class.
- **int coorX :** Stores the x coordinate of the cursor while it is rounding on the image.
- **int coorY:** Stores the x coordinate of the cursor while it is rounding on the image.
- **int utmX :** Stores the latitude values of the UTM coordinates.
- **int utmY:** Stores the longtitude values of the UTM coordinates.

- **(void) GetCoordinates(ImageData* image, int &coorX, int &coorY)**

This method gets the x and y coordinates of the cursor on image and stores them to coorx and coory.

- **(void) CalculateParameters(int coorX, int coorY , int &utmX, int &utmY)**

This method calculates the UTM coordinates according to the coorx and coory values taken from GetCoordinates( ) method and saves them in utmX and utmY.

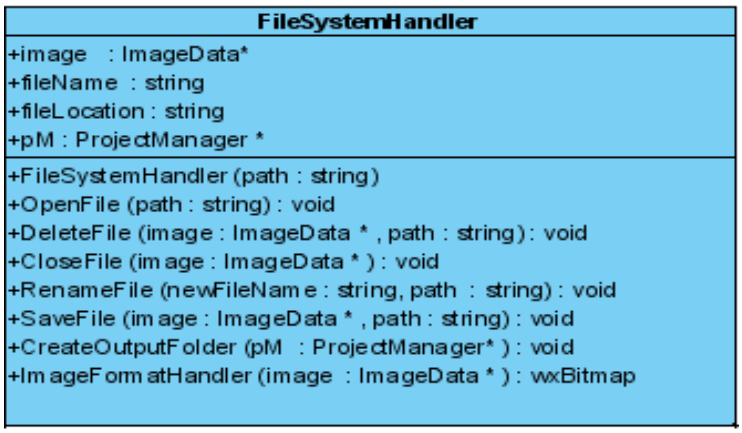
- **(void) ShowParameters(int utmX, int utmY)**

This method writes the calculated parameters utmX and utmY on status bar.

4.2.2 File System Module Class Diagrams

The class diagrams of the File System module are positioned below its module explanation.

4.2.2.1 FileSystemHandler Class



- I**mageData* image :** Points an image whose type is imageData.
- **string fileName :** It is the name of the file object .
- **string fileLocation :** It stores the path of the file. It is taken directly from user.
- **ProjectManager * pM :** Points the project manager object.

- **FileSystemHandler (string path)**

It is the constructor of File System Handler class. It takes the path of the file and extracts the file name and file location. Then, it assigns them to the class members fileName and fileLocation.

- **(void) OpenFile (string path)**

It shows the wxImage on the screen whose path is given by the fileLocation.

- **(void) DeleteFile (ImageData * image, string path)**

It destroys the imageData object and deletes the real image file whose location is taken from path .

- **(void) CloseFile (ImageData * image)**

It destroys only the imageData object.

- **(void) RenameFile (string newFileName, string path)**

It changes the name of the file whose location is taken from path.

- **(void) SaveFile (ImageData * image, string path)**

It saves the image pointed by 'ImageData* image' as a bitmap image to the location specified by path.

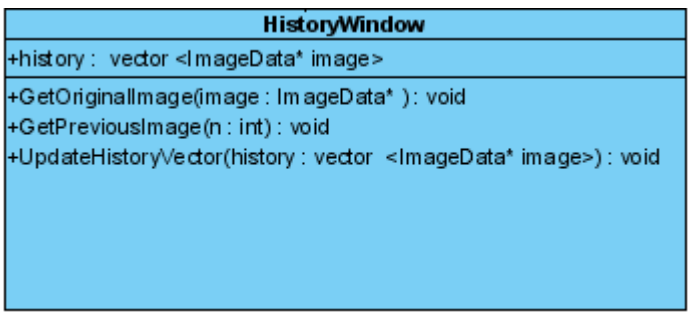
- **(void) CreateOutputFolder (ProjectManager* pM)**

It creates an output folder to store the outputs of the project in the project folder. It assignes the name 'Output' to the folder.

- **(wxBitmap) ImageFormatHandler (ImageData * image)**

It takes the IPL image pointed by 'ImageData * image' and converts it to wxBitmap**.**

4.2.2.2 HistoryWindow Class



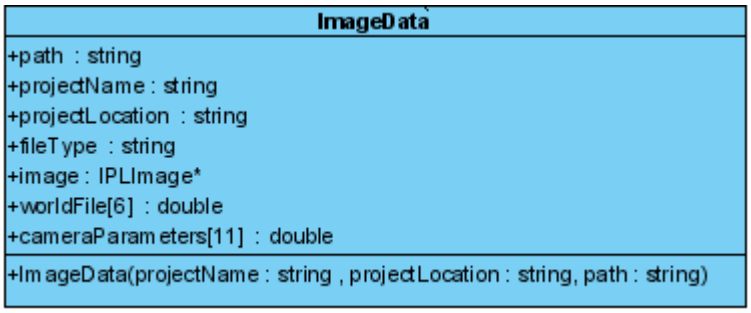
• **vector <ImageData* image> history :** It is the vector of ImageData class instances that contains the previous 3 versions of the image, to undo the processes that are done on images.

• **(void) GetOriginalImage(ImageData* image)**
Resets the processes done on image and loads the original file from project folder.

• **(void) GetPreviousImage(int n):** It undoes the n processes done on image. The integer number n can not be greater than 3.

• **(void) UpdateHistoryVector(vector <ImageData* image> history)**
This method updates the history vector if a process is done on image. New version of the image is pushed to stack and the oldest version is deleted if there are more than 3 versions in history vector.

4.2.2.3 ImageData Class



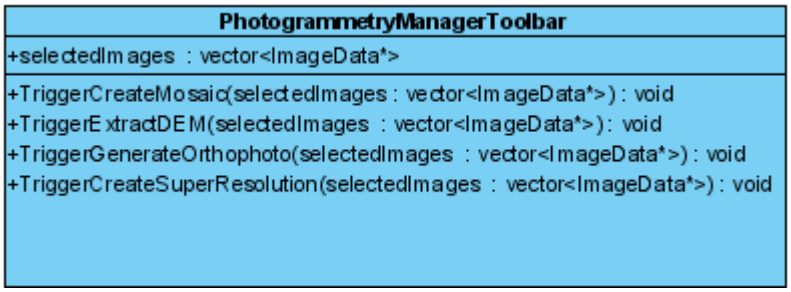
• **string projectLocation :** Stores the location of the project.
• **string projectName:** Stores the name of the project.
• **string path :** Stores the location of the image.
• **string fileType :** Stores the format of the image such as 'jpeg' .
• **IPLImage* image:** Stores the image data as an openCV image type IPLImage.
• **double worldFile[6] :** Stores the world file informations of the image.
• **double cameraParameters[11] :** Stores the intrinsic and extrinsic camera parameters for the image.

• **ImageData(string projectName, string projectLocation,string path)**
This method takes project name, location and the path of the image, and stores the file type in fileType, reads corresponding world file and stores the information in *worldFile if it exists, reads the corresponding camera parameters file and stores it in *cameraParameters, stores the image as IPLImage in image.

4.2.3 Photogrammetry Module Class Diagrams

The class diagrams of the Photogrammetry module are positioned below the photogrammetry module explanation.

4.2.3.1 PhotogrammetryManagerToolbar Class

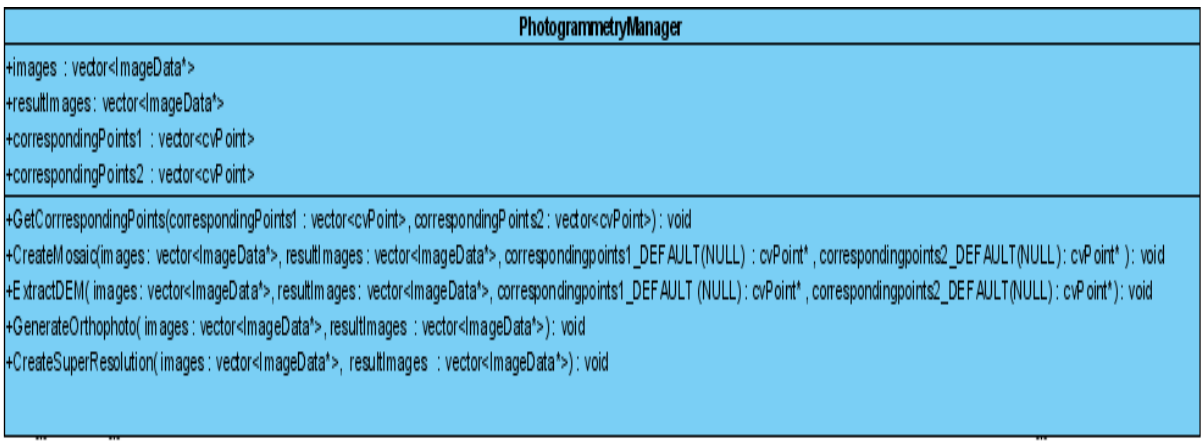


- **vector<ImageData*> selectedImages :** The selected images from the Project. “images” vector which will be created from selected images.

The four methods in the following, works according to user events. The button event taken from Photogrammetry Toolbar on the main window, calls one of these methods :

- **(void) TriggerCreateMosaic(vector<ImageData*> selectedImages)**
- **(void) TriggerExtractDEM(vector<ImageData*> selectedImages)**
- **(void) TriggerGenerateOrthophoto(vector<ImageData*> selectedImages)**
- **(void) TriggerCreateSuperResolution(vector<ImageData*> selectedImages)**

4.2.3.2 PhotogrammetryManager Class



- **vector <ImageData*> images :** It is the instance of the ImageData class. This vector contains input images which will be parameters to the photogrammetry subclasses.
- **vector <ImageData*> resultImages:** It is again the instance of the ImageData class. This vector is used for storing output images comes from the PhotogrammetryManager Class methods.
- **vector<cvPoint> correspondingPoints1 :** The manually selected corresponding points of the first image. We use openCV library for corresponding point selection.

• **vector<cvPoint> correspondingPoints2 :** The manually selected corresponding points of the second image. This points are matched-paired with the correspondingPointsFirst, e.g the first point in the CorrespondingPointsFirst is the matching of the first point in the CorrespondingPointsSecond.

• **(void) GetCorrrespondingPoints(vector<cvPoint> correspondingPoints1, vector<cvPoint> correspondingPoints2)**

This method take the positions of the left mouse clicked events from the screen and store these point values to the parameter vectors respectively.

• **(void) CreateMosaic(vector <ImageData*> images, vector <ImageData*> resultImages, cvPoint* correspondingpoints1_DEFAULT (NULL), cvPoint* correspondingpoints2_DEFAULT(NULL))**

By using these parameters, this function uses Mosaic Class functions to mosaic images in the "images" vector. At initial phase it uses first 2 images on the vector and creates a Mosaic Class object. Then output of this operation and next image on the "images" vector will be input of next mosaic operation. This function simply assigns the output mosaiced image to the "resultImages" vector. If Photolab is in manual mode and if user selects corresponding points in two images by hand, DetectCorner() and DetectMatchingPoints() steps will be skipped. As a result of this method, final version of mosaiced images will be shown in the screen and will be saved as another ImageData object with 'MOSAICED_IMAGE' flag .

• **(void) ExtractDEM(vector <ImageData*> images, vector <ImageData*> resultImages, cvPoint* correspondingpoints1_DEFAULT (NULL), cvPoint* correspondingpoints2_DEFAULT(NULL), int displayType)**

This method uses DEM Class methods to calculate the digital elevation model. From input vector "images", we use first 2 images and calculate Digital Elevation Model by the help of camera parameters. We again use corresponding points taken from user or detected automatically. "displayType" is an integer flag to set the output show type, such as contour lines and 3D relief map. As a result of this method, final version of DEM will be shown in the screen according to display type and saved in 'imageName.dem' file in the Project directory. In addition, the process type of the output image is changed with 'DEM_EXTRACTED_IMAGE' flag.

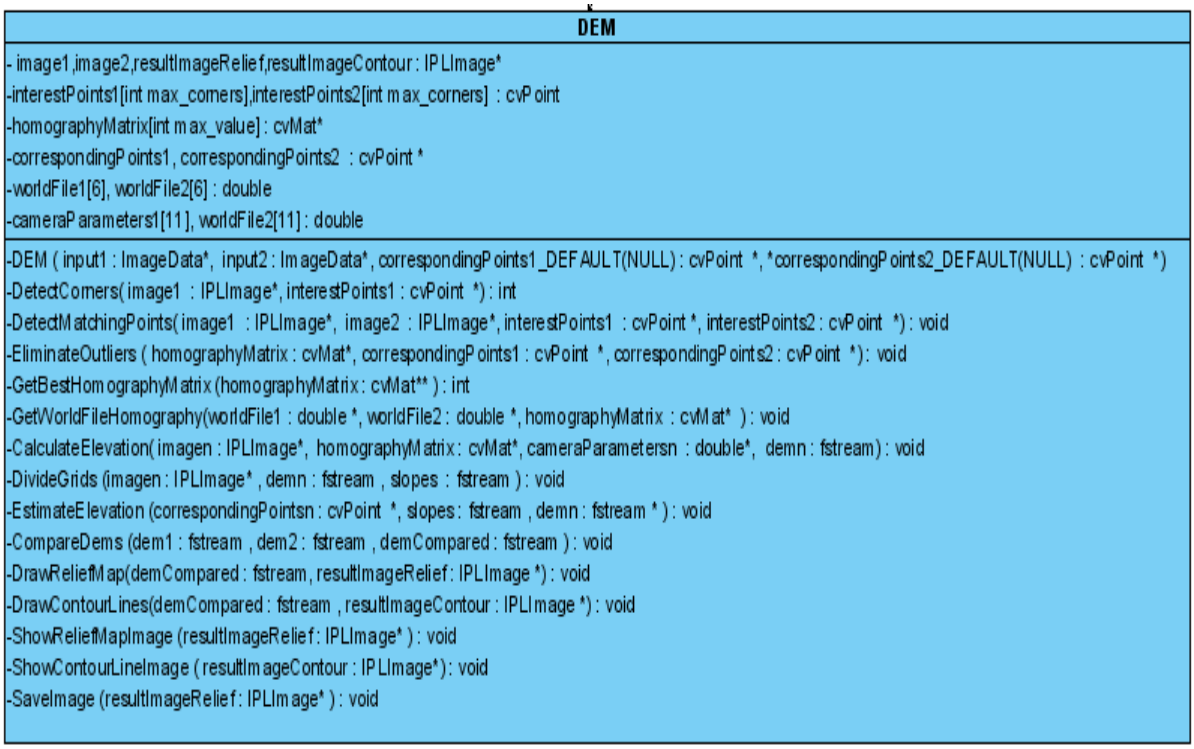
• **(void) GenerateOrthophoto(vector <ImageData*> images, vector <ImageData*> resultImages)**

Orthophoto generation uses Orthophoto Class methods. To start orthophoto generation, each image needs to have its "imageName.dem" file. If this method can not find the ".dem" file in the Project directory, it forces user to extract DEM of images or it uses ExtractDEM() method with default parameters. Final result is shown in the screen and saved as ImageData object with 'GEORECTIFIED_IMAGE' flag.

• **(void) CreateSuperResolution(vector <ImageData*> images, vector <ImageData*> resultImages)**

This method uses SuperResolution Class. The first image in the "images" vector is the original and superresolution is shown on this one. Super Resolution steps similar to Mosaic methods, however the difference is we only improve the first original image. We use all overlapped layers as parameters to calculate final resolution of original image. Final result is shown on the screen and saved as ImageData object with 'SUPER_RESOLVED_IMAGE' flag.

4.2.3.3 DEM



• **IPLImage* image1,image2,resultImageRelief,resultImageContour:** The first two members are DEM inputs. The “resultImageRelief”and “resultImageContour” are the output images.

• **cvPoint interestPoints1[int max_corners],interestPoints2[int max_corners] :** It stores the detected corner values. This values are optimized by DEM methods.

• **cvMat* homographyMatrix[int max_value] :** It contains the calculated homography matrixes of the images.

• **cvPoint * correspondingPoints1, correspondingPoints2 :** It stores the matching points of the images. If these values are not NULL,assigned by the user manually, we use them directly.

• **double worldFile1[6], worldFile2[6]:** It stores the world file parameters of the two inputs which are the inputs of the DEM() method. As default, they are NULL since images may not have world files.

• **double cameraParameters1[11], worldFile2[11]:** It stores the camera parameters of the two inputs which are the inputs of the DEM() method.

• **DEM (ImageData* input1,ImageData* input2,**
cvPoint *correspondingPoints1_DEFAULT(NULL),
cvPoint *correspondingPoints2_DEFAULT(NULL))

Constructor of the DEM Class. This constructor assigns the IPLImage of input1 and input2 to the Mosaic object members image1 and image2 .In addition,it assigns world files of input1 and input2

to the Mosaic object members worldFile1 and worldFile2. It assigns camera parameters of input1 and input2 to the DEM object members cameraParameters1 and cameraParameters2.

- **(int) DetectCorners(IPLImage* image1 , cvPoint *interestPoints1)**

By using Harris Corner Detection, this finds the corners of the first image and stores them to the interestPoints1 array. It returns the number of corners.

- **(void) DetectMatchingPoints(IPLImage* image1 , IPLImage* image2 ,cvPoint *interestPoints1, cvPoint *interestPoints2)**

cvGoodFeaturesToTrack() method of openCV library finds the interest points of the second image. While storing these values, it compares the second image with the first image interest points and updates these arrays according to the matches between them.

- **(void) EliminateOutliers (cvMat* homographyMatrix, cvPoint *correspondingPoints1, cvPoint *correspondingPoints2)**

According to the RANSAC algorithm, we calculate homography matrix between two images. In addition RANSAC eliminate the outliers by using the interest points of the activated DEM object.

- **(int) GetBestHomographyMatrix (cvMat** homographyMatrix)**

It returns the location of best homography matrix among the given calculated homography matrixes array.

- **(void) GetWorldFileHomography(double *worldFile1, double *worldFile2, cvMat* homographyMatrix)**

If the world files of the two images are given, not NULL, we can directly calculate the homography matrix and then we register images with this homography matrix.

- **(void) CalculateElevation(IPLImage* imagen, cvMat* homographyMatrix, double *cameraParametersn,fstream demn)**

This method is called for both image1 and image2. According to the homography matrix and by using the given camera parameters, it calculates the elevations of the images and stores them into the dem1 and dem2 files.

- **(void) DivideGrids (IPLImage* imagen,fstream demn,fstream slopes)**

This method is again called for both image1 and image2. It uses the calculated elevations of the two images for the dem files and calculates the slopes between two points whose elevations are calculated before. We again store this slope values into the file.

- **(void) EstimateElevation (cvPoint *correspondingPointsn,fstream slopes, fstream * demn)**

This function finds the elevation of the corresponding points of the two images by using the elevation and slope information of nearest neighbours of these points. Itadds the estimated elevations of the corresponding points to the dem files of images.

- **(void) CompareDems (fstream dem1, fstream dem2,fstream demCompared)**

Since we calculate the elevations of the matching points into two images before, by comparing them we find their exact elevations.

- **(void) DrawReliefMap(fstream demCompared, IPLImage *resultImageRelief)**

According to the compared elevations this method generates the relief map image.

- **(void) DrawContourLines(fstream demCompared, IPLImage *resultImageContour)**

According to the compared elevations this method generates the image with contour lines.

- **(void) ShowReliefMapImage (IPLImage* resultImageRelief)**

This method only displays the resultImageRelief on the screen. It is called by the DrawReliefMap method.

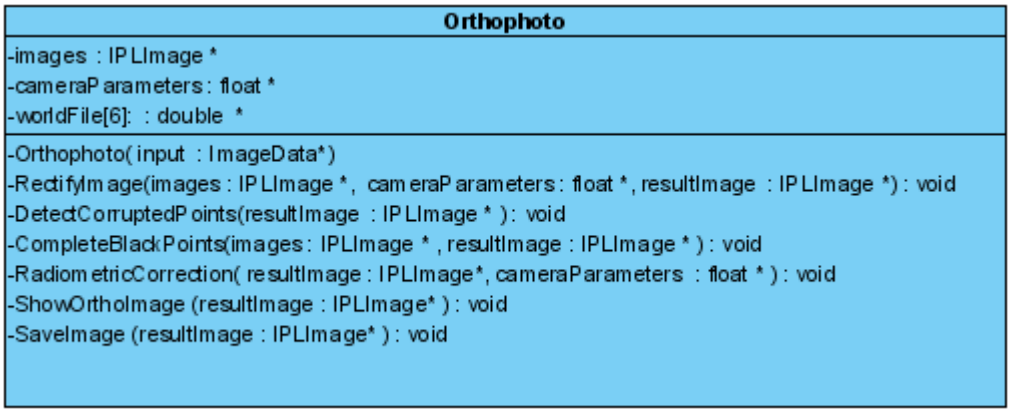
- **(void) ShowContourLineImage (IPLImage* resultImageContour)**

This method only displays the resultImageRelief on the screen. It is called by the DrawContourLines method.

- **(void) SaveImage (IPLImage* resultImageRelief)**

This method only saves the resultImage with the flag 'DEM_EXTRACTED_IMAGE' both in Project.phl file and ImageData object .

4.2.3.4 Orthophoto



- **IPLImage *images :** This array stores the input IPL images for the Orthophoto methods.
- **float *cameraParameters:** This array stores the camera parameters coming from input imageData objects.
- **double *worldFile[6]:** It is a pointer array pointing the world file of each input image.

- **Orthophoto(ImageData* input)**

Constructor. It assignes the IPL images of 'input' to the Orthophoto object member 'images'. In addition it assigns world file and camera parameters coming from input object to members of Orthophoto object.

- **(void) RectifyImage(IPLImage *images, float * cameraParameters, IPLImage *resultImage)**

This method rectifies the image to topographic view according to the camera parameters. The outer parameters determine the perspective view of the camera. 'resultImage' stores the output of rectification process.

- **(void) DetectCorruptedPoints(IPLImage * resultImage)**

It detects the areas where rectification process has destroyed. It outputs the resultImage with detected areas.

- **(void) CompleteBlackPoints(IPLImage * images, IPLImage *resultImage)**

It completes the areas which are outputs of DetectCorruptedPoints method by using Mosaic class methods. It finds all of the necessary images which have unfilled areas and mosaic them with the resultImage. The output of this method is again resultImage.

- **(void) RadiometricCorrection(IPLImage* resultImage, float * cameraParameters)**

It simply enhances the resultImage using Enhancement class methods. Camera's sensor failures and illumination differences are corrected in this method and outputs the resultImage.

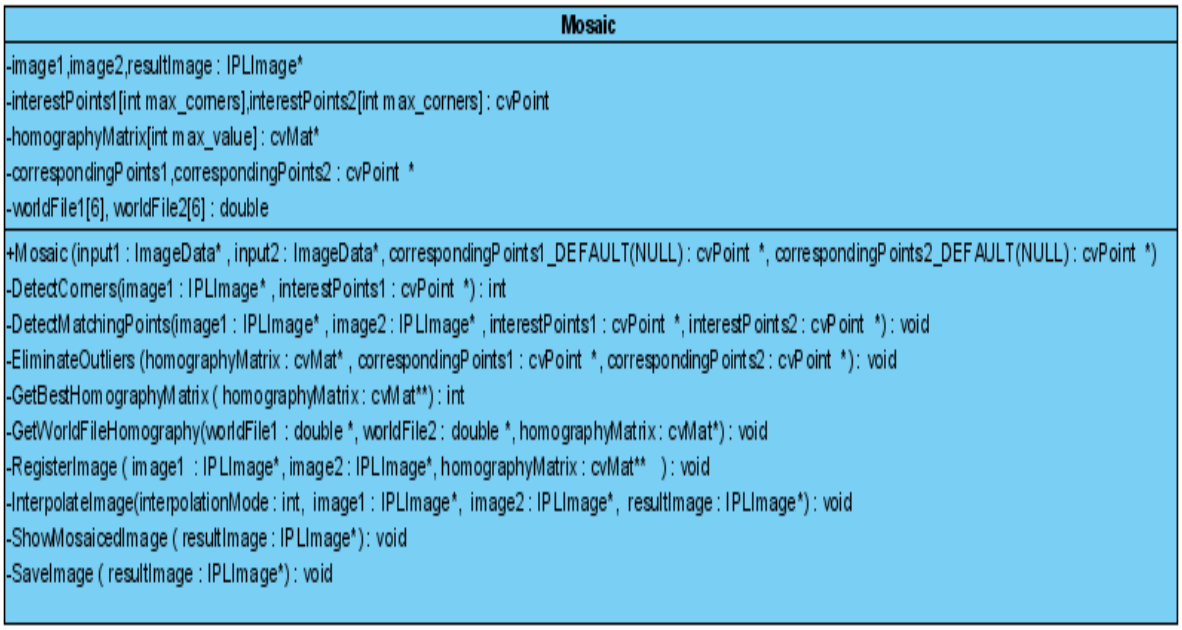
- **(void) ShowOrthoImage (IPLImage* resultImage)**

This method only displays the resultImage on the screen.

- **(void) SaveImage (IPLImage* resultImage)**

This method only saves the resultImage with the flag 'ORTHOPHOTO_IMAGE' both in Project .phl file and ImageData object.

4.2.3.5 Mosaic



- **IPLImage* image1,image2,resultImage:** The first two members are mosaic inputs. The "resultImage" is the mosaiced image.
- **cvPoint interestPoints1[int max_corners],interestPoints2[int max_corners] :** It stores the detected corner values. This values are optimized by methods.
- **cvMat* homographyMatrix[int max_value] :** It contains the calculated homography matrixes of the images.
- **cvPoint *correspondingPoints1, *correspondingPoints2 :** It stores the matching points of the images. If these values are not NULL,assigned by the user manually, we use them directly.
- **double worldFile1[6], worldFile2[6]:** It stores the world file parameters of the two inputs which are the inputs of the Mosaic() method. As default, they are NULL since images may not have world files.

- **Mosaic (ImageData* input1,ImageData* input2,**
  cvPoint *correspondingPoints1_DEFAULT(NULL),
  cvPoint *correspondingPoints2_DEFAULT(NULL))

Constructor. It assigns the IPLImage of input1 and input2 to the Mosaic object members image1 and image2 .In addition,it assigns world files of input1 and input2 to the Mosaic object members worldFile1 and worldFile2.

- **(int) DetectCorners(IPLImage* image1 , cvPoint *interestPoints1)**

By using Harris Corner Detection, this finds the corners of the first image and stores them to the interestPoints1 array. It returns the number of corners.

- **(void) DetectMatchingPoints(IPLImage* image1 , IPLImage* image2 ,cvPoint**

***interestPoints1, cvPoint *interestPoints2)**

cvGoodFeaturesToTrack() method of openCV library finds the interest points of the second image. While storing these values, it compares the second image with the first image interest points and updates these arrays according to the matches between them.

- **(void) EliminateOutliers (cvMat* homographyMatrix, cvPoint *correspondingPoints1, cvPoint *correspondingPoints2)**

According to the RANSAC algorithm, we calculate homography matrix between two images. In addition RANSAC eliminate the outliers by using the interest points of the activated Mosaic object.

- **(int) GetBestHomographyMatrix (cvMat** homographyMatrix)**

It returns the location of best homography matrix among the given calculated homography matrixes.

- **(void) GetWorldFileHomography(double *worldFile1, double *worldFile2, cvMat* homographyMatrix)**

If the world files of the two images are given, not NULL, we can directly calculate the homography matrix and then we register images with this homography matrix.

- **(void) RegisterImage (IPLImage* image1 , IPLImage* image2 ,cvMat** homographyMatrix)**

It applies transformations to the images by using the best homography matrix and so image1 and image2 are ready to be displayed as mosaic.

- **(void) InterpolateImage(int interpolationMode, IPLImage* image1, IPLImage* image2, IPLImage* resultImage)**

It contains the bilinear and bicubic interpolation algorithms. The algorithm selection is specified by interpolationMode. Also it combines the two transformed images and forms the output mosaicked image as a resultImage . Interpolation algorithms are for displaying resultImage properly.

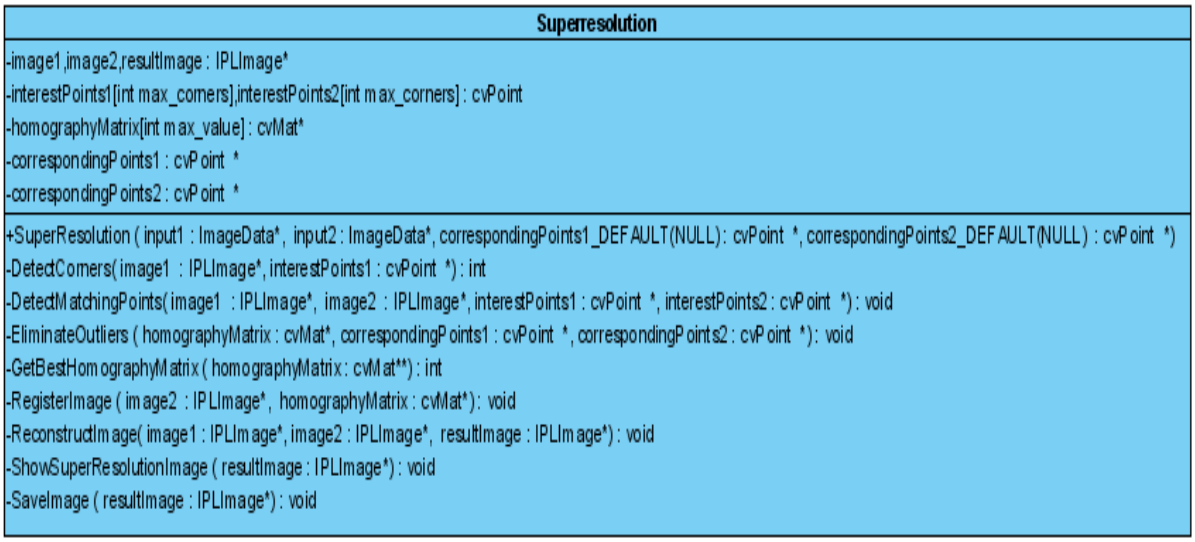
- **(void) ShowMosaicedImage (IPLImage* resultImage)**

This method only displays the resultImage on the screen. It is called by the InterpolateImage method.

- **(void) SaveImage (IPLImage* resultImage)**

This method only saves the resultImage with the flag ‘MOSAICED_IMAGE’ both in Project.phl file and ImageData object .

4.2.3.6 Superresolution



- **IPLImage* image1,image2,resultImage:** The first two members are SuperResolution inputs. The "resultImage" is the high resoluted image.
- **cvPoint interestPoints1[int max_corners],interestPoints2[int max_corners] :** It stores the detected corner values. This values are optimized by SuperResolution methods.
- **cvMat* homographyMatrix[int max_value] :** It contains the calculated homography matrixes of the images.
- **cvPoint *correspondingPoints1, *correspondingPoints2 :** It stores the matching points of the images. If these values are not NULL,assigned by the user manually, we use them directly.

- **SuperResolution (ImageData* input1,ImageData* input2,**
  cvPoint *correspondingPoints1_DEFAULT(NULL),
  cvPoint *correspondingPoints2_DEFAULT(NULL))

Constructor. It assigns the IPLImage of input1 and input2 to the SuperResolution object members image1 and image2 .

- **(int) DetectCorners(IPLImage* image1 , cvPoint *interestPoints1)**

By using Harris Corner Detection, this finds the corners of the first image and stores them to the interestPoints1 array. It returns the number of corners.

- **(void) DetectMatchingPoints(IPLImage* image1 , IPLImage* image2 ,cvPoint *interestPoints1, cvPoint *interestPoints2)**

cvGoodFeaturesToTrack() method of openCV library finds the interest points of the second image. While storing these values, it compares the second image with the first image interest points and updates these arrays according to the matches between them.

- **(void) EliminateOutliers (cvMat* homographyMatrix, cvPoint *correspondingPoints1, cvPoint *correspondingPoints2)**

According to the RANSAC algorithm, we calculate homography matrix between two images. In addition RANSAC eliminate the outliers by using the interest points.

- **(int) GetBestHomographyMatrix (cvMat** homographyMatrix)**

It returns the location of best homography matrix among the given calculated homography matrixes.

- **(void) RegisterImage (IPLImage* image2 ,cvMat* homographyMatrix)**

It applies transformations to the second image by using the best homography matrix.

- **(void)ReconstructImage(IPLImage* image1,IPLImage* image2, IPLImage* resultImage)**

Method takes first image and warped second image then buils resulting image by combining and improving resolution in parts of image1, where it intersects with image2.

- **(void) ShowSuperResolutionImage (IPLImage* resultImage)**

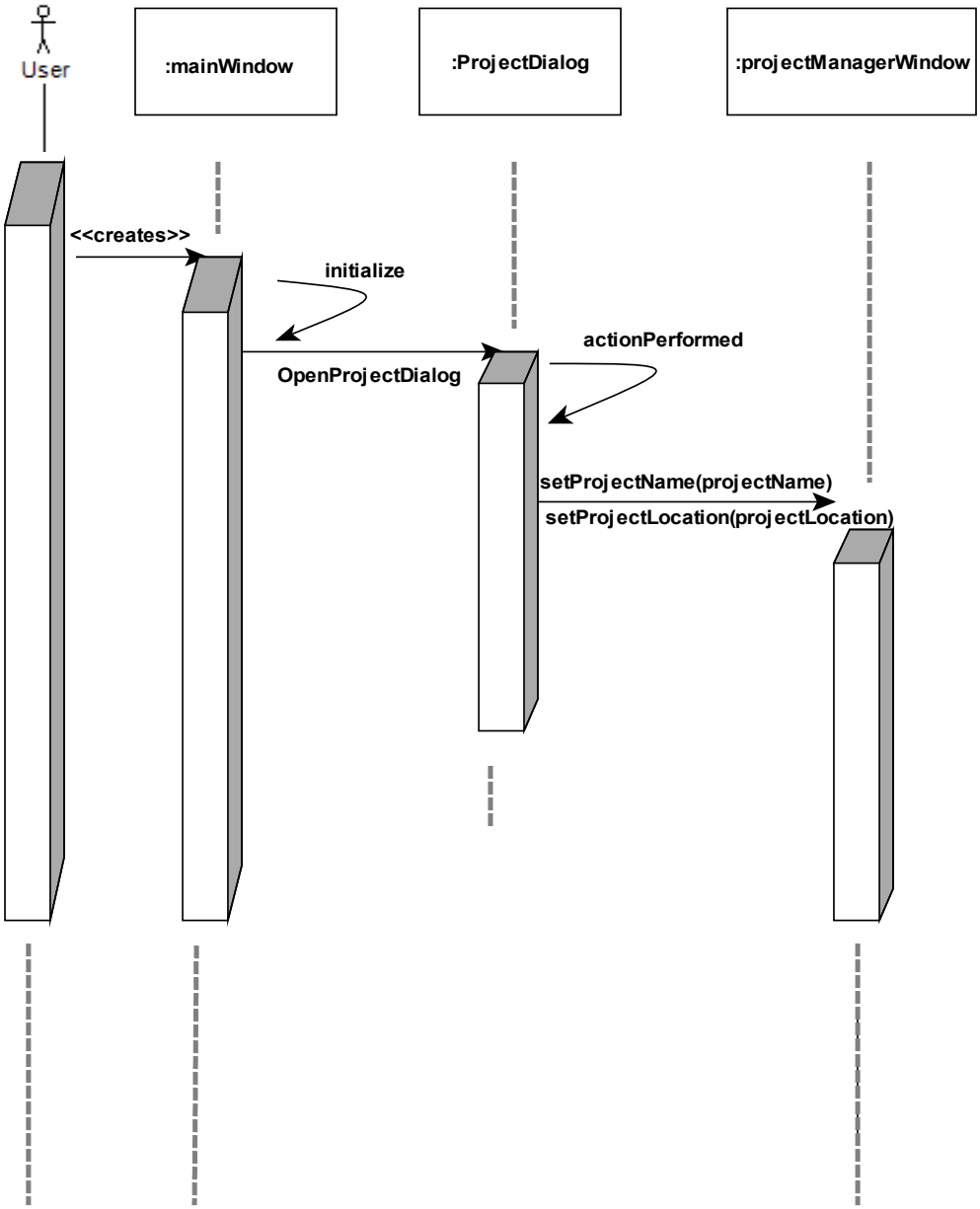
This method only displays the resultImage on the screen. It is called by the ReconstructImage method of the SuperResolution class.

- **(void) SaveImage (IPLImage* resultImage)**

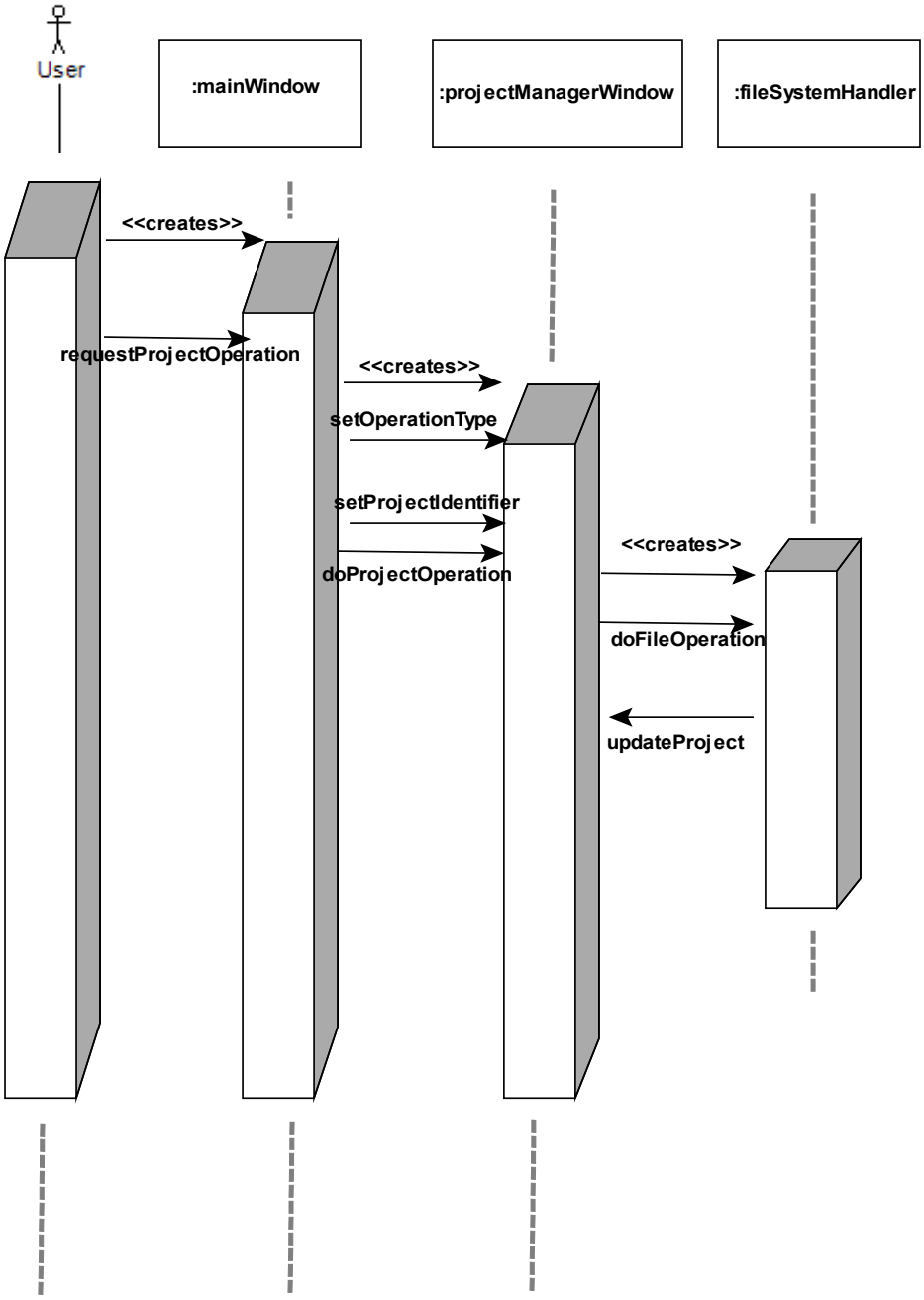
This method only saves the resultImage with the flag 'SUPER_RESOLVED_IMAGE' both in Project.phl file and ImageData object .

4.3 Sequence Diagrams

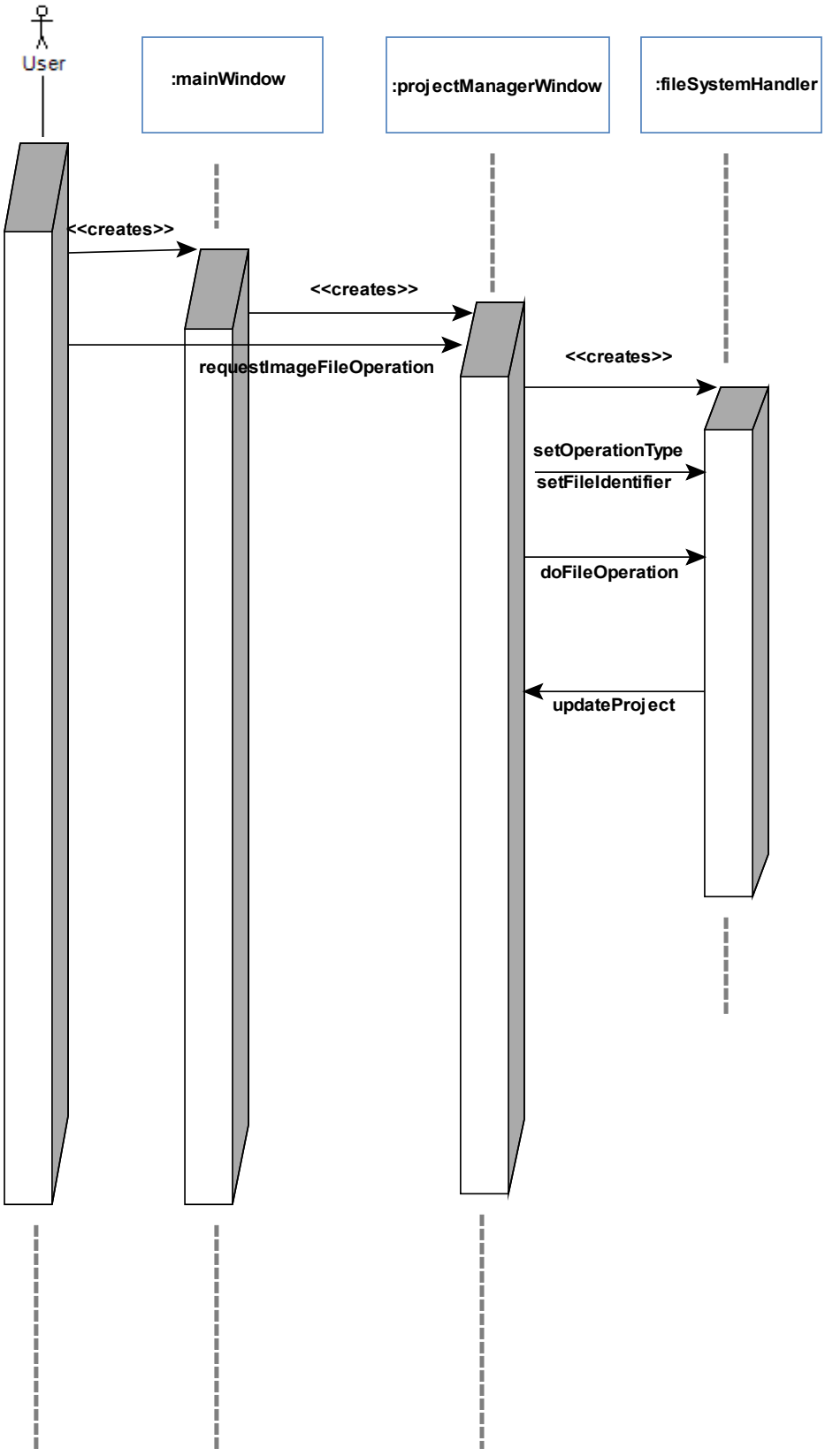
4.3.1 Create New Project



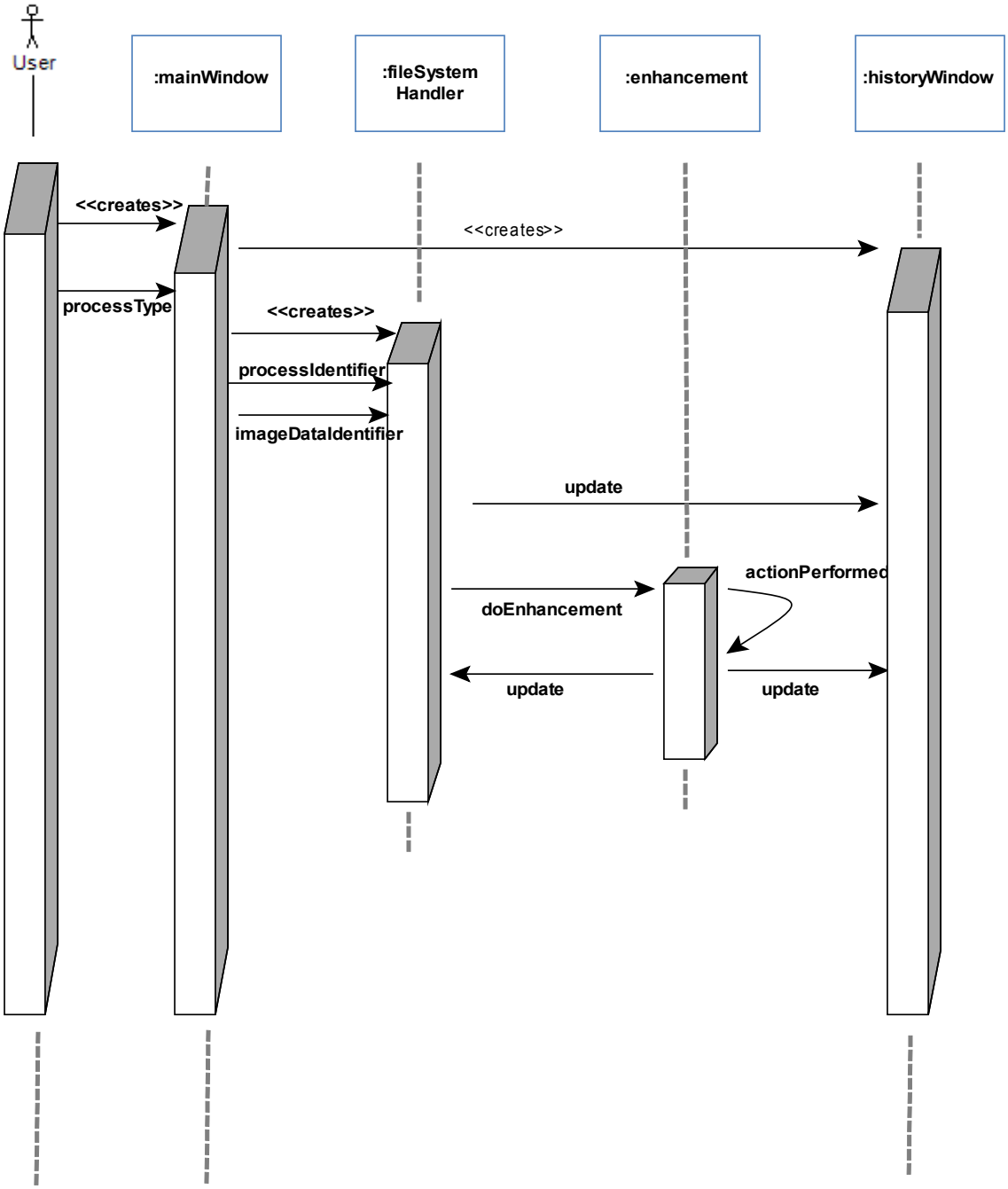
4.3.2 Project Operations



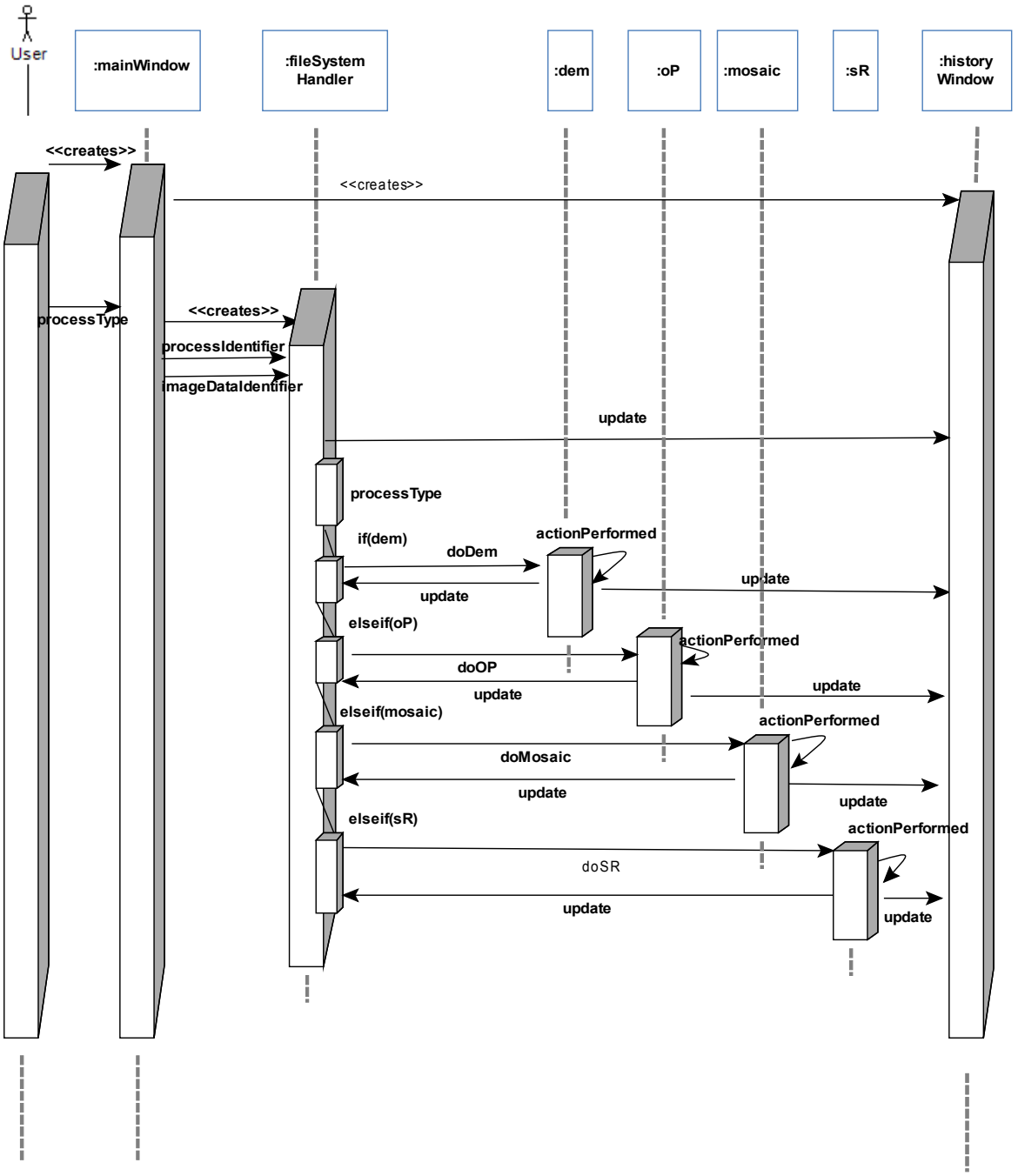
4.3.3 Image File Operations



4.3.4 Enhancement



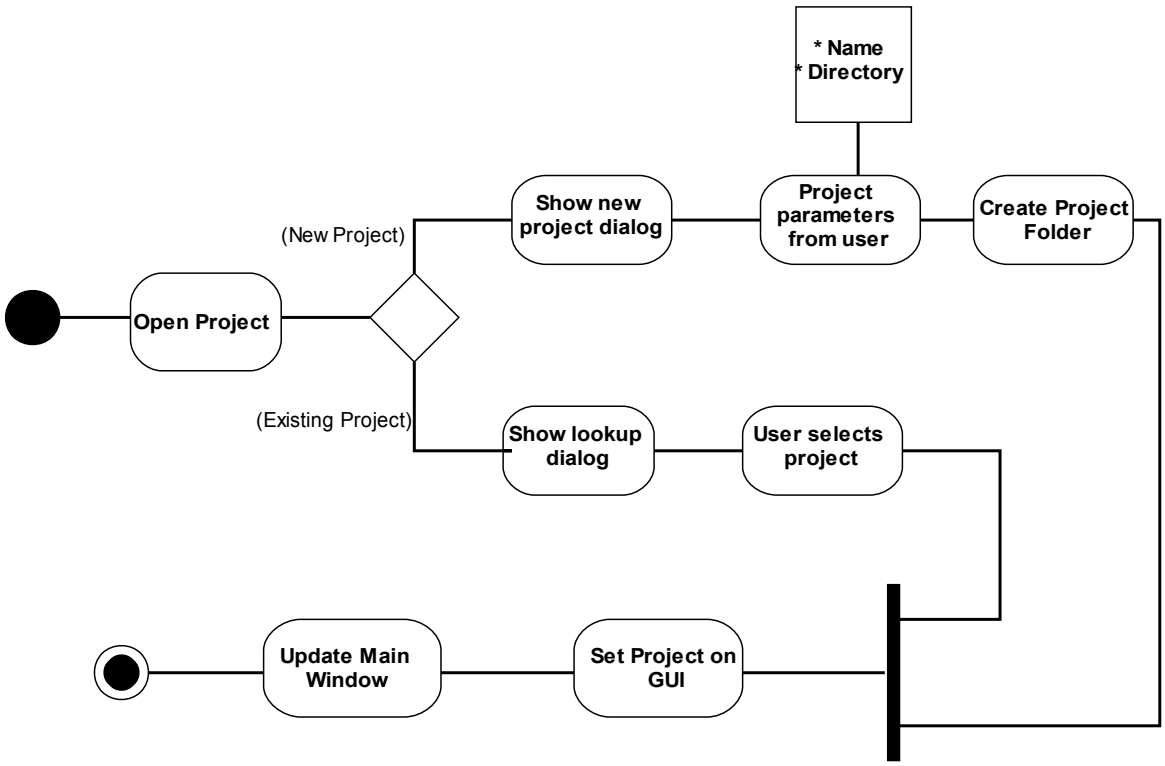
4.3.5 Photogrammetry



4.4 ACTIVITY DIAGRAMS

4.4.1 Open Project

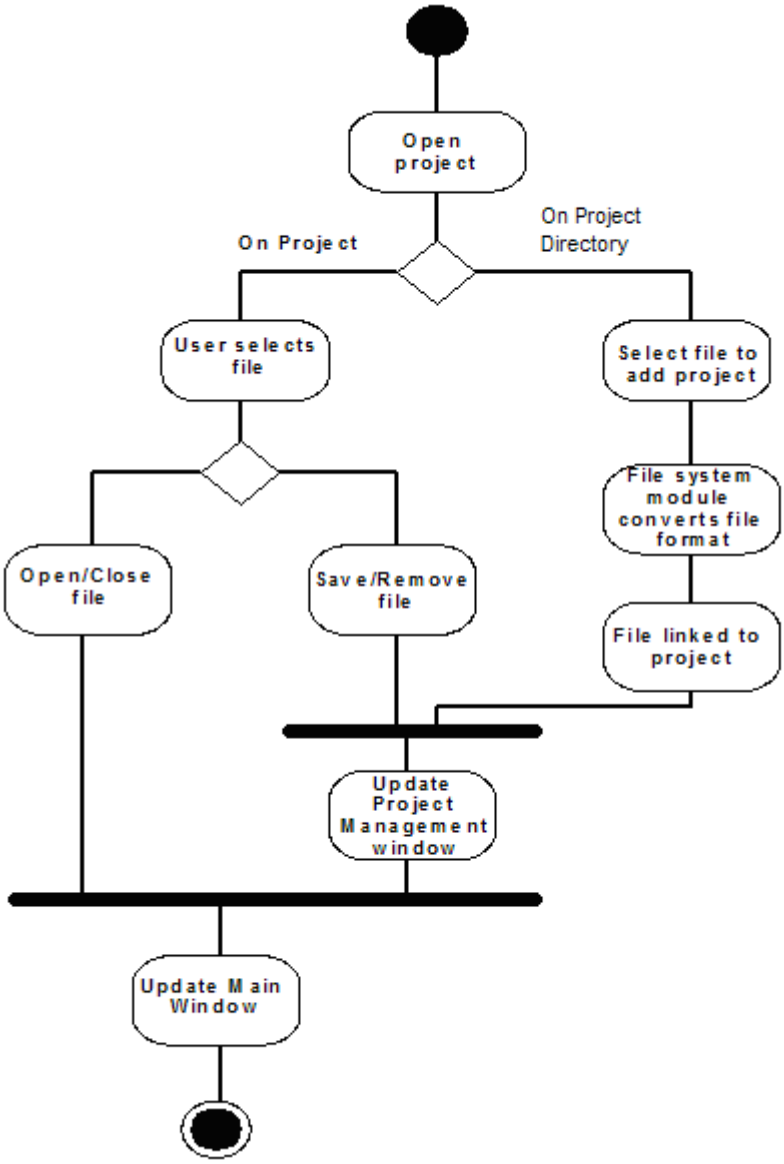
When the user initiates the PHOTOLAB program, most likely the user wants to open a project. The user will either open an existing project or create a new project. If the user wants to create a new project, PHOTOLAB shows new project dialog . The user enters a project name and determines the file location to this dialog. After getting the project parameters from the user,a project folder is created by PHOTOLAB. The user may want to open an existing project. In this case, PHOTOLAB shows the user lookup dialog so the user can select the project that he/she wants to open. In both cases, PHOTOLAB opens the project on the main window.



4.4.2 File Operations

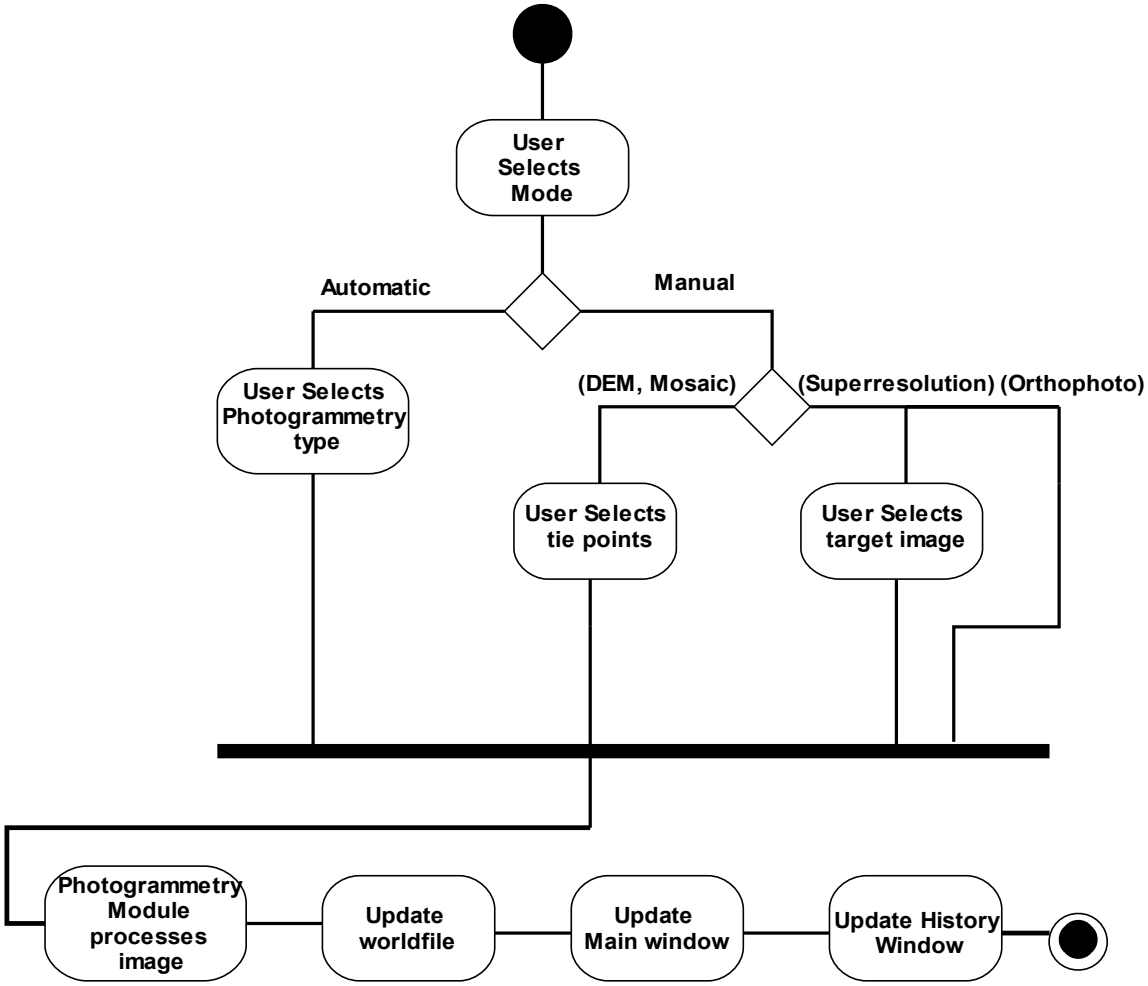
File operations start with opening a Project. User may want two different spaces depending on file operations. On Project, first user selects a file to operate. After selection, user decides on file operation type; open, close, save and remove. Since the directory tree structure will be modified , save and remove operations needs updates on Project Management Window. On Project directory user may want select files to add Project. The file format is controlled by file system module and if it

needs a conversion same module converts it into specified format. Then a link is created between file and Project. This operation also needs Project Management Window update. At the end, main window is updated and file operations finish.



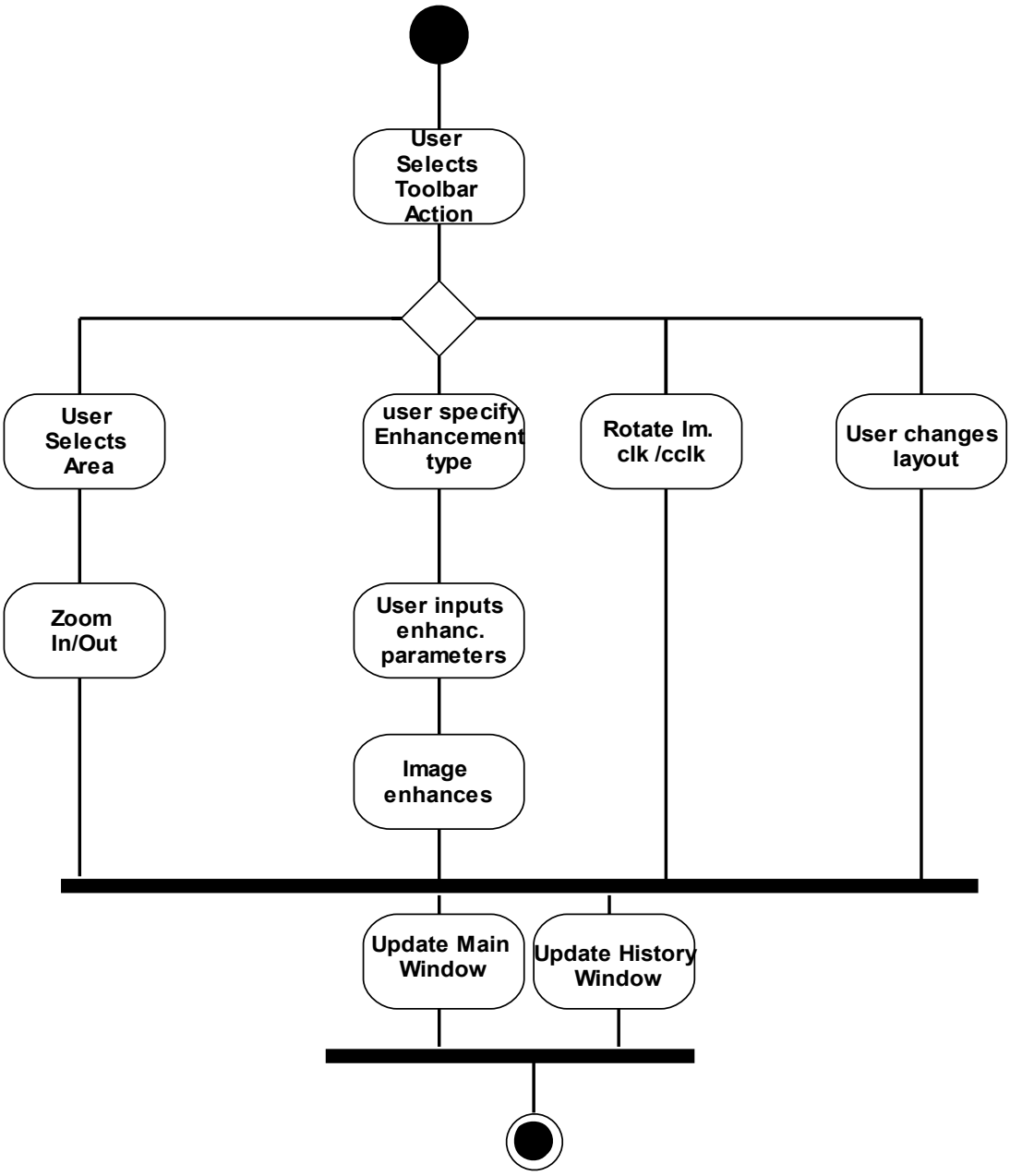
4.4.3 Photogrammetry Operations

The user selects the mode from the menu bar. PHOTOLAB gives two options to the user.User can either select the automatic mode or the manual mode. The default mode provided by PHOTOLAB is automatic mode. If the user selects manual mode, he/she either selects the tie points or the target image according to the photogrammetry process type to be processed. The user selects tie point if the process is DEM or Mosaic and selects target image if the process is SuperResolution and nothing is done if the process is OrthoPhoto. In the automatic mode PHOTOLAB wants user only to choose process type. Then PHOTOLAB processes on the images according to the selected process type.After the photogrammetry process, PHOTOLAB updates the world file,main window and history window.



4.4.4 Toolbar Actions

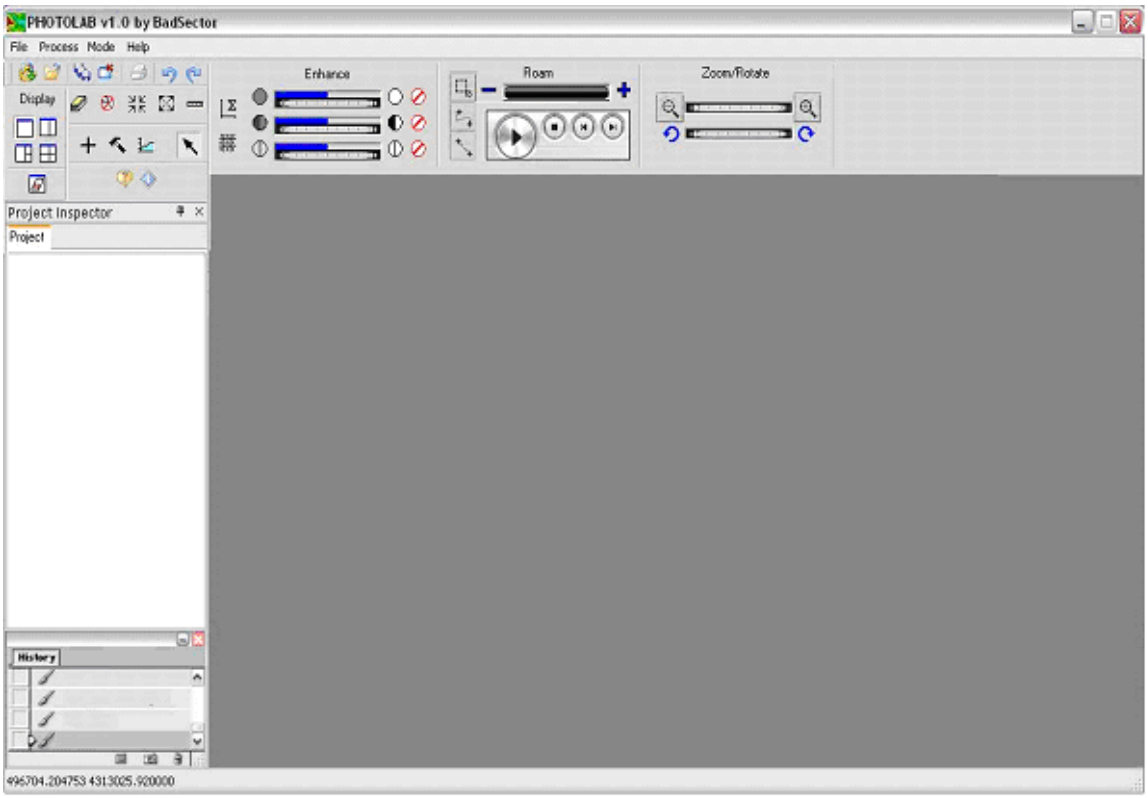
Toolbar actions provide making some operations on images. This toolbar is very crucial part of our IDE. It provides image enhancing, zooming, rotating options. Also, the layout can be changed from toolbar. The user first selects the operation to be done. After operating the process chosen by toolbar, the main window and the history window is updated.


User Selects Toolbar Action
User Selects Area
user specify Enhancement type
Rotate Im. clk /cclk
User changes layout
Zoom In/Out
User inputs enhanc. parameters
Image enhances
Update Main Window
Update History Window

5 GUI – GRAPHICAL USER INTERFACE

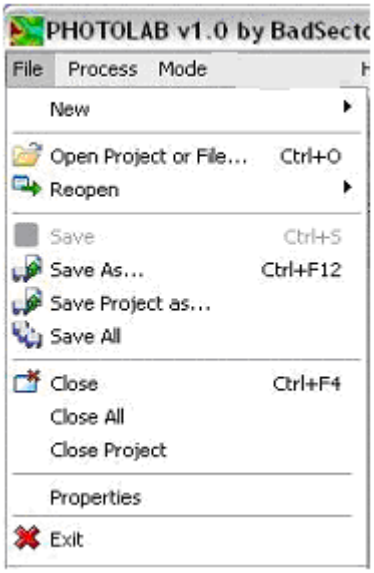
As a group we tried to design our GUI user-friendly to make the interaction with user easy and practical. In our design, all of our modules interact among each other via GUI, so this makes GUI design module, the most important part of our project.

The general graphical user interface of PHOTOLAB looks like as shown below.

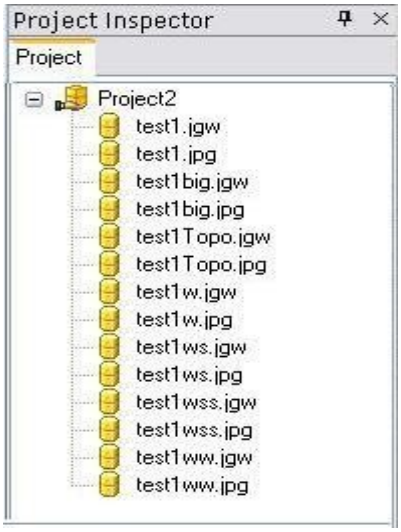


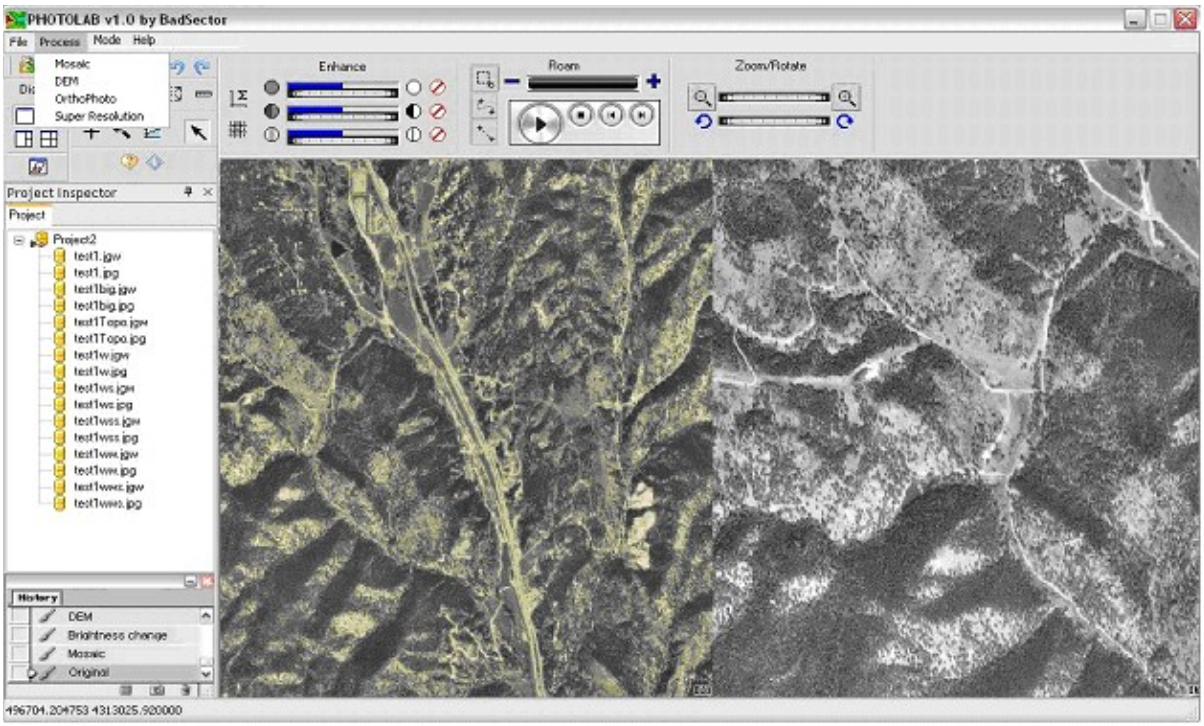
Upon starting, project inspector window is shown at left middle side of the main window. Project inspector window lists the files of the current opened project and it makes file operations easier. Just below it there is the history window. In history window there is a list of last three processes and the original version for the selected image. Besides undo and redo operations, user can turn back to the images before photogrammetry operations. In addition to project inspector and history window there is the main area for image display. At first the display area is not divided to sub display areas.

At the top of the main window, there is a menu bar which contains ‘File’, ‘Process’, ‘Mode’ and ‘Help’ menu items. To start working with PHOTOLAB, you have to open a file or project by using ‘File’ menu. The user can either open a single file or, open or create a project. You save the project and files from this menu item and you also do the closing job from here.

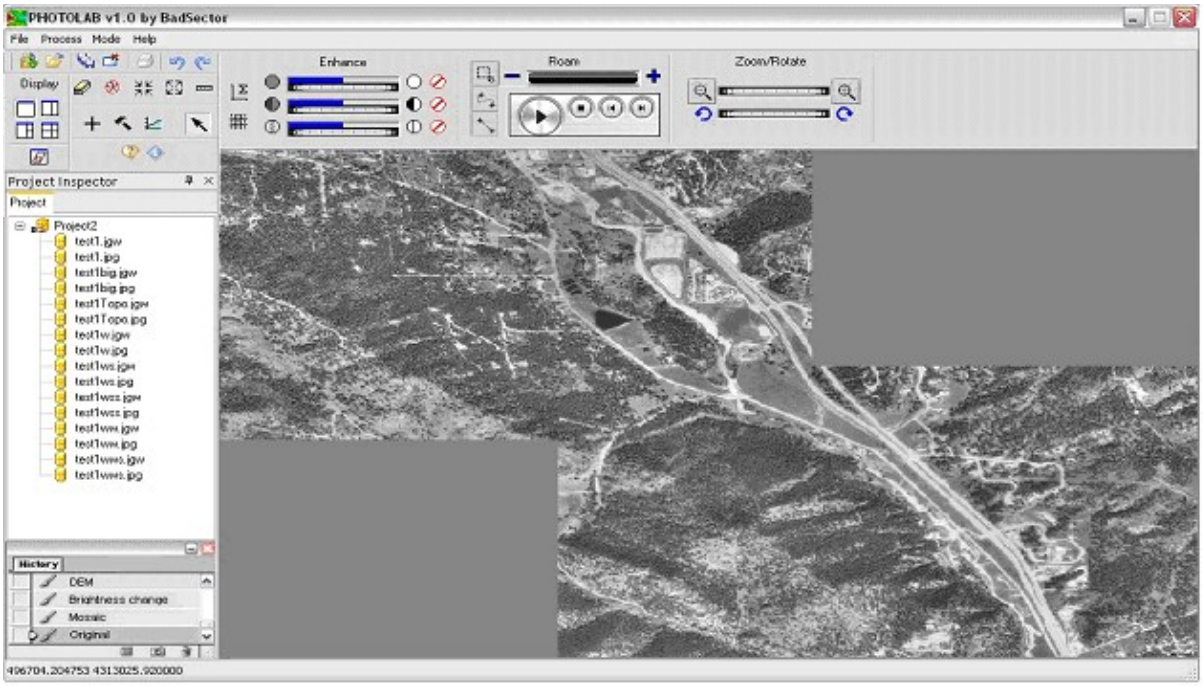


After forming a project or opening an existing one, the components of the project can be seen in project (manager) inspector window.





The second menu item is ‘Process’. The sub menu items are mosaic, DEM, Orthophoto and Super Resolution. After opening the files that are going to be processed with one of these methods, the user simply click on which process he/she wants the PHOTOLAB to perform. The result of an example mosaic process is below.



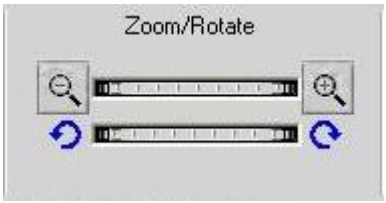
The next menu item is ‘Mode’ menu item. Mode is used for determining the tie point selection method. The user has two options, which are automatic and manual. In automatic mode, the tie points are determined by the program automatically and pointed in images, however in manual mode the user selects the tie points on images.

The last menu item is ‘Help’ menu item. PHOTOLAB provides user help files in order to make easier usage. These help files include the necessary information about the usage of the software in terms of functionalities. Help files also include a search and index box for faster search.

Above is a part of PHOTOLAB user interface toolbar. The display menu on the left provides the user to view the image(s) in five different scenes which are one-two-three-four images on the screen simultaneously or multiple images cascaded. This property of PHOTOLAB GUI is designed with MDI (Multiple Document Interface). Beside this, PHOTOLAB provides a number of other functionalities.

PHOTOLAB gives the user an opportunity to enhance the images by using various enhancement techniques such as setting the brightness values of images, contrast stretching and sharpening.

By the roaming function of PHOTOLAB (optional), the user can roam on a predefined pattern. The user can define a path either as a linear line or as a non-linear path. User can also calculate the distance between the two end points of this predefined pattern.



By the zoom/rotate function of the PHOTOLAB, the user can zoom in and out on the image and he/she can also rotate them.

At the bottom of the main window there is a status bar. The world file of the images give information about the world coordinates of each pixel in the image. PHOTOLAB uses this world file

information to calculate the original coordinates of the image. While the cursor is wandering on the image, the world coordinates of the place shown by cursor is written in status bar as UTM coordinates.

At the lower left side of the main window there is history window. The aim of the history window is to provide the user more possibilities to see the whole process step by step. PHOTOLAB records last four changes on the project and the original of the project easily. For this purpose history window shows these recorded changes for selected image.

6 SYNTAX SPECIFICATION

6.1 Classes

Class names start with a capital letter. If it has more than one word, first letter of each word is capital, too. In “x.h” files classes are divided into three parts and their order is private, protected and public members. Inside each part, members are ordered as;

1. Data structures

2. Variables

3. Functions

And there is one empty line to seperate these three. Besides, data structures, variables and functions are grouped according to their usage areas.

Opening curly brace of class is adjacent to closing parenthesis. Closing curly brace is in independent empty line.

A sample class structure is;

```
class Sample{
private:
        vector<Example> sampleVector;

        int variable1;
        int variable2;

        void DoSomething();
        void DoAnotherThing();
protected:
public:
```

}

6.2 Functions:

Function names in local areas start with lower case. If there is more than one word other words start with capital letter. Functions are implemented in x.cpp files. All implementation has a tab from the page edge Local variables are grouped at the top of the function implementation. Language concerned implementations like “if”, “while” and “for” has one tab between the nested belongings and curly braces. Opening and closing parenthesis have one space between parameters and each other. A sample function is;

```
void sampleFunction( int parameter )
{
        int variable;
        int variable2;

        while( isHappening )
        {
                doSomething( );
        }
}
```

6.3 Variables

Global variables started with capital letter. However local variables and class members have lower case first letter. If variable has more than one word other words start with capital letter, too.

6.4 Comments

There will be a comment line above each function which defines the aim of the function. The comment sentences will be formal and clear. Each member of the class “.h” files has definitions as comment. In “.cpp” files, before implementations there will be creation date of file, name of creator, modification date, name of modificator and class definition inside multiple line comment. A sample comment is;

```
/*  # Created 01.12.2007 by Serra Sinem Tekiroğlu

    # Modified 02.12.2007 by Serap Atılgan

   # This sample class comment is written for exemplification of comment specifications. */
```

7 PROCESS MODEL and PROJECT SCHEDULE

7.1 Team Structure

As a group we decided that “Democratic Decentralized (DD)” fits best to our project and project group. Our first priority is cooperation and this model ensures a high rate of cooperation since it

forces us to communicate in decision making. We appointed the tasks to the members for short durations at our weekly meetings. Another reason that we have chosen this structure is; we make decisions on problems or weekly tasks by agreement of each member. In case there is a contradiction among the group members, we chose a group leader to say the last word to prevent disagreements.

7.2 Process Model

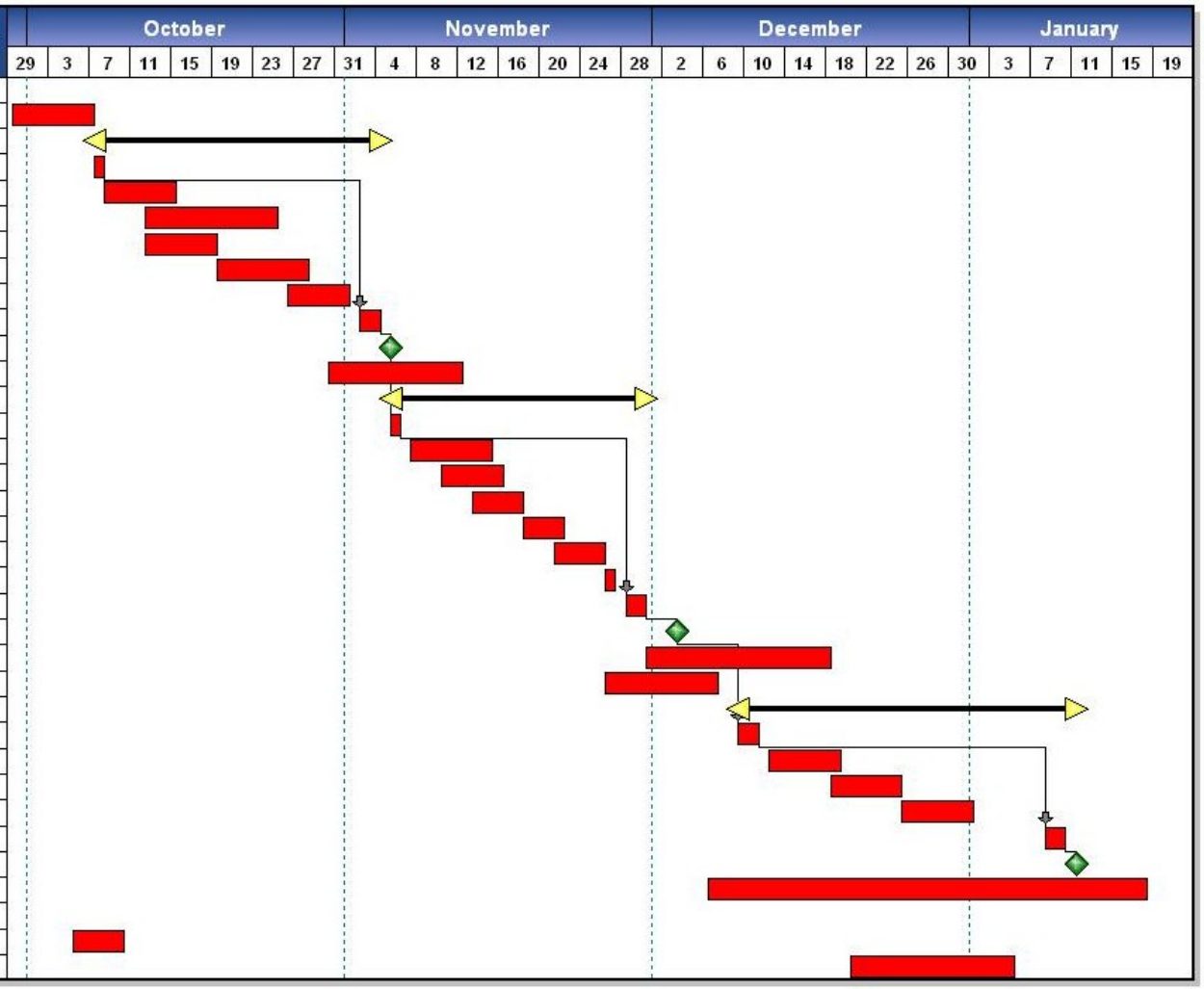
Our project team will iteratively go through planning, modeling, construction and deployment stages. For this reason, linear (waterfall) model of development best fits to our project. After the requirements analysis, we are going to make our initial design. Actually, Milsoft wanted the process model to be spiral during the design and implementation phases so they wanted more than one prototype. By this way, we will have a chance to go back and correct the faults in the design which we found during implementation.

7.3 Gantt Chart

Time Schedule of Photolab Project is shown by Gantt Chart. We have planned for a whole academic year.

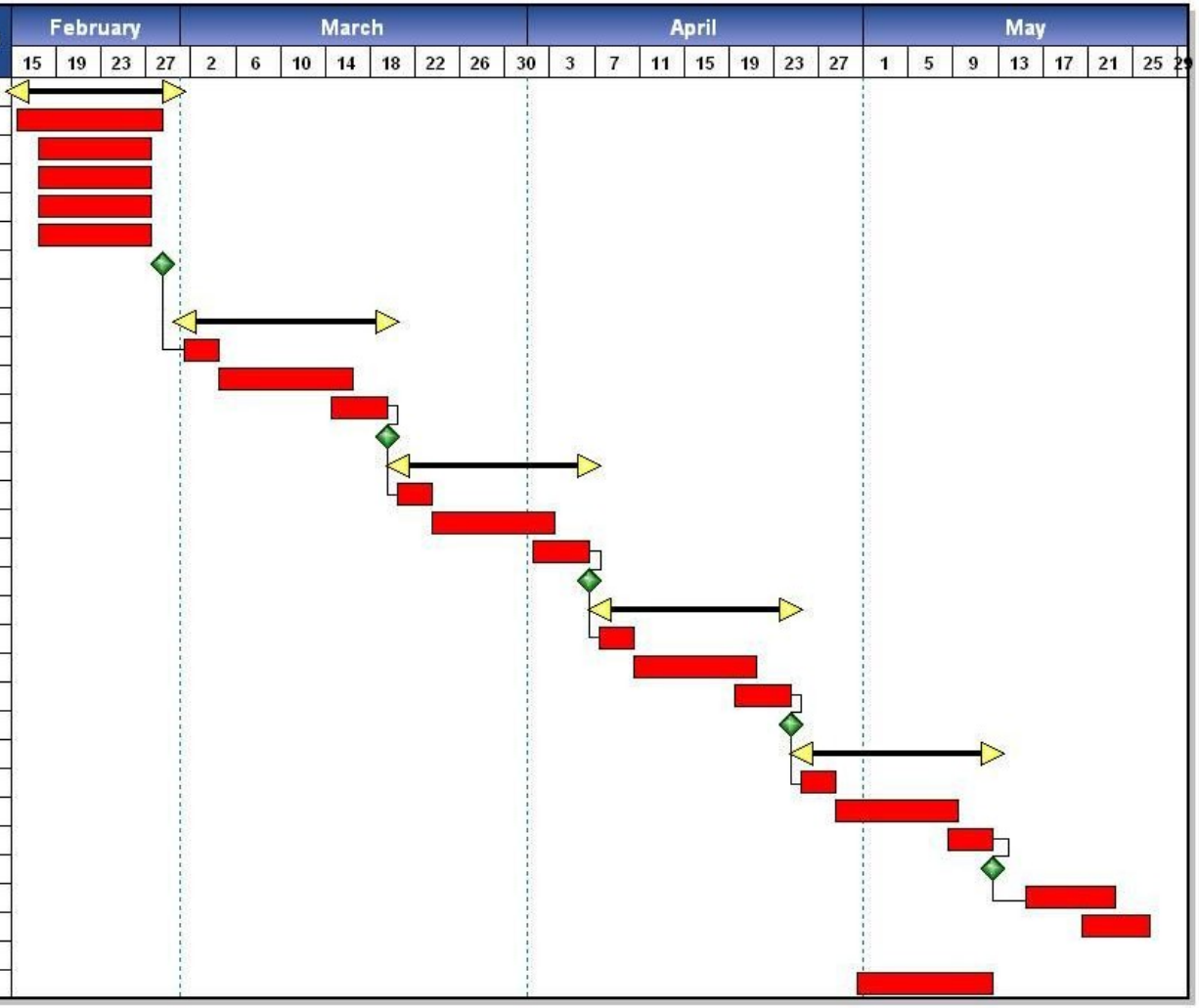
Gannt Chart

ID	Group Memb	Task	Duration
1			
2	1,2,3,4	*Teaming Up&Project Proposal*	7 days
3		Analysis Report	28 days
4	1,2	*Outline of Report*	1 day
5	1,2,3,4	*Market Research*	7 days
6	3,4	*Determining User Requirements*	13 days
7	1,2	*Determining System*	7 days
8	1,2	*Data&Behavioural Model*	9 days
9	3,4	*Functional Model*	7 days
10	1,2	*Last Version of Report*	2 days
11		MILESTONE 1	
12		Milsoft Assignment	2 weeks
13		Initial Design Report	25 days
14	3,4	*Outline of Report*	1 day
15	1,2,3,4	*Initial GUI Design*	8 days
16	3,4	*Defining Constraints*	6 days
17	1,2	*Design of Core Module*	5 days
18	1,2	*Design of Supporting Modules*	4 days
19	1,2,3,4	*Modelling*	5 days
20	3,4	*Update Schedule*	1 day
21	3,4	*Last Version of Report*	2 days
22		MILESTONE 2	
23		Milsoft Assignment	2 weeks
24	1,2,3,4	Initial Prototype Preparation	12 days
25		Final Design Report	33 days
26	1,2	*Outline of Report*	2 days
27	1,2,3,4	*Review of Modules*	7 days
28	1,2,3,4	*Final GUI Design*	7 days
29	1,2,3,4	*Final Design Verification*	7 days
30	3,4	*Last Version of Report*	2 days
31		MILESTONE 3	
32	1,2,3,4	Prototype Preparation	4 weeks
33		Other	
34	1,2	*Web Site Design*	5 days
35	3,4	*Presentation*	2 weeks



Gannt Chart

ID	Group Member	Task	Duration
36		Implementation of Toolbar	
37	1,2,3,4	*Enhancement options*	10
38	1	*Roaming*	10 days
39	2	*Zoom/Rotate*	10 days
40	3	*Display Layout*	10 days
41	4	*Basic Functions*	10 days
42		MILESTONE 4	
43		Implementation of Submodules	
44		Implementation of Mosaic	18 days
45	1,2	*Algorithm Search*	3 days
46	1,2,3,4	*Implementation*	12 days
47	3,4	*Testing*	5 days
48		MILESTONE 5	
49		Implementation of DEM	18 days
50	3,4	*Algorithm Search*	3 day
51	1,2,3,4	*Implementation*	12 days
52	1,2	*Testing*	5 days
53		MILESTONE 6	
54		Implementation of OrthoPhoto	18 days
55	1,2	*Algorithm Search*	3 days
56	1,2,3,4	*Implementation*	12 days
57	3,4	*Testing*	5 days
58		MILESTONE 7	
59		Implementation of SuperResolution	18 days
60	3,4	*Algorithm Search*	3 days
61	1,2,3,4	*Implementation*	12 days
62	1,2	*Testing*	5 days
63		MILESTONE 8	
64	1,2	Whole System Test	8 days
65	3,4	Optimization	7 days
66		Other	
67	1,2	*Presentation*	2 weeks



1-Hanife, 2-Serap, 3-Serra, 4-Meryem

8 TESTING

Testing is to detect the differences between existing and required conditions and to evaluate the overall software by analyzing it with respect to predefined validity and correctness norms. As Bad Sector team, we believe that the testing is a crucial part of a software design procedure since without testing; one can not be sure the overall validity and correctness of the final software package. Upon decisions on testing, Bad Sector team has prepared an initial test plan to follow. In subsequent sections the testing procedure will be covered according to this plan.

8.1 Test Items

Testing is a procedure which must be performed at several predefined points in the life cycle of the software development. Since testing is a very dependent and continuing activity, test plan must be developed according to these predefined levels. In order to specify these points, firstly test items must be identified. Bad Sector team has identified the items as below:

- Software Modules
  - GUI
  - File System
  - Photogrammetry
- Job Control Procedures
  - Production Scheduling and Control(refers to the prepared project schedule (Gantt Chart) and controlling time, cost and efficiency issues.)
  - Calls and Job Sequencing (refers to the job calls and the their sequence in these calls.
  - Job Control Language(“is a scripting language to instruct the system on how to run a batch job or start a Subsystem”[wikipedia].)
- Operator Procedures
  Operator procedures are the items to be tested in order to ensure that the application can run on different machines and environments.
- User Procedures
  User procedures are the items to be tested in order to ensure that user documentation is correct, complete and comprehensive.

8.2 Test Approach

As Bad Sector team we think that the software testing consumes 20 percent to 30 percent of software development resources. Therefore Bad Sector team has decided to specify an approach for all major testing tasks and for the required time estimation to do these tasks, with the minimum time & cost and maximum efficiency & satisfaction.

The team constructed the approaches by identifying the types of testing with the methods and criteria used in the testing.

8.2.1 Component Testing

Component testing is to test particular functions or code modules. It is usually the most micro scale of testing. Bad sector team plans to test the modules of Photolab with the component testing approach.

- GUI

Component Test Check List

Question to be answered:	Passed	Failed
Are the passes between subunits of GUI successful?	Yes, skip to the next question…	No, work on the unit further.
Do the shortcuts work correctly?	Yes, skip to the next question…	No, work on the unit further.
Are the menu items in correct order?	Yes, skip to the next question…	No, correct the order.
Are the appropriate menu choices active?	Yes, skip to the next question…	No, recover the activation.
Are the data interactions between subunits of GUI successful?	Yes, skip to the next unit.	No, focus on the data interactions.

- File System

Component Test Check List

Question to be answered:	Passed	Failed
Are all the file operations coherent with the File System?	Yes, skip to the next question…	No, work on the coherence.
Can all the image file types be converted to the common data type correctly?	Yes, skip to the next question…	No, work on the image type conversion.
Can File System and GUI interact successfully?	Yes, skip to the next question…	No, focus on the interaction.
Can File System and Data Library interact successfully?	Yes, skip to the next question…	No, focus on interaction.
Can video files be converted to image files correctly?	Yes, skip to the next question…	No, focus on conversion.
Can images with a relatively	Yes, skip to the next unit…	No, work on handling the

large size be handled?		large sized images further.

- Photogrammetry

Component Test Check List

Question to be answered:	Passed	Failed
Are the interactions between subunits of Photogrammetry successful?	Yes, skip to the next question…	No, work on the interactions.
Are all the algorithms efficient and fast?	Yes, skip to the next question…	No, work on the algorithms.
Can subunits work successfully?	Yes, skip the next question	No, correct the subunit.
Are the subunits coherent with the Photogrammetry module?	Yes, skip to next question…	No, work on coherence.

8.2.2 Integration Testing

As Bad Sector team, we decided to make an integration test to check the system after combining the parts (i.e. modules or individual applications such as image enhancement or photogrammetry processes) to ensure that they function together correctly. Bad Sector team can not be satisfied with a component, only working correctly on its own area. Therefore, the team will also ensure that the integration is successful while doing a continuous component testing in background.

8.2.3 Interface Testing

Bad Sector team plans to make an interface testing after completing component and integration testing successfully and solving the all critical errors. The aim of the team in doing interface testing is to check the external interfaces with Photolab in order to verify the execution times, data exchange, transmission and control. In order to make the test Bad Sector team needs to find external organizations having interfaces which can be tested with Photolab.

8.2.4 Performance Testing

As Bad Sector team, we also plan making a performance test in order to see how fast a component outputs under a particular workload or what percent of quality (time, cost, efficiency) and validity issues such as reliability or resource usage are satisfied by the system.

8.3 Pass/Fail Criteria

Bad Sector team decided pass/fail criteria for the test cases. The decisions include the suspension of a test in case an occurrence of a more urgent one. After the completion of the urgent test case,

the suspended test will be resumed from where it already is. Decisions also include the resumption of a test case until it succeeds as well as approval of a specific test case if it satisfies the criterion for all of the components forming the case.

9 CONCLUSION

The detailed design components of Photolab was simply identified in this report. As a summary, A simple Project definition and design constraints are expressed.Project is explained by architectural and component levels and object oriented diagrams. In addition possible testing scenarios are created to get ready for any cases.

At detailed design phase, Bad Sector has reviewed the whole architecture and according to current implementation state, the class structures have been rearranged. In the Initial design phase, Photolab have some problems and deficiencies upon input/output specifications or data structures. In this documentation these problems are handled and the solutions are described clearly.

In conclusion, Bad Sector has designed the whole Project in detail in order to be ready for the implementation phase. In the following duration of five months our team will concentrate on the coding and testing process.

REFERENCES

1. UML Sequence Diagram Tutorial, Effexis Software,LLC,[2005-2007]
   URL: http://www.sequencediagrameditor.com/uml/sequence-diagram.htm
2. IEEE Standard 829-1998, Standard for Software Test Documentation
   URL: http://www.ecs.csun.edu/~rlingard/COMP480/IEEETestPlanTemplate.pdf
3. High Resolution Panaromas using Image Mosaicing,Stanford University EE368 Final Project
   Laurent Meunier and Moritz Borgmann,May 2000
   URL: http://scien.stanford.edu/class/ee368/projects2000/project13/index.html

4. Norvelle, F.R., 1994. Using Iterative Orthophoto Refinements to Generate
and Correct Digital Elevation Models, Proc. Mapping and
Remote Sensing Tools for the 21st Century, ASPRS, pp. 134-142.
5. Krzystek, P., 1995. Generation of Digital Elevation Models, Second
Course in Digital Photogrammetry, Bonn, 6-10 February, Ch. 7.
6. [Davis] Davis, J. Mosaics of scenes with moving objects.(1998).Computer Vision
and Pattern Recognition
7. Feature Based Image Mosaicing,Satya Prakash Mallick,Department of Electrical and Computer Engineering, University of California, San Diego
URL: http://www-cse.ucsd.edu/classes/fa02/cse252c/smallick.pdf
8. Prof.A.Gruen&Henri Eisenbeiss(2006) UAV Photogrammetry IGP-ETH Zurich, 25/11/2007
URL: http://www.photogrammetry.ethz.ch/research/heli/index.html
9. Prof. Serge RIAZANOFF , depending on Envisat MERIS Geometry Handbook. (2008)
URL: http://www.brockmann-consult.de/beam/doc/help/visat/Orthorectification.html