



Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques

Universitat de Barcelona

Stitching Images Using Android Devices

Francisco Villalba Carballo

Director: Simone Balocco

Realitzat a: Departament de Matemàtica Aplicada i Anàlisi. UB

Gener, 2013

ABSTRACT

On this report is described in what the stitching technique consist of, which the state of art on stitching methods is and how this technique has been recently assimilated by mobile devices market.

This report also includes a full description of the developed applications which allow stitching images from pictures taken by an Android device. The first application is an Android application. The second application is a Java server.

The development, the installation, the management and configuration of the Android app and the service have been detailed in order to provide a correct understanding of the work done.

Both applications are detailed by several diagrams and explanations, which represent its internal design and how they have been implemented.

Table of Contents

1 Introduction	6
1.1 State of the Art	6
1.2 Motivation regarding the choice of the project	8
1.3 Main Goals	9
1.4 Gantt Diagrams	10
1.4.1 Planning of the Time Requirements	10
1.4.2 Summary of the Project Dedication	11
1.5 Report Scheme	12
2 Stitching Background	13
2.1 Smartphone Stitching	13
2.1.1 What is the Image Stitching?	13
2.2 State of the Art on Stitching Methods	15
3 Applications Overview	17
3.1 Use Case Diagrams	18
3.1.1 Android Application Use Case Diagrams	18
3.1.2 Server Application Use Case Diagram	26
3.2 Domain Model	27
4 Development	28
4.1 System Context Diagrams	29
4.1.1 Android Application Context Diagrams	29
4.1.2 Server Application Context Diagrams	45
When launching the server, the user has nothing to configure	45
4.2 Flowcharts	46
4.2.1 Android Application Flowcharts	47
4.2.2 Server Application Flowcharts	50
4.3 Class Diagram	52
4.3.1 DroidStitcher Class Diagram	52
4.3.2 StitcherServer Class Diagram	54
4.4 Class Specifications	55
4.4.1 DroidStitcher Classes	55
4.4.2 StitcherServerClasses	64
4.4 Event-trace Diagrams	66
4.4.1 Android Application Event-trace Diagrams	66
4.4.2 Server Application Event-trace Diagrams	71

5 User Guide	73
5.1 Application Specifics	73
5.2 Android requirements.....	74
5.3 Installing the Android application.....	75
5.4 Server requirements.....	76
5.5 Back-end Installation.....	76
5.6 Tutorial.....	79
5.6.1 DroidStitcher User Guide.....	79
5.6.2 StitcherServer User Guide.....	87
6 Conclusion.....	88
6.1 Objectives Achievement	88
6.2 References	89
6.3 Appreciation.....	90
6.4 Possible Improvement	91
Annex	92
I How Java runs MATLAB.....	92
II Working with Accelerometers.....	93
Sensors	93
The Accelerometers.....	94
III Protocol	97
Connection: Waits and Errors	99
IV Android Manifest.....	100
V Device camera usage regulations	101
Image Compression.....	101
VI Data Distribution and Storage.....	102
VII Algorithms	103
SIFT.....	103
RANSAC.....	103
VIII Image Cropping.....	104
IX Examples.....	107

1 Introduction

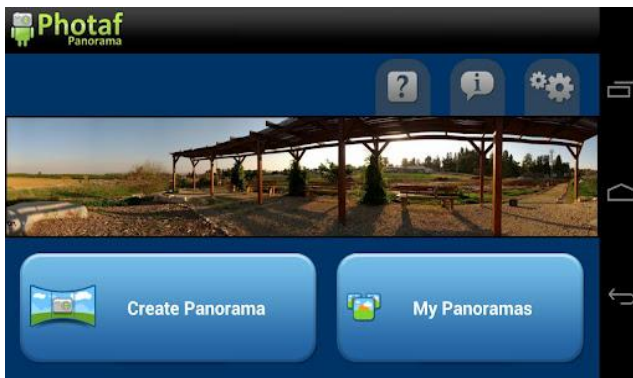
1.1 State of the Art

The aim of the project is to explore the possibility of implementing a stitching technique on an Android device. Nowadays most of the smartphones available on the market include cameras which allow the user easily record videos and take pictures.

In the first generation of telephones there were a few applications based on cameras. However, nowadays the technological advance of the smartphone's hardware allowed the appearance of camera-based applications.

When we talk about camera applications from a development perspective, the first thing that come in mind are Computer Vision techniques. One of the most popular camera applications is the image stitching. This application consists in creating a panoramic image from a collection of consecutive images.

Such applications are nowadays available on Android and iOS markets, allowing their users creating panoramic images of its surroundings.



For instance Photaf, an application that can accomplish such task.

Figure 1 illustrates an impressive result using Photaf.

Figure 1: Photaf snapshot ¹

Other relevant application developed entirely by Google is PhotoSphere, see Figure 2, which creates not just a panoramic, but a 2D projection of the device 3D surrounding.



Figure 2: PhotoSphere snapshot ²

Easily ascertainable, is the fact that the state-of-art in this kind of applications is booming into the market.

So, which improvements can we provide into this field?

¹ www.photaf.com

² www.androidpipe.com

The Android application developed for this project has not been created with commercial purposes. To work properly it needs a running service which processes the captured images. So the project is not just an Android app, is the result of combining the possibilities which a smartphone offers, such as mobility, with the processing power of a computer.

The project itself can be useful to obtain high resolution images of physical objects and panoramas.

On the previous page a question can be found. If the Android market is populated with panorama applications and stitching software can be found on the web, what can bring a new application?

The answer is because this application can work with camera pictures, not just camera previews as smartphone applications do. The stitched images can be camera previews (low resolutions) or camera pictures (high resolutions). The panoramas build by Photaf, for example, are created using camera previews.

Another achievement is the possibility of bringing such technique to old device series. PhotoSphere does not work on devices with prior Android OS 4.0 versions installed. This application also works with camera previews, not camera pictures.

So the project is a service able to provide a stitched image. This stitched image could be created from low resolution images (camera preview frames) or high resolution images (camera pictures). As the stitching process is not executed on the device, the Android application is free from executing operations that consume several resources, such as image stitching.

1.2 Motivation regarding the choice of the project

The main reason for choosing this concrete project was its development, which include Android and Computer Vision.

The project development could bring the opportunity to improve and learn how to develop an Android app, which is an interesting subject. The idea of also working with Computer Vision techniques made this project a chance for gaining knowledge. This combination also provides a great expectation for the results.

The project was initially thought as a skin mole finder software. The final application would be Android native, and it would be capable of shooting images of the skin, locate moles or irregularities presented and be able to measure changes on these moles. To top it all the shooted pictures would be mapped in a virtual skin surface, allowing the assessment of potential mole diseases.

Since the initial project was too ambitious, it was decided, in agreement with my tutor to focus on a specific task, and to optimize it for the best performances.

The final project is a mix between Android and Computer Vision fields, consisting in an application running in a mobile device able to connect with a back-end server in order to send images and obtain a stitched result. The server application processes the images given by the application, creating a new stitched image and returning it to the client.

1.3 Main Goals

The first goal to complete this project was discovering what possibilities could allow an Android smartphone on the Computer Vision fields.

After understanding how the smartphone camera works and how to gather images with the device, the next step to take was how about transferring the data wireless.

The third step was focused mainly on how to process that data and bring them back to the client.

As a summary of the work done and as an introduction to the project development, the following list explains the goals to achieve:

- Improve Android knowledge.
- Learn how to manage the smartphone camera.
- Research for the optimal way of transmitting data considering the requirements of this case.
- Fusing the back-end application with a third party algorithm for stitching images.
- Implementing an optimized way to return the result to the smartphone.

It must be noticed that the work has been mainly focused on the Android and server development, not the image processing.

Additionally, along the project we explored the use of the Android sensor. Such feature could be used in future implementation of this software for improving the results of the stitching. On the Annex is exposed why the sensors, concretely accelerometers, were useless on this case.

As a final objective for the project I decided to improve my level of technical English so I decided to write entirely this report in this foreign language.

1.4 Gantt Diagrams

On this subchapter are presented two Gantt diagrams. The first one represents a first estimation of time required on development. The second diagram corresponds to the real time devoted to the project.

1.4.1 Planning of the Time Requirements

This diagram is not detailed as the real devoted time Gantt diagram is (**1.4.2 Summary of the project dedication**).

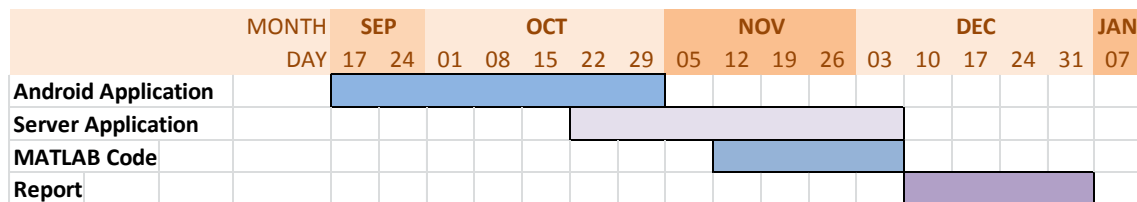


Figure 3

As it can be seen, the working time has been divided in four parts. Each part corresponds to a general task.

Tasks explanation:

- Android Application: Estimated time on developing the Android application.
- Server Application: Estimated time devoted to the Server application development.
- MATLAB code: Estimated time devoted to searching for stitching techniques and its possible implementation on the project.
- Report: Estimated time devoted on the report writing.

1.4.2 Summary of the Project Dedication

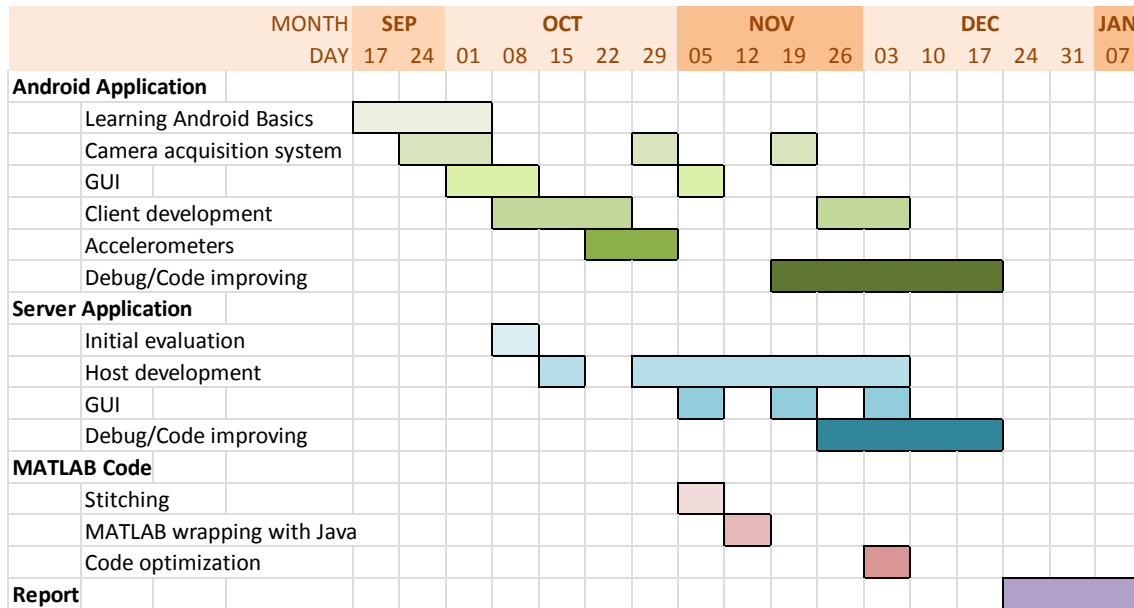


Figure 4

Tasks explanation:

- **Android Application:**
 - Learning Android Basics: Android introduction, basic knowledge.
 - Camera acquisition system: Learning the Camera basic knowledge and its proper management.
 - GUI: Graphical User Interface.
 - Client development: Connection with the host.
 - Accelerometers: Working with the sensors.
 - Debug/Code Improving: Clearing code, avoiding bugs.
- **Server Application:**
 - Initial Evaluation: Searching the best way to connect client and host.
 - Host development: Connection with the client.
 - GUI: Graphical User Interface.
 - Debug/Code Improving: Clearing code, avoiding bugs.
- **MATLAB:**
 - Stitching: Understanding what stitching is.
 - Adding MATLAB to Java: Searching for the best way.
 - Code optimization: Adding new functionalities on the code.

1.5 Report Scheme

This report is divided in six chapters, explained below:

- 1 Introduction: In this chapter are explained an introduction to image stitching on smartphones, the project personal motivation and the time devoted to the project.
- 2 Stitching Background: This chapter is focused on the stitching methodology and the issues that can arise when implementing on smartphones. It is also explained the advantages of this application against others.
- 3 Applications Overview: This chapter explains the features of the application, available for the user. Use case and domain model diagrams are also commented.
- 4 Development: This chapter exposes the most relevant diagrams from a development view, such flowcharts, system context diagrams or sytem sequence diagrams.
- 5 User Guide: This chapter describe the user guide, illustrated with screenshots and detailed explanations.
- 6 Conclusion

It is also included an annex at the end of the report.

2 Stitching Background

2.1 Smartphone Stitching

At this point it is explained what is the stitching, and what kind of problems arise when such technique is implemented.

2.1.1 What is the Image Stitching?

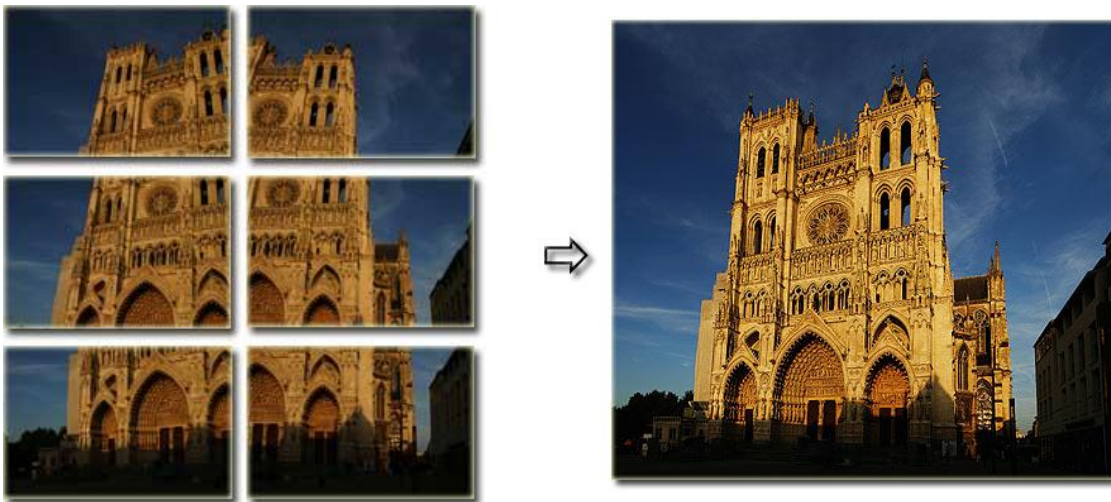


Figure 5: Image stitching³

As it can be seen on the images above, image stitching consists on creating an image of higher resolution from a set of images with lower resolution. The methodology is not a simply collage, but the final result corresponds to a real object, in this case a cathedral.

We can note the following facts:

- Some details of each lower resolution image are overlapped.
- The stitching process deforms the images using affine transformations, then, the final image frame has been cropped out.
- The final image might have a size approximately equal to the sum of the initial images sizes.
- The final image is time-displaced. The first original images were taken in a short time lapse, but their time stamps are different.
- The point of view on the initial images and the final image is the same. The observer had not moved from his position.

These inferred tips are in fact the problems that stitching brings to reality.

³ www.ptgui.com

2.1.2 Stitching and Smartphones

Consider the problem in which a bird appears on one picture, but not in others, and this part of the image is overlapped to other image. In the stitched image such bird will appear as an artifact.

Another issue that produces artifacts is the use of images not taken from the same place, which can lead to parallax errors.

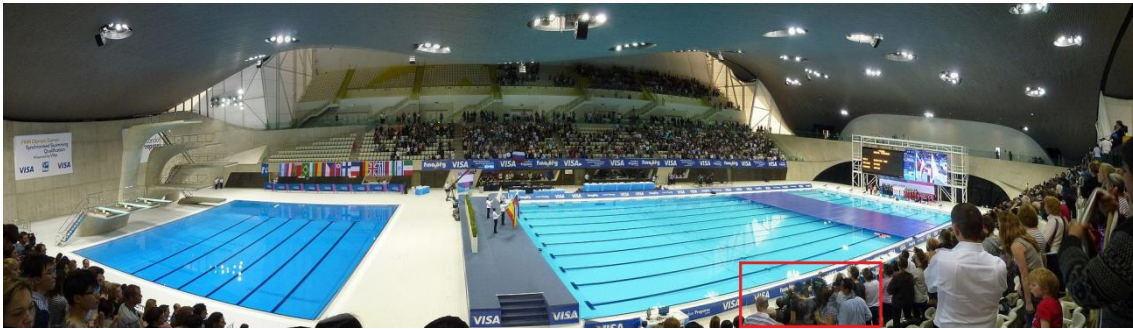


Figure 6: Stitched Image example ⁴

The above figure corresponds to a panorama taken on the London Aquatics Center interior. At first glance it looks like a perfect result, but if a close look is taken:



Figure 7: The artifacts produced by time difference can be easily discovered on this zoom shot. ⁵

In order to minimize the errors few advices should be followed:

- When capturing a panorama, the observer reference for the stitching is the photographer himself. The observer should rotate on a vertical axis which can ideally be drawn from the photographer head to his feet (stationary view).
- When the photographer shifts his camera for changing the point of view, he should not toss or shake the device. The slowest the movement is, the more images will be captured and a better result can be obtained.

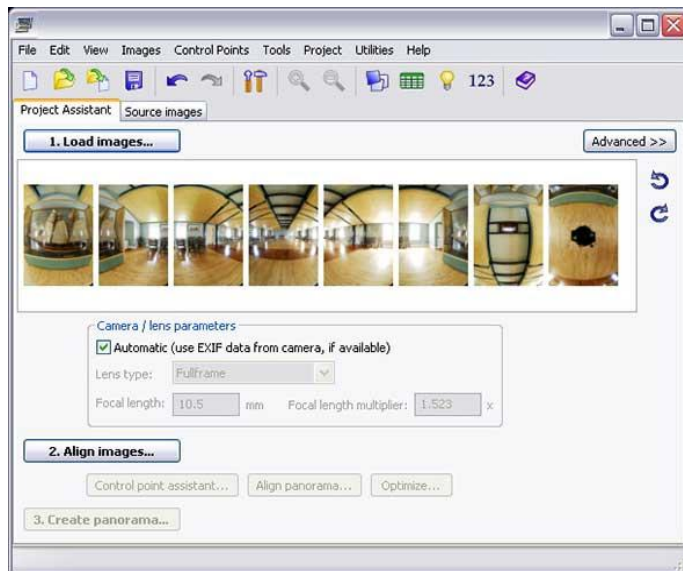
⁴ www.wikipedia.org

⁵ www.wikipedia.org

Such advices should be carefully followed when the image stitching application is used on a smartphone.

2.2 State of the Art on Stitching Methods

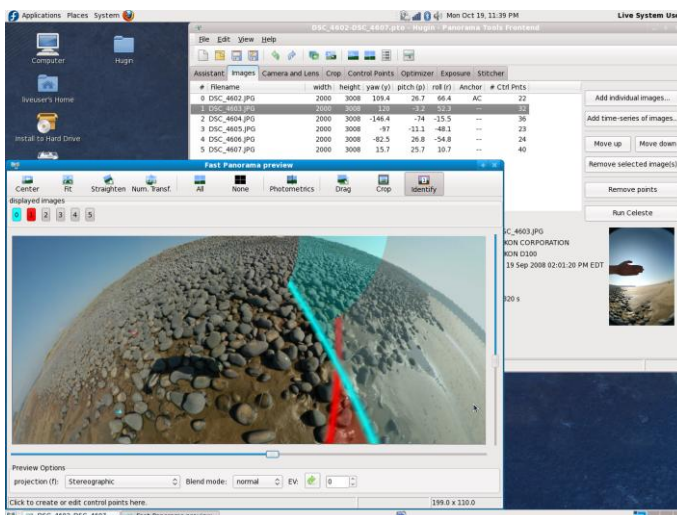
The most useful software employed on image stitching is PTGui (Graphical User Interface for Panorama Tools). This software allows the user to stitch previously obtained images and correct possible errors on the image.



As seen on *Figure 8*, images should be acquired previously, and then processed by the software. This software is based on Panorama Tools, a free source code for stitching created by Professor Helmut Dersch from the University of Applied Sciences Furtwangen⁶ as a toolkit to correctly stitch images.

*Figure 8: PTGui snapshot*⁷

Another relevant free software kit, also based on Panorama Tools, is Hugin, which provides a better experience to the user due to its capabilities on modifying images and obtaining high quality results.



*Figure 9: Hugin snapshot*⁸

This is cross-platform software and brings non-limit possibilities over the image deformation in order to improve the final product.

⁶ <http://webuser.hs-furtwangen.de/~dersch/>

⁷ www.intpicture.com

⁸ hugin.sourceforge.net

Examples above are based on Panorama Tools suite, which is developed entirely on C++ and has obtained a great reputation on the net. The panorama tool is software which development is still active and it is constantly improved.

Panorama Tools is not the only project related with image stitching. Fortunately there are other projects like the one developed by the TobW⁹ team which has been entirely developed using MATLAB (plus a bash script). The code from this team has been used on the project's image stitching process.

The main steps followed by this software are the following:

- Capture two images
- Detect **SIFT** keypoints in both images and compute the set of matching points
- Apply **RANSAC** to estimate a homography that transforms the images such that the points coincide
- Transform the images using this homography
- Blend the images together
- Capture the next image and repeat

⁹ www.tobw.net

3 Applications Overview

The diagram presented on *Figure 10* illustrates how both applications, the Android application and the server, work jointly.

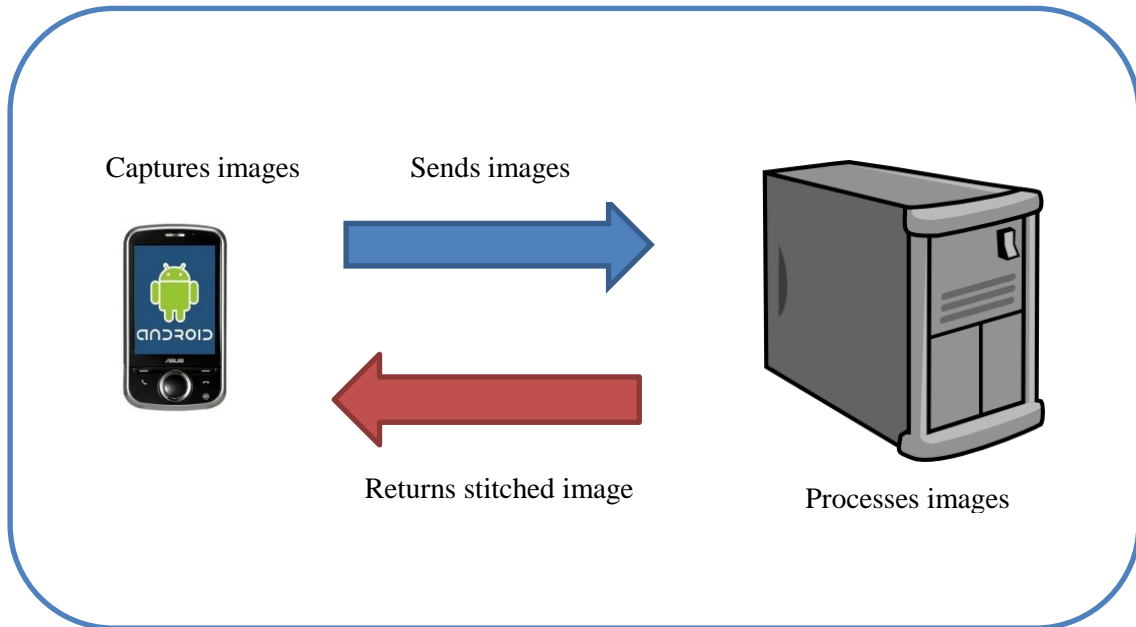


Figure 10

Before explaining the use case diagrams, is necessary to know which the application functionalities are:

- Android application (DroidStitcher):
 - Stream images to the server.
 - Configure the quality of the images used on stitching.
 - Check for images from old stitching sessions.
- Server application (StitcherServer):
 - Offer connection to multiple devices.
 - Session window generation with live-time streaming view from the device camera.
 - Stitch the images and return the result to the smartphone.

In this chapter is carefully explained the relation between the user and both applications and what are the actions the user is able to do when working with them.

On this chapter there is also a second part where is shown the application domain diagram.

3.1 Use Case Diagrams

On this subchapter will be explained the relation between the user and the application by use case diagrams. Each diagram shows the user possible actions in different contexts of the application. Due to the relevance that takes the user interaction on the Android app, the possible situations have been thoroughly outlined.

3.1.1 Android Application Use Case Diagrams

On this concrete case, the installation should be completed on the device and the application launched. The green labels found on the diagrams redirect to other diagrams, located on the figure labeled as by Fig. XX.

The diagrams disposal corresponds to the hierarchy presented on *Figure 11*.

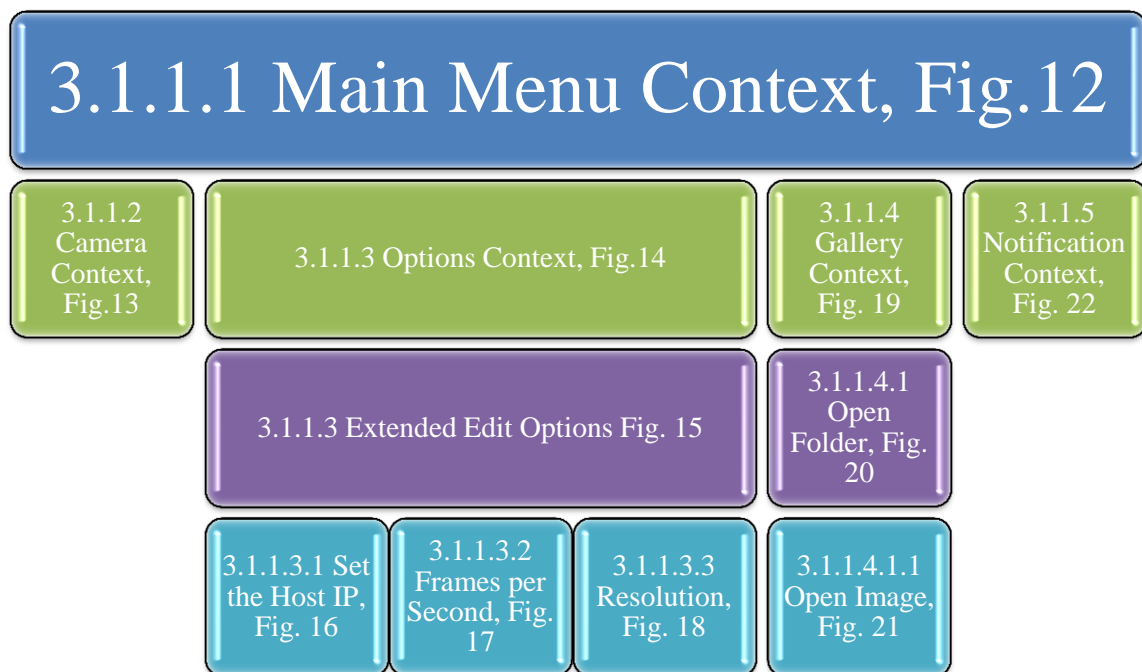


Figure 11

3.1.1.1 Main Menu Context

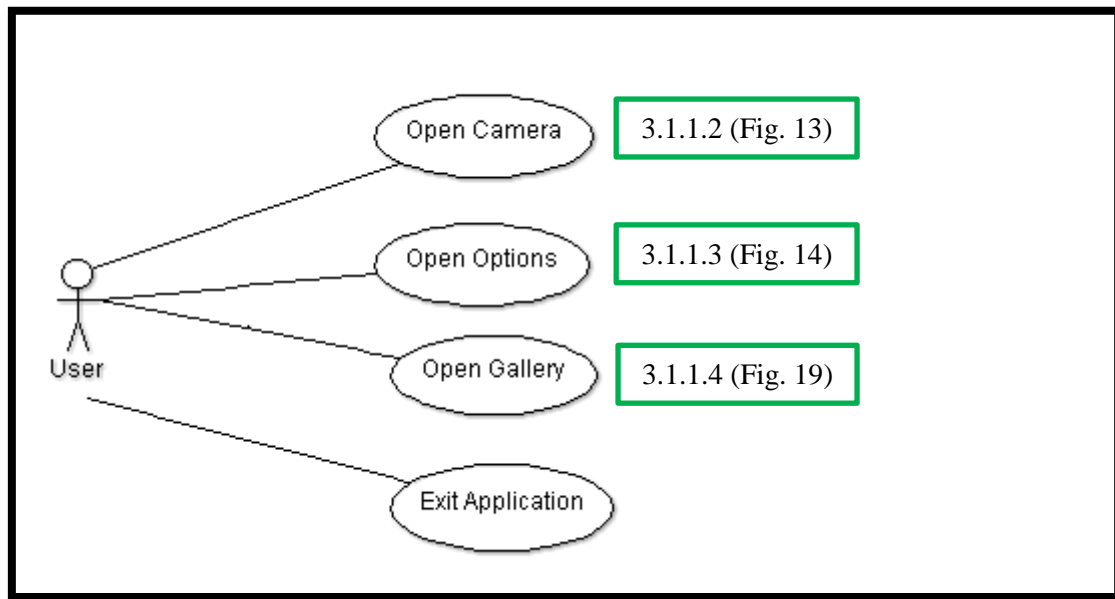


Figure 12

On this case the user can choose between:

- Open Camera: To start a new streaming session.
- Open Options: To configure the application options.
- Open Gallery: To manage previous session contents.
- Exit Application: Exit from the application doing nothing.

3.1.1.2 Camera Context

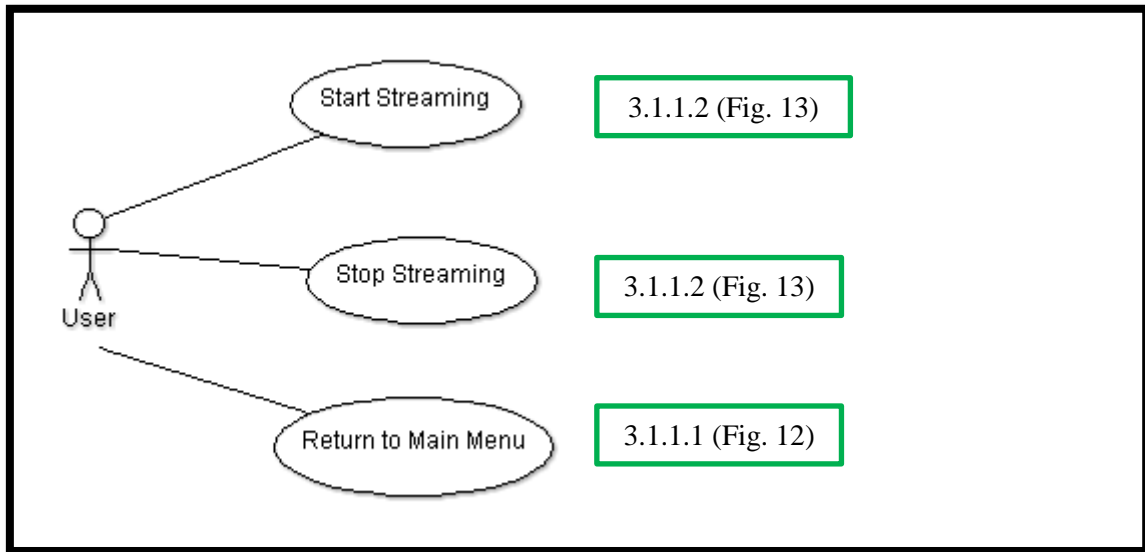


Figure 13

The user can:

- Start Streaming: Start a new **Streaming Session**.
- Stop Streaming: Finish current **Streaming Session**.
- Return to Main Menu

3.1.1.3 Options Context

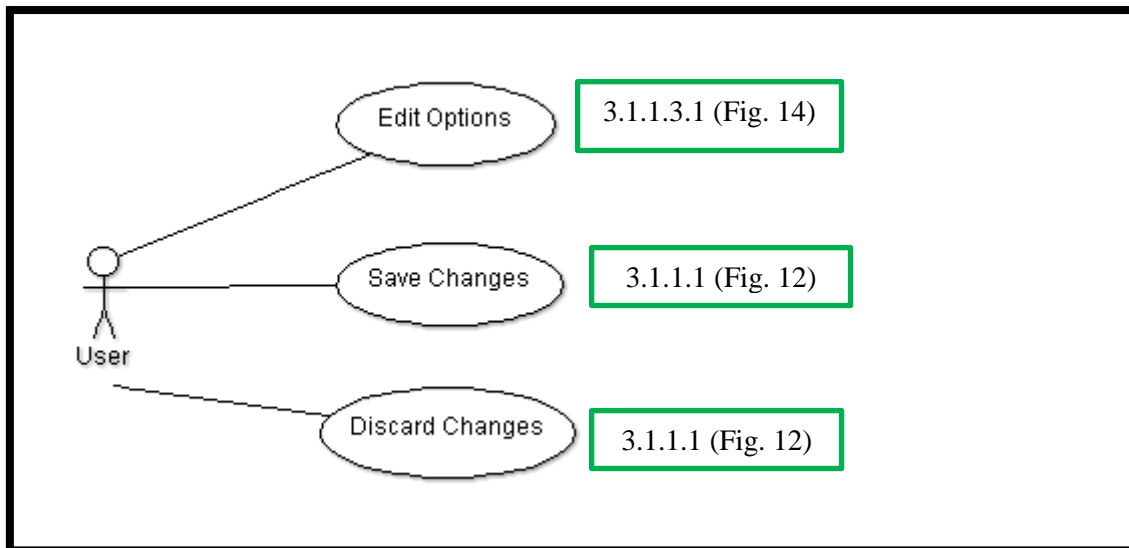


Figure 14

The user chooses between:

- Edit Options: This is NOT a new context, explained on 3.1.1.3 Extended (*Figure 15*).
- Save Changes: Save current changes and exit.
- Discard Changes: Discard current changes and exit.

3.1.1.3 (Extended) Edit Options

This subchapter is mainly focused on the “Edit Options” use case and describes these possible options.

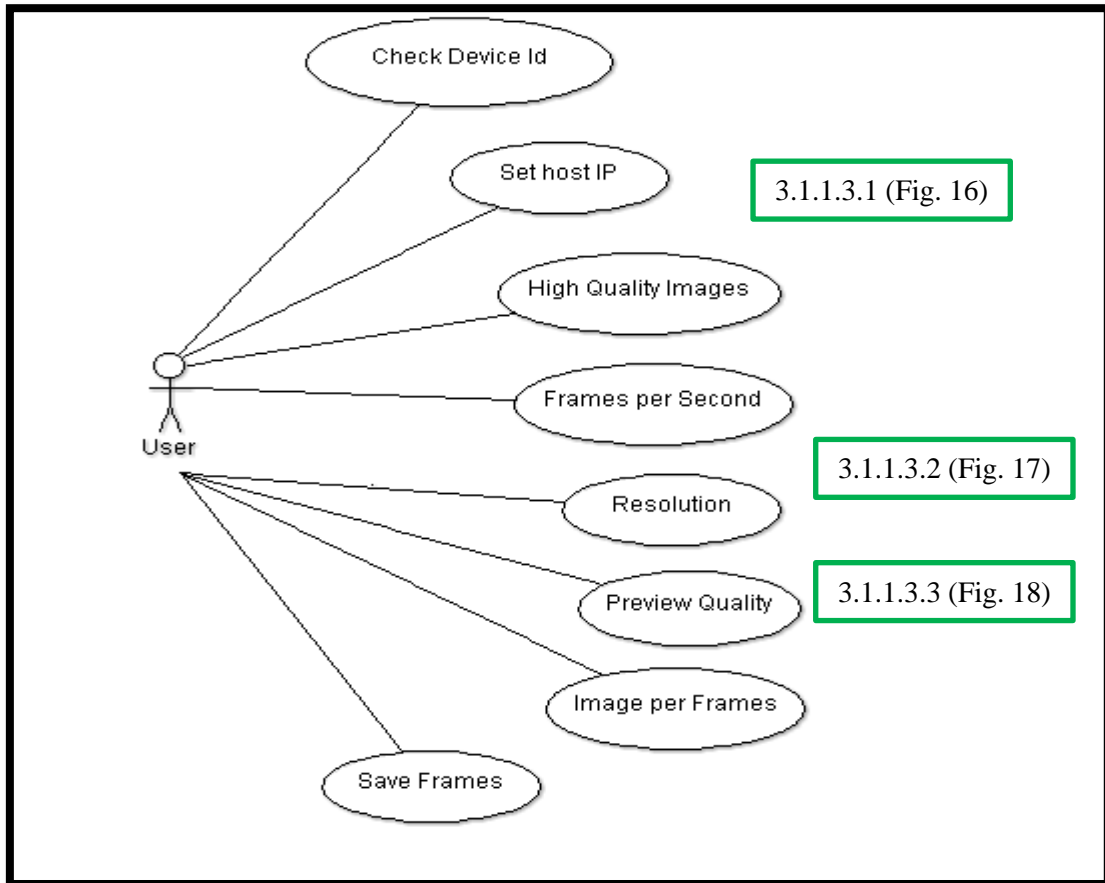


Figure 15

Options:

- Check Device Id: The device Id is composed by 2 values, the device name and the Android Identifier.
- Set host IP: The user can change the host IP and PORT (socket).
- High Quality Images: The images to stitch will be camera pictures instead previews.
- Frames per Second¹⁰: The user can select between different FPS ranges offered by the device.
- Resolution¹¹: Camera preview frames resolution.
- Preview quality: The quality of the camera preview frame compression into JPEG.
- Image per Frames: This number represents the number of camera previews frames have to pass before taking of image to stitch.
- Save Frames: Stores every preview frame captured by the camera on the host.

¹⁰ These values may differ between devices.

¹¹ These values may differ between devices.

These options grant access to configurations submenus, represented on the following diagrams:

3.1.1.3.1 Set host IP

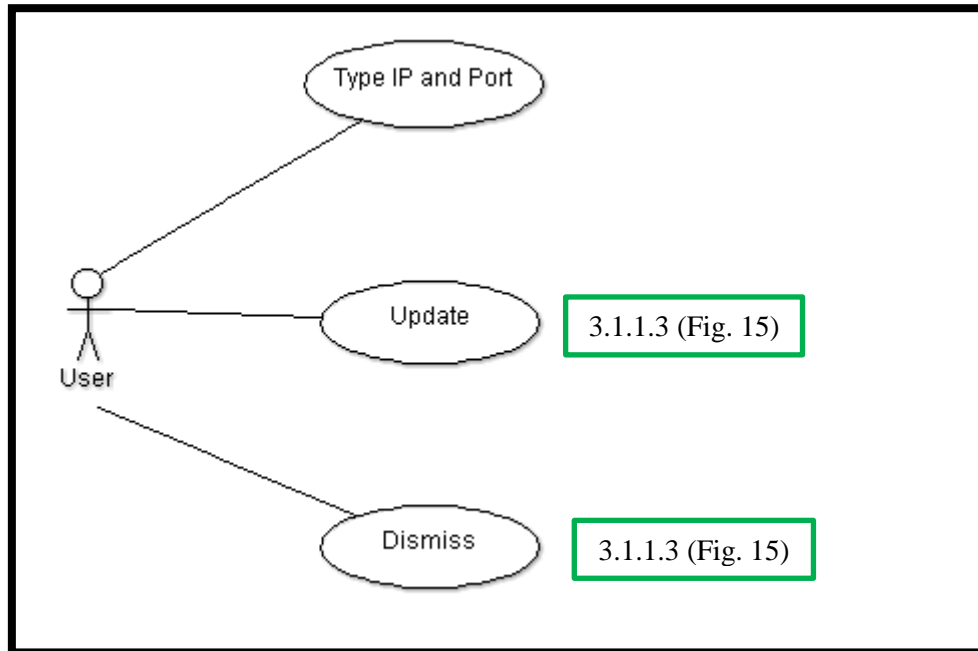


Figure 16

Available options:

- Type IP and Port: Enter the IP and Port for host connection.
- Update: Update the entered data.
- Dismiss: Exit without changes.

3.1.1.3.2 Frames per Second

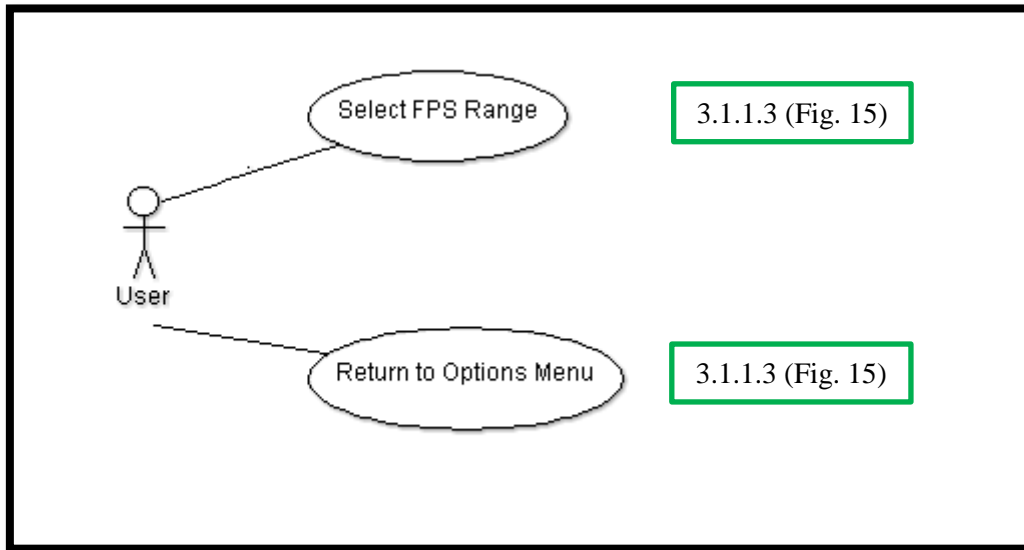


Figure 17

The user can choose one of the listed FPS ranges or return to the Options Menu.

3.1.1.3.3 Resolution

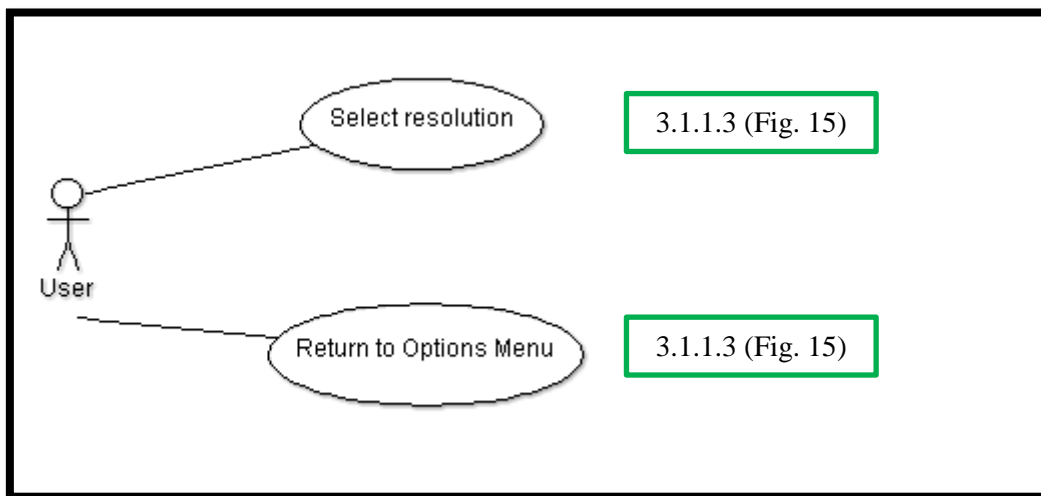


Figure 18

The user can choose between one of the listed resolutions or return to the Options Menu.

3.1.1.4 Gallery Context

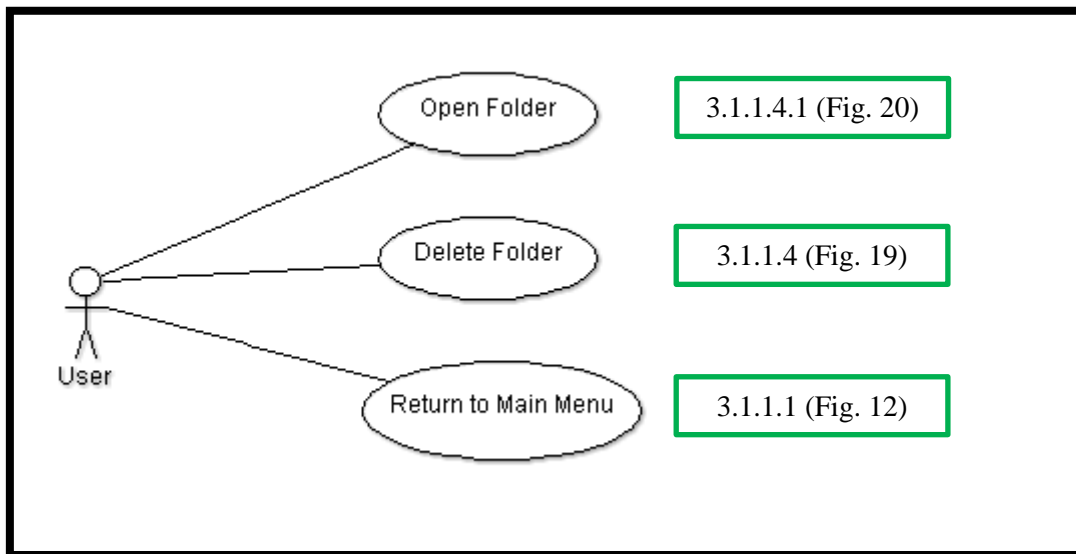


Figure 19

On this case the user can choose between:

- Open Folder: Opens the selected folder (short click).
- Delete Folder¹²: Deletes the selected folder (long click).
- Return to Main Menu.

3.1.1.4.1 Open Folder

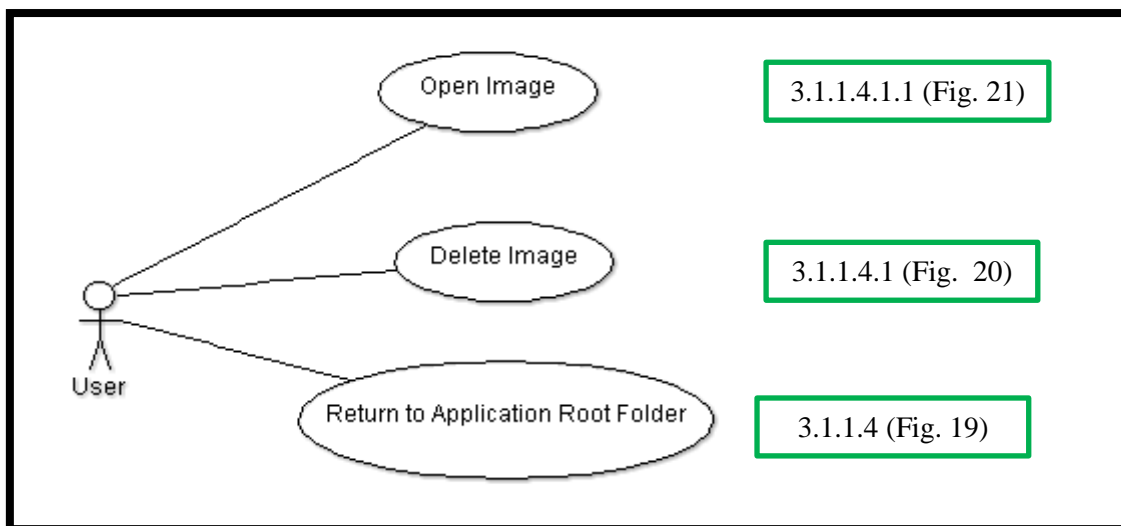


Figure 20

On this case the user can choose between:

- Open Image: Opens the selected image (short click).
- Delete Image¹³: Deletes the selected image (long click).
- Return to Folder List.

¹² A deletion dialog will appear.

¹³ A deletion dialog will appear.

3.1.1.4.1.1 Open Image



Figure 21

On this case the user can choose between:

- Slide to Previous Image: Get the previous image view.
- Slide to next Image: Get the next image view.
- Return to Image List (on the current folder).

3.1.1.5 Notification Context

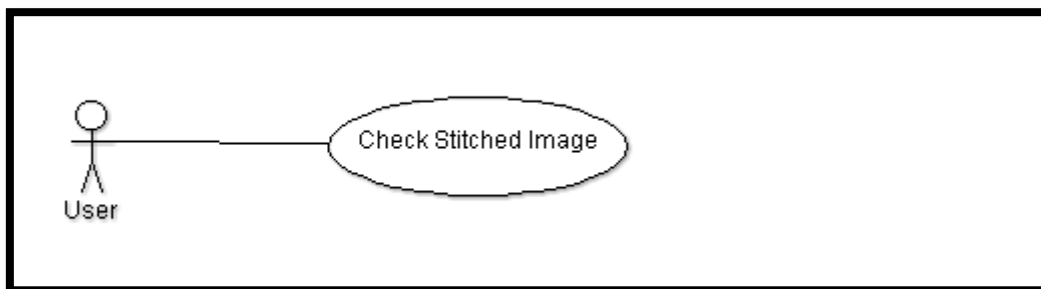


Figure 22

On this concrete case it is supposed that the user have received a notification from the service, notifying an image has been successfully, or not, stitched. If the user clicks on the notification, will get the view of that image, or a null image if the stitched process have found any problem.

3.1.2 Server Application Use Case Diagram

The server administrator has a quite simply diagram, because the only actions available are launching the server or stop it.

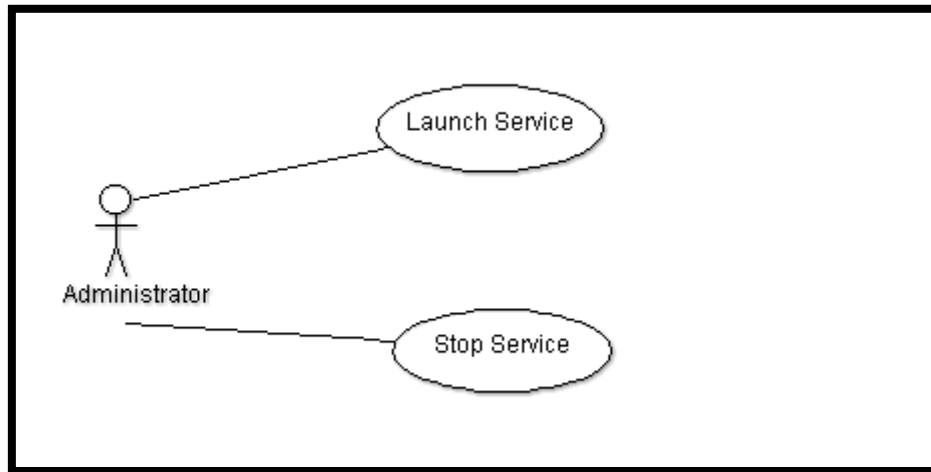


Figure 23

When a new session is started, a window will pop up on the screen allowing the user (administrator) to check information about the connection. It is also show a live stream video corresponding to the camera viewing.

3.2 Domain Model

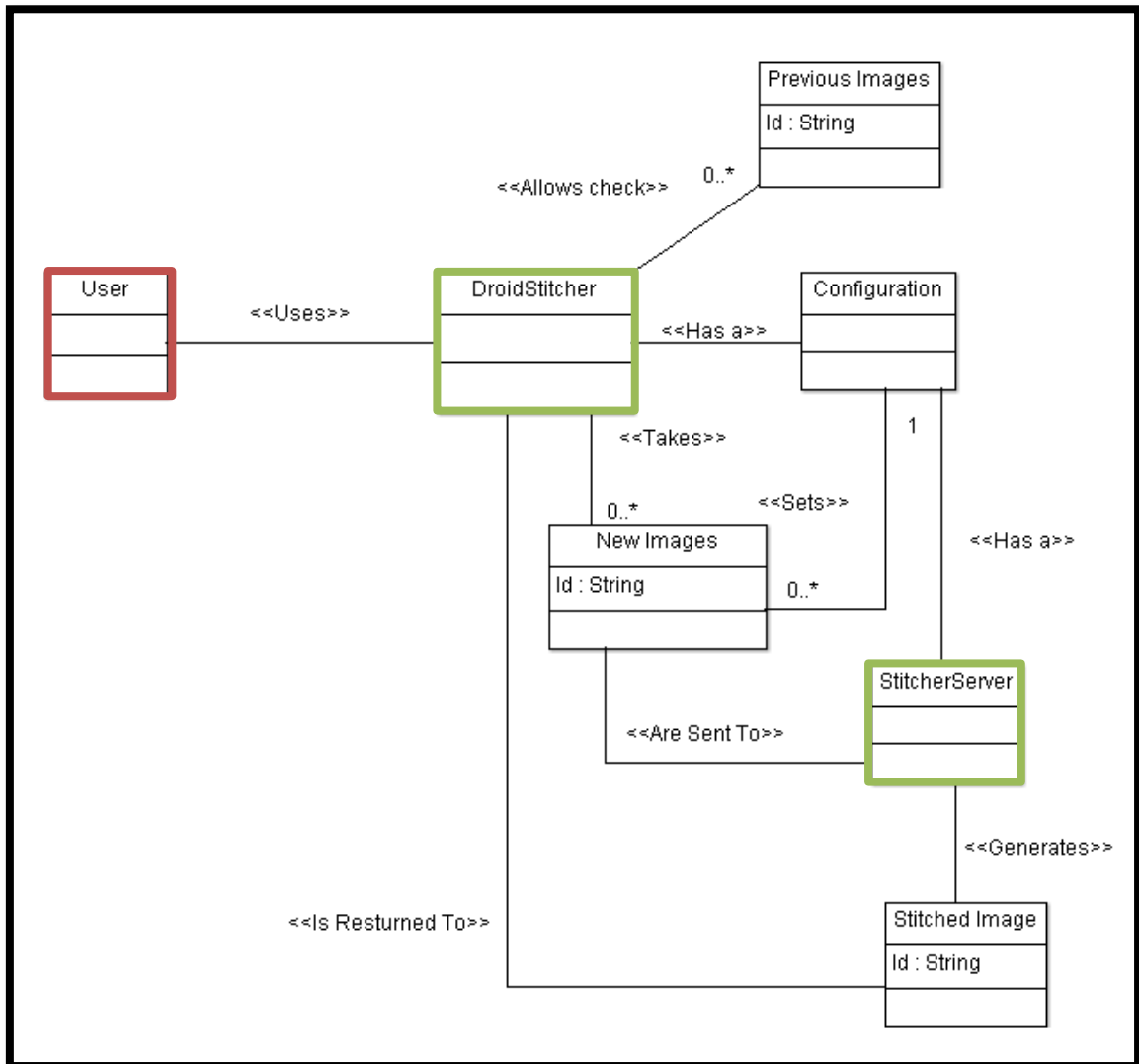


Figure 24

This diagram shows the relation between the user, the Android application and the server application. The use case diagrams can be explained inside this domain model.

The red labeled object is the user, while the green labeled objects are the applications.

On the diagram can be seen that the object called “Configuration” is related with the Android Application and the server, due to necessary adjustments done on the server when a new session starts.

As this diagram represents a concrete session, it is necessary to explain that the objects named “New Images” and “Stitched Image” will be “Previous Images” on later sessions.

4 Development

On this chapter are exposed the development related diagrams, which are:

- System Context Diagrams
- Flowcharts
- Class Diagrams
- System Sequence Diagrams

Due to the high level of interaction required by the Android application, the System Context diagrams have been thoroughly schematized. The automatic processes, which do not require the user interaction, have been detailed on flowcharts and system sequence diagrams.

There are several web sites providing information about how to implement efficient Android applications. Some of the facts that should bear in mind are the resources consumed by the application in order to avoid memory consumption or the loading time, which could deteriorate the user experience.

To not overload the application Main UI it is heavily encouraged from Android Developers to use threads when loading information, this way the user do not notice about it and the application flows in a faster way.

The guides on Android Developers bring developers tricks and tips to develop in an effectively way. For example, when creating the camera related methods, you can find a full tutorial where every part of the code is fully explained and also it is possible to use snippets to learn how to use the code.

As this chapter is focused on the implementations of both applications, should be noticed that the advices found during the development have been followed in order to design and develop well-structured applications.

4.1 System Context Diagrams

These diagrams show the interaction between the user and the applications. On these diagrams the applications are represented as black boxes and no internal performance is revealed.

4.1.1 Android Application Context Diagrams

This subchapter is focused on the context diagrams related with the Android application.

On figures 25, 26, 27 and 28 is explained the diagram hierarchy.

Starting from the application main menu, the user has three options.

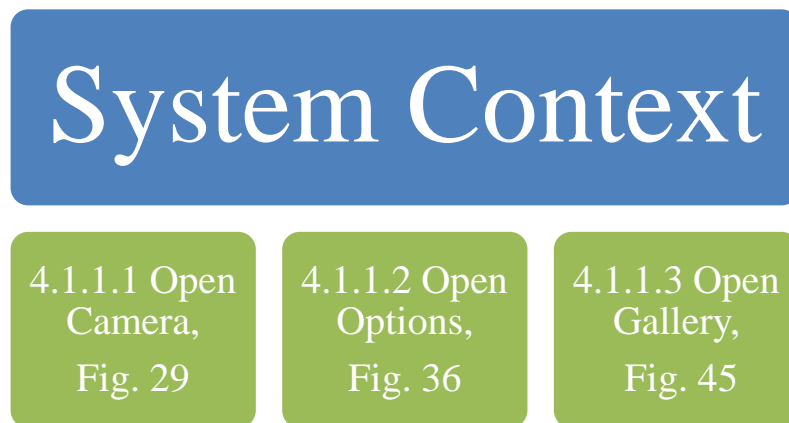


Figure 25

The context hierarchy explained on *Figure 26* is produced when the user works with the camera.

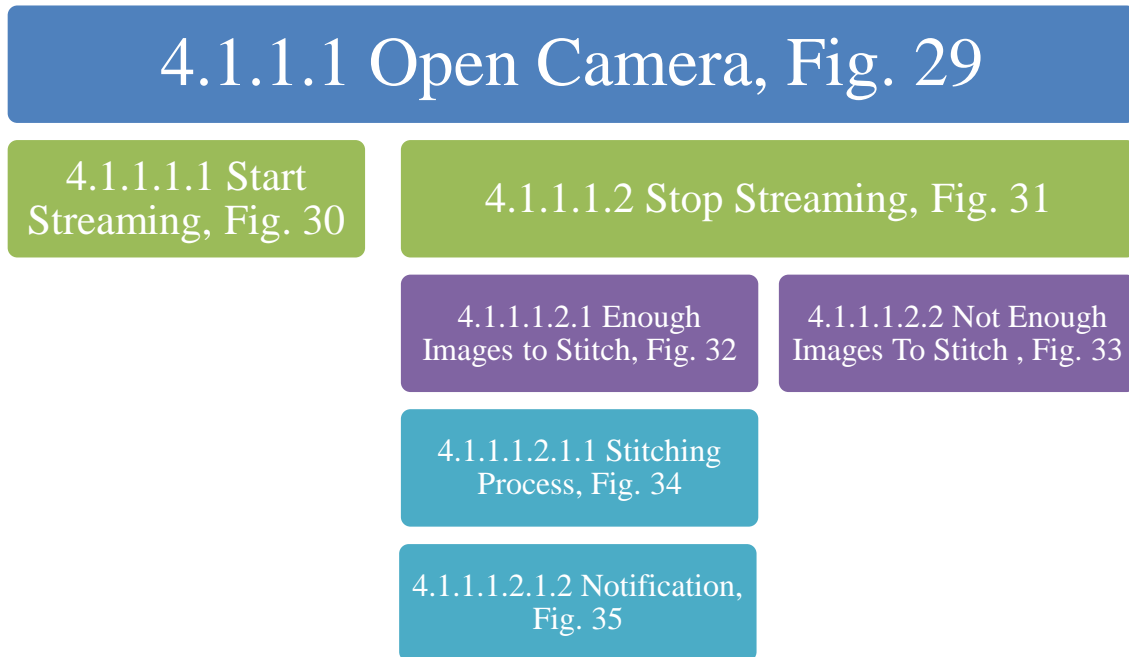


Figure 26

The context hierarchy on *Figure 27* is produced when managing the application configuration.

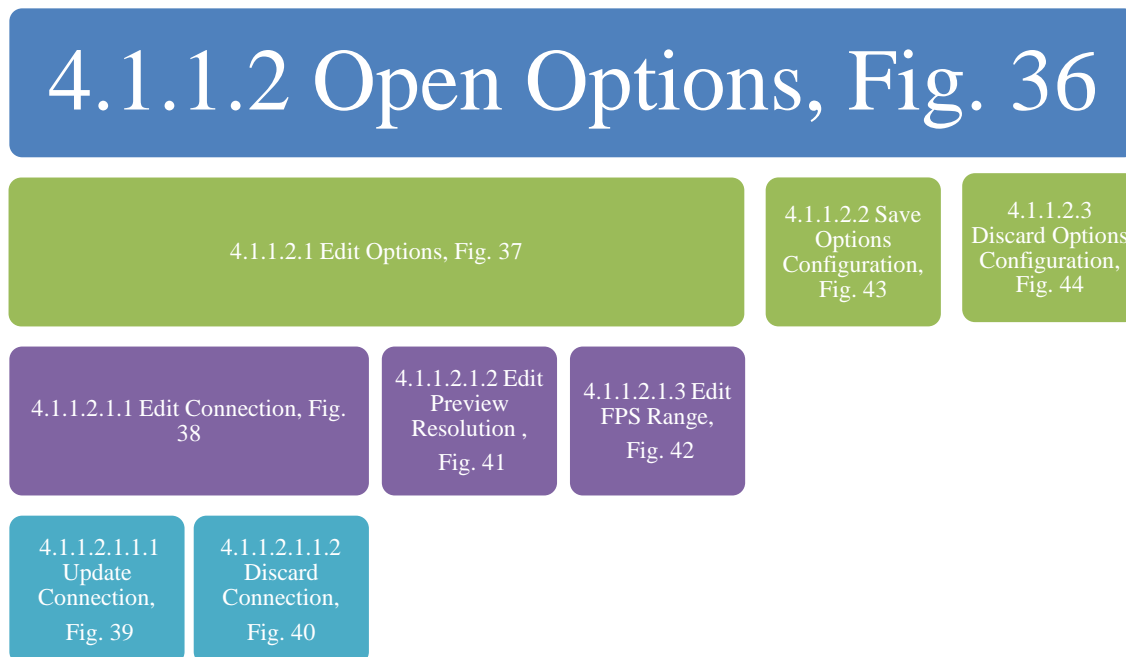


Figure 27

Figure 28 represents the context hierarchy available when exploring the gallery.

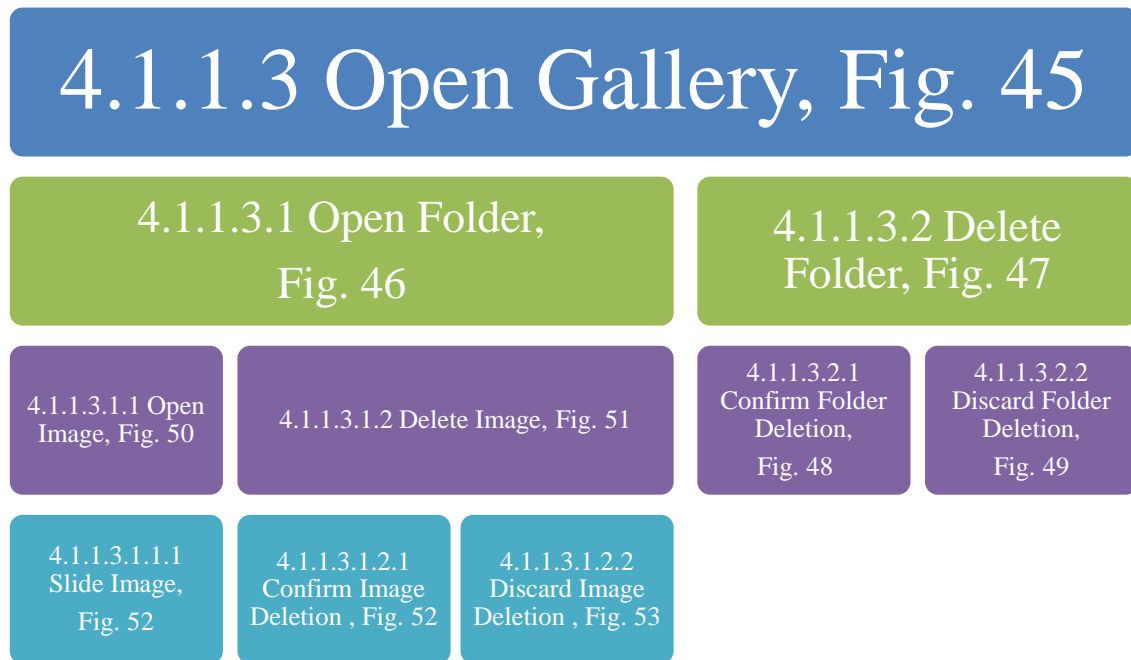


Figure 28

4.1.1.1 Open Camera

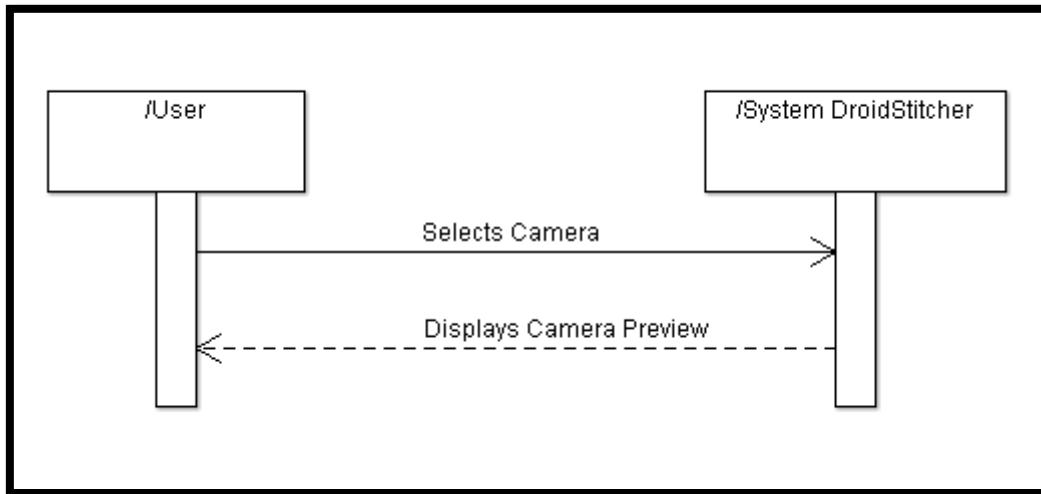


Figure 29

On this case the user has selected the option “Camera” on the application main menu.

4.1.1.1.1 Start Streaming Session

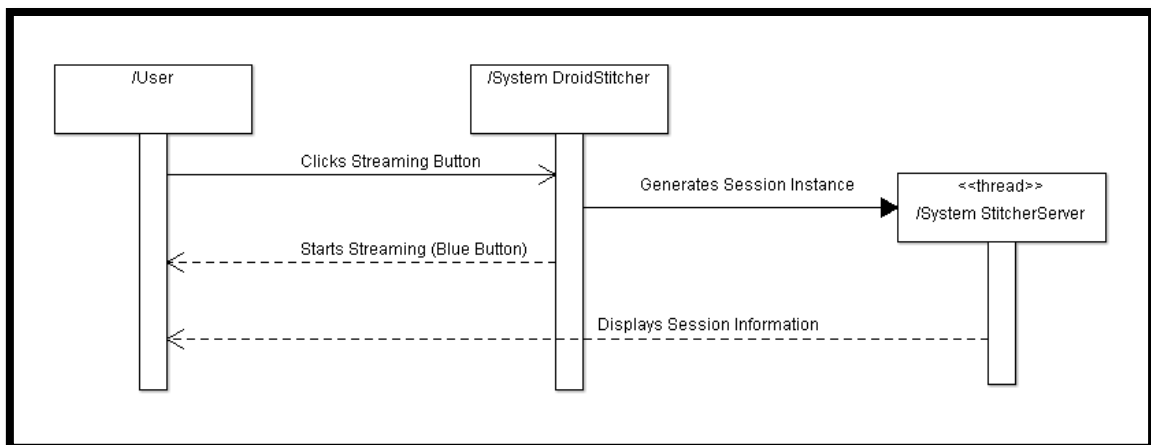


Figure 30

On this context the user clicks on the streaming button. When doing this, a new instance of the server is launched, and the Android application starts streaming preview frames. The user can also check on the computer screen the session information, as well as the images captured by the device.

4.1.1.1.2 Stop Streaming Session

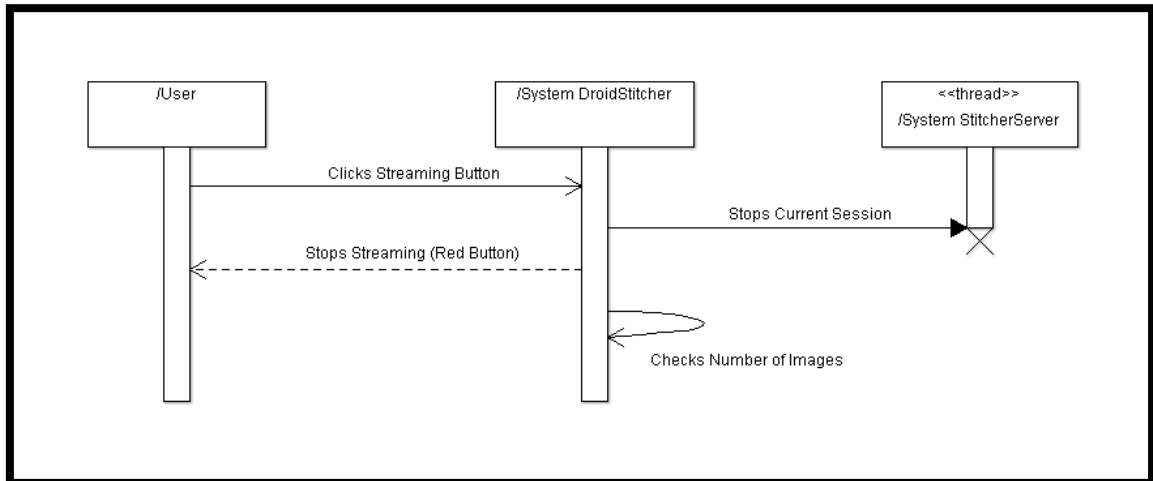


Figure 31

When the user clicks the streaming button, the session is closed. When this happens, two things can occur: have enough images to launch a stitching process on the server or not.

4.1.1.1.2.1 Enough Images to Stitch

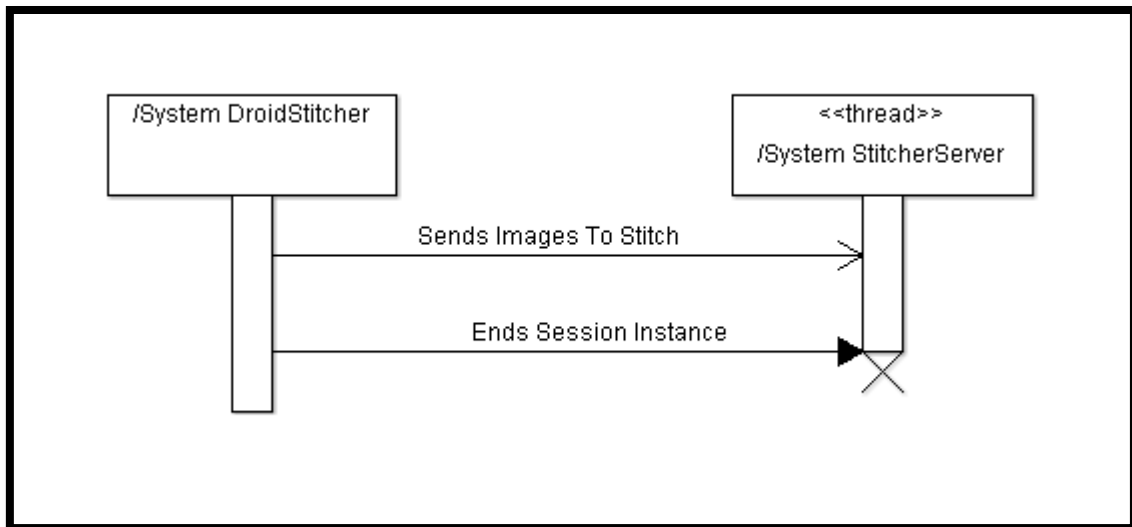


Figure 32

In this case the device has captured enough images to start the stitching process. The images are sent to the server. Then the current session is finished.

4.1.1.1.2.2 Not Enough Images to Stitch

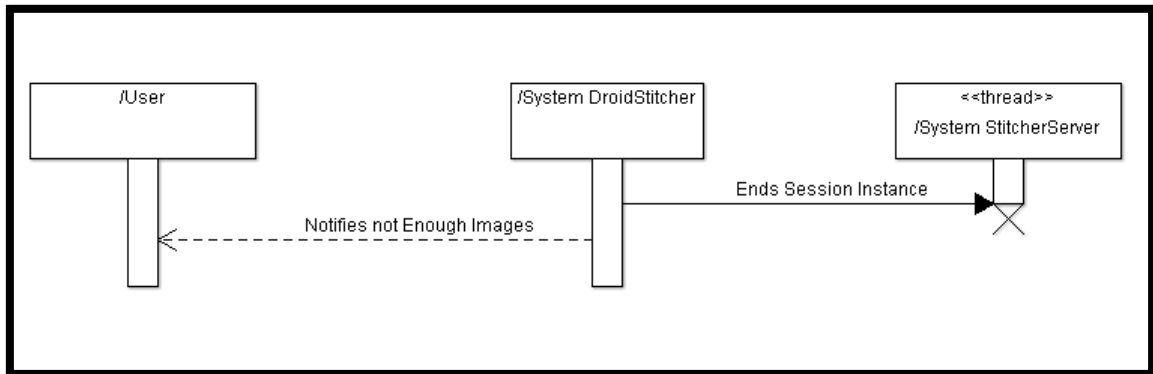


Figure 33

Otherwise, the session is finished and the user receives a message telling that there are not enough images to start the stitching process.

4.1.1.1.2.1.1 Stitching Session

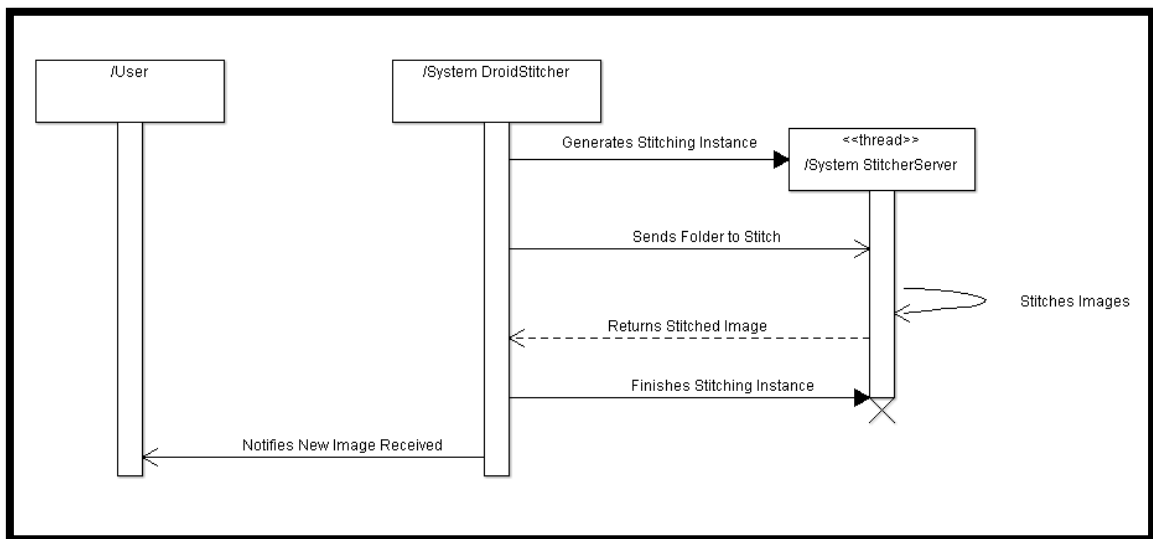


Figure 34

If the minimum number of images is achieved, a new stitching instance is created. The stitching result is returned to the Android application and the user receives a notification when this occurs.

The image could have been correctly stitched or not. In both cases the notification is received, the only thing that differs between a correct result and an incorrect result is the notification message.

4.1.1.1.2.1.2 Notification

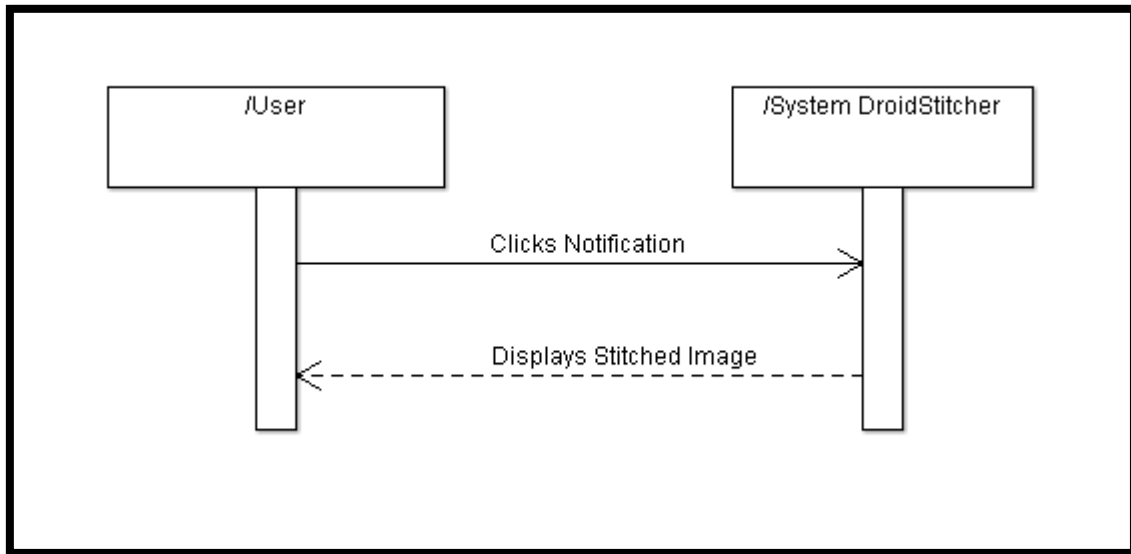


Figure 35

When the user clicks on the notification, the stitched image is displayed. If the image is null, which means the stitching process has not been successful; a “no image” icon will be displayed when clicking the notification instead of the stitched image.

4.1.1.2 Open Options

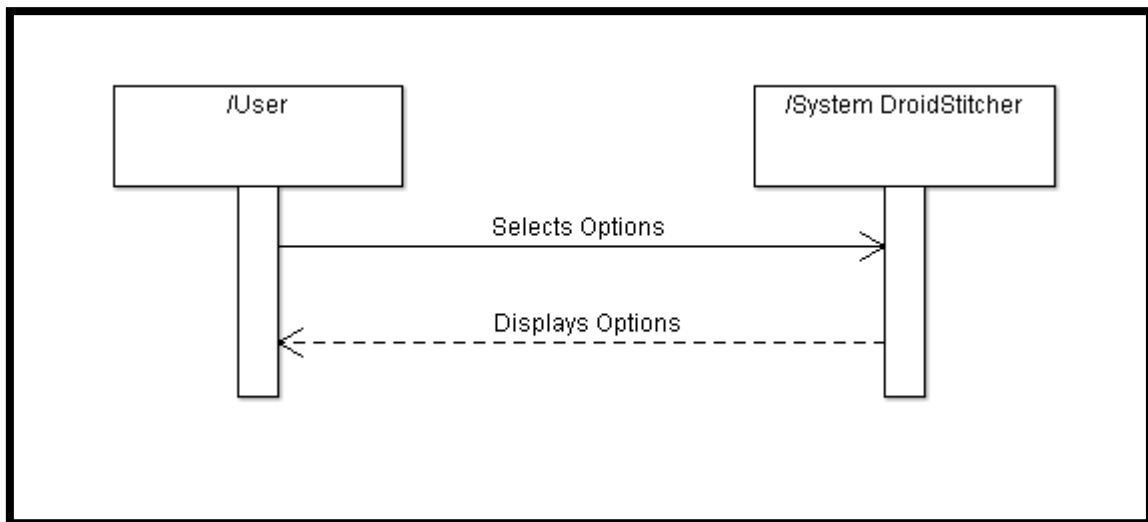


Figure 36

On this case the user has selected the option “Options” on the application main menu.

4.1.1.2.1 Edit Options

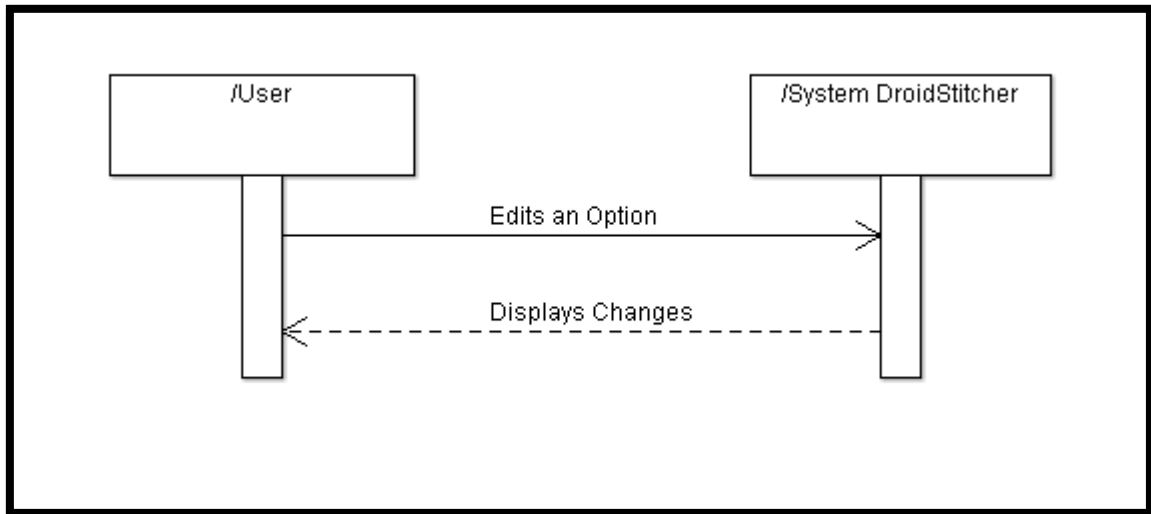


Figure 37

Every time the user changes an option value it is visible on the Options Menu.

4.1.1.2.1.1 Edit Connection

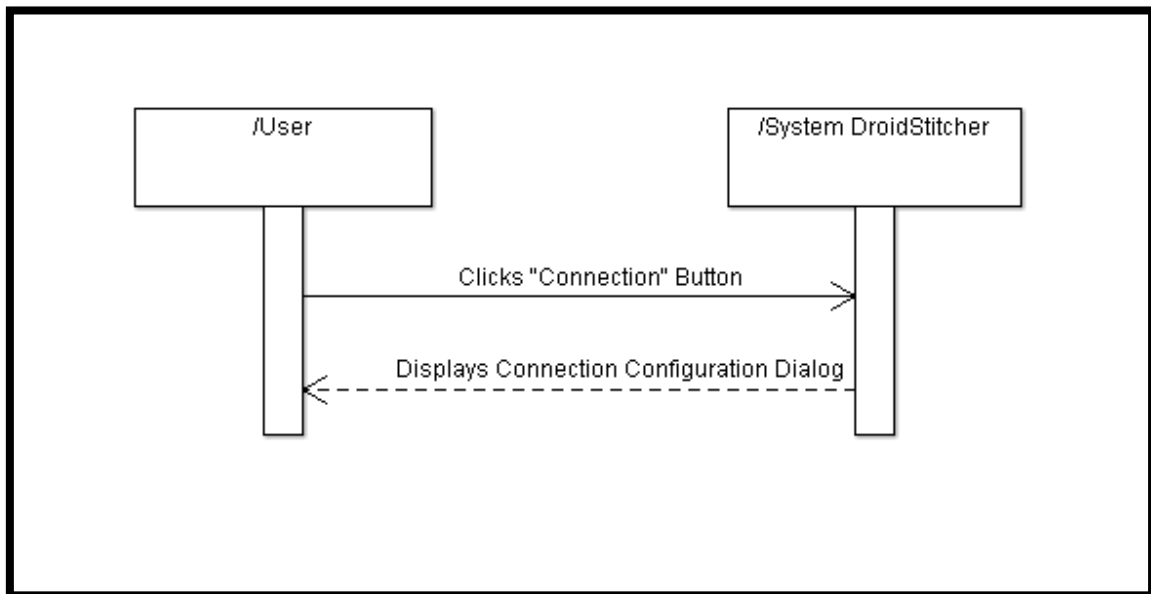


Figure 38

When the user clicks on the “Connection” button, represented by an IP and a port (XXX.XXX.XXX.XXX:XXXXX), a dialog is shown.

4.1.1.2.1.1.2 Update Connection Configuration

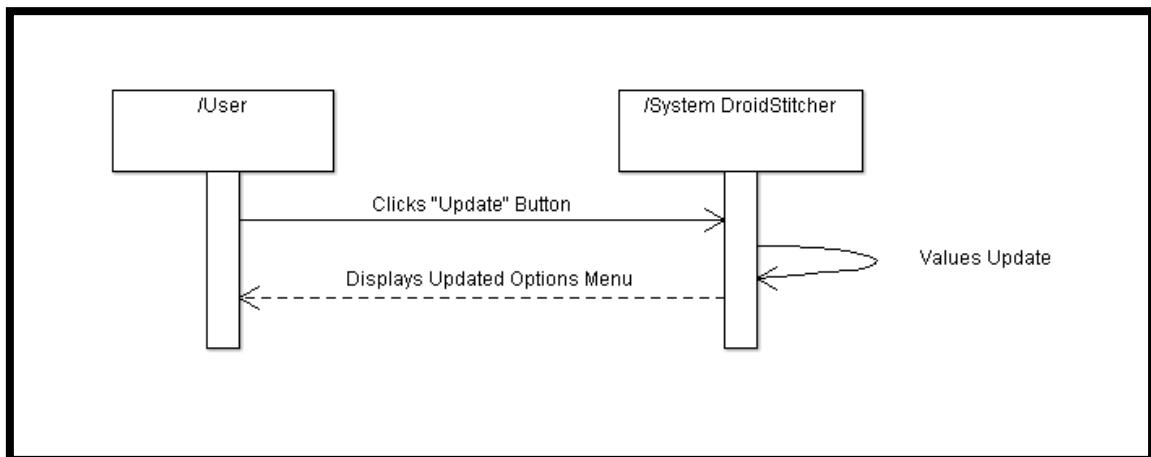


Figure 39

If the user adds a new connection configuration, it is stored and the updated configuration displayed on the Options menu.

4.1.1.2.1.1.3 Discard Connection Configuration

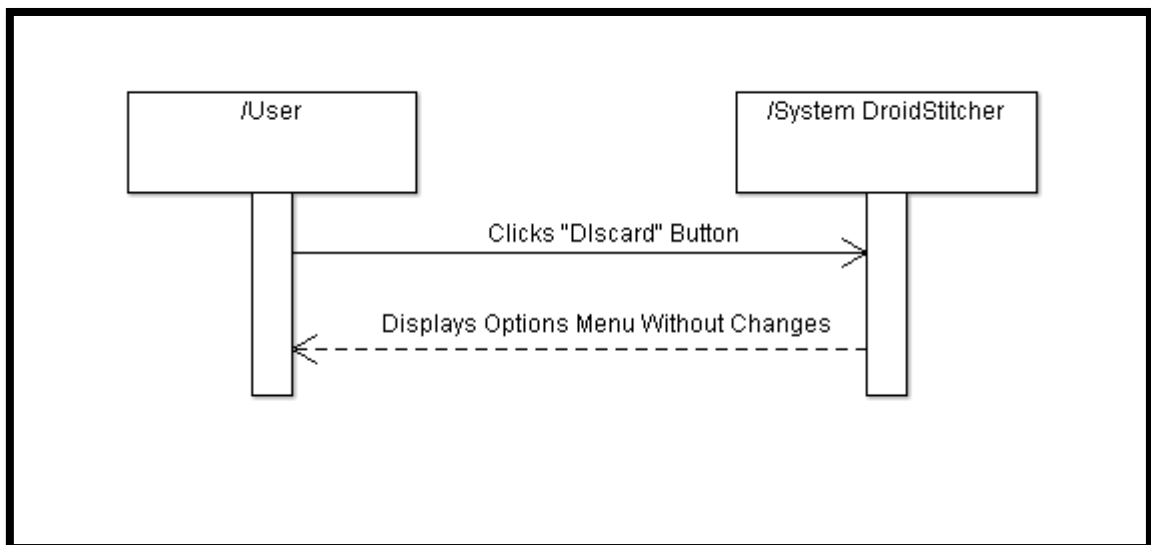


Figure 40

The user can also exit without making any change.

4.1.1.2.1.2 Selecting Preview Resolution

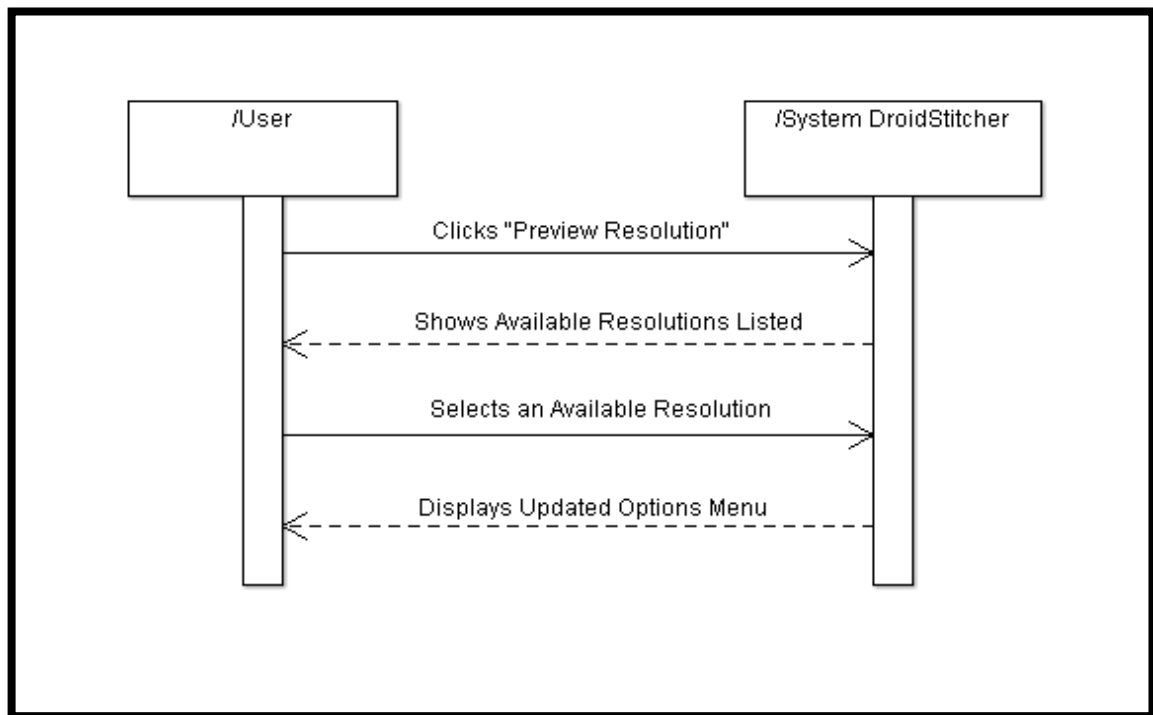


Figure 41

When selecting a new Preview Resolution, the user must choose one from the given list.

4.1.1.2.1.3 Selecting FPS Range

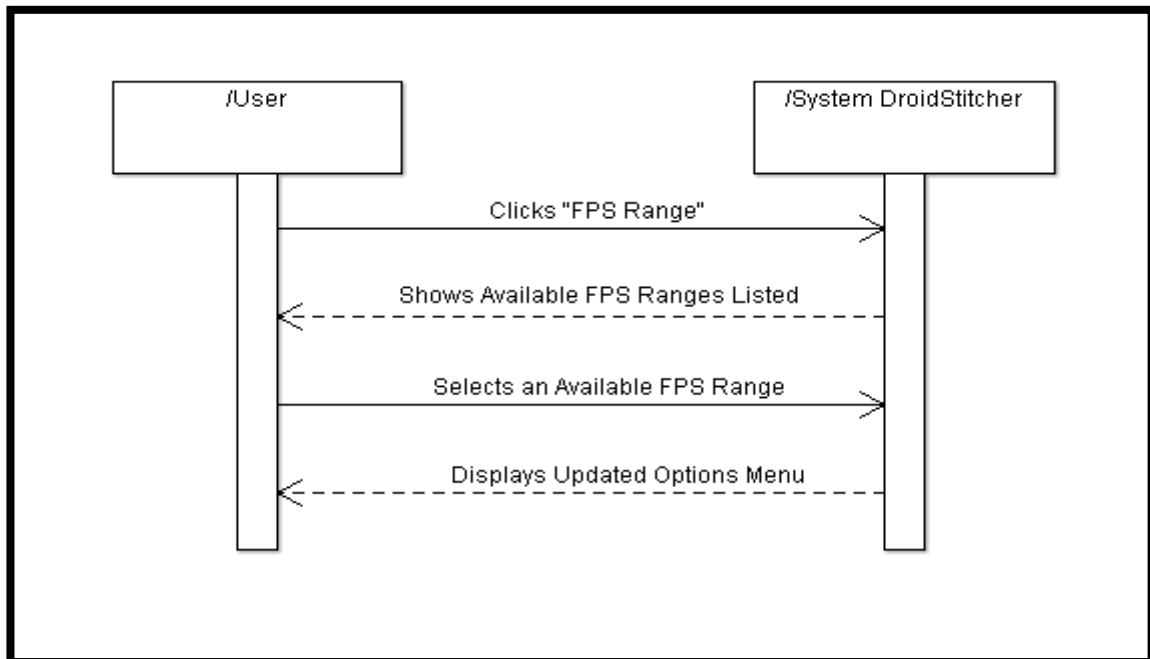


Figure 42

The user has to choose one of the available FPS Ranges from the given list.

4.1.1.2.2 Save Options Configuration

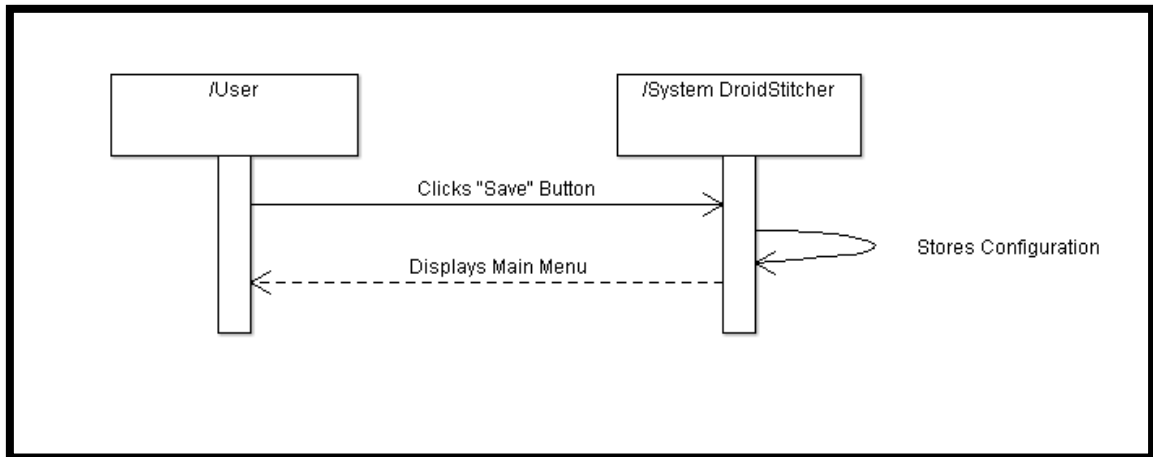


Figure 43

Once the application has been properly configured, the changes can be stored.

4.1.1.2.3 Discard Options Configuration

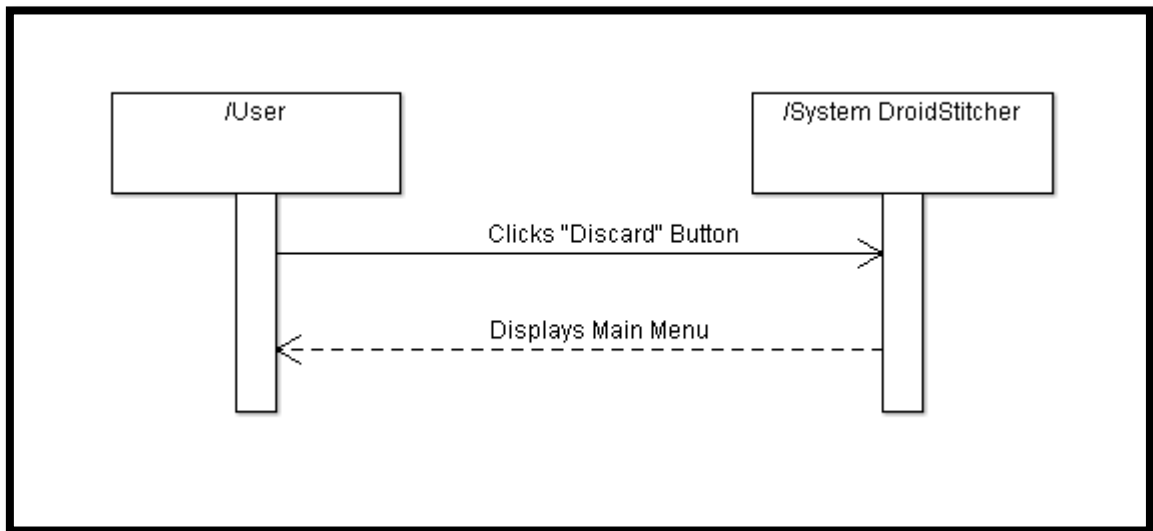


Figure 44

The user can also exit without saving changes.

4.1.1.3 Open Gallery

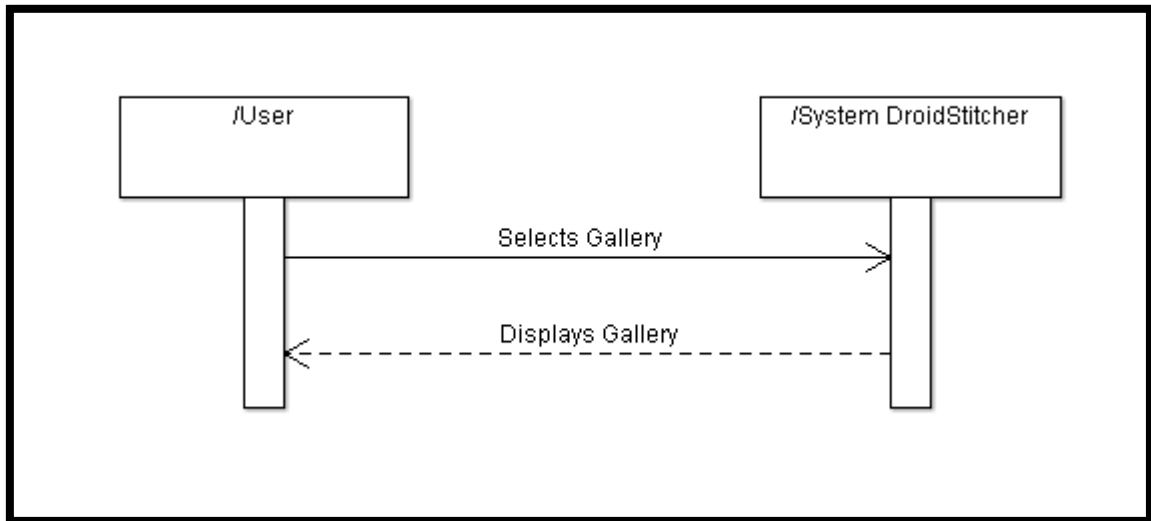


Figure 45

On this case, the user has clicked “Gallery” on the application main menu. The application root folder is listed.

4.1.1.3.1 Open Folder

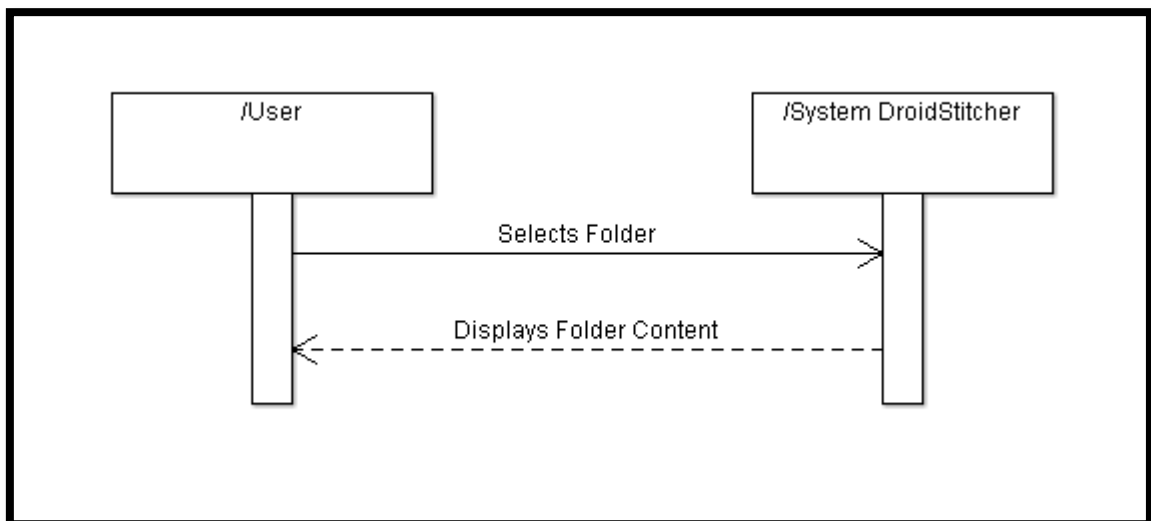


Figure 46

The user can open a listed folder by short clicking. The folder content will be displayed on the screen.

4.1.1.3.2 Delete Folder

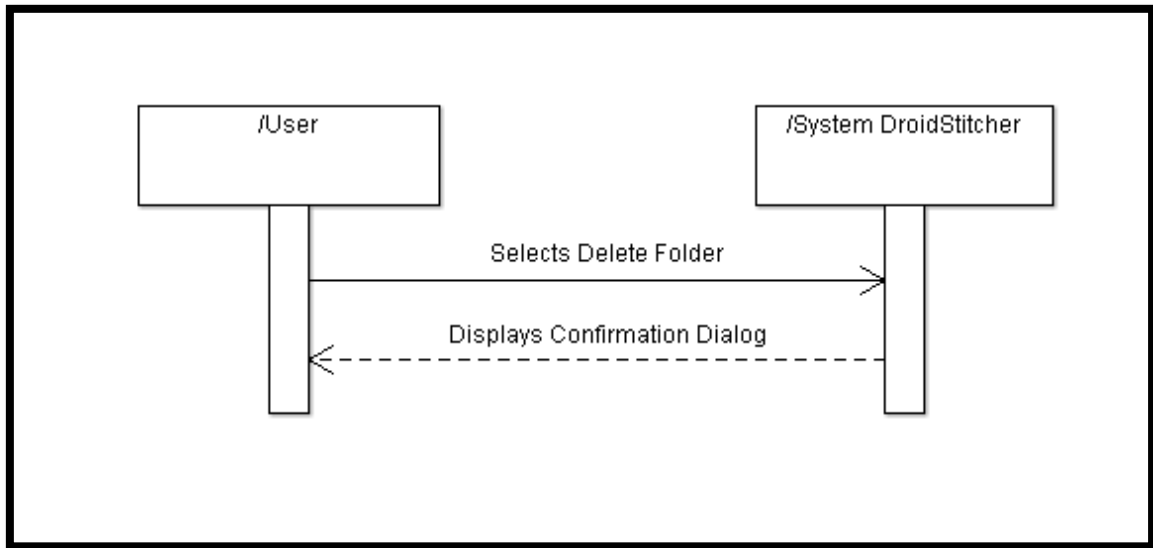


Figure 47

The user can also delete the folder by long clicking over it. A confirmation dialog will appear in this case.

4.1.1.3.2.1 Confirm Folder Deletion

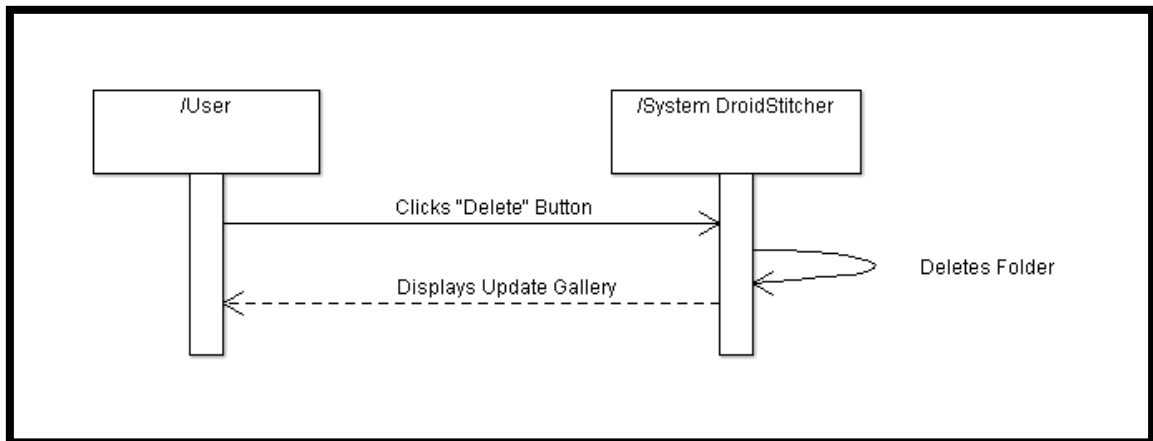


Figure 48

If the user wants to delete the folder, it is necessary to confirm.

4.1.1.3.2.2 Discard Folder Deletion

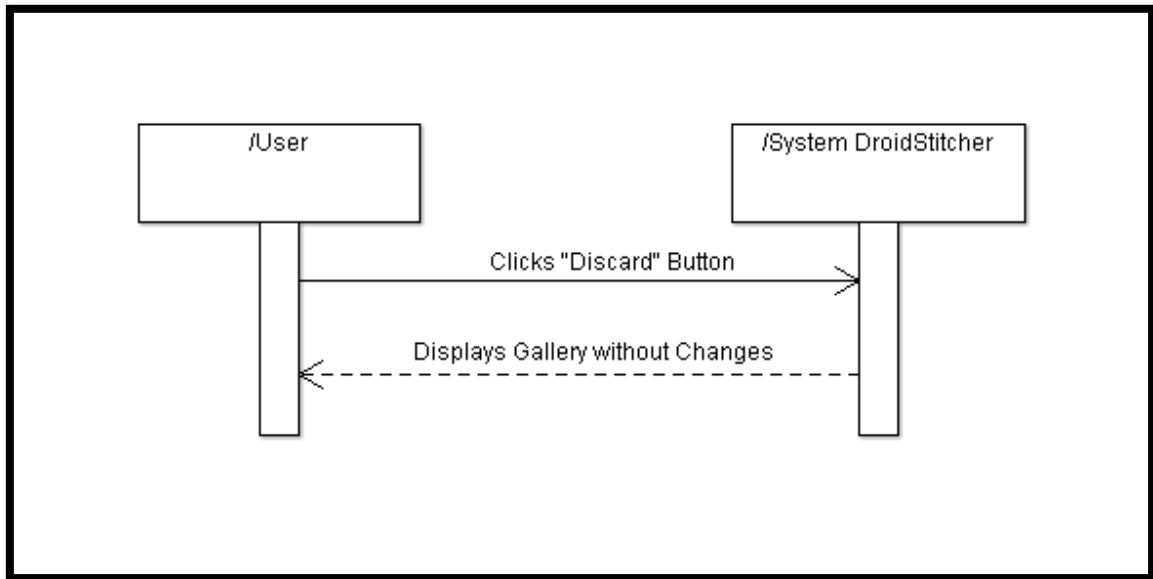


Figure 49

Otherwise, the folder is not deleted.

4.1.1.3.1 Open Image

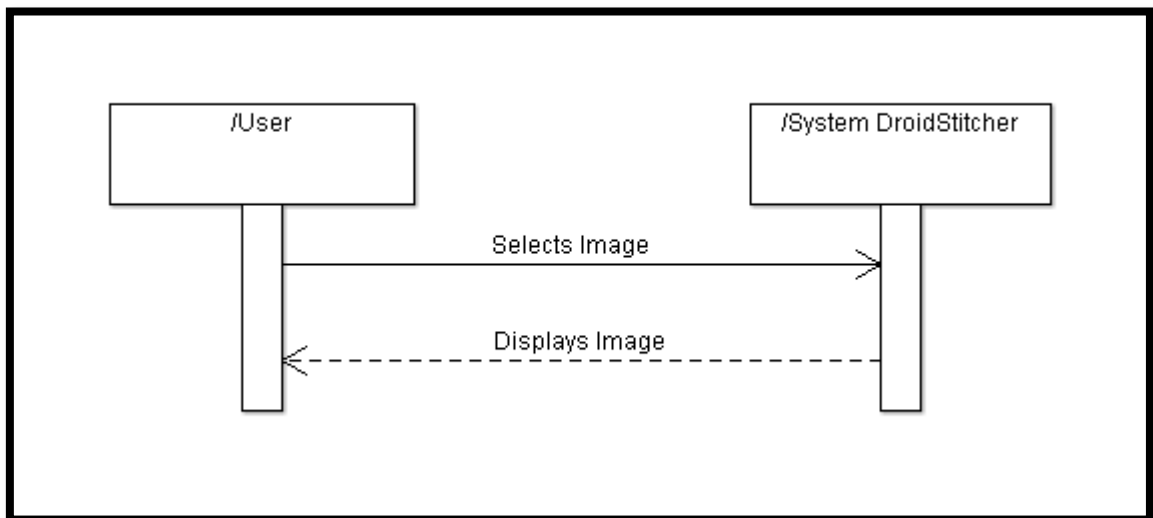


Figure 50

By a short click over an image, it is displayed on full screen.

4.1.1.3.2 Delete Image

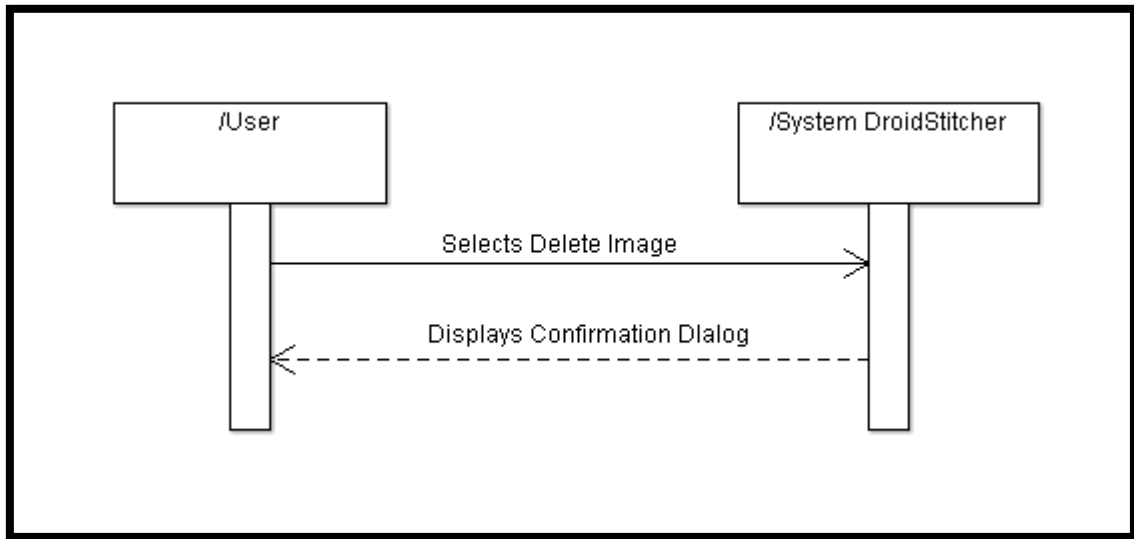


Figure 51

The images can also be deleted by a long click.

4.1.1.3.2.1 Confirm Image Deletion

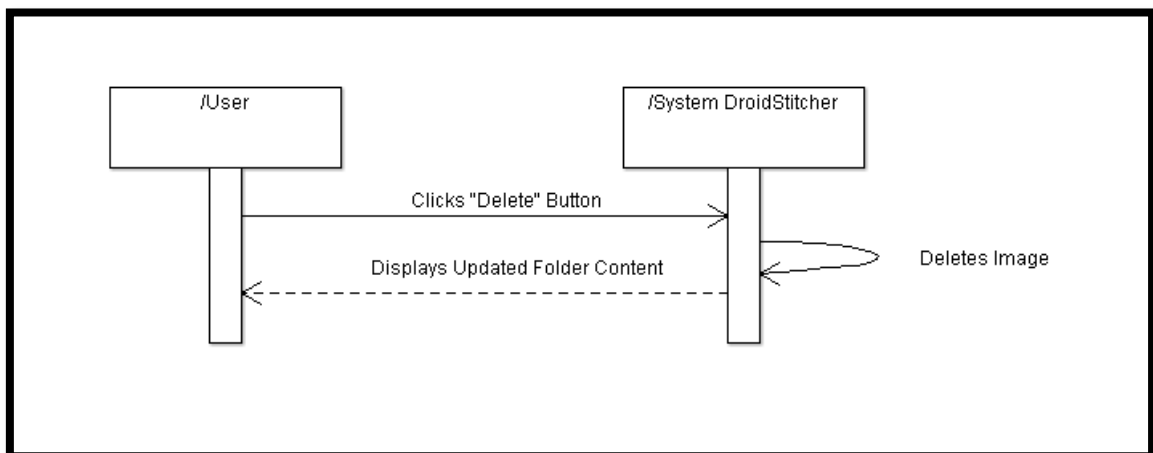


Figure 52

Confirmation is also needed in this case.

4.1.1.3.2.2 Discard Image Deletion

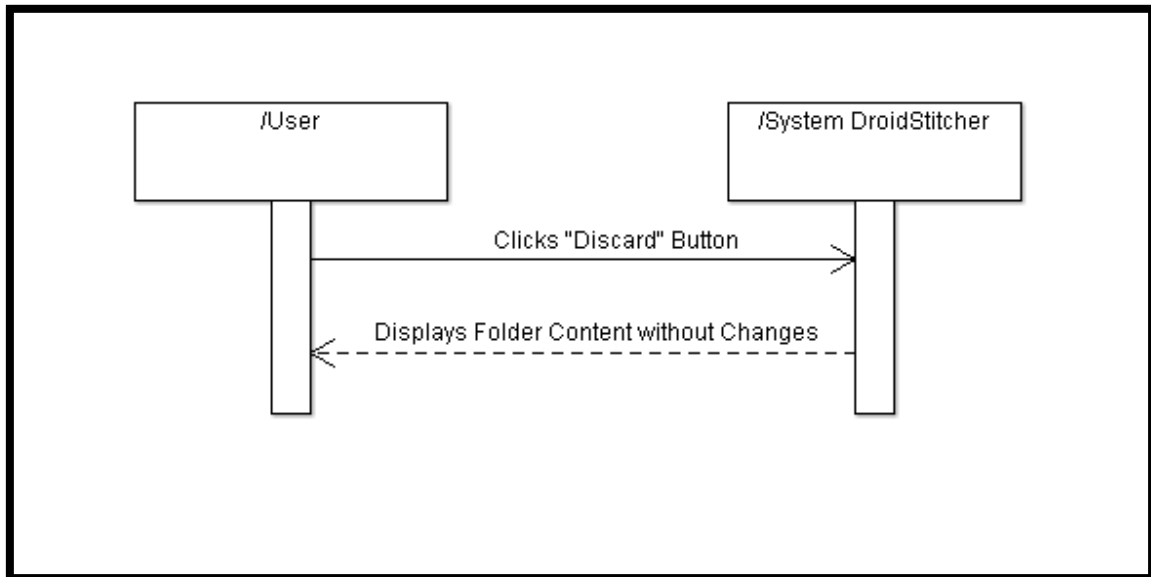


Figure 53

It is also possible to not delete the image.

4.1.1.3.1.1 Slide Image

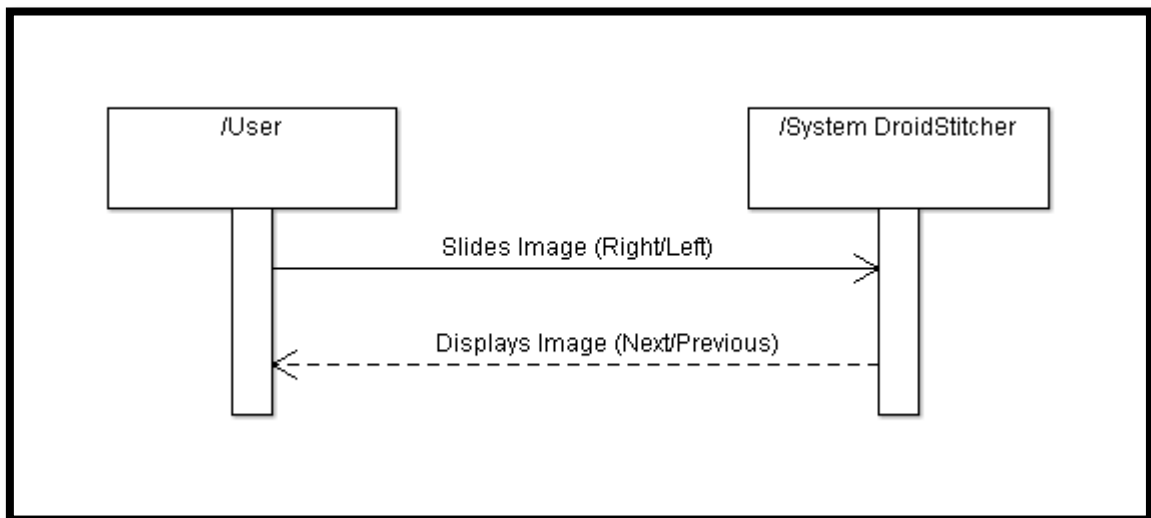


Figure 54

When the user has opened an image, it is easy to open the next or the previous one just sliding.

4.1.2 Server Application Context Diagrams

Due to the low interaction with the server application, two diagrams can explain the possible contexts.

4.1.2.1 Launching the Server Application

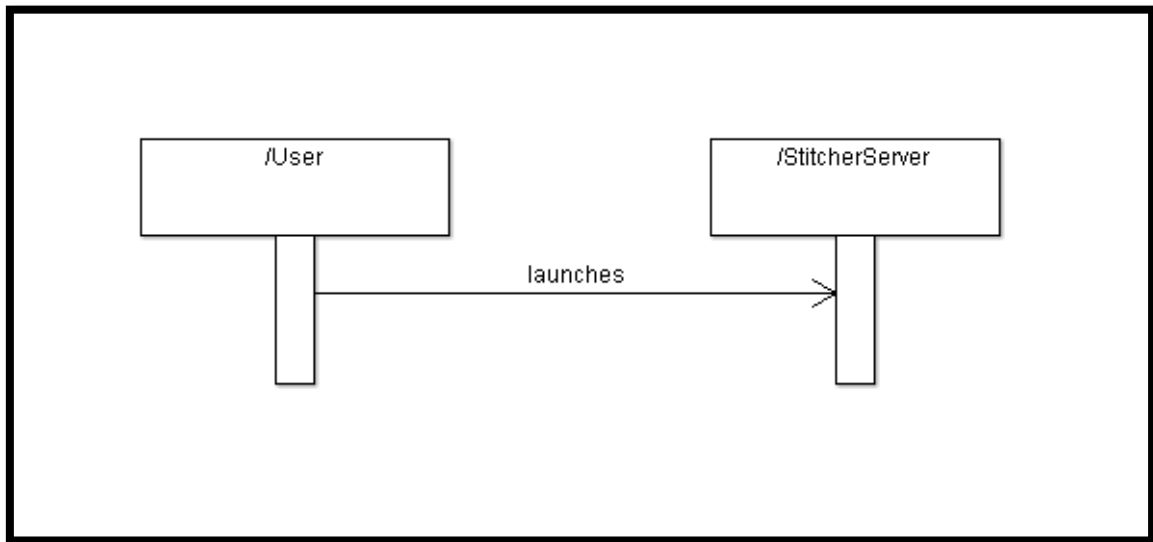


Figure 55

When launching the server, the user has nothing to configure.

4.1.2.2 Checking a new Streaming Session

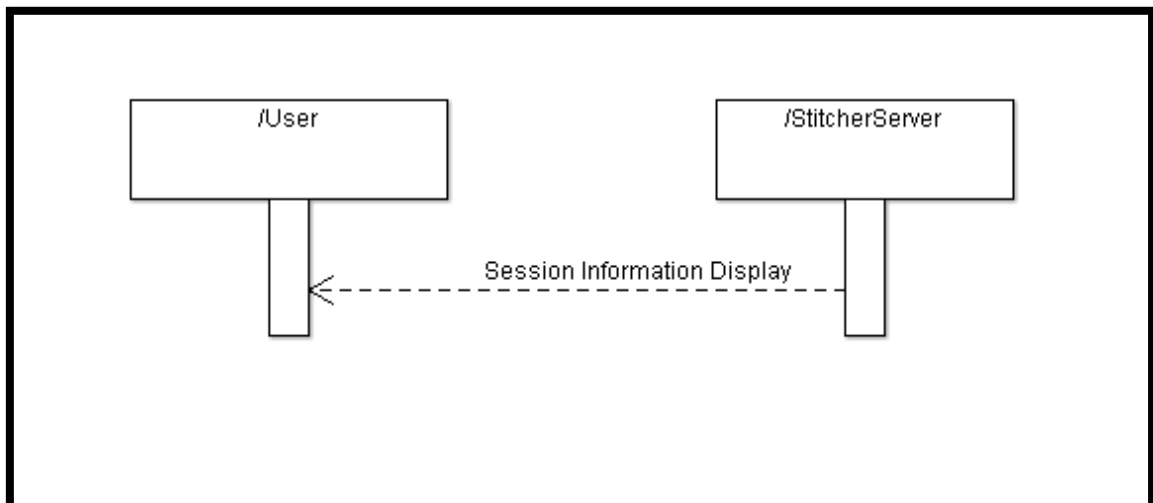


Figure 56

When a new **Streaming Session** is launched, the user can check the information received on the session display.

4.2 Flowcharts

In this subchapter are represented the most relevant workflows by flowcharts. The signs presented on the workflows are illustrated on *Table 1*.




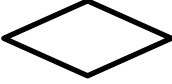


Sign	Description
	Visible change for the user.
	Requires user interaction.
	Instruction.
	Condition.
	Send Information Instruction.
	Long Wait.

Table 1

At this point it is necessary to explain which the session behavior on the application is. There are two types of sessions:

- **Streaming Session:** This session is launched by the user. It starts when the user clicks the streaming button and it finishes when the images for stitch are completely uploaded to the server.
- **Stitching Session:** This session is started by an Android Service just after the Streaming Session has finished. Its creation and destruction are hide to the user.

4.2.1 Android Application Flowcharts

Due to the Android Activities, which mainly work is related with user interaction; these chapter flowcharts represent the application parts which have a minimum user interaction.

4.2.1.1 Streaming Session

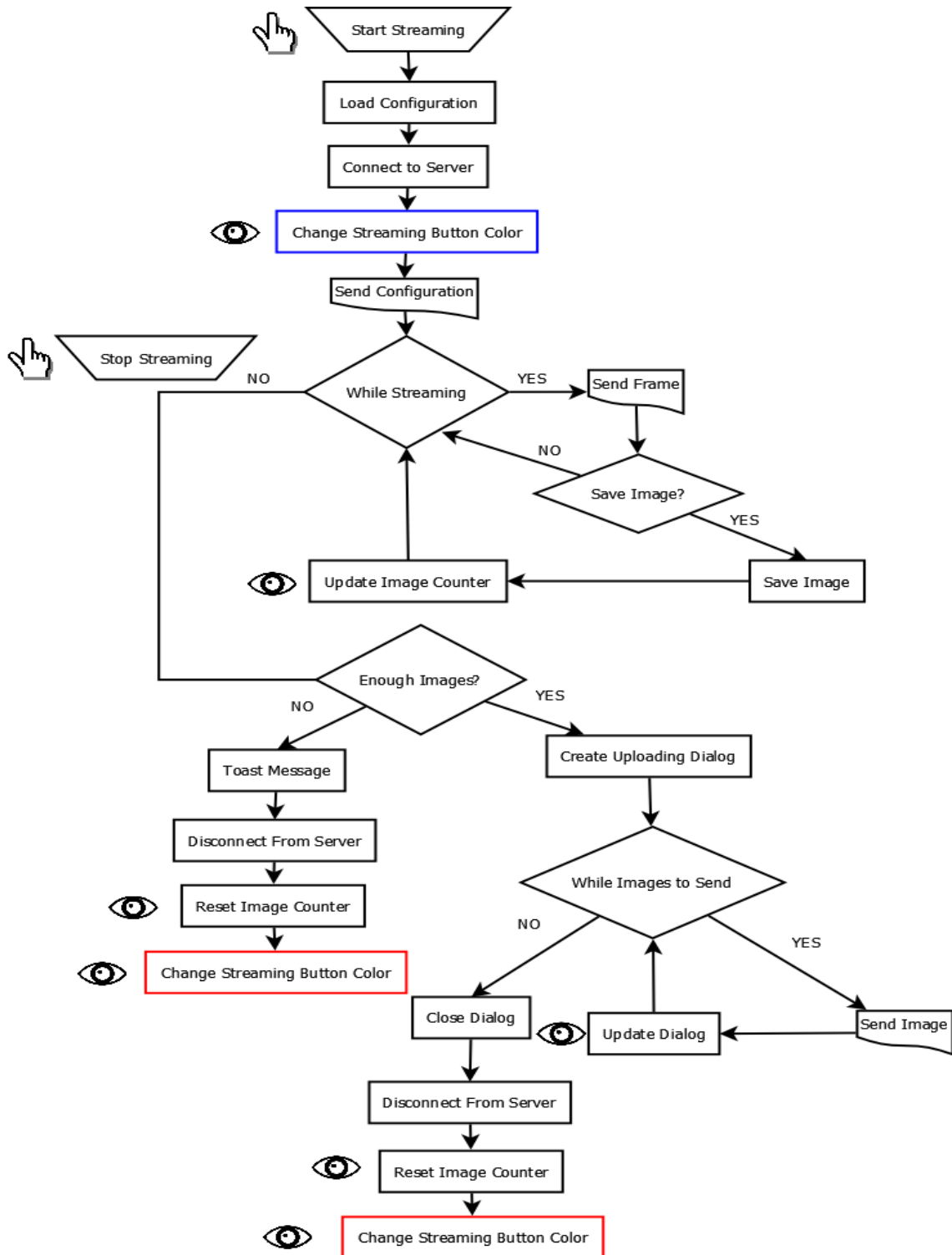


Figure 57

Figure 57 description:

Once the user has clicked the Streaming Button (*Start Streaming*) the applications load the necessary data from the stored configuration, previously modified on the Options menu.

The next step is to establish a connection with the server. If no errors are produced on this step, the Streaming Button will change its color from red to blue, so the user can check the Streaming Session has already started. Otherwise the button will not change its color and an error message will be displayed, so the user will know what is exactly producing the error and not making possible the connection.

After the connection establishment, necessary data is sent to the server, in order to configure the **Streaming Session** properly.

While the Streaming Session is running, all the frames taken by the camera smartphone are sent to the server, in order to display them on live-streaming. The Android application saves some of these frames (or takes a picture) while the Streaming Session is running. These images will be sent to the server in order to stitch them.

When the Streaming Session is finished, due to the user has clicked on the Streaming Button or has pressed the Back key on the smartphone, the stored images can be sent to the server.

The first thing to check on this case is the number of images taken and stored on the device. If the device has not stored a minimum number of images (at least two images), the stitching cannot be done, so the images are not sent and the Android application displays a message telling the number of images is too low.

If the number of images taken is high enough, then the images are sent to the server. This process is displayed to the user on a dialog which contains an uploading bar.

When the sending images process is completed, the connection with the server is closed. The **Streaming Session** finishes in order starting a **Stitching Session**.

4.2.1.2 Stitching Session

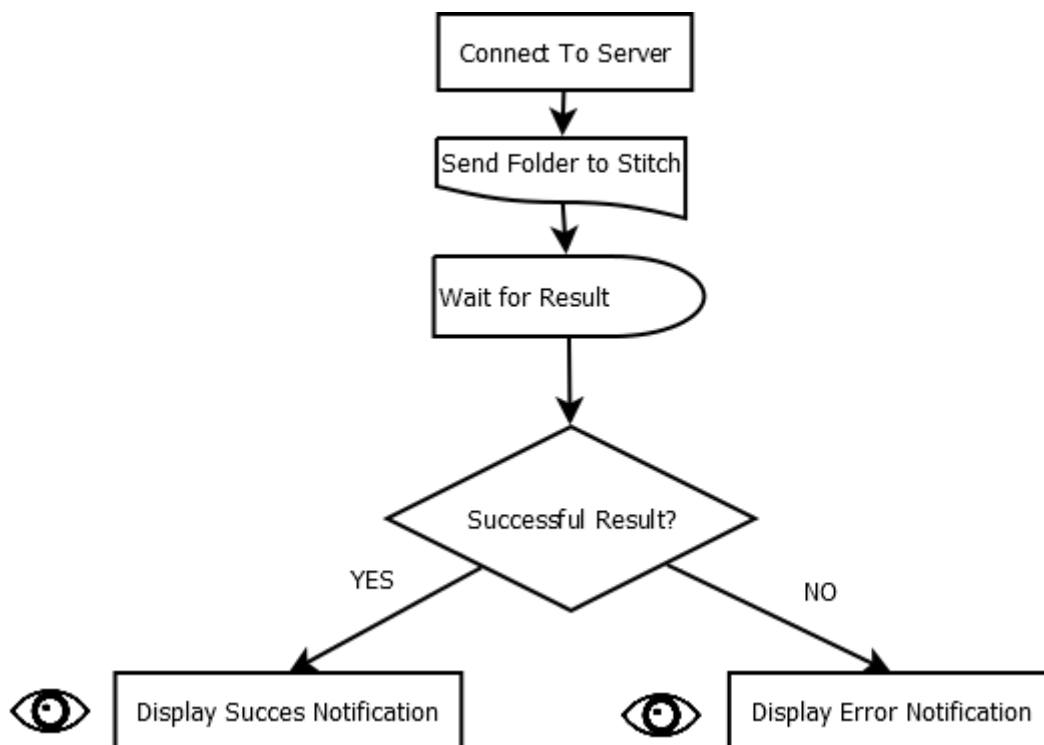


Figure 58

During the stitching session the user has no interaction with the application. Once the Streaming Session has finished, an Android Service is launched, which opens a new connection with the server. The Service asks for a folder to stitch, which corresponds to the path where the images have been stored.

The folder contents are processed, and a MATLAB instance is created to stitch the images. If the result of the stitching is properly produced, a Success Notification is displayed on the user device. The final product is shown if the Notification is clicked. It is also stored on the Session folder.

Otherwise, an Error Notification is displayed, so the user knows the images have not been correctly stitched.

4.2.2 Server Application Flowcharts

The flowcharts presented on this subchapter correspond to the Server Application processes. The user has no interaction on these cases.

4.2.2.1 Streaming Session

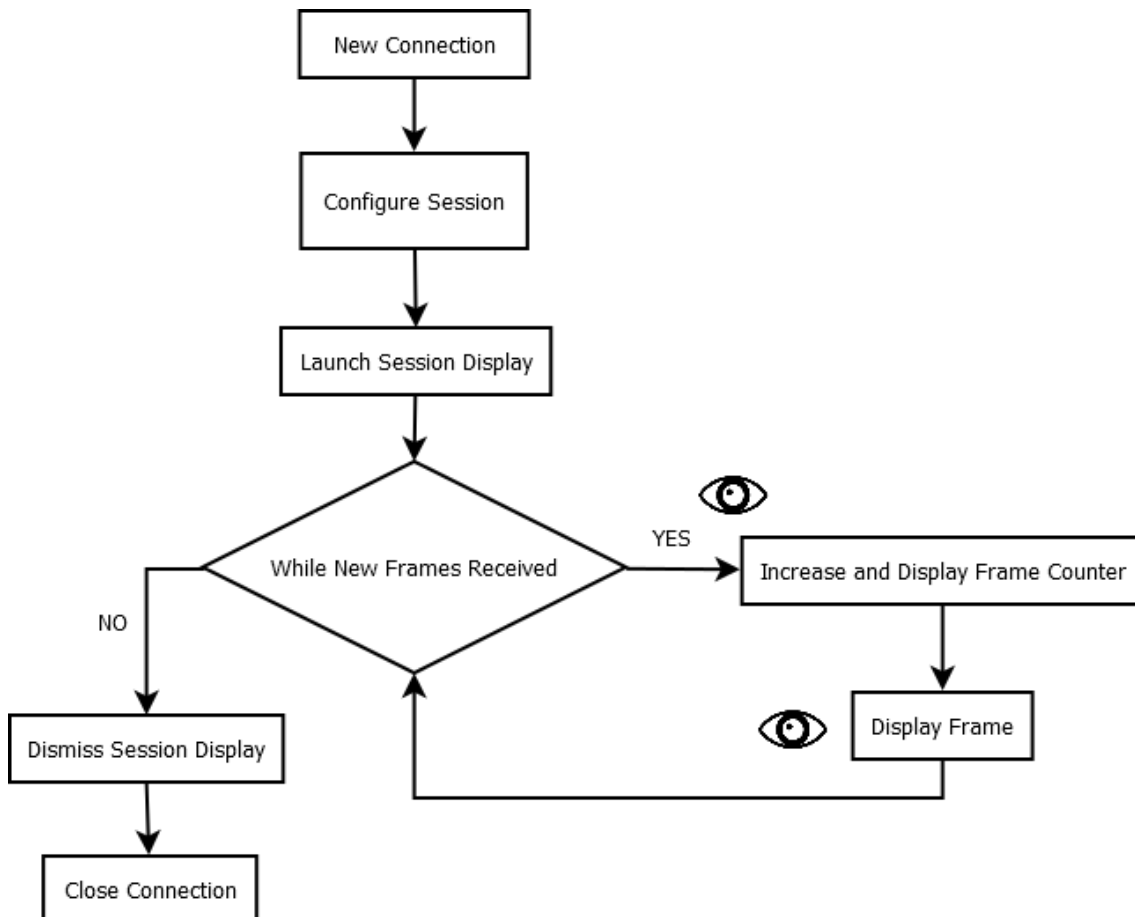


Figure 59

When a device establishes server connection, a new thread is launched and configured to the concrete session requirements. On the Streaming Session, a display is launched, in order to show relevant information from the device. While the connection is up, frames are constantly received and displayed on the screen, producing the live-streaming effect.

When the session is dismissed from the smartphone, the display is closed and also the connection.

4.2.2.2 Stitching Session

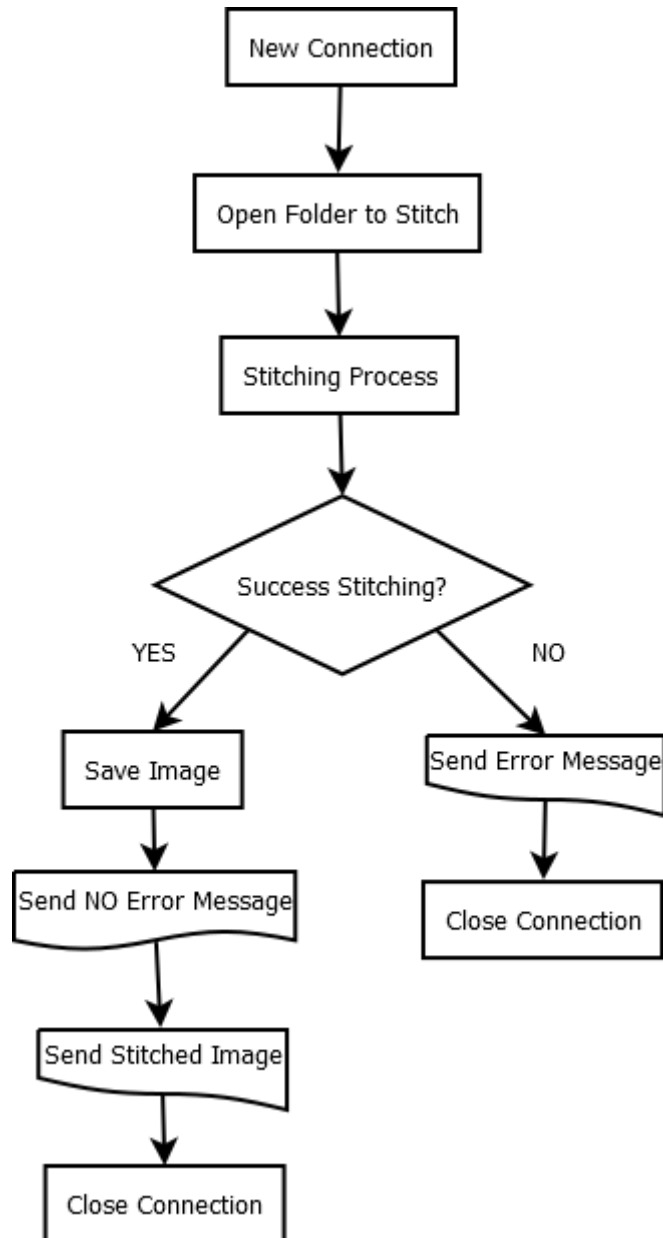


Figure 60

On the Stitching Session no display is shown. Once the necessary information is received, a MATLAB stitching process is launched.

Depending on the operation result, a different notification is sent to the device. If success, it is also sent the stitched image. This image is also stored on the server.

4.3 Class Diagram

The figures on this subchapter illustrate the code classes and their relation among the implementation. Due to the amount of methods and attributes that each class possesses, this data has not been represented on the diagrams in order to provide a clear viewing. Some classes such TextWatchers, Comparators or AsyncTask which deployment is not required for the application understanding have been deliberately omitted. Inheritance to internal Android classes has been omitted too.

4.3.1 DroidStitcher Class Diagram

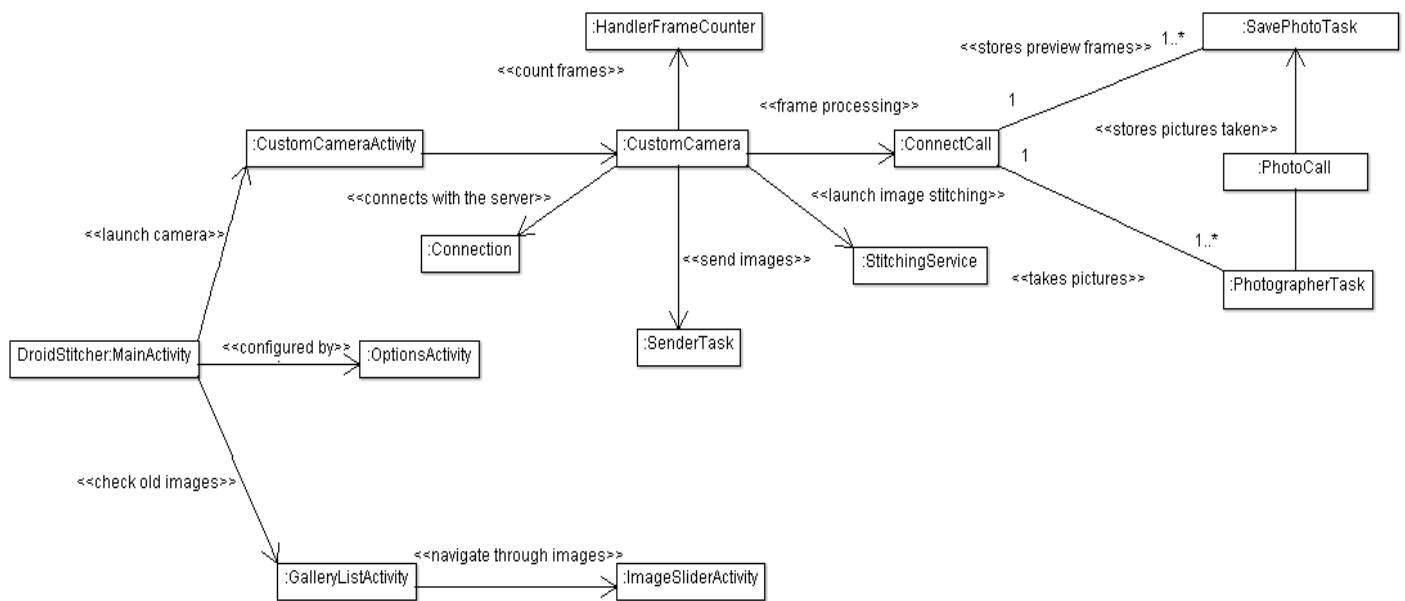


Figure 61

Figure 61 illustrates the relation between DroidStitcher Classes. The name of each class depends on its role. The names finished with “Activity” represent activities, the word “Task” is for the AsyncTask (threads) and the word “Call” represents callbacks.

The diagram starts from a simply root, which is the application main activity and extends its branches to other activities.

While OptionsActivity does not have any relevant interaction with other classes, CustomCameraActivity does. GalleryListActivity rules an ImageSliderActivity, which allows slide through images.

CustomCameraActivity generates CustomCamera. This class is responsible of the preview holder, which allows checking camera previews. This class also generates the connection with the host, managed by Connection class, the images taken counter, which is HandlerFramCounter, the sending images to server process, which is SenderTask, the responsible of retrieving the Stitched image, StitchingService, and the callback ConnectCall, which allows sending data to the host.

If the images sent to the server will be preview frames, ConnectCall directly stores them on the device using SavePhotoTask. Otherwise, the images that will be sent are camera pictures, such task will be accomplished by PhotoCall, a callback launched into the PhotographerTask that will save the image just before its shoot.

The diagram represented from CustomCameraActivity till the end corresponds to an only session. For each session just one CustomCamera will be created, just one Connection will be created, etc. The only objects that will be constantly created are the ones focused on taking pictures and saving images.

The diagram shows no relation between StitchingService and Connection, but it is clear that this service also establishes a connection with the host in order to create a Stitching Session. Actually the service accomplish its task, but without the need from other classes. The service implements on its code a new low-level socket configuration, as Connection class does.

The reason why the service starts a new connection is because the Android services run on the application main UI. To avoid that, the only possible way is to make the service run an internal thread. This thread is the one which connects with the host allowing the user exiting from the application without interrupting the connection. Hence it is not necessary the application to be opened.

The ternary relation composed by ConnectCall, PhotographerTask and SavePhotoTask allows sparing not repeating code. The image saving process is the same on both cases.

4.3.2 StitcherServer Class Diagram

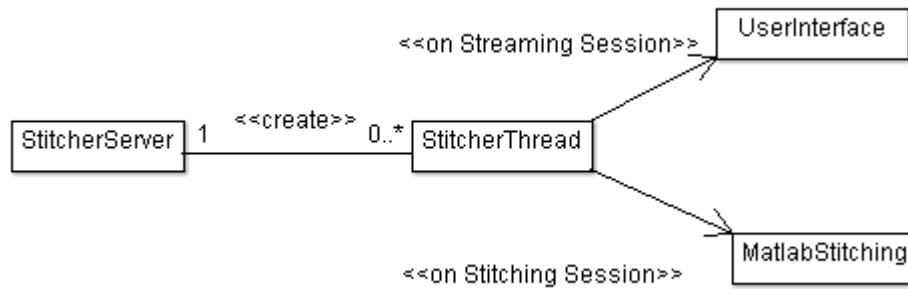


Figure 62

StitcherServer just has four relevant classes which can explain its behavior.

The first class, called `StitcherServer`, is one which is launched when the server starts. Once a device tries to connect with the host, `StitcherServer` generates a new thread which manages that concrete connection entirely.

At this point, there are two types of connections. The Streaming Sessions launched by `DroidStitcher` itself or the Stitching sessions launched by the `StitchingService`, which has been created at the end of the Streaming Session.

Depending on the session kind, established by the client, the host generates a `UserInterface`, which provides information about the connected device and the live-streaming video, or the `MatlabStitching`, a static class which allows opening a MATLAB instance and work on stitching.

`StitcherServer` allows multithreading, so several Sessions can be running at the same time. For example, if two devices are streaming images at the same time, two graphical interfaces will be shown on the server screen.

The `MatlabStitching` class is synchronized in order to avoid multiple services asking for a stitching process, the reasons for doing this are two:

- The stitching process consumes many resources from the computer. This fact could produce a collapse.
- The MATLAB stitching code generates local files. This data are images generated while stitching, files containing key points, etc. If two Stitching Sessions are running at the same time, these files can be overwritten and the final product could be erroneous.

4.4 Class Specifications

On this subchapter are explained all the classes belonging to both applications. Attributes and methods are included on every figure, and the most relevant are included on the class description.

4.4.1 DroidStitcher Classes

4.4.1.1 MainActivity

com::droidStitcher::DroidStitcher:MainActivity
onCreate(savedInstanceState : Bundle) : void onPause() : void createAppFolder() : boolean setStyle() : boolean launchCustomCameraActivity(view : View) : boolean launchGalleryListActivity(view : View) : void launchOptionsActivity(view : View) : void

Figure 63

This activity is the first created when the application is launched. This activity basically grants access to other activities, each one of them accessible by a button (view) located in its display.

4.4.1.2 CustomCameraActivity (corregir)

com::droidStitcher::CustomCameraActivity
mCamera : Camera mPreview : CustomCamera isStreaming : boolean
onCreate(savedInstanceState : Bundle) : void openCam() : void streamerButtonConfig() : void onBackPressed() : void StreamerButtonConfig()

Figure 64

This activity checks the camera availability, launches its preview and configures the streaming button behavior with listeners.

4.4.1.3 CustomCamera

com::droidStitcher::CustomCamera
<p><u>TAG</u> : String mHolder : SurfaceHolder mCamera : Camera streaming : boolean cntx : Context sname : String con : :Connection oos : ObjectOutputStream ois : ObjectInputStream framestaken : TextView appcntx : Context hfc : :HandlerFrameCounter sbutton : ImageButton connectionStatus : boolean</p>
<p><<create>> CustomCamera(context : Context,appcontext : Context,camera : Camera,sessionName : String) getPreviewResolution(appcontext : Context) : int[] getFrameRate(appcontext : Context) : int[] setFramesCounter(textView : TextView) : void resetFramesCounter() : void surfaceCreated(holder : SurfaceHolder) : void stream(mode : int) : boolean streamingValue() : boolean surfaceDestroyed(holder : SurfaceHolder) : void surfaceChanged(holder : SurfaceHolder,format : int,w : int,h : int) : void connect(user : String) : boolean autoToast(string : String) : void setStreamerButton(streamButton : ImageButton) : void setFramesCounter() setStreamerButton() connect() Verify()</p>

Figure 65

This class extends from Android SurfaceView and implements SurfaceHolder.Callback. These relations and its implementation allows this class to work as a preview holder, displaying on the device screen every frame captured by the camera.

The camera is configured loading the values stored on the Options Activity. The relevance of this class lies in the objects, such the HandleFrameCounter or Connection, which are created and managed from its code.

4.4.1.4 HandleFrameCounter

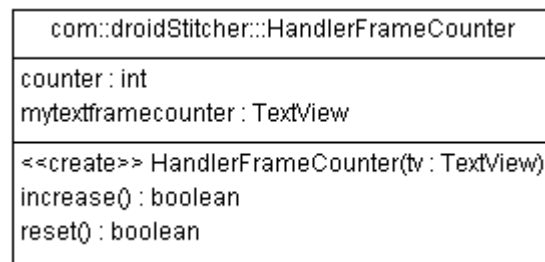


Figure 66

This handler is the responsible of managing a counter which value represents the number of images or frames stored on the Streaming Session.

4.4.1.5 Connection

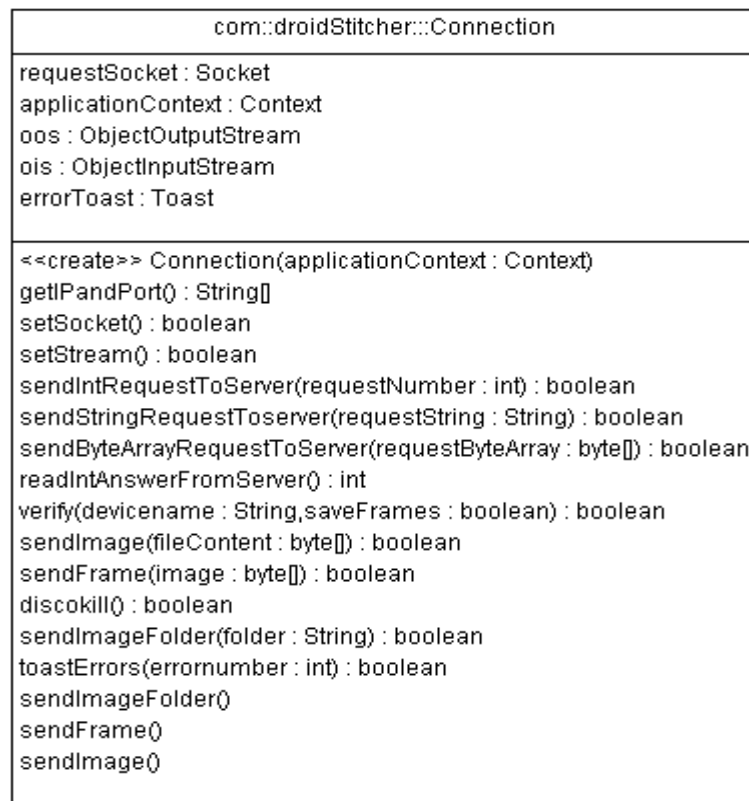


Figure 67

This class manages the connection. Every network operation required by other components like threads or objects must submit their petitions to this class.

Due to its condition its methods are synchronized and the possible network issues are corrected by returning Booleans representing the success of the operation.

The function toastErrors() is run when an error occurs, so the user can check the problem on the device screen.

4.4.1.6 SenderTask

com::droidStitcher::SenderTask
foldernow : String con : :Connection context : Context senderdialog : Dialog interrupted : boolean buttonWifi : ImageButton
<<create>> senderTask(folderbase : String,mycon : :Connection,context : Context,sbutton : ImageButton) doInBackground(params : String) : Boolean onPostExecute(success : Boolean) : void onPreExecute() : void onProgressUpdate(progress : Integer[]) : void

Figure 68

This Android AsyncTask sends the stored images from the device to the computer host. It is also responsible of updating the Upload Dialog which appears when sending those images.

4.4.1.7 Stitching service

com::droidStitcher::StitchingService
folderSaveImage : String foldersplit : String[] modeldevice : String stitchedimage : byte[] requestSocket : Socket oos : ObjectOutputStream ois : ObjectInputStream
onBind(arg0 : Intent) : IBinder stopService() : void onCreate() : void setAndShowNotification(success : boolean) : void onStart(intent : Intent,startid : int) : void

Figure 69

This Android Service asks the host for stitching a concrete folder, corresponding to the previous Streaming Session. It also stores the returned image and notifies its arrival to the user. This service runs regardless of the application status or context.

4.4.1.8 ConnectCall

droidStitcherThreading::ConnectCall
imagecounter : int mycon : Connection takepicture : boolean photocounter : int foldern : String modeimageperframes : int modeHQimages : boolean modepreviewquality : int mydialog : Dialog framelabel : TextView myhfc : HandlerFrameCounter ctx : Context mycca : CustomCamera
<<create>> ConnectCall(cca : CustomCamera, ctx : Context, HQimages : Boolean, imageperframes : int, previewquality : int, hfc : HandlerFrameCounter, con : Connection, folder : String) onPreviewFrame(arg0 : byte[], cam : Camera) : void preparePreviewImage(rawpic : byte[], cam : Camera, compressionquality : int) : byte[] preparePreviewImage()

Figure 70

This Android Callback is responsible of the image post-processing. Once a frame is taken, this callback will change its format, store it if necessary and send it to the server.

4.4.1.9 SavePhotoTask

droidStitcherThreading::SavePhotoTask
folderToSaveTheImage : String imageCounter : int ImageName : String
<<create>> SavePhotoTask(folderToSaveTheImage : String, imageCounter : int) doInBackground(jpeg : byte[]) : String

Figure 71

Whenever a camera picture or preview frame needs to be stored, this AsyncTask will store it on the required folder with the required name.

4.4.1.10 PhotographerTask

droidStitcherThreading::PhotographerTask
mCamera : Camera myfolder : String imagecounter : int
<<create>> PhotographerTask(mCamera : Camera, folderToSaveImage : String, imageCounter : int) onPostExecute(result : Void) : void doInBackground(params : Void) : Void takePicture()

Figure 72

This AsyncTask launches the camera method takePicture(), necessary for obtaining camera pictures.

4.4.1.11 PhotoCall

droidStitcherThreading::PhotoCall
folder : String imagecounter : int mCamera : Camera
<<create>> PhotoCall(mCamera : Camera, folderToSaveImage : String, imageCounter : int) onPictureTaken(data : byte[], camera : Camera) : void

Figure 73

If the images which will be stored are camera pictures and not preview frames, this callback is passed onto the takePicture() method, located on the PhotographerTask. This callback will launch the SavePhotoTask thread in order to save the image.

4.4.1.12 OptionsActivity

com::droidStitcher::OptionsActivity
<pre>onCreate(savedInstanceState : Bundle) : void setStyle() : void seekBarWatchers() : void displayPreviewFPSRangeList(view : View) : void displayPreviewResolutionList(view : View) : void showIPDialog(view : View) : void loadOldIP(dialog : Dialog) : boolean setWatchers(dialog : Dialog) : void updateIP(dialog : Dialog) : void finishOptions(view : View) : void saveOptions(view : View) : void storePreferences() : boolean loadOptions() : boolean</pre>

Figure 74

This Activity is responsible of allowing changing the application settings. It is also responsible of the dialogs which appear when modifying some settings.

The configuration, which is stored and loaded when launching and closing this activity, is stored on the Shared Preferences of the Application Context, allowing this data to be accessible from other activities.

The Shared Preferences is a .xml based data storage system. On Android, each activity can own its Shared Preferences in order to store and load values. Accessing from other activity to these values is not easy and requires the addition of explicit code. In order to avoid it, Android allows creating SharedPreferences for the application context, not just one activity, this way the data can be easily accessible from other activities or objects that have access to this context.

4.4.1.13 MyTextWatcher

com::droidStitcher::myTextWatcher
editText : EditText
<<create>> myTextWatcher(edittextsecond : EditText) afterTextChanged(s : Editable) : void beforeTextChanged(s : CharSequence,start : int,count : int,after : int) : void onTextChanged(s : CharSequence,start : int,before : int,count : int) : void

Figure 75

This TextWatcher allows jumping between the Connection Dialog fields.

4.4.1.14 GalleryListActivity

com::droidStitcher::HandlerFrameCounter::GalleryListActivity
mCurrentNode : File mLastNode : File mRootNode : File f : File mFiles : ArrayList mAdapter : CustomAdapter
onCreate(savedInstanceState : Bundle) : void onBackPressed() : void refreshFileList() : void onResume() : void deleteDialog(fileToDelete : File) : void deleteConfirmDialog(fileToDelete : File) : void onSaveInstanceState(outState : Bundle) : void onListItemClick(parentListView : ListView,view : View,position : int,id : long) : void

Figure 76

This activity allows the user managing the files stored on previous sessions. It works as a file explorer, allowing navigating through the stored files. Its root is the application main folder. It is also responsible of displaying the deletion dialogs and launching the image slider when necessary.

4.4.1.15 CustomAdapter

CustomAdapter
items : ArrayList context : Context assetManager : AssetManager typeFace : Typeface
<<create>> CustomAdapter(context : Context, textViewResourceId : int, items : ArrayList, assetManager : AssetManager) getView(position : int, convertView : View, parent : ViewGroup) : View

Figure 77

This adapter loads items on the gallery list. If the item is a folder, a folder icon will appear before its name. If the item is an image, the AsyncTask LoadImage will load its miniature.

4.4.1.16 LoadImage

LoadImage
f : File imv : ImageView
<<create>> LoadImage(imv : ImageView, file : File) onPreExecute() : void doInBackground() : Bitmap onPostExecute(result : Bitmap) : void

Figure 78

This AsyncTask loads the image miniatures to provide a preview of the image. While loading the images a question mark icon could be seen. This icon replaces the miniature since the image thumbnail is loaded.

4.4.1.17 ImageSlider

com::droidStitcher::ImageSliderActivity
currentImage : String currentFolder : String currentFolderFile : File currentImageFile : File currentFolderFiles : File[] comp : Comparator img : ImageView tempX : int
onCreate(savedInstanceState : Bundle) : void onTouch(arg0 : View, motionEvent : MotionEvent) : boolean

Figure 79

This activity allows sliding between images from a concrete folder.

4.4.2 StitcherServerClasses

4.4.2.1 ServerLauncher

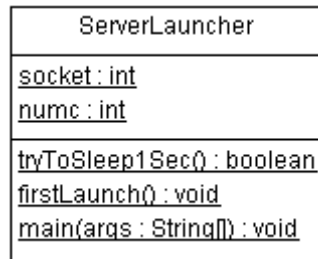


Figure 80

This class is the first created when launching the server. It is responsible of the server root folder creation, and launching connection threads when necessary.

4.4.2.2 StitcherThread

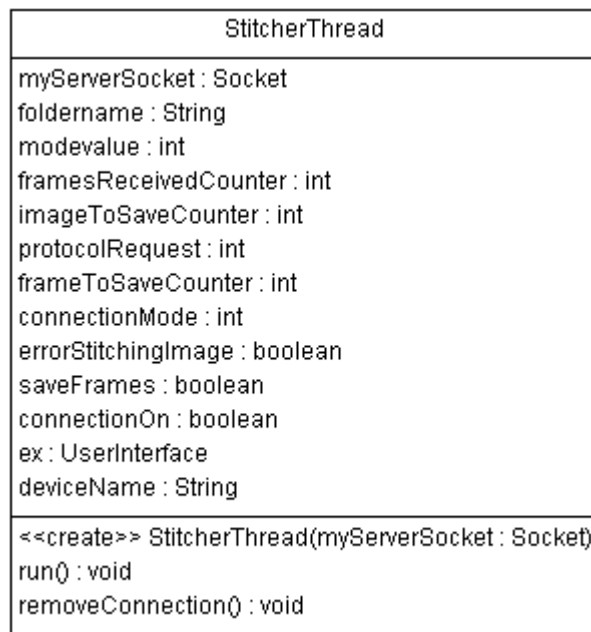


Figure 81

This thread manages the entire connection with the devices. Once a device is connected to the server, a new StitcherThread is launched and will handle that connection till its end. It is responsible of launching the User Interface on the Streaming Sessions and updating its contents.

4.4.2.3 MatlabStitching

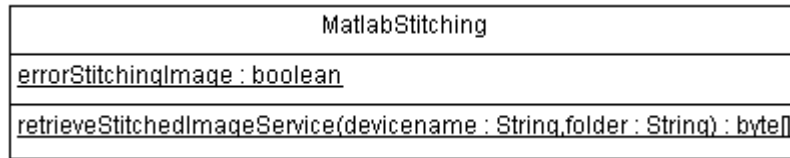


Figure 82

This class launches the stitching process. The functions called are located in the .jar compressed library deployed with MATLAB.

4.4.2.3 UserInterface

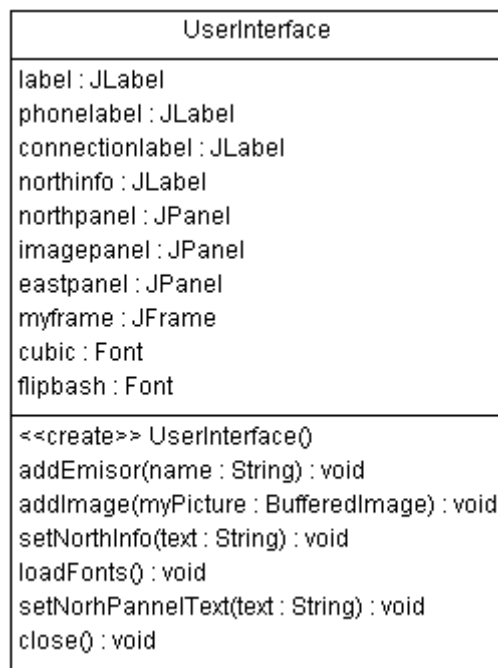


Figure 83

This class is responsible of creating the user interface. Its structure is based on modular panels. It is easy to change or add new panels. The three main panels are:

- NorthPanel: Display the number of frames received.
- Video Panel: Displays each received frame.
- Connection Panel: Displays the device running that concrete session.

4.4 Event-trace Diagrams

On this subchapter are presented the Event-trace diagrams corresponding to the applications implementation.

4.4.1 Android Application Event-trace Diagrams

4.4.1.1 Launching the Camera

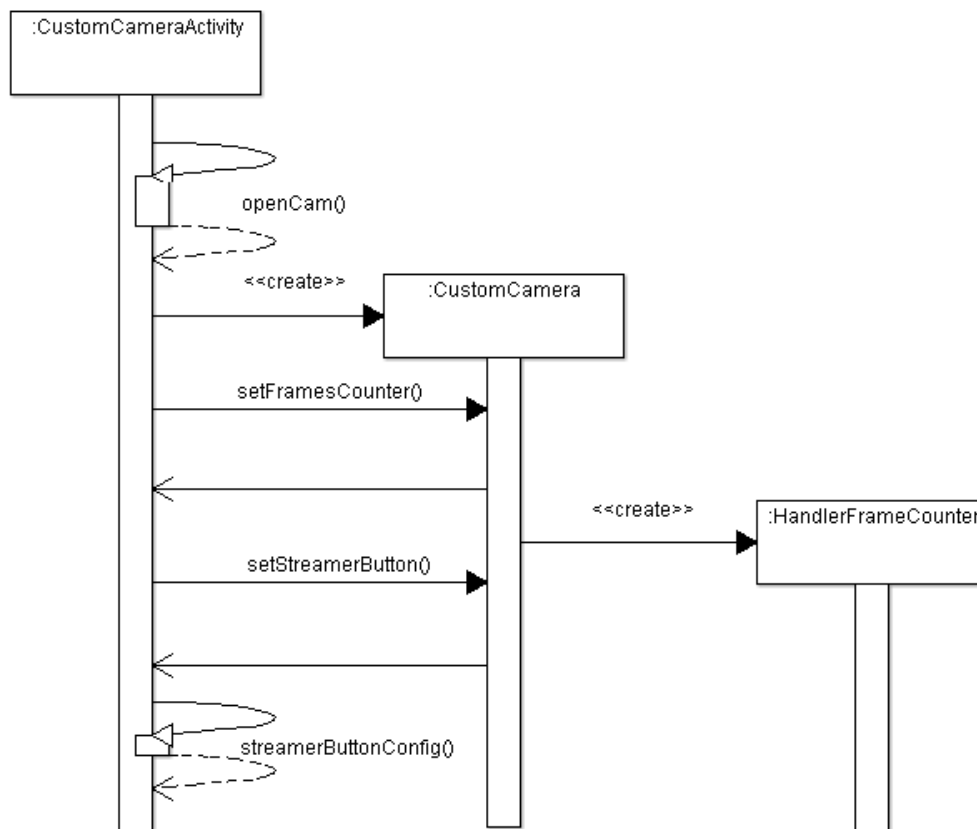


Figure 84

When opening the `CustomCameraActivity`, the first thing that is checked is the availability of the hardware camera. Once the camera is loaded, a preview holder, `CustomCamera`, is launched. Some of the parameters from the `CustomCamera` have to be set from the `CustomCameraActivity`, like the frame counter or the Streaming Button.

To handle the frame counter a new object is created, the `HandleFrameCounter`.

To finish the loading the Streaming Button listeners are set on the `CustomCameraActivity`.

4.4.1.2 Start Streaming Session

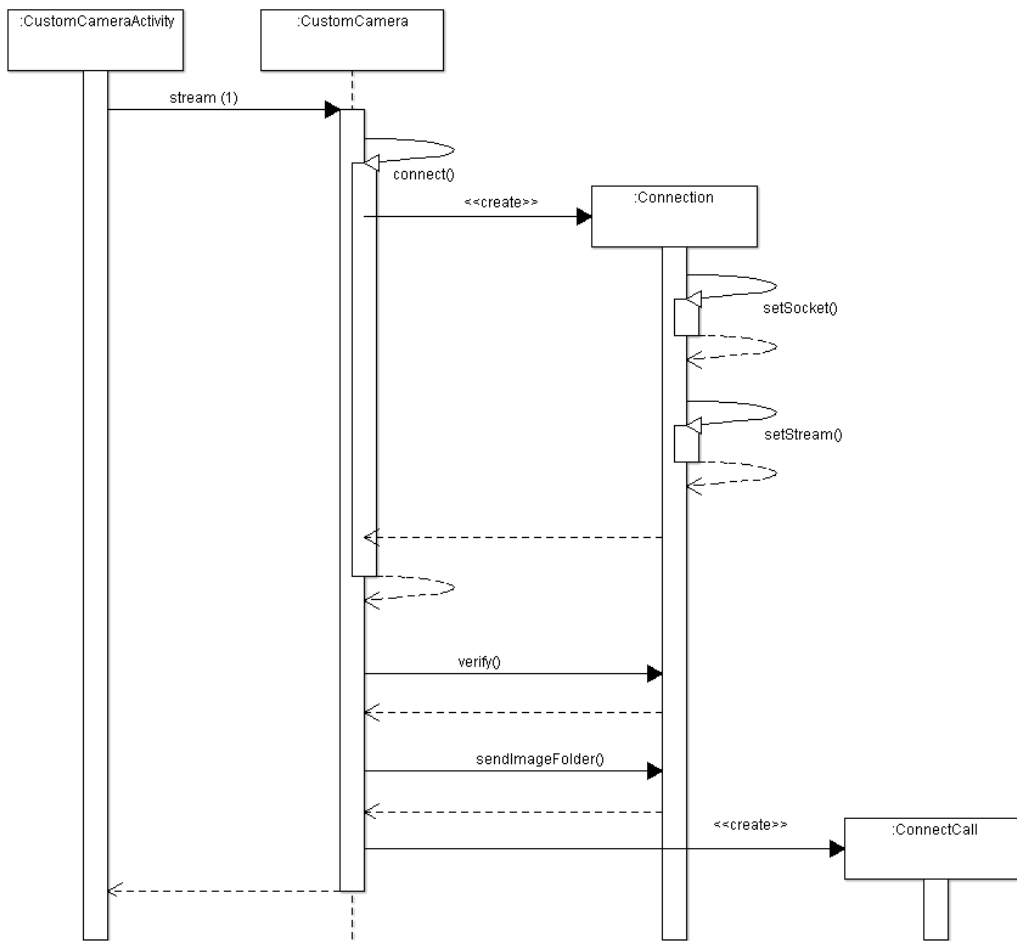


Figure 85

When the Streaming Sessions is launched, the CustomCamera object creates a new Connection object, which will handle all the connection related instructions.

The Connection object sets the socket (connect to the server) and establishes a data stream over them, in order to exchange information.

The next step to take is to send necessary session related information calling the methods `verify()` and `sendImageFolder()`.

To finish this process a ConnectCall is launched. ConnectCall is a Callback that sets the Preview Holder behavior. It allows sending every frame taken to the server and store images.

4.4.1.3 Sending Preview Frames for subsequent Stitching

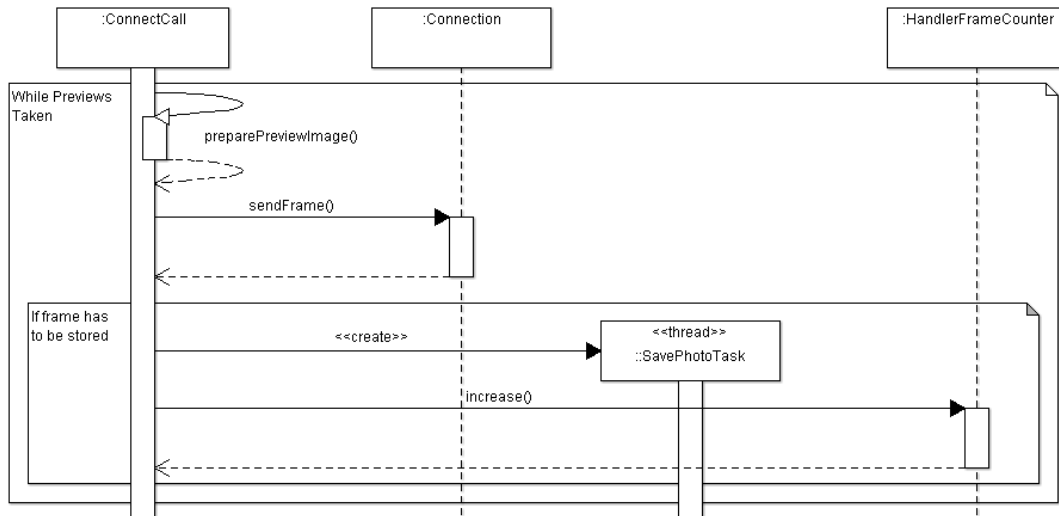


Figure 86

On *Figure 86* is presented what *ConnectCall* does in order to send images to the server and update information on the screen. On this concrete case the images to stitch will be preview frames, not camera pictures.

Once a frame is taken, it needs to be casted to a compressed image format type. The method *preparePreviewImage* compresses this frame before sending it to the server.

After the frame has been compressed and sent to the server, it could be stored if needed. If the image is stored, it will be sent again to the server when the session finishes, but this time as an image to stitch. After storing the frame, the image counter is increased.

Note that the quality of the compression and the image storage rate are configured on the Options menu.

4.4.1.4 Sending Captured Images for subsequent Stitching

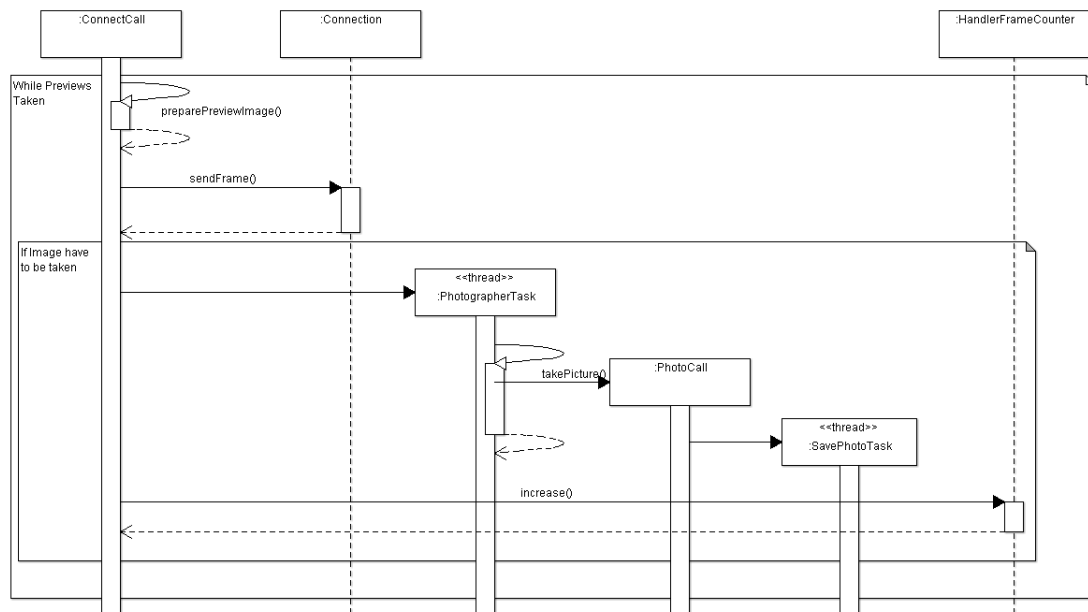


Figure 87

On this case the images stored, and consequently sent, for stitching are pictures taken by the smartphone camera.

The preview frames are sent as in **4.4.1.3 Sending Preview Frames for subsequent Stitching**, but instead of saving a concrete frame, a new picture is taken.

A new Android AsyncTask is created, which will load the takePicture process. To store the image and restart the image preview on the Preview Holder a new callback is created, called Photocall. After taken the picture, the image counter is increased.

4.4.1.5 Finish Streaming Session

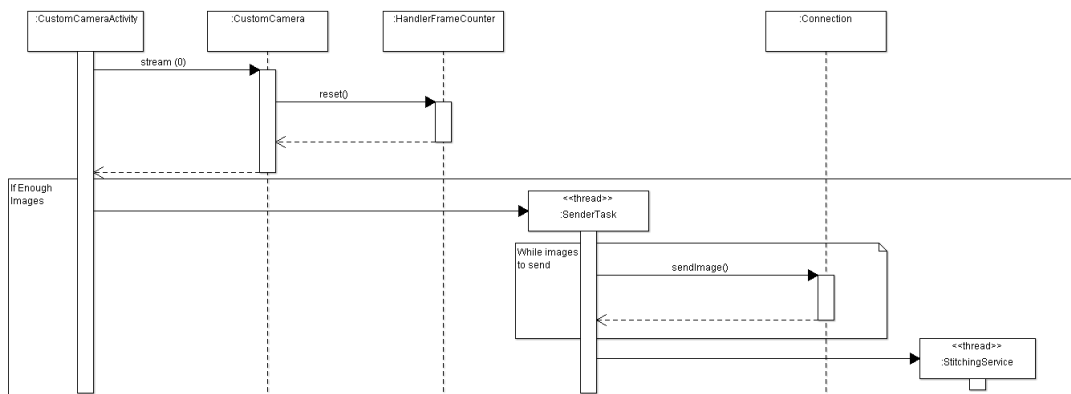


Figure 88

When the user clicks on the streaming button while the streaming session is running, the session is finished. *Figure 88* illustrates how the application solves this situation.

The first thing done is a reset over the frame counter. After that, is checked the number of images stored on this session. If enough, these images are sent one by one to the server. This function is handled by the AsyncTask `SenderTask`.

Once this is done, the connection with the server is closed and the `StitchingService` is launched. This service request for the image stitching and waits for the final product.

4.4.2 Server Application Event-trace Diagrams

4.4.2.1 Streaming Session

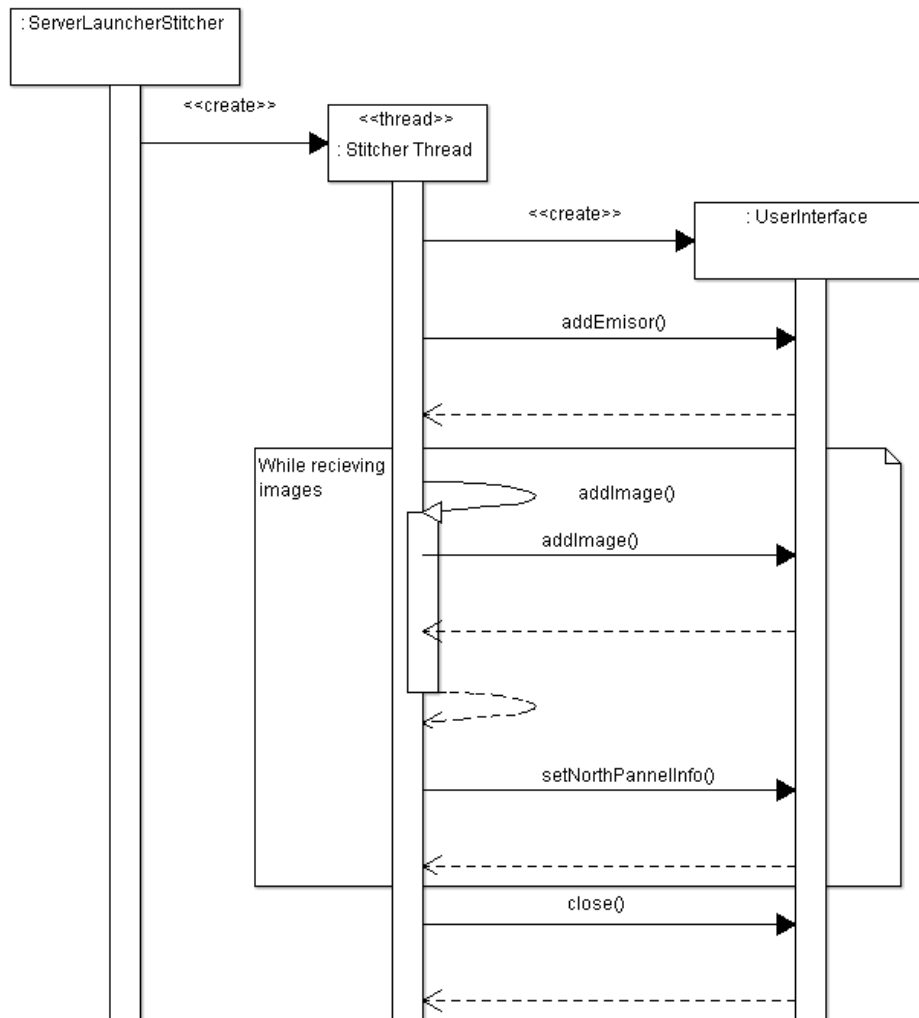


Figure 89

When a new Streaming session is launched from the smartphone, a new thread is created on the server side to handle that concrete connection. This thread, called `StitcherThread`, launches a new display, called `UserInteface`, which has the live-streaming panel and gives relevant information about the session.

When a frame is received, it is set on the view panel located on the display. Also a frame counter is increased on the Northpanel. Once the session has finished, the display is dismissed and the `StitcherThread` dies.

4.4.2.2 Stitching Session

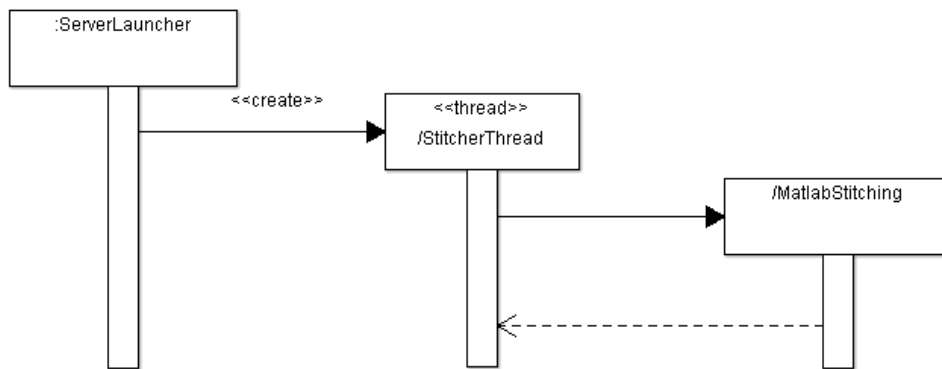


Figure 90

When stitching images a `StitcherThread` is also created. Instead of opening the session information panel the thread launches a new `MatlabStitching` instance, which tries to create the stitched image.

5 User Guide

On this chapter will be explained the necessary steps to properly run the Android application, DroidStitcher, and the server application, StitcherServer, and their requirements. It also contains the user guide which explains how to manage the Android application.

5.1 Application Specifics

DroidStitcher:



This app has been entirely developed using Eclipse and its Android SDK. The applications involved on its user interface have been Photoshop CS6, AAA Logo.

It is quite simply to use as explained below on this chapter. Its weight is about 2,22 MegaBytes.

Figure 91: DroidStitcher logo

StitcherServer:

The server application has been also developed on Eclipse. The user interface has been developed using Java Swing.

The interface on both applications has been devolved under the same pattern. The typography used is Flipbash¹⁴ and the color gamma used is white for the digits and dark electric green and black or grey for the backgrounds.

Due to the code deployment is easy to change the user interface pattern on both applications. The main aim for applying a custom style over the applications has been debugging.

¹⁴ <http://www.dafont.com/flipbash.font>

5.2 Android requirements

Android OS versions overview:

Version	Code name	Release date	API level	Distribution (December 3, 2012)
1.5	<i>Cupcake</i>	April 30, 2009	3	0.1%
1.6	<i>Donut</i>	September 15, 2009	4	0.3%
2.0–2.1	<i>Eclair</i>	October 26, 2009	7	2.7%
2.2	<i>Froyo</i>	May 20, 2010	8	10.3%
2.3–2.3.2	<i>Gingerbread</i>	December 6, 2010	9	0.2%
2.3.3–2.3.7	<i>Gingerbread</i>	February 9, 2011	10	50.6%
3.1	<i>Honeycomb</i>	May 10, 2011	12	0.4%
3.2	<i>Honeycomb</i>	July 15, 2011	13	1.2%
4.0.x	<i>Ice Cream Sandwich</i>	December 16, 2011	15	27.5%
4.1.x	<i>Jelly Bean</i>	July 9, 2012	16	5.9%
4.2	<i>Jelly Bean</i>	November 13, 2012	17	0.8%

Table 2

The Android application was developed for working on Android 2.2 (Froyo API Level 8) and below versions. Due to technical advantages it has been finally developed for Android 2.3 (Gingerbread – API Level 9) versions and below and it does NOT work on previous versions. This is explained more accurately on the development chapter.

Considering the table above, the application compatibility is:

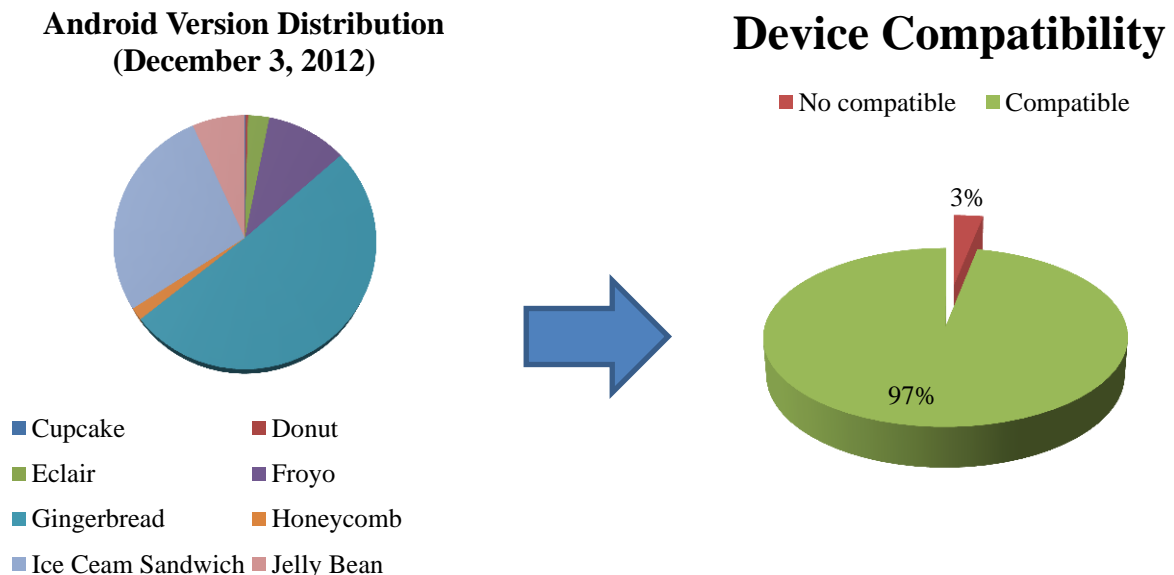


Figure 92

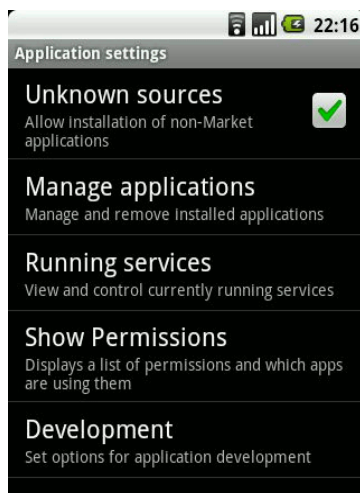
5.3 Installing the Android application

The Android device should have the next requirements to properly run the application:

- Android smartphone with 2.3 version installed (or next)
- SD card on the device with space to write on it
- Back side camera
- Local Area Network Connection

There are two possible ways of installing the application on an Android smartphone:

1 – Installing directly the APK file:



- a) Connect the smartphone via USB to the computer
- b) Check that the connection mode mounts the smartphone SD
- c) Store the .APK file on a folder chosen by the user.
- d) On the smartphone application settings menu, be sure Installation from unknown resources is ticked
- e) Disconnect the device from the USB port or change its connectivity to load battery.
- d) Search for the stored APK file, run it and complete the installation.

Figure 93

2 – Installing from Eclipse:

- a) Connect the smartphone via USB to the computer
- b) On the smartphone Development options be sure debug mode is checked
- c) Right click on the project
- d) Click on Run As > Android application

Steps c) and d) could be avoided by opening a class from the project and clicking ctrl+F11 (forces to run selected source code on Eclipse).

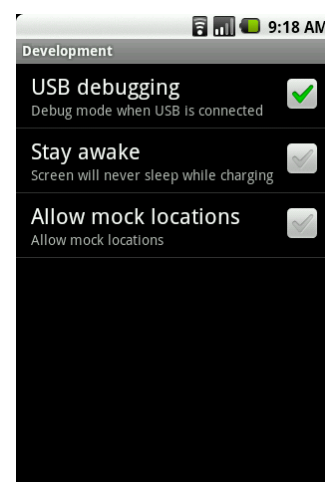


Figure 94

5.4 Server requirements

It is necessary a computer to run the server-side application. Its requirements are:

- Local Area Network Connection
- MATLAB/MCR installed on computer

The MCR, or MATLAB Compiler Runtime, is a runtime which allows execute MATLAB compiled files on a computer without MATLAB installed.

5.5 Back-end Installation

For the back-end running there are three things that should be considered:

- The need of configuring environmental variables depending on the Operating System.
- The need of changing the JRE (Java Runtime Environment) due to the JVM.
- The reference of necessary libraries.

In order to add the environmental variables needed to configure for the server run is necessary to consider the Operating System and the MATLAB runtime, which could be MATLAB itself or the MCR.

If the back-end is running on a Linux-based OS the environmental variables to configure are:

Environmental Variable	Path	Deployment
LD_LIBRARY_PATH	<matlabroot>/runtime/glnxa64	MATLAB
XAPPLRESDIR	<matlabroot>/X11/app-defaults	
LD_LIBRARY_PATH	<mcrroot>/<version>/runtime/glnxa64	MCR
XAPPLRESDIR	<mcrroot>/<version>/X11/app-defaults	

Table 3

If the back-end is running on a PC:

Environmental Variable	Path	Deployment
PATH	C:\Program Files\MATLAB\bin	MATLAB
PATH	C:\Program Files\MATLAB\MATLAB Compiler Runtime\<version>	MCR

Table 4

What are these variables?

XAPPLRESDIR: This variable is used by part of the X11 software that MATLAB uses to display most graphics on Linux/UNIX/Mac.

LD_LIBRARY_PATH: Linux specific, is an environment variable pointing to directories where the dynamic loader should look for shared libraries.

PATH: Windows specific, this environmental variable specifies a set of directories where executable programs are located. In general, each executing process or user session has its own PATH setting.

Where:

- `<matlabroot>`: is the root to MATLAB.
- `<mcrroot>`: is the root to the MCR.
- `<version>`: is the MCR version.

The second step is to change the JRE. Usually Java programs run over the system's default JRE, which has the JVM. In this concrete case the JRE must be changed, in order to avoid OS library dependency issues. (For example, working on Linux, an error is generated because the system's default JRE uses the `libjpeg.so.8` version to encode/decode jpeg images, when compiling the MATLAB code into `.jar` the library used is `libjpeg.so.6`).

In order to proceed, the JRE must be the one included in MATLAB, which path is:
`<matlabroot>/sys/java/jre/glnxa64/jre`

After configuring the JRE, it is also necessary to reference the next library:

Library	Path
javabuilder.jar	<code><matlabroot>\toolbox\javabuilder\jar\javabuilder.jar</code> or <code><mcrroot>/<version>/toolbox\javabuilder\jar\javabuilder.jar</code>

Table 5

This library could be copied and pasted on the preferred directory, for example the directory where the project is. This library allows the communication between Java and MATLAB.

To finish it is also necessary to reference the `.jar` library compiled with the MATLAB command `"deploytool"`, which contains the compiled MATLAB code.

To setup this entire previous running configuration is useful Eclipse, which allows modifying the application environment.

Using Eclipse, the configuration could be modified following these steps:

- 1 - To add environment variables.
 - a) Click on Run>Run Configurations
 - b) Locate the Java application, click on it
 - c) Select Environment label
 - d) Add the environmental variables

- 2 - To change the JRE:
 - a) Right click on the project
 - b) Click on properties
 - c) Click on Java Build Path
 - d) Select the JRE System library
 - e) Click on Edit
 - f) Click on Alternate JRE > Installed JREs
 - e) Click on Search and add the path to MATLAB JRE.

- 3 - To add referenced libraries:
 - a) Right click on the project
 - b) Click on Properties
 - c) Click on Java Build Path
 - d) Click on Add External Jar
 - e) Type the path to the new library

5.6 Tutorial

This tutorial aim is to illustrate a how-to guide to manage both applications.

5.6.1 DroidStitcher User Guide

5.6.1.1 First Launch

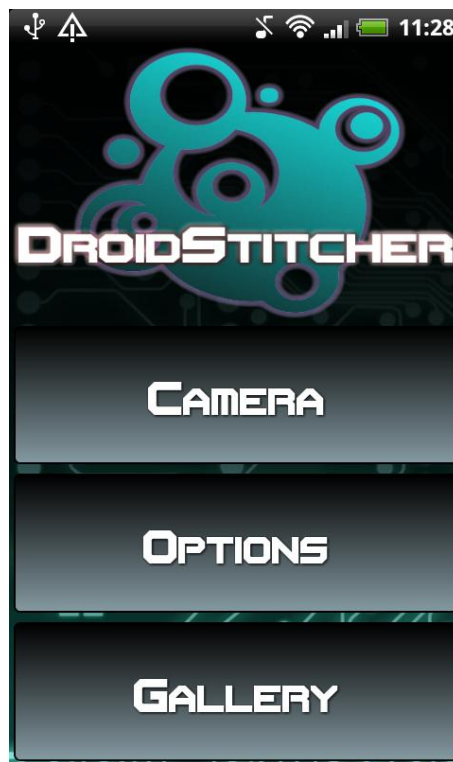


Figure 95

When launching for first time DroidStitcher, it is necessary to configure the application. If the Camera button is pressed before configuring the application, a message will pop on the screen, asking for configure the application.

As is the first time the application is launched, the Gallery will be a black screen, which means there are no stored images. In fact no sessions have been taken.

5.6.1.2 Configuration

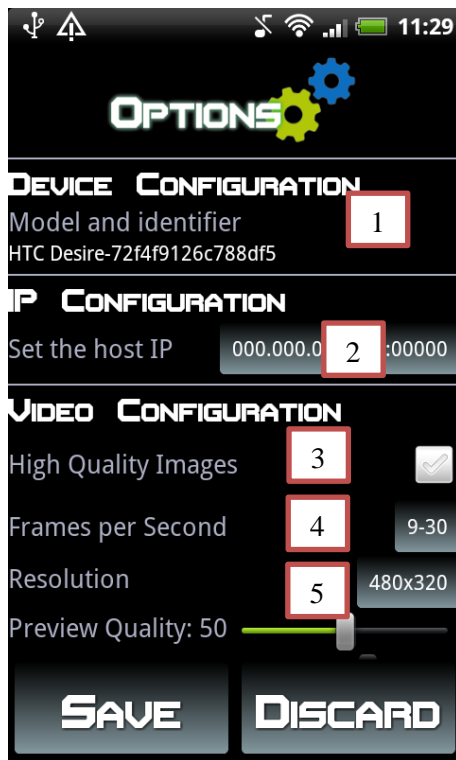


Figure 96

On *Figure 96* can be seen the application configurable options, which are:

- 1- Device Identifier: Not modifiable
- 2- Socket configuration: IP and port
- 3- High Quality Images*: Camera pictures.
- 4- Frames per second: FPS range
- 5- Resolution: Preview resolution

*If the checkbox is ticked, the images stored will be camera pictures, not preview frames. When a camera picture is taken the camera will freeze for few seconds due to the camera loading time and the picture preview display. It is recommended not to move the device while this is happening.

- 6- Preview Quality: Quality of the jpeg compression applied to the previews (10-100, one by one).
- 7- Image per Frames: Number of preview frames that have to be taken to store a frame or take picture to stitch (10-100, 10 by 10).
- 8- Save Frames: This checkbox allows storing all the preview frames on the server.

This context corresponds to diagram 3.1.1.3 (*Figure 15*).

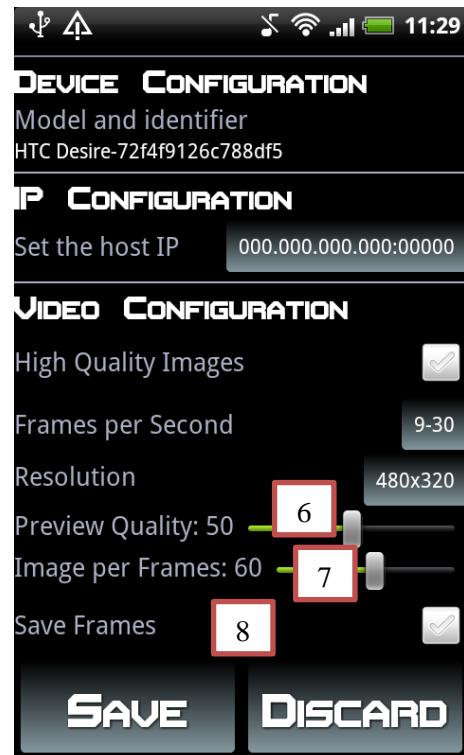
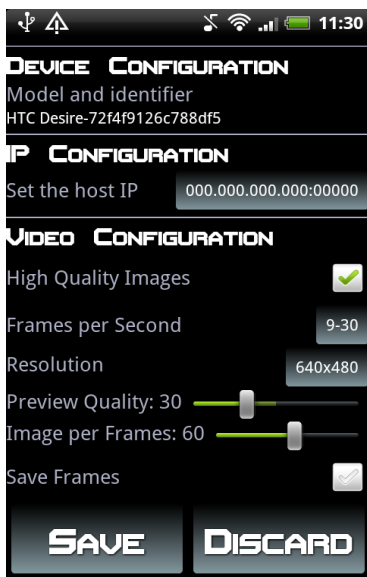


Figure 97



Note that all changes will be stored for subsequent sessions. When launching the Options menu again, the configured options will display their current value.

As an example, on *Figure 98* can be seen the displacement over the Preview quality bar, allowing the user to know its old value.

Figure 98

Configuration tips:

- A low resolution improves live time streaming performance.
- A low quality improves live time streaming performance.
- If High Quality Images is not selected, the images that will be stitched are NOT camera preview frames, so the quality and the resolution will alter the final stitched image.
- If High Quality Images is selected, the JPEG quality compression will be 100%, and the image resolution will be decided by the camera performance (megapixels).

5.6.1.2.1 Setting the Socket

When changing the socket configuration a new dialog will pop on the screen. This dialog allows the user type the host IP and port. The server default port is 2004.

Once the new data is entered, the user may press the Update button in order to save the changes. Otherwise the changes will be dismissed.

For a comfortable typing and improving the user experience, these fields have TextWatchers, which change the field focus when 3 digits have been entered (IP).

The diagram 3.1.1.3.1 (Figure 16) represents this dialog context.



Figure 99

5.6.1.2.2 Setting the FPS Range

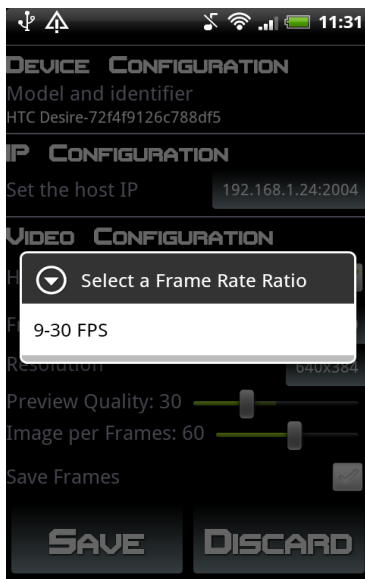


Figure 100

On Figure 100 is shown the dialog that will appear if the Frames per Second button is pressed.

As devices cameras do not work with fixed FPS, a range should be chosen from the list.

On this concrete case a single FPS range is available.

On last generation smartphones (1-2 years old models) the number of FPS ranges has been increased compared to old devices.

Diagram 3.1.1.3.2 (Figure 17) corresponds to this action.

5.6.1.2.3 Setting the Preview Resolution

When pressing the Preview Resolution button, a new dialog will be shown, similar as the presented on Figure 101.

This dialog provides a list of the resolutions supported by the device camera. As not all the devices have the same camera, this values may differ between them.

The values are first ordered by width decrease, and if the width coincides, by height decrease.

Diagram 3.1.1.3.3 (Figure 18) corresponds to this action.

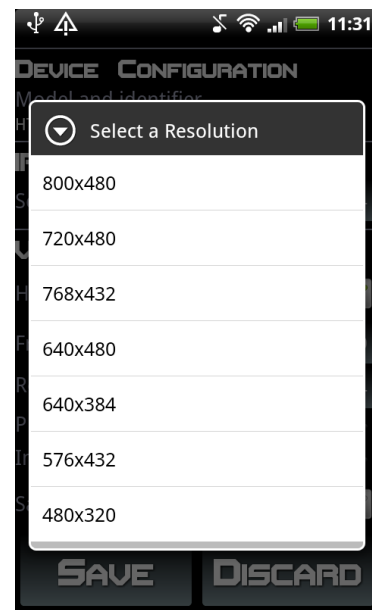


Figure 101

5.6.1.3 Saving the Configuration



On *Figure 102* is shown an example of configuration. When all the values have been properly configured then can be stored by pressing the Save Button.

The values will remain the same while not changed or application uninstalling. Hence it is important to check them to know in which conditions the application is running.

Diagram 3.1.1.3 (Figure 14) corresponds to this context.

Figure 102

5.6.1.4 The Gallery

When the Gallery button is pressed a scrollable list is opened. This list contains all the stored session folders.

The folders are ordered by its creation date, so the last session folder will be on the top of the list.

The folder name corresponds to the session starting time in the following format: HourMinuteSecond-DayMonthYear.

The data shown in brackets corresponds to the number of images stored from a concrete session.

Diagram 3.1.1.4 (Figure 19) corresponds to this context.

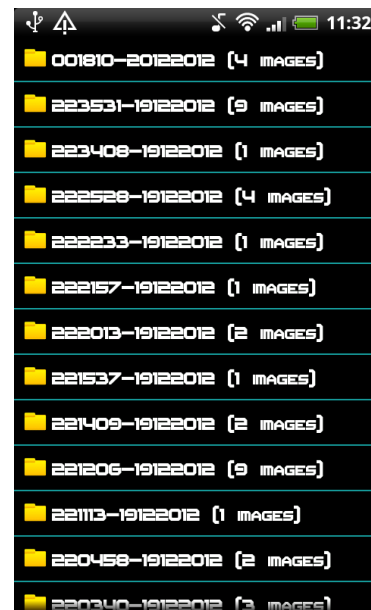
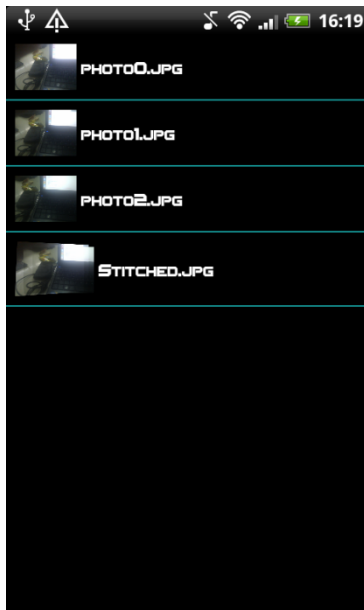


Figure 103

5.6.1.4.1 Session Folder Content



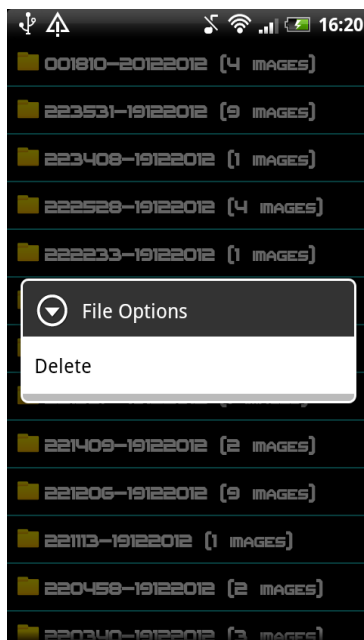
Each folder contained on the gallery has the structure presented on *Figure 104*. The images are displayed in the order they when taken. In this case, Photo0.jpg was the first image taken on the session; Photo1.jpg was the second one, etc. The last picture is the result from that session, the stitched image returned from the server.

The images name will be the same on all session folders. To avoid the confusion this fact can generate, a thumb is located before the name, so the user can check a preview of an image before opening it.

Diagram 3.1.1.4.1 (Figure 20) corresponds to this situation

Figure 104

5.6.1.4.2 Deleting Data



As the images take up so much space, they can be deleted. The images can be deleted one by one or an entire folder can be deleted instead. On both cases a deletion option pops on the screen by a long click on an item.

If the “Delete” option is clicked, then a new dialog will appear. This dialog will offer the user the opportunity to not delete the image, *Figure 106*.

Figure 105

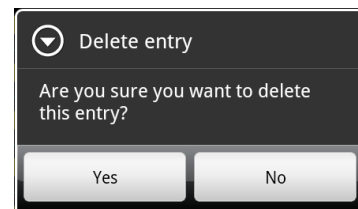
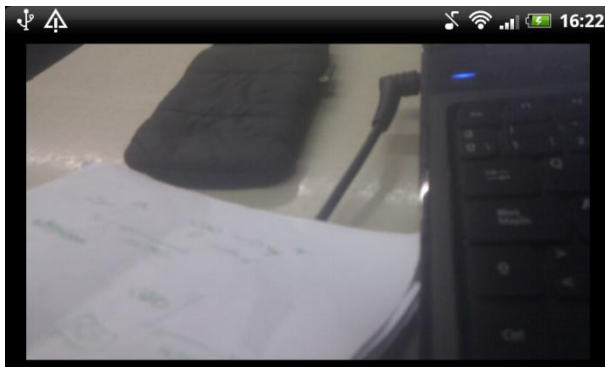


Figure 106

5.6.1.4.3 Sliding images



To improve the user experience the images contained on one folder can be swapped as slides. In order to proceed, the user has to click on the screen and drag his finger to the right (next image) or the left (previous image).

Figure 107

5.6.1.5 Managing a Streaming Session



Figure 108

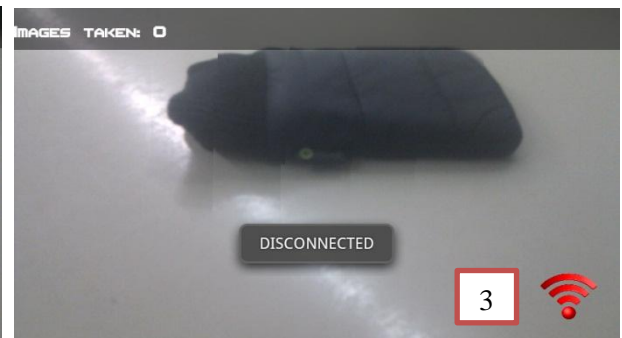


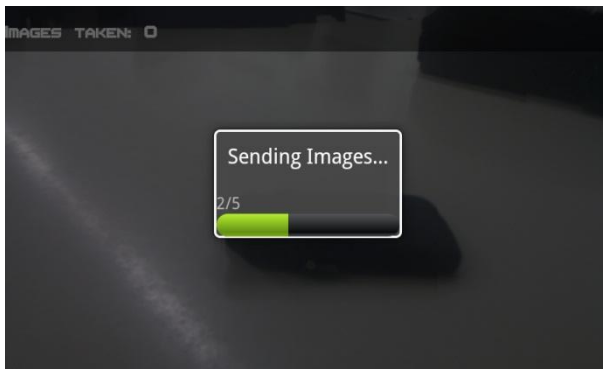
Figure 109

Once the Streaming Session has started, the Streaming button will change its color to blue. If any problem occurs while establishing the connection the color will remain red and an error message will be pop on the screen.

The user can check the number of images stored (1) on the north pan on the screen. This number will increase depending on the configuration the user has applied over that concrete session.

Once the user presses again the streaming button (2, 3) it will send the images to the server, after that the button will be red again.

5.6.1.6 Uploading Images



While the images are being sent to the server a dialog will appear on the screen. This dialog will inform the user about the uploading status.

Figure 110

5.6.1.7 Notifications

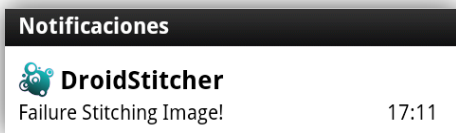


Figure 111

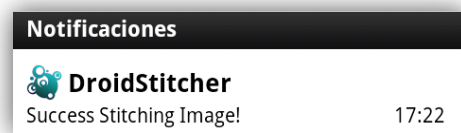


Figure 112

Depending on the stitching result, different notification will appear on the device. *Figure 111* illustrates the notification that appears when the image has not been correctly stitched. Otherwise a notification like the one shown on *Figure 112* will appear on the device.

If the notification is clicked, the stitched image, which is already stored on the device on its session folder, will be displayed. If the error notification is clicked a null image icon will be shown.

5.6.2 StitcherServer User Guide

During the Streaming Session no user interface is shown, so the only graphic support for the user when working with the server will be the Streaming Session user interface.

When the server is launch for first time, a new window will pop up. This window is not the Streaming Session interface, but provides information about the files directory and a 3 seconds countdown before launching the server listener.

5.6.2.1 Streaming Session User Interface

As seen on *Figure 113*, the window displayed has a compact appearance. The window is not resizable and it has 3 pans:

- 1 - Information pan: On this panel is shown the number of frames received on that concrete session
- 2 - Video pan: On this panel is shown the images captured by the device in real time.
- 3 - Connection pan: This panel displays the name of the device which is connected to the server.

At the end of the Streaming session the user interface is automatically closed.

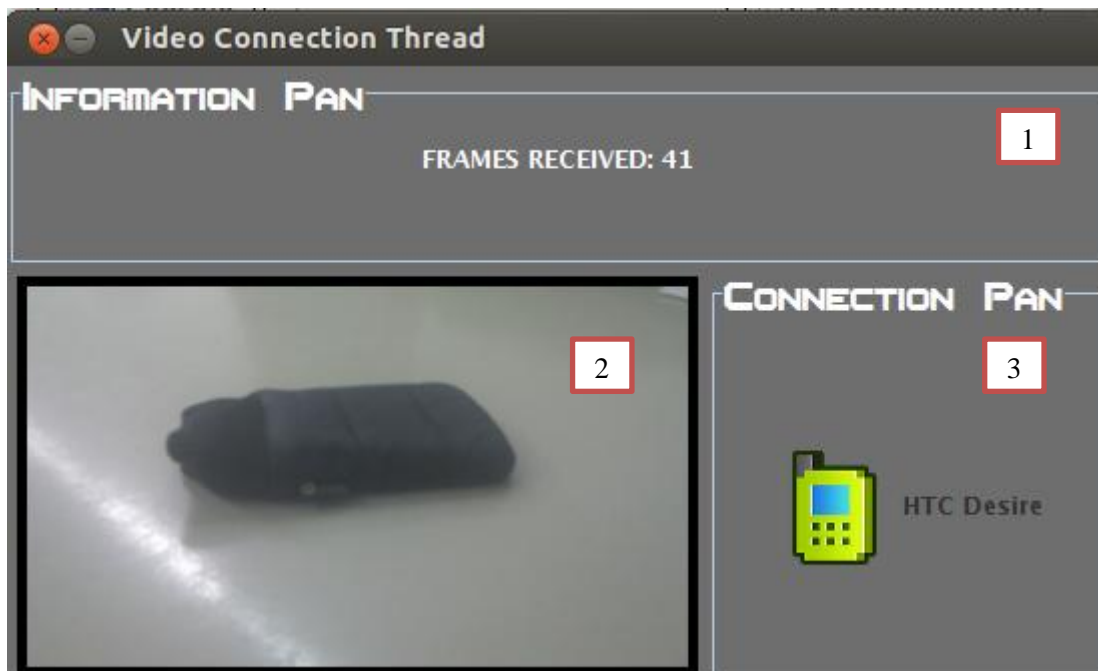


Figure 113

6 Conclusion

6.1 Objectives Achievement

I would like to stress that the main goals proposed along the project have been achieved.

From learning Android basic knowledge to creating a working service, each part of the project has been analyzed in order to obtain the better results.

I consider working with sensors has been a great experience, because it has been proved why in this case have not been useful. On the annex can be found a full explanation about this topic.

All my tutor recommendations have been considered and implemented while developing the project achieving his proposals. My recommendations were also considered and the result of our ideas has been pretty impressive.

The camera management, the method used on image stitching, how to run an Android Service, embedding MATLAB in Java, are just few things needed to understand and properly create this service.

6.2 References

For this project the main search resource has been the net. Most useful websites, in order of relevance, are:

- **www.developer.android.com:** Android development official website. Provides information about how to start developing, good practices and examples.
- **www.stackoverflow.com:** This website offers community solutions for problems found while developing.
- **www.oracle.com:** Provides guides and tutorials about how to develop using Java.
- **www.vogella.com:** Offers several tutorials about Android development
- **www.jdk7.java.net:** Oracle Java official website.
- **www.mkyong.com:** Personal blog which offers Java tutorials and examples.
- **www.mobile.tutsplus.com:** Website which has smartphone development tutorials.
- **www.developerlife.com:** All kind of tutorials for developers.
- **www.android-pro.blogspot.com.es:** Android basic knowledge tutorials.
- **www.fourcc.org:** Offers complete explanations about image formats.
- **www.mathworks.es:** MATLAB official website.

A book have been consulted:

- **Programming Android** (Java programming for the new Generation of Mobile Devices) – *Z. Mednieks, L. Dornin, G. Blake Meike & M. Nakamura.* – O'Reilly 2011

6.3 Appreciation

As a conclusion of this project, I would like to thank everyone not just for the support during the project, but for these four years coursing the major.

Without these years of learning it would have been impossible for me to finish this project, so first of all I would like to thank all the professors who have tough me most of the things I know.

I would like to thank also my university colleagues. With their company these years have become one of the most important parts of my life.

My relatives have become a strong support during these years, so it is fair to mention them on this chapter.

Last but not least, I would like to thank my tutor, Simone Balocco. Without his help, recommendations and patience this project could not have been what it is.

6.4 Possible Improvement

During the development some possible improvements were found:

The first possible improvement is the data transmission. Applying a data compression algorithm over the images that have to be sent will make the data transmission faster than it is.

Another possible improvement is related with the stitching process. As has been explained, the stitching technique is processed by MATLAB. As this code is embedded into Java, it can be modified without making any change on the server code. The fact is that the modularity achieved with this code disposal facilitates the stitching procedure improvement.

An improvement of the Streaming Session user interface could be a great achievement too. Right now is quite simply and with more work it could be a pretty useful workstation. For example, changing its viewing panels, adding additional information like the IP of the streaming device or adding more video panels on one window, this way several devices view can be checked from just one interface.

Adding new functionalities could generate a greater application. Imagine the user could choose between image stitching and other kind of procedures. If this change is introduced into the application, it would generate a device image transformation suite, allowing the user obtaining impressive results from an ordinary device.

Annex

I How Java runs MATLAB

The software used to run MATLAB code on Java is called MATLAB Builder JA. This builder encrypts MATLAB functions and generates a Java wrapper around them so that they behave just like any other Java class. These classes are portable and run on all platforms supported by MATLAB.

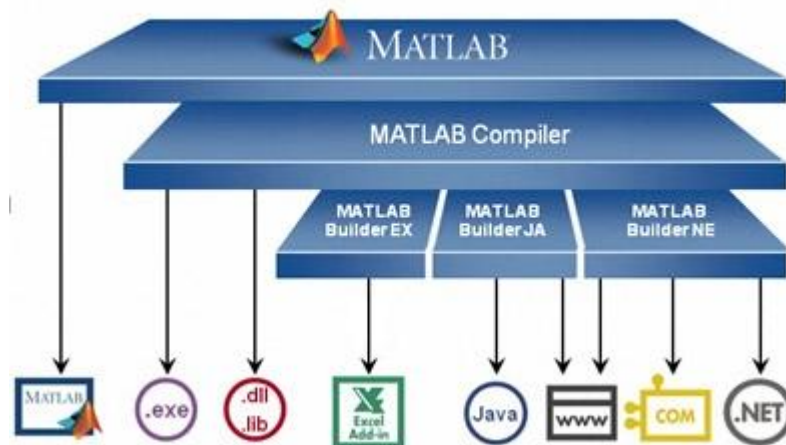


Figure 114

As seen on *Figure 114* the Compiler allows to run code developed in different platforms. On this case the opposite has been done, MATLAB code has been embedded on Java.

To deploy the MATLAB code used on this project, it has been necessary to use the *deploytool* instruction, which allows compacting the MATLAB code in a .jar library. The available formats to compact the MATLAB code by using the *deploytool* are specified by the toolboxes added with MATLAB installation, so it is not possible to compact on .jar libraries if the Java toolbox is not installed.

II Working with Accelerometers

Sensors

It is known that Android devices own a great sort of sensors. Magnetometer, accelerometer, proximity sensor... these are just a few sensors that could be found on most of Android devices.

Table 6 illustrates the whole set of sensors:

Sensor	Type	Description	Uses
<u>TYPE_ACCELEROMETER</u>	Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
<u>TYPE_AMBIENT_TEMPERATURE</u>	Hardware	Measures the ambient room temperature in degrees Celsius ($^{\circ}C$).	Monitoring temperature
<u>TYPE_GRAVITY</u>	Software or Hardware	Measures the force of gravity in m/s^2 that is applied to a device on all three physical axes (x, y, z).	Motion detection
<u>TYPE_GYROSCOPE</u>	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection
<u>TYPE_LIGHT</u>	Hardware	Measures the ambient light level (illumination) in lx.	Screen brightness.
<u>TYPE_LINEAR_ACCELERATION</u>	Software or Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Axis acceleration
<u>TYPE_MAGNETIC_FIELD</u>	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT .	Creating a compass.
<u>TYPE_ORIENTATION</u>	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the <u>getRotationMatrix()</u> method.	Device position.
<u>TYPE_PRESSURE</u>	Hardware	Measures the ambient air pressure in hPa or mbar.	Air pressure.

<u>TYPE_PROXIMITY</u>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
<u>TYPE_RELATIVE_HUMIDITY</u>	Hardware	Measures the relative ambient humidity in percent (%).	Humidity.
<u>TYPE_ROTATION_VECTOR</u>	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
<u>TYPE_TEMPERATURE</u>	Hardware	Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the <u>TYPE_AMBIENT_TEMPERATURE</u> sensor in API Level 14	Temperatures.

Table 6

It is important to know two important tips about the sensors:

- Not all sensors require a hardware implementation.
- Not all devices own the full set of sensors.

As an example, the linear acceleration sensor, which is a software sensor, was introduced on Android Gingerbread 2.3.

The Accelerometers

The first thing to learn about accelerometers is how do they work, which means what is the data gathered by the accelerometers.

Figure 115 illustrates how the coordinate system is superimposed over the device. So, if the device is tilted laterally achieving a landscape position, it is deducible that the X and Y axis will be swapped, and the Z axis will retain its position.

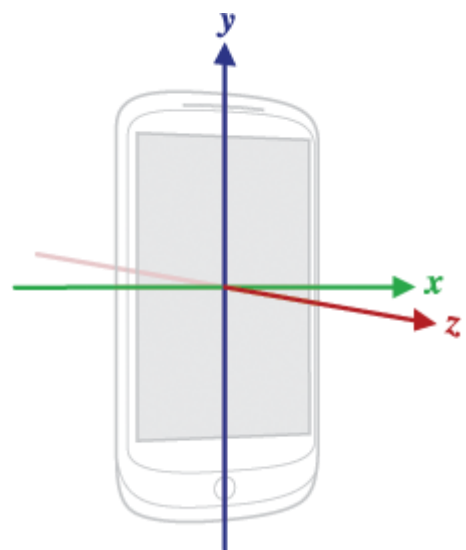


Figure 115

The next thing to know is how to initiate this sensor and how to retrieve the measured data. On code, it is necessary to start a `SensorManager` object:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);
```

Of course it is highly recommended from Android Developers to check if our device has the concrete sensor we are going to use. The following snippet illustrates how to do it:

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (mSensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION) != null){
    // Success
}
else {
    // Failure
}
```

Quite simple to use, this sensor provides a great amount of information, but it is important to know what are the accelerometers useful.

The accelerometers usage is mainly focused on detecting movement, not on distance measurement, easily estimable having time and acceleration, because of the drift produced on the acceleration measurements.

Hence it is strongly discouraged to use the accelerometers as a distance measurement tool.

A test can be useful to note the importance of this drift. It consists on laying the smartphone down a table, store the sensors measurement and check its values. The smartphone must not be moved while it is capturing data.

The result of this experiment is illustrated on *Figure 116*.

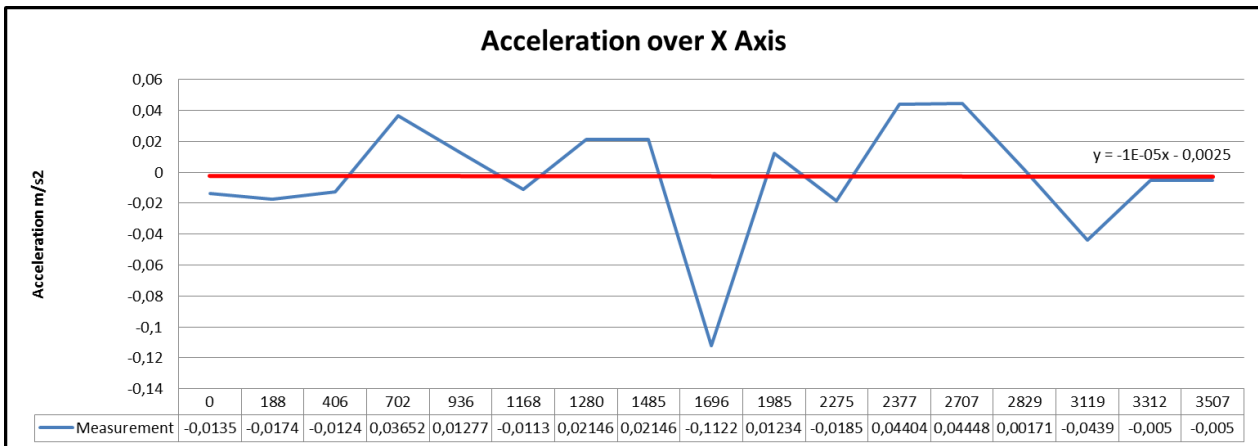


Figure 116: Graph from 18 measured samples. Actually the test had hundreds of samples taken on various sessions.

The X axis on the graph corresponds to time in milliseconds, while the Y axis corresponds to acceleration measurements on the X axis of the smartphone.

As the smartphone has not been moved while capturing acceleration measurements, all these values are closer to zero, and they do not represent anything but drift.

If the smartphone has not been shifted, the acceleration may be 0 theoretically.

The red line represents the acceleration average and as seen on its equation, it has a minimum displacement.

Considering position a summation of velocities, and velocity a summation of accelerations, the error, which is dragged through the operations, is highly increased.

Obviously this error could be threatened, but it is not easy at all, because the real problem is the gravity.

On the Android lecture titled **Sensor Fusion on Android Devices: A Revolution in Motion Processing**¹⁵ is explained how this drift makes impossible to work with the accelerometers in this sense.

This issue has a lot of importance on the project, because at a first glance it had to contain a skin mole distance measurer.

¹⁵ www.youtube.com/watch?v=C7JQ7Rpwn2k

III Protocol

When working with two applications the way of transmitting information between them takes a lot of relevance.

Some of the published open source Android applications which functionality is to stream data captured by the camera implement http servers on its code. There are also well-known protocols that could be implemented, like RTSP. Implementing existing protocols takes a lot of knowledge and time to properly introduce them into an application; it could be an entire project for itself.

On this case, assuming the amount and size of the data sent, the best way to transmitting data is using sockets. It is also important to include on the protocol decision what will be done with the data transmitted and how to manage it when sent.

Sockets provide advantages like the platform, for example allow communication between C and Java, and the protocol customization.

For this project has been developed a request-answer protocol, which grants a correct communication between the applications.

The protocol is specified on *Table 7*.

Request	Answer	Description	Android Application	Android Service
11	12	New connection	✓	✓
21	22	Send new frame	✓	✗
41	42	Close connection	✓	✓
61	62	Create stitching folder	✓	✗
81	82	Create session folder	✓	✗
91	92	Stitching request	✗	✓

Table 7

As seen above, the request and answer petitions are integers. The server will not answer a request if an error occurs while processing data. If an error happens, both applications will close the connection.

Table 7 has two fields called Android Application and Android Service. These fields represent from which part of the application is sent the request. Both are contained on the Android application, but the second one is called from the service launched when finishing the **Streaming Session**.

Each request is explained below:

New connection:

When a **Streaming Session** or **Stitching Session** is launched, this is the first request sent to the server. After sending this request, the client must send its identification (device model and OS identification) in order to proceed.

After it, is necessary to send another integer that will configure the session:

- 1: New Streaming Session
- 2: New Stitching Session

If the new session is a Streaming Session, then the client must sent an integer. This integer must be one of these values 1 (affirmative) or (0) negative. This integer represents the user choice to saving the preview frames.

Otherwise, the server has nothing to do.

Before answering, the server will create a folder for the concrete device.

Create session folder:

When starting the Streaming Session, this should be the second request sent to the server.

After sending this request, the client must provide a session folder name. This folder will store all the information related with this concrete session.

If the user has chosen to save the preview frames, the folder that will contain them will be also created.

Send new frame:

A new frame follows this request. The server will store it if necessary, and will display it at the viewing panel located on the user interface.

The server will also increase the frame received counter and will display its new value on the screen.

Close connection:

This request finishes the connection. The server will answer just before receiving it and then will close the data streams and the socket.

Create Stitching Folder:

When the server receives this request creates a new folder that will contain the images to stitch. This will happen on the first call of this request.

This request is also called when sending images to stitch, at the end of the Streaming Session.

Stitching Request:

This request is sent to the server in order to stitch images. The images are stored on a folder which name is supplied by the client just after calling this request. The server can also send two different types of answer before sending the protocol answer (92).

The first kind is “NERROR”. This answer means no error has happened while stitching, and after it the stitched image is sent. Otherwise the server will send “ERROR”, allowing the device application to know that a problem has occurred and the stitched image has not been generated properly.

Connection: Waits and Errors

If a connection error is produced both applications will automatically try to close it and dismiss all the processes involving the connection.

DroidStitcher will exit the Streaming/Stitching Session and will warn the user, while StitcherServer will dismiss the corresponding thread.

No timeouts have been established over the sockets, so in the case the StitcherServer falls when DroidStitcher is sending information, the connection interruption is not produced by a long wait. In this concrete case, the interruption comes from the data streams.

The data streams are deployed over the sockets as `ObjectOutputStream`¹⁶ and `ObjectInputStream`¹⁷ on both applications. These objects allow writing and reading Java primitive data types. When the stream is interrupted, an exception is launched, which allows to handle the issue.

Although these objects are not restricted for using a network socket as a data stream, on the project have been used for sending integers, byte arrays or strings.

¹⁶ <http://docs.oracle.com/javase/1.4.2/docs/api/java/io/ObjectOutputStream.html>

¹⁷ <http://docs.oracle.com/javase/1.4.2/docs/api/java/io/ObjectInputStream.html>

IV Android Manifest

The Android Manifest is a XML file present in all applications. The manifest contains relevant data about the application, like the activities and their configurations, permissions, etc.

When using an application, especially on smartphones, it is important to know which are the resources used by the application and why is necessary.

DroidStitcher manifest requirements are listed on *Table 8*.

Permission	Description
<i>CAMERA</i>	Allows the application to access the device camera.
<i>WRITE_EXTERNAL_STORAGE</i>	Allows writing data on the SD.
<i>VIBRATE</i>	Allows device vibration. Used on the notifications.
<i>INTERNET</i>	Allows the application performing network operations.
<i>STORAGE</i>	Allows accessing data stored on the SD.

Table 8

V Device camera usage regulations

Working with a device camera requires a well-structured implementation when launching or closing it.

As the camera is a shared resource between applications, if an app takes control over the camera is needed to release it at the appropriate moment. If the camera is not released and an application requests it, the camera preview will be not shown and the application would be useless.

Another thing to keep in mind is the fact the function **takePicture** is an asynchronous camera method. As each device takes its time to load the camera hardware, a picture cannot be taken in a synchronous way.

To take a picture properly, it is necessary to load that method on an Android AsyncTask, which will take the picture when the camera hardware has been prepared.

After taking the picture, the camera preview is always destroyed, so it is necessary to launch it again.

Image Compression

The image data format in which the camera preview frames are stored usually is YUV NV21. In order to compress them on JPEG format it is necessary to apply the following instructions:

```
YuvImage image = new YuvImage(rawpic, formats, w, h, null);
ByteArrayOutputStream out = new ByteArrayOutputStream();
Rect area = new Rect(0, 0, w, h);
image.compressToJpeg(area, compressionquality, out);
byte[] previewframetosend = out.toByteArray();
```

The rawpic variable is a byte array containing the picture in YUV NV21 format. The variable formats, w and h are camera settings variables. The image is compressed in a Rect object (rectangle) which has the size of the original picture.

Once these operations have been done, the result is a byte array which contains the picture in JPEG format.

VI Data Distribution and Storage

The data stored, fully composed by images, has been stored on the device following the hierarchy present on *Figure 117*.

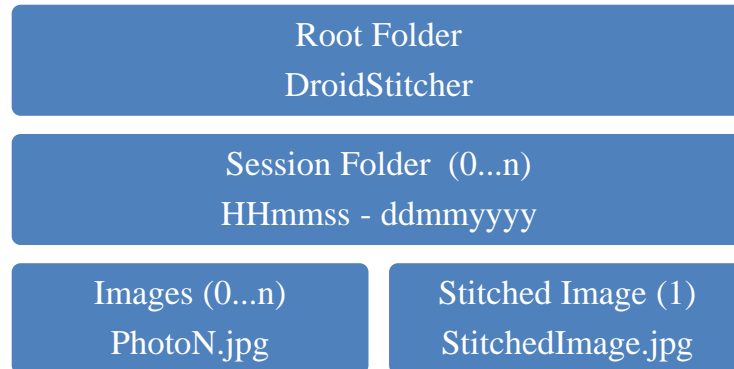


Figure 117

Where:

- Root Folder: Called “DroidStitcher” is located on the device SD root folder and contains the session folders.
- Session Folder: Contains the images related to a concrete session.
- Photo: Are the images taken on the session. N represents its number.
- Stitched Image: Is the result the session.

On the following figure it is shown how the images are stored on the server side:

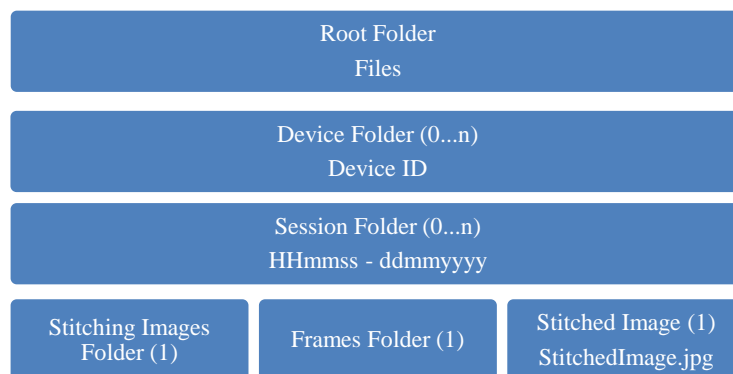


Figure 118

The root folder contains a subfolder for each device. These subfolders contain the session folders from concrete sessions. Inside each Session Folder there are the images to stitch inside the Stitching Images folder and the Stitched Image. If the user has chosen to store the frames, they are located on the Frames folder.

All images are stored in JPEG format.

VII Algorithms

SIFT

Scale-Invariant Feature Transform ¹⁸ is a Computer Vision algorithm which allows detecting features in images. It is applied in several fields like object recognition, robotic mapping or image stitching amongst others.

The extracted features bring a description of the objects presented in the image. These features are located in high contrast points such the edges of an object and are called SIFT keypoints.



As can be seen on *Figure 119*, the SIFT keypoints are located on areas where the viewing conditions, such the illumination or color, have a major contrast.

Figure 119: Image with its keypoints.

RANSAC

RANdom SAMple Consensus ¹⁹ is an iterative method to estimate parameters from a model which contains a set of data. The distance between these data must be numerically measurable.

In order to proceed with the stitching, the RANSAC algorithm will generate pairs of keypoints located on different images. From this pairing a homography is determined for each pair of images. After that, a warping process will fuse the images.



Figure 120: Pair of images with RANSAC correlation.

¹⁸ Lowe, David G. (1999). "Object recognition from local scale-invariant features"

¹⁹ Martin A. Fischler and Robert C. Bolles (June 1981). "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography"

VIII Image Cropping

A script for image cropping has been added to the MATLAB code in order to improve the final result. This script was developed by a third party ²⁰ and published as open source on MATLAB Central. This script allows finding the largest inscribed rectangle inside the figures presented on an image.

Taken as example the images given by the TobW Team for their stitching script, the image cropping procedure has been added as follows.

The first step is to obtain the images to stitch, three on this case:



Figure 121



Figure 122



Figure 123

The next step is to apply the stitching technique:



Figure 124

As can be seen on *Figure 124* the final product has an irregular black frame. This frame is produced when deforming the images to produce the stitched image.

The inscribed rectangle script cannot work directly over this image because it contains too many figures. Each color cluster could be considered as a figure.

²⁰ www.mathworks.com/matlabcentral/fileexchange/28155-inscribedrectangle

In order to solve this problem, the colored image must be turned into a greyscale image. After changing the color disposal, a threshold must be applied to obtain just two figures, the irregular frame and the “image body”:



Figure 125

As seen on *Figure 125* just two figures are on the image. Once this step is done, the script can process the image.

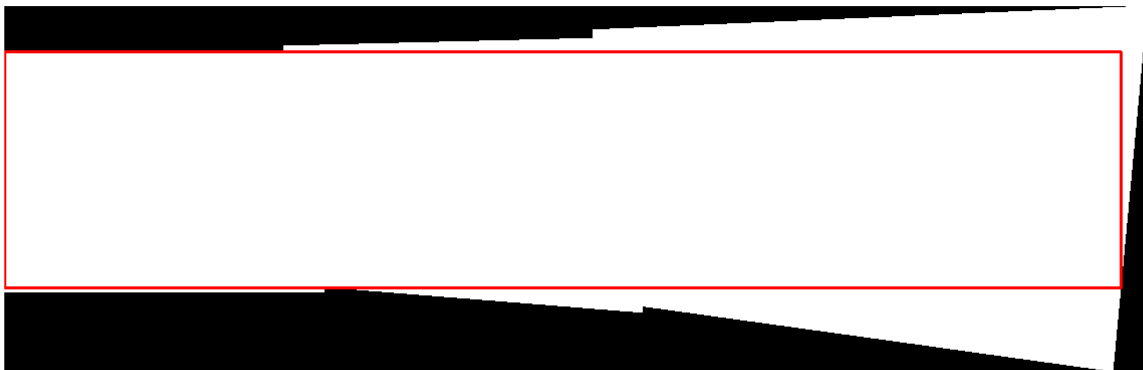


Figure 126

Figure 126 shows the largest rectangle found on the image. The image is a panorama, so the largest rectangle found is a panorama. If the stitched image had been square shaped, the largest rectangle would have had a squared shape.

Indeed this script finds any kind of rectangle inside a figure. Properties as the width or the height are not taken into account. So this script is able to work with any kind of image.

Run this code takes several time on MATLAB, so adding it to the stitching process have produced the server working time to increase.

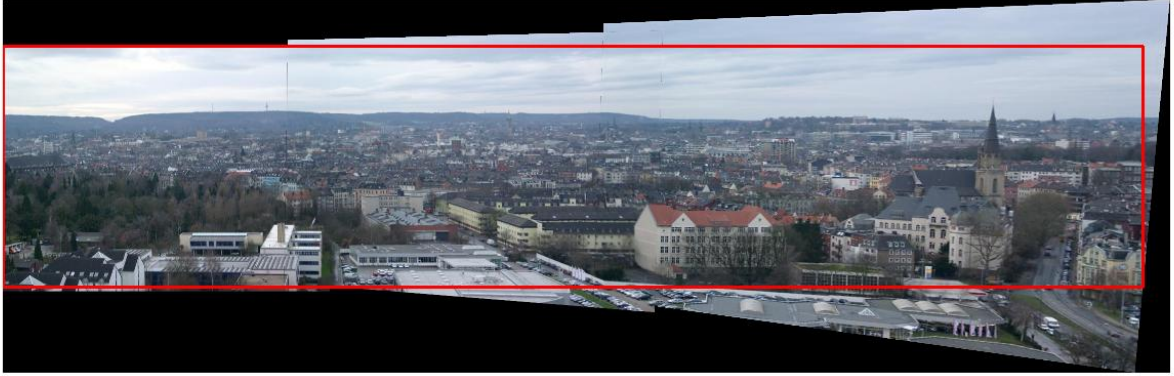


Figure 127

On *Figure 127* is shown the found rectangle over the original stitched image. As can be noticed, some parts of the image will be lost during the image cropping.



Figure 128

The final product is shown on *Figure 128*. The irregular black frame has been completely removed from the stitched image and it looks quite an impressive result.

With the exception of the data loss, this modification is a great achievement in the project. Not for the stitching process, which it does not alter, but the user experience.

IX Examples

This example has been created using preview frames (low resolution images). The dimensions of each original frame were 640x384 pixels and the stitched image dimensions are 1377x385 pixels.



Figure 129

On this case the images used on the stitching process have been camera pictures (high resolution). The original images dimensions were 2592x1952 pixels and the stitched image dimensions are 5491x1896 pixels.



Figure 130