Universitat
de les Illes Balears

Title: Developing OpenCV applications on Google Glass 2

AUTHOR: Jelena Dedović

Master's Thesis

Master's degree in Computer Engineering
(With a speciality/Itinerary in Interactive Technologies)

at the

UNIVERSITAT DE LES ILLES BALEARS

Academic year 2017/2018

Date 05.07.2018.

UIB Master's Thesis Supervisor: Dr. Ramon Mas Sansó

Developing OpenCV applications on Google Glass 2

<Jelena Dedović>
Tutor: <Dr. Ramon Mas Sansó>
Treball de fi de Màster Universitari Enginyeria Informàtica (MINF)
Universitat de les Illes Balears
07122 Palma de Mallorca
<dedovicjelena@yahoo.com>

Abstract

The purpose of this paper is to evaluate the possibility to combine , the device Google Glass 2, with one of the most used libraries for vision based implementation OpenCV for application development. This paper presents and examines the main problems encountered during application development in Google Glass 2 and how to surpass them. The paper also presents a working application called "Room color helper", to proof the concept for the feasibility of such an application. Finally, the paper reviews the possible limitations of such devices.

Key words: OpenCv, Google Glass 2, Android, Augmented Reality, Computer Vision

1 Introduction

Augmented Reality is experiencing expansion at drastic speed. Due to common confusion of virtual environments: virtual reality, augmented reality and mixed reality, I would like to point out the definitions of both augmented [1] and mixed [2] reality: *Augmented Reality (AR):* is taking digital or computer generated information, whether images, audio, video and touch or tactile sensations and overlaying them over in a real-time environment. *Mixed Reality (MR):* An interaction concept that considers how the virtual and real worlds can be combined into a unified interaction space.

The requirements that need to be fulfilled to correctly exploit augmented and mixed reality to be exploit [3] are:

- Interactive action in real time
- Combining the real and virtual world in a real environment
- Matching real and virtual objects

The popularity of AR field became much intense in the last decade, mainly due to the Google Corporation and its product *Google Glass* . Considering vision as the primary sense for human to perceive its surroundings, it's fully understandable the protagonism of smart glasses to are exploit augmented reality. Nowadays, we also have Microsoft's HoloLens [1] and we are awaiting this year One glasses by start-up Magic Leap [2] which both are exploiting mixed reality.

In addition, the newest findings are that the brain is capable to identify picture content in less than 13 milliseconds [3].If we consider only the facts that the resolution in which we are capable to see is 575 mega-pixels and that the registered frames per second are nearly 75, it is understandable that the brain has to process a vast amount of data. Nonetheless thanks to the mankind progress and desire to achieve the same benchmark in technology, it is possible to process large amounts of data with the continuously rising power of the processors, as are the resolution and capabilities of cameras. Accordingly, it is trendy to use the computer vision in all the branches of science along with its algorithms. *OpenCv (Open Source Computer Vision Library)* is one of the most known libraries used for computer vision. This is not strange, considering its multiplatform support, the multi-language support and the power of forty seven thousand people in the user community.

The aim of this paper is to see if is it feasible to combine the most popular product of AR and the most popular computer vision library in order to develop an application. Further, we will present all the drawbacks that I have encountered during this examination regarding Google Glass, OpenCV and application itself, as well as the solutions discovered on how to resolve them. The objective of this paper is to present the findings and an unified conclusion on trying to exploit the combination of these two technologies.

2 The problems

In this section we examine and present the difficulties that have been encountered while trying to build an application for Google Glass using OpenCV. The solutions to these problems will be presented later on. This section is divided in two parts, the difficulties encountered regarding Google Glass smart-ware and regarding OpenCV, with each having a brief

[1] https://www.microsoft.com/en-us/hololens
[2] https://www.magicleap.com/
[3] The most recent study of neurologist of MIT http://news.mit.edu/2014/in-the-blink-of-an-eye-0116

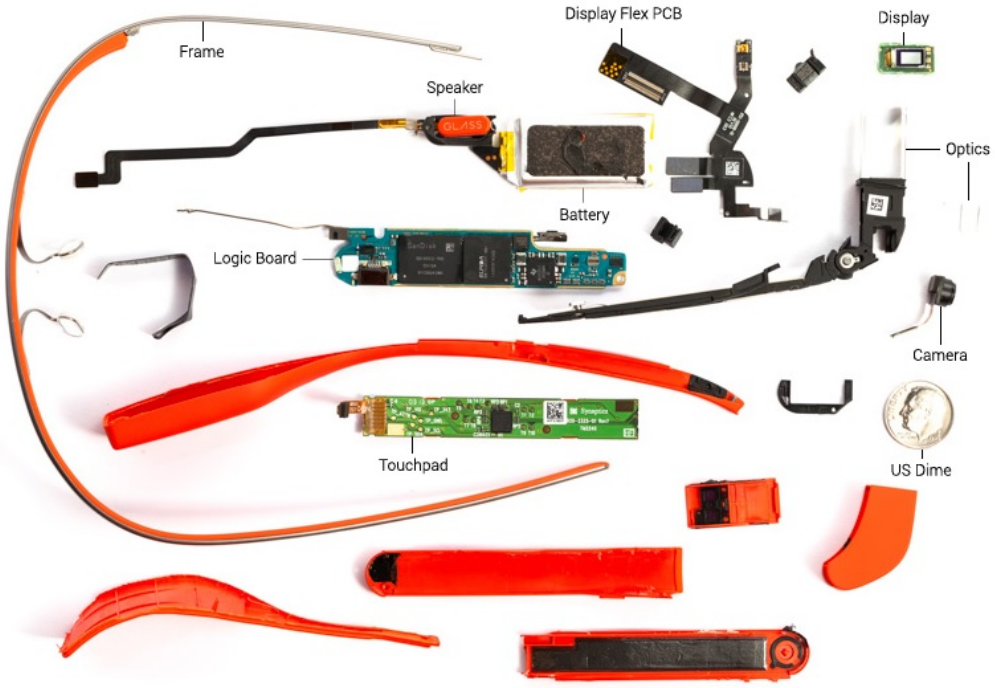

Figure 1: Google Glass hardware components

overview. However, the obstacles encountered while building application with working and enabled OpenCV on Google Glass, and resolutions to these problems, will be presented as a separated section further in the text.

2.1 Regarding Google Glass

Google Glass, we already mentioned in the introductory part, was firstly publicized back in April in 2012 , and shortly after, it went in the sales to a limited amount of users though the "Google Glass Explorer" program for the cost of 1500 dollars [4] . This piece of hardware, eventually named Glass 1 (XE – Explorer Edition), had too many usability and privacy disputes and Google finally decided to terminate Glass Explorer project in the beginning of January 2015. Although, Google continued to exist as Glass 2 (EE – Enterprise Edition), through an Enterprise program named "Glass at Work project", which is primary focused to connect with companies that want to develop business applications. *Due to the short life of the Explorer program, we have noticed that the user community is very reduced that Google support for developers depends on issues occurred during the period that Glass Explorer project was active.* For this paper purposes, I had on the disposal a Google Glass 2 device from Enterprise project.

Google Glass 1 as well as Glass 2, is equipped with a camera, optics, a display, a touch-pad and tactile sensors for navigation, which can be seen in Figure 1.[5] The device has the possibility of voice control, because it is powered with Google Now knowledge base. The Google Now is actually part of home screen launcher added in Android Kit-Kat, enabling voice control over the Glass device with the phrase: "Ok, Glass...". The software platform, runs on Android 4.4 Kit-Kat (API 19), using the special Glass Development Kit (GDK) for application development [6]. This GDK is not an entirely new development kit, in fact it supplements the existing Android SDK and runs directly on the device. GDK enables accessing the hardware, enabling the voice command, the gesture detector and so on.

During the application development we have to face several technological troubles. The *first problem* occurred with the device, the screen turned completely white, disabling any manipulation over the device, which is also called *White Screen of Death*. This condition of Glass has been reported by various users, and some of its roots can be hardware malfunctions - such as loose cable in the optics, extreme weather, the device has got too hot, or software update. Depending on the case, several of the possible solutions to resolve White Screen of Death are: hard reset, overnight stay on the charger, power it down and leave it for couple of days, erasing cache and user data, re-flashing the last EE image.

The *second problem* experienced with Glass 2 comes from *incompatibility of drivers* of Google Glass and the operating system of the machine used for the development. A similar problem has been occurring to users of different OS plat-

[4] http://glassalmanac.com/history-google-glass/

[5] http://www.catwig.com/google-glass-teardown/

[6] https://developers.google.com/glass/develop/overview

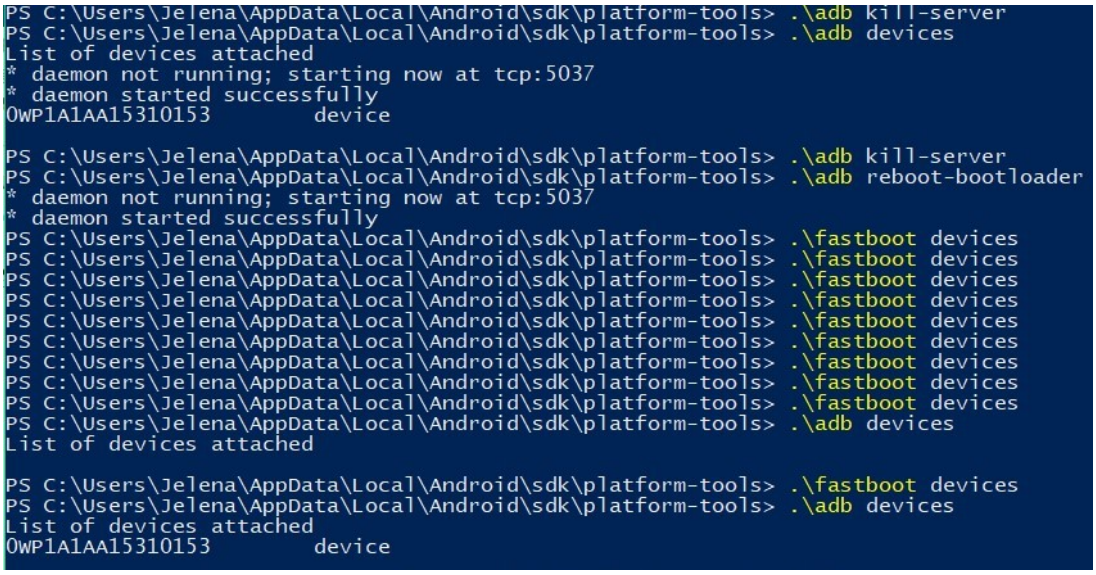

Figure 2: The Glass has not entered properly into Bootloader mode

forms, such as Linux and OS X, but is a negligible number of cases compared to the Windows 8 platform and with its inheritors Windows 8.1 and Windows 10, which was the operating system used during this study. For the development machine to communicate with Android devices, such as Google Glass (the client), the communication between server and the client is enabled via Android Debug Bridge (ADB) [7] while to modify Android file system from a server when the client is in Bootloader mode, it is used fastboot.

As the ADB and fastboot come through the Android SDK Platform-Tools package, the predisposition was that after placing the Glass in Bootloader mode, running the fastboot command in Windows Power-Shell – *fastboot devices* lists all the devices which are in Bootloader mode, which was not the case. We can see on Figure 2, the abd command is executing as it should, and fastboot command is recognized, but it does not list the device as connected - fastboot tool does not function properly.

2.2 Regarding OpenCV

The OpenCv library, is an open source library that is used for computer vision. It was issued by Intel with the main goal developing computer vision and artificial intelligence applications,and it was developed by many authors. While these advanced functions and algorithms are written mainly in C and C ++ languages, this library is cross-platform and multi-language, thanks to the interfaces or wrappers [8]. The *OpenCV4Android* is a version of the OpenCV library that is customized and optimized to work on Android devices. As the execution of the code is in native language, it is not enough to make the application dependent on the library using Gradle. Gradle is an open-source build automation tool as support for dependency management of a project [9].It is necessary to install a special OpenCv Manager as a support for Android devices, which it has no functionality but provides OpenCV API through wrappers in order to use the libraries. The second option for using the OpenCV is extracting the required algorithms from the OpenCV library and use the code through Java Native Interface, in other terms using Android NDK.

The *first problem* regarding OpenCV was to *enable library to compile* as we can see on Figure 3. Currently, the most recent OpenCV4Android library version is 3.4.1 which was released in February this year. In order to enable it as a dependent module for developing a Glassware application, the compiled API of OpenCV for Android, and the compiled API of Glass application must match. The library 3.4.1 targets Android API 21 (Android 5.0 Lollipop) for compilation while, as we could see in previous subsection, the Glassware application uses API 19 for compilation. After adjusting the compiled API of OpenCv in its Gradle file, the libraries still could not compile because Android packages (Figure 3) that are used in OpenCV4Android 3.4.1 are introduced later than API 19, in API 21. The same problem persist *for all versions from 3.1.0 to the most recent 3.4.1*.

Hence, what about *older versions* of libraries considering that development of Glassware was active and popular in that time period? We can find several proof of concept projects, about merging OpenCv with Glass application, such as Open Quartz [10]. It is an open source Google Glass project with several working samples by using the version of OpenCv library 2.4.9. Therefore, the assumption was that libraries should

[7] https://developer.android.com/studio/command-line/adb
[8] https://opencv.org/
[9] https://docs.gradle.org/current/userguide/userguide.html
[10] Official from OpenCV https://github.com/jaredsburrows/open-quartz

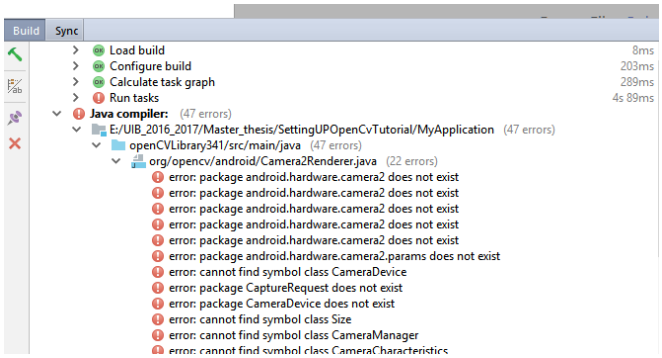

Figure 3: The OpenCv library 3.4.9. can not compile with Glass project

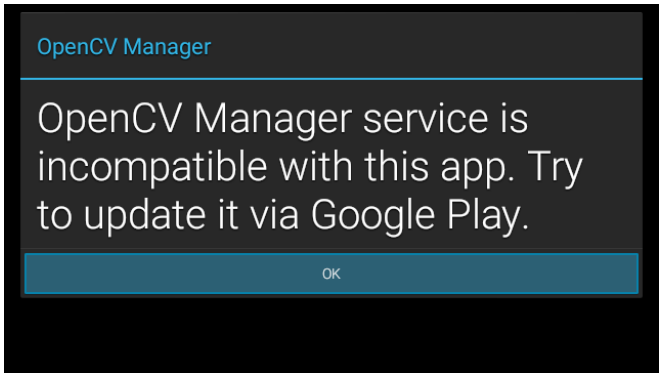

Figure 4: The OpenCv library 3.4.9. compiles, but is outdated to run on Glass 2

compile and work. So, after inclusion of the library 2.4.9 in the project, *the project indeed compiled*. But then again, when trying to run in on the Glass, the prompt window shows the option to install OpenCV Manager with predefined answer of NO. Selecting YES is not possible considering that the *OpenCV Manager application expects touch input event from mobile, and not tap event from Glass*. Considering that the OpenCV Manager is an application, the need for consent can be easily avoided using the install command through command line with Android Debug Bridge. After successful installation, running the project creates another prompt window (Figure 4) notifying the user that this *OpenCv Manager is not compatible*, and that it should update via Google Play.

The same *problem persists for all versions from 2.* edition* of OpenCV4Android libraries up to the most recent which is 2.4.13.6. Bearing in mind that the device for this paper was Glass 2 from the Enterprise Edition and *working samples were developed with Glass 1 from Explorer Edition*, the automatic update of the Manager is not possible because Glass EE software does not have Google Play, and as we could see previously, the versions from 3.1.0 and the following are not supported.

3 The solutions

After the difficulties have been presented, for both Google Glass 2 and OpenCV, in this section will discuss step by step solutions on how to resolve each one of them.

3.1 The Google Glasses solution

As we could see the White Screen of Death on Google Glass can occur for many reasons. If the problem is not of a hardware nature, the following solutions might resolve it:

- Hard reset - holding down the power button for 15 seconds
- Overnight Stay - shutting down the device and letting it charge overnight
- Power Down – turning the device off and letting it in that state for a while, optimal two days and turn back on.

If the problem still persists, the following solutions can be tried, but it is *very important and limited to the fact that the developer mode was turned on before White Screen of death occurred.*

If that is the case, we can *try to erase user data and cache* from Google Glass, like shown on Figure 5:

- Open an ADB Shell on Your Computer - this can usually be found at `C:\Users\<Your_username>\AppData\Local\Android\sdk\platform-tools`. Holding shift and right-clicking while inside the directory that contains adb and fastboot executable, open the PowerShell.
- Check whether Glass has enabled debug mode - run the command: adb devices. The output will be all devices connected to the machine, if not the new daemon will start. If this command doesn't work, the device doesn't have debug mode enabled, and further steps cannot be performed.
- Put the Glasses into Bootloader mode: run the command: adb reboot-bootloader. The Glass will turn off and shortly after display blue screen with Bootloader options.
- Check that the Glass are connected properly and that fastboot command work - run the command: fastboot devices. The output should be the same as in second step.
- Unlock the bootloader: run the command: fastboot oem unlock. The output will be a prompt asking you to confirm the unlocking. Enter yes and confirm.
- Erase user data: run the command: fastboot erase userdata.
- Erase cache: run the command: fastboot erase cache.
- Lock the bootloader before rebooting: run the command: fastboot oem lock.
- Reboot the Glass: run the command: fastboot reboot. You should see that the Glass booted the software successfully like on Figure 6.

As the previous solution didn't resolve White Screen of

```
PS C:\Users\Jelena\AppData\Local\Android\sdk\platform-tools> .\fastboot devices
0WP1A1AA15310153        fastboot
PS C:\Users\Jelena\AppData\Local\Android\sdk\platform-tools> .\fastboot oem unlock
FAILED (remote: Already Unlocked)
Finished. Total time: 0.111s
PS C:\Users\Jelena\AppData\Local\Android\sdk\platform-tools> .\fastboot
fastboot: usage: no command
PS C:\Users\Jelena\AppData\Local\Android\sdk\platform-tools> .\fastboot devices
0WP1A1AA15310153        fastboot
PS C:\Users\Jelena\AppData\Local\Android\sdk\platform-tools> .\fastboot erase userdata
Erasing 'userdata'                                 OKAY [ 26.584s]
Finished. Total time: 26.776s
PS C:\Users\Jelena\AppData\Local\Android\sdk\platform-tools> .\fastboot erase cache
Erasing 'cache'                                    OKAY [  3.675s]
Finished. Total time: 3.859s
PS C:\Users\Jelena\AppData\Local\Android\sdk\platform-tools> .\fastboot reboot
Rebooting
Finished. Total time: 0.102s
PS C:\Users\Jelena\AppData\Local\Android\sdk\platform-tools>
```

Figure 5: Checking that Glass has entered Bootloader, erasing user data and cache, and booting the device again

Death,we *tried to flash Google Glass with last working EE image*. This solution actually differs from previous one from the seventh step and further. It is necessary to go through the steps 1-6 and then perform the following ones:

- Download the factory images – after downloading EE images, place them in the same folder where is the ADB shell, in other terms, in folder `C:\Users\<Your_username>\AppData\Local\Android\sdk\platform-tools`
- Flash the Glass - run the command: flash-all. When the flashing is done, the Glass should automatically reboot and you should see the booted software successfully like on Figure 6.

Sometimes the command execution adb reboot-bootloader can stuck on it. This *hardware technique to put the Glass in bootloader mode*, is suggested by one of the Google representatives, and it follows [11] :

- Power down the device by holding the power button for 15 seconds.
- Press and hold the camera button. Keep holding it until step 5.
- Briefly press the power button.
- Wait until the LED solidly illuminates.
- Release the camera button. After this step the device should be visible to bootloader mode.
- Connect the device to the machine, and check that the Glass are connected properly, by running the command from PowerShell: fastboot devices. The Glass device should be listed just like on Figure 5.

In case of incompatible drivers, the ADB shell will execute the adb reboot-bootloader, but the device won't be recognized as connected, and it will reboot itself shortly after. As the SDK Google USB driver doesn't include all necessary drivers for

[11] https://stackoverflow.com/questions/17519865/having-issues-seeing-glass-in-fastboot

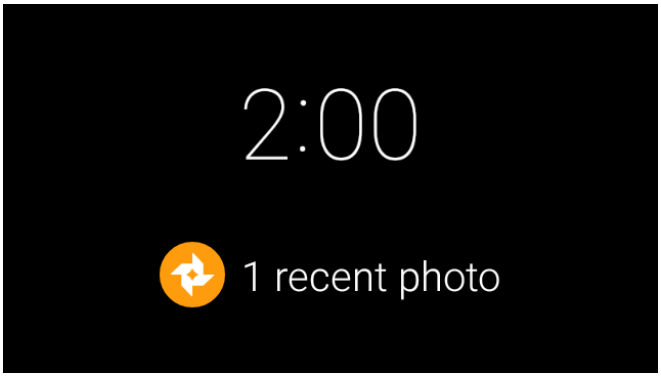

Figure 6: Glass successfully booted the software

the Glass, the *missing drivers can be installed via adb-setup-1.3.exe*. [12]. After theirs installation, in the Device Manager on Windows the Glass device needs to be chosen, and for it manually selected the Bootloader driver. After the Bootloader driver installation for the Glass, the commands from Figure 5 can run successfully, and we can see on the screen that the Glass booted its software right, such as the Figure 6 is showing.

3.2 The OpenCV solution

As we could see for the OpenCv4Android customized library for Android in previous section, all its versions up to the most recent from 2.* edition are not supported by Glass 2 software, because theirs Open CV Manager is outdated. *The Open CV Manager from 3.* edition of the library should be working*, but all the versions from 3.1.0 and forward use the API 21 as the targeted version of Android, hence the library cannot compile in the Glass project as the API must be 19

[12] The video explanation can be found on this link : https://www.youtube.com/watch?v=maLWScgXflk , while the program can be found on the following link https://forum.xda-developers.com/showthread.php?t=2588979

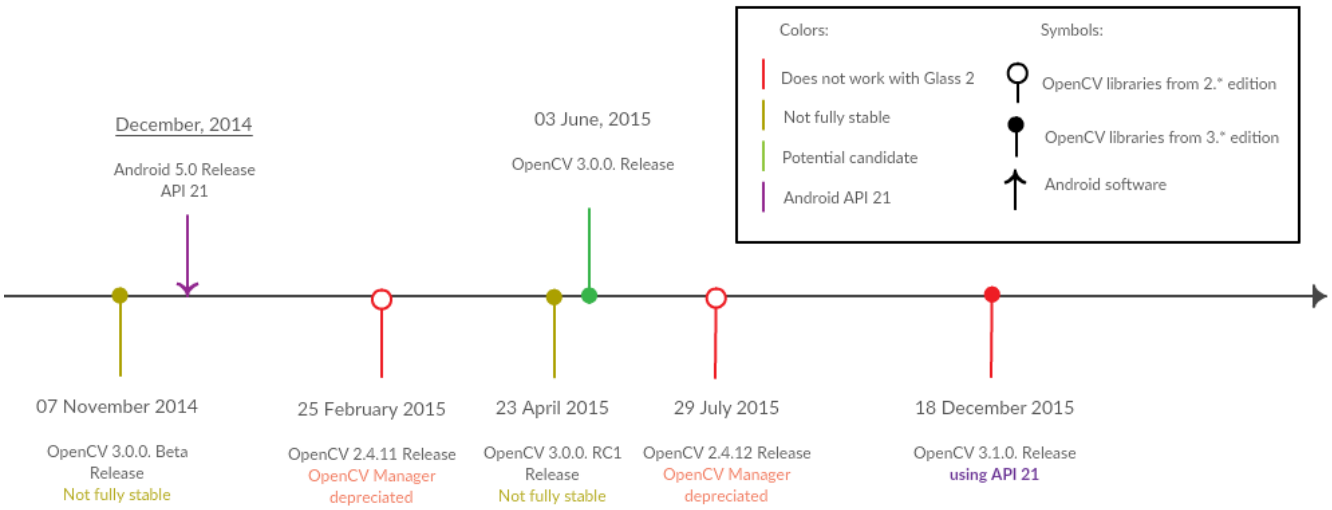

Figure 7: The timetable of the releases of OpenCV libraries and the Android version 5.0, with color indicating whether the version is supported or out of interest

for Glassware development. It is possible to use the desired algorithms from the OpenCV library through Android Native Development Kit, by using their native form written in C/C++. *But what happened with the first stable release of OpenCv4Android from 3.* edition, was the API 21 used for its development?*

If we look at the time line that the Figure 7 is showing, we can see that the first stable release from 3.* edition was in beginning of the June 2015, while the API 21 (Android 5.0 Lollipop) was released just a couple of months before, in December 2014. *Because of a relatively short time distance, the assumption is that the targeted API is not higher than 19, and that Open CV Manager could properly initialize on the Google Glass.*

In order to add the OpenCV4Anroid 3.0.0 library to the existing Glass project, we should:

- In the Android Studio we should first import the library as the module - `Menu:/File/New/Import_Module:` with the source directory `{unzip-dir}/sdk/java` where unzip-dir is location of the directory where the unzipped OpenCV4Anroid library 3.0.0 is. If we check the build.gradle file, we can see that for this library the compiled version is 14. *The assumption is still valid.*
- Now we can add the module as a dependency to Glass project - opening `Menu:/File/Project_Structure` in the *app module* on the Dependencies tab, we should add :openCVLibrary300 as a Module Dependency. And then, to fix things, we shloud clean the project, sync the Gradle and then build it. The output is showing the successful build.
- Now we can add the natives for different platform support - copying from `{unzip-dir}\sdk\native\libs` directory and everything under it to our project in `ProjectName/OpenCVLibrary310/src/main/` and renaming it to jniLibs.
- In order to load and initialize OpenCV library from the current Glass project we should add the following piece of code as shown on Figure 8 to the onCreate() method of MainActivity.java.

```
if (!OpenCVLoader.initDebug()) {
    Log.e(this.getClass().getSimpleName(), "  OpenCVLoader.initDebug(), not working.");
} else {
    Log.d(this.getClass().getSimpleName(), "  OpenCVLoader.initDebug(), working.");
}
```

Figure 8: Invocation of OpenCVLoader.initDebug() in onCreate() method will load and initialize the library

When we run the project on the Glass, we can see in the Logcat output that the OpenCV has been loaded successfully and that the Glass application is working (Figure 9).

The assumption was correct, and we finally had a running OpenCV application for Google Glass 2.

4 The application - proof of concept

In the previous sections, we have seen some of the challenges enabling Google Glass 2 for application development,and challenges enabling OpenCV for Glass 2 development. This section is dedicated to an application called "Room color helper",thought to be *proof of the concept that combining the Glass 2 and the OpenCV is feasible*, and will be also discussed

Figure 9: OpenCV 3.0.0. is running on Google Glass 2

the challenges of developing it.

Looking from the point of view of augmented reality and considering the power of the algorithms of OpenCV, *the objective* of the application emerges naturally: *seeing the results of camera stream processing in real-time*.

Bering in mind that Google Glass 2 is orientated to business as the clients, and its employees as users, the application "Room color helper" is developed for home interior designers, for facilitating the problem of room idea colors. This application enables to the users to see, in real-time, painted objects with the desired color, where the color is previously chosen from the user's environment.

The application exploits Google Now, by enabling the users to launch the application with a voice command " Ok Glass, Help me Decorating".Once the application launched, designers can choose the desired color from the surroundings and select which object to paint with a tap gesture, and apply the picked color on the object with long press with two fingers, save the current view with one finger long press and start over again with swiping right. The sequence of the application execution is shown on Figures 10 to 14.

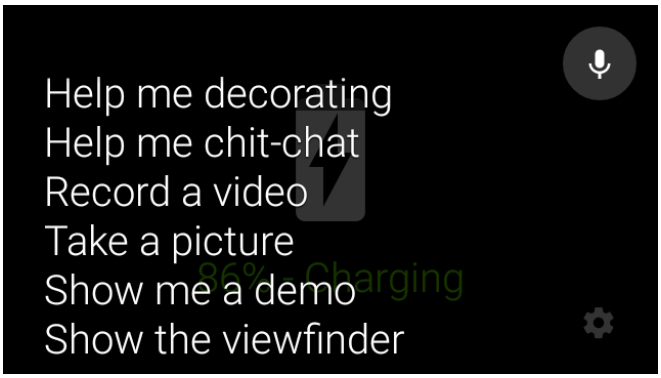

Figure 10: Launching the application with voice command

Figure 11: Welcome screen waiting for gesture tap for coosing the color, and it will be picked from the center of the screen.

4.1 Challenges

The biggest challenge for developing this application comes from the famous *unresolved bug of the OpenCV wrapper for camera view*[13], called JavaCameraView, and it concerns the screen rotation. When launching the application, the camera view is tilted 90 degrees in portrait mode, leaving the vertical black gaps on the sides of the view, regardless of Android's Manifest layout setup.

In order to see the processing results in real-time, the activity that is responsible for handling the frames must imple-

[13]https://github.com/opencv/opencv/issues/4704

Figure 12: Color is selected and can be seen as RGBA value in top left corner with HSV spectrum. The application is waiting for user to choose the object with another tap.

Figure 13: After selected object, its contours are shown and user is prompt to apply the color

Figure 14: The user can see the result and start again

ment one of the OpenCV's camera view listeners, while the instantiation of the view *requires* OpenCV's wrapper JavaCameraView for camera. Why? JavaCameraView is responsible for initialization of the device's camera and it extends CameraBridgeViewBase class, which is the base class for the interaction with Camera and OpenCV library. The main responsibility is to control when to enable the Camera, process the frame, call external listeners to make adjustments to the frame and then draw it.

Therefore, the problem with the rotation should be resolved easily via Android SDK accessing the camera instance in JavaCameraView, setting the display orientation and invoking the SurfaceHolder to set the display. But, invoking the SurfaceHolder locks the canvas on camera that is needed to be processed in the base class - CameraBridgeViewBase, in order for OpenCV to process the frames. This results in a rotated display, but without any visible output from OpenCV.

A second approach is from an OpenCV perspective, to rotate the frames before making the adjustment and its processing. We can rotate the frames in the Activity that implements the CvCameraViewListener2, because it implements the method for adjusting the frames `onCameraFrame(inputFrame:CvCameraFrame)`. We need the center of the received frame and the rotation angle to make the rotation matrix, and make a new frame with the same width and height by using the rotation matrix. How this approach results?

We have the display rotated, but the vertical black gaps remain from portrait view, and we also have two additional on the top and bottom, as the residue of performed rotation, and moreover the available view is very small. Why? Even though the frame that Glass camera captures, that is received in JavaCameraView, is 640x360 (640 width and 360 height) we can see that supported preview sizes returns portrait 360x640. Therefore, JavaCameraView will pick the best one for accommodating the received frame, 288x352. Hence, the rotation of the frame with size 288x352 will surely leave the gaps.

So, in order to fulfill the objective of real-time camera frame processing, it was needed to create our own view implementing the base class CameraBridgeViewBase, modeled on JavaCameraView, where we will rotate the frames before they are supplied to the listener, with the right resolution. The new view is called *CVGlassCameraView* with which the user have right display orientation and with which the view fill the layout, *but the cost is on processing/memory*. Rotating every frame is costly by itself, therefore if the application needs to much frame adjustment, the FPS slows down, the Glass heats up too much, as well as that there is no space to forgot `.release()` on Mats (memory leaks).

5 Conclusion

As we have seen from previous section with "Room color helper", developing OpenCV application for Google Glass 2 is feasible.

We must have into count that Google Glass 2 is not oriented to developers and that there is a very small support community, making it more demanding to resolve the encountered obstacles.

Furthermore, employing OpenCV with relaying only on published support for Android platform, is limited to OpenCV 3.0.0.version and OpenCV bugs, which in case of making real-time camera stream processing applications signifies that the developer must open the OpenCV's "black box".

References

[1] Joseph Rampolla Greg Kipper. *Augmented Reality: An Emerging Technologies Guide to AR*. Elsevier, December 2012.

[2] Group of authors. *Mixed Reality and Human-Robot Interaction*. Springer Science and Business Media, 2011.

[3] ZhiYing ZhouJayashree KarlekarDaniel HiiMiriam SchneiderWeiquan LuStephen Wittkopf. Robust Pose Estimation for Outdoor Mixed Reality with Sensor Fusion. *Universal Access in Human-Computer Interaction. Applications and Services*, pages 281–289, 2009.