Ατομική Διπλωματική Εργασία

DEVELOPING AN APPLICATION FOR WEARABLE SMART-GLASSES THAT HELPS THE VISUALLY IMPAIRED TO READ NORMAL BOOKS

Θαλής Παπακυριακού

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2017

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Developing an application
for wearable smart-glasses
that helps the visually impaired
to read normal books

Θαλής Παπακυριακού

Επιβλέπων Καθηγητής: Αντρέας Παμπόρης

Όνομα Καθηγητή: Σαμάρας Γιώργος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2017

Special thanks

I would like to thank Dr. Samaras George for encouraging me to take up a dissertation theme he provided, and for providing such an interesting theme indeed. I would also like to thank him and his lab for purchasing the really expensive Vuzix M100 augmented reality smart-glasses to aid me in completing this dissertation.

Additionally, I would like to thank Dr. Andreas Pamboris for mentoring me through the whole process of completing this dissertation. He was always there when I needed something and always came up with interesting ideas, proposals and helpful tips.

Finally, I would like to thank many of my fellow students and friends, because they provided the best brain-storming sessions every time I brought something up about the progress I had made or the problems I was facing.

Abstract

One of the difficulties visually impaired people face (whether partially or completely blind) is reading.

This diploma project is focused on explaining the process of creating an Android application for the Vuzix M100 augmented reality glasses, than helps any visually impaired person to read a normal book. The glasses, in combination with the application, eventually ended up with the following capabilities:

- They can read the page of a book with their camera and speak it to their user.
- They can receive voice commands, through which they can speak previously listened-to book pages to their user again.
- They have a “beeping” tracking mechanism for a blind user to be able to correctly position himself and the camera of the glasses above a book page.

We call the completed product “Book Eyessistant”. It is essentially a complete product for any visually impaired person that wants to read a normal book. Here are just a few of its positive traits and characteristics:

- It is suitable even for completely blind people (no need to look at the screen).
- It is simple to use and functions independently of any other device. It requires only the 4 buttons of the Vuzix M100 and speech commands from the user to be controlled.
- It uses a smart algorithm to make sure what it speaks is as close as possible to the correct contents of the page read.
- It has the ability of speaking a page to the user and at the same time read another page, implementing a kind of pipelining that saves the user time.

Apart from the process of creating the application from start to end, this diploma project additionally focuses on explaining the different considerations and heuristics that defined certain traits of the application, traits that make it stand out compared to other applications of its kind.

Table of contents

5) Google's "Ocr-reader"

6) The "Book Eyessistant" (Creating our application)

7) The "Book Eyessistant" (The final product)

A) Appendix

0) Prologue – Introduction to Augmented Reality

Technology has always been used to make the lives of people with disabilities easier. Lately, with the rapid growth of augmented reality, the possible ways of helping these people have been greatly expanded, since augmented reality is focused specifically on that, enhancing the senses of a person. By definition, augmented reality is the means of enhancing a person's sensory experience (vision, hearing etc) using technology, creating an experience that is the combination of information directly from the environment and from the technology attached to us.

"Augmented reality – where the real world is overlaid with information from the digital world – hugely expands the variety of problems students can study."

-Unknown

Our objective during this diploma project was to help the visually impaired people using augmented reality. **Specifically, we wanted to help the visually impaired people to read a book using augmented reality smart-glasses**, and we achieved that using the Vuzix M100.

"Since we cannot change reality, let us change the eyes that see reality."

-Nikos Kazantzakis

Of course, the great writer and fellow compatriot Nikos Kazantzakis wasn't referring to either augmented reality or the visually impaired when he spoke that quote, but one can easily see how it can be interpreted to serve that purpose.

This diploma project describes the timeline and sequence of events that eventually led us to set the above as our objective, and fulfilling it. But before describing the whole process from the very start to the end, let us dig a bit deeper into augmented reality and its history:

According to Wikipedia, "*Augmented reality (AR) is a live direct or indirect view of a physical, real-world environment whose elements are augmented (or supplemented) by computer-generated sensory input such as sound, video, graphics or GPS data*".

The first attempt to implement an augmented reality system dates back in 1968, when Ivan Sutherland developed the first head-mounted display system, which used computer generated graphics to show users simple wireframe drawings.

6 years later, in 1974, Myron Krueger built an 'Artificial reality' laboratory called "the Videoplace", which combined projectors with video cameras that emitted on-screen silhouettes, surrounding users in an interactive environment.

Interestingly enough, **up until 1990 the term "augmented reality" didn't officially exist. It was coined as a term during that year by Boeing researcher Tom Caudell**.

By 1998, augmented reality started making its way into entertainment, too, as technologies advanced. For example, that same year the "1st & Ten line" computer system was used during a live NFL game to cast a yellow "first down" marker on the field.

The next great evolution for augmented reality is speculated to have happened around 2013, when augmented reality hardware and software made a leap towards consumer audiences for the first time.

In 2014 Google announced shipment of "Google Glass" devices for consumers, thus starting the trend of wearable AR. That same year, the greatest AR investment to date had been made by Magic Leap, reaching an amount of 50 million US dollars. [1]

Today, augmented reality and virtual reality investment reach 1.1 billion US dollars, and many think augmented reality will actually be way bigger than virtual reality (like Apple's Tim Cook). During CES (Consumer Electronics Show) 2017 some amazing AR technologies were presented, like ODG's new models of the R-series smart-glasses (the R8 and R9), Lumus 55-degree-field-of-view AR display prototype etc. [2]

[1] The History of Augmented Reality (Infographic) | The Huffington Post
http://www.huffingtonpost.com/dennis-williams-ii/the-history-of-augmented-_b_9955048.html

[2] Here's a Virtual and Augmented Reality Check From CES 2017
http://uproxx.com/technology/virtual-augmented-reality-ces/2/

1) Selection of wearable equipment

The very first step was picking the most suitable pair of AR glasses for our dissertation. At the time (early 2016), we had the following possible options [1]:

1. CastAR
   This set of glasses consists of a camera that scans your surroundings and adjusts accordingly to project images through two micro-projectors installed on top of the frames. It has the ability to **add augmented reality objects to your view that can be moved around using a physical stick called the MagicWand**, and thus fun augmented reality games could be played using it.
2. Moverio BT-200
   The Moverio BT-200 consists of a front facing camera, a motion sensor, a built-in Dolby Digital Plus for sound, GPS, microphone, compass and projectors. **It needs to be connected to an Android device at all times, which is quite the disadvantage**. It works by projecting images at a resolution of 960×540 to the transparent glasses, allowing you to watch videos, play games, navigate and plenty more without losing sight of the physical world.
3. Meta
   The Meta glasses provide services such as motion tracking, 3D HD display and 3D surround sound, and have a camera and quality lenses. You can use gestures to move augmented reality projected objects that overlay the real world. **Their most advantageous attribute is their supreme 3D modeling ability.**

[1] 10 Forthcoming Augmented Reality & Smart Glasses You Can Buy – Hongkiat http://www.hongkiat.com/blog/augmented-reality-smart-glasses/

4. Vuzix M100

   This set of glasses relays information directly to you from a wearable monocular display. It has a lot of attributes that can be compared to the no-longer-available-for- purchase Google Glass, and a comparison between the two can be found at the link below [2]. **It supports user-made Android applications and has a 5MP camera and 1080p video.** It also supports Speech and Gesture Recognition. It's mostly focused on enterprise, commercial and medical applications.

5. Laster SeeThru

   This set of glasses supports wireless & communication with smartphone, localization and navigation, and has features like a head tracker, contacts access from phone etc.

   Instead of relying on a camera to gather information about your surroundings, it relies only on its own series of location plus a GPS. **It focuses almost entirely on sports and activities** like biking, parachuting and yachting.

6. Icis

   The greatest advantage of this pair of glasses, compared to others, is that **it contains a camera, speaker, microphone, battery and circuit board, all hidden**. They look like a pair of normal glasses. It can easily be connected to smartphones running Android, iOS and Windows platform using a Bluetooth connection.

7. ORA-S

   ORA-S comes packed with sound through an audio jack, microphone, orientation sensor, camera, Wi-Fi, 1.2GHz dual core ARM Cortex microprocessor, 1GB DDR memory with 4GB Flash memory etc. **It also enables the user to view the world from a 20 degree angle**.

[2] Hardware Comparison: Vuzix M100 vs Google Glass | Google Glass, Smart Glasses News, Vuzix Smart Glasses | Glass Apps Source http://www.glassappsource.com/google-glass/hardware-comparison-vuzix-m100-vs-google-glass.html

8. GlassUP

   This pair focuses on being your second output screen (other than your phone), thus enabling you to see all the notifications and messages on your glasses screen, without interrupting you from what you are looking at the moment. It can also send emails and comes with a monochrome display. Additionally, **GlassUp can also be used in aiding the hearing impaired and even receive translations display when talking in different languages**.

9. Atheer One

   Atheer One is primarily focused on gesture recognition. It can display 3D graphics when connected with an Android device and lets you interact with the 3D environment using your hands. **It's one of the best when it comes to gestures and 3D objects.**

10. K-Glass

    K-Glass focuses on replicating the process of how our brains form our surroundings.

    It categorizes relevant and irrelevant visual data, replicating the human brain's ability. For example, at a restaurant, K-Glass can prompt you with an overlay menu of that restaurants food.

Although, under usual circumstances, picking a pair of glasses comes after doing the research and deciding what we want to work on, in this case we picked our pair of glasses early on, and our biggest priorities where the following: We wanted the AR glasses to be as highly customizable/programmable as possible, an Android or iOS operating system was preferable, and we also wanted them to provide utilities like Speech Recognition etc. After careful consideration, we decided to go with the Vuzix M100 AR glasses because they proved the most suitable for our preferences: you could program Android applications for them, they provided utilities like Speech Recognition, Gesture Recognition and high quality video, and they kept almost every initial option available, which means their hardware or capabilities didn't limit our vision and scope of the dissertation.

2) The Vuzix M100

2.1) Introduction to the Vuzix M100

So now that we've picked our augmented reality glasses, let's take a more detailed look at them and analyze their capabilities. Note that we will only be concentrating on the important parts for the purposes of this dissertation, or we will be explaining some things derived from the Vuzix M100 Product Guide in a better and more helpful way. For the full product guide, please visit https://www.vuzix.com/support/Download

"The Vuzix M100 Smart Glasses is an Android-based wearable computer, enhanced with a monocular display, speaker, and a high definition camera able to capture video and still image pictures. It can be used as a stand-alone device, paired to work cooperatively with an Android device, or to connect wirelessly to other devices and the Internet."

-The Vuzix Corporation

2.2) Specifications

CPU:	OMAP4460 at 1.2GHz
Memory:	1GB
Storage:	4GB flash, 32GB with MicroSD
Battery:	550mAh rechargeable internal battery, up to 6 hours of usage
Display:	6:9, WQVGA, full color display with 428 x 240 resolution
Camera:	5MP photos, 1080p video
Sensors:	3 DOF gesture engine (L/R,U/D,N/F), Ambient light, GPS, Proximity, 3-degree of freedom head tracking, 3 axis gyro, 3 axis accelerometer and 3 axis mag/integrated compass
Connectivity:	Wi-Fi and Bluetooth
Sound:	Ear speaker and a noise canceling microphone
Controls:	4 control buttons, remote control app, voice navigation and gestures

2.3) Assembling and wearing the Vuzix M100

The following instructions are not what we have been initially instructed by the Vuzix M100 Product Guide, but what we, ourselves, found to be the most efficient and safe way to assemble the Vuzix M100 after a lot of assembling and testing we did ourselves:

The package includes two sets of safety glasses, one looking like the ones used by construction workers, and one looking more like the glasses a visually impaired person would use daily. The first set is more enduring, stable, and fit, but lacks appeal, while the second set of glasses is worse than the first one on the 3 properties mentioned before, but makes up for it in appeal. Let's just say the second set wouldn't attract so much attention to your person if wore outside for a walk, unlike the first one.

The package also contains 2 mounting brackets with the letter 'L' or 'R' written on them, one for each side of the glasses. Which one to use depends on the preferable eye of the user. After picking the set of safety glasses you want and a preferable eye:

- Slide the corresponding mounting bracket onto the safety glasses stem as far as it can go, with the mounting ring to the rear and the holder to the front.
- Take the M100 Glasses and place their arm in the holder of the mounting bracket.
- Snap the ear speaker of the M100 Glasses into the mounting ring of the bracket. It is very important that this step is done after placing the arm of the M100 Glasses into the holder (for safety purposes).
- If it's your first time using the Vuzix M100, charge them using their special charger and the USB cable provided with the package. Turn on the M100 Glasses by pressing the "power button" (the only button on the bottom-side of the M100). Now you can adjust the M100 so that, after wearing the safety glasses, the totality of the screen can be seen with your eye just by looking directly at it. There are various ways to do that: Slide the mounting bracket further to the back/front, grab the screen of the M100 and rotate it upwards/downwards, or grab the arm of the M100 and bend it to the left/right. The latter could be a bit dangerous for the integrity of the M100 Glasses, so if it can be avoided, it is advised to (during the tests performed by us, the arm never needed to be adjusted).
- For even more stability and comfort, you can optionally insert the "speaker cone" on the ear speaker or slide the "ear lock" onto the safety glasses to hold the M100 in place if you choose to turn your head downwards while wearing them. The latter is advised to be done after wearing the safety glasses.

2.4) Disassembling the Vuzix M100

The safest way we found for disassembling the Vuzix M100 was to first carefully remove the ear speaker from its slot by using your nails to push the mounting ring out of the speaker, being careful not to scratch the speaker itself with your nails in the process. Then you can remove the arm of the M100 from its holder and slide the mounting bracket out of the safety glasses.

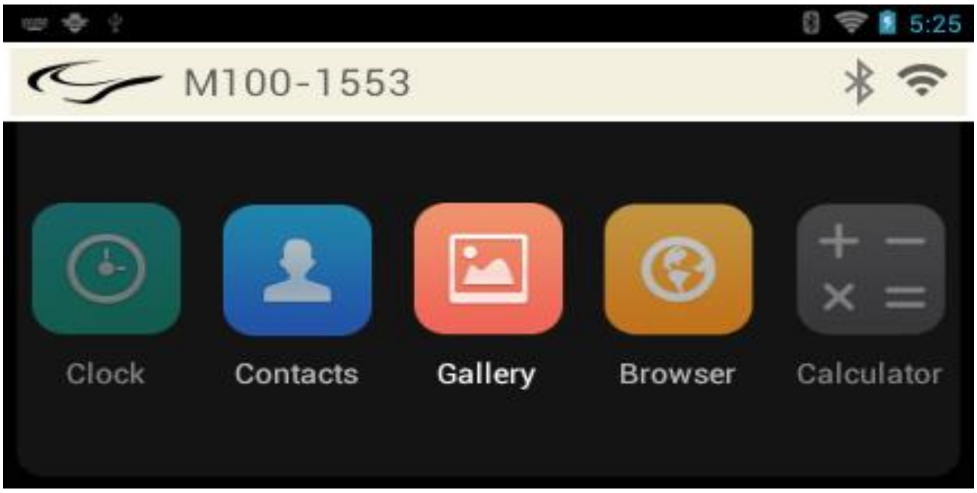
2.5) Controls

There are a total of 4 buttons on the Vuzix M100, 3 on the top side (ACTION, BACK, FRW) and 1 on the bottom side (POWER):

ACTION: Long press: Android BACK button’s default action
Short press: action/confirm
BACK: Long press: Android HOME button’s default action
Short press: directional back (doesn’t pause/destroy applications)
FRW: Long press: Android MENU button’s default action
Short press: directional forward
POWER: Long press: Power off the Vuzix M100
Short press: Sleep mode

Because 4 buttons are hardly enough to comfortably control an Android device that doesn’t support a touch screen, the “Vuzix Manager” app (available at Google Play Store) can be installed to any Android version-compatible phone and, among other functions it has, can be used as a “substitute touch screen” with which you can control the glasses more easily, for example to write long text where it’s needed.

The Vuzix M100 can also be controlled through voice commands if “Voice Recognition” is enabled, but loud places with a lot of chit-chat cause the Vuzix M100 to seriously malfunction and perform random actions, so navigating the menus through voice commands didn’t seem like a good idea to us.



2.6) Connecting the Vuzix M100 to the internet and registering them

It is necessary for this process to be explained, since any Android application we will be developing might need to have internet access to work. Additionally, this process is necessary for the user to be able to handle/update the Vuzix M100 through their “Vuzix Manager” program on the computer. The instructions provided by Vuzix are not that detailed, so we thought it was appropriate to give our own instructions derived from experience:

- Power up the Vuzix M100.
- Install the “Vuzix Manager” app, available at Google Play Store, to any compatible version Android phone.
- Long press the FRW button on the glasses and select “Ensure Visibility”. Can also be done through Settings -> Bluetooth
- Press the “Find Glasses” button on your phone and confirm that phone and glasses have been connected.
- Connect to any available Wi-Fi through Settings -> Wi-Fi. To complete usernames and passwords use the phone, it’s a lot easier than using the Vuzix M100 buttons.
- Press the “Vuzix Store” button on your phone and register at the webpage displayed. To make sure the following steps will work, you better register at the

Vuzix’s European website as both a developer and a normal member, preferably with the same credentials. Stay logged in!

- While at the “Home” page of your phone’s Vuzix Store, go to “Device Manager” and register your glasses’ serial number (for example, M00100****). You should now be able to see the device on your list, but its status will be “NOT CONNECTED”.
- Go to Settings -> App Manager on your glasses and choose “Enable app manager”. Try either “http” and “https” options if one or the other doesn’t work. If all went well, your device’s status should now be “CONNECTED” once the Vuzix Store’s “Device Manager” page on your phone is refreshed.
- (Optional 1/2) Install the “Vuzix M100 System File Manager” program, available at Vuzix’s downloads page, on your PC and open it.
- (Optional 2/2) Connect the glasses to the PC using their USB cable. If all went well, the Manager should recognize the model. For any additional information on the use of the “Vuzix Manager”, please consult the guide provided through the program itself.

2.7) Version and software updates of the Vuzix M100

Just as we laid out the hardware specifications of the Vuzix M100, It is also important to lay out its software characteristics, too, to know the capabilities of the Android device we are using. The Vuzix M100 Android version is 4.0.4, with an API Level 15 (Ice-cream Sandwich). As for the system file of the Vuzix M100 itself, it is version 2.6, with the newest version available being 2.7 (available for installation through the “Vuzix System File Manager” program). Unfortunately, updating the system file resets the glasses to their factory settings, deleting any apps currently installed on them. So we decided there was no reason to further update the system file when the 2.7 update came out, since the risk of not being able to return to the current state of progress was too great.

3) Preparations before choosing a more specific theme for our dissertation

3.1) Android Studio and our very first Android applications

One thing was definite: We were going to take advantage of the fact that you can write your own Android applications for the Vuzix M100. As a result, before deciding an area to work on, we decided it would be a good idea to first become familiar with programming for Android, and with the special libraries available only for the Vuzix M100 (the “VoiceControl” library and the “GestureSensor” library). Becoming familiar with the above and discovering any potential problems/difficulties would make it easier for us to decide what can really be achieved and what not, when thinking about the area we would focus on.

First thing we did was becoming familiar with programming for Android using the Android Studio environment. We built a lot of test applications with the help of online tutorials and tried them out on emulators and actual Android phones. We also explored the many different menus of Android Studio, the settings that could be modified etc…

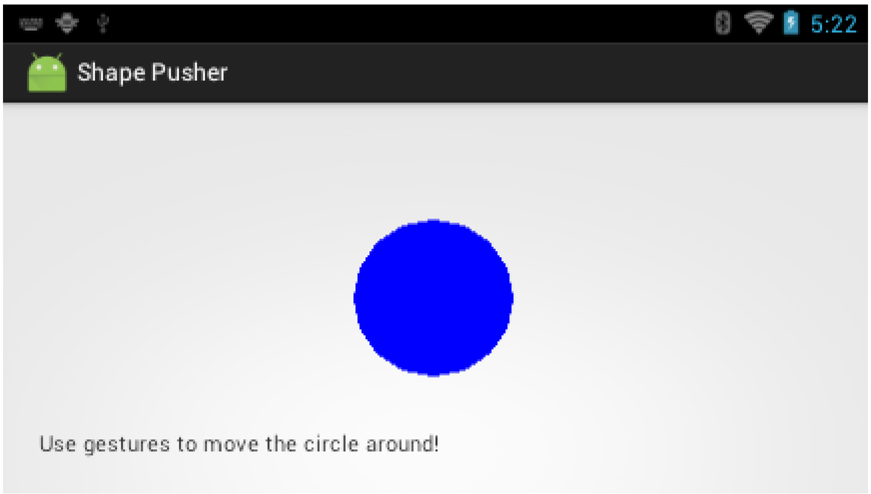
--FROM THIS POINT FORWARD, ANY TIME WE WILL BE REFFERING TO AN INSTALLATION OF AN APP ON THE GLASSES, WE WILL BE IMPLYING COMPILATION OF THE CODE AND DIRECT INSTALLATION OF THE APP FROM THE ANDROID STUDIO TO THE GLASSES USING A USB CONNECTION.--

By visiting the Vuzix website and logging in as a developer, one can download two sample applications/Android projects for the Vuzix M100: “Shape Pusher” is an application that presents the way the “GestureSensor” library works, and “Color Demo” is an application that presents the way the “VoiceControl” library works. Testing those out would make us even more comfortable in programming Android applications that made use of the additional capabilities of the Vuzix M100.

3.2) “Shape Pusher”

This application enables the user of the Vuzix M100 to perform certain gestures to move or resize a blue circle in the middle of a blank screen.

The code can only be compiled using the “Vuzix M100 SDK-Add on” compiler, the glasses’ personal compiler that can be used in Android Studio as an SDK add-on. This is the only compiler that can correctly compile the “GestureSensor” library.

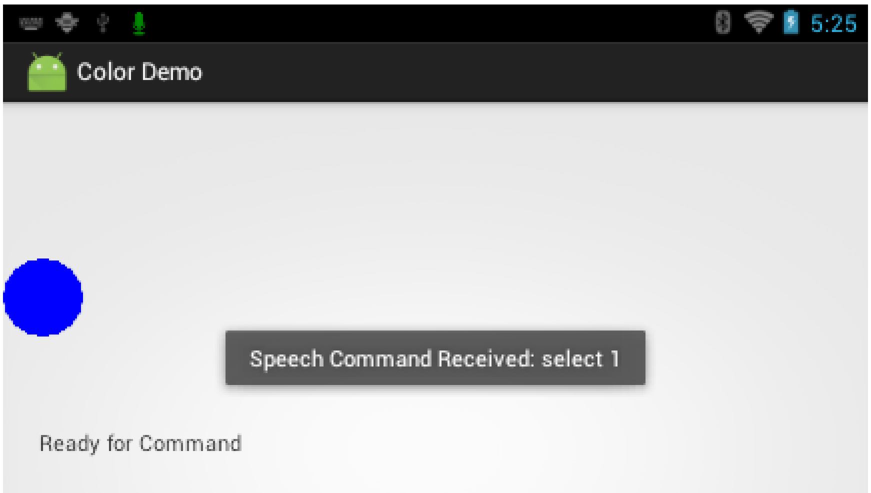


Use gestures to move the circle around!

3.3) “Color Demo”

This application enables the user to create new shapes on the blank screen, paint them and delete them, using only voice commands. For example, the command “new circle” spoken, creates a new circle somewhere where there is available space on the screen. Then, by speaking the command “select <number>”, the user can select any of the shapes currently on the screen (a shape's number is its ID by which it can be referenced, and each shape's ID is the number of total shapes that were on the screen when it was created + 1). After selecting a shape, the user can either paint it by speaking the command "color <color>" or delete it by speaking the command "delete".

Just like with the "ShapePusher", the code can only be compiled using the "Vuzix M100 SDK-Add on" compiler, since no other compiler can correctly compile the "VoiceControl" library.



Speech Command Received: select 1

Ready for Command

3.4) Compiler issues

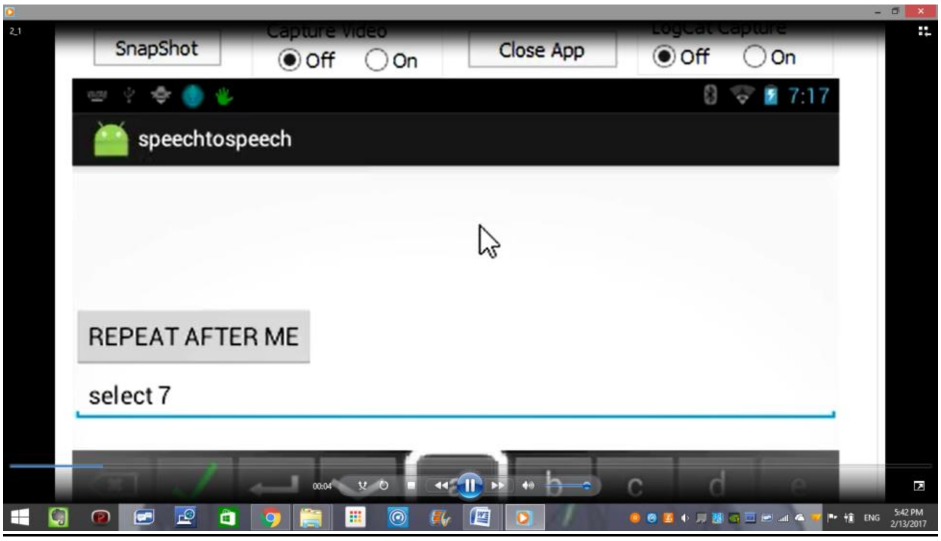
We were already facing an important issue: Any program that imported the Vuzix M100's specific libraries in its code had to be compiled using the Vuzix M100 compiler. However, the compiler itself is not updated regularly by Vuzix. This means that **any project in Android Studio that is built for an Android API Version higher than 15 (Ice-cream Sandwich) could be facing compatibility issues since the Vuzix compiler would not be able to recognize the new structure of both the project and any new code**. This has been confirmed through different tests. The best solution we derived at for when we needed to use the Vuzix libraries, (and consequently, the Vuzix compiler) was the following:

Any project compiled by the Vuzix compiler had to have the exact same, identical settings as the sample projects we described before. Additionally, any code that wasn’t compatible with the compiler had to be removed or replaced so that it is compatible with compilers for Android versions equal to or lower than API 15 (Ice-Cream Sandwich). An easy way to do that was to launch any of the sample projects and then just replace any code and layout file with the code of the program we wished to build.

3.5) Text-to-speech and our “Speech-to-speech” application

At that time, an interesting idea came to mind: We could use the “VoiceControl” library, along with any existing text-to-speech Android library, to build a “speech-to-speech” application, an application that recognized every word we spoke and spoke it back to us. Finding and learning how to use the Android text-to-speech library was easy, but at the time we had difficulties in making the system speak our words back to us immediately after we finished speaking, without any additional interference by us. So, for the moment, we decided to make the system speak the words back to us after a button was pressed.

This was an example where we used the method described above for a successful compilation with the Vuzix compiler. We opened the sample project “Color Demo” and modified only the code and a few settings to change the application’s ID etc. The compilation with the Vuzix compiler was completed successfully and the program worked like a charm.



Capture Video ● Off ○ On

LogCat Capture ● Off ○ On

select 7

4) Choosing an area to work on

4.1) Areas augmented reality solutions find use-cases on

The next big step was to find a good use case for our Vuzix M100. There are a lot of categories for use-cases of augmented reality; I present to you some of them [1]:

-Games: What better way to bring a game closer to reality, than introducing augmented reality in it? Apart from plain entertainment, augmented reality in games could also be used as a way to enhance a product's marketing, a way to bring a more immersive experience to the customer when advertising a product.

[1]Top 10 Augmented Reality Use cases | Augmented Reality Software and Solutions by Total Immersion

http://www.t-immersion.com/augmented-reality/use-cases

-Promotional purposes: Bring attention to a topic/idea by using augmented reality to promote it, since it's been proven that these days, anything backed up by a fun augmented reality application will eventually find his way through the internet and all kinds of networks, gaining popularity.

-Supporting a customer at a store: Augmented reality could provide a much better user friendly experience to any customer, helping him make better choices and making the whole process of buying what he desires easier. For example, museums have implemented augmented reality tutorials near their exhibits to help the visitor learn more about what is exhibited, easier and in a more entertaining way.

-Facilitating education and training: Augmented reality could be used in schools to demonstrate something in a more obvious and friendly-to-the-eyes way. It can also be used for the same purpose in businesses that aim to train their employees in using hard and complex machinery that just isn't as easy to do on a 2D piece of paper.

-Providing assistance and support to medical facilities: Yet another way to provide an easier and more helpful working environment for a doctor or a nurse, since, when under a stressful situation, providing more friendly-to-the-eye support could be just what's needed to make everything clearer.

-Supporting people with disabilities: Just like augmented reality can make our everyday lives easier, it can aid in reducing the everyday difficulties a disabled person faces due to their disability. For example, augmented reality can be used to aid the visually impaired in reading, in recognizing the environment, it can aid the hearing impaired in reading what is being told to them etc.…

A lot of ideas had come to mind, with one of them being using augmented reality in sports. Specifically, one of our very original ideas were augmented reality glasses for runners, which would show the runner statistics on the screen while he was running (heartbeat, km/h, distance travelled etc.). As long as the glasses were comfortable, I could easily picture myself using them, since a problem I often have is that I unintentionally speed down when I'm tired, even though I want to keep a steady pace. In the end, this whole "augmented reality in sports" idea was discarded, since we did find a lot of companies that worked specifically on augmented reality in sports, like Sensoria Fitness (and their Sensoria Running System which is a prototype of the idea described above), Senth IN1 (who focus on AR Cycling Glasses) etc.… Plus, the Vuzix M100 weren't the most comfortable glasses for the job.

4.2) Making the best out of our chosen augmented reality glasses

Before moving any further, he had to take into account that we chose "utility-based" AR glasses, which means they are better built for providing support and utilities, rather than an everyday fun accessory to use or for gaming. Not that they wouldn't be fit for the latter, but other glasses would be fitter for that, so we had to eventually focus on what they can do best. Let us reconsider the pros and cons of our Vuzix M100: They possessed the capabilities of an Android device, they were fully programmable, they had high quality speech recognition software, they had quite impressive hardware specifications (1.2 GHz CPU, 1GB RAM, 1080p camera), and everything else I mentioned above at the "The Vuzix M100" section. However, they weren't very comfortable for all-day use, and the screen was small enough that it required you to stay still and concentrated to read it. **So, to sum it up, we had some hardware with amazing computing capabilities and a wide variation of software customizations, but it lacked the appeal, comfort in moving around with and comfort in using it when tired.**

With the above in mind, the next big areas of AR applications we focused on were helping the hearing and visually impaired population. **Let us focus on the AR solutions for the hearing impaired we found on the market**, since they led us to choose focusing on an AR application for the visually impaired instead. Not because we didn't see prospect in an application for the hearing impaired, but because we saw more prospect in an application for the visually impaired. Let us remember that the Vuzix M100 glasses we had at our disposal are famous for their speech recognition system and its quality (Nuance Communication software), however we still went with the application for the visually impaired, since we thought these capabilities could prove useful in that kind of an application, too.

4.3) Previous work on AR solutions for the hearing impaired

4.3.1) Recent AR applications for the hearing impaired

1) British Sign Language on AR (augmented reality) Code

   An AR code is an image that contains information within it and can be decoded. In this application, a phone camera scans an AR Code, translates what is encrypted into British Sign Language, creates a video containing the translated motions/sign language in order and shows the resulting video to the hearing impaired user. The application can be supported by both an Android and an Apple mobile. Using an AR code means up to 3,000 words can be recorded into British Sign Language and the code will store that information. [1]

2) Sign Aloud

   Gloves that translate motion (USA sign language) into words to help non-deaf people understand sign language. They send motion to a computer via Bluetooth; the computer then searches its database for the most likely corresponding word (using a neural network-like behavior) and speaks it out loud. [2]

[1] British Sign Language on AR (Augmented Reality) Code for English publications - YouTube

https://www.youtube.com/watch?v=xAUbqE0AP9A, uploader: Tim Scannell

[2] https://www.facebook.com/viralthread/videos/554825808040454/?pnref=story1

UW undergraduate team wins $10,000 Lemelson, MIT Student Prize for gloves that translate sign language | UW Today

http://www.washington.edu/news/2016/04/12/uw-undergraduate-team-wins-10000-lemelson-mit-student-prize-for-gloves-that-translate-sign-language/

Inventors: Thomas Pryor and Navid Azodi, University of Washington students

3) The Live Time Closed Captioning System (LTCCS)

A piece of equipment that can be attached to any glasses and translates live speech into captions which are then shown on the screen of this equipment piece. The system consists of three components: a compact microphone that's clipped onto the user's clothing (as if they're a news anchor), a smartphone-sized Raspberry Pi/Adafruit-powered microcomputer that's kept in a pocket, and a Google Glass-like display. Although still in-progress, the inventor claims that the captions will appear almost instantaneously as the words are spoken, enabling the wearer to have a normal conversation with another person without pauses. [3]

4.3.2) Papers focused on AR solutions for the hearing impaired

Apart from completed or "in-progress" AR applications for aiding the hearing impaired, the following papers have been published around the same topic:

1) Although not an AR solution, but a VR solution, we decided to include this in this list. The authors of this paper suggest a multisensory approach to using VR for aiding the deaf-blind. Specifically, they were designing a navigation and support system for the dead-blind using a technique called the "tactile" technique. The navigation system itself is called "virtual leading blocks for the deaf-blind" and consists of a wearable interface for Finger-Braille, one of the commonly used communication methods among deaf-blind people in Japan, and a ubiquitous environment for barrier-free application, which consists of floor-embedded active radio-frequency identification (RFID) tags. The wearable Finger-Braille interface using two Linux-based wristwatch computers has been developed as a hybrid interface of verbal and nonverbal communication in order to inform users of their direction and position through the tactile

[3] Student-designed aid for the deaf converts speech to AR captions http://newatlas.com/live-time-closed-captioning-system/40078/, Inventor: Daniil Frants

sensation. They gave the metaphor "watermelon splitting" for the navigation by this system and verified the feasibility of the system through experiments. [1]

2) A free interactive Android mobile AR application called “Aurasma” was developed to facilitate the learning of geometry. Twenty-one elementary school children participated in an experiment. The results showed that the AR system could help the school children to finish puzzle game activities independent of the teacher’s assistance. With the use of AR display technology, the participants demonstrated improved ability to complete puzzle game tasks when compared to the use of traditional paper-based methods. Specifically, hearing impaired children can replace the need of a teacher guiding them or the need of written-down information with an application that offers a more user-friendly way to interact with the puzzle and uncover the mysteries in solving it.

   After reading through the paper, we realized that the application does not have much to offer, since it’s all about presenting videos to the little kids, but one idea that immediately comes to mind is the possibility for the phone to scan the current state of the game and suggest possible next moves or hints. Also, it would be really helpful if the application could provide a 3D representation of the puzzle, thus giving a clearer view of the situation to the kid. [2]

[1] Virtual leading blocks for the dead-blind: a real time way-finder by verbal-nonverbal hybrid interface and high density RFID tag space – IEEE Xplore Document http://ieeexplore.ieee.org/document/1310070/
2004, Authors: T. Amemiya, J. Yamashita, K. Hirota, M. Hirose

[2] Augmented reality in educational activities for children with disabilities http://www.sciencedirect.com/science/article/pii/S0141938215000566, 2016, Authors: Chien-Yu Lin, Hua-Chen Chai, Jui-ying Wang, Chien-Jung Chen, Yu-Hung Liu, Ching-Wen hen, Cheng-Wei Lin, Yu-Mei Huang

3) The authors of this paper developed an Android application to help people understand and learn the representation of letters in Sign Language. Specifically, the application scans a sequence of media cards containing a letter each, and displays on-screen the hand gesture that represents each letter in Sign Language, all at once. It basically teaches the user the sequence of hand gestures he needs to perform in order to "speak" a word (the sequence of media cards) in Sign Language. The application uses a "markerless" technique, as the authors call it, where a marker is a real world card and the camera can identify it as a "point of interest" and track it, translating the letter it contains in Sign Language. [3]

The main reason we didn't go with a hearing impaired AR application is this: **The first thing that came to mind when thinking about helping the hearing impaired was using speech-to-text software (which our glasses possessed) to translate any words spoken to text on the screen of the glasses belonging to the hearing impaired person. However, we found out that GlassUp is a brand of AR glasses currently on the market that focuses on exactly that, translating speech from many languages into text on the user's screen.** One can easily perceive that not many things can be done to compare our solution to glasses that are specifically made for that sole purpose. That, in addition to everything mentioned in the "4.2) Making the best out of our chosen wearable glasses" section, made us move on to the next area of AR applications…

[3] Augmented Reality Application of Sign Language for Deaf People in Android Based on Smartphone

http://www.sersc.org/journals/IJSEIA/vol10_no8_2016/13.pdf, 2016,

Authors: A.A K. Oka Sudana, Gusti Ayu Agung Mas Aristamy, Ni Kadek Ayu Wirdiani

4.4) Previous work on AR solutions for the visually impaired

4.4.1) Recent AR applications for the visually impaired

Having collected all the above information about the AR technologies for the hearing impaired out right now, we concluded that it would be wiser to move on with an AR solution for the visually impaired instead. Here is what we found in this area of AR applications:

1) "The vOICe for Android" is an Android application that scans the area from left to right using the camera of the Android equipment and produces certain sounds based on what it recognized, kind of like a Sound Language. It has other uses, too, but that is its main one. [1]
2) "OrCam MyEye" is a product from the company "OrCam" that was made available to the public in late 2016. It is a piece of equipment with a built in camera and a processor that can be attached to the frame of normal glasses. By having the user/wearer press a button or point at something, "MyEye" can recognize text in its camera's field of view (or where the user is pointing) and speak it back to the user, thus helping blind/visually impaired people read. Additionally, "MyEye" can also recognize faces that were previously stored in it or recognize objects whose characteristics were encountered before, in a manner, it can be somewhat "taught" to recognize them. [2]

[1] The vOICe for Android
https://play.google.com/store/search?q=the%20voice%20for%20android%20peter%20meijer&c=apps&hl=en
https://www.seeingwithsound.com/android.htm, 2016, Inventor: Peter Meijer.

[2] OrCam – See for yourself
http://www.orcam.com/, 2017, Inventors: Professor Amnon Shashua and Ziv Aviram

3) “eSight” is a pair of augmented reality glasses that, using a camera, captures everything the user is looking at, sends it to a small computer/processor attached to it and is then able to send back the same video, but modified, in order to help visually impaired people see everything in a clearer and more helpful way. For example, it gives the user the power to change the brightness of the video, the zoom, the contrast, focus on specific areas of the video screen etc. [3]

4) “AIPOLY” is an iPhone application that aids the blind/visually impaired to recognize objects. The application speaks the name of every object it recognizes when that object is in front of the camera. It can also read colors and it speaks seven different languages. The application won the “best innovation award” at CES 2017.

[3] How does eSight work?
http://www.esighteyewear.com/learn-more/how-does-esight-work ,
Inventor: Conrad Lewis

[4] Aipoly: Vision Through Artificial Intelligence
https://www.facebook.com/TechInsiderUK/videos/599478503592873/?autoplay_reason=all_page_organic_allowed&video_container_type=0&video_creator_product_type=2&app_id=2392950137&live_video_guests=0,
2017, Creators: Leon Ciciliano, Joe Daunt
http://aipoly.com/

4.4.2) Papers focused on AR solutions for the visually impaired

1) The iCare Reader is a paper published in 2003 that basically describes the 3 stages to designing an all-around complete solution for helping the visually impaired read a book by "listening" to it, using text-to-speech software. According to the paper, the first step involves the development of a table-based book reader prototype that is suitable for deployment in a library. The second step involves the development of a portable book reader prototype, which can be carried in a briefcase. The third step involves the development of a lightweight wearable text reader, which can be used for reading common print, such as signage, vending machines and hardcopy text. This text reader would consist of a miniature video camera in a pair of glasses and a belt-mounted PDA-sized computer. So far, no application with the name "iCareReader" has been identified by us that implements what the paper suggests. [1]
   (A YouTube video, also with the name "iCareReader", has been uploaded by 3 researchers which suggests the use of a pen as the camera that records the text that is to be translated to speech). [2]

[1] iCare-Reader - A Truly Portable Reading Device for the Blind http://s3.amazonaws.com/academia.edu.documents/3142715/The_iCare_Reader.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1487031500&Signature=yPCWQelMIvKMi4WvJxWO%2BSJ5KNo%3D&response-content-disposition=inline%3B%20filename%3DiCare_Reader_-_A_truly_portable_reading.pdf, 2003, Authors: Terri Hedgpeth, Mike Rush PE, Vivek Iyer, John Black, Mehmet Donderler, Sethuraman Panchanathan

[2] icare Reader - YouTube https://www.youtube.com/watch?v=oOCvTzSVz7E

2) NAVIG (Navigation Assisted by artificial Vision and GNSS) is an augmented reality system that helps the visually impaired navigate. It recognizes surrounding objects in real-time and stores helpful information about them in its database, thus creating a geographic information system. It can then provide the user with improved route selections and guidance, using spatialized semantic audio rendering.
   In the end, the paper also makes an introduction to a bio-inspired vision system that can help visually impaired users identify locations of objects via sound, perceptually. [3]

Based on the above findings and by reconsidering everything mentioned in the "4.2) Making the best out of our chosen wearable glasses" section, **we came up with the idea of building an application that would help blind/visually impaired people "read" by speaking to them the text in a book**. Moreover, the application would make use of the special capabilities of the Vuzix M100 to enhance this experience.
To confirm that such an idea was actually implementable, the important thing was to find image-to-text software that had good performance, meaning it recognized words in images well enough for us to use it for our application's purposes. And so our next stage of the research began…

[3] NAVIG: Augmented reality guidance system for the visually impaired | SpringerLink http://link.springer.com/article/10.1007/s10055-012-0213-6, 2012, Authors: Brian F. G. Katz, Slim Kammoun, Gaëtan Parseihian, Olivier Gutierrez, Adrien Brilhault, Malika Auvray, Philippe Truillet, Michel Denis, Simon Thorpe, Christophe Jouffrais

4.5) Image-to-text software and applications in the market

Here are the applications and software that we discovered while searching for the best image-to-text software that would fit our needs. During this stage of our research we also discovered some interesting papers regarding image-to-text applications, which are also presented below:

1) "Mobile vision" is a collection of Android projects/open-source code provided by Google that focus on real-time on-device vision technology. Some of these projects are focused on barcode scanning, face detection, text recognition etc. For the purposes of our selected dissertation theme, what is important to us is the project that focuses on text recognition, called "Ocr-reader". [1]
2) "Text fairy" is an Android application/OCR text scanner. It supposedly can convert images to text, correct the viewpoint of an image, edit extracted text, copy extracted text into the clipboard for use in other apps, convert the scanned page into PDF and recognize printed text from more than 50 languages. When we downloaded it and put it to the test, we realized that the whole scanning process was taking quite a lot of time to complete, and the accuracy wasn't very good either. Not that we were planning to use it in any way for our application, what we needed was code/a library and not a finished product. Still, applications like this could provide us with ideas, like functionalities for our own application, and something to compare our application to.[2]
3) "From scanned image to knowledge sharing" is a paper published in 2005 about the steps involved in creating a digital version of a book. The paper

[1] Mobile Vision | Google Developers https://developers.google.com/vision/, Google

[2] Text Fairy (OCR Text Scanner) – Android Apps on Google Play https://play.google.com/store/apps/details?id=com.renard.ocr&hl=en, 2016, Inventor: Renard Wellnitz

begins with the importance of such work, explaining that Google aims to digitalize whole grand libraries worldwide and has devoted huge amounts of money to this purpose. It then proceeds to present and explain the steps, which are [3]:

- Scanning and image processing
  - -From image to text with visual markup
  - -Data storage
- Document markup enhancement
  - -Structure Markup Enrichment
  - -Born Digital Documents
- Semantic Processing
  - -Document classification and clustering
- Identification, Indexing, Visualization and Dissemination
  - -Presentation
  - -Visualization

4) We discovered some websites that allow you to upload a PDF or an image file and convert it into a word document or text file. Some examples are "convertimagetotext.net", "onlineocr.net", "ocrconvert.com" etc…

5) Although the following information came from an article published in late 2011, it still provided us with helpful info about the OCR applications that existed at the time. "Google Goggles" is an Android application that can convert images of text to actual text, send that text to your clipboard for pasting, translate resulting text into tons of different languages, recognize business cards adding them to your phone as contacts etc. We did try the application out and found out that it lived up to its reputation: we could take pictures and "Google Goggles" would find the text in them and give us relevant information about what has been read, like search engines do. [4]

[3] From Scanned Image to Knowledge Sharing http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.72.5913&rep=rep1&type=pdf

[4] The Best Image-to-Text App for Android http://lifehacker.com/5829946/the-best-image-to-text-app-for-android

6) ABBYY Mobile OCR Engine is a software development kit (SDK) that allows developers to integrate optical character recognition technologies into Android apps. Thing is, for us to test it we had to get a free trial of the SDK and needed to fill up a form with various details and contact the company. We decided not to, since we already had discovered Google's "Ocr-reader" which would most probably be better and more helpful to us than this SDK. We, did, however, search for some videos and reviews online presenting ABBYY Mobile OCR Engine's performance, and found out that it had a pretty good performance after all. [5]

7) "Tesseract" is an OCR engine that started being developed way back in 1996 and continued as open-source software funded by Google since 2006 and on. It used to not have a GUI interface and was only usable through the command-line interface, but that has changed. Although its performance nowadays is considered to be very good, unfortunately it is not usable by an Android device, so it was not an available option for us. [6]

8) During our research we also had the chance to get familiar with the steps for developing an OCR algorithm. Even though we had more important matters to focus on for this dissertation and decided to use ready OCR software, a little knowledge on the subject could prove helpful later on. Here are, in general, the steps for developing one type of an OCR algorithm [7]:

   - Load an image
   - Convert the image to grayscale (easy and simple).

[5] Mobile OCR Engine is a Powerful Recognition SDK for Mobile Devices https://www.abbyy.com/en-ca/mobile-ocr/trial/?purpose=trial&campaign=701w0000001Jln8

[6] Tesseract (software) - Wikipedia https://en.wikipedia.org/wiki/Tesseract_(software), Inventor: Hewlett Packard

[7] Character Recognition (OCR Algorithm) – Stack Overflow http://stackoverflow.com/questions/15188104/character-recognition-ocr-algorithm

- Convert the grayscale image to binary (possible using “Otsu’s method”, which calculates the optimum threshold separating the two pixel classes (foreground pixels and background pixels) so that their combined spread (intra-class variance) is minimal, or equivalently (because the sum of pair wise squared distances is constant), so that their inter-class variance is maximal).
- Detect the rotation angle of the image (possible using the “Houg Transform” feature extraction technique, which finds imperfect instances of objects within a certain class of shapes by a voting procedure).
- Remove noise (replace any pixel that does not have a neighbor (north, east, south or west) with the same color (a similar color, using a tolerance threshold) with the average of the neighbors).
- Separate lines and areas with characters from everything else (search for vertical white gaps for layout detection. Slice along the vertical gap. For each slice, now search horizontal gaps, and slice. If the slices have the same (a similar) height, you are at line level. Otherwise repeat vertical/horizontal slicing, until you only have lines left. The last step then is again a vertical slicing, giving you the single characters (or ligatures in some cases). Long and narrow or short and wide slices are lines).
- Compare the character slices with a character library. An image of every character must be converted to appropriate character code. If performance is not the main concern, try to find the characters within different font libraries, until you can identify the font used. Then stick with that font for character recognition.
- Save the results to the selected output format, for instance, searchable PDF, DOC, RTF, TXT. You may feel like it’s important to also save the original page layout: columns, fonts, colors, pictures, background etc. In that case, you could produce the background image by replacing each character in the original image with the

background color, which is determined by interpolating pixels that not are part of the character for each pixel of the character.

Let us be honest: **The real and main reason we went with Google's "Ocr-reader" was that it was the first one we tested (since testing it out was extremely simple, it was a simple Android project with functional code) and that it was Google's product, which is an indirect guarantee of quality. Plus, if we were disappointed by its performance we could always go back and choose another of the OCR engines described above**. We were not, and although "Ocr-reader" 's performance and abilities are described later on, it's worth mentioning that one thing at which it excels and surpasses the other OCR engines is that it does NOT require a captured image to produce results, in contrast to the other OCR engines. The need for a captured image would prove quite a hard task for blind/visually impaired people. Thus, we proceed to describe the infamous Google's "Ocr-reader"….

5) Google's "Ocr-reader"

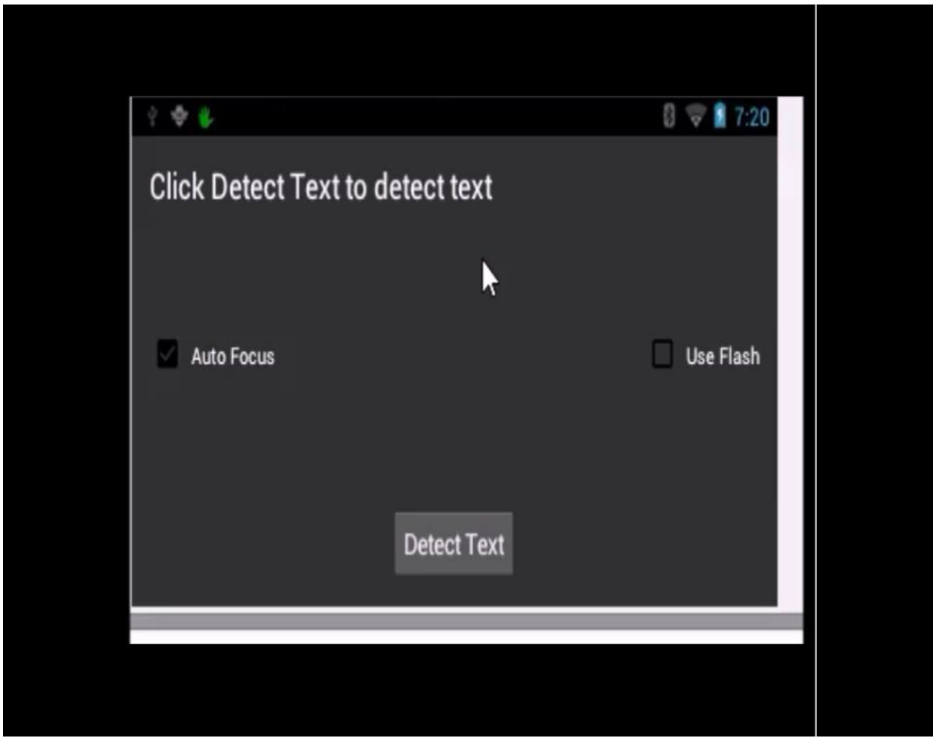
5.1) Operation and functionality of the Ocr-reader

After installing the "Ocr-reader" app on our glasses for the first time, we realized that in order to initialize the text detector, the "Google Play Services" app had to be installed on the glasses. Through closer inspection of the comments in the code of the "Ocr-reader", we realized that the glasses needed to support a "Google Play Services" version higher than 8.1. That next step proved more complicated than expected…

Although the Vuzix M100 is an Android device, it, unlike others, does not come with the "Google Play Store" and "Google Play Services" pre-installed on it. To add to the list of unfortunate events, the default browser pre-installed on the glasses was unable to access Google's "Play Store" webpage. So we attempted to install the "Google Play Store" app, in order to make the installation of any other app directly on the glasses possible. The app was downloaded through a website that provides .apk packages and was installed on the glasses through the "Vuzix M100 System File Manager" program on our PC. It was the first time such a procedure was needed, since every other app we installed on the glasses came in the form of compiled code through Android Studio, installed directly on the glasses through the USB connection. Unfortunately, the installed "Google Play Store" app failed to even launch, no matter what version of it we installed. Having no other obvious alternative, we abandoned our attempts of installing it and proceeded to install the "Google Play Services" app as an .apk package downloaded from a website. We knew that the app's version had to be higher than 8.1

for the “Ocr-reader” to work, but we didn’t know what version was compatible with our glasses. Eventually, after checking some compatibility rules between “Google Play Services” versions and Android versions, we installed the “Google Play Services” version 9.6.73. As a final step, we verified that the phone, and consequently the Ocr-reader, recognized that the “Google Play Services” app was indeed installed with the use of an app called “Play Services Info”. We were, at last, ready to use the “Ocr-reader”…

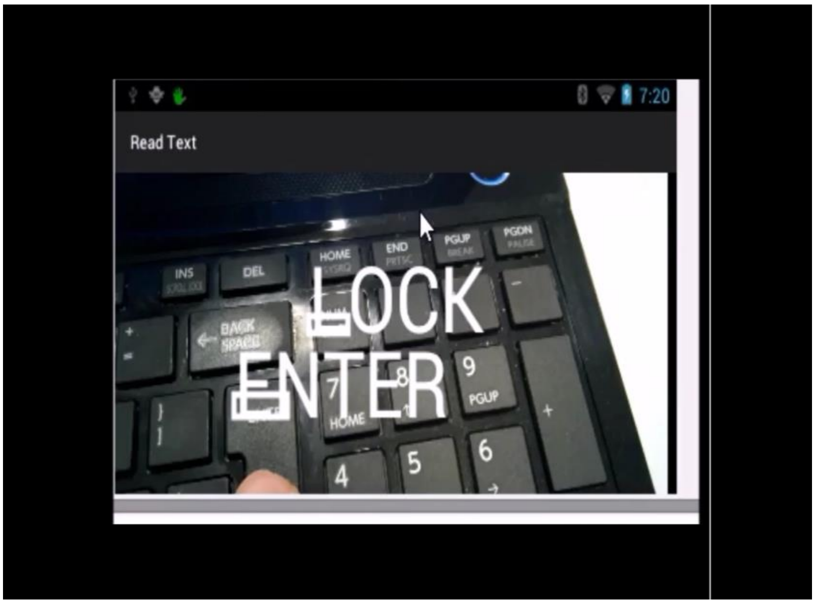
Let us know explain the general and abstract operation of the “Ocr-reader” application. Once the app launches, it gives you two options and one button. The button is used to begin reading text from whatever the camera is looking at. The first option, “auto-focus”, is for letting the camera focus on the picture when held still for a second or two, and should be enabled for best performance in what we’re trying to do. The second option, “use flash”, is for turning on the flash of the device (if available), which should also be used if available for better reading of the words (unfortunately our Vuzix M100 glasses don’t support this option, so a well lit area is a must for reading).


Click Detect Text to detect text

☑ Auto Focus

□ Use Flash

Once the button is pressed**, the camera is turned on and any text recognized in its field of view is presented on the screen with big white letters. The big text is enclosed in blocks which are directly above the text read, and are used to show where the text was recognized at. So basically, what the application tries to do is find areas where text is next to each other (like the text in books) and encloses that text in a single block, indicating that this particular area contains the enclosed letters/words**. If, for example, the camera recognizes 4 words, one at each of its field of view’s corners, 4 blocks will be displayed, one at each of the four corners, each block containing the word read at each corner. Here’s a visual example for better understanding:



At this point we couldn’t yet understand if the application was presenting blocks as soon as it created them or after all the possible blocks on a frame were identified. To find that out we needed to give the camera a field of view full of words and see how it would “react”. And a page of a book filled with words was the perfect way to do that. Our conclusion on this matter is discussed later on…

Now that we explained the abstract operation of the “Ocr-reader” application, it’s time to look at it in more detail, that is, see what the code does and how. Since the “Ocr-reader” is a complete Android project, we will only look at the files that are relevant to the functionality described before.

There are a total of 7 .java files (3 of which are responsible for the camera and its configuration) and 5 .xml files (4 of which are for the layout before the initialization of the camera, depending on the orientation of the screen etc, and 1 for the layout after the camera is initialized):

Java Files

Camera

CameraSource.java

CameraSourcePreview.java

GraphicOverlay.java

Main

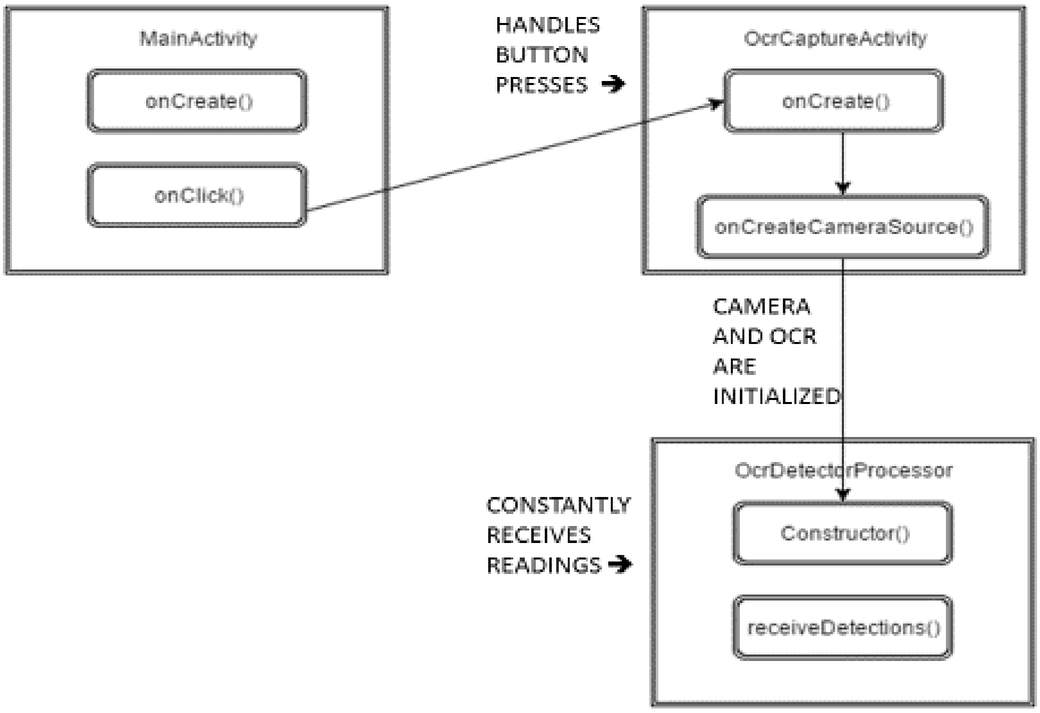
MainActivity.java

OcrCaptureActivity.java

OcrDetectorProcessor.java

OcrGraphic.java

We are going to analyze the important parts of the flow of our program, which methods /classes are called and in what sequence. Our program starts with the "MainActivity" class:



MainActivity

-onCreate(): The very first method that is called in the program. This method just assigns the correct variables to the correct buttons of this starting activity and sets a button listener on the “read_text” button.

-onClick(): Launches the camera and text detector (“OcrCaptureActivity” activity) once the “read_text” button is pressed.

OcrCaptureActivity

-onCreate(): The very first method called after the “read_text” button is pressed on the previous activity. This method calls the “createCameraSource()” method once some initial preparations are done.

-onCreateCameraSource(): This is the method whose code can be modified to correctly set the camera settings based on the camera specifications of the hardware on which the software will be installed. The text detector is created here, and the one and only instance of the “OcrDetectorProcessor” class is called and initialized.

OcrDetectorProcessor

Unlike the previous two, this class is not an activity. Once the constructor is called, any reading of a frame completed by the text detector is sent to the “receiveDetections()” method:

-receiveDetections(): This method takes any frame read, separates it into the blocks of text that were recognized by the text detector and sends them to be presented on the screen. Actually, “receiveDetections()” is more like a thread than a normal method, since it works continuously in the background whether the text detector recognizes any text or not.

So far, it seemed that the “serious work” on the code would take place in the “OcrDetectorProcessor” class, and more specifically in the “receiveDetections()” method, since this is where the blocks are initially accessible for modifications etc…

Also, closer inspection of the above structure helped us understand how the “Ocr-reader” really worked: we previously said that the application focuses on recognizing blocks of text, but we didn’t know when those blocks of text were set to be presented on the screen. Well, now we knew... **The application constantly takes photographs of the camera’s field of view (frames) and analyses them into blocks of text. Once the analysis of a photograph is complete, the resulting blocks are presented on the screen simultaneously. The rate at which the application takes these photographs depends on how much text each photograph contains: photographs filled with words means less photographs per second, and vice versa.** From now on, we will be calling these analyzed “photographs” **READINGS**, try to remember that as it’s highly important...

5.2) Notifying Google about an error in the application’s code

While inspecting the code of the “Ocr-reader”, we noticed that the application ID defined in a build.gradle file was wrong (instead of “com.google.android.gms.samples.vision.ocrreader”, the ID was “com.google.android.gms.samples.vision.barcodereader”, which is the same ID the “barcode-reader” project has). This could cause some serious issues if both applications were to be installed on the same device. And so we decided to notify Google of this issue through a discussion specifically for issues with “Mobile Vision” on GitHub. [1]

[1] Issues – googlesamples/android-vision - GitHub

https://github.com/googlesamples/android-vision/issues, Issue #175

5.3) Testing the capabilities and limits of the Ocr-reader

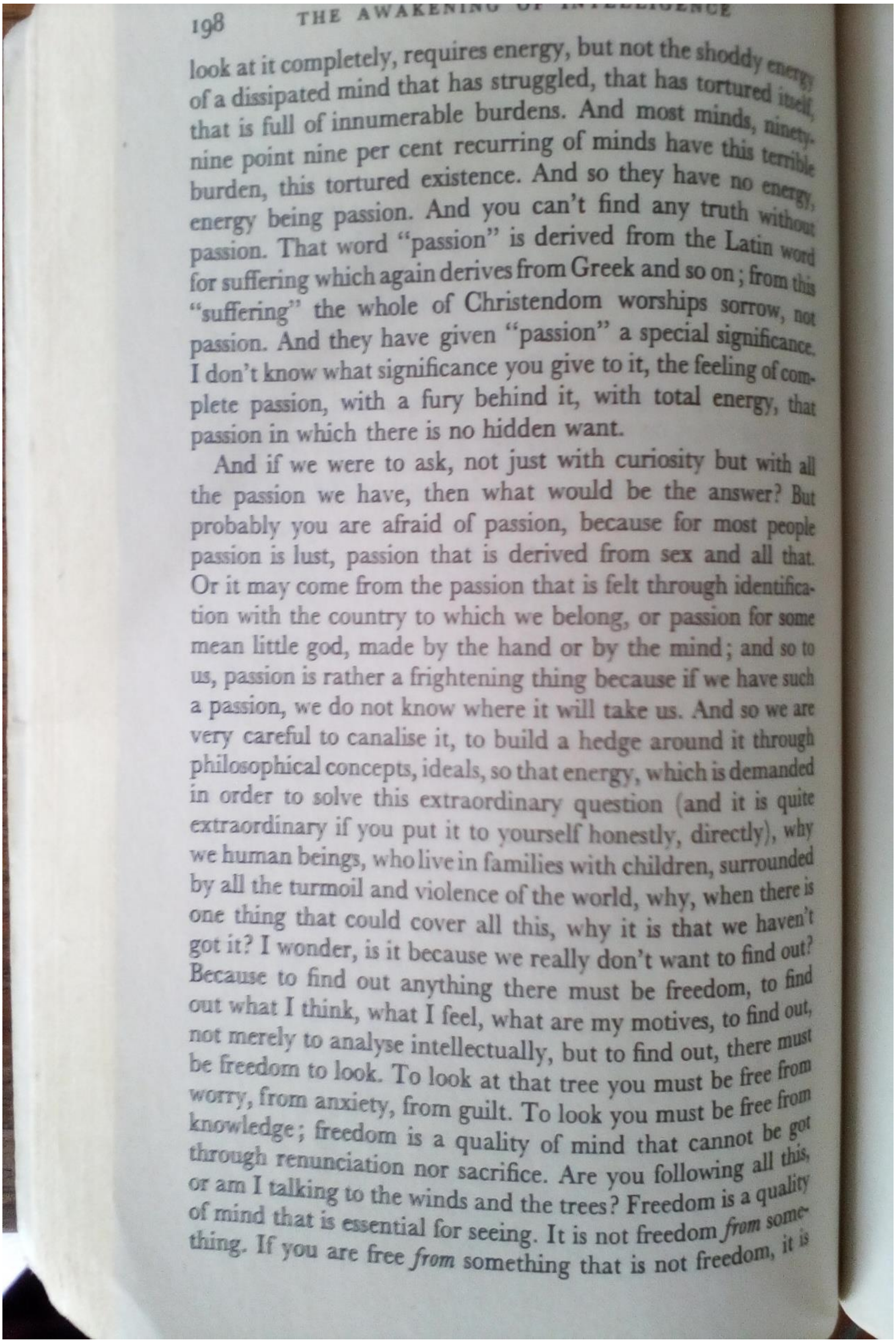
To carry out the tests, we also used (apart from the glasses) a BlackView 2000s Android phone. After some initial necessary test-installations, we installed the “first complete” version of the “Ocr-reader” application on our phone and Vuzix M100 glasses. We had made two changes to the original downloaded software:

1. We logged the read words in a simple way to help us understand what was read by the software and in what order. Whenever a block of text appeared on the screen of the app, that same block of text was immediately written in the log. [Code: Appendix section B]

2. We specified, through the software’s code, that we would be using it for a preview size of 1920 x 1080, the optimal preview size when using 1080p video/5MP like our glasses’ camera does. The default fps was 15, but we changed it to 30 since a few tests we ran proved it provided slightly better results. Note that these changes could negatively affect the performance of the software on the phone, which was using 1280 x 720 video/8MP, but that wasn’t of importance since we only cared about maximum performance on the glasses. [Code: Appendix section C]

After successfully installing the “Ocr-reader” application on both our mobile phone and Vuzix M100 glasses, the next step was testing its functionality and limits. The tests were carried out firstly on the phone and then on the AR glasses, and the main reason for that was the fact that we were a lot more confident in the correct functionality of the “Ocr-reader” application on the phone than on the glasses, after all, it was primarily designed for a phone (one indication of this was the capability to turn on the flash of the phone through the app, which caused the glasses to crash if selected). It turned out that testing the software on the phone first really helped analyze its functionality on the glasses, and we will be explaining later on why.

So let us present the results of this functionality analysis. After that we will lay out our conclusions. Originally, we had tried to read two pages simultaneously (one next to the other, like when reading an open book), but that made the performance drop significantly since the camera had to be quite far to fit both pages and the software was prioritizing horizontal reading, while we wanted it to prioritize vertical reading (first read the whole first page and then the second). As for the latter, it could be fixed, but the amount of effort required to do so would be much greater than ignoring this problem and focusing on reading on page at a time. The following image is the page we attempted to read:



look at it completely, requires energy, but not the shoddy energy of a dissipated mind that has struggled, that has tortured itself, that is full of innumerable burdens. And most minds, ninety-nine point nine per cent recurring of minds have this terrible burden, this tortured existence. And so they have no energy, energy being passion. And you can't find any truth without passion. That word "passion" is derived from the Latin word for suffering which again derives from Greek and so on; from this "suffering" the whole of Christendom worships sorrow, not passion. And they have given "passion" a special significance. I don't know what significance you give to it, the feeling of complete passion, with a fury behind it, with total energy, that passion in which there is no hidden want.

And if we were to ask, not just with curiosity but with all the passion we have, then what would be the answer? But probably you are afraid of passion, because for most people passion is lust, passion that is derived from sex and all that. Or it may come from the passion that is felt through identification with the country to which we belong, or passion for some mean little god, made by the hand or by the mind; and so to us, passion is rather a frightening thing because if we have such a passion, we do not know where it will take us. And so we are very careful to canalise it, to build a hedge around it through philosophical concepts, ideals, so that energy, which is demanded in order to solve this extraordinary question (and it is quite extraordinary if you put it to yourself honestly, directly), why we human beings, who live in families with children, surrounded by all the turmoil and violence of the world, why, when there is one thing that could cover all this, why it is that we haven't got it? I wonder, is it because we really don't want to find out? Because to find out anything there must be freedom, to find out what I think, what I feel, what are my motives, to find out, not merely to analyse intellectually, but to find out, there must be freedom to look. To look at that tree you must be free from worry, from anxiety, from guilt. To look you must be free from knowledge; freedom is a quality of mind that cannot be got through renunciation nor sacrifice. Are you following all this, or am I talking to the winds and the trees? Freedom is a quality of mind that is essential for seeing. It is not freedom *from* something. If you are free *from* something that is not freedom, it is

To make reading as easy as possible for the glasses, we put a blank piece of paper on top of the right page (since we are reading the left) and set up a lamp next to the book which was facing towards the page (so that we could put our head above the book, with the glasses facing downwards, and still not block the light shining on the page with our head).

Over the course of 2-3 days, we made a lot of tests and examined a lot of logs. Below we will be presenting the most representing result (log) for the phone and the most representing result for the glasses (note that the most representing result IS NOT the best result, but rather the log that represents the conclusions we arrived at best). The "I:" at the beginning of a set of lines indicates that the following lines were enclosed in a block by the application:

Phone

```
I: THE AWAKENING OF INTELLIGENCE
I: inds and the tr
I: of look a dissipated it completely, that requires struggled, energy, but that not has the tortaned shoddr e
   th nine ie point d ef nine per cent recurring of Aad minds zoat have this t e
I: burden, cnergy this being tortured passion. existence. And you can't And so find they any have truth no mn wi
I: for passic auffering which again derives from Greek and so on, bae
I: a. That word "passion" " is derived from the Laie w
   the whole of Christendom e worships samtwe
I: "'suffering
I: e f we hare sc
I: Freedo
I: why it is that want te
I: e but with al
I: And if we were to ask, not just with cu
   the passion we have, then what would be the an
   probably you are afraid of passion, b
   passion is lust, passion that is derived from ses and al tis
   Or it may come from the passion that is felt th
I: e for mest pemie
I: got one 2 thing that could t cover all we this,
I: I don't know what significance you give to it, the feeliag efic
I: with the country to which we belong, or pasios er
   an little god, made by the hand or by the mind; and ut
   us, passion is rather a frightening thing
   a passion, we do not know where it will take us. And so we at
   very careful to canalise it, to build a hedge ar
   philosophical concepts, ideals, so that energy,
   in order to solve this extraordinary question and
   extraordinary if you put it to yourself honestly, directy.
   we human beings, wholivein families with children, suzea
   by all the turmoil and violence of the world, why, ben ther
I: Becau to find out anything there must be freedom,
   out what I think, what I feel, what are my m motives, to
```

I: not merely to analyse intellectually, but to find oet.
be freedom to look. To look at that tree you must be
Worr from anxiety, from guilt. To look you must be
knpwledge ; frecdom is a quality of mind that ca
throu renunciation nor sacriñice. Are you follo
or am I talking to the w
of mind that is essential for seeing. l
I: pass
I: plete passion, with a fury behind it, with total enerp, de
I: n. And they have given "passion a special
I: passion in which there is no hidden w
I: THE AWAKENING OF INTKELLIENCE
I: look a at it completely, f per that cent requires has recurring encrgy. of but minds not mue kaves the shodd- milo te ted
I: nine poin
I: lowe
I: burde . this tortured existen re. And so they have tp
I: e being passio a. And you can't find any trutdl witio
passion. That word "passion" i is derived from the Latis s
I: for suffering which again derives from Greek and so on; from t
suffering" the whole of Christendom worships somroe w
I: pas
I: sion in which there is no hidden want.
I: e. And they have given "passion" a special siguificasoe
I don't know what significance you give to it, the feling of cams
plete passiom with a fury behind it, with total energ, so
I: but wit al
I: Ba
e for most peagie
I: And if we were to ask, not just with cu
the passion we have, then what would be the a
probably you are afraid of passion, b
passion is lust, passion that is derived from ses and all tia
Or it may come from the passion that is felt through ide
I: tion with the country to which we belong, or passion for sume
I: mean little god, made by the hand or by the mind; and s t
us, passion is rather a frightening thing b
I: e if we have aud
I: fal to canalise it, to build a a hedge around it tirou
I: we human beings, wholivein families with children, surrc
I: merely to analyse intellectually, but to fid out, the
be Bredom to look. Ta lok at that tree you mut be be
I: passion, we do not know where it will take us. And so we amt
I: philosophical concepts, ideals, so that energy, which ia d
a order to solve this extraordinary qustion (and it k
eraordinary ifyou put it to yourself honestly, d
I: by all the turmoil and violence of the world, why, w
me thing that could cover all this, why it is that we E
got it? I wonder, is it because we really don't want d fnd
Because to find out anything there must be freedom,
out what I think, what I feel, what are my motives, to e
I: sciation nor sacrifice. Are you follows
I: ey from ansiety, Bom guslt. To lock yes must le e
knowledge; vedom is a quality of mind that cade
I: through r
or am I talkin to the winds and the trees? Freedom
of mind that is essential for seeing. It is not freedon
thing. at you are free from something that is not fee

Glasses

I: with a fary behin it, with total energy,
I: THE AWAKENING OF INTELLIGENCE
I: but not the shoddy enere
I: nine poir
I: per cent recurring of minds have this ter
I: passion "a special Hghibicance,
I: by all the turmoil and violence of the world, why, w
I: ne thing that could cover all this, why it is that we ha
I: of that a is dissipated full of innumerable mind that has burdens. struggled, And that tnost has minds, tortured Ruinety. nelf,
I: sion. And they have give
I: I don't know what significance you give to it, the feeling of com.
plete passior with a fury behind i L with total enerev that
I: passion in which there is no hidden want.
And if we were to a
I: not just with curiosity but with all
the passio we have, then what would be the answer? But
I: probably you are afraid of passion, because for most people
passion is lust, passion that is derived from sex and all that.
it may come from the passion that is felt through identfiea-
tion with the country t to which we belong, or passion for some
I: mean little god, made by the hand or by the mind; and so to
HS, passion is rather a fnghtening thing because if we have nuch
a passio we do not know where it will take us. And so we are
very careful to canalise it, to build a a hedge around it through
philosophical concepts, ideals, so that energy, which is demanded
in order to solve this extraordinary question (and it is quite
ordinary if you put it to yourself honestly, directly), why
we human beings, wholivein familie with children, surrounded
I: got it? I wonder, is it because we really don't want to find out?
Because to find out anything there must be freedom, to find
out what I think, what I feel, what are my motives, to find out,
not merely to analyse intellectually, but to find out, there must
be freedom to look. To look at that tree you must be free froms
Worry, from anxiety from guilt. To look you must be free from
knowledge; freedom is a quality of mind that cannot be got
through renunciation nor sacrifce. Are you following all this
or am I talking to the w unds and the trees? Freedom is a qualir
of mind that is essential for seeing. It is not freedom from Some-
thing. If you are free from somethi that is not freedom, it
I: rg8
look at it completely, requir
I: burden, this tortured existence. And so they have no
energy being passion, And you can't find any truth with
passion. That word "pasuion" is derived
for suffering which againderivesfrom Greck and so on; from word tla
"'suffering'" the whole of Christendom worships sorrow, sat
I: THE AWAKENING OF INTELLIGENCE
I: rg8
I: that of Jook a dissipated is at it full completely, of innumerable mind that requires has burdens. struggled, energy, but And that not most has the minds, shoddy tortured em un

burden, mergy being this tortured passion, per cent existence. A recurring can't And of so minds find they any hav have truth this no erible

for suffering" passion. suffering That the which word whole again "passion" of derives hristendom is from derived Greek worships from the on Latin fomn word dest

I: poir

I: sthou

I: hot

I: passio And they have given "p passion a special significanes

I don't know what significan you give to it, the feeling of com. plete passion, with a fury behind it, with total en passion in which there is no hidden want.

And if we were to ask, not just with curiosity but with all the passio we have, then what would be the answer? Bou probably you are afraid of passion, b because for m passic is lust, passion that is derived from sex and all that O it may come from the passion that is felt through identifica. tion with the country to which we belong, or passion for some

I: erEY, that

I: people

I: us, a passion, passion we is rathera do not know frghtening where it thing will because take us. if we And have so we suck are

I: an litle god, made by the hand or by the mind; ,and so to

I: very carefu to canalise it, to build a hedge around it through philosophical concepts, ideals, so that energ which is demanded in order to solve this extraordinary question (and it is quite extraor rdinary if you put it to yourself honestly, directly), why we human beings, wholivein milies with children, surrounded by all the turmoil and violence of the world, why, when there a one thing that could cover all this, why it is that we haven't got it? I wonder, is it because we really don't want to find out? Because to find out anything there must be freedom, to find out what I think, what I I feel, what are my motives, to find out, not merely to analyse intellectually, but to find out, there must be freedom to look. To look at that tree you must be free from ry, from anxiety, from guilt. To look you must be free from knowledge; freedom is a quality of mind that ca anot be got arough renunciation nor sacrifice. Are you following all this, or am I talking to the winds and the trees? Freedom is a quality of mind that is essential for seeing. It is not freedom n. from something. If you are free Irom someth that is not freedom, it a

Here are the conclusions derived for both devices (phone and glasses) on the following "areas of importance" for the analysis:

- **<u>Distance of camera from page (glasses only)</u>**

  The application itself is able to recognize small words from one edge of the room to the other. So the real challenge here was to determine how close did the camera need to be to the book so that the trembling didn't mess up the reading of all those words next to each other. Also, the bigger the page the greater the chance that some parts of it would not be recognized correctly at all, even though they are taken in account when analyzing the reading. It just seemed too much information for one reading. So we decided that a good distance from the book was at least that which made the whole page visible in the camera's field of view, approximately 70cm. That was like you standing, bending your back and looking downwards at a page located between your waist and knees. The distance is such that you could barely read the words through the camera. So if a visually impaired person sat down and had a book placed 70cm below his head, he could get the results we are describing above. Of course, that doesn't sound comfortable enough.
- **<u>Time to process words before presenting them on the screen</u>**

  This is where using the phone first proved helpful. When we used the phone, it only took 2-3 seconds to show the blocks on the screen after the camera was above the page, and at first we thought it was the time needed for the camera to focus and stabilize. But when we used the glasses, the blocks took about 10 seconds to appear. This is justified by the fact that the phone has a much greater processing speed compared to the glasses (4 cores vs 2 cores). Also, immediately after the blocks appeared, the log was filled with the whole page. Another interesting observation was the fact that, by the time the blocks appeared on the screen, the application had already analyzed 2 readings, as indicated by the log filling up with 2 occurrences of the same page after a few seconds. This was confirmed through multiple tests, including leaving the camera above the page for 2-3 seconds and then moving it away. The read blocks appeared on the screen after a while, no matter where the camera was pointing. So we had confirmed that at least two readings of a page were completed by the time the blocks were ready to be shown and written in the log.

- **Accuracy of readings (words/sentences read correctly)**

  First of all, the glasses are clearly more accurate than the phone. Which was expected (and requested to be exact) since we set the preferences of the software to match the specifications of the glasses, and their video analysis is better than that of the phone (1920 x 1080 vs 1280 x 720). Still, both the phone and the glasses did poorly, being quite accurate only on a few sets of sentences of the page (usually, the first few lines of the page were barely readable). Furthermore, both the phone and the glasses were more accurate on the second reading of the page, something that can possibly be justified by the fact that the camera was a lot more stabilized by the time the second reading started.

- **Percentage of readings (how many words read out of the total number of words?)**

  Just like with accuracy, both the phone and the glasses recognized all the words in a few sets of sentences, while they did very poorly on other sentences, especially on their edges. Also, just like with accuracy, the second reading of the page recognized a much larger percentage of the total words.

- **Block sizes**

  A lot of testing showed that, from this distance, blocks can be as big as one third of a page, assuming we have a good reading. If not, blocks are a few lines long. That's a good result, the optimal would be one-page-blocks, but this too is satisfying.

- **Ordering of words and blocks**

  This is where it gets interesting. As we mentioned before, by the time the blocks appeared on the screen, the log would eventually fill with 2 occurrences of the same page. In the log, all the blocks of the first occurrence appeared before of the blocks of the 2nd occurrence, which is good, but often the blocks of the same occurrence were a little unsorted (a block of previous lines could appear after a block of following lines). So we had to find a way to overcome this problem, too. As for the ordering of the words within a block, that was fine, we had 100% accuracy.

Surely the trembling of both the hand when using the phone and the head when wearing the glasses greatly increased the difficulty the application had in recognizing the text. Especially since the distance needed to fit a page in the camera's field of view was the one analyzed above. Still, these results were not acceptable for what we were trying to accomplish. We could try to modify the code to improve the ordering of the blocks, choosing between two readings of the same block etc. However, no matter how we modified the code, we could do nothing to increase the accuracy at this distance and with this trembling. Since the "trembling" factor is a constant, the only visible remaining option was to reduce the distance of the camera from the page. As a result, we had to read something less than a page at a time. So we tried reading half a page, by covering the other half with a blank piece of paper. The results were amazing (again, these are the most representing results for the 2 halves of the page when read by the glasses):

<u>Glasses – First half of page (up to "…hidden want")</u>

```
I: 19B THE A AWw AKENING OF INTELLIGENCE
I: look at it completely, requires energy, but not the shoddy energy
  of a dissipated mind that has struggled, that has tortured itself,
  that is full of innumerable burdens. And most minds, ninety-
  nine point nine per cent recurring of minds have this terrible
  burden, this tortured existence. And so they have no energy,
I: sergy being passion. A And you can't find any truth without
  passion. That word "passion" is derived from the Latin word
  for suffering which again derives from Greek and so on ; from this
  "'suffering" the whole of Christendom worships sorrow, not
  passion. And they have given passion "" a special significance,
  I don't know what significance you give to it, the feeling of come
  plete passion, with a fury behind it, with total energy,
  passion in which there is no hidden want.
```

<u>Glasses – Second half of page</u>

```
I: passion is lust, passion that is derived from sex and all that.
I: And if we were to ask, not just with curiosity but wity all
   the passion we have, then what would be the answer? But
   probably you are alraid of passion, because for most people
I: Or it may come from the passion that is felt through identifica-
   tion with the country to which we belong, or passio for somie
   mean litte god, made by the hand or by the mind; and so to
   os, passion is rather a frightening thing because if we have such
   a passion, we do not know where it will take us. And so we are
   ry careful to canalise it, to build a hedge around it through
   philosophical concepts, ideals, so that energy, whichis demanded
   in order to solve this extraordinary question (and it is quite
   extraordinary if you put it to yourself honestly, directly), wliy
   we human beings, wholivein families with children, surrounded
   by all the turmoil and violence of the world, why, when there is
   one thing that could cover all this, why it is that we haven't
   got it? I wonder, is it because we really don't want to find out?
   Because to find out nything there must be freedom, to find
   out what I think, what Ifeel, what are my motives, to find out,
   not merely to analyse intellectually, but to find out, there must
   be freedom to look. To look at that tree you must be free from
   worry, from anxiety, from guilt. To look you must be free from
   knowledge; freedom is a quality of mind that cannot be got
   through renunciation nor sacrifice. Are you following all this,
   or am I talking to the winds and the trees? Freedom is a quality
   of mind that is essential for seeing. It is not freedom frem some-
   thing. If you are free from something that is not freedom, it is
```

- **<u>Distance of camera from page</u>**

  This time, only half the page needed to be in the field of view of the camera, and that helped reduce the required distance by a lot. By placing the camera only 40cm above the particular half we were reading, we could get the results shown above. Plus, 40cm distance from the page is a lot more comfortable for someone sitting and looking down vertically at a book placed on his knees or at a desk of that height (compared to the 70cm required before, that almost required the person to be standing to have such a distance from the book).

- **<u>Time to process words before presenting them on the screen</u>**

  This time, the blocks appeared on the screen almost instantaneously when using the phone, while it took about 7 seconds for the glasses to show the blocks. Also, both the phone and the glasses wrote 2 occurrences of the same half of the page in the log by the time the blocks appeared, just like in the tests with a whole page read.

- **<u>Accuracy of readings (words/sentences read correctly)</u>**

  In the second occurrence of the half written in the log, very few words were read wrong (in the first example above, the only words read wrong were "awakening" and "energy"), so we calculated an accuracy rate of over 98%.

- **<u>Percentage of readings (how many words read out of the total number of words?)</u>**

  In the second occurrence of the half written in the log, the percentage of readings always seemed above 98%, very few words were missed, if any.

- **<u>Block sizes</u>**

  After a lot of tests, we confirmed that the whole half that we were reading was often put in a single block, which was very fortunate. Other times, some blocks contained only a line, but the order in which they were written in the log was often correct.

- **<u>Ordering of words and blocks</u>**

  We had 100% correct ordering of the words in the blocks. As for the ordering of the blocks, sometimes some blocks-line were written in the wrong place in the log, but because of the little amount of total blocks, that seemed like it could be fixed whenever it occurred.

So it seemed we had very accurate and satisfying results when examining the second occurrence of the half page in the log. As a result, we had to make sure that the first occurrence would be ignored, or even better, used to our advantage in case the second, more accurate occurrence, missed something the first one got right. Also, we had to modify the code of the application (if possible) to correctly order the blocks that were misplaced in the log (if any). Finally, it would be great if we could modify the code to increase the accuracy even more, even though it was quite satisfying as it was.

6) The “Book Eyessistant” (Creating our application)

Yes, that was the name we decided to give to our application!! Not THE most catchy and friendly name but it sufficed. As it should probably be obvious, the name combines the words “book assistant” and “eye” to imply that basically you can use the application as a special eye to assist you with (reading) books.

6.1) Ocr-reader - Optimization 1: Ordering of blocks

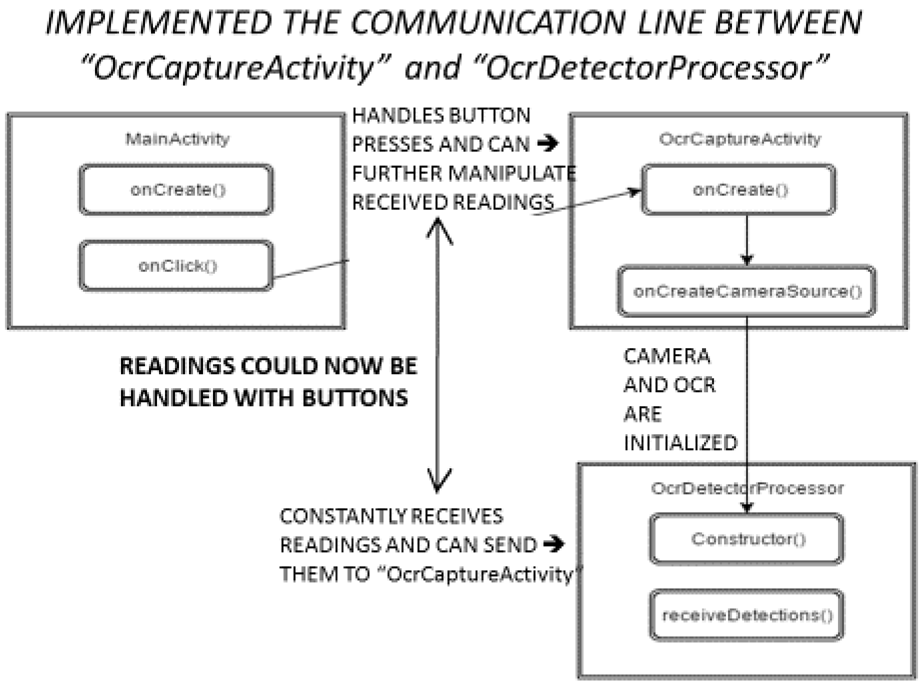
Our first attempt was to correctly order the blocks that were misplaced in the log. After carefully inspecting the code and doing some tests, we realized that the location of each block could be retrieved, and by “location” we refer to its 4 points-corners in relation to the camera’s field of view. Furthermore, the scaling of the horizontal axis

(x) starts at 0 (leftmost area of the camera's field of view) and ends at 1920 (rightmost area of the camera's field of view). The scaling of the vertical axis (y) starts at 0 (upper area of the camera's field of view) and ends at 1080 (bottom area of the cameras field of view). Thus we realized that the application defined the position of a block relatively to the resolution of our camera / the preview size we defined in the code earlier on.

What we wished for is for blocks to appear in order, from highest to lowest, and for blocks of the same height to appear in order, from leftmost to rightmost. **And we did that by comparing the 'x' and 'y' position of each block**. [Code: Appendix section D]

6.2) Ocr-reader - Optimization 2: Choosing the reading with the most recognized characters (+ initializing our text-to-speech engine)

As we mentioned before, by placing the camera above the page and waiting until the blocks appeared on the screen, we could get at least 2 readings of that page. However, the camera keeps on storing readings of its field of view even when it is not above a page, empty readings, but readings nonetheless. So what we wanted now **was to choose the reading that best represented the contents of the page**. At first, we attempted something simple: **we decided to declare the best reading based on the number of characters it contained (following the logic that more characters equals more accurate reading)**. Although that rule isn't always correct, in most cases it is. So this is how we modified the code to achieve it (we implicitly implemented a communication line between the "OcrDetectorProcessor" class which is responsible for receiving the detected blocks of text and the "OcrCaptureActivity" class which is responsible for handling the button actions):


IMPLEMENTED THE COMMUNICATION LINE BETWEEN "OcrCaptureActivity" and "OcrDetectorProcessor"
MainActivity
onCreate()
onClick()
HANDLES BUTTON PRESSES AND CAN ➔ FURTHER MANIPULATE RECEIVED READINGS
OcrCaptureActivity
onCreate()
onCreateCameraSource()
READINGS COULD NOW BE HANDLED WITH BUTTONS
CAMERA AND OCR ARE INITIALIZED
CONSTANTLY RECEIVES READINGS AND CAN SEND ➔ THEM TO "OcrCaptureActivity"
OcrDetectorProcessor
Constructor()
receiveDetections()

The following modified code also includes the initialization of our text-to-speech engine that would get the best reading decided and speak it to the user through the ear speaker. **To listen to the best reading decided, the user has to wait at least until the blocks appear on the glasses' screen, and then press the MENU android button on the phone or long press the "FRW" button on the glasses. After the button is pressed, the application starts speaking and the list containing the readings is emptied**. [Code: Appendix section F]

6.3) Ocr-reader – Revisiting compiler issues

Since we were programming an application for the Vuzix M100, and not just an Android phone, we strongly felt the need to take advantage of their special capabilities (apart from the fact that they're wearable), which were the Voice Recognition and Gesture Detection capabilities. Our next idea involved the use of the "VoiceControl" library (the idea is described in the next section). However, if we revisit the "3.4) Compiler issues" section, we shall remember that using that library is not possible without using the Vuzix compiler.

This introduced major issues, since the Vuzix compiler is too old to be able to compile the code of the “Ocr-reader”. Even if we used the technique of switching to a correctly set up Android project that we described in the “3.4) Compiler issues” section, we still had no way to change the code of the application in order to be recognized by the Vuzix compiler, it was just too much code and too much irreplaceable code, too. This consequently implied that we would be unable to use the Voice Recognition and Gesture Detection capabilities of the Vuzix M100 for our application, something extremely unfortunate.

Luckily for us, we found a way out of this seemingly dead-end. After a lot of tests and research, we discovered that, **although the compiler is unable to recognize the “VoiceControl” library when imported in the code, if the library is given directly as a .jar file dependency to the project, the normal and updated compiler IS able to recognize parts of it.** Now, that may sound strange, but it’s true, there are functions in the library that are recognizable by both compilers, and other functions that are only recognizable by the Vuzix compiler. For example, the “destroy()” function of the “VoiceControl” library is unrecognizable by the normal compiler, but we managed to find ways around it (meaning we found ways to destroy any “VoiceControl” variable initialized without calling the “destroy()” function). We were not so lucky with the “GestureSensor” library, where the normal compiler was unable to recognize the “register()” function which was crucial for using the Gesture Detector. (In the process of testing out the “GestureSensor” library for the “Ocr-reader”, we deleted a lot of code that was responsible for Gesture Detection events, events that did NOT use the “GestureSensor” library but the default Android Gesture Detection library).

So, as a verdict, we realized that we could, after all, use the Voice Recognition capability of the Vuzix M100 for our application, but not the Gesture Detection one.

6.4) Ocr-reader – Extra functionality 1: Choosing which already listened-to page we wish to hear using voice commands

Here is the idea: **The user can use voice commands to select which page (that he listened to before) he wants to listen to again**. The fact that the user uses voice commands would really benefit blind people because selecting a page in any way that involves vision is impossible (plus, a total of 4 buttons available on the glasses are too few to do that and we didn't want to use the buttons of an assisting phone since we considered that the need for an assisting phone would be a huge minus for the final product).

Based on the above considerations, and because we wanted to keep things simple, we decided that **the user would speak a number from 1 to 9, and then, by the push of a button, he would be able to hear the corresponding page**. For example, if he said '3', by pushing the button he would be able to hear the 3rd page he listened to in the past. If more than 9 pages have been listened to, then each new page added replaces the oldest one, so, after listening to the 10th page for example, saying '1' would result in listening to the 2nd page again.

Another issue we faced was which button to assign to this functionality. The MENU button was taken by "Optimization 2", and the only other buttons whose functionality could be defined was the HOME and BACK button (a lot of tests made have proven that the functionality of any button on the glasses that does not represent any of the MENU, HOME and BACK buttons can NOT be changed). Revisit the "2.5) Controls" section to see which button represents what. The problem with modifying the HOME or BACK button is that one or the other has to be responsible for exiting the application correctly, without just shutting everything down and ignoring warnings (this is bad programming). After some thinking, **what we did was to assign this new functionality we were describing to the BACK button (Vuzix M100 "ACTION" button long press)** through the "onBackPressed()" function (since no "onHomePressed()" function is available that does not involve bad programming) . We also knew that the HOME button goes through the "onPause()" -> "onStop()" functions to suspend the application, so we linked the "onStop()" function to the "onDestroy()" function to properly suspend and then terminate our application. With few words, **we assigned the default functionality of the BACK button to the HOME button**, and this was the best solution possible.

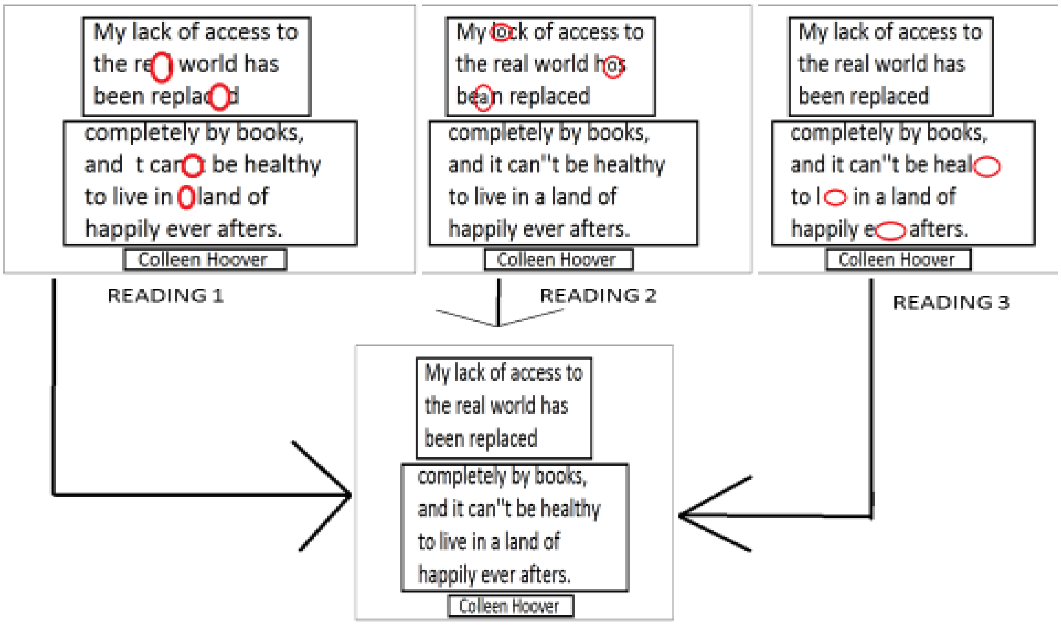
At this point, one may wonder: why use a button to listen to the page after a number is spoken? Why not have the application speak the page immediately after the user speaks the number, thus saving us an unassigned button we so hardly managed to find? Well, the problem here was that, while the application was speaking a page to us, if the “Speech Recognizer” thought it heard a number, it would immediately begin speaking the corresponding page. Generally, letting the application start to do anything just by listening to a sound is dangerous and would put restrictions on the user, such as being careful when to speak. By having the application speak only after a button is pressed, the user does not care what the “Speech Recognizer” recognizes, as long as the button is not pressed.

As for the writing of code to achieve this functionality, we spent almost a day trying to figure out why the “Speech Recognizer” wasn’t working. In the end, it was just a permission right we forgot to define in the AndroidManifest.xml file. Additionally, for optimization purposes, we only gave the “Speech Recognizer” the ability to recognize numbers. [Code: Appendix section L]

6.5) Ocr-reader - Optimization 3: Optimizing the accuracy

Perhaps the most important optimization of the code we did. This step was also the most difficult to perform because we had to take quite a lot of details in account. Recall that in section “5.5) Ocr-reader - Optimization 2: Choosing the reading with the most recognised characters (+ initializing our text-to-speech engine)” we wanted to find rules and heuristics to **choose the reading that best represented the contents of the page.** By the time we completed all those previous optimizations and functionalities, the goal had been updated to “**use a combination of readings to reconstruct the original contents of the page”.**

The easy, first step was to choose the non-empty, useful readings. For each page we wanted read, all we had to do was choose the 3-4 non-empty readings with the most characters to further work on them for reconstructing the original page. Any more than 3-4 readings would be unneeded, plus, they would require the person to stay above the page for a longer time than those few seconds needed for at least 2 readings. Now, how were we going to do that? How were we going to use 4 readings, each possibly missing some information, to reconstruct the original page?


READING 1
READING 2
READING 3
My lack of access to the real world has been replaced completely by books, and it can"t be healthy to live in a land of happily ever afters.
Colleen Hoover

The following ideas had come to mind:

- Use a dictionary to check the validity of the words of a reading and replace the wrongly read words with the correct ones. The problem with this is that there is a high possibility that some words would be special (like names, written sound effects etc.) and we didn’t want them declared as a wrongly read word by the dictionary, or even worse, to be replaced. Additionally, some wrongly read words like “zog” could be replaced by the words “dog”, “fog” etc. and a semantic check to decide the correct replacement word is nearly impossible.

- Start from the 1st character to the last and, by checking the i-th character of each reading in parallel, choose the i-th character of the reconstruction to be the most popular i-th character amongst the readings. The problem with this method was that, in case a big section of characters was missing, it could seriously mess up the whole process of parallel i-th character checks.
- Instead of making parallel character checks, make parallel word checks and choose the i-th word of the reconstruction to be the most popular i-th word amongst the readings. That way, part of the problem encountered at the character-check method could be eliminated, since we can skip ahead to the next word of each reading whenever we realize that sections of characters are missing from a reading. We will resume the process of parallel word checks once all readings have found their next common word.
- Take a lot of readings, say 20, and create our own dictionary by adding to our dictionary only those words that appear in more than 75% of the readings. Then use that dictionary in the first method/idea described. Though this solution eliminates the problem of the “special words” of the first method, it creates the need for a lot of readings.

The last idea described seemed like the best one, so we tried it out:

I. We first took all the 4 readings and added every newly encountered word to our custom dictionary. Every time we encountered a word seen before, we incremented its “seen” count.

II. **We then created a function that returned the similarity (Levenshtein Distance) of two Strings as a zero-to-one value** (1 means the Strings are identical, 0 means they are completely different Strings, not a single character in common).

III. We took the reading with the most characters and checked every single word of it: if the word was similar to a word from our dictionary (similarity greater than 0.7) and the dictionary word “seen” count was greater than 2, we replaced our word with the dictionary word. If not, we removed the word from the reading.

Unfortunately the results were not good. Before applying this update, our readings were 98% correct, and this update only worsened the results. In some cases, replacing a word or deleting it resulted in a much less coherent reading that leaving a word wrong. For example, when the word “whichis” (wrong because the space between “which” and “is” was not recognized) was compared to a word from the dictionary, it would either be deleted or replaced with “which”, when we would much rather have it spoken as “whichis” than removing “is” entirely.

In the end, we abandoned all the previously mentioned ideas and proceeded with something different but quite efficient, an optimization that is described in section 6.7…

6.6) Additional optimizations/functionalities for the completely blind

Apart from the optimizations regarding the reading of the page, we could also make adjustments so that the glasses would be handy for, not just visually impaired, but totally blind people, too. **For example, a blind person would have difficulty in knowing when the camera is in the correct position above a page and keep it still there. It would also be difficult for him/her to recognize when the blocks appeared on the screen, which is the indication that the readings are complete and the camera can be moved away**. The following last optimization and last extra functionality are focused specifically on aiding the completely blind users…

6.7) Ocr-reader - Optimization 3: Optimizing the accuracy (2) + eliminating the need for the user to know when the readings are complete

In section “6.5) Ocr-reader - Optimization 3: Optimizing the accuracy” we have been trying to increase the accuracy of the longest reading above 98% by modifying its contents, but to no avail. Seeing that modifying the contents of the reading lead us nowhere, we decided to do something else:

Instead of choosing the longest reading (most characters**) we decided to use the "similarity" (Leveshtein) function we mentioned in section "6.5) Ocr-reader - Optimization 3: Optimizing the accuracy" to find the two readings that were most similar out of the total 'x' readings. Then we would choose one of the two as our reading to be spoken at the "FRW" button press**. The logic behind this implementation is that, if a reading is extremely similar to another reading, the chances of them being accurate are greater than the chances of them being inaccurate. A reading with a lot of mistakes has very little chances of being similar with another reading since the exact same mistakes are hard to be repeated twice.

To prove that the above assumption was actually correct, we implemented a java program called "WordDistrurbing", which can be found in the "Java" folder. The program starts with an initial String containing the text in our book page. We then disturb that String by a percentage of 5, which means we replace a random 5% of its characters with other random characters. We store the disturbed String in a list, and then disturb the initial String again by a bigger percentage of 15, which means we replace a random 15% of its characters with other random characters and store the disturbed String in our list. We repeat this process until we reach a disturbance percentage of 95 and store a total of 10 disturbed Strings in our list. Finally, we take those 10 disturbed Strings and compare them to each other using the Levehstein function to find how similar they are. These are the average results collected:

String 0: 5%   disturbance
String 1: 15% disturbance
String 2: 25% disturbance
String 3: 35% disturbance
String 4: 45% disturbance
String 5: 55% disturbance
String 6: 65% disturbance
String 7: 75% disturbance
String 8: 85% disturbance
String 9: 95% disturbance

The numbers in parentheses represent the Strings compared and the following number is their similarity percentage in a scale from 0 to 1:

```
(0, 1) = 0.8108747044917257
(0, 2) = 0.7249802994483846
(0, 3) = 0.6304176516942475
(0, 4) = 0.5374310480693459
(0, 5) = 0.44602048857368004
(0, 6) = 0.35973207249802996
(0, 7) = 0.2628053585500394
(0, 8) = 0.1773049645390071
(0, 9) = 0.10795902285263988
(1, 2) = 0.6473601260835303
(1, 3) = 0.5626477541371159
(1, 4) = 0.4842395587076438
(1, 5) = 0.396769109535067
(1, 6) = 0.3219070133963751
(1, 7) = 0.23522458628841608
(1, 8) = 0.17257683215130024
(1, 9) = 0.10638297872340426
(2, 3) = 0.5011820330969267
(2, 4) = 0.43380614657210403
(2, 5) = 0.3668242710795902
(2, 6) = 0.2962962962962963
(2, 7) = 0.2127659574468085
(2, 8) = 0.16193853427895982
(2, 9) = 0.10874704491725769
(3, 4) = 0.37943262411347517
(3, 5) = 0.3187549251379039
(3, 6) = 0.2572892040977147
(3, 7) = 0.19818754925137905
(3, 8) = 0.152876280535855
(3, 9) = 0.10795902285263988
(4, 5) = 0.27344365642237983
(4, 6) = 0.2360126083530339
(4, 7) = 0.17533490937746257
(4, 8) = 0.13238770685579196
(4, 9) = 0.1099290780141844
(5, 6) = 0.20646178092986603
(5, 7) = 0.15405831363278172
(5, 8) = 0.13435776201733649
(5, 9) = 0.11977935382190702
(6, 7) = 0.1516942474389283
(6, 8) = 0.12450748620961387
(6, 9) = 0.11583924349881797
(7, 8) = 0.12135539795114263
```

```
(7, 9) = 0.11702127659574468
(8, 9) = 0.11899133175728921
```

Carefully observing the comparison results of String 0 (5% disturbance) to all the other Strings, we can see that, the less disturbed the other String (= more accurate), the more similar it is to String 0. However, the same does not apply for Strings of similar but high disturbance percentages. As we can see, Strings 8 and 9 have almost the same amount of disturbance but, since both disturbance percentages are high, they are quite dissimilar.

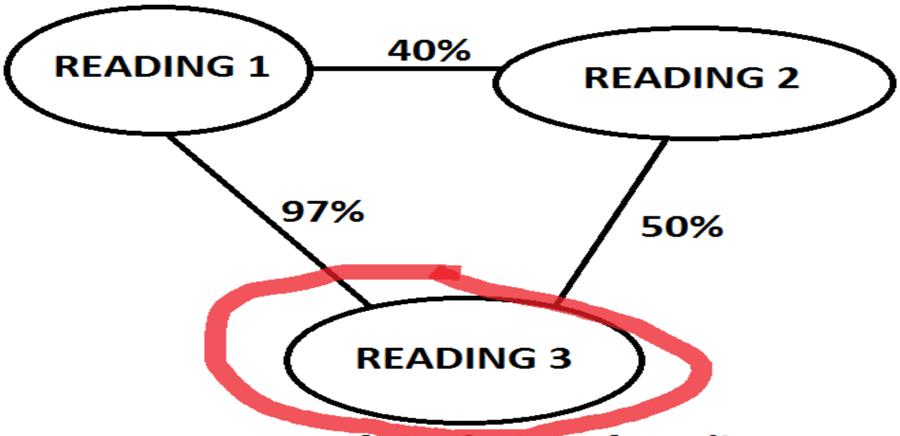
The above results and derivations further enhance the strength of our assumptions. To be completely sure, we additionally studied comparisons between 10 Strings with disturbance percentages from 0% up to 10%, and the derivations were the same.

Moving on, if we recall the modifications mentioned in section “6.2) Ocr-reader - Optimization 2: Choosing the reading with the most recognised characters (+ initializing our text-to-speech engine)”, pressing the “FRW” button would result in the application choosing the longest reading (most characters) out of those readings already collected. That caused a little problem: As we mentioned in section “6.6) Additional optimizations /functionalities for the completely blind”, it would be difficult for a blind user to know when the “reading collection” process was complete so that he presses the button. In order to fix this issue, **we changed the functionality of the “FRW” button like this: After the button is pressed, the application waits until it collects a total of ‘x’ readings. Then it applies that “similarity” heuristic mentioned above to the collected readings**. **After the best reading has been chosen, it is immediately spoken to the user**. This way, the user does not need to know how long to wait above a page for the readings to be collected, since, after the button is pressed, the application will start reading the chosen reading immediately after it has been decided.

(What number is the ‘x’ number of readings mentioned above is discussed in a later section, as this is a matter of balancing speed and accuracy for our application).

This change in the functionality of the “FRW” button gave our application another advantage: By performing slight modifications on our TextToSpeech engine, we could now begin preparing for a new reading while listening to the current reading!! How? Well, before each chosen reading is spoken, the user will listen to the application saying “You may proceed to the next page”. **After that phrase is heard, the user can press the “FRW” button to have the application prepare for the next reading to be spoken, while he still listens to the current one. As a result, a lot of valuable time can be saved**. If the next reading is ready to be spoken before the current one is over, the application adds it to a queue and starts speaking it immediately after the current one is over.

Here is a graph representing this new “similarity” heuristic for choosing the best reading and how we modified the code to achieve that, as well as how we modified the code to change the “FRW” button’s functionality: [Code: Appendix section R]



Edges: percentage of similarity of readings

6.8) Ocr-reader – Extra functionality 2: Letting the blind user know when the camera is correctly positioned above a page

Yet another important task we had at hand was finding a way to help a completely blind user know when the camera was correctly positioned above a page. This obviously had to be done using sounds as directional signals. At the same time, we didn't want to tamper with the user's temper by adding too many or too irritating sounds. So we proceeded as follows:

First of all, the directional signals are gentle beeps. There are 5 types of beeping, from the shortest and dullest one to the longest and most excitatory one. Short, continuous beeps means the user is probably not even above the page. Longer, less frequent beeps means the user is correctly positioned above the page and, the more excitatory the beep, the better his position. But what do the beeps translate to technically? Well, **the type of the beep depends on how much of the camera's field of view the blocks of a reading have covered**: If the reading only contains a small block, it means the application only manages to read a small part of the page at the user's current position. If the reading contains a lot of big blocks, it means the application probably has a whole page to it available for analysis, and so the user is probably correctly positioned above the page.

Why did we use this heuristic? Obviously it has some drawbacks: Whether you are very close to the page or further away, as long as the camera's field of view is full of text, the beeping will be as excitatory as possible, indicating you are at a good position above the page. How can we overcome this problem? The only obvious additional criterion is the amount of text enclosed in our camera's field of view: the more text, the better. However we can't correlate that with the type of beeping, it's impossible to decide how much text corresponds to what type of beeping. And so this seems to remain a drawback. Then, can we think of a completely different heuristic? Should we go lower? Should we focus on the pixels of each image to make a correlation between them and the types of beeping, rather than focusing on the blocks resulting by the image's analysis? The thing is, the lower we go, the more time these checks will take and the more they would slow our application for insignificant reasons: checking pixels to define if the camera’s field of view is full of text would be an a lot heavier and time consuming operation than just the calculating the area of the blocks of each reading. Eventually, we decided that the current heuristic was one of the best possible, and the best one thought of.

Initially, the beeping is OFF. This is because, even though the beeping is disabled when the application is speaking or after the press of a button, it can still be annoying for some people and they might not want to listen to it at times**. To turn the beeping ON, the user has to speak the number “nine” (9) and then long press the Vuzix M100 “ACTION” button**, exactly like he would do if he wanted to listen to the 9th page he listened to before. This was a simple and easy way to handle the beeping without expanding the possible inputs by the user (voice-numbers and Vuzix buttons). The beeping can be turned on and off as many times as the user wants to.

[Code: Appendix section Y]

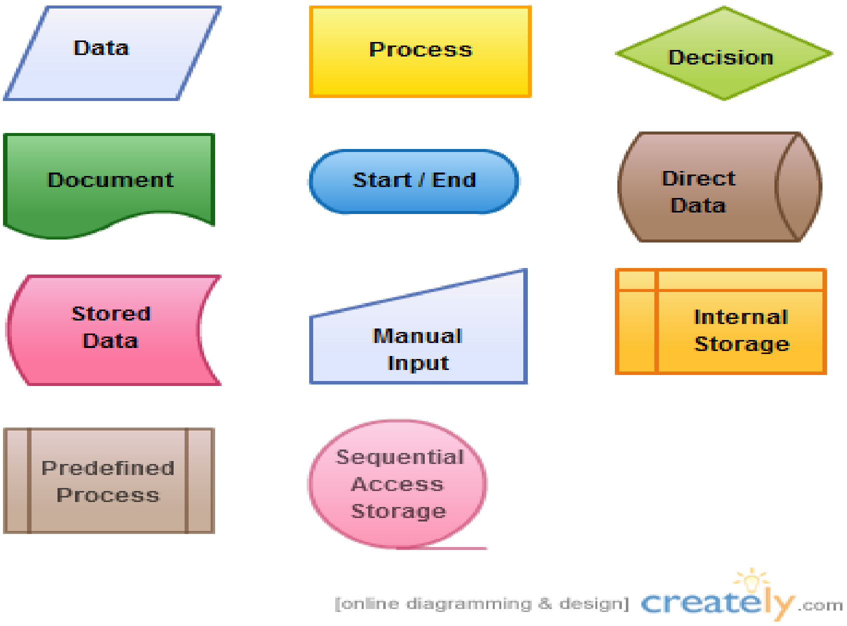
7) The “Book Eyessistant” (The final product)

7.1) Final overview of the abstract architecture

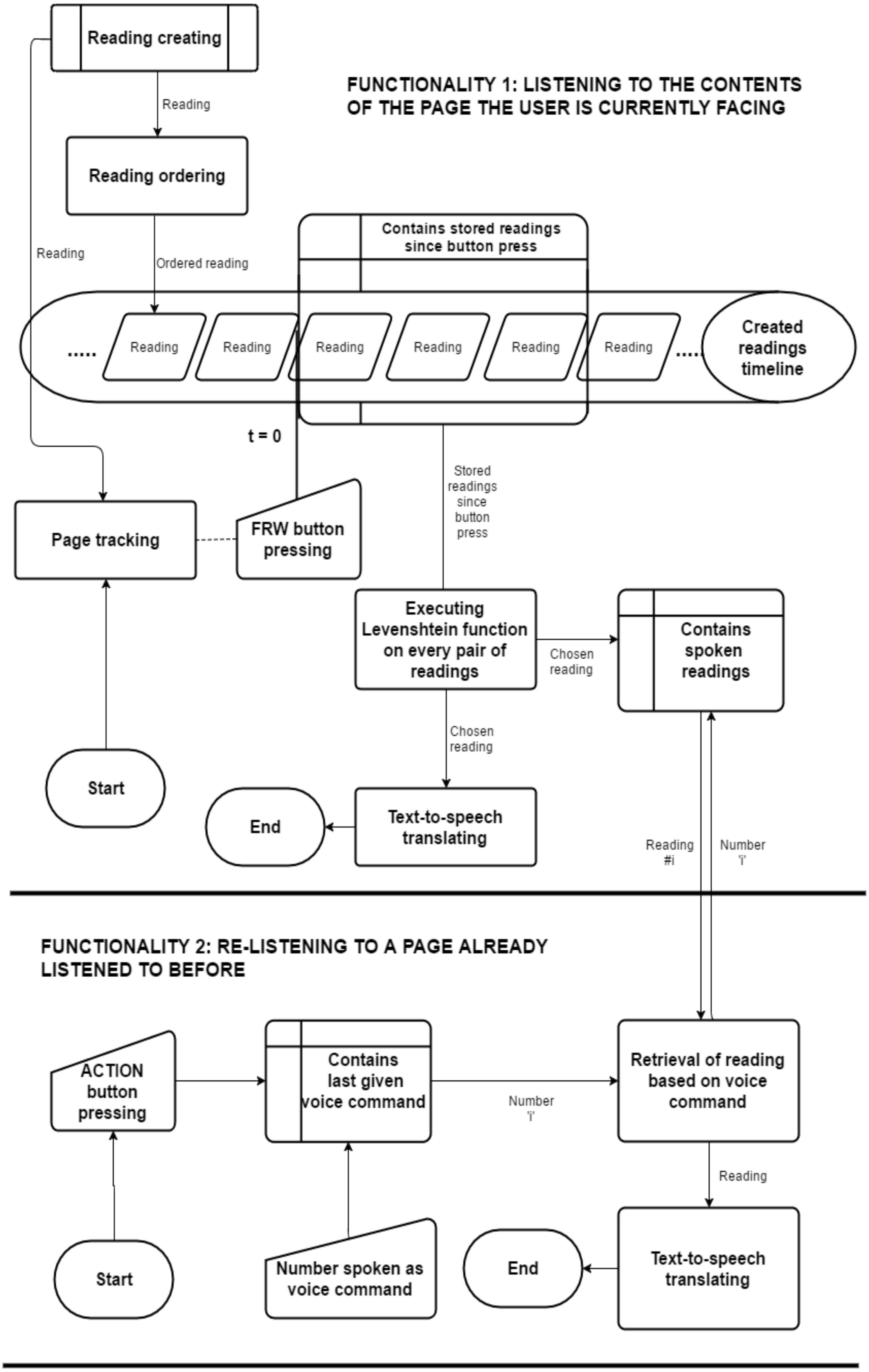
Our application for the Vuzix M100 is finally complete!! The final code of the application can be found in the “Final code of the application” folder.

Up to this point we have been describing the building process step by step, so let us know present its completed abstract architecture in a UML-like fashion:

Legend



Architecture


Reading creating
Reading
FUNCTIONALITY 1: LISTENING TO THE CONTENTS OF THE PAGE THE USER IS CURRENTLY FACING
Reading ordering
Reading
Ordered reading
Contains stored readings since button press
.....
Reading
Reading
Reading
Reading
Reading
Reading
....
Created readings timeline
t = 0
Stored readings since button press
Page tracking
FRW button pressing
Executing Levenshtein function on every pair of readings
Chosen reading
Contains spoken readings
Chosen reading
Start
End
Text-to-speech translating
Reading #i
Number 'i'
FUNCTIONALITY 2: RE-LISTENING TO A PAGE ALREADY LISTENED TO BEFORE
ACTION button pressing
Contains last given voice command
Number 'i'
Retrieval of reading based on voice command
Reading
Start
Number spoken as voice command
End
Text-to-speech translating

Brief description of units

--Reading creating (predefined process): This process continuously takes photographs of the camera's field of view and analyses them using OCR. For each analyzed photograph, it returns a list of TextBlocks, each TextBlock containing a String, representing everything written on the photograph captured. This returned list of TextBlocks is called a "reading".

--Reading ordering (process): This process reorders the list of TextBlocks of a given reading so that, if TextBlock 'i' is before TextBlock 'j', then the String of TextBlock 'i' is found before the String of TextBlock 'j' on the captured photograph. It does so using the coordinates (pixels) of every TextBlock, since the position of a TextBlock is represented by the pixels on the photograph where the String in the TextBlock was spotted.

--Created readings timeline (direct data): Represents the timeline of every reading created and ordered. Most readings in this timeline are not handled and lost.

--Page tracking (process): This process tries to indicate to the user when he has correctly positioned the camera above the page to be read. It calculates the total area of the TextBlocks of a reading, for each and every reading given. Depending on the "calculated area of reading/area of camera's field of view" ratio, the process makes certain beep sounds to help a user track and position himself correctly above the page, since bigger TextBlocks area equals more characters recognized equals more text read.

--Executing Levenshtein function on every pair of readings (process): The Levenshtein function returns a percentage that indicates how similar two Strings are. This process executes the famous "Levenshtein function" on every possible pair of readings, and returns the most identical pair. The readings this process acts upon are those stored after the user presses the FRW button to listen to the page he is currently facing.

One reading of the pair returned by this process is stored in a list of readings that have been listened-to by the user.

--Text-to-speech translating (process): This process takes a reading and speaks it to the user.

--Retrieval of reading based on voice command (process): This process takes the stored voice command and retrieves the corresponding stored reading from the list of spoken readings. How it does that is described in detail in section "6.4) Ocr-reader – Extra functionality 1: Choosing which already listened-to page we wish to hear using voice commands".

Description of the two main functionalities

We will not describe the exact steps that the user takes per functionality, the usage instructions are presented in a later section. Instead, here we will focus on explaining the flow of the program regarding the architecture presented above.

--**Functionality 1: Listening to the contents of the page the user is currently facing:** The program continuously creates readings using the "reading creating" process. Every reading created is passed through 2 processes: the "reading ordering" process which orders its blocks and the "page tracking" process which uses it to generate beeping sounds (directional signals) for the user. Once the user believes he is correctly positioned above the page and presses the FRW button, the following 'x' readings created are stored in a list. Once at least 'x' reading have been stored, the readings in the list are passed to the "Executing Levenshtein function on every pair of readings" process, which chooses one out of the two most similar readings. The chosen reading is spoken to the user by the "Text-to-speech translating" process and is stored in another list of spoken readings.

--**Functionality 2: Re-listening to a page already listened-to before:** The user uses voice commands (numbers), and the last voice command received and recognized is stored by the program. When the user decides to press the ACTION button, that last stored voice command (number) is passed on to the "Retrieval of reading based on voice command" process which retrieves the corresponding reading and finally passes it on the "Text-to-speech translating" process which speaks it to the user.

7.2) Final capabilities analysis

Let us now test our application and, since it's a finished product, analyze its capabilities in more depth and detail. Recall that in section "5.1) Operation and functionality of the Ocr-reader" we took the following attributes in account:

- Distance of camera from page
- Time to process words before presenting them on the screen
- Accuracy of readings (words/sentences read correctly)
- Percentage of readings (words read out of the total number of words)
- Block sizes
- Ordering of blocks

This time we will take a different approach for our analysis: We will ignore the last two attributes (blocks sizes and ordering of blocks) since we have already fully resolved any issues regarding those two attributes (words within blocks are 100% correctly ordered and blocks themselves are also 100% correctly ordered thanks to our first optimization). What we will do is, we will attempt to read the test-page of our book from 6 different positions. Each position will be allowing us to read a different percentage of the page: The 6 different positions are:

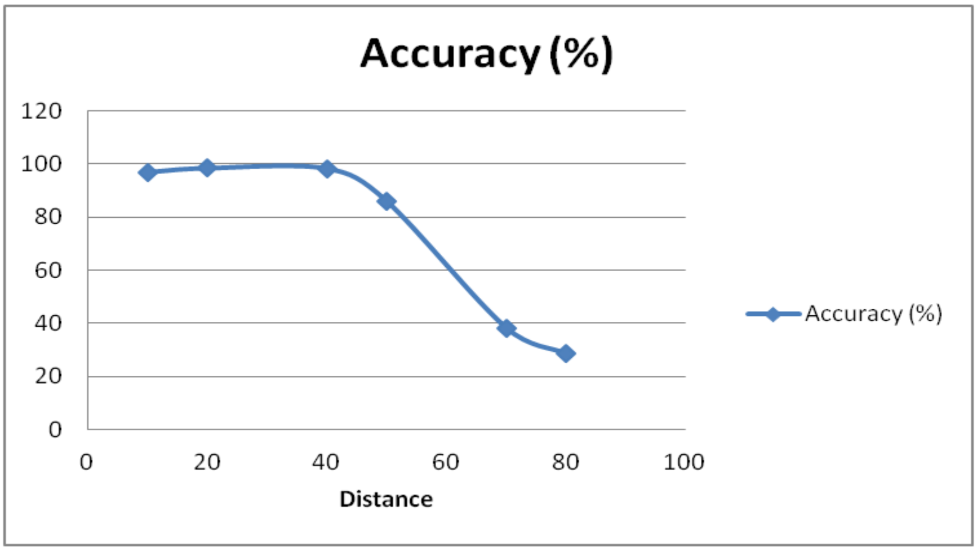
- Position 1: 80 cm above the page, allows us to read more than the whole page
- Position 2: 70 cm above the page, allows us to read the whole page
- Position 3: 50 cm above the page, allows us to read 75% of the page
- Position 4: 40 cm above the page, allows us to read 50% of the page
- Position 5: 20 cm above the page, allows us to read 25% of the page
- Position 6: 10 cm above the page, allows us to read 10% of the page

For each position, we will take the spoken reading and use our already implemented Levenshtein function to compare our reading to the original content of the page, and get a percentage of how similar the two are. Getting a percentage of how similar two strings are is actually like calculating the "accuracy of readings" and "percentage of readings" attributes both at the same time.

This analysis is focused on creating **a correlation between the distance from the page and the accuracy of our readings**, **a correlation between the distance from the page and the mistakes in the readings,** as well as **a correlation between the distance from the page and the time it takes for the application to speak the read text to us after the "FRW" button is pressed**.

Here are the results of the Levenshtein function (accuracy of readings) for each position. To see the program used for the collection of the following results, check the "WordSimilarity" java file in the "Java" folder. To see the actual captured readings, check the corresponding text files in the "Final accuracy tests" folder:

• Position 1: 80 cm above the page, whole page+:	**28.64460204885737 %**
• Position 2: 70 cm above the page, whole page:	**38.37667454688731 %**
• Position 3: 50 cm above the page, 75% of the page:	**86.21052631578947 %**
• Position 4: 40 cm above the page, 50% of the page:	**98.36734693877551 %**
• Position 5: 20 cm above the page, 25% of the page:	**98.43014128728415 %**
• Position 6: 10 cm above the page, 10% of the page:	**96.92307692307692 %**

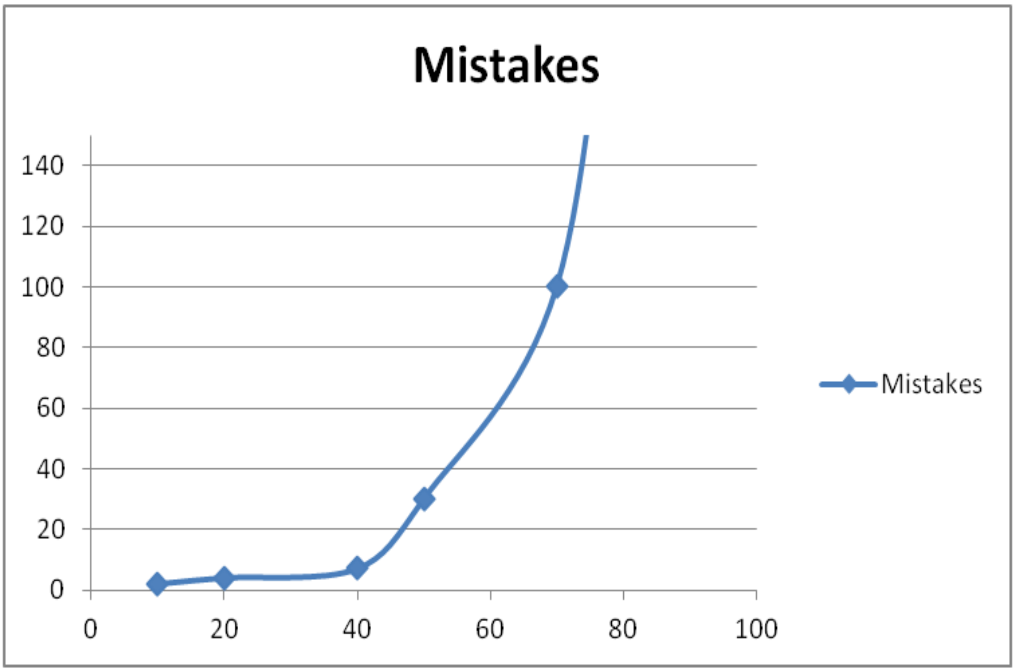


]

The reason that the average accuracy below 40cm from the page is close to or less than the average accuracy at 40cm from the page, is that, the smaller the distance, the fewer the words read, and the more impact one single mistake does to the accuracy percentage as a result.

Here is the correlation between the distance from the page and the mistakes in the readings: note that the following results were collected from the same average readings captured and analyzed in the previous correlation. Additionally, the mistakes were counted by a human because a human is more reliable in defying what a mistake is, for example if there is just one insignificant letter missing or if the whole world is unreadable. We did not count the mistakes for distances above 50cm distance from the page, since the readings were too incoherent for us to be able to identify specific mistakes in them.

- Position 3: 50 cm above the page, 75% of the page: **30+ mistakes**
- Position 4: 40 cm above the page, 50% of the page: **7 mistakes**
- Position 5: 20 cm above the page, 25% of the page: 4 mistakes
- Position 6: 10 cm above the page, 10% of the page: **2 mistakes**

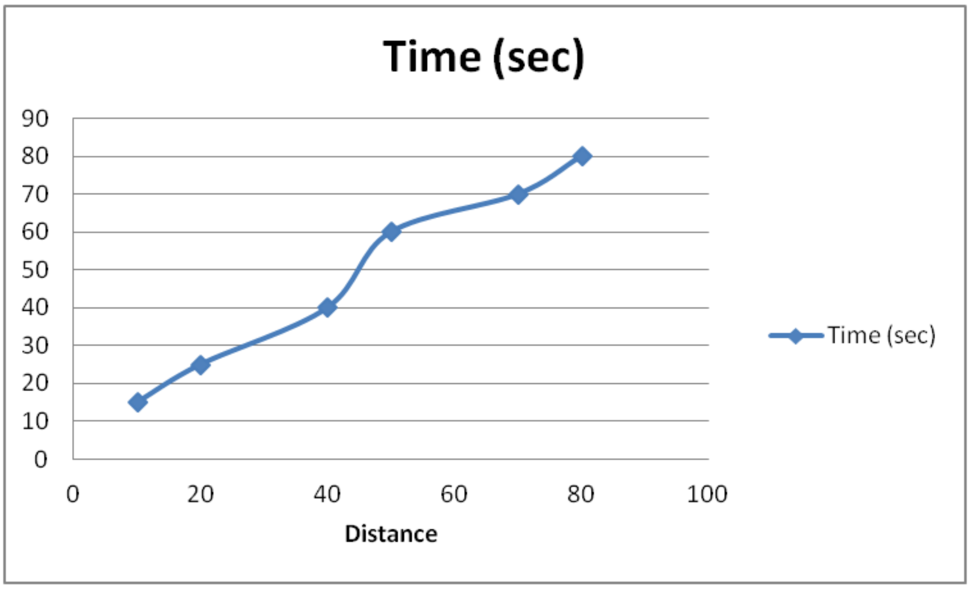


Let us now focus on the correlation between the distance from the page and the time it takes for the application to read the text for us after the "FRW" button is pressed. We previously mentioned that we perform measurements (like the Levenshtein distance) on the spoken reading for each distance. Remember that in section "6.7) Ocr-reader - Optimization 3: Optimizing the accuracy (2) + eliminating the need for the user to know when the readings are complete" we decided which reading that would be by comparing a total of 'x' captured readings with each other using the Levenshtein distance. How many are those 'x' readings? We had left that question unanswered.

Well, it doesn't seem like we really have a choice here. The minimum number of readings required for our last (3rd) optimization to function properly is 3, since there is no point in comparing a total of two readings with each other and picking one of the two. Now, setting the number of readings above 3 does increase the accuracy by a little, but also vastly increases the amount of time needed before the application starts speaking the chosen reading. And what the "time" tests showed us, as you will see below, is that we are not in a position to sacrifice any more time. The Vuzix M100 are quite slow when it comes to capturing readings. Are we doing something wrong here, or is this truly a hardware issue?

To confirm that the performance is only a hardware issue, we tested our final application on the powerful Android phone "CUBOT H2", which has a 1.3GHz quad-core processor and 3GB of RAM. You could press the "MENU" button on the phone and, having set the 'x' number of total required readings to 3, the phone would start speaking the text to you in less than 10 seconds!! How long do the Vuzix M100 take to start speaking the text when the number of captured readings required is set to 3? These are the results we collected:

• Position 1: 80 cm above the page, whole page+:	**1min+**
• Position 2: 70 cm above the page, whole page:	**1min+**
• Position 3: 50 cm above the page, 75% of the page:	**1 min**
• Position 4: 40 cm above the page, 50% of the page:	**40 sec**
• Position 5: 20 cm above the page, 25% of the page:	**25 sec**
• Position 6: 10 cm above the page, 10% of the page:	**15 sec**



Of course, the times mentioned above are an average, as this kind of measurement is extremely susceptible to bias. As you can see, the difference in speed between the CUBOT H2 and the Vuzix M100 is tremendous. (And we can't modify the code to make the Vuzix M100 faster since we can't make the whole text recognition process any faster, we use a non-modifiable built-in Google library for that). As we said, we don't really have a choice here, as 3 is the minimum number of readings required to be captured for our application to function correctly. Good thing is, the last modification in the functionality of the "FRW" button, the one that enabled us to listen to a reading and prepare for the next reading concurrently, makes this negative impact of time on our application a lot less significant, and saves the user valuable time.

Based on the above statistics we decided that, for our Vuzix M100 to function optimally, the following requirements must be met:

1. The person wearing the Vuzix M100 must have his head at most 40cm away from the page he is reading if he wishes for a truly accurate reading (considering he is trying to read a densely populated page full of small words).
2. The number of readings to be captured and analyzed after the "FRW" button is pressed is 3. This is the optimal number of readings for both accuracy and acceptable speed (for the Vuzix M100, of course).

7.3) Overview and usage instructions

Having completed the final analysis of our product, let us review what he have achieved, or better yet, let us give its usage instructions through which the whole product’s capabilities will be demonstrated. We will be describing the usage instructions for people with all kinds of visual disabilities (blind, visually impaired, visually healthy) when they are attempting to read an open book.

1. Open the book on the desk in front of you and set a directable lamp next to it, so that you can make the lamp point at a page without it being directly above the page (where you will be when reading it).
2. Assuming the Vuzix M100 are assembled and ready, wear them and run the “Book Eyessistant” application.
3. In the first screen, make sure the “Auto Focus” button is selected and the “Use Flash” button is unselected, and press the “Detect Text” button. Blind people don’t need to make these checks since that is the default configuration, all they have to do is press the “ACTION” button to initiate the camera and text detector.
4. Take 2 white sheets of paper and cover the side of the book you are not going to read yet, as well as the half of the page you are not going to focus on (since we suggested that the glasses read half a page at most at a time).
5. Move your head above the page you want to read, and as close to the text as possible. Blind people may want to activate the page tracking functionality first, by speaking the number ‘9’ and long pressing the “ACTION” button. Then they can listen to the intensity of the beeps to ensure they are indeed correctly positioned above the page.
   (During this time, the application continuously collects readings and sorts the blocks in them, storing them in its special “readings” storage).
6. When you feel ready, long press the “FRW” button and wait until the application speaks the page to you. Keep your head above the page until you hear the application speaking, and then you can move it away.

(As we said, after the “FRW” button is pressed, the application captures the next 3 readings, compares every reading to each other and selects one of the two most similar to be spoken).

7. After you‘re done listening to the page, repeat the steps 4-6 for the next page/part of the page. Optionally, you can repeat the steps 4-6 even before you are done listening to the current page.
8. If you want to listen to a previously listened-to page again, just speak a number from 1 to 9, say ‘i’, and long press the “ACTION” button to listen to the ‘i’-th page you listened-to being spoken again.
   (Here the application just retrieves the ‘i’-th page it read from the “read pages” storage and speaks it, if it correctly recognized the number ‘i’ being spoken by the user).
9. To exit the application, long press the “BACK” button.

7.4) What does the “Book Eyessistant” do better than other applications of its kind?

This whole journey of creating our own application that reads books for the visually impaired has been extraordinary, and the general success of it leaves us greatly satisfied. However, apart from the personal benefits we gained from it, is this application, in combination with the Vuzix M100 on which it runs, superior on any aspect than any other application of its kind? Does the “Book Eyessistant” do something better than other applications of its kind? Well…

The whole policy we followed when we decided to create this application was: “**Use the Vuzix M100 in a way that takes advantage of their special capabilities. Make them do one specific thing, and do it well**”.

And it turned out pretty well… Here are some of the special advantages our application, in combination with the Vuzix M100, has:

- It has been built so that it is suitable for even completely blind people. You can easily interact with the application without ever needing to look at the glasses' screen, which, as we have said previously, can be tiresome.
- The interaction is extremely easy, too. You do what you do using only 2 buttons, those on the Vuzix M100. You don't need an additional device connected, like an Android phone for example, to handle the glasses. That brings us to our next point…
- It is a completely independent application. That is, once it is installed on the Vuzix M100, it doesn't need any other device connected to them to function.
- It takes advantage of the Vuzix M100's high-quality Speech Recognition system.
- You do what you do hands-free. The only time you need to use your hands is when you press a button to have the application speak a page to you.
- Everything you do requires as little movement as possible. Both the microphone and the speaker are directly next to your head, so you do not need to move at all to make the application listen to you, or to listen to the application, like you would do with an Android phone for example.
- The application is designed to specifically handle a screen full of text, like a book page. Where a screen full of text would be another application's "weakness", something that it wouldn't be able to handle that well, our application is designed specifically to handle such situations, and it does that quite well.
- You can listen to the application speak a page at the same time that it, concurrently, prepares itself to speak the next page.
- The Vuzix M100's camera and video are top quality, which in return results in high quality readings. Also, the fact that they are glasses helps reduce trembling by a lot, and that too, adds to the "quality part". If we were talking about a phone, the trembling of the hand holding it would be much greater, resulting in lower quality readings.

Having laid out the advantages of our application, we ought to also mention its disadvantages:

- The Vuzix M100's greatest disadvantage by far is that they are slow when taking readings through our application. And that is a hardware issue: there is no way we could modify the code to make the text recognition faster, after all, we use a non-modifiable built-in Google library for that.
- The previous disadvantage results in the user being required to stand above a page and look down for about 40 seconds after he presses the "FRW" button, until the page starts being spoken and he can finally lay back. That is sometimes tiresome and can result in headaches. There is no known way around this problem, since, as we have said multiple times by now, the speed is a hardware problem. Luckily, this waiting time's negative impact is reduced by the fact that the user can listen to the application speak the most recently available page during that time.
- The Vuzix M100 can easily become uncomfortable if they are not first assembled according to the person that is about to wear them. They can be especially slippery when the user looks down to a page and may sometimes require him to hold them so that they don't fall. That problem however can easily be fixed by adjusting them better or strapping them on your head.

A) Appendix

Here we will be presenting the changes we made to the code of the “Ocr-reader” Android project, and the code we added, to turn it into our own Android application for the Vuzix M100, the “Book Eyessistant”.

Each section’s name is the current page’s letter (A-Z).

For every section of this appendix, the representation of the contents will be as follows:

bold line -> code -> arrows -> bold line -> code -> separating line -> ……

--The **bold** line represents the class.

--Below the class is the original piece of code.

--The arrows ➔➔➔➔➔➔ mean that what follows is how we modified that original piece of code.

--Below the arrows and the next bold class name is the modified code. The red lines represent the code that was added or changed.

--The separating line ---------------------- means that that class is done with, and the section may or may not continue with another class.

Section B

OcrProcessorDetector.java

```
@Override
 public void receiveDetections(Detector.Detections<TextBlock> detections) {
          mGraphicOverlay.clear();
          SparseArray<TextBlock> items = detections.getDetectedItems();
         for (int i = 0; i < items.size(); ++i) {
                   TextBlock item = items.valueAt(i);
                   OcrGraphic graphic = new OcrGraphic(mGraphicOverlay, item);
                   mGraphicOverlay.add(graphic);
          }
}
```

➔➔➔➔➔➔

OcrProcessorDetector.java

```
@Override
public void receiveDetections(Detector.Detections<TextBlock> detections) {
         mGraphicOverlay.clear();
          SparseArray<TextBlock> items = detections.getDetectedItems();
          for (int i = 0; i < items.size(); ++i) {
                   TextBlock item = items.valueAt(i);
                   if (item != null && item.getValue() != null){
                             Log.i("blax", item.getValue());
                    }
                   OcrGraphic graphic = new OcrGraphic(mGraphicOverlay, item);
                   mGraphicOverlay.add(graphic);
          }
 }
```

Section C

OcrCaptureActivity.java

```
private void createCameraSource(boolean autoFocus, boolean useFlash) {
      ……..
      // Creates and starts the camera.  Note that this uses a higher resolution in comparison
       // to other detection examples to enable the text recognizer to detect small pieces of text.
       mCameraSource =
      new CameraSource.Builder(getApplicationContext(), textRecognizer)
      .setFacing(CameraSource.CAMERA_FACING_BACK)
      .setRequestedPreviewSize(1280, 1024)
      .setRequestedFps(2.0f)
      .setFlashMode(useFlash ? Camera.Parameters.FLASH_MODE_TORCH : null)
      .setFocusMode(autoFocus ? Camera.Parameters.FOCUS_MODE_CONTINUOUS_PICTURE :
null)
      .build();
}
```

➔➔➔➔➔➔

OcrCaptureActivity.java

```
private void createCameraSource(boolean autoFocus, boolean useFlash) {
       ………
       // Creates and starts the camera.  Note that this uses a higher resolution in comparison
       // to other detection examples to enable the text recognizer to detect small pieces of text.
       mCameraSource =
       new CameraSource.Builder(getApplicationContext(), textRecognizer)
       .setFacing(CameraSource.CAMERA_FACING_BACK)
       .setRequestedPreviewSize(1920, 1080)
       .setRequestedFps(30.0f)
       .setFlashMode(useFlash ? Camera.Parameters.FLASH_MODE_TORCH : null)
       .setFocusMode(autoFocus ? Camera.Parameters.FOCUS_MODE_CONTINUOUS_PICTURE :
null)
       .build();
}
```

Section D

OcrProcessorDetector.java

```
@Override
public void receiveDetections(Detector.Detections<TextBlock> detections) {
            mGraphicOverlay.clear();
             SparseArray<TextBlock> items = detections.getDetectedItems();
             for (int i = 0; i < items.size(); ++i) {
                        TextBlock item = items.valueAt(i);
                        if (item != null && item.getValue() != null){
                                    Log.i("blax", item.getValue());
                         }
                        OcrGraphic graphic = new OcrGraphic(mGraphicOverlay, item);
                        mGraphicOverlay.add(graphic);
             }
 }
```

➔➔➔➔➔➔

OcrProcessorDetector.java

```
@Override
public void receiveDetections(Detector.Detections<TextBlock> detections) {
            mGraphicOverlay.clear();
            SparseArray<TextBlock> items = detections.getDetectedItems();
             //We will move each read block in this array, and sort the
            //blocks based on our preferences
             ArrayList<TextBlock> items2 = new ArrayList<>(items.size());
             for (int i = 0; i < items.size(); ++i) {
                    TextBlock item = items.valueAt(i);
                    if (item != null && item.getValue() != null){
                        //move block to our new array
                        items2.add(item);
                    }
                   OcrGraphic graphic = new OcrGraphic(mGraphicOverlay, item);
                   mGraphicOverlay.add(graphic);
             }
             //sort our new array based on the following criteria...
             Collections.sort(items2, new Comparator<TextBlock>() {

             public int compare(TextBlock block1, TextBlock block2) {

                      //get position of blocks (upper-left corner)
                      Point point1 =  block1.getCornerPoints()[0];
                      Point point2 =  block2.getCornerPoints()[0];
                      //if the blocks are at the same position, return that
                      //they are identical
                      if (point1.y == point2.y && point1.x == point2.x)
                                  return 0;
                      //if the first block is quite higher than the other,
                      //place it earlier in the array
                      if (point1.y < point2.y && point2.y - point1.y > 20){
                                   return -1;
                      }
```

```
                //if the first block is about the same height as the other
                //(which possibly indicates that they are at the same line)
                //check their horizontal possition and place the left one
                //earlier in the array
                else if (point1.y < point2.y){
                        if (point1.x < point2.x)
                                return -1;
                        else return 1;
                }
                //if the second block is quite higher than the first
                //block, place it earlier in the array
                else return 1;
            }

        });

        //Write the blocks of the now sorted array in the log
        for (int i = 0; i < items2.size(); ++i) {
            TextBlock item = items2.get(i);
            if (item != null && item.getValue() != null){
                    Log.i("blax", item.getValue() + "   " + item.getCornerPoints()[0].toString());
            }
        }
}
```

Section F

OcrProcessorDetector.java

```
private GraphicOverlay<OcrGraphic> mGraphicOverlay;

 OcrDetectorProcessor(GraphicOverlay<OcrGraphic> ocrGraphicOverlay) {
            mGraphicOverlay = ocrGraphicOverlay;
 }

  /**
  * Called by the detector to deliver detection results.
  * If your application called for it, this could be a place to check for
  * equivalent detections by tracking TextBlocks that are similar in location and content from
  * previous frames, or reduce noise by eliminating TextBlocks that have not persisted through
  * multiple detections.
  */
  @Override
  public void receiveDetections(Detector.Detections<TextBlock> detections) {
            …………………………………
           //sort our new array based on the following criteria...
           Collections.sort(items2, new Comparator<TextBlock>() {
                    public int compare(TextBlock block1, TextBlock block2) {
                            …………………………..
                    }
            );

           //Write the blocks of the now sorted array in the log
           for (int i = 0; i < items2.size(); ++i) {
                   …………………………..
           }
}
```

➔➔➔➔➔➔

OcrDetectorProcessor.java

```
  private GraphicOverlay<OcrGraphic> mGraphicOverlay;
  //The list that contains all the readings
  private ArrayList<ArrayList<TextBlock>> reading;


  OcrDetectorProcessor(GraphicOverlay<OcrGraphic> ocrGraphicOverlay) {
            mGraphicOverlay = ocrGraphicOverlay;
           //Initialization of the list of readings
           reading = new ArrayList<ArrayList<TextBlock>>();
  }

  //returns the list of readings to the class that requests it
  public ArrayList<ArrayList<TextBlock>> getReading(){
            return reading;
  }

  //empties the list of reading when a class commands so
  public void clearReading(){
```

```
            reading.clear();
    }

    /**
     * Called by the detector to deliver detection results.
     * If your application called for it, this could be a place to check for
     * equivalent detections by tracking TextBlocks that are similar in location and content from
     * previous frames, or reduce noise by eliminating TextBlocks that have not persisted through
     * multiple detections.
     */
    @Override
    public void receiveDetections(Detector.Detections<TextBlock> detections) {
             …………………………………..
            //sort our new array based on the following criteria...
            Collections.sort(items2, new Comparator<TextBlock>() {
                     public int compare(TextBlock block1, TextBlock block2) {
                             …………………………..
                    }
             );

            //Add this reading to the list of readings
            reading.add(items2);
            //Write the blocks of the now sorted array in the log
            for (int i = 0; i < items2.size(); ++i) {
                    …………………………..
            }
    }
```

OcrCaptureActivity.java

```
……………….
private CameraSource mCameraSource;
 private CameraSourcePreview mPreview;
private GraphicOverlay<OcrGraphic> mGraphicOverlay;
………….
/**
  * Initializes the UI and creates the detector pipeline.
  */
  @Override
 public void onCreate(Bundle icicle) {
         super.onCreate(icicle);
        setContentView(R.layout.ocr_capture);

         mPreview = (CameraSourcePreview) findViewById(R.id.preview);
        mGraphicOverlay = (GraphicOverlay<OcrGraphic>) findViewById(R.id.graphicOverlay);

         ………………..
 }

…………………

/**
* Creates and starts the camera.  Note that this uses a higher resolution in comparison
* to other detection examples to enable the ocr detector to detect small text samples
* at long distances.
 *
* Suppressing InlinedApi since there is a check that the minimum version is met before using
* the constant.
*/
@SuppressLint("InlinedApi")
private void createCameraSource(boolean autoFocus, boolean useFlash) {
        Context context = getApplicationContext();

         // A text recognizer is created to find text.  An associated processor instance
        // is set to receive the text recognition results and display graphics for each text block
        // on screen.
        TextRecognizer textRecognizer = new TextRecognizer.Builder(context).build();
        textRecognizer.setProcessor(new OcrDetectorProcessor(mGraphicOverlay));
         …………………….
}

 ………………………..

/**
* Restarts the camera.
*/
@Override
 protected void onResume() {
         super.onResume();
        startCameraSource();
}

/**
* Stops the camera.
*/
@Override
 protected void onPause() {
        super.onPause();
        if (mPreview != null) {
                mPreview.stop();
         }
}

/**
* Releases the resources associated with the camera source, the associated detectors, and the
* rest of the processing pipeline.
 */
@Override
protected void onDestroy() {
        super.onDestroy();
        if (mPreview != null) {
                mPreview.release();
        }
 }
```

➔➔➔➔➔➔

OcrCaptureActivity.java

```
……………………….
private CameraSource mCameraSource;
private CameraSourcePreview mPreview;
private GraphicOverlay<OcrGraphic> mGraphicOverlay;
 //The one and only object of type "OcrDetectorProcessor"
private OcrDetectorProcessor detectorPro;
//Our text-to-speech engine
 private TextToSpeech tts;
………………………..

/**
* Initializes the UI and creates the detector pipeline.
*/
@Override
public void onCreate(Bundle icicle) {
                 super.onCreate(icicle);
                setContentView(R.layout.ocr_capture);

                mPreview = (CameraSourcePreview) findViewById(R.id.preview);
                 mGraphicOverlay = (GraphicOverlay<OcrGraphic>) findViewById(R.id.graphicOverlay);
                 //initialization of our text-to-speech engine,
                //recognises United States English
                 tts = new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
                                @Override
                                public void onInit(int status) {
                                                if (status != TextToSpeech.ERROR){
                                                                tts.setLanguage(Locale.US);
                                                }
                                }
                });
                ………………………
}

…………………………….

 /**
* Creates and starts the camera.  Note that this uses a higher resolution in comparison
* to other detection examples to enable the ocr detector to detect small text samples
* at long distances.
*
* Suppressing InlinedApi since there is a check that the minimum version is met before using
* the constant.
*/
@SuppressLint("InlinedApi")
private void createCameraSource(boolean autoFocus, boolean useFlash) {
                Context context = getApplicationContext();

                // A text recognizer is created to find text.  An associated processor instance
                 // is set to receive the text recognition results and display graphics for each text block
                 // on screen.
                 TextRecognizer textRecognizer = new TextRecognizer.Builder(context).build();
                detectorPro = (OcrDetectorProcessor) new OcrDetectorProcessor(mGraphicOverlay);
                textRecognizer.setProcessor(detectorPro);
```

```
    ………………………….
}

//This method changes the default action for pressing the MENU android button.
//Instead of its default action, it makes the program choose the best out of the
 //so far collected readings (the one that contains the most characters) and
 //stores it in a String, so that it is later on spoken by the ear speaker / otherwise
//utilised...
 @Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_MENU) {
        //Get the whole list of readings
        ArrayList<ArrayList<TextBlock>> whole = detectorPro.getReading();
        //This will point at the best reading
        ArrayList<TextBlock> chosen = null;
        int big_size = whole.size();
        int count = 0; int max_count = Integer.MIN_VALUE;
         //For every reading...
        for (int i=0; i<big_size; i++){
             ArrayList<TextBlock> temp = whole.get(i);
            int small_size = temp.size();
            count = 0;
            //For every block of this particular reading...
            for (int j=0; j<small_size; j++){
                 //Count the characters of that block and add it to
                //the total characters in this reading
                count += temp.get(j).getValue().length();
            }
            //If this reading contains more characters than any
            //other, set this as the best reading
            if (count > max_count){
                max_count = count;
                chosen = temp;
             }
        }

         //Create a String containing the whole best reading
        String toRead = " ";
         for (int i=0; i<chosen.size(); i++){
             toRead = toRead + " " + chosen.get(i).getValue();
         }
        //Based on the target android version for this application,
         //use the correct text-to-speech function to speak the
        //best reading to the user
         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
            ttsGreater21(toRead);
        } else {
            ttsUnder20(toRead);
         }
         detectorPro.clearReading();
        return true;
    }
    return super.onKeyDown(keyCode, event);
```

```
}

//The text-to-speech  function to be used in case the target android version
//is equal to or lower than API-20
@SuppressWarnings("deprecation")
private void ttsUnder20(String text) {
            HashMap<String, String> map = new HashMap<>();
             map.put(TextToSpeech.Engine.KEY_PARAM_UTTERANCE_ID, "MessageId");
            tts.speak(text, TextToSpeech.QUEUE_FLUSH, map);
}

//The text-to-speech function to be used in case the target android version
//is greater than API-20
@TargetApi(Build.VERSION_CODES.LOLLIPOP)
private void ttsGreater21(String text) {
             String utteranceId=this.hashCode() + "";
            tts.speak(text, TextToSpeech.QUEUE_FLUSH, null, utteranceId);
}

/**
 * Restarts the camera.
 */
@Override
protected void onResume() {
            super.onResume();
             startCameraSource();
}

/**
 * Stops the camera.
 */
@Override
protected void onPause() {
            super.onPause();
             if (mPreview != null) {
                        mPreview.stop();
            }
}

/**
 * Releases the resources associated with the camera source, the associated detectors, and the
 * rest of the processing pipeline.
 */
@Override
protected void onDestroy() {
            super.onDestroy();
            if (mPreview != null) {
                        mPreview.release();
              }
            if (tts != null){
                        tts.stop();
                         tts.shutdown();
             }
}
```

Section L

OcrCaptureActivity.java

```
……………………….
private CameraSource mCameraSource;
private CameraSourcePreview mPreview;
private GraphicOverlay<OcrGraphic> mGraphicOverlay;
 //The one and only object of type "OcrDetectorProcessor"
private OcrDetectorProcessor detectorPro;
//Our text-to-speech engine
 private TextToSpeech tts;
………………………

/**
* Initializes the UI and creates the detector pipeline.
*/
@Override
public void onCreate(Bundle icicle) {
         super.onCreate(icicle);
        setContentView(R.layout.ocr_capture);

        mPreview = (CameraSourcePreview) findViewById(R.id.preview);
         mGraphicOverlay = (GraphicOverlay<OcrGraphic>) findViewById(R.id.graphicOverlay);
         //initialization of our text-to-speech engine,
        //recognises United States English
         tts = new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
                @Override
                public void onInit(int status) {
                        if (status != TextToSpeech.ERROR){
                                tts.setLanguage(Locale.US);
                        }
                }
        });
        ………………………
}

……………………………

//This method changes the default action for pressing the MENU android button.
//Instead of its default action, it makes the program choose the best out of the
 //so far collected readings (the one that contains the most characters) and
 //stores it in a String, so that it is later on spoken by the ear speaker / otherwise
//utilised...
 @Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
         if (keyCode == KeyEvent.KEYCODE_MENU) {
                ……………………………………………………..

                 //Create a String containing the whole best reading
                String toRead = " ";
                 for (int i=0; i<chosen.size(); i++){
                        toRead = toRead + " " + chosen.get(i).getValue();
                }
                //Based on the target android version for this application,
                //use the correct text-to-speech function to speak the
                //best reading to the user
                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
                        ttsGreater21(toRead);
                } else {
                        ttsUnder20(toRead);
                }
                detectorPro.clearReading();
                return true;
        }
        return super.onKeyDown(keyCode, event);
}

…………………………………………………….

/**
* Restarts the camera.
*/
@Override
protected void onResume() {
        super.onResume();
        startCameraSource();
}

/**
  * Stops the camera.
 */
@Override
protected void onPause() {
        super.onPause();
        if (mPreview != null) {
                mPreview.stop();
        }
}

/**
* Releases the resources associated with the camera source, the associated detectors, and the
* rest of the processing pipeline.
*/
@Override
protected void onDestroy() {
        super.onDestroy();
        if (mPreview != null) {
                mPreview.release();
        }
        if (tts != null){
                tts.stop();
                tts.shutdown();
        }
}
```

➔➔➔➔➔➔

OcrCaptureActivity.java

```
……………………….
private CameraSource mCameraSource;
private CameraSourcePreview mPreview;
private GraphicOverlay<OcrGraphic> mGraphicOverlay;
 //The one and only object of type "OcrDetectorProcessor"
private OcrDetectorProcessor detectorPro;
//Our text-to-speech engine
 private TextToSpeech tts;
//Our Speech Recogniser
private VoiceControl vc;
//The set of complete paragraphs read so far
private List<String> paragraphs;
//the only recognisable words (numbers) by our Speech Recogniser
private String[] numbers = { "1", "2", "3", "4", "5", "6", "7", "8", "9"};
//The number of the paragraph to be read after the user chooses one by speaking a number
private String par_to_speak;
…………………………

/**
* Initializes the UI and creates the detector pipeline.
*/
@Override
public void onCreate(Bundle icicle) {
         super.onCreate(icicle);
        setContentView(R.layout.ocr_capture);

        mPreview = (CameraSourcePreview) findViewById(R.id.preview);
        mGraphicOverlay = (GraphicOverlay<OcrGraphic>) findViewById(R.id.graphicOverlay);
         //initialization of our text-to-speech engine,
        //recognises United States English
        tts = new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener(){
                @Override
                public void onInit(int status) {
                        if (status != TextToSpeech.ERROR){
                                tts.setLanguage(Locale.US);
                        }
                }
        });
        //initialize the Speech Recogniser, remove its original grammar
        //and set its grammar to be only the first 9 numbers
        vc = new myVoiceControl(this);
         vc.removeGrammar(Constants.GRAMMAR_BASIC);
        vc.setWordlist(numbers);
        vc.on();
        //initialize list of paragraphs read
        paragraphs = new LinkedList<>();
        ………………………
}

…………………………….
```

```
//Our Speech Recogniser class. Upon recognising a word, the function
//checks if it's a valid number, and if so, prints that number on the screen
 //and sets it to be the paragraph to be read once it's time...
class myVoiceControl extends VoiceControl {
     public myVoiceControl(Context context) {
          super(context);
    }

    @Override
    protected void onRecognition(String result) {
        for (int i=0; i<numbers.length; i++){
            if (numbers[i].equals(result)){
                Toast.makeText(getApplicationContext(), result,
                Toast.LENGTH_SHORT).show();
                par_to_speak = result;
                break;
             }
        }
     }

}

//This method changes the default action for pressing the MENU android button.
//Instead of its default action, it makes the program choose the best out of the
 //so far collected readings (the one that contains the most characters) and
 //stores it in a String, so that it is later on spoken by the ear speaker / otherwise
//utilised...
 @Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
     if (keyCode == KeyEvent.KEYCODE_MENU) {
        …………………………………………………………………..

         //Create a String containing the whole best reading
        String toRead = " ";
         for (int i=0; i<chosen.size(); i++){
             toRead = toRead + " " + chosen.get(i).getValue();
         }
        //add this paragraph to the storage of paragraphs
         if (paragraphs.size() >= 9){
            paragraphs.remove(0);
        }
         paragraphs.add(toRead);
        //Based on the target android version for this application,
         //use the correct text-to-speech function to speak the
        //best reading to the user
         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
            ttsGreater21(toRead);
        } else {
            ttsUnder20(toRead);
         }
         detectorPro.clearReading();
        return true;
    }
```

```
        return super.onKeyDown(keyCode, event);
}

………………………………………………



//Once the back button is pressed, check if a correct number has been
//previously spoken by the user, and, if so, take the last spoken number
//and speak to the user the corresponding paragraph based on that number.
@Override
public void onBackPressed(){
        if (par_to_speak != null){
                switch (par_to_speak){
                   case "1": if (paragraphs.size() >= 1) ttsUnder20(paragraphs.get(0)); break;
                   case "2": if (paragraphs.size() >= 2) ttsUnder20(paragraphs.get(1)); break;
                  case "3": if (paragraphs.size() >= 3) ttsUnder20(paragraphs.get(2)); break;
                  case "4": if (paragraphs.size() >= 4) ttsUnder20(paragraphs.get(3)); break;
                  case "5": if (paragraphs.size() >= 5) ttsUnder20(paragraphs.get(4)); break;
                  case "6": if (paragraphs.size() >= 6) ttsUnder20(paragraphs.get(5)); break;
                  case "7": if (paragraphs.size() >= 7) ttsUnder20(paragraphs.get(6)); break;
                  case "8": if (paragraphs.size() >= 8) ttsUnder20(paragraphs.get(7)); break;
                  case "9": if (paragraphs.size() >= 9) ttsUnder20(paragraphs.get(8)); break;
                 default: break;
              }
       }
       par_to_speak = null;
}

/**
 * Restarts the camera.
 */
@Override
 protected void onResume() {
        super.onResume();
        startCameraSource();
        if (vc != null){
                vc.on();
         }
}

/**
* Stops the camera.
*/
@Override
protected void onPause() {
        super.onPause();
        if (mPreview != null) {
                 mPreview.stop();
        }
        if (vc != null){
                 vc.off();
         }
}
```

```
//By overriding the onStop() function, we are able to set
//our application to terminate whenever the HOME button is pressed
//instead of the back button
@Override
 protected void onStop(){
         super.onStop();
        onDestroy();
}

/**
 * Releases the resources associated with the camera source, the associated detectors, and the
 * rest of the processing pipeline.
 */
@Override
 protected void onDestroy() {
        super.onDestroy();
        if (mPreview != null) {
                mPreview.release();
         }
        if (tts != null){
                tts.stop();
                tts.shutdown();
        }
        if (vc != null) {
                vc.off();
        }
 }
```

Section R

OcrCaptureActivity.java

```
@Override
public void onCreate(Bundle icicle) {
    ...................
    tts = new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
        @Override
        public void onInit(int status) {
                if (status != TextToSpeech.ERROR){
                        tts.setLanguage(Locale.US);
                }
        }
    });
    ....................
}
//This method changes the default action for pressing the MENU android button.
//Instead of its default action, it makes the program choose the best out of the
 //so far collected readings (the one that contains the most characters) and
 //stores it in a String, so that it is later on spoken by the ear speaker / otherwise
//utilised...
 @Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
         if (keyCode == KeyEvent.KEYCODE_MENU) {
                //Get the whole list of readings
                ArrayList<ArrayList<TextBlock>> whole = detectorPro.getReading();
                //This will point at the best reading
                ArrayList<TextBlock> chosen = null;
                int big_size = whole.size();
                int count = 0; int max_count = Integer.MIN_VALUE;
                  //For every reading...
                for (int i=0; i<big_size; i++){
                         ArrayList<TextBlock> temp = whole.get(i);
                        int small_size = temp.size();
                        count = 0;
                        //For every block of this particular reading...
                        for (int j=0; j<small_size; j++){
                                 //Count the characters of that block and add it to
                                //the total characters in this reading
                                count += temp.get(j).getValue().length();
                        }
                        //If this reading contains more characters than any
                        //other, set this as the best reading
                        if (count > max_count){
                                max_count = count;
                                chosen = temp;
                         }
                }

                  //Create a String containing the whole best reading
                String toRead = " ";
                 for (int i=0; i<chosen.size(); i++){
                        toRead = toRead + " " + chosen.get(i).getValue();
                    }
                    //Based on the target android version for this application,
                    //use the correct text-to-speech function to speak the
                    //best reading to the user
                    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
                        ttsGreater21(toRead);
                    } else {
                        ttsUnder20(toRead);
                    }
                    detectorPro.clearReading();
                    return true;
                }
                return super.onKeyDown(keyCode, event);
}

//The text-to-speech function to be used in case the target android version
//is equal to or lower than API-20
@SuppressWarnings("deprecation")
private static void ttsUnder20(String text) {
        HashMap<String, String> map = new HashMap<>();
        map.put(TextToSpeech.Engine.KEY_PARAM_UTTERANCE_ID, "MessageId");
        tts.speak(text, TextToSpeech.QUEUE_FLUSH, map);
}

//The text-to-speech function to be used in case the target android version
//is greater than API-20
@TargetApi(Build.VERSION_CODES.LOLLIPOP)
private static void ttsGreater21(String text) {
        String utteranceId= code + "";
        tts.speak(text, TextToSpeech.QUEUE_FLUSH, null, utteranceId);
}
```

➔➔➔➔➔➔

OcrCaptureActivity.java

```
@Override
public void onCreate(Bundle icicle) {
    ...................
    tts = new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
        @Override
        public void onInit(int status) {
            if (status != TextToSpeech.ERROR){
                tts.setLanguage(Locale.US);
                tts.setOnUtteranceProgressListener(new UtteranceProgressListener() {
                    @Override
                    public void onStart(String utteranceId) {}
                    @Override
                    public void onDone(String utteranceId) {}
                    @Override
                    public void onError(String utteranceId) {}
                });
            }
        }
    });
    .....................
}

//This method changes the default action for pressing the MENU android button.
//Instead of its default action, it makes the program initialize a process of collecting
//readings by setting the "check" variable to 1 in the "OcrDetectorProcessor"
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
            if (keyCode == KeyEvent.KEYCODE_MENU) {
                        code = this.hashCode();
                        detectorPro.setCheck();
            }
            return super.onKeyDown(keyCode, event);
}

//This method just continues the work of the "OcrDetectorProcessor"
//here in this class, since every variable we need to modify is in
//this class and there is no need to implement intercommunication
//of the two classes
public static void continueAction(String toRead) {
            //add this paragraph to the storage of paragraphs
            if (paragraphs.size() >= 9) {
                                    paragraphs.remove(0);
            }
             paragraphs.add(toRead);
            //Based on the target android version for this application,
            //use the correct text-to-speech function to speak the
            //best reading to the user
            Log.i("blax", toRead);
            If (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
                        ttsGreater21(toRead);
             } else {
                        ttsUnder20(toRead);
            }
}


//The text-to-speech function to be used in case the target android version
//is equal to or lower than API-20
 @SuppressWarnings("deprecation")
 private static void ttsUnder20(String text) {
            HashMap<String, String> map = new HashMap<>();
            map.put(TextToSpeech.Engine.KEY_PARAM_UTTERANCE_ID, "MessageId");
            if (!tts.isSpeaking())
                        tts.speak(text, TextToSpeech.QUEUE_FLUSH, map);
            else tts.speak(text, TextToSpeech.QUEUE_ADD, map);
}

  //The text-to-speech function to be used in case the target android version
  //is greater than API-20
  @TargetApi(Build.VERSION_CODES.LOLLIPOP)
  private static void ttsGreater21(String text) {
             String utteranceId= code + "";
            if (!tts.isSpeaking())
                        tts.speak(text, TextToSpeech.QUEUE_FLUSH, null, utteranceId);
            else tts.speak(text, TextToSpeech.QUEUE_ADD, null, utteranceId);
  }
```

OcrDetectorProcessor

T

```
 private GraphicOverlay<OcrGraphic> mGraphicOverlay;
 //The list that contains all the readings
private ArrayList<ArrayList<TextBlock>> reading;


OcrDetectorProcessor(GraphicOverlay<OcrGraphic> ocrGraphicOverlay) {
        mGraphicOverlay = ocrGraphicOverlay;
        //Initialization of the list of readings
        reading = new ArrayList<ArrayList<TextBlock>>();
}

 //returns the list of readings to the class that requests it
public ArrayList<ArrayList<TextBlock>> getReading(){
        return reading;
}

 //empties the list of reading when a class commands so
public void clearReading(){
        reading.clear();
}

/**
* Called by the detector to deliver detection results.
* If your application called for it, this could be a place to check for
* equivalent detections by tracking TextBlocks that are similar in location and content from
* previous frames, or reduce noise by eliminating TextBlocks that have not persisted through
* multiple detections.
*/
@Override
public void receiveDetections(Detector.Detections<TextBlock> detections) {
         ……………………………….
        //sort our new array based on the following criteria...
        Collections.sort(items2, new Comparator<TextBlock>() {

                public int compare(TextBlock block1, TextBlock block2) {
                        …………………………..
                }
         );

        //Add this reading to the list of readings
         reading.add(items2);
         //Write the blocks of the now sorted array in the log
         for (int i = 0; i < items2.size(); ++i) {
                 …………………………..
        }
}
```

➔➔➔➔➔➔

OcrDetectorProcessor

```
……………………………………………..
 private int check = 0;
…………………………………………………..
//initialize collection of readings
public void setCheck(){
        check = 1;
        clearReading();
}

/**
* Calculates the similarity (a number within 0 and 1) between two strings.
 */
public static double similarWord(String s1, String s2){
        String longer = s1, shorter = s2;
         if (s1.length() < s2.length()) { // longer should always have greater length
                longer = s2; shorter = s1;
        }
         int longerLength = longer.length();
        if (longerLength == 0)
                { return 0; /* both strings are zero length */ }
                /* // If you have StringUtils, you can use it to calculate the edit distance:
                return (longerLength - StringUtils.getLevenshteinDistance(longer, shorter)) /
                (double) longerLength; */
                return (longerLength - editDistance(longer, shorter)) / (double) longerLength;
  }


  // Example implementation of the Levenshtein Edit Distance
 // See http://rosettacode.org/wiki/Levenshtein_distance#Java
  public static int editDistance(String s1, String s2) {
        s1 = s1.toLowerCase();
         s2 = s2.toLowerCase();

        int[] costs = new int[s2.length() + 1];
         for (int i = 0; i <= s1.length(); i++) {
                 int lastValue = i;
                for (int j = 0; j <= s2.length(); j++) {
                        if (i == 0)
                                costs[j] = j;
                        else {
                                 if (j > 0) {
                                        int newValue = costs[j - 1];
                                         if (s1.charAt(i - 1) != s2.charAt(j - 1))
                                                 newValue = Math.min(Math.min(newValue,
                                                lastValue),
                                        costs[j]) + 1;
                                        costs[j - 1] = lastValue;
                                        lastValue = newValue;
                                }
                        }
                 }
                if (i > 0)
                        costs[s2.length()] = lastValue;
         }
        return costs[s2.length()];
  }

  /**
   * Called by the detector to deliver detection results.
   * If your application called for it, this could be a place to check for
 * equivalent detections by tracking TextBlocks that are similar in location and content from
 * previous frames, or reduce noise by eliminating TextBlocks that have not persisted through
 * multiple detections.
 */
@Override
public void receiveDetections(Detector.Detections<TextBlock> detections) {
         mGraphicOverlay.clear();
        //if collection of readings was initialized
        if (check == 1) {
                    //if 6 readings have been collected
                   if (reading.size() >= 6) {
                              //stop collecting readings
                             check = 0;
                             //Get the whole list of readings
                             ArrayList<ArrayList<TextBlock>> whole = getReading();
                             //sort readings from the longest to the shortest
                             Collections.sort(whole, new Comparator<ArrayList<TextBlock>>() {

                             public int compare(ArrayList<TextBlock> reading1, ArrayList<TextBlock>
                             reading2) {

                                       int size1 = 0, size2 = 0;
                                       for (int i = 0; i < reading1.size(); i++) {
                                                  size1 += reading1.get(i).getValue().length();
                                       }
                                        for (int i = 0; i < reading2.size(); i++) {
                                                  size2 += reading2.get(i).getValue().length();
                                        }

                                        if (size1 == size2) return 0;
                                        if (size1 > size2) return -1;
                                       return 1;
                              }
                             });

                   //leave only six readings in the list
                    while (whole.size() > 6) {
                             whole.remove(whole.size() - 1);
                    }

                   //the 6 strings = 6 readings
                   String[] candidates = new String[whole.size()];

                   //fill in the 6 strings
                   for (int i = 0; i < whole.size(); i++) {
                             candidates[i] = " ";
                             for (int j = 0; j < whole.get(i).size(); j++) {
                                       candidates[i] = candidates[i] + "\n" +
                                       whole.get(i).get(j).getValue();
                             }
                              Log.i("blax", i + candidates[i]);
                   }

                    double max_sim = Double.MIN_VALUE;
        String toRead = " ";

        //find the 2 most common readings out of the 6
        for (int i = 0; i < whole.size(); i++) {
            for (int j = i + 1; j < whole.size(); j++) {
                double similarity = similarWord(candidates[i], candidates[j]);
                 Log.i("blax", i + " " + j + " " + similarity);
                If (similarity >= max_sim) {
                    max_sim = similarity;
                     toRead = candidates[i];
                 }
             }
        }
         //continue the process in the "OcrCaptureActivity" class
        OcrCaptureActivity.continueAction(toRead);
    }
    ……………………………………………………

}
```

Section Y

OcrGraphic.java

```
......................................
/* Return textbox's area */
   public int TextBlockSize(){
              TextBlock text = mText;
             if (text == null) {
                          return 0;
               }
              RectF rect = new RectF(text.getBoundingBox());
              rect.left = translateX(rect.left);
             rect.top = translateY(rect.top);
             rect.right = translateX(rect.right);
              rect.bottom = translateY(rect.bottom);
              return (int)((rect.right - rect.left) * (rect.bottom - rect.top));
 }
......................................
```

OcrDetectorProcessor.java

```
public void receiveDetections(Detector.Detections<TextBlock> detections) {
  ..................................................
   int reading_area_size = 0; //the area of the camera each reading covers
   //the max area of the camera
   final int max_area_size = OcrCaptureActivity.width * OcrCaptureActivity.height;
   for (int i = 0; i < items.size(); ++i) {
                TextBlock item = items.valueAt(i);
                 if (item != null && item.getValue() != null){
                                //move block to our new array
                                items2.add(item);
                }
                //create a graphic white box for the TextBlock "item" and put it in the Overlay
                 OcrGraphic graphic = new OcrGraphic(mGraphicOverlay, item);
                reading_area_size += graphic.TextBlockSize(); //update the covered area with each box
                mGraphicOverlay.add(graphic);
   }
   //if beeping is allowed to happen at this moment
   if (OcrCaptureActivity.beep && check == 0 && !OcrCaptureActivity.isTtsSpeaking()) {
                //Generate louder and less frequent tones the more area the text detector
                //reads, indicating you are closer to the correct positioning from the page
                if (reading_area_size > max_area_size / 2) {
                                  toneG.startTone(ToneGenerator.TONE_CDMA_ALERT_INCALL_LITE, 400);
                } else if (reading_area_size > max_area_size / 3) {
                                 toneG.startTone(ToneGenerator.TONE_CDMA_ALERT_INCALL_LITE, 300);
                 } else if (reading_area_size > max_area_size / 4) {
                                 toneG.startTone(ToneGenerator.TONE_CDMA_ALERT_INCALL_LITE, 200);
                } else if (reading_area_size > max_area_size / 5) {
                                 toneG.startTone(ToneGenerator.TONE_CDMA_ALERT_INCALL_LITE, 100);
                } else toneG.startTone(ToneGenerator.TONE_CDMA_ALERT_INCALL_LITE, 50);
   }
```

OcrCaptureActivity.java

```
public void onBackPressed(){
      if (par_to_speak != null){
         switch (par_to_speak){
               case "1": if (paragraphs.size() >= 1) ttsUnder20(paragraphs.get(0)); break;
```

```
        case "2": if (paragraphs.size() >= 2) ttsUnder20(paragraphs.get(1)); break;
        case "3": if (paragraphs.size() >= 3) ttsUnder20(paragraphs.get(2)); break;
        case "4": if (paragraphs.size() >= 4) ttsUnder20(paragraphs.get(3)); break;
        case "5": if (paragraphs.size() >= 5) ttsUnder20(paragraphs.get(4)); break;
        case "6": if (paragraphs.size() >= 6) ttsUnder20(paragraphs.get(5)); break;
        case "7": if (paragraphs.size() >= 7) ttsUnder20(paragraphs.get(6)); break;
        case "8": if (paragraphs.size() >= 8) ttsUnder20(paragraphs.get(7)); break;
        //the number 9 is also used to toggle the beep on an off
        case "9": if (paragraphs.size() >= 9) ttsUnder20(paragraphs.get(8));
              beep = !beep; break;
        default: break;
    }
  }
  par_to_speak = null;
}
```