Universidad Carlos III de Madrid

Ingeniería de Sistemas Audiovisuales

PROYECTO FIN DE CARRERA

Desarrollo de una aplicación móvil mediante la integración de realidad aumentada y comunicación oral

Autor: Edel Alejandro Pérez Cuellar

Tutor: David Griol Barres

Leganés, Septiembre 2016

Título: Desarrollo de una aplicación móvil mediante la integración de realidad aumentada y comunicación oral

Autor: Edel Alejandro Pérez Cuellar

Director: David Griol Barres

EL TRIBUNAL

Presidente:________________________________

Vocal: ________________________________

Secretario: ________________________________

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _______ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

En primer lugar me gustaría mostrar mis agradecimientos a mi tutor David Griol por su apoyo y optimismo durante el desarrollo de este proyecto.

A mi maravillosa novia, que siempre ha estado para animarme cuando lo necesitaba.

Y a mi familia, mi padre, mi madre y mi hermana por siempre ofrecerme su apoyo.

Abstract

This Bachelor project aims to create an augmented reality application which uses computer vision algorithms and technologies to provide assistance to the visually impaired in their daily tasks. The specific purpose of the application is to recognize scenes and objects after capturing them using the device's native camera. Colour recognition is also present to aid in identifying objects and their properties such as their size and distance from the capturing device, such functionality may also help individuals which suffer from colour blindness. The application also interacts with the user by speech, effectively establishing a Speech Dialogue System (SDS).

The general planning of this project was done considering the gained knowledge from project development courses done within the University of Carlos III, Madrid. In such, the required research and development was divided in a set number of stages, each with an independent number of tasks. This methodology has allowed us to define three major stages: Planning, Execution and Closing.

A Work Breakdown Structure (WBS) has been used to make the structuring of the project's planning easier.

For the development of the application, JAVA programming language was used within the Android Studio development environment. OpenCV libraries were imported and used to implement the functionalities of the developed system.
The final version of the application has thus used computer vision as a tool to provide additional information over real world scenes both with on screen representation and audible messages.

As future work it would be interesting to reduce the computational load of computer vision algorithms. It would perhaps be convenient to further improve the OpenCV library and its recent API on Android.

Keywords: Android, Augmented Reality, Computer Vision, OpenCV,
Speech Dialogue Systems

General Index

Figure Index

Chapter 1

INTRODUCTION

This chapter will introduce the present bachelor thesis and the issue under study. The general structure and study of this project will be described and how it contributes to solve the problems and circumstances found. A contrast will be made in how the Android application developed in this study contributes to what is already present.

Furthermore, the context of the current study will be described explaining the issues present regarding the physically disabled and vision impaired. The regarded objectives of this study will be listed and the budget required to accomplish them. This will further require an accurate planning which will be laid out in this chapter in a Work Breakdown Structure (WBS) diagram.

1.1 Introduction

The advance of technology aims to make tasks easier in an open field of applications, such as quality of life improvements, increased efficiency in industry and retail, military advancements and so on. The level of research and accelerated technological development have allowed complex algorithms which were historically inaccessible for computer systems of the era to now be endured by powerful processors. The cost of technological research have also decreased greatly in recent years, motivating further investigation by large companies and governmental organisations. The medical field has taken a giant leap in technological progress, now greatly relying on devices and complex machines to increase the efficiency and accuracy of diagnostics and treatment methodology.

Great progress has also been made in retail, with the product of technological research reaching the general public and sparking a great deal of interest. This has been translated to an increase in revenue as devices and internet technologies have intensively begun to participate in world markets. The increased presence of portable technology has also emphasized on the public’s interest. Such technology makes it possible for the general user to carry the hardware and software piece independently of the location. This has majorly come in the form of smart devices as smartphones, tablets, portable computers and watches.

The research in computer vision has had great results, as initially a product of military and university academic development it is now present to the public for private use or as general services provided by companies and governments. As an example, there is now a great efficiency in surveillance systems, authentication algorithms, and traffic security. The field has also greatly contributed to artificial intelligence research allowing now the introduction of autonomous robotic devices and vehicles with relevant use in medical fields, space travel and public transport.

The increased interest of the general public in technologies have also motivated investment in research of augmented reality systems that are able to add additional layers of information upon the already existing physical world scenes. This has also had great success in entertainment, providing new ways of enjoying video games and movies. Digital characters that were before confined to the computer screen now seem to travel to the three dimensional physical world, presenting themselves before the user.
As a consequence of these technological developments it is only natural that the physically or mentally disabled gain access to these systems and greatly decrease their difficulties in general everyday tasks. Those individuals which suffer from vision impairment take great benefit by using computer vision systems which may aid them in identifying objects, people and areas.

With the introduction of Android and iOS operating systems on smartphones and tablets, these advancements make the development of aid and assistance applications easier and quicker. This greatly decreases the costs of the end product and in such for the disabled or visually impaired users there is now great level of accessibility to these applications.

Google has made a great effort in increasing the accessibility on Android devices. Most devices now come with speech recognition systems which allow the visually impaired who may not be able to identify what is being presented on screen to communicate with the applications by speech. Furthermore, text-to-speech systems now also naturally come with every Android device, allowing the applications to easily communicate with the user by send relevant voice messages.

1.2 Objectives

With the research, study done in this project and the application's development it is intended that an effective system is provided for the visually impaired to assist in everyday tasks. A computer vision library and basic augmented reality is used to not only present a usable application for the visually impaired but also to those individuals who would provide personal aid. Strictly speaking, this application requires at some point the intervention of a non-visually impaired individual to configure the detection process. At key moments and situations this application should be able to provide the same feedback a visually healthy individual would obtain. In general situations, however, it is difficult to design a system which would completely replicate a healthy human visual sense. It is nevertheless intended that this application somehow increases the quality of life of visually impaired individuals and their assistants.

In such, the Android application developed will prove the users with the following benefits:

- A speech-based interactive application where it isn't required for the user to be able to see what is represented on screen. The application will use text-to-speech systems to tell the user what may be accessed and use speech recognition to understand the user's utterance. The application will however provide visible interfaces on screen so that not visually-impaired users may choose to interact with the systems through standard touches.

- Well defined and independent modules which adapt to a great variety of situations provided by the nature of computer vision systems development.

- A module dedicated to capturing and storing still images of objects saved by identification number and name.

- A module dedicated to recognize objects using the prestigious open-source computer vision OpenCV library. The module will use the database of previous stored objects or scenes and identifiers to periodically scan for objects which may appear on the scene captured by the live camera feed.

- Detection of colours also using the OpenCV library. The user may say what colour is to be detected and the system will scan the pixels for the colour in a range of HSV mapped values. The pixels detected in the HSV range will be set to white and the pixels not corresponding to the HSV range will be, in change, set to black. This might have a low-vision user distinguish an object due to the sudden high level of contrast present on the screen. Additionally, the module is to tell the user by speech the percentage of pixels present in that scene, giving the user a sense of distance and size of the object.

- Basic visual and audible augmented reality to efficiently provide information towards the user.

1.3 Development Stages

The general planning of this project was done considering the gained knowledge from project development courses done within the University of Carlos III, Madrid. In such, the required research and development was divided in a set number of stages, each with an independent number of tasks. This methodology has allowed us to define three major stages: Planning, Execution and Closing.

A Work Breakdown Structure (WBS) has been used to make the structuring of the project's planning easier. Each stage consists of a set of tasks to be completed before the execution of the next stage begins.

In the next figure the project's corresponding WBS can be observed. At the top of the figure, the three major stages can be observed and continuing towards the bottom their separate tasks.

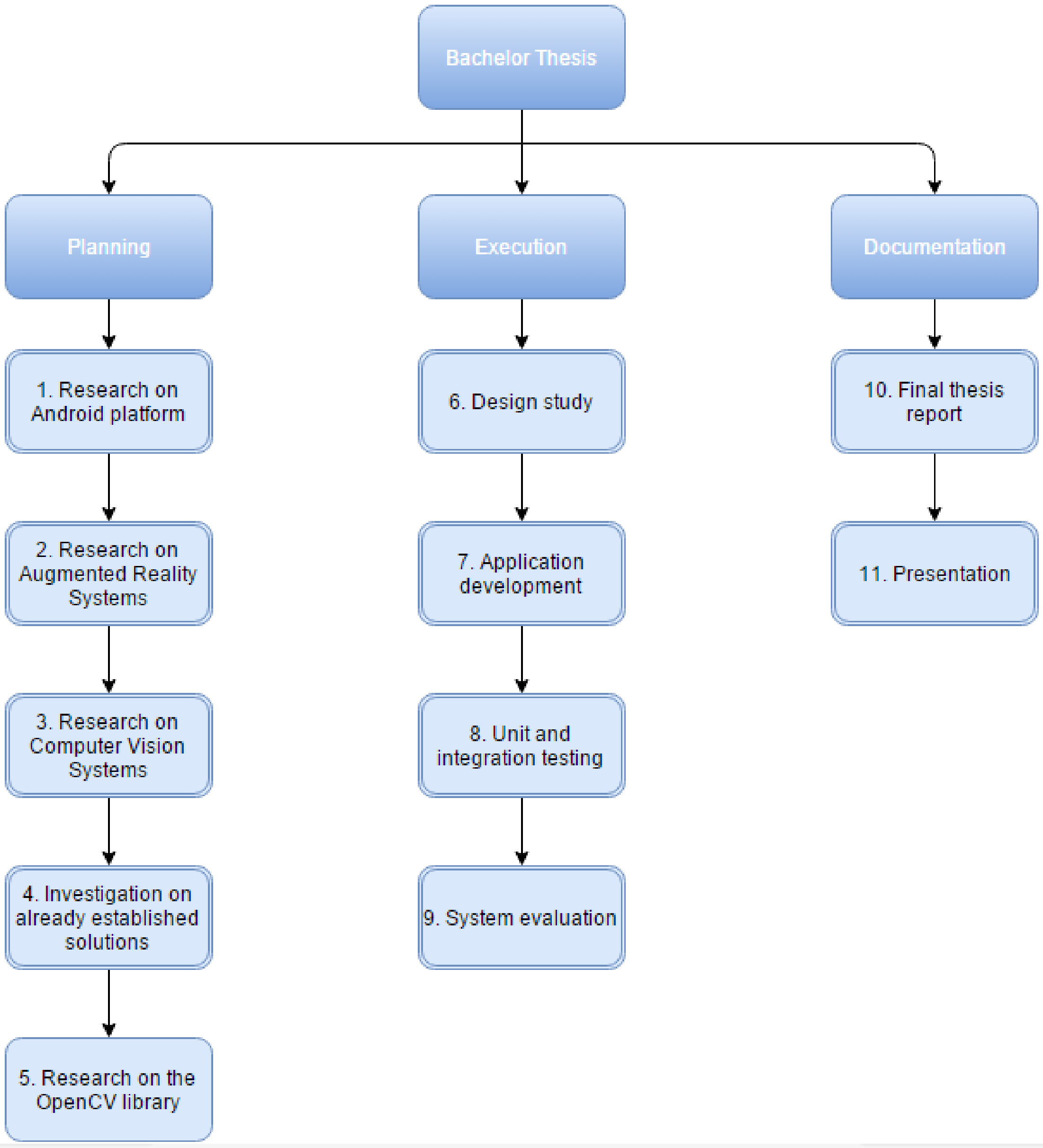

Bachelor Thesis
Planning
Execution
Documentation
1. Research on Android platform
2. Research on Augmented Reality Systems
3. Research on Computer Vision Systems
4. Investigation on already established solutions
5. Research on the OpenCV library
6. Design study
7. Application development
8. Unit and integration testing
9. System evaluation
10. Final thesis report
11. Presentation

Figure 1.1: WBS defined for the project

On the first stage of the project, the following tasks are defined:

1. **Research on Android the platform:** A study was conducted on the Android OS and the development of applications for Android devices.
2. **Research on Augmented Reality Systems:** A study was conducted on augmented reality solutions currently present on all platforms. Then, Android systems which implement augmented reality applications were studied in detail.
3. **Research on computer vision systems:** A study was conducted on computer vision and its applications.
4. **Investigation on already established solutions:** The present solutions were studied for all platforms and in detail for Android devices. A study was made on the viability of an Android application to help the visually impaired.
5. **Research on the OpenCV library:** A study was made on OpenCV and what it offers for Android development.

The next stage of the project, execution, consisted of the following tasks:

6. **Design study:** A detailed planning of the system's design was made, studying the functionalities of each module.
7. **Application development:** The code for the application was written using the Android Studio development environment.
8. **Unit and integration testing:** The code was tested for each functionality and the application was installed and executed in several Android devices.
9. **System evaluation:** The complete functionality was evaluated in different types of scenarios.

The final documentation stage consisted of the following tasks:

10. **Final thesis report:** The final report was written, detailing every aspect of the project's development.
11. **Presentation:** The project was presented.

1.4 Resources used

The following resources were used for the project’s research and development:

Hardware resources:

- Desktop computer
- Samsung Galaxy Tab 4
- Sony Xperia S Smartphone
- USB cable

Software resources:

- Android Studio development environment for Android applications
- Android SDK (Software Development Kit).
- Java JDK (Java Development Kit)
- OpenCV Android SDK: A development API for Android applications
- Google TTS synthesizer for Android devices
- Google speech recognition on Android devices
- Google Drive cloud storage
- Draw.io online software tool for the composition of flow charts.

1.5 Structure of the report

Chapter 1: Introduction. The project’s purpose is described in detail, the issues it tries to solve, the planning and the structure.

Chapter 2: State of the Art. A detailed study is made on the environment surrounding the project’s study. Augmented reality and computer vision systems are studied, as the OpenCV library is explained in detail.

Chapter 3: System description: A detailed description is made on the developed system in each of its functionalities and modules

Chapter 4: System evaluation. A methodology is followed to evaluate the application in several scenarios and for several users. The evaluation model is defined in detail and the extracted results.

Chapter 5: Final conclusions and future study. In this chapter the final conclusions are exposed as well as considerations for future studies in the present field.

Chapter 6: Project management. The temporal planning is shown in detail and the general costs are exposed.

Definitions. In this section the most important terms for this project's study are defined.

References. In this section the sources of information which were consulted for the project's study will be listed.

Chapter 2

STATE OF THE ART

2.1 Augmented Reality Systems

2.1.1. Introduction to Augmented Reality Systems

The goal of augmented reality systems is to present virtually registered or generated information directly into the physical environment. AR systems seek to shorten the gap between the real physical word and the virtual world, spatially and cognitively. Augmented reality allows to add digital information as an intrinsic part of the real world (Hölleler & Schmalstieg, 2016). This principle presents differences with virtual reality (VR) systems, where the goal is to include the user in a completely virtual environment as a substitute to a real world experience.

Many areas of computer science contribute to the development of complex AR systems, yet the definition of augmented reality as a concept is at many times not clear. AR systems must combine several characteristics. First of all it is essential that the system combines real world information and virtual information in a single scene. Furthermore the system must provide real time interactivity with the virtual information. Depending on the application, the system must register the information in 3D (Azuma, 1997). The latter is flexible, however, as text or image information may also be represented on a real world scene, or interactive voice communication for registering real time data (as in the case of this project) is also considered an AR system. Augmented reality then presents additional information added to a real world scene in real-time so as to allow the brain to process real and virtual information instantaneously through the senses (Craig, 2013).

According to Hölleler & Schamstieg (2016), to achieve its goals, it is essential for an AR system to contain three main components: a tracking component (with the goal of capturing real world information in real-time), a registration component (so as to adapt the input information to the virtual model managed by the system), a visualization component (for the representation of real and virtual data on the same scene) and a spatial model (a database to store information of the real world and the virtual world separately and the relations made between them). In the figure below we can observe a general schema of this concept.

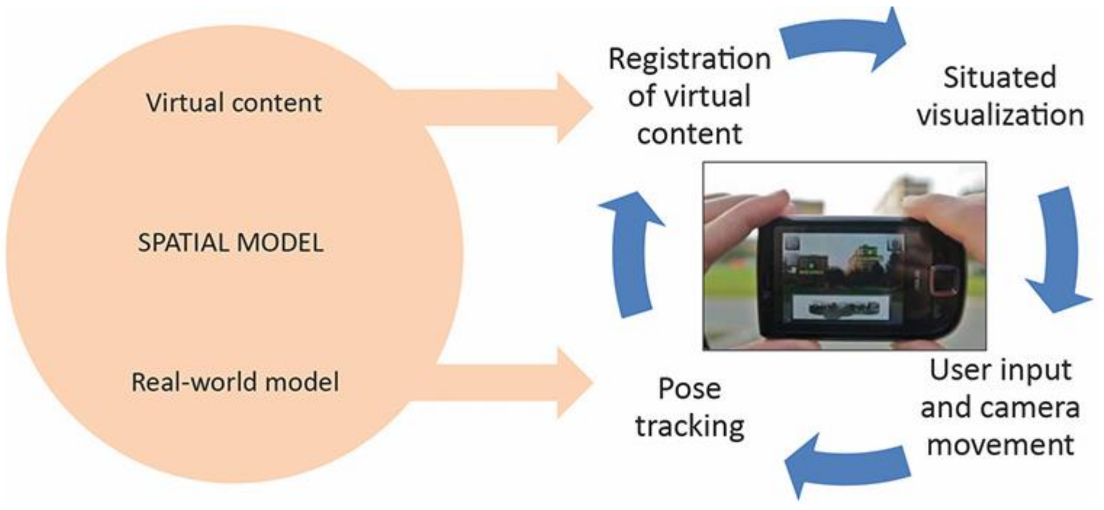


Figure 2.1: Components of an AR system

With the concept of augmented reality, it is essential to consider the idea of spatial registration where the information has a physical space or location in the real world in the same way the physical counterpart to the digital information would have (Craig, 2013). In other words, a virtual object must be placed and positioned by the AR system in the same way a physical object is placed, abiding physical laws.

2.1.2. History of augmented reality

It could be said that there has been a long history of events in which additional information has been overlaid over the real physical world, however the biggest advances in computer-generated augmented reality can be traced to the 1960s (Hölleler & Schmalstieg, 2016). Ivan Sutherland was one of the first researchers in setting the scene in which both virtual reality and augmented reality systems would later be developed with the introduction of the *ultimate display*. According to Sutherland (1965), the ultimate display would be a room which can control the existence of matter. Objects generated in such an area would have the same physical properties and their existence would cause the same consequences and events upon the world as their naturally generated counterparts. Sutherland would later proceed to construct the first VR system in 1968, a heavy head-mounted display.

Other works have continued the research into augmented reality systems. In 1974, Myron Krueger built an “artificial reality” laboratory by the name of “The Videoplace”. It consisted of combined projectors with video cameras that emitted onscreen silhouettes, surrounding users in an interactive environment.

The military domain would quickly take notice of the advantages in augmented reality systems, researching, developing and researching their own. In 1992, Louis Rosenberg developed the named “Virtual Fixtures” AR system for the United States Airforce. It allowed for the military to control a full upper-body exoskeleton which allowed them to guide machinery and perform tasks from a remote operating space.

The concept of augmented reality would later make its appearance in the entertainment scene, with the arrival of the “1st and Ten” line computer system in 1998, casting a virtual yellow marker on an NFL game. Research on AR system would later drastically increase, with only a year later the military working on the Battlefield Augmented Reality System (BARS) devices for soldier training. Later that same year NASA would use augmented reality terrain and navigation display to fly the first X-38 spacecraft.

In the year 2000, Hirokazu Kato released the open-source software library ARToolkit, still widely used today. The library uses video tracking to overlay computer graphics on video cameras ARToolkit would later support augmented reality on web browsers in 2009. Print media would also take advantage of AR systems in 2009, with Esquire Magazine prompting readers to scan the example cover to make Robert Downey Jr. come alive on stage.

In recent years, car manufacturers have integrated augmented reality for vehicle service manuals. In 2013, the Volkswagen MARTA app was released which provided a virtual guide with step-by-step instructions on vehicle service. The system allowed technicians to foresee how a repair process would look on the present vehicle.

Google was quick to step onto the scene with the announcement of the Google Glass device in 2014, starting a trend of wearable augmented reality devices. Investments on AR systems increased dramatically in the following years. Going from fifty million dollars to one billion in only two years from 2014 to 2016.

2.1.3. Applications of augmented reality

Augmented reality research presents itself as a significant promise in decreasing the challenges in cyber and physical system visualization and interaction for multiple domains such as medicine, construction, advertising, manufacturing and gaming (White, 2014). The concept however is increasingly challenging with research to overcome these challenges based on precise localization, information fusion and complex information visualization among others. Many AR system applications have appeared throughout history and the trend has increased in recent years.

The industrial field has risen up as an important part of AR practice and research. AR devices have historically proven to be expensive and of non-efficient use in industry. However the appearance of smartphones has proven to be an inflection point decreasing the costs and difficulty to access augmented reality mediums. Many current-generation mobile devices, smartphones and tables contain a great variety of sophisticated sensors, powerful processing and storage systems and persistent network connections (White, 2014). Recently appearing head-mounted devices such as Google Glass are also gaining relevance in the industrial and medical scheme.

AR driving experiences have also become greatly relevant. Challenges and risks are present however, greatly increasing the possibility of distractions and endangering safety. New safety systems for automobiles are working to ensure the safety of the occupants such as automatic steering and breaking systems. This allows the AR research on this field to have more freedom. Companies such as BMW have begun research on AR windshield displays, allowing additional information to be presented for the driver such as road hazards, weather reports and GPS information.

The medical field has found great use in augmented reality systems for anatomic study, with mobile devices being able to track the position of objects or markers to render muscle and bone structure in 3D which would be located in the specific location.

The recent trend for the use of commercial and non-commercial unmanned aerial vehicles (UAVs) has allowed for many potential AR applications by increasing the user’s perception of the real world. By the use of UAVs information can be gathered for use in augmented reality systems such as terrain topological data and construction 3-D modelling.

The tourism industry takes advantage of AR systems with the use of smartphone and tablet apps which are able to locate and describe relevant touristic or historical areas, restaurants, hotels and services. By using both VR and AR system advances, touristic companies are able to offer complete alternate reality experiences in leading touristic sites by overlaying 3-D information.

2.2. Computer Vision

2.2.1 Introduction to computer vision

Humans and many animal species are able to perceive the three-dimensional world through a complex visual system which begins at the eyes and culminates in the brain where the visual nerve impulse signal travelling through the optical nerve is interpreted and thus we experience what we call vision. This happens with apparent ease, yet many complex processes occur, much of which are still under intense study as we struggle to understand how interconnected neurons are able to create vision. In such, the brain is able to not only perceive real world objects but also make out detail within them, such as colour, shape, translucency, subtle patterns. In the case of other humans we are quickly able to determine their emotions from their facial appearance (Szeliski, 2010).

Research in computer vision is done with the goal to simulate or replicate this phenomenon with the inclusion of complex algorithms and mathematical techniques to study three-dimensional objects and their properties within a scene. Many breakthroughs have been made, devising systems which are able to create 3-D model representations of real world objects through spatial photography, track people that move along complex scenery or backgrounds and even recognise people within a crowd studying their facial features and clothing. However, it is still not possible to devise a system that works with the clarity of the human visual system. In great part, it is because developing vision systems requires us to tackle an inverse problem where the goal is to recover unknown data given insufficient information to come to a solution (Szeliski, 2010).

Due to this lack of information, we find ourselves forced to use physics-based probabilistic models to come to potential solutions to the problem. This is called modelling and it also has a limit. Modelling the complexity of the world requires great processing power.

Computer vision has a great variety of applications but the approach for solving this problem usually requires several separate tasks:

- **Recognition:** Tackles the classical problem where the system has to determine if the object is present in the scene, or if a certain feature is apparent. The best algorithms for this task handle convolutional neural networks where an approach is made to the animal visual cortex.
- **Motion analysis:** An estimation is made regarding the motion or velocity of the object in a particular scene or even of the camera device itself (egomotion). An effort is made then to follow the object's trajectory throughout the scene.
- **Scene reconstruction:** An attempt is made to reconstruct the scene into a 3-D model representation in a digital coordinate system.
- **Image restoration:** Several filters are applied to improve the quality of the image for its processing. Gaussian or median filters are relevant for noise removal.

2.2.2. Applications of computer vision

Computer vision is being used in the modern world in several research and application areas such as:

- **Optical character recognition (OCR):** Systems which are able to identify written letters and produce a meaning.
- **Machine inspection:** By the use of stereo vision (algorithm in which multiple overlapping images are taken to produce a detailed 3D version of the object) a detailed study is made on the state of a machine's parts or shapes.
- **Retail:** Presence recognition to control customer check-in and check-out of retail areas.
- Medical imaging: Dermatological study of skin conditions, brain morphology with age or pre-operative and post-operative studies.
- **Automotive safety:** Detection of obstacles to ensure driver and pedestrian's safety in crowded areas or dangerous roads.

- **Surveillance:** Detection of intruders, highway traffic counting and speed estimation.
- **Security:** Fingerprint, retina and facial recognition to control user access systems (authentication).

2.3. Spoken Dialogue Systems

2.3.1. Introduction to spoken dialogue systems

Spoken Dialogue Systems allow interaction with a certain application through speech and additional input and output modalities. Such interaction is made possible through Speech Recognition processes and Text-to-Speech algorithms. Thus, it is essential for any application which makes use of the features of the Spoken Dialogue System to contain the former and latter functionalities. These systems are useful when we wish to simulate human-to-human interaction when visibility of a UI is not readily available. Academic researchers on dialogue systems often have the goal of exploring how systems may allow more spontaneous language use (Skantze, 2007).

The process, however, may face risks in its correct functioning due to certain variations in the input process. Language based differences and pronunciation can produce a variability which may hinder the complete Spoken Dialogue System. Errors may also be induced from unpredictable ambient noise during the capture process. To face this dilemma, the algorithm must be well equipped to face these variations increasing the system's robustness.

2.3.2. Architecture of a spoken dialogue system

Spoken Dialogue Systems present complex processes and algorithms hence their operation may be divided in different sub-processes and technologies. The entire process may be represented in a pipeline approach with a system that takes user utterance as input and delivers system utterance as output (Skantze, 2007).

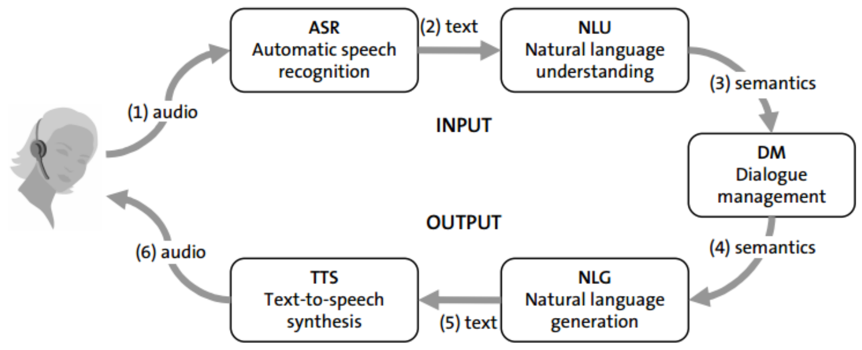


Figure 2.2: Pipeline approach for processes of a Spoken Dialogue System

From the figure above we can observe that the module that will directly process the user's speech will be the Automatic Speech Recognition module (ASR). From this input, the module will create a text based hypothesis which will serve as input for the next module, the Natural language Understanding module (NLU). The NLU module will not consider the dialogue context, it will attempt to resolve the input text utterance and produce a semantic based response. That is, it will attempt to find the meaning of the input hypothesis and determine if it is useful for the implementation. Such response will be managed by the Dialog Manager (DM) which will establish operations based on the input, accessing system resources of databases or registering the user's instructions. It will then decide which action is to be taken or the response the system must produce.

The response will be naturally an output dialog, thus a semantic-based response must be given to be handled by the Natural Language Generation module (NLG). The NLG module will use the input semantic to produce a text based response which will be used by the Text-to-Speech module to produce an acoustic signal, the output or response of the entire system in the form of speech.

2.3.2.1. Automatic Speech Recognition module

The Automatic Speech Recognition module (ASR) as a feature of computational linguistics will be able to transform speech into text in readable format for the application. In other words, the module with attempt to recognize the utterance of the user and serve as output a sequence of recognized words by the use of sound signal treatment techniques. Acoustic modelling and language modelling are important concepts to make the system's recognition process more effective. Thus, the system usually consists of several stages, an acoustic signal modelling stage where the signal will be treated to remove noise and interference and a language modelling stage where the system will attempt to relate sounds with language semantic sequences.

Many modern speech recognition systems use Hidden Markov Models. That is, statistical models which serve a series of symbols or quantities in its output phase. HMMs take advantage of the property of an acoustic speech signal which can be considered stationary during a certain amount of time (around milliseconds). In other words, in a small amount of time speech can be approximated as a stationary stochastic process.

Other speech recognition systems may make use of dynamic time warping. An algorithm that measures the similarity between two sequences which may be different in time or speed. This is especially useful when we wish to consider different speaking speeds for a ground of individuals.

2.3.2.2 Natural Language Understanding module

The Natural Language Understanding module (NLU) will attempt to obtain a sematic representation of the recognized sequences obtained from the speech recognition module. In other words, the system will perform what can be considered a conversion from natural language to a semantic language. However, still maintaining the original meaning of the transmitted message. The NLU module will thus need to perform a language based analysis dividing individual words into tokens or lexemes (parts of the word which are invariable and contain the semantic meaning) and morphemes (additional parts of the words which will alter its original meaning). The system will further analyse the complete phrases attempting to deduce their meaning from that of the individual words obtained previously. As a final step, the system may attempt to alter the deduced meaning by analysing the context in which each individual phrase is detected.

2.3.2.3. Dialogue Management module

The Dialogue Management module may be considered the ‘core’ of the complete speech dialogue system. It will handle the input commands from the user by interpreting the semantic meaning of the identified message then start processes or obtain information in the application accordingly.

Furthermore, the Dialogue Management module must update the dialogue context and handle the context according to the obtained interpretation. All participant modules with the dialogue system must be handled and coordinated according to each of their individual function in the general scheme.

2.3.2.4. Natural Language Generation module

The Natural Language Generation module will receive the semantic representation of the response of the system and will further perform the conversion into natural language phrases which can be easily understood by the user. This is essential if we wish to simulate human-to-human interaction and make our system seem less "robotic".

2.3.2.5. Text to Speech synthesis module

The Text to Speech synthesis module (TTS) has the purpose of taking the response of the system in natural language and transform it into an acoustic speech signal understandable by the user. TTS modules usually consist of back end processes (conversion of words and symbols into spoken phrases) and front end processes (acoustic signal generation with human characteristics).

2.4. Voice recognition in Android

Voice recognition in Android can be managed through a special type of intent called *RecognizerIntent*. An intent in Android is defined as a mechanism which allows users to coordinate functions and activities to achieve a certain task. It can be seen as the action of flipping a switch, actions are taken for an event occurring after the switch change state. By handling different actions taken by the *RecognizerIntent* it will call different activities which will perform specific actions. Commonly used actions are listed as the following:

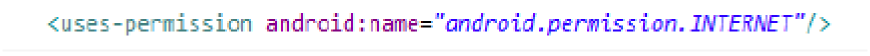
- ACTION_GET_LANGUAGE_DETAILS: Send a broadcast intent that will take the broadcast metadata of an activity under ACTION_WEB_SEARCH.

- ACTION_RECOGNIZE_SPEECH: Begin an activity that will listen for user speech upon a trigger event or gesture and send it through a speech recognizer.

- **ACTION_VOICE_SEARCH_HANDS_FREE:** Begin an activity that will listen for user speech but without the need for a trigger event or gesture. It will then send it through a speech recognizer to initiate a web search or trigger another action.

- **ACTION_WEB_SEARCH:** Begin an activity that will listen for user speech upon a trigger event or gesture then send it through a speech recognizer for use in a web search or to trigger another action.

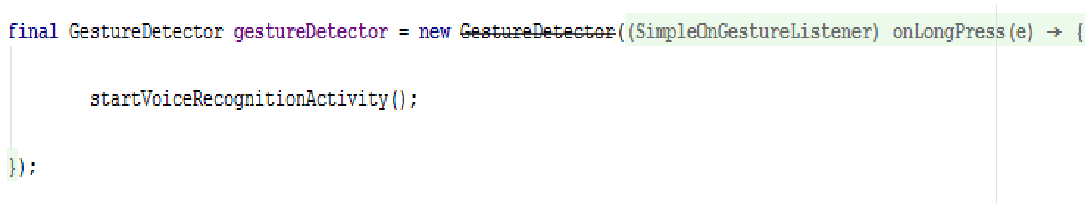
Speech recognition will be handled by Google which will make use of a server database to identify the input message. Thus, internet access will be required, either from mobile data or Wi-Fi-access. Thus, the following line of code in the figure below must be added to the android manifest in order to ask permission from the application to make use of the internet service installed on the phone.



```
<uses-permission android:name="android.permission.INTERNET"/>
```

Figure 2.3: Permission settings in Android Studio:

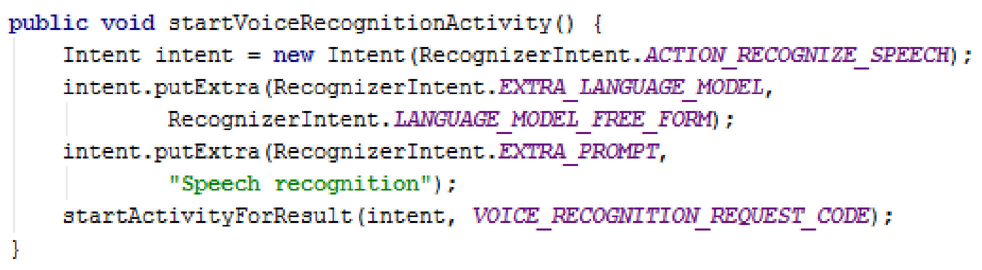
Additionally, the trigger event for the voice recognition system must be handled in Android as a gesture on screen. A tap-on-screen, hold-on-screen or swipe-on-screen event is sufficient. In the following figure, we can observe the aforementioned requirements:



```
final GestureDetector gestureDetector = new ~~GestureDetector~~((SimpleOnGestureListener) onLongPress(e) → {

        startVoiceRecognitionActivity();

});
```

Figure 2.4: Gesture detector code on Android Studio

As we can observe in the figure above, by instantiating the *GestureDetector* class we handle a *LongPress* event which will detect when a touch has been held on the screen for two or three seconds. If such event occurs, the method which initiates the voice recognition is called. In the following figure we observe this method in detail:



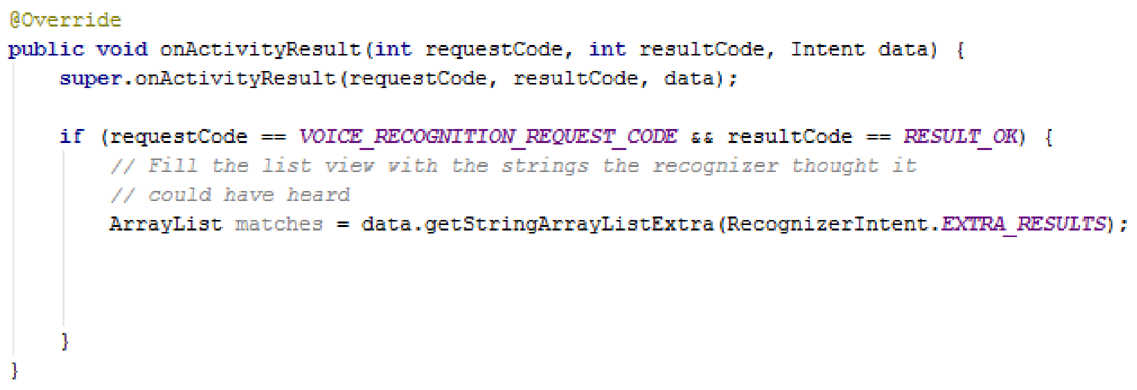
```
public void startVoiceRecognitionActivity() {
    Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
            RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
    intent.putExtra(RecognizerIntent.EXTRA_PROMPT,
            "Speech recognition");
    startActivityForResult(intent, VOICE_RECOGNITION_REQUEST_CODE);
}
```

Figure 2.5: Recognizer intent configuration code

In the previous figure we may observe how the Intent class is instantiated to be able to call upon its methods. The type of intent is passed as argument with the action set to *ACTION_RECOGNIZE_SPEECH* so as to begin the activity responsible for speech recognition with Google. Additionally, we give information to the speech recognizer about the language using the *putExtra* method *EXTRA_LANGUAGE_MODEL* is to inform the speech recognizer that additional language information is to be considered and *LANGUAGE_MODEL_FREE_FORM* is to inform the recognizer to use a language model based on free-form speech recognition. The *EXTRA_PROMPT* action indicates to the recognizer that an additional text prompt should be shown to the user when required to speak.

It is highly convenient to get the recognized speech from the activity in charge of the speech recognition. For this purpose, we can make a call to the *startActivityForResult* method which will return the recognized speech through the *onActivityResult* method. The *VOICE_RECOGNITION_REQUEST_CODE* tag is added in order to identify the activity which made the call to the intent in the first place.

As mentioned earlier, the identified speech must be taken from the result of the speech recognition activity. In the code, we can override the corresponding OnActivityResult method whereas arguments we receive the returning VOICE_RECOGNITION_REQUEST_CODE as an identifier of the intent that executed the activity and the identified speech data as well as shown in the following figure:



```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == VOICE_RECOGNITION_REQUEST_CODE && resultCode == RESULT_OK) {
        // Fill the list view with the strings the recognizer thought it
        // could have heard
        ArrayList matches = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);



    }
}
```

Figure 2.6: Activity result fetching code

As we can see in the figure above, we receive the identifier code in the argument *requestCode*, the second argument *resultCode* will indicate the status of the conversion, and the third argument *data* will provide the *Intent* with the identified speech from the recognizer activity.

In the method’s implementation body we make an *if* statement to check if the identifier code is the correct one (to check if this is the speech conversion we asked for) and we also check with the *resultCode* if the conversion had no errors. In the case in which statement is true, we enter the '*if*' statement and save the data in an *ArrayList* (a collection of data with individual elements or positions akin to a matrix or vector) by calling to the *getStringArrayListExtra* method. In the first element or position of the array we will find the most likely recognized word or phrase, the system's assurance decreases as we access the higher positions.

2.5. Text to speech synthesis in Android

Text to speech conversions in Android development can be done in a simple manner by the use of the integrated *TextToSpeech* class. By instantiating the class we are able to call its methods which allow to configure the converter and to easily convert any given string to spoken language. Furthermore, Androids allows the users to configure the text to speech module through its settings, without having to access any sort of code.

The user must first open the system's settings page. In the following figure we can observe the settings page for a Sony Xperia S on Android version 4.1.2.

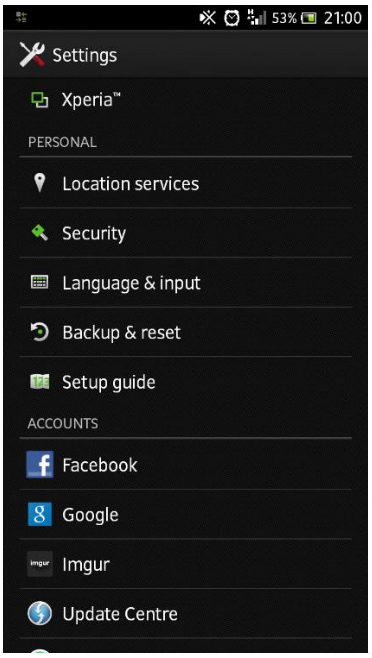


Figure 2.7: Settings page for Android 4.1.2

From here the user must access the option “Language & input” which will show the configuration page for system language and speech settings. In the following figure we can observe a screen picture of the page.

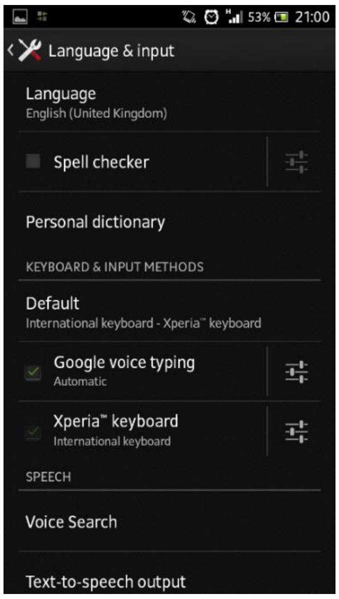

53% 21:00
Language & input
Language
English (United Kingdom)
Spell checker
Personal dictionary
KEYBOARD & INPUT METHODS
Default
International keyboard - Xperia™ keyboard
Google voice typing
Automatic
Xperia™ keyboard
International keyboard
SPEECH
Voice Search
Text-to-speech output

Figure 2.8: Language & input settings on Android 4.1.2

By setting the system’s language, the output Text-to-speech language will also change accordingly. The user may also access the settings for the input soft keyboard of the device on this page. Under the *SPEECH* section the user can gain access to the Text-to-speech output settings which will open the page observed in the figure below.

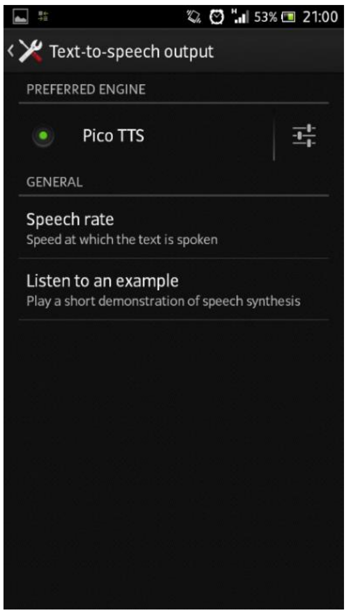

53% 21:00
Text-to-speech output
PREFERRED ENGINE
Pico TTS
GENERAL
Speech rate
Speed at which the text is spoken
Listen to an example
Play a short demonstration of speech synthesis

Figure 2.9: Text-to-speech output settings on Android 4.1.2

From this area the user may choose the preferred speech output engine. In this case only one engine is installed on the device. Text-to-speech output engines are readily available on the Google Play store for download. If more than one engine was installed on the example device, several options would be present for election and configuration. The user may additionally listen through the device's loudspeaker to an example of the configured speech output. Finally, the user may alter the speed at

which the output voice speaks the converted message as observed in the figure below.

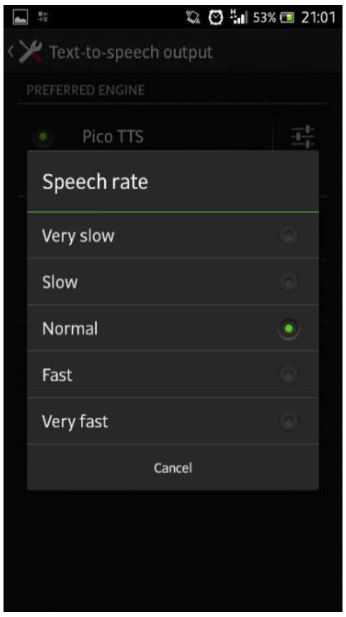
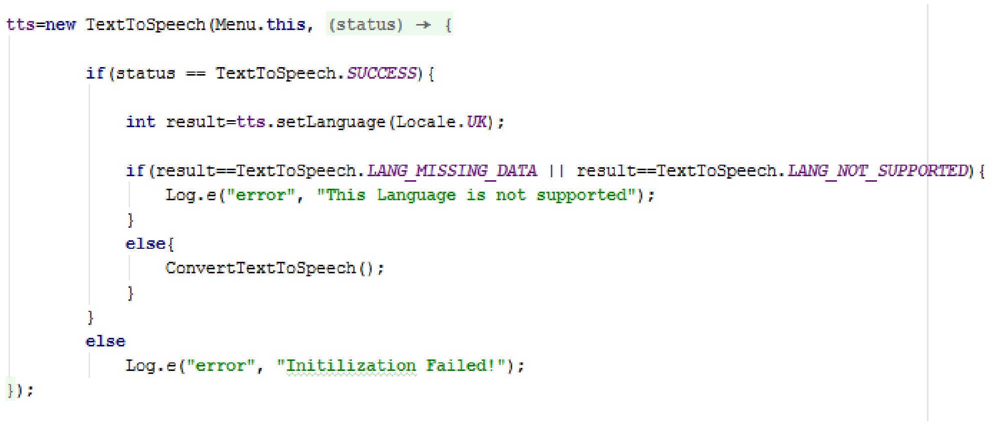


Figure 2.10: Speech rate settings on Android 4.1.2

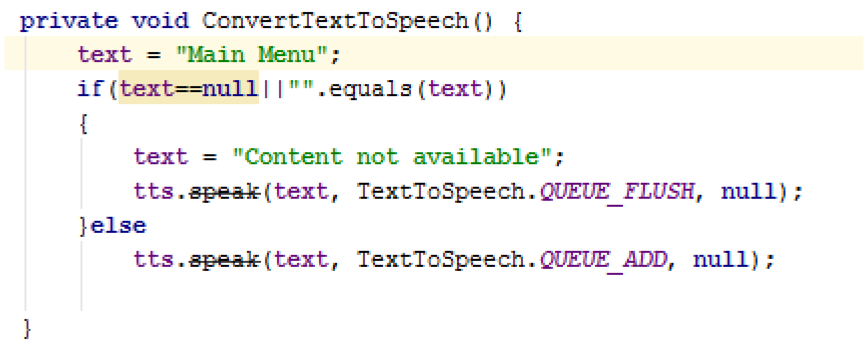
For the application to use the configured Text-to-Speech module, the developer must instantiate the class, as mentioned earlier. This has to necessarily be done within the *onCreate* method since this is our main method in the activity, the first to be called when the activity initiates its lifecycle. In the following figure we observe how this is done:



```
tts=new TextToSpeech(Menu.this, (status) -> {

        if(status == TextToSpeech.SUCCESS){

            int result=tts.setLanguage(Locale.UK);

            if(result==TextToSpeech.LANG_MISSING_DATA || result==TextToSpeech.LANG_NOT_SUPPORTED){
                Log.e("error", "This Language is not supported");
            }
            else{
                ConvertTextToSpeech();
            }
        }
        else
            Log.e("error", "Initilization Failed!");
});
```

Figure 2.11: Text to Speech initialization code

In the figure we can observe how the *TextToSpeech* class is instantiated and implemented within Android. First a check is made on the status of the module, if it returns a positive response we proceed inside the *'if'* statement. Then, we must set the language for the module, in this case to English UK, the method call returns a status report. If the status returns as LANG_MISSING_DATA or LANG_NOT_SUPPORTED that means that the language is not installed on the device or the language is not supported on the speech engine respectively. If such is not the case a method call to the *ConvertTextToSpeech* method is made, which will be the method involved in converting written text to spoken language. In the following figure we can observe the implementation of this method.



```
private void ConvertTextToSpeech() {
    text = "Main Menu";
    if(text==null||"".equals(text))
    {
        text = "Content not available";
        tts.~~speak~~(text, TextToSpeech.QUEUE_FLUSH, null);
    }else
        tts.~~speak~~(text, TextToSpeech.QUEUE_ADD, null);

}
```

Figure 2.12: ConvertTextToSpeech method for text to speech conversion

The method shown in the figure above makes a direct call to the *speak* method in the TextToSpeech class will directly convert the text inserted in the first argument call. First, in terms of convenience, the text to be converted should be stored in the string variable (variable type for storing information without the purpose of performing arithmetic operations).

An *'if'* statement follows checking if there was actually information stored in the variable. If there isn't the text stored is set to indicate "Content not available" and a method call to the *speak* method is made so as to make the message audible. However, if there is in fact information stored in the variable, the '*if´* statement will proceed to its *'else'* statement where we simply proceed to convert the stored text into an audible message.

In the second argument we handle the *QUEUE_FLUSH* and *QUEUE_ADD* constants. When text is added to the TextToSpeech converter, it will include the text information inside its playback queue as a new entry. The text will then be converted when all other entries proceeding it in the queue finish their conversion into audible messages. By specifying the *QUEUE_ADD* constant in the second argument of the *speak* method call we indicate that the text entry is to be added to the playback queue. In the other case, where the *QUEUE_FLUSH* constant is specified, all entries in the playback queue (media to be played and text to be synthesized) are dropped and replaced by the new entry.

The third argument indicates intent bundle parameters for the request. In the case shown in the figure above it can be set to *null* but supported parameters are: *KEY_PARAM_STREAM* (key to specify the audio stream type to be used when speaking text or playing back a file)*, KEY_PARAM_VOLUME* (key to specify the speech volume relative to the current stream type volume used when speaking text) and *KEY_PARA_PAN* (key to specify how the speech is panned from left to right when speaking text).

2.6. Android Apps for the disabled and visually impaired

The aforementioned technologies and many others have been used to help people who have movement difficulties or sensory impairments. By the use of android devices their daily tasks can become simpler by using built in Google apps or custom made apps installed on the device. In the table below, we can find a list of several relevant apps which can be found on the google play store.

Name	Description	Language	Android version
Google Talkback	TalkBack is an accessibility service that helps blind and vision-impaired users interact with their devices. TalkBack adds spoken, audible, and vibration feedback to your device.	All Android OS languages.	Varies with device. Available free for download.
Voice Access	Voice Access is an accessibility service that helps users who have difficulty manipulating a touch screen.	English, Spanish, French and several.	Android 5.0. Lollipop and above. Available free for download.
BrailleBack	BrailleBack is an Accessibility Service that helps blind users make use of braille devices.	All Android OS languages	Android 4.1 JellyBean and above. Available free for download.
Tecla Access	Tecla is a set of tools that provides access to mobile devices, such as smartphones and tablets, for those who are unable to manipulate them due to disease or disability.	English	Android 2.0. Éclair and above. Available free for download.
Magnifying Glass	Allows for magnifying areas of the screen on the device.	English	Android 4.0 Ice Cream Sandwich and above. Available free for download with optional pro version.
Big Launcher	BIG Launcher makes the smartphone suitable for people with eye diseases, motor problems or the legally blind.	English	Android 2.1 Éclair and above. Purchasable for €9.99.

2.7. OpenCV Library

2.7.1. Introduction to OpenCV

OpenCV is an open-source library distributed under a BSD free software licence which allows the source code to be used and changed freely for commercial and academic purposes if certain conditions are met although with minimal requirements. The OpenCV library, originally developed by Intel, is currently mostly used within computer vision and image processing based development. Since its initial appearance in 1999 the library has been used in varied applications, from security systems with the use of motion tracking systems and face recognition algorithms to process control applications where object detection is needed. The library consists of more than 2500 algorithms divided in elaborate sets of classic computer vision and machine learning algorithms. The latter is provided by a general purpose Machine Learning Library (Bradski, 2008).

OpenCV provides interfaces for the C++, C and Python programming languages. Recently support for Java and Android has been added. The library places importance in efficiency and real-time processing. Its support for hardware acceleration and multi-core processing makes this possible.

2.7.2. Installing OpenCV on Android Studio

In this section we will list the steps required to include the latest OpenCV SDK in an Android Studio project for application development. After correctly importing the SDK, all available functions and data structures contained in the Java API should be accessible to the developer.

The first step is to download the latest SDK file from the OpenCV website. For that, we must open the opencv.org webpage and click on the Downloads section.

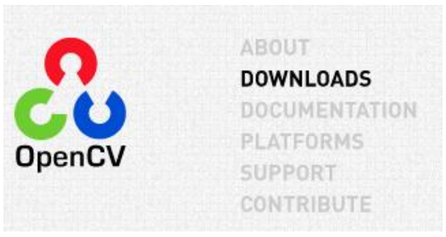


Figure 2.13: OpenCV page downloads selection

In the “Downloads” section, we may observe that the latest version available for Android OS is OpenCV 3.1, the second step is to click on the Android download link to get the zip file.

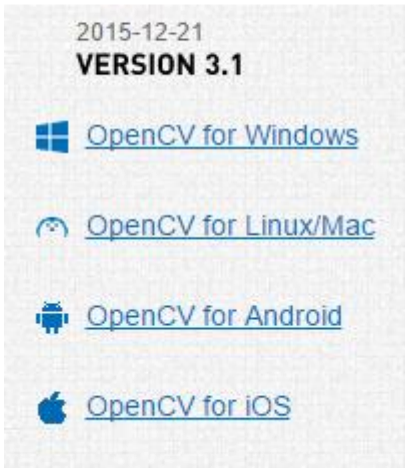
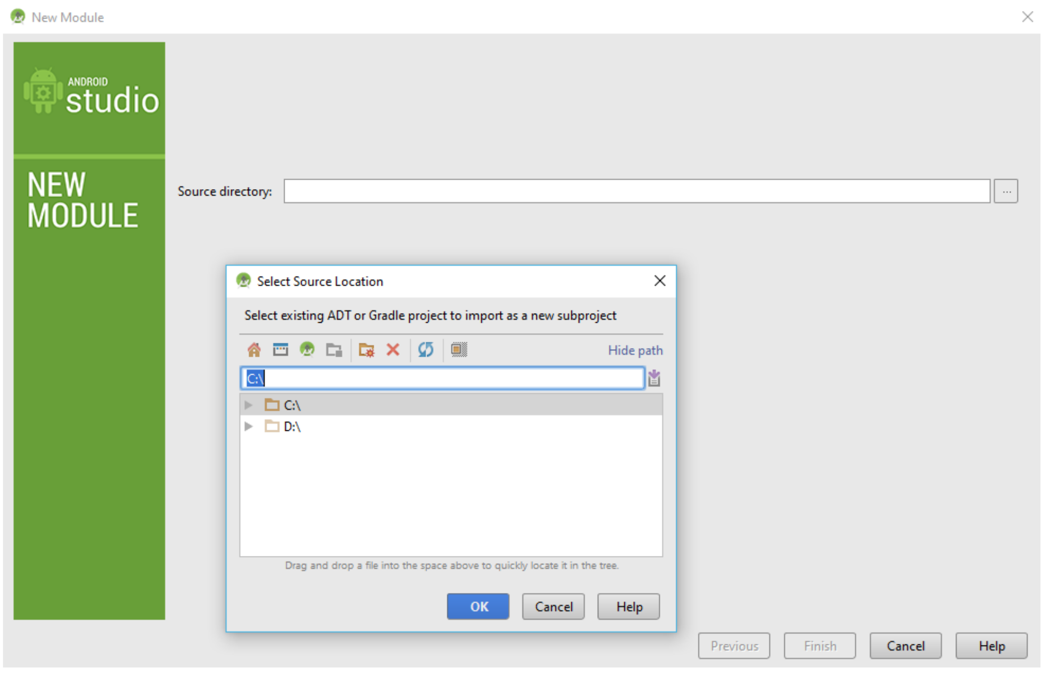


Figure 2.14: OpenCV page platform selection

After the download is complete, we must decompress the zip file and place it in an easily accessible folder. Afterwards in Android Studio, the third step is to choose the unzipped SDK folder from File -> New -> Import Module.


New Module

ANDROID studio

NEW MODULE

Source directory:

Select Source Location

Select existing ADT or Gradle project to import as a new subproject

Hide path

C:\

C:\

D:\

Drag and drop a file into the space above to quickly locate it in the tree.

Figure 2.15: Module import in Android Studio

A new build gradle will be added to the project corresponding to that of the imported OpenCV SDK, the fourth step is to make it match the build gradle of our native project changing the *compileSdkVersion*, *buildToolsVersion*, *minSdkVersion* and *targetSdkVersion* parameters.

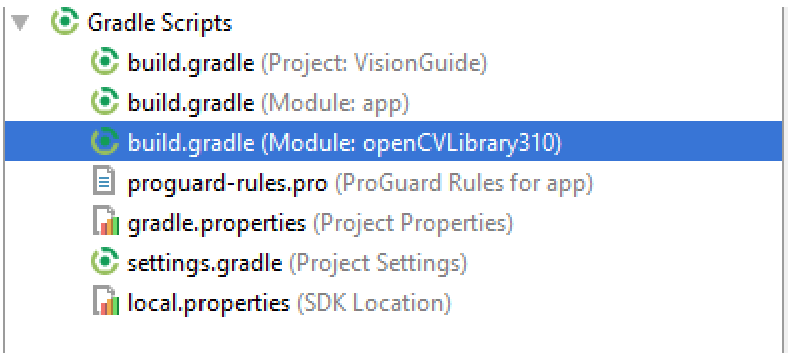
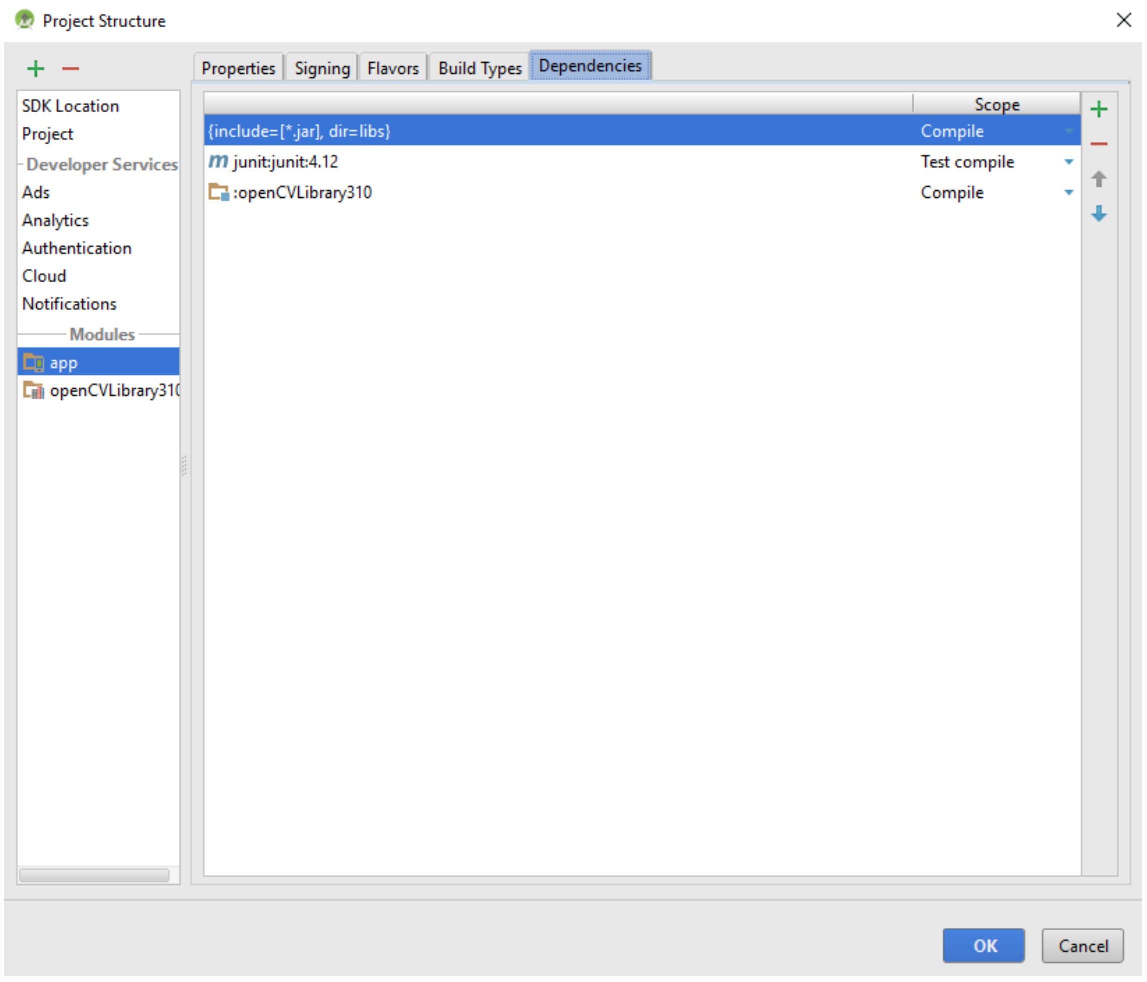

Gradle Scripts
build.gradle (Project: VisionGuide)
build.gradle (Module: app)
build.gradle (Module: openCVLibrary310)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

Figure 2.16: OpenCV gradle selection in Android Studio.

Now we must add the module dependency for our project. We must find the dependencies tab from Application -> Module Settings. The fifth step is to click on the '+' icon then on "Choose Module Dependency" and last on the imported OpenCV module we will have established the dependency within our project.


Project Structure

	Scope
{include=[*.jar], dir=libs}	Compile
junit:junit:4.12	Test compile
:openCVLibrary310	Compile

Figure 2.17: Project dependencies selection in Android Studio

The last step is to copy the libs folder under sdk/native and paste it in our app/src/main folder. Afterwards, we must rename the folder to jniLibs. This is done because Android Studio expects native libs in the app/src/main/jniLibs folder location instead of the older libs folder.

2.7.3. The OpenCV Manager

The OpenCV Manager was introduced by NVIDIA after the release of OpenCV version 2.4.2. It has become essential for any Android device that wishes to run an Android based application. The OpenCV manager is readily available on the Google Play Store or installable by using the Tegra Android Development Pack. The goal of the OpenCV manager is to manage the libraries, update them and selecting the most optimized versions depending on the device.

By dynamically linking the OpenCV libraries (instead of statically linking them where they form part of the application), the OpenCV Manager can more efficiently install any available updates. Dynamically linking the libraries means they are linked on runtime, when the application is launched. In the case of static linking, the libraries and the application would need to be updated strictly together, this meaning that upon any update of an OpenCV library, the application would need to be re-released entirely. By the use of dynamic linking, the OpenCV Manager is also able to detect hardware automatically.

2.7.4. OpenCV sample applications

The OpenCV SDK provides sample applications and five tutorials which help with the initial development steps in OpenCV. These tutorials and samples are meant to work as frameworks or foundation for the application’s development. The samples and tutorials are listed in the following:

- Android Camera: This tutorial does not use OpenCV libraries at all, but it presents itself as a skeleton for any application which uses the Android native camera.
- Add OpenCV: A tutorial which demonstrate a simple example of how a call to the OpenCV library is done.

- OpenCV Camera: This sample is similar to the Android native camera sample but provides a framework for using the OpenCV camera instead.
- Add Native OpenCV: This tutorial teaches how to set OpenCV as a native part of the application development by the use of JNI.
- JAVA/C++: This tutorial demonstrates how to use C++ and Java OpenCV APIs in the same application.
- Image- manipulations: A sample which shows how OpenCV is sued for processing and manipulating input images.
- 15-puzzle: A sample which is an implemented game showing what possible development is with OpenCV. Readily available on the Google Play Store.
- Face-detection: A sample which serves as a simple implementation of a face-detection system on Android.
- Colour-blob-detection: A sample showing a simple implementation of a colour blob tracker in real-time.

2.7.5. OpenCV Architecture

The OpenCV library presents a modular structure. The distributed package consists of several individual shared or static libraries such as the following:

- **Core functionality module (core):** This module presents the basic functions that will be used within other modules, including the multi-dimensional array Mat and other basic data structures
- **HighGUI module:** Module that makes possible the representation of information and basic UI capabilities.
- **Image processing module (imgproc):** Module for the processing of images providing geometric transformations, perspective deviation, filtering, colour manipulation, etc.
- **Video module:** This module, dedicated to real-time video processing, allows for object tracking algorithms and motion estimation.
- **Calib3d module:** Containing 3D reconstruction algorithms, this module additionally implements multiple-view geometry algorithms and object pose estimation.

- **Features2D module:** By using this module the developer is able to work with descriptor detecting algorithms and feature matching.
- **Objdetect module:** This module allows instancing of pre-defined classes for object detection (for example, face and eye detection, vehicle detection).
- **Videoio module:** Mainly used for video capturing and video codecs.
- **GPU module**: This module allows the GPU acceleration of processes and algorithms.

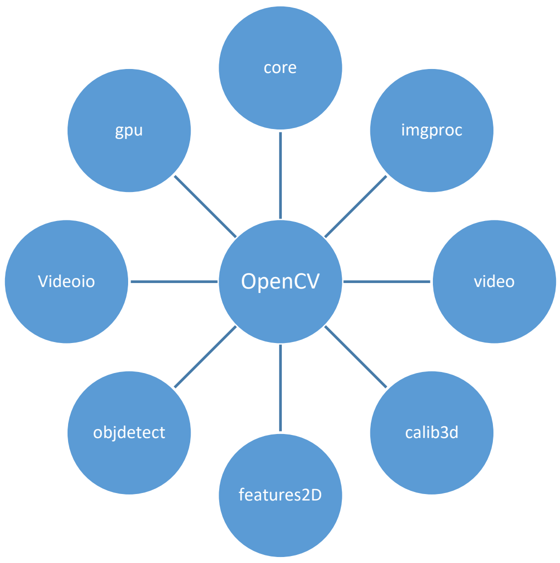


Figure 2.18: OpenCV modules diagram

2.7.5.1. Core functionality module

OpenCV works fundamentally with matrix representations of digital images obtained via digital capturing devices such as cameras and scanners, or artificially created images (OpenCV 3.1.0 Tutorials). Within the cells of the stored matrix relevant information of each pixel is stored for future access and manipulation.

With the use of the *Mat* class, OpenCV is able to handle the numeric data information contained in these matrices. The *Mat* class is in itself composed of the matrix header, which contains relevant information on the nature of the stored data matrix (such as size, storing method, etc.), and a pointer towards the matrix itself containing the pixel values in its cells. Other simpler data structures may also be defined using OpenCV, such as *Point* which is a simple data structure class with two integer parameters x and y (Bradski, 2008). Others such as *Rect* may be defined in a more elaborate manner by specifying its parameters x, y, width and height. Furthermore, *Scalar* is described as a set four double-precision numbers, it contains a single member *val* which points to an array containing the aforementioned information in floating-point format (Bradski, 2008). In any case, constructor methods are available for the developer to define and initialize these data structures.

2.7.5.2. HighGUI module

The HighGUI module keeps operations such as interactions with hardware and file systems simple for the developer. With the HighGUI we are able to easily display, read and write on images and video or to display additional UI elements for extra functionality.

From the hardware perspective, the HighGUI module allows to easily access the latest feed from the camera, which can be tedious if done directly working with the operating system.

The module also allows to load and save images. For this purpose the module offers a pair of load and save functions which, if correctly supplied with the required parameters, handle the task of decoding and encoding for the developer.

Furthermore, we are able to handle mouse and keyboard interactions on any window and element we render by using the HighGUI module. The task of creating a simple UI with buttons and sliders becomes quick and simple.

2.7.5.3. Image processing module

With this module we are able to perform more advanced operations for manipulating image structures than in the core and highgui modules. In other words, using this module we are able to process images in a more generic way, as images instead of arrays and matrices of values.

Frequent image processing operations such as smoothing or blurring become possible when using the imgproc module. Sometimes it is of interest to reduce noise on an image, for which smoothing operations become relevant. OpenCV applies a filter to the image which in itself may be of different types (linear, median, bilateral, guassian) the most common being the linear filter.

Morphological transformations such as dilation and erosion are also available and they allow us to remove noise, joining disparate elements in an image or isolating individual elements. Morphology is also used to find intensity bumps or holes in an image and to find image gradients. The operation consists of convoluting an image A with a kernel B (OpenCV 3.1.0 Tutorials).

Other operations such as opening, closing, morphological gradient, image resize, top hat and black hat are available for use with the module in image transformation purposes.

2.7.5.4. Video module

The video module provides methods and algorithms to real-time object tracking which becomes relevant in video processing when we focus on a single or determined set of objects in a scene. Such can be done with the use of the image processing modules by working with each video frame, however with this module we are able to understand the motion of the object.

Although not absolutely required, the first task is usually to identify the object to study in each frame so that we may track it in subsequent frames. Techniques for tracking unidentified objects typically involve tracking visually significant key points where we may observe differentiating details (Bradski, 2008).

The second task, modelling, will allow us to take the variable information obtained from object or key point tracking and use mathematical approximations for the objects trajectory.

Additionally, with background subtraction (BS) techniques we are able to filter or isolate the object of interest by generating a binary image containing the pixels belonging to the moving objects in the scene (OpenCV 3.1.0 Background Subtraction Tutorial). The binary image or foreground mask is mainly obtained by subtracting the current frame and a background model containing everything that can be considered as background in the current scene.

2.7.5.5. Calib3d module

With the calib3d module we are able to handle camera calibration to mathematically correct deviations imposed by the pinhole model when the real three dimensional world scene is captured by camera lenses. The physical world is also handled in real physical units, thus it is important to find a good relation with the cameras measurement units (pixels) if we wish to reconstruct the three-dimensional scene. For this purpose we are able to perform the mathematical homography transformation.

2.7.5.6. Features2D module

The features2D module implements the methods and algorithms for the detection of features and descriptors on input images. Moreover, it allows feature matching between two distinct images. Features of a certain object or scene can be defined in terms of OpenCV as characteristics of the scene which we can easily identify (OpenCV 3.1.0 Features2D Tutorials). The feature detection algorithms search for different types of features, those being edges, corners (also known as interest points) and blobs (also known as regions of interest). Corners represent an area where two edges intersect, the direction of the edges change thus a high variation of the gradient will be present in that region, something OpenCV can easily detect. A similar situation is present with edges, where from a set of pixels to another a detectable change occurs.

The features2D module implements the necessary functions and interfaces to also draw the key points or matches between two images as small coloured or black circles. Several algorithms are available within the library for the feature detection itself. The OpenCV 3.1.0 base package contains the following:

- **BRISK:** Feature detection algorithm used to construct binary key points using a sampling pattern (where the sampling points will take place) composed of concentric rings. Afterwards, a smoothing Gaussian filter is applied to each sample and the binary key points are determined by performing intensity comparisons. The algorithm additionally works with an orientation compensation mechanism, thus becoming to a certain extent invariant to key point rotation.
- **BRIEF:** Algorithm for constructing binary key points and descriptors, it has the distinct feature of not having a defined sampling patterns and thus samples for descriptor construction are taken in random manner. It makes use of a smoothing Gaussian filter to make the sampling stage less sensitive to noise.

- **ORB:** Similar to BRIEF AND BRISK it constructs binary key points and is supported by an orientation compensation mechanism which adds robustness for rotated key points. ORB additionally learns the most optimal sampling pairs instead of choosing them randomly. Sampling pairs are later used to compare and thus build the final descriptor.
- **FAST:** A computationally efficient feature and descriptor extraction algorithm it uses a circular sampling pattern of 16 pixels (Bresenham circle of radius 3) to detect features. It further makes use of machine learning to improve the algorithm's detection
- **SURF:** This algorithm is well known for its robustness and effectivity at detecting features within a scene. However it is also computationally intensive. It makes use of multi-resolution techniques to obtain coordinates from the input images. It then creates different versions of the input image but with a reduced bandwidth using Laplacian or Gaussian Pyramids. A blur effect will occur in the image and thus constructing a Scale-Space. This makes the interest points invariant to scale.
- **FREAK:** This feature detection and descriptor extractor algorithm shares features from both BRISK and ORB. It makes use of a pre-defined retinal sampling pattern and additionally uses machine learning processes to get the most optimal sampling pairs. It contains an orientation compensation mechanism similar to that of BRISK.
- **MSER:** Algorithm used for blob detection in a given scene it extracts from the image a certain amount of co-variant regions called MSERs. The MSER regions are stable connected components of the image. The algorithm is based on taking regions which a nearly the same through a wide range of thresholds.
- **SIFT:** Predecessor to SURF it can robustly identify objects in the presence of visual noise our partial occlusion. It presents itself as invariant to scale changes and orientation or illumination changes.

2.8. OpenCV Android Apps for the visually impaired

Research on the OpenCV library endures to create augmented reality and computer vision systems which can aid the visually impaired or individuals with mental illnesses. With the introduction of OpenCV on the Android platform, many apps are being developed which can provide such aid on Android devices and allow those individuals who have difficulties with vision or discerning objects with artificial on-board vision in the provided app.

Name	Description	Languages	Android version
Colour Assist	This app aids people with colour-blindness and assists them with distinguishing different colours using the camera on their phone.	English	Android 2.2 Froyo and above. Application is free for download and use.
Visual Coin Counter	This app uses computer vision algorithms to detect coins and then add up the values.	English	Android 2.2 Froyo and above. Application is free for download and use. Pro version available to remove ads.
ICsee	ICSee applies special filters to the real-time image taken from the camera of the smartphone/ tablet, making identifying indiscernible objects and texts easier.	English	Android 4.0 Ice Cream Sandwich. Application is free for download and use.
Sign Language Interpreter	Sign Language Interpreter will convert signs from ASL and ISL to appropriate text and audio.	English	Android 2.2 Froyo and above. Application is free for download and use.
ReadEasy	ReadEasy is a mobile application designed to help visually impaired to read signs.	Italian	Android 4.0 Ice Cream Sandwich. Application is free for download and use.

Chapter 3

SYSTEM DESCRIPTION

3.1 General overview

The Vision Guide application consists of four main modules each with independent processes which contribute to the general functionality of the system. These modules are named as the following:

- **Main Menu:** This is the core of the application. By use of this activity the user will be able to launch the other modules by touching on their icons on the screen or by communicating their name through speech recognition.
- **Object Capture:** This activity is in charge of capturing still images of objects and saving them in the database giving them a name and identifier.
- **Object Recognizer:** This activity's main objective is to launch the OpenCV library to process camera frames, recognize objects in real time, and show information on screen and by audible message.
- **Colour Recognizer:** This activity applies an algorithm to calcite the amount of colour on the scene in a pre-defined HSV range.

On the application's start up, a welcome screen greets the user with the application's name on screen and by an audible message. By touching on the screen the user will be able to access one of the most important modules of the application: The Main Menu.

The main goal in this implementation style is to allow separate functionalities independent of each other, providing a multi-purpose application ideally to provide aid in different facets of human vision.

In this chapter a detailed study will be done on the general functioning of the system. The functionality of each module and activity will be studied independently. From the front-end coding of what the user sees or hears to the back-end coding of how the algorithms that makes the core functionalities possible such as the object recognition.

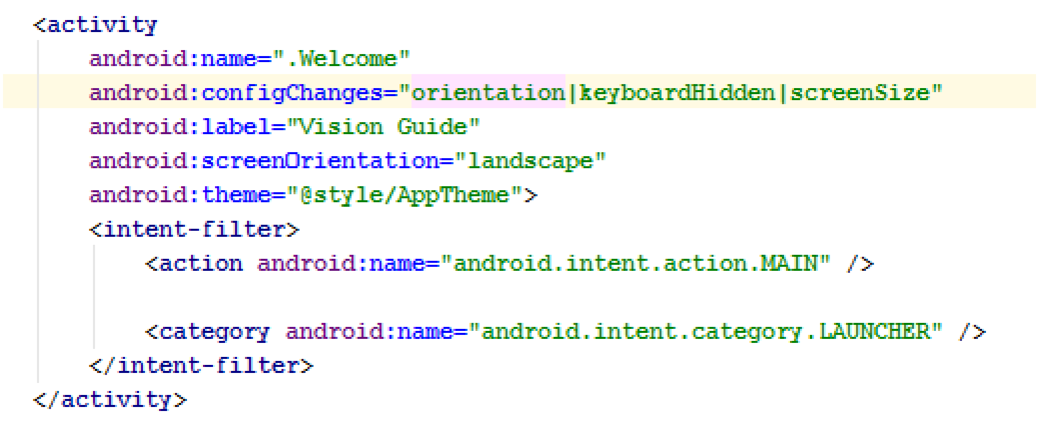
3.2. The Welcome Screen

Upon the application’s start up, the Welcome Screen activity is launched. A full screen image is shown containing the application’s name and with the *Action Bar* (area of the screen in an activity where titles, icons and menus are located) hidden. Since the main functionalities of the application require a landscape horizontal view, this is already imposed on this activity. Thus, the entire application will strictly work in landscape mode.

Figure 3.1: Screen image of the Welcome screen activity

To ensure that the *Action Bar* stays hidden, the activity must be configured in the *AndroidManifest.xml.* There are several options, we can either make the applications theme be *Theme.NoTitleBar.Fullscreen* or set *configChanges* to *screenSize*. In this activity, the latter method was chosen.

Furthermore, also within the *AndroidManisfest.xml.* file it is necessary to indicate that this activity will be the launcher activity, that is, the first activity that will show upon the application’s start up. To do this, an intent filter must be specified. And thus combining the former configuration and the intent filter, the configuration of the activity in the manifest will be the following:

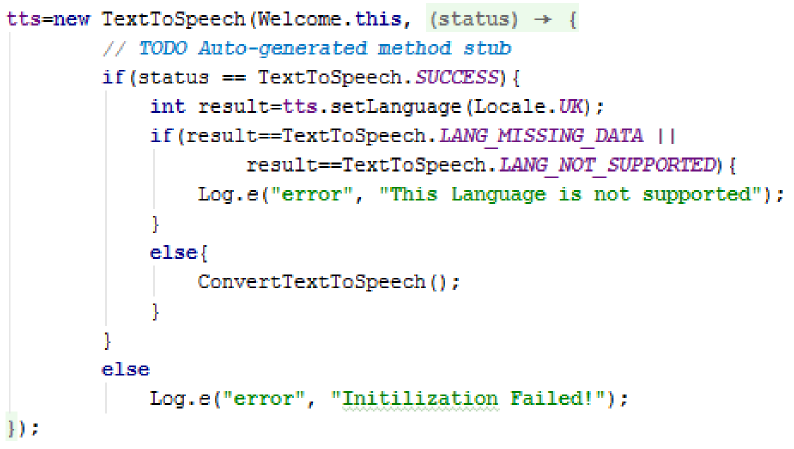


```
<activity
    android:name=".Welcome"
    android:configChanges="orientation|keyboardHidden|screenSize"
    android:label="Vision Guide"
    android:screenOrientation="landscape"
    android:theme="@style/AppTheme">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Figure 3.2: Welcome activity configuration in manifest file

Notice how the configuration is placed within the confines of the *activity* tags. The first attribute to be specified is the name of the activity for Android Studio. The next line states the aforementioned configuration for the full screen and horizontal image. A label is specified to be able to identify the activity within Android Studio. By setting the *screenOrientation* attribute to landscape, we force the activity to strictly run in landscape mode, this means the layout will be horizontal and will prevent Android's automatic screen orientation adjustment system to change it. And, by specifying the *theme* attribute, the colours and styles of the entire activity it set up.

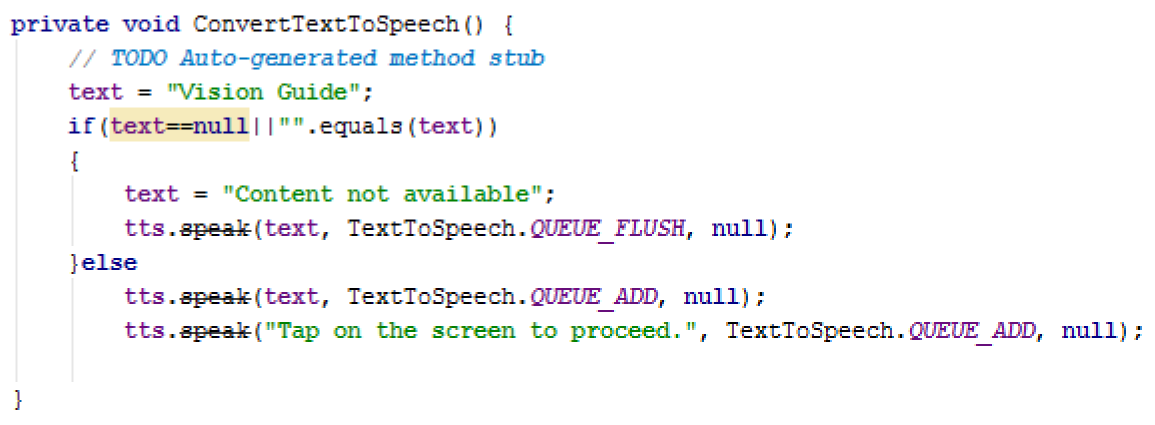
With the goal of sending an audible welcome message to the user upon the activity's initiation, it is necessary to start the TTS module within the *onCreate* method, which will be as an analogy to the *Main* method in Eclipse the first method to be executed. This implementation is shown in the following figure:



```
tts=new TextToSpeech(Welcome.this, (status) → {
        // TODO Auto-generated method stub
        if(status == TextToSpeech.SUCCESS){
            int result=tts.setLanguage(Locale.UK);
            if(result==TextToSpeech.LANG_MISSING_DATA ||
                    result==TextToSpeech.LANG_NOT_SUPPORTED){
                Log.e("error", "This Language is not supported");
            }
            else{
                ConvertTextToSpeech();
            }
        }
        else
            Log.e("error", "Initilization Failed!");
});
```

Figure 3.3: TTS module configuration in Welcome activity

As we can observe in the figure above, the TTS module will make a call to the *ConvertTextToSpeech* method if the set up returned no errors. This method will be in charge of calling the speech conversion methods with the text we wish to convert as argument. The implementation is shown in the next figure:

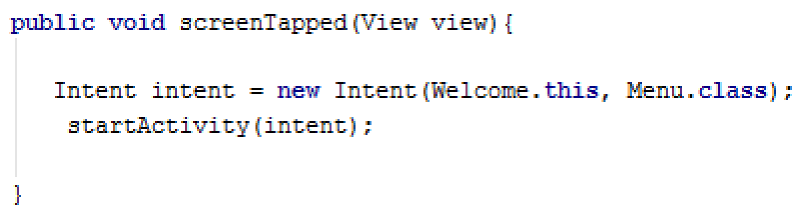


```
private void ConvertTextToSpeech() {
    // TODO Auto-generated method stub
    text = "Vision Guide";
    if(text==null||"".equals(text))
    {
        text = "Content not available";
        tts.~~speak~~(text, TextToSpeech.QUEUE_FLUSH, null);
    }else
        tts.~~speak~~(text, TextToSpeech.QUEUE_ADD, null);
        tts.~~speak~~("Tap on the screen to proceed.", TextToSpeech.QUEUE_ADD, null);

}
```

Figure 3.4: ConvertTextToSpeech method implementation in Welcome activity

In the figure above, we can observe that the text that will be converted to an audible message will be “Vision Guide”, the name of the application. Thus it is expected that the user hears the name of the application upon start-up.

Furthermore, to allow the user to advance to the next activity upon touching the screen, a method must be added to handle this event. This method’s implementation is shown in the next figure:



```
public void screenTapped(View view){

   Intent intent = new Intent(Welcome.this, Menu.class);
    startActivity(intent);

}
```

Figure 3.5: screenTapped method implementation in Welcome activity

In the next figure a flow chart is shown showing how this activity works following the processes that have been explained.

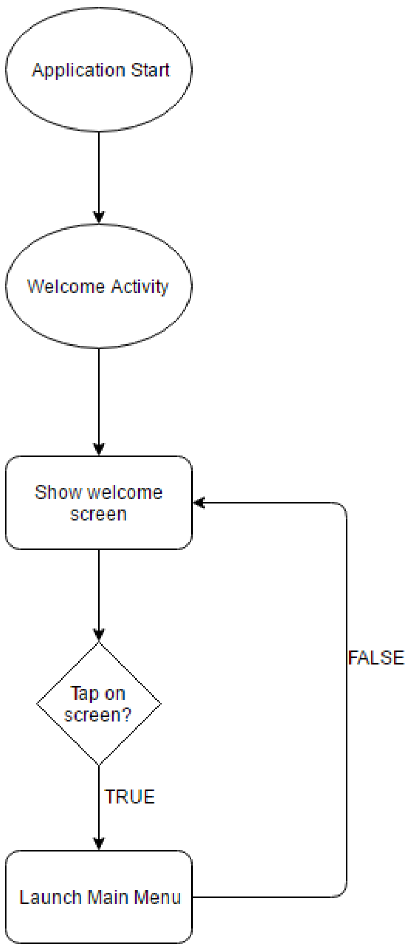
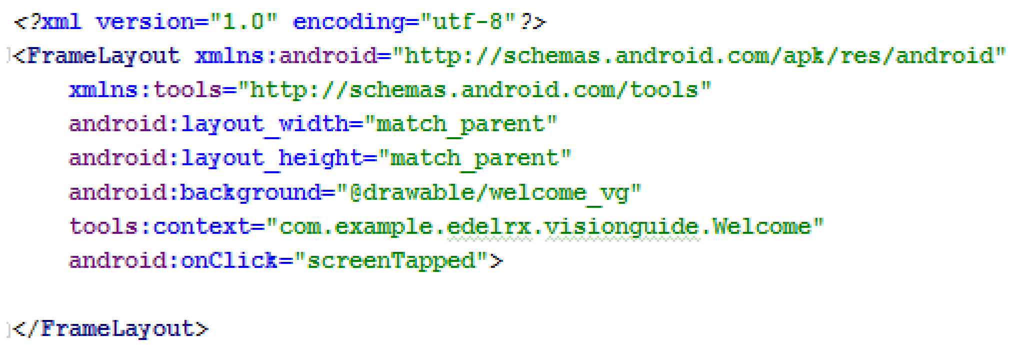


Figure 3.6: Flow chart for Welcome activity functionality

The method shown in the figure above will handle an *onClick* event on the specified *View,* it will then create an intent which will allow us to end the current activity's lifecycle and start the next activity. To specify which View in which this event is handled, we must add the specification onto the layout file corresponding to the activity. The layout file handles the graphic interface of the activity, thus here we will add or create UI elements and establish their relation with the back-end JAVA code. The layout XML file for the *Welcome* activity is shown in the next figure.



```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/welcome_vg"
    tools:context="com.example.edelrx.visionguide.Welcome"
    android:onClick="screenTapped">

</FrameLayout>
```

Figure 3.7: Layout configuration for Welcome activity

In the figure above we can observe how the activity contains a *FrameLayout* which functions as a container for other Views or layout components. It is also able specify a number of properties. In the *android:background* property we are able to specify a background image for this layout. In this case we have chosen an image which is present in the *drawable* container file which corresponds to the full screen image we will see when starting the application. Last, the *android:onClick* property specifies which method will be called when a click or touch event occurs on the screen. In this case, the previous method mentioned which takes us to the next activity.

3.3. The Main Menu

As in the case of the *Welcome* activity, the device will send an audible message to the user announcing the name of this activity so that in case the user has vision impairment he/she will easily be able to know which module is currently active.

This activity contains an *Action Bar* which shows the user the name of the activity. It will also contain three separate images on the screen, which upon touch will send the user to the separate modules of the system. The screen for this activity is shown in the following image:

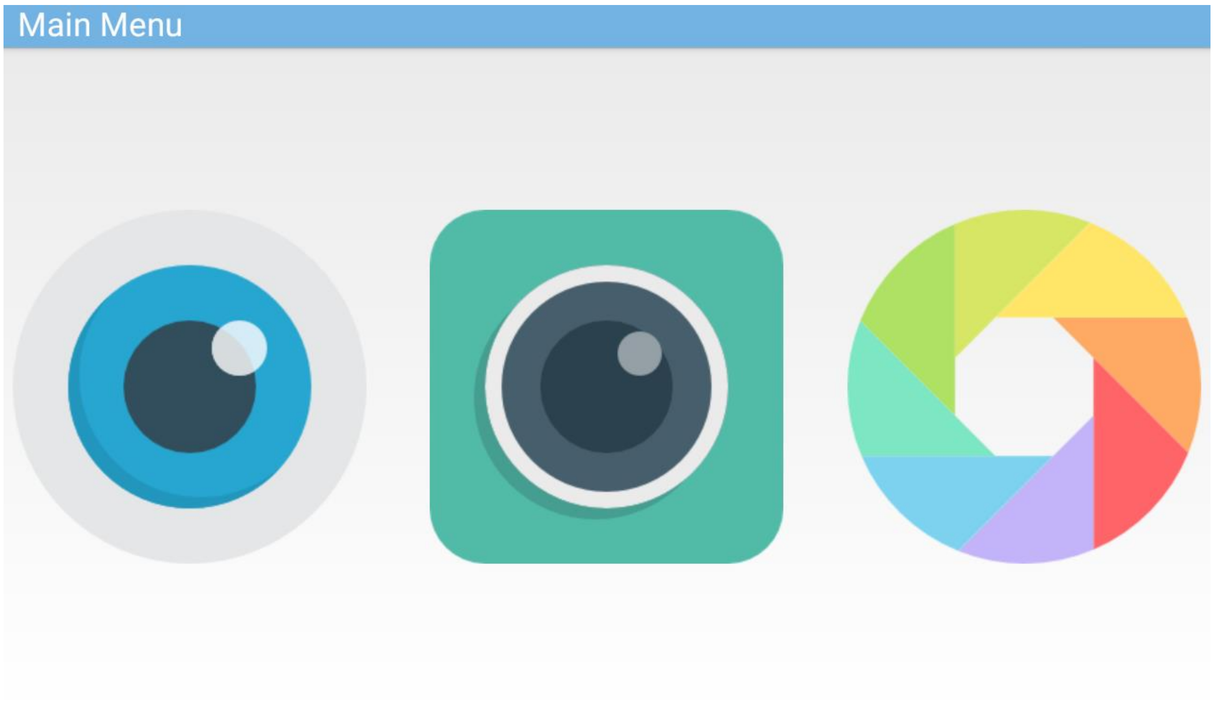
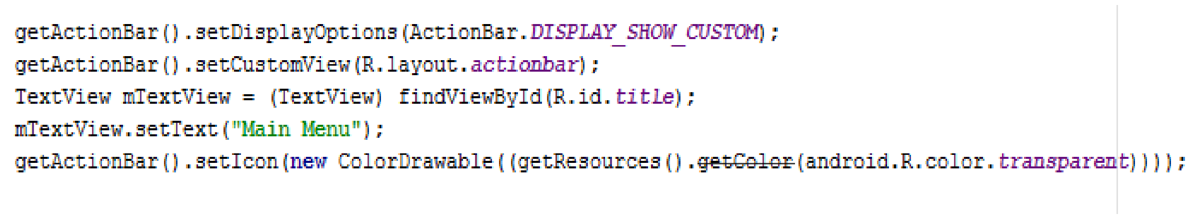


Figure 3.8: Screen image of the Main Menu activity

Since this activity does contain a visible *Action Bar* this must be specified in the code. Within the *on*Create method the *Action Bar* must be obtained. The following figure shows the set-up for this specific case.

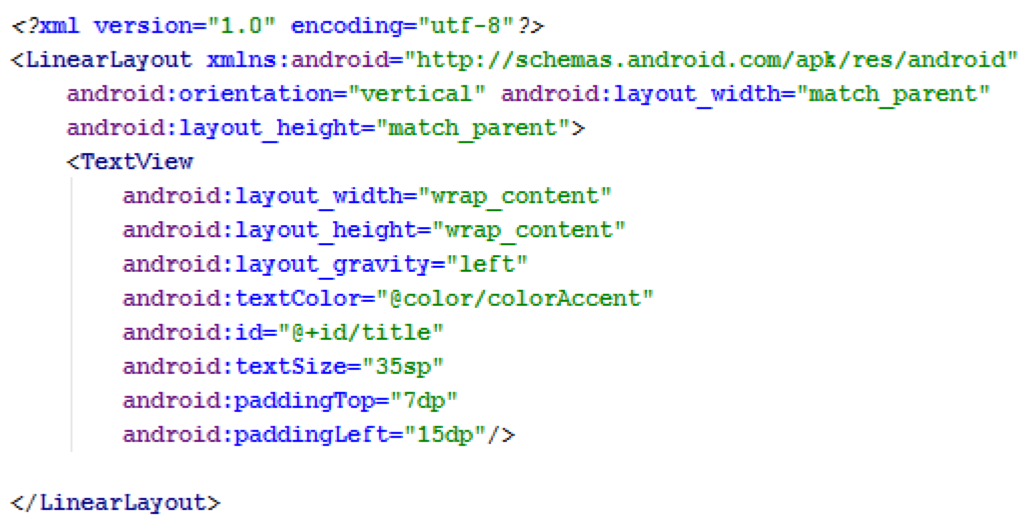


```
getActionBar().setDisplayOptions(ActionBar.DISPLAY_SHOW_CUSTOM);
getActionBar().setCustomView(R.layout.actionbar);
TextView mTextView = (TextView) findViewById(R.id.title);
mTextView.setText("Main Menu");
getActionBar().setIcon(new ColorDrawable((getResources().~~getColor~~(android.R.color.transparent))));
```

Figure 3.9: Action Bar configuration in the Main Menu activity

On the first line of code, we first call to the method *getActionBar()* to obtain an instance of the *Action Bar*, then the display is set to custom so that we may freely manipulate its graphic style. On the next line of code, the layout for this Action Bar is set to an XML file stored in the layout folder. The environment will search within this folder for the XML file to organise the elements to be shown. When the layout file has been set, we are able to access the *title* field and edit the title by making a call to the *setText* method. Last, we disable the visibility of the application's icon which is shown on the *Action Bar* by default.

The layout file for the *Action Bar* is shown in the next figure:

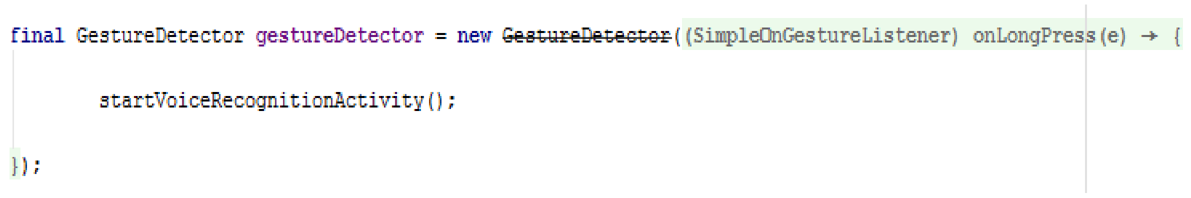


```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="left"
        android:textColor="@color/colorAccent"
        android:id="@+id/title"
        android:textSize="35sp"
        android:paddingTop="7dp"
        android:paddingLeft="15dp"/>

</LinearLayout>
```

Figure 3.10: Action Bar layout configuration in Main Menu activity

As we can observe the in the figure above, the basic container will be a *LinearLayout*. The difference with the previously studied *FrameLayout* is that the *LinearLayout* strictly places the elements below one another while the *FrameLayout* allows for more placement freedom. A *TextView* element is added which is the one accessed when we set the *Action Bar* title shown in the previous figure. To be able to find this *TextView* we must specify an *android:id*. As we saw in the previous figure, we obtained this *TextView* by searching for the specified *android:id*.

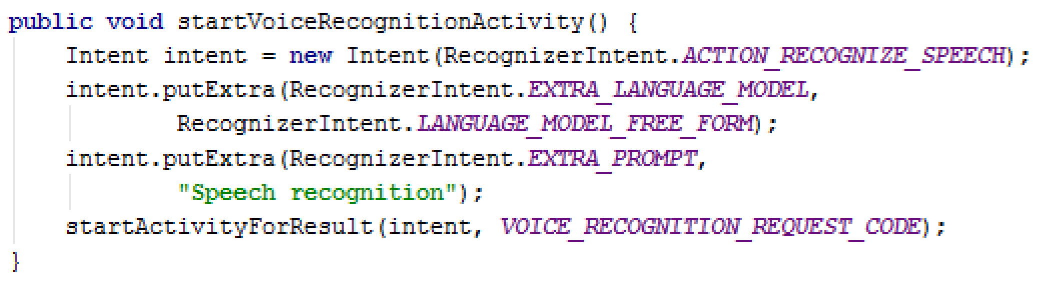
Since this activity will redirect the user to the modules of the system, it is necessary for it to contain a voice recognition module. The user will be able to access the module by holding a touch on the screen, after a few seconds the voice recognition system will prompt the user for speech. This gesture must be set and handled within the code, as shown in the following figure:



```java
final GestureDetector gestureDetector = new ~~GestureDetector~~((SimpleOnGestureListener) onLongPress(e) → {

        startVoiceRecognitionActivity();

});
```

Figure 3.11: gestureDetector configuration in Main Menu activity

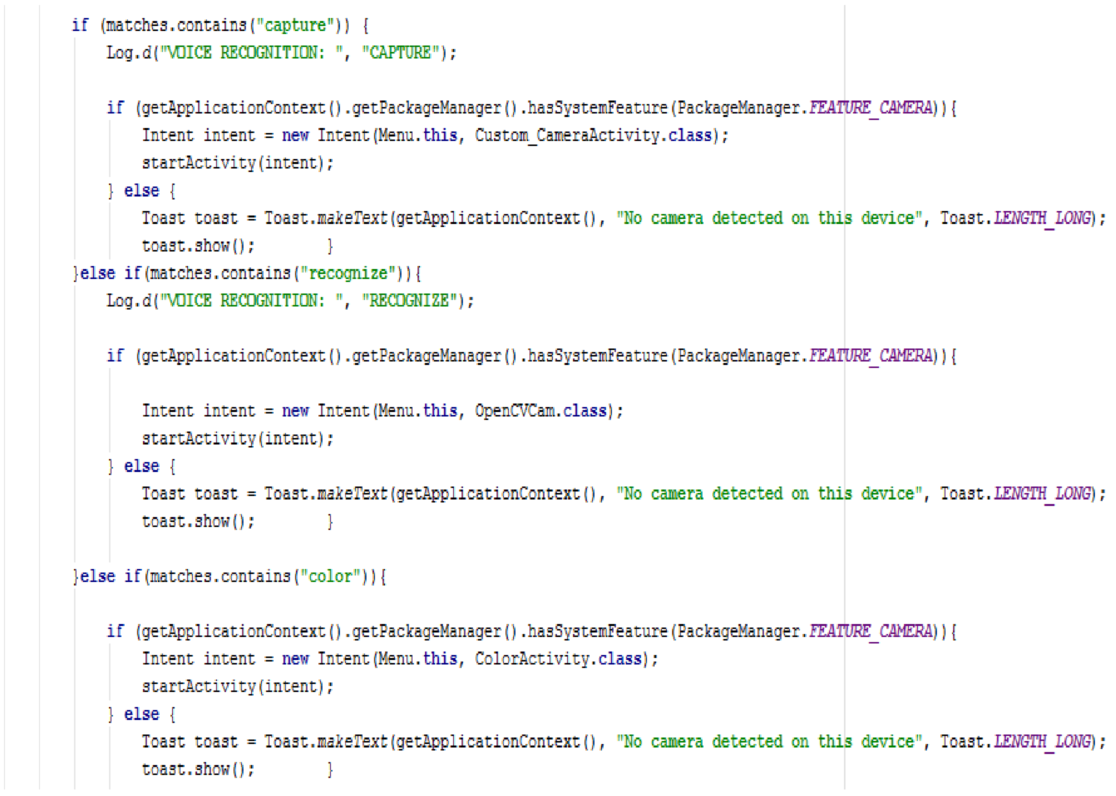
The method above will create a listener long press event which will occur when the user holds a touch on the screen for longer than two seconds. When this happens, a call to the *startVoiceRecognitionActivity* method is made, which will handle the start-up configuration for the speech recognition module. The implementation of this method is shown in the next figure:



```
public void startVoiceRecognitionActivity() {
    Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
            RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
    intent.putExtra(RecognizerIntent.EXTRA_PROMPT,
            "Speech recognition");
    startActivityForResult(intent, VOICE_RECOGNITION_REQUEST_CODE);
}
```

Figure 3.12: Speech recognition intent configuration in Main Menu activity

As we can observe in the previous figure, an intent is created to configure the speech recognition module. The first configuration begins when we make a call to the *putExtra* method which allows us to set additional parameters. The language settings are thus applied, and an additional prompt is added to the user upon initiation for the speech recognition module. Last, the speech recognition activity is started by calling the *startActivityFoResult* method. As arguments we specify the configuration intent we have just created, and an identifier tag which we will use to know which activity is returning results. With this, the activity in charge for Google's speech recognition will start and will return information as result, which we will be able to obtain by overriding the *onActivityResult* method. The implementation of the previously mentioned method is shown in the next figure.



```
if (matches.contains("capture")) {
    Log.d("VOICE RECOGNITION: ", "CAPTURE");

    if (getApplicationContext().getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA)){
        Intent intent = new Intent(Menu.this, Custom_CameraActivity.class);
        startActivity(intent);
    } else {
        Toast toast = Toast.makeText(getApplicationContext(), "No camera detected on this device", Toast.LENGTH_LONG);
        toast.show();       }
}else if(matches.contains("recognize")){
    Log.d("VOICE RECOGNITION: ", "RECOGNIZE");

    if (getApplicationContext().getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA)){

        Intent intent = new Intent(Menu.this, OpenCVCam.class);
        startActivity(intent);
    } else {
        Toast toast = Toast.makeText(getApplicationContext(), "No camera detected on this device", Toast.LENGTH_LONG);
        toast.show();       }

}else if(matches.contains("color")){

    if (getApplicationContext().getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA)){
        Intent intent = new Intent(Menu.this, ColorActivity.class);
        startActivity(intent);
    } else {
        Toast toast = Toast.makeText(getApplicationContext(), "No camera detected on this device", Toast.LENGTH_LONG);
        toast.show();       }
```

Figure 3.13: Main menu activity speech interaction configuration code

In the previous figure we can observe that the *@Override* tag indicates that we are overriding the method (replacing or writing its implementation code). This tag is not strictly necessary, if we were to delete it, the code would compile. The *onActivityResult* method receives three arguments: *requestCode, resultCode,* and *data.* The first argument, *requestCode*, indicates the indentification code we had set up before, this is to verify this is our corresponding method call. The second argument, *resultCode,* indicates the status of the result returned. If *RESULT_OK* is specified then the returned data is valid. The last argument, *data*, returns the *Intent* with the resultant data from our method call.

Within the implementation of the method an *'if'* statement checks for the identification and status of the conversion. In case everything is according to the requirements we enter the statement to recover the resultant data. To recover a list of strings of all recognized words or phrases a call to the *getStringArrayListExtra* method is made and additionally *RecognizerIntent.EXTRA_RESULTS* is set as argument to specify that what we wish to retrieve is the results of the speech conversion. This is stored in the *matches* variable.

At this point, we are able to search within the list if a certain word was recognized. Since this speech recognition module will be used to access other modules of the system, it is of our interest to search for words which the user might mention when trying to access one of the modules. Thus, the first word we search for is "capture". Inside the *'if'* condition statement, we make a call to the *contains* method which will return "true" if the word specified in its argument is indeed present in the list. In this case, the '*if'* will proceed to its inner code where redirection can take place.

Before redirecting the user towards the module in charge of capturing objects, we make a check to see if the present device has a camera since it is a strict requirement for the module's core functioning. If the device contains a functioning camera, the intent is made to end this activity's lifecycle and initiate the next one. If no camera is present, a *Toast* message is displayed (a short message on the device's screen).

This same process is repeated for the other cases, if we wish to access the object recognition module we make a check to see if the list of recognized words contains "recognizer". And finally if we wish to access the colour recognition module we make a check to see if the list of recognized words contains "colour".

A flow chart is included in the next figure, summarizing the process that have been explained:

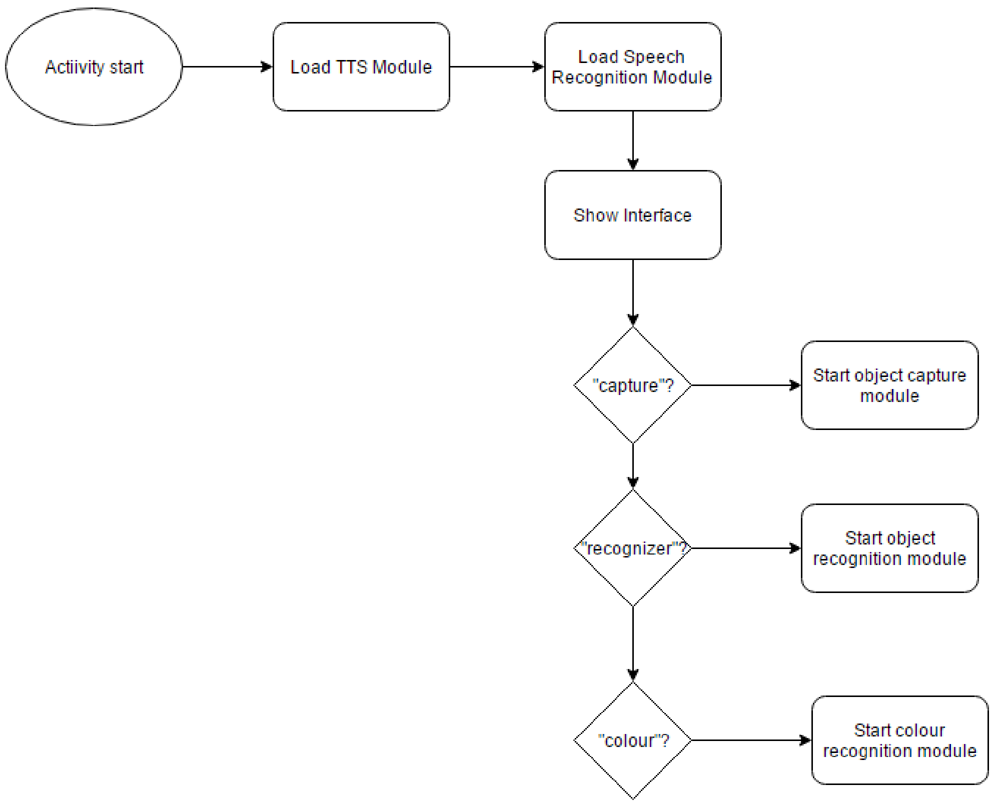
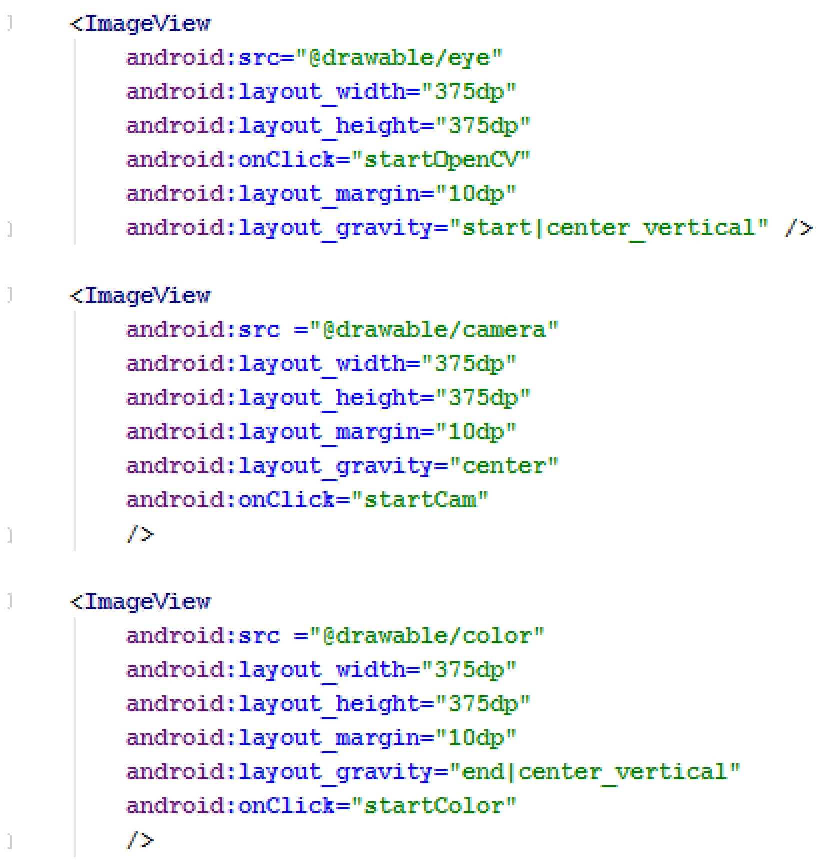

Actiivity start
Load TTS Module
Load Speech Recognition Module
Show Interface
"capture"?
Start object capture module
"recognizer"?
Start object recognition module
"colour"?
Start colour recognition module

Figure 3.14: Flow chart for Main Menu activity functionality

This system is designed to help the visually impaired navigating through the application. Three functioning buttons are added, however, for users without vision impairment. By touching these buttons the system's corresponding module will be launched. These buttons consist of images with a specified method when an *onClick* event occurs. The layout file including these three buttons is shown in the next figure.



```
<ImageView
    android:src="@drawable/eye"
    android:layout_width="375dp"
    android:layout_height="375dp"
    android:onClick="startOpenCV"
    android:layout_margin="10dp"
    android:layout_gravity="start|center_vertical" />

<ImageView
    android:src ="@drawable/camera"
    android:layout_width="375dp"
    android:layout_height="375dp"
    android:layout_margin="10dp"
    android:layout_gravity="center"
    android:onClick="startCam"
    />

<ImageView
    android:src ="@drawable/color"
    android:layout_width="375dp"
    android:layout_height="375dp"
    android:layout_margin="10dp"
    android:layout_gravity="end|center_vertical"
    android:onClick="startColor"
    />
```

Figure 3.15: Main Menu layout configuration

As we can observe in the figure above, three images are added in the *FrameLayout* container, which allows us freedom with their placement. For each image, a *layout_width* and *layout_height* attribute is specified, to handle dimensions on the screen. Their positions are set by adjusting the *layout_gravity* attributed. For the left-most image "*start|center_vertical"* is specified, for the centered image *"center"* is specified and for the right-most image *"end|center_vertical"* is specified. Furthermore, to allow appropriate separation between each image, an amout of *layout_margin* is specified.

Finally, to handle a touch event on each screen and effectively setting them as buttons a different method is specified within the *onClick* attribute which will handle the appropriate change of activity lifecycle for the corresponding button.

3.4. Object capture

When the user chooses to start the object capture module, the corresponding activity will be launched. This activity will receive frames from the camera, thus the user will be able to see what the camera captures on the activity's screen. Additionally, a button is added to capture an object. The idea is that the user places the object which is to be saved in front of the camera and presses the capture button. The interface will then ask the user to write the name of the object so as to store it in the database. In the following figure we show a screen image of the activity:

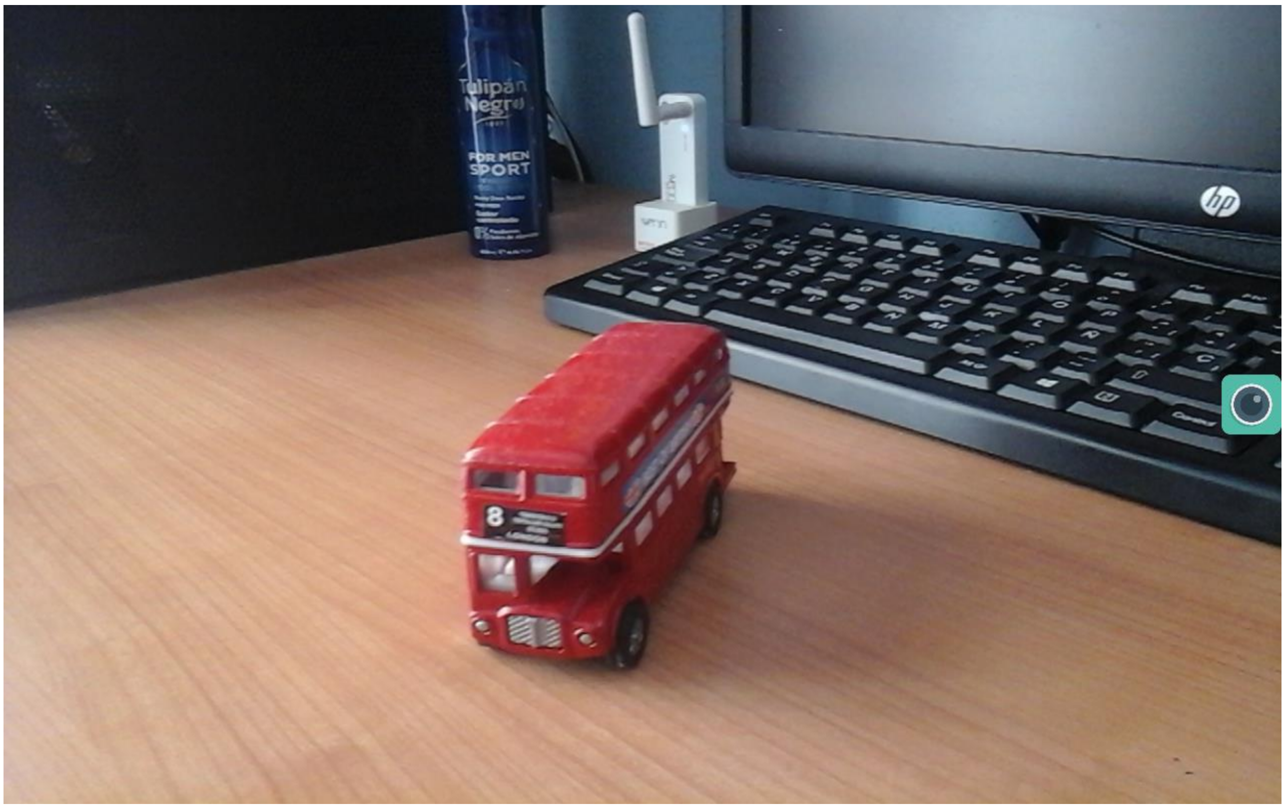
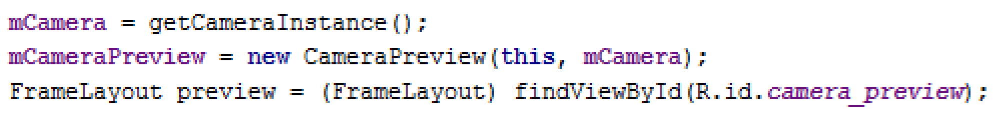


Figure 3.16: Screen image of the object capture activity

This activity works with the Android native camera, thus we must obtain the camera instance and initiate the preview that will be shown on screen. In the following figure the code that configures this process is shown.

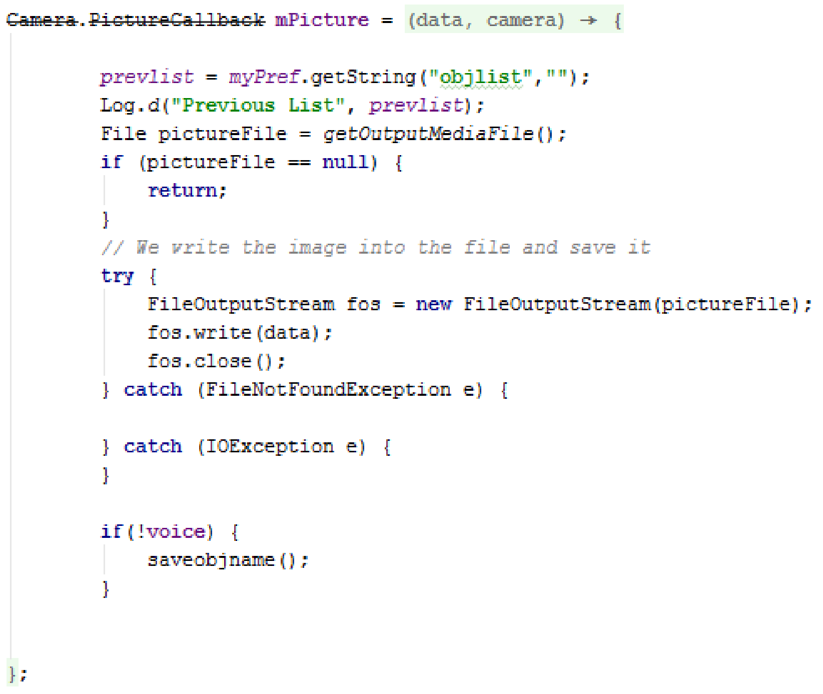


```
mCamera = getCameraInstance();
mCameraPreview = new CameraPreview(this, mCamera);
FrameLayout preview = (FrameLayout) findViewById(R.id.camera_preview);
```

Figure 3.17: Came instance fetching for Object Capture activity

As we can observe in the previous figure, an instance of the Android native camera is obtained by calling to the *getCameraInstance* method and stored within the *Camera* variable *mCamera.* With the instance obtained, we are able to construct the preview in the next line of code. This is what the user will be able to see on the screen. By adding the next line of code, we ensure that the preview is placed within the UI elements of the acitivity.

As with all modules of the system, this activity implements speech recognition. And thus the user with visual impairment is able to capture an object by touching on the screen to activate the speech recognition module and saying the word “capture”. At this point, the application will save the current camera frame by making a method call. In the next figure, this method call is shown.

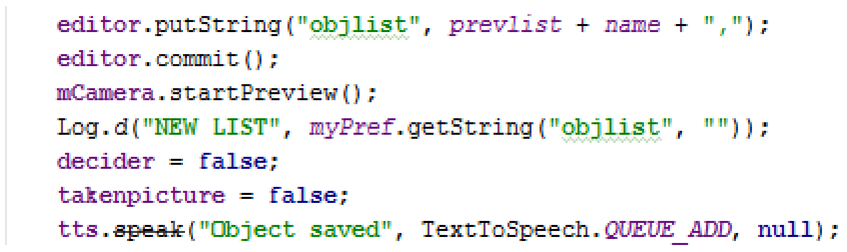


```
~~Camera~~.~~PictureCallback~~ mPicture = (data, camera) → {

        prevlist = myPref.getString("objlist","");
        Log.d("Previous List", prevlist);
        File pictureFile = getOutputMediaFile();
        if (pictureFile == null) {
            return;
        }
        // We write the image into the file and save it
        try {
            FileOutputStream fos = new FileOutputStream(pictureFile);
            fos.write(data);
            fos.close();
        } catch (FileNotFoundException e) {

        } catch (IOException e) {
        }

        if(!voice) {
            saveobjname();
        }


};
```

Figure 3.17: Picture saving on the Object Capture activity

As we can observe in the previous figure, the first step is to load the previous list of saved picture names so that we may add upon it the new picture. Then, by making a call to the *getOutputMediaFile* method, we are able to obtain the file location in the system where we will be able to place the image. Within the try/catch statement an output stream is initiated onto the file directory configured earlier and the data transfer is made. At this point, the image is saved into the device’s memory.

Furthermore, the device will ask the user to say the name of the captured object. After the speech recognition module effectively gets the user’s utterance the *SharedPreferences* are used so that we may save the name for this picture in local storage. In the next figure we can observe the implementation of this code.



```
editor.putString("objlist", prevlist + name + ",");
editor.commit();
mCamera.startPreview();
Log.d("NEW LIST", myPref.getString("objlist", ""));
decider = false;
takenpicture = false;
tts.~~speak~~("Object saved", TextToSpeech.QUEUE_ADD, null);
```

Figure 3.18: Object name list save in Object Capture activity

As mentioned earlier, in the first line of code the new name is added to the previous list of names and by making a call to the *commit* method the data is effectively stored in local. Since Android pauses the camera preview when an image is taken it is necessary to restart it by calling to *startPreview()* on the *mCamera* object. The *decider* and *takenpicture* are boolean variables (they take values of “true” or “false”) used to control the speech dialogue with the user. Last, we use the Text-to-Speech module to make the user know that the object has been saved in the database.

In the next figure, a flow chart of the activity’s functioning is shown.

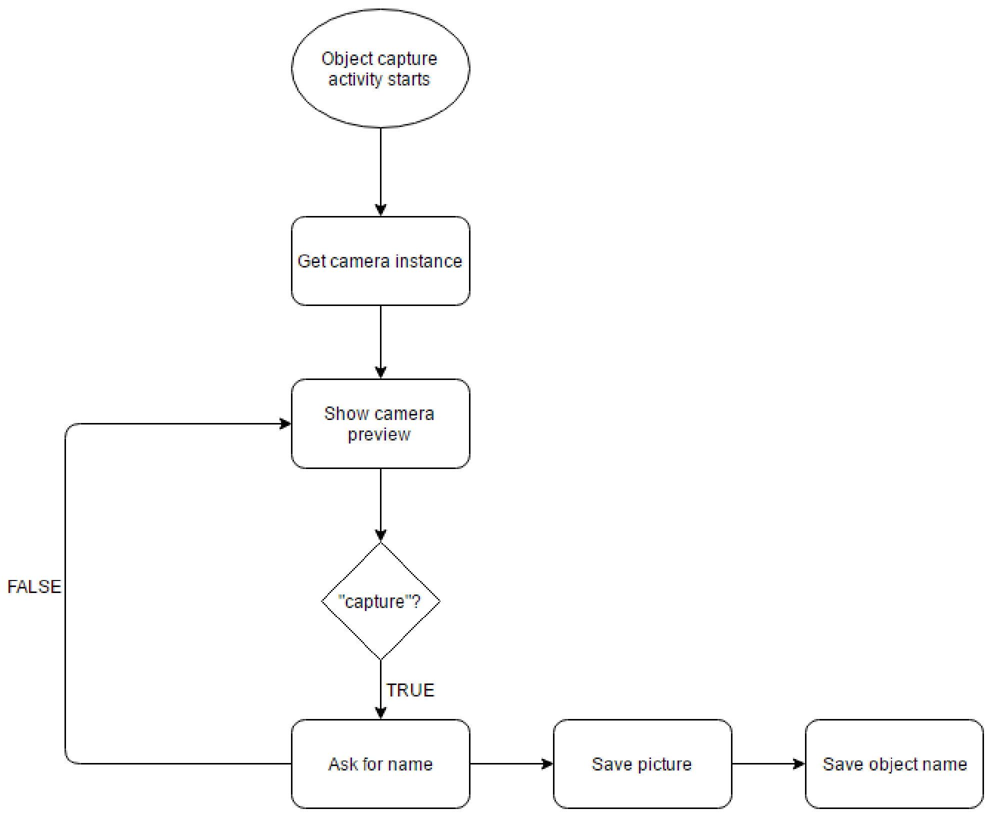


Figure 3.19: Flow chart of Object Capture activity functionality

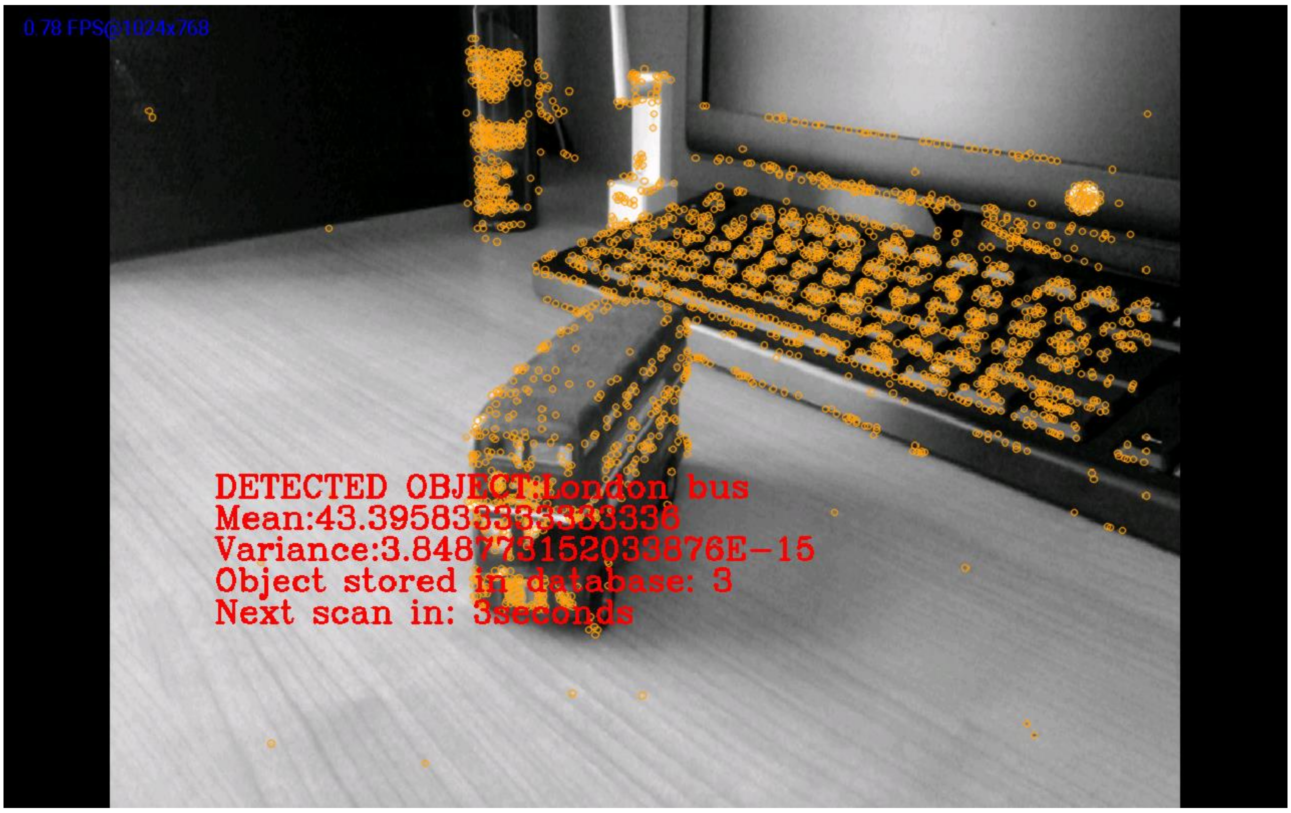
3.5. Object recognizer

The object recognizer activity launches the object recognition module where the system will use the OpenCV library to run algorithms in order to look relevant key points that describe an object stored in the database in real time. The algorithms will first look for details in a scene: borders and corners upon a black and white version of the input camera frames, it will then run filtering mechanisms to find the details which may actually describe an object instead of background information and will then periodically check for similar key point patterns of objects stored in the database.

The module will additionally add information on the screen indicating the name of the recognized object, if no object is detected it will also show the according message. It will also indicate the number of objects stored in the database and as additional information the mean distance and variance of the matches between the object present on the screen and the object the object stored in the database, as a measure of certainty.

Since the scan is periodic, a countdown for the next scan is shown on screen. This countdown will increase as the database increases since the algorithms are expensive on system resource this criteria ensures stability.

In the next figure a screen image of the interface is shown for this module.

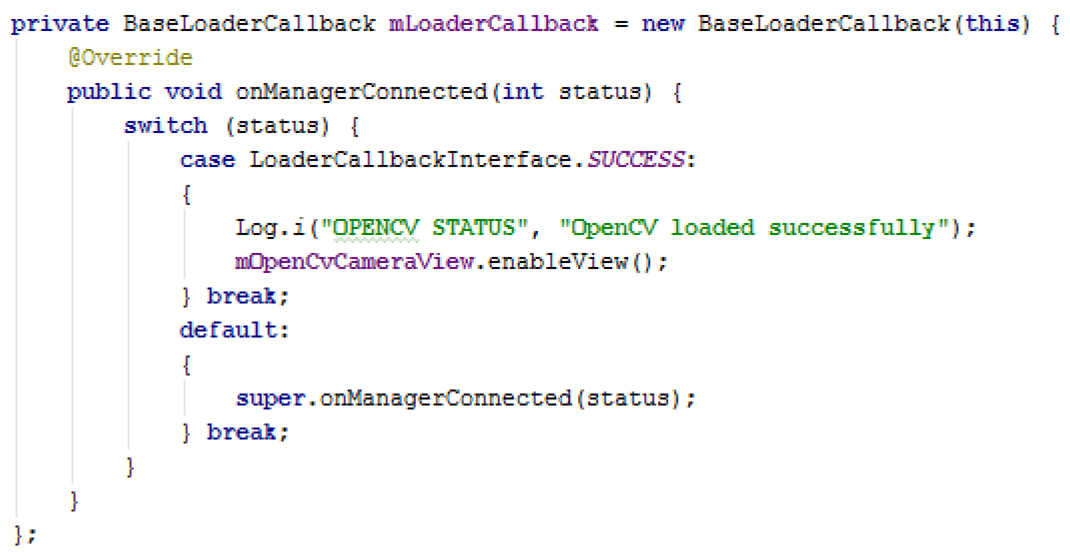


```
0.78 FPS@1024x768
DETECTED OBJECT:London bus
Mean:43.395833333333336
Variance:3.8487731520333876E-15
Object stored in database: 3
Next scan in: 3seconds
```

Figure 3.20: Screen image of the Object Recognizer activity

In order to make use of the OpenCV libraries, it is convenient to load them first. This comes from the fact that the load takes a certain amount of time, and the code might execute faster than the loading process resulting in an error as the called methods and variables are not found.

In the figure below we can observe an implementation of this process.

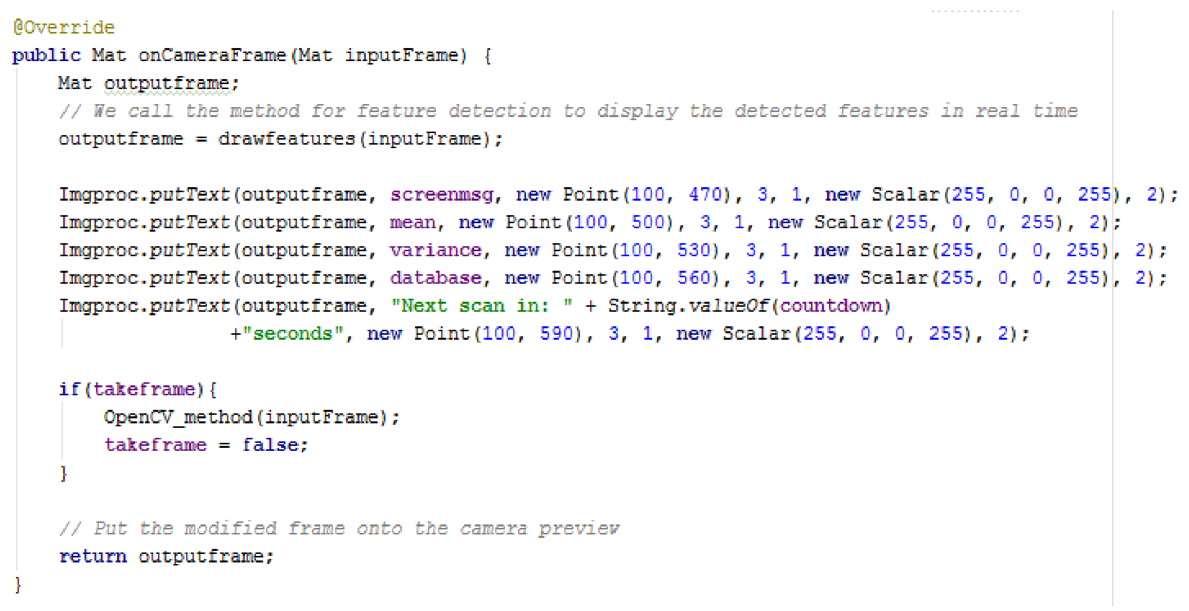


```
private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
    @Override
    public void onManagerConnected(int status) {
        switch (status) {
            case LoaderCallbackInterface.SUCCESS:
            {
                Log.i("OPENCV STATUS", "OpenCV loaded successfully");
                mOpenCvCameraView.enableView();
            } break;
            default:
            {
                super.onManagerConnected(status);
            } break;
        }
    }
};
```

Figure 3.21: OpenCV library load in Object Recognizer activity

As we can observe in the figure, a *BaseLoaderCallback is created,* this will allow us to initialize the OpenCV libraries before the activity code is executed. By overriding the *onManagerConnected* method we can state the functionalities that are to take place when the OpenCV Manager runs. In this case, a *switch* statement is made. This implementation will check if the statement in each state is true and in such a case execute the code held within the statement. In this implementation, the system checks for a success in loading the libraries by the OpenCV manager. It will then make a call to *enableView* to show the activity's interface, showing the live camera feed using OpenCV.

Upon initiation the system will be begin processing each camera frame, it will draw the detected scene features on each frame and the information regarding the detection. This approach will update the features and information at a high rate. In the next figure, we see the implementation of this system.

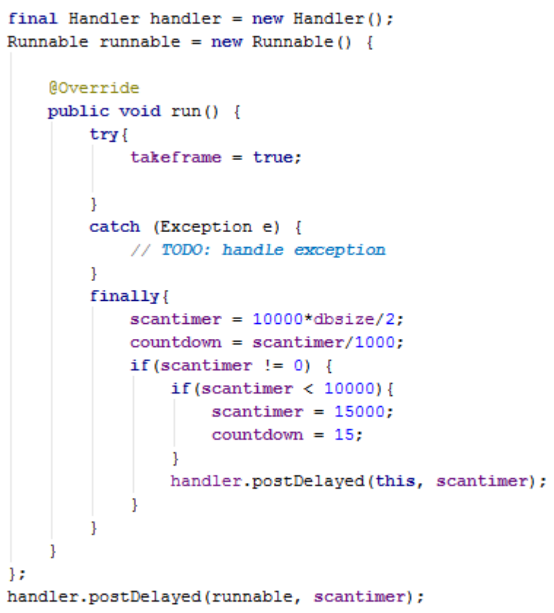


```
@Override
public Mat onCameraFrame(Mat inputFrame) {
    Mat outputframe;
    // We call the method for feature detection to display the detected features in real time
    outputframe = drawfeatures(inputFrame);

    Imgproc.putText(outputframe, screenmsg, new Point(100, 470), 3, 1, new Scalar(255, 0, 0, 255), 2);
    Imgproc.putText(outputframe, mean, new Point(100, 500), 3, 1, new Scalar(255, 0, 0, 255), 2);
    Imgproc.putText(outputframe, variance, new Point(100, 530), 3, 1, new Scalar(255, 0, 0, 255), 2);
    Imgproc.putText(outputframe, database, new Point(100, 560), 3, 1, new Scalar(255, 0, 0, 255), 2);
    Imgproc.putText(outputframe, "Next scan in: " + String.valueOf(countdown)
                +"seconds", new Point(100, 590), 3, 1, new Scalar(255, 0, 0, 255), 2);

    if(takeframe){
        OpenCV_method(inputFrame);
        takeframe = false;
    }

    // Put the modified frame onto the camera preview
    return outputframe;
}
```

Figure 3.22: Camera frame processing in Object Recognizer activity

By overriding the *onCameraFrame* method we are able to alter the frames coming from the camera feed and show the modified frame on the camera preview instead. A call is first made to the *drawfeatures* method which will run the OpenCV algorithms to detect and show the features or details detected in the scene. Next, several lines of code are dedicated to screen representation of relevant information such as the name of the object detected, the mean and variance distance of the detected math and the number of objects stored in the database. Additionally the countdown for the next scan is shown.

Since the detection is done periodically, a boolean *takeframe* is handled to begin the object detection. This Boolean value (true or false) is set by a handler whose function is to activate section of code every set amount of seconds. In the next figure we can observe an implementation of the handler.



```
final Handler handler = new Handler();
Runnable runnable = new Runnable() {

    @Override
    public void run() {
        try{
            takeframe = true;

        }
        catch (Exception e) {
            // TODO: handle exception
        }
        finally{
            scantimer = 10000*dbsize/2;
            countdown = scantimer/1000;
            if(scantimer != 0) {
                if(scantimer < 10000){
                    scantimer = 15000;
                    countdown = 15;
                }
                handler.postDelayed(this, scantimer);
            }
        }
    }
};
handler.postDelayed(runnable, scantimer);
```

Figure 3.23: Handler configuration in Object Recognizer activity

In the figure above we can observe that after the handler's instantiation a *Runnable* is created. This is what the handler will manage. Within the runner we make a call to the *run* method and inside we implement the code that is to be executed. The Boolean value is immediately set to true and later within the *finally* statement the time for the next scan when the boolean will be set to true again is set. In this case, the time for the next scan is set depending on the size of the database. A simple mathematical operation is then done and an '*if*' statement ensures that the timer is never too small. Last, the handler sets the time for the next execution of this runnable.

Nevertheless, a call is made to the *drawfeatures* on every camera frame to update the detected features in real time. In the next figure the implementation of the *drawfeatures* method is shown.

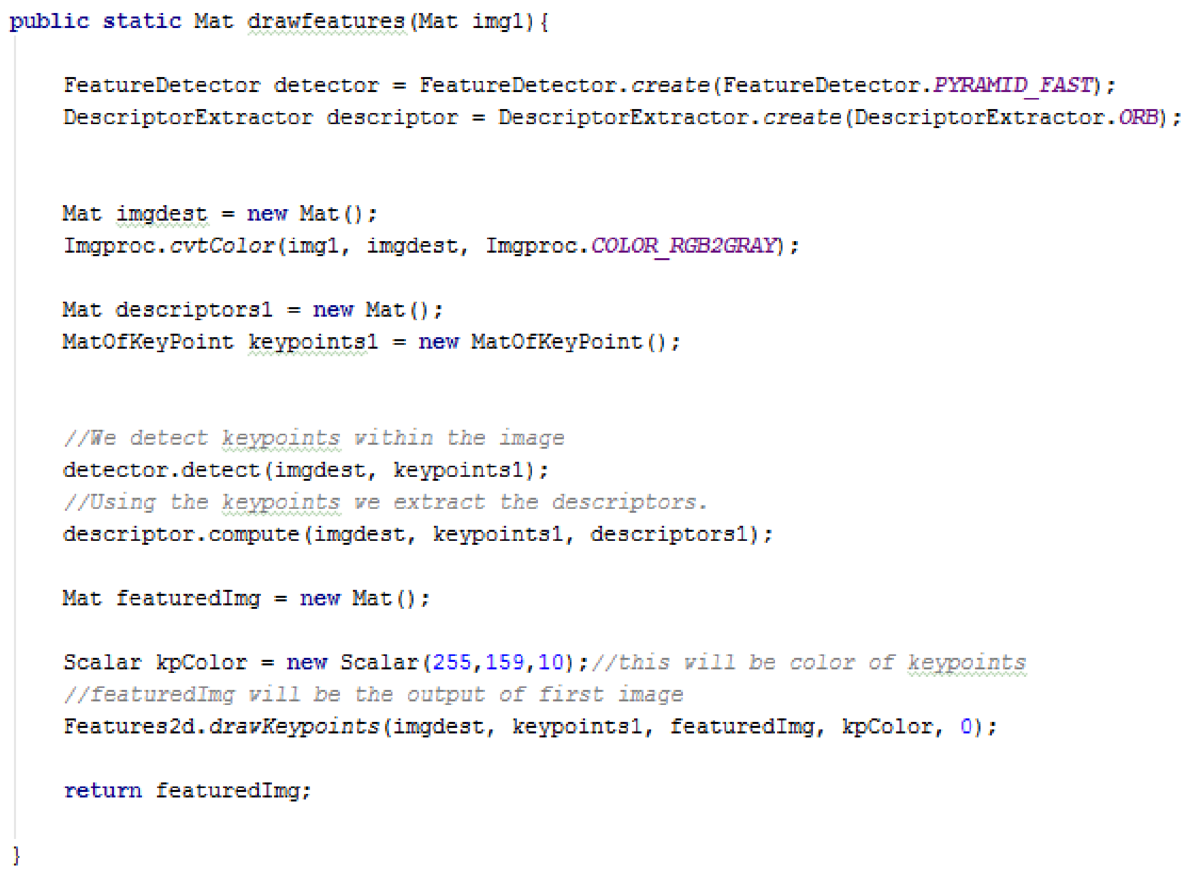

```java
public static Mat drawfeatures(Mat img1){

    FeatureDetector detector = FeatureDetector.create(FeatureDetector.PYRAMID_FAST);
    DescriptorExtractor descriptor = DescriptorExtractor.create(DescriptorExtractor.ORB);


    Mat imgdest = new Mat();
    Imgproc.cvtColor(img1, imgdest, Imgproc.COLOR_RGB2GRAY);

    Mat descriptors1 = new Mat();
    MatOfKeyPoint keypoints1 = new MatOfKeyPoint();


    //We detect keypoints within the image
    detector.detect(imgdest, keypoints1);
    //Using the keypoints we extract the descriptors.
    descriptor.compute(imgdest, keypoints1, descriptors1);

    Mat featuredImg = new Mat();

    Scalar kpColor = new Scalar(255,159,10);//this will be color of keypoints
    //featuredImg will be the output of first image
    Features2d.drawKeypoints(imgdest, keypoints1, featuredImg, kpColor, 0);

    return featuredImg;

}
```

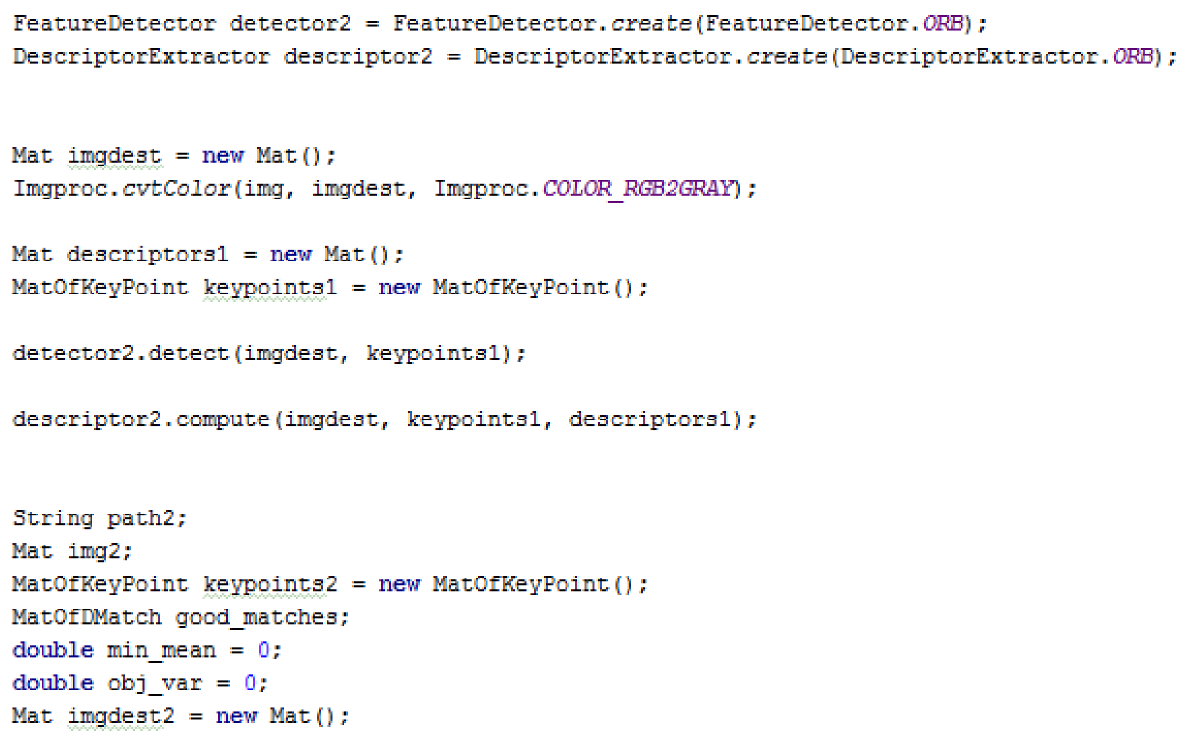
Figure 3.24: Draw features on camera frames in Object Recognizer activity

This method receives the image as argument in the form of a *Mat* object which is OpenCV's data matrix representation of digital images. First, a call is made to OpenCV library methods to create the feature detector and descriptor extractor. The feature detector is in charge of finding borders and corners which may define the details in a relevant object. The descriptor extractor will use the detected features to define the most relevant detected details that define the object.

Next, the image is converted to a grey colour space. This makes the detection of borders and corners easier for the algorithm. A call is made to the *detect* method to get the localization of the key points and to store them in the *MatOfKeyPoint* variable *keypoints1.* These key points are then used to create the descriptors by making a call to the *compute* method in the *descriptor interface*. Finally, by calling the *drawKeypoints* OpenCV method within the *Features2D* interface module and passing as arguments the image to be modified, the detected key points, the destination image and the colour of the key points we will get the feature drawn image.

When a scan is to be made, the application will take a still image from the live camera feed and compare it will the other still images saved in the database, looking for the closest match. The closest match is produced when the mean distance between the detected features is the smallest. When such event occurs the algorithm will fetch for the name of the object corresponding to the still image and communicate it to the user by the use of speech and on screen representation.

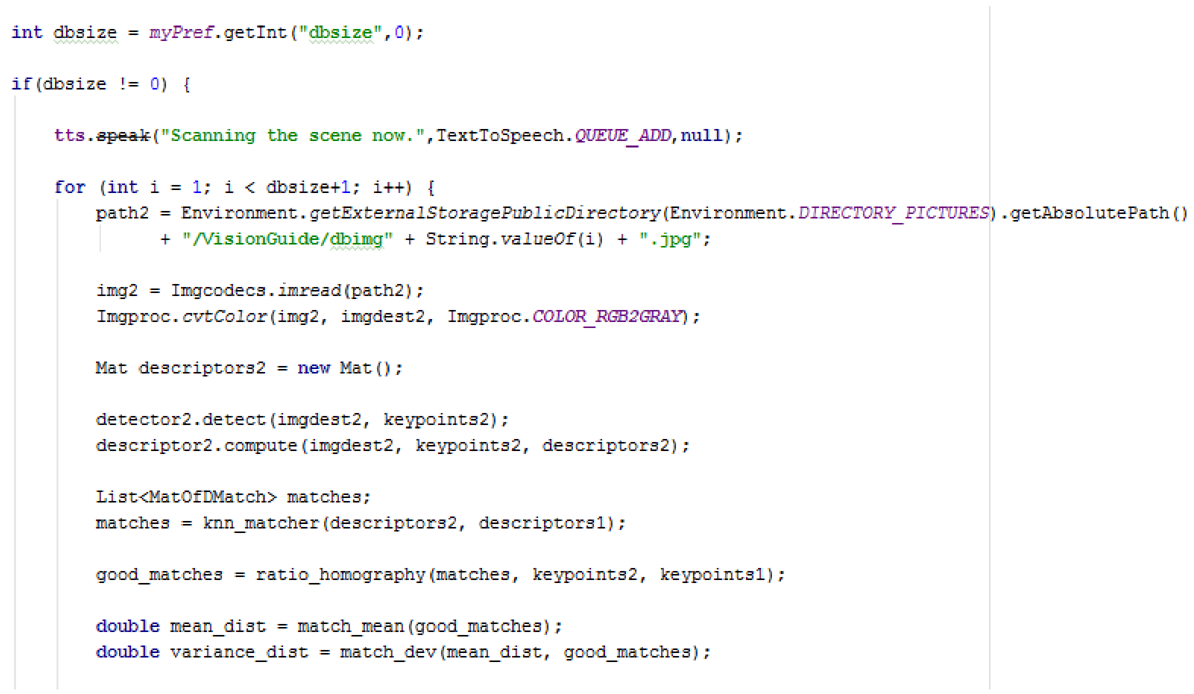
In the figure below the implementation of this functionality can be observed.



```
FeatureDetector detector2 = FeatureDetector.create(FeatureDetector.ORB);
DescriptorExtractor descriptor2 = DescriptorExtractor.create(DescriptorExtractor.ORB);


Mat imgdest = new Mat();
Imgproc.cvtColor(img, imgdest, Imgproc.COLOR_RGB2GRAY);

Mat descriptors1 = new Mat();
MatOfKeyPoint keypoints1 = new MatOfKeyPoint();

detector2.detect(imgdest, keypoints1);

descriptor2.compute(imgdest, keypoints1, descriptors1);


String path2;
Mat img2;
MatOfKeyPoint keypoints2 = new MatOfKeyPoint();
MatOfDMatch good_matches;
double min_mean = 0;
double obj_var = 0;
Mat imgdest2 = new Mat();
```

Figure 3.25: Detected key points in database images in Object Recognizer activity

As we can observe the first step is to detect features in a similar way it was done for drawing features on each frame with the exception that in this implementation it is done on a single camera frame when the scan is made. Now, a second image *img2* will be scanned for features corresponding to the image stored in the database.



```
int dbsize = myPref.getInt("dbsize",0);

if(dbsize != 0) {

    tts.~~speak~~("Scanning the scene now.",TextToSpeech.QUEUE_ADD,null);

    for (int i = 1; i < dbsize+1; i++) {
        path2 = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES).getAbsolutePath()
                + "/VisionGuide/dbimg" + String.valueOf(i) + ".jpg";

        img2 = Imgcodecs.imread(path2);
        Imgproc.cvtColor(img2, imgdest2, Imgproc.COLOR_RGB2GRAY);

        Mat descriptors2 = new Mat();

        detector2.detect(imgdest2, keypoints2);
        descriptor2.compute(imgdest2, keypoints2, descriptors2);

        List<MatOfDMatch> matches;
        matches = knn_matcher(descriptors2, descriptors1);

        good_matches = ratio_homography(matches, keypoints2, keypoints1);

        double mean_dist = match_mean(good_matches);
        double variance_dist = match_dev(mean_dist, good_matches);
```

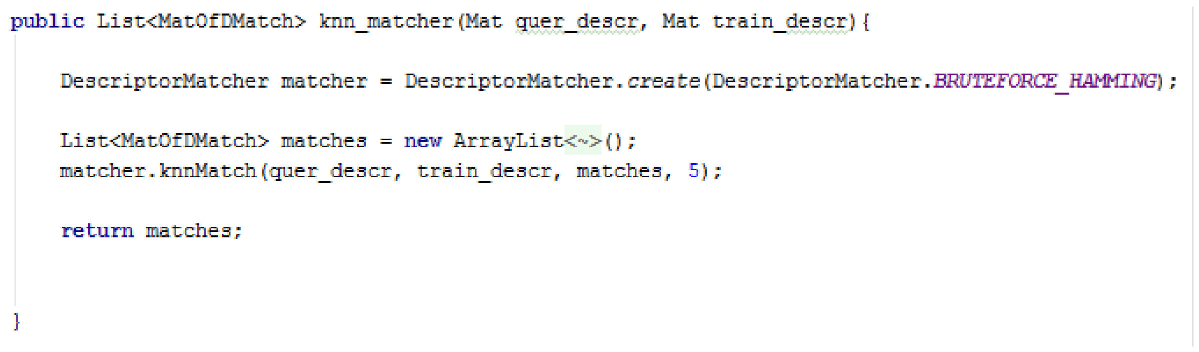
Figure 3.26: Searching the database for matches in Object Recognizer activity

We can observe in the figure above how the database scan is done. First we find within the local store the size of the database, a value increased and stored each time a new image is added. Next, if the database has images we proceed to scan for each image. Within the database, each image is stored with the name *dbimg* and an identifier number, which for convenient simplicity it correspond to its order withint he database. Thus, the next thing that is done is the path towards the stored image is constructed. The image is stored within the *Pictures* directory within the local storage. By calling to the method *getExternalStoragePublicDirectory* and passing as argument the directory we wish to obtain the method will return a path towards the folder. We then add the subfolder the image is located in, its database name and identifier number which is increased on every new iteration.

The next steps are to get the features and descriptors for the database image corresponding to the current iteration and then find matches with the features of the image coming from the live camera feed. First, a list of matches object is declared and a method call is made to *knn_matcher* to find the most optimal matches between the descriptors of both images. Then, the matches are filtered by making a call to the *ratio_homography* method which does a ratio test and a *RANSAC* test to find the most relevant matches, removing noise data.

The mean distance between the matches is calculated by making a call to the *match_mean* method and the variance of the distance between all matches by calling to the *match_dev* method.

Before we continue with the ‘for’ loop, it is convenient to study the aforementioned methods in detail. The first, shown in the next figure, will be the *knn_matcher* method.

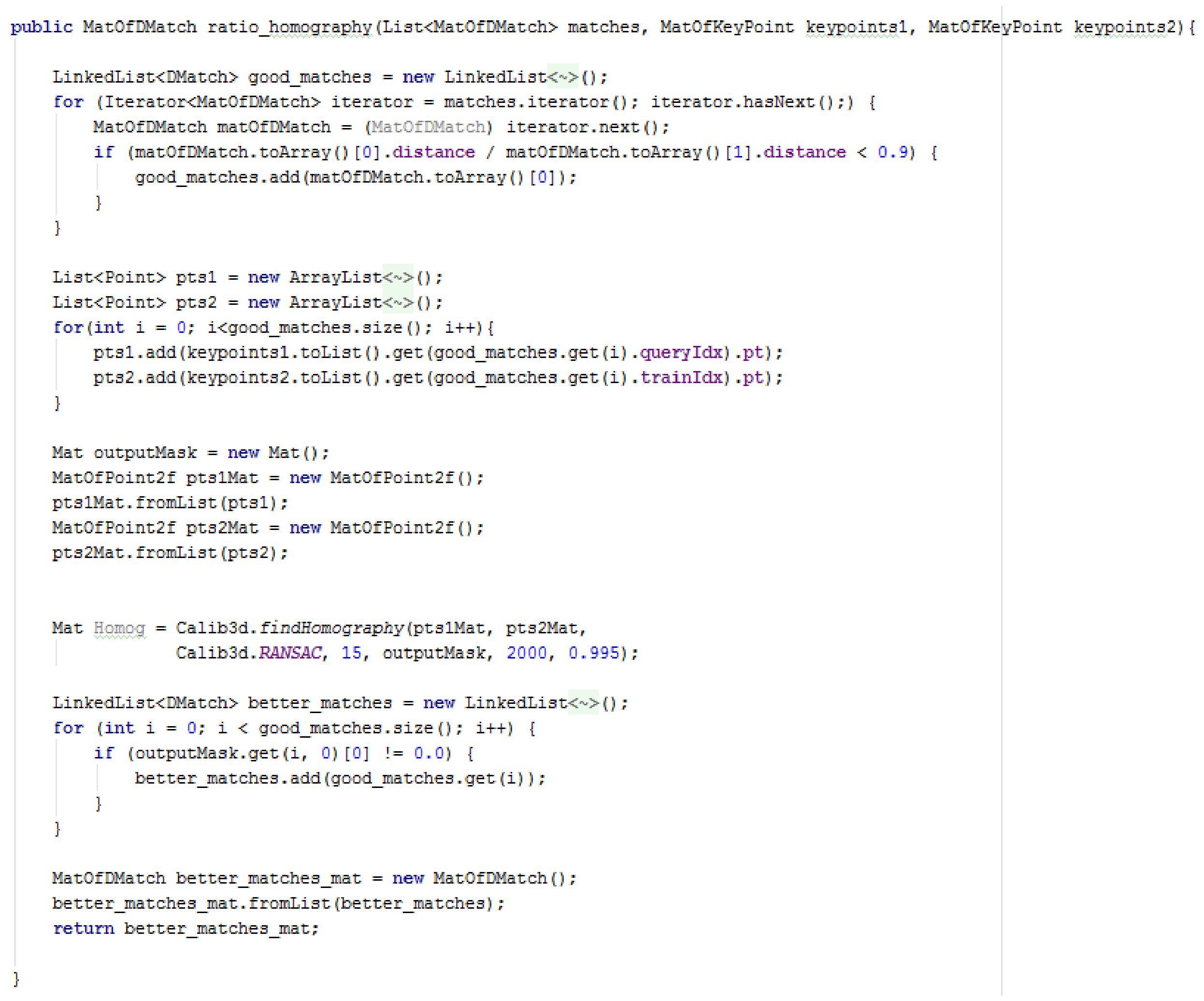


```
public List<MatOfDMatch> knn_matcher(Mat quer_descr, Mat train_descr){

    DescriptorMatcher matcher = DescriptorMatcher.create(DescriptorMatcher.BRUTEFORCE_HAMMING);

    List<MatOfDMatch> matches = new ArrayList<~>();
    matcher.knnMatch(quer_descr, train_descr, matches, 5);

    return matches;


}
```

Figure 3.27: Using the KNN algorithm to find matches in Object Recognizer activity

In the figure above we can observe how the *knn_matcher* is implemented. The descriptors corresponding to both images is passed as argument. Next the *DescriporMatcher* is created from the OpenCV library. This object will be in charge of finding the closest matches using bruteforce hamming (simple coordinate distance comparison). By making a call to the *knnMatch* OpenCV method, it will compute the most optimal matches by filtering with the K-Nearest-Neighbours algorithm, giving a greater relevance to nearby points than isolated coordinates.

Now, as shown in the next figure, we will study in detail the ratio and *RANSAC* test.

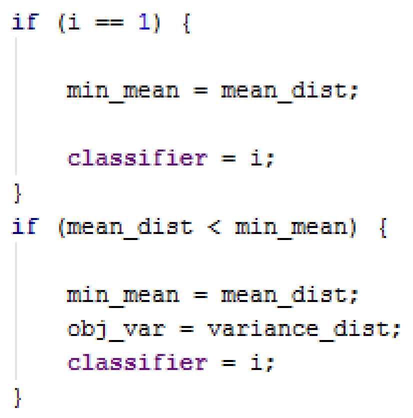


```java
public MatOfDMatch ratio_homography(List<MatOfDMatch> matches, MatOfKeyPoint keypoints1, MatOfKeyPoint keypoints2){

    LinkedList<DMatch> good_matches = new LinkedList<~>();
    for (Iterator<MatOfDMatch> iterator = matches.iterator(); iterator.hasNext();) {
        MatOfDMatch matOfDMatch = (MatOfDMatch) iterator.next();
        if (matOfDMatch.toArray()[0].distance / matOfDMatch.toArray()[1].distance < 0.9) {
            good_matches.add(matOfDMatch.toArray()[0]);
        }
    }

    List<Point> pts1 = new ArrayList<~>();
    List<Point> pts2 = new ArrayList<~>();
    for(int i = 0; i<good_matches.size(); i++){
        pts1.add(keypoints1.toList().get(good_matches.get(i).queryIdx).pt);
        pts2.add(keypoints2.toList().get(good_matches.get(i).trainIdx).pt);
    }

    Mat outputMask = new Mat();
    MatOfPoint2f pts1Mat = new MatOfPoint2f();
    pts1Mat.fromList(pts1);
    MatOfPoint2f pts2Mat = new MatOfPoint2f();
    pts2Mat.fromList(pts2);


    Mat Homog = Calib3d.findHomography(pts1Mat, pts2Mat,
            Calib3d.RANSAC, 15, outputMask, 2000, 0.995);

    LinkedList<DMatch> better_matches = new LinkedList<~>();
    for (int i = 0; i < good_matches.size(); i++) {
        if (outputMask.get(i, 0)[0] != 0.0) {
            better_matches.add(good_matches.get(i));
        }
    }

    MatOfDMatch better_matches_mat = new MatOfDMatch();
    better_matches_mat.fromList(better_matches);
    return better_matches_mat;

}
```

Figure 3.28: Filtering matches using ratio test and RANSAC algorithm in Object Recognizer activity

As it can be observed in the previous figure, an iterator is used to go through the matches list and find patterns using the ratio test for relevant key points. Each key point is compared with the next by division, if the result is smaller than 0.9, it is considered a relevant match and it is added to a new list of good matches. Then the *RANSAC* algorithm is used which filters the points according to perspective changes.

In the next figure, returning to the for loop iteration, it is shown how the minimum mean distance, corresponding to the recognized object is determined.

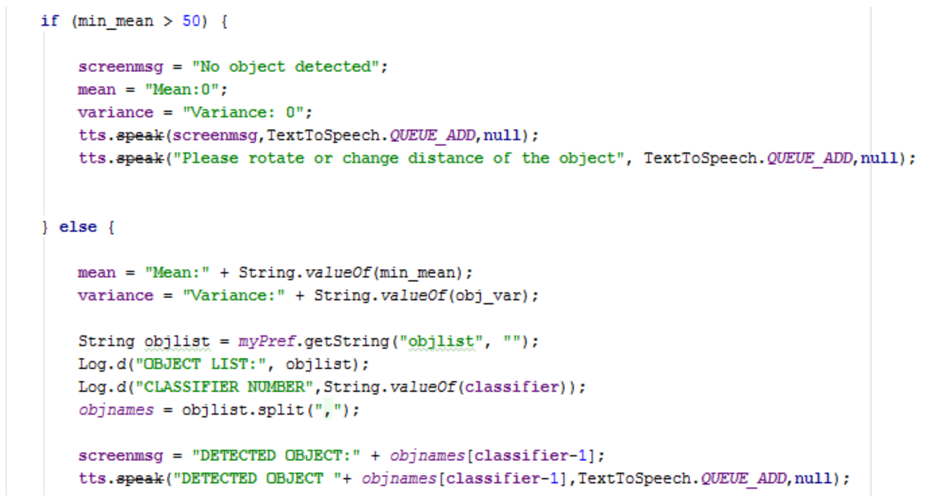


```
if (i == 1) {

    min_mean = mean_dist;

    classifier = i;
}
if (mean_dist < min_mean) {

    min_mean = mean_dist;
    obj_var = variance_dist;
    classifier = i;
}
```

Figure 3.29: Finding the minimum mean distance in the Object Recognizer activity

The first '*If'* statement checks if we are in the first iteration, in such a case the minimum distance is set to the first scanned distance. Then, the next *'if'* statement checks if there is another scanned distance smaller than our stored minimum distance, in such a case the minimum distance is set to that distance and the classifier variable is set to the iteration number. This will allow us to locate which image in the database contains the smallest distance since they are saved with their order number.

In the next figure, we exit the *'for'* loop and the detection process is shown.



```
if (min_mean > 50) {

    screenmsg = "No object detected";
    mean = "Mean:0";
    variance = "Variance: 0";
    tts.~~speak~~(screenmsg,TextToSpeech.QUEUE_ADD,null);
    tts.~~speak~~("Please rotate or change distance of the object", TextToSpeech.QUEUE_ADD,null);


} else {

    mean = "Mean:" + String.valueOf(min_mean);
    variance = "Variance:" + String.valueOf(obj_var);

    String objlist = myPref.getString("objlist", "");
    Log.d("OBJECT LIST:", objlist);
    Log.d("CLASSIFIER NUMBER",String.valueOf(classifier));
    objnames = objlist.split(",");

    screenmsg = "DETECTED OBJECT:" + objnames[classifier-1];
    tts.~~speak~~("DETECTED OBJECT "+ objnames[classifier-1],TextToSpeech.QUEUE_ADD,null);
```

Figure 3.27: Finding the recognized object's name in the Object Recognizer activity

If we refer to the figure above, first a check is made to see if the minimum mean distance is below a threshold. If the threshold is exceeded we deduce no relevant detection was made, this is done to avoid false detections. In the other case however, we conclude that there may be a relevant object within the camera scene and the minimum mean distance and minimum variance is shown on screen. Furthermore, the name of the object is fetched within the list of names stored in local storage, and the exact position of the detected object is found within the list to extract its name. The name is then put on screen and said to the user by speech using the text-to-speech module. This name corresponds to that which the user inserted when the object was captured.

As a summary, the complete functioning of this activity's algorithm is shown in the next flow chart figure.

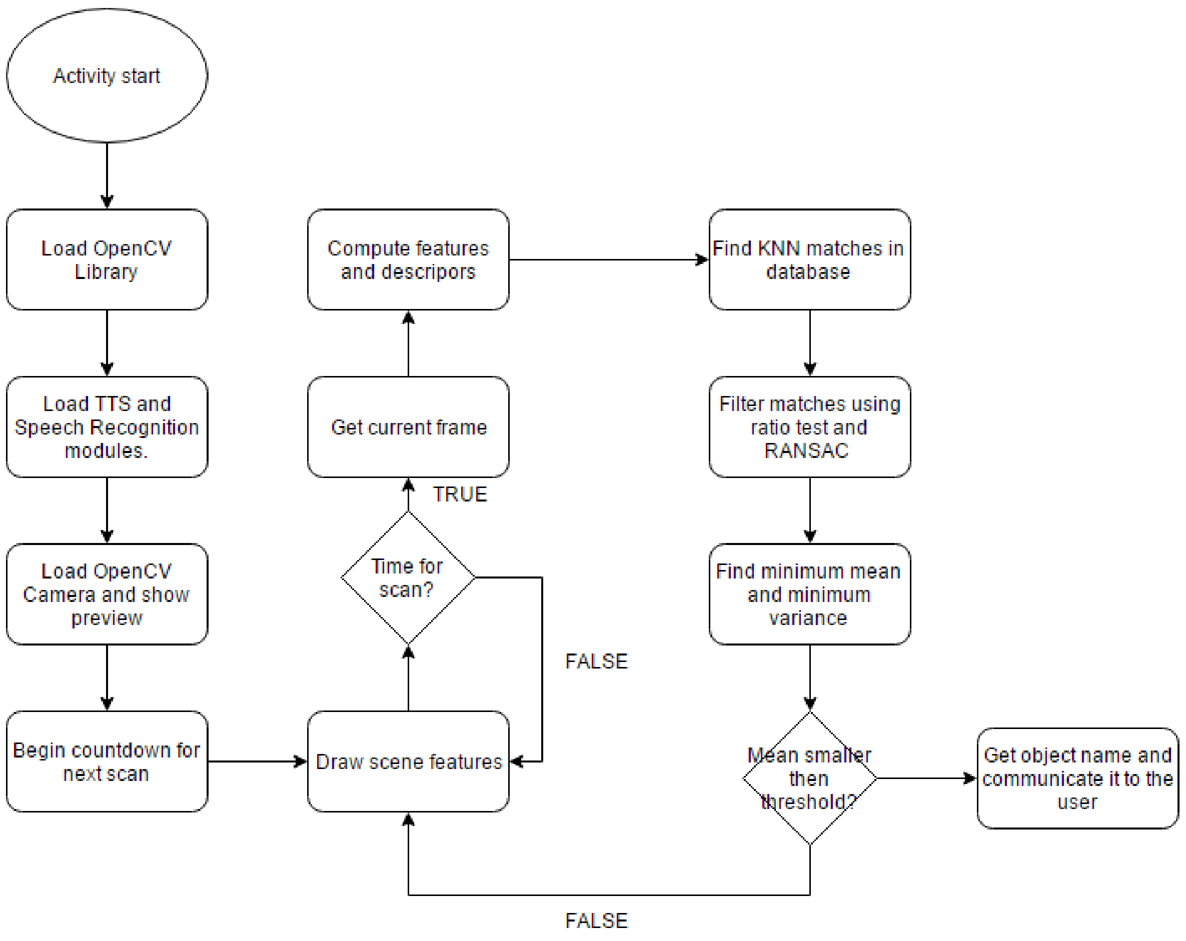
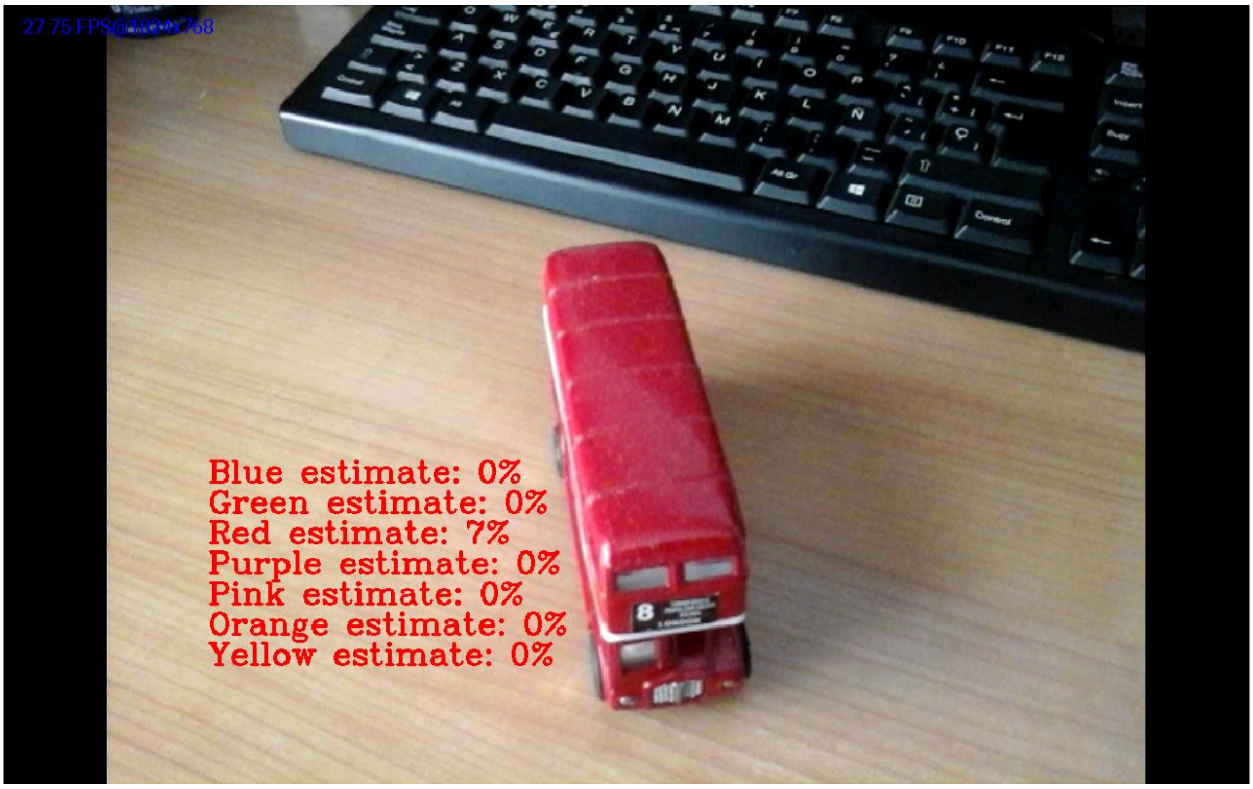

Activity start
Load OpenCV Library
Load TTS and Speech Recognition modules.
Load OpenCV Camera and show preview
Begin countdown for next scan
Draw scene features
Time for scan?
TRUE
FALSE
Get current frame
Compute features and descripors
Find KNN matches in database
Filter matches using ratio test and RANSAC
Find minimum mean and minimum variance
Mean smaller then threshold?
Get object name and communicate it to the user
FALSE

Figure 3.30: Flow chat for the Object Recognizer activity functionality

3.6. Colour recognizer

The colour recognition activity's goal is to be able to inform the user of the colours present in a determined scene. On screen information will show the user the percentage of that colour present in the scene according to the total number of pixels present in that HSV range. The user may communicate to the application which colour is to be scanned and flowingly the application will check and convert the pixels present in that HSV range to white and the pixels not in the HSV range to black. With this sudden high level of contrast it is intended for the users with low-vision to be able to tell which object or scene is present in the camera scene or for users with difficulties telling colours to be receive the colour information they require. In the next figure, the main interface is shown.

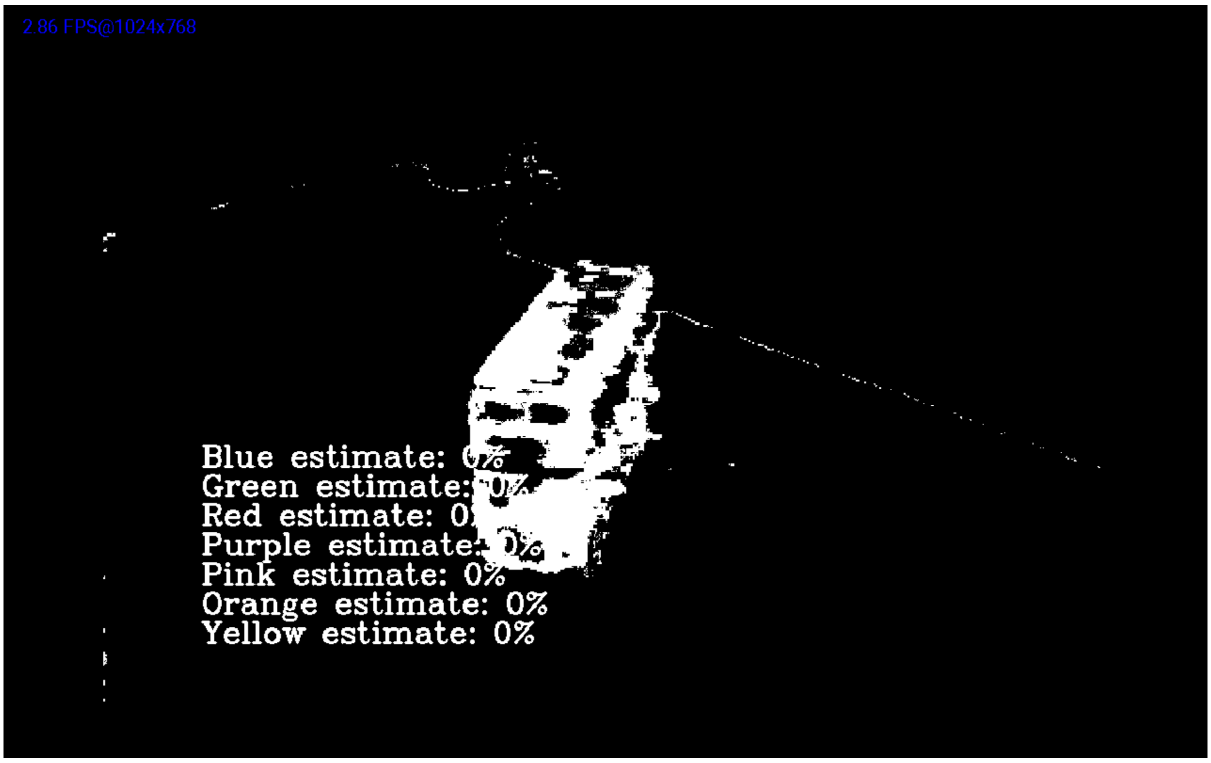


```
27.75 FPS@1024x768
```

```
Blue estimate: 0%
Green estimate: 0%
Red estimate: 7%
Purple estimate: 0%
Pink estimate: 0%
Orange estimate: 0%
Yellow estimate: 0%
```

Figure 3.31: Screen image of the Colour Recognition activity

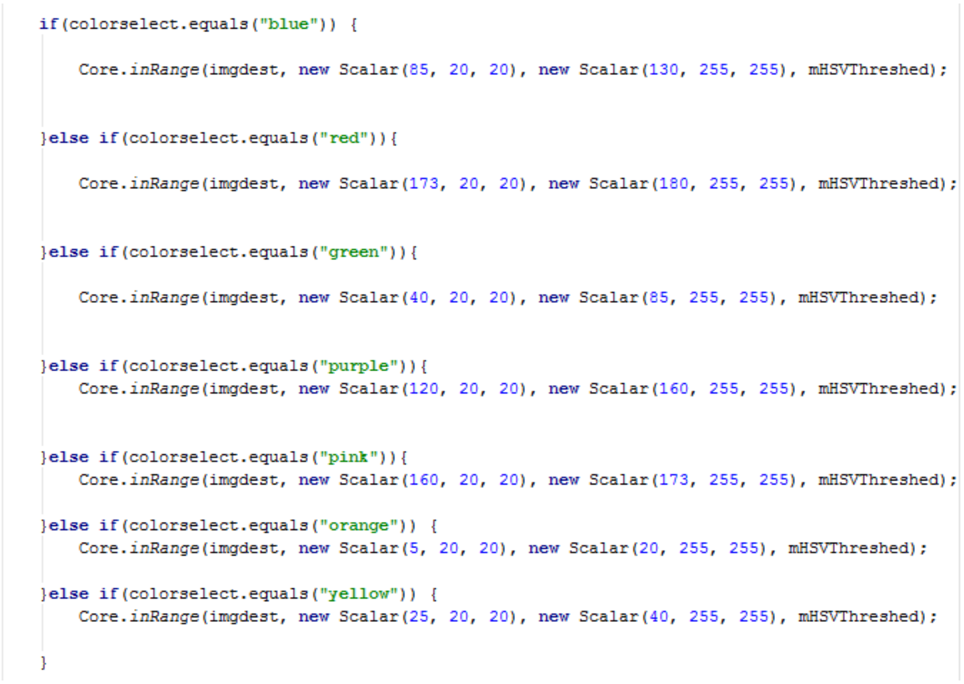
And in the next figure a screen capture is shown when the algorithm scans for a colour.



```
2.86 FPS@1024x768

Blue estimate: 0%
Green estimate: 0%
Red estimate: 0%
Purple estimate: 0%
Pink estimate: 0%
Orange estimate: 0%
Yellow estimate: 0%
```

Figure 3.32: Screen image of the Color Recognizer activity when scanning.

To detect colour, the system will use speech recognition to listen to what the user wishes to scan. At that moment, the appropriate method will be called which uses OpenCV's libraries to read the image pixels in a fast and efficient manner and determine if they are within the HSV range. The next figure shows the implementation of this functionality.



```java
if(colorselect.equals("blue")) {

    Core.inRange(imgdest, new Scalar(85, 20, 20), new Scalar(130, 255, 255), mHSVThreshed);


}else if(colorselect.equals("red")){

    Core.inRange(imgdest, new Scalar(173, 20, 20), new Scalar(180, 255, 255), mHSVThreshed);


}else if(colorselect.equals("green")){

    Core.inRange(imgdest, new Scalar(40, 20, 20), new Scalar(85, 255, 255), mHSVThreshed);


}else if(colorselect.equals("purple")){
    Core.inRange(imgdest, new Scalar(120, 20, 20), new Scalar(160, 255, 255), mHSVThreshed);


}else if(colorselect.equals("pink")){
    Core.inRange(imgdest, new Scalar(160, 20, 20), new Scalar(173, 255, 255), mHSVThreshed);

}else if(colorselect.equals("orange")) {
    Core.inRange(imgdest, new Scalar(5, 20, 20), new Scalar(20, 255, 255), mHSVThreshed);

}else if(colorselect.equals("yellow")) {
    Core.inRange(imgdest, new Scalar(25, 20, 20), new Scalar(40, 255, 255), mHSVThreshed);

}
```

Figure 3.33: Finding colours in the Colour Recognition activity

As it can be observed in the previous figure the camera frame is passed as input argument, similar to the object recognition module and additionally the colour the user requires, communicated by speech. This information is used to run multiple *'if'* statements and accessing their inner code depending on the selected colour. In all cases, however a call to the core method of the OpenCV library *inRange* is made, whereas arguments the input image is set, the HSV range to be scanned and the destination image. This OpenCV method will single-handedly convert the pixels present in the defined HSV range to white and the others to absolute black.

As a summary of the functioning of this activity, the flow chart is shown in the next figure.

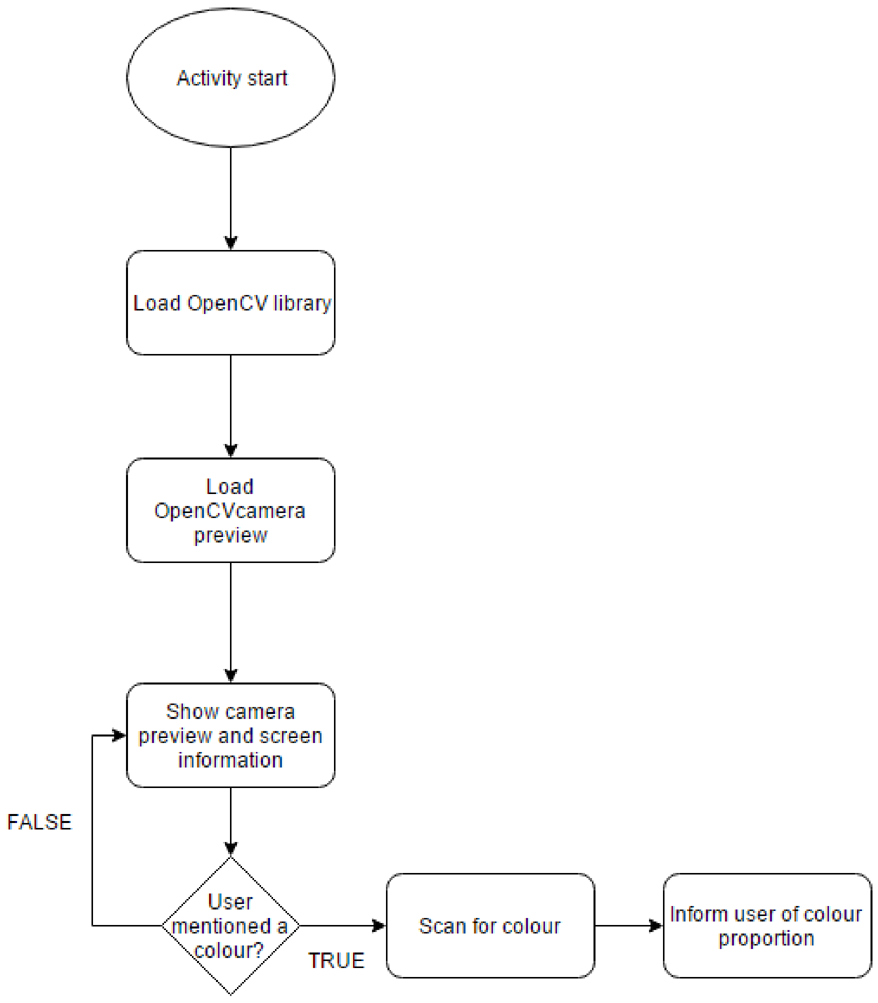

Activity start
Load OpenCV library
Load OpenCVcamera preview
Show camera preview and screen information
FALSE
User mentioned a colour?
TRUE
Scan for colour
Inform user of colour proportion

Figure 3.34: Flow chat for the Colour Recognition activity

Chapter 4

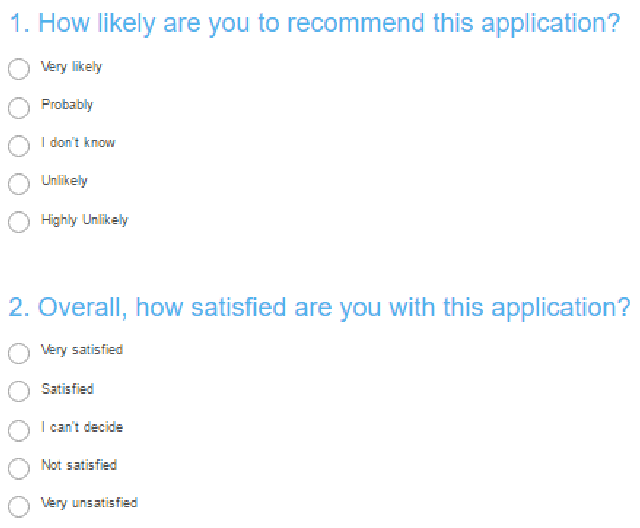
SYSTEM EVALUATION

4.1 System evaluation methodology and results

For the evaluation of the present application several candidates were carefully chosen since the nature of the developed system's functionalities may only be useful to the visually impaired and other individuals who wish to aid them by making use of this application. The Android device was handed out to the participants with the application installed and the tests were made in varying environments in an attempt to imitate daily use.

After a week of testing, the users were given a questionnaire with questions designed to evaluate the system's quality and usefulness. The results were taken and several statistics were made. The total number of participants were ten and consisted of a mixed group of both visually and non-visually impaired individuals.

The first set of questions are designed to gather information regarding the application's general functioning. In the next figures the first and second questions and their results are shown.



1. How likely are you to recommend this application?

○ Very likely
○ Probably
○ I don't know
○ Unlikely
○ Highly Unlikely

2. Overall, how satisfied are you with this application?

○ Very satisfied
○ Satisfied
○ I can't decide
○ Not satisfied
○ Very unsatisfied

Figure 4.1: First question pairs of the system evaluation survey

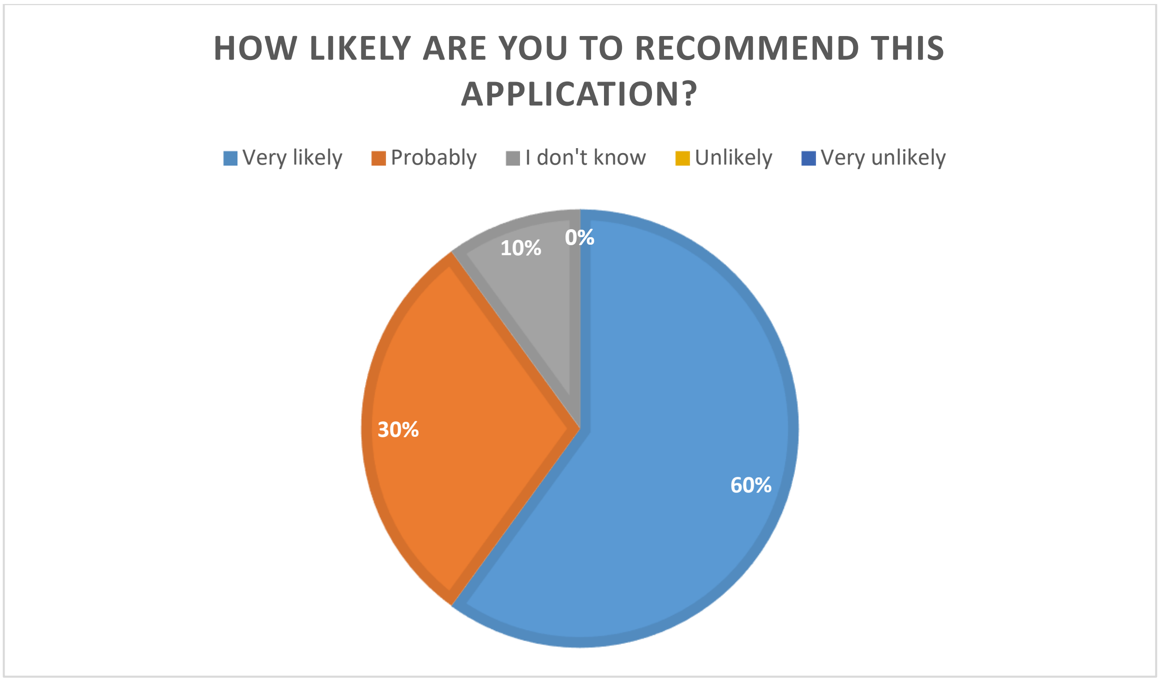


Figure 4.2: First question results

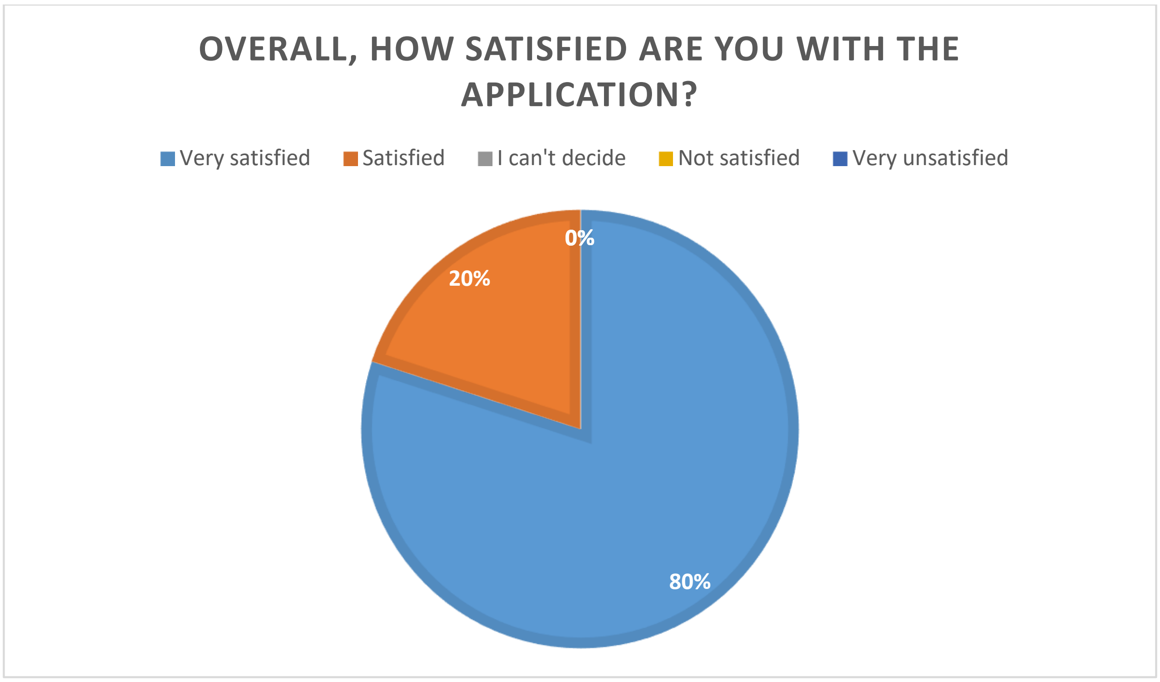
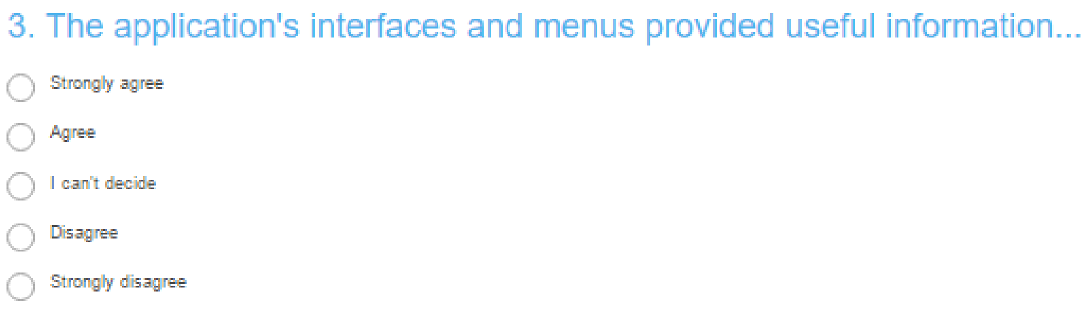


Figure 4.3: Second question results



3. The application's interfaces and menus provided useful information...

- ◯ Strongly agree
- ◯ Agree
- ◯ I can't decide
- ◯ Disagree
- ◯ Strongly disagree

Figure 4.4: Third question of the system evaluation survey

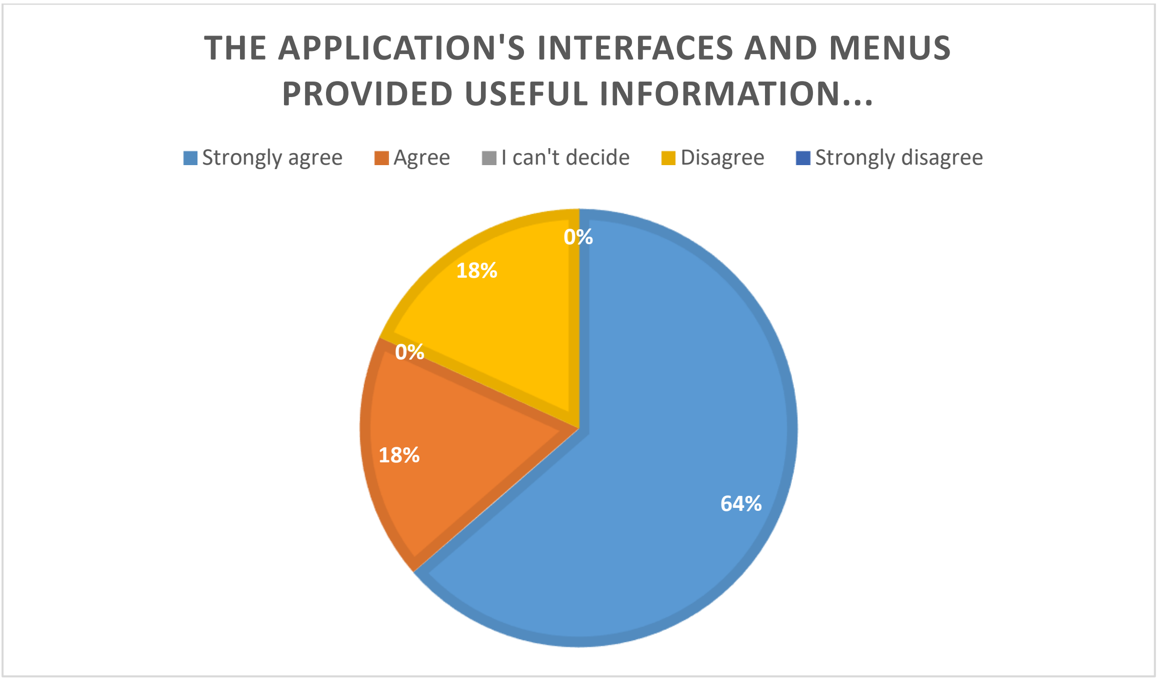
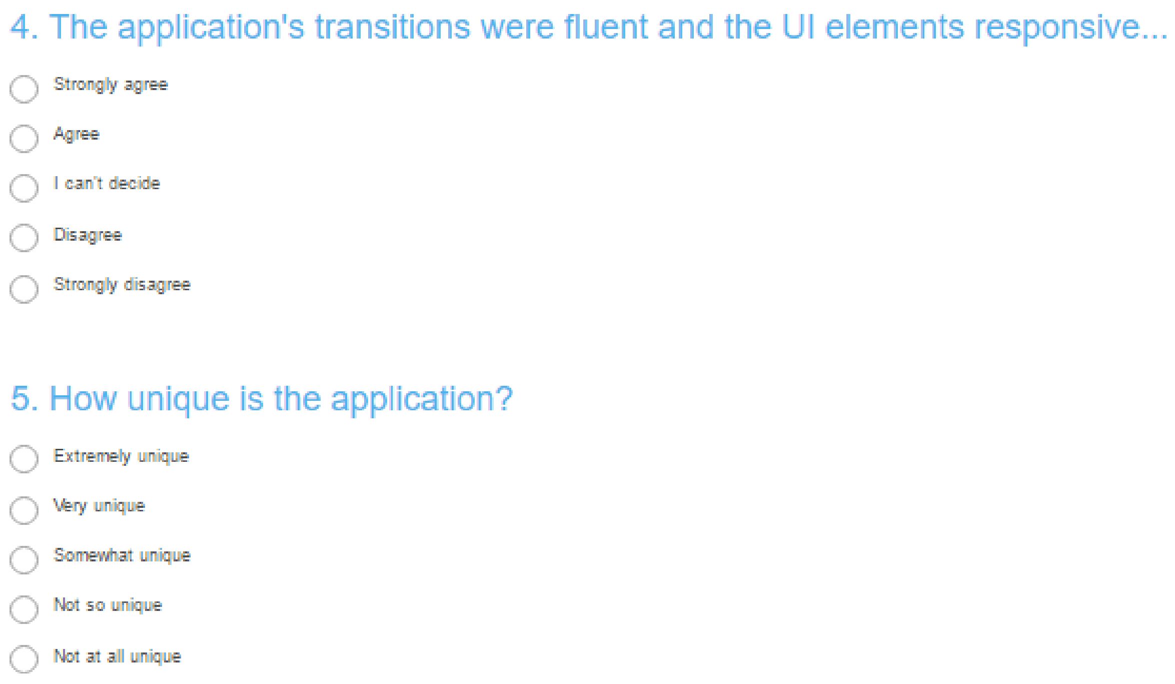


Figure 4.5: Results for the third question

In the next figures, the next pair of questions regarding general system satisfaction are shown and their results:



4. The application's transitions were fluent and the UI elements responsive...

- ○ Strongly agree
- ○ Agree
- ○ I can't decide
- ○ Disagree
- ○ Strongly disagree

5. How unique is the application?

- ○ Extremely unique
- ○ Very unique
- ○ Somewhat unique
- ○ Not so unique
- ○ Not at all unique

Figure 4.6: Second set of questions in the system evaluation survey

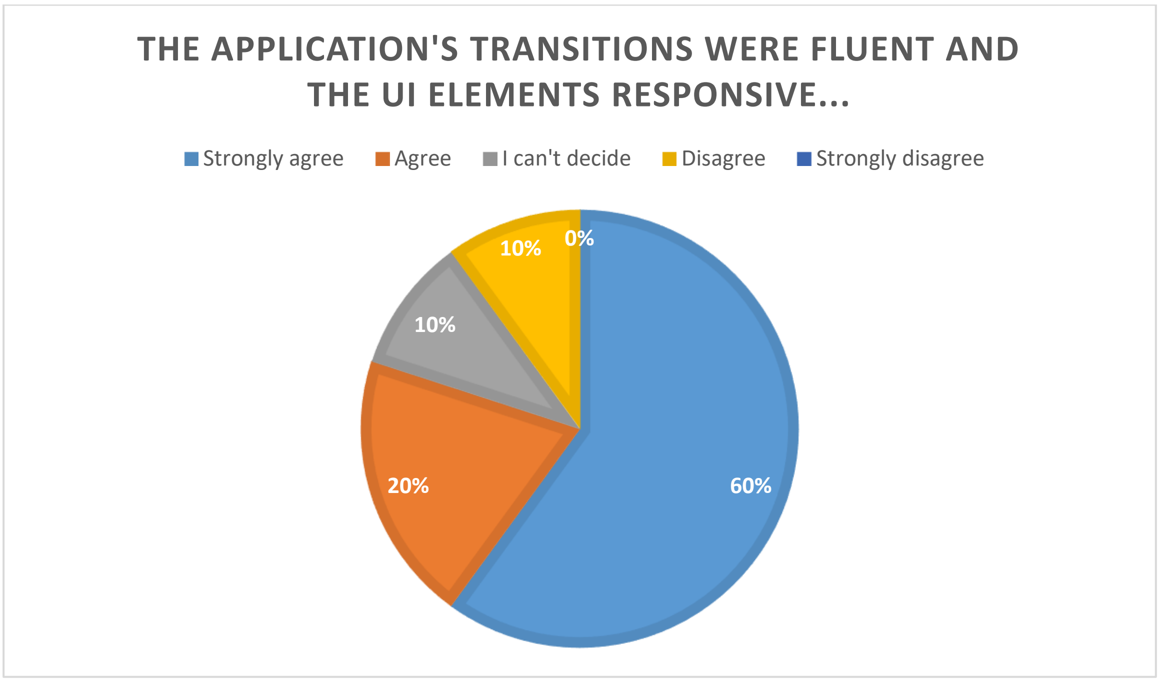


Figure 4.7: Fourth question results

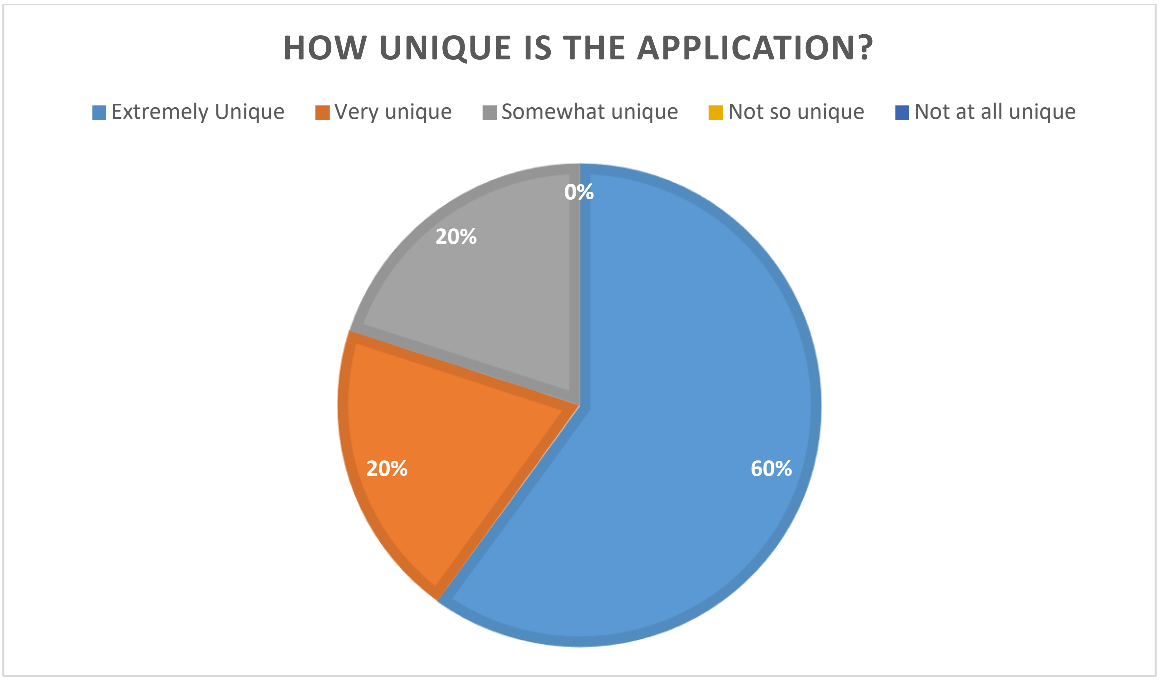
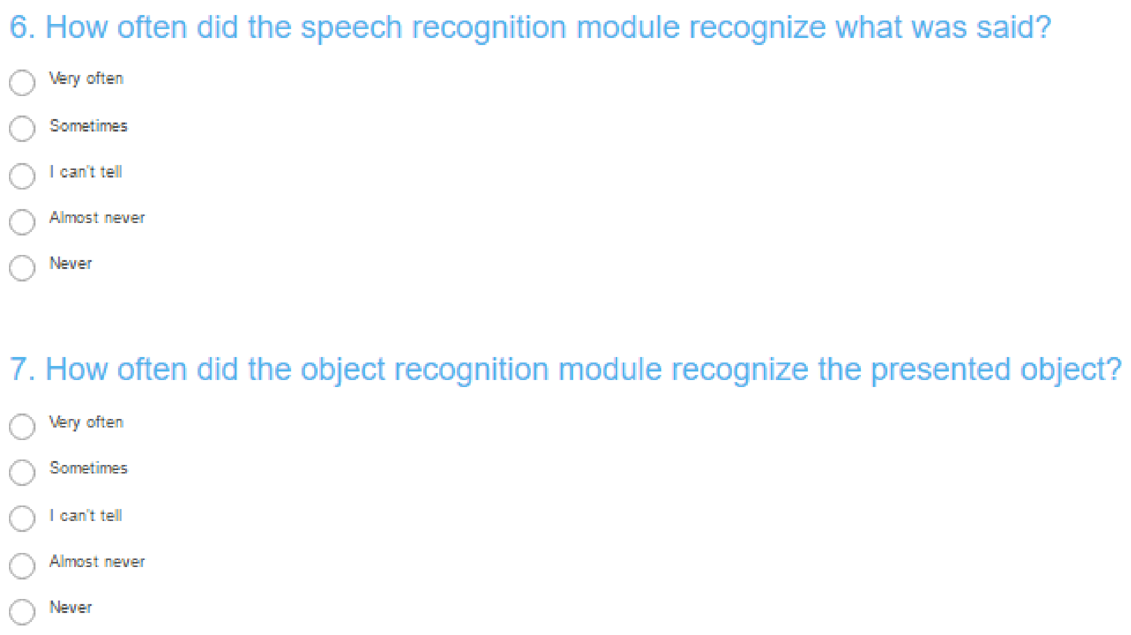


Figure 4.8: Results for the fifth question

The next figures now show the second part of the survey where the questions recovered information regarding the detailed functioning of the application as a computer vision and augmented reality system for the visually impaired.



6. How often did the speech recognition module recognize what was said?

- ○ Very often
- ○ Sometimes
- ○ I can't tell
- ○ Almost never
- ○ Never

7. How often did the object recognition module recognize the presented object?

- ○ Very often
- ○ Sometimes
- ○ I can't tell
- ○ Almost never
- ○ Never

Figure 4.9: Question six and seven of the system evaluation survey

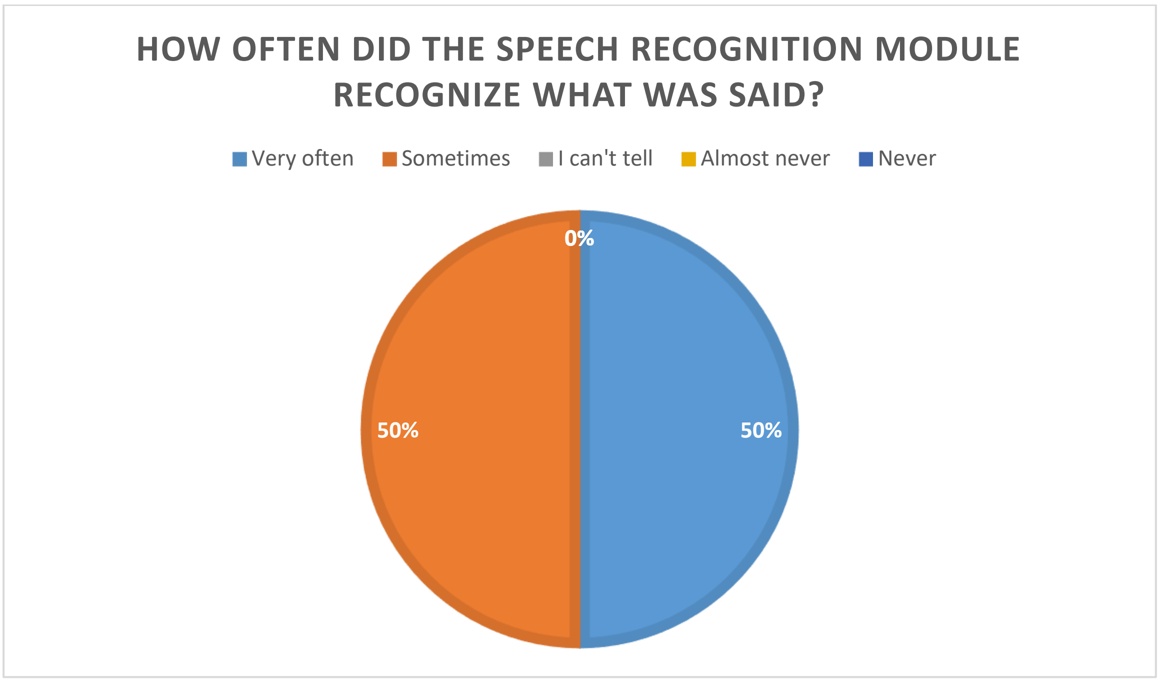


Figure 4.10: Results of the sixth question

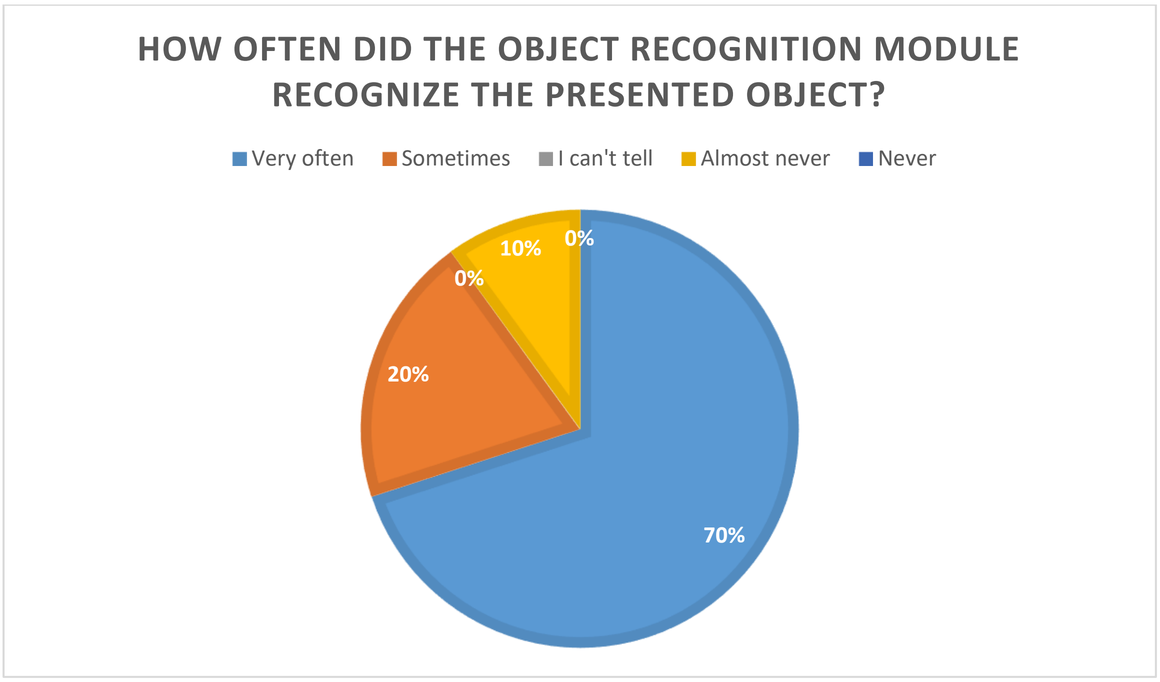
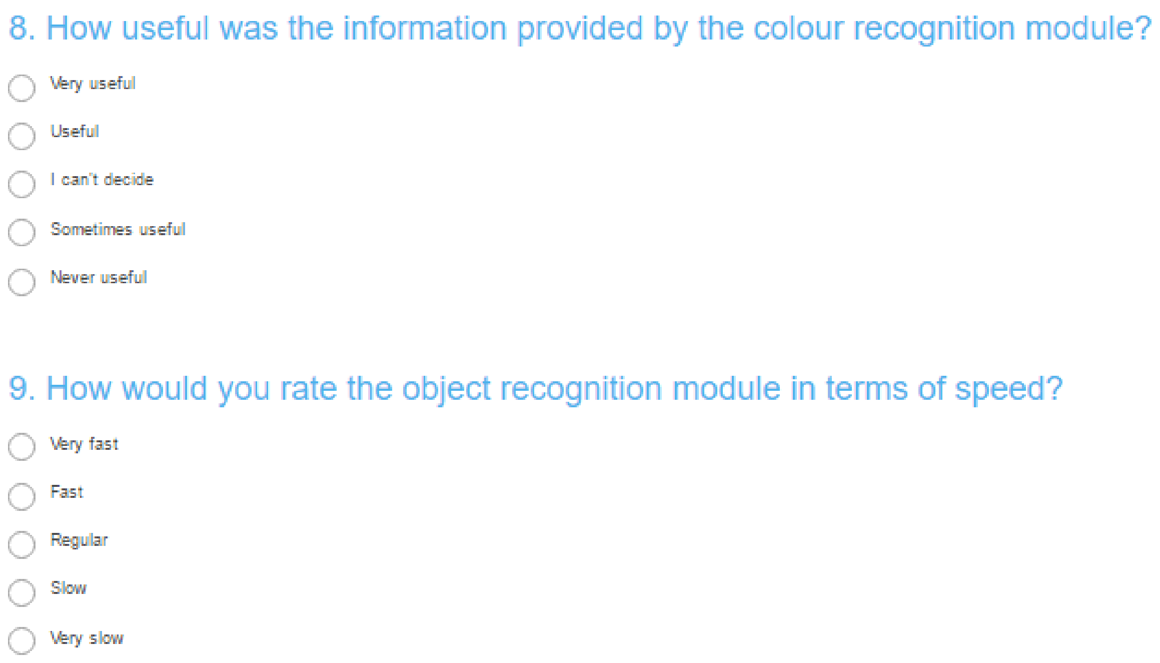


Figure 4.11: Results of the seventh question



8. How useful was the information provided by the colour recognition module?

○ Very useful

○ Useful

○ I can't decide

○ Sometimes useful

○ Never useful

9. How would you rate the object recognition module in terms of speed?

○ Very fast

○ Fast

○ Regular

○ Slow

○ Very slow

Figure 4.12: Question eight and nine of the system evaluation survey

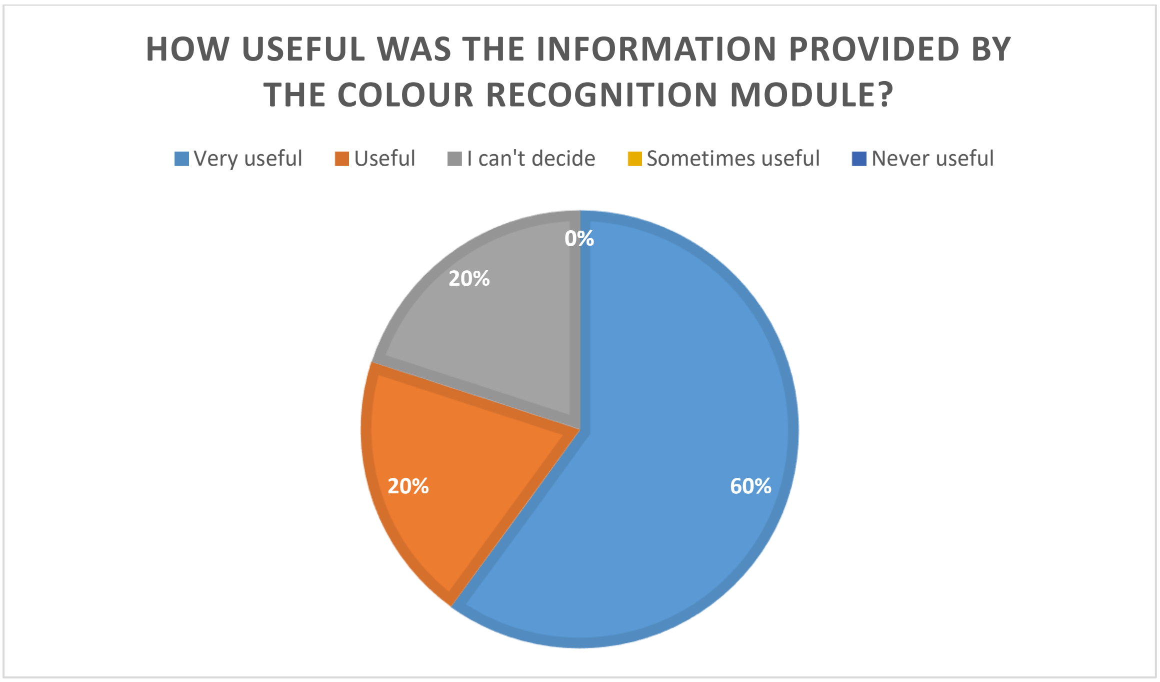
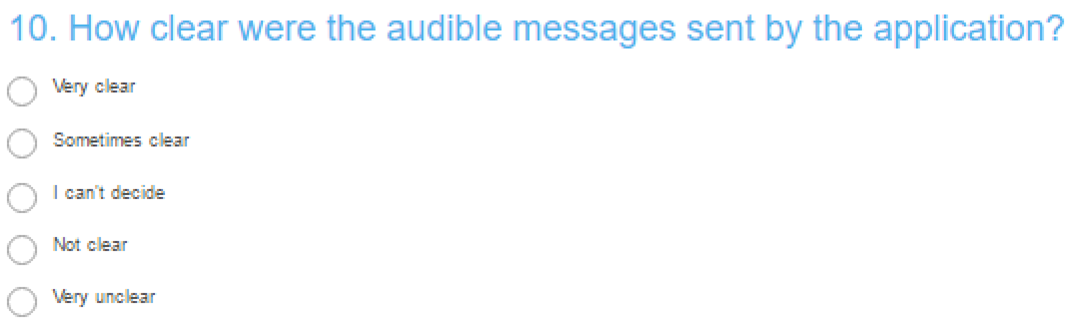


Figure 4.13: Result for question eight


10. How clear were the audible messages sent by the application?

- ○ Very clear
- ○ Sometimes clear
- ○ I can't decide
- ○ Not clear
- ○ Very unclear

Figure 4.14: Question ten of the system evaluation survey

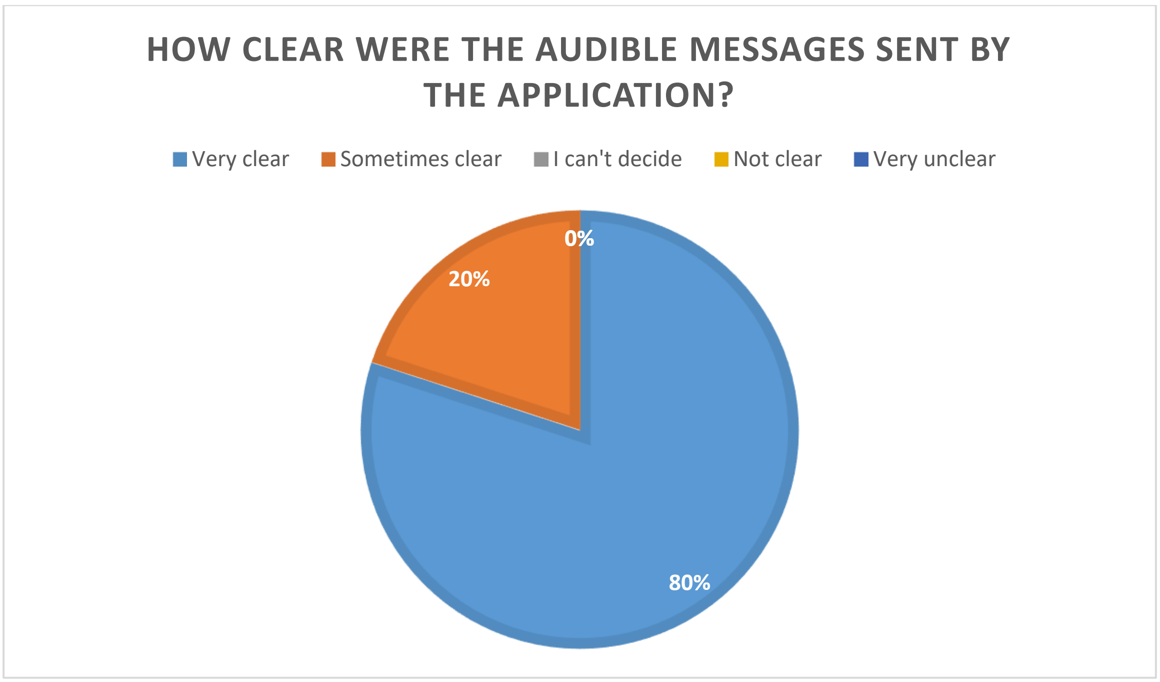


Figure 4.15: Results for question 10

4.2 Conclusions

From the results extracted from the survey it can be observed that there was general satisfaction with the application’s quality. The UI elements were reportedly clear enough and visually attractive. A majority of the users would recommend the application.

Regarding the systems performance as an augmented reality and computer vision system for the visually impaired, it seems the algorithms have proven sometimes slow and the colour information provided not always useful.

Chapter 5

FINAL CONCLUSIONS AND FUTURE STUDY

5.1 Final conclusions

During the development of this projects we have observed how complex computer systems are progressively becoming more present in our daily lives. Technologies which were before unfeasible and subject of fantasy writings and movies are now becoming a reality. It is now possible to integrate advanced technologies for daily use by the general public. Many years of intense research in computer vision are now producing great applications in medical, surveillance and security systems. Augmented reality systems are quickly becoming common place in entertainment and academic investigation.

The sudden increase of digital technologies brings a great amount of interest from the general public where many are willing to spend large amounts of money in expensive technologies. This has motivated a bigger investment by companies in investigating and developing new and more advanced technology.

Portable technologies are now becoming common-place. Smartphones, tablets, and smart watches allow to carry technology everywhere and function independent of the localization. Recent advances in Internet technologies allow to easily communicate over vast distances.

With the development of faster processors in portable technology, it is now easier to run complex algorithms with significant effects. Standard devices can now perform a variety of tasks without suffering the side-effects of computationally heavy processes.

The costs of technological development have decreased greatly in recent years, and much of it is now also present to the general public. This means that software applications can readily be developed by standard users and sold freely. In the case of Android applications, hundreds of applications are made by amateur developers and sold freely on the Google Play Store.

The physically disabled and visually impaired can take great benefit from this great and accelerated advance in technology. Present portable systems such as Smartphones and tablets can install applications which may assist them in their daily tasks. Augmented reality can help add additional layers of information which can aid the disabled. Computer vision systems can assist the visually impaired in communicating to them the world they cannot see.

The development of this project and the application has proven that creating a system which completely replaces personal assistance is difficult, but technology can be integrated to this process and motivate others to help the disabled.

With the completion of the application's development it can be said that the objectives have been met. The application can effectively aid a visually impaired individual in not only identifying objects present in a scene but also to orientate and recognize areas since the application studies the distinct features of any given scene. Additionally, the colour recognition functionality can provide aid to individuals who suffer from colour blindness.

It was also intended with the development of this application to motivate the development of augmented reality and speech dialogue systems to provide assistance to those who need it, independently from the entertainment domain.

5.2 Future study

As future work it would be interesting to reduce the computational load of computer vision algorithms. As an open source project, the OpenCV library is open to improvement. It would perhaps be convenient to further improve the OpenCV library and its recent API on Android.

The application Vision Guide could be subject to several improvements upon its functionalities. It might be interesting to combine the colour and object detection modules to improve object or scene detection. Furthermore, the dialogue system could be improved to provide a more fluent conversation with the user and an additional module which reads to the user the text present in the scene captured by the camera.

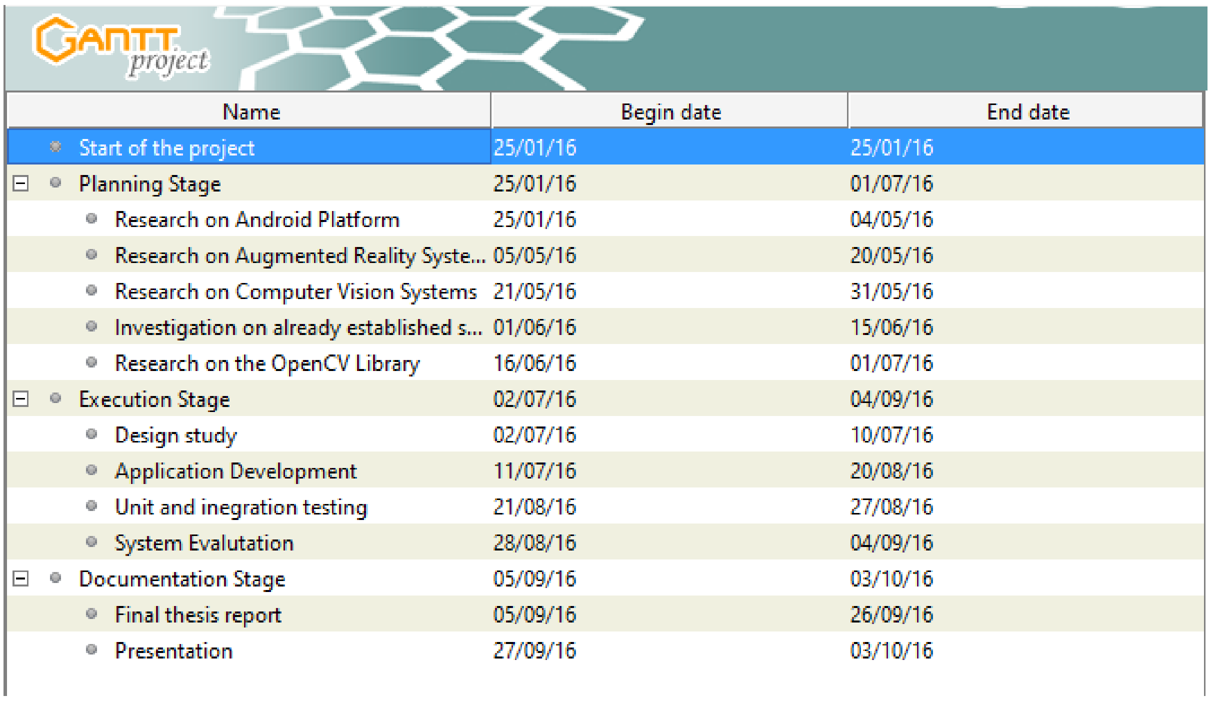
Chapter 6

PROJECT MANAGEMENT

6.1 Temporal planning

The project's temporal planning was managed by the free software tool GanttProject with which Gantt diagrams can be created with relative simplicity. The purpose of the Gantt diagram is to visually show the time dedicated to each individual tasks required for the project's completion.

The duration of each task has been set considering a dedication time of 3 hours per day. The next figure shows a table with all tasks and their durations elaborated within the GanttProject tool.



Name	Begin date	End date
Start of the project	25/01/16	25/01/16
Planning Stage	25/01/16	01/07/16
Research on Android Platform	25/01/16	04/05/16
Research on Augmented Reality Syste...	05/05/16	20/05/16
Research on Computer Vision Systems	21/05/16	31/05/16
Investigation on already established s...	01/06/16	15/06/16
Research on the OpenCV Library	16/06/16	01/07/16
Execution Stage	02/07/16	04/09/16
Design study	02/07/16	10/07/16
Application Development	11/07/16	20/08/16
Unit and inegration testing	21/08/16	27/08/16
System Evalutation	28/08/16	04/09/16
Documentation Stage	05/09/16	03/10/16
Final thesis report	05/09/16	26/09/16
Presentation	27/09/16	03/10/16

Figure 6.1: Table of project stages shown within the Gantt Project tool

In the next figure the Gantt diagram is shown which was drawn by the software using the previous table.

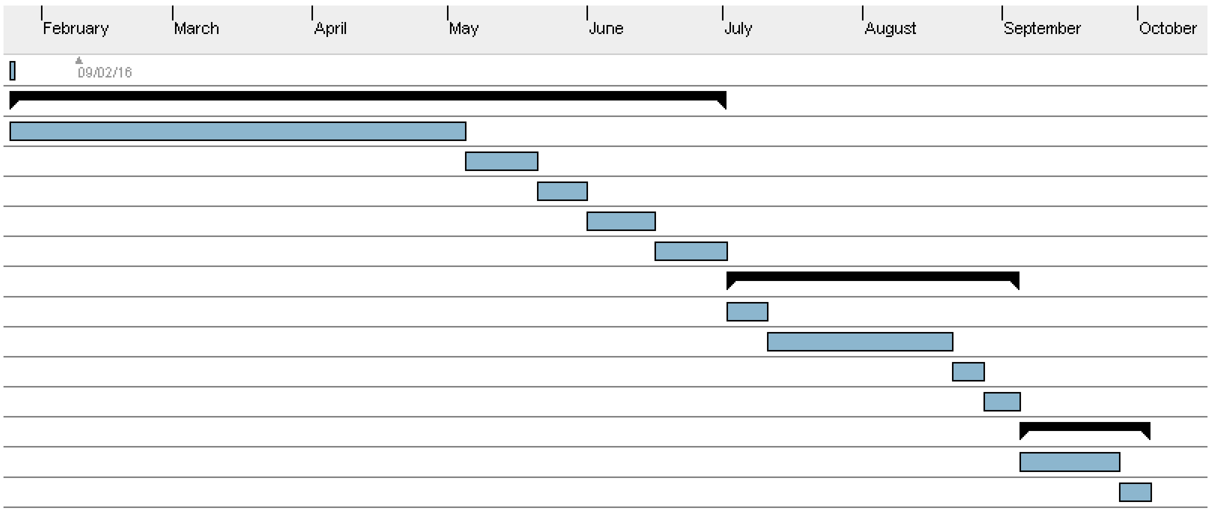


Figure 6.2: Gantt diagram drawn using the Gantt Project software

The total duration of the project has been of 253 days. The total duration of the project was longer than expected due to the required investigation on Android application development which was previously not known.

6.2 Budget

In this section the project's research and development costs are listed in order to estimate the required budget.

Human resources:

For calculating the costs of human resources, the following formula is used extracted from the template provided by the university (uc3m, 2014).

Cost = (duration (days) * daily hours) / (human dedication per month) * cost dedication

- Duration in days: **253**
- Daily hours: **3**
- Human dedication per month: **131,25**
- Cost dedication (engineer): **2694,39**

- Total cost result: **15 581,27 €**

Hardware resources:

- Desktop computer: **600 €**
- Smartphone Xperia S: **241,45 €**
- Samsung Galaxy Tab 4: **281,95 €**
- USB Cable: **5 €**

Software resources:

- Android Studio development environment for Android applications: **0 €**
- Android SDK (Software Development Kit): **0 €**
- Java JDK (Java Development Kit): **0 €**
- OpenCV Android SDK: A development API for Android applications: **0 €**
- Google TTS synthesizer for Android devices: **0 €**
- Google speech recognition on Android devices: **0 €**

Other costs:

- Indirect costs: **3 525,80 €**
- Redemption costs: **110,05 €**

Total costs category	**Total budget costs**
Human resources	15 581,27 €
Hardware resources	1 128,94 €
Software resources	0 €
Indirect costs	3 525,80 €
Redemption costs	110,05 €
Total costs without taxes	20 346,06 €
Total costs with taxes (0,21)	24 618,73 €

The total costs of the project is thus TWENTY-FOUR THOUSAND SIX HUNDRED AND EIGHTEEN EURO AND SEVENTY-THREE CENT.

DEFINITIONS

Activity: Defined Class which corresponds to an independent screen of the application.

Android Studio: Development environment for Android applications in JAVA code.

API: Set of methods and calls which provide functionalities and processes, representing an abstraction layer for the developer.

Application: Set of activities and methods which compose basic and complex functionalities in an Android software.

Augmented Reality (AR): Augmented reality is a real world environment view whose elements have been augmented by computer generated data.

Automatic Speech Recognition (ASR): System in which the user's utterance is identified by the device in real time.

Computer Vision (CV): Field of study which deals how computers can simulate the human visual sensory system and gain a high level of understanding of images and videos.

Gantt Diagram: Graphical representation with the purpose of showing the duration of project stages and tasks.

Intent: Android system component which expresses an action generally regarding a set of data.

Java Development Kit (JDK): Set of tools that allow the developer to create programs and applications in the JAVA programming language.

Layout: Android development component that defines the visual structure of the user interface.

Object recognition: System by which a certain object presented in the camera scene is identified.

OpenCV: Library for developing computer vision software.

OpenCV Manager: Control software that updates and links OpenCV libraries on execution.

RecognizerIntent: Class of the *android.speech* packet which provides the necessary constants for the integration of voice recognition systems in Android applications.

Software Development Kit (SDK): Set of tools that allow the developer to create an application in a determined programming language or platform.

Spoken Dialogue System (SDS): System which allows interaction with the device through regular speech simulating human interaction.

Text to Speech Synthesis (TTS): System which allows the conversion of written text to spoken language by the device.

Virtual Reality (VR): Virtual reality consists of using computer generated information to replicate a real or imaginary environment.

Work Breakdown Structure (WBS): Tool used to visually analyze the project's planning with each of its stages and corresponding tasks.

REFERENCES

Schmalstieg, D., & Höllerer, T. (2016). *Augmented reality: Principles and practice*. Retrieved September 18, 2016, from http://proquest.safaribooksonline.com.strauss.uc3m.es:8080/9780133153217

Azuma, R. T. (1997, August). A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments, 6*(4), 355-385.

Craig, A. B. (2013). *Understanding augmented reality: Concepts and applications*. Retrieved September 18, 2016, from http://proquest.safaribooksonline.com.strauss.uc3m.es:8080/9780240824086

By Augment. May 12th, 2016. Industry & Augment News. Infographic: The History of Augmented Reality - Augment News. Retrieved September 18, 2016, from http://www.augment.com/blog/infographic-lengthy-history-augmented-reality/

White, Jules. Schmidt C., Douglas. Golparvar-Fard, Mani (2014, February). Applications of augmented reality. Proceedings of the IEE. Vol 1002 No. 2

Szeliski, R. (2011). *Computer vision: Algorithms and applications*. Retrieved September 18, 2016, from http://szeliski.org/Book/

Skantze, G. (2007). *Error handling in spoken dialogue systems: Managing uncertainty, grounding and miscommunication*. Retrieved September 18, 2016, from http://www.sciencedirect.com.strauss.uc3m.es:8080/science/article/pii/S0167639304

SpeechRecognizer. (n.d.). Retrieved September 19, 2016, from https://developer.android.com/reference/android/speech/SpeechRecognizer.html

TextToSpeech. (n.d.). Retrieved September 19, 2016, from https://developer.android.com/reference/android/speech/tts/TextToSpeech.html

Board, C. I. (n.d.). Apps for People with Disabilities and Older People. Retrieved September 19, 2016, from http://www.assistireland.ie/eng/Information/Information_Sheets/Apps_for_People_with_Disabilities_and_Older_People.html

Gregori, B. E. (n.d.). Developing OpenCV computer vision apps for the Android platform. Retrieved September 19, 2016, from http://www.embedded.com/design/programming-languages-and-tools/4406164/Developing-OpenCV-computer-vision-apps-for-the-Android-platform

Tutorial on Binary Descriptors – part 1. (2016). Retrieved September 19, 2016, from https://gilscvblog.com/2013/08/26/tutorial-on-binary-descriptors-part-1/

OpenCV: Introduction. (n.d.). Retrieved September 19, 2016, from http://docs.opencv.org/3.1.0/d1/dfb/intro.html

Bradski, G. R., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. Beijing: O'Reilly.

OpenCV - Google Play. (n.d.). Retrieved September 19, 2016, from https://play.google.com/store/search?q=opencv