

Coursework Submission Cover Sheet

Please use Adobe Reader to complete this form. Other applications may cause incompatibility issues.

Student Number

Module Code

Submission date

Hours spent on this exercise

Special Provision

(Please place an x in the box above if you have provided appropriate evidence of need to the Disability & Dyslexia Service and have requested this adjustment).

Group Submission

For group submissions, *each member of the group must submit a copy of the coversheet.* Please include the student number of the group member tasked with submitting the assignment.

Student number of submitting group member

By submitting this cover sheet you are confirming that the submission has been checked, and that the submitted files are final and complete.

Declaration

By submitting this cover sheet you are accepting the terms of the following declaration.

I hereby declare that the attached submission (or my contribution to it in the case of group submissions) is all my own work, that it has not previously been submitted for assessment and that I have not knowingly allowed it to be copied by another student. I understand that deceiving or attempting to deceive examiners by passing off the work of another writer, as one's own is plagiarism. I also understand that plagiarising another's work or knowingly allowing another student to plagiarise from my work is against the University regulations and that doing so will result in loss of marks and possible disciplinary proceedings.

Contents

Introduction.....	3
Prior Research & Background	4
How is object recognition currently achieved?.....	4
Choosing a descriptor part 1: HOG and texture descriptors.....	4
Choosing a descriptor part 2: SIFT, SURF, FAST with BRIEF key point detectors and descriptors	5
Selecting a smart phone operating system.....	8
Libraries that implement the methods discovered	8
System Specification and Design.....	9
System Scope + Boundaries	9
Requirements.....	9
Functional Requirements	9
Non-Functional Requirements	11
Use Cases	14
Software Work Models – User Interface & Background System.....	16
System Architecture	17
Implementation	19
Complications when implementing.....	19
Critical code documentation	20
System User Interface, Testing & Evaluation	25
UI Prototype and System Usability Testing: Think-Aloud User Evaluation	25
The Implemented UI.....	33
System Experiments	36
Introduction	36
1. Angle of Acceptance	37
2. Dolly Capacity / Object Scale Robustness	38
3. Noise Limit.....	39
4. Library Limit.....	41
5. Object Distortion	42
System Functionality Testing and Evaluation.....	44
Future Adaptations.....	51
Helping visually impaired recognise and find objects	51
Finding lost objects.....	51

Conclusions.....	53
Reflection	54
Bibliography.....	56

Introduction

Object recognition, that is finding and identifying different objects in images, video streams or similar media can be achieved on computers where processing and memory power is relatively large, but there are less examples of image recognition in smartphone or portable devices where the ability to perform recognition is far more capped.

The objective of the project is to produce an application for a smartphone device that can recognise user defined objects in an environment of noise through the device's camera. As smartphones have the added advantage of a freely moving camera to perform recognition on, this can then act as a basis for future developments such as helping visually impaired recognise specific objects or lost items.

The main objective can be split into a series of goals; Research into how object recognition maybe achieved and how it may be applied to a smartphone device, designing the system using the research as a foundation for assumptions, implementing the system itself, testing the boundaries and functionality of the system created, documentation of how the existing system may be applied to solve the future developments like that previously mentioned and finally drawing conclusions on the implemented system in comparison to the original design and objective.

Prior Research & Background

In order to create object recognition on smartphones a number of different fields need to be considered, these include; how object recognition is currently performed, what different methods or techniques used in object recognition will be best implemented to give the maximum performance and still be suitable on a smartphone, what smartphone operating systems are available and finally what recognition technologies or libraries exist that could aid the programming process.

How is object recognition currently achieved?

Object recognition requires three key components in order to function. This first is a single or set of images that are either identical or similar visually to the object you wish to recognise. Officially these are known as training images as they are used to train the detector in detecting objects of a similar type using the set. The second key component is the descriptor/method used to gain data from those training images in order to make a comparison. Here there are mainly two kinds of descriptors, texture descriptors and key point descriptors [29]. Texture descriptors uniformly process the entire training image that often results in a high number of parameters being gained as a description. Meanwhile key point descriptors only process pixels that have a high enough “uniqueness” value, before processing each pixel is given a uniqueness value to measure how likely that pixel will be to recognise in another image, the top valued pixels are the key points analysed further to give the parameters. Within these two kinds are descriptor methods available for detecting a vast range of data, from those that detect edges/gradients or corners to others using features, templates and image segmentation. The final component is the matcher itself, which is the way in which the data detected is matched with the query image (in the case of this project the frames seen through the phone camera) to determine whether or not the object exists in the query image and how it will then be highlighted.

Whether or not a single or set of images is used and how the data will be matched depends on the descriptor used to gain data from training images. As there are many descriptors available a selection of the most popular will be considered, each will be analysed in turn as to how it works and how affective it may be before ranking them in order of priority of implementation.

Choosing a descriptor part 1: HOG and texture descriptors

The first to consider is HOG (Histogram of Orientated Gradients); HOG is a type of texture descriptor that splits a training image into a number of localised cells [28]. For each cell, the shape of structures within is captured through calculating orientation gradients. Orientated gradients are determined for each cell by analysing pixels within the cell in turn; determining whether there is an edge that passes through the pixel, what orientation the edge is at and how visible it is. The result of the orientation gradients calculated for a region is added to a bin. When all cells have been monitored, the bin is then analysed as a whole to determine a consensus of the gradients and calculate edges for the entire image.

HOG will often produce parameters for each cell that are very similar to one another, performing HOG on a single training image gives little value. Often the HOG descriptor will need thousands of training images stored in a classifier that contains both positive and negative examples in order to calculate a threshold for matching and function appropriately for classification and recognition. The same is for other popular texture descriptors such as LBP (Local Binary Patterns) and Haar, though their methods and type of data retrieved may differ, still thousands of training images sorted in a classifier are need [32].

When considering transferring texture descriptor techniques to a smartphone device there is a major concern with the storage capacity of the platform. Even if through scraping many images can be maintained, smartphones do not hold the same memory of a standard computer or laptop. Storing thousands of images in order for the application to work without external storage may not

be feasible. For a numerical example, a face detector used in a publication on Haar training was recorded to have used a total of 8000 positive and negative training images [21]. A Motorola Moto G (representing a common budget smartphone) has a storage capacity of 5.52 GB in total. Each image taken using the camera takes up roughly 300KB, this leads to a total of 2.4 GB or 43.48% of the systems total capacity being used to track a single object type for the application. When considering other phone functions such as messaging, applications, audio, images and videos not used by the recogniser, using nearly half the storage of the phone is inefficient.

The result is that HOG and other texture descriptors will be of the lowest priority when choosing a descriptor for smartphone transfer. This means that further research will focus on Key point detectors that need only a single image of an object to create matches and compare similarities. The impact this will have on the smartphone system will be it can no longer recognise objects at a high semantic level, which is what texture descriptors excel at doing. To expand, it will not be able to recognise two distinct objects as being the same thing, such as two different cats as being in the same family “cat”, but using key point detectors will be able to recognise them as two separate entities provided there is at least an image of each and that those images to some operable degree look similar to the training images.

Choosing a descriptor part 2: SIFT, SURF, FAST with BRIEF key point detectors and descriptors

SURF (Speeded Up Robust Features) and SIFT (Scale-Invariant Feature Transform) are scale and rotation invariant [38] [39], this means the size or orientation of the object in the query image compared to the training image does not affect its matching performance. SIFT and SURF are also both real value descriptors [15], meaning the resultant vectors produced by their key point feature detection are in real values. Being in real value means that when matching between features in query and training images the distances (measure of similarity between two descriptors) are calculated using Euclidian distance. Whereby the Euclidian distance $d(P,Q)$ between two vectors, P and Q containing values $(P_1, P_2 \dots P_N)$ and $(Q_1, Q_2 \dots Q_N)$ respectively is given by the following Pythagorean theorem [20]:

$$d(P, Q) = d(Q, P) = \sqrt{(Q_1 - P_1)^2 + (Q_2 - P_2)^2 + \dots + (Q_N - P_N)^2} = \sqrt{\sum_{i=1}^N (Q_i - P_i)^2}$$

The main difference between SIFT and SURF is the methods used in calculating key point/pixels and their features. SIFT creates various scales of the image through a scale space, which by smoothing and resampling the image results in a series of images at different scales [34]. SIFT then applies a difference of Gaussians function (a method of effectively “subtracting” one blurred image from another [35]) between adjacent images in the space to calculate maxima and minima, the extrema calculated act as the points of interest on the image. These points are further narrowed in key point localisation [18], where unstable points are removed and then real value descriptors made by combining together remaining key points in a local region that sum the orientation and scale data of each separate point it encompasses.

SURF also builds a scale space, but instead uses an integral image of the original training image (whereby an integral image is the original image split into summed areas of pixels rather than a series of individual pixels) and uses a Hessian matrix approximation to find key points [5]. The real value descriptors are calculated using a set of quadratic grids of a set size covering the entire image. Each grid is orientated to the dominant direction of the feature under the grid, each descriptor is then the sum of x and y values relative to the orientation of the grid.

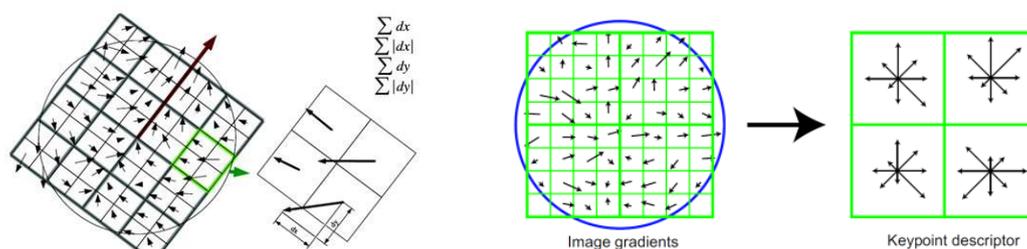


Fig. 1. Creating descriptors in SURF (left) and SIFT (right)

As a result of using an integral image, for the same size image SURF is much faster at computing matches than SIFT but SIFT normally finds and matches more key points resulting in greater accuracy. In a research paper by the International Journal of Innovative Research in Computing and Communication Engineering [25], SIFT calculated 46% more matches between the same two images than SURF however SURF had an approximately 3x faster run time. For a smartphone device application that needs to run and compare images through a camera frame as close to real time as possible, which also has less processing power than a standard computer, SIFT may be too slow to transfer to a mobile device. However SURF being faster and still having good performance as well as being scale and rotationally invariant is a strong candidate.

FAST (Features from Accelerated Segment Test) with BRIEF (Binary Robust Independent Elementary Features) and ORB (Orientated FAST and Rotated-BRIEF) both can create binary descriptors. When matching between features in query and training images the distances are calculated using Hamming distance. A hamming distance value is an integer representing the number of coefficients in which two descriptors differ (the higher the value the more differences there are) [12]. For instance, the Hamming distance between two binary values “10111” and “10010” is 2.

FAST compares and extracts feature of a training image by detecting corners [6]; it achieves this by using a circle of 16 pixels (also known as a Bresenham circle of radius 3) as a classifier on each pixel in the image to determine if they are corner points or not. Each pixel in the Bresenham circle is labelled 1 to 16 clockwise; the pixel under consideration, P , is at the centre of the circle and given an intensity value, I_p . An intensity threshold is set, then if a set N of contiguous pixels out of the 16 are either lighter or darker than I_p that pixel is considered a key corner point. The algorithm is made faster by only comparing pixels 1, 5, 9 and 13 of the circle first. If 3 out of the 4 intensity values at those pixels are not above or below the threshold then the point can be discarded early. Otherwise, cycle through the remaining 16 to see if the other pixels that are above or below threshold add to a total of N .

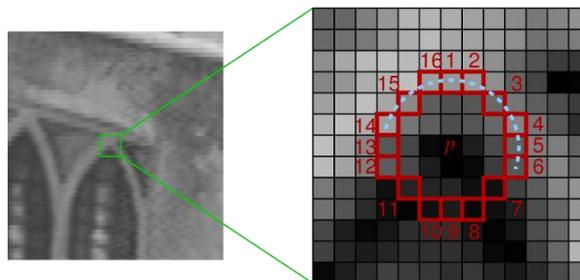


Fig. 2. FAST & the Bresenham circle

FAST by itself does not cover implementation to compute descriptors of the corner points for matching, for this it needs another separate method BRIEF [19][11]. BRIEF creates a binary string as the descriptor for a region around each key point by performing a number of binary tests equal to the length of the string, 1 is a passed test and 0 is a failed test. Each test compares two points (P_1 and P_2) around the set region of the key corner point. If P_1 has a greater intensity value than P_2 a binary result of 1 is recorded in the binary string, otherwise it is a 0. This gives each key point a unique binary identifier acting as the descriptor in the training image, which can be measured against identifiers of the query image to match descriptors that are most similar (through Hamming distance). Although FAST with BRIEF descriptors are robust to lighting, blur, and perspective distortion in a similar way to SIFT and SURF, they do not take into account scale or rotation and are not invariant or robust in these fields.

ORB builds off of both FAST and BRIEF methods but adjusts them to make them rotation invariant and give more robustness to scale (but not invariance) at the cost of additional computing time. How ORB achieves this is by first making the FAST feature detector incorporate both scale and

rotation data. Firstly, FAST features are filtered so that only the most corner like points are taken into the key point set, this is so a measure of “cornerness” is imposed and only the best and most useful key point need take up descriptor computation. This is done through a Harris corner measure [4], to begin a threshold is set to gain more than N points out of all the ordered points (where N is the target number of key points wanted), these N+ points are measured and ranked in accordance to the Harris corner measure then the top N are selected. Only these top N key points are continued on to be given scale robustness, scale invariance and be made descriptors for. Scale robustness is introduced to the top key points through a scale pyramid of the image, producing FAST features for each filtered key point at every level of the pyramid. Orientation data is added to the select corner key points by an intensity centroid [26]. The intensity centroid makes the assumption that a corner’s intensity is offset from its centre, through calculating the moments of a circular patch of radius R around the corner key point the centroid can be found. Finding the vector from the centre to the centroid may then be used to create an orientation degree. Mathematically this follows:

. Finding the moments of a patch, M, where p and q are the order of moments:

$$M_{pq} = \sum_{x,y} x^p y^q I(x, y)$$

. Finding the intensity centroid using these moments:

$$C = \left(\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}} \right)$$

. Calculating orientation from the vector between centre of corner, O, to the centroid, C:

$$\theta = \text{atan2}(M_{01}, M_{10})$$

Once all key points have been given orientation values and some degree of scale is considered, BRIEF descriptors of the key points are then made rotation invariant by using the orientation of a key point to “steer” BRIEF calculations in a relative direction using a corresponding rotation matrix.

FAST with BRIEF and ORB have quicker runtimes in both the matching and detecting process when compared to SIFT and SURF. This is because when compared to Euclidian distance, Hamming distance is calculated far faster when matching two images [27]. Furthermore their methods of finding key points are also computed at a greater speed. For example, in a research paper on “ORB: an efficient alternative to SIFT or SURF” [9], the time per frame in milliseconds that FAST and ORB took to detect objects through the same video string was 8.68ms and 15.30ms respectively meanwhile it took 217.30ms for SURF and 5228.70ms for SIFT to compute the same function. However, SIFT and SURF are more robust than FAST with BRIEF and ORB as they are both scale and orientation invariant rather than only rotation invariant for ORB or neither invariance for FAST with BRIEF. Considering this with the fact that SURF is still relatively fast in comparison to SIFT, SURF will still remain the top priority. ORB being faster than SURF and more robust than FAST with BRIEF is a second priority whilst FAST with BRIEF will be third as SIFT will drop camera frames per second significantly more making the application seem unusable despite being better at matching.

To re-iterate, the priority order in which the descriptors and methods discussed will be implemented in if the other is unsuccessful will be:

1. SURF
2. ORB
3. FAST with BRIEF
4. SIFT

5. HOG

*MSER was not researched as a possible descriptor to be used by the system despite being in the initial report. Having already evaluated two additional key point descriptors (FAST with BRIEF and ORB) it was decided that another popular descriptor that focuses on key points would not need to be reviewed and would waste resources to investigate rather than start implementation. A sufficient range of descriptors had already been identified along with the top three choices being sufficiently quick enough to port to smartphones without additional investigation.

Selecting a smart phone operating system

The two choices in phone operating systems available to test and implement on are coincidentally the most popular for smartphones at present day, Android and Apple iOS. Both have well-known application stores and technologies that support mobile developers. When choosing between the two it was a matter of which application was programmed in a language most suitable for the strengths of the programmer, or if both are coded similarly which contained the most support for testing new applications (especially in the context of object recognition and the need to access the system camera etc.)

iOS applications are programmed in Objective-C using the Xcode IDE [14], Objective-C is an object orientated language built on top of the existing C language. Meanwhile Android applications are coded in Java [8], mainly using either the specific Android Studio or Eclipse IDE. Given that the programming language most recognisable is Java, and that there would be little time to learn Objective-C in addition to transferring object recognition to a smartphone it was decided to prioritise using an Android rather than iOS operating system to host the recognition application.

Libraries that implement the methods discovered

There are two popular open source libraries that contain methods/algorithms for image processing and recognition (i.e. functions for SURF detection and matching available), they are VLFeat and OpenCV [16]. VLFeat is written in C with an interface for Matlab and supports Windows, Mac OS X and Linux operating systems [41]. Meanwhile OpenCV is written in optimised C/C++ with an interface for C, C++, Python and Java and supports Windows, Linux, Mac OS, iOS and Android [22]. Given that Android has been selected as the operating system of best choice and that Java is the preferred language, OpenCV having a Java interface and Android compatibility gives it a greater priority over VLFeat that has neither in this context. The resources needed to re-write the relevant parts of the VLFeat library for Android compatibility as well as creating a corresponding Java interface could take over creating the application that uses and tests it. Therefore with VLFeat, there is less of a guarantee that the end result will be a smartphone application that can perform object recognition to some extent, whereas OpenCV has the documented tools to begin using recognition algorithms on smartphones from the start. With this in mind, OpenCV will take first priority over VLFeat when it comes to applying a recognition library on the smartphone being used.

System Specification and Design

System Scope + Boundaries

The system scope and boundary is a succinctly written description of what the system is designed to be from research. This is used to help fuse all the results of research into a more condensed form so that requirements of the system can be more easily drafted. The system scope and boundaries are as follows:

The smartphone system will be programmed in Java for Android devices. In order to perform object recognition the system will need two key components. One component is constantly connected to the Android camera that displays the stream of frames received on screen whilst giving the option to capture the current frame being viewed. The captured image will be saved in a location accessible on the smartphone; Android provides a public Media Gallery that is suitable this [1]. From this section the system will also show the results of object recognition using SURF methods accessed from the OpenCV library via a java interface. The recognition function will be between the query image (the frames in the camera) and the training images available to the application. This component will be called the “finder”.

The second component stores the training images to be compared and recognised in the camera frames shown in the finder. This component will be called the “library” and will be able to; display training images in some library list form, add images from the Media Gallery, delete images that have been added and select images in the library to be cropped. Cropping is necessary in order to only have a training image the size of the object itself so that no other features but the object are recognised in the camera view. Android also provides an in-built crop mechanism that is suitable to serve as the crop screen [31].

Requirements

Functional requirements were derived using the scope and boundaries description to provide a more concise list of deliverables so that it is easier to break down and implement parts of the problem at hand. Both non-functional and functional requirements also help serve as a benchmark when evaluating the results of what was implemented with the specification of the system design.

Functional Requirements

Finder view

1. The finder must capture images through the mobile device’s camera and save them to an external storage location

Acceptance Criteria:

- . On response to a button or touch command, the application must be able to get the frame data that the camera received at that time and write to an image file
- . This image file data must be sent and stored in external storage than can be accessed again when needed (such as the Android Media Gallery discussed)

Justification:

This is one of the key requirements using the method chosen to perform object recognition; the user must be able to take an image of the object to recognise otherwise there will be no training image to search for in the camera frames.

2. The finder must be able to perform feature matching between images in the library and frames processed by the camera

Acceptance Criteria:

- . The camera in response to a system button must switch between performing no processing on camera frames to on each frame, trying to match features from images in the library to that frame
- . A user must not be able to perform feature matching if the library is empty/ there are no training images to use

Justification:

The central requirement in order for the application to adhere to its scope and description, if the system cannot match features in anyway then it has not solved the problem or brief in much capacity. Performing a feature match on null data when the library is empty will most likely lead to computational errors, to avoid the entire application shutting down the system should never initiate feature comparison with an empty library and should instead display an error message.

3. The finder will display positive matches through outlining the matched library object in the camera display using a graphical overlay

Acceptance Criteria:

- . The graphical overlay must be bound around the object in order to highlight its location
- . To distinguish multiple objects being tracked, each object image in the library must have the option of being named or labelled and this label must also show on the recognised item in the frame as part of the overlay
- . The graphical overlay must be visible and distinguishable against other backgrounds and entities in frame

Justification:

Performing the background processing for matching yields no results to a user unless they can see the outcome, making this requirement equally as important as the other functions.

4. The finder must contain functionality to switch to the library view and from the library view back to the finder

Acceptance Criteria:

- . On button click, the system must be able to navigate to the library view, displaying relevant UI features and functionalities for that view as necessary
- . The library view must have a return option to navigate back to the finder

Justification:

In order to decrease clutter on a single screen it is best to split functionality into relevant groups and give each group its own user interface, this requirement is therefore necessary to navigate between the groups chosen. This case being functions involving the library group, which is editing and adding of the training images as well as the finder group, that is functions involving the initiation and results of object recognition.

Library view

1. The system's library must store images selected from other storage facilities on the mobile device

Acceptance Criteria:

- . On button click or touch, the library must be able to navigate to the external storage interface
- . From this interface, the user must be able to select a single image from a listed selection of all images taken
- . Once selected, the image data must be sent and stored in the library
- . After selection, the user must be prompted with an optional text entry field to label said image with a string that is stored in the library alongside it
- . The system must return to the library after the above has been performed
- . The library must be able to hold any quantity of images

Justification:

The library must be a separate storage from where all the images are saved to in order to increase flexibility of the programme. For example a user may take an image, add it to the library separately, remove it from the library and add the same image again without having to re-take the photo. This would not be the case if all images taken were stored directly into the library without use of an external storage.

2. The system's library should remove images from its own storage

Acceptance Criteria:

- . Via the use of a menu button, the user must be able to select an image and it be removed along with its label from the library storage

Justification:

This requirement is important to allow freedom of action for the user during a session using the application. If an object has been recognised or is no longer needed to be tracked then removal of the items training image from the library will free up computational space for other objects of higher priority. A delete function will also ensure the system will not become overloaded performing matching on objects that do not need to be found and will not compete for space on the camera view by displaying recognition graphics for an object that has no longer a desire to be recognised.

3. The system's library should support a crop function for images within its storage

Acceptance Criteria:

- . On selection of a menu button, the user must be able to choose an image to be cropped and have its data sent and displayed to a crop screen where the system will also navigate to
- . The image must be able to be cropped from each of its four edges, with the flexibility to crop as much of a percentage of each edge as desired
- . After cropping selection the user must be able to select a save button, whereby the system navigates back to the library screen with the previously cropped image data replacing the uncropped data of the same image
- . The string label for this image must remain the same

Justification:

Cropping will be an essential requirement in order to focus the recogniser to find features that are only present on the objects itself and not additional features from around the environment to which the images object was taken in.

Non-Functional Requirements

Performance

1. The initial camera fps (frames per second) before running matching functionality alongside should match the performance of the default camera specification for that device

Acceptance Criteria:

. For the Motorola Moto G device used in testing and showcasing the system, this should be 30 fps

Justification:

As running the feature matching code on each frame will decrease fps, having the highest possible fps for the camera before matching takes place will ensure that the decrease is the lowest it can be for that device.

2. The resolution for the screen size of the application camera should match that of the maximum resolution available for that size on the device

Acceptance Criteria:

. The widescreen (16:9) camera for the test device operates at 864x480 pixel resolution

Justification:

The greater the resolution the more detail the object to be recognised will be in, this will further result in more features accurately being picked up by the matcher and theoretically lead to better matching. It is therefore important for the camera's resolution being used by the application to match the best available resolution the device has for the widescreen.

3. The time taken to switch between finder and library views should be equivalent to the expectations for loading times

Acceptance Criteria:

. The time to switch between views should be ≤ 2 seconds

Justification:

Users will often lose interest in an application if loading times are too long or functions take too long to initiate/work. From an independent survey taken by Akami and Gomez [30] the wait time before user's loose interest of a web page is 2 seconds or less. Though web pages perform differently to in-built applications, this should act as a benchmark for the attention span of typical users from initialising a function to seeing its results on screen.

4. Response times between switching from matching to non-matching modes should be instantaneous

Acceptance Criteria:

. It should take ≤ 2 second to switch between standard camera view and feature matching view

Justification:

See "Justification" for performance requirement 3.

5. The finder should match objects in the library with camera frames at real-time

Acceptance Criteria:

. Finder should always use the most recent frame in the data stream from the camera to perform match processing on

Justification:

Object recognition needs to occur as the camera view is translated across a scene, such that the user knows in real time when an object to be recognised is in view of the camera and what that object is. Using other methods such as performing recognition on a pre-recorded video would not suit the scope or description of what the recogniser is tasked with.

Reliability

1. Matching functionality should use tried and tested feature extraction and comparison methods

Acceptance Criteria:

. The system should use the top priority feature extraction method, SURF, as the basis for extracting features and matching descriptors

Justification:

The methods of feature extraction explored in the research phase of the project have been tried and tested to have measurable success among many sample types and spaces. This allows for comparison and selection of the best method for the application. Creating a new feature extraction method will not give enough time to test as extensively as previously designed methods to the extent that it can be indefinitely compared and justified that it is better for the application than an existing technique.

Compatibility

1. The system should accommodate for different models of phone using Android

Acceptance Criteria:

. Techniques used should not be hardcoded for a specific device specification

Justification:

The application was designed to be used on Android mobiles, as there are many types of mobiles using Android by multiple manufacturers it would be unbeneficial to make the system optimal only for the test device. Examples of this would be setting the camera view to be a specific pixel size that fitted the test screen well but appeared too small or large on other mobile screens, or setting the resolution to match the maximum capacity of the test device but be incompatible with a device with a smaller capacity.

Usability

. UI should adhere and be designed towards a well validated set of heuristics

Acceptance Criteria:

. The heuristic set chosen shall be 'Nelson's usability principles' for its popularity and credit
 . There should be evidence of these principles being considered when designing and testing the interface

Justification:

Rather than designing and creating an interface based on what appears good to the creator and to test against no real benchmark. It is deemed best to follow approved guidelines that have been modelled specifically to help interfaces in terms of usability, then to test the interface in accordance

to how well it met those principles. This more structured approach should result in a user centric rather than personalised interface and also save time when testing as results can be more easily contextualised and measured.

Use Cases

Using use cases will help understand how separate functions are meant to respond in different circumstances from the actor's perspective viewing the interface. If referenced to whilst implementing a function, the number of errors found in the testing of that function should be reduced as all the possible scenarios in which a user can use that function and the way it should respond to user commands would have been already explored.

Capturing an object's image

Basic flow:

1. User selects the menu option to take an image
2. User presses a button or touches the camera view
3. System undergoes image capture of the current camera frame
4. System displays a message indicating the image has been saved

Viewing the library

Basic flow:

1. User selects menu option to navigate to the library view
2. System replaces the finder view on the main activity with the library view

Adding an image in the library

Basic flow:

1. User selects menu option to add an image
2. System redirects to Android Media Gallery
3. User selects the image to add from the list in the Gallery
4. System prompts user to enter a text label for the image
5. User enters a name into the text entry field
6. User selects the confirmation button for the name
7. System adds image and text to the library list
8. System redirects back to the library view

Alternative flows:

A. No image data saved in Gallery:

1. User selects menu option to add an image
2. System redirects to Android Media Gallery
3. No image data can be seen, user selects cancel option in Gallery
4. System redirects back to the library view

B. User does not wish to add a name:

1. System prompts user to enter a text label for the image
2. User selects cancel option on the text label pop-up
3. System adds image and null text to the library list
4. System redirects back to the library view

Removing an image in the library

Basic flow:

1. User selects menu option to delete an image
2. User touches a library entry to delete
3. System deletes the image and text for that entry
4. System updates the library list so that the entry can no longer be seen on the interface

Alternative flows:**A. No entries are in the library:**

1. User selects menu option to delete an image
2. System does not respond to an on touch delete request in the empty library list until an entry is added

B. User tries to delete the final library entry whilst tracker is still active

1. User selects menu option to delete an image
2. User touches the last library entry to delete
3. System responds with an error message stating the library cannot be empty whilst the tracker is on

Cropping an image in the library**Basic flow:**

1. User selects menu option to crop an image
2. User touches a library entry to crop
3. System sends image data of that entry to the Android crop screen
3. System redirects to Android crop screen
4. User drags corner controls to crop the image
5. User selects the save button
6. System gets the data from the crop and replaces the relevant entries original image with the cropped image
7. System redirects back to the library view

Alternative flows:**A. No entries are in the library:**

1. User selects menu option to crop an image
2. System does not respond to an on touch crop request in the empty library list until an entry is added

Viewing the recogniser/finder**Basic flow:**

1. User selects menu option to navigate to the finder view
2. System replaces the library view on the main activity with the finder view

Matching library images with camera frames**Basic flow:**

1. User selects menu option on finder screen to track objects in the library
2. System matches library images with each camera frame
3. System highlights positive matches on each frame in the camera view using a graphical overlay containing a box around the object and name at the centre

Alternative flows:

A. No entries are in the library

1. User selects menu option on finder screen to track objects in the library
2. System displays an error message stating that the library is empty

B. User wishes to stop matching

1. User selects menu option on finder screen to stop tracking objects in the library
2. System finished processing on the last frame received
3. System displays recognition results of the last processed frame
4. System stops processing further frames for matches between library images

Software Work Models – User Interface & Background System

Work models help to give structure to implementation, so that a pattern of outputs can be chosen and followed to most suit the problem at hand. The execution of such vast amounts of work for a system cannot be humanly comprehended without thinking of a work model. Improvising as to what kind of work should be done when and how could lead to running out of time on parts of the system when closing towards the deadline. This can be avoided with the consideration of appropriate work models for the task and achieve end results of a higher grade.

The background system, meaning the coding of the functional requirements, will be implemented using an incremental development model. This will involve breaking down the system into subtypes, implementing and testing each subtype whilst layering the functionalities on top of one another to eventually result in a full system. Incremental development was chosen due to the size of the system, time given to implement and “safety” of the model itself. In example, given the fairly short period of time to implement it is vital that the core capabilities of the system (the matching process between training and camera frame) can be seen to work to some degree before moving on to less essential functions. An incremental model supports this by allowing the core capability to be its own subtype, implementing and testing it first to check that it works thus assuring safety that the core functions do perform by themselves before integrating with other less important subtypes.

Furthermore the incremental model was chosen due to the lack of disadvantages it has in the context of this system project. For instance, the task and time invested to break down the system into sub-functions has already been achieved in majority through creating the functional requirements. Delegating the sub-functions out which may have been another large task in a group implementation is not necessary when there is only a single programmer. Moreover, through prior research the system architecture has already been envisaged and requirements drawn from those conclusions which are a necessary step needed when following the incremental model. The extra time spent testing each segment may add some extra time when compared to implementing a complete and untested system, but the assurance that each component works is a good compensation and may eventually save time later on as you do not need to scan the entire code looking for solutions to errors once the system is complete.

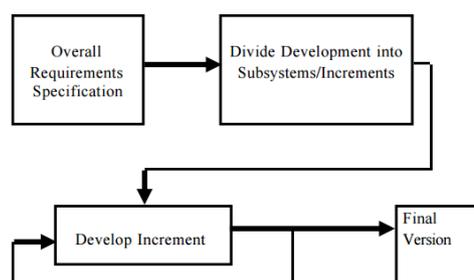


Fig. 3. Incremental Development Model

For the user interface in terms of layout, usability and aesthetics a prototyping model was chosen to be followed. This is because as the part of the system with exposure to the user audience it seemed logical to follow a user based implementation pattern. The prototyping model involves creating a prototype that is made using fewer resources than that of the real implementation, the prototype is tested to gain initial feedback and the UI changed accordingly before investing fully into implementing the real interface. Although the model will not strictly be followed as there will probably only be time to perform a single iteration of prototype testing, gaining early feedback should result in better, more user centred decisions when approaching the interface that will be integrated into the system. There is also no worry that users will think the system is 'nearly built' or will be reluctant to wait for the finished product, which are often disadvantages of prototyping, as there is no planned commercial release of the system. Furthermore, creating prototypes and documenting the changes may result in harder project management in a group implementation but should be easily overcome when there is only a single human resource containing all the knowledge about the system.

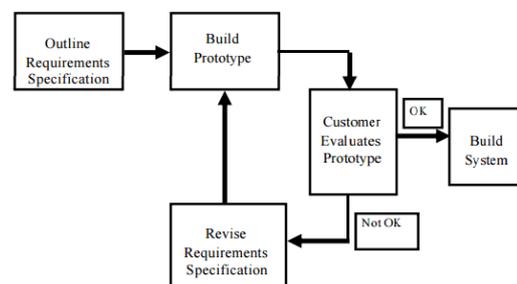


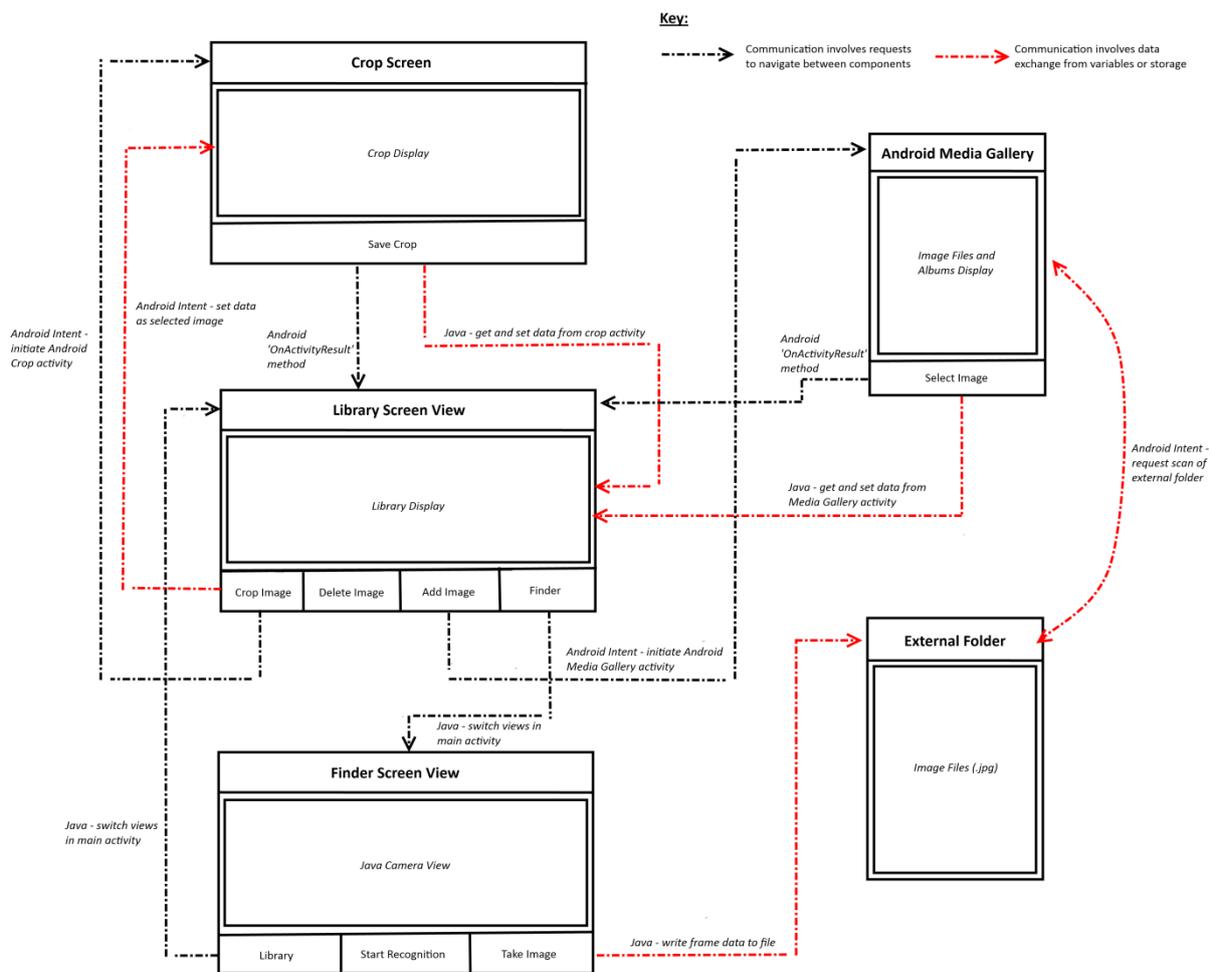
Fig. 4. Prototype Development Model

System Architecture

This section contains a high level diagram of the system architecture in terms of its interactions between different applications or components and the data exchanges that take places. The architecture diagram was created to be a useful reference when coding sub-functions, such that the picture of the entire system and how it interacts is not lost when focusing on specific system parts.

When labelling communication arrows with "Android Intent", "activity" and "Android 'OnActivityResult' Method"; an Android intent is an abstract message that can be sent to Android specific applications to instruct them to perform specific functions dependent on what has been sent in the intent and which Android component it was sent to. An 'activity' in android refers to the component of the application that provides the screen and the UI in which the user interacts with. The main activity represents the activity for the object recogniser system itself and must be created and managed as part of the system code whilst activities for Android applications like Media Gallery are already generated. Android application activities can be called through an intent from the main activity in order to replace it on screen, giving new functions available to the user depending on which applications activity is called. The 'OnActivityResult' method lies within the main activity and is called from the Android application on exit of its own activity. The method handles the return of data used whilst away from the main activity, allowing on return for the programmer via Java code to handle result data gained whilst the system was displaying the other application.

Finder and library screen are documented as 'views' and not separate screens as they are both part of the same main activity. They are effectively the same screen but with different elements on them depending on which view is active. Views were chosen over two separate screens in this case so that they could share image data more easily (such as passing images stored in the library to the matcher) using the same variables without having to continuously send intents to one another.



Implementation

Complications when implementing

Accessing the Android Media Gallery

The Android Media Gallery was chosen to store saved images taken with the camera in preference to only storing in a separate location on the phone for consistency purposes, that a user would expect if taking a photo with the default camera would be stored in the Media Gallery so should the same functionality for this application. It also made handling the image storage and selecting appropriate images to add to the library easier as the interface and storage management was already implemented for the Media Gallery. However, during implementation there were complications with accessing and saving to the Media Gallery that took some time to overcome. At first the image was stored in an external location, then through a media scanner intent populated with the image data taken from the external file location, a broadcast can be sent out containing the intent directed at the Media Gallery broadcast receivers. This then forces the media scanner to scan the file into the Gallery.

This method in theory worked but was inefficient in practice, as when testing the system whilst programming, whenever an image was taken only occasionally could it be seen in the Media Gallery. The Smartphone would have to be restarted to force a complete phone scan in order for all the previously taken images to appear back into the Gallery. This is a major usability error as having to restart the device in order for the system to fully function most times when an image is taken takes up a user's time and affects other open applications that then have to be closed as a result. To eventually solve this problem the method had to be re-implemented such that no broadcasters or intents were used. Instead the data taken (for organisational purposes), file type and file data were directly stored into a Content Provider. A Content Provider is an in-built Android class that manages a repository of data (in this case the 3 data sets mentioned) in one application for use in another (the Android Media Gallery in this case). The method 'insert' in class Content Resolver then sends these values as a row of data entries in a table and inserts it into the Media Gallery directly without the need to scan. This new method proved successful and solved the previous error that could have been detrimental to the usability of the system.

Old method for accessing Android Media Gallery

```
public void AddToLibrary(final String fileName){

    Log.i(TAG,"Adding to Gallery");
    Intent mediaScanIntent = new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE);
    File f = new File(fileName);
    Uri contentUri = Uri.fromFile(f);
    mediaScanIntent.setData(contentUri);
    sendBroadcast(mediaScanIntent);

}
```

New method for accessing Android Media Gallery

```
public void AddToLibrary(final String fileName){
    ContentValues values = new ContentValues();
    values.put(Images.Media.DATE_TAKEN, System.currentTimeMillis());
    values.put(Images.Media.MIME_TYPE, "image/jpeg");
```

```

        values.put(MediaStore.MediaColumns.DATA, fileName);
        getContentResolver().insert(Images.Media.EXTERNAL_CONTENT_URI, values);
    }

```

Native C++ code

OpenCV is written in optimised C/C++ but contains a Java interface, a vast majority of the documentation on the Android OpenCV library relevant to the system such as using feature detection and matching was therefore also written in C++ language. This meant that the application could either use native C language alongside the Java code, such that the documentation can more easily be translated and applied to solving matching problems, or it would have to be rewritten from C++ to Java using the interface OpenCV provided but had little documentation on. The inherent benefit of using C++ is the time it saves at first from not having to be rewritten. However, using C++ may lead to future problems when needing to tweak functionality or solve errors as there was little knowledge or experience in writing in C++ when compared to Java. It was decided that although it was much easier to transfer documentation straight to a C++ file in the Android Java package, using the Java interface would benefit much more when needing to understand and edit its contents. Furthermore, the time taken to re-write the code could have still been outweighed by the time it would take when trying to edit and add to a language with limited experience in.

SURF with OpenCV Android library

Despite SURF feature detection being part of the standard OpenCV library, when the Android OpenCV library was installed it was found that SURF detection was not included in the package due to patent issues [17]. As previously discussed SURF detection was initially chosen over FAST and ORB detection as it was both supposedly scale and rotation invariant yet faster than SIFT. As a result of this, the next highest priority detection method from research, ORB, was selected as a replacement. The result should be a faster matching algorithm that has less performance maxima for certain contexts due to the lack of robust scale invariance.

Critical code documentation

Storing images in the library

When an image is selected from the Android Gallery and is given a label, the bitmap data from the image and its label represented as a string are stored in two ArrayLists (`image` and `descrip`) of the given type at the same index '`Position`'. The application's library is then essentially an Android ListView represented as an Nx2 table through an adapter. Each row in the table stores a TextView object and an ImageView object that are populated with the bitmap data and string data from the two ArrayLists respectively at index '`Position`', meanwhile N is the size of the ArrayLists (i.e. the number of images added to the library). The adapter acts as platform for the ListView to display and gain access to the table data items and is also what forces the ListView to refresh its display whenever a Bitmap is added, removed or cropped from the ArrayList.

Starting the matching process and retrieving library images

When 'Track objects in library' is selected the systems camera display enters the feature matching mode. Instead of just displaying each camera frame in RGBA format with no prior processing (view mode set to `VIEW_MODE_RGBA`) the system now retrieves images in the library to undergo a matching process for each camera frame (view mode set to `VIEW_MODE_FEATURES`). To do this the system simply cycles through the bitmap ArrayList until it reach the last data item in the list. In turn the system gets the Bitmap data at a given index ,converts it to a compatible Mat format

that the OpenCV library uses to perform matching, undergoes the matching and necessary highlighting functions before moving on to the next index with the same frame. The frame and any relevant highlights on top are only shown in the camera display once all the bitmap ArrayList has been cycled through.

```

case VIEW_MODE_RGBA:
    mRgba = inputFrame.rgba();
    break;
case VIEW_MODE_FEATURES:
    mRgba = inputFrame.rgba();

    ... Selecting type of detector, matcher and extractor to use in the matching process ...

    for (int x = 0; x < image.size(); x++){
        Bitmap bm = image.get(x);
        Mat ImageMat = new Mat ( bm.getHeight(), bm.getWidth(),
            CvType.CV_8U, new Scalar(4));
        Utils.bitmapToMat(bm, ImageMat);

        ... Matching and highlighting ...
    }
Return mRgba

```

Finding matches between training image and camera frames

The matches that are created for pairs of feature points between library/training image and camera frame depend on the type of matcher as well as the kind of descriptor (either binary or real value) used. First all the ORB features of both training image and camera frame are calculated using the “detect” method of OpenCV and stored in Mat arrays “points2” and “points” respectively. These points are then computed to binary descriptors (multidimensional vectors) using the OpenCV “compute” method so that they can be matched. Finally, the descriptors of the training image are matched with the most relevant descriptor in the camera frame as a result of using the brute force descriptor matcher. DMatch stores the camera frame descriptor index and train descriptor index for each match and the distance (how close they are to perfectly matching each other) between them.

```

FeatureDetector fast = FeatureDetector.create(FeatureDetector.ORB);
DescriptorExtractor Extractor = DescriptorExtractor.create
(DescriptorExtractor.ORB);
DescriptorMatcher Matcher =DescriptorMatcher.create(DescriptorMatcher.BRUTEFORCE);

MatOfKeyPoint points2 = new MatOfKeyPoint();
fast.detect(ImageMat, points2);

MatOfKeyPoint points = new MatOfKeyPoint();
fast.detect(mRgba, points);

Mat descriptor1 = new Mat();
Mat descriptor2 = new Mat();
MatOfDMatch matches = new MatOfDMatch();

Extractor.compute(ImageMat, points2, descriptor2);
Extractor.compute(mRgba, points, descriptor1);

Matcher.match(descriptor1,descriptor2, matches);
List<DMatch> matchesList = matches.toList();

```

Selecting “good” matches

Once all the matches between the training image of the object and the camera frame have been computed using the OpenCV library, it is still necessary to prune the set of matches. This is because the BRUTEFORCE matching method OpenCV uses will match all descriptors of the training image with the descriptors most like it in the camera frame. You could therefore get a pair of descriptors that are matched to be most similar but in reality are vastly different and do not reflect features of the true object at all. Without selecting good matches the system will try to highlight everything that is most similar to the image of the object that can be seen in the frame and will therefore not successfully recognise when the true object is not in frame.

Matches are pruned using the “distance” value stored with every matching pair as part of the matching method; the distance value is a measure of similarity between the two descriptors. As previously discussed in the research, using ORB creates descriptors that are binary vectors; the distance between the two is measured using the Hamming technique. To prune the system first finds the maximum and minimum distances in the set of matches found. Then the system cycles through every match, only adding matches to a separate “good” set that are less than $X * \text{minimum distance}$ where X is some constant. The constant chosen must be decided carefully, as too high will let too many bad matches through and highlight points on too many irrelevant objects. Meanwhile too lower constant will make slight deformations in camera position or object placement from the original training image result in the object not being recognised despite being in view. The constant “2.5” was chosen to be the best value for the system after testing with values near the OpenCV recommended distance threshold of 3 [23]. The result of pruning is a set of ‘good’ matches containing the indexes of the pair of descriptors and their distance values that matched and passed the good matches threshold in an OpenCV MatOfDMatch structure.

```

Double max_dist = 0.0;
Double min_dist = 100.0;

for (int i = 0; i < matchesList.size(); i++) {
    Double dist = (double) matchesList.get(i).distance;
    if (dist < min_dist)
        min_dist = dist;
    if (dist > max_dist)
        max_dist = dist;
}

LinkedList<DMatch> good_matches = new LinkedList<DMatch>();

for (int i = 0; i < matchesList.size(); i++){
    if (matchesList.get(i).distance <= (2.5 * min_dist)) {
        good_matches.addLast(matchesList.get(i));
    }
}

MatOfDMatch goodMatches = new MatOfDMatch();
goodMatches.fromList(good_matches);

```

Highlighting the object in the camera frames

As the structure `MatOfDMatch` only stores the indexes in both training and camera frame of the points that match, it is necessary first to get the point values at those indexes in order to properly map the object in the training image onto its relative position in the camera frame. This is done by cycling through the `good_matches` `MatOfDMatch` with the indexes of all good matches previously

filtered, getting their point values at that index and storing them in a list. This list then needs to be converted into a `MatOfPoint` format for OpenCV to map them.

The mapping and highlighting code is surrounded by an if statement that does not execute unless the number of good points found from matches is greater than Y, where Y is yet another constant that controls whether or not an entity is mapped highlighted on top of the frame. A minimum value of 4 is needed in order to perform the mapping process and plot a rectangular outline however this integer was raised to 10 for this system in order to avoid highlighting objects that have features close to the training image but are not the training image object.

Mapping the points and highlighting the outline of the training image into its correct relative position in the camera frame is achieved mainly using the OpenCV library methods “`findHomography`” and “`perspectiveTransform`”. “`findHomography`” is used to produce the most probable perspective transformation/ homography matrix between the good points that matched in the object image and the camera frame. It achieves this by using a RANSAC (random sample consensus) approach [36], distinguishing in the camera frame inliers (points that resemble the object/training image) from outliers by iteratively and randomly sampling many different subsets of point pairs until the end of the sampling and the best subset (the subset which has most inliers) is then used to produce the resultant homography matrix. RANSAC was chosen over other OpenCV library methods such as default “0” and “LMEDS” as it is the only method that works well with any ratio of outliers to inliers [24]. Which when considering the amount of potential noise the system may have to transform onto (as it will be unlikely the camera frame will be in a position identical to the object image in the library), made RANSAC the best option.

This homography matrix is then used along with a 5x1 matrix “`tmp_corners`” (where the first four rows represent each corner point of the training/library image and the last row is the centre point of the library image) to undergo the actual transform of the points in “`tmp_corners`” onto their relative positions in the camera frame. The transformed points are stored in the output array “`scene_corners`” in the same index they started at in “`tmp_corners`” (for example the last index in “`scene_corners`” is the centre point of the other transformed corners etc.). Digital lines are then drawn between the points of each transformed corner in the scene using OpenCV method ‘`line()`’ and the string label of the library image is placed at the centre transformed point so that the mapping can be seen on the camera frame and the object recognised by the user.

```
List<KeyPoint> keypoints1_List = points.toList();
List<KeyPoint> keypoints2_List = points2.toList();

LinkedList<Point> objList = new LinkedList<Point>();
LinkedList<Point> sceneList = new LinkedList<Point>();

for(int i=0;i<good_matches.size();i++){
    objList.addLast(keypoints2_List.get(good_matches.get(i).trainIdx).pt);
    sceneList.addLast(keypoints1_List.get(good_matches.get(i).queryIdx).pt);
}

MatOfPoint2f obj = new MatOfPoint2f();
MatOfPoint2f scene = new MatOfPoint2f();
obj.fromList(objList);
scene.fromList(sceneList);

if (objList.size() > 10){

    Mat H = Calib3d.findHomography(obj, scene,Calib3d.RANSAC, 5);
    Mat tmp_corners = new Mat(5,1,CvType.CV_32FC2);
    Mat scene_corners = new Mat(5,1,CvType.CV_32FC2);

    tmp_corners.put(0, 0, new double[] {0,0});
    tmp_corners.put(1, 0, new double[] {ImageMat.cols(),0});
```

```

tmp_corners.put(2, 0, new double[] {ImageMat.cols(),ImageMat.rows()});
tmp_corners.put(3, 0, new double[] {0,ImageMat.rows()});
tmp_corners.put(4, 0, new double [] {(0 + ImageMat.cols())/2,(0 +
ImageMat.rows())/2 });

Core.perspectiveTransform(tmp_corners,scene_corners, H);

Core.Line(mRgba, new Point(scene_corners.get(0,0)), new
Point(scene_corners.get(1,0)), new Scalar(0, 255, 0),4);

Core.Line(mRgba, new Point(scene_corners.get(1,0)), new
Point(scene_corners.get(2,0)), new Scalar(0, 255, 0),4);

Core.Line(mRgba, new Point(scene_corners.get(2,0)), new
Point(scene_corners.get(3,0)), new Scalar(0, 255, 0),4);
Core.Line(mRgba, new Point(scene_corners.get(3,0)), new
Point(scene_corners.get(0,0)), new Scalar(0, 255, 0),4);

Core.putText(mRgba, descrip.get(x), new Point(scene_corners.get(4,0)), 1,2,
new Scalar(0,255,0), 2);
}

```

System User Interface, Testing & Evaluation

UI Prototype and System Usability Testing: Think-Aloud User Evaluation

The user interface testing phase consisted of following protocol for a think-aloud user evaluation. The test itself was performed on the paper prototype UI, a paper prototype was chosen over using prototype software due to the speed of production in comparison and relative simplicity of the UI at hand. Testing was done on a prototype in order to identify and solve any major usability errors before committing high quantities of resources integrating it with the Android Java system.

The test itself was carried out on three potential users, one male aged 20 and two females ages 21 and 54. From prior interview the persons of age 20 and 21 technical experiences with mobile software were no greater than frequent phone use for texting, internet browsing and social media applications thus representing an average non-professional IT capability for their age. The female aged 54 had limited technical experience with mobiles, only occasionally texting and using internet browsing but did not use any phone applications.

The test was taken inside a 5x4 metre room, the noise conditions inside were quiet and the room contained a desk, chair, two paper prototypes (one for demo purposes), TP-330 Cam Link camera stand, list of 3 tasks and a Motorola Moto G mobile device using default software for video and audio recording . A facilitator and human computer were also present in the room during the test. The facilitator was tasked with activating the audio/video recording, briefing and prompting the user, demonstrating the pilot example as well as setting up the paper prototype. Meanwhile the human computer was responsible for manipulating the paper interface to respond as the system would to the user's interactions with the prototype.

The test was conducted as follows; only one user would be in the test room at any time taking the test. Before a new tester had entered the room the paper prototype would be set up by the facilitator to resemble the starting screen of the application, and the mobile device would be set on the camera stand to view the entire prototype on the desk. One of the users would then be invited into the room to start their test and asked to take a seat next to the paper prototype. The facilitator then explained that the experiment is recorded by video and audio via the mobile device, and that the user is to perform a think-aloud walkthrough for 3 tasks on a paper prototype for a new mobile application aimed at tracking objects through the mobile's camera. It was explained that the human computer would be acting as the system would in response to their actions. The facilitator then demonstrated an example think-aloud walkthrough on the demo paper prototype and task with the human computer. Once finished the facilitator then asked the user to perform a mock task on the demo paper prototype as well, treating it as if it were a real task and prompting the user to think aloud their actions when necessary. After the mock and demonstration was completed the facilitator then asked if the user was ready to begin the proper recorded task with the paper prototype. All responded yes but if there was a negative response the facilitator would go through mock and demonstration once again. The recording started as soon as the user responded that they were ready, the facilitator would begin the recording by stating the gender and age of the user then told the user the first task. The task list was for subject reference to remind them of their goal whilst the oral announcements by the facilitator were used as marker for the recording to help in analysis. Once the user had completed a task the facilitator would announce the next task. During the entire test the facilitator would prompt the user to speak aloud their thoughts and processes when necessary. Once all tasks were completed the tester was then asked to leave the room.

The results of the tests were 9 video recordings, 3 recordings per tester for each of the 3 tasks. Recordings were analysed by comparing their reactions and ability to complete the tasks using the paper UI in accordance to well-known usability heuristic, 'Neilsen's usability principles'. The principles when compared to each task were also given a compliancy and severity rating, indicating if the UI successfully supports the principle and if not the severity in which it did not achieve the

principle. Below shows the compliancy key, severity table used with a negative compliance, and tabled results that summarises the analysis of all users for each task. T₁, T₂ and T₃ refer to testers aged 20, 21 and 54 respectively.

Compliancy Key:

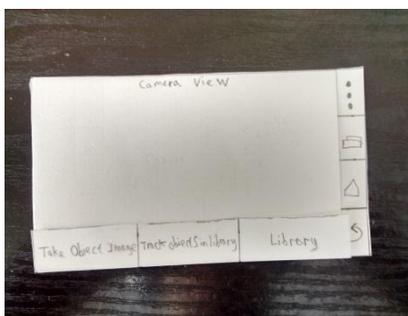
- . '+' = Positive compliance, prototype interface is good and supports the principle well
- . '-' = Negative compliance, prototype interface did not support the principle to some severity extent

Severity Rankings:

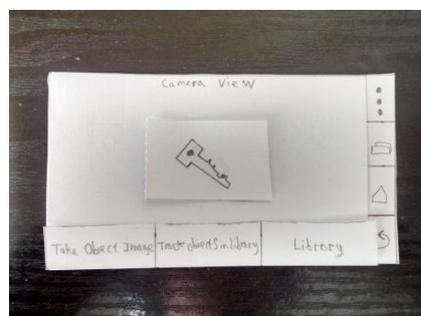
Severity Rankings	
Rating	Definition
0	Violates a heuristic but doesn't seem to be a usability problem.
1	Superficial usability problem: may be easily overcome by user or occurs extremely infrequently. Does not need to be fixed for next release unless extra time is available.
2	Minor usability problem: may occur more frequently or be more difficult to overcome. Fixing this should be given low priority for next release.
3	Major usability problem: occurs frequently and persistently or users may be unable or unaware of how to fix the problem. Important to fix, so should be given high priority.
4	Usability catastrophe: Seriously impairs use of product and cannot be overcome by users. Imperative to fix this before product can be released.

Task 1 Results: "Take an image of an object through the camera, add the image to the library and give it a name"

Paper States Involved:



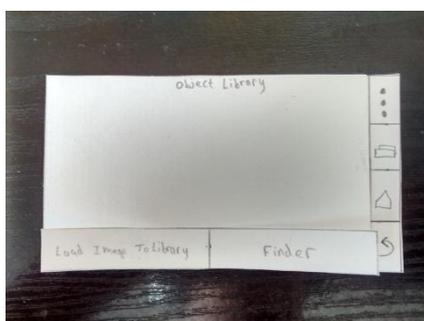
Camera view on launch



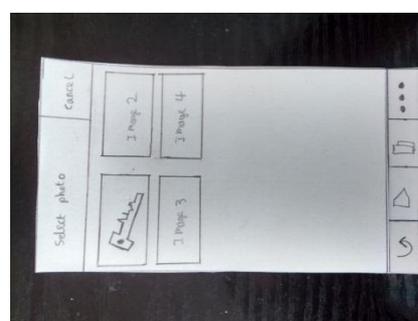
Camera view – Object in scene



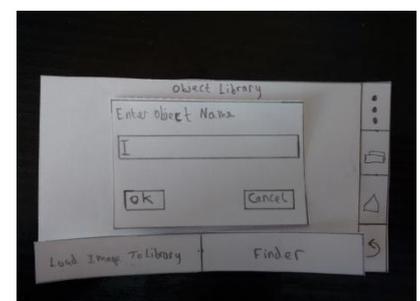
Camera view – Image taken



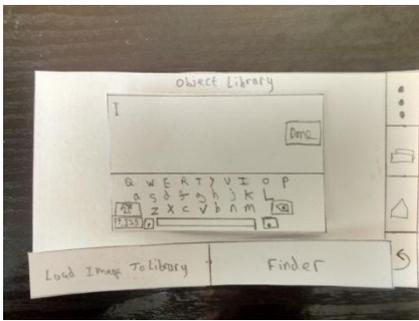
Library view



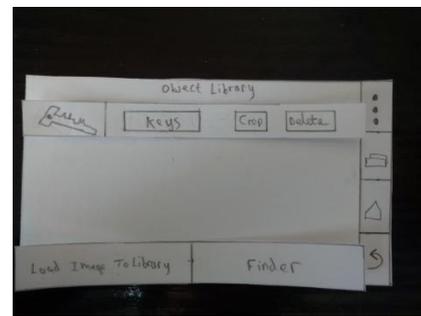
Android Media Gallery



Label entry after image select from Android Media Gallery



Text entry after text field is selected in label entry



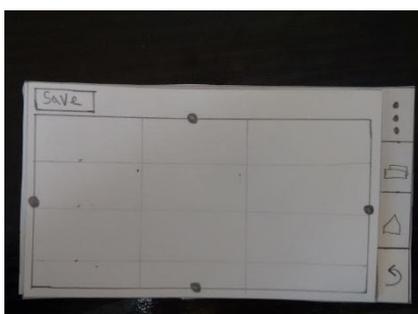
Library view – Single image and label added

Usability Principle	Compliance	Severity	Written Comments
Visibility of system status	-	2	When taking an image through the camera, T ₃ understood that clicking the 'Take Object Image' menu option initialised the camera but took time to realise that touching the screen is what was needed to take the image. Incorporating a pop-up text to explain that touching the screen executes image capture may help users who are used to physical button applications see this functionality in quicker time.
Match between system and real world	+	NA	Labels for menu items were clear enough to the users to not confuse as to their technical functionality. All testers seemed ok with the concept that the 'library' was the place where the images were stored locally.
User control and freedom	+	NA	No errors were made during this task that required the freedom to return to a previous state, in the case that the library was accidentally accessed too early the 'finder' menu item should be enough to return the user back to the camera to take an image of the object.
Consistency and standards	+	NA	All users seemed to be able to distinguish between the different controls available in both library and finder states. Users could recognise menu items that were not relevant to the current task and always pressed the right touch buttons to achieve a sub-goal.
Error prevention	+	NA	As no errors were made and the intent to add an image to the library can be easily cancelled, no additional error prevention such as confirmation windows were deemed necessary for these states and tasks. Including confirmation windows when no errors have occurred during testing could hinder the efficiency and speed of progressing to the task.
Recognition rather than recall	+	NA	The only knowledge the application assumes the user remembers is what image they wish to add to the library.

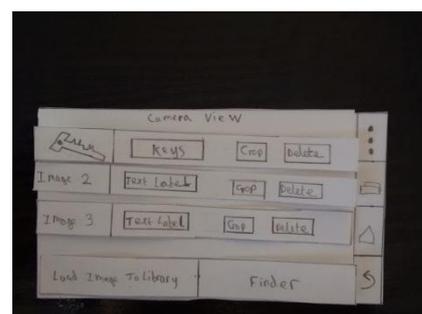
			This is however facilitated by displaying all images on the system through the Android Media Gallery such that they can browse all images until the correct item is found. Other than this the options available in Finder and Library states were suitable enough to avoid confusion.
Flexibility and efficiency of use	+	NA	Although the UI has no shortcuts for experienced users (there is only one way to take a picture and add it to the library), the actions themselves could be completed quickly (it took 3 seconds to navigate through the Android Media Gallery prototype and add the selected image).
Aesthetic and minimalistic design	+	NA	The design was simplistic enough that no user was confused about how to access the library or add an image once in the library state.
Help users recognise, diagnose and recover from errors	+	NA	No errors were made during the task, however if a user was to try and track an object when no image is in the library a message stating 'Error: No Image in Library' has been prepared to pop-up. This should be enough to warn users that they are not currently searching for anything and need to add an image to the library in order to work.
Help and documentation	-	2	As with the "Visibility of system status" principle analysis for this task, no help was made clear when T ₃ did not understand how to take a picture. The pop-up text explaining what to do when the 'Take object image' is clicked should be enough to solve the issues with both these principles.

Task 2 Results: "Add another 2 images, Crop a single image, delete all the images in the library and re-add the original image"

Additional Paper States Involved:



Crop screen



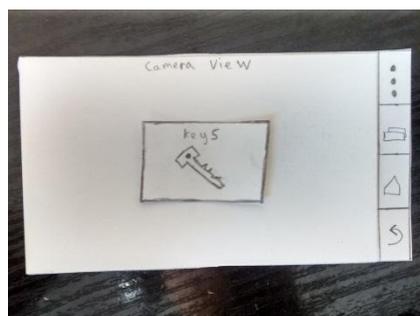
Library view – Multiple images and labels added

Usability Principle	Compliance	Severity	Written Comments
Visibility of system status	+	NA	Library, crop and the Android Media Gallery screens were clear enough for each tester to know which part of the system they were in once the menu item was clicked to navigate towards them. The library list was updated in response to crop, delete and add actions in a way that the users recognised which image they had just cropped, deleted or added without any usability concern.
Match between system and real world	+	NA	All users understood the terminology of 'cropping' an image and that it involved trimming the borders to focus on the object. Having the objects in the library appear as a chronological list (starting from the top being the first image added) was suitable enough that users understood what object images were in the library at any time and could navigate between them with confidence.
User control and freedom	+	NA	User T ₂ accidentally clicked 'Load image to library' a third time, default 'cancel' button in the Android Media Gallery (where the images are saved and selected to go to the library) was acceptable enough for the user who no longer needed to add an image to return to the library.
Consistency and standards	+	NA	With crop and Android Media Gallery screens there was no confusion as to the controls associated with them. All users understood the concept that to crop an image they had to drag the highlighted corners until it fit around the object and that to add an object to the library from the Media Album they had to touch the image.
Error prevention	+	NA	Despite T ₁ commenting that he could have had accidentally miss-spelled the wrong name when adding an image during the text entry state, returning to the applications label entry state after would have allowed him to re-enter the text entry state again by clicking on the text field to fix the error.
Recognition rather than recall	+	NA	Much like in ' <i>Recognition rather than recall</i> ' for task 1, the system only assumes that the user knows what the original image is before adding it. There were no issues in using recall to find the

			image in the Media Album, although recognition would be preferred, this instance of recall facilitated by the Media Album is deemed acceptable enough for this iteration.
Flexibility and efficiency of use	-	2	When re-adding the original image, T ₁ commented that he would have preferred to not retype the name of an item he had previously to save time. Although it would save time to store deleted images and items in a temporary cache such that if they are re-added from the library again the system would recognise its name, this is not just a change to the UI. The extra resources and processing power needed to implement this system change may not be feasible in this implementation. As it was expressed as only a minor inconvenience this functionality could be added in future adaptations.
Aesthetic and minimalistic design	-	2	User T ₂ commented how the library looked 'kind of cluttered' when multiple objects were added. Perhaps taking away the 'crop' and 'delete' buttons from each library entry and instead incorporating them as single menu buttons will remove unnecessarily repeated controls and free up space.
Help users recognise, diagnose and recover from errors	+	NA	There is cancel button in the Android Media Gallery screen to return to the library if necessary, as seen in the ' <i>User control and freedom</i> ' principle for this task it was effective in solving navigation errors for T ₂ guiding the user back to their intended state. For the crop screen, despite there being no cancel button the users were familiar with clicking the save button to return to the library despite how much they cropped.
Help and documentation	+	NA	The controls for adding, delete and cropping were intuitive enough that no user needed additional help or guidance when completing any of the tasks.

Task 3 Results: "Start and stop the recognition search for the object in the camera"

Additional Paper States Involved:



Camera view – Object recognised in view

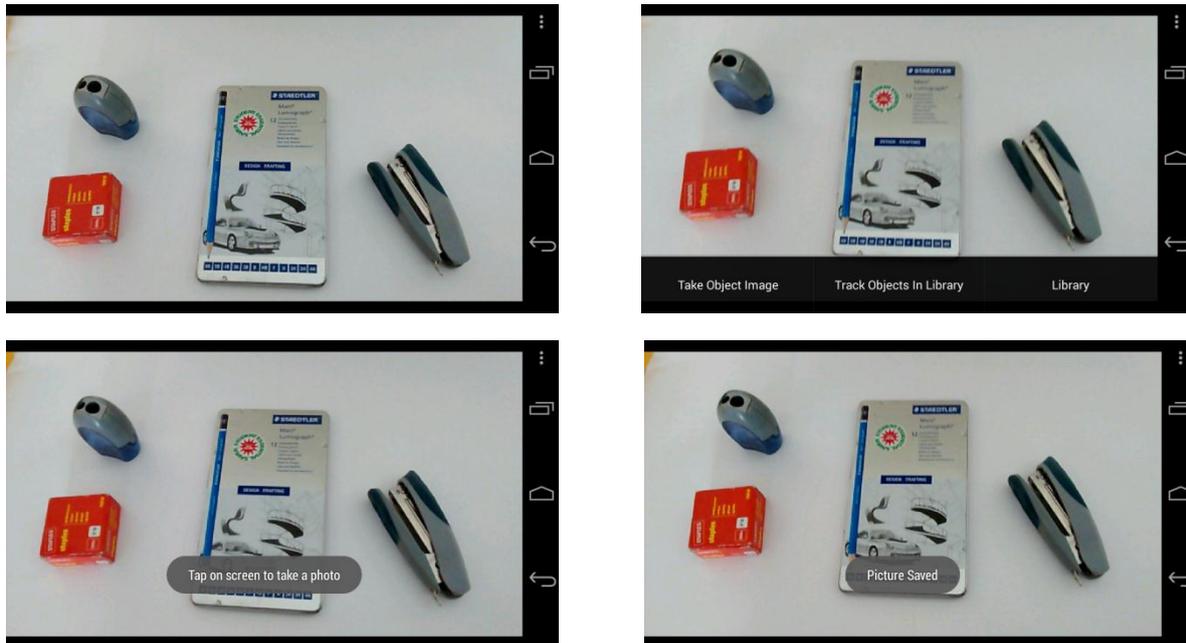
Usability Principle	Compliance	Severity	Written Comments
Visibility of system status	-	3	T ₃ commented that with black being such a popular shade for many objects, highlighting them when recognised with a black outline box and text may seem invisible on the screen in practice. With this in mind the colour of the bounding box and text will have to be changed to a brighter and less popular colour, otherwise the functionality of the system may be undermined by UI choices.
Match between system and real world	+	NA	Using a box and text approach to highlighting recognised objects through the camera did not seem unusual to any of the users despite the comment on its colour and how it might affect performance (see <i>Visibility of system status</i> in task 3).
User control and freedom	+	NA	Users recognised that the menu item 'Track objects in library' changed to 'Stop object tracking' and that they could stop object tracking at anytime.
Consistency and standards	+	NA	As with ' <i>User control and freedom</i> ' and ' <i>match between system and real world</i> ' heuristics for this task users were not confused by the controls for searching an object or indeed the way in which recognised objects would be displayed.
Error prevention	+	NA	No user had accidentally started object tracking during the experiment, however if this case did occur and the library wasn't empty it was deemed that the menu item 'Stop object tracking' would be enough of a counter measure to return to a desired state, especially considering how all users notices the change in the menu item. A confirmation screen may just decrease efficiency when a user intends to execute object tracking.

Recognition rather than recall	+	NA	All images and their relevant labels are listed in the local library such that a user always has a prompt as to what objects they are searching for in the finder. There was no confusion from any user involving a lack of knowledge as to what the application was searching for or the controls needed to press to activate the search.
Flexibility and efficiency of use	+	NA	Users could move from the library screen to the finder and initialise the object tracking in 3.5 seconds. Considering that object tracking will often be turned on and off in application use and that the library will be often accessed in and out of, this speed was concluded to be quick enough to satisfy the consistent change in screen, as none of the users commented negatively on the length of time it took to navigate around the system UI during the experiment.
Aesthetic and minimalistic design	+	NA	With the camera view taking up the entire UI of the finder screen and the menu items visibility being toggle able through in-built android controls, no signs of complaint were made from any of the users about irrelevant controls or features taking up space.
Help users recognise, diagnose and recover from errors	+	NA	See ' <i>Error prevention</i> ' and ' <i>Help and documentation</i> ' for this task.
Help and documentation	+	NA	The change in menu item name from 'Track objects in library' to 'Stop object tracking' was enough help and documentation for the users in the experiment to know whether or not the system was in a tracking mode.

The user evaluation test on the paper prototype revealed three usability flaws in the current design; the lack of system visibility when taking the image of an object, the "crowded" appearance of the library when multiple items are held and the concern of the visibility of the recognition graphics when an object had been successfully matched with a training image. Although there was only enough time to perform user evaluation on three possible audience targets, and that there could be other usability errors spotted if more potential users were tested, the sample size was large enough to provide suitable enough feedback on strengths and weaknesses of the intended UI. These usability flaws will be taken into consideration upon implementing the real interface and logged justifications of changes can be seen in "The Implemented UI" section of this report.

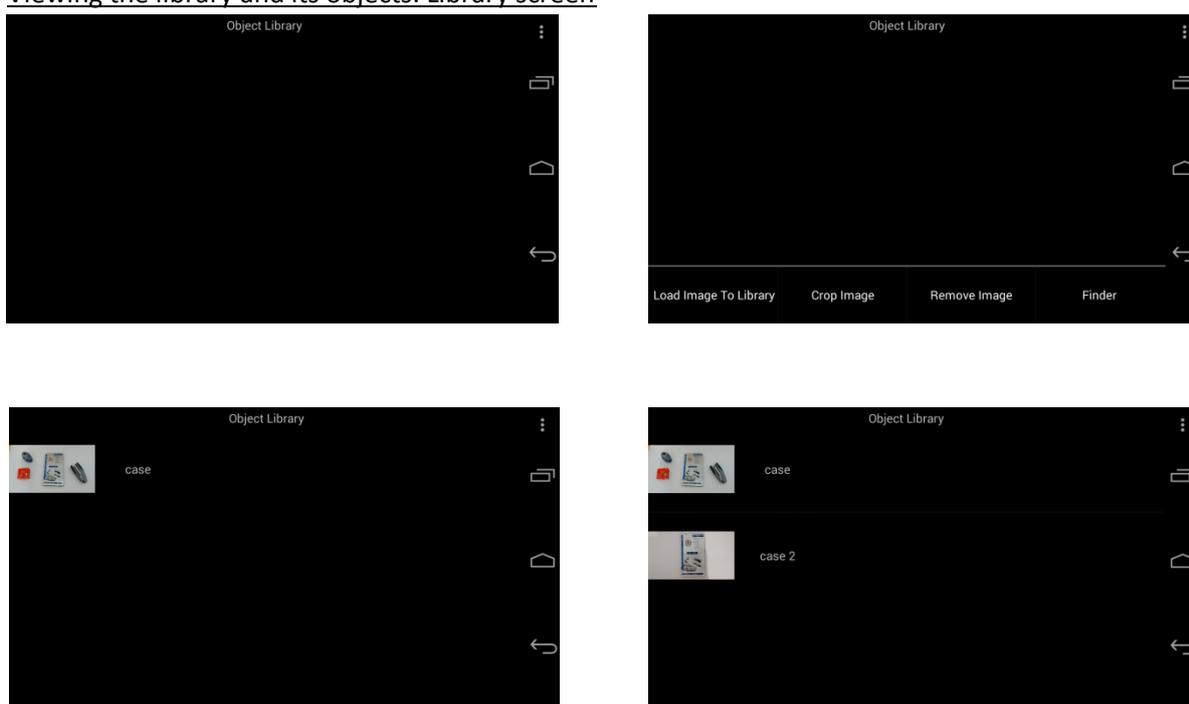
The Implemented UI

Start-up screen and taking an image: Finder screen



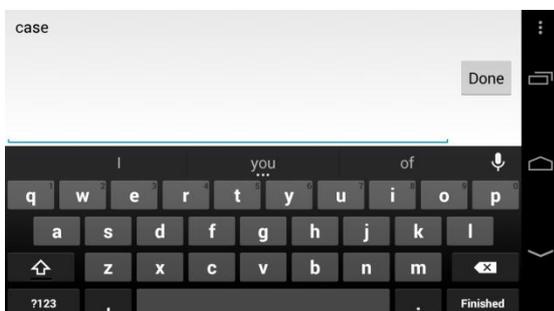
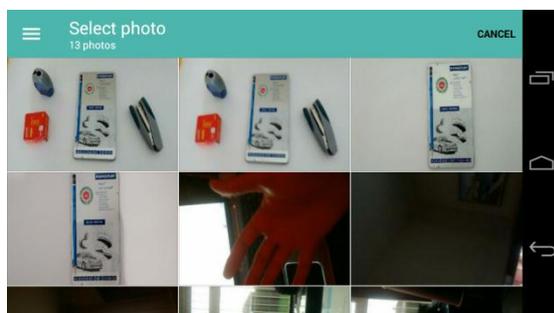
The finder screen follows a centre stage design pattern, whereby the entire UI is taken up by the java camera view. This was chosen as the most appropriate approach as the camera is where all the functionality of the finder screen takes place. The menu items can be toggled to appear so that they only take up space on the screen when necessary, and do not distract the user when they are searching through the camera view to track an object or take a picture. As a result of the prototype testing, pop-up text now appears and fades out after 3 seconds of selecting the 'Take Object Image'. Text instructs the user on how to take an image and when it has been successfully saved to the system therefore helping to solve the issues involving the 'help and documentation' heuristic with the original prototype.

Viewing the library and its objects: Library screen



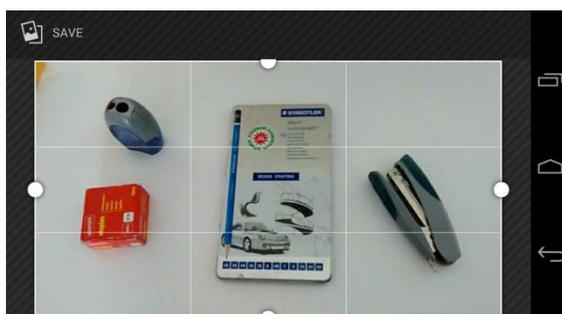
For visibility purposes, black and white has been selected to be the primary colours used in the library and throughout the system. Not only is white on black the best combination to read and grab attention with titles and labels [40], it helps avoid human factors involving the eye and colour wavelengths (such as the use of dark blue appearing as black as the eyes age [7]) which must be considered if the application is too help the elderly who statistically have the poorer sight. For the library itself, a black background also frames the images added and do not detract from the content of the image such that it is hard to tell what images are loaded into the library. Crop and Remove image buttons have been removed from each library entry and added as menu items to create a more minimalistic design after the prototype experiments. The style of the library entries are set like a list such that all items have equal visual priority and can be easily distinguished from one another. Much like with the screen finder, menu items can be toggled on and off so that they do not unnecessarily detract from items in the library.

Adding an image to library: Android Media Gallery and labelling pop-up



The media library follows a grid of equals design pattern, giving each image equal space and rich opportunity for users to browse through available images and select the most relevant item. Meanwhile the labelling pop-up contains a cancel option for “user control and freedom”, supporting users that do not wish to attach a label to the image added.

Cropping an image: Crop screen



Object Tracking enabled: Finder screen



As a result of the prototype experiments, the bounding box and label text has had its colour changed from black to light green. This was so that when an object is successfully recognised, it can be easily detected against most common environments and items. The colour choice also helps with the principle that there should be a 'match between system and real world', as green is often associated with a working system which would be preferable to a red colour that has connotations of system failure [37].

System Experiments

Introduction

Experiments were performed to test the limits of the system in different scenarios in addition to testing functionality and usability. Each report of an experiment consists of an abstract, list of equipment, method (with accompanying diagram if necessary), results tables/graphs and written conclusion.

The phone equipment used in these experiments is limited to one device that operates at a resolution of 864x480 pixels. As resolution affects the detail of features when comparing between library images and screen capture, especially at different distances, certain results from experiments Dolly Capacity, Library Limit and Noise Limit may yield different values with screens of different resolutions.

In some experiments a 'best case' and 'worst case' object is used to try and estimate lower and upper level bounds of the system. The best and worst case objects were selected based on their complexity and kept constant between experiments that used them. The complexity of an object consists of two factors, the first is the number of features on and around the object and the second is the regularity in its shape.

For the best case the object needs to be regular in shape, contain many features and not deform upon being moved. A tin pencil case was therefore chosen for its regular rectangular shape and graphics on the front to provide the large number of features needed. Meanwhile the worst case object needed to be irregular in shape and contain significantly less features than the best case. For this a set of keys on a key ring was selected to provide fewer features to be detected (due to the lack of graphics on the item) and also be non-standard in shape.

Best Case Object



Worst Case Object



Visibility rankings for all experiments are based off a pre-defined chart as shown below:

Visibility Rankings	Description	Example Image
5	Object is clearly recognised, bounding box wraps around object perfectly at the dimensions of the training image with no stutter between frames and text is at the centre of the object.	

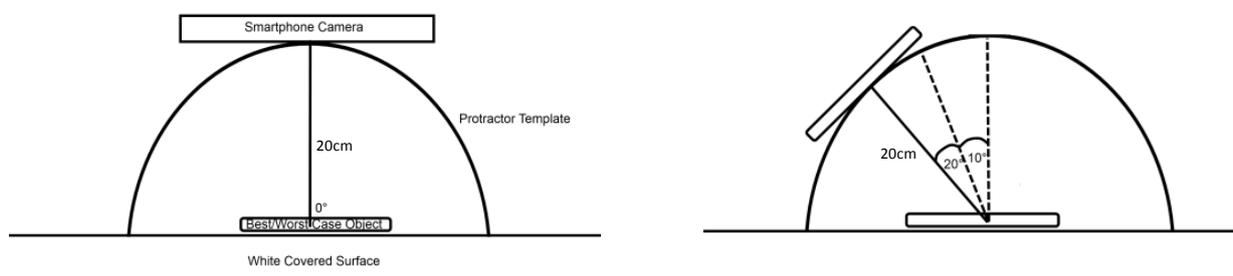
4	Bounding box warps around the object but on occasional frame does not perfectly fit the dimensions of the training image.	
3	Between frames bounding box and text ranges from slight miss-fitting (as in visibility rank 4) to out of place, but still is constantly recognised and has the overlay on the objects relative location.	
2	Object is constantly tracked between frames but bounding box or text has no particular form or location on the recognised object.	
1	Text or bounding box lines can be seen occasionally between frames on the object with no particular form.	
0	No recognition overlay or wrong object has been recognised.	-

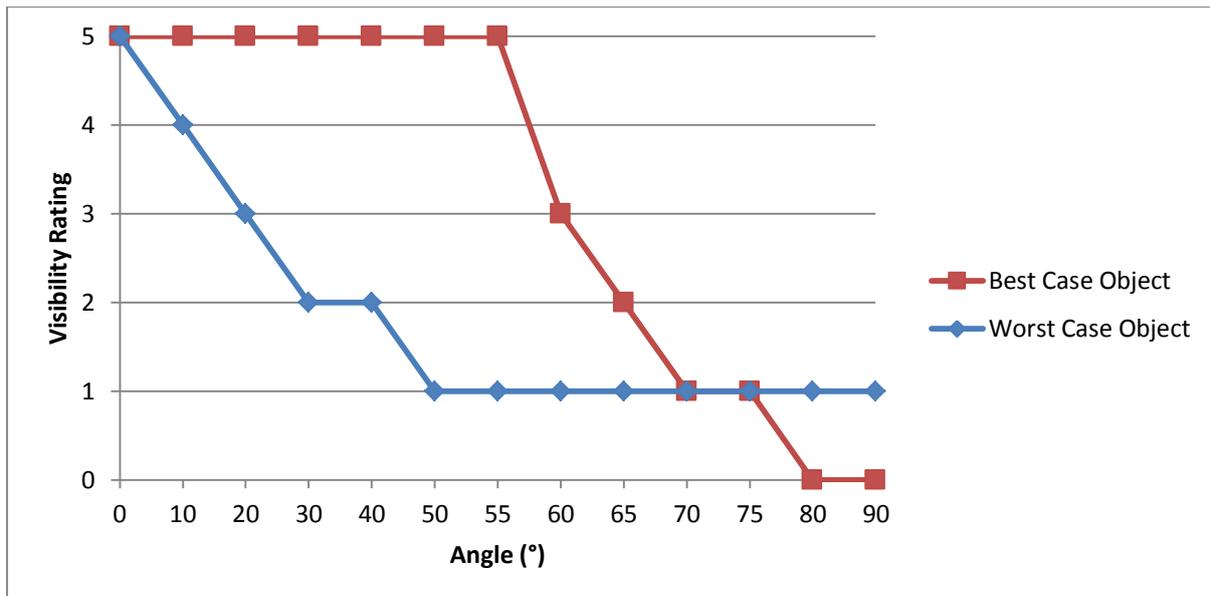
1. Angle of Acceptance

The angle in which the camera can be moved around an object before the system loses visibility of it.

- . Protractor Template
- . 30cm Rule
- . White Cover (used to place on top of the test surface, such that no features of the surface will be picked up in addition to the objects added to the scene)
- . Best and Worst Case Object

The camera was positioned 20 cm away from the best case object, facing perpendicular down the Y plane such that the camera lens was pointing down at the object directly below. The initial image was taken and loaded to the library; this position was then treated as 0° from the centre point of the object where the image was taken. The camera visibility was recorded at 0 , then at $\pm 10^\circ$ increments about the centre point using a template protractor until the object was no longer visible. At each increment the 20cm distance was kept constant. The test was then repeated for the worst case object.





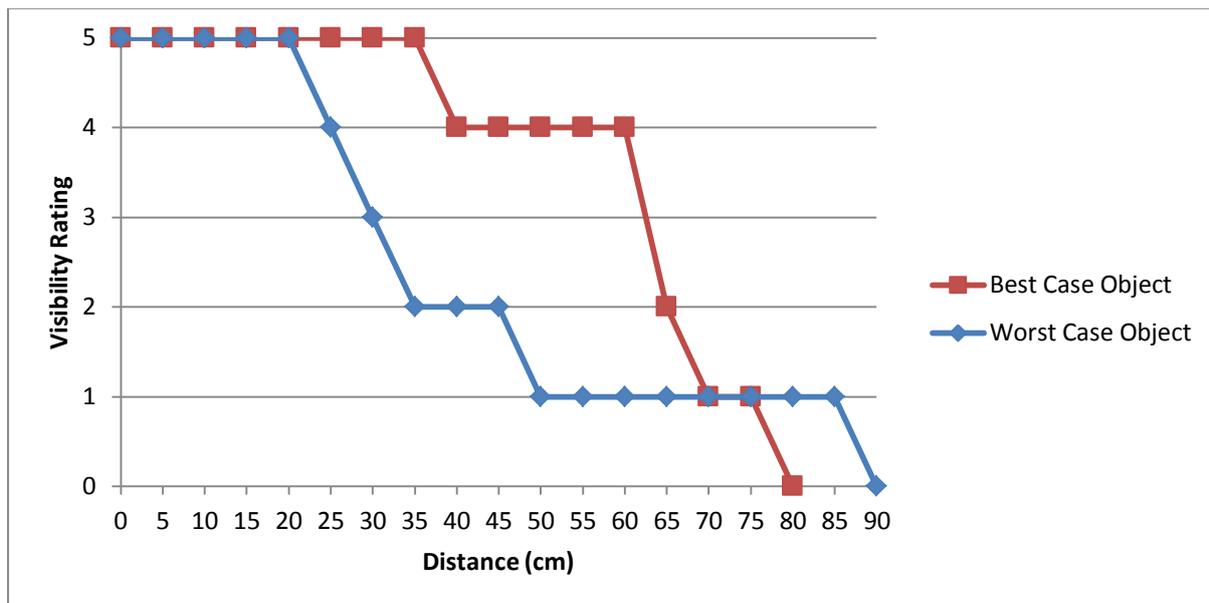
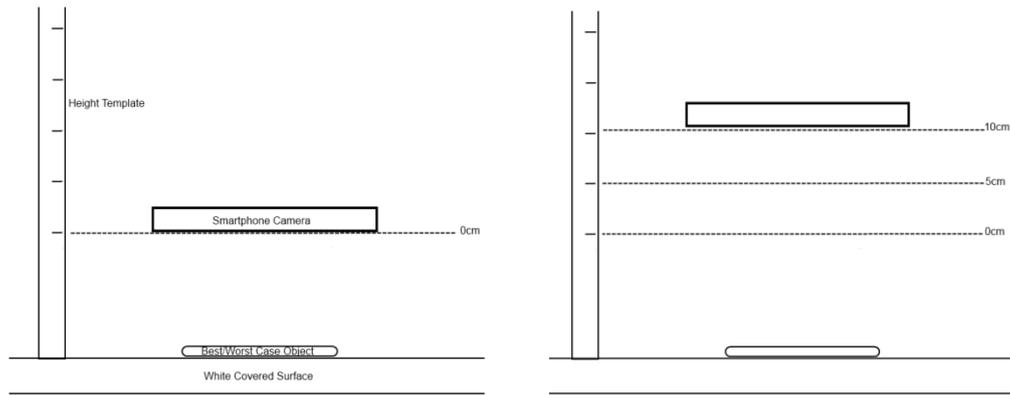
It can be seen that in the best case the system can be rotated up to 55° from the angle of the image in the library and still function at best capacity. However, past this threshold there is a rapid decline in performance dropping to the lowest rank over the next 15°. Meanwhile with a worst case object the system only works perfectly when the angle of the camera view is identical to the angle at which the image was taken. Although the drop from highest to lowest rank is over a far larger degree when in the worst case (50°), the system may operate at its lowest performance for some objects when tilted to 50° yet be optimal for others. With this best case object it is no longer visible to the system past 80° yet the worst case will always be visible up to 90°. This is most likely due to regularity of the shapes; from a 90° or side perspective if the object is flat such as a rectangle all the features on the face where the image was recorded are hidden. On the other hand an irregular object may still have certain parts of the front face showing when viewing from the side. Granting in this case only enough features for the system to recognise at its lowest performance.

2. Dolly Capacity / Object Scale Robustness

The distance the camera can dolly away from an object before losing visibility, (as there is no lens zoom function as part of the system, the only way to decrease or increase the scale of the object in the camera view would be to dolly (physically move the camera not the lens away or toward the object)).

- . Height Template
- . 30cm Rule
- . White Cover
- . Best and Worst Case Object

Like the start of experiment 1, the camera was positioned facing down toward the centre of the best case object. The starting distance away from the object is variable and is set depending on how close the object needs to be to fill the camera view such that its width or height borders the edge of the lens view. The initial image was then taken and loaded to the library; this position is treated as 0cm away from the object. The camera visibility was recorded at 0, then at increments of +5cm away keeping the angle to the object the same throughout. The test was then repeated for the worst case object.



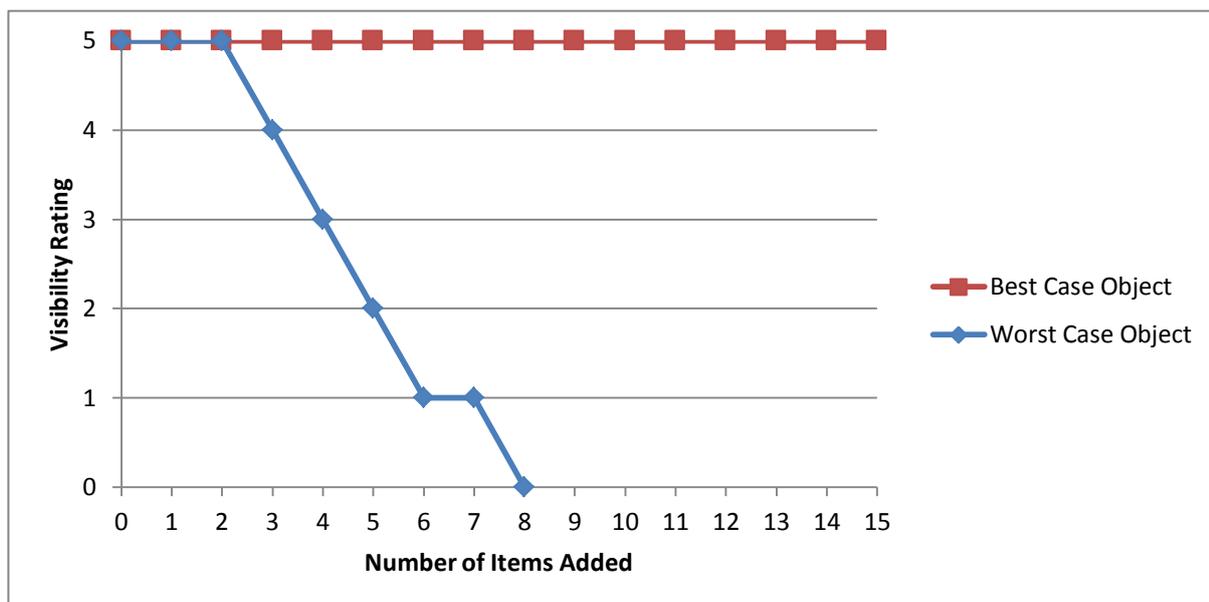
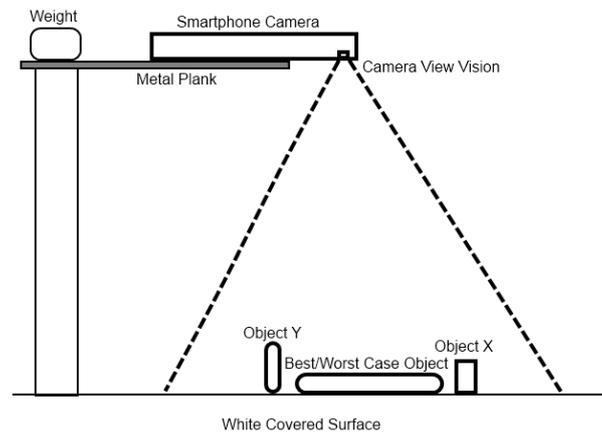
Despite ORB having no scale invariance, there is still some flexibility that the camera can distance itself from an object. From the results 20cm appears to be the threshold for optimal performance in both worst and best cases, at best the system is optimal a further 15cm. Much like with angle of acceptance, there is a steep fall-off point with best case objects whereby visibility sharply falls after taken 60cm away, reducing to the lowest rank over an additional 10cm. In a similar pattern to the previous experiment too, there is only a short time in which the best case object operates at the lowest rank before becoming unrecognisable. Furthermore, the worst case object reduces to the lowest visibility rating 20 cm before its best case counterpart yet similarly will continue to operate at lowest performance further than the best case (10cm). This must once again be due to the irregular shape of the worst case object being more distinguishable at extremes than the graphics and shape on the best case item.

3. Noise Limit

The number of other items that can be in the camera view before either camera fps becomes too low to be responsive or the object is no longer visible to the system.

- . 30cm Rule
- . Metal Plank
- . Weight
- . White Cover
- . Set of objects

During both object experiments, the camera was kept at the same distance to the table surface with the object to be recognised at the centre point of the view. To start the object is alone in the camera view, the library image is taken and cropped to the size of the object, the camera fps and object visibility rating is noted (camera fps is recorded and displayed on the phone using an OpenCV library method). Then one at a time, other non-library items are added from a pre-defined set, positioned within 1cm of another object in scene but not overlapping the library item, recording fps and visibility each time. This continues until the visibility is rated 0, the set has all been placed or the fps becomes too low to be responsive.



In the worst case, the system will only take 3 additional objects being close to the tracked entity in the view before the visibility is affected. The decline in visibility is rather linear when the number of items is greater than 3, falling one rank per every other item added close to the object in the view until at the lowest rank. There is of course a leniency in the results given what particular items are added, but the experiment shows that an estimate of 7 to 8 items will be the maxima in worst cases. When compared to the best case the difference between lower and upper bounds seems wide, this is most likely due to the difference in the number of features and the distinctness of those features between best and worst case. As the less features an object you wish to track has and the more common they are the more likely visibility can reduce by mismatching x amount of features with other similar objects. The best case maintains perfect visibility even after all 15 items in the set have been added, this could be as a result of the graphics on the front of the tin providing many unique

features to the tin itself. To test the limits of a best case object the view was further filled with 15 additional items until the screen was entirely filled with non-library objects, representing a maximum for the number of additional features possible to be added on screen at the fixed distance. When the best case item was then re added it was still recognised at a visibility of 5. It can be therefore assumed that an object matching a best case description will always be spotted at a rank of 5, provided it is within the angle of acceptance and dolly range that yields a result of 5 and that the library article remains uncovered by other objects. Below shows the table result and image of filling the view with maximum items.

Best Case Object		
Objects added to scene	Camera Fps	Visibility Rating
...
30	2.11	5
Image of Result		
		

4. Library Limit

The experiment records two parts, the first is the number of objects that can be stored into the library and simultaneously be in view before any library object no longer becomes visible. The second is the number of objects that can be stored into the library before the system becomes overloaded.

- . 30cm Rule
- . Metal Plank
- . Weight
- . White Cover
- . Set of objects

The camera was set in a fixed position 42 cm over the cover using the metal plank. Objects in the set were added one at a time into the view, with each one being added an image was taken, added to the library and cropped to the perimeter of only that object. With each additional object the camera fps and visibility ratings of each object in the set were recorded. Once any visibility rating of a subset of objects becomes 0 the ratings will stop being recorded, the objects in view will be removed but other items will continue being added to the library and fps noted until system failure.

Objects Loaded in Library	Camera FPS	Visibility Ratings
1	3	{5}
2	1.12	{5, 5}
3	0.65	{3, 5, 3}
4	0.57	{0,4,4,2/0}
5	0.32	x
6	0.30	x
7	0.26	x
8	-	-

Though the objects in the set for this experiment were thought to be distinct, critical problems occurred with the system in terms of the visibility of the library objects as soon as 4 were in both the library and the view. Object₁ seemed to be matched within other features of object₂, the system could no longer distinguish between two separate objects and thus object₁ visibility was noted at 0. Meanwhile object₄ on some frames was found and other frames were not recognised at all, however upon removing one object whilst keeping the library size at 4 the system could once again recognise each item individually. This concludes that although the system cannot minimally function if the number of objects in the library and view is > 4, there is still a capacity to hold more library objects provided not all of them are in view at once. The results of part 1 of the experiment however only give an estimate, the number of variables in terms of different objects that could be in the library as well as camera angles and position could yield different results. For the second part of the experiment, once eight objects were added to the library the system was overloaded and forced to close. This may give a total library capacity of 7 (regardless of whether the 7 library items are in the camera view) but in reality the camera fps becomes too low once 5 objects are in library that a moving camera could be deemed too unresponsive for user purposes.

5. Object Distortion

An experiment to test the angle the object can be orientated around in comparison to the original image before no longer being visible. Documentation on how much of the object can be covered before no longer being visible.

- . Protractor
- . 30cm Rule
- . Metal Plank
- . Weight
- . White Cover
- . Best Case Object (the worst case object was not included in the experiment as its overall shape would change upon rotation, adding more than just a rotation variable influencing the results)

The camera was set in a fixed position 42 cm over a blank surface, the best case object was then placed in the centre of the camera view. An image was taken of the object and added to the library with its size cropped around the item perimeter. Marks were then drawn around the object to reference its starting position and to make a bearing in which to rotate the object. Starting at 0°, the objects visibility rating was recorded before rotating the object +45° about its centre. This was repeated until a full 360° rotation had been made.

Object Rotation (°)	Visibility Rating
0	5
45	5

90	5
135	4-5
180	5
225	5
270	4-5
315	5

The two ways an object could be distorted without physically changing its shape are through rotation and covering. When rotation/orientation is concerned, it can be seen from the table that research was correct, ORB is orientation invariant and therefore an object can be rotated a full 360° from the origin of the library image and still be recognised competently by the system. The amount that an object can be covered and still recognised is too highly variable between objects to be accurately estimated, as it depends where on the object a majority of features are found. The percentage an object can be covered before becoming invisible to the system is also controlled in the code. The code only highlights the object if the number of good matches between training image and camera frame is greater than X (X being hard coded as 10). Therefore, so long as X features of the object are visible to the camera it will be displayed. There is however a trade-off, as the lower X is set to the easier it is for the system to match the training image with an object in the camera frame that is not the library object.

System Functionality Testing and Evaluation

Test cases are used to ensure the functionality performs as expected by checking if a pre-defined set of use cases or steps creates determined results or an error. Test cases were carried out by a single tester who had no involvement in the implementation. Test cases are useful so long as an extensive amount of different procedures for performing similar tasks are considered. The following should represent a large subset of the steps that can be possibly undergone whilst using the system giving a strong indication of how error prone the system is whilst running. However, even if all are passed that does not directly mean that the system is error free, as there could be a minor subset of unconsidered paths to use the system that have not yet been realised and therefore untested.

Test Case ID:	Description:	Preconditions:	Input:
Image_Capture	Take an image of the test object	-	. Test Object (set of keys)
Steps:	Expected Result:	Status:	Comments:
1. Launch the application by clicking the "Object Recogniser" app on the default phone menu	Finder screen should appear in mobile monitor, consisting of the java camera view and Android menu bar.	Pass	
2. From the launch/finder screen, tap the "... " option from the Android menu bar (on the right hand side)	Finder screen menu bar should appear at the bottom of the monitor overlaying the camera view.	Pass	
3. Tap the "Take Object Image" option from the finder screen menu	Finder screen menu bar should close and system message "Tap on screen to take photo" should appear for a 3 second duration.	Pass	
4. Position the camera 20cm away from the centre of the test object, with the camera lens facing down towards the surface, tap anywhere on the camera view	System message "Picture Saved" should appear for a 3 second duration.	Pass	
All steps successfully completed for ID "Image_Capture" with no errors or additional observations.			

Test Case ID:	Description:	Preconditions:	
Image_Add	Add an image to the library	User has launched the application and taken an image of an object	
Steps:	Expected Result:	Status:	Comments:
1. From the finder/launch screen, tap the "... " option from the Android menu bar (on the right hand side)	Finder screen menu bar should appear at the bottom of the monitor overlaying the camera view.	Pass	
2. Tap the "Library" option from the finder menu	Screen should switch to library view, consisting of an empty library (appears	Pass	

	black) with title "Object Library" centred at the top of the screen.		
3. From the library screen, tap the "... " option from the Android menu bar	Library screen menu bar should appear at the bottom of the monitor overlaying the library interface.	Pass	
4. Tap the "Load Image To Library" option from the library menu	Screen should switch to Android Media Gallery view, consisting of albums "Phone", "0" and "Screenshot" with a blue banner on the top of the screen and 'cancel' button aligned to the right inside the banner.	Pass	
5. Select and tap the album "0" from the Android Media Gallery	Albums on the screen should disappear and be replaced with the images inside album "0", this album should only contain the image of the object just taken.	Pass	
6. Select and tap the image of the test object from the album	Screen should return to the library, with a pop-up box on top of the screen titled "Enter Object Name", pop-up consists of a text entry field, "OK" and "Cancel" button.	Pass	
7. Touch the text field on the label pop-up	Screen should be replaced with the Android Text Entry view, consisting of a half blank screen with button "Done" to the right and an Android Keyboard UI underneath.	Pass	
8. Type name "Keys", using the Android Keyboard and select "Done"	Blank screen should update when the keyboard is pressed to reflect the character or keyboard function chosen, once "Done" is selected screen should navigate back to the pop-up box now with the string 'Keys' in the text entry field.	Pass	
9. Tap "OK" on the label pop-up	A row should have been added to the library, consisting of the image	Pass	

	chosen from the album on the left and its given label to the right.		
All steps successfully completed for ID "Image_Add" with no errors or additional observations.			

It is important to consider the alternative flows as well as the basic functional components of the system, so that even in a scenario where a user wishes to return back to a previous screen is tested for errors. This is why even test cases such as "Android_Gallery_Cancel" and "Object_Label_Cancel" are documented even though they only consist of a single step, at some point they could be important for error prevention or recovery and would need to have a return function that is proven to work.

Test Case ID:	Description:	Preconditions:	
Android_Gallery_Cancel	Return from the Android Media Gallery without selecting an image	From test case "Image_Add", performed steps 1 – 4.	
Steps:	Expected Result:	Status:	Comments:
1. From the Android Media Gallery, select and tap the "cancel" touch button	Screen should navigate back to library in the same state it was left upon entering the Android Media Gallery.	Pass	
All steps successfully completed for ID "Android_Gallery_Cancel" with no errors or additional observations.			

Test Case ID:	Description:	Preconditions:	
Object_Label_Cancel	Cancel labelling an object about to be loaded to the library	From test case "Image_Add", performed steps 1 – 6.	
Steps:	Expected Result:	Status:	Comments:
1. From the label pop-up, tap the "cancel" button	A row should have been added to the library, consisting only of the image chosen from the album and no text label	Pass	
All steps successfully completed for ID "Object_Label_Cancel" with no errors or additional observations.			

Test Case ID:	Description:	Preconditions:	
Image_Crop	Crop an image in the library	User has loaded and labelled 2 images from the Android Media Gallery to the library and is on the library screen.	
Steps:	Expected Result:	Status:	Comments:
1. From the library screen, tap the "..." option from the Android menu bar	Library screen menu bar should appear at the bottom of the monitor overlaying the library interface.	Pass	
2. Tap the "Crop Image" option from the library	Library screen menu bar should close.	Pass	

menu			
3. Tap the first library entry row with the image and label in	The first library entry row should highlight in yellow when pressed, on release screen should switch to the Android Crop view containing the image selected, a white rectangle around the perimeter of the image with four white circles acting as crop points at the centre of each rectangle edge and a "Save" button at the top of the screen aligned to the left.	Pass	
4. Click and drag the four crop points around the object in the image	On dragging a point, the edge of the rectangle should move with the relative motion of the drag, highlighting only the portion of the image inside the area of the rectangle (this is the area that is cropped).	Pass	
5. From the crop screen, touch the "Save" button	Screen should switch to library view in the same state as in step 1, except the row selected for cropping should contain the newly cropped image with the same text label.	Pass	The cropped image had successfully replaced the original image in the row selected whilst keeping the relevant label, but the row that was selected which started at the top of the list is now at the bottom of the list.
All steps successfully completed for ID "Image_Delete_02" with no errors and 1 observation.			

For the above test case, tester had noticed the change in index of the selected row to be cropped, this occurs because upon return from the crop screen the cropped image and its original text label are added as a new entry into the library and the corresponding original image with its text label are deleted at the index that the selected row was in. The function executes this way rather than replacing the original image at the selected index due to issues with transferring the index integer from the "onItemClick" method that handles on touch events in the library to the "onActivityResult" method that handles returning from the crop activity. As this does not affect the crop functionality itself and with the library designed to give equal weight to all items it holds regardless of index this observation has been recorded but will not be acted upon in this iteration.

Test Case ID:	Description:	Preconditions:	
Image_Delete_01	Delete an image from the library	User has loaded and labelled an image from the Android Media Gallery and is on the library screen.	
Steps:	Expected Result:	Status:	Comments:
1. From the library screen, tap the “...” option from the Android menu bar	Library screen menu bar should appear at the bottom of the monitor overlaying the library interface.	Pass	
2. Tap the “Remove Image” option from the library menu	Library screen menu bar should close.	Pass	
3. Tap the library entry row with the image and label in	Library entry row should highlight in yellow when pressed, on release the row should be deleted from the library.	Pass	
All steps successfully completed for ID “Image_Delete_01” with no errors or additional observations.			

Test Case ID:	Description:	Preconditions:	
Image_Delete_02	Delete the last image from the library with the object tracker still active	User has loaded and labelled a single image from the Android Media Gallery and is on the library screen.	
Steps:	Expected Result:	Status:	Comments:
1. From the library screen, tap the “...” option from the Android menu bar	Library screen menu bar should appear at the bottom of the monitor overlaying the library interface.	Pass	
2. Tap the “Finder” option from the library menu	Screen should switch to the finder view consisting of the java camera view.	Pass	
3. From the finder screen, tap the “...” option from the Android menu bar	Finder screen menu bar should appear at the bottom of the monitor overlaying the camera view.	Pass	
4. Tap the “Track Objects In Library” option from the finder screen menu	Finder screen menu bar should disappear, no noticeable change in the java camera except a drop in fps (frames per second) making it seem less responsive to camera movement.	Pass	
5. Tap the “...” option from the Android menu bar	Finder screen menu bar should appear at the bottom of the monitor overlaying the camera	Pass	

	view.		
6. Tap the "Library" option from the finder menu	Screen should switch to library view in the same state as in step 1 (with the library containing a single item represented as a row with an image and label inside.)	Pass	
7. From the library screen, tap the "..." option from the Android menu bar	Library screen menu bar should appear at the bottom of the monitor overlaying the library interface.	Pass	
8. Tap the "Remove Image" option from the library menu	Library screen menu bar should close.	Pass	
9. Tap the library entry row with the image and label in	Library entry row should highlight in yellow when pressed, on release a system message should appear with text: "Please stop object tracking before emptying library" for 3 seconds.	Pass	
All steps successfully completed for ID "Image_Delete_02" with no errors or additional observations.			

Test Case ID:	Description:	Preconditions:	
Object_Track_01	Track the test object when the library has no images stored	The library is empty and the user in on the finder screen	
Steps:	Expected Result:	Status:	Comments:
1. From the finder screen, tap the "..." option from the Android menu bar	Finder screen menu bar should appear at the bottom of the monitor overlaying the camera view.	Pass	
2. Tap the 'Track Objects In Library' option from the finder screen menu	Finder screen menu bar should close and system message "Error: Library is empty" should appear for a 3 second duration.	Pass	
All steps successfully completed for ID "Object_Track_01" with no errors or additional observations.			

Test Case ID:	Description:	Preconditions:	Input:
Object_Track_02	Track the test object when the library has an image of the object stored.	From the test case "Image_Capture", performed all steps, user has added the specific test image from test case "Image_Capture" to the library with label "Keys" and is on the finder screen	. Test Object (set of keys)

Steps:	Expected Result:	Status:	Comments:
1. Position the camera 30cm away from the centre of the object, with the camera lens facing down towards the surface	NA	NA	
2. From the finder screen, tap the “...” option from the Android menu bar (on the right hand side)	Finder screen menu bar should appear at the bottom of the monitor overlaying the camera view.	Pass	
3. Tap the “Track Objects In Library” option from the finder screen menu	A green rectangle should appear around the test object in the camera view; the rectangle has a size equal to the dimensions of its image in the library (the closer the crop around the object in the image is the closer the rectangle is in the camera view). The label “Keys” should appear in the centre of the test object in the camera view.	Pass	
4. Remove the test object from the camera view	Camera View should not recognise or highlight anything.	Pass	
All steps successfully completed for ID “Object_Track_02” with no errors or additional observations.			

Eight out of nine test cases were filled with no errors or observations whilst “Image_crop” did have a documented observation it still passed all stages in the test. This shows that the system should be competent at executing the different ways in which its functions can be performed (not considering the small percentage of possible unperceived test cases that could exist but were not thought of.) The main disadvantages of test cases other than the previously mentioned percentage of unknowns are its scalability with system expansions and the time investment. Test cases would have to be archived appropriately if the system were to grow, perhaps transfer data into some data warehouse so that their knowledge is not lost and can be referred to if there were ever new software designers working on a bigger project using the object recogniser as a basis. In this instance there was enough time to create and test the test cases so that the system can be thoroughly checked, if ever there was a tighter time constraint then an alternative could be exploratory testing. Exploratory testing involves giving users a timed window to use and “play” with the system and does not require the prior setup of test case forms or structure (though there may be some guidance involved). This in some cases will generate more error recognition that a standard set of test cases as the tester could explore one of the unknown paths during the time frame; however the results or errors discovered are never certain with exploratory testing and should only replace test cases if a deadline to deliver a tested system leaves little room for formal documentation.

Future Adaptations

As expressed in the introduction, the system was designed as a building block for future possibilities, here will be discussion of the main adaptations if given more time the system could adhere to and what theatrically could be done in order to achieve this.

Helping visually impaired recognise and find objects

The system currently has all the basic recognition functionality needed to help visually impaired persons find or recognise an object, however the interface in its current state is not suitable for use by those with a lesser vision as it relies on visuals to show results. Adding additional features such as audio may make the application more suitable. For instance, instead of just adding a text label when images are added to the library perhaps a voice record option could also be implemented. Android already has a MediaRecorder API available to do this whereby audio can be recorded using the device's microphone, stopped, played back and saved [2]. Then, in addition to a bounding box and text label being displayed on recognised items from training images the system could also play the relevant recording attached to that image in the library. The audio could be played in regular intervals so long as the object is recognised or in view of the camera, or alternatively a command could be pressed to play the recording back again once it has played once. To further this idea the system could have the option of being controlled through voice command, so that it can be operated without needing to interact with the interface. Once again Java has an in-built voice recognition activity that could help with this [10], where on command the Android system will use the microphone to convert speech into a list of strings. You can then use this list to execute certain functions depending if certain keyword strings are in the list or not. Given the relative ease of adding these extra audio functionalities, tuning the system to cater for the more visually impaired would be very plausible for the future.

Finding lost objects

Can the system implemented be used to help even non-visually impaired find custom items? Currently one of the most successful methods to find lost objects using a smartphone device is to use tags. Tags are miniature integrated circuits, with enough memory to hold a unique electronic product code and further information so that it can be read from [13]. Tags are attached to antennas that allow them to communicate with RFID (Radio Frequency Identification) readers at certain frequencies and ranges revealing the ID of the tag and its relative location. These readers when integrated into mobile systems can give tracking functionality if handled and processed accordingly by the device. Other tags use Bluetooth signals that can be received by smartphones without the need to use RFID readers [33] [3]. In either case, the use of tags imposes a manufacturing cost on the tracking system (especially those using RFID) which can lead to expensive products.

Using image recognition like the methods implemented for this system removes the need for additional hardware but at the cost of more processing from the device and is less robust. The main problem currently is the object cannot be completely obscured by the camera in order to function, as the system needs to view a proportion of the object to gain enough descriptors to compare with training images. This then raises the question that if an object has to be seen in full or partial view by the phone's camera in order to be recognised, the system is no better than a human with standard vision trying to find an object alone and is perhaps even slower at doing so.

In order to have the possibility of knowing if an object in a training image has been covered up or moved out of site of the camera view the system must always be on and monitoring the tracked objects, so that it may look into some frame archive and tell that at a point in time the detected object was moved off screen or placed inside/under an item that is not recognisable. From

this the system could theoretically give a last known position co-ordinate on the screen (provided the camera is fixed) or tell which side of the screen an object had left. This solution fits less with using the object recognition methods on a mobile device and more with a set of fixed cameras in a room such as a CCTV system. There could be the possibility of connecting the phone to a set of such cameras wirelessly, using their camera data instead of the device's as the query images and incorporating the additional functionality explained above to be of more use to the non-visually impaired. This could be more accurate than using tags by showing an objects last location in a real world environment, as RFID readers cannot get exact co-ordinates of tags without the addition of GPS. Though this then adds many further complications such as setting up a vast set of cameras as well as the processing requirements, power and privacy issues involved with having a consistently active camera recording. Therefore, although there is a possibility for the system to be developed further in order to help the non-visually impaired find commonly missing items. The certain changes in architecture and additional work needed to perform the development may require too many input resources and not enough output for the current detection method.

Conclusions

To re-iterate; the objective of the project was to produce an application for a mobile device that can recognise user defined objects in an environment of noise through the device's camera. Reviewing the system description from the initial report shows one objective that was not implemented with much success; "The project will need to be able to allow users to add objects they wish to distinguish. This will be done by creating a repository of images at multiple angles for added objects via steadily rotating the camera around the entity at certain points." Although the finished system can add objects and if the user wishes multiple angles of the same object to the library, the experiments performed revealed the top library capacity is 7 before a chance of system crashing and 4 before possibly affecting the recognition in the camera view. These capacities are minimal in order to support adding multiple angles of a single object before being a detriment to the system. The programme in its current state is far more suited to detecting an object from a single face (within the angle of acceptance) if multiple objects wish to be found or a single object from a few perspectives if only that object is to be recognised. This highlights the importance of the experiments performed, as it could have been concluded that the system satisfies this objective successfully by assuming the library has a large capacity without considering the capacity of the whole system.

However excluding this objective, a mobile device application for Android smartphones has indeed been implemented, the application can recognise objects using ORB detection and matching on images taken by the user. The experiments prove that in the best case noise is not a consideration, so long as the object is in view and in the worst case a certain noise threshold is acceptable. The recognition results can also be seen as additional graphics on top of the device camera whose frames are used as the query image when matching. The finished system does satisfy the requirements drafted in the design section, further proven from the test cases. Therefore, a vast majority of sub-goals from the main objective have been successfully fulfilled with positive results and it is reasonable to assume the main objective has been met to a high degree.

Reflection

The individual project is essentially a problem solving task that can be split into segments; developing an understanding of the problem to a point where a solution can be made, using understanding to create plans and designs to help solve the problem, creating the solution from the designs and research, and finally testing the solution is adequate or valid enough to solve the initial problem. For each section there will be a discussion of the method(s) used to accomplish it, how beneficial the methods were under timed conditions and what methods or learning's can be applied to future applications.

Firstly, developing an understanding of the problem, which in this case was the research needed to give a competent understanding of how to implement object recognition on smartphones whilst adhering to my skillset. In order to achieve this the object was decomposed into smaller sections of research such as how does object recognition work, what techniques are available and what smartphones will most likely support those techniques with a programme I am comfortable with. The method of decomposition was very successful when drawing conclusions from research and building up knowledge. It also prevented an overload of information which could have occurred when trying to research everything at once. However it did need additional management in order to re-compose the sub solutions to bring back the context of the end result. In this case it was relatively easy to relate all the segments together at once, especially considering there was only one human resource with all the knowledge of each segment and no sharing or externalising was needed. Applying decomposition and combining to future projects with similar success may need more management steps especially considering group tasks or larger problems. A structure such as externalisation meetings for group tasks that involve discussions of individual findings, or re-combing segments of related information in steps until all the knowledge is once again interrelated for larger problems could help with this. Either way so long as there is an assessment to judge that the payoff of an increased understanding is greater than the extra time invested in management and decomposing the problem this method will suit well for other projects.

There were many plans and designs made during the project in order to aid the development of the solutions and to make sure it was implemented in time; from Gantt charts in the initial report to requirements and architecture diagrams in the design section. What was consistent throughout each plan or design was a justification. The list of potential charts, drawings and documentations in a design phase are endless that designing the solution to the problem even when guided through research could theoretically take up more time than the implementation itself. Applying and thinking if a model is really necessary helped focus the design process and avoided wasting resources creating additional material that did not actually contribute to better implementation or time management. A more critical and justified approach to all areas, not just the designing of the solution, will help clarify actions and have similar benefits of that discussed. This may seem like an obvious method, but justifying actions and material can get lost once the project is past its research phase and only thought of again at the end of the project when reflecting back. Keeping and documenting a justified approach throughout helped greatly in this project to keep on track to accomplishing the main objective on time and will continue to be applied in the future for hopefully similar results.

For implementing the object recognition software, even when following design requirements and research results there were still unforeseen barriers such as those discussed in "complications when implementing". One specific problem, "SURF with OpenCV Android library", which occurred upon finding out that SURF was not available for Android could have been prevented from a more iterative style of research. To expand, having known that SURF was to be used in the system and later deciding to use Android and OpenCV. Combining research conclusions and performing a secondary round of research looking at the specific OpenCV library for Android would have found that it did not contain SURF and prevented this problem. To learn from this, although prior in the reflection it was mentioned that decomposition was good at building up knowledge.

Using this built up knowledge and testing theories against once more for other tasks could help solve problems only discovered this time when creating the solution.

When testing the solution, following approved guidelines and test structures such as usability heuristics, think-aloud evaluation and test cases helped I thought make the testing seem official and validated. Even in the self-conducted experiments, following a common patterned form of documenting experiments such as; abstract, equipment list, method, results and conclusions had a similar affect. Testing is about assuring others not just the creator that the system is functional and usable, using recognisable test methods gave confidence that the system should be working and achieves its objectives. Another positive of using pre-defined testing and experiment methods/documentation is that they are already in an applicable format to other scenarios. The same usability heuristics for example could easily be used on another system interface for the same testing purposes as this one. Given the discussed benefits, the method of using existing test formats will be applied again for subsequent solutions for prospective problems.

Overall I feel that there were some good methods used throughout the entire project that can be further applied elsewhere. As with the group project last year, meetings with the supervisor were key in expanding knowledge, producing results on a regular schedule and giving guidance when necessary and I hope similar guidance systems can be applied in the future. Even reflecting on the methods missed, such as the iterative like research approach, can lead to the creation of more improved methods and principles to add to those that proved useful during the project. The ultimate result of the reflection process is to create continuously improving results in the next manner of work without repeating past mistakes. Making reflection itself a method to be transferred on in the future, to forever improve and learn and create improving solutions to new problems as a result.

Bibliography

1. Android Developer, *MediStore.Images*, Viewed 08/02/2015 At:
<http://developer.android.com/reference/android/provider/MediaStore.Images.Media.html>
2. Android Developer, *Performing Audio Capture*, Viewed 30/04/2015 At:
<http://developer.android.com/guide/topics/media/audio-capture.html>
3. Ben Coxworth, *StickNFind system uses your phone and coin-like tags to find lost items*, Viewed 30/04/2015 At: <http://www.gizmag.com/sticknfind-finding-system/25238/>
4. C. Harris and M. Stephens, 1986, "*A combined corner and edge detector*", pages 147–151
5. Caleb Woodruff, *Feature Detection and Matching*, Viewed 03/02/2015 At:
<https://courses.cs.washington.edu/courses/cse576/13sp/projects/project1/artifacts/woodrc/index.htm>
6. Deepak G. Viswanathan, *Features from Accelerated Segment Test (FAST)*, Viewed 04/02/2015 At:
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV1011/AV1FeaturefromAcceleratedSegmentTest.pdf
7. Dr Alia Abdelmoty, *Human Factors: Colour Desig*, Viewed 30/04/2015 At:
https://learningcentral.cf.ac.uk/bbcswebdav/pid-3028051-dt-content-rid-4304435_2/courses/1314-CM2101/lect10-2101-%20Colour%20Design-st-slides.pdf
8. Eric Ravenscraft, *I Want to Write Android Apps. Where Do I Start?*, Viewed 05/02/2015 At:
<http://lifelacker.com/i-want-to-write-android-apps-where-do-i-start-1643818268>
9. Ethan Rublee, Vincent Rabaud, Kurt Konolige & Gary Bradski, *ORB: an efficient alternative to SIFT or SURF*, Viewed 04/02/2015 At:
https://www.willowgarage.com/sites/default/files/orb_final.pdf
10. Eveliotc, *How To: Voice Commands into an android application*, Viewed 30/04/2015 At:
<http://stackoverflow.com/questions/11798337/how-to-voice-commands-into-an-android-application>
11. Gil Levi, *A tutorial on binary descriptors – part 2 – The BRIEF descriptor*, Viewed 04/02/2015 At: <https://gilscvblog.wordpress.com/2013/09/19/a-tutorial-on-binary-descriptors-part-2-the-brief-descriptor/>
12. Grant D. McKenzie, *How to Calculate Hamming Distance*, Viewed 04/02/2015 At:
<http://classroom.synonym.com/calculate-hamming-distance-2656.html>
13. Imping, *How Do RIFD System Work*, Viewed 30/04/2015 At:
<http://www.impinj.com/resources/about-rfid/how-do-rfid-systems-work/>
14. iOS Developer Library, *Start Developing iOS Apps Today*, Viewed 06/02/2015 At:
<https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/>

15. Jones V, *What does the distance attribute in DMatches mean?*, Viewed 03/02/2015 At: <http://stackoverflow.com/questions/16996800/what-does-the-distance-attribute-in-dmatches-mean>
16. Konstantinos Avgerinakis, *Which are the best open source tools for image processing and computer vision?*, Viewed 06/02/2015 At: http://www.researchgate.net/post/Which_are_the_best_open_source_tools_for_image_processing_and_computer_vision
17. Lanshan, *A problem in OpenCV with Android using SURF*, Viewed 02/03/2015 At: <http://answers.opencv.org/question/14966/a-problem-in-opencv-with-android-using-surf/>
18. Lin Longfei, *Image Processing: Difference between SURF and SIFT?*, Viewed 03/02/2015 At: <http://www.quora.com/Image-Processing/Difference-between-SURF-and-SIFT-where-and-when-to-use-this-algo>
19. Michael Calonder, Vincent Lepetit, Christoph Strecha, & Pascal Fua, *BRIEF: Binary Robust Independent Elementary Feature*, Viewed 04/02/2015 At: <https://www.robots.ox.ac.uk/~vgg/rg/papers/CalonderLSF10.pdf>
20. Michael Stanford, *Measures of distance between samples: Euclidean*, Viewed 03/02/2015 At: <http://www.econ.upf.edu/~michael/stanford/maeb4.pdf>
21. Naotoshi Seo, *Tutorial: OpenCV haartraining (Rapid Object Detection With A Cascade of Boosted Classifiers Based on Haar-like Features)*, Viewed 02/02/2015 At: <http://note.sonots.com/SciSoftware/haartraining.html#Kuranov>
22. OpenCV, *About OpenCV*, Viewed 06/02/2015 At: <http://opencv.org/about.html>
23. OpenCV, *Features2D + Homography to find a known object*, Viewed 03/03/2015 At: http://docs.opencv.org/doc/tutorials/features2d/feature_homography/feature_homography.html
24. OpenCV, *Camera Calibration and 3D Reconstruction*, Viewed 04/03/2015 At: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=findhomography#findhomography
25. P M Panchal, S R Panchal & S K Shah, *A Comparison of SIFT and SURF*, Viewed 03/02/2015 At: http://www.ijircce.com/upload/2013/april/21_V1204057_A%20Comparison_H.pdf
26. P. L. Rosin, 1999, *"Measuring corner properties. Computer Vision and Image Understanding"*, 73(2):291 – 307
27. Rbaleksandar, *The pro and con of BRIEF and ORB compared to SIFT*, Viewed 04/02/2015 At: <http://stackoverflow.com/questions/13226554/the-pro-and-con-of-brief-and-orb-compared-to-sift>
28. Samarth Brahmhatt & Mert Kilickaya, *What is "Histogram of Oriented Gradients" and how does it work?*, Viewed 02/02/2015, At: <http://www.quora.com/What-is-Histogram-of-Oriented-Gradients-and-how-does-it-work>

29. Sammy, *How to match 2 HOG for object detection?*, Viewed 02/02/2015 At: <http://answers.opencv.org/question/877/how-to-match-2-hog-for-object-detection/>
30. Sherice Jacob, *Speed Is A Killer – Why Decreasing Page Load Time Can Drastically Increase Conversion*, Viewed 08/02/2015 At: <https://blog.kissmetrics.com/speed-is-a-killer/>
31. Sue Smith, *Capture and Crop an Image with the Device Camera*, Viewed 08/02/2015 At: <http://code.tutsplus.com/tutorials/capture-and-crop-an-image-with-the-device-camera--mobile-11458>
32. The MathWorks, *Train a Cascade Object Detector*, Viewed 02/02/2015 At: <http://uk.mathworks.com/help/vision/ug/train-a-cascade-object-detector.html>
33. Tile, *How Tile Works*, Viewed 30/04/2015 At: <https://www.thetileapp.com/how-it-works>
34. Tony Lindeburg, *Scale Invariant Feature Transform*, Viewed 03/02/2015 At: http://www.scholarpedia.org/article/Scale_Invariant_Feature_Transform
35. Unknown, *Difference of Gaussians*, Viewed 03/02/2015 At: http://en.wikipedia.org/wiki/Difference_of_Gaussians
36. Unknown, *RANSAC*, Viewed 04/03/2015 At: <http://en.wikipedia.org/wiki/RANSAC>
37. Unknown, *What Colors Mean*, Viewed 30/04/2015 At: <http://www.factmonster.com/ipka/A0769383.html>
38. Unknown, *Scale-invariant feature transform*, Viewed 03/02/2015 At: http://en.wikipedia.org/wiki/Scale-invariant_feature_transform
39. Unknown, *SURF (Speeded Up Robust Features)*, Viewed 03/02/2015 At: <http://en.wikipedia.org/wiki/SURF>
40. UxMovement, *When to Use White Text on a Dark Background*, Viewed 30/04/2015 At: <http://uxmovement.com/content/when-to-use-white-text-on-a-dark-background/>
41. VLFeat.org, *About VLFeat*, Viewed 06/02/2015 At: <http://www.vlfeat.org/about.html>

Image references

Fig. 1. *Creating descriptors in SURF (left) and SIFT (right)*, Viewed 03/02/2015 At: <http://www.quora.com/Image-Processing/Difference-between-SURF-and-SIFT-where-and-when-to-use-this-algo>

Fig. 2. *FAST & the Bresenham circle*, Viewed 04/02/2015 At: <http://www.edwardrosten.com/work/fast.html>

Fig. 3. *Incremental Development Model*, Viewed 09/02/2015 At: https://learningcentral.cf.ac.uk/bbcswebdav/pid-2763056-dt-content-rid-3198509_2/courses/1213-CM1202/Week%204%20-%20after%20waterfall.pdf

Fig. 4. *Prototype Development Model*, Viewed 09/02/2015 At:
https://learningcentral.cf.ac.uk/bbcswebdav/pid-2763056-dt-content-rid-3198509_2/courses/1213-CM1202/Week%204%20-%20after%20waterfall.pdf