Computer Science
University of Otago

COSC490 Interim Report

# Fast Mobile Pose Estimation

*Author:*
Nicky Crawford

*Supervisor:*
Brendan McCane

July 19, 2013

# Contents

# 1 Objective

Augmented reality (AR) is a technology that adds elements to a live view of the real-world. This idea has been around since the early 1900's, first mentioned in a novel by L. Frank Baum. Applications of AR started appearing in the second half of the 20th century and have since been used in fields as diverse as medicine and gaming [10]. In order to convincingly display an element as if it were part of the real-world, its orientation, position and scale must be updated as the camera moves. Most current AR applications use markers to track movement. The problem with this is that a very specific marker is required which often must be printed, and the application only works while this marker is in view of the camera. What we really want to do is get rid of these markers completely and instead track natural features. The aim of my project is to investigate and employ a system which tracks natural features, and uses a suitable fast position and pose estimation algorithm to implement markerless AR on a mobile device.

# 2 Method

The system I aim to create will augment a live video stream on a mobile device. My approach to achieve this involves four key stages:

1. *Feature detection* - Analyse each frame of the video in order to find points of interest such as the corners of objects.

2. *Feature matching* - Compare the features found in two consecutive video frames in order to determine which correspond to the same item in the real-world.

3. *Relative Pose Estimation* - Using the matches found, calculate how the view of the real-world has been transformed as a result of the camera moving.

4. *Augment the scene* - Draw some element in the scene and transform it using the relative pose calculation. The idea is to give the illusion that this element exists in the real-world.

So far the focus has been on feature detection and feature matching so section 3 will discuss methods of feature detection from the literature. Section

4 will outline what work I have done so far and covers stages 1 and 2 above. Finally section 5 will describe what I have left to do, particularly stages 3 and 4.

# 3 Feature Detection in the Literature

Feature detection is a technique used in computer vision to find points of interest, often corners, in an image. There are many ways to do this, though most tend to look at changes in lighting or intensity of the pixels. The most important property of a feature detector is that it detects features consistently. That is, the same features will be detected when present in consecutive scenes so they can be matched.

## 3.1 Feature Descriptor vs. Feature Detector

One concept that can cause confusion is the difference between a feature detector and a feature descriptor. A feature detector finds the points of interest in a scene using a specific algorithm. A feature descriptor on the other hand is a way of representing the points detected. This may contain information like orientation, or how confident you are that the point is a feature. In the next four sections I will describe four feature detectors and in the fourth section I will also describe the feature descriptor used by the ORB detector.

## 3.2 Moravec Interest Operator

One of the early corner detectors was created by Moravec [5] during an experiment to navigate a robot through indoor and outdoor spaces using pictures taken by a camera on the robot. Moravec used an *interest operator* routine to detect corners or "features". This took a 4 x 4 pixel window in the image which was being considered as a feature. This window was moved horizontally, vertically and diagonally, and the sum of square differences was calculated for the intensity of corresponding pixels in the original and translated windows. The minimum intensity variation from these translations was used as the window's interest measure. This test was carried out over the entire image with windows spaced half a window width apart.

Using the minimum directional variance worked because you only got a high variation in intensity in all four directions when you have found a corner, e.g. in cases C and D in Figure 1. An edge, e.g. case B, gave a high variation in intensity perpendicular to the edge, but low variation along it. A flat surface such as the interior of an object or a background, case A, gave little variation in any direction.

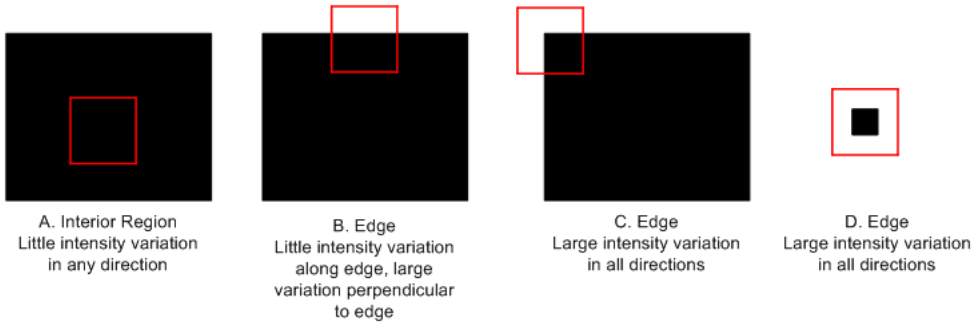


Figure 1: Moravec's interest operator routine looks at window of pixels in an image. Cases C and D will give a high variation in the intensity of corresponding pixels when the window is translated in all directions indicating a corner [7].

## 3.3 Harris and Stephens Corner Detection

Harris and Stephens [2] identified a number of problems with Moravec's technique:

1. The intensity variation was only measured in a discrete set of directions because the window was only shifted at 45 degree angles. This meant that if the image was in a different orientation, you would get a different result.

   By analytically expanding Moravec's calculation, Harris and Stephens came up with a formula to calculate the intensity variation for all possible small shifts of the window.

2. The intensity variation calculated was noisy because the window used was binary and rectangular. Being rectangular meant that the distance from the centre to the outside of the window was not uniform, and being

binary, it did not take account of this difference in distance - either the pixel was in or out of the window.

To resolve both these problems Harris and Stephens proposed using a smooth circular window such as a Gaussian.

3. The Moravec operator detected edges as corners too easily since pixellation and other imperfections could make the edges look like corners to the operator. Using eigenvalues they found it was possible to differentiate between corners, edges and flat surfaces despite noise in the image.

Moravec's algorithm was simple and efficient, but being simple it clearly had problems as pointed out by both Harris and Stephens, and Moravec himself. Harris and Stephens made some improvements to the algorithm, and other techniques have been employed since to alleviate these and other issues associated with feature detection.

## 3.4 Scale Invariant Feature Transform (SIFT)

A more recent feature detector was developed by David Lowe [3] to help identify objects in different images. Lowe noted that previous corner detectors only identified features at a single scale, which meant that when the scale of the image changed, different points would be detected as features. Lowe's method which he termed Scale Invariant Feature Transform (SIFT) involves finding maxima and minima of a difference of Gaussian function applied in scale space. He achieved this by creating a series of images which were the result of repeatedly smoothing the original using a Gaussian function. From these a series of Difference of Gaussian images were created by subtracting successive Gaussian images, see Figure 2.

The features were extracted from the difference of Gaussian images by comparing each sample point with its nearest neighbours at the same scale, at the scale below and the scale above, see Figure 3. A point would only be chosen as a feature if it was bigger or smaller than all its neighbours at the same scale, the scale above and the scale below. Most points would be eliminated within the first few checks so it was not too computationally expensive.

By searching for features that could be reliably detected across different scales, Lowe's method was able to be scale invariant.

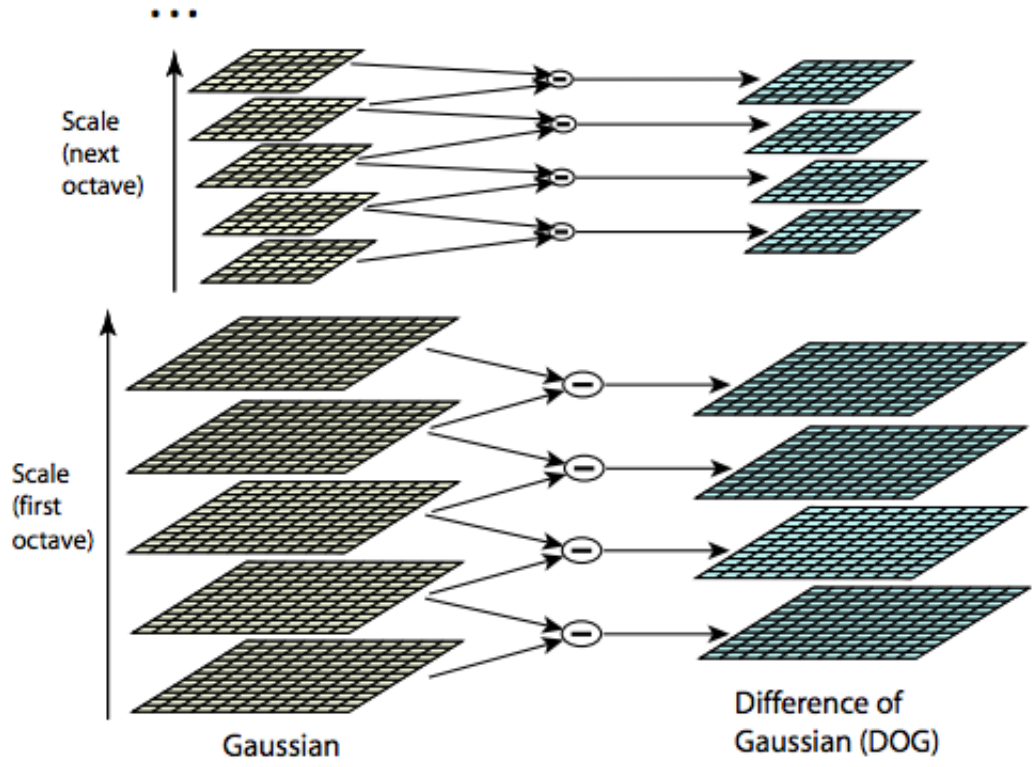


Figure 2: A series of Gaussian images (left) are created by repeatedly smoothing the original image with a Gaussian blur. A series of Difference of Gaussians images (right) are created by subtracting successive Gaussian images [4]

## 3.5 ORB

In a 2011 publication, Rublee et al. [9] commended the performance of SIFT, however they asserted it was not good for real-time systems and low-power devices such as cellphones. They proposed their own feature detector, ORB, which they claimed performed as well as SIFT while being almost twice as fast. ORB was composed of two parts - oFAST (oriented FAST) and rBRIEF (Rotation-Aware BRIEF).

FAST [8] was another method of feature detection which worked by looking at a circle of pixels around a candidate corner. If it found an arc of $n$ contiguous pixels in the circle which were all brighter or all darker than the centre pixel plus a threshold, the centre would be considered a corner, see

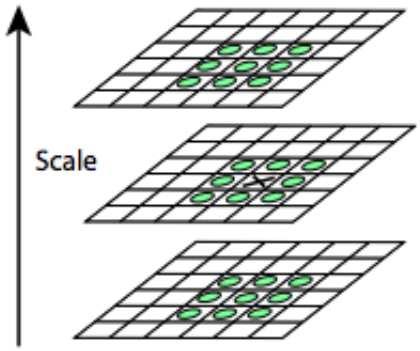


Figure 3: Point X will be considered a feature if it is a maximum or minimum among neighbouring pixels highlighted in green [4].

Figure 4. oFAST started by using FAST and then employed techniques from Harris and Stevens to give a measure of cornerness - a number indicating the degree to which they believe the pixel is a corner. From this they picked the top N points they believed were corners. To add orientation, oFAST created a vector from the centre of the corner to the intensity centroid.

rBRIEF was used to compute the descriptor. The original version of BRIEF [1] took a smoothed image patch p, and created a descriptor by comparing the intensities of pixels x and y using the following binary test:

$$\tau(\boldsymbol{p}; x, y) := \begin{cases} 1 & : \boldsymbol{p}(x) < \boldsymbol{p}(y) \\ 0 & : \boldsymbol{p}(x) \geq \boldsymbol{p}(y) \end{cases} \tag{1}$$

This test will return a 1 if pixel x is brighter than pixel y, otherwise a 0 is returned. To work out which pixels, x and y, would be compared, a Gaussian distribution around the centre of the patch was used. No exact details about the use of the Gaussian distribution were given. A 256 bit binary descriptor was created from 256 comparisons of pixels. BRIEF was very sensitive to in-plane rotation, so Rublee et al. created "steered BRIEF" which rotated the image points according to the orientation of the image. They also employed a greedy search to make sure they got points with high variance and that were uncorrelated.

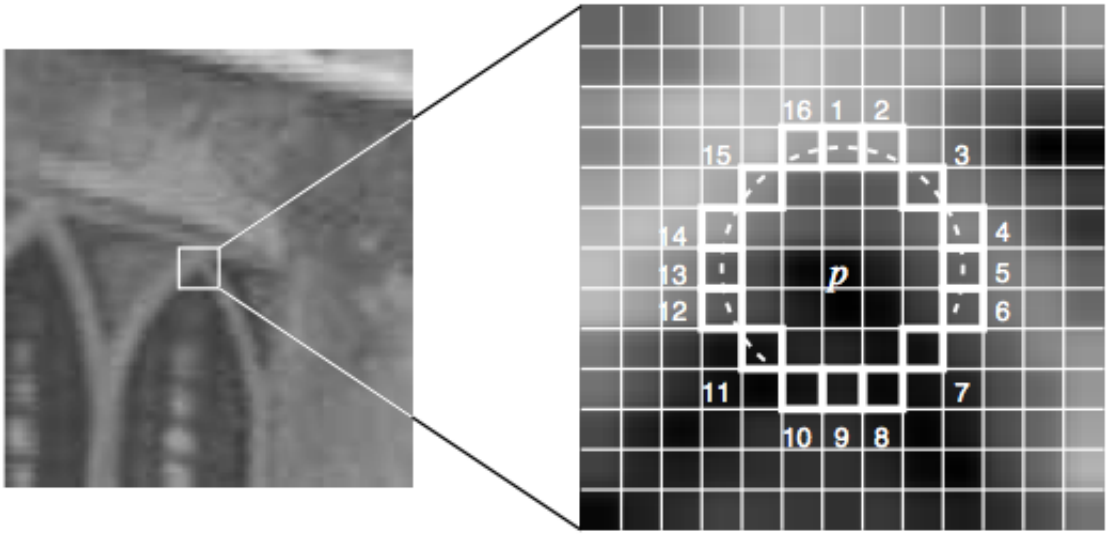


Figure 4: FAST detected corners by checking if there were an arc of pixels (here 12) that were all brighter in intensity or all darker in intensity than the centre pixel $p$ [8].

# 4 My Work

## 4.1 Setup

All of my work has been written in Objective-C and C++ with the help of OpenCV (Open Source Computer Vision Library) version 2.4.5. This was written and compiled in Xcode version 4.6 on an iMac running OS X 10.8.3. The code is written to be run on an iPhone or iPod Touch running iOS version 6.1 or higher and has been tested on a fourth generation 8GB iPod Touch running iOS 6.1.

## 4.2 Feature Detection

I chose to use ORB for feature detection in my project. It was designed for use on low-power devices such as cellphones so is a sensible choice for my project. It was also one of the available feature detectors in OpenCV.

I started with a simple case - detecting features in a static image. OpenCV was able to do a lot of the hard work for me - it took in an image stored

in OpenCV's Mat (matrix) data type and found the features using the ORB method with only a few method calls. There were also some nice drawing methods so you could visually see the points that were detected as features. The result of the feature detection can be seen in Figure 5.

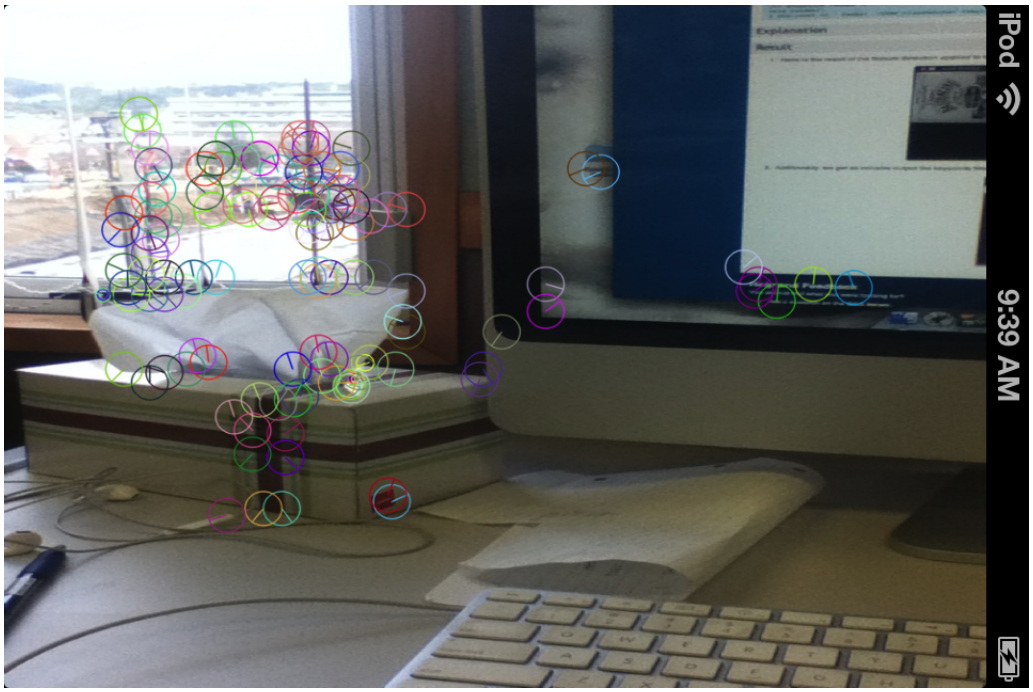


Figure 5: OpenCV's implementation of ORB feature detection working on a picture taken with the iPod. The coloured circles are the points that have been detected as features and the lines indicate their orientation. (Colours have been randomly assigned and have no meaning)

Next I got feature detection working on a live video stream. Using the camera on the iPod, each frame was processed in the same way the static image, and features were drawn on screen. I had to add code to get input from the video camera, but the most of the existing feature detection code could be reused.

## 4.3 Feature Matching

### 4.3.1 OpenCV

Using OpenCV I had two options for matching features found in consecutive frames of the video stream - FLANN and Brute Force. FLANN is a method that uses an approximate nearest neighbour search to find the closest matching descriptor. It is optimised for large datasets and high dimensional features by sacrificing accuracy [6].

The brute force matcher works as you might expect - it goes through every feature descriptor from the first image and works out the closest match by trying each from the second. This is a more precise method, but obviously it suffers in speed.

To work out whether two features are a good match, i.e. they are similar image patches, the hamming distance between the two descriptors is calculated. As described above, the descriptor for an ORB feature is a string of bits. The hamming distance between two of these describes how many corresponding bits are different, so a small value means the image patches are similar. No individual bit has any more weight on the hamming distance than any other, which means no pixel in the image patch is more significant than the others.

### 4.3.2 My Implementation

I decided to use the brute force method for my project as I will only be dealing with a relatively small number of points in each image (around 20 - 100). With fewer points I needed the matches to be more accurate and the cost of speed should not be too high. If the speed of matching becomes an issue, I may revise my choice.

Again OpenCV had some very helpful methods for implementing feature matching and visualising it on screen. I was able to build on my feature detecting code in order to implement matching on two static images, and subsequently on consecutive frames of the video feed. The results of feature matching on two static images taken by the iPod can be seen in Figure 6.

Not all of the matches found are necessarily going to be correct. To try and minimise the number of incorrect matches cross checking can be used. This means we only count matches as being "good" if they go both ways - that is we match the descriptors in the first image to those in the second, and also match the descriptors in the second to those in the first. If two

descriptors are matched with each other in both cases, they will be used. There is a flag that can be set to do this during the brute force matching process. I implemented cross checking myself to verify the flag in the matcher was doing what I expected. I got the same results using my implementation and the built in one, so I left cross checking to the matcher.

### 4.3.3 Results

The results of feature matching as seen in Figure 6 were poor. From 60 features detected in each image only 15 matches were retrieved, and these matches were not particularly good correspondences. The expected results were lines between the images that are relatively parallel indicating the transformation of the scene. Instead the lines possess no uniformity, and it is clear that incorrect matches exist simply by looking at the points matched in each image.

## 4.4 Difficulties

I have had relatively few difficulties with my project at this point. A big part of my research so far has been learning about feature detectors in the literature. Reading the original papers was not always easy when I had little knowledge of computer vision and the mathematics was not necessarily simple to follow. I looked at other explanations of the concepts in each paper on the internet as well as asking my supervisor for help in order to understand the papers.

Getting into OpenCV was another challenge. I had never written in C++ before and I found using it another hurdle. The tutorials on the OpenCV website were a useful starting place and example code written by others was also helpful. There were a lot of little things that tripped me up like needing to convert images to different colour formats at various points, and having copy the image into a separate Mat for drawing matches. However I managed to find enough help on the internet and in example code to sort out all these issues.

# 5 Future Work

## 5.1 Feature Matching

Currently I have implemented feature matching on static images with poor results and on the video feed with no visual output. My next task will be to find out why the matching results are not good whether this is because of the test images I used, the number of features I detected or some other reason. Once I am happy with that I would like to modify the program doing matching on the video feed to show both the current frame and the previous one. This will allow me to see what is going on verify matching is working correctly.

## 5.2 Relative Pose Estimation

I have not yet started doing any work on calculating the relative pose of the camera between the frames of the video. I have discussed the options with my supervisor and he has suggested two appropriate methods I could use - the eight-point algorithm or RANdom SAmple Consensus (RANSAC). Both are methods used to calculate the fundamental matrix used in computer vision to relate corresponding points in stereo images. I will need to do background reading on these techniques and work out which I will implement.

## 5.3 Augmenting the Scene

In order to augment the video feed, I will need to complete all of the above and look into OpenCV's drawing methods. I will also need to work out how to transform this added element using the fundamental matrix or any other information calculated during the relative pose estimation stage.

## 5.4 Revision of Aims and Objectives, and Schedule

My objectives from the start of the year focused on testing the performance of algorithms for detecting key points in a scene, matching these points in a subsequent scene and estimating camera movement between the two. I have spent a lot of time reviewing the literature on feature detecting techniques and I feel I have gained enough insight from this to make my choice of feature detectors.

While I would like to keep the aim of my project the same (outlined in sections 1 and 2), my revised objectives will be as follows:

- A literature review will be performed on techniques for detecting key points in a scene. (complete)
- An appropriate feature detection technique will be chosen implemented for use on a mobile device. (complete)
- Feature matching will be implemented using the features previously detected. (in progress)
- A literature review will be performed on algorithms for pose estimation such as the eight-point algorithm and RANSAC. (to do)
- An appropriate algorithm will be implemented to calculate the relative pose of the camera on a mobile device. (to do)
- If the above objectives are completed with time to spare, a scene will be augmented. This may be used to create a mobile application involving user interaction of some description depending on time restrictions.

Revised Schedule:

**21July - 1 August**

- Investigate and fix feature matching problem on static images
- Implement feature matching on video stream and verify it is working

**4 August - 1 September** *(including mid-semester break)*

- Relative pose estimation literature review
- Implement Relative pose estimation and verify it works

**1 September - 15 September**

- Augment scene

**15 September - 29 September**

- Write final report

# References


[1] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: binary robust independent elementary features. In *Computer Vision–ECCV 2010*, pages 778–792. Springer, 2010.

[2] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[3] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[4] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[5] Hans P Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, DTIC Document, 1980.

[6] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[7] Parks and Gravel. Corner detectors. `http://kiwi.cs.dal.ca/~dparks/CornerDetection/moravec.htm`, 2013. [Online; accessed 18-June-2013].

[8] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006*, pages 430–443. Springer, 2006.

[9] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[10] Wikipedia. Augmented reality — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Augmented_reality`, 2013. [Online; accessed 17-July-2004].

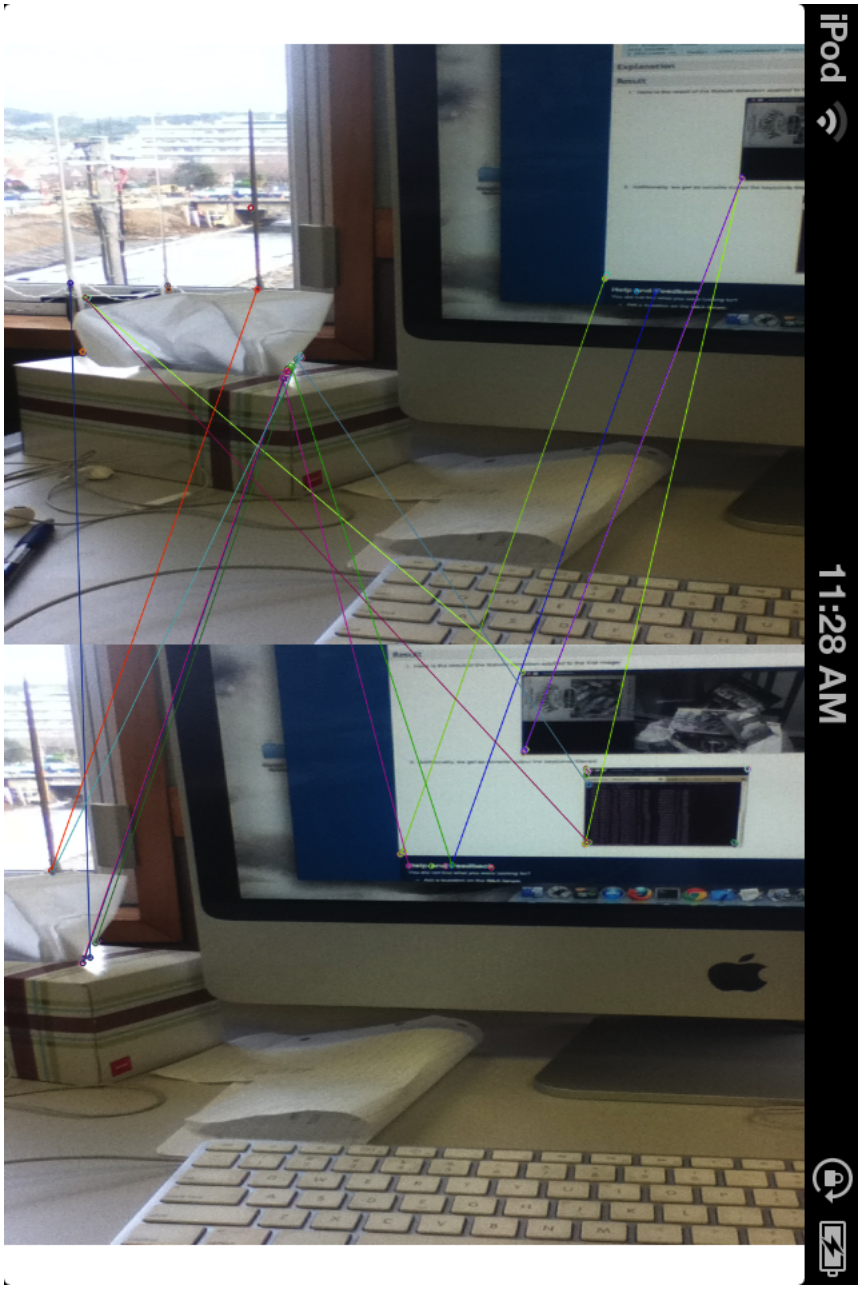


Figure 6: OpenCV's implementation of ORB feature detection with feature matching working on two pictures taken with the iPod. The coloured lines indicate features which have been matched. (Again colours have been randomly assigned and have no meaning)