Chapter 5

Tests and Results

In this chapter the tests that were done on the fingerprinting system are explained and the results for each test is shown and interpreted. These tests were done with the intention to improve the frame fingerprinting system, by choosing appropriate parameters based on the test results, or to show the limits of the system in terms of robustness, efficiency, accuracy and speed. Another test is used to do a trade-off between the use of SIFT and SURF to detect key points and then the tests are done to show the video fingerprinting's effectiveness and prove that it runs in real-time. At the end of the chapter the validation and verification of the system is done.

5.1 Experimental Setup: Frame Fingerprinting

For all the tests done on the frame fingerprinting algorithm, the following steps were followed:

- **System set up:**

  In all the tests the normal frame fingerprinting system is used as described in Section

4.1. Only the particular parameter value or method being tested was changed. For the *bits per hash* and *hashes per frame* tests, the corresponding parameter values were set and for the *hashing method optimization* and *SIFT vs. SURF* tests, the function was selected in code to correspond to the hashing function being tested. The *codecs and distortion* tests didn't require any alteration of the normal system.

- **Videos' frame fingerprints added to the database:**
  In this step, the database was filled with the frame fingerprints of 30 music videos. The 30 music videos will be referred to as the *"test videos"* throughout this chapter. Music videos were used because of the wide array of images and scenes used in the footage. For most of the tests, excluding the *codecs* test in Subsection 5.4.1, AVI videos with XviD encoding were used to generate the fingerprints for the database.

- **Videos converted and distorted:**
  Matching the same videos to the database that were used to generate the fingerprints for the database defeats the purpose of testing the system, because all the hashes would be matched successfully. This meant that alternate sets of *test videos* had to be created, by adding distortions, so it could be used to test the matching capabilities of the system.
  Using a program called FormatFactory, the *test videos* were converted to use different wrappers and codecs. Different codecs introduce different compression distortions as described in Subsection 2.3.2. The different wrapper and codec combinations used in these tests are shown in Table 5.1. The wrapper's name will be used to refer to the specific wrapper and codec combination throughout this chapter.

Wrapper	Full Name	Codec
AVI	Ausio Video Interlaced	XviD
MP4	MPEG-4 Part 14	DivX
FLV	Flash Video Format	FLV1
MPEG	Motion Picture Experts Group	MPEG-1

Table 5.1: Table of video wrapper and codec combinations used in the tests.

To do the tests with blur, scaling and rotation distortion applied to the frames,

key frames were detected in the same *test videos* that were used to generate the database and then functions in the EmguCV library were used to distort the frames before they were matched to the database.

- **Test videos' frames detected:**

  The sets of *test videos* were then run through the system and every key frame detected by the KFD was matched to the database. Every match result contains the VFID that scored the highest number of hash matches along with the specific number of hash matches the VFID acquired, the number of hashes generated for the frame and the unknown video's name and frame number. The VFID can be used to find the corresponding video name and frame position in the database and if it matches up to the unknown frame's information, a true positive match was made. The results of the matches were saved to an excel file. The second best match of the key frames to the database was also saved to determine the number of matches the best false positives acquire.

- **Raw result data processed:**

  The frame match results were processed and used to create frequency histograms for a visual representation of the frame detection. The frequency histograms show the *number of frames* (y-axis) that had a certain number of *hash matches* (x-axis) to the database and they are displayed as line graphs, instead of bar graphs, to allow the viewer to quickly and easily see the distribution of detections for the different test sets.

  In Figure 5.1 an example of a such a frequency histogram is shown. There are **three points** plotted on the line that will be used to explain how the graphs should be read. At the *circular point* the y-axis value is 90 and the x-axis value is 20, which means that 90 of all the frames that were matched to the database got 20 or 21 hash matches. At the *diamond shaped point*, the graph shows that 310 frames had 34 or 35 hash matches. The *triangular point* is exactly over the *More* point on the x-axis. At this point the graph indicates that there are 20 frames that got more than 50 hash matches. This happens if duplicate hashes were generated from the frame.

  As most of the test result distributions are very close to normal distributions, the av-

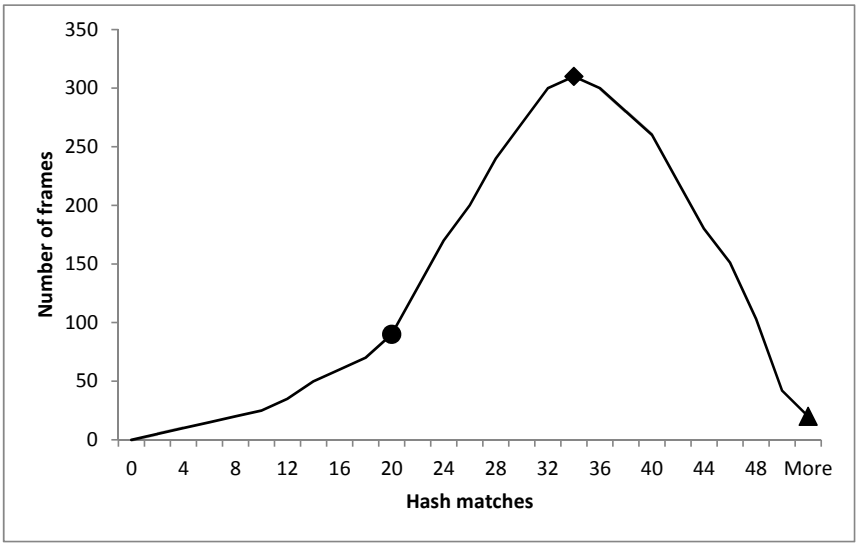


Figure 5.1: Example frequency histogram of results.

erages and standard deviations will be shown in tables. The higher the distribution's average is to the number of *hashes per frame*, the better. The best false positive matches' averages and standard deviations will also be shown in these tables so they can be compared to the results obtained from the true positive matches. The usual number of key frames detected in the *test videos* (30 music videos) are between 3600 and 3700 frames, allowing more than enough data to obtain an understanding of the detection capabilities.

Although this testing procedure stays relatively constant throughout the whole testing phase, specific test information will be stated in the test sections where needed.

5.2 Parameter Optimization

This section contains the tests that were done in order to improve the performance of the system by determining the best parameters to use for frame detection. The parameters that were optimized are the *bits per hash*, *hashes per frame* and the *detection threshold*.

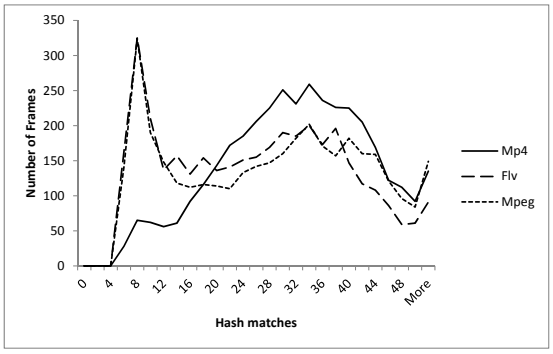
5.2.1 Bits per hash

As mentioned in the previous chapter, the frame fingerprinting technique that was developed, uses key points found in an image or frame and creates a set of hash values from pairs of these key points. Each hash value is made up of four normalized values calculated from the key point pairs, as explained in Subsection 4.1.1. This means that each hash contains a certain number of bits, and to match an unknown image or frame to the database, its hashes should match, bit for bit, with a known frame's hashes.

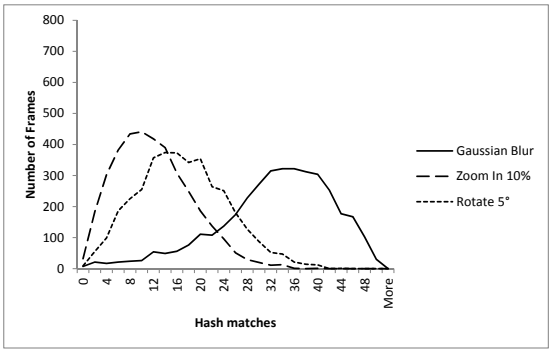
The *bits per hash* value is therefore very important, as it determines the tolerance allowed for each value used to create a hash, and subsequently the successful matching of frames. The higher the number of bits used, the higher the resolution, thus the hashes are more specific and its harder to get matches. On the other hand, using less bits to create hashes, means that its easier to get matches, but it increases the chances of false matches. It also increases the chance of getting two or more of the same hash values in one frame. Thus, it boils down to a trade-off between accuracy and robustness.

This test was set up as explained in Section 5.1 and repeated three times using 12, 16 and 20 *bits per hash* for the different tests. The fingerprint database was filled with fingerprints generated from the AVI *test videos*. After the database was filled with the fingerprints, six different sets of videos were matched to the database and the detection results saved. The first three sets of videos are the *test videos* in MP4, FLV and MPEG format and the last three sets are the AVI *test videos* with Gaussian blur (9×9 filter size), uniform scaling (zoomed in 10%) and rotation (5° right) applied, respectively. The results from the MP4, FLV and MPEG test sets will be called the *"codec results"* and the results from the other three test sets, the *"distortion results"*.

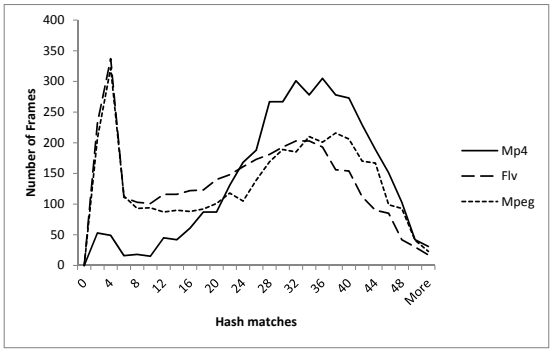
In Figure 5.2 the results are shown in the form of frequency histograms as explained in Section 5.1. The frequency histograms are made up of 3685 frame match's data for the *distortion results*, because 3685 key frames are detected in the AVI *test videos*. The *codec results* consist of different numbers of frames, because different key frames are detected by the KFD when using videos with different codecs. The number of key frames detected



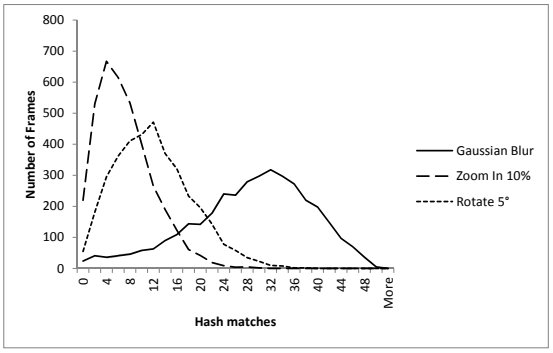
(a) 12 bit hashes *codec results*



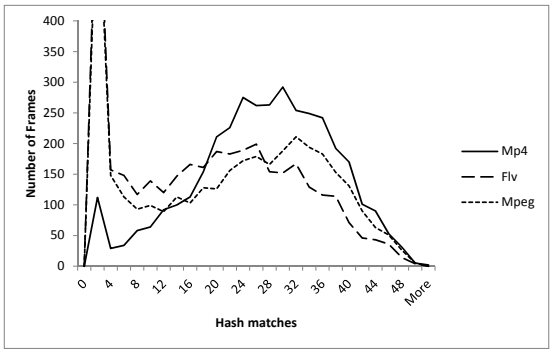
(b) 12 bit hashes *distortion results*



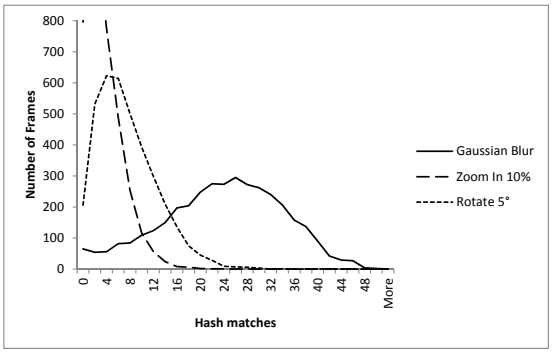
(c) 16 bit hashes *codec results*



(d) 16 bit hashes *distortion results*



(e) 20 bit hashes *codec results*



(f) 20 bit hashes *distortion results*

Figure 5.2: Results for *bits per hash* test.

for MP4, FLV and MPEG are 3674, 3642 and 3618, respectively.

The match averages and standard deviations for the true positive (best matches) and best false positive matches (second best matches) are summarized in Table 5.2. The match average and standard deviation for all the false positives and *distortion results* are calculated using every single detection. The match averages and standard deviations for the true positive matches of the *codec results* are calculated using only the hash matches that are above the false positives' match average + standard deviation. This means that the matches that are above 11, 5 and 3 will be used for the 12, 16 and 20 *bits per hash* test results, respectively. This is done so that the detection distribution can be analysed without the influence of the outliers.

Codec or Distortion	Bits per hash	Best match average	Best match standard deviation	Second match average	Second match standard deviation
MP4	12	32.86	10.32	8.20	3.01
MP4	16	32.65	9.16	2.99	2.18
MP4	20	27.44	9.49	1.35	1.36
FLV	12	30.42	10.96	8.07	2.96
FLV	16	27.41	11.11	2.94	2.08
FLV	20	22.54	10.77	1.31	1.26
MPEG	12	32.60	11.50	8.08	2.93
MPEG	16	29.87	11.59	2.91	1.97
MPEG	20	25.58	11.18	1.31	1.25
Gaussian Blur	12	32.17	9.72	8.21	3.11
Gaussian Blur	16	28.05	10.21	3.02	2.14
Gaussian Blur	20	23.24	10.11	1.36	1.31
Zoom in 10%	12	11.52	6.48	8.01	2.63
Zoom in 10%	16	6.65	4.8	2.77	1.12
Zoom in 10%	20	3.06	3.02	1.22	0.54
Rotate 5°	12	16.45	7.55	8.21	2.94
Rotate 5°	16	11.53	6.51	2.88	1.51
Rotate 5°	20	6.96	5.02	1.28	0.81

Table 5.2: Quantitive results for *bits per hash* test.

If one studies the graphs in Figure 5.2 one can see that the data distribution is almost

normal, except for the outliers at the very low values with the *codec results*. These outliers are caused by the frame detector algorithm, because frames are sometimes detected as key frames in the MP4, FLV or MPEG *test videos* that do not correspond to the key frame detected in the AVI *test videos*.

If one compares the data seen in Table 5.2, one can see that the 12 bit hashes have a higher match average than the 16 bit hashes, but the lower resolution of the 12 bit hashes caused a big increase in the averages of the secondary matches, from ± 3 to ± 8. In this case the 16 bit hashes prove advantageous, as it keeps the hash uniqueness higher (65536 hash values for 16 bit hashes and 4096 hash values for 12 bit hashes) with only a small decrease in hash match average. The 16 bit hashes' low secondary match average also allows one to choose a much lower detection threshold, which will be discussed in the following section.

The 20 bit hashes, on the other hand, have lower averages than the 16 bit hashes, because of the loss in robustness. The lower robustness is especially evident in the *distortion results*, which can be seen by comparing Figure 5.2(d) and Figure 5.2(f). After all is considered, the 16 bit hashes are the best choice for the system, because they provide a balance between uniqueness, robustness and efficiency that are advantageous for the system as a whole.

5.2.2 Hashes per frame and detection threshold

In the previous test, the *bits per hash* parameter was investigated and a conclusion was drawn that 16 *bits per hash* is the best value to use in this system, but that is not the only parameter that influences frame matches. The number of hashes created per frame is also a very important aspect, as it influences detection rates as well as database usage. After the number of *hashes per frame* is decided, the *detection threshold* can be set for the number of hash matches needed to detect a positive frame match.

Every single fingerprinting system makes use of thresholds to determine if a query fingerprint has a legitimate match in the database. This frame fingerprinting technique makes

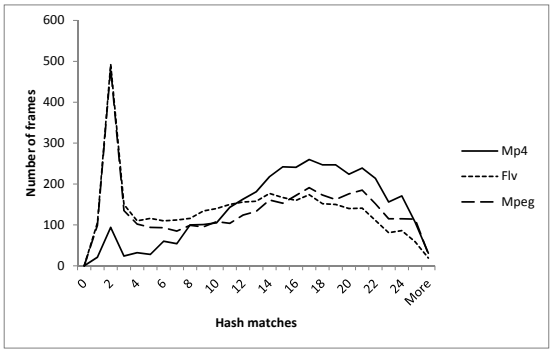
use of a *minimum hash matches* threshold to determine if the match is successful. If the best VFID match's number of matches is not above the threshold, it is not considered a match. Both the *hashes per frame* and the *minimum hash matches* threshold is determined by the same set of tests as the *minimum hash matches* threshold is dependant on the number of hashes used to fingerprint a frame.

This test was done the exact same way as the *bits per hash* test, but instead of changing the *bits per hash* value, the maximum *hashes per frame* was set to 25, 50 and 100 for the three test set-ups. The results for the different test set-ups are shown in Figure 5.3 and consists of between 3600 and 3700 frame matches' data. The differences are due to the fact that frames are only added to the database if their number of hashes reach the maximum *hashes per frame* value.

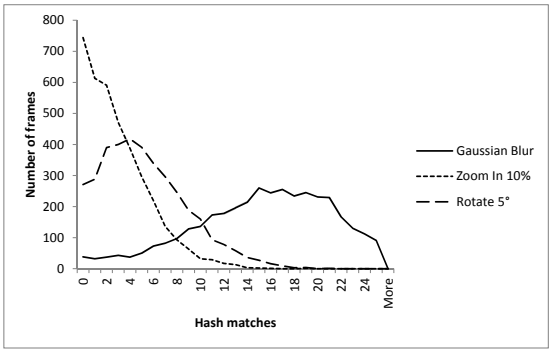
The test also included determining the best false positive match (second best match) of the frame, as with the *bits per hash* test, and are included in Table 5.3, along with the true positive matches' (best matches) data. The averages and standard deviations for the *codec results*, again only make use of the hash matches that got values above the false positive distribution, which is 3, 5 and 9 for the 25, 50 and 100 *hashes per frame* results, respectively.

If the results in Figure 5.3 and Table 5.3 are examined, it is clear to see that the distribution of the number of hash matches stay in relation when using 25, 50 or 100 *hashes per frame*. This means that the hashes are still robust, even if they make use of the smaller key points. The deciding factor would have to be the database usage. 25 *hashes per frame* would be the best, but it would make the selection of a threshold more difficult because of the low number of maximum hashes. With 50 or 100 *hashes per frame* the number of hashes is high enough to confidently choose a threshold, but 100 hashes would take up double the space in the database that 50 hashes would, so the best choice would be the 50 hashes per frame.

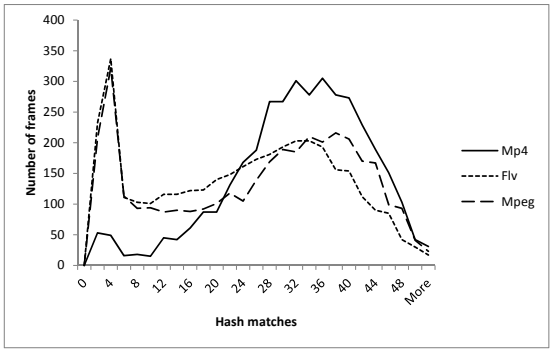
After close inspection of the results, it was seen that frame matches that had more hash matches than 10% of the hashes used per frame are positive matches, almost 100% of the



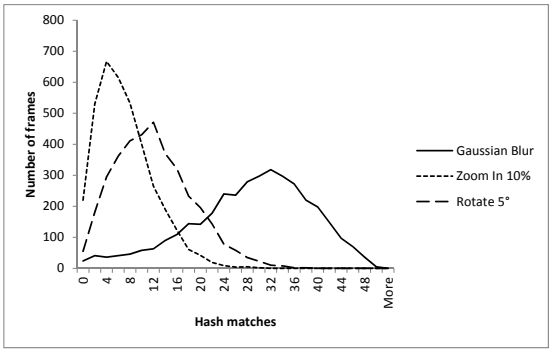
(a) 25 hashes *codec results*



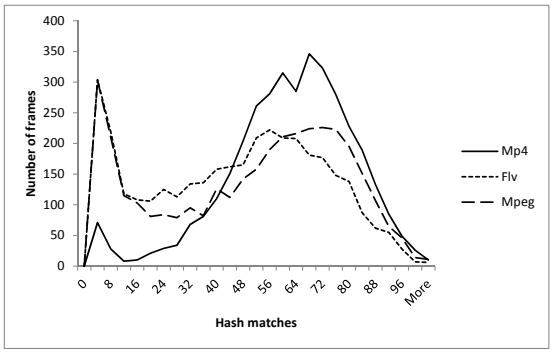
(b) 25 hashes *distortion results*



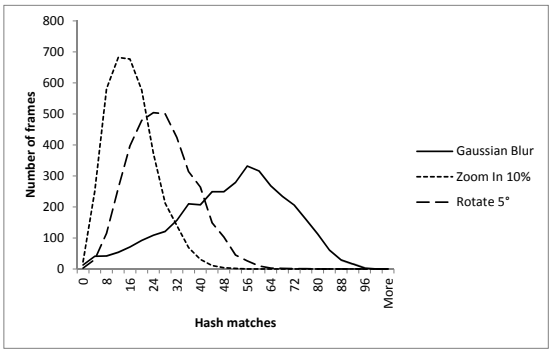
(c) 50 hashes *codec results*



(d) 50 hashes *distortion results*



(e) 100 hashes *codec results*



(f) 100 hashes *distortion results*

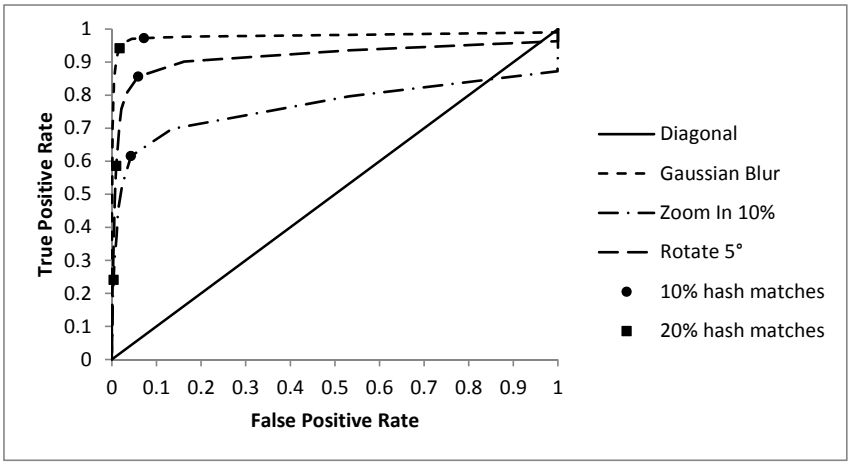
Figure 5.3: Results for *hashes per frame* test.

Codec or Distortion	Hashes per frame	Best match average	Best match standard deviation	Second match average	Second match standard deviation
MP4	25	16.61	5.13	2.01	1.11
MP4	50	32.65	9.16	2.99	2.18
MP4	100	62.29	16.71	4.76	4.06
FLV	25	14.31	5.79	1.95	1.06
FLV	50	27.41	11.11	2.94	2.08
FLV	100	51.18	21.31	4.69	4.02
MPEG	25	15.56	5.98	1.95	1.04
MPEG	50	29.87	11.59	2.91	1.96
MPEG	100	56.59	21.98	4.65	3.69
Gaussian blur	25	15.24	5.73	2.03	1.17
Gaussian blur	50	28.05	10.21	3.02	2.14
Gaussian blur	100	49.31	18.97	4.69	3.68
Zoom in 10%	25	2.95	2.71	1.93	0.63
Zoom in 10%	50	6.65	4.81	2.77	1.12
Zoom in 10%	100	15.07	8.18	4.31	1.88
Rotate 5°	25	5.24	3.62	1.97	0.81
Rotate 5°	50	11.53	6.51	2.88	1.51
Rotate 5°	100	25.48	10.91	4.53	2.70

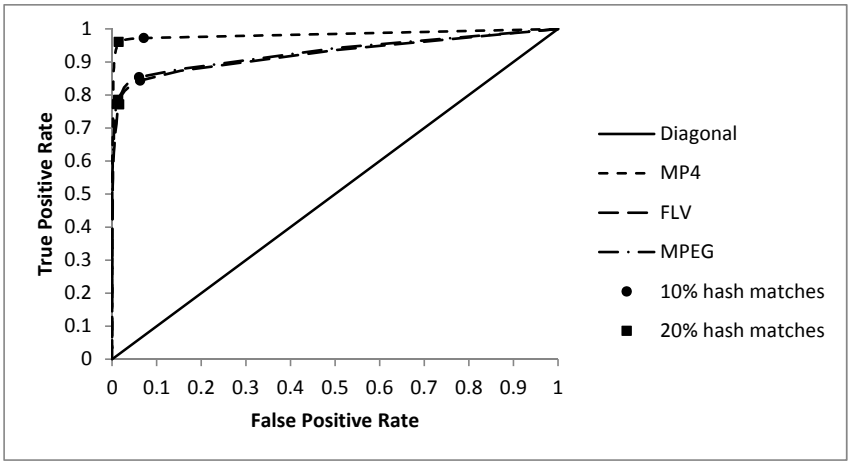
Table 5.3: Quantitive results for *hashes per frame* test.

time. This is also seen if one looks at the secondary match results in Table 5.3 and Table 5.2. The average of the secondary hashes for 50 *hashes per frame* and 16 *bits per hash* are always in the range of 3, with a deviation of 2. This means the highest standard deviation from the average value is ±5 and 5 of 50 is 10%. There may still be exceptions, so it is safe to say that the *minimum hash matches* threshold can be set to 20% of the *hashes per frame* value. So, for detection with 50 *hashes per frame*, any frame match that has 10 or more hash matches can be considered a positive match.

This value was also checked using Receiver Operator Characteristics (ROC) curves. A ROC curve is a graphical plot which illustrates the ability of a system to discriminate between positive cases and negative cases as the discrimination threshold is varied [35]. It is created by plotting the True Positive Rate (TPR) against the False Positive Rate



(a) ROC curves for *distortion results*



(b) ROC curves for *codec results*

Figure 5.4: ROC curves for tests done with 50 *hashes per frame* and 16 *bits per hash*.

(FPR) at the different threshold values. The TPR is the ratio of true positive above the discrimination threshold to the total number of true positive, while the FPR is the ratio of false positives above the discrimination threshold to the total number of false positives.

In Figure 5.4, the ROC curves are shown for the tests where 50 *hashes per frame* and 16 *bits per hash* were used. The closer the curve comes to the (0,1) point, were the TPR is 100% and the FPR is 0%, the better. Thus, we want the TPR to be as high as possible, while keeping the FPR as low as possible when choosing a threshold. It is clear that the FPR is significantly higher at 10% hash matches than at 20% hash matches. This does not mean that frame matches that get a hash match value between 10% and 20% are incorrect, it only means that there is a very slight chance that is is incorrect. Choosing 20% hash matches for the *minimum hash matches* threshold is therefore a very safe choice.

5.3 Hashing Method Optimization

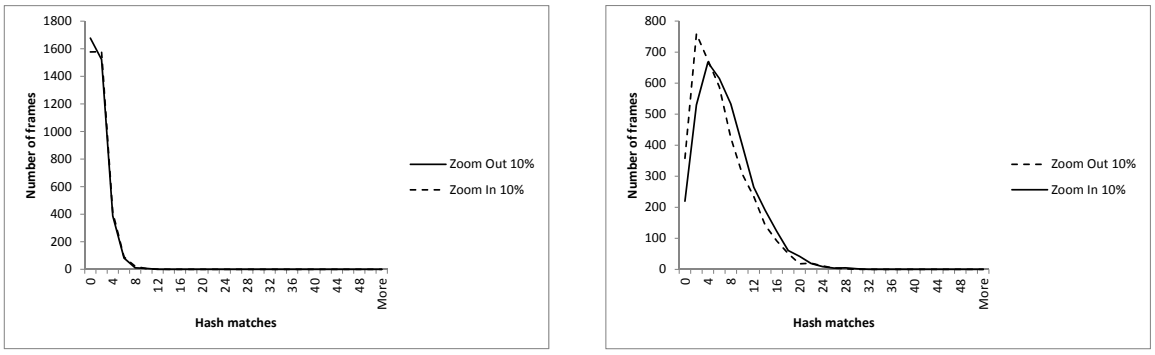
In this section the aim of the tests are to determine if detection can be improved by changing the algorithm used to create the hashes. In each test one of the variables in question will be tested to see what set-up gives the best results. The default method developed and used to create hashes is explained in Subsection 4.1.1.

5.3.1 Relative or fixed distance

The distance hash value represents the distance two paired key points are from each other. Two methods can be used to calculate this value. The first method calculates a relative distance by dividing the distance between the key points by the main key point's scale value. The second method uses a fixed maximum distance to determine the hash value. It is obvious that using the relative distance method will make the hashes more robust to scaling distortions, but the effect this choice has on detection in general still has to be tested and a decision made.

This test was done as described in Section 5.1, repeated twice: Once for relative distance hashing and another time for fixed distance hashing. Each time a database was created using the AVI *test videos* and the altered video test sets were run through the detection algorithm.

After the tests were done, the results showed minimal differences in detection rates while using the different hashing algorithms, with relative distance hashing giving slightly better results. The main difference between the two methods is the detection rates with scale distortion present. The hash match distribution results for the detections made with the scale distorted *test videos* are shown in Figure 5.5. As one can see, detection is greatly affected when the frames are scaled. This test results show that relative distance hashing's hash matches distribution is higher than the fixed distance's distribution and is thus the best way to calculate the point distance used in the hashes.



(a) Fixed distance hashes

(b) Relative distance hashes

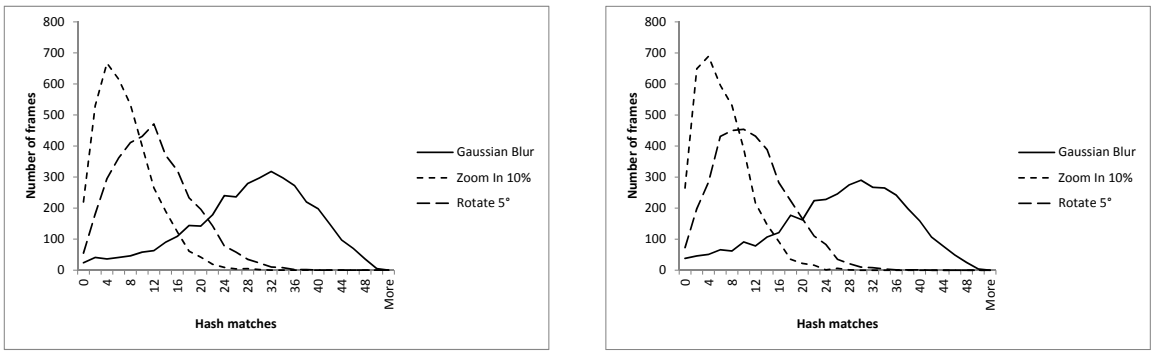
Figure 5.5: Results for the fixed and relative hashing methods using scale distorted *test videos*.

5.3.2 Sectional magnitude ratio

In Subsection 4.1.1, one of the hash values (H_1) is calculated using the ratio between the primary and secondary key points' magnitudes. The primary key point's magnitude is always bigger, so the effective range of the magnitude ratio is 0 to 1. After seeing the database saturation results shown in Figure 5.17(a) in Section 5.6, it was clear that

the magnitude ratio values are not distributed evenly among the hashes. The number of hashes that have magnitude ratios below 0.25 ($\frac{4}{16} = 0.25$) are minute compared to the rest of the hashes, therefore this test was created and executed to determine if the frame fingerprinting system can be improved by only creating hashes if their magnitude ratios are between 0.25 and 1. Only a section of the full magnitude ratio range is used with this method and therefore this test is called the sectional magnitude ratio test.

This test was set up exactly the same way as the previous test, using normal hashing (full magnitude range) and sectional magnitude ratio hashing for the two set-ups. The results displayed in Figure 5.6 show that there are minimal differences while using the full scale of magnitude ratios or only a section of it. Therefore the whole range of magnitude ratios will be used to allow the system to create hashes even if the scale ratio value is below 0.25.



(a) Full magnitude range

(b) Sectional magnitude range

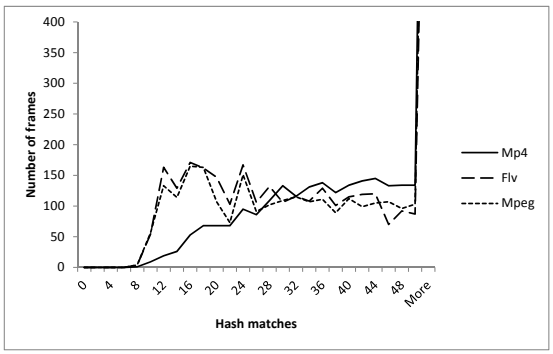
Figure 5.6: Results for the sectional magnitude ratio test.

5.3.3 Alternate hashing method

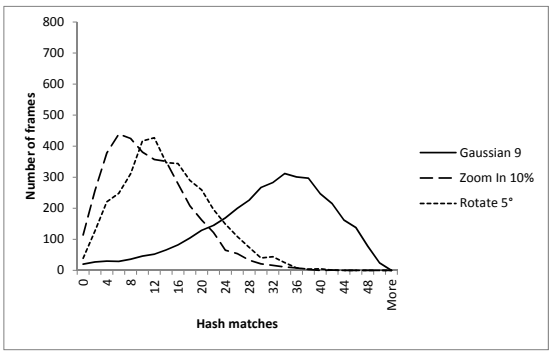
In this test the objective is to see if matching can be improved by not making the hashes rotation invariant, thus ignoring the key point's direction when calculating hashes. This test is important, as most television signals won't have rotation present and the alternate hashing may prove advantageous if it can detect frames with more certainty.

The hashes generated using the alternate hashing will consist of the magnitude ratio, relative distance to the secondary point and the direction to the secondary point. These values will be calculated in exactly the same way as the normal hashing method, except the direction to the secondary point, which won't be relative, as the key points don't have orientations.

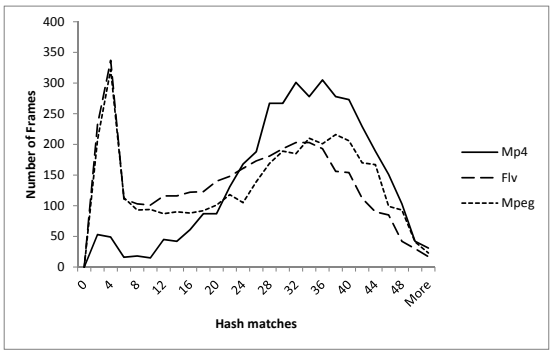
The test was set up just as described in Section 5.1 and repeated twice: Once using normal hashing and once using alternate hashing. The results for the test are displayed in Figure 5.7. The results obtained using the MP4, FLV and MPEG *test videos* are the *codec results* and the results obtained from applying blur, scaling and rotation distortions to the frames are the *distortion results*.



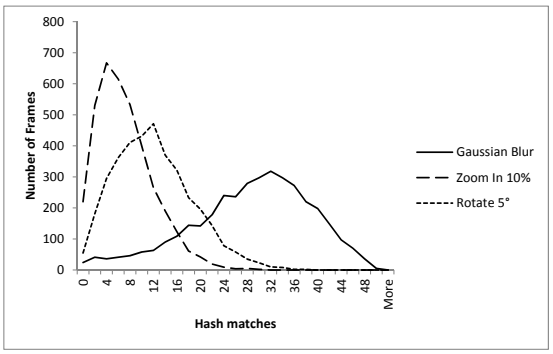
(a) Alternate hashing *codec results*



(b) Alternate hashing *distortion results*



(c) Normal hashing *codec results*



(d) Normal hashing *distortion results*

Figure 5.7: Results for alternate hashing test.

The quantitative values for the test is given in Table 5.4. The average and standard deviation of the frame detections are shown. The values were calculated using every hash

Codec or Distortion	Hashing method	Best match average	Best match standard deviation	Second match average	Second match standard deviation
MP4	Alternate	52.47	-	17.92	7.69
MP4	Normal	31.76	-	2.99	2.18
FLV	Alternate	42.62	-	17.29	7.59
FLV	Normal	23.19	-	2.94	2.08
MPEG	Alternate	46.22	-	17.38	7.42
MPEG	Normal	25.51	-	2.91	1.96
Gaussian blur	Alternate	30.53	10.93	17.93	8.01
Gaussian blur	Normal	28.05	10.21	3.02	2.14
Zoom in 10%	Alternate	10.66	6.91	17.98	7.59
Zoom in 10%	Normal	6.65	4.81	2.77	1.12
Rotate 5°	Alternate	13.64	7.36	18.45	8.77
Rotate 5°	Normal	11.53	6.51	2.88	1.51

Table 5.4: Quantitive results for *alternate hashing* test.

match value acquired during the test. The standard deviation for the true positives of the *codec results* are not shown, because the distributions are not normal. The reason for this is that the number of repeated hashes created from the frames are so high using the alternate hashing method, that the number of hash matches go way beyond the number of hashes calculated, as shown in Figure 5.7(a).

The average detection rates of the rotation invariant hashes are much higher than the normal hashing method, but the secondary false positive detection rates are also much higher, in the range of ±18 (see Table 5.4). This is a bad situation, because the detection threshold has to be high enough to eliminate false detection and with a false detection average of ±18, the threshold would would have to be extremely high. Such a high threshold would cause many false negative detection and therefore the normal hashing method is much better.

The ROC curves for the *distortion results* are also shown in Figure 5.8. In a ROC curve, the *diagonal* line represents random guessing between true and false positives. This means that a line above the *diagonal* is better than random and below the *diagonal* is worse than

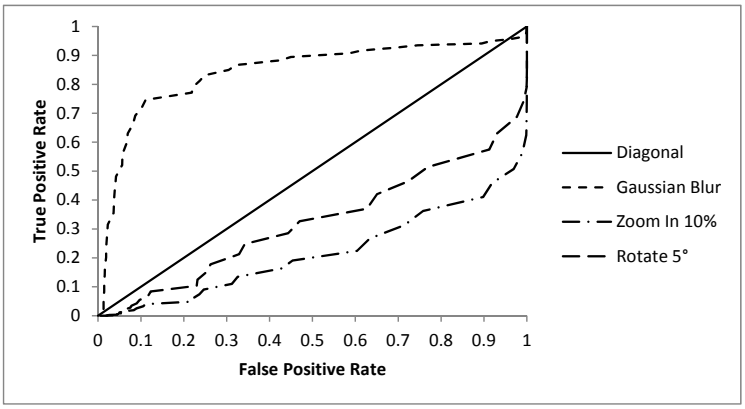


Figure 5.8: ROC curves for the *distortion results* of the alternate hashing method.

random. As one can see, the Rotate 5° and Zoom In 10% results' curves are below the *diagonal*, which is poor, because a threshold cannot be set to distinguish between true positives and false positives.

5.4 Codecs and Distortion

In this section the tests are done to determine the capabilities and limits of the system when the videos' frames are detected with different image distortions. By using different video codecs, the frames are distorted because of compression of the data, thus tests have to be done to evaluate the effect of the compression distortion on detection. The effect of common image distortions like blur, scaling and rotation also have to be tested.

While using the system for advertisement tracking, the chances of receiving a television broadcast with heavy distortion is very rare. These tests are only done to make sure that the system can handle distortions, as it will encounter these distortions very seldom in practice.

5.4.1 Codecs

In this test, the *test videos* (30 music videos) were fingerprinted and added to the database. These videos were converted to other wrappers and codecs (see Table 5.1) using Format-Factory and then the converted videos' key frames were matched to the database. The first set of tests included matching AVI, FLV and MPEG *test videos* to a database generated from the MP4 *test videos*. The results for these tests are shown in Figure 5.9. The tests were then repeated with a database generated from the AVI *test videos*, this time matching MP4, FLV and MPEG *test videos*, and these tests' results are displayed in Figure 5.10.

As one can see, the compression distortion causes some of the hashes to be missed while matching. This was to be expected, as different codecs introduce little distortions into the videos' frames, causing the hash generating algorithm to create some hashes that are slightly different from the originally fingerprinted frame's hashes. Even with this distortion present the algorithm can still detect the frames, because most of the detections' hash matches are above the *minimum hash match* threshold that was set in Subsection 5.2.2.

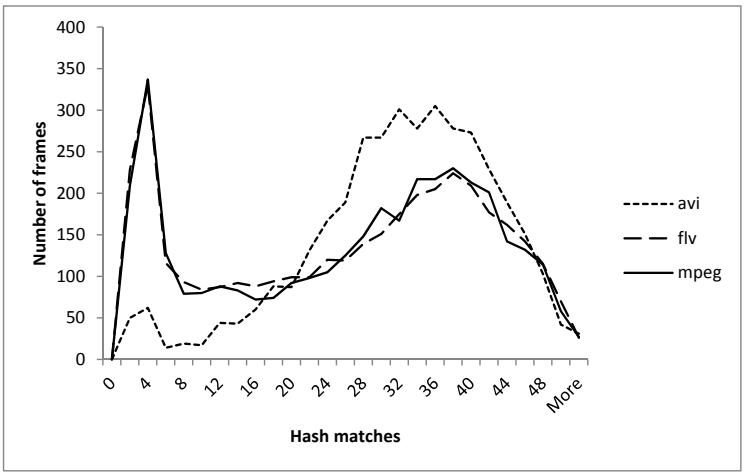


Figure 5.9: Results for codecs tests with MP4 *test video* database.

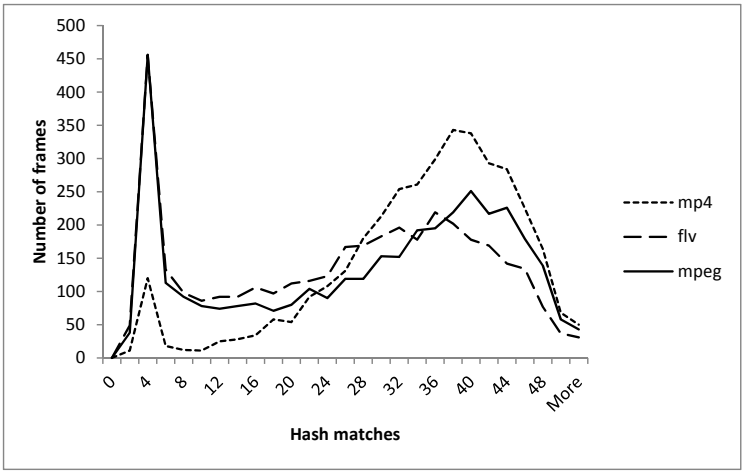


Figure 5.10: Results for codecs tests with AVI *test video* database.

5.4.2 Distortion

This test was done by adding the fingerprints generated from the AVI *test videos* to the database and then matching the same videos' key frames to the database, but applying distortions to the frames before they were matched. Distortions were applied to frames by using built in image distortion functions in the EmguCV library. Examples of the distorted frames with their SIFT key points projected on them are displayed in Figure 5.14.

A lot of different levels of blur, scaling and rotation distortions were used during the tests, but only one distortion was applied to the frames at a time. The detection results for the tests are shown in Figures 5.11, 5.12 and 5.13. The true positive hash match distributions are shown in these figures, but the best false positive matches were also determine by finding the best frame match to the database, excluding the original frame.

After inspection of the results, one will notice that Gaussian blur, shown in Figure 5.11, does not affect detection that much. The reason for this is that the SIFT algorithm that is used to detect the key points, uses Gaussian blur to smooth the images before detecting the key points (see Section 3.2), therefore the extra Gaussian blur doesn't have a big

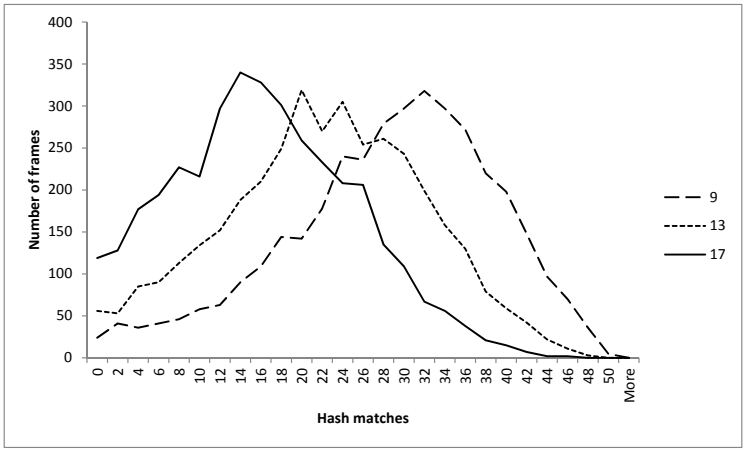


Figure 5.11: Results with Gaussian blur applied to the test frames. The numbers 9, 13 and 17 refer to Gaussian filter sizes of 9×9, 13×13 and 17×17 used to filter the image.

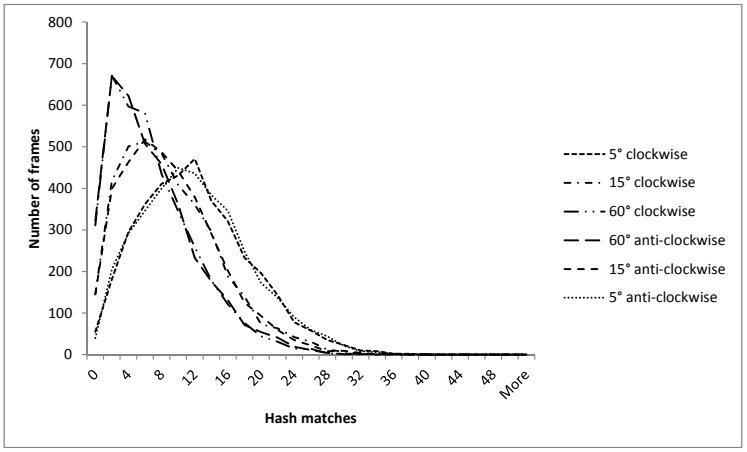


Figure 5.12: Results for the test frames with different angles of rotation.

influence.

The lower detection rates seen in the rotation test results, displayed in Figure 5.12, can be attributed to the fact that extra key points are detected or some key points aren't detected after rotation. Also, small perturbations in the values may affect the hashes enough to not match exactly. Even with these conditions, most of the frame matches' corresponding hashes are enough to differentiate it from non-matches.

As for the linear scaling or resizing of the frames, the detection rates, displayed in Figure 5.13, can also be attributed to the fact that new key points are detected or key points

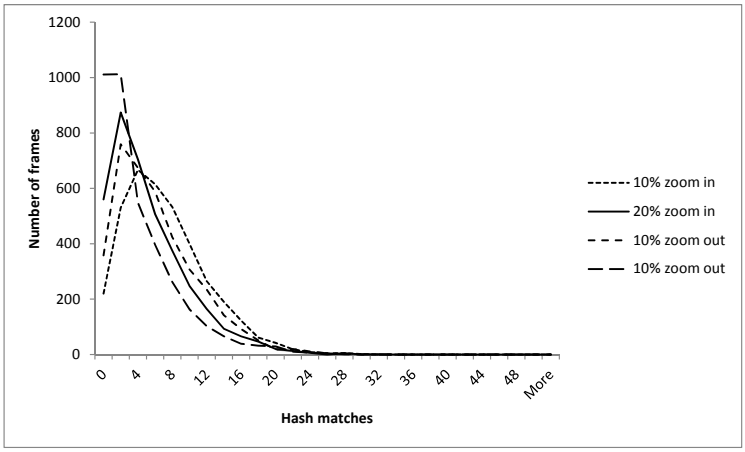


Figure 5.13: Results for linear scaling/resizing of the test frames.

are missed after the frame is scaled. If the frame isn't scaled a lot, the right key points may still be detected, but if the scaling is drastic, the key points will certainly be missed, because only a certain scale of key points are detected for a frame to create the fingerprint with.

These tests show how the system reacts to distortion and it may not be as robust when compared to other existing fingerprinting techniques, but the lower level of robustness was the result of a trade-off taken into consideration when developing the fingerprinting system. To reach real-time detection speeds the system cannot be overly robust, and thus only minor distortion are handled by the system.

5.5 SIFT vs. SURF

In this test we compare the use of SIFT and SURF key points in the frame fingerprinting system. A database is created from the fingerprints generated from the AVI *test videos* and then the MP4, FLV and MPEG *test videos* are fingerprinted and matched to the database while the the hash match results are saved. AVI *test videos'* key frames were also distorted and matched to the database to test the effects of distortion on detection rates. The distortions applied were Gaussian blur, linear scaling and rotation. This test

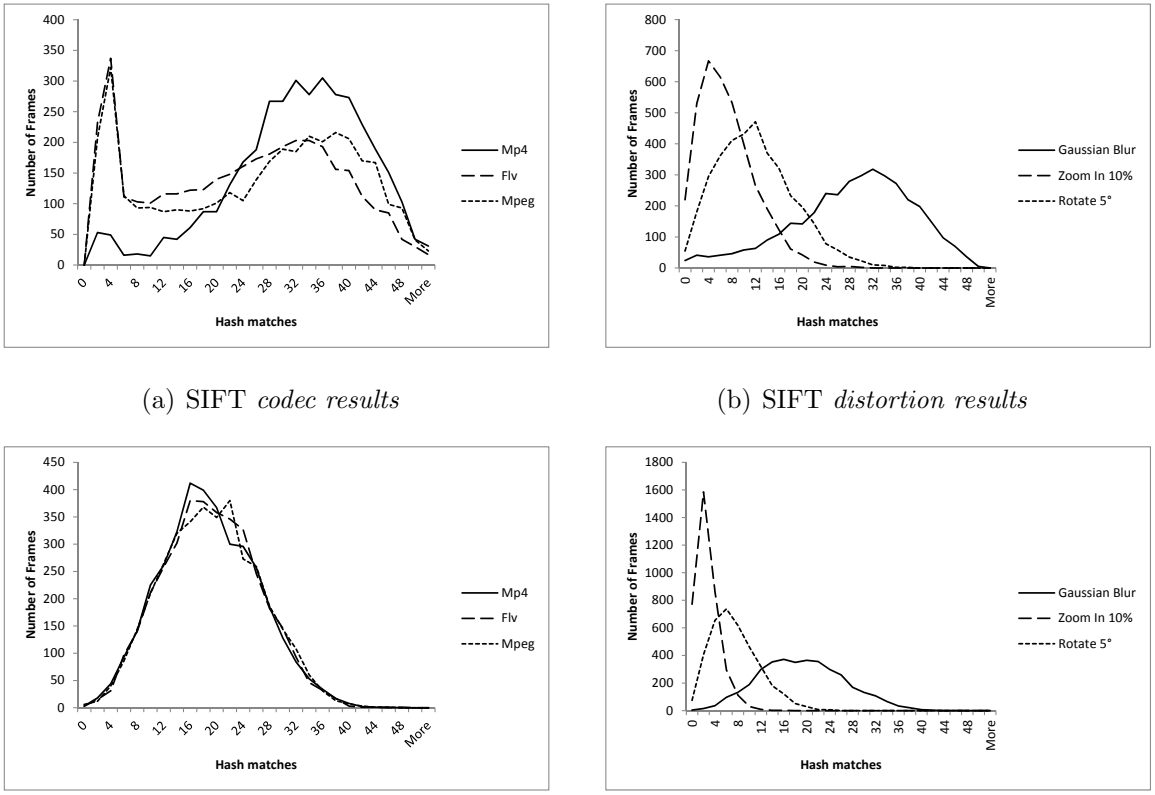
(a) Example frame (b) Gaussian blurred

(c) Zoomed in 10% (d) Zoomed out 10%

(e) Rotated 15° (f) Rotated 60°

Figure 5.14: SIFT key points are displayed on a frame and distorted versions of the frame.

was repeated twice, using SIFT key points the first time and SURF key points the second.



(a) SIFT *codec results*

(b) SIFT *distortion results*

(c) SURF *codec results*

(d) SURF *distortion results*

Figure 5.15: Results for the SIFT and SURF tests.

If one looks at the results of the SIFT and SURF tests in Figure 5.15, it is clear to see that the detection rates are much higher using the SIFT key points. The reason for this may be the fact that the SIFT key points are more repeatable under the conditions. The conclusion drawn from this test is that the SIFT key points produce better results and will thus be used in the video fingerprinting system.

5.6 Database Saturation

An important aspect in any fingerprinting system is the database. In this case, the database is made up of spots that get filled with appropriate VFID values. In this test,

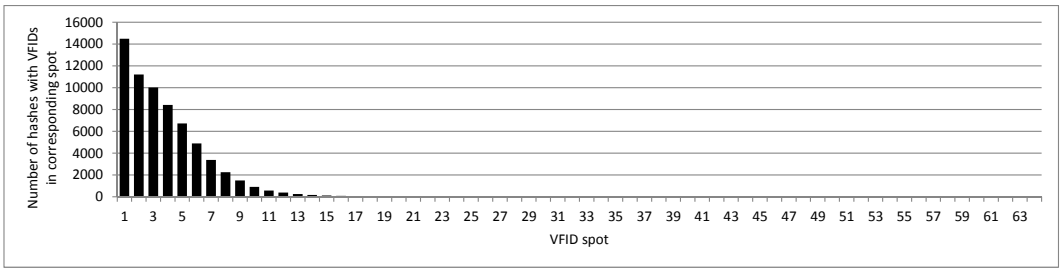
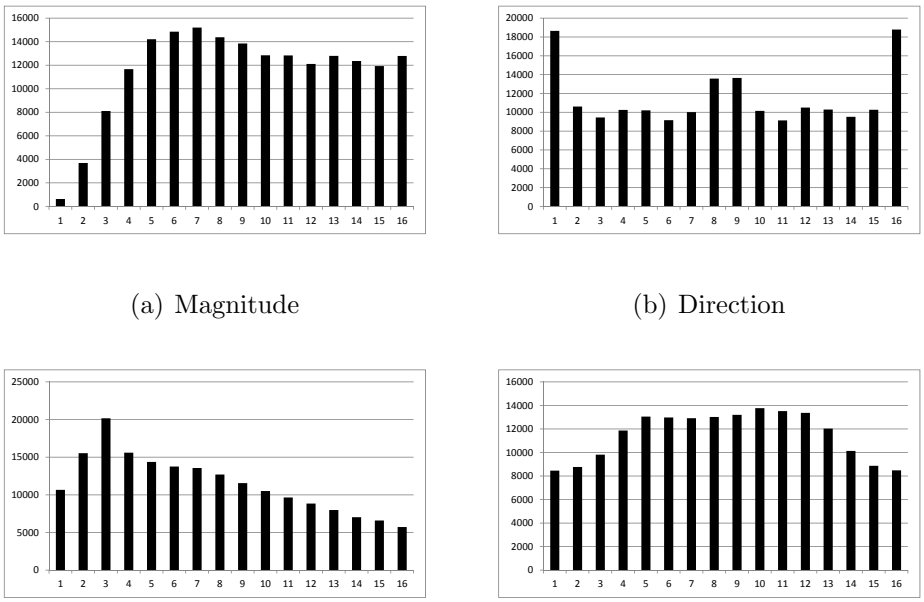


Figure 5.16: A frequency histogram of the number of hashes containing a VFID in the corresponding spot.

the database will be monitored and meta data will be extracted to determine how fast the database fills up and how the fingerprint information is distributed.

To test how quickly the database fills up, a set of videos (30 music videos) with a combined length of 1 hour, 57 minutes and 13 seconds (1.954 hours) was run through the system and fingerprinted. After the fingerprints were added to the database, the information showed in Figure 5.16 was extracted from the database.

3687 key frames were detected in the AVI *test videos*. This means that a total of 184350 hash values were calculated and added to the database, because every key frame's fingerprint consists of 50 hashes. The number of key frames per second of video data is calculated to be approximately 0.52 with this set of test data. The database can take 4194304 VFID values with 16 bit hashes and 64 VFID spots per hash, which means it can take ±44 and a half hours' fingerprints ($\frac{4194304}{184350} \times 1.954 hours = 44.457 hours$) if it fills up uniformly, but it doesn't. In Figure 5.17 the distribution of hash values are shown. With the hash value distributions not exactly uniform, some hashes' VFID spots will fill up faster than others'. Therefore the effective size of the database will be smaller, meaning that it will not be able to save 44.5 hours' video fingerprints without some loss in fingerprint data, because the algorithm will discard fingerprint hashes if all the slots of the corresponding hash value is used. The database may be able to hold 30 hours' video fingerprinting data by looking at the hash value distributions in Figure 5.17 and still be reliable, but no concrete testing has been done. If the database is too small for the application, the number of VFID spots per hash can be increased. This increase may



(a) Magnitude

(b) Direction

(c) Distance

(d) Point Direction

Figure 5.17: Distribution histograms showing how the different hash values are distributed among all the hashes.

influence search times, but not by a lot, especially if a SQL server implementation is used.

5.7 Key Frame Detector (KFD)

Key frame detection is very important for the successful matching of a video's frames. The same frames have to be selected as key frames every time a video is run through the algorithm, so matches will be found in the database. This means that repeatability is one of the most important characteristics of a KFD. In this section the robustness of the KFD developed for this video fingerprinting system is tested.

The recall and precision measurements are used to evaluate information retrieval systems, and can be defines as:

Wrapper	Codec	t_p	f_p	f_n	Precision	Recall
avi	XviD	3594	93	89	0.975	0.976
mpg	mpeg1	3568	78	116	0.979	0.968

Table 5.5: Key Frame Detector (KFD) test results.

$$precision = \frac{t_p}{t_p + f_p} \tag{5.1}$$

$$recall = \frac{t_p}{t_p + f_n} \tag{5.2}$$

Where t_p is the true positives (correct detections), f_p the false positives (wrong detections) and f_n the false negatives (missed detections). The results are shown in Table 5.5 after comparing the key frames detected in AVI and MPEG videos to the key frames in MP4 videos.

The results show that a lot of the key frames detected in the AVI and MPEG videos correspond to the key frames in the MP4 videos, even though this is a very basic KFD design. This is very good, because the *recall* values calculated are very high and it means that the detector's repeatability is great. The results prove that the KFD can function properly and that it can be used to detect key frames reliably in a video stream.

5.8 Video Detection

In this section the best parameters, selected according to the frame fingerprinting test results, are used, along with a KFD, to detect advertisements broadcast over a television channel. This test is important to show that the system can function as a whole, although all the subsections of the system have been tested and have been proven to function properly.

5.8.1 Experimental setup

To test the video detection system, a day's broadcast of SABC 2, a local South African television channel, was made. All the advertisements were then extracted from the video stream, using a video editing program, and saved separately. The advertisements were then fingerprinted and added to the database using 50 hashes per key frame and 16 bit hashes.

After the advertisements were successfully added to the database, the original 22 hour video stream (3am to 1am) was run through the detector. The video wasn't converted to another codec and no distortion was added.

5.8.2 Results

In Table 5.6 the results are shown, with the first line only referring to the advertisements and if they were detected or not during the day's video footage, whereas the the second line represents all the advertisements shown during the day, repeats included.

	t_p	f_n	f_p	Precision	Recall
Advertisements	124	4	0	1	0.969
Advertisements (repeats included)	188	9	0	1	0.954

Table 5.6: Results of advertisement tracking/broadcast monitoring using video fingerprinting.

As one can see, the repeats were also detected, even though only the first occurrence of the advertisement was added to the database. After further investigation, it was noted that the advertisements that were missed was because the KFD failed to detect sufficient key frames in each of the videos. This is because the KFD detects abrupt changes and these advertisements didn't have any.

5.9 Real-time System

In this section the system was tested and timed to prove that it monitors videos in real-time. An input stream with a frame rate of 25 and a resolution of 320 × 240 pixels was used. The tests were run on a HP Probook, with an Intel® Core™ i3 CPU @ 2.40 GHz, 4 GB RAM and an ATI Radeon HD 530v graphics card on a 32-bit Windows 7.

Add/Detect	Videos	Time (mm:ss)	Key frames
Add	30 music videos	06:43	3685
Detect	30 music videos	10:24	3685
Add	128 advertisements	05:27	1988

Table 5.7: Adding and detection times for videos used during testing.

In Table 5.7, the time it took to add videos to the database and detect videos are shown. The total time of the 30 music videos is 1 hour, 57 minutes and 13 seconds and the total time of the advertisements is 1 hour, 11 minutes and 26 seconds. Each set of videos was fingerprinted and added to an empty database to determine the fingerprinting time and the database was filled with 30 music video's fingerprints when the detection was done. In Table 5.8 the detect time for each hour of television broadcast is shown along with the total time and the average time per hour. The database contained the 128 advertisements' fingerprints while the SABC 2 broadcast videos were detected.

With all this information on the fingerprint adding and detection times one can easily see that the algorithm takes very little time in relation to the total video length to detect videos or add them to the database. The detection times are the most important, because it influences the speed of the system. The add times are only mentioned for interest's sake.

In Table 5.9 the average frame detection times are shown for the cases where MP4 and FLV videos were matched to a database of AVI video fingerprints. The average time is ±85 ms for a detection which accounts for 5 minutes and 23 seconds of the 10 minute and 24 second total time it took to detect 30 MP4 music videos. This means that the other 5 minutes of the detection time was used to read the videos' frames and detect key frames,

Hour	Detection time (mm:ss)	Video length (mm:ss)
03h00	01:44	34:55
04h00	01:26	30:50
05h00	01:46	33:55
06h00	03:29	59:42
07h00	02:41	59:42
08h00	03:43	59:44
09h00	02:57	59:44
10h00	04:09	59:44
11h00	02:19	35:49
12h00	02:53	59:44
13h00	03:30	59:44
14h00	03:54	59:44
15h00	04:14	59:44
16h00	03:35	59:44
17h00	03:33	59:44
18h00	03:28	59:44
19h00	03:27	59:44
20h00	03:58	59:44
21h00	03:16	59:42
22h00	04:07	59:42
23h00	04:27	59:41
24h00	04:01	59:59
Total	74:31	20:10:56
Average	03:23	55:03

Table 5.8: Detection times for SABC 2 video.

resulting in an average of 0.131 ms (1 hour, 57 minutes and 13 seconds of video @ 25 fps) of processing time per frame for reading and key frame detection.

A normal video runs at a frame rate of 25, thus the frames are refreshed every 40 ms. This means that it takes 2 or 3 times longer to match a key frame to the database than it takes to refresh a video's frame, but this isn't a problem, because key frames are only detected roughly every 2 seconds or 50 frames (see Section 5.6). This gives the system more than enough time to read frames, determine if they are key frames and match them to the database if they are, resulting in a system that can easily run in real time.

Test number	MP4 Average	MP4 Standard deviation	FLV Average	FLV Standard deviation
1	84.81	21.06	85.94	22.95
2	84.88	20.95	85.27	21.24
3	84.96	20.91	84.90	23.78

Table 5.9: Frame detect time (in milliseconds) using MP4 and FLV videos

5.10 Verification and Validation

In this dissertation a video fingerprinting algorithm was implemented as a software product, therefore the standard verification and validation techniques for software models will be used. For **verification**, one has to prove that the model was built right. This means that the input parameters and logical structures of the conceptual model should be correctly implemented in software [36]. Sargent defines verification in [37] as: "Ensuring that the computer program of the computerized model and its implementation are correct", while Law and Kelton [38] define it as: "Verification is determining that a simulation computer program performs as intended, i.e., debugging the computer program."

Validatation of a software product, on the other hand, involves proving the correct model was implemented. This is done by proving that the model is an accurate representation of the real system by comparing the model to the actual system behaviour [36]. The definition given by Sargent for validation is: "Substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model", and by Law and Kelton: "Validation is concerned with determining whether the conceptual simulation model (as opposed to the computer program) is an accurate representation of the system under study."

5.10.1 Verification

To verify the software program, it was debugged to make sure the conceptual model's logic was correctly programmed. The code was also double checked by my study leader, Professor W.C. Venter, to make sure that the logic and implementation of the program was correct. The main functions used in the program are discussed below along with the methods used for their verification.

- **Preprocessing frames:** This functionality involves the normalization of frames, so the rest of the algorithm receives the input in a format it knows. To open video files, acquire the frames, resize them and convert them to gray scale, the open source library, EmguCV, was used. To prove that the frames were converted correctly, the output was displayed on the screen for visual inspection and the parameters of the image object was monitored in debug mode.
- **Acquiring SIFT key points:** This function in the algorithm was also "borrowed" from the open source library, EmguCV. To prove that the key points, were in fact detected correctly, another function was written to display them on the frame images. In the OpenCV documentation, a comparison is made between the original SIFT algorithm's results and their results.
- **Creating hashes:** The hash creation algorithm was developed for this video fingerprinting algorithm and is discussed in Section 4.1. To prove that the hashes are created correctly, one can look at the results.
- **Save fingerprint:** To verify that the program saves fingerprints correctly, the database was opened and inspected after fingerprints were added. The VFID values were saved in the correct hash value's spots in the database, and the extra video data was also saved correctly.
- **Match fingerprints:** The matching function was verified by creating fingerprints for frames and saving them to the database and then fingerprinting the same frames again and running the match algorithm. All the hashes were matched with those in the database, proving that the search and match function works correctly.

- **Detecting key frames:** To ensure that the KFD functioned correctly it was tested as shown in Section 5.7. The function was also debugged, stepping through videos, frame by frame and monitoring what the KFD function does. This is also how the threshold and parameter values were chosen for the KFD.
- **Adding distortion for testing:** While testing, distortion was added to the frames to test the robustness of the fingerprinting system. To ensure that the distortions were applied correctly to the frame, the output was displayed and visually inspected. An example of the distorted frames are shown in Figure 5.14.
- **Converting video to other formats:** The videos were converted to other formats using the program FormatFactory. To ensure that the videos were correctly converted, a program called GSpot (v2.70a) was used to inspect the parameters of the videos after conversion.

5.10.2 Validation

To validate the software implementation it is must be proven that the correct model was implemented for the intended application. This means that the system must be proved to be a video fingerprinting, but it cannot be compared to a previously implemented version of the model, because the model designed in this dissertation is a new one.

Thus, to prove that the software program is an implementation of a video fingerprinting system, the system will be inspected to see if it behaves as expected, by comparing results to a prediction made beforehand. In Section 4.2, the tests were done to show that advertisements were detected in a television broadcast video stream. The fact that the system detects advertisements, that were previously added to the database, in a unknown video stream, proves that the system does what it is supposed to do and that it conforms to the predictions made about it at the start of the dissertation.

In the tests in Chapter 5, predictions were also made concerning the expected results after distortion was added or the hashing algorithm changed slightly. This is also a form of validation that strengthens the validity of the model and its implementation.