Fast Feature Detection with a Graphics Processing Unit Implementation

Charles Bibby and Ian Reid

Dept of Engineering Science
University of Oxford
Oxford, OX13PJ, UK
cbibby@robots.ox.ac.uk,ian.reid@eng.ox.ac.uk

Abstract. We present a fast feature detector that uses the Graphics Processing Unit (GPU) to extract distinctive features from an image, which are suitable for reliable matching between different views of an object or a scene. This detector is a particularly useful tool for the computer vision community because it is: (i) simple to implement; (ii) extremely fast; (iii) operates stand alone - requires no post-processing or filtering of detected features. We compare the fast feature detector to other similar published algorithms and demonstrate that it extracts features in desirable/salient locations, repeatedly and more rapidly than other scale-space detectors. We further describe implementations of two demanding real-time application scenarios: (i) real-time detection and tracking of multiple visual targets; (ii) real-time visual SLAM, with the results of each confirming the efficacy of the new features. In this latter application we make a comparison between full frame feature detection and Davison's [3] active search technique and discuss the trade off between the two.

1 Introduction

This paper presents a fast feature detector that uses the Graphics Processing Unit (GPU) to extract distinctive features from an image. Our analysis shows that our features demonstrate good repeatability between pairs of images, over a variety of different image content and transformation types. More compellingly we show that these features are useful for mobile vision tasks such as visual SLAM. This is demonstrated in section 5 with a SLAM example, which uses our features as a replacement for more typical corner features [10], [4] and compares full frame feature detection against an active search approach.

Our motivation for developing this feature detector was primarily the problem of detecting and tracking multiple visual targets at sea. We use our feature detector to find features that may correspond to a target and then cluster these features into small constellations, which are tracked using a Probabilistic Data Association Filter (PDAF) [1]. The feature detector therefore needs to use as little of our 33ms frame budget as possible so that other components of the tracking system have time to run. Section 4 shows how the feature detector fits into the multi-target tracking system.

The problem with many existing techniques [5], [10] is that they are either slow or require a post-processing stage to filter features that are poor for matching. For example Lowe uses a Difference of Gaussian (DOG) feature detector to obtain scale-invariant features, and then applies a post-processing stage rejecting those which may be poorly or ambiguously described by his SIFT descriptor [5]. Our feature detector which we term the Fast Radial Blob Detector (FRBD) can run in a mode that has been designed specifically to reduce the detection of poor features, in particular suppressing edge responses which are bad for matching because they suffer from the aperture problem. We look at a GPU implementation of both the DOG detector and the FRBD, showing that with a standard PC it is possible to achieve full frame feature detection on a 640x480 monochrome sequence, or 320x240 colour sequence, at ≈ 100 frames per second.

The core contributions of this paper are: (i) we propose a novel fast feature detector; (ii) we provide a comparison of feature detectors over a variety of data sets; (iii) we show how our feature detector can be used as an alternative to many widely used features within computer vision; (iv) we present a comparison between 'brute-force' full-frame feature detection and the idea of active search as suggested by Davison [3].

The paper is organised as follows: we begin in section 2 by proposing our FRBD algorithm, then in section 3 we compare the FRBD to other feature detectors, section 4 looks at a multi-target detection and tracking application, section 5 looks at a visual SLAM application [3], and finally in section 6 we conclude with a discussion.

2 The Fast Radial Blob Detector

Given an image I, we would like to acquire a set of feature $F(i), i \leftarrow [1..N]$, that we deem to be salient, see Figure 1.

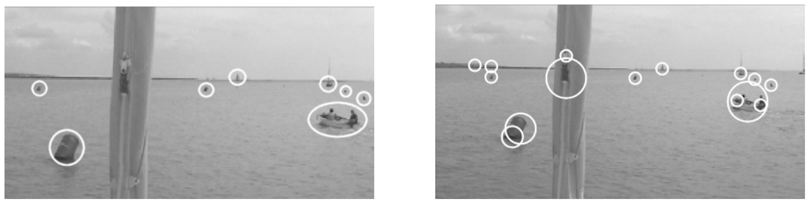


Fig. 1. (left) A typical image, with features of interest manually highlighted in white; (right) what we can achieve with our FRBD.

Our inspiration for the FRBD is based on the principles behind the Laplacian of Gaussian (LOG) operator

$$LoG(x,y) \triangleq \triangle G(x,y,\sigma) = \frac{\delta^2 G(x,y,\sigma)}{\delta^2 x} + \frac{\delta^2 G(x,y,\sigma)}{\delta^2 y}, \qquad (1)$$

where $G(x, y, \sigma)$ is $2D$ gaussian with scale σ. The LOG will respond to regions of high curvature in the scale-space image

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \tag{2}$$

which is a desirable property of a feature detector because it finds regions where the image gradients are changing quickly e.g. blobs, corners and edges. The problem with this technique is that its computational complexity is $O(N\sigma^2)$ where N is the number of pixels, making it slow in practice for any significant σ.

One alternative, which has enjoyed a renaissance in recent times due to its use by Lowe in his SIFT work, is the Difference of Gaussian (DOG) operator. It is an approximation to the LOG, which can be computed more rapidly.

2.1 The FRBD

Our FRBD takes this approximation one step further by using second-order finite differencing to approximate how the smoothed image $L(x, y, \sigma)$ changes at a given pixel. We use a circular geometric construction with radius r and sample at eight discrete orientations, see Figure 2.

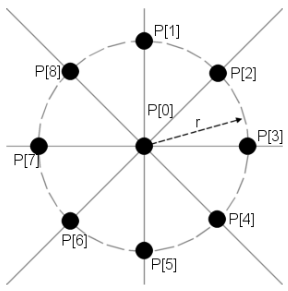


Fig. 2. Fast Radial Blob Detector.

Using these sample points $P[0 \ldots 8]$ we compute the average pixel difference around pixel $P[0]$ defined as

$$F(x, y, r) = abs(\sum_{i=1}^{8} P[0] - P[i]), \tag{3}$$

where $P[0]$ is centred on the pixel $L(x, y, \sigma)$ and the locations for $P[1 \ldots 8]$ are constructed according to Figure 2. This average pixel difference $F(x, y, r)$ is identical to convolving the original image I with the kernels shown in Figures 4(a) and 4(b). Features are then extracted by maximising over r, the rate of change of $F(x, y, r)$ which is calculated by first-order finite differencing

$$R(x, y, r) = F(x, y, r) - F(x, y, r - 1). \tag{4}$$

2.2 Edge Suppression

Optionally $R(x, y, r)$ can be multiplied by the minimum pixel difference

$$F_{min}(x, y, r) = \min_i(|P[0] - P[i]|), \tag{5}$$

which has the effect of suppressing locations that do not exhibit changes in intensity in *all* directions e.g. edges. We use this facility in our multi-target tracking framework to suppress edge responses, which removes the need for a post-processing stage to filter the features.

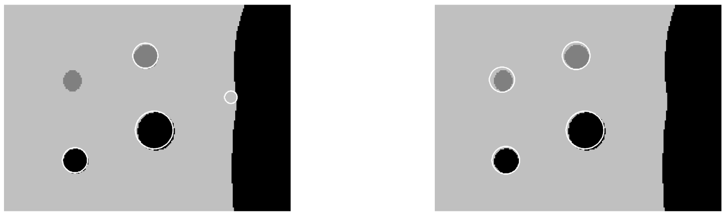


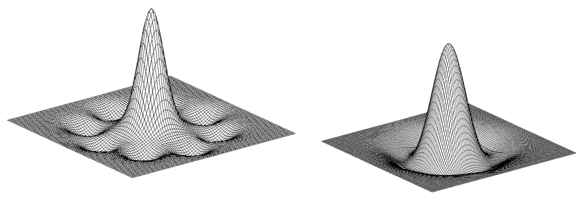
Fig. 3. Shows how edge suppression stops the detection of edges: (left) no suppression and a target is detected on the black edge; (right) suppression is on and all four blobs are detected.

2.3 Parameters

The FRBD has two parameters: (i) the radial range $r \leftarrow [r_{inner} \ldots r_{outer}]$ to search over; (ii) the width σ of the gaussian smoothing. Figures 4(a) and 4(b) illustrate the effects of these parameters on the kernel $F(x, y, r)$. If σ is too small for the current r then some of the pixels along the circumference no longer contribute to the response, which means that the detector is no longer rotationally invariant, see Figure 4(a). In contrast Figure 4(b) shows good balance of σ against r where the influence from the pixels on the circumference is approximately rotationally invariant. In practice we have found that you can search a small range of r for a given σ without seeing any artifacts due to sampling along the eight orientations. This is one reason why the FRBD is faster than other scale-space methods because with a single gaussian convolution several separate scales are searched. For our experiments we let $r \leftarrow [1 \ldots 5]$ and $\sigma = 2$.

3 Comparison of Feature Detectors

We have compared our FRBD against the following feature detectors:



(a) Kernel ($\sigma = 1$, $r = 5$) (b) Kernel ($\sigma = 2$, $r = 5$)

Fig. 4. FRBD kernel examples.

- **Rosten's FAST detector** [8]: This detector works directly on the image I. It searches for chains of pixels on the Bresenham circle (radius=3) where the absolute difference between them and the centre pixel $I(x, y)$ is greater than a preset threshold. Rosten develops a series of clever optimisation tricks to make this run at speeds of around 400Hz making it probably the fastest feature detector to date. There are various versions of the FAST detector relating to the number of pixels that are required to be in a chain before a feature is detected, we have compared FAST_9, FAST_12 and FAST_16, which will detect weak corners, corners and blobs respectively.
- **DOG / LOG** [5]: LOG, and its faster counterpart DOG, detects image regions that exhibit high curvature in intensity values. They will detect blobs followed by corners and finally edges.
- **Harris' corners** [4]: Looks for image regions that exhibit two large image gradients, i.e. corners.

3.1 Repeatability

To test the performance of our FRBD against these feature detectors we used a repeatability measure as proposed by [6], which is the ratio of correct matches against possible matches between two images I_1 and I_2. A feature detected in I_1 is said to match if when projected into I_2 there is a feature, detected in I_2, within a five pixel radius. Though this metric has weaknesses (in particular it can not distinguish between correct and incorrect matches, merely proximity) it does provide some useful insight and has been used by others [6], [9]. The reason we use this metric is to avoid the need for an image descriptor for feature matching, which may favour certain features over others. We have used the data set used by Mikolajczyk [6] and extended it to include some examples of typical environments in our lab. The data set contains pairs of images with varying content and transformations between them which include: viewpoint changes, zooming, rotation, lighting changes and blurring. Figure 5 shows five of the twenty image pairs used for testing and Figure 6 shows a typical repeatability graph. The repeatability graphs exhibited different characteristics and although there were certain detectors that seemed to do better than others it was hard to draw any conclusions. In order to make more sense of the results we generated a ranking table, which for every graph (i.e. each image pair) at every sample point (i.e. number of features detected) we generated a ranking. These rankings were then accumulated into histograms for each detector. Thus, our FRBD is highest ranked for 160 of the possible 400 sample points (20 samples per image pair and 20 pairs).

3.2 Speed

We compared the speed of DOG and FRBD implemented on the GPU, and we know the performance of Rosten's FAST detector. Although we have not implemented optimised versions of Harris or LOG, there appears to be no reason why they could not be implemented on the GPU for speed. Rosten's FAST detector is the most rapid at $400Hz$ followed by our FRBD at $100Hz$ and DOG at $30Hz$. Harris on the GPU would probably lie between FRBD and DOG, and LOG would be slowest (for significant σ).

This makes our FRBD and Rosten's FAST detector ideal tools for real-time computer vision tasks, for example: Visual SLAM [3], or real-time structure from motion [7]. The main benefits of our detector over FAST are: (i) it computes a value for feature strength; (ii) it calculates a scale; (iii) the gaussian blur stage makes it more resilient to image noise.

3.3 Analysis of results

Figure 7 shows the histograms of the ranking positions for the eight feature detectors. The FRBD generates the most promising histogram with the most dominant bin being 1st place and rarely coming anywhere below 5th place. The FRBD with edge suppression and FAST_9 give middle performances with FAST_9 normally taking 3rd, 4th or 5th place. DOG, LOG, and HARRIS show no particular pattern, which might be a trade off between the more structured scenes in the data set, which favour HARRIS, and the more natural scenes (fewer high contrast corners), which favour LOG or DOG. Finally FAST_12 and FAST_16 exhibit the worst performances, with FAST_16 coming last for the majority of the time because it only responds to small blob features, which some images might not exhibit very often.

Although these histograms have shown some interesting trends they have also reinforced the fact that the performance of a given detector is highly dependent on image content. For this reason we suggest they are taken with certain skepticism along with similar repeatability results in the literature. We recognise the need for a more valuable performance measure that relates to the expected performance in a SLAM or visual odometry system.

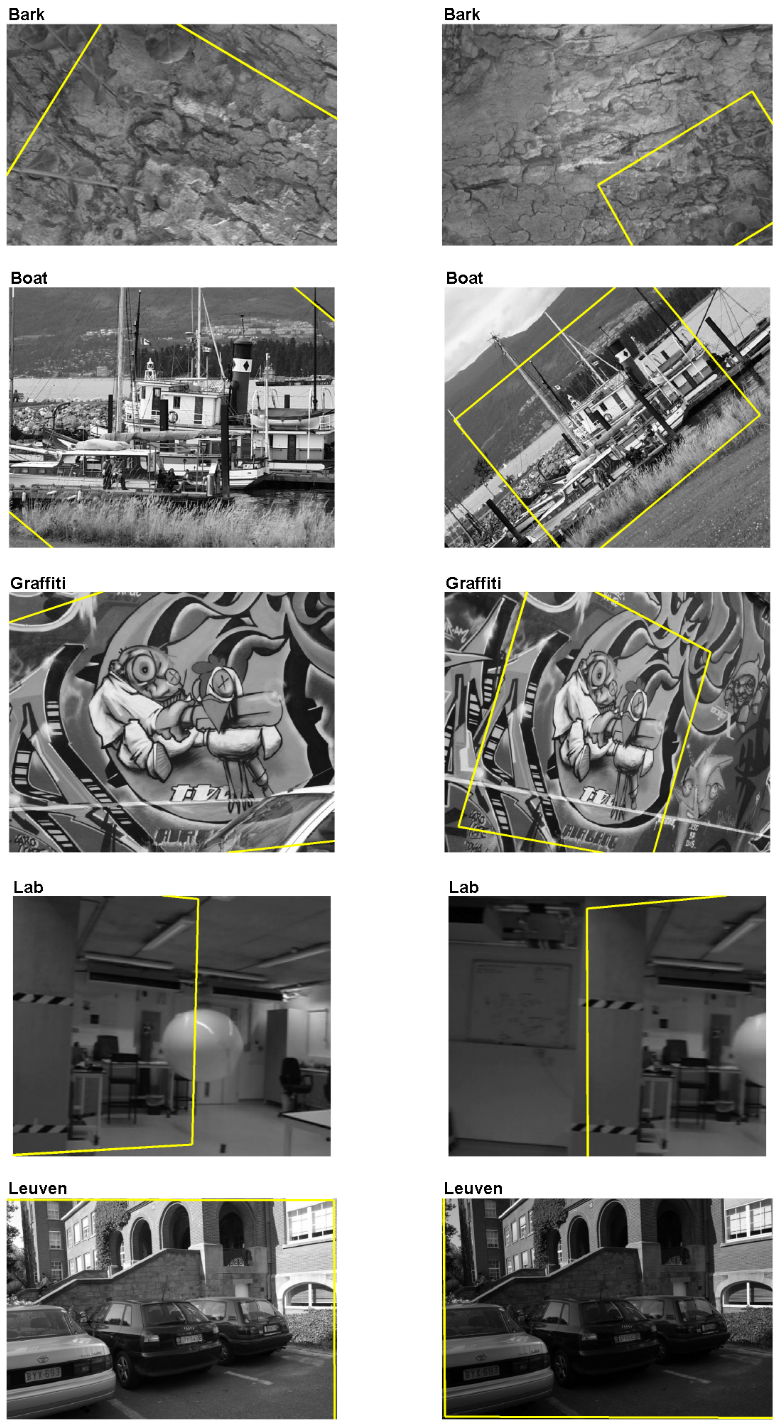


Fig. 5. Example data sets: (1st row) zoom and viewpoint; (2nd row) zoom and rotation; (3rd row) viewpoint; (4th row) viewpoint; (5th row) lighting change. The superimposed quadrilaterals represent the homography of $I1$ in $I2$ and vice versa.

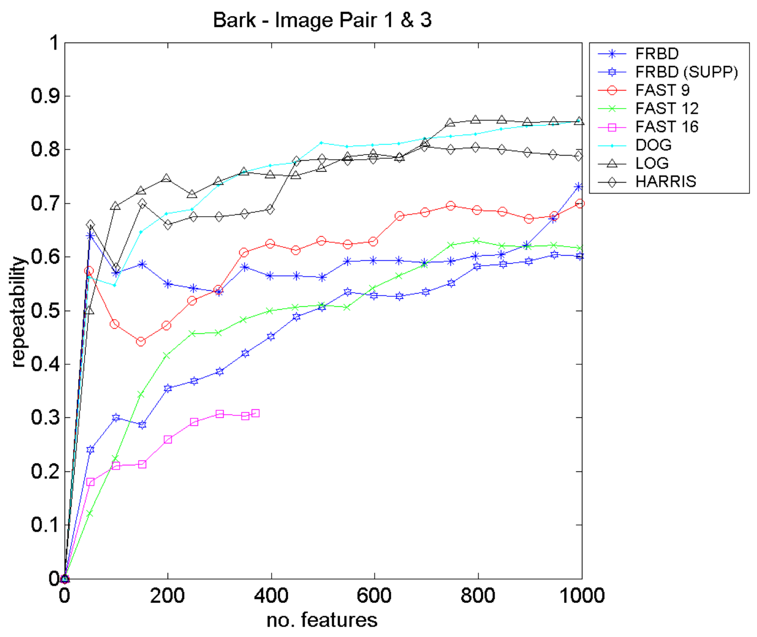


Fig. 6. Bark data set repeatability graph.

4 Application - Detection and Tracking of Multiple Visual Targets

We use the features $F(i), i \leftarrow [1..N]$ detected by the FRBD algorithm to find regions in the image that may belong to an object/target of interest. Before trying to track the features we perform a clustering stage using a minimum spanning tree [1] [2] to try and group features that may belong to a single target. We then use a bank of Probabilistic Data Association Filters (PDAFs) [1] to track, and classify these targets as either clutter or temporally salient. We define a temporally salient target to be one which receives measurements in most frames that can be explained by our prior motion model.

The PDAFs greedily use all of the measurements within their gate [2]. This means that a measurement can only be used once and cannot be assigned to more than one filter. After all of the PDAFs have completed an update any left over measurements are then used to instantiate a new PDAF. A simple check on the covariance of each filter is made at every iteration, so that any filters with high uncertainty can be terminated, freeing resources for other potential targets.

[1] Given a weighted undirected graph, a minimum spanning tree contains all of the vertices in the graph and the sum of the weights on its edges are minimal.

[2] Gate refers to an ellipsoid in state space, centered on the state estimate of the filter, with a size defined by a $n\sigma$ bound. For our system we use a 2σ bound.

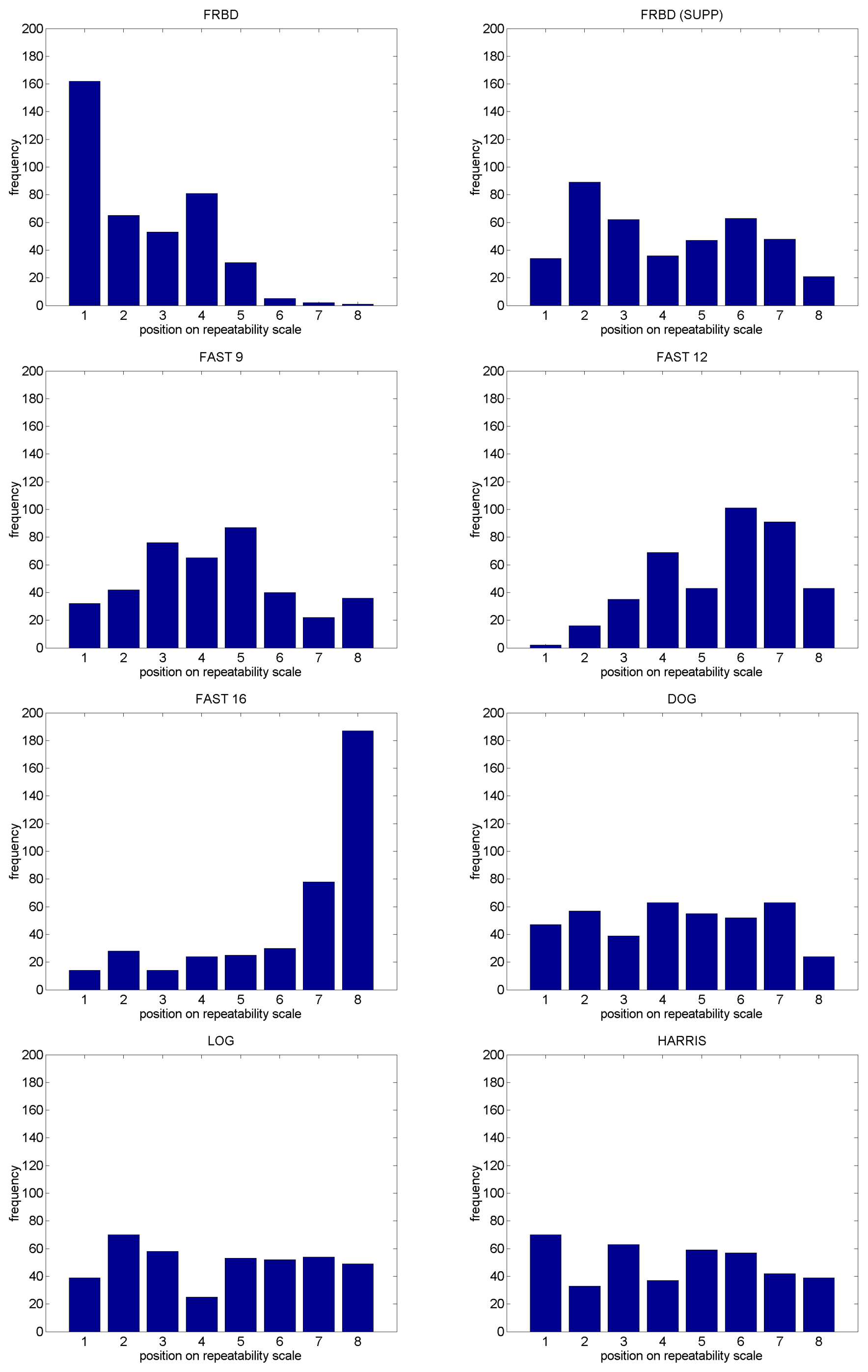


Fig. 7. Histograms of ranking table - for example the FRBD came 1st for 160 of the 400 data points and FAST_12 came 6th for 100 of the 400 data points.

4.1 Results

Figure 8, shows a typical sample frame and a movie strip from system in action, where the current tracked targets are highlighted and assigned a unique identifier. Note that in the movie strip the two initial targets maintain the same ID for the time they are in view. This is because the FRBD features are stable between frames and so the PDAFs corresponding to the two targets receive good measurements in most frames. Figure 9 shows the run-times for varying numbers of targets. The final algorithm - i.e. detection, clustering, and the PDAF filters - runs at about 80 frames per second on typical sequences but reduces to around 50 frames per second when the number of targets tracked reaches 50 .

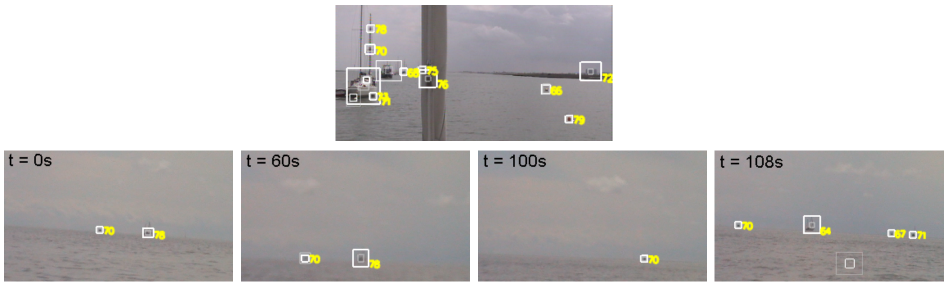


Fig. 8. (Top) Sample frame from system in action; (Bottom) movie strip of boats on the horizon.

These results show that for this particular application area, which is of significant interest to us, the FRBD features can be tracked stably over long periods of time. Although these initial results are good we need to carry out more analysis to evaluate their performance on general tracking problems. We are also interested in using the output of this multi-target tracking system to drive the focus of attention of an active camera system.

5 Application - Visual SLAM

We use Davison's framework for visual SLAM [3], which estimates in real-time both the position of the camera and the locations of current landmarks, using an Extended Kalman Filter framework. In order to achieve real-time performance, Davison used the idea of active search for feature matching; i.e he searches using SSD for a set of landmarks within projected uncertainty ellipses. The landmarks are chosen based on their visibility and uncertainty. Thus he adopts a top-down (model-driven) approach. In contrast, because we can compute the FRBD features so rapidly we can introduce a bottom-up element.

We have made three changes to Davison's original framework: (i) we carry out full frame feature detection using the FRBD and extract the N-best[3] features;

[3] For our testing we set $N = 80$.

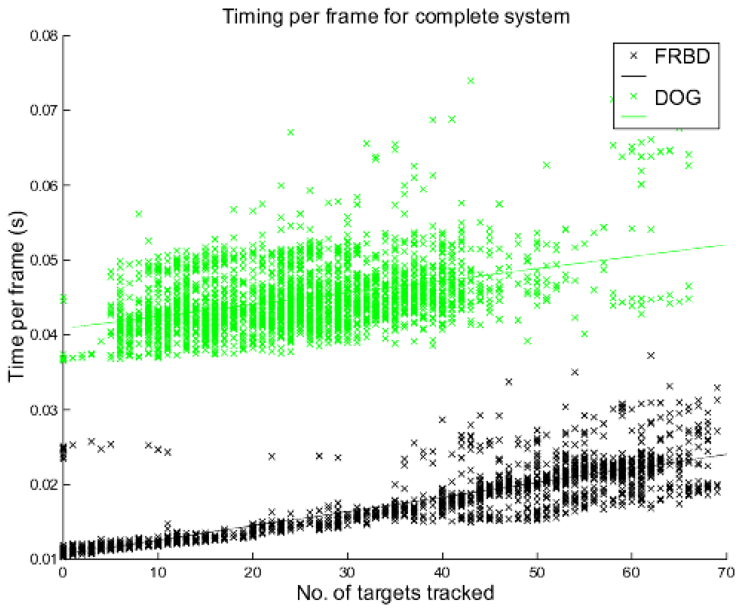


Fig. 9. Run-time against number of targets.

(ii) when a new landmark is required Davison's algorithm carries out an exhaustive search within a randomly chosen non-overlapping window[4], and selects the corner feature that has the highest response and is above a set threshold. In contrast, we pick the most salient from the N-best FRBD features that is above a set threshold and lies within the non-overlapping window; (iii) For each new frame Davison carries out his active search to find the locations that best match the selected landmarks. Rather than carrying out an exhaustive search, we use our N-best features to find locations that lie within the projected uncertainty ellipses and then pick the best matches using these locations.

The use of the FRBD has two main effects: (i) it adds a constant time per frame for the feature detection; (ii) it puts an upper bound on the computation required in each frame (for a given number of landmarks) that is significantly lower than the active search approach - this is because we only use the N-best feature locations for the potential detection and matching of landmarks as opposed to potentially using every pixel in the image.

5.1 Results

Figure 10 shows a typical timing sequence from a pre-recorded video sequence. The graph is broken into three sections for discussion:

[4] A window that does not overlap any existing landmarks.

– Section A - The camera is stationary for the first 100 frames after which it begins to move, and the approximately constant time process of detecting new landmarks begins.
– Section B - The camera is moving and more landmarks are required to localise the camera, which means that the time for the active search increases. The time for the FRBD also increases but by a smaller amount as it only uses locations that correspond to feature detections. Note that the dashed lines illustrate that in some frames the detection of new landmarks is carried out.
– Section C - Erratic camera motion occurs and so the uncertainty in the landmarks increases, which means that more pixel locations have to be examined by the active search. The dashed hand-fitted line shows that the active search begins to increase above the FRBD.

Figure 11 shows a timing sequence that corresponds to the situation where the SLAM algorithm gets temporarily lost. Again the graph is broken into sections for discussion:

– Section A - Comparable to section C in Figure 10.
– Section B - The estimation of the camera position is now wrong, which means that the uncertainty in all landmarks grows significantly. The dashed line shows how the active search now begins to peak and on occasions no longer operates in real-time. This is particularly interesting as the upper bound on the FRBD feature approach could easily be tight enough to guarantee real-time performance for a set number of landmarks.

In order to test the reliability of the FRBD features compared to the corner features normally used, we carried out 30 test runs for both. The search for new landmarks in both algorithms contains a random component, which means that no two runs are the same. By using a demanding video sequence, we look at the number of times the whole sequence is run and the camera finishes at the correct location and has successfully mapped the landmarks visible in the last frame. The original corner features had a success rate of 70%, whereas the FRBD features achieved 90%. This could be because the FRBD constrains the possible locations for matching a landmark to locations of detected features, which correspond to salient regions in the image, whereas the active search can potentially match a landmark anywhere in the image.

Our results demonstrate how the trade off in computation between full-frame feature detection and active search occurs, and how restricting landmark matching to feature locations appears to improve the robustness of SLAM. We are investigating other potential benefits of full-frame feature detection in SLAM, for instance by checking features against the database of landmarks to allow loop closing even when the state estimate has drifted and normal active search would no longer match the correct landmarks.

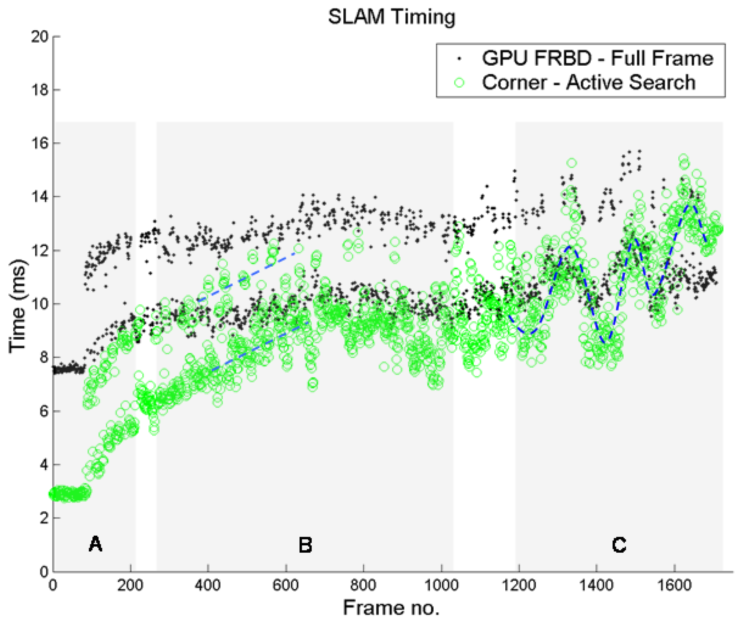


Fig. 10. Illustrates the trade off between full frame detection and active search.

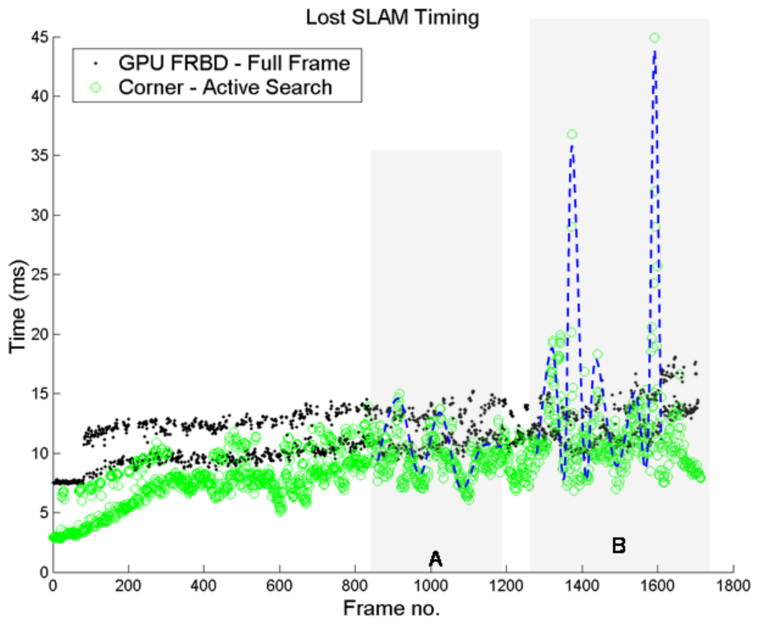


Fig. 11. Shows how full frame detection can ensure real-time performance.

6 Discussion

We have developed a feature detector, which works both *quickly* (640x480 video at 95 frames per second) and *robustly* (exhibits the highest repeatability on our data set). We have obtained promising results with a couple of applications: Detection and tracking of multiple visual targets, and a well known visual SLAM algorithm. We have also implemented Nister's preemptive RANSAC [7] and used the FRBD to replace the DOG detection stage, which makes a *significant* time saving of approxiamtely 9ms per frame. This leads to the potential for building a hybrid version of Davison's visual SLAM that uses Nister's preemptive RANSAC to provide visual odometry and improve the prediction stage between frames.

The FRBD also provides the opportunity to revisit the pros and cons of a top-down versus a bottom-up approach within visual SLAM, as to some extent it removes the great time divide between the two. For instance Davison's efficient active search method could be used when the uncertainty in landmarks is small, and a simple switch to full frame detection could be made when the uncertainty grows to the point that the benefits of active search disappear. The full frame detection could also help with the problems of loop closing, as the features provide sensible locations to look for familiar landmarks that may have already been mapped.

Although the repeatability experiments do give insight into performance, we have already highlighted a degree of skepticism in using this measure as an indicator of feature detector performance. It may tell you that a feature detector produces features in the same image locations but these may not be good for generating matchable features. Although Mikolajczyk [6] has looked at matching performance in more detail, we are interested in testing the actual performance of features in a real SLAM scenario, for instance by looking at average feature lifetime in a running SLAM system.

Acknowledgment

This work is carried out under the sponsorship of Servowatch Systems Ltd, who work in collaboration with the Royal National Lifeboat Institution.

References

1. Y. Bar-Shalom and T. Fortmann. *Tracking and data association.* Academic Press, 1988.
2. T. H. Cormen, C. E. Leiserson, and R. L.Rivest. *Introduction to Algorithms.* The MIT Electrical Engineering and Computer Science Series. MIT Press/McGraw Hill, 1990.
3. A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proc. 9th Int'l Conf. on Computer Vision, Nice*, 2003.
4. C. G. Harris and M. Stephens. A combined corner and edge detector. In *Proc. 4th Alvey Vision Conf., Manchester*, pages 147–151, 1988.

5. D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.
6. K. Mikolajczyk. *Detection of local features invariant to affines transformations*. PhD thesis, INPG, Grenoble, 2002.
7. D. Nister. Preemptive ransac for live structure and motion estimation. In *Proc. 9th Int'l Conf. on Computer Vision, Nice*, 2003.
8. E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *Proc. 10th Int'l Conf. on Computer Vision, Beijing*, volume 2, pages 1508–1515, 2005.
9. E. Rosten and T. Drummond. Machine learning for very high speed corner detection (to appear). In *Proc. 9th European Conf. on Computer Vision, Graz, Austria*, 2006.
10. J. Shi and C. Tomasi. Good features to track. *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 593–600, 1994.