Evaluation of Feature Detectors for KLT based Feature Tracking using the Odroid U3

Ben Barnes, Dinuka Abeywardena, Sarath Kodagoda and Gamini Dissanayake
Centre for Autonomous Systems, University of Technology, Sydney, Australia
Ben.Barnes@student.uts.edu.au, {Dinuka.Abeywardena, Sarath.Kodagoda, Gamini.Dissanayake}@uts.edu.au

Abstract

Feature tracking is an integral part of most vision-based state estimation frameworks. However, tracking features at a sufficient frame rate is a challenging task for mobile robots such as Micro Aerial Vehicles (MAVs) due to their fast dynamics and limited on-board computing resources. Recent developments in smartphone processors have led to embedded computing platforms that are ideal on-board computers for MAV state estimation. This paper analyses the performance of a Kanade-Lucas-Thomasi (KLT) based feature tracker on a state-of-the-art embedded computing platform suitable for on-board MAV state estimation. It compares the performance of different implementations of the feature tracker using four different low-complexity feature detectors. The experimental results presented herein may serve as guidelines for the selection of a feature detector, image resolution, framerate and feature quantity when developing on-board feature tracking systems based on ARM Cortex-A9 embedded computers.

1 Introduction

Feature tracking is an integral part of most vision based state estimation frameworks. It consists of three basic components: a feature detector, a feature tracker and a feature manager. The task of the feature detector is to identify salient points in each image that have the highest probability of being uniquely identified in subsequent images. The output of the detector is an array of points, each represented by a two-dimensional pixel coordinate. The tracker takes this array of points and attempts to determine the corresponding location of each feature in subsequent images. As the image sequence progresses, some of the feature points being tracked may be lost. This may be due to the feature moving out of the camera's field of view, or transforming such that the tracker cannot identify a correspondence with the previous frame. The feature manager addresses this by ensuring the number of features being tracked remains above a given threshold. It employs the feature detector to find new feature points, replacing those that have been lost. The feature manager is responsible for maintaining an even feature point distribution across the image, thereby also ensuring that each tracked feature is unique.

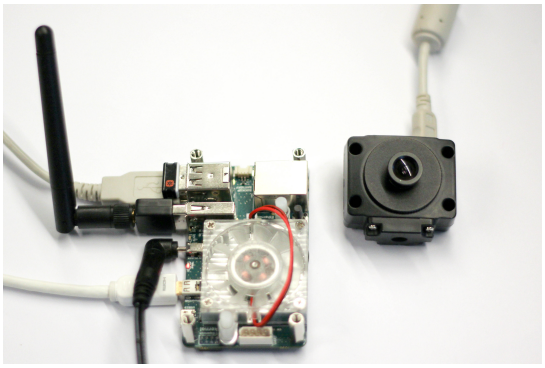

Figure 1: Experimental set-up showing the Odroid U3 and the Point Grey Firefly camera

There are two broad classes of feature tracking method on which the feature tracker may be based. The first approach uses the Kanade-Lucas-Thomasi (KLT) tracker [Lucas *et al.*, 1981], [Tomasi and Kanade, 1991] which uses spatial intensity information to search for a feature match. The second approach is to extract unique descriptors for features that can be matched against descriptors extracted from subsequent images. Given a robust descriptor such as SIFT [Lowe, 2004] or SURF [Bay *et al.*, 2006], this second approach can produce better tracking results. However, extracting and matching feature descriptors involves significantly more computational work. Therefore, KLT is the preferred method when computing resources are limited, such as in the case of an on-board state estimator for a Micro Aerial

Vehicle (MAV) [Hwangbo *et al.*, 2009], [Sanfourche *et al.*, 2013].

Whilst there are many feature detection algorithms available, only a handful of them are suitable for a resource constrained embedded platform. Prominent among these are Harris corners [Harris and Stephens, 1988], Good Features to Track (GFTT) [Shi and Tomasi, 1994], Features from Accelerated Segment Test (FAST) [Rosten and Drummond, 2006] and Adaptive and Generic Accelerated Segment Test features (AGAST) [Mair *et al.*, 2010]. There are also variants of these detectors such as scale-invariant AGAST (BRISK) [Leutenegger *et al.*, 2011] and Pyramidal FAST (ORB) [Rublee *et al.*, 2011].

The main objective of this paper is to evaluate the suitability of these feature detectors for a KLT based feature tracking algorithm. This feature tracker is a component module of a real-time state estimation framework for a quadrotor MAV based on the fusion of visual and inertial odometry. The visual-inertial fusion algorithm is expected to be designed in a structure-less manner without relying on estimating the 3D position of the environmental features. For this reason, feature trackers based on extracting and matching descriptors were deemed inappropriate as their computational requirements increase substantially when an estimated 3D position of the feature is not available.

Payload and power budgets limit the embedded computer platforms suitable for use on MAVs. A typical set-up for MAV state estimation and control would combine a simple microcontroller board with a more sophisticated on-board computer. The microcontroller board interfaces with the sensors and provides real-time control of the actuators, while the on-board computer handles the computationally intensive tasks such as visual information processing. Recent developments in multicore smartphone processors have resulted in embedded computing platforms with ideal characteristics for MAV on-board computers. From the available options, the Odroid U3[1] was chosen as the on-board computer for the quadrotor MAV state estimator, of which this work forms an integral part. Therefore, this paper focuses on analysing the performance of feature tracking implementations on-board the Odroid U3 with the aim of identifying the most suitable detector for a future MAV state estimator implementation.

This paper makes two contributions that have not been reported in the previous literature. First it evaluates the suitability of one of the new low-complexity feature detectors (AGAST) with three well established detectors (FAST, GFTT and Harris) for KLT based tracking. Though AGAST has been compared with FAST before [Canclini *et al.*, 2013], there the aim was to identify feature detectors that would facilitate descriptor based matching. Therefore the feature detectors were not evaluated for their suitability in KLT based tracking. KLT works best when the features have strong gradients in both image directions. While GFTT explicitly uses this metric to identify good features to track, detectors such as AGAST and FAST do not rely on image gradients to identify strong corners. Though this makes them more computationally efficient than GFTT, the resulting features may not be ideal for tracking with KLT. Therefore, FAST and AGAST must be compared with GFTT whilst using KLT for tracking to evaluate their suitability as detectors for the task at hand. To the best of our knowledge, this is the first time such an evaluation has been made.

As the second contribution, this paper analyses and reports on the performance of the Odroid U3 for the task of feature tracking. The Odroid U3 is selected as the experimental platform because it represents a class of recent light-weight embedded computer platforms based on the ARM Cortex-A9 architecture. We present details on how each feature detector performs on the Odroid U3 by making use of open-source implementations. We also present details on the performance of a comprehensive feature tracking implementation on-board the Odroid U3 and compare the results obtained using different detectors and image resolutions. This analysis provides valuable information about the suitability of the Odroid U3 and other similar embedded computing platforms for real-time computer vision-based implementations.

The remainder of this paper is organized as follows. Section 2 discusses the related work in feature detector evaluations. Section 3 details the methodology adopted here for the evaluation of feature detectors and section 4 details the experimental set-up and results. Section 5 concludes this paper along with a brief description of future work.

2 Related Work

There are several works in the literature which compare the performance of different feature detectors. [Gil *et al.*, 2010] analysed the repeatability of several feature detectors including Harris corners, SIFT and SURF. There, the requirement was to identify a suitable feature detector for descriptor based matching and tracking of features between images. They did not evaluate the computational requirements of each detector. A comparison of the computational requirements of six different feature detectors is detailed in [Nourani-Vatani *et al.*, 2012]. They evaluated a modification of the Canny edge detector, Harris corners, GFTT, SIFT, SURF and random points and concluded that the Canny edge detector performed the best. However, it should be noted that fea-

[1] http://hardkernel.com/main/main.php

tures on edges, such as those returned by the Canny edge detector, are not suitable for tracking due to the "aperture problem". They also reported both GFTT and Harris corners as having similar performance in terms of the processing time and the number of tracked features. More recently, an evaluation of low complexity feature detectors was presented in [Canclini *et al.*, 2013]. They compared the processing time required for FAST, AGAST, BRISK and ORB feature detectors and concluded that FAST and AGAST were the fastest followed by BRISK and ORB respectively. In terms of repeatability, they showed that both FAST and AGAST perform poorly when compared to BRISK and ORB. However, as is noted in [Nourani-Vatani *et al.*, 2012], the detectors with higher repeatability may not necessarily be suitable for feature tracking with KLT. For example, blob detectors such as SIFT and SURF might detect large image patches with near uniform intensity as features but such features cannot be reliably tracked with KLT. The accuracy and speed of FAST and GFTT when employed for KLT based tracking was evaluated in [Sanfourche *et al.*, 2013]. They concluded that FAST outperforms GFTT in both metrics.

3 Methodology

The test platform consisted of an Odroid U3 running an Odroid-specific release of the Xubuntu 13.10 operating system, available from Hardkernel[2]. The open computer vision library (OpenCV) version 2.4.9 was installed on the Odroid U3 and was used to provide all of the computer vision-specific functions in this experiment. Software was written in C++ to evaluate the performance of four feature extractors available through OpenCV: Harris, GFTT, FAST and AGAST.

OpenCV does not provide an isolated AGAST feature detector. However, OpenCV's BRISK implementation does use multi-scale AGAST for feature detection. It is possible to configure OpenCV's BRISK to detect features using only a single scale, thereby operating as an AGAST detector. However, it was not clear whether this would result in any additional overhead compared to a native AGAST feature detector. For this reason, we evaluated two different AGAST implementations for speed of execution. The first was OpenCV's implementation of BRISK, configured for single-scale extraction. The second was a standalone C++ AGAST implementation written by the authors of [Mair *et al.*, 2010].

For each feature detector, the OpenCV implementations include several parameters which can be adjusted to fine tune its performance. The values of parameters used in the experiments are listed in Table 1. The pyramidal KLT implementation of OpenCV was used to track the detected features between images; the KLT parameters are listed in Table 2.

[2]http://goo.gl/gGqE85

Table 2: Parameters used for KLT

Parameter	Value
winSize	31×31
maxLevel	3
criteria	count(30) or epsilon(0.01)
flags	OPTFLOW_LK_GET_MIN_EIGENVALS
minEigThreshold	0.08

OpenCV provides functions for time measurement. These were used for all extractor time profiling. Two functions are provided by OpenCV, *getTickCount* and *getTickFrequency*. The function *getTickCount* returns the number of fixed-period 'ticks' since some epoch, such as system boot. The function *getTickFrequency* returns the number of ticks per second. By taking two samples with *getTickCount*, subtracting the first sample from the second, and dividing by the result of *getTickFrequency*, the time in seconds between two events can be determined.

Two image sequences were used for the experiments. The first sequence was gathered using a Point Grey Firefly MV FMVU-13S2C camera connected to the Odroid U3 via a USB interface (see Figure 1). This camera was selected for its external shutter triggering capability, which may be used in later experiments to synchronise inertial measurements with video frames. Point Grey provide an ARM Linux package consisting of libraries, a configuration utility and a C++ API for this camera called FlyCapture2 which was installed on the Odroid U3 platform. The images from this camera were gathered at a rate of 30Hz whilst the camera was moved by hand in an office-sized environment. These images had a resolution of 640×480 pixels. Figure 2 presents a sample image from this sequence.

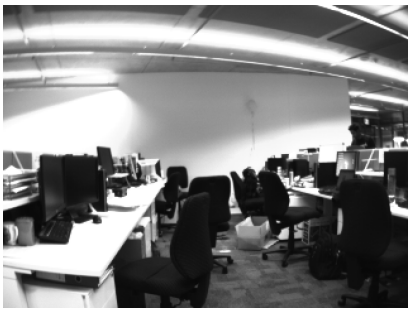

Figure 2: An image from the Firefly sequence.

At the time of data collection, the Point Grey camera was not attached to a quadrotor MAV and as such

Table 1: Parameters used for Feature Detectors

GFTT	**Harris**	**FAST**	**AGAST**
qualityLevel = 0.01	qualityLevel = 0.01	nonmaxSuppression = 1	octaves = 0
minDistance = 3	minDistance = 3	type = TYPE_9_16	patternScale = 1.0
blockSize = 3	blockSize = 3		
useHarrisDetector = 0	useHarrisDetector = 1		
	k = 0.04		

that image set did not reflect the true characteristics of a sequence gathered from an on-board camera. For this reason, a second data set gathered from the front camera of the ARDrone 1 quadrotor MAV [3] was also used for the experiments. These images had a resolution of 320×240 pixels. They were collected approximately at 10Hz whilst the AR Drone was flying aggressively in a small room-sized environment. Figure 3 presents a sample image from this sequence.

Figure 3: An image from the ARDrone sequence.

We performed three different tests to evaluate the suitability of the four feature detectors for tracking. The three tests were designed with the following objectives in mind:

- To determine the execution speeds of the various feature detectors
- To determine the suitability of each detector for KLT based tracking
- To evaluate the overall performance of a feature tracking algorithm based on each detector

All experiments for these three tests were conducted on-board the Odroid U3. All code was compiled using GCC/G++ with Thread Building Blocks (TBB) and ARM Neon support enabled. Both TBB and Neon support were essential in reducing the computation times to those reported in the next section. The source code for these experiments has been made available online[4] under the GNU GPL v3 license. A video of the second and third tests is also available[5].

[3] www.ardrone-flyers.com/ar-drone-specs.html

[4] https://github.com/benwbarnes/ACRA-2014

4 Experiments

4.1 Feature Detection Speed Test

The aim of the first test was to evaluate the execution speed of each detector. As this test required extracting a large number of feature points, an image of a stone wall with a high amount of texture was used as a target (see Fig. 4). To run this test, this image was first loaded into memory. A desired number of features was taken as a parameter, with each feature detector in turn being asked to provide this number of features. The time taken for detection was recorded for different numbers of features. This information may be used in subsequent projects to determine the number of features a given detector should attempt to detect, given the frame rate at which the images need to be processed.

Figure 4: Stone wall image used for feature detection speed test.

Whilst the Harris and GFTT functions provided by OpenCV take a maximum point count as a parameter, FAST and AGAST do not. Rather, these extractors are controlled through the use of a 'threshold' against which points are compared, with those having a quality measure less than the threshold being discarded. Thus, the number of points identified by these extractors can be controlled by adjusting the threshold. To determine

[5] https://www.youtube.com/watch?v=sVSzd-fIPCo

a best-case time for feature extraction with these algorithms, the threshold was set at an initially low value and incrementally increased until the number of detected features reached the target level. For FAST and the standalone AGAST implementation, it was possible to adjust these parameters rapidly enough so that recording 200 points of data was feasible (1-200 features in steps of 1 feature). However, the AGAST implementation provided by OpenCV as part of BRISK does not allow dynamic modification of the threshold; this parameter is set at object creation. With object creation taking approximately 500 ms, the brute-force method for finding an appropriate threshold was not feasible for 200 data points. Instead, data was recorded in steps of 10 features with the threshold being adjusted manually until the number of features reached the desired level.

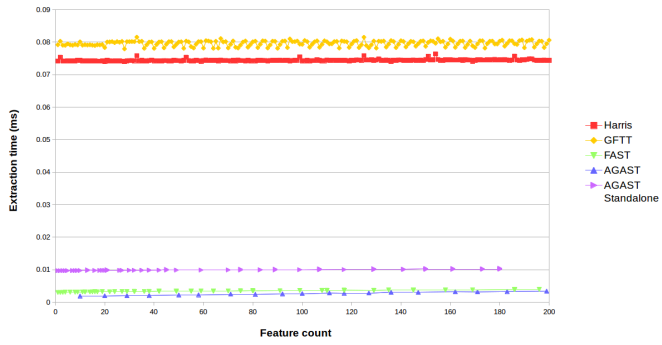

Figure 5: Feature detection speeds for all detectors.

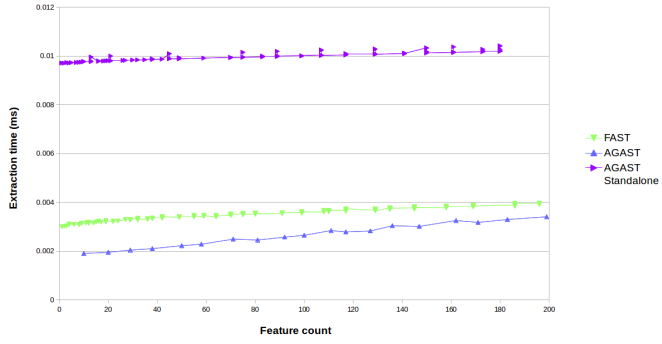

Figure 6: Feature detection speeds for FAST and AGAST detectors.

The feature detection times for each of the detectors are illustrated in Fig. 5. It can be seen that Harris and GFTT behave similarly to each other in terms of extraction time, as do FAST and AGAST to each other. Broadly speaking, the AST-based detectors outperform Harris and GFTT by a factor of approximately 10-20. The enlarged graph of the AST-based detectors, illustrated in Fig. 6, shows that the OpenCV implementation of AGAST (as provided by BRISK) is the fastest of the set. It is interesting to note that it is approximately 5 times faster than the non-OpenCV implementation. FAST closely approaches the speed of AGAST, in this case operating around 1.5 times slower.

4.2 Feature Lifetime Test

The aim of the second test was to determine the suitability of each detector for KLT based tracking. This test was performed using both the ARDrone and Firefly image sequences described above. First, an initial set of features was extracted from the first frame of the video sequences. Then KLT was used to track the features between frames, with any features that could not be tracked being removed. This was repeated until no features remained or the video sequence terminated. The rate of feature loss was then used as a comparison between extractor algorithms. There are three reasons why a given feature would be removed from the tracker: a feature moving beyond the camera's field of view, feature occlusion, or the feature not being robust enough for tracking. As the same image sequence was used with all four detectors, the first two conditions remained the same. Therefore, this method provides an insight into the quality of the features provided by each detector in terms of KLT based tracking.

To balance between tracking accuracy and feature loss, the KLT error threshold was adjusted manually using a visual output of the features being tracked. An initial set of 50 features for the Firefly, and 30 features for the ARDrone was extracted. For the Firefly and ARDrone tests a KLT minimum eigenvalue threshold of 0.08 was used.

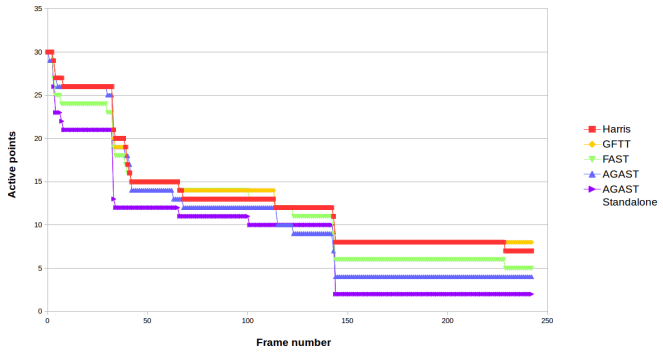

Figure 7: Feature lifetime results for the ARDrone sequence.

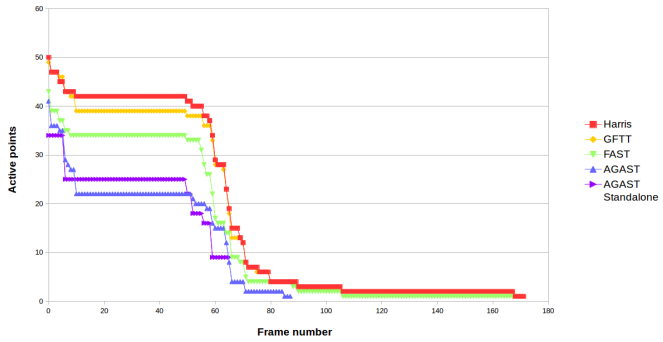

Figure 8: Feature lifetime results for the Firefly sequence.

The results of this test for the ARDrone and Firefly sequences are presented in Figures 7 and 8, respectively. They demonstrate that there are no significant differences in the performance of features extracted by different detectors. The Firefly data shows some separation from frames 10-50; this is largely due to the FAST extractor finding less initial features with a quality greater than its given threshold. Overall, this test demonstrates that given a single image from which to draw features, there appears to be no compelling reason to select one extractor over another based on feature quality alone.

4.3 Point Management Test

The third test combined the detection and tracking with a simple feature management method to evaluate the performance of feature tracking as a whole. The aim of this test was to gain a better understanding of the capabilities of the Odroid U3 in detecting and tracking features. For this, a simple feature management routine was included. This routine maintained a fixed number of features by replacing any features lost during tracking. To prevent overlapping features from being generated, a mask was used which defined an exclusion zone around existing features. This also ensured an even spread of features over the image. This test was designed to simulate a real-world application of KLT optical flow, and allowed for a realistic assessment of the Odroid U3's capabilities when using the various extractors.

This test provided data about both extractor speed and feature quality and was performed using both the ARDrone and Firefly image sequences described above. The resolution of the Firefly sequence is more than twice that of the ARDrone sequence. This resulted in different computation times for the two sequences. The results of this test for the two image sequences are presented in Figures 9 - 16. Note that results for the standalone version of AGAST are not presented as its speed and tracking performance were comparable to the OpenCV version.

In these figures, the tracking time is the time taken by KLT to track the existing feature points. The management time is the time taken by the point management module to detect new features to compensate for those lost during tracking. The total loop time is the total time required to process each image and is the combination of the tracking and management times. Also, points tracked is the number of points successfully tracked in each image and points added is the number of new features added to the tracker by the point management module.

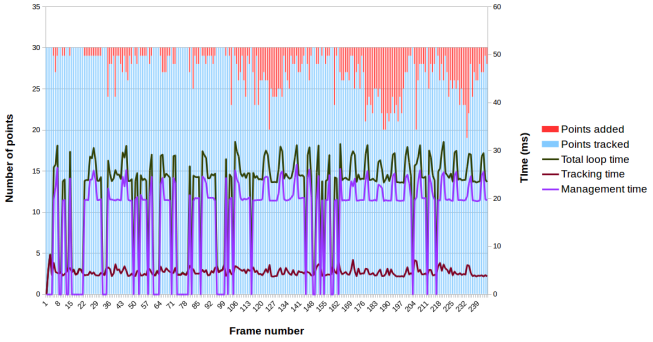

Figure 9: Point management results for GFTT with the ARDrone sequence.

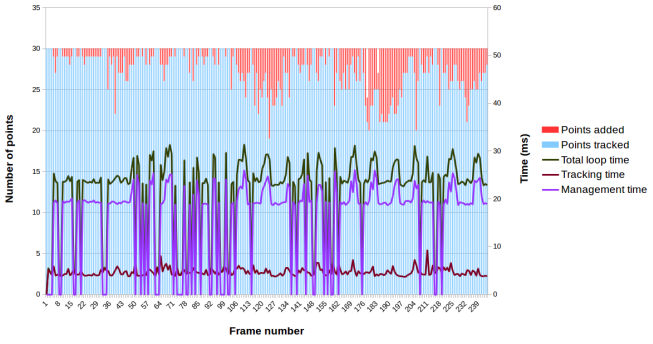

Figure 10: Point management results for Harris with the ARDrone sequence.

For the Harris and GFTT extractors, these results appear to agree with expectations from the first two experiments. Feature loss is moderate and reasonably consistent over the course of the test. Processing time is essentially constant regardless of the number of new features required, unless zero features are required in which case the management time drops almost to zero. Tracking time remains roughly constant throughout.

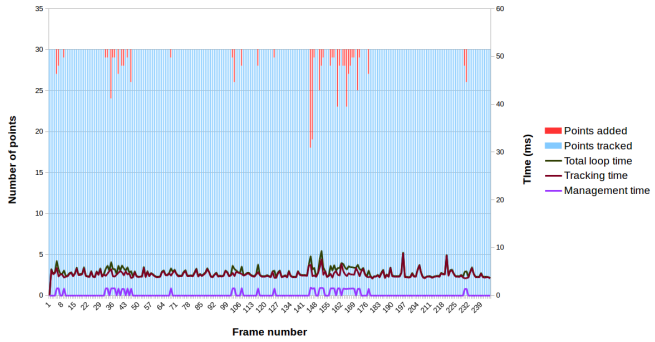

Figure 11: Point management results for FAST with the ARDrone sequence.

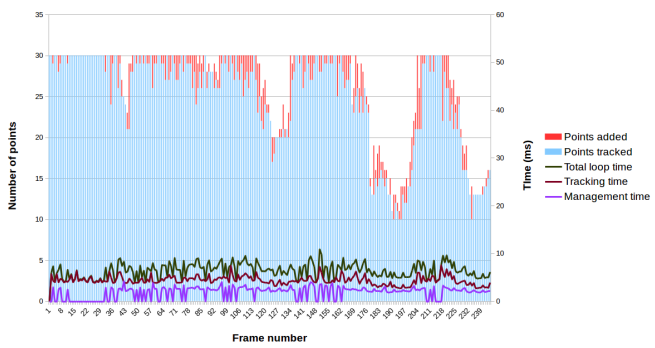

Figure 12: Point management results for AGAST with the ARDrone sequence.

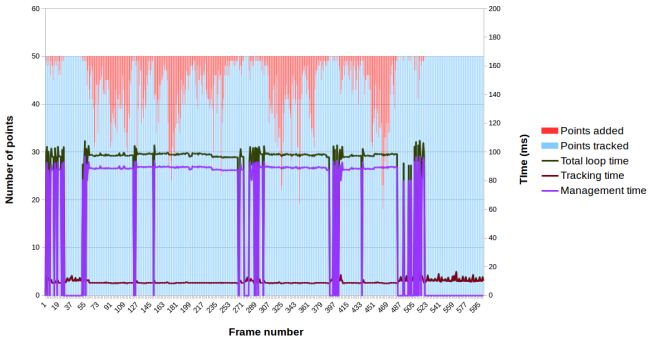

Figure 13: Point management results for GFTT with the Firefly sequence.

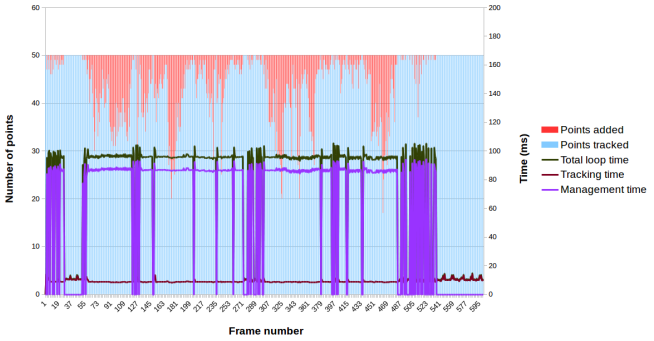

Figure 14: Point management results for Harris with the Firefly sequence.

Compared to Harris and GFTT, FAST loses far less features per frame on average. Additionally, the processing time is much lower, as well as being more consistent whether or not additional features are required. This appears to conflict with the feature lifetime tests, which indicated no significant difference in the rate of feature loss for the various extractors. However, this is not an accurate comparison as the feature lifetime tests did not consider average performance over a sequence of frames. Whilst the performance of the features extracted from a specific source frame may have been similar, over a longer timescale the differences in extractor performance become apparent. This manifests as a greater average rate of feature loss with features extracted using Harris and GFTT than FAST.

The OpenCV implementation of AGAST offers arguably the least favourable performance, with significant feature loss on average and a processing time slightly exceeding that of FAST. The ARDrone test also shows that AGAST was unable to maintain the full complement of 30 features, the only extractor for which this was the case. However, on average AGAST can operate three times faster than both GFTT and Harris, and it maybe possible to configure it to detect more feature points during each step. In this case, such an adjustment was not made in order to maintain the same settings for all detectors.

Comparing the execution times, it can be seen that both AGAST and FAST based feature trackers can operate at 100Hz with the ARDrone sequence. For the Firefly sequence, this reduces to 50Hz for FAST and 20Hz for AGAST. Also note that the times taken for point management and feature tracking are similar both for FAST and AGAST. For GFTT and Harris, the time taken for point management far exceeds that for tracking.

5 Conclusion and Future Work

The results presented in the previous section demonstrate that all four detectors are able to detect features that can be tracked reasonably well with KLT. However, in terms of tracking speed, the implementations based on FAST and AGAST are clearly superior to those based on

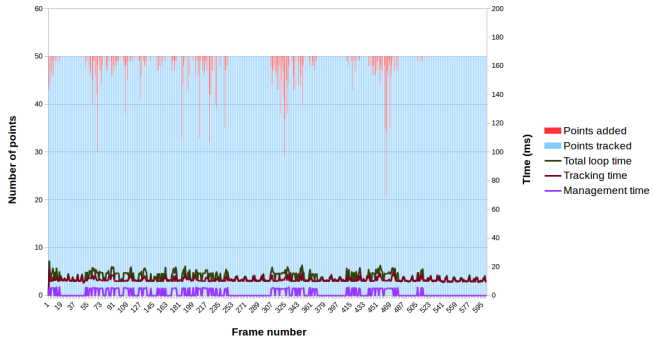

Figure 15: Point management results for FAST with the Firefly sequence.

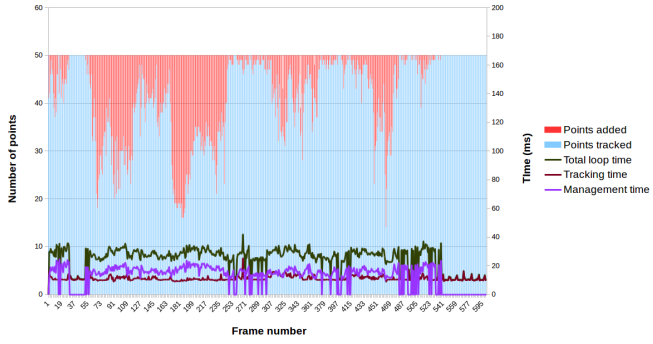

Figure 16: Point management results for AGAST with the Firefly sequence.

GFTT and Harris. Comparing FAST against AGAST, we see that the former results in faster and more robust tracking than the latter. Therefore, it can be concluded that FAST is the most suitable feature detector of the four evaluated, for the task of KLT based feature tracking.

The results of the point management experiments provided a clear insight into the performance of the Odroid U3 in feature tracking task. It was demonstrated that the Odroid U3 was capable of tracking an average of 50 environmental features at 50Hz on images with a resolution of 640×480. This far exceeds the real-time performance requirements for a camera with a frame rate of 30Hz. However, as the Point Grey Firefly camera is capable of capturing images at a maximum frequency of 60Hz, it is desirable to further improve the performance of the feature tracking implementation on-board the Odroid U3.

Our future work focuses on two main tasks. First, we aim to further decrease the feature tracking and management time of the FAST-based implementation by leveraging the multiple processing cores available on the Odroid U3. Second, we aim to use the visual information provided by the feature tracker in a real-time on-board state estimator for a quadrotor MAV. We believe these to be exciting research areas, leading to a successful field deployment of quadrotor MAVs.

References

[Bay *et al.*, 2006] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006*, pages 404–417. Springer, 2006.

[Canclini *et al.*, 2013] A Canclini, M Cesana, A Redondi, M Tagliasacchi, J Ascenso, and R Cilla. Evaluation of low-complexity visual feature detectors and descriptors. In *Digital Signal Processing (DSP), 2013 18th International Conference on*, pages 1–7. IEEE, 2013.

[Gil *et al.*, 2010] Arturo Gil, Oscar Martinez Mozos, Monica Ballesta, and Oscar Reinoso. A comparative evaluation of interest point detectors and local descriptors for visual slam. *Machine Vision and Applications*, 21(6):905–920, 2010.

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

[Hwangbo *et al.*, 2009] Myung Hwangbo, Jun-Sik Kim, and Takeo Kanade. Inertial-aided klt feature tracking for a moving camera. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 1909–1916. IEEE, 2009.

[Leutenegger *et al.*, 2011] Stefan Leutenegger, Margarita Chli, and Roland Yves Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.

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

[Lucas *et al.*, 1981] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[Mair *et al.*, 2010] Elmar Mair, Gregory D Hager, Darius Burschka, Michael Suppa, and Gerhard Hirzinger. Adaptive and generic corner detection based on the accelerated segment test. In *Computer Vision–ECCV 2010*, pages 183–196. Springer, 2010.

[Nourani-Vatani *et al.*, 2012] Navid Nourani-Vatani, P Borges, and J Roberts. A study of feature extraction

algorithms for optical flow tracking. In *Australasian Conference on Robotics and Automation*, 2012.

[Rosten and Drummond, 2006] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, pages 430–443, May 2006.

[Rublee *et al.*, 2011] 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.

[Sanfourche *et al.*, 2013] Martial Sanfourche, Vincent Vittori, and Guy Le Besnerais. evo: a realtime embedded stereo odometry for mav applications. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 2107–2114. IEEE, 2013.

[Shi and Tomasi, 1994] Jianbo Shi and Carlo Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[Tomasi and Kanade, 1991] Carlo Tomasi and Takeo Kanade. *Detection and tracking of point features*. School of Computer Science, Carnegie Mellon Univ. Pittsburgh, 1991.