Lightweight image processing toolpack for low-power and low-cost optical SST triangulation stations for cataloguing in LEO regime

Konrad Bojar
KB-Innotech, ul. Bukowiecka 92, 03-893 Warsaw, Poland

ABSTRACT

In this paper we provide a thorough analysis of a lightweight image processing toolpack which serves as an SST station on-board imaging data processing block designed with strong power budget limitations. We start the statement of the problem and with a description of the main design principles of the toolpack, especially requirements on the power budget and the data bandwidth; it becomes apparent that these requirements are in fact the main force driving the design at the single observation station level. Once the hardware is decided, the software design can be performed; the interface-driven breakdown structure is obtained in this first step. The breakdown structure comprises a background extractor, a star extractor, and a streak extractor; the latter is the most computationally exhaustive. In order to get the streak extractor module fast enough while still performing well in terms of sensitivity and number of false positives per image, we give up methods based on patterns, working in transformed domains or employing model-blind generic search methods. Instead, we use a collective method which is a mix of the well-known concept of the line detector based on Markov random fields and voting methods already known to be useful for streak detection. Our approach is highly optimized and to that end we follow concepts based on the iterated conditional modes in the multiresolution pyramid framework, all that with numerous reductions and performance improvement tricks, like lookup tables or moving the problem to fixed-point domain. We describe our test setup in terms of the hardware and software, test methodology, and some results to indicate to what extent our method proves efficient for the assumed use case.

1. INTRODUCTION

The commercial space surveillance and tracking (SST) market is still considered to be in its infancy stage, but there are already some rapid technology development trends visible. A recent market research [1] reveals that currently 80% of commercial SST revenues are generated by governments, and this number will fall to 60% until 2035. Moreover, in 2035 73% of the revenue will be generated by services targeted at the low Earth orbit (LEO) while currently it is only 46%. It follows then that for the coming decade the governmental market is of key interest. In this paper we focus on the European framework, namely the EU SST Partnership framework. This Partnership has clear objectives, defined in the regulation 2021/696 [2], and the recent implementing decision 2022/1245 [3]. One of the main performance criteria considered in the said documents is related to the cataloguing capability, and this has already been widely discussed in [4] and in greater detail in [5]. In order to obtain acceptable results, two key performance indicators (KPIs) have been defined: coverage and fraction of well-observed objects for each and every orbital regime, in particular for LEO; this definition is different for radars and for optical sensors. The first KPI refers to coverage of the object population by the observation network. The object is counted in when it is observed at least once in the predefined period. For the second KPI, a LEO object is considered well-observed when it is seen more than once in 24 hours. This will appear to be a deciding criterion just in a while. There is yet another system-wide KPI, the timeliness, measuring how fast the system is able to process the data: any data delivered long after the act of actual measurement is useless. In purely operational terms this means that we cannot stack results without delivering them to the database. From the point of view of a single SST station, the available processing time is orders of magnitude shorter than usual timeliness values (a fraction of a second versus 24 hours). In order to obtain the desired added value in terms of the mentioned KPIs there are several possible solutions. In this paper we are going to discuss the option of a dense sensor network consisting of numerous identical, autonomous nodes in every very large area (VLA). A single triangulation measurement generated by this network requires that the same object is seen by at least two stations at the same time. This turns the dense network even denser and it means that many stations should deployed in areas with no infrastructure whatsoever, like small islands (Pacific VLA), deserts (Oceania VLA), or desolate mountain peaks (South America VLA). The concept of triangulation networks is already being put to the test under the OmniSkyNET [6] and Optical Fence [7, Ch. 4] projects, but high density of the network implies even lower resources available at the station location. In order to make another step forward, it is necessary to impose stringent requirements on power consumption and data bandwidth for the station design, like it

was done in [8] (some aspects of the power consumption problem have also noticed in [9], but resulted from COTS-oriented design principles). In this paper we will discuss one of enabling image processing technologies for low-power and low-bandwidth SST stations for cataloguing in LEO regime. Our discussion will be developed from the point of view of the software, but in a hardware-aware manner, retaining knowledge on what our platform is capable of and what is particularly good at. We will discuss a fixed optical setup, and therefore our sensors will be of very wide field of view type. Naturally, such a setup is in practice limiting the magnitude of objects available for observations, irrespective of which commercial off-the-shelf (COTS) sensor we use; this effect has already been taken into account at the trade-off analysis stage of the architecture study.

In Section 2 we describe the design of the single station, justifying hardware decisions and software design principles. In Section 3 we outline the design of the image processing toolpack, focusing on principles of its functionality, the data pipeline structure and necessary internal interfaces between blocks. In Section 4 we discuss the workbench used for testing, implementation of the image processing toolpack, and obtained results. We discuss main parameters and factors affecting performance of the system. In Section 5 we draw conclusions and indicate future work directions.

2. SINGLE STATION DESIGN

A preliminary version of the observation network is already operational, hence we do not need to redo detailed design of the single station from scratch. All necessary details, together with a review of competing solutions, can be found in [6]. In simplest words, the station consists of a power subsystem, a network subsystems, a watchdog, a computing subsystem, and an imaging subsystem (containing a synchronizing component providing global clock across the network), clearly following the paradigm of edge computing. The target version of the system shall not rely on external infrastructure, and this includes the mains power. Therefore we adopt a requirement that the computing and imaging subsystems, providing station's core functionality, should not require more than 70W of continuous power in the steady state. This requirement does not cover events like the cold start or some other rare events which may occur in the system, because they do not pose a practical problem and can be easily dealt with. The adopted value of 70W allows the station to utilize solar panels of reasonable size. When it comes to the bandwidth requirement, the expected throughput is on the order of one gigabyte of tracklet data per night or on the order of 100MB/h of sustained throughput during the night of 12 hours; again, no rare events like calibration counted in, just the data required by the central node to compute matches and generate the tracking data messages (TDMs). This means that the 256kbps link is sufficient for this solution and it poses no problem nowadays to have a satellite uplink to handle this stream. We will not delve into data size optimization any further in this paper.

Once the power budget decision has been made, it is the time to perform a quick trade-off analysis to choose the hardware platform. With no a priori decisions on the processing hardware platform architecture, we can choose from platforms driven by the x86 architecture, the ARM-Cortex architecture, or one of existing FPGA-based architectures, possibly in a system-on-chip setup. All FPGA-based architectures are ruled out already in the first step because they require engineering of a tailored software and this is too much of an effort for our solution (and too long in terms of time to market). Moreover, any FPGA-CPU tandem, be it a hard-core version, like Xilinx Zynq, or a soft-core version, like Xilinx Microblaze, is ruled out for the same reason. The ARM-Cortex architecture is a good competitor for the x86 architecture, especially the Cortex-M 64-bit architecture. Practically all OTS software packages we are going to use have their versions optimized for Cortex-M, like standard mathematical packages (Neon), image processing toolpacks (OpenCV), and camera driver toolpacks (FLIR Spinnaker). However, due to the fact that we are using a substantial amount of in-house legacy code, we opt for the x86 architecture. Given the power constraint of 70W for the cameras and the platform, it follows clearly that there is no power budget left for any currently available graphics processing unit (GPU).

We have decided that the x86 platform should be controlled by a Linux operating system for full control over processes and scheduling, including forced preemption, and process CPU affinity control. This control is easily exerted by our user-level software using the /proc and the /sys subsystems and signals. All used OTS libraries are simple utilities, for example communication libraries, FITS format generators, or camera APIs. On the top of that software stack we build our solution. Before we pass to that stage we should decide what should be the driving force of our design. We follow the top-down approach, dividing the system breakdown structure until we arrive at blocks simple enough. At the design stage we could not be sure about final performance of any block on the platform of choice. Therefore, the design should allow to exchange any blocks without reworking the rest of the processing

pipeline. It follows that the modular, interface-driven design was one of the best options and we pursued best practices taken from the ECSS-E-ST-10-24C standard which formalizes this type of design. Once we strip off the usual auxiliary piece of the software and performed CPU load analysis, it becomes apparent that the image processing toolpack consumes over 80% of the available CPU capacity. Detailed design guidelines for this toolpack are given in the next section, and actual implementation details can be found in Section 4.

3. DESIGN OF THE IMAGE PROCESSING TOOLPACK

The main goal of the data processing toolpack is to reduce the data incoming from the cameras (~600MB/s) to a stream of on the order of no more than 50kB/s per camera. In practical terms this means that for every image we should expect no more than several streaks detected, including false positives. Keeping that in mind we identify three main functional blocks of the toolpack: the background modeler, the star extractor, and the streak detector; these blocks are mentioned in the data flow order. There are also smaller auxiliary blocks, but they do not bring much new into our discussion at this level of detail. It should be also noted that there is no need to perform typical astronomical preprocessing and calibrations on the observation station, like dynamic flat field correction, fixed-pattern noise removal, dark frame subtraction, pixel cross-talk compensation, lens aberration correction, etc. The only effects that matter are vignetting and sunset/sunrise effect, and here we assume that this effect is corrected straightforwardly after frame acquisition. All geometric parasitic effects can be compensated for in the network central node using the astrometric calibration performed at the station, and no mentioned intensity-related effects disturb our algorithms.

The three main functional blocks have the following roles assigned:

1. The background modeler module serves as the region of interest (ROI) extractor and is meant to remove big irrelevant objects, mainly clouds, and, occasionally, tree branches, poles and alike structures, if the location was good enough for other reasons to accept the trade-off of having them in the field of view. Non-saturated stars are unlikely to be removed due to their small size. A lower resolution image generated here is small enough to be sent several times a minute to the central node as a preview for visual checks.
2. The role of the star extractor module is twofold. On the one hand it provides the data for the astrometry block which is used for calibration. On the other hand it removes stars from the ROI. It is important to have stars removed because the streak extraction which comes next is, roughly speaking, based on a concept of a distributed evidence and the higher the intensity of a pixel, the stronger the evidence.
3. The streak extractor module provides the core functionality of the station. It extracts candidates for tracklets and thus reduces the data to the target 50kB/s per camera. This does not mean that every frame complies with this constraint, but it is satisfied on the average. Final removal of false positives is very effectively performed at the central node of the network by timestamp matching and geometric consistency checks [10]. Therefore we care for false positives only in terms of maximal data bandwidth, not in usual terms of false positives per image (FPI).

Before we pass to definitions of interfaces and detailed architectural considerations, it is a good place to briefly discuss how our approach is related to already known solutions of the problems of streak detection and extraction. A noticeable group of streak detection algorithms uses modelless methods like convolutional neural networks [11]; such algorithms are out of scope of our approach, although very useful for problems where little or no knowledge on structure of the problem is implemented into the algorithm. Another group of algorithms is based on pattern matching [12, 13]; these algorithms are too slow for us. The next group is based on image transforms like the Radon transform [14, 15]; we are not going to use methods working in any transformed domain due to problems with short streaks and high computational complexity. Some streak detection methods are built using assumptions on the streak model, assumptions reflecting physical properties of image acquisition hardware and exposure conditions. It is common to assume that every streak is a straight line segment convolved with the Gaussian point spread function (PSF). This form is especially useful for analytic streak detection methods, for example the maximum likelihood (ML) method outlined in [16]. The next assumption is related to the exposure conditions. When relatively long exposures are assumed, long streaks are obtained without any need for power-consuming synthetic tracking techniques [17]. We adopt both of the abovementioned assumptions in our solution. Our method is a mixture of a line detector in the multiresolution Markov random field (MRF) framework and a voting scheme inspired by the streak detection method based on the Hough transform [8]. Since our design is based on interfaces and is architecture aware, we need to pinpoint interface features which will reflect algorithm-specific needs and address the CPU and memory architecture at the same time.

The functional blocks enumerated above have the following interfaces and architecture-related features:

1. The background modeler block
   Input: vignetting- and sun-uncorrected camera image
   Output: multiresolution binary ROI mask, 8 bits per pixel (due to memory architecture), 3x3-redundant
   Architectural considerations:
   a) All constituents, together with the vignetting and sunset/sunrise compensation, are fixed-point manipulations on pixels or their 2x2 neighborhoods. In general, the ability to process this data as a coherent stream, with no random memory access, is favored over implementations with several streams. Hence, it is useful to prepare an auxiliary redundant structure where pixels are remembered together with their neighborhoods. At later steps the appropriate grouping is 3x3, yielding 3x3-redundancy, hence the output is 3x3 redundant already here.
   b) The ROI mask must be updated faster than the environment changes its appearance, but some less involved processing can be performed every frame.
   c) This block operates on a multiresolution pyramid [18] which is used extensively later on. In this way we effectively filter out the Gaussian PSF effect. The first-choice method in presence of the Gaussian PSF – the Wiener deconvolution filter – performs poorly here and we are not going to deconvolve the input image at any stage.
2. The star extractor block
   Input: vignetting- and sun-corrected camera image
   Output: multiresolution binary ROI mask, 8 bits per pixel (due to memory architecture), 3x3-redundant
   Architectural considerations:
   a) This functionality could be a priori realized in at least two distinct ways.
      i. The first option to choose from is to store the catalog of objects in the form of right ascension, declination, spectral data, and shape data for non-stars, and compute locations of those objects in the image coordinate system. The next step is to prepare the ROI mask on the fly using the precalibrated optical axis data, the time stamp, and the geometric distortion model. This solution requires recalculation of a part of the catalog at every generation of the ROI mask.
      ii. The second option to choose from is to store the precomputed, time-tagged ROI masks in the internal database and to pick the correct one every frame. The latter option requires storage for almost 200 million ROI masks (per year, for six cameras, one mask per one second) and this is way too much, even if an efficient compression scheme is applied.
   b) We propose a trade-off allowing us to avoid floating-point computations but at the price of storage. We store precomputed locations of objects, but we prepare the ROI mask on the fly using multiresolution templates for each object type. It should be pointed out that we do not perform detection of stars at the cost of larger templates and more elaborate templates for saturated stars. In the next section we will see that the magnitude limit of our system makes this approach efficient.
3. The streak detection block
   Input: vignetting- and sun-corrected camera image, both 3x3-redundant multiresolution binary ROI masks
   Output: tracklet list
   Architectural considerations and further breakdown structure:
   a) This functionality is a cascade of three algorithms, each with its input and output interface requirements. We will present this decomposition, describing the inputs and the outputs along the cascade.
   b) The first algorithm is the multiresolution MRF line detector inspired by [19]; clique definitions are derived from pixel 8-adjacency, hence 3x3-redundancy is the most desired data structure. Although it is typical to evaluate the posterior distribution of the MRF with the simulated annealing, for example by means of the Gibbs sampler, we do not follow this path for computational complexity reasons. We apply the iterated conditional modes concepts (ICM) which corresponds to instantaneous freezing in the simulated annealing framework [20]; the algorithm is completely deterministic. The output of this step is the 3x3-redundant multiresolution probability map which can be interpreted as the multiresolution fuzzy ROI mask, where each pixel in the pyramid is represented by an 8-bit value instead of logical values (0 and 1) only.
   c) The second algorithm resembles the voting method given in [8] and uses all results produced in previous steps, especially the multiresolution probability map generated by the MRF which provides a major improvement in performance. The driving force of this algorithm is the improved

Hough transformation [21], but limited to pixels which are indicated by the said probability map as likely to lie on a streak. The output of this algorithms is the voting map.

d) The third algorithm extracts most salient lines from the voting maps, clips them to segments and calculates the tracklet list which is the output of the whole streak detection block. In this algorithm we assume mixed pixel adjacency neighborhoods, as opposed to the first algorithm.

4. BENCHMARKING SETUP AND IMPLEMENTATION AND THE RESULTS

Our test bench is a single industrial PC of Intel Comet Lake architecture and with no GPU onboard. Specifically, we used the Onlogic HX500, a platform based on the Intel Q470 chipset with a 6-core Intel Core i5-10500T processor and 8GB of DDR4-2133 and NVMe storage. The cameras used in the test are 20 megapixel FLIR Black Fly S with the Sony IMX183 monochromatic sensor and VST VS50085 lens; the resolution is 5472x3648 with ≈10" per pixel and 12 bits, represented using 16 bits on the target platform. The HX500 motherboard layout allows connecting six such cameras without saturating its transfer capabilities. However, due to unknown routing of transmission lines from physical ports to Intel Q470 PCI-E lanes, and it could not be deduced from the bus tree visible in the operating system, it is safe to use two cameras with the Gig-E interface and four cameras with the USB3 interface. From the chipset block diagram [22] it follows that all USB3 ports may share the same PCI-E lane. We expect throughputs on the order of 100MB/s (5fps) of useful payload, hence to be on the safe side, it is best to utilize also two Ethernet ports which definitely do not share the PCI-E lane with USB3 ports. The north-south bus has throughput of around 4GB/s in one direction and that is much more than needed. The total power consumption of the HX500 platform is 43.4W when powered by the 24V line [23]; moreover the power consumption in the S3 ACPI state is below 2W. Taking a typical power conversion factor of 84% ≈ 5/6 adds another 20% and we arrive at 52W of power consumption. Each camera consumes typically 3W, hence adding six of these we arrive at 70W of total consumption. Therefore, the described hardware setup allows seamless transfer of the camera images to the internal memory and stays within the power budget.

The hardware described above is running the Debian Linux operating system with 5.4 kernel. All the cameras are synchronized by an external trigger, mimicking the target setup in which they are triggered by, for example, the GPS 1pps signal. Hence all the cameras start transmitting in the same time. It is therefore crucial to decrease the load on the platform and thread racing by setting the IRQ throttling and IRQ affinity to some fixed core; this is done using the abovementioned /proc subsystem. This noticeably reduces the average load, supposedly due to a lower cache invalidation rate. We do not interfere with the camera driver's stack and do not optimize its transfers because these transfers are DMA transfers, so after throttling of the interrupts all the image acquisition threads are easily handled by a single core. It must be noted that all automatic adjustments performed by the camera should be turned off: exposure correction, gamma correction, and gain automatic adjustment in case of cameras of our choice. It is a good place to mention that optical performance of our setup allows us to make observations up to 10.5mag (stationary objects), with full width at half maximum of (FWHM) of the PSF exceeding 6 pixels (≈1') and falling below 2 pixels nowhere in the image; this is the total effect, with chromatic aberrations and all possible geometric defects of the lens already counted in. Taking into consideration the exposure conditions, the lens geometry, the optical axis elevation, and the apparent angular velocity of LEO objects [24], in the acquired images we should expect streaks of lengths from 15 to 80 pixels.

The implementation details of the image processing toolpack described in the previous section are as follows:

A. The background modeler block updates the background mask every 1 second and this means every 5 frames, but the smoothing is performed every frame. The first step is the vignetting and sunset/sunrise correction, followed by updating the 5-frame moving average image at full resolution using the Intel IPP library [25]; this library is further on referred to as IPP. It should be noted that we do not use any IPP functions which are not sequential and could mess our CPU affinity settings. Then we perform the 8x8 resolution reduction using the Haar wavelet pyramid scheme (from the IPP). The resulting image resolution is 684x456. The reduced frame average value is calculated during this process. Every 5 frames we calculate the background mask by thresholding the image after subtracting the (vignetting- and sunset/sunrise-compensated) average and taking the absolute value. Next, we remove isolated pixels from the ROI; connectivity is understood in the sense of mixed adjacency. In practice this step is merged with the thresholding step and no additional pass is necessary. We end this block with calculation of the 3x3-redundant structure of the corrected image and the ROI mask pyramid.

This module has two free parameters which can be adjusted: the averaging window size and the threshold value. It is also possible to use different threshold values at different times of the day (or the night, one should say) to address the fact that observations made at dusk and dawn have larger clear sky intensity and therefore larger value of the threshold is required than during the astronomical or nautical night; we do not pursue this possibility here. It is also worth noting that stars useful for astrometry are not removed here because they are too small. The 8x8 resolution reduction at the step 2a turns them into isolated pixels which are subsequently removed in the step 2c. Should there be a problem with the astronomical seeing resulting in wider point spread functions, there are fast morphological algorithms to remove larger but still small connected groups of pixels of the binary ROI mask [26] (present in the IPP).

B. The star extractor module takes a (precomputed during deployment) list of objects not dimmer than 10m and their location within the image in the image coordinate system, and imposes latest astrometric corrections. At every moment there are practically never more than 200 such objects within the field of view and the full database is smaller than 10 gigabytes; exact value depends on the cut-off magnitude value adopted and can be much smaller when objects which are never visible are removed. Stars and other distant objects (eg. the Moon) move by as much as 1.5 pixels per second, or up to 0.3pix/frame in our setup, depending on the latitude of installation and on the line of sight. The only task to be performed every frame is a collection of binary logical operations on mask templates and the star mask; we use the IPP version wherever possible. Due to low number of operations, this block is the most lightweight one in terms of CPU load. In the cold start mode, when no astrometric corrections are available, this module cannot produce highest quality results. It has been found that one cannot rely indefinitely on the deployment-level calibration and during the cold start it is necessary to perform the full astrometric calibration.

C1. The first streak detector subblock implements the multiresolution MRF line detector. The algorithm starts from the lowest resolution, that is 684x456. Since the PSF FWHM does not exceed 6 pixels, this means that the streak width is one pixel. This, together with 3x3 neighborhoods used for clique potentials, guarantees correct detection, even in the case when the streak is shared by two neighboring pixels. We use clique potentials inspired by those in [19], but adopted simplifications and modifications allowed us satisfy conditions necessary to use the ICM algorithm [20] for rapid field evaluation. We initialize the field with the image itself and perform a fixed number of 15 iterations. Going up the multiresolution pyramid we take the upscaled result as the starting point for evaluation of the next level. This means that if the streak width (measured by the PSF FWHM) is 4 or less, it is often that it is missed on the first level of the pyramid. However, it is then found during evaluation of the second level, but such a case gets a special treatment. We evaluate the MRF at three out of four pyramid levels and the highest level of original resolution is just upscaled; evaluation at the highest level would require larger cliques to bring in any new quality. Due to our 3x3-redundant memory organization, we work in the pipeline at all pyramid levels and use IPP arithmetic operations wherever possible.

This module has several free parameters and criteria which can be adjusted, number of the ICM iterations and the stop condition being the most important. We chose not to introduce an overall measure of convergence because its evaluation would require to perform an additional operation along the ICM probability update, not to mention the fact that the ICM does not guarantee convergence. It is computationally more efficient to give up evaluation of such a measure and to adopt a safe, fixed value for the number of iterations.

C2. The second streak detector subblock implements the voting algorithm extracting line parameters of streaks in the image. The first step is to narrow down the ROI mask obtained by the background modeler using the output from the previous step. The key assumptions is that the MRF map calculated there contains mostly low values in the background region and significantly higher values where there is a chance for a streak. Operationally this map can be thought of as a gain map. This gain map is thresholded using the Otsu automatic thresholding scheme [27], and then the thresholded map is morphologically closed with a 3x3 structuring element of all ones. The thresholded gain map is then used to clip the original gain map. In this way the gain map is “clipped” to interesting regions and values outside these regions are pushed down to zero. It is important that pixels above the threshold and their immediate vicinity are not modified (in [8] the image itself is thresholded and the binary image is processed from that point). Then comes the most complicated component of our solution, the kernel-based Hough transformation [21]. This transformation in itself is lengthy in terms of lines of code (with heavy use of the IPP), already after some improvements that could be made under the assumption that the background is quite uniform. We will not delve into this algorithm any deeper because it is a fairly standard image processing tool. However, it is not standard to

evaluate this transformation on a small piece of the image indicated by the clipped gain map. This pixel preselection scheme results in a voting map in which peaks are found more easily.

C3. The last streak detector subblock implements the tracklet generation given the voting map from the previous step. The voting scheme design implies that streaks are indicated by maxima of the voting map. It is important to note that our cascade detection scheme admits relatively limited number of pixels to the Hough transformation step, hence the voting map features low pollution by the background. Therefore, detection of peaks in the voting map is a relatively simple task and we take 10 locally maximal values and 10 locally maximal values of the voting map integrated over 3-by-3 cells. The second option provides correct peak detector response in cases of peaks dissolved over neighboring voting map cells. The next step is to verify which of our 20 streak candidates are actually streaks. Given a maximum in the voting map we know its corresponding line parameters. We calculate the intensity map along the streak line and check whether this map contains the assumed streak pattern; we have assumed that the streak is a line segment convolved with the Gaussian PSF. This job is performed using a simple summation scheme with moving ends and can be thought of as a rectangle fitting. Once we know the approximate beginning and the end of the confirmed streaks, we calculate the corresponding tracklet data and prepare the corresponding image regions to be sent to the central node of the network. There is no need at this moment to extract the streak ends precisely, it will be redone much more accurately at the central node of the network.

We have tested our toolpack using images acquired by the hardware. In order to provide reliable test conditions, during several runs at good weather the video streams were recorded. The toolpack was then benchmarked using selected streams, we do not describe here any tests performed live. We have chosen a sequence of 10000 consecutive frames acquired on the 1st of March 2022. A typical preprocessed frame, together with its wavelet decomposition approximations is shown in Fig. 1. below. A typical streak, together with its wavelet decomposition is in turn shown in Fig. 2. It is apparent that at the lowest pyramid level the streak is salient and the background is uniformized. This is the low-pass effect of the Haar wavelets.

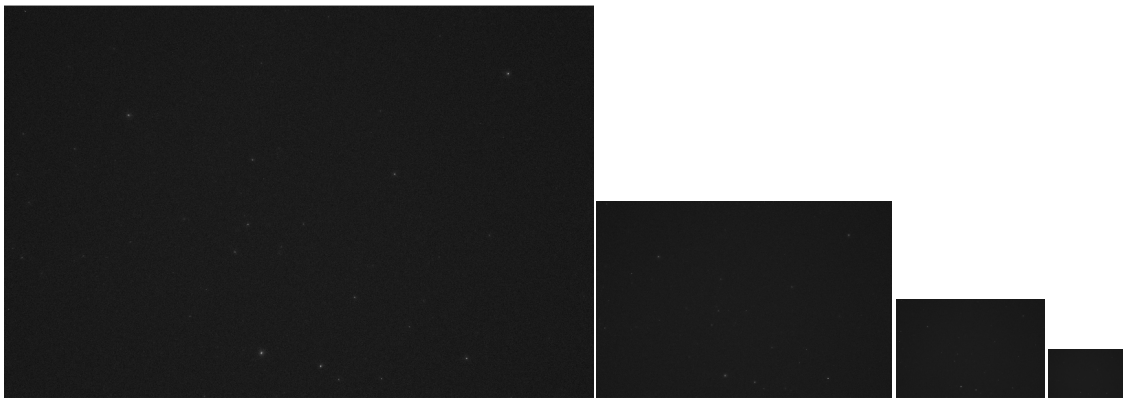

Fig. 1. A typical preprocessed frame, together with its wavelet decomposition approximations. Zoom in for details.

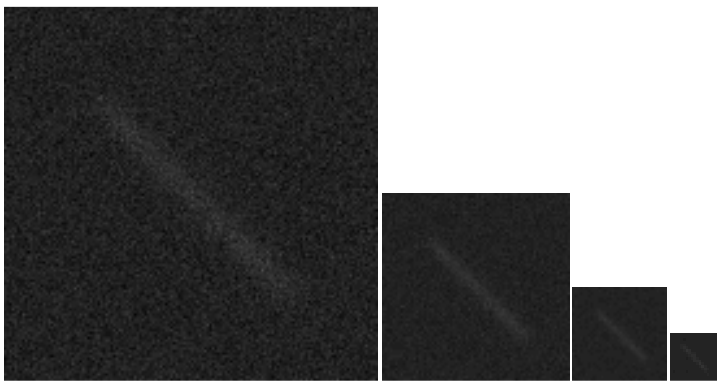

Fig. 2. A typical long streak, together with its wavelet decomposition approximations. Grayscale values multiplied by a factor of 5 for presentation. Zoom in for details.

The MRF line detector has no problem with detecting the streak shown as the rightmost picture in Fig. 2 above and in most cases returns very good results at the lowest pyramid level. This is one of the key properties assuring efficiency of our detection scheme, because at higher pyramid levels the processing starts from results from one level down. When a streak is present in the image, the Hough detection scheme provides a clear maximum in the voting map, as expected, and it is no problem at all to pinpoint it. The streak candidate verification also works very efficiently due to the minimum length criterion. Typical streak detection results are shown in Fig. 3 below.

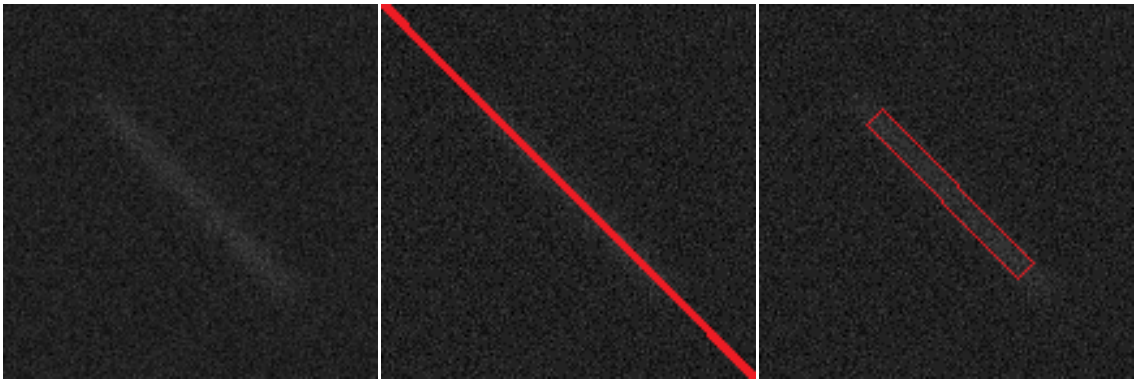

Fig. 3. From left to right: a typical long streak, the line detection result, the streak detection result. Grayscale values multiplied by a factor of 5 for presentation.

Our algorithm was capable to detect all streaks in the selected test sequence and the FPI rate did not exceed 6 which is considered a good result. It was observed that the free parameters of the algorithm can be fine-tuned to produce average FPI rates as low as 1.2 but at the cost of missing streaks of SNR close to the assumed cut-off value. In this study we did not pay attention the limiting SNR of the proposed detection scheme. There exist methods of detecting streaks where the SNR is low [13, 16]. In our approach the limiting factor stems from the initial MRF detector performance at the lowest resolution and from performance of the final block where confirmation of the streak presence takes place. The MRF block is insensitive to low SNR streaks, but the SNR is here calculated for the lowest resolution image, not the original. An in-depth discussion of the limiting SNR (object magnitude) and its impact on the precision and recall metrics is beyond scope of this paper.

5. CONCLUSIONS

The streak detection task can be effectively performed by a low-power platform, suitable for deployment in locations with no mains power. Following the path of architecture-aware top-down interface-driven design for all functional blocks of the image processing algorithm leads to a very efficient software ecosystem capable of working in the soft real-time regime. For support of these claims, implementation details of the software were presented and illustrated with some results of numerical experiments. The proposed algorithm performs well for fairly salient streaks, but it fails for low SNR streaks which could still be recovered by other known methods of larger computational complexity. Apparently, the detection algorithm contains many free parameters and therefore has also other operating points, with values of precision and FPI rate other than presented. Therefore, it is planned to evaluate the full free receiver operating characteristic (FROC) in order to allow control over the FPI rate which immediately translates into the upstream data bandwidth of the whole triangulation station. In future it is also planned to introduce stitching conditions for streaks in consecutive frames in order to employ the fact that LEO objects move continuously across the field of view, from one image boundary to another, as opposed to fireball events which can start and end anywhere; currently such tracklet stitching takes place at the central node of the network.

6. ACKNOWLEDGEMENTS

The author of this paper is grateful to Cilium Engineering and Stanisław Kozłowski for supporting this research by means of observational data and numerous fruitful discussions on how to improve triangulation technologies in a business-oriented manner.

7. REFERENCES

[1] Euroconsult. Commercial Space Surveillance and Tracking Final Report, *UK Space Agency Reports*. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/917912/Euroconsult_-_Commercial_SST_Market_-_for_publication.pdf, 17 March 2020, accessed: 14 August 2022.
[2] Regulation (EU) 2021/696 of the European Parliament and of the Council of 28 April 2021 establishing the Union Space Programme and the European Union Agency for the Space Programme and repealing Regulations (EU) No 912/2010, (EU) No 1285/2013 and (EU) No 377/2014 and Decision No 541/2014/EU, *Official Journal of the European Union*, L170:69-148, 28 April 2021.
[3] Commission Implementing Decision (EU) 2022/1245 of 15 July 2022 laying down rules and procedures for the application of Regulation (EU) 2021/696 of the European Parliament and of the Council as regards the participation of Member States in the SST sub-component, the establishment of the SST Partnership and the development of the initial key performance indicators, *Official Journal of the European Union*, L190:166-190, 15 July 2022.
[4] V. Morand, J. Alves, J. Gelhaus, S. George, J. H. Hermoso, E. Veluttini. System level studies to design optical surveillance networks in the frame of the EUSST Support Framework, *Proceedings of the Advanced Maui Optical and Space Surveillance Technologies Conference*, 2018.
[5] J. M. Hermoso, I. Urdampiletta, V. Morand, E. Delande, J. Gelhaus, E. Veluttini, et al. System approach to analyze the performance of the EU Space Surveillance and Tracking system, *Proceedings of the Advanced Maui Optical and Space Surveillance Technologies Conference*, 2021.
[6] S. Kozłowski, P. Sybilski, A. Olech, A. Raj, P. Żołądek, M. Litwicki, et al. OmniSky: Wide angle multi-camera station network concept for re-entry detection, *Proceedings of the 1st NEO and Debris Detection Conference*, 2019.
[7] M. Polkowska, A. Chimicz. Status of SSA activities in Poland and recent SST developments, *Journal of Space Safety Engineering*, 2022, in press.
[8] M. Towner, M. Cupak, J. Deshayes, R. Howie, B. Hartig, J. Paxman, et al. Fireball streak detection with minimal CPU processing requirements and the automation of the Desert Fireball Network data processing pipeline, *Publications of the Astronomical Society of Australia*, 37:e008, 2020.
[9] R. Danescu, R. Itu, M. Muresan, A. Rednic, V. Turcu. SST Anywhere - A portable solution for wide field Low Earth Orbit surveillance, *Remote Sensing*, 14(8): 1905, 2022.
[10] Z. Ceplecha, Geometric, dynamic, orbital and photometric data on meteoroids from photographic fireball networks, Bulletin of Astronomical Institutes of Czechoslovakia, 38:222-234, 1987.
[11] L. Varela, L. Boucheron, N. Malone, N. Spurlock. Streak detection in wide field of view images using Convolutional Neural Networks (CNNs), *Proceedings of the Advanced Maui Optical and Space Surveillance Technologies Conference*, 2019.
[12] A. Vananti, K. Schild, T. Schildknecht. Streak detection algorithm for space debris detection on optical images, *Proceedings of the Advanced Maui Optical and Space Surveillance Technologies Conference*, 2015.
[13] A. Vananti, K. Schild, T. Schildknecht. Improved detection of faint streaks based on a streak-like spatial filter, *Advances in Space Research*, 65(1): 364-378, 2020.
[14] P. Hickson. A fast algorithm for the detection of faint orbital debris tracks in optical images, *Advances in Space Research*, 62(11): 3078-3085, 2018.
[15] G. Nir, B. Zackay, E. Ofek. Optimal and Efficient Streak Detection in Astronomical Images, *The Astronomical Journal*, 156:229, 2018.
[16] W. Dawson, M. Schneider, C. Kamath. Blind detection of ultra-faint streaks with a maximum likelihood method, *Proceedings of the Advanced Maui Optical and Space Surveillance Technologies Conference*, 2016.
[17] J. Zscheile, P. Wagner, R.-A. Lorbeer, B. Guthier. Synthetic tracking for orbital object detection in LEO. *Proceedings of the ESA NEO and Debris Detection Conference*, 2019.
[18] V. Cantoni, M. Ferretti. *Pyramidal Architectures for Computer Vision*, Springer, 1994.
[19] C. He, Z. Liao, F. Yang, X. Deng, M. Liao. A novel linear feature detector for SAR images, *EURASIP Journal on Advances in Signal Processing*, 235, 2012.
[20] J. Besag. On the statistical analysis of dirty pictures, *Journal of the Royal Statistical Society: Series B*, 48(3):259-279, 1986.
[21] L. Fernandes, M. Oliveira. Real-time line detection through an improved Hough transform voting scheme, *Pattern Recognition*, 41(1):299-314, 2008.
[22] Intel. Product Brief, Intel Q470 Chipset, https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/q470-chipset-brief.pdf, accessed: 15 August 2022.

[23] OnLogic. HX500 / HX600 Product Manual, https://static.onlogic.com/resources/manuals/OnLogic-HX500-HX600-Manual-v2.pdf, accessed: 15 August 2022.
[24] J. McGraw, M. Ackermann, J. Martin, P. Zimmer. The Air Force space surveillance telescope, Proceedings *Proceedings of the Advanced Maui Optical and Space Surveillance Technologies Conference*, 2003.
[25] Intel. Intel® Integrated Performance Primitives Developer Reference: Developer Reference, Volume 2: Image Processing. Version 2022. https://software.intel.com/content/dam/develop/external/us/en/documents/ippi.pdf, accessed: 14 August 2022.
[26] P. Soille. *Morphological Image Analysis: Principles and Applications*, Springer, 2004.
[27] N. Otsu. A Threshold Selection Method from Gray-Level Histograms, *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62-66, 1979.