GPU based Video Object Tracking on PTZ Cameras

Cevahir Cigla, Kemal Emrecan Sahin and Fikret Alim
Aselsan Inc.
Turkey
ccigla,kesahin,fikretalim@aselsan.com.tr

Abstract

In this study, an embedded Pan-Tilt-Zoom (PTZ) tracker system design is proposed that is based on NVIDIA Tegra K1-X1 mobile GPU platform. For this purpose, state-of-the-art correlation filter (CF) based video object tracking (VOT) algorithms are exploited regarding their high performance. Each algorithmic step is carefully implemented on GPU that further increases the efficiency and decreases execution times. The PTZ control is designed to track human targets by centralizing within the image coordinates where the targets have limited speed but obvious appearance changes. Incorporating on-board decode and encode capability of Tegra platform as well as angular position control, the presented approach enables 50-100 fps target tracking for HD (1920x1080) videos on K1 and X1 correspondingly. This is to our best knowledge the first efficient implementation of CF trackers on a mobile GPU platform with use of multiple features, scale and background adaptation. This study extends the scope of accuracy focused VOT research to platform optimized efficient implementations for real-time high resolution video tracking.

1. Introduction

VOT is a common tool for a large variety of computer vision applications such as video based surveillance and autonomous robotic systems. The recent VOT benchmarks and challenges [1], [2], [3], [4], [5] had an impact on tracking research by providing a common evaluation platform for the comparison of algorithms. The key contributions of these benchmarks are the labeled videos and availability of open source algorithms. In that way, new studies can easily use the outputs of state-of-the-art techniques to compare results on the same platform with much less effort.

Despite the advantages provided by these benchmarks, there are two significant issues that have not been discussed adequately; computational complexity and specific tracking applications. These benchmarks mostly focus on the accuracy of tracking performances with respect to various evaluation metrics. The computational complexity is the secondary focus where timings are achieved for CPU devices with negligible efforts on optimizations. The algorithms are mostly implemented on high level languages (such as MATLAB) and high compute capability devices (such as workstations) which are far from deployment level on low cost embedded systems. The second issue is the specific tracking application problems. VOT benchmarks involve large variety of videos for very different applications (fish tracking in aquarium or tracking football players) that definitely require special assumptions. In that manner, after an overall evaluation, winning algorithms may not be the most efficient or best performing algorithms for special purposes. In this study, we focus on real-time HD video PTZ tracking on cost efficient mobile platforms and provide a framework for real-time GPU implementations of VOT algorithms. Nvidia Tegra K1 and X1 series are chosen as the target platforms which have revolutionized mobile computing with their effective GPU capabilities. Based on the real-time constraints, we have incorporated correlation based filter and its state-of-the-art variants as the base algorithms for the proposed framework.

The related work on VOT is discussed in section 2, which is followed by the details of GPU implementations of the chosen correlation filter trackers. The fourth section is devoted to the hardware and data framework of the overall system that is composed of a PTZ camera and a low cost GPU device. Giving the experimental results, the paper concludes by the discussions and future work.

2. Related Work

The evolution of VOT algorithms can be observed by the annual VOT challenges [1]-[5] that provide a comprehensive overview. According to the recent VOT challenge [5], almost half of the competitors exploit correlation filter tracking and one third of the competitors take advantage of convolutional neural networks (CNN) based features. Discriminative correlation filter is the common approach for the top ten performing algorithms. In the early years of these challenges, KCF [6] has been the game changer approaches

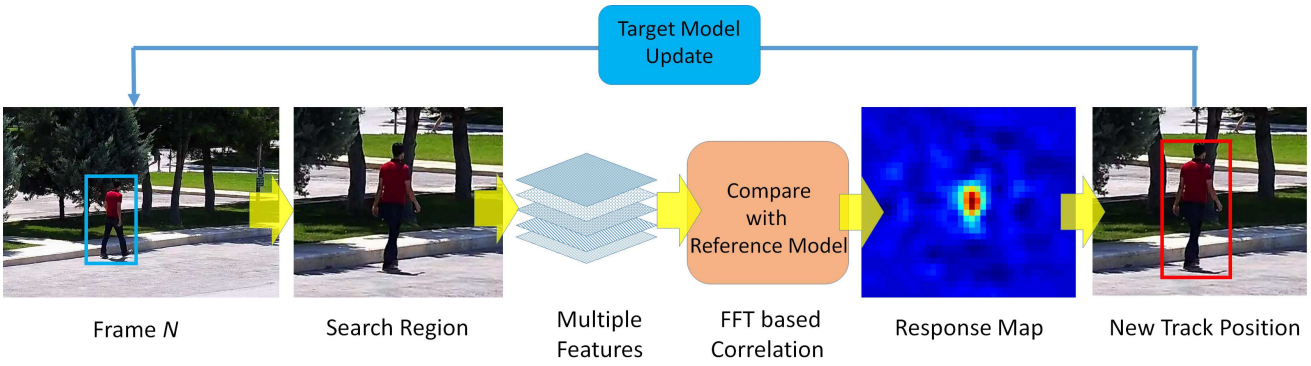

Figure 1: The flow of correlation based trackers

that introduced efficient correlation based on Fourier Transforms and multi-dimensional features. Even though the recent top ten trackers [5] vary in terms of feature utilization, Histogram-of-Gradients (HOG) [7] and color names [8] are the common features that are exploited by all trackers.

Statistics of trackers in the recent challenges indicate that significant amount of research is devoted for correlation filters. Circular shifts and Fast Fourier Transform are the efficient fundamental tools of CFs. The common flowchart of CF trackers is given in Figure 1, where the target template is correlated in the following frames within a search region based on the extracted multiple features. Then new position of the target is estimated by the response map obtained through FFT based correlation. Finally, target model is updated as the tracks are successful. One of the pioneering CF based approaches is DSST which exploits intensity features to represent a target. This work is modified by use of HOG as multi-dimensional features and kernelized correlation with more robust tracking [6]. The performance and computation balance has provided KCF to be one of the most exploited approach in recent years. Almost each CF tracker utilizes multi-dimensional features increasing robustness and accuracy. SAMF [9] introduced scale adaptation and use of color features where different scales of the target template are correlated to find the best match. This has boosted-up robustness and applicability of CF for tracking. Recently, Staple [10] incorporated histogram features to handle dynamic targets with shape changes. Response maps of HOG and histogram based features are merged to find the best match. On the other hand, context-aware CF (CFCA) [11] and discriminative CF tracker [12] have been proposed to handle background changes around the target that improves accuracy and robustness.

VOT challenges provide an excellent environment to compare trackers with the open source implementations. In most cases, algorithms are implemented on CPU with high level languages (MATLAB) that restricts the trial on mobile or embedded systems. Even though they provide flexible experimental comparison especially on tracking accuracy, execution time evaluation is mostly restricted on CPU implementations which is not sufficient to give ideas for real-world systems. In VOT 2017, there are only two significant trackers [12] and [13] that are implemented in GPU with C implementations, while most GPU implementations are given in MATLAB which requires less effort and optimization. Recently, [14] proposed a GPU implementation of KCF based on deep comparison network with single scale adaptation on Tegra X1. It is also important to note that, most of the trackers utilize high power GPUs to enable deep learning approaches. The best performing real-time tracker in VOT 2017 [12] is almost ten times slower than KCF on single scale.

As the camera technology develops, video resolutions and frame-rates increase and the need for faster trackers is inevitable. According to [15], complexity of the trackers losses their advantages over fundamental trackers such as DSST and KCF for higher frame rates. The real-world scenarios require scale adaptation as well which increases the computation time linearly in general. Even though DSST provides a novel solution for scale adaptation, it is eight times slower than KCF. At that point, special attention is required to map those efficient algorithms to GPU apart from the fundamental parallelization tools in high level languages. Thus, mobile platforms can benefit these efforts to cope with fast developing camera technologies.

The other branch of VOT research is devoted to improve and utilize the real time trackers on pan-tilt-zoom (PTZ) cameras. [16] proposes a real time vision system that focuses on control of the camera and object tracking under different conditions by using a SONY network camera SNC-RZ30N. An optical flow background estimation method is exploited in [17] for real-time VOT using Point-Grey Research USB 3.0 Flea-3 cameras with PTU-D46-17 pan/tilt units. [18] copes with low frame rates using SONY IP PTZ cameras(SNC-RZ50N and SNC-RZ25N). Adaptive background generation and moving object detection are uti-

lized over Panasonic PTZ camera in [19]. [20] proposes position tracking that focuses on the estimation of object position using IP-Surveillance system contains AXIS PTZ-215 cameras. It is important to note that, none of these works provide any analysis of their VOT algorithms as well as performance over a Wide Area Network(WAN) that may cause fluctuation in response time of camera control between PTZ camera and VOT system.

In this study, we propose GPU implementations of well known and recent CF trackers SAMF [9], Staple [10] and $CFCA$ [11] with scale adaptation on NVIDIA Tegra K1-X1 modules which are accepted as efficient mobile platforms. Preserving reported performances of the trackers, the proposed implementation provides up to x3 speed-up compared to full CPU implementation on the same platform. Besides, we incorporate camera control on preselected Dome cameras to design a smart tracker camera that works on 1920x1080 video with 50 fps.

3. Algorithm Selection and GPU implementation

The algorithmic complexity and availability are the key constraints that are faced during the selection of algorithms for a real-time performing system. At that point, we focus on the mostly exploited CF trackers which provide sufficient space for improvement on GPU. CF trackers have common functional blocks as illustrated in Figure 1. As mentioned previously, there are many types of CF trackers which have various differences on each step. Among these choices we have chosen a representative set of trackers that have comprehensive features important in terms of GPU implementation.

3.1. Algorithms

Kernelized correlation filter [6] has provided significant performance boost-up as well as robustness by the use of multiple features such as HOG. In SAMF [9], KCF has been extended to handle changes in the scale, as well as incorporated an efficient color representation with HOG features. Afterwards, these features have been a standard for CF trackers with their proven reliability and robustness against various conditions. The scale adaptation is also an important factor that enables trackers to cope with size changes of targets especially in surveillance systems, where position of the cameras are generally fixed and the distance of the targets change. It is clear that SAMF has the fundamental capabilities of CF trackers, multiple feature utilization, scale adaptation and efficient FFT based correlation. Thus, we have chosen SAMF as one of the baseline trackers for GPU implementation. It can easily be mapped to KCF by exploiting single scale and omitting color features.

Recently, staple [10] exploited histogram features to handle shape changes of the targets. For this purpose, two response maps, multiple feature based correlation and foreground/background histogram similarity, are extracted and merged by weighted summation. In [10], it is shown that use of histogram similarity improves the tracking especially for non-rigid objects with marginal burden on computational complexity. In the case of surveillance tracking, it is important to handle shape changes since the targets are mostly human. Therefore, we have chosen Staple tracker as the second baseline for our GPU implementation framework.

Adaptation of trackers according to the environmental changes is another important factor for robustness. The update of target model is sometimes insufficient when the background is dynamic and highly textured. $CFCA$ [11] provides a solution to adapt the target models by use of negative samples around the target. Hence, in any direction of the target, negative samples provide the tracker to learn regions where the response should give low correlation scores. In that way, background-foreground mixing is prevented. It is a novel idea to incorporate negative samples into the target model, which forms the third approach to be implemented in GPU.

As indicated in [5], there are lots of variations of CF trackers improving the performances each year. One recent approach that has been exploited excessively in VOT challenges is the use of deep features instead of hand crafted ones. That seems to improve the performances however increasing computational complexity as well [14]. The complexity is critical for the deployment on mobile devices which is the main point of this study. For the time being, we focus on GPU optimization of fundamental CF tracking features, use of multiple features (HOG-color names-histogram), scale, shape, environment adaptation and FFT based correlation. It is one of the best ways to exploit deep features designed for specific targets however, there is still room for the real time deployment of those features, which is considered as a future direction of this study.

In the chosen CF tracker set, each method has modifications over the base KCF algorithm. They mostly vary on the additional features as indicated previously. In Figure 2, the regions exploited to extract additional features are shown where the orange region corresponds to the fundamental HOG and color name features, the red area indicates the region for histogram calculation in staple [10], and the blue areas define the negative response regions around to target in [11].

3.2. GPU implementation

The target platform for the GPU implementation of CF trackers is the well known mobile Nvidia Tegra K1-X1 boards. In general 16x16 processing blocks are utilized with 256 threads, as shown in Figure 3, that gives the most efficient tiling for K1. The thread are organized in coalesced

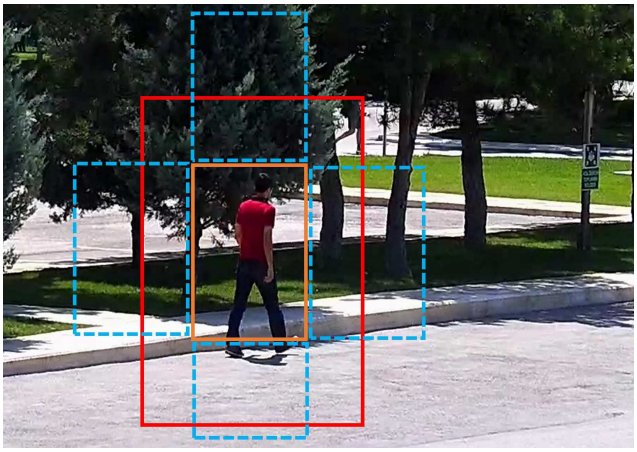

Figure 2: The regions defined for different features exploited by various CF trackers, HOG and color name features that are common for all CF trackers, histogram features exploited in Staple, and the false target regions for context-aware CF tracker

memory pattern to get fast and efficient reads as shown in the middle of the first row of Figure 3. Besides RGB images are stored in *unsigned char4* format so that each thread gets *rgb* values with one memory access. Having efficient memory access, the details of GPU implementations are presented in three steps according to the framework given in Figure 1.

3.2.1 Feature Extraction

Feature extraction starts with the region crop around the target center and CPU to GPU data transfer. In order to minimize the data transfer which has a significant impact on the computational complexity, the region corresponding to the largest scale is cropped which is then re-cropped for smaller scales as shown in the second row of Figure 3. Thus, number of memory copies is set to one and independent of the number of scales that will be searched. Once the related region is copied to GPU device memory, HOG features and compact color names representation are extracted according to the tiling in Figure 3. Coalesced memory access is used to enable fast RGB image processing through low number of memory accesses. The atomic built-in functions (i.e.*atomicAdd*) are efficient for the calculation of histograms during HOG feature extraction. The color names features are extracted by fast look-up tables with the same tiling configuration in Figure 3. The feature extraction, excluding the region crop, is conducted for each scale independently. Throughout the extraction of features, bottleneck is formed along the histogram calculation which depends on the atomic functions. It is still quite efficient on GPU since the size of the target region is limited (96x128) in order to guarantee a processing time that meets

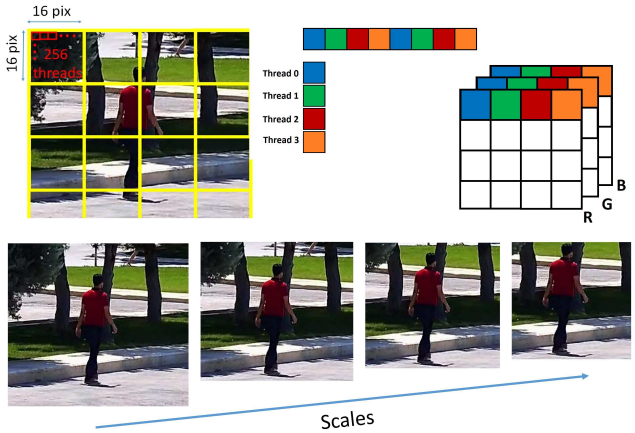

Figure 3: The tiling for GPU processing, only one CPU-GPU memory copy is performed for highest scale.

the final tracking speed. Thus, even the initial target size is larger that the pre-defined sizes, the cropped region is down-sampled to the desired size, which does not affect the tracking performance of considerably large objects.

3.2.2 Response Calculation

Once feature representation of the search region is ready, the correlation response is calculated by the use of built-in FFT functions of CUDA. Multiple FFT operations can be conducted at the same time, yielding an efficient implementation for correlation. The FFT of target model and the search region are multiplied element-wise for each channel independently with same thread tiling given in Figure 3 which is followed by inverse FFT. Then the response of each feature channel is added to obtain the final response of the search region. There are minor differences between the implemented CF trackers, in SAMF the inverse FFT is calculated for each channel independently in a kernelized way then the addition is performed over the channels. On the other hand, for the linear kernel version of SAMF and the remaining trackers, only one inverse FFT calculation is performed after the summation of channel FFTs.

In the CFCA tracker, additional operations are performed for the negative response regions which follow exactly the same approach described previously. Negative response regions are included in the target model. The regions that look like the negative sample result in low correlation scores and prevents tracker from stucking on background. Histogram similarity check provides additional response map that is introduced in staple. Having the background/foreground histogram distributions, Foreground similarity of each pixel is calculated in the search region which is achieved by fast look-up tables along the two histograms through 16x16 tiling. This response is resized to the FFT response map and then added for the final result.

Localization of the target is achieved by finding the maximum correlation in the response map through built-in *atomicMax* function in CUDA. Currently, we exploit pixel precision tracking however it can easily be extended to sub-pixel accuracy by polynomial fit around the peak response point.

3.2.3 Model Update

The last step of visual tracking is the update of target model with the use of recent observation. For this purpose, feature extraction is performed on the new target position within a region defined by target size. The new representation is incorporated into the target model by weighted summation which is easily achieved by pixel-wise additions on 16x16 grids.

In PTZ tracking scenario, it is required to pan and tilt the camera in order to hold position of the target within the center of image. For this purpose, detected target positions in image coordinates *(u,v)* are converted into pan and tilt degrees in camera coordinates as:

$$\Delta\theta = \tan^{-1}\frac{u}{vc(\phi)+fc(\phi)}$$
$$\Delta\phi = -\phi + \tan^{-1}\frac{vc(\phi)+s(\phi)}{vs(\Delta\theta)+c(\Delta\theta)[-vs(\phi)+c(\phi)]} \quad (1)$$

where ϕ is the current tilt angle position of the camera, $\Delta\theta$ and $\Delta\phi$ are the pan-tilt angles to be applied to get the target at the center of the image, c and s are the abbreviations of cos and sin for simplicity. It is also important to note that, targeting PTZ cameras to the exact pan-tilt angles is not a good way to provide smooth tracking which results in hysteresis with fast camera motion. Instead, we define speed parameter for the camera, which aims to get target in the image center within a pre-defined time interval (eg. 2 seconds). In that way, the camera slows down as the target gets closer to the center and overshooting is prevented. The calculation of angles and the related camera speeds are conducted on CPU which require very low computational burden.

4. System Outline

The system architecture of the proposed smart tracker camera is given in Figure 4. The system consists of a PTZ camera, Tegra K1 or X1 board and video clients. In subsection 4.1, the overview of hardware structure of the system architecture is given which is followed by the the software structure.

4.1. Hardware Structure

As shown in the figure 4, ethernet is the main interface that connects PTZ Camera and a Tegra as well as Video clients to the tracker camera. Hence, two separated ethernet interfaces are required on Tegra platform that is met by a mini-PCIE ethernet adapter on Tegra board. The PTZ camera is an IP Camera with HD resolution. PTZ controls are made over HTTP protocol through the known camera field-of-view (FOV) and pan-tilt speeds. The camera video stream is controlled by RTSP and the video stream is transmitted by RTP over UDP.

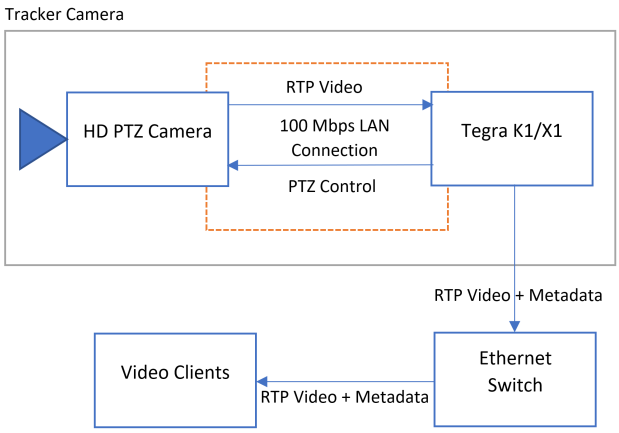

Figure 4: Hardware Structure of Video Object Tracking

Tegra carrier boards may vary according to performance requirements. In this study, we have tested several boards such as ConnectTech Astro Carrier for TX1 [21], NVidia EVMs Jetson TK1 [22] and Jetson TX1 [23], Apalis TK1 module [24] and Ixora carrier board [25] and custom boards for TX1. Tegra platform takes video stream from IP Camera over ethernet interface and sends RTP video + metadata over the second ethernet interface. RTSP server and RTSP client are also supported with the help of our software development. Thus, video clients can get video stream from the tracker camera by RTP/RTSP.

4.2. Software Structure

Software implementation consists of two main pipelines shown with dashed rectangles as seen in Figure 5. The video stream and PTZ camera control related information are transferred over colored links in which red link represents the video stream, blue link contains the pan-tilt-zoom information and the orange link transmits the metadata generated by tracking algorithm. In the first pipeline, there are three blocks; rtp stream receiver, gstreamer decoder and the tracking algorithm. Correspondingly, streaming of video from a target PTZ camera, decoding of h264-encoded video stream into required format for tracking algorithm and finally CF-based tracking are followed in this pipeline. Video stream is gathered via the real-time streaming protocol based client written in QT. H264-encoded stream is decoded with a similar pipeline seen in Figure 6 that is forwarded to the VOT algorithm block.

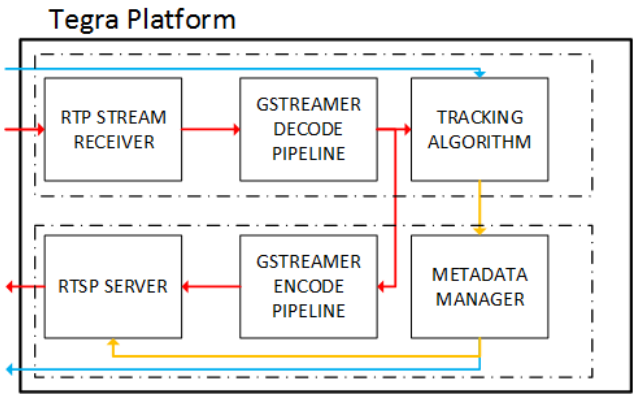

Figure 5: Software Structure of Video Object Tracking

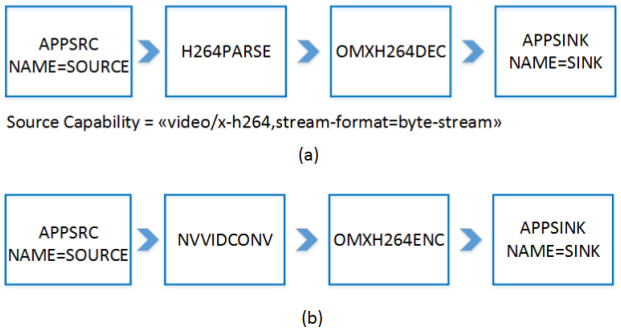

Figure 6: Gstreamer Decode Pipeline

In the second pipeline, there are metadata manager, gstreamer encode and rtsp server. Metadata manager is responsible for parsing the final pan-tilt speeds determined by the VOT block. Once the pan-tilt speed and the direction of movement are determined, the control command of PTZ camera is constructed. This command is transmitted to the motor controller of the PTZ camera. The gstreamer encodes the decoded video stream as in Figure 6b which is also sent to video clients in h-264 format.

The information flow in blue link (between PTZ camera and tegra platform) is provided by HTTP-based server-client communication protocols. Tegra platform acts as HTTP-based communication client that requests pan-tilt related information using GET commands and sets the pan-tilt speed of PTZ camera using POST commands.

5. Experimental Results

The proposed PTZ tracking system is analyzed in three steps. In the first part, a comparison is given between CPU and GPU implementations through the executions timings and the tracking accuracy. The algorithms implemented on GPU are compared in terms of execution speed in the following subsection. The overall system performance is discussed in the final part mostly focusing on the timings and the final tracking capability. The experiments are conducted on two different Nvidia Jetson boards; TK1 and TX1.

5.1. CPU-GPU Comparison

In this study, GPU implementation is developed over the open source C code provided by [6] where multi-scale version of KCF is also implemented which actually corresponds to SAMF. Thus, the algorithmic CPU-GPU comparison is provided over this approach that covers the most critical steps of CF trackers including HOG and color name features, multi-scale extension, FFT based correlation and detecting new target position. Having the verified GPU implementation of each step, it is rather easy to extend with use of histogram similarity in staple [10] and CFCA [11].

In order to be fair, CPU code is also optimized for faster execution. During the experiments, we have exploited two of the recent tracking benchmarks [26][27] as well as our surveillance videos captured for human tracking that involves 50 videos with 1920x1080 resolution (totally 15000 frames). In order to verify GPU implementation, we run CPU and GPU codes consecutively on the same videos with same initial target definitions. The average absolute horizontal and vertical position differences are 1.3 and 1.2 pixels over 15000 frames with HD resolution. These results indicate that GPU implementation produces almost the same output with the original CPU version. The tracking outputs visualized in Figure 7 indicates that the proposed GPU implementation (SAMF Gauss) has almost same output compared to original CPU version.

The execution times of CPU and GPU versions of SAMF [9] are given in Figure 8, and it is clear that GPU provides x3 speed-up on the Tegra platform. The modified CPU version of SAMF is also 4 times more efficient that the same code experimented in [14] (8 fps on 1280x720 with X1). Hence, the proposed GPU version enables x12 improvement compared to the experiments given in a recent study [14]. We have chosen three different target sizes (100x100, 200x200 and 300x300) and run the experiments on full resolution. Throughout the experiments, five different scales (*1.1-1.05-1-0.95-0.9*) are utilized to adapt target size changes which has linear effect on the computational complexity. It is important to note that, GPU improvement is more obvious as the target size increases. The detailed computation time analysis (K1 module) for three main steps of CF trackers is given in Table 1 where the target size is chosen to be 200x200. According to the results, HOG feature extraction is the most time consuming step which is followed by the FFT calculations. The improvement by GPU is much more obvious for the feature extraction step which almost provides x7 speed-up. On the overall, proposed GPU implementation enables up to 50 fps on Tegra K1 for HD videos, which is further speeded up by 2 on X1 module.

Figure 7: Visual outputs of implemented GPU algorithms SAMF linear, SAMF Gauss, CFCA, Staple and SAMF Gauss in CPU.

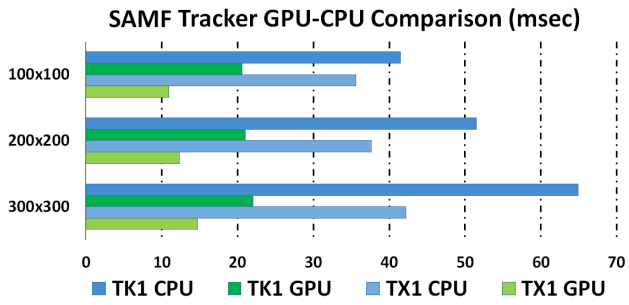

Figure 8: The average CPU-GPU computation times on Jetson TK1 and TX1 for three different target sizes.

5.2. Methods on GPU

The computation times of provided GPU implementations for different CF trackers are provided in Table 1. The Gaussian kernel version of SAMF requires 30% more computation compared to the linear kernelized version. In staple, use of histogram response map introduces additional *1.5 msec* while in context aware tracker additional feature extraction along negative response regions almost triples computation in the model update step. As the algorithm complexity increases with additional steps, execution times go up to 32 msec which is still within the conventional 25 fps bounds. Compared to the GPU implementation of KCF with deep comparison networks in [14] where tracking takes 22 msec on 1280x720 video (49 msec for HD resolution), proposed GPU implementation of the most complex chosen algorithm (staple) requires almost 70% less computation with 16 msec tracking on 1920x1080 video. It is important to note that CPU-GPU memory transfer of HD frames takes almost 30% of the computation on K1 board, while on X1 board this part can be handled through pinned memory utilization. However, this advantage is not reflected to the comparison in this study that could further increase system capability especially on X1 boards.

The accuracy of the chosen CF trackers have been analyzed comprehensively in VOT challenges [1]-[5], there-

Table 1: Computation time distribution of CF trackers on Jetson TK1 for a target with size of 200x200 pixels, *corresponds to CPU implementation

Time(msec)	CPU-GPU transfer	Feat. Extr.	Detection	Model Training	Total
SAMF_linear	6.4	6.9	3.5	3.6	20.4
SAMF_Gauss	6.4	6.9	8.5	4.8	26.6
CFCA	6.4	6.9	4.1	12.4	29.8
Staple	6.4	8.5	8.7	7.8	31.4
***SAMF_linear**	-	48.1	11.7	4.6	64.4

Table 2: The performance of implemented GPU trackers on Jetson TK1 and TX1.

Time(msec)	SAMF_linear	SAMF_Gauss	CFCA	Staple
TK1	20.4	26.7	29.3	34.7
TX1	12.2	15.2	16.8	20.0

fore we do not provide additional comparison. On the other hand, some typical visual results are illustrated in Figure 7 for the sake of completeness. The timing performances of trackers on Tegra K1 and X1 boards are given in Table 2. It is clear that, X1 provides almost x2 speed-up without any exception over different methods.

5.3. Overall System timing

The timing on the overall system architecture is given in Figure 12, where the video encode-decode take 1-2 msec that are negligible compared to the VOT algorithm block. The management time of the PTZ camera control (duration of GET and POST request commands) is one of the most critical aspects that affect the system behavior. The duration of GET request command has an average response time around 450 msec which limits the frequency of pan-tilt requests provided by Tegra platform. Since the software system has the initial pan-tilt information and track speed, the same pan-tilt speeds are utilized until next pan-tilt information is requested. The response time of POST command is approximately 100 msec hence the software system sends a POST request around every 200 msec. As mentioned previously, pan-tilt speed is estimated to get the target in the center of the camera within 2-3 seconds. Thus, the overall response and request interval of GET and POST command do not cause any issue to PTZ camera control.

Typical system outputs during a real world tracking scenario are illustrated in Figure 10. The system is mostly designed to track human under surveillance scenario where speed and distance of the target is limited. Thus, the delays introduced during get-set commands on PTZ, do not have significant impact on the tracking capability. As the target gets closer to the camera center, the tracking speed decreases in order to prevent hysteresis. It is clear that the system is robust against background and target appearance

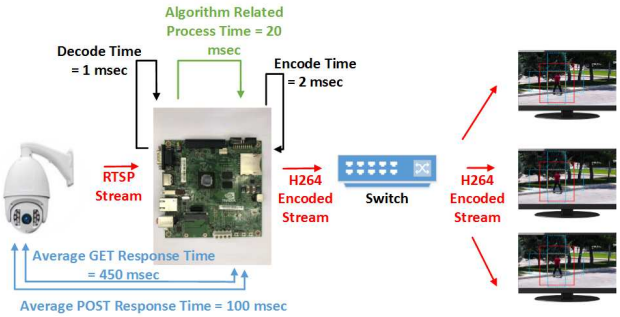

Figure 9: System architecture and the overall timing

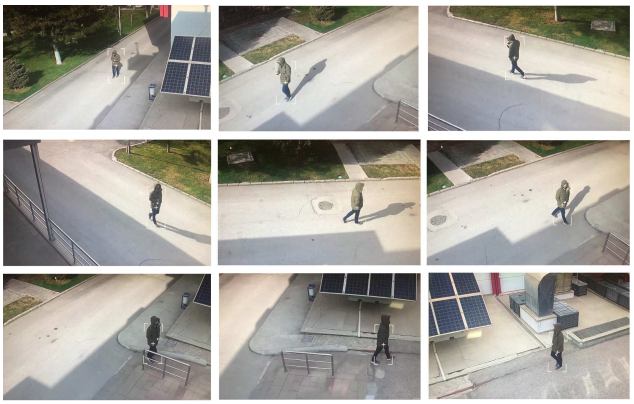

Figure 10: The Visual outputs of the PTZ tracking system that runs in real-time on HD videos

changes and tracks the human target properly.

6. Conclusion

In this study, an efficient GPU implementation framework of well known correlation based trackers is presented which is further incorporated into PTZ tracking system. Streaming HD videos through surveillance cameras, performing on board decode and encoding, the proposed system is able to track human targets with high frame rates i.e. 50 and 100 fps for Tegra K1 and X1 consecutively. Incorporating system on chip versions of these mobile GPU platforms, this seems to be an important contribution for the future of smart cameras. Having implemented four versions of CF based trackers with hand crafted features, multi-scale and background adaptation, we plan to extend the proposed framework for deep features that are designed for specific classes of targets depending on the application.

References

[1] Matej Kristan et'al. The visual object tracking vot2013 challenge results. *ICCV Visual Object Tracking Workshop*, Dec 2013. 1, 7

[2] Matej Kristan et'al. The visual object tracking vot2014 challenge results. *ECCV, Visual Object Tracking Workshop*, 2014. 1

[3] Matej Kristan aet'al. The visual object tracking vot2015 challenge results. *ICCV Visual Object Tracking Workshop*, Dec 2015. 1

[4] Matej Kristan aet'al. The visual object tracking vot2016 challenge results. *ECCV Visual Object Tracking Workshop*, Dec 2016. 1

[5] Matej Kristan et'al. The visual object tracking vot2017 challenge results. *ICCV Visual Object Tracking Workshop*, 2017. 1, 2, 3, 7

[6] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2015. 1, 2, 3, 6

[7] Girshick R. McAllester D. Ramanan D. Felzenszwalb, P. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2010. 2

[8] Schmid C. Verbeek J.J. Larlus D. Van de Weijer, J. Learning color names for real-world applications. *Image Processing, IEEE Transactions on*, 2009. 2

[9] Yang Li and Jianke Zhu. A scale adaptive kernel correlation filter tracker with feature integration. In *Computer Vision - ECCV 2014 Workshops: Zurich, Switzerland, September 6-7 and 12, 2014, Proceedings, Part II*, pages 254–265. Springer International Publishing, 2014. 2, 3, 6

[10] Luca Bertinetto, Jack Valmadre, Stuart Golodetz, Ondrej Miksik, and Philip H. S. Torr. Staple: Complementary learners for real-time tracking. June 2016. 2, 3, 6

[11] Matthias Mueller, Neil Smith, and Bernard Ghanem. Context-aware correlation filter tracking. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 3, 6

[12] Luka ehovin Ji Matas Alan Lukei, Tom Voj and Matej Kristan. Discriminative correlation filter tracker with channel and spatial reliability. 2018. 2

[13] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In *European Conference Computer Vision (ECCV)*, 2016. 2

[14] Krishneel Chaudhar, Moju Zhao, Fan Shi, Xiangyu Chen, Kei Okada, and Masayuki Inaba. Robust real-time visual tracking using dual-frame deep comparison network integrated with correlation filters. In *nternational Conference on Intelligent Robots and Systems (IROS)*, 2017. 2, 3, 6, 7

[15] Hamed Kiani Galoogahi, Ashton Fagg, Chen Huang, Deva Ramanan, and Simon Lucey. Need for speed: A benchmark for higher frame rate object tracking. *arXiv preprint arXiv:1703.05884*, 2017. 2

[16] Gerard Medioni Thang Dinh, Qian Yu. Real time tracking using an active pan-tilt-zoom network camera. In *International Conference on Intelligent Robots and Systems, IEEE/RSJ*, 2009. 2

[17] Jonathan T. Black Daniel D. Doyle, Alan L. Jennings. Optical flow background estimation for real-time pan/tilt camera object tracking. *Elsevier, Volume 48, pp 195-207*, 2014. 2

[18] Guillaume-Alexandre Bilodeau Parisa Darvish Zadeh Varcheie. People tracking using a network-based ptz camera. *Machine Vision and Applications,, Volume 22, Issue 4, pp 671-690*, 2011. 2

[19] Andreas Koschan Besma R. Abidi Mongi A. Abidi Sangkyu Kang, Joon-Ki Paik. Real-time video tracking using ptz cameras. In *Proceedings Volume 5132, Sixth International Conference on Quality Control by Artificial Vision*, 2003. 3

[20] Chao-Yang Lee Shou-Jen Lin Chu-Sing Yang, Ren-Hao Chen. Ptz camera based position tracking in ip- surveillance system. In *3rd International Conference on Sensing Technology*, 2008. 3

[21] Astro carrier board users guide. http://www.connecttech.com/pdf/CTIM-ASG001_Manual.pdf. 5

[22] Jetson tk1 development kit specification. http://developer.download.nvidia.com/embedded/jetson/TK1/docs/3_HWDesignDev/JTK1_DevKit_Specification.pdf. 5

[23] Jetson tx1 development kit product sheet. http://images.nvidia.com/content/tegra/embedded-systems/pdf/JTX1-DevKit-Product-sheet.pdf?ncid=pa-blo-ftrs27-3860. 5

[24] Toradex apalis tk1 module datasheet. https://docs.toradex.com/103129-apalis-tk1-datasheet.pdf. 5

[25] Toradex ixora carrier board datasheet. https://docs.toradex.com/101430-apalis-arm-ixora-datasheet.pdf. 5

[26] Matthias Mueller, Neil Smith, and Bernard Ghanem. A benchmark and simulator for uav tracking. In *Proc. of the European Conference on Computer Vision (ECCV)*, 2016. 6

[27] Anton Milan, Laura Leal-Taixé, Ian D. Reid, Stefan Roth, and Konrad Schindler. MOT16: A benchmark for multi-object tracking. *CoRR*, abs/1603.00831, 2016. 6