Transport and Communications Science Journal

BUILDING AN IMAGE PROCESSING PROGRAM FOR THE VEHICLE FIRE CONTROL SYSTEM

Xuan Tung Vu

Weapons Institute, Vietnam Defence Industry, 51, Phu Dien, Phu Dien, Bac Tu Liem, Hanoi, Viet Nam

ARTICLE INFO

TYPE: Research Article
Received: 15/05/2024
Revised: 11/06/2024
Accepted: 14/06/2024
Published online: 15/09/2024
https://doi.org/10.47869/tcsj.75.7.5

* *Corresponding author*
Email: vuxuantung0511@gmail.com

Abstract. In the world, the integration of controlled weapons into combat vehicles has been done for a long time and many weapon manufacturers have utilized image processing software to enhance the combat effectiveness of weapon systems, resulting in positive outcomes. Fire control systems, especially fire control systems on vehicles, require requirements for processing speed, durability as well as flexibility, which are essential when fighting the enemy. Kernelized Correlation Filters image processing algorithms and Temple Matching algorithms have promoted the advantages of image processing with vehicle fire control system in the weapons field. In this article, from the analysis of the Kernelized Correlation Filters image processing algorithm and the Temple Matching algorithm on hardware platforms suitable for vehicle fire control systems, the authors built a software program based on taking advantage of the powerful parallel computing capabilities of GPUs applied to 12.7 mm gun fire control systems installed on vehicles. The experiments demonstrated the results of handling targets in the field after completely installing all components of the weapon complex on the vehicle.

Keywords: Board Jetson AI, Orange Pi 5, Kernelized Correlation Filters, Temple Matching, Linear Correlation Filter.

1. INTRODUCTION

For real-time image processing problems, it is common to use an Intel Core I CPU or an AMD Ryzen CPU that has enough computing power. However, this option is not suitable when installed on a vehicle because these platforms consume a lot of energy, have difficulty dissipating heat, and require a lot of space to install [1]. Currently, hardware devices for mobile systems mainly use ARM CPUs, which have many advantages such as low energy consumption, improved processing speed, compactness, resistance to impact, and low heat emission when operating. Moreover, they have a widespread application development community, are easy to purchase, have many support tools, and are highly inheritable [1], [2]. NVIDIA's Jetson AI board [3] is often chosen to perform image processing problems on mobile or outdoor systems [3,4], and in particular, the Orange Pi 5 Plus Board uses OpenCL technology for parallel processing. Although it is not as convenient and powerful as CUDA technology on the Jetson board, it has the advantage of generating less heat [3].

Image processing algorithms in vehicle fire control systems can be mentioned as follows: (1) Temple Matching algorithm [5, 6] is a technique for finding regions of matching images (similar) to sample image (patch); (2) The KCF (Kernelized Correlation Filters) image object tracking algorithm [7] works according to the following general principle: the object to be tracked is usually selected by a rectangular bounding box.

In terms of weapons systems, the installation of controlled weapon systems on combat vehicles is much more difficult because the space is small and difficult to install, and providing enough stable energy for the electrical system to operate is difficult. Because the space in the car is tight and cramped, finding a heat dissipation solution for electronic components is not simple. The vehicle's operating conditions include vibrations and exposure to sunlight and rain, so ensuring the system is stable and accurate is not easy.

In this paper, the authors propose a solution to build an image processing program (according to Temple Matching and KCF algorithms) for the vehicle's fire control system. The computer that integrates this program must be compact, impact-resistant, and have little heat loss, but must still have enough computing power to process camera images in real-time.

Recent studies have shown significant advancements in GPU-based image processing for real-time applications. For instance, in publication [8], the authors demonstrated the use of heterogeneous computing and edge computing to accelerate anomaly detection in multispectral images, highlighting the efficiency of GPU-based solutions in processing large datasets. Similarly, in publication [9] and [10], the authors explored fast 2D and 3D image processing with OpenCL, emphasizing the potential of GPU acceleration in various imaging tasks. These advancements provide a broader context for the current research, underscoring the viability of GPU-based image processing in embedded systems.

2. RELATED WORKS

2.1. Selecting image processing algorithms in building image processing applications on fire control computers

Figure 1 presents the details of the image processing algorithm flow chart, in which the central part of this flow chart is the image processing algorithm that is parallelised on the cores of the video card on the embedded board. Here, the authors integrate two image processing algorithms for experimental evaluation: Temple Matching and KCF. Each algorithm has different advantages and disadvantages, suitable for each different testing situation, so we decided to put both options on the central control software.

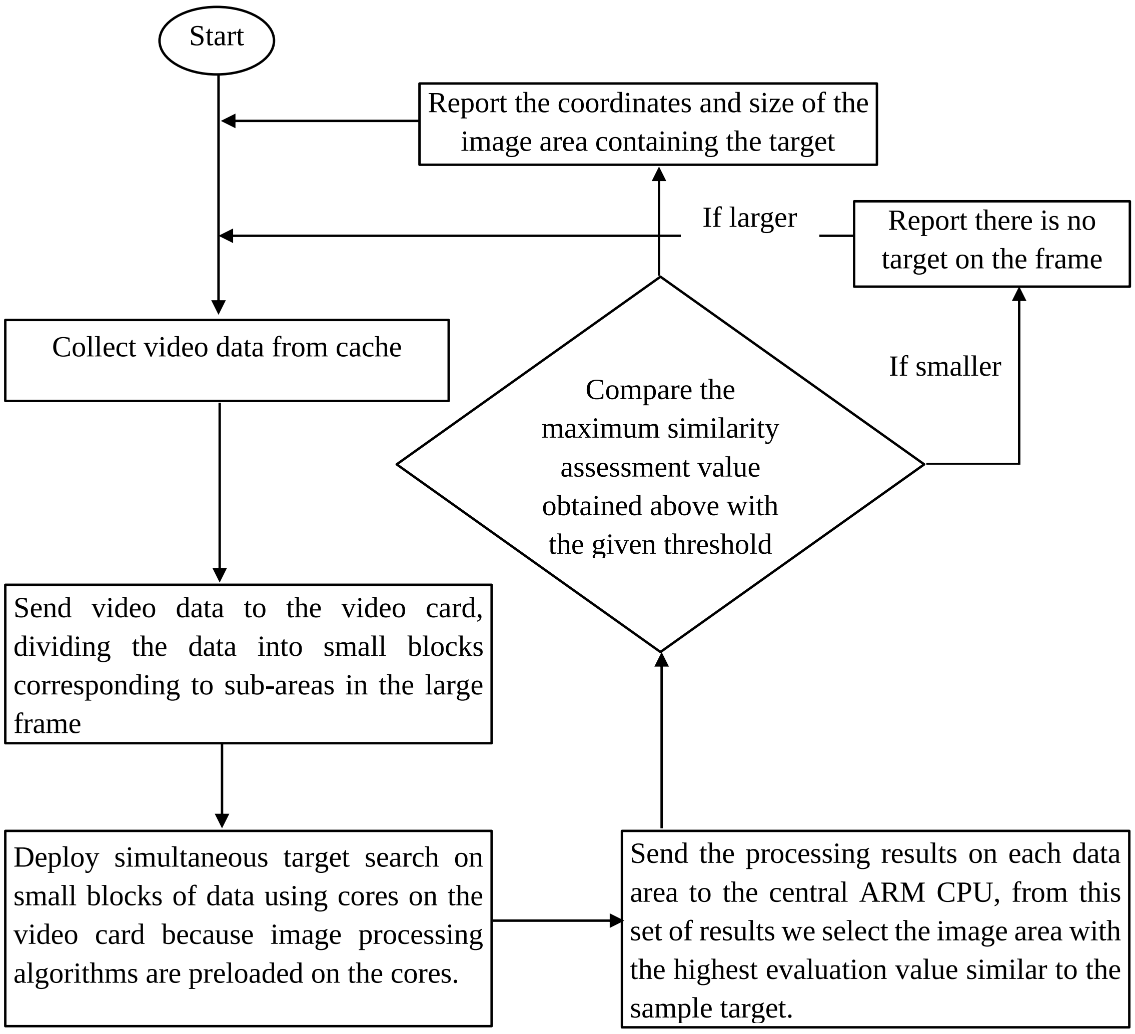
Start
Report the coordinates and size of the image area containing the target
If larger
Report there is no target on the frame
Collect video data from cache
If smaller
Compare the maximum similarity assessment value obtained above with the given threshold
Send video data to the video card, dividing the data into small blocks corresponding to sub-areas in the large frame
Deploy simultaneous target search on small blocks of data using cores on the video card because image processing algorithms are preloaded on the cores.
Send the processing results on each data area to the central ARM CPU, from this set of results we select the image area with the highest evaluation value similar to the sample target.

Figure 1. The flow chart of image processing algorithm.

1) Temple Matching image processing algorithm [5]: is a technique for finding regions of an image that match (similar) to a sample image (patch), in other words, fitting a puzzle piece to the actual image by "sliding" that puzzle piece onto the input image (like 2D convolution) and compare samples and fragments based on some normalisation (Table 1). It returns a grayscale image, where each pixel represents how well its neighbors match the sample.

Operating principle: If the input image has size (WxH) and the sample image has size (wxh), the output image will have size (W-w+1,H-h+1). After receiving the result, the minMaxLoc() function is used to find where the maximum/minimum value is. This value it taken as the top left corner of the rectangle and (w,h) is taken as the width and height of the rectangle. That rectangle will be the sample area to search. Extracting and finding the maximum/minimum value is performed based on the correlation value functions and corresponding mathematical formulas in Table 1.

Table 1. Details of correlation value functions used in the Temple Matching algorithm.

Function	Formula
TIM_SQDDIFF	$R(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2$
TIM_SQDDIFF_NORMED	$\mathrm{R(x,y)} = \frac{\sum_{x',y'}(T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2}\sqrt{\sum_{x',y'} I(x+x',y+y')^2}}$
TM_CCORR	$R(x,y) = \sum_{x',y'} (T(x',y').I(x+x',y+y')$
TM_CCORR_NORMED	$\mathrm{R(x,y)}= \frac{\sum_{x',y'}(T(x',y').I(x+x',y+y')}{\sqrt{\sum_{x',y'} T(x',y')^2}\sqrt{\sum_{x',y'} I(x+x',y+y')^2}}$
TM_CCOEFF	$R(x,y) = \sum_{x',y'} (T'(x',y').I'(x+x',y+y'))$
TM_CCOEFF_NORMED	$\mathrm{R(x,y)}= \frac{\sum_{x',y'}(T'(x',y').I'(x+x',y+y')}{\sqrt{\sum_{x',y'} T'(x',y')^2}\sqrt{\sum_{x',y'} I'(x+x',y+y')^2}}$

In this paper, each segmented image region will generate a value by one of the above list of Temple Matching functions and this process is calculated in a core of the video card. The advantage of this process in the Temple Matching algorithm is that it can use multiple Graphics Cards. The more cores the card has, the greater the number of Temple Matching functions that are calculated simultaneously, and thus it will save processing time in one cycle of collecting an image frame.

2) KCF image processing algorithm [7]: The basic idea of correlated filter tracking is to estimate an optimal image filter such that filtering with the input image produces the desired response. The desired response typically has a Gaussian shape centered on the target location, so that the score decreases with distance. The filter is trained from translated (shifted) versions of the target patch. On testing, the filter response is evaluated and the maximum gives the new position of the target. The filter is trained online and continuously updated with every frame so that the tracker adapts to moderately sized target changes. The main advantage of the correlation filter tracker is computational efficiency. The reason is that the calculation can be performed efficiently in the Fourier domain. Therefore, the tracker runs in super real-time, several hundred FPS. There are both linear and non-linear (Kernel) versions of the tracker derived from Henriques' uniform least squares principle. The operation of this KCF image processing algorithm is mainly shown in the following modules: (1) Linear Correlation Filter; (2) Kernelized Correlation Filter.

(1) Module 1 (Linear Correlation Filter):

The optimal linear filter w is found by solving the least squares problem after normalization:

$$min_w(||Xw - y||^2 + \lambda||Xw - y||)^2 \qquad (1)$$

Where, X is the cyclic matrix of the input image. The rows of X store all possible cyclic image shifts, y is the sample target image, λ is the regularizer weights. Then, the equation of (1) is rewritten as following:

$$\mathrm{w} = w = (X^T X + \lambda \mathrm{I})^{-1} X^T X \qquad (2)$$

Since X is a cyclic matrix, w in (2) can be quickly calculated by operations in the Fourier domain

$$w = \frac{\hat{x} \otimes \hat{y}}{\lambda + \hat{x} \otimes \hat{y}} \qquad (3)$$

In which, "^" is the complex Fourier value; "⊗" represents multiplication by components;

Furthermore, the filter response on the test image is not calculated using a sliding window but can be done more efficiently by the following method

$$\hat{r} = \hat{z}^* \otimes \widehat{w} \qquad (4)$$

In the LCF linear correlation filter, the filter has the disadvantage of accuracy, which will not be as high as the kernel correlation filter because this filter is represented in a nonlinear form, so it characterizes the object better. In this paper, to overcome this drawback, the authors took advantage of the LCF linear correlation filter's advantage of being fast to calculate because the filter is represented in linear form. This helps improve the performance of the KCF image processing algorithm.

<u>(2) Module 2 (Kernelized Correlation Filter):</u>

The "kernel-trick" is used by mapping the input data to a non-linear functionx−>φ(x) and representing the solution as a linear combination $\mathrm{w} = \sum_{i=1}^{n} \alpha_i \varphi(x_i)$. Then, the algorithm needs to find

$$min_\alpha(||K\alpha - y||^2 + \lambda \alpha^T K \alpha) \qquad (5)$$

In which, matrix K is the kernel matrix with convolutional elements $k_{i,j} = \varphi(x_i)^T \varphi(x_i)$. Therefore, the original problem becomes non-linear. The method to find the result for equation (5) is understood as:

$$\alpha = (K + \lambda \mathrm{I})^{-1} y \qquad (6)$$

It can be effectively calculated by

$$\dot{\alpha} = \frac{\dot{y}}{\hat{k}^{xx} + \lambda} \qquad (7)$$

and with fast detection response

$$\hat{r} = \widehat{k}^{xx} \odot \hat{\alpha} \qquad (8)$$

Then, by substitution $k^{xx'} = F^{-1}(\hat{x}^* \odot \hat{x}')$ into (7) and (8), We will obtain the results of the linear correlation tracker (3) and (4)

The KCF correlation filter has the disadvantage that the calculation time is more than the linear correlation filter because the representation function is more complicated. This is a weakness that reduces the execution efficiency of the KCF algorithm. In this article, the authors have overcome it by using functions (7) and (8) to produce more accurate results than the linear correlation filter, this is shown in equation (3) and (4).

(3) Detailed Implementation of Algorithms:

KCF Algorithm:

The KCF algorithm estimates an optimal image filter such that filtering with the input image produces the desired response. The filter is trained online and continuously updated with each frame, ensuring the tracker adapts to moderate changes in the target. The implementation involves the following steps:

- ✓ Initialization: The target is selected using a rectangular bounding box.
- ✓ Training: The filter is trained from translated versions of the target patch using the least squares method.
- ✓ Detection: The filter response is evaluated, and the maximum response gives the new position of the target.

Temple Matching Algorithm:

The Temple Matching algorithm finds regions of an image that match a sample image by sliding the sample over the input image and comparing regions based on normalization. The implementation involves:

- ✓ Input Preparation: The input image and sample image are prepared.
- ✓ Matching: The sample image is slid over the input image, and correlation value functions (e.g., TIM_SQDDIFF, TM_CCORR) are used to find matching regions.
- ✓ Result Extraction: The minMaxLoc() function identifies the top left corner of the matching region, and the sample area is defined based on this location.

2.2. Selecting image processing hardware solution for fire control computer

With the disadvantage of computing power, it cannot be compared to Intel Core I CPU systems or AMD Ryzen CPU systems, although they are continuously improved each year by increasing clock speeds, increasing cache capacity and adding more cores ARM [1]. If we only take advantage of the power of ARM CPUs, we cannot ensure that the image processing problem of the fire control computer can operate in real time. With the advent of ARM CPU systems, NVIDIA's CUDA tool and OpenCL ([2], [8], [9], [10]), the ability to integrate on mobile embedded boards using ARM CPUs has become helps solve image processing problems. The GPU (Graphic Process Unit) graphics processing chip is essentially a combination of many single processing cores (the number can range from 128 to 2048 units). Each core is a separate CPU whose processing speed can range from 600 Mhz to GHz. Although each GPU core also has its own amount of resources such as cache and RAM, it cannot compare in terms of processing power of each ARM core of the CPU (clocking up to > 2Ghz). Because the number of GPU cores is very large, the total amount of computation the GPU can handle is much larger than the total amount of computation that the ARM CPU cores can produce. Therefore, theoretically if we could share the computation part of the program Image processing for GPUs while ARM cores only play the role of data coordination and peripheral communication, the program can completely operate in real time

Currently, with the support of NVIDIA's Jetson AI board [4], it helps to perform image processing problems on mobile or outdoor systems more conveniently. This board has many advantages such as powerful hardware resources, receiving a lot of support from NVIDIA and a large development community, so application development has many advantages. Besides the advantages, there are many disadvantages such as high cost and high heat dissipation, so dissipating heat for them is difficult, especially in cramped environments. Furthermore, it will be quite difficult to buy large quantities of high-end boards, so there are many shortcomings when applying on the vehicles.

In this paper, to solve the problem of image processing on a fire control computer, the authors used Jetson boards to test and evaluate the algorithm to save time. Then, to overcome the disadvantages of the algorithm operating on the Jetson board, the authors chose an alternative solution using the Orange Pi 5 Plus board. This board uses OpenCL technology for parallel processing. Although it is not as convenient and powerful as CUDA technology on the Jetson board (but still meets the requirements of real-time image processing), it radiates less heat and is more affordable and easier to buy in larger quantities than the Jetson Board at the same level of processing capacity.

2.3. Experimental setup and hardware configuration

The experiments were conducted using two different hardware platforms: Nvidia Jetson Orin NX and Orange Pi 5. The Nvidia Jetson Orin NX features a powerful GPU with 1024 CUDA cores and an ARM Cortex-A78AE CPU, while the Orange Pi 5 uses an ARM Cortex-A55 CPU and supports OpenCL for parallel processing. Both platforms were equipped with 8GB of RAM. The image processing program was tested under real-world conditions, including target detection and tracking in outdoor environments. The targets included a UAV drone and a mobile shooting target. The performance of the algorithms was evaluated based on processing time and target accuracy.

3. RESULTS AND DISCUSSION

Both KCF and Temple matching algorithms are performed in real time which means the target detection process is performed while waiting for the next image frame to be collected. Two tables are presenting the evaluation of the processing performance of the two algorithms on different hardware platforms. We used algorithm evaluations on outdoor scenes to increase realism when used in practice. Here, the assessment of catching a UAV drone and tracking a moving target as a shooting target on the field will be conducted (Figure 2).

Figure 2 shows the results in the situation of detecting and tracking Drone targets using the KCF algorithm on two hardware platforms Orange Pi 5 and Nvidia Jetson Orin NX. This target-tracking imaging program system is installed and controls firepower on the vehicle. To evaluate the quality and performance of the program during targeted combat, the authors conducted experiments to evaluate the effectiveness of the program in different areas and terrains.

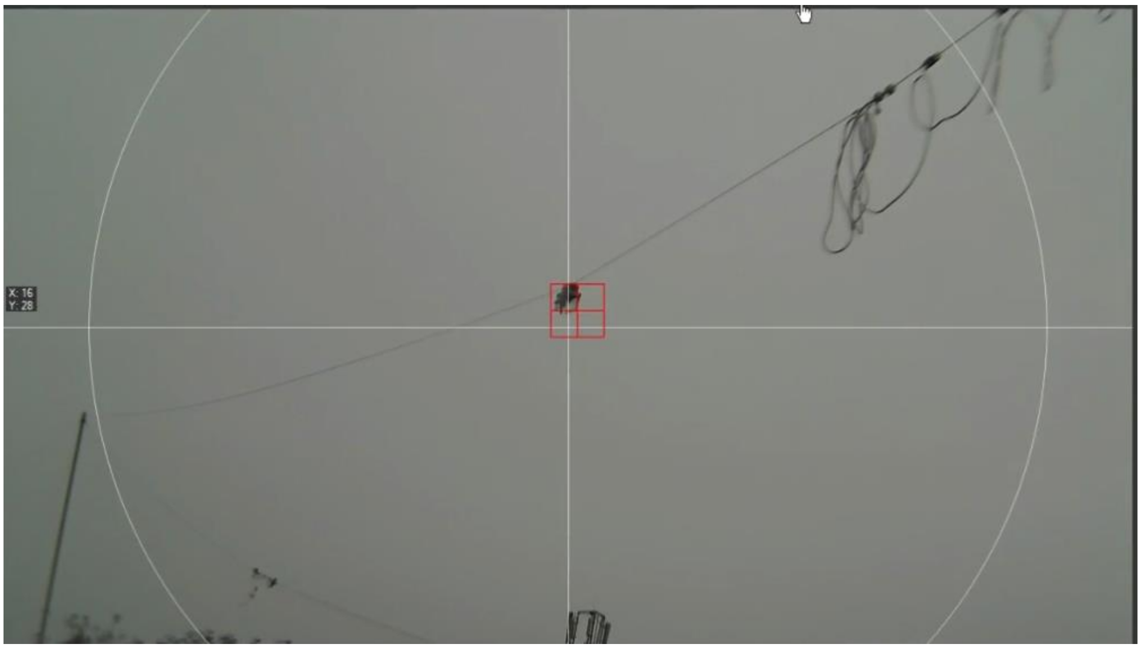

Figure 2. Results of displaying Drone tracking using KCF algorithm on two hardware platforms Orange Pi 5 and Nvidia Jetson Orin NX.

Table 2. Compare the performance of two algorithms on the Nvidia Jetson Orin NX platform.

N0	**KCF algorithm**			**Temple Matching algorithm**		
	Input sample	**Processing time (ms)**	**Target accuracy (Pixel, Pixel) in two axes (x, y)**	**Input sample**	**Processing time (s)**	**Target accuracy (Pixel, Pixel) in two axes (x, y)**
1	Drone	5.1	(0,1)	Drone	4.1	(0,0)
2	Drone	5.2	(1,0)	Drone	4.2	(0,0)
3	Drone	5.0	(1,1)	Drone	4.3	(0,0)
4	Drone	5.2	(1,1)	Drone	4.1	(0,0)
5	Drone	5.3	(1,1)	Drone	4.3	(0,0)
6	Target	5.2	(1,0)	Target	4.1	(1,0)
7	Target	5.1	(1,0)	Target	4.1	(1,0)
8	Target	5.0	(1,1)	Target	4.2	(0,0)
9	Target	5.3	(1,1)	Target	4.25	(0,0)
10	Target	5.2	(0,1)	Target	4.3	(0,1)

The standard for evaluating a well-functioning image processing program is that the target image does not lose focus during its existence on the screen and that the center of the target frame does not fluctuate when the target is stationary (to apply the problem of aiming and shooting). In particular, the authors installed both the KFC algorithm and the Temple Matching algorithm on two hardware platforms, Orange Pi 5 and Nvidia Jetson Orin NX, to evaluate the processing quality of the program.

Table 2 shows detailed performance results of the KCF algorithm and the Temple Matching algorithm on the Nvidia Jetson Orin NX platform when the image processing program for the fire control system is deployed. The results show that the image processing program on the author's vehicle fire control system operates well and stably. For the KCF algorithm, the target tracking time is about 5.0 (ms) to 5.3 (ms) with high accuracy. For the

Temple Matching algorithm, the target tracking time is about 4.1(s) to 4.3(s) with high accuracy.

Table 3. Compare the performance of two algorithms on the Orange Pi 5 platform.

N0	KCF algorithm			Temple Matching algorithm		
	Input sample	**Processing time (ms)**	**Target accuracy (Pixel, Pixel) in two axes (x, y)**	**Input sample**	**Processing time (s)**	**Target accuracy (Pixel, Pixel) in two axes (x, y)**
1	Drone	15.3	(1,0)	Drone	13.2	(0,0)
2	Drone	15.2	(1,1)	Drone	13.0	(0,0)
3	Drone	15.0	(1,1)	Drone	13.1	(0,0)
4	Drone	15.2	(1,1)	Drone	13.1	(0,0)
5	Drone	15.3	(1,0)	Drone	13.3	(0,0)
6	Target	15.0	(0,1)	Target	13.1	(1,0)
7	Target	15.25	(1,0)	Target	13.1	(1,0)
8	Target	15.2	(1,1)	Target	13.2	(0,0)
9	Target	15.3	(1,1)	Target	13.25	(0,0)
10	Target	15.2	(1,0)	Target	13.3	(0,1)

Table 3 shows detailed results of the operation of the KCF algorithm and the Temple Matching algorithm on the Orange Pi 5 platform when the image processing program for the fire control system is deployed. The results show that the image processing program on the author's vehicle fire control system works well and stably. For the KCF algorithm, the target tracking time is about 15.0 (ms) to 15.3 (ms) with high accuracy. For the Temple Matching algorithm, the target tracking time is about 13.0(s) to 13.3(s) with high accuracy. This was tested by the authors to evaluate operating algorithms with two objects: Drones and mobile shooting targets, specifically:

- ✓ ***KCF algorithm on two platforms: Nvidia Jetson Orin NX and Orange Pi 5***

The objects are drones: The drone object moves undulating (non-linear trajectory) and rotates around itself (object shape is changed) but is still firmly tracked by the software throughout the object's existence on the screen, but the target frame centre fluctuates by 1 pixel when the subject is stationary, and the software will not automatically detect the target when it reappears after it exits the frame.

The objects are mobile shooting targets: The object moves linearly and sometimes enters a partially obscured area but is still tracked firmly by the software throughout the object's existence on the screen, but the target frame centre fluctuates by 1 pixel when the object is exposed. A stationary object will not be automatically detected by the software when it reappears after it leaves the camera's field of view.

- ✓ ***Temple Matching algorithm on two platforms: Nvidia Jetson Orin NX and Orange Pi 5***

The objects are drones: When the drone object moves undulating (non-linear trajectory) and does not rotate around itself, it is still firmly tracked by the software even when the object reappears in the camera's field of view. At the same time, the centre of the frame does not fluctuate when the object is stationary, but when the object rotates around itself (the object's shape is changed), the software captures the object very poorly and does not meet the requirements.

The objects are mobile shooting targets: If the object moves linearly and preserves its original shape, it is still firmly tracked by the software even when the object reappears in the camera field of view. At the same time, the target frame does not fluctuate when the object is stationary, but when the object changes its original shape while appearing in the camera's field of view, the results of tracking the object are very poor and do not meet the requirements.

To provide a comprehensive comparison, the processing times of the KCF and Temple Matching algorithms were also evaluated on traditional CPU systems such as Intel Core I and AMD Ryzen. The results are shown in Table 4.

Table 4. Comparison of processing times on traditional CPUs.

Algorithm	**Platform**	**Processing Time (ms)**
KCF	Intel Core I	10.5 - 12.0
KCF	AMD Ryzen	9.8 - 11.2
Temple Matching	Intel Core I	6.1 - 7.3
Temple Matching	AMD Ryzen	5.9 - 6.8

4. CONCLUSION

The author of this paper has conducted an analysis of two algorithms, highlighting their respective advantages and disadvantages. Both algorithms have distinct strengths and weaknesses, and it would be unwise to entirely discard either one. This is because there are numerous situations in the field, and each algorithm is better suited for certain types of scenarios. The results achieved by conducting experiments are shown specifically:

The KCF algorithm is suitable for situations where the object changes its original shape, but when used to destroy the target with many different shots, the user must choose again. Because the camera shakes between shots, the software cannot track the target. When the system stabilises, the algorithm does not automatically detect the target.

The Temple Matching algorithm is well-suited for scenarios where the object being tracked maintains its original shape. This algorithm is capable of firing multiple rounds of bullets at a target without the need to reselect it. Additionally, due to the target frame's stability during the tracking process, the probability of successfully destroying the target is higher with the Temple Matching algorithm compared to the KCF algorithm.

ACKNOWLEDGMENT

The authors acknowledge the Weapons Institute-Vietnam Defence Industry for supporting this work.

REFERENCES

[1]. S. J. Lee, Challenges of Real-time Processing with Embedded Vision for IoT Applications, in 2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), (2022) 1–6. https://doi.org/10.1109/ICECCME55909.2022.9988338.

[2]. I. Rodriguez-Ferrandez, L. Kosmidis, M. M. Trompouki, D. Steenari, F. J. Cazorla, Evaluating the Computational Capabilities of Embedded Multicore and GPU Platforms for On-Board Image Processing, in 2023 European Data Handling & Data Processing Conference (EDHPC), (2023) 1–7.

https://doi.org/10.23919/EDHPC59100.2023.10395928.

[3]. J. Suder, K. Podbucki, T. Marciniak, Power Requirements Evaluation of Embedded Devices for Real-Time Video Line Detection, Energies, 16 (2023) 6677. https://doi.org/10.3390/en16186677.

[4]. M. Barnell, C. Raymond, S. Smiley, D. Isereau, D. Brown, Ultra Low-Power Deep Learning Applications at the Edge with Jetson Orin AGX Hardware, in 2022 IEEE High Performance Extreme Computing Conference (HPEC), (2022) 1–4. https://doi.org/10.1109/HPEC55821.2022.9926369.

[5]. R. Brunelli, Computational Aspects of Template Matching, in Template Matching Techniques in Computer Vision, Wiley, (2009) 201–219. https://doi.org/10.1002/9780470744055.ch10.

[6]. Y. ABE, T. Fukuda, M. Shikano, F. Arai, Y. Tanaka, Vision Based Navigation System for Autonomous Mobile Robot. Locomotive Experiments Based on Variable Template Matching, JSME International Journal Series C, 43 (2000) 408–414. https://doi.org/10.1299/jsmec.43.408.

[7]. J. F. Henriques, R. Caseiro, P. Martins, J. Batista, High-Speed Tracking with Kernelized Correlation Filters, IEEE Transactions on Pattern Analysis and Machine Intelligence, 37 (2015) 583–596. https://doi.org/10.1109/TPAMI.2014.2345390.

[8]. J. López-Fandiño, D. B. Heras, F. Argüello, Using heterogeneous computing and edge computing to accelerate anomaly detection in remotely sensed multispectral images, The Journal of Supercomputing, (2024). https://doi.org/10.1007/s11227-024-05918-z.

[9]. D. O. Dantas, H. Danilo Passos Leal, D. O. B. Sousa, Fast 2D and 3D image processing with OpenCL, in 2015 IEEE International Conference on Image Processing (ICIP), (2015) 4858–4862. https://doi.org/10.1109/ICIP.2015.7351730.

[10].R. S. Dehal, C. Munjal, A. A. Ansari, A. S. Kushwaha, GPU Computing Revolution: CUDA, in 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN*)*, (2018) 197–201. https://doi.org/10.1109/ICACCCN.2018.8748495.