# High Performance Automatic Target Recognition

Misiker Tadesse[#], Eneyew Adugna[#]

[#]*School of Electrical and Computer Engineering, Addis Ababa University*
*Addis Ababa, Ethiopia*

[1]misiker@aait.edu.et

[2]eneyew.adugna@aait.edu.et

*Abstract*— **Designing a vision system, which was motivated by that of the human eye, has been done since the introduction of digital computing devices. Its computational complexity hinders it from the required accuracy and flexibility achievable by these systems. A real-time Automatic Target Recognition system having the ability of detection, identification and tracking of pre-specified targets even with bad visual artifacts is developed. This work investigates an improved combination of detection and recognition algorithms to reach a better solution to the problem of target detection and recognition while aiming to fit the design of a minimal system. A speed up of 2.4 is achieved, by using a non-preeminent GPU, over brute force implementations.**

***Keywords*— Target detection, heterogeneous computing, GPU, Background subtraction, Target recognition**

## I. INTRODUCTION

Automatic Target Recognition(ATR) can be referred to as detecting and recognizing targets in a sensor data using digital signal processing. ATR has evolved to be a very important part of intelligent systems both for military and civil applications. These applications operate on streams of data and hence are typically characterized by real-time performance requirements. Various improvements have been done and are being done from various vantage points. Signal processing algorithms, processor technology, sensor technology improvements have a great impact on the evolution of ATR. Processor improvement has brought with it parallel architectures which are suited to the application of signal processing for applications which are data intensive. ATR falling in this category can take advantage of the parallelism in the processors. A successful teal-time ATR algorithm must meet performance (timing) and accuracy requirements imposed on it for tracking objects of concern irrespective of the external environmental conditions. Noise, occlusion and change in lighting intensity are prominent challenges to ATR algorithms. Intermediate stages in ATR such as noise cancellation operations and intensive recognition operations are trade-offs to speed. To compensate for the speed drop due to the added features on the algorithm, a parallel computing chip particularly GPU is used to assist ATR algorithm to accelerate the process of recognition and related intensive computations. Not every phase in the ATR algorithm is able to be executed in parallel. The essence is to redesign serial implementation of the algorithm to run on GPU to achieve a better speedup of the whole system.

## II. BACKGROUND THEORY

In this section terms and concepts used in this thesis are thoroughly discussed.

A. ***Automatic Target Recognition*** The basic principle behind ATR is detecting and recognizing specific items of interest (or targets) in an image gathered from complex background (or clutter) by a non-flawless sensor, which inevitably introduces noise in the gathered signal. Main challenges in ATR can be categorized in to: elimination of noise from a noisy signal gathered or extracting targets from a complex surrounding. ATR problems generally include a front-end target detection (or segmentation) stage [4]. The main aim of the detection stage is to minimize the amount of data on which the process of target recognition is done. This means detection refers to finding a probable target of interest which needs to be verified in the recognition stage. The target detection stage minimizes the number of computation required while making sure not to miss targets of interest. Figure 1 shows the elimination of background clutter and non-target clutter in an ATR pipeline.

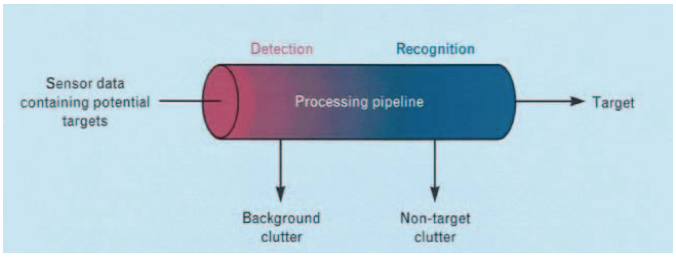


Figure 1: Conceptual data flow in ATR systems [4]

Generally the ATR algorithm can be divided in to five major steps: Sensing, Detection/ Segmentation, Feature Extraction, Target Recognition, and Post-Recognition processing.

B. ***Segmentation:*** Segmentation is the first step in any image analysis and target recognition application. It is most essential and most difficult task and determines the performance of the whole system. It is basically dividing the image into different regions which are heterogeneous, and any two regions in the same category are homogeneous. If $Homogineity()$ is a homogeneity predicate defined on a group of connected pixels, then segmentation is a partition of the set F in to connected subset or regions $\{S_1, S_2, \dots, S_n\}$, such that:

$$\bigcup_{i=1}^{n} S_i = F \,, with\, S_i \cap S_j = \emptyset \;\; (i \neq j) \qquad (Eq.1)$$

The homogeneity predicate $Homogeneity(S_i) = True$ for all segmented homogeneous region,$S_i$ and$Homogeneity(S_i \cup S_j) = Flase$, where$i \neq j$, and $S_i$ and $S_j$ are adjacent regions. For target recognition applications the subset/regions are 2, the target and the background clutter.

C. ***Parallelism in computation*** Several types of parallelism are performed in computing. It can be classified as: Instruction level parallelism, Task level parallelism, data level parallelism [7]. While implementing parallel algorithms, the most essential part to consider is data dependencies between the operations. The instruction in an algorithm or part of an algorithm can be performed in parallel, if there is no data dependency between each other, as shown in Figure 2.

One approach for accelerating computationally intensive tasks is to transfer some or all of the computations to dedicated hardware such as Field Programmable Gate Array (FPGA) or Graphics Processing Units (GPU). FPGAs are essentially field programmable logic devices where any logic function can be implemented by synthesizing the design on the device. This FPGA approach was very successfully applied to template matching speed up in [3] and to the acceleration of various signal processing operations [4]. In spites of the achievable performance gains, FPGAs are often expensive and involve highly complex implementation. Apart from that, FPGAs implementation lack the flexibility requirement set by some applications as its most parameters are fixed at the compile time of the design.

GPU is a reduced cost speed up alternative to FPGAs. Driven by the ever increasing market demand for realistic 3D games, GPU devices has evolved into a highly parallel, multithreaded, multi-core processing units with enormous computational power. In terms of Floating Point

**978-1-4799-7498-6/15/$31.00 ©2015 IEEE**

Instruction Per Second (FLOPS) or Giga FLOPS (GFLOPs), modern commodity GPUs are overshadowing CPU performance [18].

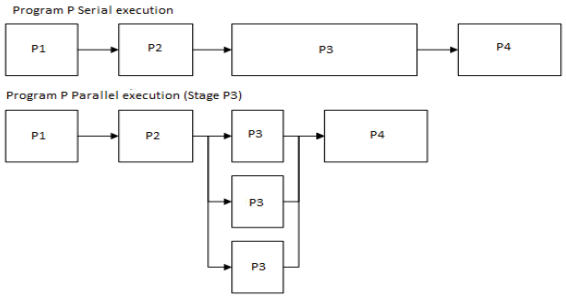


Figure 2: Performance improvement executing part of algorithm in parallel [7]

The computational capability of GPUs have been growing at average yearly rate of 1.7 (pixels/second) to 2.3 (vertices/ second), which is a significant margin over the average yearly rate of roughly 1.4 for CPU performance [18].

## III. PREVIOUS WORKS

Various researches have been done to improve the performance of automatic recognition, in terms of algorithm improvements and hardware software co-processing. This section summarizes the various works that have been done in the area of target recognition.

**Algorithmic Improvement**

Algorithmic improvements in ATR have been done through improving of one or more of the algorithms in the pipeline, for the overall improvement of one of the metrics of ATR.

***Segmentation*** One popular method that is developed in this context is the watershed algorithm [6]. This algorithm approaches the problem by converting lines in an image into "mountains" and uniform regions into "valleys" that can be used to help segment objects. Another popular work, Grab-cut algorithm, uses an iterative Background/foreground segmentation grouping the pixels into clusters of similar colors and introducing boundaries between foreground and background pixels. Grabcut unlike mathematical morphology based algorithms [6][15] has a high accuracy but is computationally an expensive algorithm. Mean Shift Segmentation [15] uses nonparametric estimator of density to image segmentation. Mean shift algorithm has significant limitation if the local characteristics of the pixels vary considerably across the domain; it is difficult to select common optimal influence region [15]. As a result of this, in a segmented image with varying color, some objects may appear too coarse while others are too fine.

***Hardware software co-processing***

The performance in the improvement of ATR attained using hardware co-processing is dealt with terms of different metrics of computation: Speed [8][11][12][13][14], Accuracy of result[10][11][14], power efficiency[9]. In [9], an FPGA based face recognition system which is reasonably power efficient than floating point architecture and can be employed for portable applications was designed using Eigenvalues. It shows a significant amount of reduction in power dissipation at the cost of a slight decrement in the accuracy of the algorithm [9]. In [12] implementation of a template matching algorithm on Synthetic Aperture Radar (SAR) was developed at Myricom for performance improvement. The system was migrated to the multiple, parallel, FPGA computing nodes connected by Myrinet. A linear systolic implementation of a partly scalable ATR algorithm is mapped to the Splash 2 FPGA platform[13]. The column oriented processors with distributed Splash 2 memories were used throughout the design to achieve the high performance and the scalability of the system.

Since the emergence of general purpose GPU, the computer vision industry has transferred its attention to the processing power of GPU devices to fulfil the ever increasing computational requirement of computer vision applications. In [14] a real time computer vision based power line extraction solution is considered for active Unmanned Aerial Vehicle (UAV) guidance. A collinear line segments fitting algorithm considering global and local collected data together with multiple collinear measurements. NVIDIA GPU is used to speed up the implementation of the algorithm to outperform previous baseline line detection algorithms, for real time application. GPU-based implementation of target and anomaly detection algorithms for hyperspectral data analysis has been discussed in [19] and [20].

## IV. SYSTEM DESIGN AND IMPLEMENTATION

In this section the methodology followed for the proposed work is discussed. The C++ programming language is used for this work, and because of its processing speed, Open Computer Vision Library (OpenCV) framework is used for the implementation. The algorithm starts after an RGB image sequence is obtained from video source, video file or digital camera. The algorithm for this work incorporates: Capturing Frame, Object Detection/Segmentation, Target Recognition and Target Tracking. Capturing frame involves accepting input video from camera or input video file. The segmentation phase minimizes the search area for the recognition phase by getting the region of interest (ROI) on the captured frame. The recognition phase then examines ROI's detected from the segmentation phase by comparing with predefined target templates. The target template used is a uniquely identifying feature of the particular target. The target template, which basically can be either a shape based or color based identifying feature of the target, is compared with the potential targets using template matching algorithm. The target template used for this work is a composite template which is a combination of the color feature of the target and the shape feature of the target.

**Target Detection**

This section focuses on how to isolate objects or parts of objects from the rest of the image. For efficient resource utilization the region of interest has to be detected before executing further otherwise unnecessary processing will be done. For this work the target detection is a composite process which includes color segmentation and background subtraction; which separately are weaker algorithms, but when combined give better performance.

***Color Segmentation***

*RGB to HSV color space*

The conversion from the RGB color space to the HSV is done using the following equations:

$$h' = \begin{cases} \frac{g-b}{\delta} \; if \; r = max \\ \frac{2+(b-r)}{\delta} \; if \; g = max \\ \frac{4+(r-g)}{\delta} \; if \; b = max \end{cases} \quad \text{(Eq. 2)}$$

$$v = \max \quad \text{(Eq. 3)}$$

$$s = \frac{max - min}{max} \quad \text{(Eq. 4)}$$

$$h = 0.167 * h' \quad \text{(Eq. 5)}$$

Where: max = max(r,g,b)
min = min(r,g,b)
δ =max - min

For $r, g, b \in [0 \ldots 1]$, Equation 3.1 – Equation 3.4 gives the corresponding result $h, s, v \in [0 \ldots 1]$. The hue parameter of the target template contains the minimum and maximum hue values (<Minimum_Hue, Maximum_Hue> pair) contained in the target. This range could be small or large depending on the color content of the target. Usually targets are composed of a single color; for those situations the Hue range will be very narrow. The Range (Frame) operation will give a bitmap, which has for each pixel: For $r, g, b \in [0 \ldots 1]$, Equations 2 – 5 give the corresponding result $h, s, v \in [0 \ldots 1]$. The hue parameter of the target template contains the minimum and maximum hue values (<Minimum_Hue, Maximum_Hue> pair) contained in the target. This range could be small or large depending on the color content of the target. Usually targets are composed of a single color; for those situations the Hue range will be very narrow. The Range (Frame) operation will give a bitmap, which has for each pixel:

$$Hue_{Frame} = Range\,(Frame) \quad \text{(Eq. 6)}$$

$$Hue_{Frame(x,y)} = \begin{cases} 1, & if\ Minimum_{Hue} < Frame(x,y) < Maximum_{Hue} \\ 0, & Otherwise \end{cases}$$

The discontinuity in the Hue component of the HSV image on Red (around 0 and 360°) which is at the division where Hue 'rolls over', is dealt with by always setting hue values to be within the 0-360° range using modulus arithmetic operation[16]. So, whenever the hue range being considered contains the discontinuity, the above operation is modified as:

$$Hue\_Frame(x,y) = \begin{cases} 1, & if\ Minimum_{Hue} < Frame(x,y) < 360 \\ & \text{or } 0 < Frame(x,y) < Maximum\_Hue \\ 0, & Otherwise \end{cases}$$

***Back Ground Subtraction***

Background subtraction is a popular motion detection technique [1]. It operates by simple image subtraction of the current frame from the background to find the foreground moving objects. This technique is very efficient for static camera configuration. But for moving camera, the background changes, this technique has to be assisted by other techniques to compensate the movement.

*Averaging background*

Averaging background method involves learning the average and standard deviation of each pixel to model the background. For this purpose the operations Accumulate() to accumulate learned background, Absolute_difference() and Threshold() operations to segment in to background and foreground models. The mean is approximated by a running average method which is given by:

$$Acc\ (x,y) = (1 - LR) * Acc\ (x,y) + LR * Image(x,y) \quad (3.8)$$

Where, *LR* is the learning rate of the algorithm

The standard deviation is given by[2]:

$$\sigma^2 = \sum_{i=0}^{N-1} (x_i - Mean(x))^2 \quad (\text{Eq. } 7)$$

The problem with this equation is that it needs making one pass through the images to compute $Mean(x)$ and then a second pass to compute $\sigma^2$. For this work however, after simple algebraic manipulations, it is approximated by:

$$\sigma^2 = \left(\frac{1}{N}\sum_{i=0}^{N-1} {x_i}^2\right) - \left(\frac{1}{N}\sum_{i=0}^{N-1} x_i\right)^2 \quad (\text{Eq. } 8)$$

Using this form, we can accumulate both the pixel values and their squares in a single pass. Then, the variance of a single pixel is found using the average of the square minus the square of the average.

*Morphological Operations*

To avoid salt and paper noises due to camera uncertainties that the extracted frame passes through noise cleaning, which involves Median Filter followed by DEED (Dilate, Erode, Erode, and Dilate) operation.

$$Median\_Filter(in, out, val):$$

$$Out(x,y) = Median(in(i,j)), for: \begin{cases} x-(v-1)/2 < i < x-(v+1)/2 \\ y-(v-1)/2 < j < y-(v+1)/2 \end{cases}$$

Next phase this image is passed through a thresholding function which labels any difference big enough as foreground object:

Threshold (extracted_frame, foreground_frame, th).

$$foreground_{(x,y)} = \begin{cases} 0\ if\ extracted & \geq th \\ 1\ if\ extracted & < th \end{cases} \quad (\text{Eq. } 9)$$

*Template matching*

Templates of a target contain templates of the targets at varying sizes and directions. Size of particular extracted frame (ROI) is used to select particular templates to correlate with the detected potential targets. For this work, target templates are produced in such a way that to contain 6 pixels extra in every direction (Top/Bottom/Left/Right) of the template. So, the output contains 13 x 13 pixel normalized correlation output Matrix whose values range from 0(total mismatch) to 1(Full match). In this work a match greater than 0.5 is accepted, in order to recognize targets which are partially obstructed. The output of the Recognition (template matching) phase is the maximum correlation result from the correlation output matrix; Depending on whose value target is either recognized or not. Algorithm 1 shows the template matching algorithm used for this work, given an input Image Frame and Target Template gives the best matching position and the normalized match quality.

***MatchTemplate***
*Input*: $Frame, Template$
*Output*: $Match_{Qualty}, Match_{Position}$
*for* $i$ *in* 0 *to* $Frame.Height - Template.Height + 1$ *do*
  *for* $j$ *in* 0 *to* $Frame.Width - Template.Width + 1$ *do*
    $Quality \leftarrow Compute(Frame, Template, Position(i,j))$
    *if* $Quality > Mathc_{Quality}$ *then*
      $Match_{Position} \leftarrow Position$
      $Match_{Qualty} \leftarrow Quality$
    *End if*
  *End for*
*End for*
*Return* $Match_{Position}, Match_{Qualty}$
*End* ***MatchTemplate***

*Algorithm 1: CPU based implementation template matching algorithm*

$\boldsymbol{Compute}(Template, Frame, Position)$
$Output \leftarrow Quality$
$Intermediate\ Variables\ temp1, temp2, temp3 \leftarrow 0$
*for* $x$ *in* 0 *to* $Template.Heght$ *do*
  *for* $y$ *in* 0 *to* $Template.Width$ *do*
    $temp1 \leftarrow temp1 + [Template(x,y) - Frame(Position.i + x, Position.j + y)]^2$
    $temp2 \leftarrow temp2 + [Template(x,y)]^2$
    $temp3 \leftarrow temp3 + [Frame(Position.i + x, Position.j + y)]^2$
  *End for*
*End for*
$Normalization_{Factor} \leftarrow sqrt(temp2.temp3)$
$Quality \leftarrow temp1/Normalization_{Factor}$
*Return Quality*
*End* ***Compute***

*Algorithm 2: Computation for template matching algorithm*

*GPU Implementation*

Considering GPUs are intended to enhance the performance of particularly data parallel applications, such template matching. CUDA framework is used to implement and analyze the template matching algorithms on GPUs for this work. The compute operation is the same as the compute operation for CPU, Algorithm 2, the only difference in this case the computation is performed on the GPU. One dimensional thread block was used where a single block processes a single ROI (13 x 13 region). The CPU is responsible for initializing the GPU and transferring the data between system and device global memory.

$\boldsymbol{ComputeGPU}(Template, Frame, Position)$
*Input*: $Frame, Template$
*Output*: $Match_{Qualty}, Match_{Position}$
  $i \leftarrow ThreadID$
  $x = i\ /\ (Frame.Width - Template.Width + 1)$
  $y = i\ \%\ (Frame.Width - Template.Width + 1)$
    $Quality \leftarrow Compute(Frame, Template, Position(x,y))$
    *if* $Quality > Match_{Quality}$ *then*
      $Match_{Position} \leftarrow Position$
      $Match_{Qualty} \leftarrow Quality$
    *End if* $ThreadID = 0$
      $Normalization_{Factor} \leftarrow sqrt(temp2.temp3)$
      $Quality \leftarrow temp1/Normalization_{Factor}$
  *Return Quality*
  *End* ***ComputeGPU***

*Algorithm 3: GPU based implementation template matching algorithm*

## V. RESULTS AND DISCUSSION

The algorithm is tested on an Intel Core i3-2100 CPU at 3.1 GHz frequency, with CUDA enabled GeForce 8400GS GPU. GeForce 8400 GS is a low-cost integrated graphics chip featuring 2 SMs and a memory bandwidth of 9.6 GB/s [19]. It GPU has 8 CUDA cores, with a processor clock of 1340 MHz, 64 bit memory

interface for a dedicated video memory of 1024 MB and PCI Express x16 Gen2 bus.

**Test Data**

The algorithm is tested on four video sets, Ideal Target, Simple Background, Complex Background and Similar Colored Background; for simple Targets and Complex targets. Two types of lighting conditions were considered, Indoor and Outdoor. The test set included video files, with frame rate of 25 FPS and have a resolution of VGA (640 x 480 pixels) for all experiments, except for the speed up test which used resolutions in Table 3. The data sets considered for the experiment included various colored, sized and shaped target templates in the scene. For checking robustness of the algorithm noisy targets are considered, since they have generally been one of the common problems in the area of automatic target recognition.

**Performance metrics used**

Performance metrics used include metrics used for testing detection and recognition of the algorithm.

The metrics used for testing the detection phase are:

*Detection rate*, which is the ratio of number of target pixels detected to the total number of target pixels available in a frame. The average is taken for the frames in the videos.

*Miss rate,* is the ratio of missed/undetected target pixels to the total number of target pixels available in a frame.

*False detection* is the ratio of number of non-target pixels detected to the total number of pixels available in a frame

The metrics used to test the recognition phase are:

.*Degree of recognition* represents the result of the recognition phase, a normalized template matching algorithm result. It has a value between 0 and 1, 0 being total mismatch and1 full match.

*False recognition rate* is the ratio of frames with false recognition to the total number of frames.

*Speedup* is the ratio of the computation time of the implementation on CPU to the computation time of the implementation on GPU.

## Results

The algorithm is tested for different performance metrics varying the important parameters of the algorithm. Table 1 shows the results found considering varying threshold values of the detection phase (Eq. 9). A standard VGA resolution video is used for this experiment.

| Threshold Value | Average Time/ Frame(ms) | % of Target misses |
|---|---|---|
| 0 | 2.3 | 98 |
| 1 | 4.6 | 94 |
| 2 | 6.3 | 93 |
| 3 | 6.3 | 91 |
| 4 | 8.0 | 83 |
| 5 | 9.2 | 63 |
| 6 | 10.2 | 66 |
| 7 | 13.7 | 45 |
| 8 | 14.2 | 36 |
| 9 | 17.1 | 31 |
| 10 | 19.8 | 12 |
| 11 | 21.2 | 8 |
| 12 | 24.9 | 5.5 |
| 13 | 25.4 | 3 |
| 14 | 34.6 | 2.5 |
| 15 | 49.7 | 2 |
| 16 | 59.6 | 2 |
| 17 | 62.7 | 1.5 |

Table 1: Performance of the system for varying threshold values

The variation of the detection rate/false detection rate as a function of the color range is tested, and the result of the experiment on the data in Figure 3.

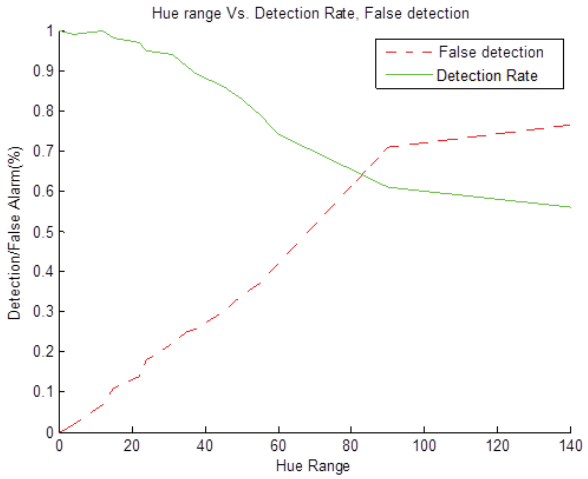


Figure 3: Variation of detection rate with Color range.

*Degree of Recognition for varying scenes*

In this test the degree of recognition is considered for different scene types both for indoor and outdoor lighting conditions. The performance is measured in terms of the average time taken per frame, showing the time required for every frame being considered.

| Lighting condition | Scene Type | Avg. Time/Frame | False Recognition | Degree of Recognition |
|---|---|---|---|---|
| | Ideal | 7.34 | 0 | 1 |
| Indoor | Simple | 10.89 | 0 | 0.964 |
| | Complex | 22.48 | 0.071 | 0.9367 |
| | Similar | 38.11 | 0.942 | 0.7877 |
| Outdoor | Simple | 15.4 | 0.01 | 0.93 |
| | Complex | 23.7 | 0.192 | 0.8545 |
| | Similar | 44.62 | 0.86 | 0.6999 |

Table 2: Performance of the algorithm for varying scene and lighting conditions

Next the target recognition phase of the algorithm is tested when the size of the target varies. The test is taken for all the scenes, simple, complex and similar under both indoor and outdoor scenarios and the average of the tests is taken. As can be seen from the graph in Figure 4 below, the degree of the recognition deteriorates as the object size grows.

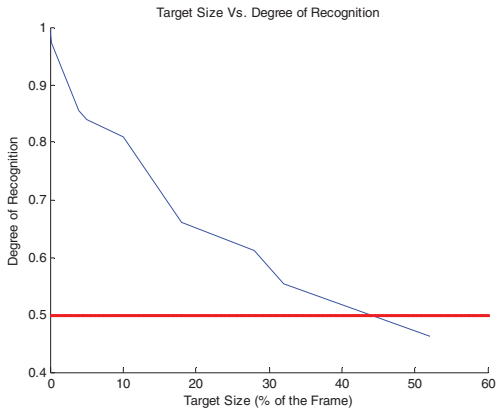


Figure 4: Target Template Size Vs. Recognition

**GPU Implementation Test**

The first call of functions on GPU is quite slow; it took 9.38 seconds for our case. So for performance measuring, it is necessary to do dummy function call and only then perform time tests. For GPU test videos with varying standard frame sizes has been tested, varying the frame sizes from Quarter Video Graphics Array (320 x 240) to HDTV Quad Extended Graphics Array (1920x1080).

| Standard Name | Size | Aspect | Average Time/Frame(ms) | Speedup |
|---|---|---|---|---|

| | | Ratio | CPU | GPU | |
|---|---|---|---|---|---|
| QVGA | 320 x 240 | 4:3 | 7.60 | 9.40 | 0.8085 |
| HVGA | 640 x 240 | 8:3 | 13.97 | 13.89 | 1.0058 |
| VGA | 640 x 480 | 4:3 | 24.83 | 15.33 | 1.6197 |
| SVGA | 800 x 600 | 4:3 | 50.67 | 25.40 | 1.9949 |
| XGA | 1024 x 768 | 4:3 | 69.50 | 34.67 | 2.0046 |
| XGA+ | 1152 x 768 | 3:2 | 78.11 | 35.20 | 2.2190 |
| | 1152 x 864 | 4:3 | 86.44 | 35.61 | 2.4274 |
| SXGA | 1280 x 1024 | 5:4 | 94.50 | 47.33 | 1.9966 |
| SXGA+ | 1400 x 1050 | 4:3 | 130.67 | 65.05 | 2.0088 |
| UXGA | 1600 x 1200 | 4:3 | 151.33 | 65.60 | 2.3069 |
| QXGA | 2048 x 1536 | 4:3 | 276.67 | 131.07 | 2.1109 |

Table 3: Speedup of the GPU implementation of ATR

As the video size grows the performance of the algorithm gets better on GPU-based implementation of the algorithm. Figure 5 shows the speed up of the algorithm using GPU-based implementation.

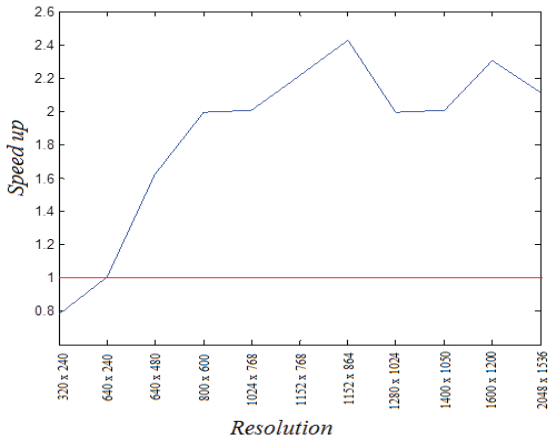


Figure 56: Speed up of the GPU-implementation over CPU-implementation

**Discussion**

The performance of the algorithm varied significantly when the threshold value of the background subtraction phase is varied. Minimizing the threshold value improved the computation time, but the target miss rate is affected severely. On the other hand, increasing the threshold level improved the target detection miss rate but at the cost the computation time. We found the threshold level of 13 to be an acceptable with the performance not affected severely, with in a real-time frame rate region, and an acceptable level of target miss rate.

For recognition phase of the algorithm, the degree of recognition varied significantly with scene type and lighting conditions. For scenes where targets have the same color as the background the false detection rate increased, depreciating the computation time of the algorithm. Test has shown that targets with lower Hue range (narrow Hue range) have better results for similar colored targets and backgrounds.

For the GPU based implementation of the algorithm the initialization time required for the device makes it lagging for live target processing. To overcome this, the target processing has to be done on CPU at start and proceed on GPU after the initialization.

## VI. CONCLUSION AND FUTURE WORK

This work demonstrates a performance improvement gained by using a hybrid algorithm for automatic target recognition. Both shape based and color based segmentation algorithms were used to get an improved performance and detection rate compared to each one of them separately.

The target recognition algorithm was tested both on CPU and GPU implementations. An appropriate choice of part of the algorithm is made for GPU implementation, with minimum data transfer. Computations where CPU is as fast as GPU remained at CPU as it appends unnecessary I/O transfer between CPU and GPU. Due to the inherent parallelism of the recognition algorithm, it is suited for GPU implementation. Since these is a component of the computation that work on huge data, Single Instruction multiple data units, exhibiting high degree of independence.

The performance of the system is tested for robustness and for variation in the properties of targets and the environment behind. The performance gain for using a co-processing on GPU has also been tested. A performance speed up is gained through the use of GPU.

The need for performance improvement in multimedia applications computation has been and will be an ever increasing challenge in digital signal processing. Meanwhile the performance improvement of digital devices is increasing. Performance improvement of Computer vision algorithms can be achieved through the use upcoming digital devices by mapping the algorithm or part of the algorithm on the device and taking advantage of the nature of the specific device used.

The algorithm implemented in this work only considers 2-dimensional views, but with an improved performance in the devices, 3-dimensional views can be implemented with this algorithm. In addition a single camera configuration is considered for this work, future works with device improvements can be mapped to more than one integrated camera systems.

It is also worth noting that the accuracy of algorithm can be improved further by incorporating advanced detection methods like Steerable filters for edge detection and methods for prediction probability.

## VII. REFERENCES


[1] Ufuk Suat Aydin, Traffic Sign Recognition, Middle East Technical University, May 2009.

[2] Second Lieutenant Michael A. Tanner, Image Processing for Multiple-Target Tracking on a Graphics Processing Unit, US Air Force, March 2009.

[3] Oscar Mateo Lozano • Kazuhiro Otsuka, Real-time Visual Tracker by Stream Processing, J Sign Process Syst. DOI 10.1007/s11265-008-0250-2

[4] Alok Whig, Stream Processor Based Real-Time Visual Tracking Using Appearance Based Approach, University Of Florida, 2009

[5] F. Meyer, “Color image segmentation,” Proceedings of the International Conference on Image Processing and Its Applications (pp. 303–306), 1992.

[6] Amdahl, G. "The validity of the single processor approach to achieving large-scale computing capabilities". Proceedings of AFIPS Spring Joint Computer Conference, Atlantic City, N.J., AFIPS Press, 1967, pp. 483–85.

[7] John L. Gustafson, "Reevaluating Amdahl's Law", Communications of the ACM 31(5), 1988, pp. 532–33

[8] Scott Hemmert, Brad Hutchings and Anshul Malvi. An Application-Specific Compiler for High-Speed Binary Image Morphology, Brigham Young University.

[9] Sajid, M. M. Ahmed, I. Taj, M. Humayun, and F.Hameed, Design of High Performance FPGA Based Face Recognition System, Mohammad Ali Jinnah University, Islamabad, Pakistan.

[10] Xiaohan Chen and Natalia A. Schmid, On the Limits of Target Recognition in the Presence of Atmospheric Effects. West Virginia University, Morgantown, WV 26506

[11] Tien-Hsin Chao and Thomas Lu, Automatic Target Recognition (ATR) Performance Improvement Using integrated Grayscale Optical Correlator and Neural Network, California Institute of Technology

[12] Young H.Cho, Optimized Automatic Target Recognition Algorithm on Scalable Myrinet/Field Programmable Array Nodes,The University of Texas at Austin, U.S.A.

[13] Michael Rencher and Brad L. Hutchings, Automated Target Recognition on Splash. Brigham Young University, Provo, UT 84602

[14] Mahsa Maghami,Michael C. Koval.Social Network Analysis for Automatic Target Recognition in SwarmRobotics, University of RochesterRochester, NY, USA

[15] Jue Wang, Bo Thiesson, Yingqing Xu, Michael Cohen. Image and Video Segmentation by Anisotropic Kernel Mean Shift, Microsoft Research (Asia and Redmond).

[16] Darrin Cardani, Adventures in HSV Space, AdobeSystems, Inc.

[17] Image indexing using color histogram in the CIELUV color space

[18] Gaurav S. Sukhatme The Path to Autonomous Robots, Los Angeles, California 90089-2905

[19] Abel Paz ; Antonio Plaza, GPU implementation of target and anomaly detection algorithms for remotely sensed hyperspectral image analysis, IEEE Geoscience and Remote Sensing Letters, Vol. 10, No. 2, March 2013.

[20] Bernabe, S., et al, GPU Implementation of an Automatic Target Detection and Classification Algorithm for hyperspectral Image Analysis, University of Extremadura, Avda. de la Universidad s/n 10071 C´aceres, Spain