Parallelizing Image Processing Algorithms for Face Recognition on Multicore Platforms

Kausar Mia[1]
Department of CSE
Daffodil International University
Dhaka, Bangladesh

Mr. Md Assaduzzaman[3]
Department of CSE
Daffodil International University
Dhaka, Bangladesh

Arnab Saha[5]
Department of CSE
Daffodil International University
Dhaka, Bangladesh

Tariqul Islam[2]
Department of CSE
Daffodil International University
Dhaka, Bangladesh

Tajim Md. Niamat Ullah Akhund[4]
Department of CSE
Daffodil International University
Dhaka, Bangladesh

Sonjoy Prosad Shaha[6]
Department of CSE
Daffodil International University
Dhaka, Bangladesh

Md. Abdur Razzak[7]
Department of CSE
Daffodil International University
Dhaka, Bangladesh

Angkur Dhar[8]
Department of CSE
Daffodil International University
Dhaka, Bangladesh

***Abstract*—A good face detection system should have the ability to identify objects with varying degrees of illumination and orientation. It should also be able to respond to all the possible variations in the image. The image of the face depends on the relative camera face pose such as the nose and one eye. The appearance of a face is directly influenced by the facial expression of a person and partially occluded by objects around it. One of the most important and necessary conditions for face recognition is to exclude the background of reliable face classification techniques. However, the face can appear in complex backgrounds and different positions. The face recognition system can mistake some areas of the background for faces. This paper solves some face recognition problems including segmenting, extracting and identifying facial features that are thought to face from the background.**

***Keywords*—Image processing; multi-core platforms; machine learning; face recognition; parallelizing**

I. Introduction

People have an excellent ability to analyse images and we can recognize the face very reliably. Indeed, humans can easily find the surrounding face despite difficult situations such as obscure parts of the face or heavy lightning. Facial recognition is considered a prerequisite for many computer vision applications such as security, surveillance, and content-based image search. So much research has been done to automatically replicate this process on machines [1]. For face detection methods, several authors have defined a vast range of methods for face detection [2]. The feature-invariant approach is primarily aimed at finding structural features that are present even when "the pose, the angle of view, or lighting conditions change". People are expected to easily recognize faces, so they need constant properties or capabilities for these fluctuations. The problem with this approach is that lighting, noise, and occlusion can severely damage image features. The template matching process defines some standard patterns for the face. These patterns are stored in templates to describe the entire face or individual facial features. The correlation between the entered pattern and the saved pattern is calculated for recognition. In the appearance-based method in contrast to template matching, the model is trained from a series of training images aimed at capturing typical variations in facial appearance. One of the problems with this approach is that training the model can take hours or even days. Face recognition method considerations represent the boundaries between knowledge-based methods and template comparison methods. The latter is usually because it implicitly requires human knowledge to define a face template.

The aim of this work is to implement parallelize Image Processing Algorithms for Face Recognition on Multi-core Platforms. In order to reach our goal we started by analyzing the challenges associated with face detection which involves factors such as pose, presence or absence of structural components, facial expression and imaging conditions. When the face detection methods were analyzed, it was determined that the ones based on learning algorithms (appearance-based) provides better results. Because these methods eliminate the possibility of modeling error, which can occur when face knowledge is either insufficient or inaccurate.

II. Related Works

Face recognition techniques based on learning algorithms (appearance-based) have recently received a lot of attention to eliminating potential modelling errors due to incomplete or inaccurate face knowledge. A neural network-based face recognition framework developed by Rowley et al. [3] proposed, reasonably connected NN (neural network) examines small windows in the image to determine if each window contains a face. Fig. 1 shows Neural network face detection.

Authors of [1] ilustrated the understandings of computer vision and image processing. Authors of [2] mentioned many

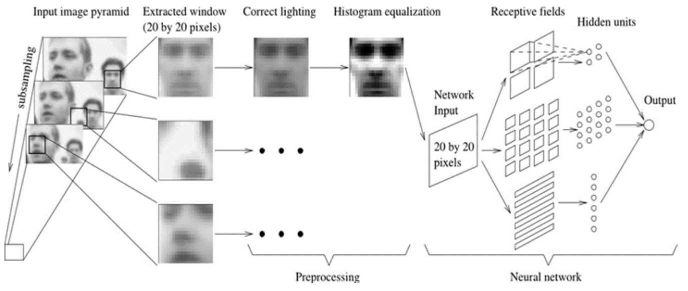

Fig. 1. Neural Network Face Detection.

approach of detecting face in an image. Authors of [4] used deep learning for image recognition. Authors of [3] used neural network for face detection. Authors of [5] showed an improved algorithm in this aspect. Authors of [6] showed GPU performance in computer vision aspect. Authors of [7] worked on Parallelizing face detection. High perfomance computing for image processing is analysed by authors of [8]. Haar-like features were implemented by authors of [9] and [10]. For face detection CNN can be used [11][12] for robust face detection. Multiple faces can be detected with adaboost and Camshaft algorithms [13]. Moreover, artificial intelligence and internet of things based solution can make changes in medical sectors [14], hotel sectors [15], covid-19 patient management sectors [16], and many other sectors [17]. Remote sensing [18] makes a great change in virus-affected people monitoring [19]. IoT-based systems are also secure [20]. These systems can also help in agriculture [21], poultry farm [22], disable people management [23], electronic voting [24], gaming [25], farming [26], nursing [27], remote data sensing [28], virus affected area monitoring [29][30] from remote places [31]. IoT based irrigation systems are helping farmers [32] and farming [33]. Robotics and IoT are doing great in medical fields also [34][35]. So, in multicore platform parallelizing image processing algorithm should work better for face recognition and help mankind.

III. Face Detection Optimization

We would like to optimize the face detection task to reduce its computation time [4][3]. To do so it's important to understand how it works inside. The face recognition task first creates an image pyramid, which is a multi-scale representation of the image. This makes the face recognition scale invariant and it recognizes faces in the same recognition window. Fig. 2 shows Pyramid of scales.

On each scale it is important to notice that the size of the detection window is the same in all the scale. This process is such that smaller faces are detected on early scales while bigger faces are detected later. Fig. 3 shows Detection window (sub-window).

A. The Scale Factor

It allows the creation of the image pyramid, by re-scaling the input image we can resize a larger face to a smaller one. Each scale will reduce the size by 10% concerning the previous one. The algorithm works leisurely since a lot of scales are created. It is possible to grow 1.40 for faster detection with

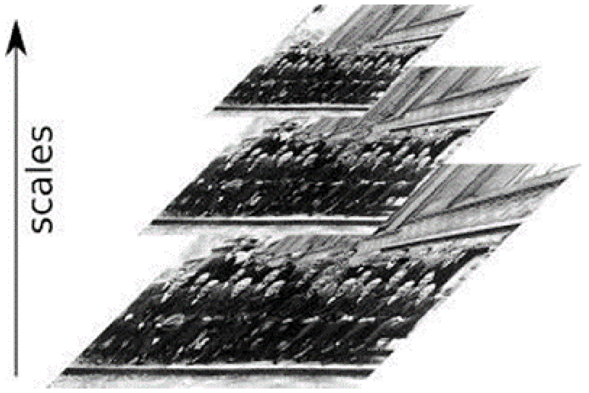

Fig. 2. Pyramid of Scales.

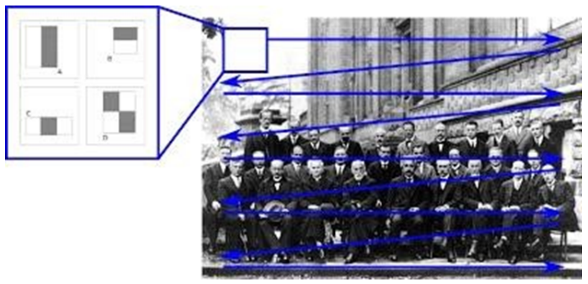

Fig. 3. Detection Window (Sub-Window).

the missing some faces altogether. To precisely detect the face, all the scales that return a face result are merged into one final result. In Fig. 4, 46 scales were generated. Scales (1 - 35) and (43 - 46) did not find any face, scales (36 - 42) did find a face (left image), and all the scales that found a face are merged into one final result in the right image.

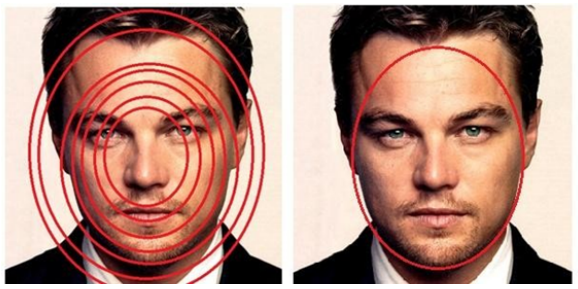

Fig. 4. 46 Scales were Generated. Scales (1 - 35) and (43 - 46) did not find any Face, Scales (36 - 42) did find a Face (Left Image), and All the Scales that found a Face are Merged into One Final Result in the Right Image.

B. Minimum and Maximum Object Size

The standard for this parameter is usually [30, 30] pixels, and for bigger objects than the size, we want this to be ignored. Now that the parameters are known, we would like to work with the low scale factor (1.1) so that all the possible faces are detected but also with a few numbers of scales. The stopping condition for the creation of scales is based on the two parameters defined above (min and max object size). Fig. 5 shows Stopping condition for the creation of scales. The size

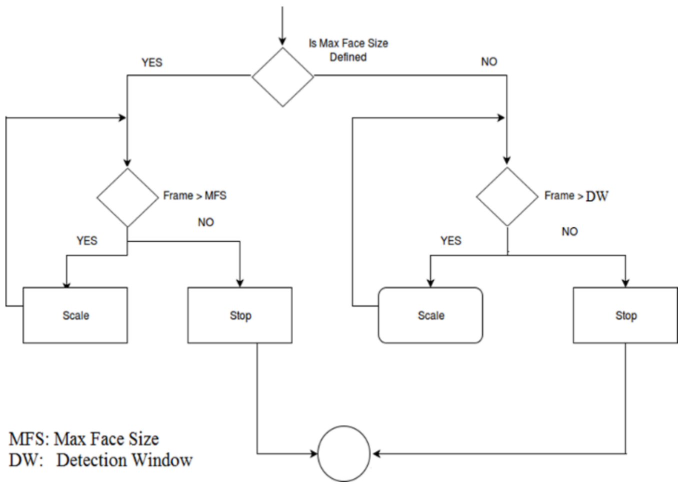

Fig. 5. Stopping Condition for the Creation of Scales.

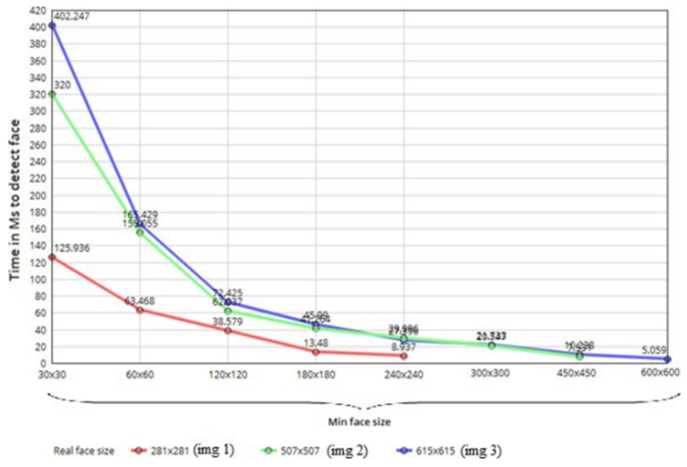

Fig. 8. Computation Decreased as the Min Face size Gets Closer to the Real Face Size.

of the detection window is defined as double the min object size. So only objects that a greater than the min object size and smaller than the detection window are considered. Fig. 6 shows the detection window. By experiment was possible

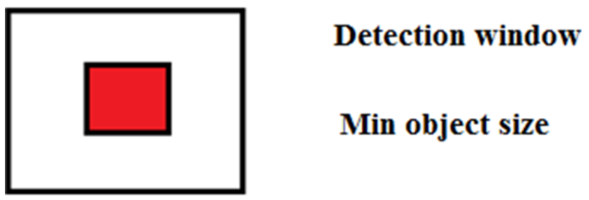

Fig. 6. Detection Window.

to notice that the most important factor to optimize the face-detection is the min object size, as it directly affects the size of the detection window and the number of scales [5]. A final optimization solution for a stream of images can be seen as: first, we start by considering the worst-case scenario in which the parameter for min face size is (30, 30) and a scale factor of 10% which will generate a lot of scales once a face is found, adjust the detection window to the size of the face for the next frames (min_face_size = real_face_size ∗0.8). This process allows reducing drastically the number of scales as they are determined based on the previously detected face dimensions as it's possible to see from Fig. 7. As it is possible to notice

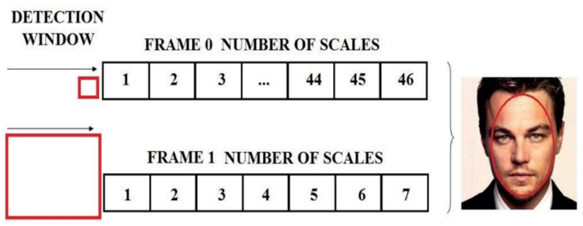

Fig. 7. Optimized Solution.

from Fig. 8, the computation time decrease drastically as the min face size gets closer to the real face size and the number of scales is reduced.

This face detection optimization should be applied only for cases where the goal is to find exactly one face or at most multiple faces that are somehow equidistant from the camera. When the algorithm is applied for the first time (left image), we do not have any a priori knowledge of the size of the faces, and almost all of them are found [36]. Because of the image pyramid of scales, the smallest faces (the ones much farther away from the camera) are the first to be found. Then the intermediate faces (the ones in the center) and finally the largest faces (the ones in the front line). In this way, the largest faces are the last ones to be found, so they will set the threshold of what size of face to look for. When the algorithm is applied for the second time (right image) only the front-line faces are found. Table (I) shows the Styles Summary.

TABLE I. STYLES SUMMARY

Paragraph Style Name (MS Word)	Font Size	Spacing Before	Spacing After
Title_text	20 (B)	0 pt.	24 pt.
Authors_name	12 (B)	0 pt.	12 pt.
Authors_aff	10	0 pt.	0 pt.
Abstract	10 (B)	0 pt.	4 pt.
Keywords	10	12 pt.	12 pt.
Body_text	10	0 pt.	0 pt.
Section_heading	10 (B)	12 pt.	4 pt.
Subsection_heading	10 (B)	12 pt.	4 pt.
Equations	Eqn	12 pt.	12 pt.
Figure	10	12 pt.	0 pt.
Figure_no	10 (I)	6 pt.	12 pt.
Table	10 pt.	0 pt.	12 pt.
Table_no	10 (I)	12 pt.	6 pt.
Reference	10	0 pt.	3 pt.

IV. PARALLEL IMPLEMENTATION

We will start by studying the parallel patterns and tools that will be applied later on to parallelize the application. The potential sources of parallelism of the Viola-Jones algorithm [37] will be also analysed followed by a detailed study of the OpenCV parallel implementation [6]. An improvement to the OpenCV parallelism will also be addressed along with the GPU implementation of the Viola-Jones algorithm. We will discuss techniques for the design of parallel computations and the tools that allow implementing such techniques [38].

A. Pipeline Paradigm

This paradigm is defined for the stream of computations only. It is applied when we have a computation that can be expressed as a composition of functions. The service time of the pipeline depends on the slowest stage. This paradigm is generally characterized by a latency (time between the input of a data and the output of its result after processing) worst than the sequential version due to the communication between the stages. The Pipeline may be used to improve the service time, however, to remove application bottlenecks with this paradigm we must ensure that the number of functions is equal to or greater than the optimal parallelism degree and the stages are somehow balanced. Fig. 9 shows the Pipeline process.

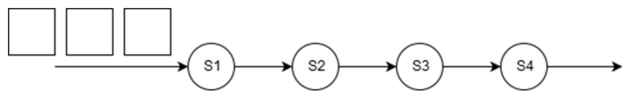

Fig. 9. Pipeline Process.

B. Embarrassingly Parallel Paradigm or Farm

This is a streaming paradigm that replicates the same function that distinct stream elements can be processed by DIM in parallel. This paradigm is composed of three modules: “Emitter, Workers, and Collector”. The emitter provides to send every received input stream to one of the workers according to a certain policy. Each input element is transmitted to exactly one worker. Each worker applies the function F to the received data and sends the result to the collector which transmits each of them to the output stream. To apply the farm paradigm the function F must be a pure function, that’s it without an internal state. The farm paradigm is good for applications where the order of the output is not so important as the relative speed of the workers can be different. In Fig. 10, we can see the farm as a three-stage Pipeline, in which the emitter is the first stage, the workers in the second, and the collector in the last stage.

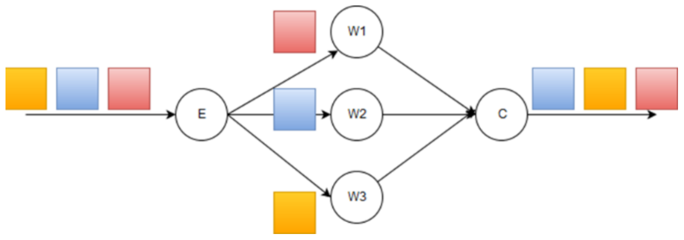

Fig. 10. Embarrassingly Parallel or Farm.

C. Map or Parallel for Paradigm

This is a data-parallel paradigm that can be applied on streams as well as on single data values. This paradigm is characterized by data partitioning and by function replication. For stream-based computations, besides service time, it also optimizes latency and memory capacity [39] which is not the case with the paradigms introduced above. During the execution of function F, each of the functions on its local data exclusively in the Map paradigm, without any collaboration with the other workers. Parallel for is a paradigm characterized by parallelizing sequential iterative for loops with independent iterations [7]. Fig. 11 shows the Map paradigm.

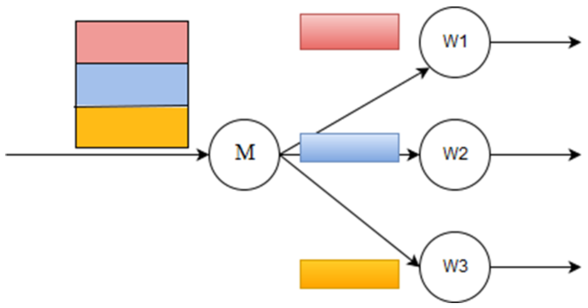

Fig. 11. Map Paradigm.

D. Fast Flow (FF)

Fast Flow is a C++ parallel programming framework that focuses on high-level, pattern-based parallel programming. It supports streaming and data parallelism. It allows expressing the combination of different patterns which increases the parallelism exploitation possibilities [8].

V. Parallel Implementation of OpenCV

OpenMP is a programming framework that greatly simplifies writing multi-threaded programs in Fortran, C, and C++. This programming framework provides loop parallelization with parallel for. Besides being able to implement parallel loops when iterations are independent, it also has the notion of task parallelism which allows assigning different dependent or independent tasks to different threads. To write efficient programs with OpenMP, we need to understand all the additional parameters of the OpenMP pragmas and also the concurrent accesses performed on shared data structures. To implement parallelization, OpenCV uses a hierarchy of tools [9]. The first tool found in the user machine is the one that is going to be applied. For our specific case, all the parallelization implemented by OpenCV was performed with OpenMP. For the object detection task, OpenCV automatically parallelizes the data in each scale with a row-wise Parallel which divides the computation space into a collection of detection windows. Each detection window is calculated independently without any need for synchronization. If the user doesn’t specify the number of threads to be used, OpenCV will automatically use all of the available threads. Fig. 12 shows the OpenCV parallelization inside each scale.

A. Convert an RGB Image to a Grayscale

There is no data dependency between the three channels composing an RGB image when performing the grayscale conversion [40]. In this way, we can exploit the Map paradigm to parallelize this task. Synchronization is needed to have the equivalent grayscale of each channel. The final step is to merge the three-channel grayscale images into a single one. Fig. 13 shows the process of Converting an RGB image to a grayscale.

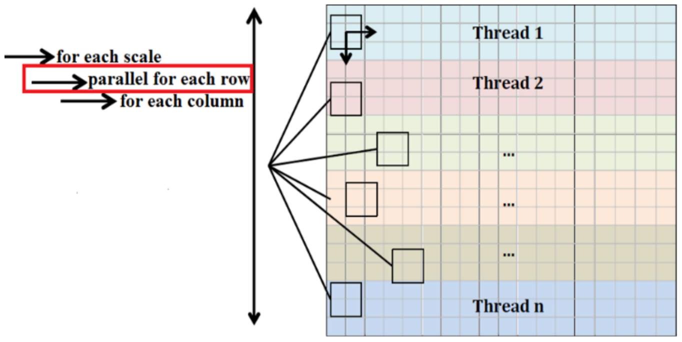

Fig. 12. OpenCV Parallelization Inside each Scale.

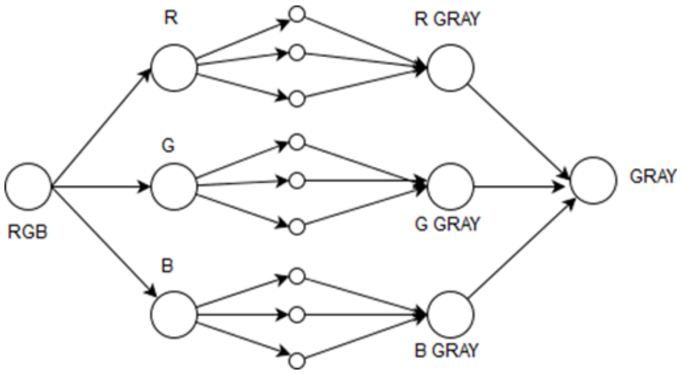

Fig. 13. Converting an RGB Image to a Grayscale.

B. Histogram Equalization

When performing histogram equalization, it's possible to exploit the Map paradigm in which the image is split to compute the histogram H of the source [11]. A synchronization point is needed to have the final histogram H. The computation of the histogram integral image is performed using a pair of recurrences. Transforming H´ as a look-up table can also be performed independently by applying the Map paradigm. Fig. 14 shows the process of Histogram equalization.

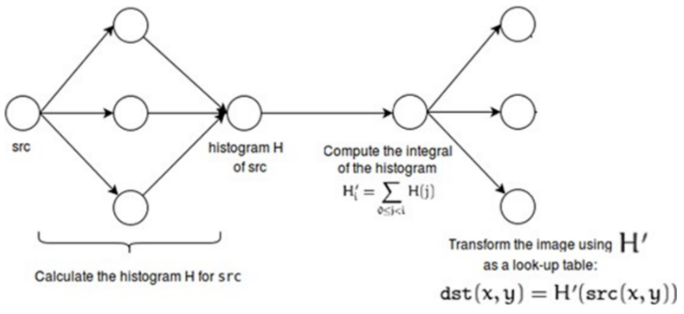
src
histogram H of src
Compute the integral of the histogram
$H'_i = \sum_{0 \le j < i} H(j)$
Transform the image using H' as a look-up table:
$dst(x,y) = H'(src(x,y))$
Calculate the histogram H for src

Fig. 14. Histogram Equalization.

C. Detection Window

The face detection computation space is partitioned into a set of detection windows, which can be computed independently without the need for synchronization. The schema for this case is the same as Fig. 11 showed for the Map paradigm in which the input image is partitioned and detection windows would be applied in parallel in each of the partitions.

D. Haar-Feature Calculation and selection

Inside each detection window no data dependence on the feature value calculation and selection in such a way that each feature can be calculated and selected in a completely independent way [12][10]. All the selected features are sent to the cascade of classifiers shown in Fig. 15.

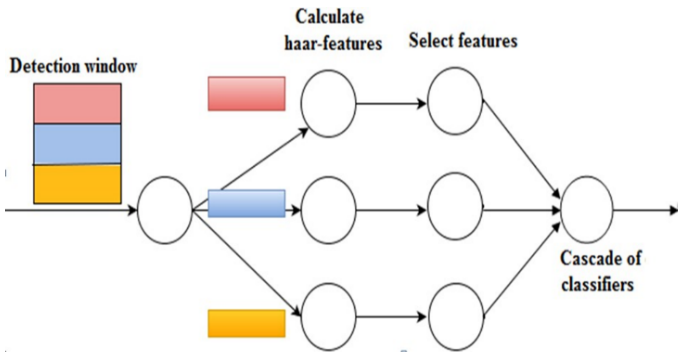

Fig. 15. Haar-Feature Calculation and Selection.

E. Feature Evaluation

All the chosen functions are evaluated through the levels of the cascade classifiers which may be parallelized to the usage of the pipeline paradigm. Map paradigm in which each feature instead of being partitioned is sent entirely to all the filters. This process is known as multicast. A synchronization point is needed to compute the outcome of the stage. The detection window is automatically discarded if the outcome of one of the stages is negative. The face detection cascade of classifiers has 22 stages. The number of filters for each stage can be seen in Fig. 16 below.

Stage	0	1	2	3	4	5	6	7	8	9	10	11	12
Filters	3	15	21	39	33	44	50	51	56	71	80	103	111

Stage	13	14	15	16	17	18	19	20	21
Filters	102	135	137	140	159	177	181	210	213

Fig. 16. Stages of Face Detection Classifier.

Fig. 17 shows Cascade of classifiers.

F. Feature Evaluation

We must scale down the input picture according to the supplied scale factor until it is the same size as the detection window to locate all potential faces of various sizes. Without the requirement for synchronization, each scale of the incoming picture may be handled in parallel. The schema for this case is the same as the figure shown on the Farm paradigm in such a way that each scale would be sent to a worker. The overall parallelization scheme can be seen in Fig. 18 below which the red circles represent.

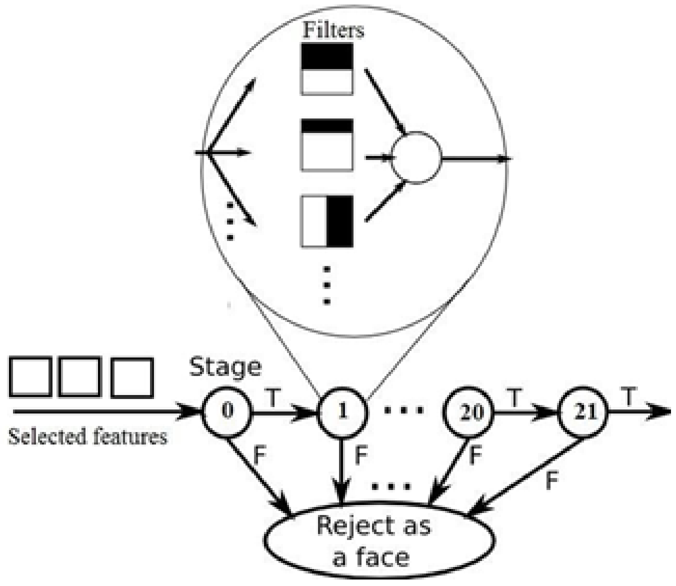

Fig. 17. Cascade of Classifiers.

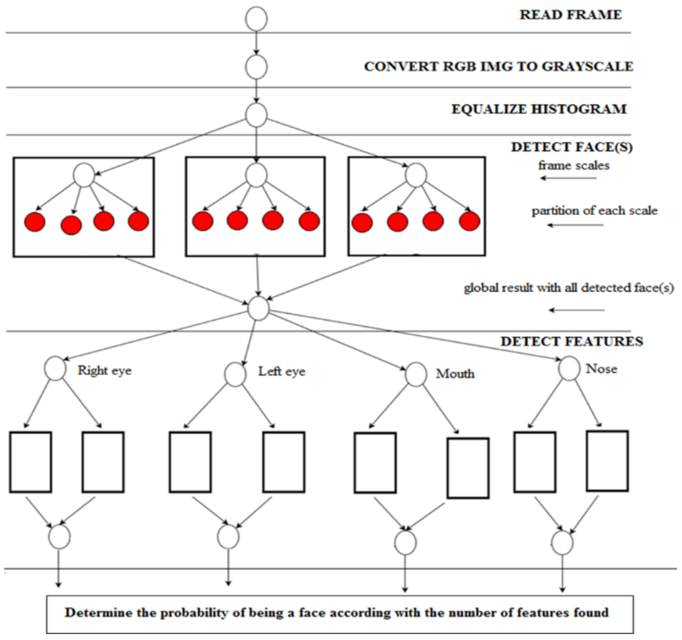
READ FRAME
CONVERT RGB IMG TO GRAYSCALE
EQUALIZE HISTOGRAM
DETECT FACE(S)
frame scales
partition of each scale
global result with all detected face(s)
DETECT FEATURES
Right eye
Left eye
Mouth
Nose
Determine the probability of being a face according with the number of features found

Fig. 18. Overall Parallelization Scheme.

VI. Additional Parallelism to the OpenCV Viola-Jones Implementation

We will focus on providing additional parallelism to the OpenCV implementation of the Viola-Jones algorithm. We will also discuss ways of improving the existing OpenCV parallel implementation. The most straightforward approach to parallelizing a stream computation made up of phases that must be executed serially is to implement a pipeline paradigm. One important aspect to take into account when using the pipeline paradigm is that the stages must be somehow balanced to exploit all the benefits of this parallelization. The proposed pipeline stages are in Fig. 19. For the results that support this pipeline configuration to implement this paradigm, we will

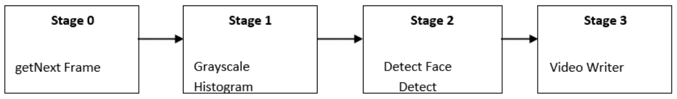

Fig. 19. Stages.

use FastFlow. The pipeline completion time depends on the slowest stage which in this specific case is stage 2. This stage completion time can be approximated by the face detection task time, so the cost model of the pipeline can be derived as Tc = N x face_detection() where N is the size of the stream. Therefore, to fully exploit the pipeline paradigm and take into account the completion time of each task, it becomes very important to be able to parallelize the face detection procedure internally.

A. *Face Detection Task Parallelization*

The face detection task processes a pyramid of scales of the same frame. The image pyramid allows face detection to be scale-invariant. There is no dependency between the processing relative to the different scales, they can be executed in parallel without any need for synchronization. The computation time is different between the scales due to their different size while applying the same detection window. Fig. 20 shows Image pyramid.

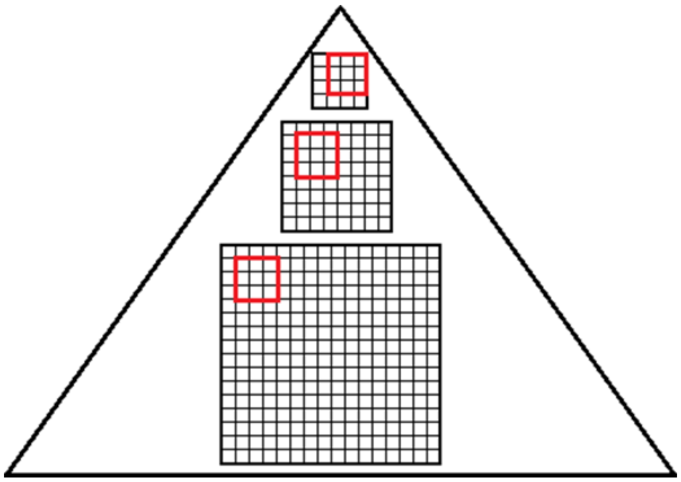

Fig. 20. Image Pyramid.

B. *Considerations about OpenCV Parallel Implementation*

OpenCV internally parallelizes the computation relative to each scale using the parallel paradigm. Externally, the scales are executed sequentially, one after the other. Even though the image scales are of different sizes, the same number of threads is assigned to compute all of them. For this specific case, the face detection task generated an image pyramid of 46 scales. We measured the time of each scale using a thread number between 1 and the maximum number of threads available (24 in this case). As evidenced in Fig. 21, the scalability achieved with 24 threads is very significant for scales of bigger size while on the other hand is negligible for scales of smaller size. If we consider the efficiency, it decreases as the number

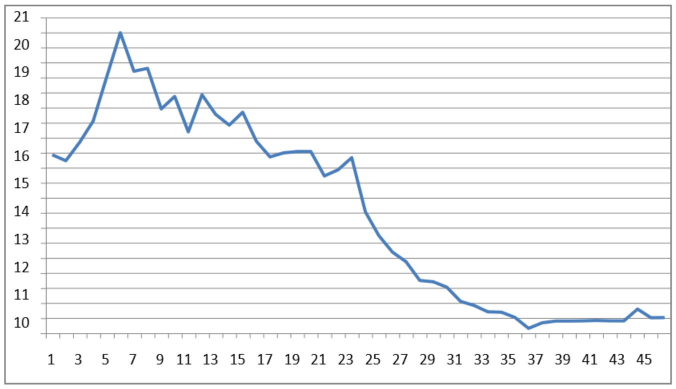
Fig. 21. Scalability Achieved with 24 Threads.

of threads increases Fig. 22. When considering the efficiency of each scale with 24 threads Fig. 23, it decreases drastically from the second half (from scale 24 to 46 of Fig. 23).

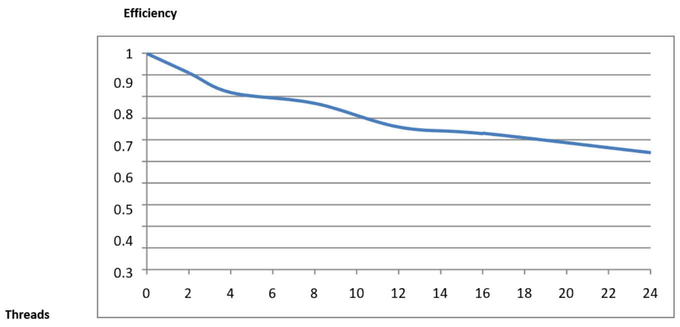

Fig. 22. Efficiency Achieved (1).

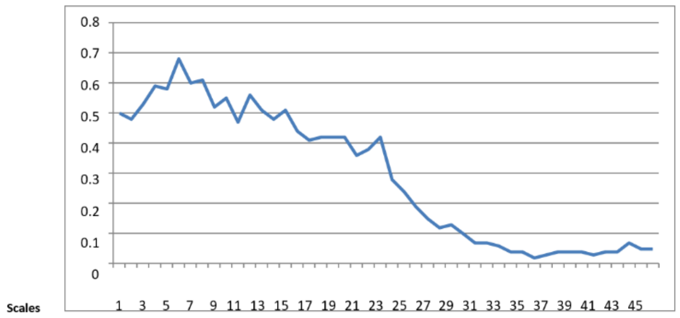

Fig. 23. Efficiency Achieved (2).

C. Improvement to the OpenCV Parallel Implementation

Analyzing deeply the results obtained from the Fig. 23 above, it's possible to realize that we can optimize the parallelization with 24 threads in the following alternative ways shown in Fig. 24.

Since in both alternatives we are parallelizing for loops, OpenMP is the most suitable parallelization tool. We can see that executing in parallel from scale 24 to 46 but inside executing serially each scale, without taking into account any

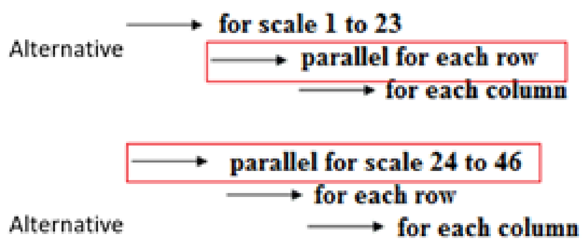

Fig. 24. Parallel Implementation.

overhead, the computation time can be roughly approximated to the time spent to execute the slowest scale (which is scale 24) with 20.62 milliseconds. But if we take the OpenCV approach which executes the scales serially while performing a row-wise parallel computation inside each scale, the execution time from scale 24 to 46 with 24 threads is roughly the sum of the times of each scale on this range which is 44.437 milliseconds. So, with the optimized approach, we can get about 2.15x speed up for the OpenCV approach. Assigning computation of different scales to different threads would fall under the category of embarrassingly parallel or farm paradigm. A dynamic load balance technique can be applied in this case such that when each scale finishes its execution, it frees the resource. In all circumstances when the exact quantity of labor of activities cannot be exactly stated before the activities take place, this strategy may provide better load balancing. Based also on other performed tests which demonstrated similar (scalability) results like the ones shown before, we divide the total number of scales by half. The heavy-sized scales in the first half would be parallelized using a row-wise parallel, while the light-sized scales in the second half would be parallelized by computing each scale in parallel. From Fig. 25 is possible to notice that the face detection parallelization is rather coarse-grained as the best scalability was achieved while the smaller one achieved the worst result.

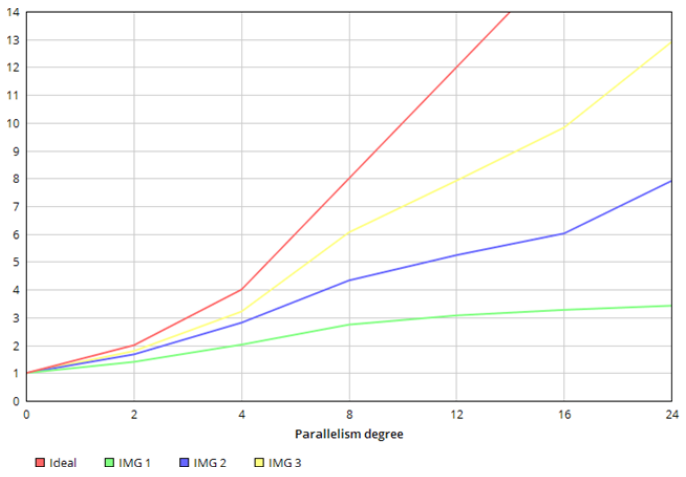

Fig. 25. Face Detection Parallelization.

D. Specific Approach: Single Face Detection

For this specific case, before finding a face the parallelization technique is the same as the one described above in the proposed heuristic solution. Once the face is found, taking advantage of the optimization, where the number of scales and the computation time is reduced drastically, as it is possible to see from the Table (II) below, the parallelization paradigm that could be applied is the farm where each scale would be assigned to a specific number of the worker. One could argue that after finding a face no further parallelization is needed taking into account that the number of scales and the computation time of each scale is reduced drastically. For our specific case, we can see from Table (II) that the sequential implementation would take as much as 4.01 milliseconds to complete while in the best-case scenario, without overheads, the parallel implementation would take 0.95 milliseconds. It is worth remembering that the results of the Tab. 2 were obtained with the largest image of our experiments (Size 2448 x 2448) which represents the worst case of the face sizes. When introducing further parallelization to this part, we need to take into account the trade-off between the gains of the parallelization and the additional overheads that it comes with as even in the best-case scenario the gains can be very negligible (around 3 ms for our case).

TABLE II. Optimized Results for Single Face Detection

Scale	Size	1-thread[ms]
1	2348x2348	0.87
2	2134x2134	0.95
3	1940x1940	0.80
4	1764x1764	0.76
5	1604x1604	0.44
6	1458x1458	0.18
7	1325x1325	0.01

E. Feature Detection Parallelization

In this case, we are not interested in parallelizing each feature detection phase internally, as the optimization proposed already allows for achieving higher speed up. The important point is to run all the features (eyes, mouth, and nose) in parallel using the MISD pattern which is a variation of the Map paradigm where each worker computes in different codes. Fig. 26 shows Feature detection parallelization.

A possible approach for detecting features in parallel for multiple faces could be applying the Farm paradigm in which each face would be assigned to a certain work and inside each worker [41], the MISD paradigm would compute the features in parallel in Fig. 27.

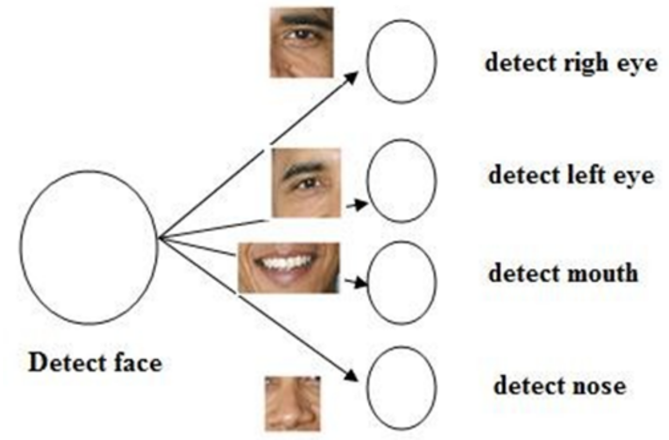

Fig. 26. Feature Detection Parallelization.

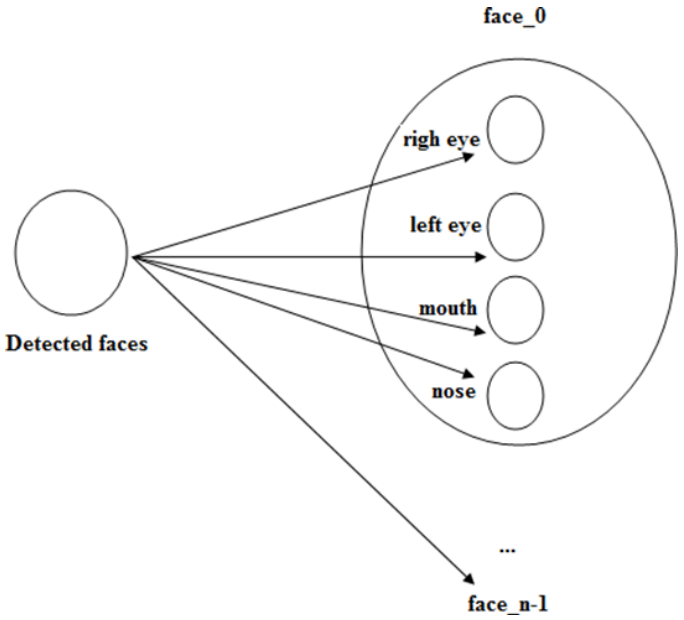

Fig. 27. Detecting Features in Parallel for Multiple Faces.

VII. Results and Discussion

Our authors saw the testing on this application, and it was determined that splitting the face did not improve performance, therefore we opted to use the four-stage pipeline. When using the Multiple Face Detection Approach to test the application, the smallest image was able to achieve real-time processing with the GPU (around 38 FPS) and quasi-real-time processing with the CPU parallelized version (around 10 FPS), while the intermediate size image was able to achieve quasi-real-time processing with GPU (Around 17 FPS) [42]. The least and intermediate photos were able to achieve real-time processing with all implementations save the originally sequential one when we ran the same trials with the Single Face Detection Approach. With the parallel CPU and GPU implementations, the largest image was able to accomplish real-time processing [43] [44]. When the application was tested on video frames, it was discovered that the size of the face had a direct impact on the sequential and parallel CPU implementations optimized for single face detection. We demonstrated that for the largest image size, the optimization introduced, allowed to decrease in the number of scales from 46 to 7 and the number of processed sub-windows from 8 443 722 to 704 which implies a reduction of 99.99% of processed sub-windows. The reliability tests showed that our application is very reliable in different scenarios such as distance, light conditions, occlusion, and rotation [45]. We demonstrated that it is possible to coexist the two parallel programming frameworks utilized in our application (Fast Flow and OpenMP) to exploit all the available resources on CPU Multi-Core Platforms.

VIII. Conclusion

This paper aimed to Parallelize Image Processing Algorithms for Face Recognition on Multicore Platforms. The heavy-sized scales in the first half would be parallelized using a row-wise parallel, while the light-sized scales in the second half would be parallelized by computing each scale in parallel. When the face detection methodologies were analyzed, was concluded that the ones based on learning algorithms (appearance-based) allow for achieving better results because eliminate the potential modeling error as a result of incomplete or inaccurate face knowledge. The state-of-the-art algorithms can process images extremely rapidly based on three innovative contributions: "The integral image, the AdaBoost learning algorithm, and a cascade of classifiers". The Viola-Jones algorithm is still not optimal as the detector becomes unreliable for faces with a certain rotation, the detector may fail when the faces are very dark while the background is relatively light and the detector fails on significantly occluded faces. To render the face detection process faster two optimizations were proposed. The first optimizes the face detection task based on the size of the face found on the previous frame. This optimization allows decreasing drastically the computation time. The drawback of this optimization is that it should be applied only for one face, or at most multiple faces that are somehow equidistant from the camera. The second optimization also allows decreasing the computation time as the size of the facial features to look for is based on a ratio to the size of the face found. To parallelize the entire face detection process, it was shown that the pipeline paradigm is the most suitable solution as the whole process is performed in a sequence of phases. To take advantage of the face detection parallelization on GPUs specifies that we need to minimize access to devise memory, transfers to and from the GPU must be timed to coincide with computation, and memory accesses to the GPU's global and shared memory must prevent bank conflicts, and branching inside kernels should be kept to a minimum. Our tests have demonstrated that our application can achieve real-time processing rates in different scenarios and it is reliable in various situations. A theoretical study on the reasons why the Viola-Jones algorithm may not allow multi-core platforms to be exploited to their full capacity. Future work includes studying in more detail the aspects and implementing them to verify their efficiency. Also, a more detailed analysis of some of the principles mentioned to take full advantage of the GPU hardware requires further investigation.

References

[1] E. Hjelmås and B. K. Low, "Face detection: A survey," *Computer vision and image understanding*, vol. 83, no. 3, pp. 236–274, 2001.

[2] M.-H. Yang, D. J. Kriegman, and N. Ahuja, "Detecting faces in images: A survey," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 1, pp. 34–58, 2002.

[3] H. A. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 20, no. 1, pp. 23–38, 1998.

[4] S. Yang, P. Luo, C.-C. Loy, and X. Tang, "From facial parts responses to face detection: A deep learning approach," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 3676–3684.

[5] G.-W. Kim and D.-S. Kang, "Improved camshift algorithm based on kalman filter," *Adv. Sci. Technol. Lett*, vol. 98, pp. 135–137, 2015.

[6] D. Hefenbrock, J. Oberg, N. T. N. Thanh, R. Kastner, and S. B. Baden, "Accelerating viola-jones face detection to fpga-level using gpus," in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2010, pp. 11–18.

[7] M. W. Joshua Miguel, Jordan Saleh. Parallelizing face detection in software. [Online]. Available: DepartmentofElectricalandComputerEngineering,UniversityofToronto, 2013.

[8] M. Vanneschi, "High performance computing: parallel processing models and architectures," in *High performance computing*. Pisa University Press, 2014, pp. 1–552.

[9] H. Commin, "Robust real-time extraction of fiducial facial feature points using haar-like features," *arXiv preprint arXiv:1505.04286*, 2015.

[10] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *Proceedings. international conference on image processing*, vol. 1. IEEE, 2002, pp. I–I.

[11] I. Kalinovskii and V. Spitsyn, "Compact convolutional neural network cascade for face detection," *arXiv preprint arXiv:1508.01292*, 2015.

[12] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, no. 2, pp. 137–154, 2004.

[13] S. Ash with Kumar and S. Kubakaddi., "Multiple face detection and tracking using adaboost and camshaft algorithm," *Int. Journal of Research Studies in Science, Engineering and Technology*, vol. 1, no. 1, pp. 8–8, 2014.

[14] N. U. Akhund, T. Md, M. Mahi, J. Nayeen, A. Hasnat Tanvir, M. Mahmud, and M. S. Kaiser, "Adeptness: Alzheimer's disease patient management system using pervasive sensors-early prototype and preliminary results," in *International conference on brain informatics*. Springer, 2018, pp. 413–422.

[15] T. M. N. U. Akhund, W. B. Jyoty, M. A. B. Siddik, N. T. Newaz, S. A. Al Wahid, and M. M. Sarker, "Iot based low-cost robotic agent design for disabled and covid-19 virus affected people," in *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*. IEEE, 2020, pp. 23–26.

[16] T. M. N. U. Akhund, M. A. B. Siddik, M. R. Hossain, M. M. Rahman, N. T. Newaz, and M. Saifuzzaman, "Iot waiter bot: a low cost iot based multi functioned robot for restaurants," in *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*. IEEE, 2020, pp. 1174–1178.

[17] F. I. Suny, M. R. Fahim, M. Rahman, N. T. Newaz, T. M. Akhund, N. Ullah *et al.*, "Iot past, present, and future a literary survey," in *Information and Communication Technology for Competitive Strategies (ICTCS 2020)*. Springer, 2021, pp. 393–402.

[18] T. M. N. U. Akhund, N. T. Newaz, and M. R. Hossain, "Low-cost remote sensing iot based smartphone controlled robot for virus affected people," 2020.

[19] T. M. Akhund, N. Ullah, N. T. Newaz, M. Rakib Hossain, and M. Shamim Kaiser, "Low-cost smartphone-controlled remote sensing iot robot," in *Information and Communication Technology for Competitive Strategies (ICTCS 2020)*. Springer, 2021, pp. 569–576.

[20] N. T. Newaz, M. R. Haque, T. M. N. U. Akhund, T. Khatun, M. Biswas, and M. A. Yousuf, "Iot security perspectives and probable solution," in *2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*. IEEE, 2021, pp. 81–86.

[21] M. Biswas, T. M. N. U. Akhund, M. J. Ferdous, S. Kar, A. Anis, and S. A. Shanto, "Biot: Blockchain based smart agriculture with internet of thing," in *2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*. IEEE, 2021, pp. 75–80.

[22] T. M. Akhund, N. Ullah, S. R. Snigdha, M. Reza, N. T. Newaz, M. Saifuzzaman, M. R. Rashel *et al.*, "Self-powered iot-based design for multi-purpose smart poultry farm," in *International Conference on Information and Communication Technology for Intelligent Systems*. Springer, 2020, pp. 43–51.

[23] T. M. Akhund, N. Ullah, G. Roy, A. Adhikary, A. Alam, N. T. Newaz, M. Rana Rashel, M. Abu Yousuf *et al.*, "Snappy wheelchair: An iot-based flex controlled robotic wheel chair for disabled people," in *Information and Communication Technology for Competitive Strategies (ICTCS 2020)*. Springer, 2021, pp. 803–812.

[24] M. M. Sarker, M. A. I. Shah, T. Akhund, and M. S. Uddin, "An approach of automated electronic voting management system for bangladesh using biometric fingerprint," *International Journal of Advanced Engineering Research and Science*, vol. 3, no. 11, p. 236907, 2016.

[25] M. Biswas, N. U. Akhund, T. Md, M. Mahbub, S. Islam, S. Md, S. Sorna, M. Shamim Kaiser *et al.*, “A survey on predicting player’s performance and team recommendation in game of cricket using machine learning,” in *Information and Communication Technology for Competitive Strategies (ICTCS 2020)*. Springer, 2022, pp. 223–230.

[26] M. A. Hasan and T. Akhund, “An approach to create iot based automated smart farming system for paddy cultivation.”

[27] T. Akhund, “Study and implementation of multi-purpose iot nurse-bot.”

[28] T. Akhund, I. A. Sagar, and M. M. Sarker, “Remote temperature sensing line following robot with bluetooth data sending capability,” in *International Conference on Recent Advances in Mathematical and Physical Sciences (ICRAMPS)*, 2018.

[29] T. Akhund, N. T. Newaz, and M. M. Sarker, “Posture recognizer robot with remote sensing for virus invaded area people,” *Journal of Information Technology (JIT)*, vol. 9, pp. 1–6, 2020.

[30] T. Akhund, “Remote sensing iot based android controlled robot,” *Methodology*, vol. 9, no. 11, 2018.

[31] T. M. N. U. Akhund, N. T. Newaz, and M. R. Hossain, “Low-cost remote sensing iot based smartphone controlled robot for virus affected people,” 2020.

[32] T. M. Akhund, N. Ullah, N. T. Newaz, Z. Zaman, A. Sultana, A. Barros, and M. Whaiduzzaman, “Iot-based low-cost automated irrigation system for smart farming,” in *Intelligent Sustainable Systems*. Springer, 2022, pp. 83–91.

[33] M. Suny, F. Islam, T. Khatun, Z. Zaman, M. Fahim, M. Roshed, M. Islam, R. Jesmin, T. M. Akhund, N. Ullah *et al.*, “Smart agricultural system using iot,” in *Intelligent Sustainable Systems*. Springer, 2022, pp. 73–82.

[34] A. H. Himel, F. A. Boby, S. Saba, T. M. Akhund, N. Ullah, and K. Ali, “Contribution of robotics in medical applications a literary survey,” in *Intelligent Sustainable Systems*. Springer, 2022, pp. 247–255.

[35] T. M. N. U. Akhund, M. Hossain, K. Kubra, Nurjahan, A. Barros, and M. Whaiduzzaman, “Iot based low-cost posture and bluetooth controlled robot for disabled and virus affected people,” *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 8, 2022. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2022.0130879

[36] O. T. C++. Histogram equalization of grayscale or color image. Accessed: April, 2022. [Online]. Available: http://opencv-srf.blogspot.it/2013/08/histogram-equalization.html

[37] R. Ladu, “Gpu: A formal introduction.”

[38] O. T. C++. Histograms - 2: Histogram equalization. Accessed: April, 2022. [Online]. Available: http://docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html{\#}gs

[39] C. U. C. for Advanced Computing. Memory coalescing. Accessed: May, 2022. [Online]. Available: https://cvw.cac.cornell.edu/gpu/coalesced?AspxAutoDetectCookieSupport=1

[40] OpenCV. The opencv reference manual.

[41] M. Danelutto. Distributed systems paradigms and models. [Online]. Available: TeachingmaterialofLaureaMagistraleinComputerScienceandNetworking

[42] NVidia. What is gpu computing. Accessed May, 2022. [Online]. Available: http://www.nvidia.com/object/what-is-GPU-computing.html

[43] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011, vol. 24.

[44] N. Prabhakar, V. Vaithiyanathan, A. P. Sharma, A. Singh, and P. Singhal, “Object tracking using frame differencing and template matching,” *Research Journal of Applied Sciences, Engineering and Technology*, vol. 4, no. 24, pp. 5497–5501, 2012.

[45] A. Jain, J. Bharti, and M. Gupta, “Improvements in opencv’s viola jones algorithm in face detection-tilted face detection,” *International Journal on Signal and Image Processing*, vol. 5, no. 2, p. 21, 2014.