Parallelized Chromaticity-Based Shadow Detection and Removal

Madeleine Armstrong, Coale Cooper, Rituparna Dey

mbarmstrong@email.arizona.edu, cjcoopr@email.arizona.edu, rdey1@email.arizona.edu

Group 2

Brief (less than 200 words) technical abstract of the proposed project	The presence of shadows in an image can negatively impact computer vision applications that depend on accuracy of object recognition. Identification and removal of shadows is thus critical in surveillance applications involving target detection and tracking. As the amount of data involved in image and video manipulation rapidly increases, new levels of computational power and algorithmic efficiency become necessary. Several known methods exist for shadow detection removal based on different features of color and image processing. In this study we propose a chromaticity-based shadow removal algorithm based on an existing serialized Matlab code. By using the CUDA computing platform and API, we aim to achieve speedup by parallelizing the algorithm on a NVIDIA Tesla P100 GPU.
Brief (less than 200 words) discussion of anticipated benefits	Compared to the serialized version of the code, we anticipate a significant speedup with a parallelized approach. Using the high-performance capability of the Tesla P100 GPU allows the innumerous pixel-wise calculations to be performed simultaneously, resulting in faster execution without compromising output quality. Our algorithm will result in modified images without the detrimental effects of illumination and shadows. Applications like the Controlled Environment Plant Production (CEPP) that rely on computer vision sensing are made simpler and more efficient using sophisticated shadow removal algorithms.
A maximum of eight key words that describe the project	shadow, parallelism, GPU, CUDA, image processing, chromaticity, Otsu's method

1 IDENTIFICATION AND SIGNIFICANCE OF THE PROBLEM OR OPPORTUNITY

1.1 Statement of Problem

Image processing utilizes computer algorithms to efficiently process the large amounts of raw pixel data needed for image/video identification and manipulation. The accuracy of these systems may be impacted by unwanted conditions such as noise or light, for which human vision automatically adjusts or corrects. Shadows are one such condition that may alter or conceal necessary parts of an image. In video surveillance applications where target detection and object tracking are crucial, it is beneficial to digitally remove shadows for better image recognition.

The application that this study focuses on is plant surveillance and contact sensing using computer vision. This is for use in Controlled Environment Plant Production (CEPP) systems within greenhouse settings, where computer vision monitors and analyzes the physical characteristics and needs of plants. When done by humans, this process is labor-intensive and costly; computer vision thus provides an effective alternative. The images that we use for this study come from this application.

Problem Background

The shadow removal algorithm selected for this study is based on [1]. This study describes a chromaticity-based shadow detection and removal algorithm which consists of ten steps, outlined in Figure 1, to produce a Red, Green, Blue (RGB) shadowless image. These ten steps are characterized in five key processes: (1) colorspace transformation, (2) thresholding and Otsu's method, (3) convolution, (4) erosion, and (5) result integration.

These five processes perform multiple sets of calculations using the pixel data of the entire image, with many of these calculations being the same across every pixel. This motivates using the parallel power of the GPU to reduce the computation time. For each of the five processes, there are multiple optimization areas where implementing GPU kernels will take advantage of the repeated calculations. Then combining the accelerated kernels, an end-to-end shadow removal solution is captured, resulting in a significant reduction of processing time.

1.2 Current Level of Technology

As stated above, the approach for this shadow removal algorithm is well studied in [1]. The algorithm has not only been shown to effectively remove shadows, it has also been shown to benefit from GPU acceleration. Secondly, there exists a sequential implementation in Matlab for this project to reference as a starting point and proof of concept.

Current methods in shadow detection can be classified according to feature into four categories: chromaticity, physical, geometry, and texture [3]. This paper will use a chromaticity-based approach, meaning utilizing a measure of color that is independent of intensity. Many chromaticity methods are relatively simple and computationally inexpensive but may produce more noisy results due to the calculations being performed at a pixel level [3].

Chromaticity-based shadow detection and removal algorithms involve color space transformations and pixel-wise computations.

1.3 Limitations of Current Technology

Identifying and adjusting for shadows in an image is computationally expensive, requiring techniques such as pixel-wise transformations and analysis of small "neighborhoods" of pixels. As the technology for image capturing and resolution evolves, the amount of pixel data and computations demand faster and more sophisticated methods of image processing. The parallel structure of GPUs allows blocks of data to be processed simultaneously, greatly increasing the speed of algorithms for high-resolution image and video processing.

1.4 Proposed Solution

In this work, we will use the chromaticity-based shadow detection and removal algorithm demonstrated in [1]. We will duplicate the same shadow removal algorithm by implementing their five processes: (1) colorspace transformation, (2) thresholding and Otsu's method, (3) convolution, (4) erosion, and (5) result integration. These five processes will be mapped to several GPU kernel implementations using efficient optimization methodologies studied in classes and discussed in [1], such as use of shared memory instead of global memory, careful selection of tiling methodologies to maximise shared memory usage, coalesced read and write, use of proper exit conditions in conditional looping etc. Then, utilizing and parallelizing the provided Matlab code, we will directly compare the speedup of our GPU implementation to the demonstrated speedup of 21.67x on an 18 megapixel image found in [1].

In the following subsections the details of the proposed solution are defined. First a background is given in section 1.4.1 followed by a discussion of detailed methodology of the solution in 1.4.2. Section 1.4.3 then relates the solution to current technology and finally 1.4.4 proposes a validation and evaluation strategy.

1.4.1 Background

Since we are trying to implement a optimised GPU based high performance shadow removal algorithm, we have decided to go with the chromaticity based algorithm proposed in [1]. The methodology proposed in [1] involves wide spread pixel-wise transformations ideal for parallelization on GPUs and high optimization methodologies such as Global memory usage optimization, efficient use of tiling methodology, efficient thread divergence handling, effective use of exit conditions in conditional loops, making use of coalesced memory read and write wherever needed etc.

We referred to the below papers/studies as well before arriving at our decision to use the solution provided in [1] as our implementation algorithm.

- Silva et al. [6], implemented an altered chromaticity-based shadow detection and removal algorithm on a GPU where they are able to achieve shadow removal in 0.03685 seconds on a 1024x1024 image.
- Macedo et al. [4], proposed another chromaticity-based algorithm on GPU parallelization where the image is split into different channels and each channel is binarized using Otsu's method [5] before being used to create a shadow mask of the image. This method is suitable for shadow detection only.
- Wu et al. [7] suggested a shadow removal algorithm for moving targets based on HSV and texture features . First, the foreground is detected by background subtraction and the shadow is detected by HSV color space transformation. The extracted background is then processed by the local variance and OTSU method. Since the local variance and the OTSU threshold not only work on moving objects, but also eliminate the shadow, the moving target is obtained by OR operating on the processed results of local variance and OTSU method. This method works well for shadow removal in video processing.

1.4.2 Statement of Solution

In this study, we will duplicate the methodology outlined in [1] for the GPU implementation of the chromaticity-based shadow detection and removal algorithm. Figure 1 shows the five intermediate steps of this algorithm.

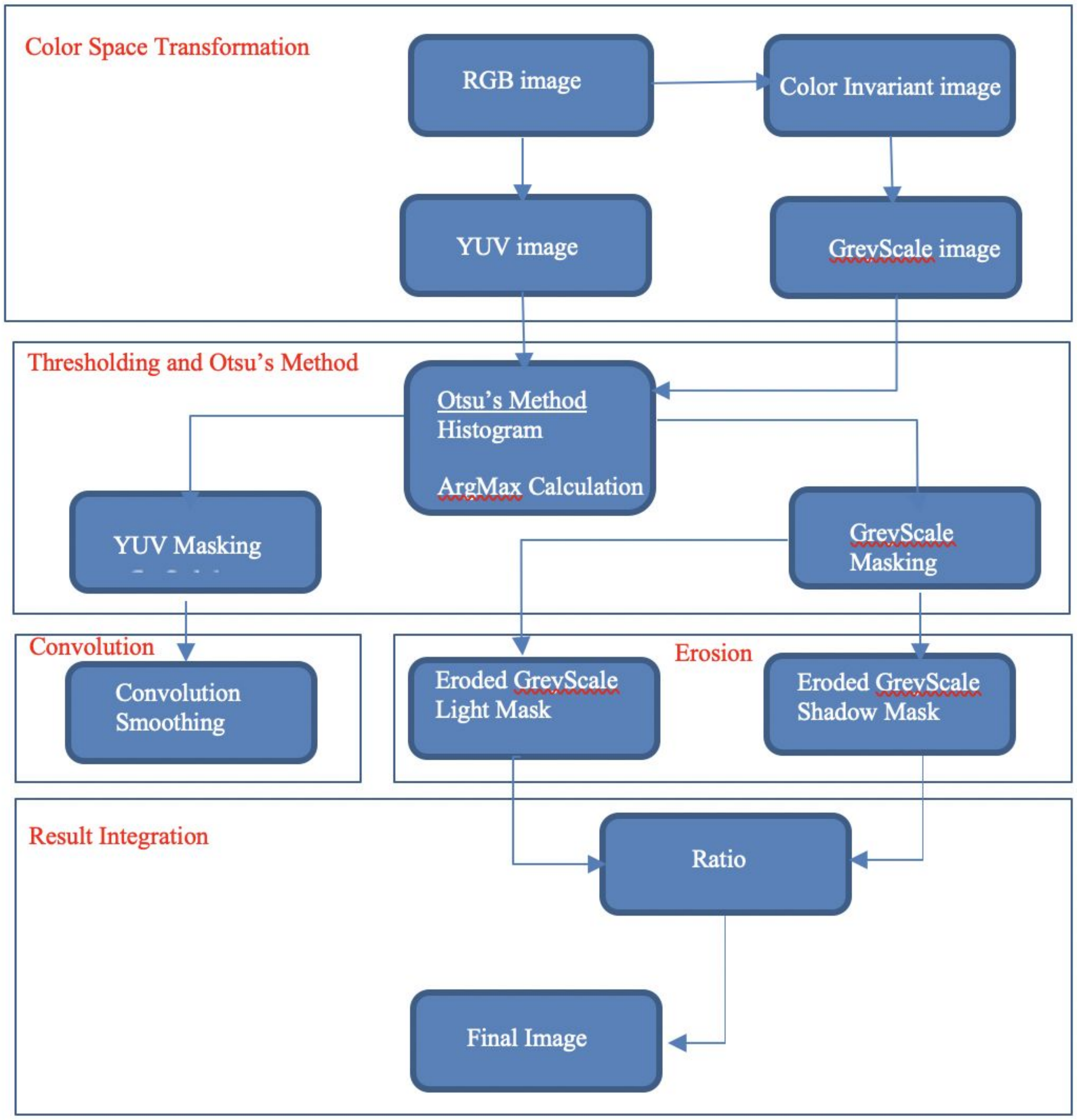

Figure 1: Flowchart of algorithm steps [1]

1) Colorspace Transformation: This process takes in the original RGB image and generates YUV and grayscale versions for use in following steps. The greyscale image is formed by first using an invariant image that is generated from the original RGB input image. The color invariant (CI) image is then converted into a grayscale image.

A single GPU kernel will be constructed to read in the single input RGB image and then output the two converted images. The single kernel is chosen to avoid the transfer of the input image multiple times to the GPU's global memory. In addition the memory layout of the input and output images will be considered carefully to take advantage of coalesced reads and writes from global memory. No optimizations will be performed on shared memory as every thread operates on independent data from a single pixel.

2) Thresholding and Otsu's Method: After the colorspace transformation, two masks will be generated using Otsu's method. One mask is generated from the grayscale image and the other mask is generated using the U component of the YUV image.

To create the masks first a histogram of pixels is computed using the color converted images. This histogram is an approximation of the probability density function of pixel intensities and is the input to Otsu's method. After creating the histogram, the problem can be viewed through a probabilistic lens by analyzing the zeroth (Equation 1) and first (Equation 2) order cumulative moments of the image up to the threshold k

$$\omega(k) = \sum_{i=1}^{k} p_i \qquad (1)$$

$$\mu(k) = \sum_{i=1}^{k} i p_i \qquad (2)$$

To determine the quality of a given threshold *k*, Otsu defines two variance measures: the between-class variance ($\sigma(k)$) and the total variance (σ_{T2}), which are defined using Equations 3 and 4.

$$\sigma_B^2(k) = \frac{(\mu_T \omega(k) - \mu(k))^2}{\omega(k)(1 - \omega(k))} \qquad (3)$$

$$\sigma_T^2 = \sum_{i=1}^{L} (i - \mu_T)^2 p_i \qquad (4)$$

where μ_T is the mean of the entire histogram. A good threshold will have a high variance between classes relative to the variance of the entire histogram. Using this intuition, a "goodness" metric of the threshold can be defined as shown in Equation 5.

$$\eta(k) = \frac{\sigma_B^2(k)}{\sigma_T^2} \qquad (5)$$

The threshold value returned by Otsu's method to create the masks, is the k that maximises the function in Eq. 5 mentioned above.

In the GPU implementation, the algorithm will be split into six kernels mimicking the approach used in [1].

- Kernel 1: Takes in the single-channel image and creates a histogram of pixel intensities.
- Kernel 2: Uses the histogram of pixel intensities to obtain $\omega(k)$ as defined in Equation1.
- Kernel 3: Uses the histogram of pixel intensities created to obtain $\mu(k)$ as defined in Equation 2.
- Kernel 4: Calculates $\sigma_B^2(k)$ for every bin in the histogram (every possible threshold).
- Kernel 5: An argmax kernel which is used to calculate the threshold value of *k* that maximizes $\sigma_B^2(k)$.
- Kernel 6: Takes the single-channel input image and the threshold and creates a binarized image based on whether the pixel was less than or greater than the calculated threshold.

When implementing each of these 6 kernels it is noted that kernels 1 and 6 are the only computations executed on the input image size. Kernels 2,3,4 and 5 are a function of bins in the histogram which provides a significantly smaller amount of opportunities for GPU thread parallelization.

For kernel 1 a parallelized histogram generator optimization will be explored. The key consideration here is memory collision that will occur when binning the large amount of pixels into a relatively small amount of bins. The more collisions that occur the more serial the implementation becomes due to the amount of threads waiting on other threads to complete writing to memory. It should be noted that [1] reported a slowdown when implementing this kernel relative to the serial matlab implementation.

Kernel 6 provides an opportunity for parallelization but is relatively simple to implement as every thread is mapped to a single pixel which gives no opportunity for optimizing shared memory. The only consideration here will be making sure the input and output image take advantage of coalesced reads and writes.

Kernels 2, 3, 4 and 5 have less opportunity for speedup however group 2 questions why these kernels were implemented separately and not as a single execution. Equations 1-5 can be computed in a single kernel call where the only return value is the maximum threshold used for the binarized image. This single kernel will be explored and compared to the individual kernels 2-5 used in [1].

3) Convolution: In this step we generate the convolution-pixel, by summing up the selected image pixel, and its neighboring pixels, multiplied and accumulated with the kernel matrix as shown in Equation 6.

$$f[x,y] * g[x,y] = \sum_{i=1}^{n} \sum_{k=1}^{m} f[i,k] * g[x-i, y-k] \qquad (6)$$

In this equation g[x,y] represents the image matrix and f[x, y] represents the kernel matrix that will be sliding across each pixel of g[x,y].

The convolution implementation of [1] explored three versions: global history convolution, shared-memory 2D convolution and shared-memory 1D convolution.

The global convolution will require each thread to bring in multiple pixels. This will be rather inefficient by placing significant stress on the global memory.

To reduce the global memory access, a 2D shared memory convolution will be implemented. A tiling approach will be beneficial here due to the amount of predictable data accesses used by each computation. Each thread will need access to a set of neighboring pixels in order to do its respective calculation. The key here is neighboring pixels. This will allow the implementation to make use of the shared memory space so all threads can bring in pixel data and share that data with other threads in the corresponding block. This has the opportunity to significantly reduce the overall global memory accesses. Careful consideration will be made when selecting a tile size to fit completely in the shared memory space.

We will also implement 1D convolution by breaking up the computation into a series of row-tiles and column-tiles, rather than 2D tiles. The first kernel executes the row-wise convolution followed by the second kernel executing the column-wise convolution. Once both are completed, the final result will be written back into global memory. The 1D convolution kernels will also make use of the shared memory space to maximize global memory access.

A timing analysis of these three implementations will be compared and the one with the lowest execution time will be selected.

4) Erosion: Image erosion process is quite similar to that of image convolution and involves passing a filter over the data that reduces the value of each pixel based on interactions with neighboring pixel values. In parallel with the 2D smoothing process, we take the output masks from Otsu’s method and pass them through an erosion process to produce two closely related masks, a light mask and a dark mask, that have slight overlap in the transition region between light and shadow and can be used to selectively perform operations on the light or dark regions of the image, respectively.

Based on NVIDIA profiler results,will optimise the process of moving data from global to shared memory by applying a tiling process, with individual blocks loading padded chunks of data from the global memory and working privately on that. We will perform a final write back to the global memory only upon completion of all threading activity. Through declaring structural element as constants, we've optimized caching of the structural element that must be read by every thread. In this step each output pixel will be assigned a thread that will gather all relevant pixels from global memory and perform the erosion process with what it fetched.For optimisation, will be using a flattened 1D shared memory access since using shared memory as a 2D array appeared to generate extra load instructions for each of the accesses.

5) Result Integration: In this final step, we first create six maps of the eroded shadow and multiply light masks by the RGB of the input image. Next we accumulate these six maps along with the eroded shadow and lights masks, which requires a total of eight reductions to get the average value of each array. We then use these average values to create our RGB ratio values. Finally, we use the RGB ratio values to map with the input image to remove the shadow and create our final output.

The result integration is mapped to four different kernels as follows:

- Kernel 1: Maps the input image multiplied by the shadow and light masks
- Kernel 2: Sums up the light arrays, shadow array and the eroded array
- Kernel 3: Calculates the Red, Green and Blue ratios from the sum obtained in Kernel 2 to produce the shadowless image
- Kernel 4: Uses the Red, Green and Blue ratios produced in Kernel 3 and the input image to remove the shadow and create the final output image.

Kernels 1 and 4 have independent threads and so they can be parallelised. Also these kernels can benefit from coalesced read by implementing reordering the input image into Red, Green and Blue arrays.

Kernel 3 will be using reduction methodology to sum up the input arrays. In this process each thread will sum up two elements and save to shared memory. By using zero padding in shared memory, will be reducing thread divergence. Also use of shared memory indexing should reduce shared memory conflict and thread divergence.

1.4.3 How Solution Overcomes Current Technology

As our main focus is to realize a low-latency and high-throughput shadow removal, we propose a chromaticity- based algorithm. This choice compliments the utilization of GPUs, as a chromaticity-based algorithm primarily consists of pixel-wise transformations that are suitable for parallelization on GPUs. The goal here is not focused on creating the 'best' algorithm at removing shadows but to use an existing well understood algorithm in [1] to achieve the same speedup of 21.67x on an 18 megapixel image compared to the same method implemented in Matlab.

The speedups of [1] are listed in Table 1 for each phase of the shadow removal algorithm.

Table 1: Predicted speedups

Process	MATLAB	CUDA-C	Speedup
Colorspace Transformation	2.0544	0.00996	206.265x
Threshold & Otsu's	0.0914	0.00820	11.146x
Convolution	0.0768	0.07177	1.0701x
Erosion	0.339	0.06091	5.5656x
Result Integration	1.130	0.02095	53.9379x

1.4.4 Validation and Evaluation Strategy

There will need to be multiple considerations when evaluating the performance of the newly implemented GPU accelerated shadow removal algorithm. First, the algorithm needs to maintain the same effectiveness in removing shadows as the baseline Matlab implementation. The team will run the same image through both the baseline and the new GPU accelerated implementations. The output will be verified by eye as an initial check of correctness. A pixel-by-pixel comparison will then be conducted between the two output images to quantify any difference in shadow removal effectiveness. These steps will then be repeated on other images of multiple sizes, each containing a varying degree of shadows.

Second, and the most important evaluation, will be the performance of the GPU implemented shadow removal algorithm. The team will gather a timing analysis of the Matlab serial implementation and each of the 5 sub processes of the algorithm. A respective timing analysis of the GPU implementation and subprocess will then be collected for comparison directly to the serial implementation. The timing analysis will be repeated on five different images provided by the CEPP application of dimensions 5226x3484, 3269x2268, 472x595, 624x468, and 4540x4548.

In addition, the NVIDIA profiler will be used throughout each kernel's development and integration in order to capture the efficiency of the GPU code. This evaluation could consist of but is not limited to, global and shared memory utilization, compute vs memory utilization and thread divergence metrics.

2 Technical objectives

This solution shall provide an accelerated shadow removal algorithm written in C, using the CUDA API. The new implementation will be derived from the study conducted in [1] using reference shadow removal code constructed in Matlab. The Matlab code and the timing analysis contained in [1] will act as a baseline reference to quantify the effectiveness of group 2’s GPU implementation with the goal of achieving the same speedup of 21.67x on an 18 megapixel image obtained in [1]. The team will end with a report and presentation summarizing the approach and speedup results, comparing this directly to the findings from [1].

2.1 Project Milestones and Tasks

2.1.1 Understanding the shadow removal algorithm

The team’s first step is to understand all computations that make up the complete shadow removal algorithm. Here the goal is to develop a detailed breakdown of the techniques and characteristics used to implement this algorithm. A current implementation has been provided in Matlab which will be used as a reference point for the team. In addition, using the Matlab code, a timing analysis will be run to establish baseline metrics to be compared against the future GPU implementation.

Tasks:

- Understand color space transformation - **Madeleine**
- Understand erosion - **Madeleine**
- Understand convolution - **Coale**
- Understand result integration - **Ritu**
- Understand Otsu method - **Coale**
- Timing analysis - **Ritu**

2.1.2 Host-only implementation

Here, the team will implement a host-only version of the shadow removal algorithm. This host only solution will be written in c which will provide the structure of the teams implementation in which GPU kernels will later replace the serial implementations.

Tasks:

- Implement input image reader - **Ritu**
- Implement serial color space transformation - **Madeleine**

- Implement serial erosion - **Madeleine**
- Implement serial convolution - **Coale**
- Implement serial result integration - **Ritu**
- Implement serial Otsu method - **Coale**

2.1.3 Implement kernels in CUDA

Each team member will individually be responsible for designing and implementing a kernel or set of kernels to optimize a piece of the shadow removal computation Multiple different optimization methods discussed in this class must be explored and each kernel will need to be profiled throughout the development process. Team members shall provide justification for their chosen implementation based on running the NVIDIA profiling tools and provide data driven analysis on the results.

Tasks:

- Implement kernel for color space transformation - **Madeleine**
- Implement kernel for erosion - **Madeleine**
- Implement kernel convolution - **Coale**
- Implement kernel result integration - **Ritu**
- Implement kernel Otsu method - **Coale**

2.1.4 Integrate kernels together

Once the individual kernels are completed they will be integrated into the host only implementation to replace the respective serial sections of the code.

Tasks:

- Team effort to coordinate integration

2.1.5 Profile complete GPU algorithm implementation

Following completion of the integration a detailed analysis of the entire shadow removal algorithm will be conducted using the NVIDIA profiling tools. Metrics will be gathered and analyzed for comparison against the baseline. At this point it may be necessary to revisit some of the individual kernels to make some small optimization improvements.

Tasks:

- Team effort to profile integration

2.1.6 Compile report

Finally, a report and presentation will be created summarizing the team's findings.

Tasks:

- Each team member write section on implementation of respective kernel
- Schedule class presentation

2.2 Work Plan Schedule

Table 2: Task schedule

Task	**Description**	**Weeks**					
		1	**2**	**3**	**4**	**5**	**6**
	Understanding the alg.	x					
	Host only implementation		x	x			
	Implement CUDA kernels			x	x	x	
	Integrate kernels					x	
	Profile completed GPU alg.					x	x
	Write final report					x	x

References

[1] A. Akoglu, J. Mack, R. Raettig, E. Richter, B. Unal and S. Valancius, “Accelerated Shadow Detection and Removal Method,” *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*, Abu Dhabi, United Arab Emirates, November 3-7 2019, pp. 1-8, doi: 10.1109/AICCSA47632.2019.9035242.

[2] P. Kumar, A. Lee and K. Sengupta, "A comparative study of different color spaces for foreground and shadow detection for traffic monitoring system," *Proceedings. The IEEE 5th International Conference on Intelligent Transportation Systems*, Singapore, 2002, pp. 100-105, doi: 10.1109/ITSC.2002.1041196.

[3] A. Sanin, C. Sanderson and B. C. Lovell, “Shadow detection: A survey and comparative evaluation of recent methods,” *Pattern Recognition*, vol. 45, no. 4, pp. 1684-1695, April 2012.

[4] M. C. F. Macedo, V. P. Nascimento, and A. C. S. Souza, “Real-time shadow detection using multi-channel binarization and noise removal,” *Journal of Real-Time Image Processing*, June 2018.

[5] N. Otsu, “A threshold selection method from gray-level histograms,” *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, Jan 1979.

[6] G. F. Silva, G. B. Carneiro, R. Doth, L. A. Amaral, and D. F. de Azevedo, “Near real-time shadow detection and removal in aerial motion imagery application,” *ISPRS Journal of Photogrammetry and Remote Sensing*, 140:104 – 121, 2018.

[7] M. Wu, R. Chen and Y. Tong, "Shadow Elimination Algorithm Using Color and Texture Features", *Computational Intelligence and Neuroscience*, vol. 2020, Article ID 2075781, 2020, https://doi.org/10.1155/2020/2075781.