TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

AZAZ AHMAD

GENETIC ALGORITHMS BASED CAMERA AUTOFOCUS OPTIMIZATION

Master of Science thesis

Examiner: Prof. Karen Egiazarian
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 8th May 2013

ABSTRACT

AZAZ AHMAD: Genetic Algorithms Based Camera Autofocus Optimization
Tampere University of Technology
Master of Science Thesis, 43 pages, 4 Appendix pages
March 2016
Masters Degree Programme in Information Technology
Major: Signal Processing
Examiner: Professor Karen Eguiazarian (Egiazarian)

Keywords: Mobile camera, Autofocus, Optimization, Genetic Algorithms

Autofocus is critical for a camera system due to its massive impact on image quality. It is essential to get correct focus on the region of interest without user interaction. Contrast based focusing is the most prevalent form of focusing which uses statistics from the image signal processor (ISP) to guide lens movements. An autofocus system consists of numerous hardware and software components and each component is especially optimized. Autofocus system design consists of repetitive and tedious field tests on real scenes. This approach, however, is very time consuming and laborious.

This thesis presents an optimization methodology to expedite autofocus design and improve camera performance. We propose the use of genetic algorithms (a branch of evolutionary algorithms) to improve autofocus. Genetic algorithms are derived from the biological model of evolution and natural selection. In this thesis, we create an environment in which potential solutions can evolve. We demonstrate the effectiveness of our method by optimizing focus kernel and step-length of a camera lens. The fitness of these parameters is measured using the contrast from focus statistics, shape of the focus statistics curve and time taken for focusing.

Input images from different ambient conditions are captured using consumer phones and are utilized in optimization to get effective clues for autofocus. Autofocus design resulting from our methodology is tested in retail phones to verify its application and efficiency. Improvements in autofocus are observed including enhanced contrast extraction and reduced time-to-focus. Biggest performance gain is seen in low-light as the available contrast is low and it becomes even more important to obtain better focus statistics. Our autofocus design results are deployed to commercial camera phones which proves the effectiveness. Optimizing camera autofocus is a very industry specific topic and this thesis presents a possible solution to this optimization problem.

PREFACE

This thesis study was performed while working in Nokia Imaging Organization and Microsoft Camera Team at Tampere, Finland.

I would like to thank my colleagues for providing an inspiring atmosphere and answering my questions. I wish to express my gratitude to Markus Vartiainen who was my mentor throughout my thesis work. I want to thank Professor Karen Egiazarian from Tampere University of Technology who was the examiner of this work and provided useful tips during the writing process.

Finally, I want to thank my family for their constant motivation and encouragement as none of this would have happened without their love and faith in me.

Tampere, March 16, 2016

Azaz Ahmad

TABLE OF CONTENTS

TERMS AND DEFINITIONS

AEC	Auto Exposure Control
AF	Autofocus
AWB	Auto White Balance
BBPSO	Bare-Bones Particle Swarm Optimization
CDAF	Contrast Detection Autofocus
DSLR	Digital Single Lens reflex
DSP	Digital Signal Processor
EA	Evolutionary Algorithm
GA	Genetic Algorithm
HCS	Hill Climbing Search
IQ	Image Quality
IR	Infra-Red
ISP	Image Signal Processor
JPEG	Joint Photographic Experts Group
NN	Neural Network
OEM	Original Equipment Manufacturer
PSO	Particle Swarm Optimization

1. INTRODUCTION

1.1 Motivation and Objective

In today's world digital cameras are ubiquitous. Phones, watches, spectacle glasses, security checkpoints etc. are a few examples of extensive camera usage. Smartphones are at the fore front of digital camera revolution. There are estimates that more than 1 trillion images were captured in 2015, thanks to the cameras in our phones [1]. To satisfy this consumer savviness, image quality (IQ) is the number one concern for all camera manufacturers. Image quality has to be tuned specifically for individual camera systems which is a laborious yet important task.

Modern digital cameras consist of numerous hardware and software components. After the raw data is captured by the image sensor, it is sent to the Image Signal Processor (ISP). A number of algorithms are applied over this data before we get the final image. This set of algorithms is normally referred to as Image Processing Pipeline. Schematic diagram of a typical image processing pipeline is shown in Figure 1.1.

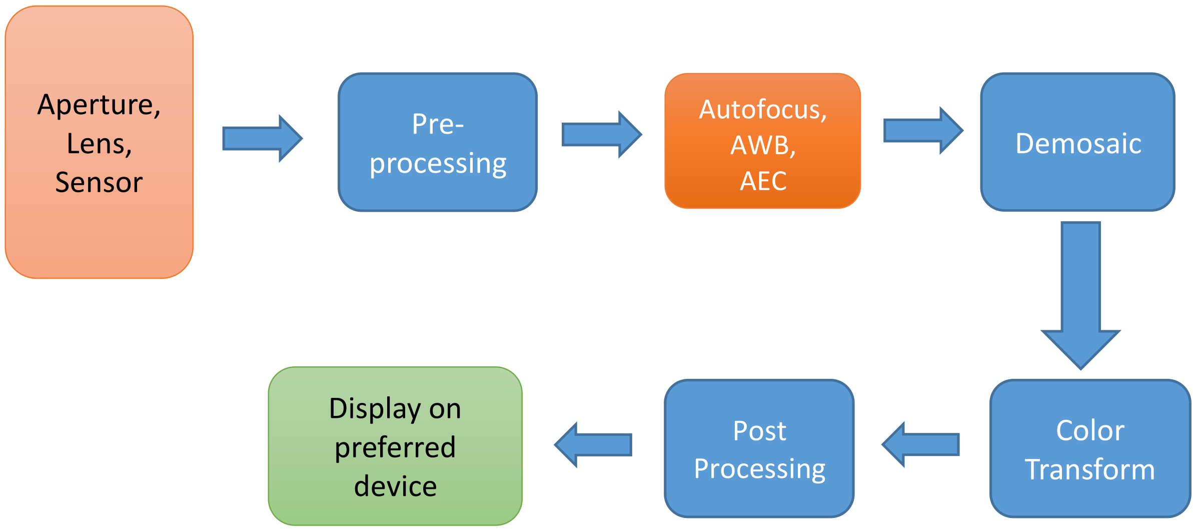

Figure 1.1: *Digital Image Processing Pipeline.*

Each component of this image processing pipeline is individually optimized by the camera manufacturer to get the best possible outcome from this imaging device. An important block of this pipeline is *autofocus*. Autofocus is crucial for a camera system due to its huge impact on perceived image quality. Failure of autofocus is one of the top reasons driving consumer dissatisfaction [2]. One approach for autofocus optimization is to perform repetitive, time-consuming field tests on real scenes. Although this approach gives us acceptable results eventually, manual optimization usually requires several iterations and refinements from the camera development team.

The *objective* of this thesis is to present an alternate methodology for optimizing autofocus design. The proposed method would save significant time and effort of camera engineers, helping them focus on algorithmic developments rather than optimization problems. In success, the resulting design parameters generated by this system would improve autofocus performance in commercial camera phones. The system would work seamlessly for multi objective optimizations. Capability to readily add new criteria for fitness evaluation would open exciting opportunities for prototyping solutions targeting difficult situations such as low-light conditions. This system, once implemented, would require minimal efforts of camera engineers for optimizing future hardware.

To achieve the goal, we will use genetic algorithms (GAs), a branch of evolutionary algorithms (EAs). Genetic algorithms were introduced by Goldberg and Holland in 1988 [3]. In Section 2.4, we present prior art showing the use of GAs in the camera domain. We will build upon current research in genetic algorithms to optimize camera autofocus. Another *objective* of this work is to present a criterion for comparing image sharpness. In the camera domain, subjective testing holds critical importance. Engineers compare images and decide which image is visually more pleasing. This is challenging as there is no fixed metric for image quality and individual preferences for image quality lead to subjective opinions. Consequently, the results could be biased and more inclined towards the individual taste of the tester. In this thesis, we try to address this problem by comparing parameters against each other objectively using a numerical *fitness value* associated with each solution candidate. Our optimization method can be extended to other scenarios where laborious manual optimization is performed. By implementing this system, we also discovered the limitations posed by genetic algorithms to camera optimization. These will be presented towards the end of the thesis (Section 3.3).

1.2 Structure of the thesis

This thesis consists of one theory chapter, one implementation and analysis chapter, and a conclusion chapter towards the end. Chapter 2 consists of the theoretical background. The reader will be given a brief overview of image processing in a phone camera. Autofocus functionality is presented to help the reader understand parameters affecting the autofocus system. Afterwards, an introduction to genetic and evolutionary algorithms is presented as the optimization framework implemented for this thesis will be based on genetic algorithms. Some existing optimization methods based on genetic algorithms are also presented. Chapter 3 consists of implementation details of the autofocus optimization system designed for this thesis. We will present important building blocks of this system and show how theoretical ideas of genetic algorithms are transformed to practical use in this system. We will present the experimental results and discuss important results derived from this work. Chapter 4 ends the thesis by presenting conclusions.

2. THEORETICAL BACKGROUND

The aim of this thesis is to help the imaging engineers with camera optimization. A mobile phone camera has plenty of tunable parameters and the image quality depends upon the optimization of these parameters. In this chapter, we will provide brief introduction towards camera imaging pipeline and the crucial role of autofocus in image capturing. Later on, we will present brief introduction of Evolutionary and Genetic Algorithms. In the last section, some recent research methods for optimization using genetic algorithms are presented.

2.1 Overview of Image Processing in Digital Cameras

A digital camera consists of several software and hardware modules. It captures scenes and preserves them for later consumption. Digital cameras have completely revolutionized our moment capturing. There are many different types of digital cameras. Digital Single Lens reflex (DSLR) are for more serious photographers for high quality shots. Digital cameras on our smartphones proved to be the game changer. Millions of photographs are taken every day because of smartphone cameras. Action cameras are another popular genre which has seen a huge interest from consumers recently. These days we also have cameras in watches and spectacle glasses. Since this thesis is about autofocus parameters in a phone camera, we will keep the emphasis of the discussion on phone cameras.

The hardware of a modern digital camera borrows many characteristics from the original pinhole camera. Having said that, modern digital cameras have evolved considerably from their predecessors. An image capturing system consists of a lens system, an image sensor to convert light into electrical signals, a processor to digest the incoming information and an output medium. The scene is illuminated usually by the ambient lighting which maybe sunlight or room lighting. In not-so-bright conditions, the flash of the camera can be used to illuminate the scene. The light reflected by the scene enters the lens system and the lens focuses the light beams onto the image sensor [4][5]. The image sensor converts the photons of light into voltage and sends this raw data to a Digital Signal Processor (DSP). In camera imaging, we call this processor Image Signal Processor (ISP). The ISP will consume the raw data from the sensor and produce statistics which will be used by different imaging algorithms for image perfection. The data from ISP is fed to algorithms including Auto white balance (AWB), Auto Exposure Control (AEC) and Autofocus (AF) among others. Finally, the image is displayed on a display module. The display module doesn’t only display the final image but is used as a viewfinder during capture as well. The image is enhanced using multiple enhancement algorithms. The final image is encoded in a compressed format and is saved to memory.

The lens system is a very important part of the digital camera. Normally there are a number of lenses packed into a lens system but the whole set is many times referred to simply as a lens. The purpose of the lens is to direct light onto the imaging sensor. The lens controls light using reflection and refraction properties of light. A typical lens system consists of 2-3 lenses, an aperture which may be fixed or movable, and some filters. The aperture of the lens controls the amount of light entering the camera. It plays a very important role in image capturing as exposure and depth of field both are functions of aperture. Mobile phone cameras have usually fixed aperture due to size and cost restraints. The size of the aperture is defined as F-number, where smaller number designates wider aperture [6]. The lens system contains Infra-red (IR) filter to reduce IR rays from reaching the sensor. The IR rays naturally affect the voltage on the sensor as the sensor has significant sensitivity to the IR rays and will produce errors. Hence we use IR filter to block them. The light from the lens system lands on the image sensor which converts light rays into digital signals. Sensors are basically semiconductor devices having a light sensitive surface. In digital cameras, the material and design is so chosen that only visible spectrum reaches the photosensitive part of the sensor.

After a raw image is captured by the sensor, there are errors in the captured data as the image capturing mechanism is not perfect. We apply a number of corrections before feeding the data to other algorithms. These corrections include Pixel Correction, Linearization, Noise reduction and Shading and distortion correction on the input data. After preprocessing, white balance correction is applied on the data. Demosaicing and color transform happens afterwards. In the post processing, we apply Gamma correction, Sharpening and enhancements to make the image look more pleasing. There are many variations for these algorithms which can be found in literature. Interested readers can refer to [8] for details.

2.2 Autofocus

Focusing is a fundamental part of the camera image pipeline. Focus of the bundle of rays is the point from which a portion of light rays diverges or converges. “3A” is a common term in imaging which refers to Auto White Balance, Auto Exposure and *Autofocus*. Autofocus helps the user to get the correct focus on the subject of interest such that the user does not have to adjust the focus manually. Autofocus is critical for a camera system due to its massive impact on perception of usability and image quality [9]. If an image is not in proper focus, the whole image is ruined no matter how good everything else performed. The human eye is very sensitive to blurriness and an un-sharp image is very unpleasing. The topic of Focusing is somewhat subjective and people may have different opinions whether something is in focus or not. But majority of the times, we can agree on focused and un-focused images. Autofocus (AF) systems can be broadly categorized into two sub groups: Active and Passive. Active Focusing measures the distance to the subject independently of the optical system. Examples include:

- Ultrasonic sound waves
- Infrared light
- Laser method

Passive focusing measures distance to the subject by analyzing the captured image. Scene is captured by sensor and is then analyzed. Examples are:

- Contrast based method
- Phase detection

When considering focusing techniques, many factors play a role in deciding which technique to use. Speed of focusing, accuracy of focus, cost and power consumption of the focusing system are some of the main things to consider when choosing which technique to go with. Apart from autofocus, manual focus can be used by the users to focus on objects of their liking or in those situations where automatic focusing is failing repeated.

Contrast based focusing is the most prevalent form of focusing. It falls under the category of passive focusing. This technique requires no extra hardware. Once an algorithm for this technique has been developed, this algorithm can be used for high end as well as low end cameras. The algorithm needs data from the sensor and applies the algorithm on the input data. The main concept in contrast based focusing is that a sharp image contains more high-frequencies than an un-sharp image [6]. Contrast based focusing can be understood with the help of a curve where focus values are plotted. This curve is referred to as the focus curve. Figure 2.1 shows an example focus curve.

Focus value is used to estimate the amount of sharpness of the image and is calculated as:

$$FV = \sum_x \sum_y (G_x^2 + G_y^2) \qquad (2.1)$$

where G_x and G_y are gradients in x and y directions respectively [9].

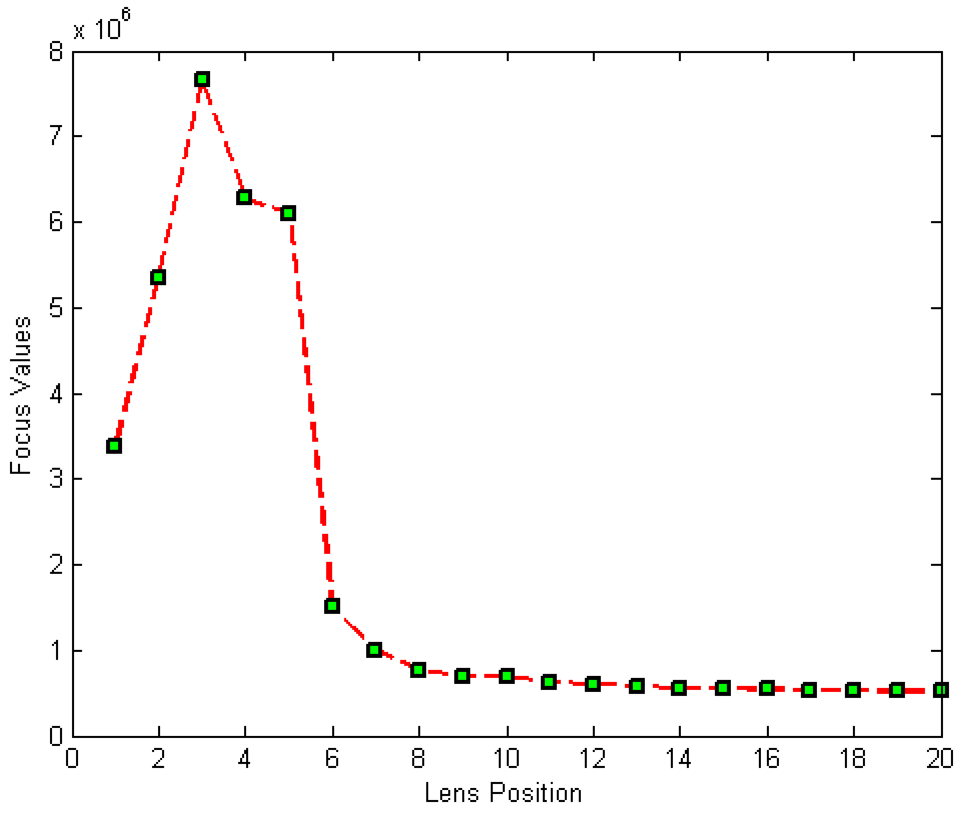

Figure 2.1: *A typical focus curve, showing focus values against lens movement.*

The focus values are plotted as the lens moves along the range. The peak focus value means the image is sharpest at this lens position. The algorithm will move the lens based on the focus values already recorded. The sharpness values (called focus values) guide the algorithm. The scenes with a lot of details normally have a narrow curve where the peak can be clearly seen. This is not the case in low light situations or bland scene. The focusing algorithm in Contrast Detection Autofocus (CDAF) is responsible for finding the lens position where the image is sharpest. A single focus value doesn't mean anything. The algorithm need a few focus values to try to reach some decision [6]. This results in contrast based focusing being inherently slow. Basic Hill climbing search can be used to find the focusing peak. Figure 2.2 shows basic Hill Climbing Search (HCS). The lens is moved to either far end or near end and then the lens start moving to the opposite side. A single pass can be made to reach the decision. However the lens step length needs to be small to guarantee a better result. Another approach is to use multiple passes over the entire range of lens movement. In the first pass, the steps are bigger. This gives some idea of the curve's peak. In the next pass, finer steps are taken around the supposed peak.

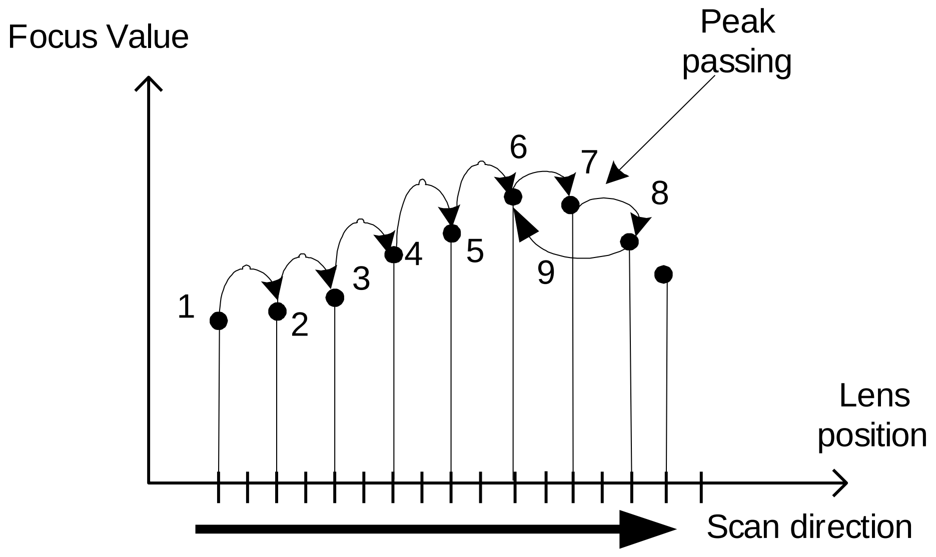

Figure 2.2: *Hill Climbing Search (HCS) [9].*

Another important thing to consider in CDAF is ROI, also called focus windows, that is, the area that is used for focusing. The ROI can be divided into multiple focus windows [6] [10].

Figure 2.3 shows different possibilities.

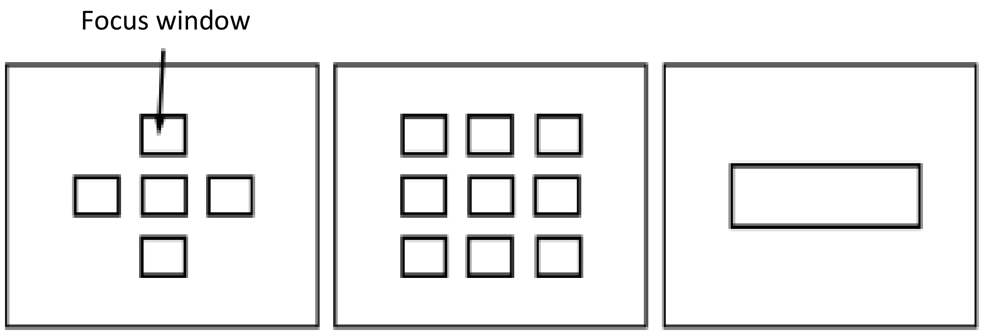

Figure 2.3: *Different examples of focus windows [6].*

Different configurations can be used for focusing. Also different weighting can be given to different windows. Usually the subject is in the center so higher weighting can be given to the window in the center [10]. Multiple windows also help in checking the confidence of the windows and thus help to check the integrity of the data.

Generally, luminance gradients are used to calculate sharpness in an image. Higher values mean better sharpness. Many methods exist for gradient calculation [11]. A very simple example can be given using the Sobel operator. The Sobel operator consists of two kernels. If we have an image A, then the computation is defined as:

$$Gx = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \; and \; Gx = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

where * shows convolution operation. G_x contains horizontal derivative and G_y contains the vertical derivative. The total gradient at any image point can then be calculated as:

$$G = \sqrt{G_x^2 + G_y^2}$$

Figure 2.4 shows the effect of Sobel operator.

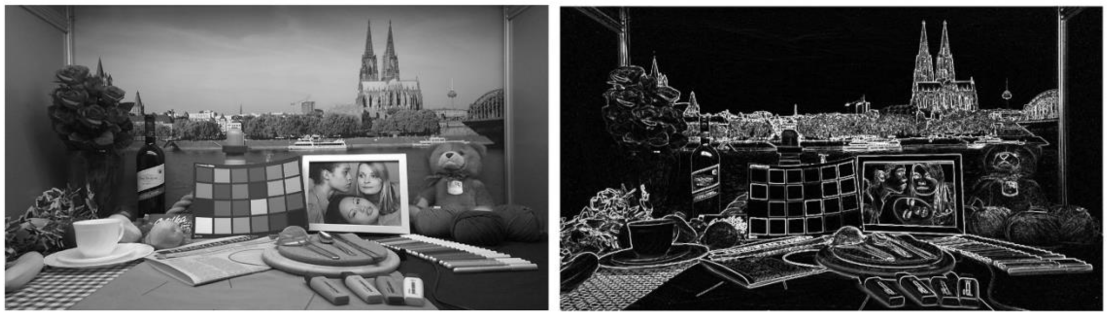

Figure 2.4: *Original image (left), Sobel operated image (right).*

Similar to Sobel, there exist other operators as well for example, Prewitt operator and Roberts cross operator. The pixel-to-pixel gradients in the image are summed up according to the focus windows and all the gradients are given as one value for each window. This value is the sum of the gradients of all pixels in that window. Similar to weighting the windows, the gradients can also be weighted for example they can be center weighted. To some extent, this achieves the same effect as window weighting but it is less complex in implementation.

Focus Kernel

The operator used for calculating image gradients is referred to as Focus Kernel. This is one of the parameters that will be used as an example to show the working of our optimization system. We will configure this kernel to the ISP for calculating focus statistics. These statistics will be used in the focusing algorithm. These days ISPs can take multiple kernels as input and operate these kernels over raw data. However, multiple kernels are found in high end, expensive chipsets and the normal mid-to-low end chipsets still have one ISP that use one kernel. We will use our optimization system to find the optimal kernel for focus statistics. A good focus kernel is able to extract maximum details from the image. The method used to compare different kernels is presented in Chapter 3.

Lens step length

Lens step length is the second parameter that we will optimize in our thesis. Figure 2.5 shows step length for lens movement. If the lens makes a great number of stops during a focusing

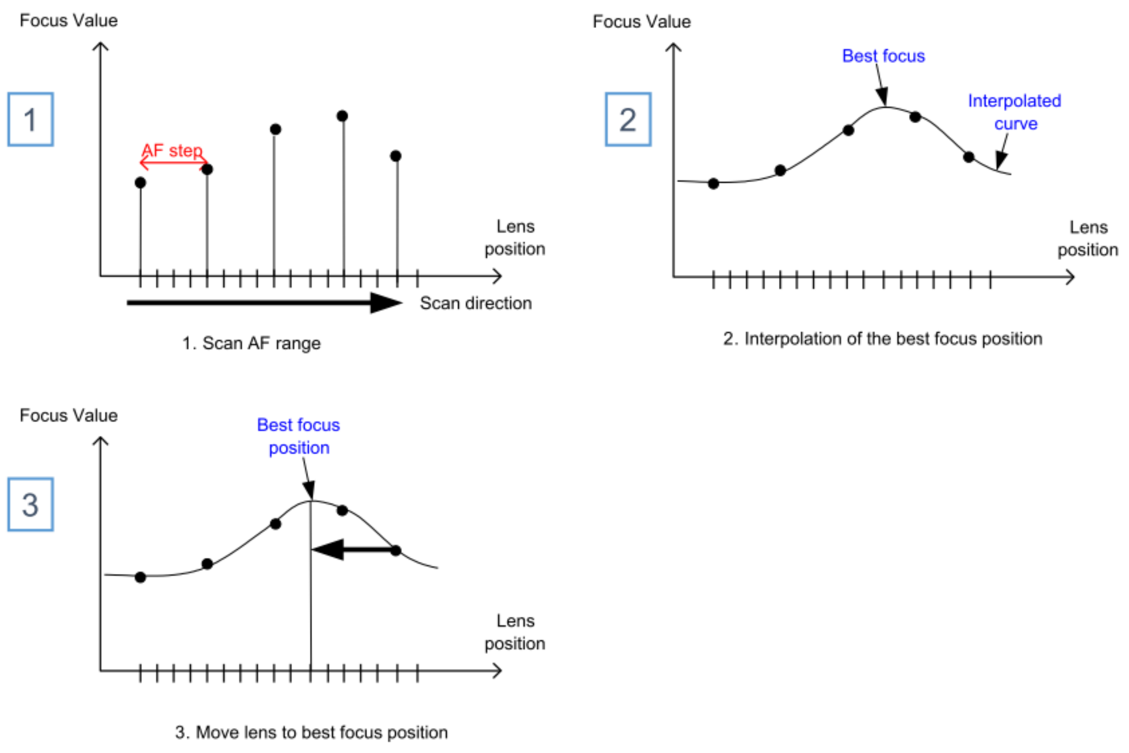

Figure 2.5: *Step-length used for lens movement [9].*

sequence, it will take a lot of time as well as the power consumption will go up. On the contrary, if the lens makes too few stops, we may miss the focusing peak and the results might be wrong, or we may have to make another focusing sequence to find the sharpest point. Greater number of stops also means more data to process resulting in high number of calculations.

2.3 Overview of Evolutionary and Genetic Algorithms

Software is written by humans to solve problems. The way a software works is the reflection of how the programmer thinks the problem should be solved. The programmer first identifies all the tasks the program should solve and then programs the computer to solve the problem in a particular way. Development of software is very precise and deterministic. The computer will do exactly what it is told to do. This approach has been successfully used by most of the software around us. Another property of this deterministic methodology is that it is excellent for getting exact answers. However, not all problems have exact answers and even if they do have exact answers, it may be highly computationally expensive to get the exact answer. Therefore in such situations it is acceptable to find a good enough answer rather than wasting time in looking for a perfect answer [12].

Evolutionary Algorithms (EA) describe a solving strategy where we use computer simulations of evolutionary processes. Evolution assists organisms to adapt to their environments. Evolution algorithms are based on this evolutionary principle. Evolutionary Algorithms have proved to be very useful for solving difficult optimization problems [13]. They perform global random search. EAs are a class of heuristics that are used to successfully solve problems that are very complex. An important benefit of EAs is that they are very flexible. EAs quality metric is their fitness to the objective target and a robust behavior. They are an adaptive concept for problem solving, especially complex optimization problem [14].

A variety of evolutionary algorithms have been proposed. These include Genetic Algorithms, Evolution Strategies, Evolutionary programming and Classifier Systems. All of these are strongly related but independently developed. They all share a common base for performing evolution via selection, mutation, and reproduction [15]. Figure 2.6 shows the families of evolutionary algorithms.

In an evolutionary algorithm, we start by generating a population of possible solutions. This population is then evolved over and over by many generations. Evolution happens according to the rules of selection. The selection procedure is similar to the natural selection, that is, survival of the fittest. The candidate solutions are evaluated against a fitness function. Each candidate solution gets a value for its fitness using which it is compared against others in that environment. Reproduction uses the individuals with the highest fitness. Recombination and mutation help to provide general heuristics for exploration.

The quality of the candidate solutions improves over time. After a certain threshold is reached, we can stop our algorithm and use the evolved solution. Pseudo code of EA is shown in Figure 2.7. The initialization of the candidate solutions is done either randomly or in an educated manner. The fitness (performance) of the initial candidates is evaluated against a fitness function. Then we enter a loop of iterations. Each loop round equates one generation. We increase the time counter per generation. From the initial population of candidates, a sub-population is selected for offspring generation.

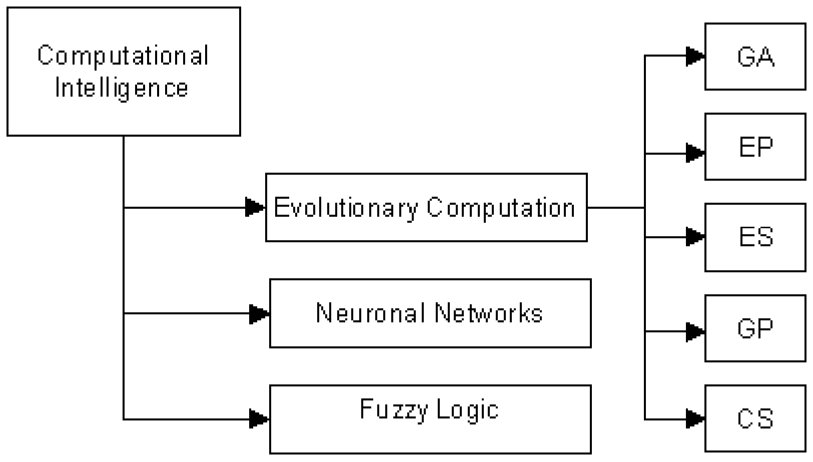

Figure 2.6: *Families of evolutionary algorithms [14].*

This selection can be done on the basis of quality or randomly. Afterwards, the selected candidates are mixed with each other to produce new candidates. The new candidates are then mutated stochastically to introduce subtle changes. The fitness of the current population is measured and selection is done for the next round. Genetic Algorithms (GA) are a sub type of Evolutionary Algorithms. Goldberg and Holland (1988) presented the idea for Genetic Algorithms. A GA has no initial knowledge of the correct solution and depends on the response of the system to reach the optimal solution [16]. Machine learning field can exist without looking at the process of natural evolution. However, the robust mechanism of evolution presented to us by nature contains enormous information which can certainly be used by humans to develop machine learning and other related fields. A look at the machine learning field today reveals that machine learning is indeed inspired in many ways by natural evolution. Neural networks for one is a field which borrows ideas from nature. Similarly Genetic Algorithms are utilizing the principles of nature as their core algorithm [3][17]. Genetic Algorithms are probability based searching mechanism operating on large search spaces. A very important property of GAs is that they are inherently parallel as they use a distributed set of samples to generate new samples. One benefit of GAs is that new requirements can be inserted into an existing program without too much hassle. So our systems builds on what we already have. This helps to improve the system by testing new rules one by one and improves the performance of the whole system with the passage of time. Genetic algorithms are very useful if very little prior information is available about the problem and the best way to measure the quality of solutions is using the problem environment.

```
// start with an initial time
            t := 0;
            // initialize a usually random population of individuals
            initpopulation P (t);

            // evaluate fitness of all initial individuals in population
            evaluate P (t);

            // test for termination criterion (time, fitness, etc.)
            while not done do

               // increase the time counter
               t := t + 1;

               // select sub-population for offspring production
               P' := selectparents P (t);

               // recombine the "genes" of selected parents
               recombine P' (t);

               // perturb the mated population stochastically
               mutate P' (t);

               // evaluate it's new fitness
               evaluate P' (t);

               // select the survivors from actual fitness
               P := survive P,P' (t);
            od
end EA
```

Figure 2.7: *Pseudo Code of an Evolutionary Algorithm [15].*

Genetic algorithms consist of binary search spaces $\{0,1\}^l$. Mutation of candidate solutions is done with a small probability. Unlike other evolutionary algorithms, GAs emphasize on "recombination". The selection of the parents which are to be recombined is done stochastically. The parameters that are controlling the run are kept constant. The size of the population in a single run also remains constant. Bit string is the genotype space i.e. $\{0,1\}^l$ and problem domain representation is normally referred to as phenotype space. Genotype space is mapped to phenotype by encoding/decoding operations. The mutation and recombination happen on genotypes.

Crossover or recombination is an important part of genetic algorithms. Crossover can be done in many ways. In 1-point crossover, we will choose just any point on the two parents. We will cut parents at the crossover point and produce children. In n-point crossover, we choose n crossover points to combine different parents. A third type of crossover is uniform crossover where for each $i \in \{1, \ldots\ldots l\}$ we flip a coin. If we get 'head', we will copy bit i from Parent 1 to Offspring 1, Parent 2 to Offspring 2. However if we get 'tail', then we exchange bit i from Parent 1 to Offspring 2, Parent 2 to Offspring 1 [18]. Figure 2.8 depicts different crossover techniques.

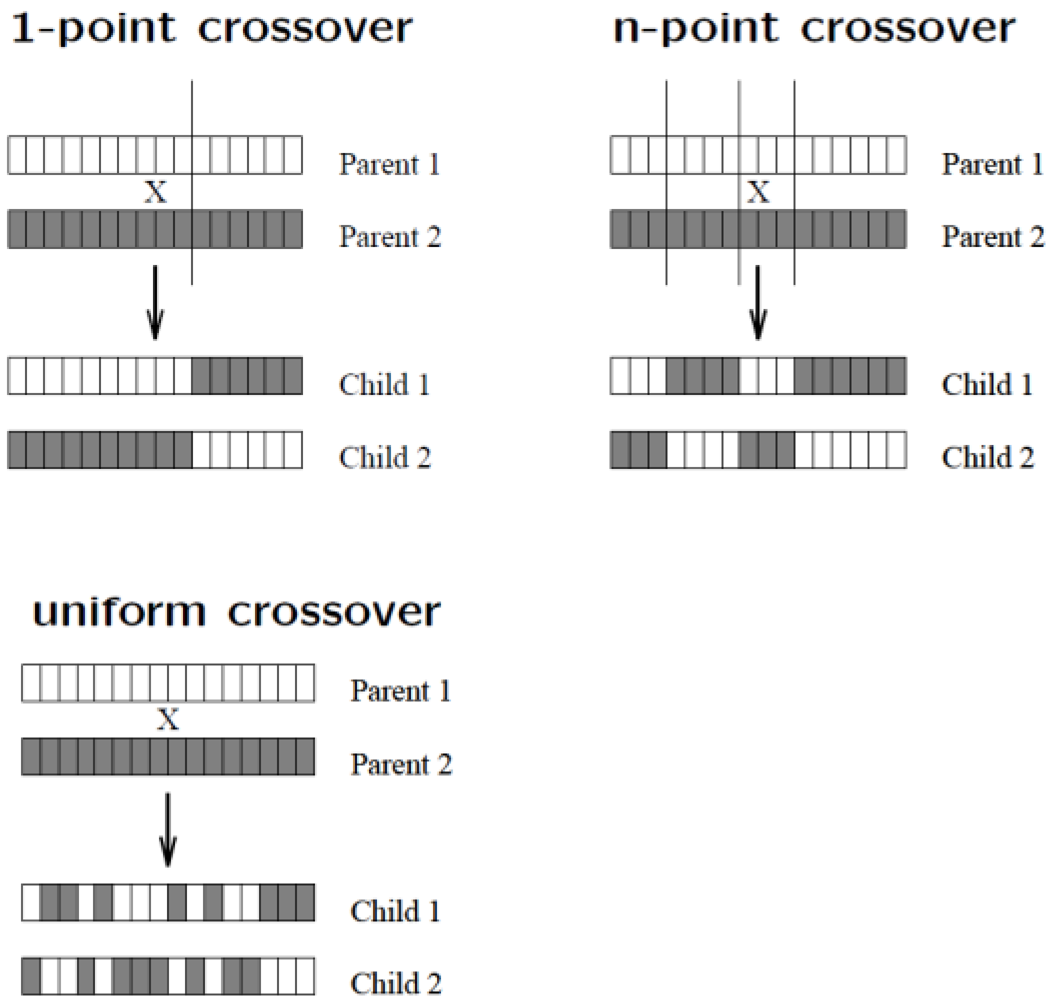

Figure 2.8: *Different crossover possibilities [18].*

Mutation is done by changing parents with some probability, *p*. We can randomly choose which point should be mutated. Also, we can choose how many points should be mutated in a given parent. The important thing is not to mutate too much as to risk changing the original parent too much. Figure 2.9 shows a parent is mutated to get a new candidate solution.

We can relate crossover to exploration and mutation to exploitation [18]. In crossover (*exploration*), we aim to discover promising areas in the search space, i.e., we try to gain information about the problem. In mutation (*exploitation*), we optimize within a promising area, i.e., we use existing information. Crossover makes large jumps to an area between the two parents. Mutation on the other hand, stays near and close by the original parent. Both mutation and crossover seem very simple yet they are so powerful and give excellent results for extremely complex problems.

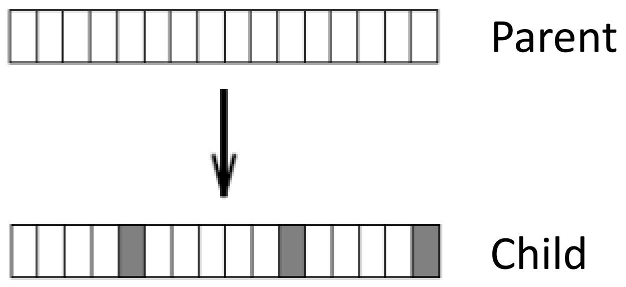

Figure 2.9: An Example of Mutation Operation [18].

2.4 Optimization Methods in Camera Imaging

A prior art search did not disclose the actual idea of providing auto focus parameters with the aid of genetic algorithms. We will present some recent research where genetic algorithms and similar approaches have been used in camera and imaging domain.

Much of the recent research in autofocus area has centered on SAR (Synthetic Aperture Radar) imaging. SAR related autofocus methods and their optimizations can be found in [19][20][21][22] and [23]. Recent research can be found targeting autofocus algorithmic improvements. For example, [24] discusses low-light specific autofocus system model and [25] discusses training based auto-focus for mobile-phone cameras. Autofocus research concerning infrared cameras can be found in [26][27] and [28]. Liquid lens and sharpness measurement based autofocus strategies can be studied in [29]. In [30] authors present a simulation tool for digital autofocus design. They propose to split a dynamic video scene into individual frames and then use the focus profile of each image for evaluating the performance of autofocus search strategy. Paper [31] describes neural network (NN) based AF optimization. Here neural network (NN) determines the main part of the image from a pattern of brightness values. NN receives the input, learns the models of the desired objects and outputs the main parts of scenes. This main part is then sent to the ALU for focus statistics calculations. In [32] neural network is used to predict future lens position which is an important output of the autofocus algorithm.

Particle swarm optimization (PSO) is a famous optimization approach, first proposed by Kennedy and Eberhart in 1995 [33]. The standard PSO uses a velocity based formula to iteratively update particle positions to converge to a solution towards global optimum. Several variations of PSO have been proposed since its introduction. Bare-bones particle swarm optimization (BBPSO) [34] is one variation of the canonical PSO [33][35]. BBPSO optimization has been used in many applications including control, system identification, and image processing. An interesting AF search algorithm using bare bones particle swarm optimization is presented in [36]. In computer vision, camera calibration technique is used to effectively display a 3D object onto 2D image plane. Prior research involving camera and genetic algorithms shows a lot of publications on camera calibration. Papers [37], [38], [39], [40] and [41] are a few GA based camera calibration examples. In [42], GA and PSO are used together for camera calibration.

Paper [43] describes systems, methods, and apparatus for camera tuning. The publication does not discuss autofocus tunings but presents general methods that can be used for camera tuning purposes. The tunings addressed in this publication are post processing tunings to enhance the captured image, for example, to reduce artifacts and to increase edge sharpness. The publication presents the use of GAs in an abstract manner and does not go into details for the implementation. Interested reader can refer to the publication for further reading. Researchers in [44] use genetic algorithms to find out-of-focus blur estimation. Two rough estimates for the solution are used to reach the actual solution. The presented method precisely estimates the blur parameters by using a genetic algorithm which was designed using frequency domain analysis of the image. The genetic algorithm was employed to estimate signal to noise ratio and image degradation parameters.

Research can be found where GAs are used in combination with other optimization techniques. For example, in [45] authors used the genetic algorithm to optimize neural networks. The optical coefficients of biological tissue were determined using a genetic algorithm based back propagation (GA-BP) neural network. The result showed that this neural network was much more precise. Genetic algorithm based back propagation algorithm combines the benefits of genetic algorithms and back propagation, and the result is more computationally efficient and accurate than the original BP neural work. Paper [46] proposes the use of neural network and genetic algorithms to estimate camera white balance illuminant in color images. In cameras, white balance is a tough problem to solve. The proposed method uses a genetic algorithm to reduce the size of the neural network as well as to identify those areas of the scene that are most important for white balance algorithm. The method achieved an average accuracy of 98.8% for scene illuminants and 97.5% for the camera the white balancing illuminants. In [47] authors present an overview of nature inspired optimization techniques for camera calibration. The authors compare simulated annealing, PSO and GA in solving the camera calibration problem in computer vision.

Image stabilization techniques are employed to remove unwanted shaking in video camera systems. An example of genetic algorithm application for digital image stabilization in low signal-to-noise ratio (SNR) is presented in [48]. The author claims that the proposed method allows both stabilization and de-noising. Using genetic algorithms for stereo cameras is an interesting application as well. Papers [49] and [50] present the use of GAs for stereo camera calibration. Paper [50] uses Laplace crossovers and power mutation operators are used to get the next generation of candidates. The proposed method claims robustness under varying amounts of image noise.

Paper [51] investigates the behavior of a simple genetic algorithm for searching bright and edge pixels in an image. Reproduction is done by population substitution based on crossover operation. Two hundred cells evolve for approximately one hundred generations. It was found that the simple genetic algorithm behaved in a stable manner, and solved some of the challenges of conventional convolution filters. The bright areas and edges in any image were identified correctly using this GA based scheme.

Currently, optimization of autofocus parameters is a manual and tedious process. Different OEMs (original equipment manufacturer) produce several phones in a year. Almost all of these products have different camera modules dependent on different price tiers. Consequently, these different modules require separate parameters. Not only the cameras are different but these products also differ in the processor (CPU) which brings its own set of parameters and complicates the optimization process further. Modern cameras are expected to capture good images in different situations automatically, for example, indoor lighting situation, sunny, cloudy, low light at night or in a dark room etc. In most cases, we need separate parameters for these situations for optimal results. Focus Kernel, one example of parameters, is optimized by testing and experimentation. The testing is repeatedly performed throughout the development process. Digital filter design techniques from Signal processing domain are employed and the results are pitted against each other for performance comparison. Many times it can be noticed that a certain kernel is performing adequately well in bright lighting situation but the same kernel does not give acceptable results in low lighting due to the flat focusing curve. Also, a kernel may be very good from sharpness calculations point of view but it may not be good for the autofocus algorithm. This is because the autofocus algorithm needs a focus curve which is easy to follow and can eventually take the lens to the sharpest point. For example, a focus curve resembling a triangle is better than a curve that rises and drops very sharply. When a new kernel is designed, it is deployed to the phone and the phone is taken to the lab where all the basic testing is done. Extensive field testing is also carried out with each parameter. The images from the device are then extracted and are evaluated on the bigger display by the autofocus engineer who decides on the efficacy of this kernel. Another tunable parameter in the autofocus algorithm is the lens step length, that is, how big the lens should jump. This depends on many factors including ambient conditions, camera hardware characteristics, processor power etc. The autofocus engineers have to experiment and test with different values to find the most suitable tuning. It would have been really useful if this tuning was generated by an automated mechanism, and the autofocus engineer would invest his time in algorithmic improvements instead. Autofocus is something that gets the attention at later stages of camera development because, for a new product, the spotlight is first on getting the whole image pipeline working. The continuous integration nightly builds verify the software integrity and identify the software errors, but mostly its actual field testing that can verify autofocus integrity. Thus it is possible that during early development, the AF parameters are not the best one and they slip through until the last moments. This can cause un-necessary pressure on the AF development/testing team and in some cases can even cause products to be delayed. This highlights the critical nature of parameters. An automatic system under these circumstances would definitely be very useful.

The framework presented in this thesis will almost reverse the current process. We will take the test images once, that is, we will capture images from all the test scenarios. We will feed all these images as the input to our system. The system will start with an initial kernel and this kernel will evolve using the principles of selection, crossover and mutation which are presented in Section 2.3. Each kernel is ranked according to its fitness and the best kernels are selected for the next round. After the ending condition is met, the system will output the best kernel found so far. This is a significant improvement over the previous process. The same optimization mechanism can be used for all other focus parameters, for example, focus window size, focus window shape, lens step length, etc.

In this section, we provided some references to the literature where genetic algorithms and similar approaches have been used to optimize camera parameters. As seen above, not much prior research is found on optimization of autofocus parameter design. One reason for that can be the fact that AF is a very industry specific topic. Companies develop their own algorithms and try to keep their tuning mechanisms under wraps, using them for competitive advantage. Although it is a valid idea to publish patents describing these approaches, but patents are not much use if they can’t be detected. There is a risk the competitors can copy the process and reduce the competitive edge, while at the same time not paying any license revenues as it can be very difficult to find if any competitor is using the patented algorithm internally. In the next section, we will describe the implementation details of the proposed optimization system.

3. AUTOFOCUS OPTIMIZATION SYSTEM

This chapter presents algorithm design and implementation details of the proposed optimization system and shows how ideas derived from genetic algorithms are used in this study. The main building blocks are explained, that is, how the fitness of a solution candidate is evaluated, how the input data is gathered and how we reach the final, optimized solution. Experimental results and discussion is presented towards the end of this chapter.

3.1 Algorithm Design

Figure 3.1 shows the high level structure of our design. Each parent produces children whose quantity depends on the input parameters. Some children are produced by mutation and some by crossover. Afterwards, the best solutions are chosen and they act as parents for next iteration (generation). This continues until we satisfy the ending condition.

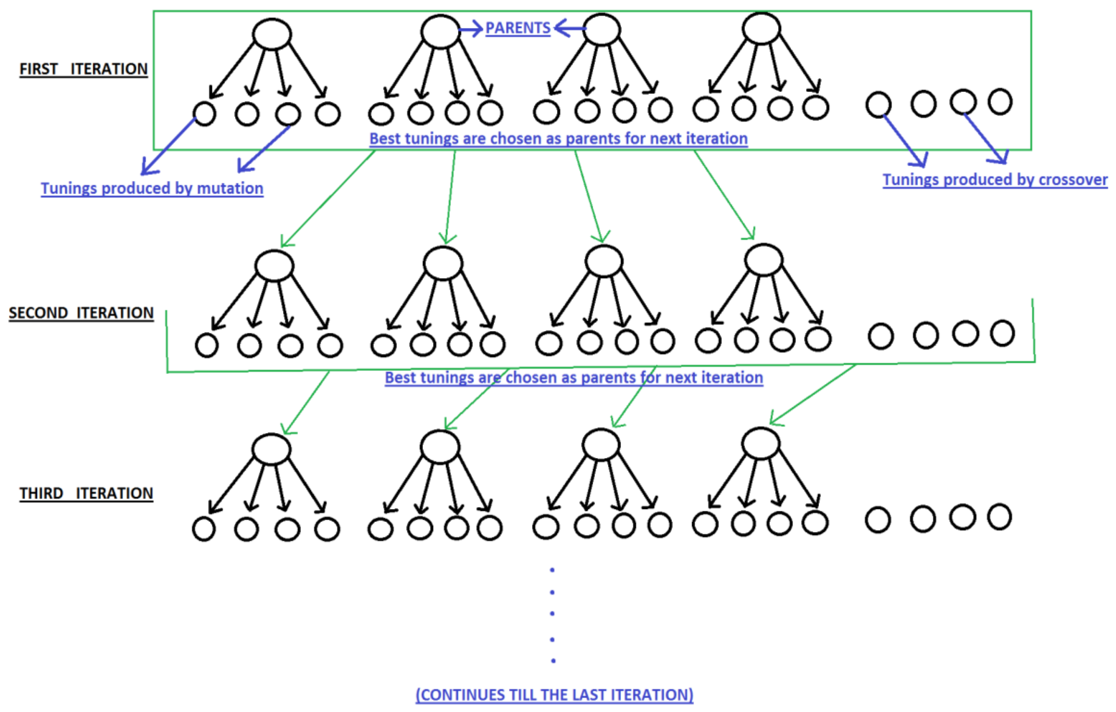
PARENTS
FIRST ITERATION
Best tunings are chosen as parents for next iteration
Tunings produced by mutation
Tunings produced by crossover
SECOND ITERATION
Best tunings are chosen as parents for next iteration
THIRD ITERATION
(CONTINUES TILL THE LAST ITERATION)

Figure 3.1: *Overview of the Genetic Algorithm based design*

In genetic algorithm, mutation is very important in the architecture of the algorithm. The main target for the mutation is to give the freedom to evolve from scratch and reach the optimized result. Mutation should not limit the evolution in any particular way. It should let the population of candidates evolve without applying much constraints. In our design, mutation is done by randomly adding or subtracting small-magnitude values to the parent candidates. The change is kept small and controlled so that the overall structure of the parent is not completely altered and the skeletal features from the parent are still seen in the child. The mutation function follows the biological ideas of mutation and evolves slowly. However, at the same time, we have to ensure mutation is producing acceptable results and that we converge to a solution following this path. We also have to mitigate the possibility that the mutation might get trapped in local maxima. In our system, the parent kernel is mutated in the following way:

```
for counter = 1:3

    i = randi(10,1);
    Current_Kernel(i) = Current_Kernel(i) + 1;
    k = randi(10,1);
    Current_Kernel(k) = Current_Kernel(k) - 1;

end
```

The input kernel in our case consists of 2 rows and 5 columns. Both the rows of the input kernel are concatenated to form a vector of 1 x 10 dimension. We randomly choose an index ‘i’ from this vector and add ‘+1’ to the value of this index. Similarly we randomly select another index ‘j’ and add ‘-1’ to its value. We repeat this process three times. The reason for adding/subtracting ‘1’ is that we want to keep the original shape of the kernel intact and try to perform the evolution at a slow pace. This helps to keep the mutation more natural. If we add big abrupt changes to the mutation, we may have difficulty reaching a stable state. Bigger changes can get us out of the optimal solution and the best solution may get discarded/un-noticed if we change its values too quickly. Therefore we mutate the kernel by adding or subtracting ‘1’ and thus the kernel is not changed drastically after every mutation. This mutated kernel is output to the main algorithm which applies it over the input image-series whereby fitness of the kernel is evaluated. The second parameter that our system is optimizing is lens step-length. When the algorithm starts, step-length of every candidate is initialized to one. We mutate the step-length randomly but in a controlled manner. Range for the step-length mutation is chosen as:

range_for_StepLength = ceil(Image_series_Length / 10);

Mutated lens step-length is computed as:

StepLength = StepLength + randi([-range_for_StepLength range_for_StepLength],1,1);

The addition (or subtraction) factor is named ‘range_for_StepLength’. We limit this value depending upon the number of images in the series. A random value is chosen between -r*ange_for_StepLength* and +*range_for_StepLength.* If our input image-series consists of 34 images, then from above calculations, the mutated step can be +/- 4 from the original value. If we have an input image series of length 67, then our mutated step can be +/- 7 from the original value. Therefore this range of mutation varies dynamically depending on the number of images in the input-series. If the number of images in a series is higher, step-length has a wider range to mutate. This helps the algorithm to adjust accordingly.

Below is the pseudo code for the mutate() function:

```
% First change the input kernel (2 x 5) into a single array (1 x 10) i.e. concatenate both %
rows into a  single row.
%
% // LOOP STARTS (loop 3 times)
% Pick a random location from this (1 x 10) array:
% Add +1 to the value of this location
% Again pick a random location from this array:
% Add -1 to the value of this location%
% // LOOP ENDS
%
% Do the un-concatenation and output the kernel as 2 x 5
% Steplength mutation:
% pick a random number between [-range_for_StepLength, range_for_StepLength]
% add this number to input steplength
% check that resulting StepLength is not negative/zero
% if resulting StepLength is negative/zero, pick up the random number again
% and repeat above process until we get positive value for resulting StepLength.
```

The above pseudo code calculates the mutated values for Focus Kernel and Lens Step-length for one candidate. The above code will be run for all candidates to produce mutations for all the candidate solutions. These candidate solutions will then be applied over input image-series and the ones with best performance will be selected for the next iteration of the algorithm. In crossover (also called recombination), we mix two or more parents to produce new candidates. The new solution will get the best features of both parents and will give us results which carry properties of both parents. The whole evolutionary algorithm approach is based on experimentation and this crossover technique gives us a handy tool to explore more diverse solutions. Below is the pseudo code for our crossover approach:

```
% randomly select two parent candidates
% Add their corresponding kernels to eachother and then divde the result by 2
% Add their corresponding steplengths to eachother and then divide by 2
% Get rid of fractions by using ceiling operators
```

This simple crossover method gives us unique and highly effective candidates to be used in our genetic algorithm based system.

3.1.1 Fitness Evaluation Criteria

The fitness (or performance) of a candidate solution is determined using magnitude of the contrast, time-to-focus and shape of the plotted focus values. Focus values (also called sharpness values) are obtained by high pass filtering the input image-series. To get focus values for an image, we extract a sub image from the input image. This sub image will be our region of interest.

Sub-image width = Total_image_width / 3;
Sub-image height = Total_image_height / 3;

Based on above values of width and height, we extract the sub-image from the center of original image. These co-ordinates stay the same for all the images in all the series to make our results comparable.

The focus values are calculated as follows:

[Focus_values] = highpass_filtering(inputImages, candidate.kernel, candidate.StepLength);

The above function *highpass_filtering()* takes the focus kernel and step-length as input parameters and applies these parameters over the input series. The output of this function consists of sharpness value or as we call it, the ‘focus value’ for each image. The output *Focus_values* is a vector of focus values where each focus value corresponds to the sharpness of one image in that image-series. This sub-image area can be increased or decreased, giving us corresponding change in running time. Also we can change the coordinates to get focus values for different parts of the image. After getting focus values, we can calculate the performance of the candidate.

After getting the focus values for an image-series, we first check three conditions regarding the focus values. If any of these conditions fail, the performance of the candidate is said to be zero. These conditions are:

1. If the maximum value for Focus-values occurs in the first image of the input series, i.e.

   maxValue_index = = Focus_values (1)

   then performance is said to be zero (no peak is found). Figure 3.2 shows an example of this scenario.

2. If the maximum value for focus values occurs in the last image, i.e.

   maxValue_index == size (Focus_values, 2)

   then performance is said to be zero (no peak is found). Figure 3.3 shows an example of this scenario.

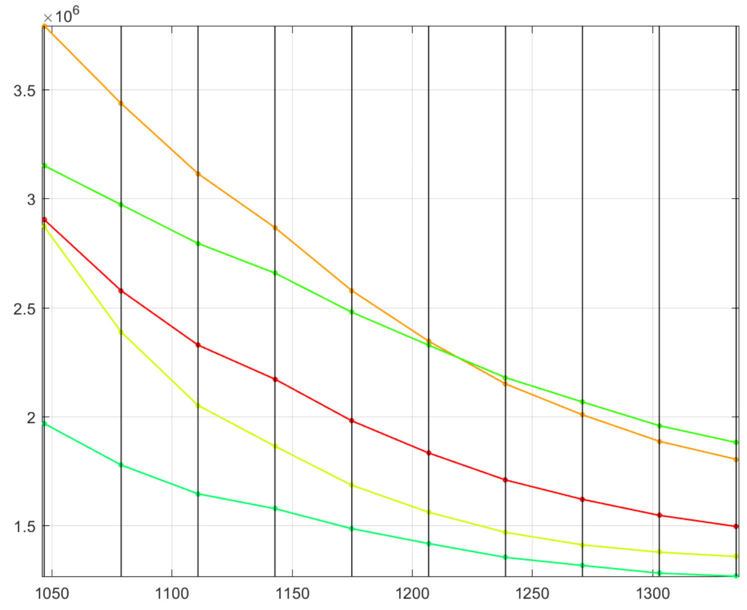

Figure 3.2: *Example of focus curves where maximum Focus-value occurs in the first image of the input series and therefore no peak is found.*

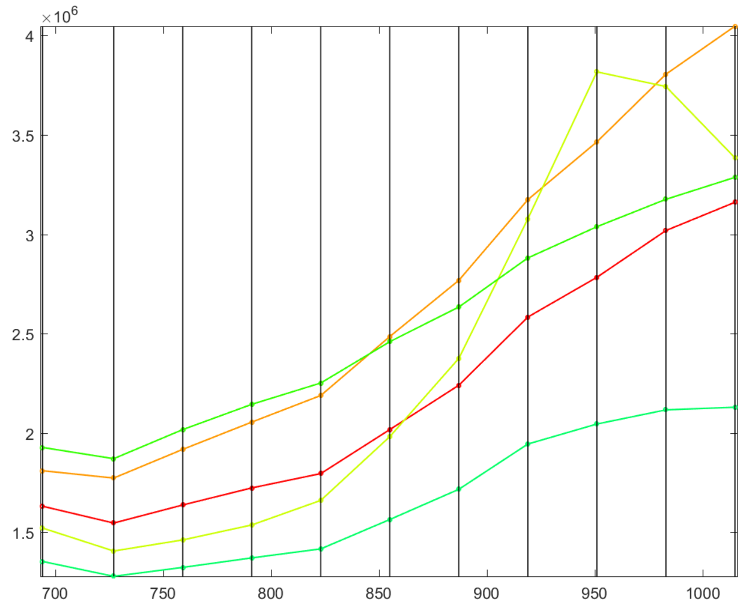

Figure 3.3: *Example of focus curves where maximum focus value occurs in the last image of the input series and therefore no peak is found.*

3. Derivative check

Here we check the integrity of input data. A good focus curve should have values rising and then falling. However, if there is movement is the scene or if the input data is not of good quality (e.g. noise) then we may have an oscillating curve and we want to discard that result. We do that by checking the derivative of the curve. If a certain curve has ‘derivative = 0’ at more than one instances, this means the curve is oscillating and is not a good curve. So any candidate solution having more than 1 ‘derivative = 0’ would be discarded. Figure 3.4 shows an example of this scenario.

The overall fitness performance of a solution candidate is the sum of following three

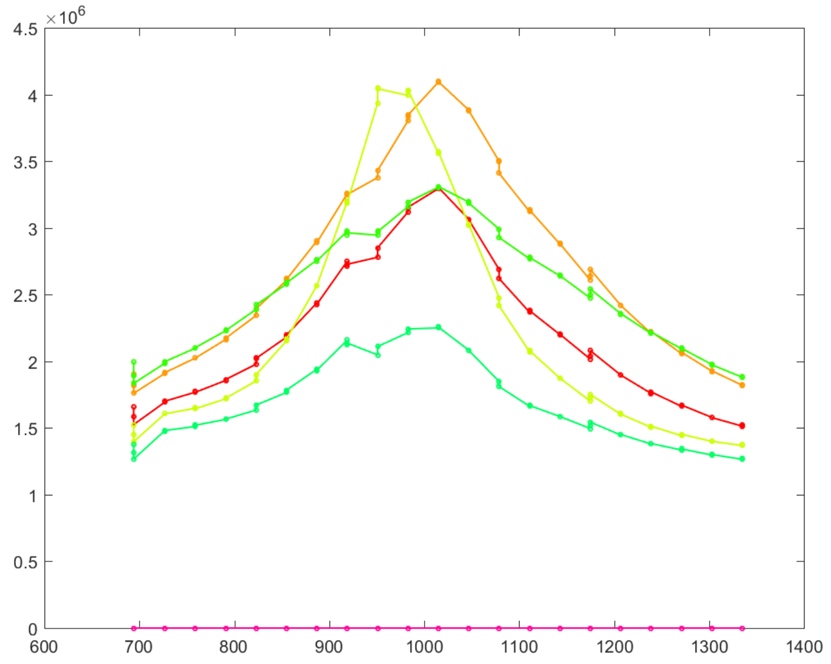

Figure 3.4: *The green curve at the bottom is an example of a bad focus curve.*

criteria:

a) Contrast contribution
b) Step-length contribution
c) Shape contribution

The next section will describe the details regarding these criteria and how do we calculate them.

a) Contrast contribution

The higher the contrast produced by a solution candidate, the higher is the performance, i.e.

$$\text{Contrast} \propto \text{performance of candidate}$$

The contrast calculations can be more easily understood if we consider the focus curves. Focus curves are plotted by using the focus values against each image in the image-series. While calculating performance, we consider the right-side contrast of the curve and the left-side contrast of the curve separately and join these two (in a weighted form) to get the final contrast value. Using focus curve, we will calculate the left contrast as:

Contrast_left = max(Focus_values)/ min(Focus_values(1:maxValue_index));

where Focus_values is the array containing the focus values for all the images in an image-series. Afterwards we will calculate the right contrast as:

Contrast_right = max(Focus_values)/ min(Focus_values(maxValue_index:last_index_of_array));

We will combine the results for both these contrasts to get contrast contribution for this candidate. We use the following formula for this:

*Contrast_contribution = ((0.5 * Contrast_left + 0.5 * Contrast_right) / 2);*

Also, in above equation, we give equal weights to left and right contrast. We experimented with different weights and decided to use equal weights for them. The division factor of 2 was added based on experimentation. Without this division factor, contrast results would dominate the other results.

Figure 3.5 , Figure 3.6 and Figure 3.7 show three example focus curves producing contrast values of 2.1783, 2.8095 and 2.2800 respectively. These contrast values are obtained using the formula presented above. We choose candidates that produce higher contrast.

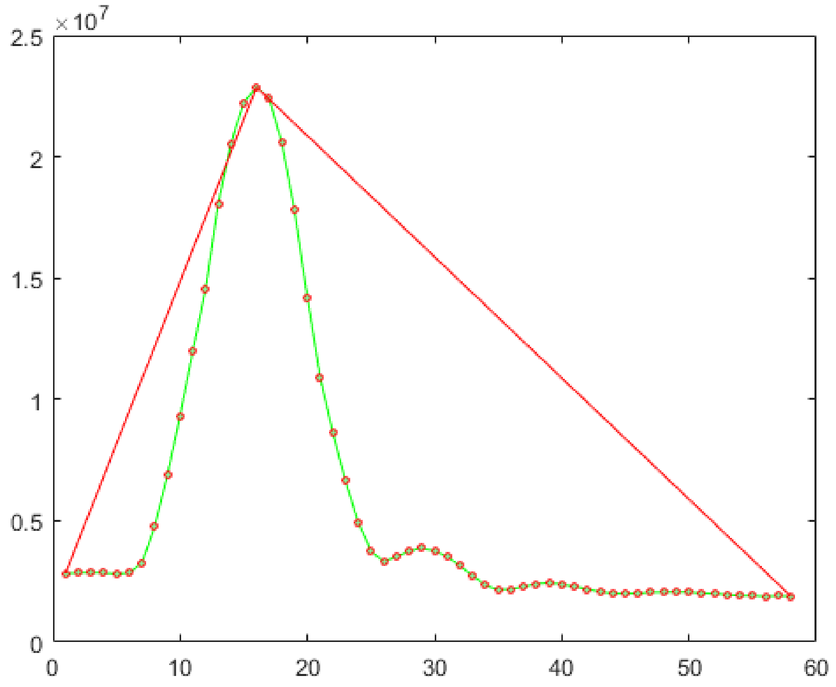

Figure 3.5: *Focus curve having contrast value of 2.1783*

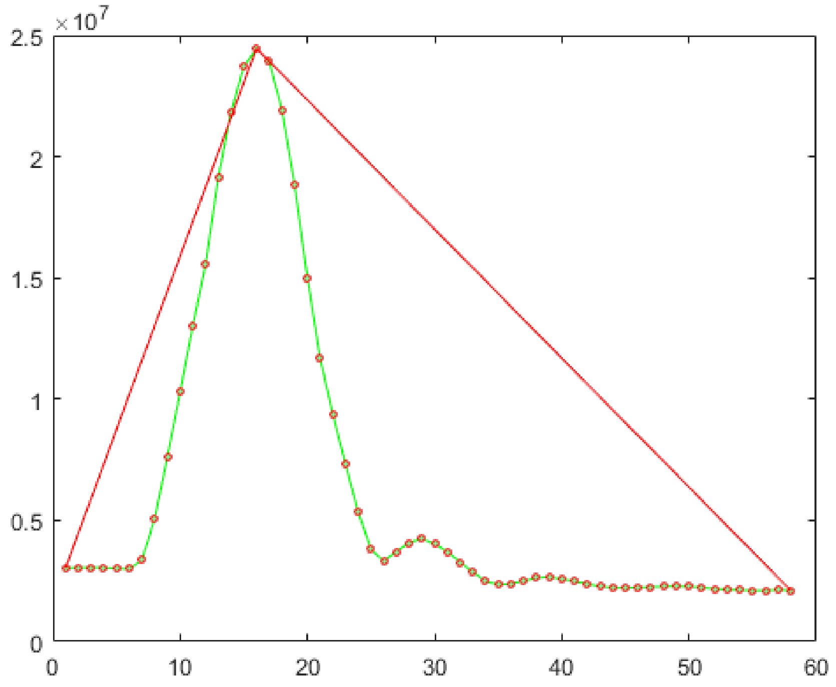

Figure 3.6: *Focus curve having contrast value of 2.8095*

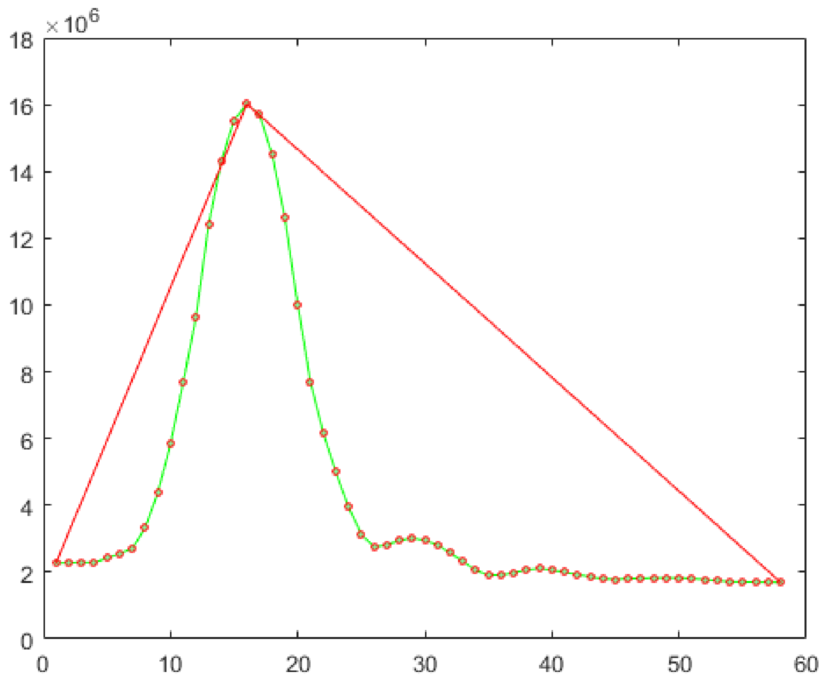

Figure 3.7: *Focus curve having contrast value of 2.2800*

b) Step-length contribution

The greater the step-length, the higher is the performance of the candidate, i.e.

Step-length ∝ performance of candidate

So performance for step-length = 1 will be much less than performance for step-length = 5. This is because we want to get to optimal focusing in as little time as possible. If the step-length is smaller, it will take more time to reach the focus decision. When step-length = 1, lens goes to every step (lens position) and calculates the focus values. Going to every lens position takes a lot of time from mechanical point of view and then calculating the focus values at these points can be even more time consuming from data processing point of view. Therefore more weighting has been given to step-lengths with higher values. However, it is important to note that if the step length gets too high, then there is a high possibility to miss the focus-peak and thus we may end up with lower focus-peaks resulting in less-focused images. To avoid such a situation, we have introduced three pre-conditions described earlier in this chapter. The formula used for calculating step-length contribution is as follows:

Step_Length_contribution = (candidate.StepLength / Image_series_Length);

c) Shape Contribution

We give higher weightage to the focus curve that fits a triangle better. We plot a focus curve based on the input image-series data and draw a triangle over it. We calculate the difference between the corresponding points on the focus curve and the triangle. The less the difference, the greater is the performance of the candidate. Figure 3.8 shows a focus value curve. In Figure 3.9, we plot a triangle over this curve. The difference in the focus curve and the triangle is used to calculate the shape contribution.

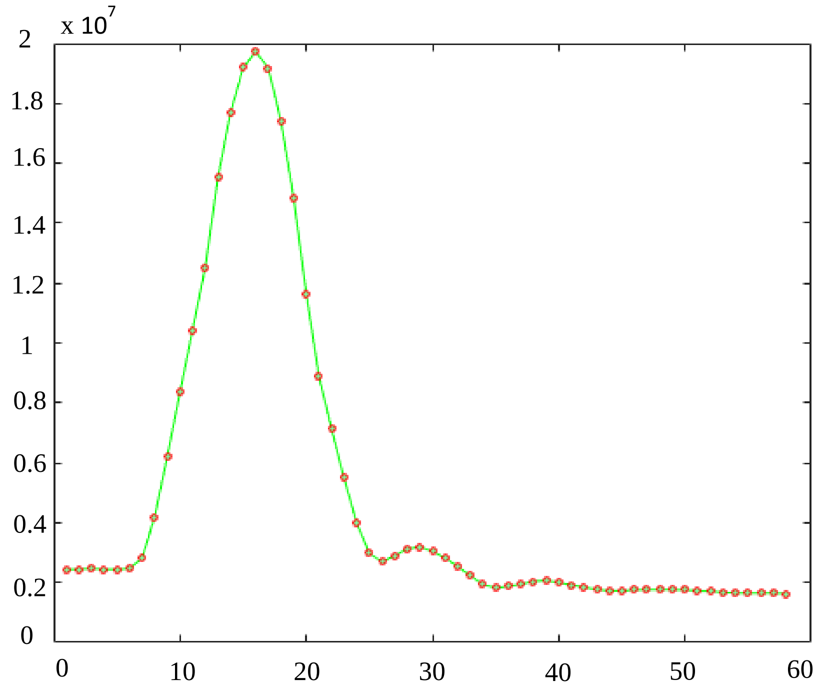

Figure 3.8: *A focus curve to be used for shape calculation*

Now we draw the triangle using the starting point, the peak point and then the last point. This is shown in Figure 3.9. The formula used for calculating shape contribution is as follows:

Total_diff = triangle_diff_calc(Focus_values);
normalized_diff = Total_diff / (max(Focus_values) - min(Focus_values));
Shape_contribution = (1/normalized_diff);

Total_diff is the difference between corresponding points in the focus curve and the triangle. This result is normalized using the maximum and minimum focus values so that our results are comparable across multiple inputs. Figure 3.10 and Figure 3.11 show focus curves, with shape contribution values of 0.1709 and 0.1760 respectively. Focus curve in Figure 3.10 is narrower around the peak and quite flat in the blurred regions when compared to Figure 3.11. A narrow and flat curve is not good from AF searching algorithm point of view and therefore curve in Figure 3.10 gets a lower fitness value than one in Figure 3.11.

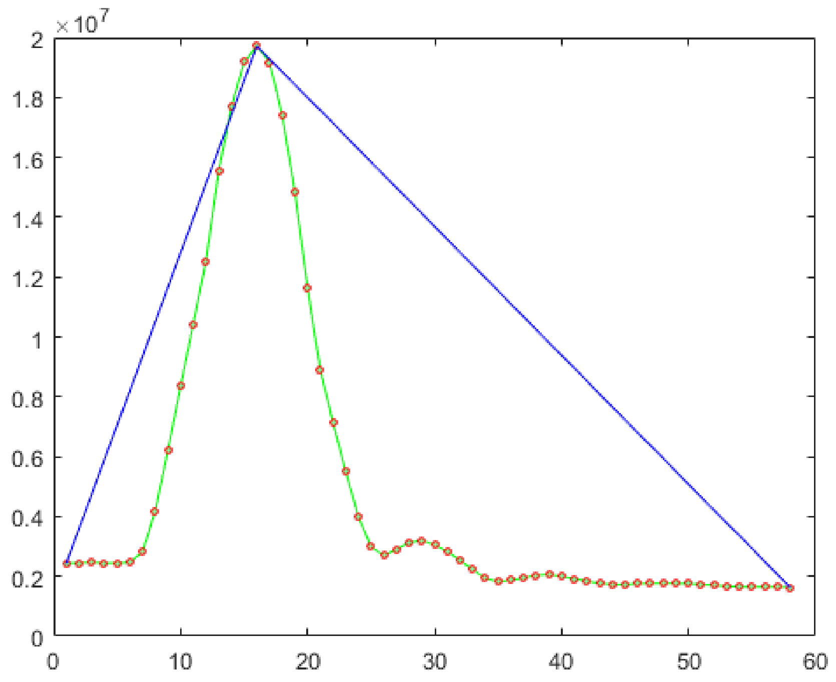

Figure 3.9: *Adding triangle over the focus curve in Figure 3.8 for calculating shape contribution*

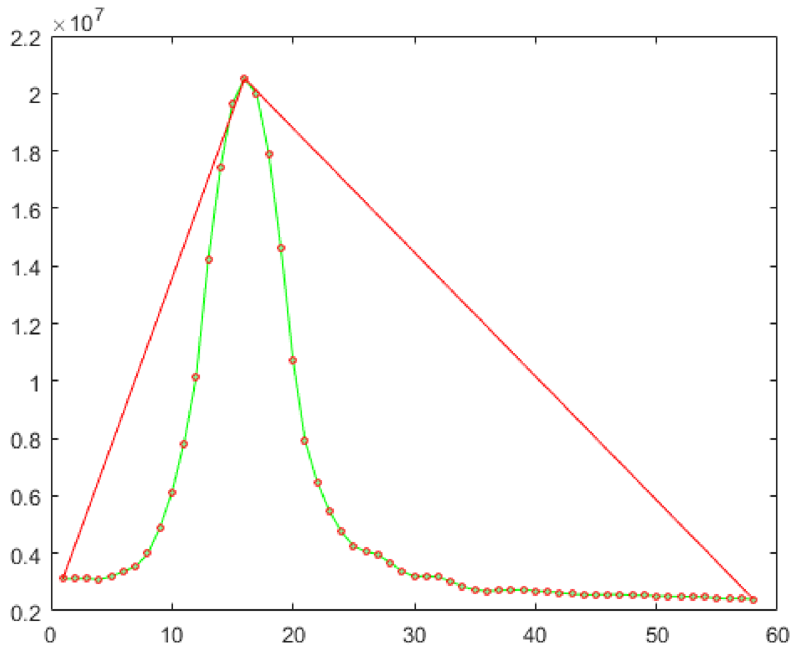

Figure 3.10: *A relatively narrow focus curve having shape contribution value of 0.1709*

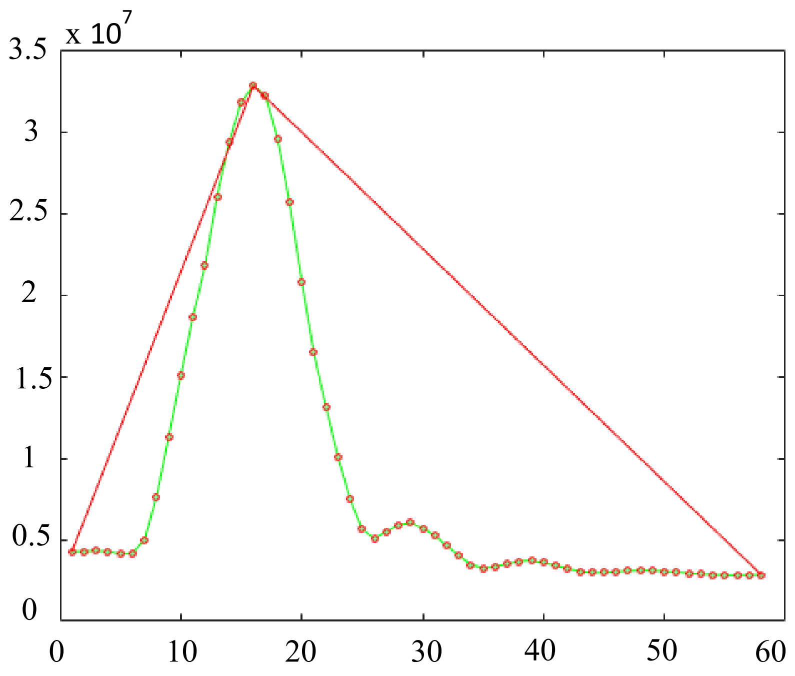

Figure 3.11: *Relatively wider focus curve having shape contribution value of 0.1760*

After calculating all three individual performances, the total performance of a solution candidate is sum of all three performances:

$$\textit{Total performance} = W_c * \textit{contrast contribution} + W_l * \textit{step length contribution} + W_s * \textit{shape contribution}$$

where W_c, W_l and W_s are the respective weights. In our experiments, we use $W_c = 0.5$, $W_l = 0.5$ and $W_s = 3$. These weigthing factors are obtained by extensive experimentation and are used so that one criterion does not dominate the other values and vice-versa. This 'Total performance' is a candidate's performance for one input series. After performance evaluations for this series, we will repeat the same process to calculate performance for all the other series in the input.

We have set a minimum threshold for the performance of a candidate. If a candidate's performance is lower than this threshold, we declare that candidate to have failed for the series it was operating upon. If a candidate fails for an input series which is flagged as critical, that candidate is immediately discarded and we start calculations for the next available candidate. To calculate minimum performance threshold, we ran our system for 4000 iterations with 100 candidates in each iteration. A bright-light series was used as input. The resulting kernel and step-size settings were then applied to a low-light series. The performance of these settings over the low-light scene came out to be 0.3507. We took 15% of this low-light performance ('0.3507') and set this as the minimum threshold for performance. In numerical terms, this value came out to be:

$$0.15 * 0.3507 = 0.0526$$

So any performance value below ‘0.0526’ will be assumed to be ‘zero’ and the candidate producing such low values will be discarded.

An important question in the working of this system is to decide how long it should be run, that is, how many iterations? The decision to stop iterating can be taken either by selecting the best kernel after a fixed number of iterations, for example, 4000, 6000 iterations etc. Another option is to make the stopping condition a function of rate-of-change of results, that is, if the rate-of-change of performance is flat or nearly flat for a certain number of iterations, the system can stop iterating further.

3.1.2 A-priori expectation based weighting

If we have more than one image-series as input, then it is possible that the input contains some low-light image-series and some bright-light image-series. The overall performance of a candidate is calculated by summing up performance of that candidate for all the input series. The contrast obtained from low-light image-series will be much lower than the contrast from the bright-light conditions, even when the candidate applied on both of them is exactly the same. Therefore bright-light results will dominate the final score. Based on this observation, enhancement has been made to the scoring of candidates. In an effort to neutralize this dominating effect of bright light image-series, we will trim the performance values of bright-light and other high values, and at the same time amplify the low-light results. We call it scaling. Scaling is done because even a low contrast from a dark scene is acceptable (and good) as that is what we expect from a dark scene. Therefore if a dark scene gives us reasonable amount of contrast, we will amplify the performance values of dark scene. Similarly, in the case of a bright scene, if we are getting very high contrast values, we will trim the high values of this bright scene and scale it down to make it comparable to the other series. In this way, we will mitigate the dominating effect of bright light image-series.

The formula used for scaling is as follows:

$$avg = \frac{sum\ (performances)}{length\ (performances)}$$

$$y = performances^{0.75}$$

$$out = 1 + \left(avg - \frac{avg}{y}\right) * 1.15$$

where *performances* is an array containing performance values of all input image-series. The above formula helps us in scaling i.e. it amplifies the smaller values and trims the higher values. This effect can be easily seen in the example provided below. We assume that we have different image-series as input with different contrast values, i.e.

in = [0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00] ;

Figure 3.12 shows the effect of scaling.

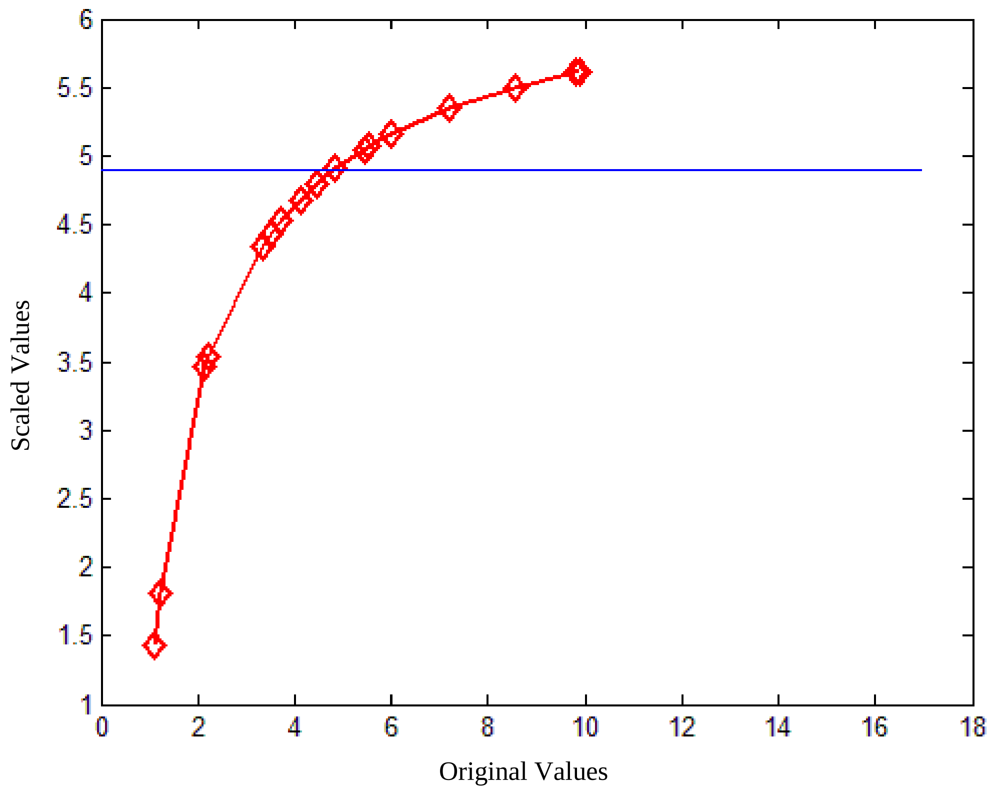

Figure 3.12: *Effect of scaling.*

Table 3.1 depicts the effect of scaling over sample input. From this table it is clear that lower values have been amplified and higher values trimmed so that performance results from different series remain comparable and bright-light contrast may not dominate the low light contrast.

Table 3.1: *Effect of scaling on numerical values*

Performance values before scaling	**Performance values after scaling**
(in)	**(out)**
0.10	0.10
0.20	0.36
0.30	0.46
0.40	0.51
0.50	0.54
0.60	0.57
0.70	0.59
0.80	0.60
0.90	0.61
1.00	0.62

3.2 Implementation

This system is implemented using Mathworks MATLAB. All the scripts for this framework are written in Matlab code. To gather input for our system, a Windows Phone application is developed. The target of this optimization system is that we will provide input data to the system and the system will output the required autofocus parameters by analyzing this input. We have structured our system in a flexible way and any new parameters that need optimization can be easily added. The data structure consists of parameters that are to be optimized. Each possible tuning structure is called a 'candidate solution' or simply a 'candidate' in this thesis. As discussed in Chapter 2, there are several tunable parameters in an autofocus system. As a proof of concept, we will optimize two parameters in our implementation, that is, *Focus Kernel* and *Lens Step-length*.

In any machine learning algorithm, the size of input is an important element which affects the results. The bigger the input, the more comprehensive the results will be and higher will be the quality of the results. The input to our system consists of sequences of images and each sequence is referred to as an *image-series* in this thesis. Each image-series contains images of the same scene under the same conditions, but with different lens positions, for the whole range of lens movement. This mimics the focus sweep performed by the autofocus algorithm while trying to find the sharpest lens position. We will use multiple image-series, under varying conditions, for example, a low-light scene, a bright-light scene, macro scene, infinity scene, snowy scene etc. If we want to capture an image series containing 34 images and the whole range of lens movement is from 0 till 1000 DAC values, then images are captured at the following lens positions:

LensPositions[34] = {0, 30, 60, 90,…, 900, 930, 960, 1000};

The greater the number of input series, the higher will be the running time of the algorithm. Similarly, the number of images per image-series is also directly proportional to the running time. Reliability of our results is directly proportional to the size of input. As mentioned above, we want to capture a number of image-series which will then act as an input to our algorithm. The feature to capture an image-series at different lens positions is not available in the publicly available camera application. Therefore, to generate input, we developed an application for Windows Phone. This application was developed using Windows Phone 8 SDK. Microsoft's Visual Studio was used for the development of this application. The application was designed such that we enter the desired lens positions and the application captures images at those positions by moving the lens to all the required positions. The images captured by our phone application are in JPEG format. After getting the image-series, we extract the part of the image which will be used for focus calculations (called Region of Interest, ROI). In most of the cases, ROI is in the center of the image. However, it can be anywhere depending upon the user requirement. The ROI will then be operated upon for focus calculations and we will get our image sharpness values based on these calculations.

When we apply our focus calculations over JPEG images, they give us a good estimate for the image sharpness. However, JPEGs don't carry the same information as the original raw images. Phone camera JPEGs have been operated upon by heavy de-noising algorithms. Plus a lot of image information has been lost as JPEG is lossy image format. The purpose of our system is to calculate focus parameters which will be used in an actual device. Therefore it is important that the input data received by our system should be the same as it is being received in the device so that the contrast calculations are done in exactly the same manner in our system and the actual device. In a device, contrast calculations (used for autofocus) are done by an Image Signal Processor (ISP) based on 2x2 binned raw data received from the sensor. In order to simulate the same processing behavior, we feed 2x2 binned raw images to our tool.

Our algorithm will operate upon many different input images (captured in different ambient conditions). Focusing works much better in bright conditions and probability of failure is much higher in dark conditions. The fitness of a solution candidate is calculated by combining the fitness performance of that candidate over all the input image-series. If one of the series is failing and others are passing, the overall combined result may still have a high value. By simply looking at the final cumulative fitness value, we might not notice that our candidate solution performed poorly on one of the series. To handle this situation, we have introduced two input data paths:

a) Important Input path
b) Normal Input path

Those input series that have a high likelihood of poor contrast (e.g. low-light), we will place them in path (a). Other series are placed in the folder with path (b). If a candidate solution performs poorly on input series placed in "Important Input Path", that candidate is discarded immediately. This way we ensure that the final solution candidate performs reasonably well on the input series placed in path (a).

A number of running parameters that govern the working of the system can be set by the user to customize the system. *Number of iterations* is very important in any genetic algorithm. It controls the number of times our algorithm is run. The greater the number of iterations, the more reliable our results will be. *Number of candidates* in a particular iteration play an equally important role in the quality of final results. Number of candidates in one iteration is determined by the three parameters: Number of parents, number of mutations and number of crossovers. *Number of parents* determines the amount of candidates selected after each iteration. When an iteration completes, *N* best candidates are selected to act as parents in the next iteration. These parents will be used to produce further candidates via crossover and mutation operations. *Number of mutations* determines how many candidates will be generated by mutation from each parent. The parents are mixed with each other to produce more candidate solutions called crossovers. *Number of crossovers* defines how many crossover-based candidates will be produced in one iteration. Thus, total number of candidates in each iteration can be calculated as

*Number of candidates = number of parents + number of parents * number of mutations + number of crossovers*

For example, if number of parents = 3, number of mutations = 4, and number of crossovers = 5, then number of candidates in one iteration will be 20 (3 + 3 x 4 + 5). Number of candidates is directly proportional to the reliability of results. If we have a large number of candidates in an iteration, then the room for experimentation is much higher. The algorithm can experiment with a variety of different solutions. The best candidate chosen from a large pool of different candidates is more likely to be of good quality than a candidate which is chosen among a smaller pool of candidates. *Number of iterations* can be kept anywhere between 50-400 or even more.

3.3 Experimental Results and Discussion

We run our optimization system for different inputs and report the results and findings. The input consists of multiple image-series. We use our Windows Phone Application designed specifically to generate our required input (see Section 3.1.1). This phone application can be used for capturing JPEG as well as raw images. We enter the length of image-series and the application captures the required image sequence. In the following experiments, we are optimizing Focus Kernel and Lens Step-length. We initialize lens step-length to ‘1’ and Focus Kernel to a basic kernel, i.e.

Kernel_0 = [1, 0, 0, 0, 0;   0, 0, 0, 0, -1];

We take raw images under different illuminations and at different distances to the subject. Table 3.2 shows 20 image-series we took to be used as possible inputs. For each distance, we captured an image-series for 5 different illuminations.

Table 3.2: *Input data*

Distance (meters)	**Illumination (lux)**				
0.41	4	12	218	1500	2450
1.00	4	12	218	1500	2450
2.15	4	12	218	1500	2450
3.61	4	12	218	1500	2450

Each image-series contains 60 raw images. The whole range of lens movement is from 0 to 1000. We divide this range into 60 points and take these images. The first image is taken at lens position: 0 (beyond infinity) and the 60th image is taken at lens-position: 1000 (macro)

From the above 20 possible inputs, we choose the following three scenarios to demonstrate the working of our system:

A. Bright-Light image-series
B. Low-Light image-series
C. Multiple image-series

We run our system for different number of iterations, with variable number of candidates explored in each iteration. Below are the tabular results:

A. Bright Light image-series

We use a bright light image-series with following specifications:

Luxlevel: 2450 lux; Distance to subject: 1.00 meter; Number of images in the series: 60

Table 3.3: *Optimization results for Bright Light image-series*

# of iterations	# of candidates per iteration	Number of solutions explored	Performance value	Time taken (hh:mm:ss)	Initial Kernel	Resulting kernel	Resulting Step-length (%)	Result location (Iteration; Candidate #)
1	5	5	3.0775	00:00:28	1 0 0 0 0 0 0 0 0 -1	0 0 0 0 0 0 1 1 1 -3	5	1;3
1	15	15	2.7366	00:01:39	1 0 0 0 0 0 0 0 0 -1	2 1 -2 0 -1 0 0 1 0 -1	2	1;2
3	5	15	2.6302	00:00:56	1 0 0 0 0 0 0 0 0 -1	1 0 -1 1 0 -1 -1 2 -1 0	7	1;2
3	15	45	3.1021	00:03:10	1 0 0 0 0 0 0 0 0 -1	0 -2 -1 2 1 0 0 -1 1 0	5	3;6
10	5	50	3.1169	00:02:31	1 0 0 0 0 0 0 0 0 -1	1 0 3 -2 -1 2 -4 -5 2 4	5	8;3
10	15	150	3.4041	00:11:08	1 0 0 0 0 0 0 0 0 -1	-1 0 0 0 1 0 2 2 -1 -3	2	10;5
15	15	225	3.3415	00:15:58	1 0 0 0 0 0 0 0 0 -1	2 3 -2 -4 0 0 -2 0 2 1	3	14;6

B. Low-Light situation

Luxlevel: 4 lux; Distance to subject: 1.00 meter; Number of images in the series: 60

Table 3.4: *Optimization results for Low Light Image-series*

# of iterations	# of candidates per iteration	Number of solutions explored	Performance value	Time taken (hh:mm:ss)	Initial Kernel	Resulting kernel	Resulting Step-length (%)	Result location (Iteration; Candidate #)
1	5	5	0.75262	00:00:28	1 0 0 0 0 0 0 0 0 -1	-1 1 1 0 0 1 -1 0 0 -1	3	1;2
1	15	15	0.88757	00:01:38	1 0 0 0 0 0 0 0 0 -1	-1 0 1 0 0 0 1 0 1 -2	7	1;1
3	5	15	0.97959	00:01:03	1 0 0 0 0 0 0 0 0 -1	0 0 0 0 0 1 0 -1 0 0	12	1;2
3	15	45	0.99134	00:02:31	1 0 0 0 0 0 0 0 0 -1	1 1 0 -1 -1 -1 -1 1 1 0	12	2;4
10	5	50	1.0414	00:01:45	1 0 0 0 0 0 0 0 0 -1	2 -1 -2 0 1 2 0 0 -2 0	19	3;2
10	15	150	1.0444	00:01:45	1 0 0 0 0 0 0 0 0 -1	2 2 -1 -4 0 0 3 1 -1 -2	12	10;7
15	15	225	1.0447	00:13:27	1 0 0 0 0 0 0 0 0 -1	2 1 0 0 0 2 -1 -3 -2 1	24	14;4

C. Mixed-Light situation

Table 3.5: *Optimization results for Mixed Light image-series*

# of iterations	# of candidates per iteration	Number of solutions explored	Performance value	Time taken (hh:mm:ss)	Initial Kernel	Resulting kernel	Resulting Step-length (%)	Result location (Iteration; Candidate #)
1	5	5	6.2856	00:01:02	1 0 0 0 0 0 0 0 0 -1	2 -1 -1 0 1 0 0 0 -1 0	12	1;4
1	15	15	5.5251	00:05:16	1 0 0 0 0 0 0 0 0 -1	2 0 0 0 0 0 1 0 -1 -2	5	1;1
3	5	15	6.5796	00:02:04	1 0 0 0 0 0 0 0 0 -1	0 0 0 1 0 0 1 2 -1 -3	5	3;2
3	15	45	6.2843	00:07:44	1 0 0 0 0 0 0 0 0 -1	-1 -1 2 0 -1 2 2 0 -2 -1	12	3;2
10	5	50	6.9989	00:07:39	1 0 0 0 0 0 0 0 0 -1	1 1 1 -1 -2 -2 0 0 2 0	12	2;3
10	15	150	6.959	00:19:42	1 0 0 0 0 0 0 0 0 -1	-2 -1 2 1 1 0 2 1 0 -4	24	8;2
15	15	225	6.9072	00:37:04	1 0 0 0 0 0 0 0 0 -1	-1 0 0 1 0 0 1 1 -1 -1	19	10;2

We use three different image series in mixed-light situation. We use a bright light image-series, low light image-series and a medium light image-series with lux levels 2450 lux, 218 lux and 4 lux respectively. Distance to subject is 1.00 meter and number of images in each series is 60. Figure 3.13 shows the one of the test chart under different illuminations.

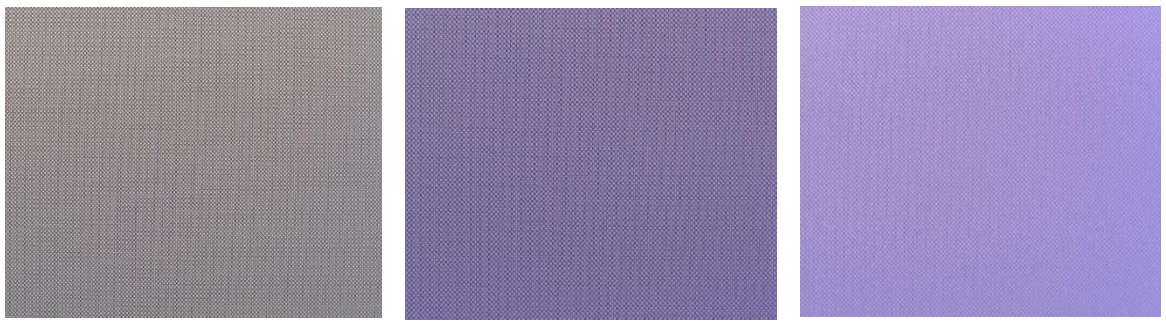
(a) 2450 lux (b) 218 lux (c) 4 lux

Figure 3.13: *Test charts under different illuminations, depicting varying level of details*

In the above three experiments, we run our system for a set of 7 different input parameters. We run with different iteration count and different candidates per iteration. Each row in the Tables 3.3 – 3.5 shows results from one run of the system. The first column shows the number of iterations (generations) in that run. The second column shows candidates per iteration. Number of candidates depends on the number of mutations and crossovers defined by the user. The third column shows total number of solution candidates explored in the whole run. This is calculated by multiplying number of iterations by number of candidates in each iteration. The fourth column shows the performance of the best candidate in that run. This performance is the sum of individual performances of contrast, step length and shape of the focus curve. Time taken by the whole run is shown next. Initial kernel, resulting kernel and resulting step-length are shown in the next columns. The step-length result is mentioned in terms of percentage of the total movement range, that is, a step-length result of 10 means the lens should jump 10% of total lens range in each step. The last column shows the instance in the run where the best candidate solution was found. So a value of 1;3 means the best candidate was found in iteration 1, candidate number 3. From Table 3.3, we see that as we increase the number of explored solutions, the quality of the results keeps getting better. The run with 150 candidates produces the highest performing solution. Moreover, since it is a randomized process, it is possible to stumble upon the optimal parameter during the initial iteration and the remaining iterations don't reach a better solution. This can be seen, for example, in Table 3.3 where first row shows better performance than the next 2 rows when the number of iterations in these next rows are 15 compared with a mere 5 in the first row. An example execution of our system (with Iterations = 3 and Number of candidates = 4) is presented in Appendix A.

To find suitable parameters for particular ambient conditions, we run the system and feed input series containing images from a specific ambient condition. We run the tool for 4000 iterations with 200 candidate solutions per iteration. After all the iterations are finished, we obtain parameters that perform very efficiently for that specific ambient condition. Biggest performance gains are seen in low-light scenes as the available contrast is low and it becomes even more important to obtain better focus statistics. These parameters are deployed to the actual consumer devices. For the sake of company confidentiality, we are not disclosing the specific results here.

Next, we conduct an experiment where we use the previous result as input and build upon previous best known solution to find a better solution. This way we keep on improving our solution. This is a good way to overcome the inherent slowness of genetic algorithms.

Table 3.6: *Improving results using previous results*

Initial value	Initial performance	Iterations	Candidates	Result	Result performance
0 0 0 0 0 0 0 0 0 0	0	3	7	0 -1 1 -1 -1 -1 3 1 0 -1	2.8046
0 -1 1 -1 -1 -1 3 1 0 -1	2.8046	3	7	0 -1 1 1 0 0 3 0 -1 -3	3.1004
0 -1 1 1 0 0 3 0 -1 -3	3.1004	3	7	0 -3 1 2 1 -1 3 2 -1 -4	3.3714
0 -3 1 2 1 -1 3 2 -1 -4	3.3714	3	7	-2 -2 1 2 1 -1 4 2 -1 -4	3.6217

Table 3.6 is an illustrative example of how we can use previous results in genetic algorithms. In all 4 instances, we run the optimization system for 3 iterations, with 7 solution candidates in each iteration. After the first run, we reach a performance of 2.8046. For the next run, we initialize the system with the previous results. We observe that we reach performance value of 3.1004 which is higher than the previous best. Similarly, after 4 runs, we have iteratively improved our result as evident from the last column of Table 3.6.

This GA based approach saves our time wasted in trial and error. It helps us to reach near-optimal solution. These results carry with them a numerical *performance* factor which gives us an objective measure of the quality of the images. This system presents a prototype for optimizing focusing and imaging related parameters. The system is easily extendable, that is, right now it optimizes only Focus Kernel and lens step-length. However, it can be easily extended to cover more parameters, for example, size of AF window, number of AF windows, de-noising filters etc. After getting an optimized result, we can set that result as input to the next run. So our results keep evolving with each run of the tool. The proposed solution does not require initial guesses to start the system. Normally, complex problem solving requires a deep understanding of the underlying concepts. However, an interesting benefit of GA based approach is that the human operator of this system need not be well versed with the complex science behind the parameters being optimized. For example, in high pass filter design case, we do not need complex mathematical filter calculations and designs, rather we can design the fitness function and run the GA based system. So if we know how the end results should look like, we design our fitness function accordingly and the GAs take responsibility of producing required results. The complex filter design process is hence completely eliminated from the loop.

One drawback of genetic algorithm based approach is the issue that is inherently present in any evolutionary algorithm, that is, genetic algorithms do not guarantee that the solution presented by them is the absolute best. The final solution is just the best solution among all the explored solutions. Further, since it is an evolutionary algorithm, which direction it will converge is never known in advance and as a result it may not reach the same solution each time. This problem is mitigated by running the algorithm for a very large number of iterations to ensure our algorithm explores as many solutions as possible. The greater the number of explored solutions, the greater is the probability that our algorithm will reach the near-optimal solution. It is also to be noted that GA based system is not a *fully automatic* system. This system helps in reaching a near-optimal solution. However, the final result still needs to be evaluated and verified by a human tester. Another challenge using GAs in the imaging domain is the design of fitness function. Imaging is a subjective domain. People can have different opinions and tastes on the quality of autofocus, colors, sharpness etc. Taste is even effected by regions, for example, people in Asia have different preferences for white balance than people in Europe. Therefore designing a fitness function to satisfy such subjective criteria is not trivial.

A GA based system is designed in a very specific way according to specific fitness criteria. The system is very useful when the same parameters need to be optimized for similar hardware. However if the nature of the hardware or the parameters is such that it is evolving rather fast, then the structure of the system has to be changed repeatedly to conform to the new criteria and this may not be time efficient. For example, in the autofocus algorithm, the parameters will change as the algorithm evolves. Therefore there is a risk this optimization system becomes deprecated rather fast. This has to be considered carefully when deciding whether to invest time in this optimization approach versus other available methods.

Currently, our system is optimizing two autofocus parameters, that is, focus kernel and lens step-length. Further development can be done to encompass other focusing parameters, for example, focus window shape, focus window size, focus kernel shape, focus algorithm pipeline enhancements etc. Further criteria can be added for evolution, for example, characteristic curves or models for different situations can help to find parameters best suitable for these situations. Similarly, characteristic curves for different camera modules can be added as a criterion for optimizing specific camera modules. In the future, this system can be expanded to other areas of imaging, for example, auto white balance and auto exposure parameters can be optimized using the approach presented in this thesis. The system can be made in such a way that it gets smarter with every new test set, that is, as more and more tests are performed on the system, it gets smarter by learning from results and this refinement process continues. An interesting future work in the context of this study could be to replace GA with some other optimization method (Simulated Annealing, Particle Swam Optimization etc.) and compare the convergence and robustness of the results.

4. CONCLUSION

The motivation of this thesis work was to help the camera engineers by presenting an alternative to tedious subjective evaluations required for field tests. We presented autofocus optimization based on the genetic algorithm. Optimization of two autofocus design parameters, that is, focus kernel and lens step-length was investigated. Experiments were performed using raw images from three different ambient conditions. The performance of the proposed system, such as robustness, convergence and reliability, was demonstrated. The resulting optimized parameters were tested in commercial devices and were able to extract enhanced sharpness statistics and reduced focusing time. Biggest improvements were observed in low-light where good quality statistics are essential to avoid focus failures.

At first, analysis of the image processing pipeline and autofocus was performed. Features of the camera components and their principles were studied. Introduction to the genetic algorithm was presented. An evolutionary environment was created to compare potential solutions whereby an objective function was designed in the context of optimizing autofocus. The fitness criteria comprised three metrics, that is, image contrast, speed of focusing and shape of the focus curve. Image contrast was measured using the focus profile, where left and right halves of the curve were weighted separately based on experiments. Focusing speed was proportional to the number of steps required to reach the highest-contrast lens position. Shape of the focus curve was graded against known focus profiles. The results of above fitness criteria were further scaled based on a-priori knowledge of focus profile characteristics.

Autofocus optimization presented in this thesis saves considerable time especially when parameters need to be optimized repeatedly due to new hardware. Once the optimization criteria have been designed and implemented, next generation cameras can be optimized just by capturing images and feeding these images to this system. The proposed solution is very flexible as new parameters requiring optimization can be easily added along with any new metrics for quality evaluations. Future improvements could include optimization of focus window size, shape, kernel orientation etc. This technique could be expanded to other areas of the imaging pipeline, for example, auto white balance, auto exposure and de-noising. There is, however, no guarantee a genetic algorithm will find the global optimum as there is a risk that the evolution might proceed in the wrong direction, never reaching the globally optimal solution. Another drawback of this method is a high computational requirement which limits the use of this method in phones as a real time application. Despite these drawbacks, the proposed system provides an autofocus optimization paradigm and opens a discussion for future research and developments.

REFERENCES

[1] Heyman S. Photos, Photos Everywhere, The New York Times. Available (accessed on 17.1.2016):http://www.nytimes.com/2015/07/23/arts/international/photos-photos-everywhere.html?_r=1

[2] Adams Jr JE. A fully automatic digital camera image refocusing algorithm. IEEE IVMSP Workshop, 2011 (pp. 81-86).

[3] Goldberg DE, Holland JH. Genetic algorithms and machine learning. Machine learning. 1988 Oct 1;3 (2):95-9.

[4] Kalevo O. Advanced Camera & Optics; Nokia internal training material. 2008.

[5] Spaulding K, Parulski K. Color image processing for digital cameras. Digital Color Imaging Handbook. CRC Press; 2002. Available from: http://dx.doi.org/10.1201/9781420041484.ch12.

[6] Nakamura J. Image Sensors and Signal Processing for Digital Still Cameras. 1st ed. Boca Raton, FL: CRC Press, Taylor & Francis Group; 2006.

[7] Nummela V. Camera Lenses; Nokia internal training material. 2008.

[8] Ramanath R., Snyder W., Yoo Y., and M. Drew, “Color image processing pipeline,” Signal Processing Magazine, IEEE, vol. 22, no. 1, Jan 2005.

[9] Krestyannikov E. and Samurov V. Nokia internal training material, 2009.

[10] Han JW, Kim JH, Lee HT, Ko SJ. A novel training based auto-focus for mobile-phone cameras. IEEE Transactions on Consumer Electronics. February 2011;57(1):232–238.

[11] He J, Zhou R, Hong Z. Modified fast climbing search auto-focus algorithm with adaptive step size searching technique for digital camera. IEEE Transactions on Consumer Electronics, May 2003;49(2):257–262.

[12] Dyer DW. A Practical Guide to the Watchmaker Framework. Available (accessed on 21.11.2015): http://watchmaker.uncommons.org/manual/ch01.html#d0e62

[13] Bäck T., Hammel U., and Schwefel H.-P., “Evolutionary computation: Comments on the history and current state,” IEEE Transactions on Evolutionary Computation. Vol. 1, no. 1, pp. 3–17, 1993.

[14] NEO Research Group, Introduction on Evolutionary Algorithms, University of Malaga. Available (accessed on 18.11.2015): http://neo.lcc.uma.es/opticomm/introea.html

[15] AI Group, Carnegie Mellon School of Computer Science. Available (accessed on 21.12.2015): https://www.cs.cmu.edu/Groups/AI/util/html/faqs/ai/genetic/part2/faq-doc-1.html

[16] Rajesh RJ, Kavitha P. Camera gimbal stabilization using conventional PID controller and evolutionary algorithms. IEEE International Conference on Computer, Communication and Control (IC4), Sep 10, 2015 (pp. 1-6).

[17] Goldberg DE. Genetic Algorithms in search, optimization and machine learning. Addison-Wesley, 1989.

[18] Back T. LIACS, Leiden University. Available (accessed on 19.10.2015): http://liacs.leidenuniv.nl/~csnaco/EA/slides/4.1%20-%20GA%20Basics.pdf

[19] Liu KH, Wiesel A, Munson DC. Synthetic aperture radar autofocus via semi definite relaxation. IEEE Transactions on Image Processing. June 2013;22(6):2317-26.

[20] Haiyang C, Daiyin Z, Jindong Z. FPGA Implementation of Two SAR Autofocus Algorithms. IEEE 11th International Conference on Dependable, Autonomic and Secure Computing (DASC), Dec 21 2013 (pp. 148-152).

[21] Zhang BJ, Zhang XL, Wei SJ. A circular SAR image autofocus algorithm based on minimum entropy. IEEE 5th Asia-Pacific Conference on Synthetic Aperture Radar (APSAR), Sep 1, 2015 (pp. 152-155).

[22] Berger T, Hamran SE, Odegaard N, Oyan MJ, Damsgard L. Stripmap autofocus of short range Ku-band synthetic aperture radar data. IEEE Radar Conference (RadarCon), May 10, 2015 (pp. 0243-0247).

[23] Hu K, Zhang X, He S, Zhao H, Shi J. A Less-Memory and High-Efficiency Autofocus Back Projection Algorithm for SAR Imaging. IEEE Geoscience and Remote Sensing Letters. Apr 2015;12(4):890-4.

[24] Gamadia M, Kehtarnavaz N. Enhanced low-light auto-focus system model in digital still and cell-phone cameras. 16th IEEE International Conference on Image Processing (ICIP), Nov 7, 2009 (pp. 2677-2680).

[25] Han JW, Kim JH, Lee HT, Ko SJ. A novel training based auto-focus for mobile-phone cameras. IEEE Transactions on Consumer Electronics. Feb 2011;57(1):232-8.

[26] Cakir S, Cetin AE. Autofocus of infrared cameras based on the cumulative probability of blur detection. IEEE Signal Processing and Communications Applications Conference (SIU), 2014 (pp. 1975-1978).

[27] Demir HS. A Comparison of Focus Measures for Autofocus Systems in IR Cameras. IEEE Signal Processing and Communications Applications Conference (SIU), 2015. (pp. 1558 - 1561).

[28] Vertan C, Florea C, Florea L, Badoiu S. On the performance of focus measures in infrared and near-infrared imagery. In International Symposium on Signals, Circuits and Systems (ISSCS), Jul 9, 2015 (pp. 1-4).

[29] Rasti P, Kiefer R, Anbarjafari G. Autofocus liquid lens by using sharpness measurement. 23th Signal Processing and Communications Applications Conference (SIU), May 16, 2015 (pp. 608-611).

[30] Tsai DC, Tsai ZM, Chen HH. A simulation tool for digital autofocus design. In International Conference on Consumer Electronics (ICCE), Jan 11, 2013 (pp. 224-225).

[31] Yamasaki, M. and Toyofuku, T. and Itoh, J. and Kodama, S. Focus detection apparatus using neural network means. US Patent 4,965,443. 1990. Available: http://www.google.com/patents/US4965443

[32] Nikon Corp., Autofocusing device for camera. 1996.

[33] Kennedy J., and Eberhart R. Particle swarm optimization. IEEE International Conference on Neural Networks, 1995.Vol. 4, pp. 1942–1948.

[34] Kennedy J. Bare bones particle swarms. Proceedings of the Swarm Intelligence Symposium, 2003. SIS’03. 2003 IEEE, pp. 80–87.

[35] Poli R., Kennedy J., and Blackwell T. “Particle swarm optimization,” Swarm intelligence, vol. 1, no. 1, pp. 33–57, 2007.

[36] Bahadur IM, Mills JK. Robust autofocusing in microscopy using particle swarm optimization. In IEEE International Conference on Mechatronics and Automation (ICMA), Aug 4, 2013 (pp. 213-218).

[37] Roberts M, Naftel AJ. A genetic algorithm approach to camera calibration in 3D machine vision. In IEE Colloquium on Genetic Algorithms in Image Processing and Vision, 1994 (pp. 12-1).

[38] Wan-Yu L, Kai X. A camera calibration method based on neural network optimized by genetic algorithm. In IEEE International Conference on Systems, Man and Cybernetics, 2007. (pp. 2748-2753).

[39] QiShen L, LiCai L, ZeTao J. A camera self-calibration method based on hybrid optimization algorithm. In Second International Symposium on Electronic Commerce and Security, 2009. ISECS'09. May 22, 2009 (Vol. 2, pp. 60-64).

[40] Merras M, Saaidi A, Nazih AG, Satori K. A new method of camera self-calibration with varying intrinsic parameters using an improved genetic algorithm. In 8th International Conference on Intelligent Systems: Theories and Applications (SITA), 2013 (pp. 1-8).

[41] Bouchouicha M, Puech W. A non-linear camera calibration with genetic algorithms. In 7th International Symposium on Signal Processing and Its Applications, 2003. (Vol. 2, pp. 189-192).

[42] Li J, Yang Y, Fu G. Camera self-calibration method based on GA-PSO algorithm. In IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS), 2011 (pp. 149-152).

[43] Chan VH. and Ravirala NS. Systems, methods, and apparatus for camera tuning and systems, methods, and apparatus for reference pattern generation. WO Patent App. PCT/US2010/034,079. 2010. Available: http://www.google.com/patents/wo2010129893a2

[44] Moghaddam ME. Out of focus blur estimation using genetic algorithm. In 15th International Conference on Systems, Signals and Image Processing, 2008. IWSSIP 2008. (pp. 417-420). IEEE.

[45] Zhang L, Zhang X. Measurement of the Optical Properties Using Genetic Algorithm Optimized Neural Networks. In IEEE Symposium on Photonics and Optoelectronics (SOPO). May 16, 2011 (pp. 1-4).

[46] Karungaru S, Fukumi M, Akamatsu N. Neural networks and genetic algorithms for learning the scene illumination in color images. In IEEE International Symposium on

Computational Intelligence in Robotics and Automation, 2003. Jul 16, 2003 (Vol. 3, pp. 1085-1089).

[47] Bilal K, Qureshi J. Nature inspired optimization techniques for camera calibration. In 4th International Conference on Emerging Technologies, 2008. ICET 2008. Oct 18, 2008 (pp. 27-31).

[48] Naït-Ali A. Genetic algorithms for blind digital image stabilization under very low SNR. IEEE Transactions on Consumer Electronics. Aug 2007;53(3):857-63.

[49] Abellard A, Bouchouicha M, Ben Khelifa MM. A genetic algorithm application to stereo calibration. In IEEE International Symposium on Computational Intelligence in Robotics and Automation, 2005. CIRA 2005. Jun 27, 2005 (pp. 285-290).

[50] Kumar S, Thakur M, Raman B, Sukavanam N. Stereo camera calibration using real coded genetic algorithm. In IEEE Region 10 Conference TENCON - Nov 19, 2008 (pp. 1-5).

[51] Egan TM, Picton PD. Behaviour of a simple genetic algorithm searching for bright and edge pixels in an image. In IEE Colloquium on Genetic Algorithms in Image Processing and Vision, 1994 (pp. 2-1).

Appendix A: Sample execution of the system

To demonstrate working of the system, we ran the tool with the following input parameters:

```
MAX_ITERATIONS = 2;

NUMBER_OF_PARENTS = 3;
NUMBER_OF_MUTATIONS = 2;
NUMBER_OF_CROSSOVERS = 2;
```

Following is the example run of our system:

```
Number of input image-series = 1

          ***************** ITERATION # 1 HAS STARTED *****************

1)   Candidate_Performace is 1.9337. Value of StepLength is 4 and value of

Candidate_Kernel =
    1     0     1     0     1
   -1    -1     0     0    -1

 2)   Candidate_Performace is 2.6171. Value of StepLength is 1 and value of

Candidate_Kernel =
    2     1     0    -1     0
    0     1    -1     0    -2

 3)   Candidate_Performace is 1.8006. Value of StepLength is 8 and value of

Candidate_Kernel =
    1     0     0     1     0
    1    -1     1    -2    -1

 4)   Candidate_Performace is 2.746. Value of StepLength is 6 and value of

Candidate_Kernel =
    1     0     1    -1     0
   -2     1     0     1    -1

 5)   Candidate_Performace is 2.9467. Value of StepLength is 2 and value of

Candidate_Kernel =
    1     1    -2    -1     2
    0     0     0     0    -1
 6)    Candidate_Performace is 2.3824. Value of StepLength is 1 and value of

Candidate_Kernel =
     1     1    -1     0     0
     0    -1     1     0    -1

 7)    Candidate_Performace is 2.1892. Value of StepLength is 1 and value of

Candidate_Kernel =
     1     0     0     0     0
     0     0     0     0    -1

 8)    Candidate_Performace is 2.1892. Value of StepLength is 1 and value of

Candidate_Kernel =
     1     0     0     0     0
     0     0     0     0    -1

9)    Candidate_Performace is 2.1892. Value of StepLength is 1 and value of

Candidate_Kernel =
     1     0     0     0     0
     0     0     0     0    -1

 10)    Candidate_Performace is 2.1892. Value of StepLength is 1 and value of

Candidate_Kernel =
     1     0     0     0     0
     0     0     0     0    -1

 11)    Candidate_Performace is 2.1892. Value of StepLength is 1 and value of

Candidate_Kernel =
     1     0     0     0     0
     0     0     0     0    -1


  ITERATION # 1 HAS FINISHED and Best_Performance from Iteration # 1 is 2.9467
with StepLength = 2 and

Kernel =
     1     1    -2    -1     2
     0     0     0     0    -1


             **************** ITERATION # 2 HAS STARTED ****************
```

1) Candidate_Performace is 3.0963. Value of StepLength is 3 and value of

```
Candidate_Kernel =
     1     1    -3    -1     2
     0     0    -1     1     0
```

2) Candidate_Performace is 3.1755. Value of StepLength is 4 and value of

```
Candidate_Kernel =
     1     1    -2    -1     1
     0     0     0     0     0
```

3) Candidate_Performace is 2.9144. Value of StepLength is 9 and value of

```
Candidate_Kernel =
     0     0    -1     1     0
    -2     2     2     0    -2
```

4) Candidate_Performace is 2.1058. Value of StepLength is 4 and value of

```
Candidate_Kernel =
     1     2     1    -1     1
    -2     0    -2    -1     1
```

5) Candidate_Performace is 2.5413. Value of StepLength is 2 and value of

```
Candidate_Kernel =
     3     0     0    -1    -1
     1     2    -2     0    -2
```

6) Candidate_Performace is 2.7719. Value of StepLength is 1 and value of

```
Candidate_Kernel =
     1     1    -1    -1     1
    -1     1     0     1    -2
```

7) Candidate_Performace is 2.9467. Value of StepLength is 2 and value of

```
Candidate_Kernel =
     1     1    -2    -1     2
     0     0     0     0    -1
```

8) Candidate_Performace is 2.746. Value of StepLength is 6 and value of

```
Candidate_Kernel =
     1     0     1    -1     0
```

```
   -2    1    0    1   -1

 9)    Candidate_Performace is 2.6171. Value of StepLength is 1 and value of

Candidate_Kernel =
     2    1    0   -1    0
     0    1   -1    0   -2

 10)    Candidate_Performace is 2.4752. Value of StepLength is 4 and value of

Candidate_Kernel =
     2    0    0   -1    1
    -1    0    0    0   -1

 11)    Candidate_Performace is 2.8496. Value of StepLength is 2 and value of

Candidate_Kernel =
     1    1   -1   -1    1
     0    0    0    0   -1

 ITERATION # 2 HAS FINISHED and Best_Performance from Iteration # 2 is 3.1755 with
StepLength = 4 and

Kernel =
     1    1   -2   -1    1
     0    0    0    0    0

*************************************************************
*********************  RESULTS  ****************************
*************************************************************

Number of input series = 1
Number of images in each series = 59

Number of Iterations = 2
Number of Candidates (in each iteration) = 11

Winning_Candidate occured in Iteration  2 with Candidate #  2
Winning_Candidate Performance = 3.1755
Winning_Candidate StepLength = 4   ( 7 % )

Winning_Candidate_Kernel =
     1    1   -2   -1    1
     0    0    0    0    0

Program took 2 mins, 13.0 secs
 *************************************************************
```