Design Optimization of a Microelectromechanical Electric Field Sensor Using Genetic Algorithms

By Mark Roy

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES OF
THE UNIVERSITY OF MANITOBA
IN PARTIAL FULFILMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF MANITOBA

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba

Dedicated to my beautiful wife Serena,

for her loving support,

and to

all of my teachers and professors

who have helped guide me throughout my education.

Abstract

This thesis studies the application of a multi-objective niched Pareto genetic algorithm on the design optimization of an electric field mill sensor. The original sensor requires resonant operation. The objective of the algorithm presented is to optimize the geometry eliminating the need for resonant operation which can be difficult to maintain in the presence of an unpredictable changing environment. The algorithm evaluates each design using finite element simulations. A population of sensor designs is evolved towards an optimal Pareto frontier of solutions. Several candidate solutions are selected that offer superior displacement, frequency, and stress concentrations. These designs were modified for fabrication using the PolyMUMPs fabrication process but failed to operate due to the process. In order to fabricate the sensors in-house with a silicon-on-glass process, an anodic bonding apparatus has been designed, built, and tested.

Acknowledgements

I owe my deepest gratitude to my advisor, Dr. Cyrus Shafai for his guidance, knowledge, and support that helped make this thesis and my graduate studies possible. Dr. Shafai is truly devoted to advancing technology and to teaching, as is plainly visible to anybody who has the chance to meet with him and share in his excitement for groundbreaking technology and novel ideas.

I would also like to thank Dwayne Chrusch, the operations manager of the Nanofabrication Laboratory, for his assistance with cleanroom procedures and invaliable assistance with the design and assembly of the anodic bonding apparatus and associated vacuum equipment.

I would also like to thank my colleagues Gayan Wijiweera whose research has made my own research possible, and Yu Zhou whose research and insight on the electric field mill project has been of great value to my own work. Thanks also to all of the other students in the Nano Fabrication Systems Laboratory for their supporting research and interest.

Thanks to Amy Dario for her administrative help filling out forms, applying for scholarships, reminding me of deadlines and most importantly helping to track down stray faculty. Amy is able to keep the ECE graduate program running smoothly through her tireless efforts and dedication.

I would also like to thank Zoran Trajkoski, Sinisa Janjic, and Ken Biegun for their help ordering components and manufacturing printed circuit boards. Thanks also to Cory Smit for his assistance in the machine shop.

My gratitude to Mount-First Ng and Guy Jonatschick for their support and maintenance of the computers, networks, and software that was extensively used throughout my thesis.

I would like to extend my appreciation also to Manitoba Hydro and the province of Manitoba for funding received that helped make this research possible. Thanks to CMC microsystems for their industry liason work making available software and fabrication processes for academic use.

And last, but certainly not least, I would like to extend my sincere gratitude to my wife and family for their love and support throughout my academic career, whom without I would not have been able to succeed.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Electric Field Mills

It is often desirable to measure electric fields in manufacturing and research. In manufacturing, electrostatic charging can be a serious problem that needs to be monitored to prevent, for example, damage to sensitive electronic devices being manufactured, the accidental ignition of flammable gasses, or to assess the risk of shock to employees [1]. It is also useful for studying the electric fields present in the atmosphere during various phenomena such as lightning [3]. There has even been some research to suggest that measurement of electric field changes in the earth could be used a predictor of earthquakes [8]. Another equally important research area is the study of electric fields beneath high voltage dc (HVDC) power lines and to determine their environmental impact at ground level [2]. These applications and others drive the need for a cheap, effective, and sensitive electric field sensor.

Traditional field mills consist of an electric motor that drives an earthed rotor above a fixed set of electrodes [1]. Figure 1.1 illustrates a traditional field mill based on the field mills described in [1, 2]. The primary chopper spins at a constant angular velocity causing the electrodes beneath to be alternately exposed or shielded from the

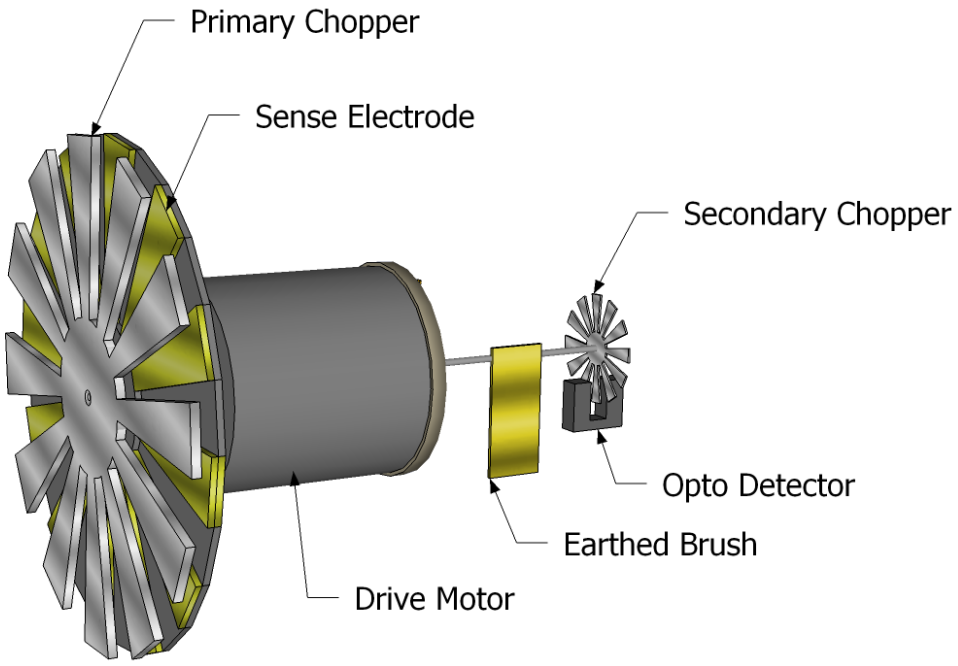


Figure 1.1: *A traditional field mill based on the field mills described in [1, 2]*

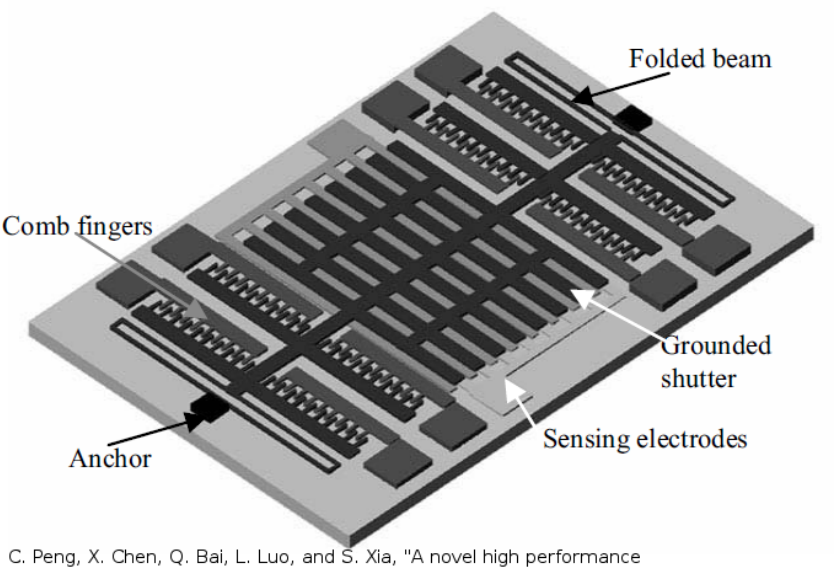
electric field. The electric field induces a current in the sense electrode that can be measured electrically. A secondary chopper and LED/phototransistor detector are configured to generate a synchronous signal used to measure the phase of the sense currents ac component. The sense current is related to the electric field E by the following general equation [3]:

$$i_e(t) = \epsilon_o E \frac{dA_e}{dt} \tag{1.1}$$

where E denotes the electric field magnitude, and A_e denotes the exposed area of the sense electrode as a function of time. The electric field magnitude E can be recovered directly from the ac component. Using this type of apparatus, it is possible to accurately measure values from 100 V/m to 100 kV/m [2].

1.2 Micro Electric Field Sensors

In recent years, there has been some research into the development of micro-electric field sensors. The small scale of a micro mill allows for a sensor with similar sensitivity range down to 100 V/m while significantly lowering cost and power usage [3]. In the case of atmospheric electric field study, miniaturized sensors can easily be included on disposable weather balloons where traditional field mills may be too expensive and heavy to outfit on a unrecoverable balloon [9]. Figure 1.2 shows an electrostatic micro field mill that was designed by [3]. It operates on the same principles as the traditional mill of Figure 1.1 but with translational oscillations as opposed to a continuous rotational motion. The grounded shutter is suspended above a fixed set of sensing electrodes by folded beam springs. The shutter is then oscillated at its resonant frequency, ~ 4.1 kHz, by electrostatic comb actuators which allows for sufficient motion of the shutter at very low power dissipation. This sensor has been used to successfully measure atmospheric electrostatic field variations at altitudes as high as 8000 m [3].



C. Peng, X. Chen, Q. Bai, L. Luo, and S. Xia, "A novel high performance micromechanical resonant electrostatic feld sensor used in atmospheric electric feld detection," in MEMS 2006, Jan. 2006, pp. 698-701. Copyright 2006 IEEE

Figure 1.2: *An electrostatic micro field mill design [3].*

Figure 1.3 shows another design for an electrostatic field mill [4]. This design consists of a single 10 μm by 10 μm sense electrode that is periodically exposed to the incident electric field by an aperture in a grounded shuttle mass [4]. Like the last design, the shuttle mass is supported by folded beam springs and is also driven at its resonant frequency, 7.6 kHz, by electrostatic comb actuators. The sensor was shown to have a sensitivity of approximately 35 $\mu V/kVm^{-}1$ and was tested in fields as large as 500 kV/m [4].

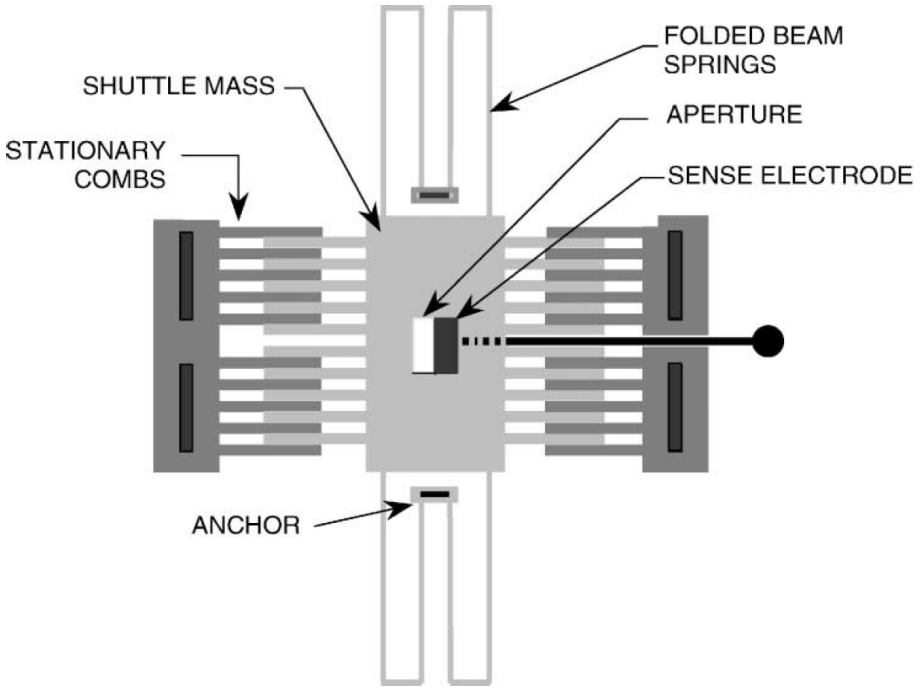


Figure 1.3: *Another electrostatic micro field mill design [4].*

One drawback with using electrostatic actuation is that the actuation voltages range from $25 - 100$ V [4, 3] which at the scale of a micro-sensor appears itself as a very large electric field. This interference may limit the field sensitivity at the lower end of the range. Using bent beam thermal actuators a field mill device was designed and fabricated by [10]. Their field mill was able to measure a field down to 240.8 V/m [10].

1.3 Other Electric Field Sensors

The field mill type electric field sensor is by no means the only way of measuring electric fields. There are also several sensor designs that measure electric field strength optically. One such method is based on the deflection of an electron beam in the presence of an electric field [9]. A very sharp emission tip made from silicon is used to generate the field emission of electrons [9]. An anode is placed some distance above the field emission tip and features a number of concentric rings. In the presence of an electric field, the electrons from the field emission tip are deflected linearly and collected by the anode rings. The principle has been tested using a glass vacuum tube and a thermal electron source and has been shown to work for fields as high as 150 kVm [9].

Another electric field sensor that has been developed is based around electro-optical crystals such as $LiNbO_3$ and $LiTaO_3$ [11].

1.4 Electrothermal Field Mill Design

The micro electric field sensor optimized in this thesis was the electrothermal field mill developed by G. Wijeweera [5] at the University of Manitoba, pictured in Figure 1.4. This field mill consists of a perforated shutter mass, suspended by loop springs at its corners, and is actuated by thermal actuators. The thermal actuators are connected to the shutter through a connecting lever that provides a mechanical advantage converting the large force afforded by the thermal actuators to displacement. The shutter, measuring 1 mm by 1 mm, is grounded through one of the four supporting springs providing shielding from an incident electric field to the electrodes beneath. The comb-shaped electrodes beneath the shutter are offset from one another so that one set of electrodes is completely exposed when the other is shielded

by the shutter providing a differential signal. When operated at its natural resonant frequency of approximately 3796 Hz, the shutter displacement is large enough to completely shield/expose the 14 µm wide electrode combs beneath [5]. The 14 µm size for the electrodes was chosen as the minimum feature size in the Pyrex cavity of the MicraGEM fabrication process [12].

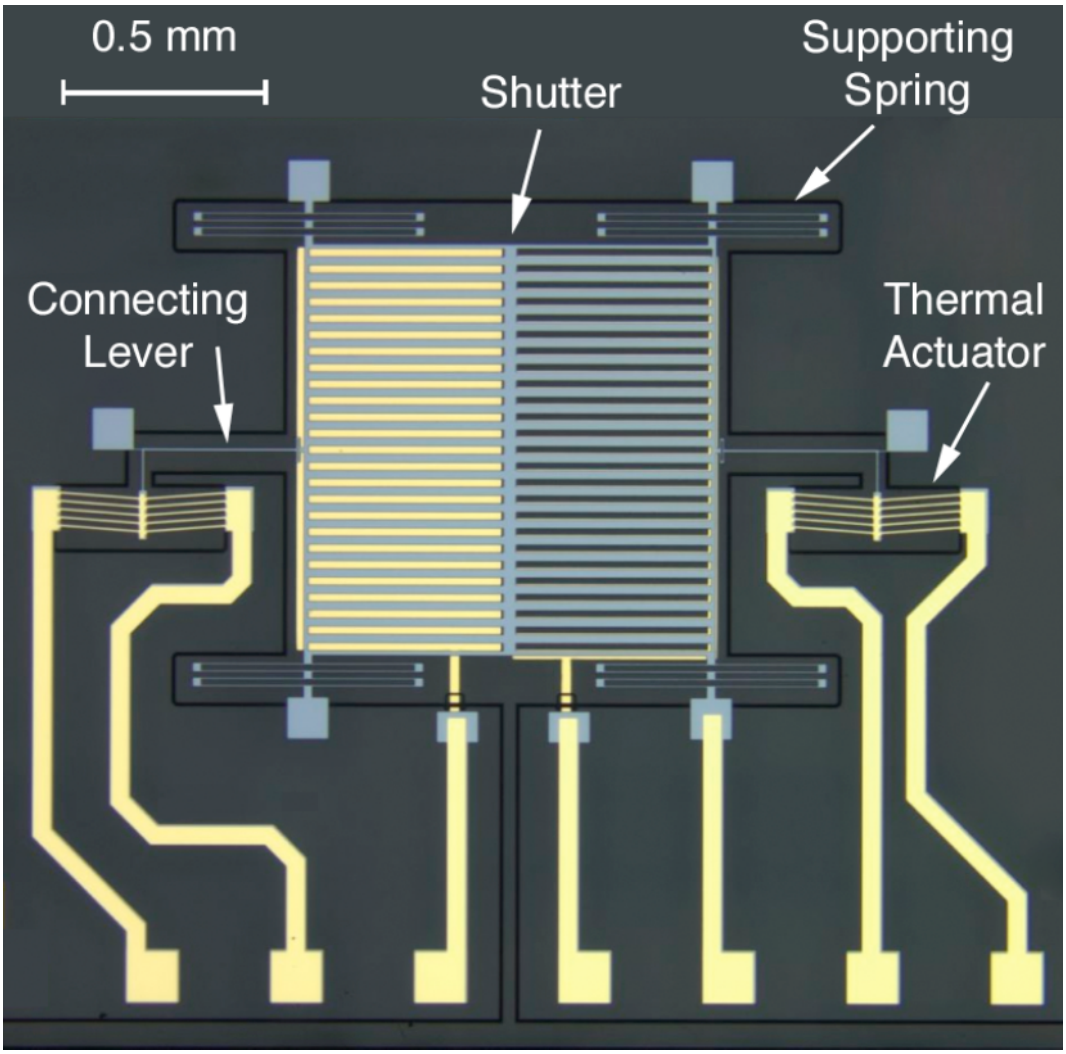


Figure 1.4: *Picture of an electrothermally actuated electric filed mill sensor [5].*

Figure 1.5 shows the response of the sensor to electric fields from 42 V/m to 5000 V/m. This sensor has a very good response to small electric fields compared to the electrostatic field mills discussed in section 1.2 since it is driven by low voltage thermal actuators and makes use of offset electrodes to provide a differential signal

which assists with rejection of common mode interference. Also, because the springs are responsible for the back-swing of the field mill's shutter, the actuation frequency is half that of the sensing frequency, separating it in the frequency domain as a source of noise from the sensor's response.

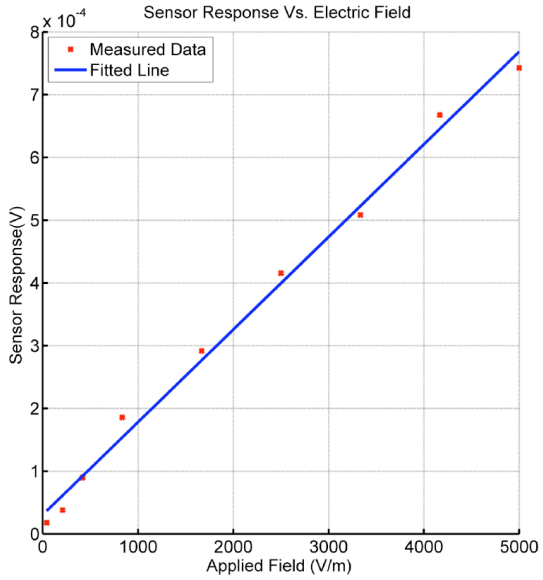


Figure 1.5: *Sensor response of electrothermally actuated field mill design [5].*

Although this sensor design was proven to work quite well, the requirement of resonant operation complicates packaging since the sensor must be operated in a vacuum to maintain the high-quality resonance necessary to achieve the 14 µm displacement. It was found that the resonant frequency of the sensor is sensitive to changes in vacuum pressure or ambient temperature. This is supported by research that shows a frequency shift in microbeam polysilicon resonators when temperature or pressure are varied [13]. It was also found that a large electric field ($\sim 100\,\text{kV}$ or greater) causes a small upward force on the shutter. The force exerted on the shutter can be approximated by using the parallel plate electrostatic force equation [14]:

$$F_{es} = \frac{\epsilon_r \epsilon_0 A V^2}{2x^2} \tag{1.2}$$

where ϵ_r and ϵ_0 are the relative and free space permittivity constants respectively, A

is the area of the shutter, V is the voltage potential on a plate above the shutter, and x is the distance above the shutter that the plate is positioned. The area A is approximately $5.25 \times 10^{-7}\,\mathrm{m}^2$ (1 mm by 1 mm minus the area of 50 slits measuring 475 µm by 20 µm). The voltage potential V is fixed at 1 V and the plate is positioned 10 µm above the shutter, resulting in a field of 100 kV/m. Using these values in the F_{es} equation above yields an upward force of ~ 23 nN. As the field increases, the magnitude of the force on the shutter increases quadratically, reaching ~ 580 nN at a field strength 500 kV/m. The pull is not enough to cause significant vertical displacement of the shutter, but is enough to cause a slight shift in resonant frequency. In order for a resonant sensor to operate in the presence of a large electric field and to withstand environmental changes in pressure or temperature, it is necessary to either electronically track changes the resonant frequency or to modify the design of the sensor to allow for non-resonant operation.

1.5 Automated Design Optimization

The natural progression of computer aided design (CAD) of microelectromechanical (MEM) devices and systems is for computer software to take a larger role in the design by algorithmically determining the best size, shape, and placement of components. A design can be loosely or tightly constrained, depending on the complexity or requirements for the system. When loosely constrained, an algorithm may be allowed to choose between different components, beams, springs, and actuators, for example, that are used to synthesize the device. A tightly constrained algorithm, on the other hand, may be restricted to a pre-determined device geometry in which the algorithm is assigned the task of sizing and tuning mechanical components such as beams, actuators, and springs.

Many different algorithms exist to suit this type of design, the simplest being

an iterative search of a single design parameter. For example, when tuning the frequency of a resonator, the size of a square proof-mass may be iteratively incremented to achieve the desired resonant frequency. To evaluate a designs performance, an algorithm may solve a simple pre-determined equation, a more complex dynamic equation, or it could be fed to a finite element simulation, SPICE simulation, or any number of other simulations that are available. Some such optimization algorithms considered here for MEMS design include simulated annealing, genetic algorithms, particle swarm optimization, seeker optimization, topology optimization, and tabu search algorithms. The best algorithmic tool for the job depends on the problem, and more importantly, the size and shape of the objective space. Given the multitude of options available for evaluating a design and algorithms for guiding the search for an optimal solution, a modular optimization framework has been proposed by [15] allowing designers to pick and choose evaluation tools and algorithms to guide their design. However, according to the generally accepted *no free lunch* theorem [16], no black box optimization algorithm provides, on average, better performance than any other algorithm across all types of optimization problem. It is important to note, however, that the *no free lunch* theorems pertain to black-box algorithms which operate abstractly from the problem and its traits. Essentially, it proves that there is no "silver bullet" when it comes to optimization algorithms.

1.5.1 Simulated Annealing

Simulated annealing is a stochastic search algorithm that models the controlled annealing of a metal alloy as it is cooled, achieving a minimal energy state [17]. In the case of MEMS design, the algorithm starts with a randomly generated design. An initial temperature and step size vector are specified before running the simulated annealing algorithm. The design is then analyzed by simulation or dynamic equation and the objectives are extracted and used to formulate the "energy" of the state. A

potential new design is generated by randomly perturbing the current design parameters using the step size vector [17]. This new designs is again evaluated and if a it has a lower energy than the current design, it is accepted as a more optimal design from which to continue. If, however, the design has a higher energy, then it may still be accepted by the Metropolis criterion [17]:

$$p' < e^{(E'-E)/T_k} \tag{1.3}$$

where p' is a randomly generated value in the range $[0, 1]$, E' and E are the energies of the proposed and current designs respectively, and T_k is the global parameter for the current temperature of the system as it cools. If the above Metropolis criterion is met, then the proposed solution replaces the current solution. This probability of accepting designs that perform worse than the current design allows the algorithm to "hill-climb", and escape local optimums in search of a better global optimum [17]. This process is repeated until there is little further change to the energy of the current design state, indicating that thermal equilibrium has been reached. At this point, the global temperature T_k, and step size vectors are reduced and the algorithm repeats. This allows the algorithm to progressively optimize smaller details of the design. Once the target cool temperature has been reached or the desired objectives are met, the algorithm is terminated.

In [17], simulated annealing was successfully used to optimize the design of a microresonator based gyroscope. Simulated annealing has also been applied to the design of a MEMS resonator by [18]. For a detailed review of the simulated annealing algorithm and it's applications to various types of problems, please refer to [19].

1.5.2 Genetic Algorithms

Genetic algorithms are a class of algorithms based on the concept of evolution. GAs are useful for finding the global optimum of a function that is either complex or unknown. A solution is encoded into a series of parameters that make up a single "genome". An initial population of genomes is generated at random and various genetic operators are applied in order to pass desirable traits to the next generation. Some such genetic operators include selection, crossover, elitism, and mutation. Selection is the method by which a solution is chosen to continue as the basis for the next generation. Crossover is a method of breeding two or more solutions from the selected set and combining them to form new solutions that inherit traits from their parents. Elitism is a mechanism which preserves good solutions by adding them to the next generation as-is without crossover or mutation. Finally, mutation is a mechanism by which new genetic material is introduced by randomly perturbing the parameters of solutions. Specific GAs may implement different genetic operators or apply them differently in order to suit their specific needs. A good overview of genetic algorithms and evolutionary computing can be found in [20].

Genetic algorithms have been applied to both the synthesis of MEMS [21] and the optimization existing MEMS designs [22, 23, 24]. The design of a reconfigurable microstrip antenna was optimized using genetic algorithms [22]. Others have applied GAs to the optimization of miniature piezoelectric forceps [23], accelerometers [24], resonators [7] and many other devices. A review of the current state of GAs applied to MEMS design is in [25]. For more details on the specifics of genetic algorithms, please refer to chapter 2.

1.5.3 Particle Swarm Optimization

Particle swarm optimization is a class of algorithms that attempt to mimic the behaviour of swarms in nature. For example, a flock of birds is able to fly together in unison, avoiding predators. In this case, the social behaviour of the birds provides an evolutionary benefit to the group as a whole [26]. A similar comparison can be made to schools of fish. When applied to an optimization problem, a population of candidate solutions is encoded as a set of "particles" in a swarm. Each particle maintains a position and velocity vector within the search space that is mapped to the design variables of the problem. Each particle in the swarm is evaluated using a simulation, or dynamic equation. After each pseudo-time increment, every particle in the swarm updates its position and velocity based on the fitness value of each particle, the best fitness value of the swarm, and a randomly generated real value, using a few simple equations. Particles are attracted towards their previous best position as well as the best position of the population [27]. This process is then repeated until, hopefully, the swarm is guided towards a global optimum [26]. Other operators such as the best ever position, maximum velocity, and inertia can be added to further tune the performance of the algorithm.

Particle swarm optimization has been shown to perform as well or better than other algorithms, including genetic algorithms, for a few small structural design problems in [26]. These problems include optimization of the cross-sectional area of beam elements in a truss, and optimization of stress distribution in a torque arm [26]. An advantage of particle swarm optimization algorithms is that they can be quite simple to implement when compared to more complex algorithms such as evolutionary algorithms or simulated annealing. Also, like genetic algorithms, particle swarm optimization lends itself very well to parallelization. Particle swarm optimization has been successfully used by [27] to optimize the geometry of micro-resonator load cells

for the force and strain measurements of materials or micro-devices. The competing objective functions used were maximum force resolution and range.

1.5.4 Seeker Optimization

Similar to particle swarm optimization, described above, seeker optimization operates on a group of "seekers" whose position is mapped to the design variables of the problem. The behaviour of each seeker is modelled to be similar to how an intelligent person might search for a solution to a problem [28]. A group of seekers is initialized with uniformly random distribution throughout the parametric design space. To update their positions in the next pseudo-time interval $t+1$, their positions are updated according to a simple equation [28]:

$$\vec{x}(t+1) = \vec{c} + \vec{d} \cdot \vec{r} \cdot \sqrt{-log(\vec{\mu})} \tag{1.4}$$

where $\vec{c}$ is the seekers current position, $\vec{d}$ is the direction that the seeker should travel, $\vec{r}$ is the allowed search radius of the seeker, and $\vec{\mu}$ is a trust degree parameter. The algorithm chooses the direction of travel, $\vec{d}$ based on a few simple rules. It considers the current direction of the seeker and whether or not it has improved in fitness since the last time interval, the direction of travel towards or away from a seeker with the highest fitness in a local neighbourhood region, and the direction of travel towards or away from the global seeker with the highest fitness. It may also consider the direction of travel towards or away from the seekers previously best fitness value. The search radius is an important algorithm parameter that is chosen based on the design variables' range and the distribution in objective space; it can be difficult to choose, but there are several suggested methods for choosing a search radius presented in [28]. The final parameter, $\vec{\mu}$, determines the level of trust in a given seeker; it is based on the relative fitness of the seekers position with that of the rest of the group. The

seeker with the best fitness would have the highest possible trust degree $\mu = 1$, while seekers with lower fitness values would be assigned a lower trust degree. A Seeker optimization algorithm was compared to particle swarm optimization and continuous genetic algorithm by optimizing six non-linear functions of increasing complexity and variable count [28]. It was found that seeker optimization was able to optimize the variables faster than both particle swarm and genetic algorithms with up to 10 variables.

Although still relatively new, a seeker optimization algorithm has been successfully employed to the optimization of a variable-capacitance micromotor design [29]. The objectives of the optimization algorithm were to maximize torque while minimizing ripple torque by varying the geometry of the motor design. The fitness was evaluated using finite element modelling with COMSOL multiphysics. The design was also optimized using a genetic algorithm to compare its performance with seeker optimization. It was found that the seeker optimization algorithm performed significantly faster than the genetic algorithm [29].

1.5.5 Topology Optimization

Topology optimization is a method of optimal design commonly used for the design of compliant mechanisms. In an optimal compliant mechanism, stress and strain are uniformly distributed throughout the entire structure, making it possible to be fabricated with fewer moving pieces and lower stress concentrations that ultimately result in a more robust and compact device. In MEMS, compliant mechanisms are commonly used for displacement amplifiers [6, 30], grippers [31], and other mechanisms. A compliant mechanism topology optimization problem starts with a bounding box that limits the size of the device. Input and output ports are assigned, each of which has a specific location, and prescribed force and displacement vectors. The structure is then initialized as a complete ground truss structure consisting of nodes

and beams uniformly distributed throughout [6]. Each beam element is assigned a variable cross-sectional area that determines its stiffness. Nodes may optionally be constrained in one or more axes; at least one node should be constrained for the problem to be solvable. Once the problem has been defined, a global stiffness matrix is generated and solved as a large linear system [6]. The solution to the matrix is the required cross-sectional area for each beam element. Once solved, beam elements below a set minimum can be removed from the truss revealing the desired topology for the compliant mechanism. Next, the exact position of nodes is sometimes perturbed algorithmically to try and further optimize the geometry.

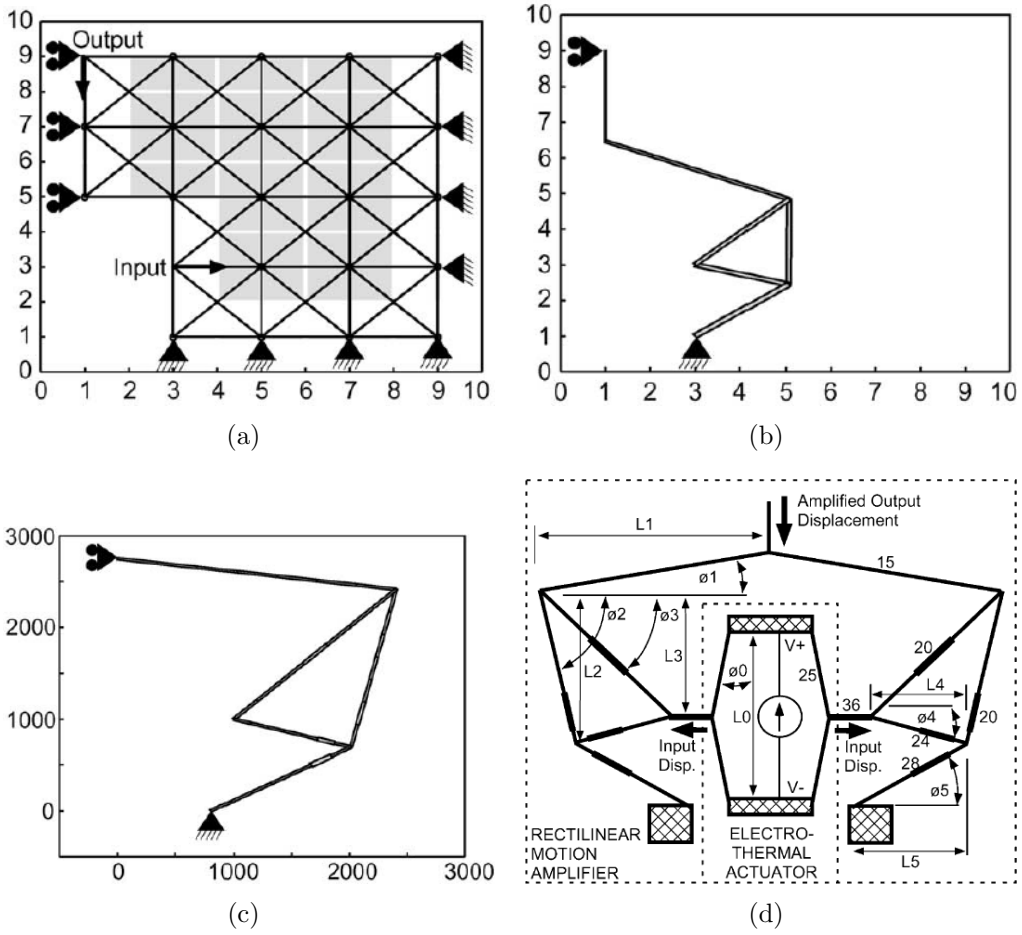


Figure 1.6: *A displacement amplifying mechanism designed for an electrothermal actuator [6]. Figure (a) shows the initial ground truss structure with constraints. Figure (b) shows the solved topology. Figure (c) shows the mechanism after the node locations have been optimized. Figure (d) shows the finished mechanism connected with its thermal actuators.*

This type of topology optimization has been applied successfully to the design of MEMS devices. In [30] displacement amplifying compliant mechanisms are designed using topology and size optimization methods for use in sensor applications. Others have used topology optimization to design high amplification microtransmissions for electrothermal actuators, which typically provide large force but poor displacement [6]. Figure 1.6 shows an example of a compliance mechanism designed by [6]. Starting with a ground truss structure in Figure 1.6(a), the problem is defined by setting the input and output ports, as well as adding constraints. In this example, the design is symmetrical so the entire left side of the truss is constrained only in one axis. Figure 1.6(b) shows the topology of the mechanism found by solving the stiffness matrix equations. Next, the node locations are shifted slightly (they are constrained within the grey boxes seen on the truss in Figure 1.6(a)) to further optimize the mechanical advantage. Finally, Figure 1.6(d) shows how the finished compliant mechanism is connected to an electrothermal actuator. The final fabricated mechanisms were able to achieve significant displacement amplification as high as 9.3 [6]. .

1.5.6 Tabu Search Optimization

Tabu (or taboo) search optimization operates by choosing a starting solution s randomly located within the parametric design space [32]. The design space is discretized into adjacent spheres or hyper-rectangles of a predetermined size. Several closely related neighbours are selected from the solution, a neighbour being defined as a randomly chosen solution within an adjacent hyper-rectangle or sphere in design space. Each of the neighbour solutions is evaluated and the centre of the neighbouring hyper-rectangle with the best fitness value is chosen as the new starting solution, regardless of whether it is better than the current solution [32]. The previous solution is then added to a "tabu" list which keeps track of previously chosen solutions. In order to prevent cycles from forming, a solution may not be chosen for the next iter-

ation if it is in the tabu list [32]. Once the tabu list is full, the first solution added to the list is removed. This allows the algorithm to eventually go back and explore the same area again, but attempts to limit local cycles. A second list of promising areas is also maintained. If when traversing the search space an unacceptable deterioration of the fitness value is encountered in all directions around an area, then the centre of that area is added to a list of promising areas [32]. After a specified number of iterations without the discovery of a new promising area, the algorithm then proceeds to identify the most promising area in the list for a more intensive search. During the intensified search, the tabu list is reset and the area is searched again using the same tabu search method above with the starting point being the centre of the most promising area. After a specified number of iterations with no further improvement, the size of the discretization hyper-rectangle (or sphere) is halved, and the intensified search repeats. After a specified number of intensification iterations, the algorithm is complete, and the best solution found is returned. The algorithm is compared to other algorithms, including variations of tabu search and simulated annealing, and is found to perform equally or better for functions up to 100 variables [32].

1.6 Research Objectives

The main research objective of this thesis is to investigate the mechanics necessary for non-resonant operation of the micromachined electric field mill designed by [5]. It is desirable to operate below the resonant frequency to alleviate issues with tracking resonance through changing environmental conditions such as temperature, pressure, and electric field. In order to achieve this goal, new finite element models are created and analyzed using transient simulations that had not previously been done. This thesis also attempts to validate the results of these simulations by comparing them to measurements made on fabricated sensors and electrothermal actuators.

After reviewing several optimization algorithms, genetic algorithms are selected as the evolutionary algorithm to be used for optimization of the electric field sensor. Genetic algorithms and their application to MEMS are further investigated and applied. Genetic algorithms were chosen because of the depth of research already available and their application to MEMS whereas other optimization algorithms such as particle swarm optimization, seeker optimization, and tabu search, are still relatively new concepts, especially when applied to MEMS design optimization. Simulated annealing, also a popular optimization algorithm with a vast amount of available research, is not as well suited to MEMS design because it does not evolve a population of solutions that explore available compromises, concentrating on only one "optimal" solution. Simulated annealing is also very dependant on the randomly chosen starting design, whereas GAs should perform equally well regardless of the fitness of the initial population. Topology optimization, while interesting, is very well suited to the design of compliant mechanisms and has readily been applied to the design of compliant MEMS. However, it was decided that topology optimization is not suitable for the optimization of the electric field sensor in its entirety since all aspects of the design can not be easily represented in a single stiffness matrix.

Finally, it is the objective of this thesis to develop an anodic bonding process, a necessary tool in the development of an in-house MicraGEM fabrication. A functioning anodic bonding apparatus is designed and presented.

1.7 Document Layout

This thesis is organized into five main chapters. Chapter two provides an overview of genetic algorithms and how they have been applied by others to design of microelectromechanical systems. Chapter three goes on to discuss the specific genetic algorithm developed for this thesis and its application to the optimization of the electrothermal

actuated field mill design introduced in section 1.4. Chapter four provides a detailed analysis of the finite element models used to simulate the operation of the electric field mill designs. Chapter five outlines the PolyMUMPS and MicraGEM fabrication process' used to fabricate these sensors and discusses some of the problems associated with the resulting devices. Chapter six describes the design and operation of an anodic bonding apparatus for use with an in-house fabrication process based on the MicraGEM designs.

Chapter 2

Genetic Algorithms and MEMS

2.1 Introduction

The design process for microelectromechanical systems (MEMS) has traditionally been based on simplified dynamic equations that describe the operation of the device being designed. These dynamic equations are solved and a resulting device is designed and built. More recently, finite element modelling and simulation tools have become available that allow the designer to simulate complex devices and analyze aspects such as stresses, dynamic and transient response, thus allowing the designer to "virtually" test their designs before fabrication.

This chapter demonstrates the use of genetic algorithms (GAs) to further automate the design process. Figure 2.1 shows an overview of a genetic algorithm applied to the optimization of a MEM device. After the initial population of solutions has been generated, each of the solutions is evaluated; in this case using FE analysis. Once evaluated, the resulting objectives are used to select a subset of the population to continue to the next generation. The selected solutions are then "mated" using crossover and mutation. The procedure repeats until one or more suitable solutions are present in the population or the maximum number of iterations has elapsed.

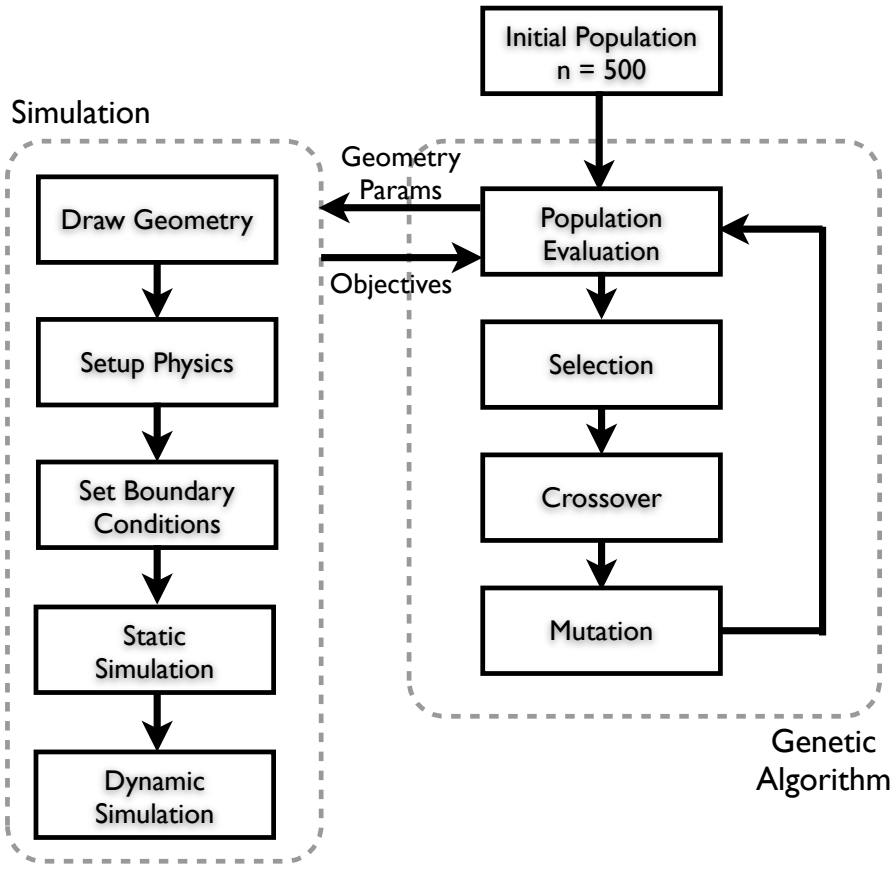


Figure 2.1: *Genetic Algorithm Flowchart*

The simplest GAs will evaluate only one objective, and can be used to minimize or maximize that objective by ranking the population according to the objective. Single objective GAs have previously been applied to MEMS in the optimal design of an RF switch [33] from known equations to maximize resonant frequency. However, often the MEMS designer must choose between multiple conflicting objectives in order to reach a compromise that adequately meets all objectives. This can be incorporated in multiple ways; the simplest of which is by using a weighted sum of the objectives to form one objective function, effectively converting the multi-objective problem into a simpler single objective problem. A weighted sum GA was used in [34] to optimize the design of a MEM switch with the objectives of minimizing switching time and capacitance. The weighted sum method works well with one or two objectives, but can

become more difficult with more objectives. Additionally, it requires more profound knowledge of the problem and can be difficult to choose weighting coefficients when objectives have highly differing ranges. For example, a shear stress objective could be in the range of $10^6 Pa$ to $10^9 Pa$ while temperature might only be in the range $273K$ to $850K$, requiring a large scaling factor to bring the shear stress down into a comparable range as the temperature. This can become even more complex when considering that maybe only a small portion of a range is desirable. For example, a resonant frequency range might be from $1kHz$ to $100kHz$ but only the range from $1kHz$ to $4kHz$ might be desirable. As more objectives are added, it quickly becomes apparent that a lot of consideration needs to be given to the scaling and mapping of objectives in order to achieve optimal results.

Multi-objective GAs have previously been used to synthesize and optimize the design of MEMS. For example in [7] and [35] multi-objective GAs were used to synthesize and design a MEM resonator in which the supporting springs are comprised of multiple beam elements of varying thickness and angle. Because of the free-form nature of the spring design, a human expert was required to intervene and rank a portion of each population so that the resulting devices are manufacturable, and to avoid problems like near-misses where spring elements may collide when in operation. Other multi-objective algorithms use the concept of Pareto dominance [36] in which a solution is said to dominate another solution if it is superior in all objectives. This concept allows us to compare apples to apples in the sense that the algorithm never needs to combine or compare different objectives eliminating the need for scaling factors.

2.2 Selection

Selection is the main source of drive for a GA. It is the method by which solutions are compared and culled, causing the overall population to improve. This section will outline several basic selection methods that are commonly used. The methods discussed here include fitness proportionate selection, truncation selection and tournament selection. All of these methods require a "fitness" value f to be assigned to each solution. This fitness value may come directly from evaluating a complex mathematical equation, such as a dynamic equation that describes the operation of a MEM device. Typically, this evaluation is the most computationally intensive aspect of a GA.

The fitness function f must be a fair representation of the desirable trait(s) to be optimized and is crucial to the success of a genetic algorithm that uses these selection methods. Ideally, the fitness function will be smooth with few local peaks. However, due to the nature of GAs, it is not usually possible to know the exact form or distribution of the fitness function over the solution space before running the algorithm.

2.2.1 Fitness Proportionate Selection

Fitness proportionate selection (aka roulette wheel selection [37]) is one of the earlier proposed methods of selection. In this method, each genome in the population receives a probability p_s of being selected that is proportional to the fitness of the solution relative to the sum of fitness for the entire population (see Equation 2.1).

$$p_s(j) = \frac{f_j}{\sum f} \tag{2.1}$$

This can be implemented by sorting the population according to fitness and then generating a uniformly distributed random number in the range $[0, 1]$. The selected

solution is then the first genome in the sorted set whose probability p_s, summed with all previous probabilities in the sorted set, is greater than the randomly generated number. This is analogous to spinning a roulette wheel where each solution receives a slice of the wheel proportionate to their fitness. This procedure is then repeated until the number of desired solutions has been selected.

An advantage of this method of selection is that every genome has a finite probability of making it to the next generation. This is important since it preserves genetic diversity. A solution that is "poor" in one generation may still have advantageous genetic traits that are dominated by other "poor" traits, thus it has a chance to survive to the next generation where it may pass on its advantageous traits. This maintenance of genetic diversity comes at the cost of selection pressure; "poor" solutions that are selected for their potential to improve the population later reduce the number of "good" solutions that are selected now.

2.2.2 Truncation Selection

Truncation selection is another simple method for selecting solutions. Much like fitness proportionate selection, the genomes are sorted according to their fitness f. Next, the truncation method simply takes the "best" n solutions from the set. A drawback of this type of selection is that "poor" solutions below the threshold will never proceed to the next generation, thus reducing genetic diversity.

2.2.3 Tournament Selection

Tournament selection is a compromise between fitness proportionate and truncation selection. Instead of ranking a solution compared to the entire population, a smaller sub-population is selected at random. The sub-population is then ranked and either truncation or fitness proportionate selection can be used to choose the "winner" of the tournament. This type of selection can be tuned by changing the size of the

sub-population to strike a balance between selection pressure and genetic diversity. A larger sub-population will increase selection pressure while decreasing the size will increase genetic diversity.

2.2.4 Weighted Sum Multi-Objective Selection

The selection methods discussed previously operate well for problems with a single objective, because the objective can be easily mapped to the fitness function linearly or using a quadratic, for example. However, when multiple objectives are required, the fitness function necessarily becomes more complex. The simplest way to accommodate multiple objectives using the previously discussed selection methods is to use a weighted sum approach. Each objective is assigned a weight and then all objectives are summed into a single fitness function. This may be simple with only a few objectives but can become complicated quickly as objectives have different range and may be prioritized differently.

For example, in the design of a thermal actuator the most important objective might be displacement while other objectives might include low temperature, and high mechanical resonant frequency. First, each objective must be normalized. This can be done by estimating the maximum and minimum values of the objective. If the range is unknown, a simple way to estimate is to evaluate the first randomly generated population and then take a look at the distribution of each objective; it is important to note, however, that the first random population is likely to be poor and the ranges will not reflect the range of the final population. Next a weight is assigned to each objective in order of priority. Finally, the fitness can be calculated as a function of each objective. In the thermal actuator example, let's assume the displacement falls into a range of $0 - 3\,\mu\text{m}$, the temperature may fall in the range $273 - 800\,\text{K}$ and the resonant frequency may be in the range $1 - 10\,\text{kHz}$. The weighting coefficients will be $w_{disp} = 0.6$ for displacement, $w_t = 0.2$ for temperature, and $w_f = 0.2$ for

resonant frequency. This gives highest priority to displacement and equal priority to temperature and frequency.

$$f(i) = w_{disp} \times \frac{(O_{disp}(i))}{3} - w_t \times \frac{(O_t(i) - 273)}{527} + w_f \times \frac{(O_f(i) - 1)}{9} \tag{2.2}$$

In Equation 2.2 above, $O_{disp}(i)$ is the displacement objective (μm), $O_t(i)$ is the temperature objective (k), and $O_f(i)$ is the frequency objective (kHz). Note, that the weights and resulting fitness are unitless and so the actual scale of the fitness does not matter since it maps the objectives monotonically.

Table 2.1 shows several fictional actuator designs with objectives mapped using the above fitness function. Note, that design three has the best displacement, and so it has the highest fitness. Unexpectedly, design four has the lowest displacement but has the second highest fitness due to its desirable temperature and resonant frequency. This illustrates how crucial the weighting coefficients are; in this case, the frequency and temperature weights should likely be reduced. It may take several attempts at running the genetic algorithm before suitable coefficients are found and quickly becomes difficult to manage once there are more than a few objectives in play.

Table 2.1: *Fictional Actuator Design Objectives*

Design #	Displacement (μm)	Temperature (K)	Resonant Frequency (kHz)	Fitness
1	1.03	425	1.523	0.1599
2	1.45	358	3.245	0.3076
3	2.54	589	1.824	0.4064
4	0.87	298	8.235	0.3252

Another issue with the above example that may not be immediately apparent is the fact that the resonant frequency distribution is heavily skewed to the higher frequencies. Although high frequencies are desirable, it is usually at the expense of displacement since a high resonant frequency means that the structure is stiffer and more difficult to move. In practice, when a weighted sum genetic algorithm was applied to the design of the electric field mill of this thesis, it was found that the

resonant frequency objective would dominate the other objectives even with a very low weight. To overcome this problem, the resonant frequency objective can be mapped to a different function. In this case for example, an inverted parabola (Equation 2.3) centred at $2kHz$ could be used. $2kHz$ is chosen as a reasonable compromise between frequency and stiffness. With this mapping, the GA will optimize the design towards one which is near 2kHz.

$$O_{mapped} = -(\frac{O_{freq}}{1000} - 2)^2 + 100; \tag{2.3}$$

The weighted sum approach was applied to the geometrical optimization of the electric field mill designed in [5]. The algorithm was set to optimize 25 geometrical parameters with five objectives including minimum actuator temperature, minimum shutter temperature, maximum shutter displacement, maximum XY shear stress, and maximum resonant frequency. Figure 2.2 shows the results of running the weighted sum GA for 15 generations. Each of the three plots shows the normalized and weighted objectives for each of the 100 genomes in the population sorted by their combined fitness value. In the first generation, there is a lot of variation in each of the objectives. As each generation evolves the solutions towards the "optimal" as defined by the fitness function, it is observed that the shutter displacement objective begins to dominate. In generation 10, the temperature and resonant frequency objectives have very little effect on the fitness value. To try and increase the sensitivity to temperature, the weights for the temperature objectives were increased at generation 13, but it seemed to have little effect. In generation 15, you can see that each of the objectives has converged to a fairly stable value. The variance of the design parameters of generation 15 has also been reduced significantly and the resulting sensor geometries are all very similar.

If the designer were not happy with the proposed similar solutions of the last generation, their only recourse with a weighted sum algorithm would be to modify

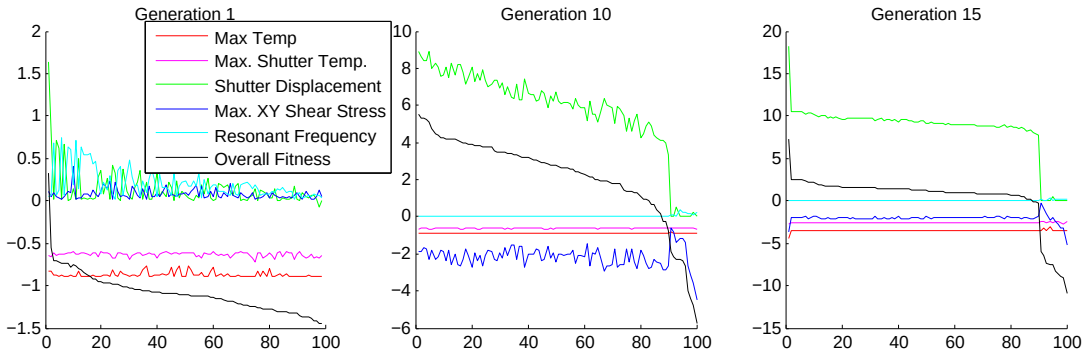


Figure 2.2: *Normalized Objectives and Combined Fitness from Generations 1, 10, and 15 of Weighted Sum GA.*

the weights and run the algorithm again. This is less than ideal since it is so costly to re-run all of the simulations. Other methods of selection can be used to alleviate the need for choosing a fitness function that will cause convergence towards one "optimal" solution. One such method is niched Pareto tournament selection, discussed in the next section.

2.2.5 Niched Pareto Tournament Selection

Unlike the weighted sum selection method discussed above, the niched Pareto tournament selection method utilizes the concept of Pareto dominance to compare genomes [38, 36]. In the niched Pareto tournament selection method, a genome is said to "dominate" another if it is superior in all objectives. This alleviates the need to combine and compare different objectives and requires no scaling factors, mapping functions, or weighting coefficients. Niching is a method used to prevent the population from converging to one genome; this works by giving a higher probability of selecting a genome if it has fewer "neighbours" in objective space.

This selection method operates by randomly selecting from the current population two solutions as well as a random tournament sub-population. Each solution in the tournament sub-population is then compared in terms of Pareto dominance to the two selected solutions. The tournament sub-population size is chosen to be large enough to

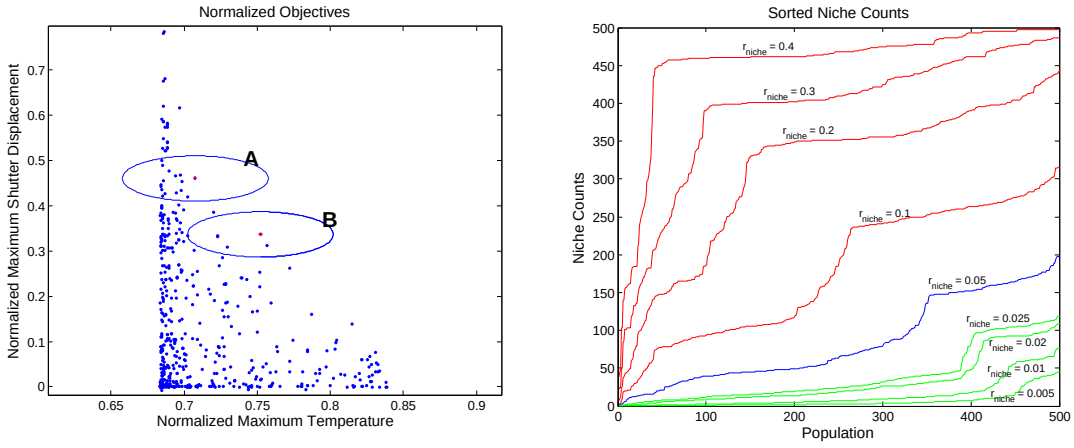
provide significant selection pressure to drive the solution set quickly towards a Pareto front, but not too large or it may prevent many lesser solutions from maintaining their genetic material in the new generation. If any solution in the tournament sub-population dominates one of the two selected solutions but not the other, then the non-dominated solution is selected. If both solutions are dominated or not dominated by the tournament sub-population then they are considered to be equal and a niching strategy is employed.

Niching is used to preserve the distribution of solutions preventing the population from converging to one solution. This is important so that the final output from the algorithm is a set of *different* non-dominated solutions that adequately represent the available compromises between objectives allowing the designer to choose the solution(s) that best meet the objectives. The niching strategy used in this thesis counts the number of solutions within a specified niche radius in objective space. This is done by counting all solutions in the current population who meet the criteria in Equation 2.4 for both selected solutions. The solution with a lower count is then selected as the more unique solution.

$$\sqrt{\sum_{i} (O_{a,i} - O_{b,i})^2} < r_{niche} \tag{2.4}$$

In Equation 2.4, $O_{a,i}$ is the normalized objective i of solution a; $O_{b,i}$ is the normalized objective i of an arbitrary solution b and r_{niche} is a constant defined for the algorithm. The objectives are normalized to a range of zero to one with higher values being superior. This normalization may be based on an expected range for the objective or through a tailored mapping function. The value of r_{niche} is chosen to be just large enough so that the sorted niche counts is an increasing function with few flat spots. If the niche radius is too small then many solutions will not have any neighbours and they cannot be compared by niche count. Similarly, if the niche radius is too large then there will be many solutions with very large niche counts and

it may no longer adequately represent a solutions uniqueness in the set. The initial niche count distribution at generation zero may not be adequate in later generations and so the niche radius should be reconsidered throughout the genetic algorithm.



(a) (b)

Figure 2.3: *Normalized objectives with niching circles plotted (a) and sorted niche counts for various niche radii (b) for a population size of 500.*

Figure 2.3 shows an example of niching taken from generation 20 of the niched Pareto genetic algorithm used in Chapter 3. The left figure shows just two objectives; maximum temperature and shutter displacement that have been normalized and plotted. Each of the two ellipses plotted shows a circle of radius 0.05 around two arbitrarily selected members of the population (note that the normalized maximum temperature axis has been stretched to improve readability). If these two points were tied during niched Pareto tournament selection, then the niche counts would be the number of other solutions located within the plotted circles A and B; 16 and 6 respectively. In order to preserve the uniqueness of the population, the solution with the lower niche count of 6 would be selected. The rightmost plot of Figure 2.3 shows the niche count distributions of the same population for several niche radii. For large niche radii (coloured red) there are many very large counts with a very low slope; similarly, for very small niche radii (coloured green) there are many low niche counts also with a shallow slope. It is desirable to choose a compromise between the two

extremes and so the niche radius 0.05 was selected because it has a steeper slope with only a very small flat ridge. A niche radius of 0.1 may have also been suitable.

Unlike the selection methods discussed earlier, niched Pareto tournament selection causes the population to converge towards the Pareto frontier [36]. The Pareto frontier represents the solutions in objective space that are non-dominated by any other solution and provides a small set of solutions to consider based on the available trade-offs between objectives. Generally, the niche radius should be re-evaluated as the population converges towards the Pareto frontier.

Niched Pareto tournament selection is repeated until approximately half of the next generation have been chosen from the current population. These solutions are the basis for crossover where each solution is "bred" with another solution and two children are created that inherit some properties of both parents.

2.3 Crossover

Once a number of solutions have been selected from the current generation, crossover is applied to combine the selected solutions into new genomes. This serves to propagate and combine beneficial traits from solutions. A commonly used simple single point crossover method starts by randomly choosing two parent solutions from the selected sub-population. From these parents, two child solutions are created. The first child is initialized with the properties $[1, n]$ from the first parent and $(n, 25]$ from the second parent, where n is a randomly generated crossover point. Similarly, the second child is initialized with the first n properties from the second parent and the remaining properties from the first. This is illustrated in Figure 2.4 in which two parent solutions are combined to create two children. Single point crossover can be extended to multi-point crossover by adding additional crossover points. Elitism can also be applied during crossover to preserve fit solutions. This can be implemented

by simply leaving the parents in with the children, forcing the children to compete with their parents for survival in the next generation.

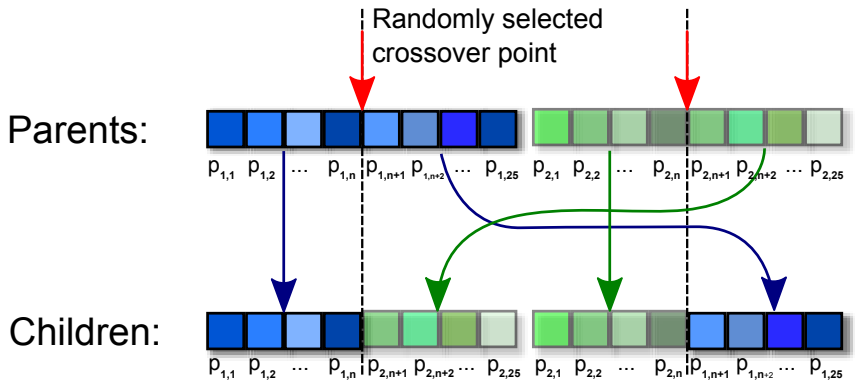


Figure 2.4: *A simplified illustration of crossover. The children are comprised of parameters from both parents.*

2.4 Mutation

The final step before evaluating the new population is to perform mutation. Mutation is the main source of new genetic material. Mutation is a very simple process when the genome is encoded as a binary string; each bit has a finite probability of being "flipped".

However, the process is a bit more complex when dealing with a genome whose properties are encoded as real values. Several methods exist that can be applied to real coded genetic algorithms; random mutation, non-uniform mutation, and Gaussian mutation, to name a few common mutation operators. Random mutation simply replaces the real valued gene with a new value that is within its allowable range [39]. An issue with this type of mutation is that as the algorithm progresses, the mutation step size remains fixed, making it difficult for the algorithm to hone-in on a global optimum. Non-uniform mutation, however, deterministically controls the step size by considering the number of generations already computed and exponentially reducing

the effect of mutation as the algorithm progresses [40]. For this reason, non-uniform mutation has become one of the most widely used mutation operators for real-valued genetic algorithms [40]. Gaussian mutation is another popular operator in which Gaussian noise is added to the genes value with the standard deviation determined by the range of the property and the desired step-size [41].

The mutation operator employed in this thesis is similar to the uniform and Gaussian mutation operators and attempts to emulate the same process as bit-wise mutation operates by assigning to each real valued property of each genome a probability M_o of being mutated. For each property, a uniformly distributed random number is generated; if this number is less than the probability M_o then that property is mutated. To determine how much to mutate the property, a mutation factor M_f is used and another uniformly distributed random number R, in the range $[-0.5, 0.5]$ is generated. These parameters are kept separate to simplify the coding of the mutation operator and to isolate the random R from the tunable mutation factor M_f. Equation 2.5 is then used to update the value of the property, and it is repeatedly executed in a loop until a value within the allowed property range is achieved. In Equation 2.5, $o_{a,i}$ is the non-normalized property i of genome a and $o_{range,i}$ is the allowed range for property i.

$$Value = o_{a,i} + R * o_{range,i} * M_f \tag{2.5}$$

The values of M_o and M_f must be tuned for each application of the genetic algorithm. If the mutation occurs too often or changes a solution dramatically, the solutions are more likely to "jitter" around the optimal solutions. Conversely, if there is not enough mutation, then it may take longer for the algorithm to converge and it is more likely to become trapped in sub-optimal peaks or valleys. Non-uniform mutation and other self-adapting mutation operators may be a good choice for future MEMS optimization genetic algorithms since they do not require re-evaluation when

running the algorithm.

2.5 Multi-Objective Niched Pareto Genetic Algorithm Applied to a Simple Resonator

As a test, a multi-objective niched Pareto genetic algorithm was applied to the optimization of a simple resonator design. The purpose of this test was to compare the results of the algorithm to the multi-objective interactive evolutionary computation (IEC) genetic algorithm developed by [7]. The optimized resonator consists of a square shuttle mass suspended by four springs; much like the shutter of the electric field mill design [5]. However, unlike the springs of the field mill design which are constrained to a box geometry, the springs of the resonator design are built from several adjoining beams. In order to simplify the design, the resonator has no actuation mechanism, however, electrostatic combs could easily be added for actuation and frequency sensing without significantly impacting the results of the simulation. Each beam element of the spring is allowed to have an arbitrary width, length, and angle from the previous element (or shuttle mass if it is the first element) within an allowable range. The unconnected edge of the last spring beam element is constrained and considered to be anchored to the substrate. The model used during simulation is one half of the resonator structure, symmetric about the x axis at the centre of the shuttle mass. Table 2.2 outlines the allowed geometrical ranges for the resonator. The allowed geometric ranges of Table 2.2 were the same as those of [7].

Table 2.2: *Geometric Ranges for Resonator Design*

Parameter	Allowed Range
Shuttle Mass Width	$200 - 400\,\mu m$
Shuttle Mass Length	$200 - 400\,\mu m$
Spring Beam Width	$2 - 10\,\mu m$
Spring Beam Length	$10 - 100\,\mu m$
Spring Beam Angle	$-90 - 90°$
Number of Spring Beams	1-7

The algorithm evaluates the objectives of resonant frequency and geometrical area of each design using a dynamic finite element simulation using COMSOL multiphysics. The target resonant frequency was 100 kHz. The niched Pareto genetic algorithm is the same as the one described in chapter 3, using niched Pareto tournament selection, single point crossover, elitism, and real-value mutation. The specific parameters used for this instance of the genetic algorithm are listed in Table 2.3.

Table 2.3: *Initial GA Parameters for Resonator Design*

Parameter	Value
Population Size	500
Tournament Population Size (t_{dom})	100
Selection Size	100
Niche Radius (r_{niche})	0.05
Mutation Odds (M_o)	0.05
Mutation Factor (M_f)	0.1

Figure 2.5 shows a sample of 10 sensor designs from the 39th generation of the genetic algorithm whose resonant frequency is within 0.5% of the target frequency, 100 kHz. The algorithm was allowed to run for several hours, and then stopped and the results evaluated. As such, it is likely that the algorithm could have been stopped sooner while still meeting the design objectives. An interesting aspect to note about the designs in Figure 2.5 is that many of them show springs that have been folded back towards the centre of the shuttle mass. Once mirrored about the shuttle mass, the springs would conflict with each other. In order to solve this problem, the springs should be flipped to the opposite edge of the design. The reason that the algorithm

chose to do this is because it is set to minimize area, and by folding the springs back towards the mass, the area is indeed minimized. This could be solved at the algorithm level by either modelling the complete resonator, or adding constraints to prevent the springs from conflicting with their mirrored counterparts. The design of Figure 2.5(d) is also quite promising, since it features springs that fold back on themselves, resulting in the least area of the designs presented in Figure 2.5.

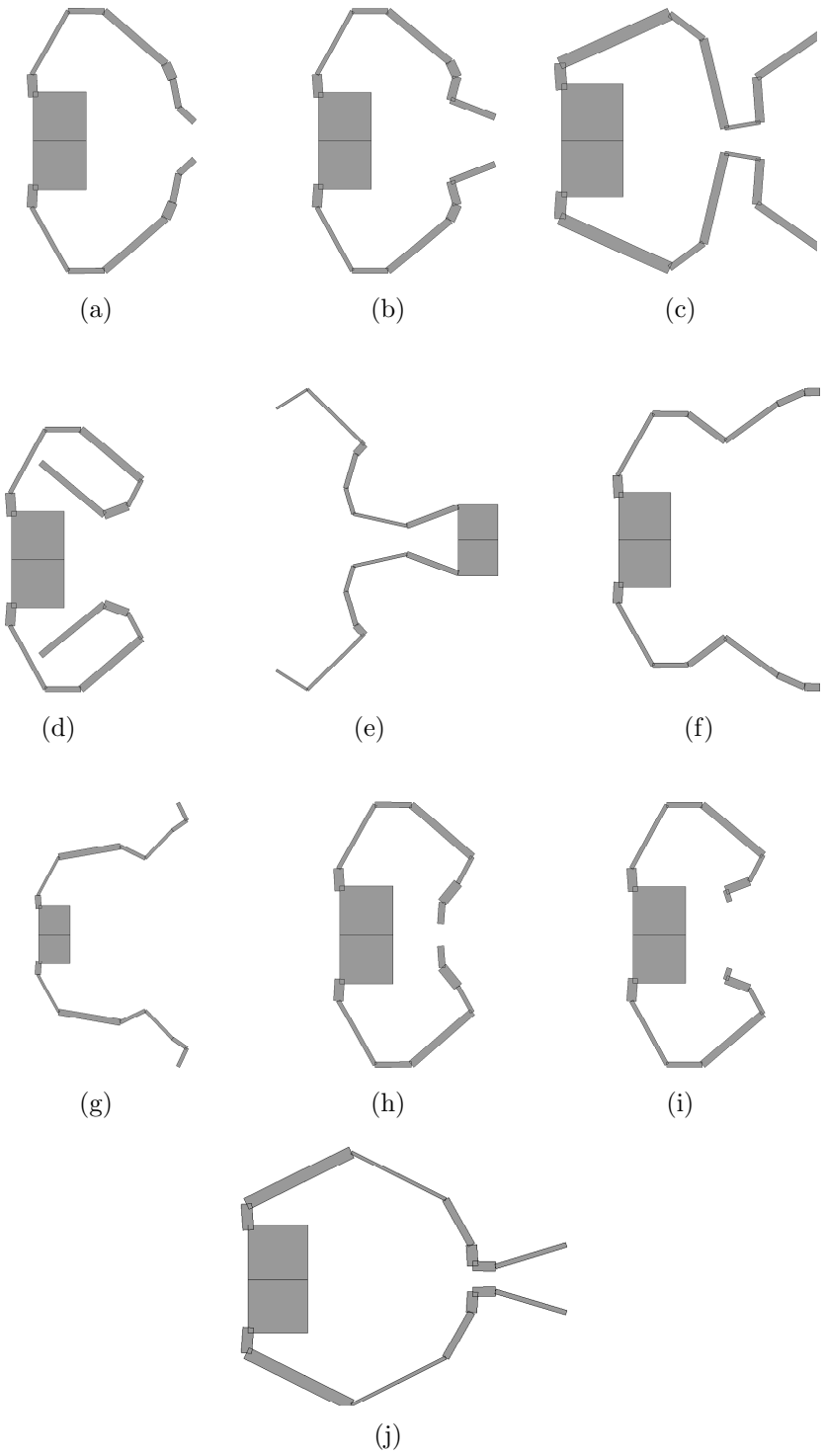
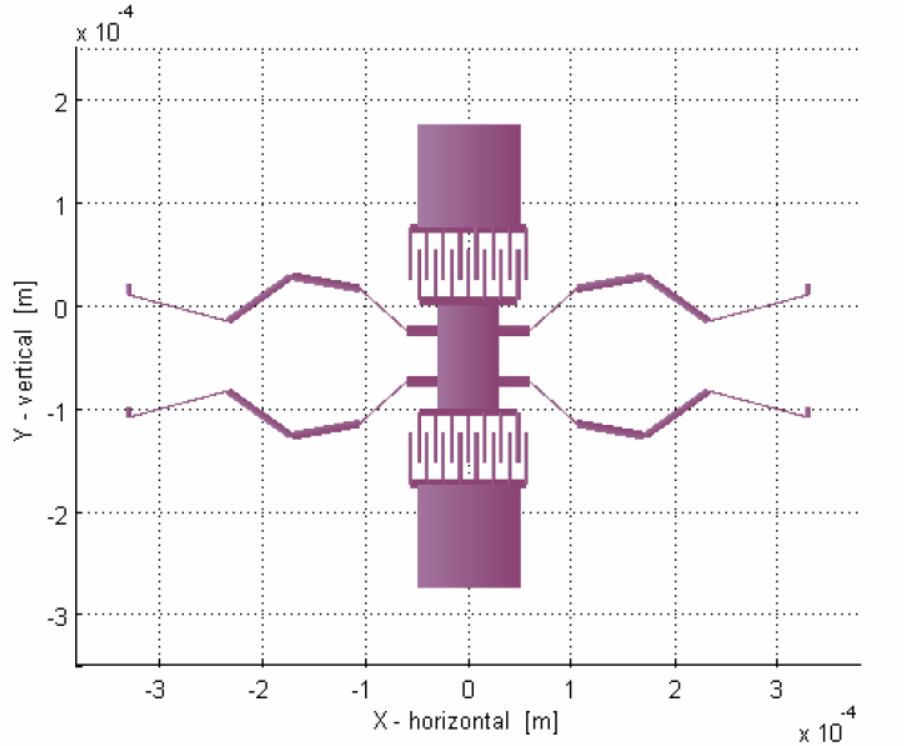


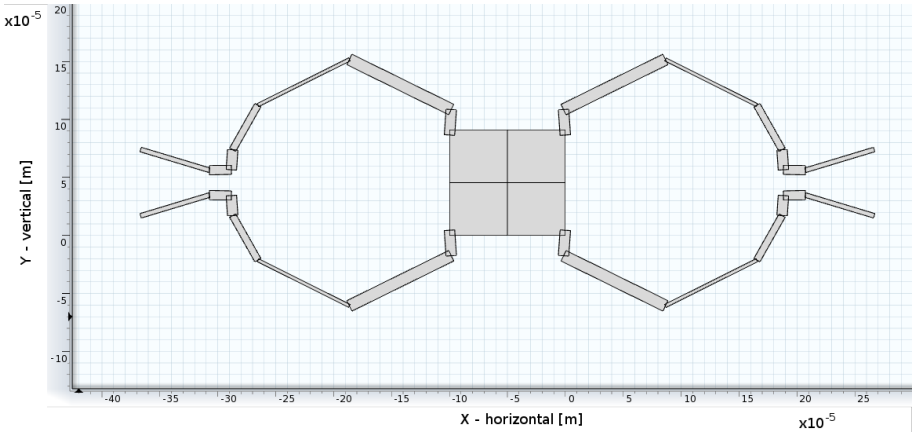
Figure 2.5: *Several resonator designs within 0.5% of the target frequency from generation 39. The resonators are not plotted to the same scale, but are presented in order to compare their spring geometry.*

Figure 2.6(a) shows an example of a sensor that was designed using the IEC genetic algorithm [7]. Figure 2.6(b) shows the design of Figure 2.5(j) that has been mirrored with flipped springs for comparison. The similarities between the two designs generated by different algorithms is quite remarkable. Their algorithm periodically requires the user to rank each individual of the population according to the user's expert opinion. For example, the sensor in Figure 2.6(a) was chosen partly because its spring beams do not contain sharp corners that could cause premature failure due to stress concentrations [7]. Experts are also used to judge other criteria such as springs that are too close together and may collide during operation, and the overall manufacturability of a design. However, the algorithm presented here is able to autonomously generate many different solutions, as pictured in Figure 2.5 that meet the objectives, from which a user could choose a solution that meets their desired criteria. Specific influences such as close spring loops or high stress concentrations could easily be added as additional objectives to the genetic algorithm allowing for a larger set of suitable designs at the completion of the algorithm.

Another variation on the IEC presented by Kamalian et al. has been developed to reduce human fatigue while evolving a design [42]. Instead of ranking each design in a generation, as in [7], the user instead has a more supervisory role and is only required to promote or demote a few of the designs in each generation [42]. Interactive evolutionary computation could be a useful tool for MEMS design when it is difficult to quantify the desired objectives of a design.



(a)



(b)

Figure 2.6: *Niched Pareto GA resonator design compared to IEC GA resonator design [7]. (a) Shows a high scoring design generated with the IEC GA. (b) A comparable design optimized automatically by the niched Pareto GA.*

Chapter 3

Genetic Algorithm Optimization of the Electric Field Sensor

3.1 Introduction

Considering the previous background on genetic algorithms, a niched Pareto genetic algorithm was designed and implemented to optimize the geometry of the electric field mill designed in [5]. The original sensor design was insufficient to operate outside of resonance and therefore required optimization. This chapter outlines the specific design of the algorithm and presents the results generated by it. Niched Pareto tournament selection was chosen as the selection method as well as the real-valued mutation operator described in section 2.4. The flowchart in Figure 3.1 depicts the specific algorithm used to optimize the electric field mill design.

3.2 Encoding and Geometric Parameters

The geometric parameters selected for optimization are summarized in Table 3.1. The nominal values listed are the dimensions used in the sensor designed and fabricated in [5] and was used as a guide when deciding the parameter ranges allowed

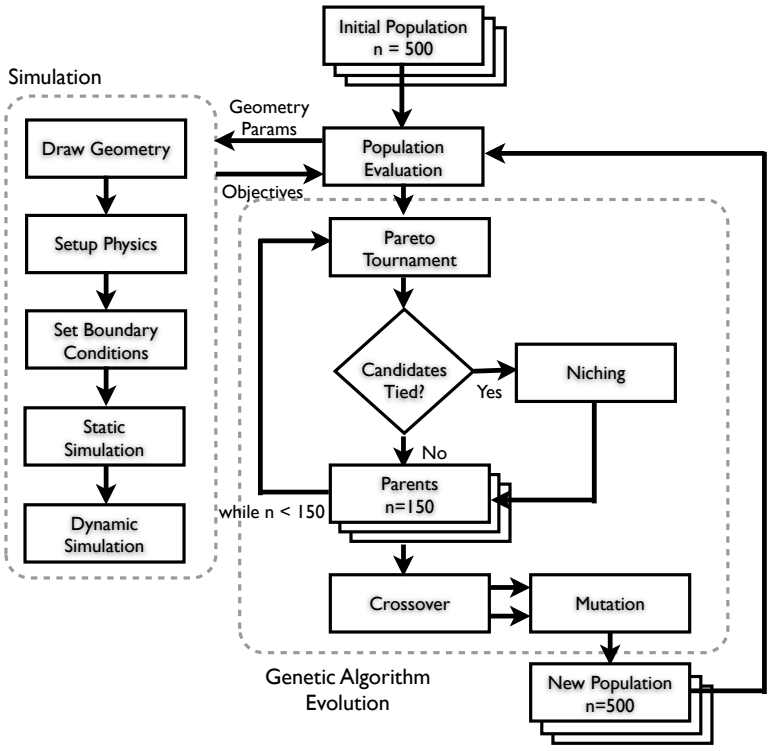


Figure 3.1: *Niched-Pareto genetic algorithm as applied to the optimization of the Electric Field Mill design.*

by the GA. The algorithm may choose any value within the range, to a tolerance of 0.001 μm for length parameters, 0.01° for angle parameters, and 1 for integer parameters. The maximum and minimum values of the parameters were selected to give the genetic algorithm enough range while still ensuring that a device could be manufactured. For example, the minimum allowable width of 3 μm was chosen for all beam elements. Likewise, the maximum values were chosen to limit the maximum aspect ratio of beams as well as the final size of the device. The actuator beam angle was restricted to values above 1 ° to ensure that the beam will expand in the desired direction without buckling. The maximum actuator beam angle was chosen to be 10° because it is significantly larger than the nominal value calculated in [5]. In general, the parameter ranges should be reconsidered if the genetic algorithm appears to be converging towards one of the parameter limits. With all 25 parameters accounted,

this makes for over 10^{106} possible sensor designs, orders of magnitude higher than many estimates of the number of atoms in the observable universe. The parameters in Table 3.1 were encoded as an ordered list of floating point values. This phenotypic [20] representation was chosen as opposed to a genotypic [20] representation to simplify the encoding of each genome while allowing each parameter to be confined to a specified range. The ease of encoding comes at the expense of more complex crossover and mutation operators.

Table 3.1: *Geometric Parameters*

Geometric Property	Nominal	Min.	Max.
Actuator			
No. Beams	5	1	30
Beam Spacing	12 µm	5 µm	20 µm
Beam Length	190 µm	50 µm	300 µm
Beam Width	6 µm	3 µm	6 µm
Beam Angle	4.5°	1°	10°
Shuttle Width	20 µm	5 µm	20 µm
Lever			
Connect. Beam Width	5 µm	3 µm	10 µm
Connect. Beam Length	95 µm	75 µm	200 µm
Short Beam Width	3 µm	3 µm	10 µm
Short Beam Length	20 µm	10 µm	200 µm
Long Beam Width	4 µm	3 µm	20 µm
Long Beam Length	375 µm	100 µm	2000 µm
Connect. Spring Width	10 µm	7 µm	30 µm
Connect. Spring Height	60 µm	15 µm	200 µm
Connect. Part Width	10 µm	1 µm	20 µm
Connect. Part Height	9 µm	3 µm	20 µm
Spring			
No. Elements	2	1	3
Thickness	21 µm	4 µm	10 µm
Total Width	560 µm	200 µm	1400 µm
Hole Width	15 µm	3 µm	50 µm
End Thickness	20 µm	3 µm	50 µm
Connect. Beam Width	20 µm	3 µm	50 µm
Shutter-Spring Beam Length	20 µm	3 µm	50 µm
Spring-Spring Beam Length	15 µm	3 µm	50 µm
Spring-Anchor Beam Length	25 µm	3 µm	50 µm

3.3 Modelling and Simulation

The method for evaluating each iteration was finite element analysis using the COMSOL Multiphysics [43] software with MATLAB [44]. The model used was a 2D symmetric model of the shutter structure, springs, levers, and thermal actuators. The

three physics application modes used were conductive dc media to model the electric potential throughout the structure, joule heating to model the resistive heating of the thermal actuators, and finally the plane strain application mode to couple thermal expansion into the model. The material properties used for the simulations, as reported for the fabrication process in [5], are summarized in Table 4.1. The MATLAB code for the COMSOL model and simulation are provided in Appendix 7.2.

Two simulations were performed for each candidate solution; a static simulation used to evaluate the shutter displacement, temperature and XY shear stress; and a dynamic simulation which is used to evaluate the resonant frequency of the structure. Since the slits in the shutter were designed to be 5 μm wide, only 5 μm of displacement is required. To simplify simulation and comparison of models, the electric potential across the actuators was fixed at 1 volt. However, a solution with more than 5 μm displacement is still preferable since it would allow the voltage to be reduced, thus reducing the maximum temperature and power consumption of the device. In the event that a given geometry is not possible due to overlapping structure or simply does not converge in finite element analysis, then it is removed from the population and either the components that cause the overlap are randomized or the entire sensor is replaced with a new randomly generated geometry. For complete details of the simulations performed, please refer to Chapter 4.

3.4 Genetic Algorithm Parameters

The selection method chosen for the algorithm is niched Pareto tournament selection [36, 38]. This method was chosen for its ability to evolve the population without the need for scaling factors or objective weighting coefficients and also because it causes convergence towards the Pareto set of solutions instead of just one "optimal" solution. The objectives to be optimized were shutter displacement, maximum tem-

perature, maximum shear stress, and resonant frequency.

The initial population size was chosen to be 500, large enough to maintain a fair distribution of solutions throughout the objective space, but small enough so that each generation can be evaluated in a reasonable amount of time. The population was first created at random by uniformly choosing parameter values that fall within the allowable ranges defined in Table 3.1. The tournament population size, t_{dom}, was set to 100 which was found to provide suitable selection pressure while still maintaining genetic diversity.

The niche radius, r_{niche} was chosen to be 0.05 but was re-evaluated at generation 34 and increased to 0.1 in order to maintain a diverse set of solutions along the Pareto frontier. Figure 3.2 shows the sorted niche counts for the population at generation 34. Plotted are the niche counts for an r_{niche} of 0.05 (blue) and 0.1 (red). The larger niche radius exhibits a better slope with fewer flat spots as discussed in section 2.2.5.

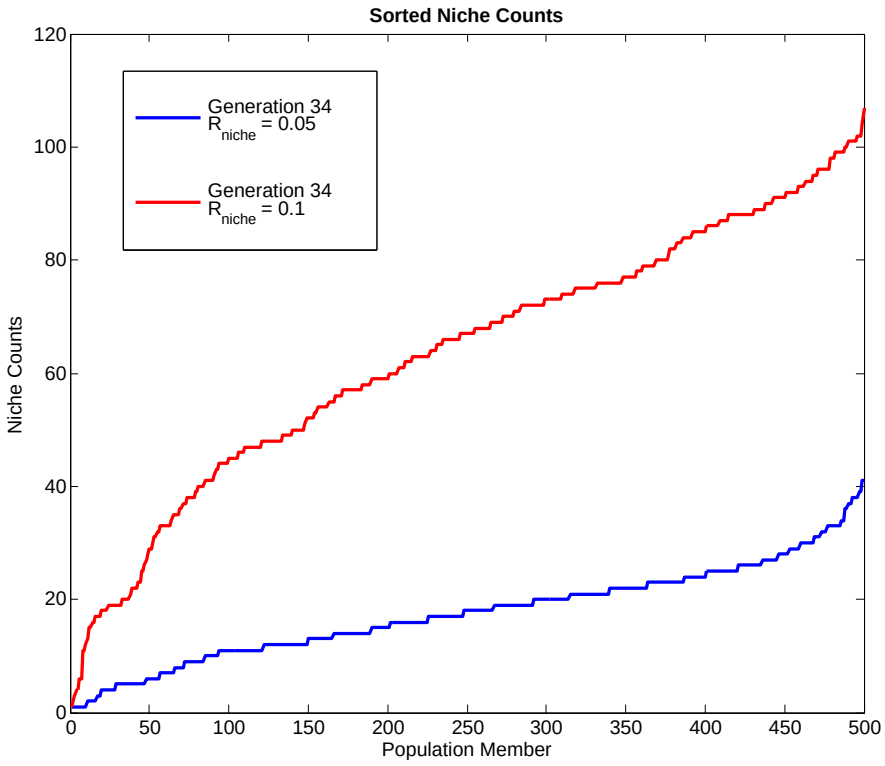


Figure 3.2: *Niche radius evaluation at generation 34.*

The algorithm was configured to select 150 solutions using niched Pareto tournament selection which were added without crossover to the next generation (elitism). The remaining 350 solutions are generated using random single-point crossover using the 150 solutions selected by niched Pareto tournament selection as the pool of potential parents. The mutation odds, M_o, and mutation factor, M_f were 0.3 and 0.2 respectively for the first 20 generations and then was reduced to 0.2 and 0.1 respectively in order to reduce the amount of genetic material being introduced and fine-tune the population for the later generations. Table 3.2 summarizes the initial parameters used in the genetic algorithm.

Table 3.2: *Initial GA Parameters*

Parameter	Value
Population Size	500
Tournament Population Size (t_{dom})	100
Selection Size	150
Niche Radius (r_{niche})	0.05
Mutation Odds (M_o)	0.3
Mutation Factor (M_f)	0.2

3.5 Finishing Criteria

One difficulty with genetic algorithms can be determining when to stop since it is not possible to always know whether your population has reached a global optimum. A number of finishing criteria can be used such as the distribution of niche counts or the convergence of the population towards the desired objectives. Figure 3.3 shows a plot of the maximum displacement and the number of solutions with at least 5 µm of shutter displacement.

The shutter displacement curve begins to level off with little or no improvement past generation 40. Similarly, the number of workable solutions with displacement of at least 5 µm peaks just after generation 40. The algorithm was allowed to run for a

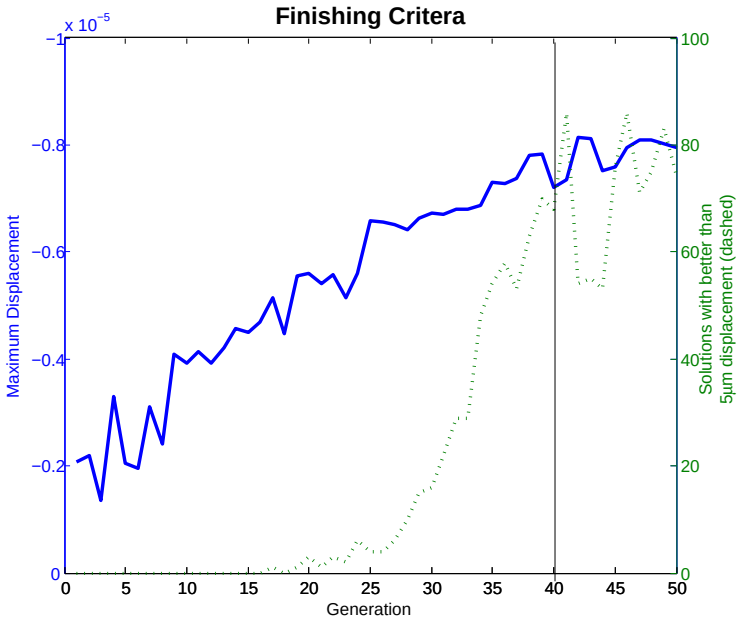


Figure 3.3: *Finishing Criteria*

few more generations to see if it would further improve, but it reached a maximum value of approximately 8 µm shutter displacement and 80 solutions with more than 5 µm shutter displacement. It is possible that the results could be improved slightly by reducing the odds of mutation M_o and mutation factor M_f, however the results were deemed acceptable so the algorithm was stopped. The final set of results taken from the run of the algorithm combines the solutions from generations 40 to 50 by selecting only the non-dominated solutions from the 10 generations. From this Pareto set of solutions, the MEMS designer can select one or more designs.

3.6 Results

Figure 3.4 shows the progression from an initial random population in generation 1 to generation 45. The colours in Figure 3.4 represent the resonant frequency of each solution while the three axes represent maximum temperature, shutter displacement and maximum XY shear stress. As can be seen in the plot for generation 1, the solutions are all fairly poor with high resonant frequencies and little or no displacement

while maximum temperature and shear stress seem to be distributed over a wide range as they depend highly on the parameters of the actuator. For larger plots of each generations' objectives, please refer to appendix 7.2.

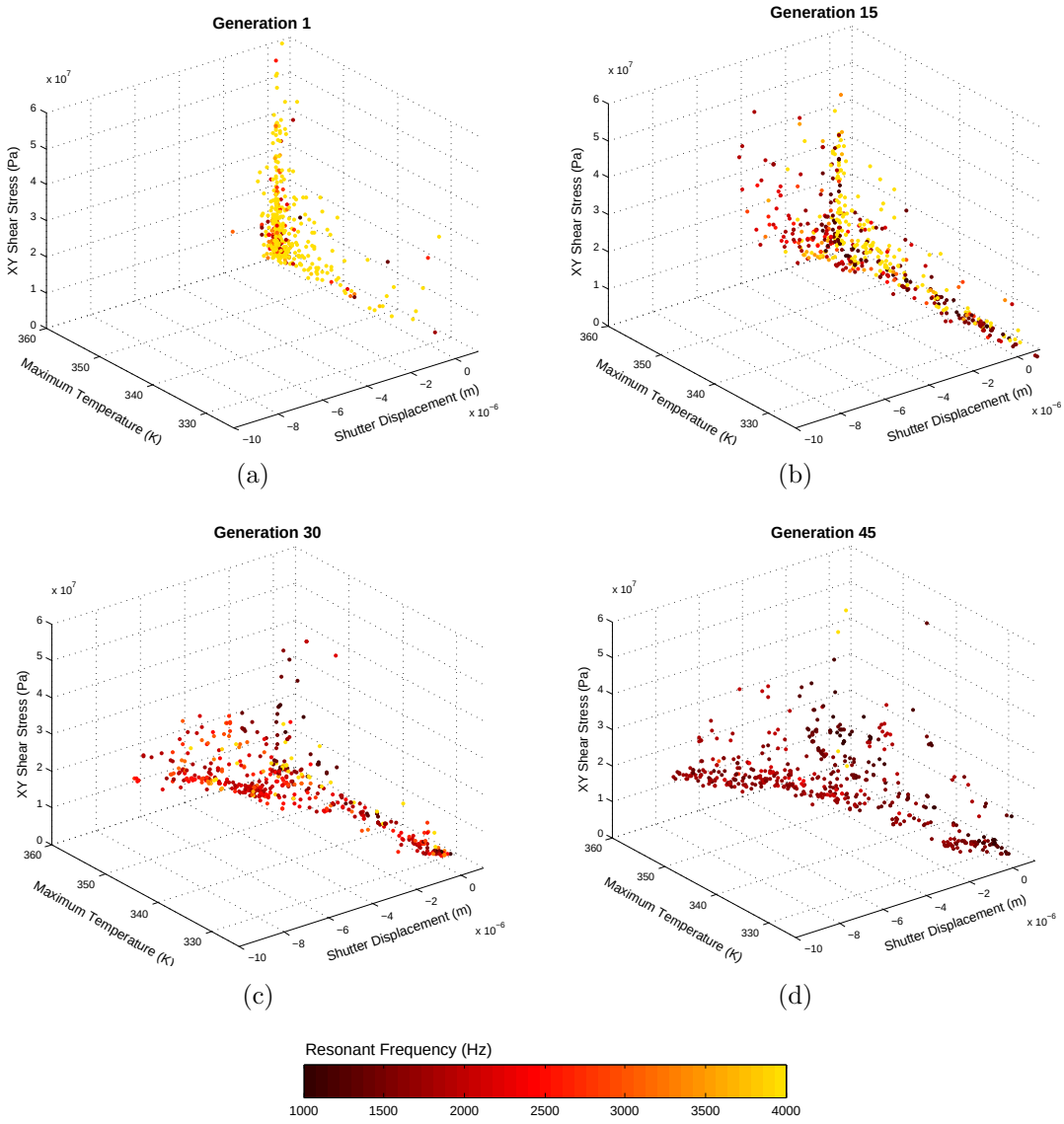


Figure 3.4: *Plots of maximum XY shear stress (Pa) vs. maximum temperature (K) vs. shutter displacement (m) for generations 1 (a), 15 (b), 30 (c) and 45 (d) showing convergence towards a Pareto front.*

By generation 15, Figure 3.4 shows that the population has begun to improve with more shutter displacement and fewer high stress solutions. It is also interesting to note that the solutions are more evenly distributed in the temperature range. However, the

majority of the resonant frequencies are still too high and the shutter displacement is still too low for a practical sensor. Generation 30 shows vast improvements since generation 15 and the emergence of the first few suitable solutions with displacements approaching 5 µm is visible near the far left end of the displacement axis. There are still many solutions clustered with high resonant frequencies and low displacement, but there are fewer than in generation 15. In generation 30, the Pareto front is starting to become visible with the relationship between maximum temperature and shutter displacement.

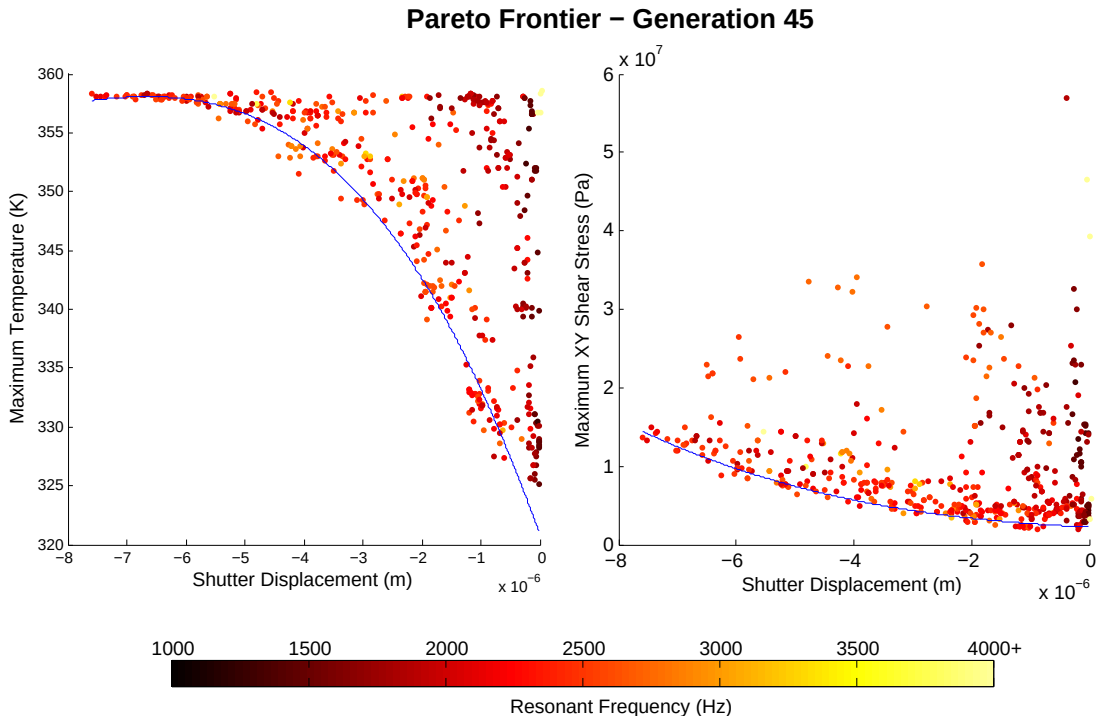


Figure 3.5: *Pareto Front*

The plot of the generation 45 shows clear convergence towards a Pareto front, most clearly seen in the plane of shutter displacement and maximum temperature. Also visible is a relationship between shutter displacement and shear stress. These two relationships are more plainly seen in Figure 3.5 which shows the 2D projections of generation 45 onto the plane of maximum temperature and shutter displacement and the plane of XY shear stress and shutter displacement. The two curves in Figure

3.5 are fit to the Pareto front in each plot which represents the trade-offs between maximum temperature, shutter displacement and maximum XY shear stress. In reality, the Pareto front extends into four dimensions; one for each of the four objectives, but this is difficult to visualize and the two relationships depicted in Figure 3.5 were the most clearly visible.

Unlike a weighted sum GA or a single objective algorithm, this method allows the designer to see the compromises available between conflicting objectives and choose the solution(s) that best meet the design requirements. For example, out of the solutions found, different objectives can be prioritized; a high frequency design could be selected from the Pareto set at the expense of displacement and shear stress, while a design with a large displacement can be chosen to reduce power at the expense of resonant frequency. Similarly, a design with the lowest shear stress could be selected to maximize the operational lifetime of the sensor while requiring a higher voltage to achieve the necessary 5 μm displacement.

Several solutions have been selected and are summarized in Table 3.3, along with designs that were optimized by hand or with a weighted sum GA. To achieve the needed 5 μm displacement the actuator voltage was incrementally increased or decreased for each solution in Table 3.3 and re-simulated until 5 μm displacement was achieved. The first solution was chosen because it had the largest displacement at 1 V drive voltage of the solution set. The second was chosen for its high resonant frequency with a 1 V displacement that is still greater than 5 μm. Similarly, the low shear stress solution was selected as the lowest shear stress solution with a displacement that is still close to the desired objective. Finally, the last solution was the original design from [5]. This design required resonant frequency operation, however here it is being compared in non-resonant operation and so needs a much higher voltage to achieve the needed displacement.

Table 3.3: *Selected Solutions from GA (cases 1-3) that meet the* 5 µm *displacement requirement, along with the design of [5] (case 4).*

Case #	Voltage (V)	Temp. (K)	Shear Stress (MPa)	Freq. (Hz)
1. High Displacement	0.80	334.93	13.91	2897.0
2. High Frequency	0.96	352.60	23.86	3511.1
3. Low Stress	1.1	357.76	5.52	2311.4
4. Design in [5] (non-resonance)	3.2	907.14	352.82	3857.9

3.7 Population Size

Population size is an important factor to consider when designing any genetic algorithm. Choosing an adequate size will depend on the nature of the problem, the size of the search space, and the size of objective space. Choosing a small population will reduce the time it takes to compute a single generation but also reduces the amount of genetic diversity that can be sustained within the population. For the case where niched Pareto tournament selection is used, it also reduces the possible coverage of the Pareto front. Choosing a larger population size has the consequence of increased computational cost in evaluating each generation, however it may require fewer generations for a larger population to reach the same level of optimization. A smaller population size also requires less mutation; as population size decreases it becomes increasingly detrimental to the overall population when a good solution is lost through mutation, whereas in a larger population there would likely be several similar solutions clustered around the same point in objective space and the loss of one or two due to mutation would have less of an impact on the overall population.

To compare the effectiveness of different population sizes, the same genetic algorithm was run again with a lower mutation factor M_f and odds M_o with population sizes of 100, 200, and 300. Table 3.4 below summarizes the parameters used for the various algorithm runs.

Figure 3.6 below shows the results of running the GA with the aforementioned parameters of Table 3.4. Since many parameters must be changed dramatically to

Table 3.4: *GA parameters for various population sizes.*

Population Size	Generations	M_f	M_o	t_{dom}	r_{niche}
100	0 - 31	0.1	0.05	20	0.1
200	0 - 31	0.1	0.05	100	0.1
	32 - 55	0.01	0.025	100	0.1
300	0 - 36	0.1	0.05	100	0.1
	37 - 48	0.01	0.025	100	0.1
500	0 - 20	0.3	0.2	100	0.05
	21 - 33	0.2	0.1	100	0.05
	34 - 50	0.2	0.1	100	0.1

accommodate the change in population size, it is difficult to say for certain which population size is "best" for this problem, but by comparing the rate of convergence on the most important objective, shutter displacement, it can be seen how the different population sizes performed relatively. The figure plots the best shutter displacement of the current generation from each run of the algorithm against the number of simulations run at that point in the algorithm. It is plotted against number of simulation runs instead of generation count since the population sizes vary and the number of simulations is a fair indicator of the amount of computational time taken. Please see section 3.8 for a discussion on computational time.

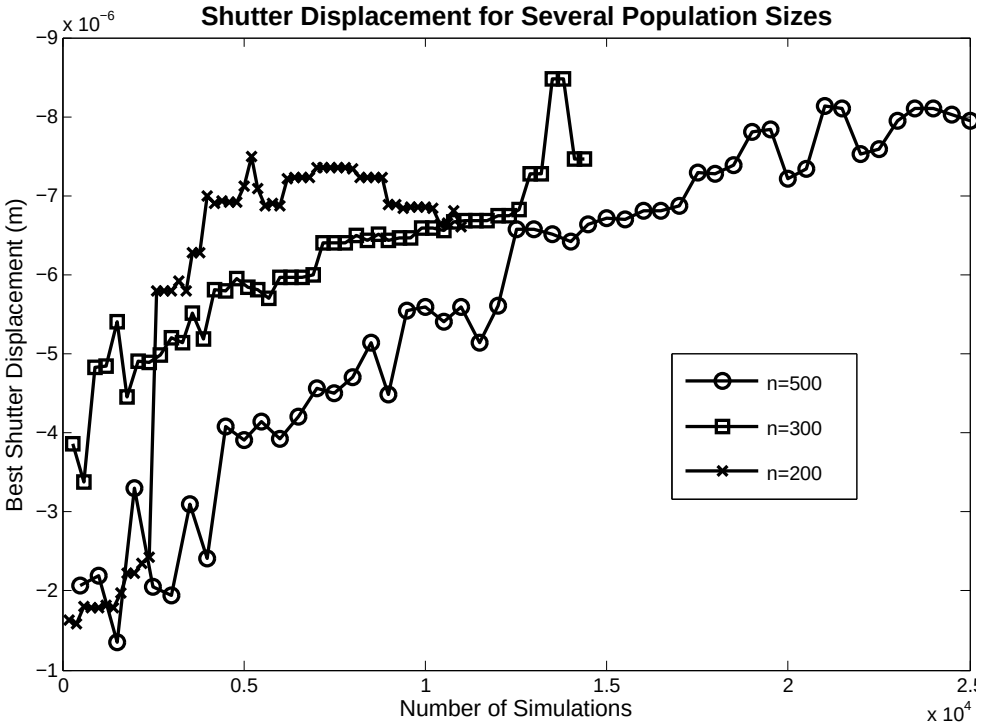


Figure 3.6: *Shutter displacement vs number of simulations for population sizes of n=200, n=300 and n=500.*

According to Figure 3.6, the population size of 300 yields the best rate of convergence for shutter displacement of the three population sizes plotted. Although, the smaller population size of 200 converges faster, it does not reach the same level of optimization as with the larger populations. The population size 100 failed to produce any solutions that meet the minimum requirement of 5 μm shutter displacement.

If the results from the smaller populations are deemed adequate, then it can return a result the fastest. However, this is a very limited view of the problem since there are more objectives and the entire Pareto frontier to examine, not just one objective. Consider Figure 3.7 which shows the final generations' objectives plotted on 3D axes. Although the extreme point of each plot shows similar maximum displacement, the population size of 200 exhibits clumping and it does not have very good coverage. The population size of 300 does quite a bit better but is still nothing like the even coverage seen with the population size of 500.

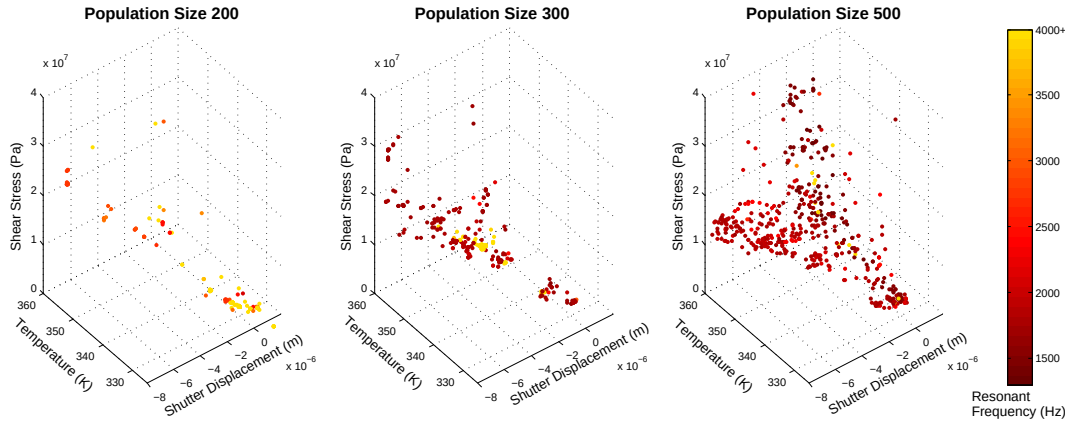


Figure 3.7: *Final generations' objectives for population sizes of n=200, n=300 and n=500.*

3.8 GA Computational Time

As discussed in the previous section on population size, a larger population necessarily requires more computational time to complete. The static and dynamic simulations of a symmetric field mill model take approximately 30 seconds to complete a single iteration. To put this in perspective, it takes approximately 4 hours to evaluate a single generation of population size 500. Testing was carried out to see the effect running the COMSOL simulations with different computer hardware. It was tested on machines with two, four, and eight cores and it was found that there was little improvement gained by allocating more than two cores to the simulations and for the majority of time during a simulation the cores would not be fully utilized. This is because the majority of time spent during each simulation is not spent solving, but rather setting up the geometry and assembling the large matrices to be solved.

However, genetic algorithms lend themselves very well to be parallelized. Instead of running only one simulation at a time, it is possible to run several at a time on the same multi-core machine, or even across multiple machines configured to run Matlab and COMSOL. It was found by experimentation that the computer hardware could easily handle one concurrent simulation for every two cores in the machine. If more

simulations are run, then they begin to take longer than 30 seconds each to complete. Using this method of parallelization, a master node runs the genetic algorithm and then distributes the population amongst multiple machines or cores for evaluation. Distributing a population of 500 sensor designs to four eight-core machines, each running four concurrent simulations, the time to compute each generation is reduced from 4 hours to approximately 20 minutes, a speed-up factor of 12. This makes it possible to run 50 generations in $\sim$ 17 hours.

Chapter 4

Finite Element Analysis Simulation

4.1 Introduction

Finite element analysis is the basis for evaluating the field mill designs in this thesis. Static and dynamic simulations were performed using the COMSOL multiphysics software during the genetic algorithm processing, as described in Chapter 2. These simulations were simplified by using a coarse mesh and symmetry in order to reduce the time to evaluate each design iteration. This chapter discusses the full details of the simulations done during and after the genetic algorithm. Finally, this chapter also discusses experiments performed to measure and validate the thermal and mechanical properties found during simulations.

4.2 Static Simulation

The first simulations performed were 2D static simulations of the complete structure. This simulation couples together the electric currents, heat transfer, and solid mechanics physics modules of COMSOL. In order to validate the simulation, the structure is first modelled using the same geometric parameters as the electric field sensor designed in [5], henceforth referred to as "the original design". The exact

geometric parameters used for the validation model are the nominal geometric parameters listed in Table 3.1. The material properties used for the simulation are provided in Table 4.1. These material properties were the same as those used in the original sensor design simulations and are found in the MicraGEM fabrication process manual [12]. Note, the electric conductivity property is the effective conductivity of a thin layer of gold electroplated on a seed layer of chrome on top of a 10 µm doped single crystal silicon substrate and was empirically determined by [5] shortly after fabrication. It has been observed that the resistivity of the gold increases over time as the chrome seed layer diffuses into the gold, however this effect could be reduced by using a different seed layer with lower diffusivity [45].

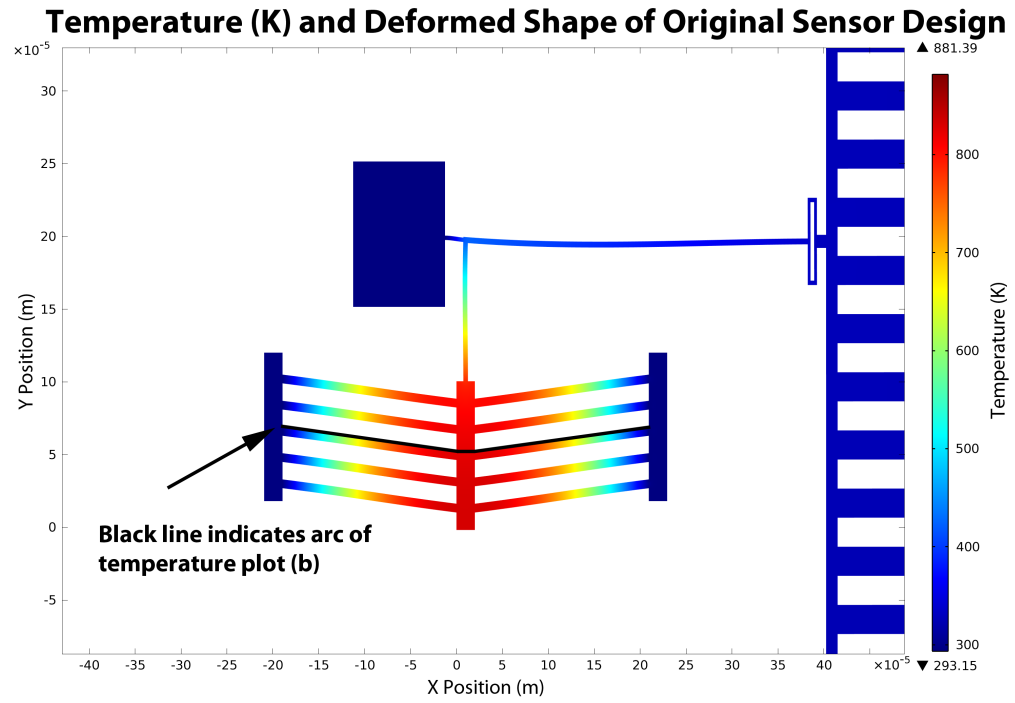
Table 4.1: *Material Properties*

Electric Conductivity (Au on Si)	78431.373 S/m
Electric Conductivity (Si)	2 S/m
Thermal Conductivity (Si)	150 W m^{-1} K^{-1}
Heat Capacity (Si)	385 J kg^{-1} K^{-1}
Density (Si)	2330 kg/m^3
Young's Modulus (Si)	1.295×10^{11} Pa
Poisson's Ratio (Si)	0.22

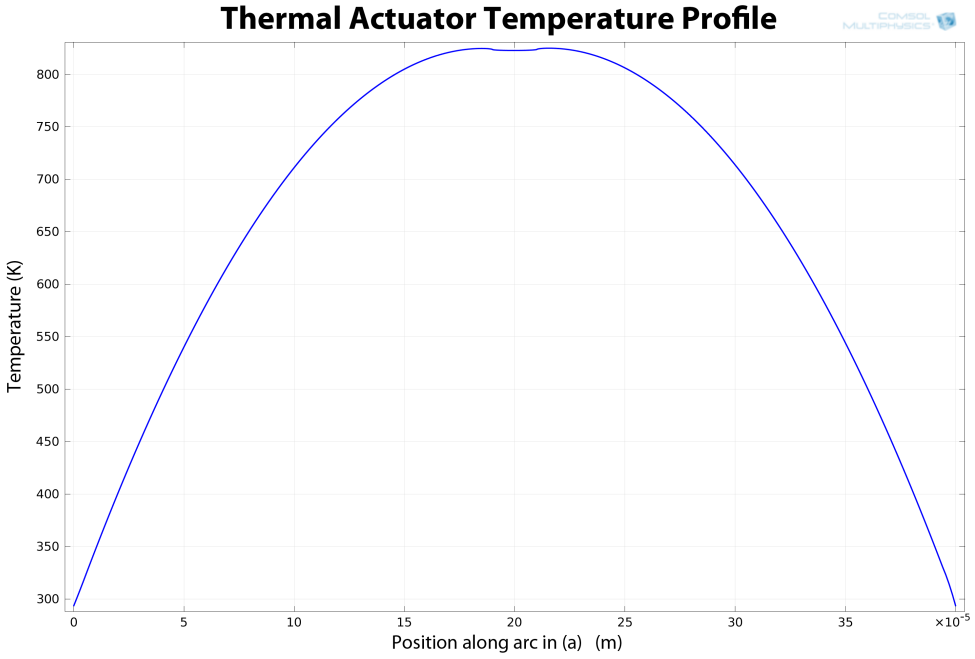
The model used for the static, dynamic, and transient simulations is the same 2D finite element model. The model assumes that the structure material is a linear elastic material with thermal expansion. The model also assumes geometric nonlinearity due to the relatively large displacements involved. The material is considered isotropic with a constant Young's modulus, Poisson's ratio, and density. The model also assumes a constant isotropic thermal conductivity. There are fixed constraint boundary conditions at all anchor points that prevents displacement at those points. For the symmetric simulations, the boundary at the centre of the shutter is set to be symmetric. The anchor points are also set to have a constant temperature of 293 K and a constant voltage potential of 0 V for the spring and lever anchors, and ±0.5 V for the actuator anchor points. The mesh used is a free triangle mesh whose size is

made sufficiently small such that the simulation results remain constant with further reductions in mesh size. All of the simulations are solved using the COMSOL direct solver, MUMPS, with a relative tolerance of 0.0010.

Figure 4.1 shows the results of a static simulation for the original sensor design. Figure 4.1(a) depicts the steady state temperature distribution of the thermal actuator and lever structure with deformation. A voltage potential of $\pm 1.5\,\mathrm{V}$ was applied across the thermal actuators. Figure 4.1(b) shows the temperature profile along the line indicated in (a). This temperature distribution is in close agreement with the model presented in [5] with the exception of a small flat spot in the middle due to the added thermal mass and reduced resistance of the shuttle.



(a)



(b)

Figure 4.1: *Simulated temperature distribution of the thermal actuator structure. Figure (a) shows the temperature distribution of the actuator and connecting lever structure while Figure (b) shows the temperature profile across the thermal actuator indicated by the black arc plotted in (a).*

Figure 4.2 shows the voltage potential distribution of the same static simulation again with the deformation of the structure. As shown, a differential voltage is applied across the symmetric thermal actuators, such that the potential of the shutter and lever mechanisms is zero. This is in agreement with previous simulations and measurements done by [5].

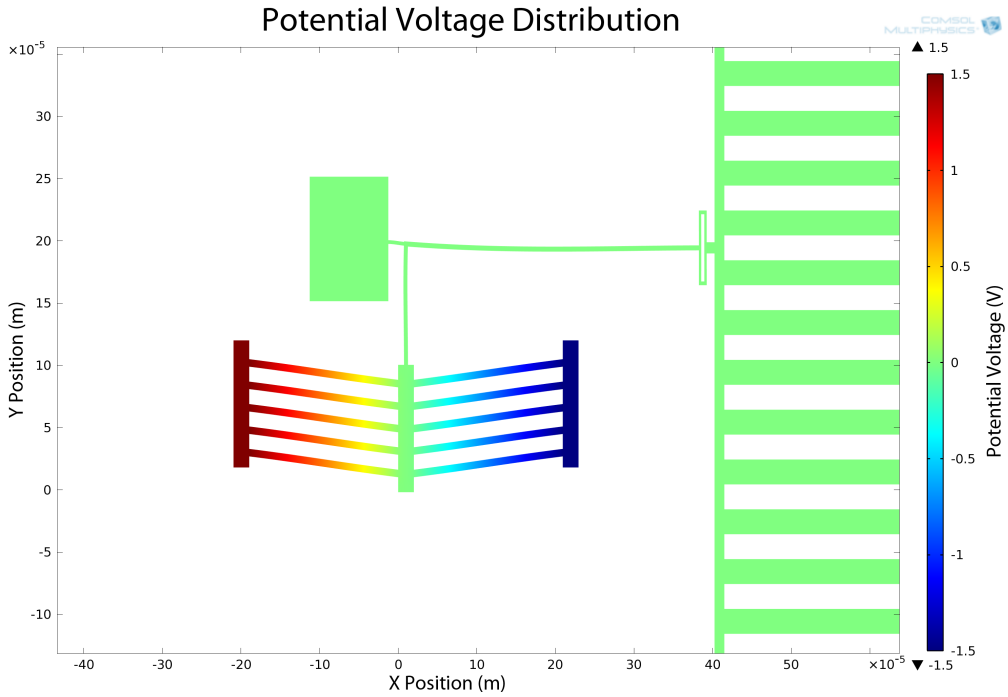


Figure 4.2: *Voltage Potential Distribution*

Figure 4.3 shows the deformed shape of the structure magnified by a factor of 10 to make the shape more plainly visible. It is also coloured according to the y-axis displacement. The point of maximal displacement is not in the shutter structure as might be expected, but in the lever that connects it to the thermal actuators. This indicates a sub-optimal design of the lever structure as it is unable to transfer the complete force of the actuators into the shutter. In this case, the shutter displacement is approximately 4.68 µm while the displacement in the lever is 5.68 µm, a full micron larger. Figure 4.4 shows the same plot for a sensor that has been optimized by genetic algorithm (case #1 of Table 3.3). The lever structure in this case is considerably

thicker and stiffer which is able to more efficiently transfer the force from the thermal actuators. The maximum displacement of this GA optimized design occurs in the shutter at approximately 7.78 µm for an applied voltage potential of 1 V.

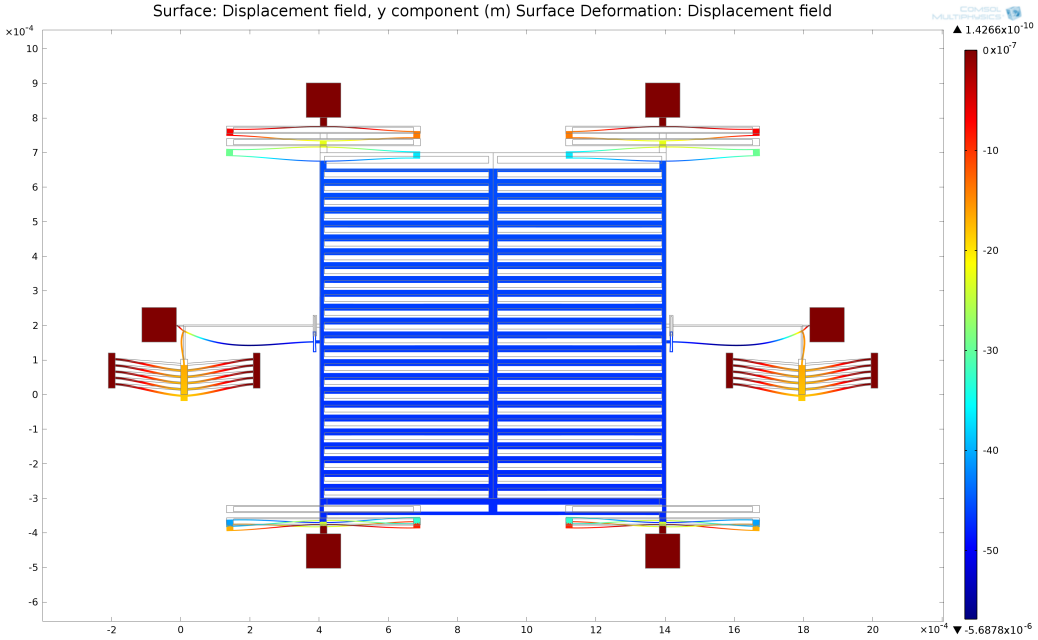


Figure 4.3: *Deformed shape of the entire original structure.*

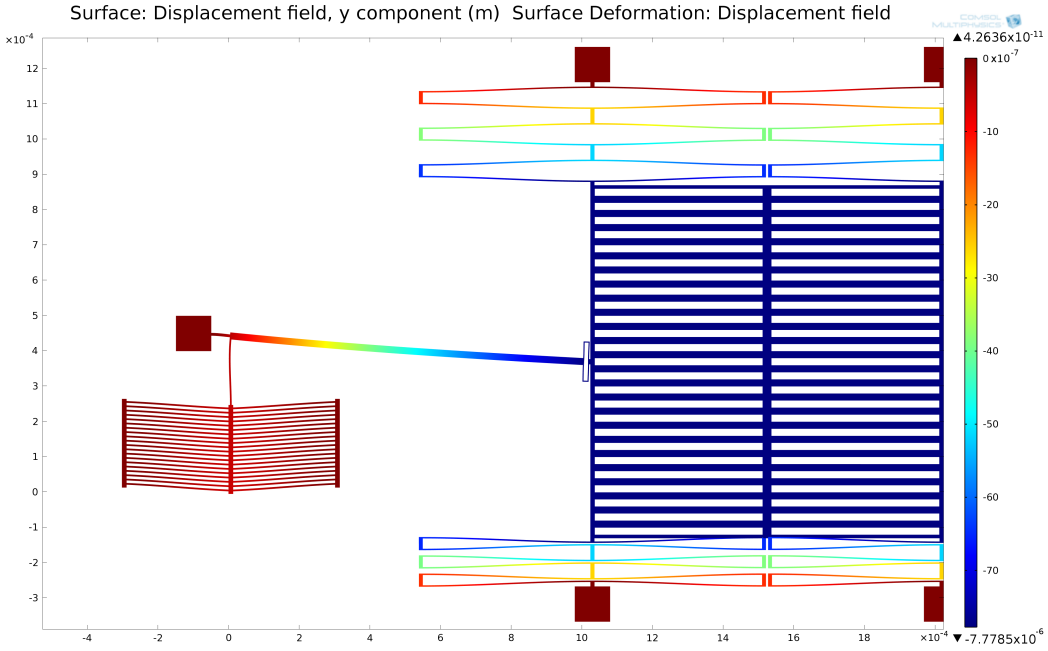


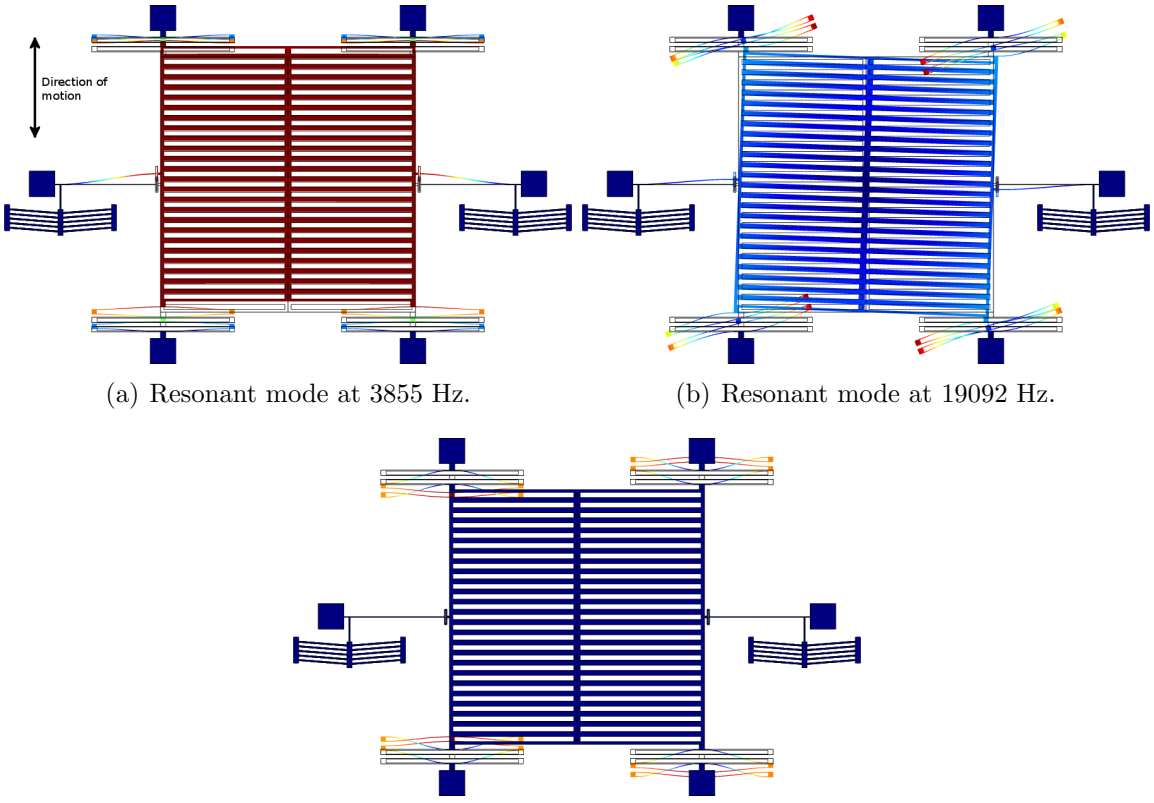
Figure 4.4: *Deformed shape of a GA optimized structure.*

4.3 Dynamic Simulation

The dynamic simulations performed were used to evaluate the natural resonant frequencies of the sensor designs. The same model used during the 2D static simulations was used, but with only the solid mechanics physics module. The eigenfrequency solver was then used to determine the resonant frequency. It was found that the simulated resonant frequency for the original design is 3857.9 Hz, which is in agreement with the calculated and measured resonant frequency for the design in [5], therefore, validating the model.

The eigenfrequency simulation calculates several resonant frequencies and their corresponding modes. The primary resonant frequency of interest is the mode in which the sensor oscillates in-plane with the wafer surface and perpendicular to the axis between the actuators, as shown in Figure 4.5(a). Two other resonant modes that were calculated are shown in Figures 4.5(b) and (c). They include a rotational mode where the structure rotates about the centre of the shutter, and resonant modes of the shutter support springs. The simulation was repeated with the same model extruded by 10 µm into the third dimenstion. In addition to the in-plane resonant modes calculated using the 2D simulation, several out-of-plane modes were found. Figure 4.6(a) to (c) show three modes of interest from this simulation. Figure 4.6(a) shows a resonant mode in which the shutter displaces upward. This mode is the lowest frequency of the undesirable modes, but is almost double the frequency of the primary mode, well above the regular operating frequency. The modes of Figure 4.6(b) and (c) show show rotational modes in which the shutter is rotated about the two in-plane axis at approximately 13208 Hz and 15185 Hz respectively. In the figures, the displacement pictured has been scaled in order to observe the shape of the resonant modes. The modes of resonance above the resonant mode of Figure 4.5(a) at 3855 Hz are far above the expected range of operational frequency and are not expected to

cause a problem with the normal operation of the sensor. Throughout the remainder of this thesis, when referring to the resonant frequency of the sensor design, it simply refers to the frequency of the first resonant mode in which the shutter is displaced in-plane with the wafer surface and perpendicular to the axis between the actuators.

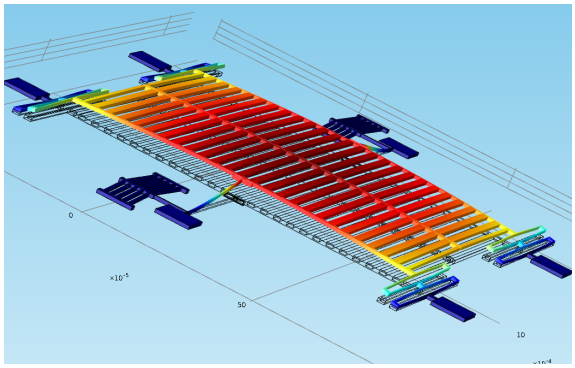


(a) Resonant mode at 3855 Hz.

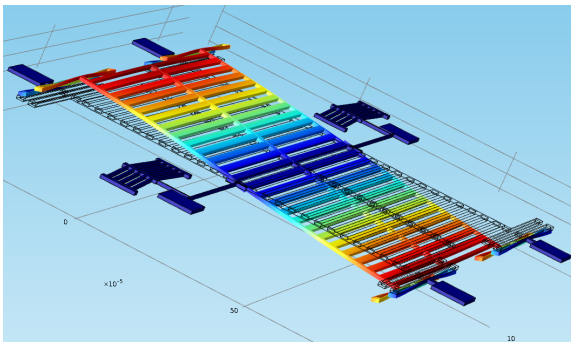
(b) Resonant mode at 19092 Hz.

(c) Resonant mode at 37697 Hz.

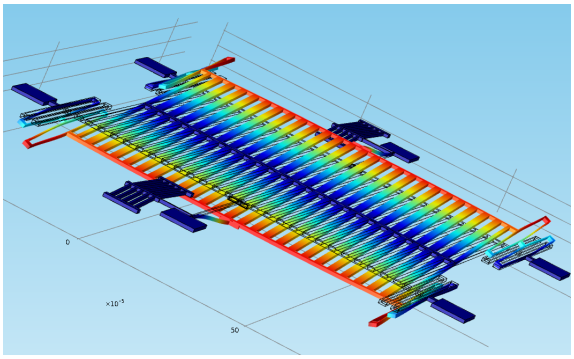
Figure 4.5: *Three in-plane resonant modes of the original field mill design.*



(a) Out-of-plane resonant mode at 7338 Hz.



(b) Out-of-plane resonant mode at 13208 Hz.



(c) Out-of-plane resonant mode at 15185 Hz.

Figure 4.6: *Three out-of-plane resonant modes of the original field mill design.*

4.4 Transient Simulation

Transient simulations were performed in order to determine sensor response to a changing drive signal. This allows for evaluation of the thermal time constant which cannot be evaluated using the static or dynamic simulations. The thermal actuators are able to heat up very quickly through joule heating, but after the voltage is removed, they must cool at a slower rate primarily by conduction through the actuator beams. As the operating frequency of the thermal actuators is increased beyond the thermal actuator cool down frequency supported by the thermal time constant, the actuator displacement is reduced.

4.4.1 Thermal Time Constant

In [5] the thermal time constant of a simple bent beam thermal actuator was estimated using a simple model of a thermal actuator. The model consists of a single straight beam connected at either end to the substrate which is held at a constant temperature of 25 °C. This model assumes that each beam of the thermal actuator is isolated from every other beam and that the only heat transfer occurs through the actuator beam and into the substrate at either end. The differential equation for this system is provided in Equation 4.1 [5],

$$k\frac{d^2T}{dx^2} + q = \rho C_p \frac{dT}{dt} \tag{4.1}$$

where k is the thermal conductivity of silicon, T is temperature, q is the rate of energy generated per unit volume, and ρ and C_p are the density and specific heat of silicon respectively. Setting the rate of energy generation, $q = 0$, and the initial temperature distribution to that of the heated actuator allows the differential equation to be solved for temperature as a function of time and position as it cools. The solution to this system yields the cooling thermal time constant τ_c as a function of length,

$$\tau_c = \frac{(2l)^2 \rho C_p}{k\pi^2} \tag{4.2}$$

where l is the length of one half-beam of the thermal actuator. For the original field mill design, the length of the thermal actuator half-beam is 190 µm, which yields an estimated time constant of approximately 87.5 µs. This model provides a rough estimate of the thermal time constant, however it is not entirely accurate since it does not account for the added thermal mass of the actuator shuttle, nor does it account for the heat loss through the connecting beam to the lever structure and its anchor.

To get a more accurate idea of the thermal time constant, it was modelled using transient analysis with COMSOL. A potential of $1\,V$ is applied to the actuators from $t = 0$ until $t_0 = 1.5$ ms. After t_0 the voltage potential is removed. To ensure that

the solver is able to converge, the waveform is smoothed by a feature of COMSOL to create a continuous first derivative for a small interval around the transition zone. The results of this simulation are plotted in Figure 4.7. The solid line represents the temperature at the centre of the thermal actuator as it is heated through joule heating and is allowed to cool. The dashed line indicates the corresponding shutter displacement. The cooling time constant, τ_c is extracted by measuring the time it takes for the structure to cool from its peak temperature to 36.6788% ($\sim 1\tau_c$) of the temperature difference. In this case, it is measured to be approximately $\tau_{\mathrm{c}} = 128.75\,\mu\mathrm{s}$. Likewise, the heating thermal time constant, τ_h is measured to be approximately $\tau_{\mathrm{h}} = 118.4\,\mu\mathrm{s}$. The heating time constant is slightly faster because the heat is generated throughout the entire actuator whereas it must be cooled by conduction through the substrate anchors.

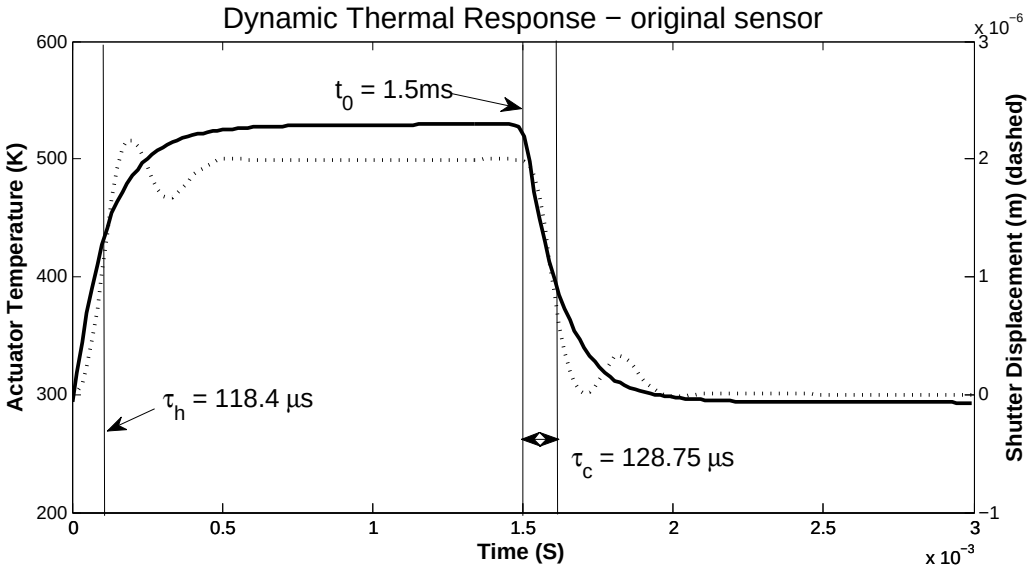


Figure 4.7: *Simulated Step Response of the original sensor design*

Given these time constants, the step response of the thermal actuators can be

approximated by the following equation:

$$T(t) = T_0 + \begin{cases} (T_1 - T_0)(1 - e^{\frac{-t}{\tau_h}}) & : 0 < t \leq t_0 \\ (T_1 - T_0)e^{\frac{-(t-t_0)}{\tau_c}} & : t > t_0 \end{cases} \tag{4.3}$$

where T_0 is the initial "cold" temperature, T_1 is the steady state "hot" temperature, t_0 is the time that the voltage potential is removed, and finally τ_h and τ_c are the heating and cooling time constants respectively.

When driven by a sinusoidal drive signal, the actuators heat during the first quarter period reaching the maximum temperature at $\frac{\pi}{2}$ radians. The actuators are then able to cool for the next quarter period as the drive voltage drops to zero. If the cooling thermal time constant is less than one quarter wavelength of the drive signal, then the temperature will closely follow the sinusoid. If, however, the cooling thermal time constant is greater than one quarter period then the actuators will be unable to follow the drive signal and will not reach the initial cold temperature T_0. At π radians, the cycle repeats but with the opposite polarity; heating until $\frac{3}{2}\pi$ and then cooling until 2π. This effectively doubles the operating frequency since the actuators heat and cool twice during each period of the drive signal. Since the mechanical response is the primary interest of the thermal actuators, from this point on when referring to the "operating frequency", it is considered to be double the drive frequency.

Figure 4.8 plots the cold temperature that the actuators are able to reach as a percentage of the difference between T_1 and T_0 as a function of operating frequency. For example, 100% would indicate that the actuators were able to cool completely back to T_0 at the specified operating frequency. Likewise, 10% would indicate that the actuators were able to cool only by 10% at the specified frequency. The solid line in Figure 4.8 shows the theoretical percentage based on Equation 4.3 versus operating frequency. The plotted data points are gathered from a set of transient simulations in which the applied waveform was sinusoidal with an amplitude of 1 V

and frequency swept in 50 Hz increments from 150 Hz to 2700 Hz. The peak-to-peak temperature value is then extracted and the percentage is calculated based on the maximum and initial temperatures for each simulation. We can clearly see agreement between the transient simulations and the theoretical values based upon the time constant τ_c measured from simulation. In order to achieve the same displacement at a higher operating frequency it is necessary to increase the applied voltage.

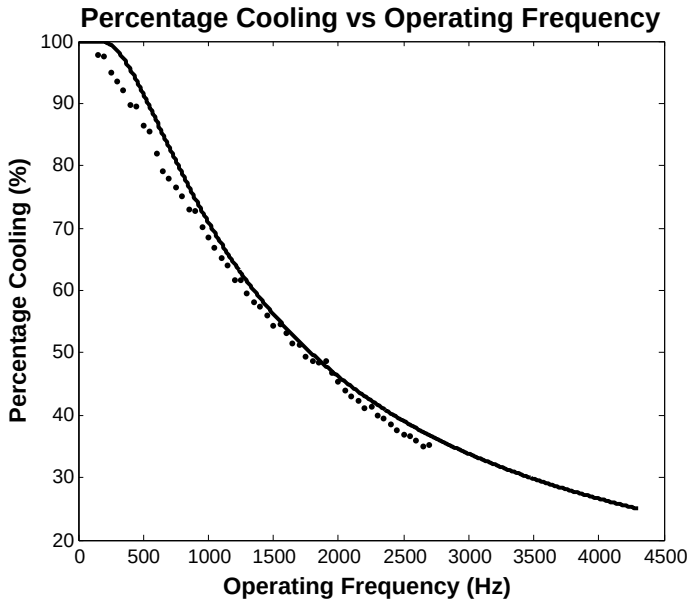


Figure 4.8: *Thermal actuator response vs operating frequency. The solid line represents the analytical values based on the measured time constant of* 128.75 µs*. The data points plotted are based on measurements taken from transient simulations.*

Others have studied the effects of the thermal time constant on chevron based actuators, such as R. Hickey et al. [46]. They designed estimates for the heating and cooling thermal time constants for both chevron and hot/cold arm thermal actuators. Hickey et al. estimated a thermal time constant of 134 µs for a chevron actuator design and simulated a more accurate thermal time constant of 30 µs [46]. Hickey et al. also developed a process for measuring the thermal time constant using a beam splitter and a laser probe microscope. With this apparatus they were able to accurately measure the thermal time constant of their chevron actuator to be approximately 60 ± 10 µs. Their actuator is significantly faster than the one presented here because it is roughly half the size.

In order to further validate the simulations presented here, the geometry of the COMSOL model was updated to match the dimensions given by Hickey et al. After applying the same transient simulation used to generate the data in Figure 4.8, the cooling thermal time constant t_c was measured to be approximately 63 µs. This measurement is in very close agreement with the measured thermal time constant of

[46]. Hickey et al. also present similar findings on the thermal frequency response and the necessity to increase operating voltage to achieve the same displacement at higher frequencies.

Several sensor designs selected from the results of the genetic algorithm are compared by their thermal time constants in Table 4.2. These designs are the same as those selected in Table 3.3 in Chapter 2. Although the designs listed have reasonable resonant frequencies, the thermal time constants significantly limit the operating frequency. This was not an issue when running the original sensor since a large mechanical amplification can be achieved when operating at resonance.

Table 4.2: *Thermal Time Constants*

Case #	Voltage (V)	Freq. (Hz)	τ_c (μs)	τ_h (μs)	80% Cooling Freq. (Hz)
1. High Displacement	0.80	2897.0	250.63	246.34	789.06
2. High Frequency	0.96	3511.1	275.55	255.57	725.58
3. Low Stress	1.1	2311.4	230.89	219.18	856.5
4. Design in [5]	3.2	3857.9	128.75	118.14	1536.02

In order to combat the problem of reduced displacement due to high thermal time constants, the thermal time constant could be added as an objective of the genetic algorithm. However, since the transient simulations that extract the time constant of a design are relatively computationally intensive, the objective could instead be estimated using Equation 4.2 with very little additional computation. Going back to the results from the genetic algorithm, new solutions can also be selected with the time constant objective in mind. This also shows the flexibility of a Pareto set genetic algorithm since the previously computed designs can easily be compared in terms of new objectives without having to run the entire genetic algorithm again. However, it is likely that better results could be found if the algorithm is re-run with this objective added.

Table 4.3 below shows several newly selected solutions for the three previous cases with superior thermal time constants. The newly selected solutions compromise the

displacement of the thermal actuators for a faster thermal time constant. The result is that they require a higher actuating voltage to achieve the 5 μm displacement requirement which also corresponds to an increase in maximum temperature and stress.

Table 4.3: *Re-selected solutions for cases 1-3 of Table 4.2 to have faster thermal time constants at the expense of the other objectives.*

Case #	Voltage (V)	Freq. (Hz)	τ_c (μs)	τ_h (μs)	80% Cooling Freq. (Hz)
1. High Displacement	0.92	2284.5	188.77	178.57	1047.63
2. High Frequency	1.26	3325.0	137.35	123.43	1439.77
3. Low Stress	1.13	2328.7	199.80	174.52	989.78

Figure 4.9 shows the results of transient simulations with varying voltage and a fixed operating frequency of 1.7 kHz for case #2 from Table 4.2. At this frequency, the thermal actuators have enough time to cool to approximately 50.5 % of their steady state temperature. Although the steady state displacement at $1V$ is relatively large at $\sim 5\,\mu m$, the voltage must be increased significantly beyond that in order for the peak-peak displacement of the actuators to be sufficient ($\sim 5\,\mu m$). This dashed lines in Figure 4.9 indicate that greater than 1.6 V is needed to achieve the necessary displacement of 5 μm. This minimum voltage will increase with frequency because of the effect of the thermal time constant. The increase in peak-peak shutter displacement is roughly linear with an increase in operating voltage.

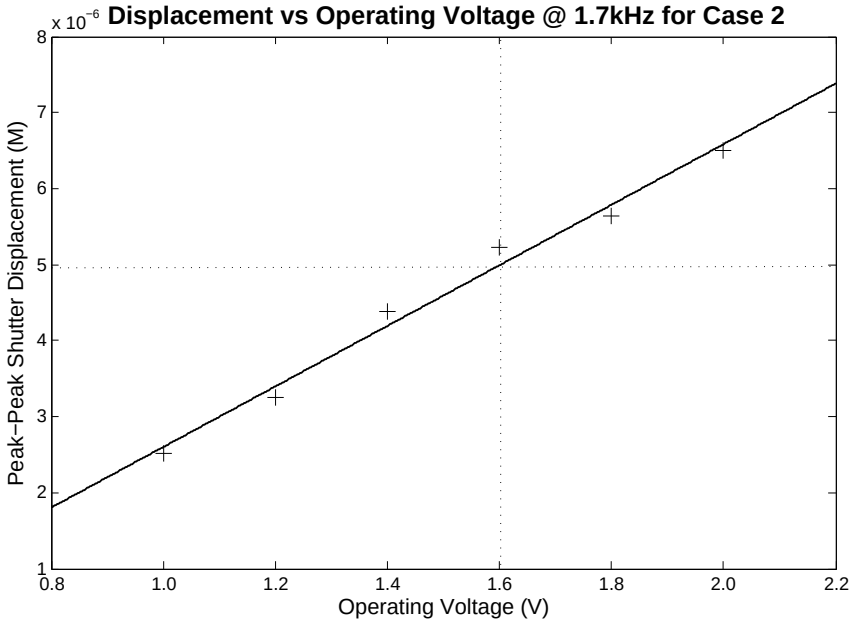


Figure 4.9: *The points plotted correspond to the peak-peak displacement at* 1.7 kHz *achieved with design case #2 from Table 4.2 as the operating voltage is increased.*

4.4.2 Experimental Measurement of Thermal Time Constant

The thermal time constant can be measured directly. This is accomplished using the experiment outlined below (see Figure 4.10). This procedure is similar to that used by [47] to measure the thermal properties of microbolometers. A thermal actuator device is connected as part of a Wheatstone bridge driven by a function generator. The function generator is configured to output a 100 Hz, 1.7 V_{pp} square wave with 50% duty cycle and a voltage offset of 0.9 V. The offset is important so that there is always a current flowing through the Wheatstone bridge making it possible to measure changes in resistance at all times. As the actuator heats up, its resistance changes as a function of temperature. This change in resistance relative to the other resistors in the Wheatstone bridge is measured as a voltage difference across the bridge. The resistor R is chosen to be a value close to the room temperature resistance of the thermal actuator, which in this case was measured to be 58 Ω, and so the closest standard value resistor, 56 Ω, was chosen. If possible, the resistors should be as precise as

possible; any difference in resistor values will present as an offset voltage across the Wheatstone bridge. The resistor R must also be able to dissipate higher power than the actuator with negligible change in resistance due to Joule heating. During the high cycle of the square wave, the actuator is heated and its resistance and voltage drop increases. When the low cycle begins, the actuator cools and so its resistance and voltage drop reduce. This can be seen as exponential curves during each half of the square wave cycle. The signal from the Wheatstone bridge is then fed into a high-impedance non-inverting amplifier which makes it possible to more accurately record small changes in resistance.

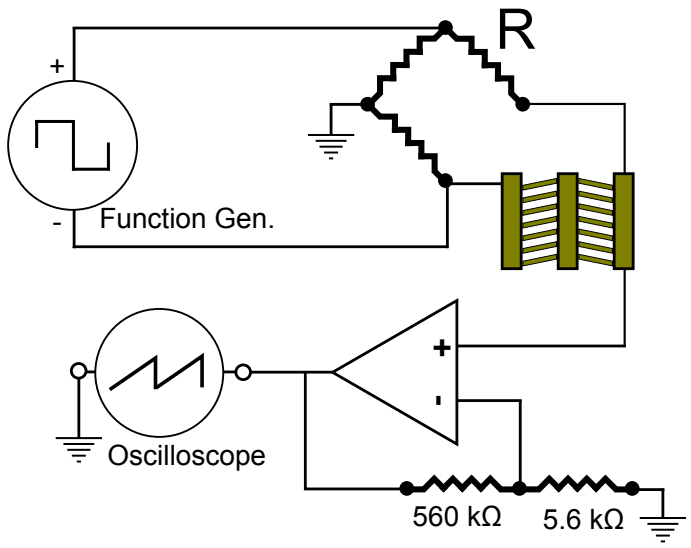


Figure 4.10: *Experimental Measurement of Actuator Time Constant*

Figure 4.11 shows a photo of the actuator being tested. For a scale reference, each actuator arm is approximately 190 µm long. The electrical resistance of the actuator and connecting traces (not pictured in Figure 4.11) is estimated using the resistivity of gold, $\rho = 2.44 \times 10^{-8}\,\Omega\,\mathrm{m}$, and the mechanical dimensions estimated from Figure 4.11. This resistance estimate is approximately $8 - 12\,\Omega$. This estimate also includes the added resistance of the bond wires, $\sim 0.2\,\Omega$, based on a rule-of-

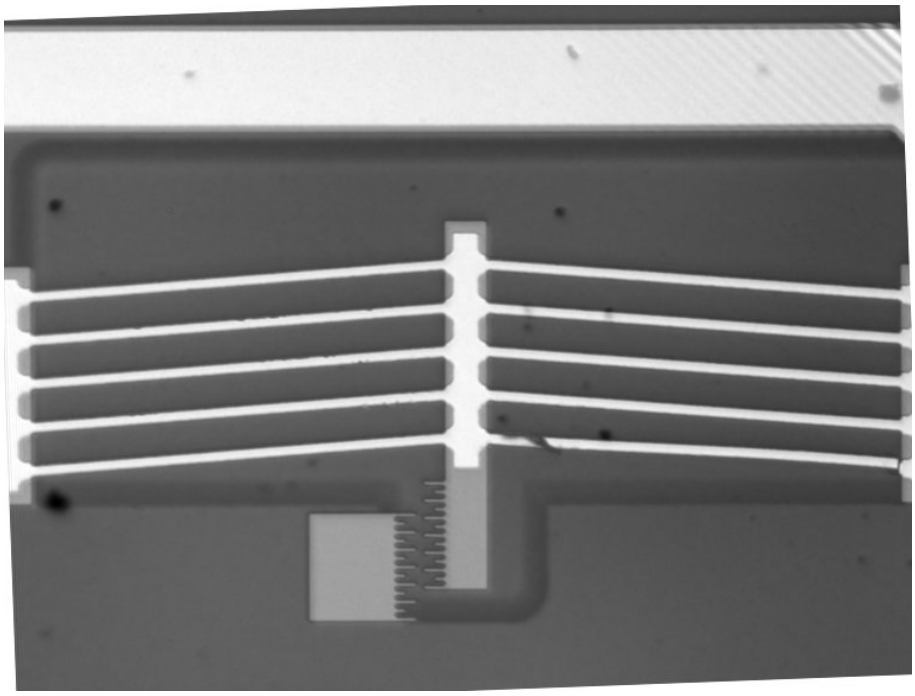


Figure 4.11: *Actuator Under Test*

thumb estimate of 1 Ω per inch [48]. However, actual resistance measurements of the actuator yield a resistance of approximately 58 Ω. It is believed that the electrical resistance is significantly higher than theoretical estimates because of chromium/gold interdiffusion, as reported by [45]. The MicraGEM [12] process makes use of a 25 nm adhesion layer of chromium to which a 75 nm layer of gold is deposited. After being exposed to high temperatures, for example, during manufacturing or regular operation of the thermal actuator, interdiffusion of the chromium and gold occurs and has a dramatic effect on the electrical resistance of the gold. The resistance can be increased by as much as an order of magnitude depending on the maximum temperature and the amount of time held at that temperature [45]. The effects of interdiffusion could be reduced by introducing a thin diffusion barrier layer between the chromium and gold, such as nickel, which better preserves the resistance of the gold [49].

Figure 4.12 shows a screen capture from the oscilloscope used in the experiment. The actuator being tested is one of the original designs built by [5]. The oscilloscope is set to capture the cooling half of the cycle. The gain of the amplifier circuit is

approximately 101. The cooling thermal time constant measured is approximately 150 µs. This time constant is quite close to the theoretical value of 125 µs from simulation. The discrepancy in thermal time constant could be explained by a number of factors such as reduced thermal conductivity due to interdiffusion of the gold conductor and chromium adhesion layers, reduced thermal conductivity of the underlying silicon due to dopants, as well as a reduced thermal conductivity due to increased temperature.

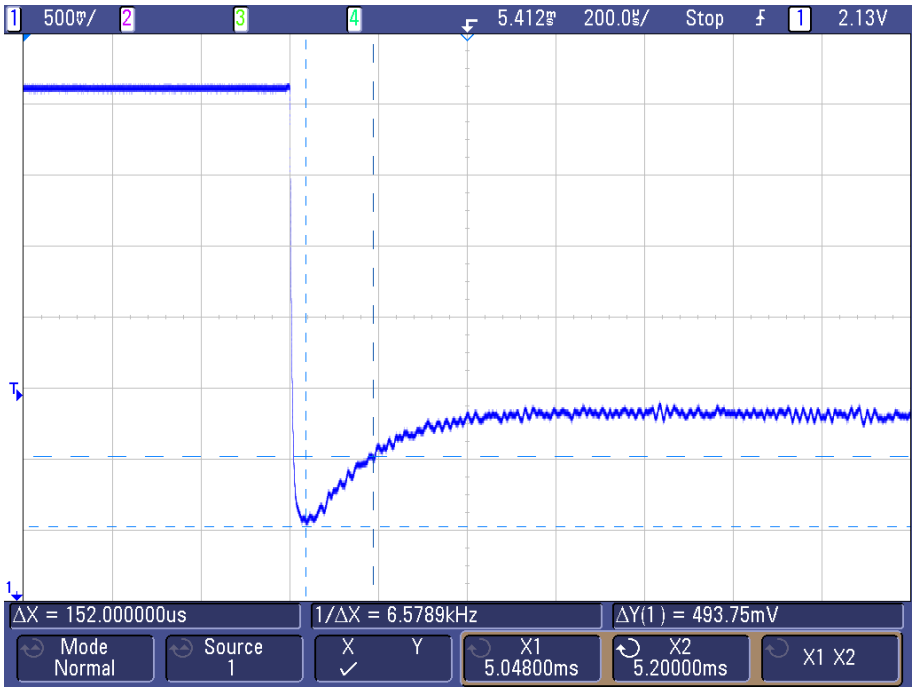


Figure 4.12: *Time Constant Measurement Results*

4.4.3 Mechanical Dynamic Response

In this section, the effect of resonance on the mechanical response of the sensor is examined. Figure 4.13 shows the results of a set of transient simulations with a constant actuation voltage of 2 V and operating frequency swept from 300 Hz to 5400 Hz. These simulations do not consider damping forces and can be considered as the case when the sensor is operated at high vacuum. In this scenario, it is interesting

to note from Figure 4.13 that the mechanical amplification of displacement due to approaching resonance seems to increase at a rate slightly greater than the reduction of peak-peak actuator displacement. After approximately 2500 Hz the mechanical amplification begins to dominate and the peak-peak displacement rapidly increases as it gets closer to resonance. This is why the original design is able to operate at very low power, however, it is very sensitive to changes in resonant frequency which can occur due to slight variations in temperature, pressure, or even variations in rapidly changing strong electric fields. Although tempting, it is not presently possible to measure the quality factor of the resonance peak using these data because the transient simulation does not converge for the case when operating exactly at the natural resonant frequency.

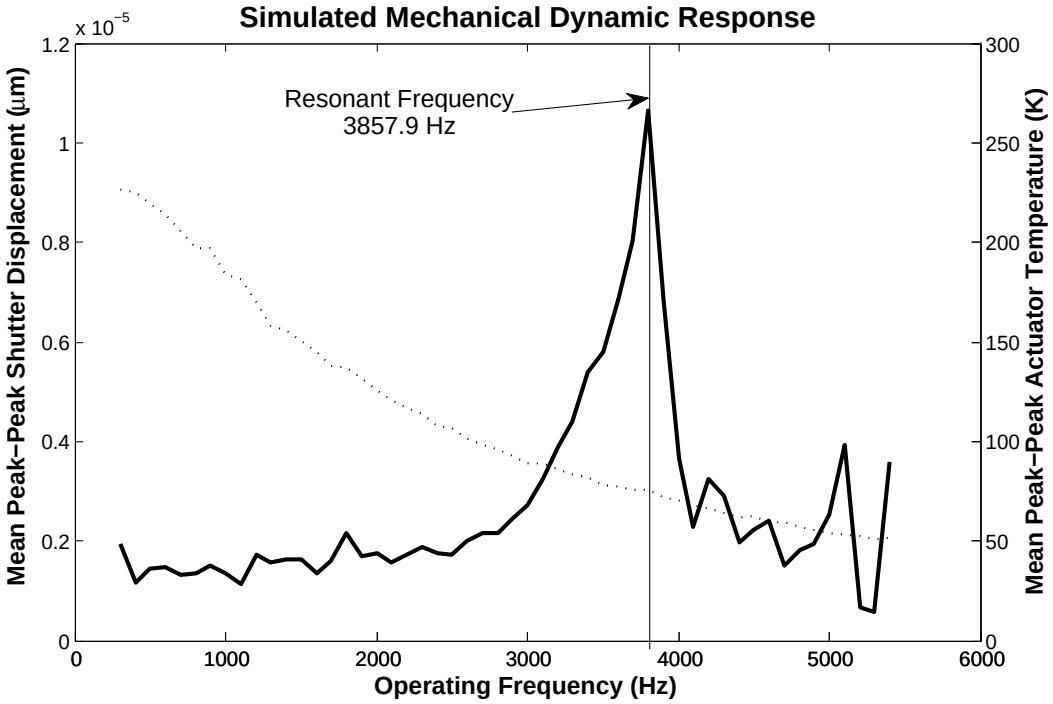


Figure 4.13: *Simulated Mechanical Dynamic Response. The solid line plots the mean peak-peak shutter displacement vs operating frequency while the dashed line depicts the mean peak-peak temperature of the actuator for the original sensor design.*

Damped Mechanical Response

Figure 4.14 plots the peak-peak amplitude of shutter displacement vs. operating frequency for four levels of damping. The black line shows the same undamped response of Figure 4.13 while the magenta, blue and red plots show the response with increasing levels of damping. As expected, with increased damping the mechanical amplification becomes lessened until it is essentially eliminated as in the red plot. The second effect of increased damping is a reduced resonant frequency.

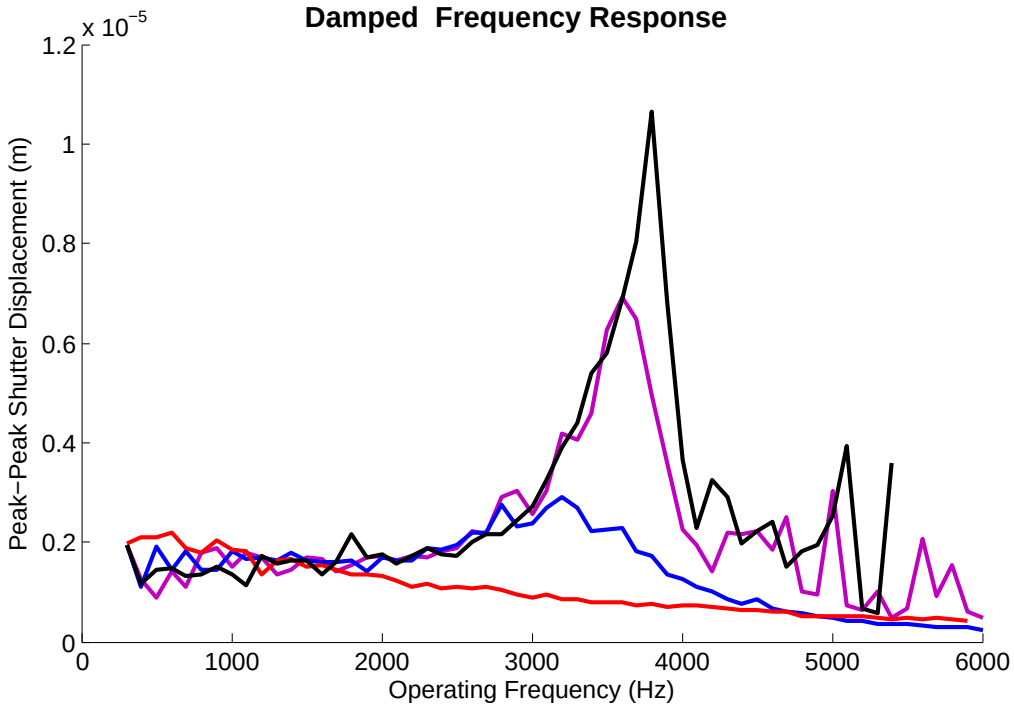


Figure 4.14: *Simulated Damped Mechanical Dynamic Response. The black plot shows the same undamped mechanical response as in Figure 4.13. The magenta, blue and red plots show frequency response for the same sensor with increased damping.*

In the totally damped case (red) the response is dominated by the thermal time constant of the actuators with no observable mechanical amplification. The totally damped case is shown again in Figure 4.15 along with the peak-peak actuator temperature. As expected, the mechanical response in the totally damped case follows closely the thermal response of the actuators. When the sensor is exposed to air at

atmospheric pressure, the damped response is expected to be somewhere between the totally damped case (red) and the undamped case (black).

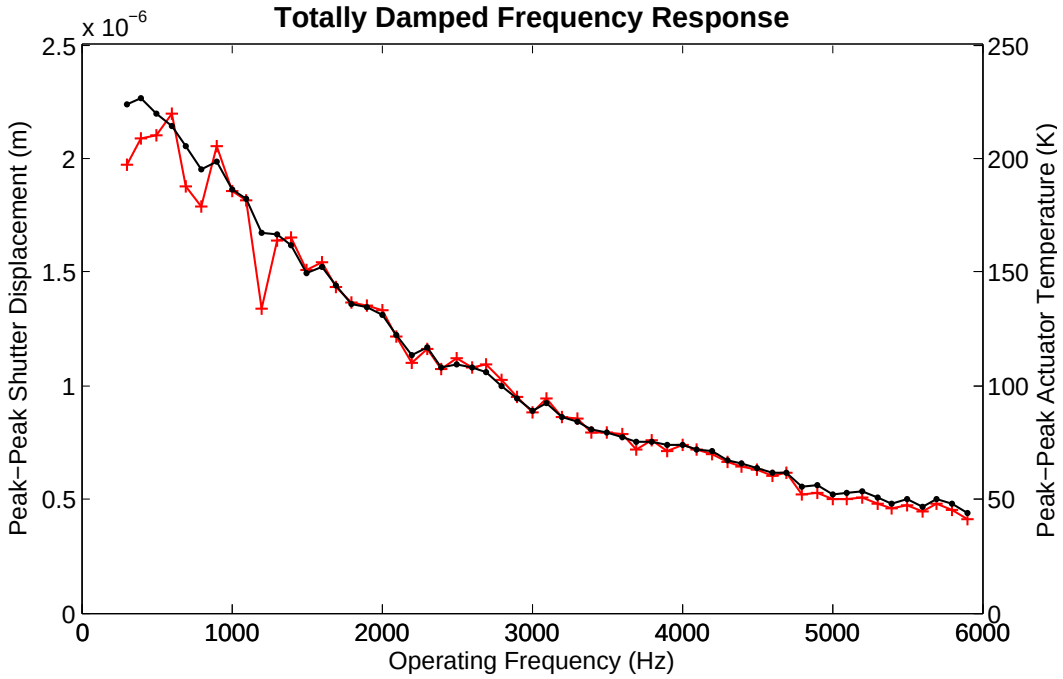
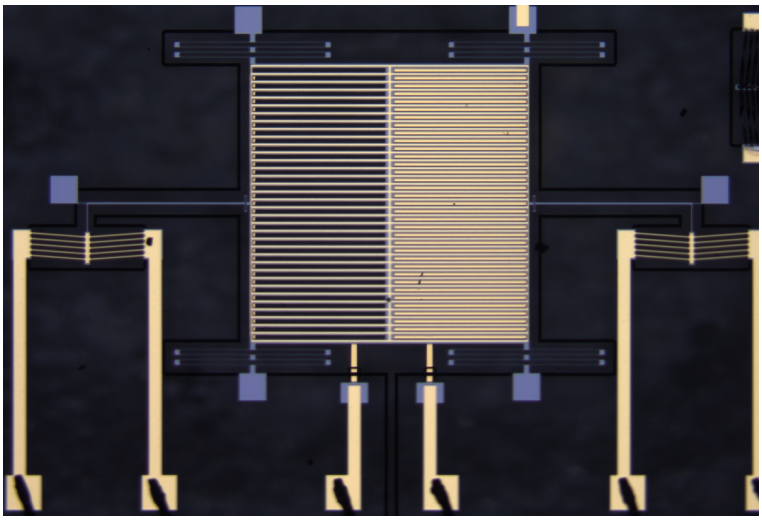


Figure 4.15: *Mechanical resonance is totally damped. Shutter displacement (red) closely follows the thermal actuator temperature (black).*

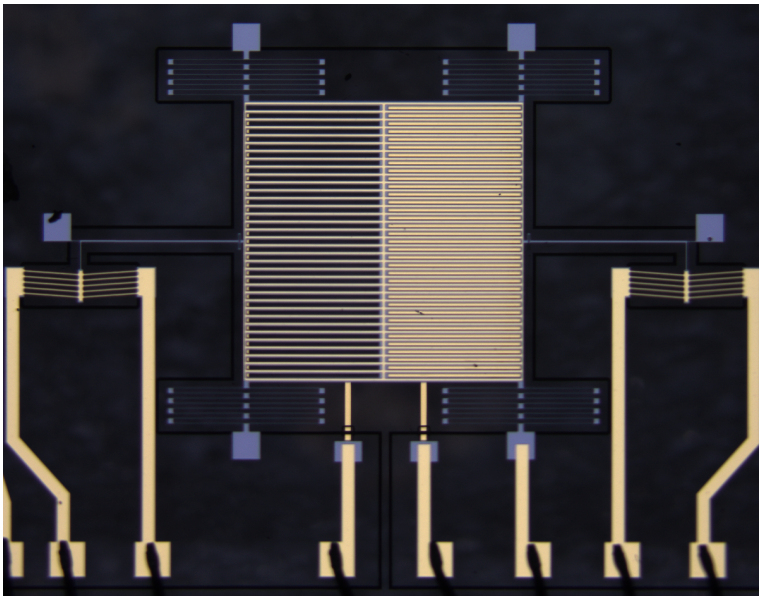
4.4.4 Optical Measurement of Dynamic Response

Two original designs were tested using optical measurements of shutter displacement. The devices tested were manufactured using the MicraGEM [12] process which is detailed in Chapter 5. The measurements were performed using a Fogale Photomap 3D optical profiler [50]. The two devices pictured in Figure 4.16 were dynamically measured using the 3D optical profiler. The first device, Figure 4.16(a) is the nominal design listed in Table 3.1 with two spring loops for each set of the supporting springs. The second device, Figure 4.16(b) is the same design but with an additional two spring loops.

The sensor is fixed beneath the 3D profiler at atmospheric pressure and actuated using a sinusoidal waveform with an amplitude of approximately 380 mV and no dc



(a) Photograph of two spring field mill design.

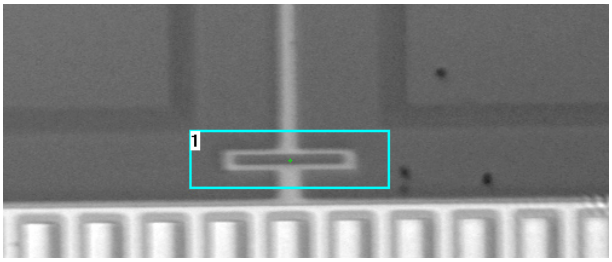


(b) Photograph of four spring field mill design.

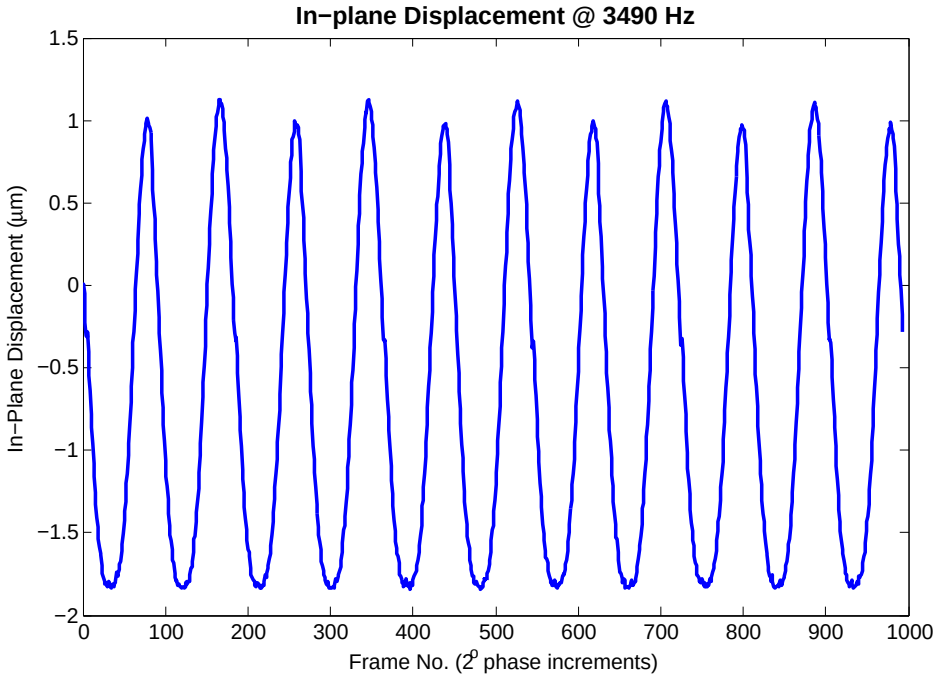
Figure 4.16: *Photographs of the two sensors whose dynamic response is measured using a 3D optical profiler. Photo (a) shows the two spring design whose geometry matches the nominal parameters listed in Table 3.1. Photo, (b), shows the same design with two additional spring loops.*

offset. The profiler records frames synchronously using strobed LED lighting at 2° phase increments and records these frames to produce a slowed video of the sensor's operation. This corresponds to 180 samples per actuation cycle. This allows the profiler to capture the entire motion of the structure over many complete cycles. The

Photomap analysis software is able to then measure in-plane displacements from one frame to the next using digital image processing and correlation techniques. Figure 4.17 shows the results of this technique for one frequency, 3490 Hz which was found to be close to the resonant frequency as it produces the largest mechanical response. As expected, near the resonant frequency the peak-peak shutter displacement is largest at approximately 2.85 µm displacement.



(a)



(b)

Figure 4.17: *Horizontal in-plane displacement measured near resonance. (a) shows a single frame from the optical 3D profiler with a rectangular selection of the four-spring sensor pictured in Figure 4.16b, which is analyzed for horizontal in-plane displacements. (b) shows the resulting plot from the analysis when operated at* 3490 Hz *(close to the resonant frequency).*

This in-plane displacement measurement is repeated at 100 Hz increments from 200 Hz to 5000 Hz with a few more closely spaced measurements added near the resonant frequency for both of the sensors pictured in Figure 4.16. Since the optical profiler is capturing the response at each phase from different cycles of actuation, there is error due to each cycle not being exactly the same as the last; for example minute changes in temperature, air pressure, or vibration are sources of error. A digital low-pass filter implemented in MATLAB is used to remove this noise and allows more precise locating of peaks in the mechanical response. Each measurement is passed through a 20th order low-pass finite impulse response filter with a cut-off frequency that is set at 4 % of the sample frequency. These parameters were chosen experimentally to provide the least amount of filtering while eliminating enough of the measurement-to-measurement error for analysis. For the example of Figure 4.17, the actuation frequency is 1745 Hz and it is sampled 180 times per cycle. This leads to a sample rate of 314.1 kHz, 4% of which is the cut-off frequency, 12.564 kHz, well above the frequency of any expected mechanical response. For the lowest actuation frequency of 100 Hz, this cut-off frequency would be 720 Hz, still well above the response frequency of 200 Hz. The peak-peak response is then determined from the filtered data by subtracting the mean value of the upper peaks from the mean value of the lower peaks.

The analyzed response for each of the two sensors is plotted in Figure 4.18. The response of the two spring-loop design is plotted in blue while the response of the four spring-loop design is plotted in red. As expected, the springs of the four spring-loop design are not as stiff resulting in a larger response at a lower resonant frequency. The response also resembles the simulated damped responses of Figure 4.14 albeit with a much lower displacement due to the non-optimized geometry of the designs being tested.

It is also interesting to note that the low-frequency response (below ~ 2500 Hz)

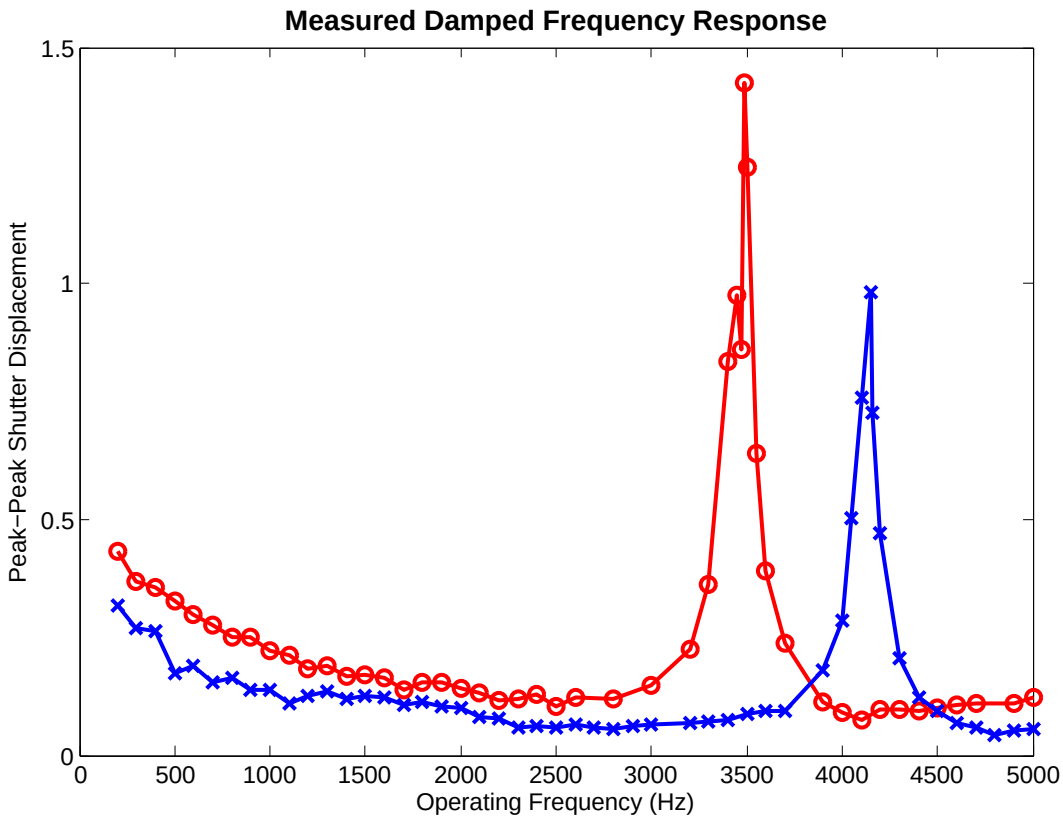


Figure 4.18: *The measured mechanical response for sensors pictured in Figure 4.16. The red plot is the response of the four spring-loop design while the blue plot is that of the two spring-loop design.*

roughly matches the expected thermal response of Figure 4.15. This is shown in Figure 4.19 which plots the measured shutter displacement (normalized) for the four-loop spring design of Figure 4.17(b). The red and green plots show the theoretical frequency response for a system with time constant of $t_c = 128.75\,\mu s$ (red) and $t_c = 150\,\mu s$ (green). These time constants correspond to the time constants measured from simulation (Figure 4.7) and from direct electrical measurement (section 4.4.2) respectively. As expected, the optical measurements agree with the simulated and measured thermal time constants. At frequencies above 2000 Hz, it is observed that the displacement begins to deviate upwards from the expected thermal time constant curves. This deviation is explained by the mechanical amplification that begins as the frequency is increased towards the resonant frequency.

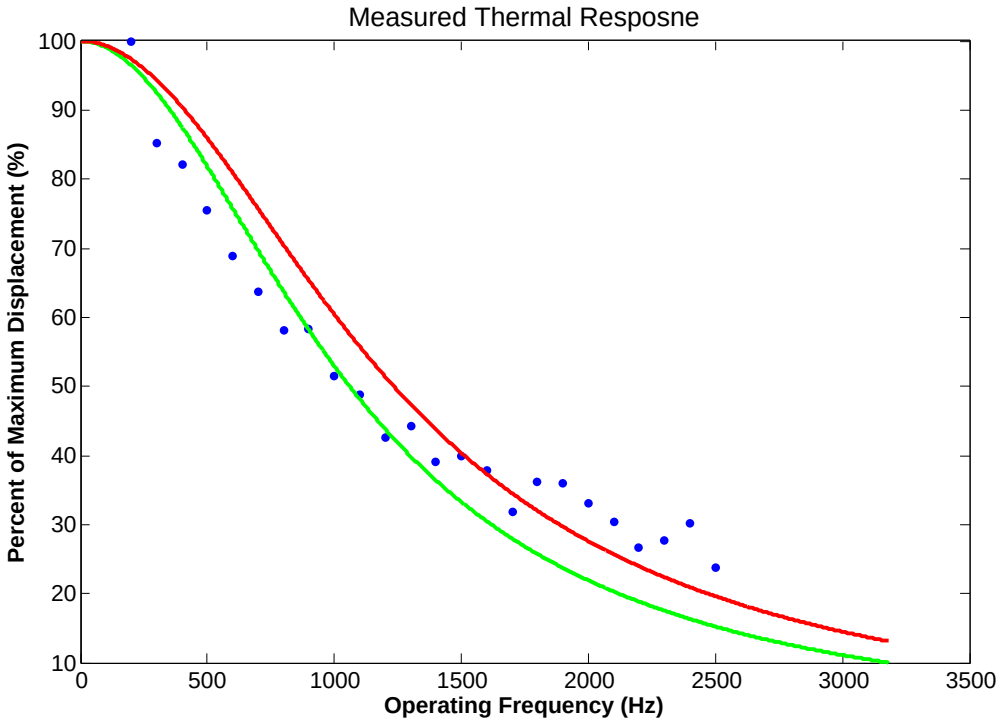


Figure 4.19: *The normalized low frequency mechanical response of the four spring-loop design (dots) plotted along side the theoretical thermal response with a time constant* $t_c = 128.75\,\mu s$ *(red) and* $t_c = 150\,\mu s$ *(green)*

.

Chapter 5

Fabrication

5.1 Introduction

The original electric field mills developed in [5] were manufactured using the MicraGEM [12] process through CMC Microsystems. At the time of writing, this process was no longer being offered through CMC Microsystems for academic research purposes. Therefore, it was decided to try and adapt the designs to the PolyMUMPS fabrication process which was currently available. This chapter presents both processes as applied to the fabrication of the electric field mill. While the resulting devices manufactured using the PolyMUMPS process failed to effectively operate, they are presented and analyzed here. The MicraGEM process is also described as well as some of the work done to recreate a similar process in-house.

5.2 PolyMUMPS

PolyMUMPS is a surface micromachining process where polycrystalline silicon and phosphosilicate glass (PSG) are alternately added in layers. Phosphosilicate glass is a commonly used compound that consists primarily of silicon dioxide (SiO_2) and P_2O_5. Each layer is subsequently patterned and etched to build-up the device. Finally, the

supporting PSG layers are completely etched releasing the finished device structure.

Figures 5.1 - 5.7 illustrate this process by showing a simplified structure with features similar to those used in the field mill design. The full details of the Poly-MUMPs process and design rules can be found in the process manual [51]. Figure 5.1 shows the starting n-type silicon wafer substrate with a 600 nm thick layer of silicon nitride deposited with low pressure chemical vapour deposition (LPCVD). The silicon nitride layer serves as an electrical isolation of the poly-silicon and the substrate, and also acts as a lower friction surface for any moving structures above. On the silicon nitride, a 500 nm layer of poly-silicon is deposited using LPCVD. This layer is patterned and etched using lithography and reactive ion etching (RIE) to form the bottom electrodes and traces for the field mill sensor.

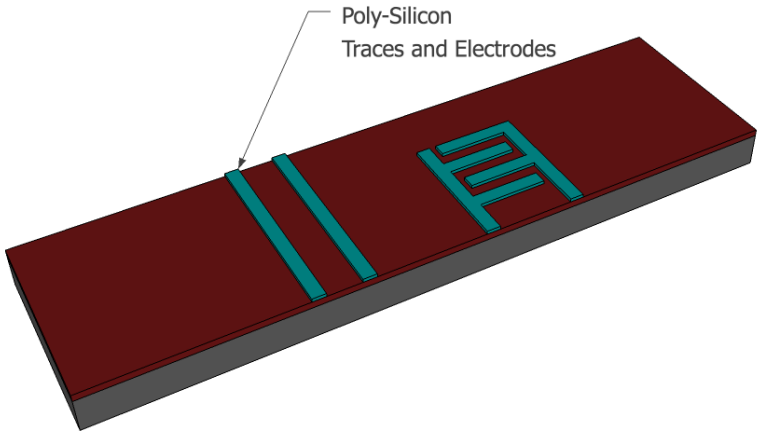


Figure 5.1: *Silicon nitride is deposited on an n-type wafer. A layer of poly-silicon is deposited and patterned to form traces and bottom electrodes.*

Next, a 2 µm sacrificial layer of PSG is deposited by LPCVD, Figure 5.2. This layer is first annealed at 1050 °C for one hour which causes the phosphorous from the PSG to dope the polysilicon bottom electrode layer below, making it conductive. The sacrificial PSG layer also defines the height of the shutter above the electrodes below. Not pictured in Figure 5.2, the PSG layer is patterned and etched to create

dimples in the centre of the shutter. These dimples are later filled by the next layer of poly-silicon and help support the device layer above to prevent it from collapsing and sticking to the lower poly-silicon layer. The PSG is then patterned and etched with anchor holes, pictured in Figure 5.3. These holes anchor the upper device layers to the substrate.

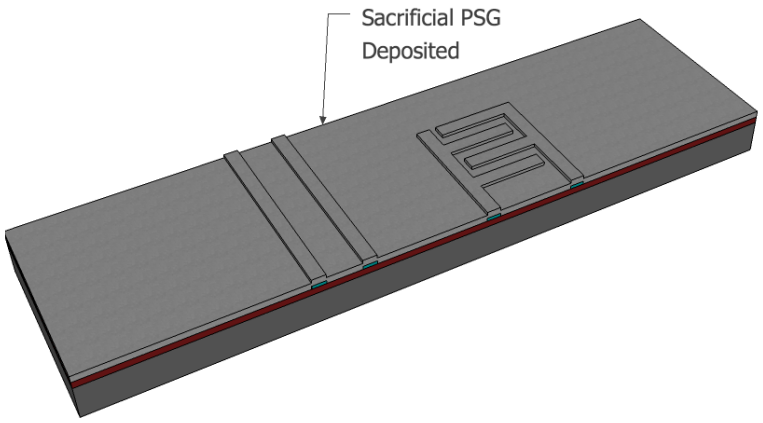


Figure 5.2: *An oxide is deposited covering both the nitride and poly-silicon layers.*

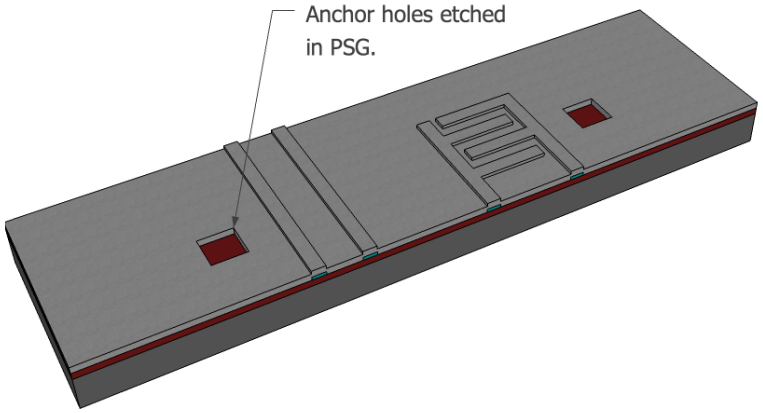


Figure 5.3: *Anchor holes are etched in the PSG.*

A 2 µm layer of poly-silicon is then deposited with LPCVD, see Figure 5.4. A thin PSG hard mask is then deposited and annealed to dope the poly-silicon device layer. This anneal also helps to relieve stress that may exist in the poly-silicon layer. The PSG hard mask is used to define the device layer rather than photoresist since it will better withstand ion bombardment during RIE. After etching, the PSG mask is stripped, resulting in a the structure pictured in Figure 5.5. Notice that the pattern of the lower electrode layer transfers to the upper device layers and appears as bumps on the patterned device layer.

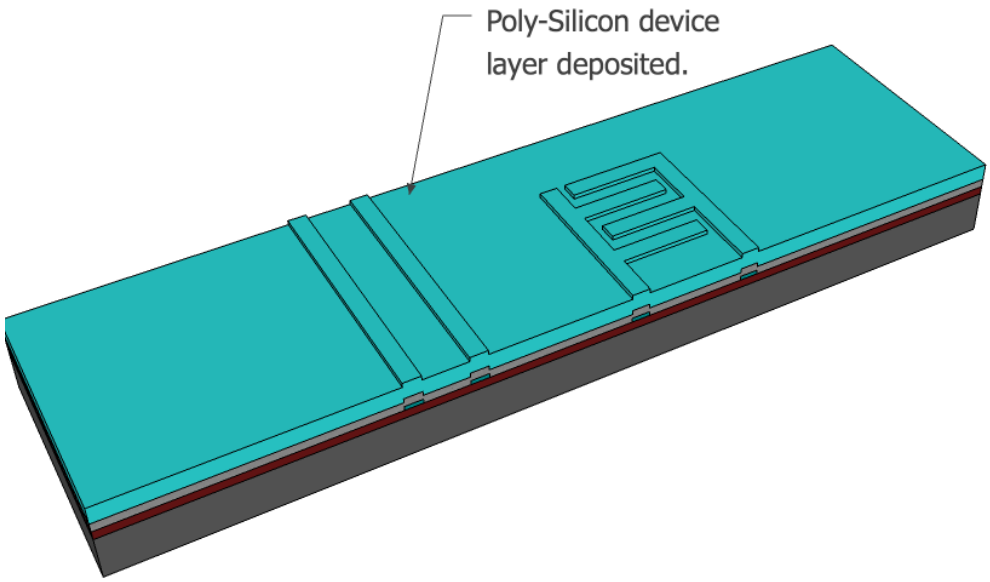


Figure 5.4: *A layer of poly-silicon is deposited.*

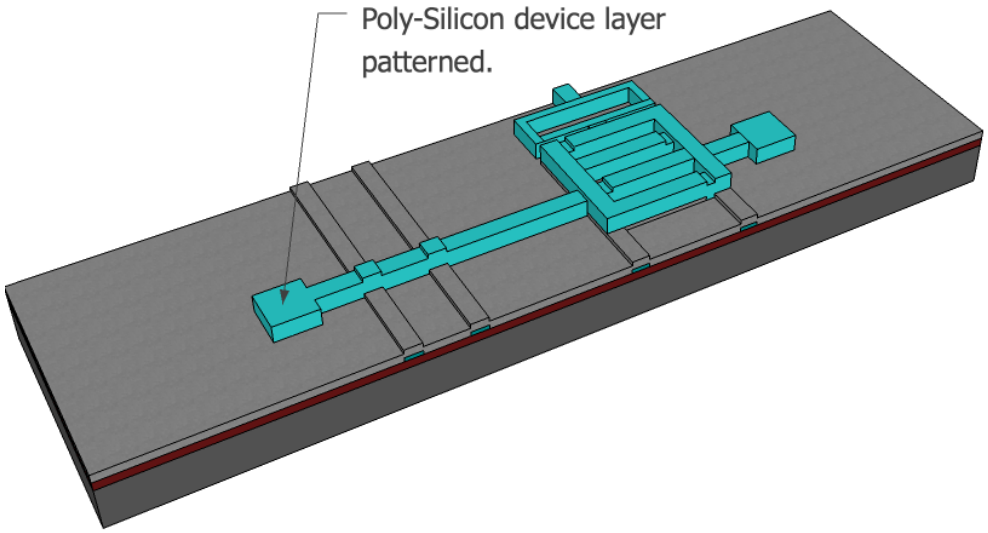


Figure 5.5: *The poly-silicon device layer is patterned and etched.*

A second PSG oxide, 0.75 µm thick, is deposited on top of the poly-silicon device layer. This layer is patterned twice using two masks at different depths of field; one mask is used to open the PSG to mechanically and electrically connect the next poly-silicon layer to the previously etched poly-silicon layer, while the second mask is used to open an anchor hole to the substrate through the top-most PSG layer and the previously deposited PSG layer. The second poly-silicon layer is not used in the electric field mill design since it must be at least 2 µm smaller than the previous poly-silicon layer which is not possible over thin sections such as the springs. However, since the process is shared amongst other users, the poly-silicon and hard mask layers are still deposited and subsequently removed.

The wafer is lithographically patterned for a 0.5 µm thick layer of metal. The metal is then deposited and patterned using lift-off, see Figure 5.6. The metal layer is used for bonding pads, and also serves as the top-electrodes on the shutter of the electric field mill. After the wafer has been diced into chips, the structure can be

released by HF wet etching which removes the supporting layers of PSG. The released structure is illustrated in Figure 5.7.

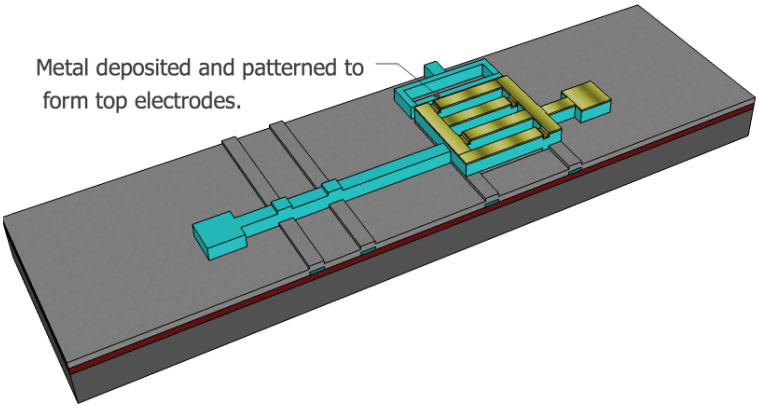


Figure 5.6: *A metal layer is deposited and patterned using lift-off.*

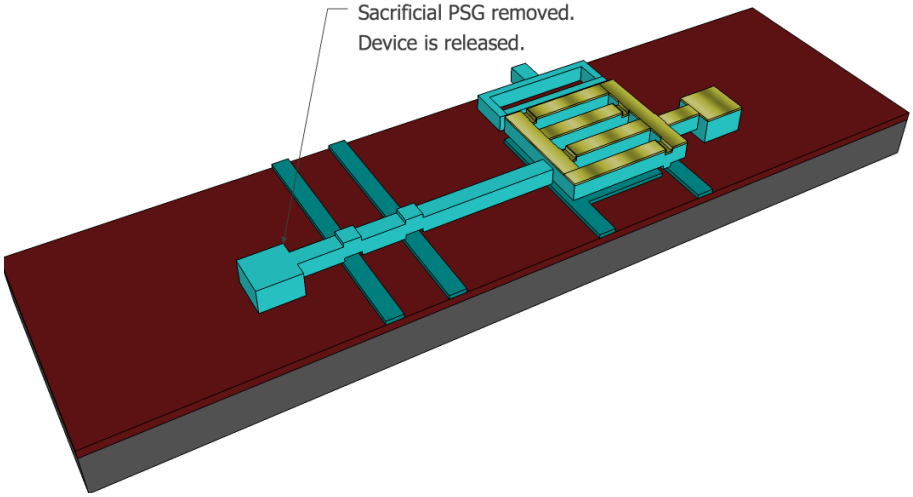
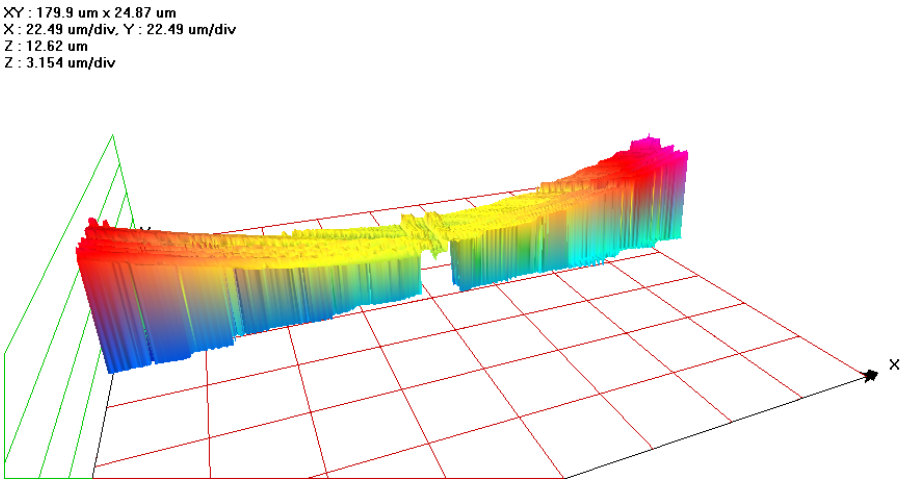


Figure 5.7: *The final poly-silicon layer is deposited and patterned.*

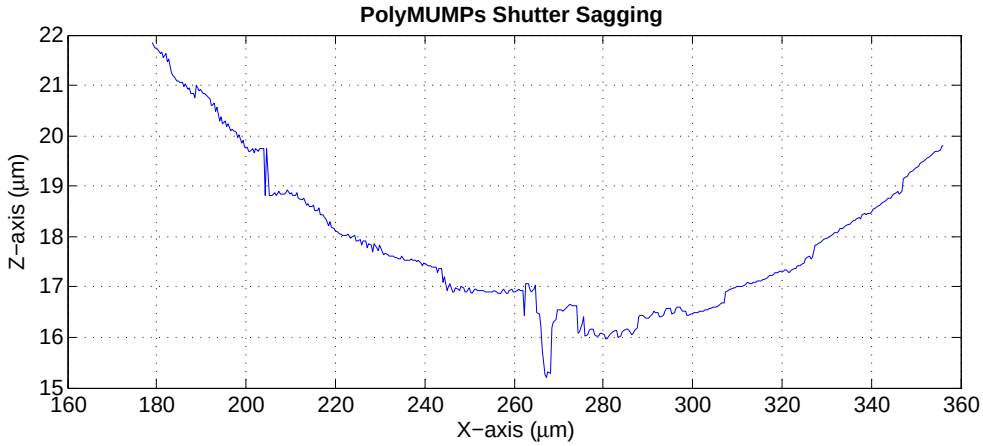
5.2.1 Failure Analysis of PolyMUMPS Sensors

The electric field sensors fabricated were based on the designs provided by the genetic algorithm, similar to the designs provided in Table 3.3. To meet the PolyMUMPs design rules, a few changes were required, such as increasing the width of the connecting spring and shutter springs. However, the main difference between the fabricated PolyMUMPs design and the sensor design optimized for the MicraGEM process is the thickness of the device layer. In the MicraGEM process, the device layer is relatively thick at 10 μm whereas in the PolyMUMPs process the thickness was that of the first device layer, only 2 μm. The second device layer was not used since it must be 2 μm smaller than the device layer below to allow for slight miss-alignments in the process.

After testing, it was found that the fabricated PolyMUMPs devices did not show any measurable displacement of the shutter when the thermal actuators were activated. A Fogale Photomap 3D optical profiler [50] was used to analyze the structure of the fabricated sensors. Figure 5.8 shows a cross section of the shutter of a PolyMUMPs electric field mill. Figure 5.8(a) shows a 3D representation of the shutter; sagging in the centre and increasing in height as it moves outwards towards the supporting springs and levers. Figure 5.8(b) shows the height along a horizontal cut-line drawn across the top of the shutter, indicating that the centre of the shutter is sagging as much as 5 μm.



(a)



(b)

Figure 5.8: *A 3D cross section of a PolyMUMPs sensor's shutter. Shown in (a), a 3D optical profile of the field mill shutter fabricated using the PolyMUMPs process. The profile of a cut-line across the top of the shutter is shown in (b).*

Figure 5.9 shows the 3D scan of the PolyMUMPs actuator before (a) and after (b) applying a 0.7 V actuation voltage. Both 3D profiles were first flattened using the Fogale proprietary software [50] to remove slight tilt of the substrate in the package. The darker colours in the 3D profiles indicate a larger out-of-plane displacement. Apparent in Figure 5.9(b) is a slight vertical displacement at the centre of the thermal actuator. The vertical displacement is slightly higher at the far end of the thermal actuator, as would be expected, where it is not connected to the lever structure.

This indicates that the actuator is moving vertically rather than displacing the lever structure horizontally in-plane.

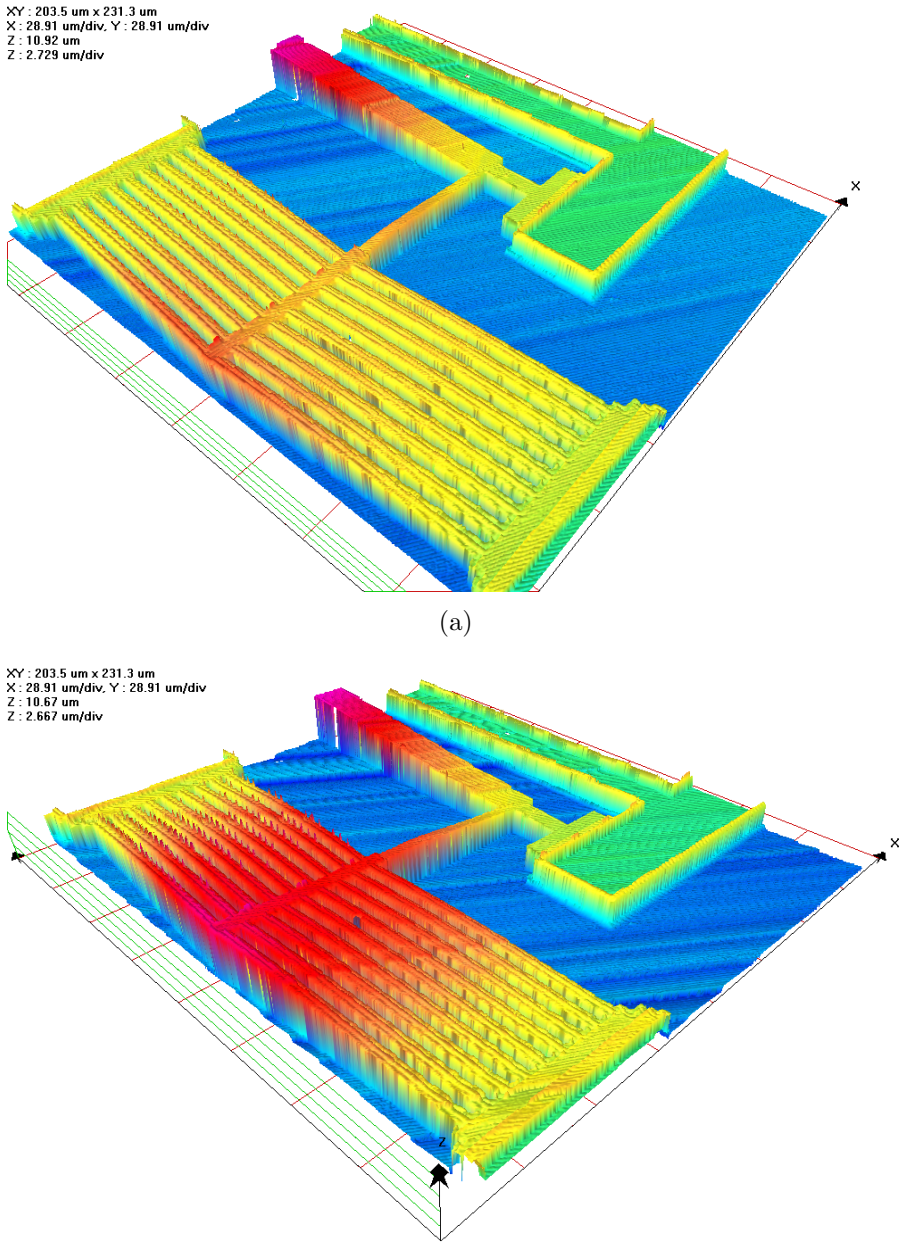
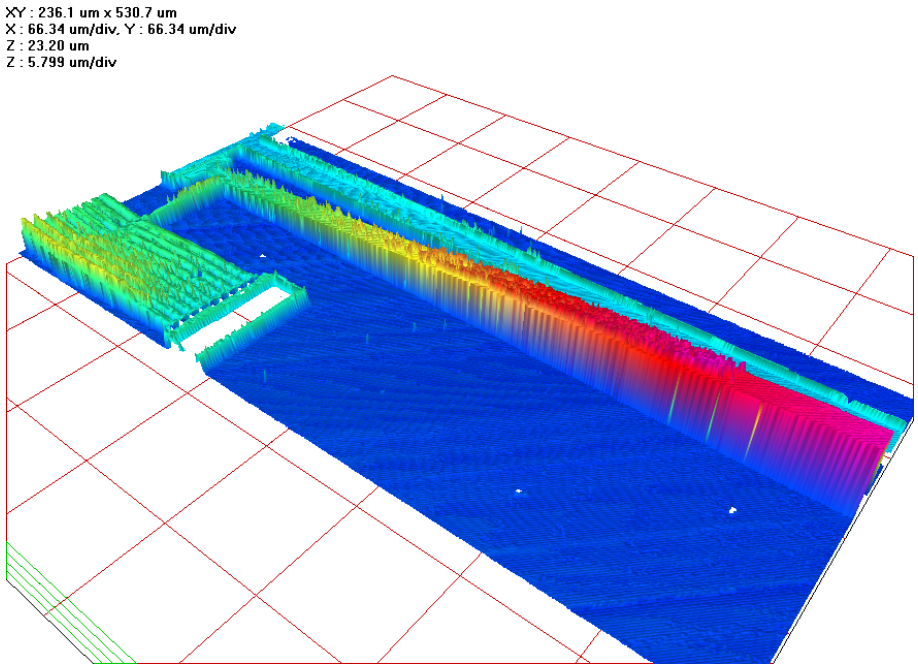
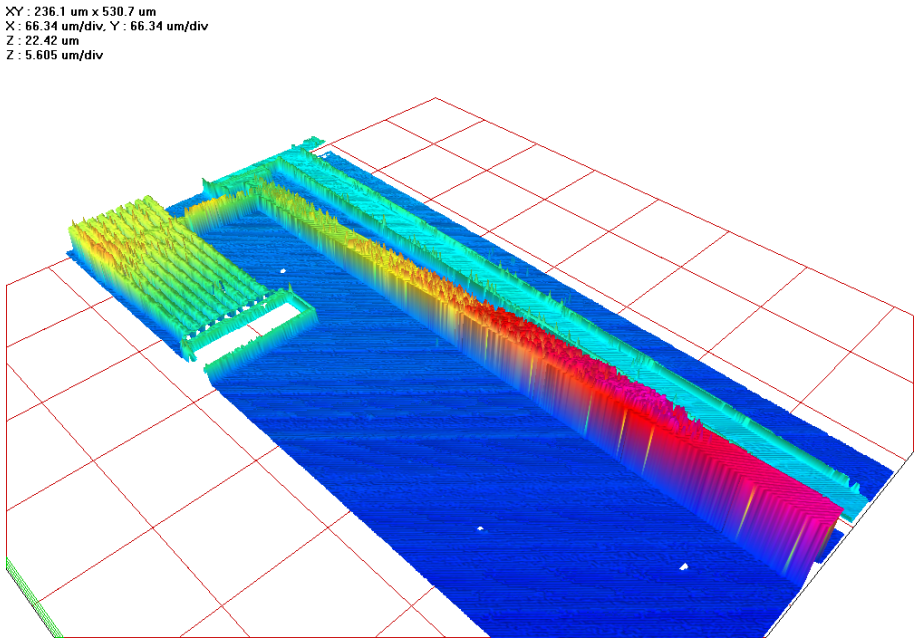


Figure 5.9: *The PolyMUMPs sensor actuator before (a), and after (b) applying a* 0.7 V *actuation voltage.*

Figures 5.10 - 5.12 shows the 3D profiles of the lever, shutter, and springs of the same device before and after applying a 0.7 V actuation voltage. Apart from the thermal actuator, there is no apparent difference in the displacement of the lever, shutter, or springs when comparing the optical scans. The shutter is not being displaced as expected because the actuators are moving out-of-plane. In the MicraGEM process, the large 10 μm thickness of the device layer, and also the fact that it is single crystal silicon rather than polymorphous silicon gives it a large out-of-plane stiffness. However, because the PolyMUMPs device layer is only 2 μm thick and is made from polymorphous silicon, it has a lower out-of-plane stiffness resulting in the sag of the shutter, and the out-of-plane displacement of the thermal actuators.

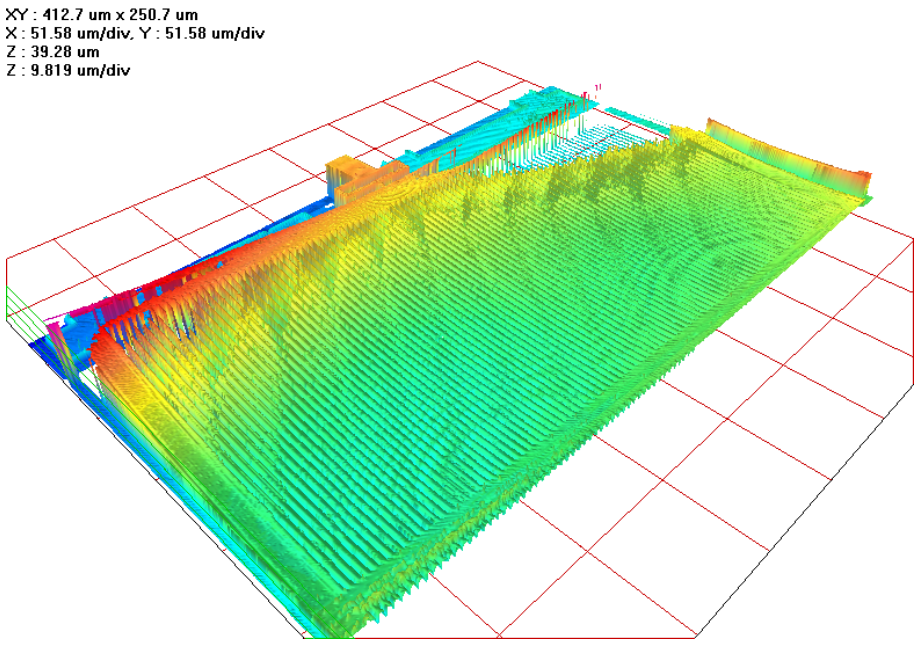


(a)

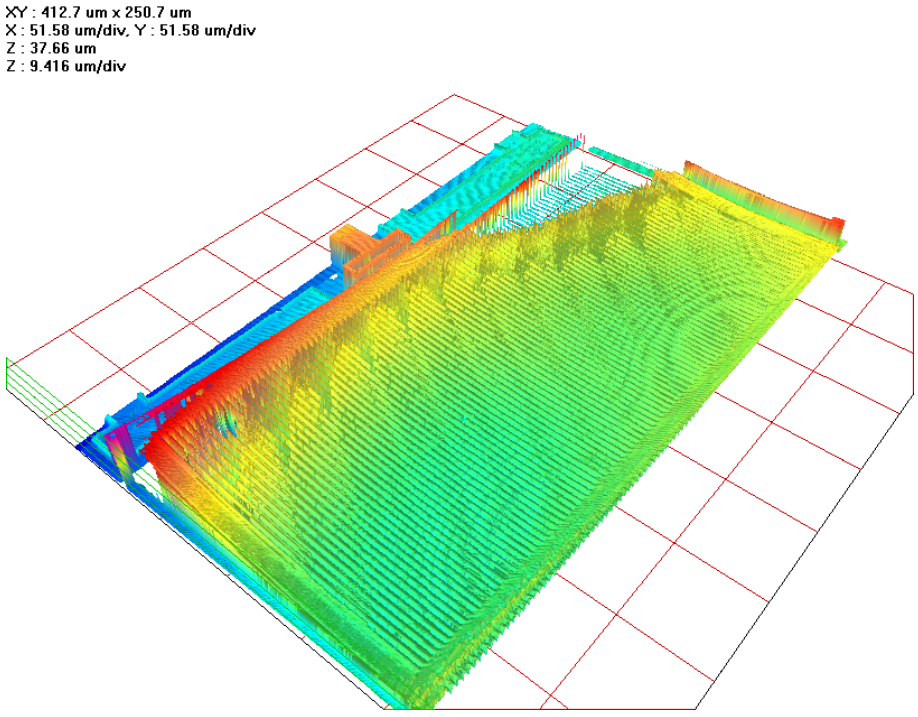


(b)

Figure 5.10: *The PolyMUMPs sensor's lever structure before (a), and after (b) applying a* 0.7 V *actuation voltage.*

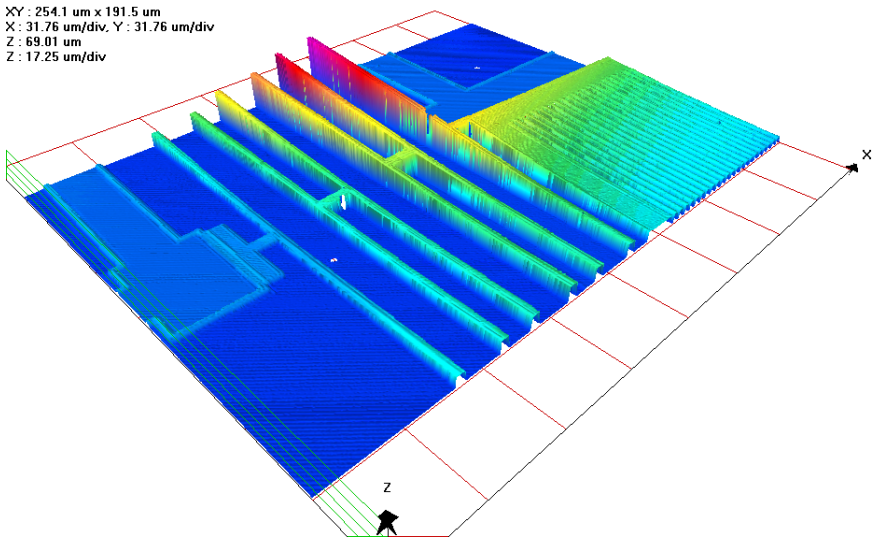


(a)

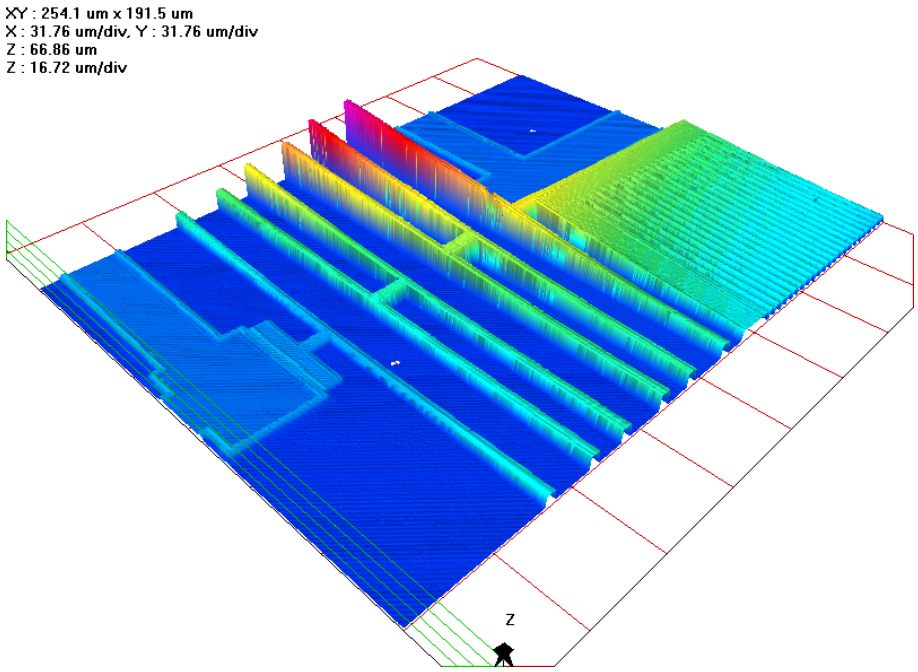


(b)

Figure 5.11: *The PolyMUMPs sensor shutter before (a), and after (b) applying a* 0.7 V *actuation voltage.*



(a)



(b)

Figure 5.12: *The PolyMUMPs sensor springs before (a), and after (b) applying a* 0.7 V *actuation voltage.*

5.3 MicraGEM

The MicraGEM process is a silicon-on-glass fabrication process originally offered for academic use by the Canadian Microelectronics Corporation in collaboration with Micralyne, Inc [12]. The process begins with a 7740 Pyrex wafer 525 µm thick. The Pyrex is etched using two masks at two depths; 2 µm and 10 µm, see Figure 5.13. Next, a metal layer is patterned using a lift-off technique. The metal layer consists of 50 nm titanium, 50 nm platinum, followed by 200 nm of gold [12]. The metal layer may be deposited in the etched pit or on the surface of the Pyrex, as illustrated in Figure 5.14.

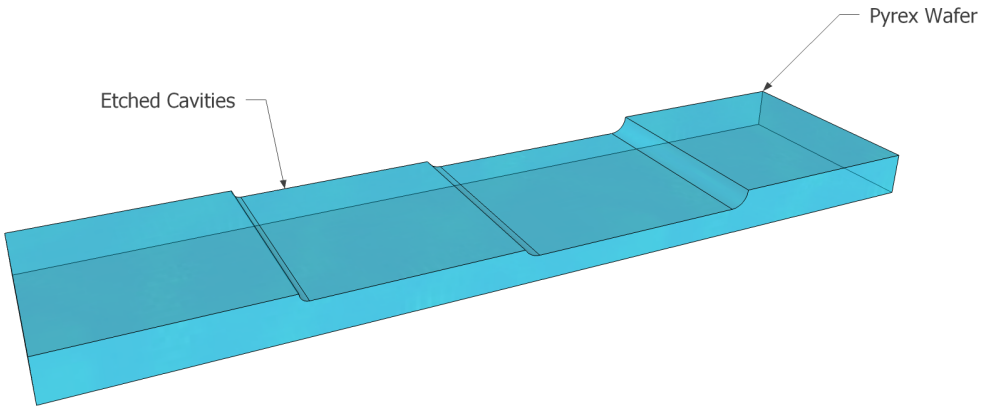


Figure 5.13: *An etched Pyrex wafer.*

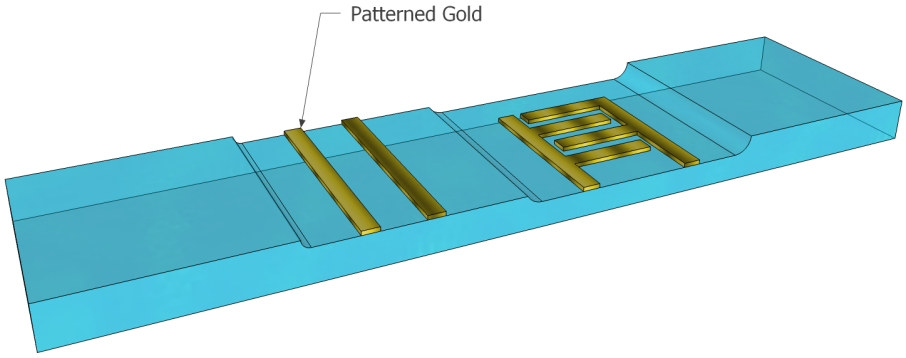


Figure 5.14: *Gold metal is deposited using a lift-off technique.*

At this stage, a silicon-on-insulator (SOI) wafer is anodically bonded to the Pyrex. Anodic bonding is discussed in greater details in Chapter 6. The SOI wafer used has a 10 µm single crystal silicon device layer, followed by a buried oxide and 525 µm silicon handle wafer, illustrated in Figure 5.15. Next, the handle wafer and buried oxide are removed by wet etching, as in Figure 5.16. A second metal layer consisting of approximately 10 nm chrome and 75 nm of gold is deposited and lithographically patterned on top of the silicon device layer, see Figure 5.17. In the electric field mill design, this metal layer is used as a conductor for the thermal actuators and as the grounding top-electrodes on the shutter structure. Finally, the wafer is lithographically patterned and etched using a specially designed anisotropic plasma etch to form the device and release it, Figure 5.18.

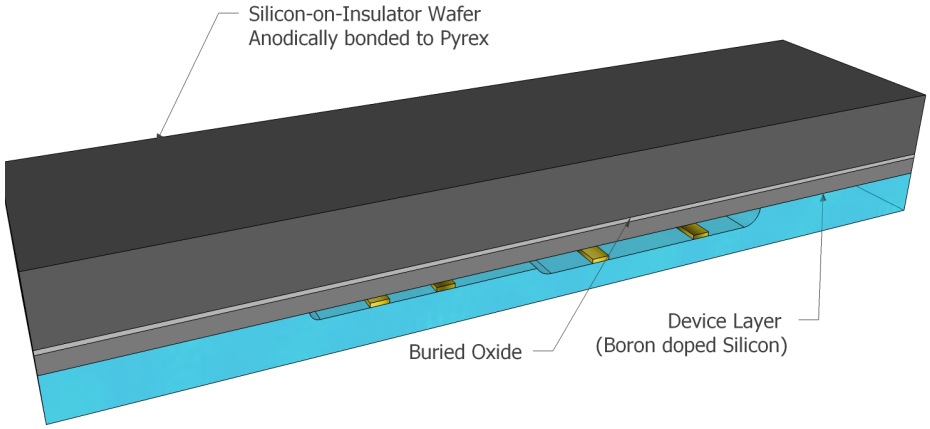


Figure 5.15: *A silicon-on-insulator wafer is anodically bonded to the Pyrex.*

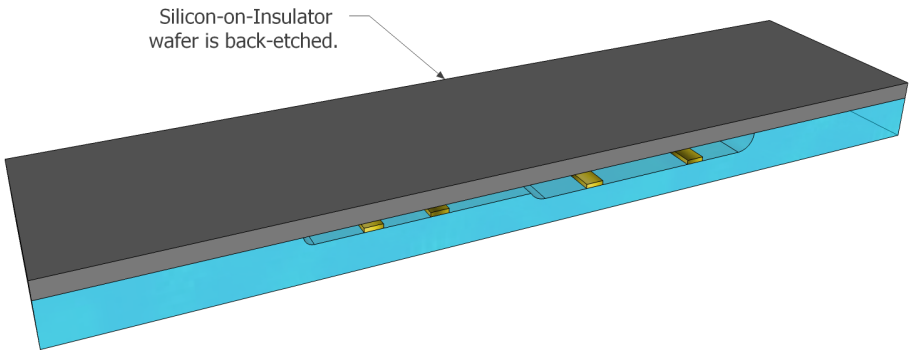


Figure 5.16: *Silicon-on-insulator wafer is back-etched, removing the handle wafer and buried oxide.*

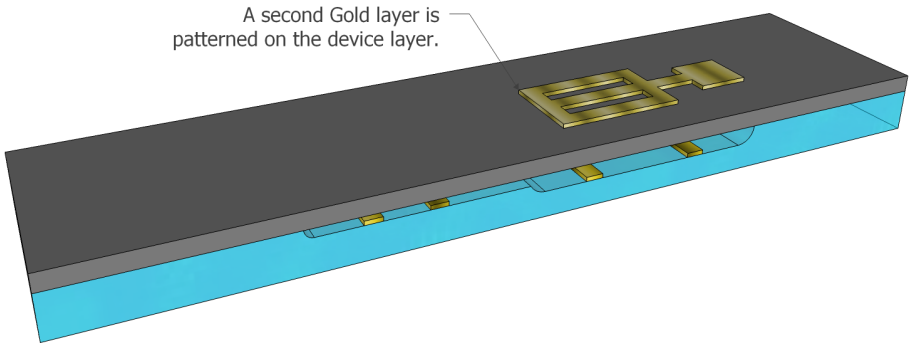


Figure 5.17: *Metal is deposited on top of the device layer, forming the top-electrodes and actuator traces.*

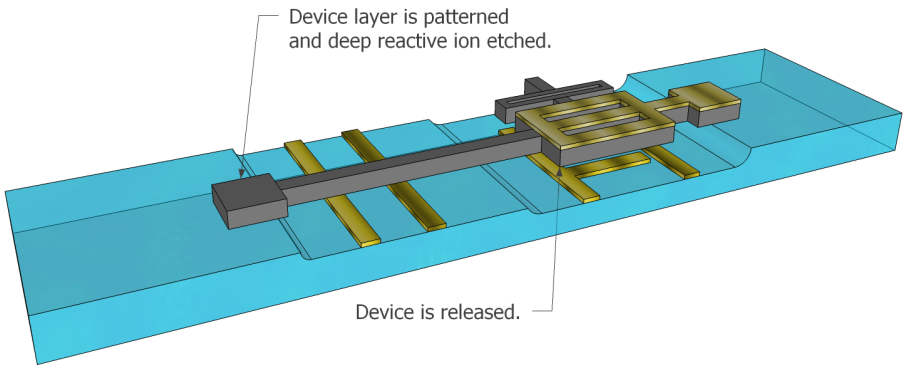


Figure 5.18: *The device layer is patterned and etched, releasing the completed structure.*

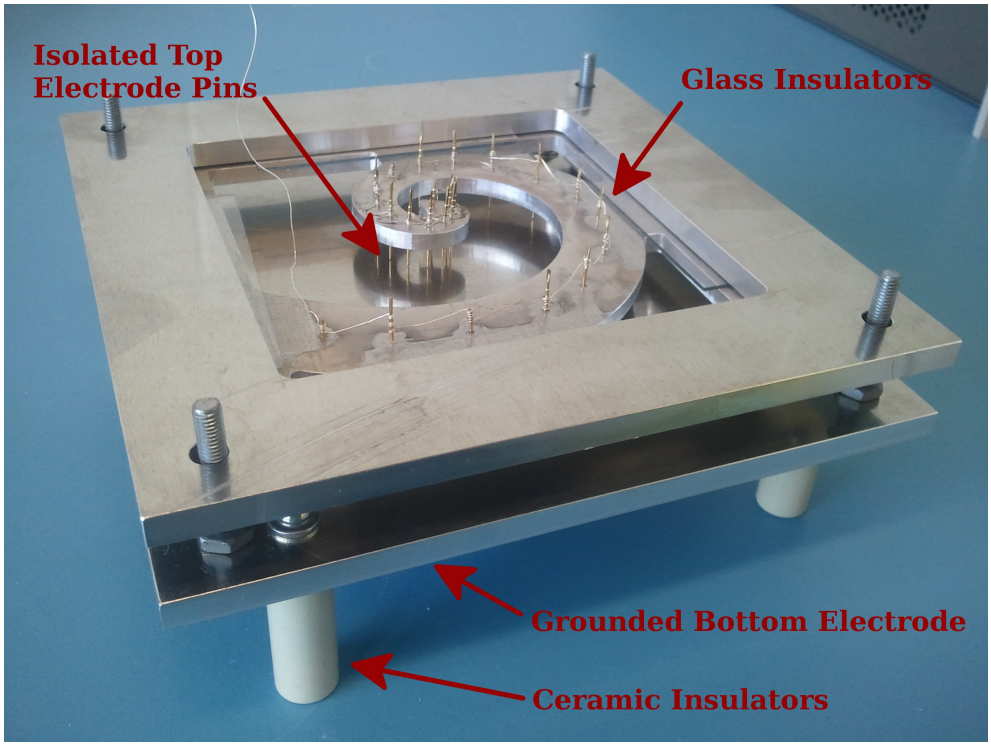
The MicraGEM process has a few advantages over the PolyMUMPs process for the electric field mill sensor design. Firstly, and most importantly, it offers a relatively thick device layer of single crystal silicon which has a sufficient out-of-plane stiffness for the electric field mill. This prevents the actuators from displacing vertically, as in the sensors manufactured using the PolyMUMPs process. The process is also quite a bit simpler than the PolyMUMPs process, making use of only 5 masks compared to the 8 masks of PolyMUMPs. However, the MicraGEM process may be more expensive since it requires the use of an SOI wafer.

Chapter 6

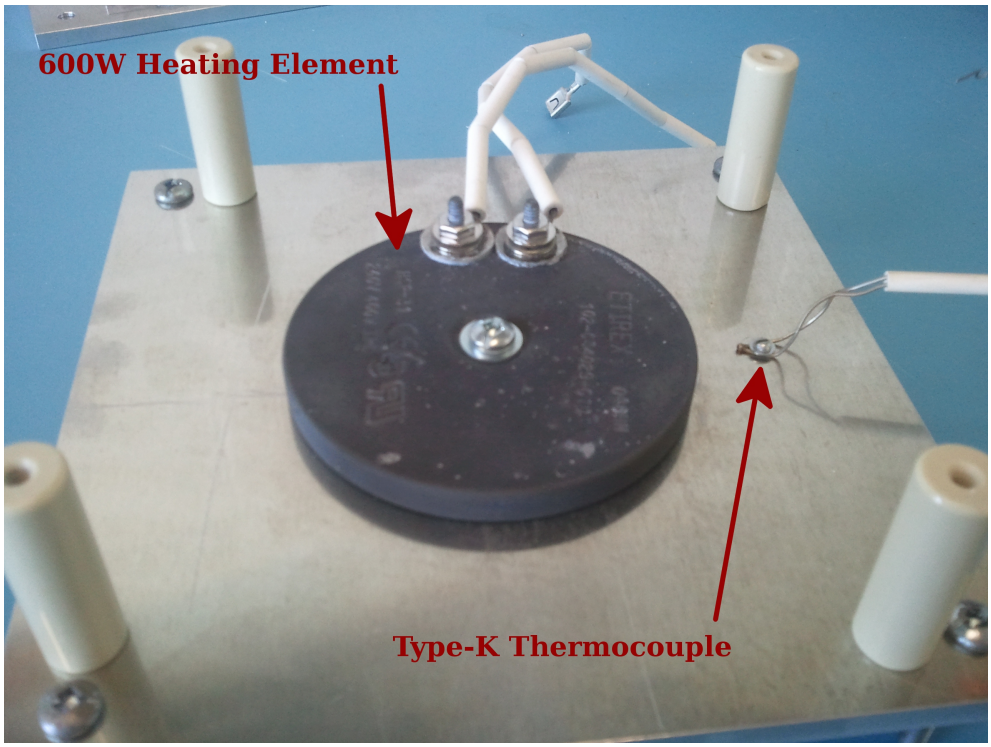
Anodic Bonding Apparatus

6.1 Introduction

Anodic bonding is a mechanism used to permanently bond a glass to a conducting substrate such as silicon. The process may also be known as field-assisted thermal bonding or electrostatic bonding. The glass used in anodic bonding to silicon is commonly Corning #7740 (Pyrex), but other sodium rich glasses such as soda lime #0080, potash soda lead #0120, and aluminosilicate #1720 can also be used [52]. To reduce thermal stress in the bond, glass with a thermal expansion constant similar to the substrate wafer should be used [14]. Bonding is achieved by bringing a cleaned Si wafer into contact with the desired glass wafer. The bonding will work if the silicon has a thin oxide, but it has been found that a stronger bond can be achieved if the oxide is first removed [52, 14]. First, the wafer stack is heated to typically $200 - 500\,^{\circ}\mathrm{C}$. Next, a dc voltage is applied across the wafer stack typically in the range of $100 - 1000\,\mathrm{V}$ through a point electrode on the glass and a ground plate beneath the silicon wafer. Figure 6.1 shows a photograph of the anodic bonding apparatus developed for this thesis.



(a)



(b)

Figure 6.1: *Anodic Bonding Apparatus.*

The bonding process centres around sodium ions in the glass [52, 53, 14] . At the elevated temperatures, the sodium ions become mobile within the glass and migrate towards the cathode, creating a space-charge region at the interface. The voltage drop occurs primarily across this small space-charge region causing a very large electrostatic force that pulls the glass into intimate contact with the silicon wafer, where the elevated temperature assists the formation of covalent bonds between the glass and the silicon surface [52]. As sodium ions are pulled from the site of bonding it leaves behind oxygen ions which are pulled towards the silicon surface where it forms an oxide and becomes non-conducting causing a bonding front to expand outwards from the point electrode [54].

The expanding bonding front slows the farther it travels from the point electrode due to the increased area of the space charge region. Depending on bonding parameters such as voltage, temperature and the exact types of materials being bonded, the bond may take a long time to complete.

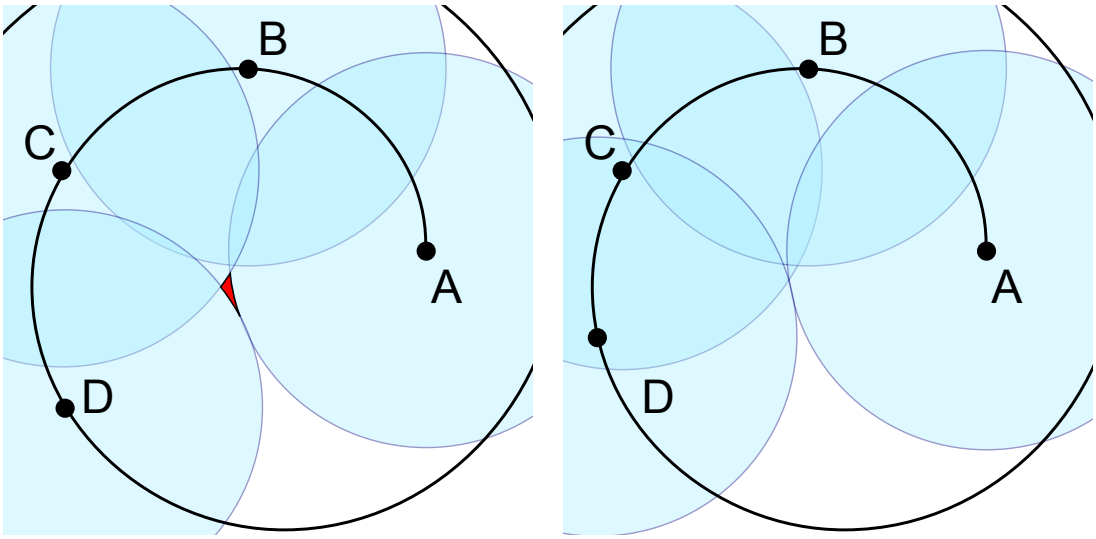
6.1.1 Spiral Electrode Array

The elevated temperature over an extended time period may cause damage to microstructures present on the wafers. In order to reduce the time required to complete a bond and to improve bond quality, it is possible to use multiple point electrodes in a spiral arrangement [55]. A spiral is mathematically designed so that the bonding fronts from adjacent electrodes meet at the same time, ensuring that any gasses have a way to escape the bonding without becoming trapped beneath the Pyrex glass.

To calculate which points along the spiral to select, a polar coordinate system is used. The spiral follows the following equation from [55]:

$$r(\theta) = K_r(1 + K_a\theta)\theta \tag{6.1}$$

where K_r and K_a are parameters that determine the shape of the spiral. The initial parameters used were $K_r = 1$ and $K_a = 10$. The first point added is at $\theta = 2\pi$. This point is used since a traditional spiral infinitely approaches the origin but never reaches it. The second point along the spiral was taken at $\theta = \frac{3}{2}\pi$. Additional points must be located along the spiral such that the circular bonding fronts of each of the three points intersects so that no gas between the glass and silicon could become trapped. Consider Figure 6.2 which shows a portion of a spiral, and the four points A, B, C, and D. Point A is the first electrode pin, located at $\theta = 2\pi$. Point B is the second electrode pin, located at $\theta = \frac{3}{2}\pi$. Points C and D are located at potential electrode positions along the spiral. As highlighted in red in Figure 6.2(a), a small space exists between the bonding fronts that could trap gas beneath the surface of the glass, resulting in an unbonded region. This can be adjusted by moving electrode D farther back along the spiral so that the bonding fronts from electrodes A, C, and D intersect at the same point, as illustrated in Figure 6.2(b).



(a) (b)

Figure 6.2: *Spiral electrode geometry calculation. (a) shows a bad electrode geometry that could trap gas beneath the glass during anodic bonding. (b) shows the same geometry, with the position of electrode D adjusted to eliminate the trapped gas problem.*

All electrode points along the spiral must be chosen in this manner to prevent

trapped gasses. An electrode arrangement was designed in this manner using MATLAB [44] and fabricated into an aluminum plate with holes positioned at each of the point electrodes. A test-pin receptacle is epoxied into each hole using a high vacuum compatible leak sealant epoxy. The pin receptacles can then be selectively loaded with gold plated spring pins that are pressed against the surface of the Pyrex glass to be bonded. Figure 6.3 shows the geometry of the electrode plate. The fabricated plate is visible in the photograph of Figure 6.1(a). Areas of the electrode plate are milled out so that the samples are visible through the top window of the vacuum chamber during bonding.

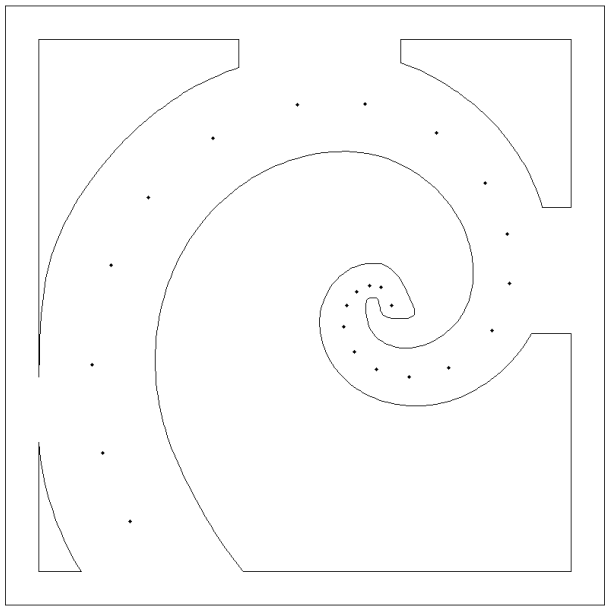


Figure 6.3: *Electrode plate fabrication drawing.*

6.1.2 Heating Controller

The anodic bonding apparatus requires a controlled heater to maintain the temperature under vacuum when bonding. The controller that was designed and built is pictured in Figure 6.4. The schematic for the controller is available in Appendix 7.2. The controller has a high gain rail-to-rail op-amp for sampling the small voltage generated by the type-k thermocouple. The op-amp, in a non-inverting amplifier

configuration, provides a gain of approximately 258. By design, the op-amp uses the negative thermocouple lead as a reference instead of ground which requires that the hot thermocouple junction be grounded. This is beneficial since it allows the aluminum plate in the chamber to be used directly as the ground plate for anodic bonding without the need for electrical isolation between the thermocouple and the ground plate. The heater controller features an ambient temperature sensor used to compensate for the thermocouples cold-junction temperature. There is a USB connection used for logging temperature and control values as well as a liquid crystal display (LCD) which displays the current temperature, set-point temperature, ambient temperature, and displays a small line graph of the temperature over the past 20 minutes of operation. For further details on the design and operation of the heater controller, please see Appendix 7.2

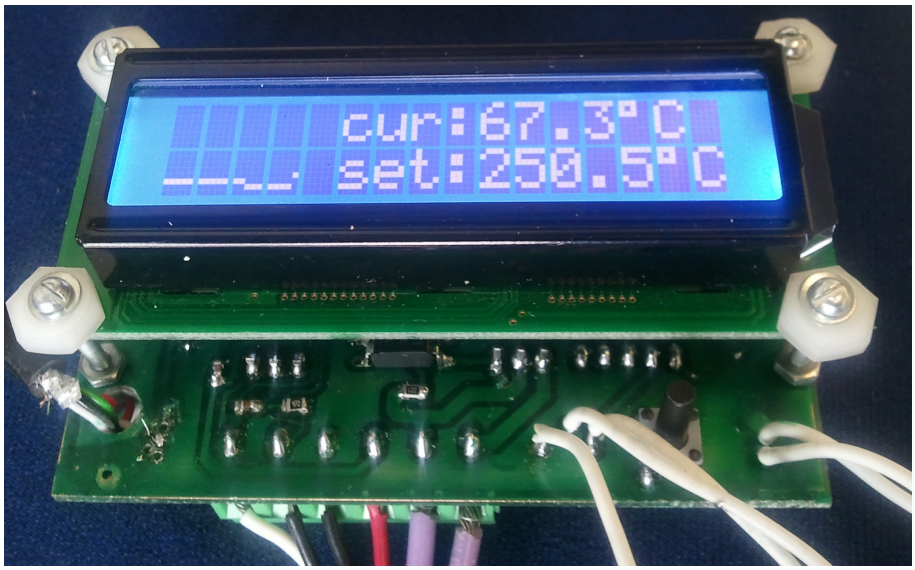


Figure 6.4: *Heater Controller*

6.1.3 Anodic Bonding Results

With the heater controller built and functioning, the anodic bonding apparatus was tested by doing a series of bonds using 3" p-type silicon wafers and 2" squares

of 0.5 mm thickness Pyrex glass. The glass and wafer samples were cleaned for five minutes using a mixture of deionized water and Sparkleen detergent [61] applied with a sponge in swirling motions. After applying the soap mixture, the samples are rinsed in deionized water for 3 minutes. Some of the samples were additionally rinsed with acetone, then Isopropyl Alcohol (IPA), followed by Methanol, and again with deionized water to remove any residues that may have been left by the soap.

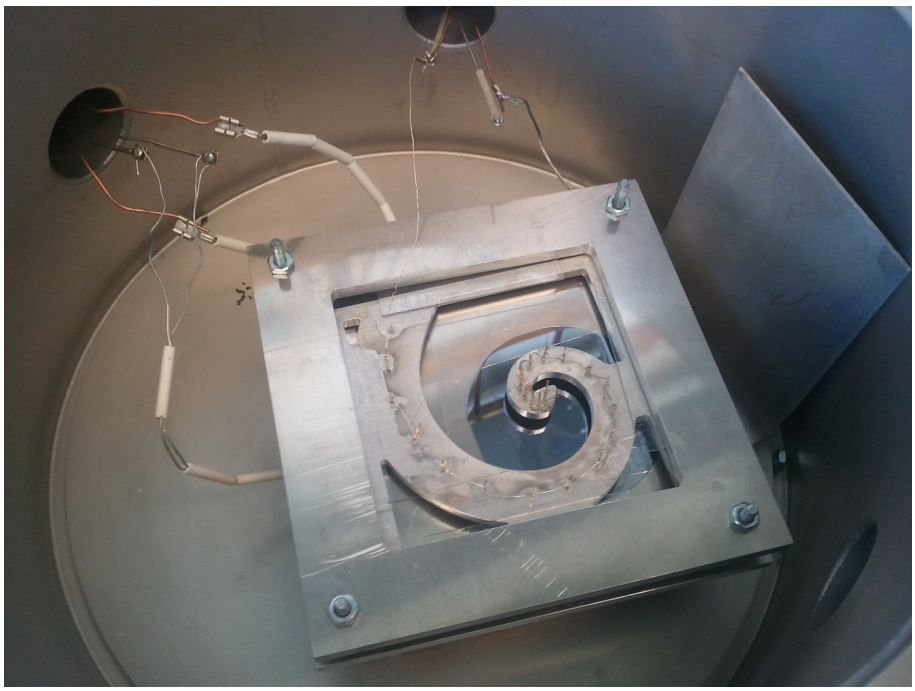
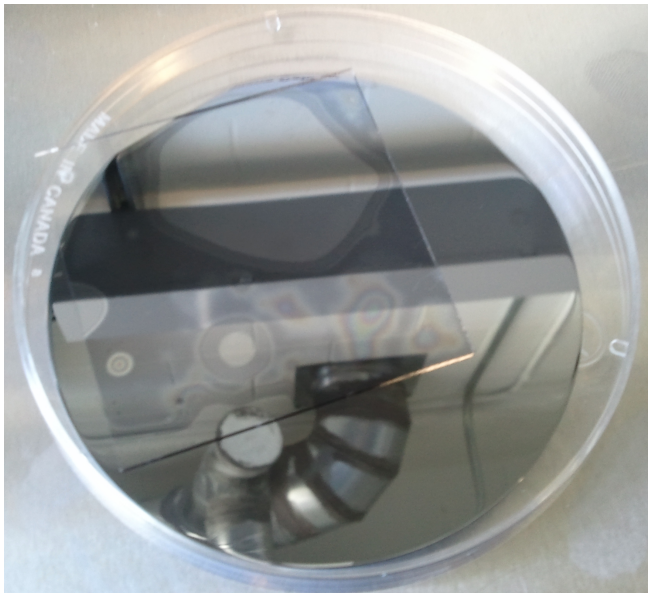


Figure 6.5: *Bonding apparatus setup inside of vacuum chamber with wafer/Pyrex samples positioned.*

After cleaning and drying with dry nitrogen, the Pyrex is placed on top of the silicon wafer and slightly pressed together. Figure 6.6(a) shows a photograph of the first samples cleaned and pressed together prior to bonding. The wafers are then placed inside the bonding chamber as pictured in to Figure 6.5, the top plate of the bonding apparatus is tightened by hand so that the top electrode pins are slightly depressed against the Pyrex. A multimeter is used to test each vacuum feed through to assure that the high-voltage connection is not shorted to ground and that the heater power lines are not shorted to ground or each other. The chamber is

then pumped down with a roughing pump and the heater is activated with a set-point temperature of 250 °C. Note that the bonding chamber also has a turbo pump available, however it was not used for these preliminary tests. After the temperature has reached 250 °C, the high voltage power supply (Stanford Research Systems, Inc. model PS310/1250V-25W) is activated and the applied voltage is gradually ramped up in 100 V increments to 1000 V. The observed current at $1000V$ is typically between $30 - 80\mu A$. The voltage is applied while monitoring the current which after a few minutes drops by approximately $10 - 20\,\mu$A. Once the current is stable, the 1000 V is applied for an additional 10 minutes after which the high voltage power supply is turned off. The chamber is then vented and allowed to cool back to room temperature before the samples are removed. Figure 6.6(b) shows the results of the first samples bonded using this process. This sample was not rinsed with acetone/IPA/methanol as described above. Note the clearly visible diffraction rings present on the bonded samples, each of which has a particle of dust at its centre. This dust is present because the samples were not cleaned inside a particle controlled cleanroom and the wafer was allowed to sit while the Pyrex was cleaned.

Figures 6.7(a) and 6.7(b) show two more samples that were cleaned inside the cleanroom and rinsed with acetone/IPA/methanol which shows considerable improvement in bonded area, however a few dust particles were still present. Also notice that both bonded pairs have an unbonded region along one edge of the Pyrex wafer; this region corresponds to the side of the wafer that was gripped with metal wafer tongs to handle the wafers during and after cleaning. It is likely that the wafer tongs left a residue during the cleaning process due to insufficient rinsing under and around the tongs. It is also possible that the metal tongs leave small scratches or imperfections on the glass which could also prevent bonding in that area. Using a piece of Pyrex glass larger than the silicon wafer may reduce or eliminate the unbonded region caused by the tongs. The samples of Figure 6.7(a) also has a crack along one edge of the Pyrex



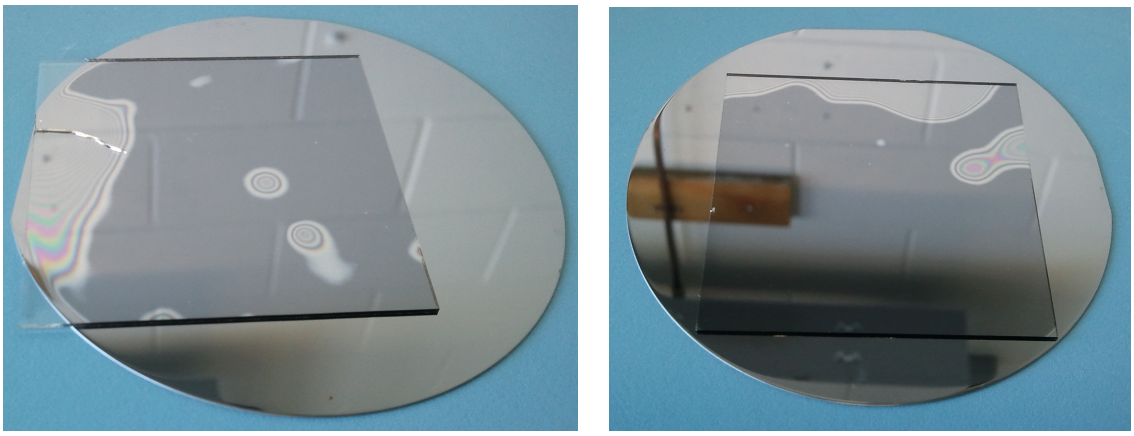
(a)

(b)

Figure 6.6: *Pyrex and silicon wafer samples before (a) and after (b) anodic bonding.*

caused possibly by thermal shock from being cooled too quickly as the samples were removed before fully cooling.

Figure 6.8 shows a final pair of samples that was cleaned inside the cleanroom with a slightly different cleaning process. After cleaning the Pyrex sample with soap solution for five minutes, it is rinsed for three minutes with deionized water, then quickly rinsed with acetone, IPA, and then methanol to remove any residues. The



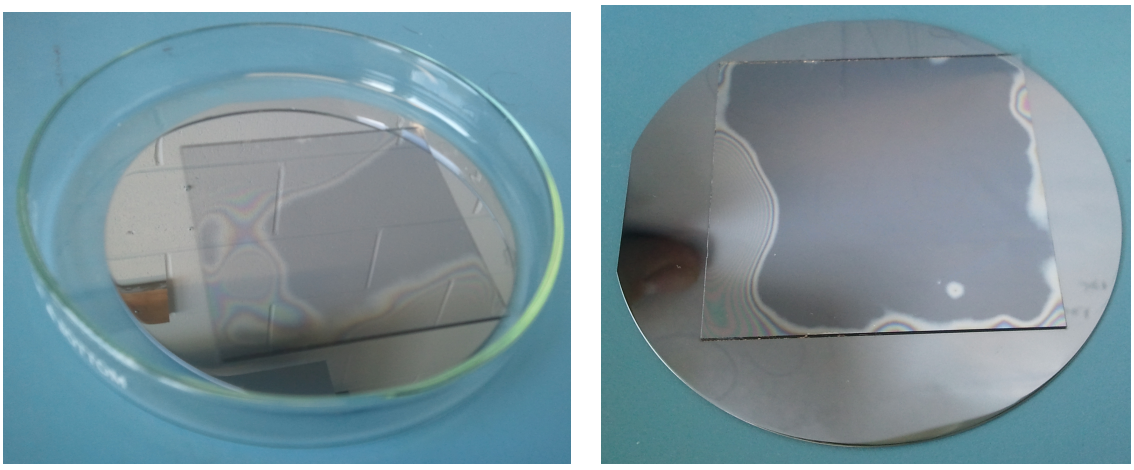
(a) (b)

Figure 6.7: *Two more Pyrex and silicon samples after anodic bonding. Note the improved bond area with fewer dust particles.*

wafer is then lightly rinsed again and then submerged in deionized water while the silicon wafer undergoes the same cleaning process. This prevents any dust from settling on the Pyrex while the silicon is cleaned. Care must be taken to maintain the orientation of the Pyrex wafer when submerged so that the cleaned side is bonded. After the silicon wafer has been cleaned and dried with dry nitrogen the Pyrex sample is removed from the water, lightly rinsed and dried with nitrogen before being placed on top of the silicon wafer for bonding.

6.2 Difficulties with Anodic Bonding

During the design process of the anodic bonding apparatus, several problems were encountered. The first major problem was with the thermocouple, it was not sufficiently attached to the aluminum bottom plate. When the vacuum was applied and the heater turned on, the poor connection between the thermocouple and the plate caused a low reading on the thermocouple which caused the controller to loose control. This out-of-control heating melted the aluminum plate and Pyrex sample. The solution was to tightly screw the thermocouple into the aluminum plate to ensure



(a) (b)

Figure 6.8: *A final wafer/Pyrex pair before (a) and after (b) bonding. Samples were cleaned in the cleanroom with Acetone/IPA/Methanol rinse. The Pyrex sample was kept submerged in deionized water while the silicon wafer was cleaned.*

adequate thermal connection.

The next major problem encountered was with the heating element itself. The first revision of the design used a ceramic clad heating element, similar to the type found in many consumer hot-plates and stoves. The element was securely bolted to the plate using machined aluminum straps. The porous ceramic cladding of the element is a poor thermal conductor by itself, and the primary heating mechanism is convection of air in and around the ceramic cladding of the element to the aluminum plate above. The effect of this is that when in vacuum, the heating element may get very hot, but it is unable to conduct enough heat into the aluminum plate, resulting in a temperature rise of only $\sim 20 - 30\,^{\circ}\mathrm{C}$ after being heated for about a minute. If the element is then deactivated and the chamber vented, the air allows the element to conduct to the plate and the temperature of the plate almost immediately jumps to over $100\,^{\circ}\mathrm{C}$. In an attempt to solve this problem, a high temperature vacuum compatible two-part epoxy, EPO-TEK H74F [62], was used to bond the element directly to the aluminum plate, with the hopes that the thermally conductive epoxy would adequately transfer the heat from the element to the plate above. The epoxy

datasheet [62] reports a high working temperature of 250 °C with an intermittent temperature as high as 350 °C. It was hoped that if the aluminum plate temperature was limited to 250 °C, the epoxy would be suitable. However, it seems that the element would become significantly hotter than the degradation temperature of the epoxy, 486 °C, resulting in the degradation of the epoxy. As the epoxy degrades in vacuum, the thermal contact between the element and the aluminum plate is also degraded. This degrading of the epoxy causes a positive feedback and quickly results in the total failure of the epoxy. This problem was solved by substituting an industrial heating element that is clad in steel instead of ceramic, and is simply bolted to the aluminum plate. In this case, the conductivity of the steel cladding is sufficient to heat the aluminum plate.

Chapter 7

Conclusion

7.1 What has been accomplished?

During the research of this thesis, genetic algorithms and their application to MEMS design has been examined and implemented. A multi-objective niched Pareto genetic algorithm has been designed and applied to the design optimization of a complex micromachined electric field mill. The primary purpose of optimization of the electric field mill was to eliminate the requirement of resonant operation of the sensor. The algorithm evaluates designs based on the results of finite element simulations performed using the COMSOL Multiphysics software. The simulations are validated using measurements taken from previously fabricated electric field mill sensors and thermal actuators designed by others. The algorithm evolves a population of sensor designs towards the Pareto frontier of optimal solutions by utilizing crossover, mutation, and niched Pareto tournament selection operators. Design compromises can then be selected from the optimized sensor population to meet multiple criteria, such as high displacement, high frequency, or low stress.

There have been other evolutionary algorithms for design and optimization of MEM structures, however, most of these algorithms are single objective, relatively

simple, or not evaluated using a finite element model. Others who have designed multi-objective optimization algorithms for complex MEMS have required human expert interaction to help guide the algorithm, which is not required by the algorithm presented here. The niched Pareto approach also differs from other genetic algorithms in that the solution set converges not to a single "best" solution, but rather returns a set of non-dominated solutions that approximate the Pareto front. Applying the genetic algorithm only to the optimization as opposed to design synthesis simplifies the search space requiring little additional input. Simulations performed include static, dynamic, and transient simulations which accurately model the desired behaviour of the sensors. Transient simulations had not been previously studied for the electric field mill design and have been used to examine the mechanical and thermal frequency response of the field mill structure with and without damping. These simulations are also compared to the frequency response obtained using an optical 3D profiler.

Several optimized designs were selected for fabrication using the PolyMUMPs process. However, the resulting devices failed to function because of the reduced stiffness of the thin device layer. The failure mode of the fabricated devices has been analyzed. The original sensor was designed for the MicraGEM process, which at the time of writing is no longer available for academic use. In order to fabricate the sensors, work has been started towards designing an in-house fabrication process based off the MicraGEM process. To that end, an anodic bonding apparatus has been designed, built, and tested. The anodic bonding apparatus is intended for use in the in-house MicraGEM process.

7.2 What else could be done?

There are many avenues available for future research on the topic of automated MEMS design optimization. The niched Pareto genetic algorithm presented here

could be applied to many other devices and MEM structures. Analysis could be done to determine the optimal genetic algorithm parameters for use with various types of MEMS optimization problems. New objectives can be added to the list of objectives currently studied by the algorithm. A more in-depth investigation into the affects of population size and rates of mutation on convergence as well as the use of other more complex genetic operators such as multi-point crossover also warrants further investigation. It would be valuable to study other types of evolutionary algorithms and their application to MEMS, such as particle swarm optimization, seeker optimization, and tabu search optimization, to name a few.

The optimized micromachined electric field mill designed here has yet to be fabricated, tested, and characterized. Once the sensor has been fabricated, future work could focus around further optimization of the sensor design. Research could also be done to integrate drive and sense electronics into the sensor package.

References

[1] J. N. Chubb, "Two new designs of field mill type fieldmeters not requiring earthing of rotating chopper," *IEEE Transactions on Industry Applications*, vol. 26, no. 6, pp. 1178–1181, Nov. 1990.

[2] P. S. Maruvada and R. D. Dallaire, "Development of field-mill instruments for ground-level and above-ground electric field measurement under hvdc transmission lines," *Power Apparatus and Systems, IEEE Transactions on*, vol. 102, pp. 738–744, 1983.

[3] C. Peng, X. Chen, Q. Bai, L. Luo, and S. Xia, "A novel high performance micromechanical resonant electrostatic field sensor used in atmospheric electric field detection," in *MEMS 2006*, Jan. 2006, pp. 698–701.

[4] M. N. Horenstein and P. R. Stone, "A micro-aperture electrostatic field mill based on mems technology," *Journal of Electrostatics*, vol. 51-52, pp. 515–521, 2001.

[5] G. Wijeweera, "Design of a micromachined electric field sensor using a thermal actuating shutter," Master's thesis, University of Manitoba, Dept. of Electrical and Computer Engineering, 2008.

[6] L. L. Chur, J. A. Hetrick, and Y. B. Gianchandani, "High amplification compliant microtransmissions for rectilinear electrothermal actuators," *Sensors and Actuators: A Physical*, vol. 97-98, pp. 776–783, Nov. 2002.

[7] R. Kamalian, H. Takagi, and A. M. Agogino1, "Optimized design of MEMS by evolutionary multi-objective optimization with interactive evolutionary computation," in *GECCO 2004, LNCS 3103*. Springer-Verlag Berline Heidelberg, 2004, pp. 1030–1041.

[8] P. Varotsos and K. Alexopoulos, "Physical properties of the variations of the electric field of the earth preceding earthquakes, i," *Tectonophysics*, vol. 110, pp. 73–98, 1984.

[9] Z. Cui, C. Gong, and S. Xia, "Design and modelling of a microsensor for atmospheric electric field measurement," in *Design, Test, Integration and Packaging of MEMS/MOEMS*, May 2003, pp. 154–158.

[10] X. Chen, C. Peng, H. Tao, C. Ye, Q. Bai, S. Chen, and S. Xia, "Thermally driven micro-electrostatic fieldmeter," *Sensors and Actuators A: Physical*, vol. 132, pp. 677–682, 2006.

[11] N. Kuwabara, K. Tajima, R. Koboyashi, and F. Amemiya, "Development and analysis of electric field sensor using $linbo_3$ optical modulator," *Electromagnetic Compatibility, IEEE Transactions on*, vol. 34, no. 4, pp. 391–396, Nov. 1992.

[12] C. M. Corporation, *Introduction to Micragem: A Silicon-on-Insulator Based Micromachining Process*, 3rd ed., Dec. 2004.

[13] J. D. Zook, D. W. Burns, H. Guckel, J. J. Sniegowski, R. L. Engelstad, and Z. Feng, "Characteristics of polysilicon resonant microbeams," *Sensors and Actuators A: Physical*, vol. 35, no. 1, pp. 51–59, Oct. 1992.

[14] G. T. A. Kovacs, *Micromachined Transducers Sourcebook*. McGraw-Hill, 1998.

[15] P. Schneider, A. Schneider, and P. Schwarz, "A modular approach for simulation-based optimization of mems," *Microelectronics Journal*, pp. 29–38, 2002.

[16] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, Apr. 1997.

[17] A. Ongkodjojo and F. E. H. Tay, "Global optimization and design for micro-electromechanical systems devices based on simulated annealing," *Journal of Micromechanics and Microengineering*, vol. 12, pp. 878–897, 2002.

[18] G. K. Fedder and T. Mukherjee, "Physical design for surface-micromachined mems," in *5th ACM/SIGDA Physical Design Workshop*, 1996, pp. 53–60.

[19] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4591, pp. 671–680, May 1983.

[20] K. A. D. Jong, *Evolutionary Computing: a unified approach.* Cambridge, MA 02142: The MIT Press, 2006.

[21] H. Li and K. K. Antonsson, "Evolutionary techniques in mems synthesis," in *Proceedings of ASME Design Engineering Technical Conferences*, Sep. 1998.

[22] S. Xiao, B.-Z. Wang, X.-S. Yang, and G. Wang, "Reconfigurable microstrip antenna design based on genetic algorithm," in *Antennas and Propogation Society International Symposium*, vol. 1. IEEE, 2003, pp. 407–410.

[23] K. Susanto and B. Yang, "Genetic algorithm based optimization design of miniature piezoelectric forceps," in *International Conference on Robotics and Automation.* IEEE, Apr. 2004, pp. 1358–1363.

[24] J. K. Coultate, C. H. J. Fox, S. McWilliam, and A. R. Malvern, "Application of optimal and robust design methods to a mems accelerometer," *Sensors and Actuators A: Physical*, vol. 142, pp. 88–96, Apr. 2008.

[25] H. Zhu, P. Wang, and Z. Fan, "Evolutionary design optimization of mems: A brief review," in *Industrial Technology (ICIT), 2010 IEEE International Conference on*, 2010, pp. 1684–1687.

[26] P. C. Fourie and A. A. Groenwold, "The particle swarm optimization algorithm in size and shape optimization," *Struct Multidisc Optim*, pp. 259–267, 2002.

[27] A. Torrents, K. Azgin, S. W. Godfrey, E. S. Topalli, T. Akin, and L. Valdevit, "Mems resonant load cells for micro-mechanical test frames: feasibility study and optimal design," *Journal of Micromechanics and Microengineering*, vol. 20, no. 12, Nov. 2010.

[28] C. Dai, W. Chen, and Y. Zhu, "Seeker optimization algorithm," in *International Conference on Computational Intelligence and Security*, vol. 1, 2006, pp. 225–229.

[29] A. Ketabi and M. J. Navardi, "Optimization shape of variable-capacitance micromotor using seeker optimization algorithm," *Journal of Electrical Engineering and Technology*, vol. 7, no. 2, pp. 212–220, 2012.

[30] G. Kirshnan and G. K. Ananthasuresh, "Evaluation and design of displacement-amplifying compliant mechanisms for sensor applications," *Journal of Mechanical Design*, vol. 130, Oct. 2008.

[31] V. Seidermann, S. Butefisch, and S. Buttgenbach, "Fabrication and investigation of in-plane compliant su8 structures for mems and their application ot micro valves and micro grippers," *Sensors and Actuators: A Physical*, vol. 97-98, pp. 457–461, Nov. 2002.

[32] R. Chelouah and P. Siarry, "Tabu search applied to global optimization," *European Journal of Operational Research*, vol. 123, pp. 256–270, 2000.

[33] Y. Gong, F. Zhao, H. Xin, J. Lin, and Q. Bai, "Simulation and optimal design for RF MEMS cantilevered beam switch," in *2009 ETP International Conference on Future Computer and Communication*. IEEE Computer Society, 2009, pp. 84–87.

[34] G. Leu, S. Simion, and A. Serbanescu, "MEMS optimization using genetic algorithms," in *CAS 2004 Proceedings*, vol. 2, 2004, pp. 475–478.

[35] N. Zhou, A. Agogino, and K. S. J. Pister, "Automated design synthesis for micro-electro-mechanical systems," in *Proceedings of DETC 2002: Design Automation*, 2002.

[36] J. Horn and N. Nafpliotis, "Multiobjective optimization using the niched pareto genetic algorithm," in *Proceedings of the First IEEE Conference on Evolutionary Computation*, vol. 1. IEEE World Congress on Computational Intelligence, 1994, pp. 82–87.

[37] D. E. Goldberg and K. Deb, *A comparative analysis of selection schemes used in genetic algorithms*. Morgan Kaufmann Publishers, 1991, pp. 69–93.

[38] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach," in *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, Nov. 1999, pp. 257–271.

[39] K. Deep and M. Thakur, "A new mutation operator for real coded genetic algorithms," *Applied mathematics and Computation*, vol. 193, pp. 211–230, 2007.

[40] F. Herrera and M. Lozano, "Two-loop real-coded genetic algorithms with adaptive control of mutation step sizes," *Applied Intelligence*, vol. 13, no. 3, pp. 187–204, 2000.

[41] R. Hinterding, "Gaussian mutation and self-adaption for numeric genetic algorithms," in *Evoloutionary Computation, IEEE International Conference on*, vol. 1, 1995.

[42] R. Kamalian, Y. Zhang, H. Takagi, and A. M. Agogino, "Reduced human fatigue interactive evolutionary computation for micromachine design," in *Proceedings of the Fourth International Conference on Machine Learning and Cybernetics*, Aug. 2005, pp. 5666–5671.

[43] *Comsol Multiphysics 4.0a*, COMSOL Inc., 1997-2010. [Online]. Available: http://www.comsol.com/

[44] *MATLAB R2009b*, The MathWorks Inc., Natick, Massachusetts, 2009. [Online]. Available: http://www.mathworks.com/products/matlab/

[45] J. R. Rairden, C. A. Neugebauer, and R. A. Sigsbee, "Interdiffusion in thin conductor films - chromium/gold, nickel/gold, and chromium silicide/gold," *Metallurgical Transactions*, vol. 2, pp. 719–721, Mar. 1971.

[46] R. Hickey, D. Sameoto, T. Hubbard, and M. Kujath, "Time and frequency response of two-arm micromachined thermal actuators," *Journal of Micromechanics and Microengineering*, vol. 13, pp. 40–46, Nov. 2002.

[47] X. Gu, G. Karunasiri, and G. Chen, "Determination of thermal parameters of microbolometers using a single electrical measurement," *Applied Physics Letters*, vol. 72, no. 15, pp. 1881–1883, Apr. 1998.

[48] E. Bogatin, *Signal and Power Integrity Simplified*, 2nd ed. Prentice Hall, 2011.

[49] J. R. Rairden, C. A. Neugebauer, and R. A. Sigsbee, "Interdiffusion in thin conducter filmes - chromium/gold, nickel/gold and chromium silicide/gold," *Metallurgical and Materials Transactions B*, vol. 2, no. 3, pp. 719–722, 1971.

[50] *PHOTOMAP 3D: Interferometric sensor for fast 3D measurements*, FOGALE nanotech, accessed May 2012. [Online]. Available: http://www.fogale.com/optical_profilers/pages/photomap3d.php

[51] D. Koester, A. Cowen, R. Mahadevan, and B. Hardy, *PolyMUMPs Design Handbook: a MUMPs Process*, 8th ed., MEMSCAP Inc., 2002.

[52] M. J. Madou, *Fundamentals of Microfabrication: The Science of Miniatruization*, 2nd ed. CRC Press LLC, 2002.

[53] S. A. Campbell, *The Science and Engineering of Microelectronic Fabrication*, 2nd ed. Oxford University Press, 2001.

[54] T. M. H. Lee, D. H. Y. Lee, C. Y. N. Liaw, A. I. K. Lao, and I.-M. Hsing, "Detailed characterization of anodic bonding process between glass and thin-film coated silicon substrates," *Sensors and Actuators A: Physical*, vol. 86, pp. 103–107, Oct. 2000.

[55] J.-T. Huang and H.-A. Yang, "Improvement of bonding time and quality of anodic bonding using the spiral arrangement of multiple point electrodes," *Sensors and Actuators A: Physical*, vol. 102, pp. 1–5, Dec. 2002.

[56] *TABLE 9 Type K Thermocouple: thermoelectric voltage as a function of temperature (°C); reference junctions at 0°C*, Pyromation, Inc, accessed May 2012. [Online]. Available: http://www.pyromation.com/TechInfo/Tables.aspx

[57] *TC1046 and TC1047/A Voltage Output Temperature Sensor Family*, Microchip Technology, Inc., 2004.

[58] C. Valenti, *TB051: Precision Temperature Measurement Technical Brief*, Microchip Technology, Inc., 2001.

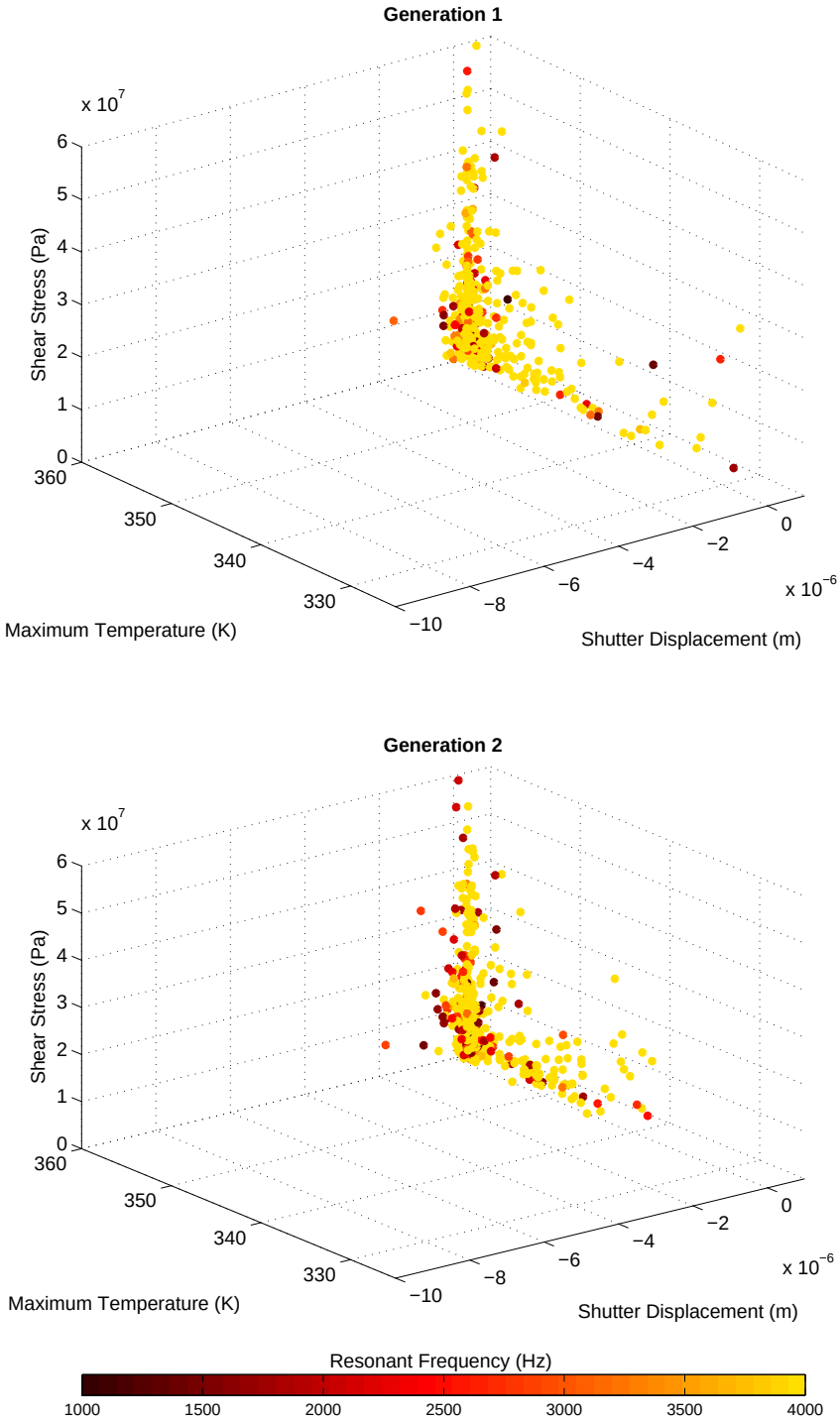
[59] *USB Framework for PIC18, PIC24 & PIC32*, 2nd ed., Microchip, Inc. [Online]. Available: http://www.microchip.com

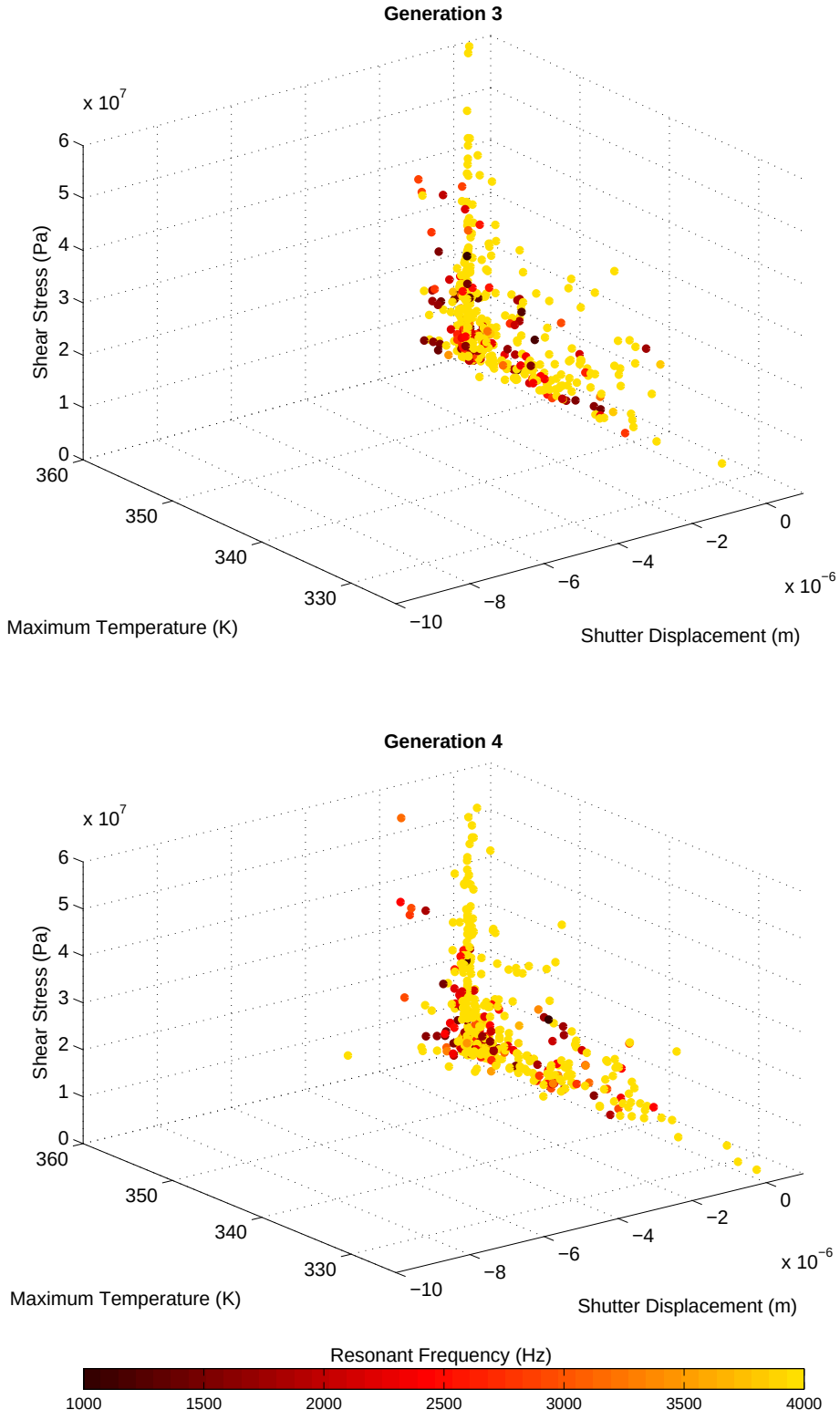
[60] K. Ogata, *Modern control Engineering*, 5th ed. Prentice Hall, 2010.

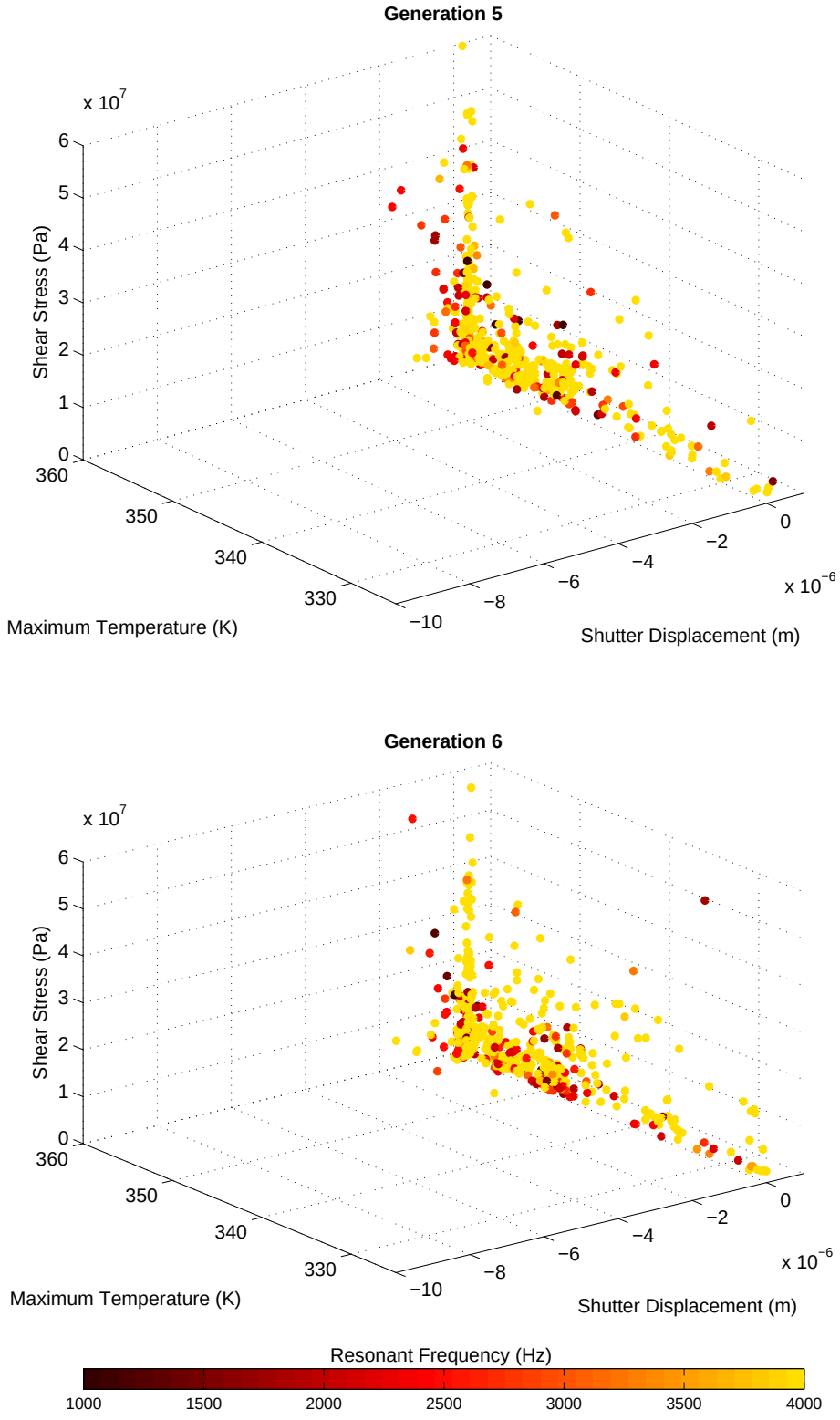
[61] *Fisherbrand Sparkleen Detergent*, Fisher Scientific, 2012. [Online]. Available: http://www.fishersci.com

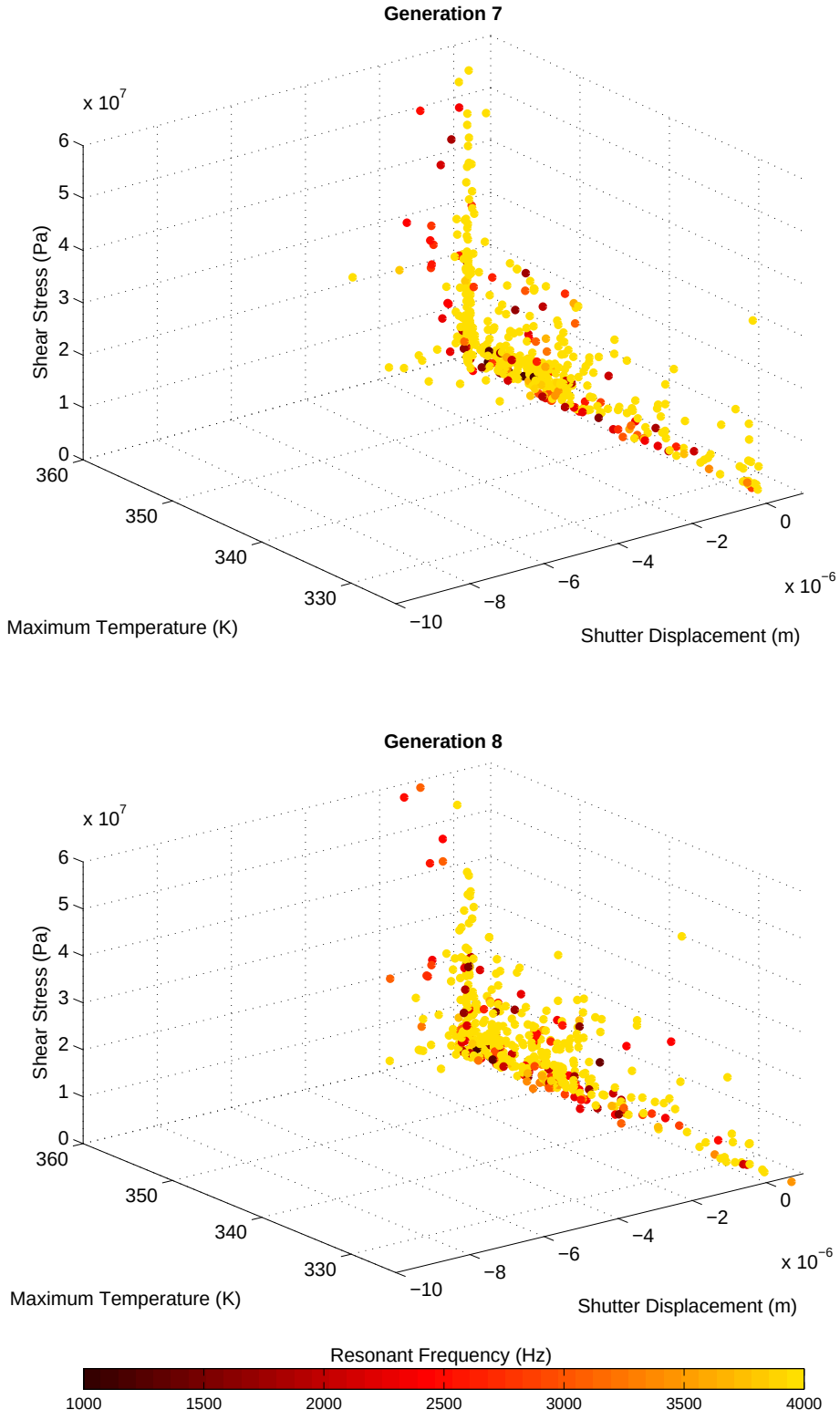
[62] *EPO-TEK® H74F Product Information Sheet*, Epoxy Technology, Inc., accessed June 2012. [Online]. Available: http://epotek.com

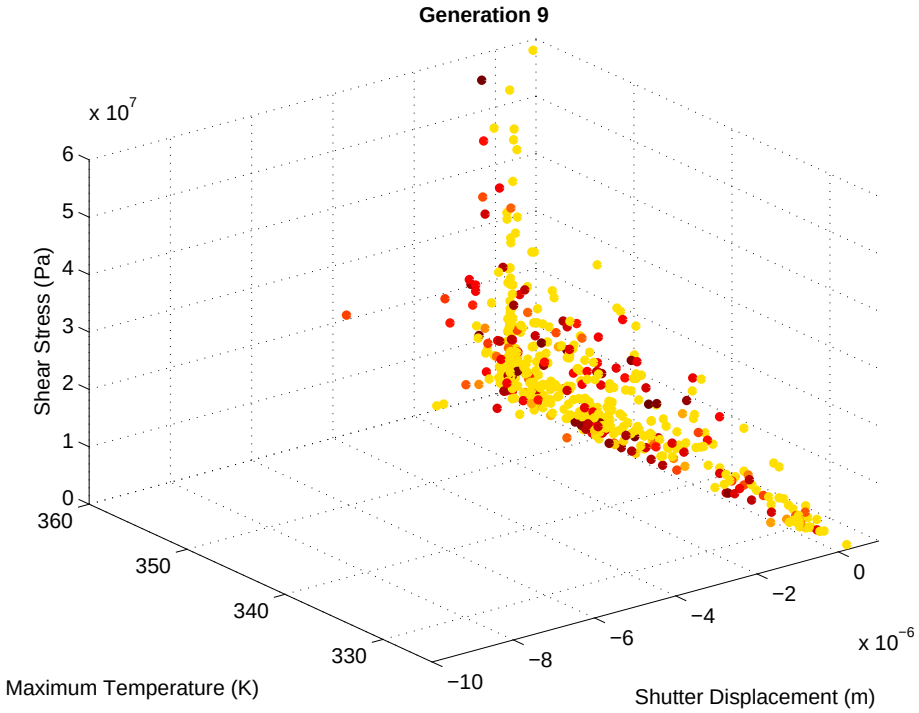
Appendix A: Complete Results of GA

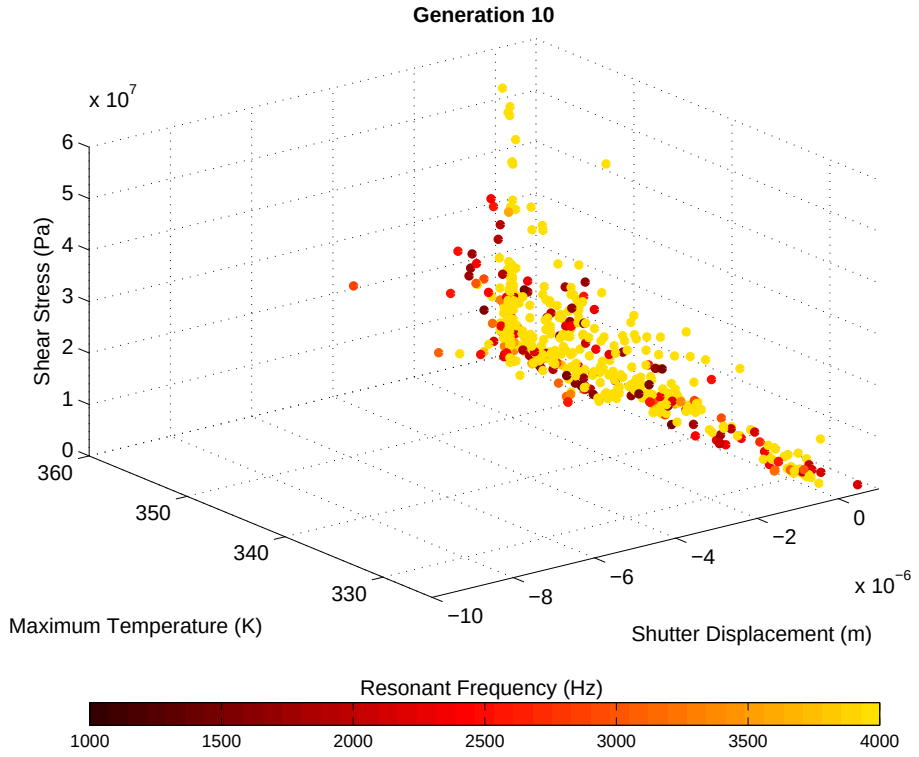


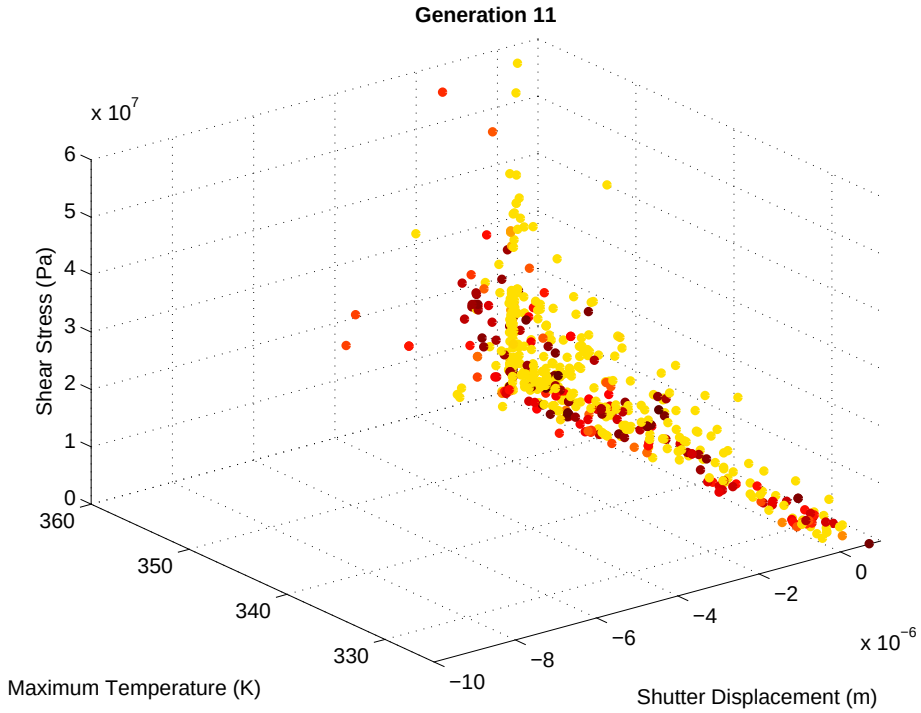

Generation 3
x 10^7
Shear Stress (Pa)
6
5
4
3
2
1
0
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Generation 4
x 10^7
Shear Stress (Pa)
6
5
4
3
2
1
0
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

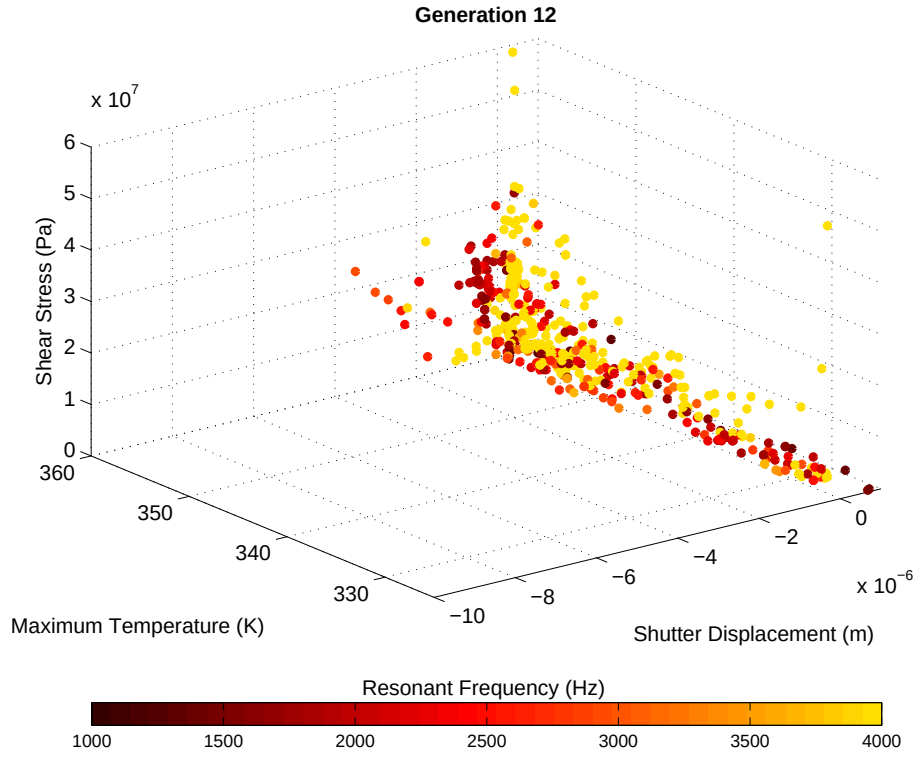

Generation 5
x 10^7
6
5
4
3
2
1
0
Shear Stress (Pa)
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Generation 6
x 10^7
6
5
4
3
2
1
0
Shear Stress (Pa)
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

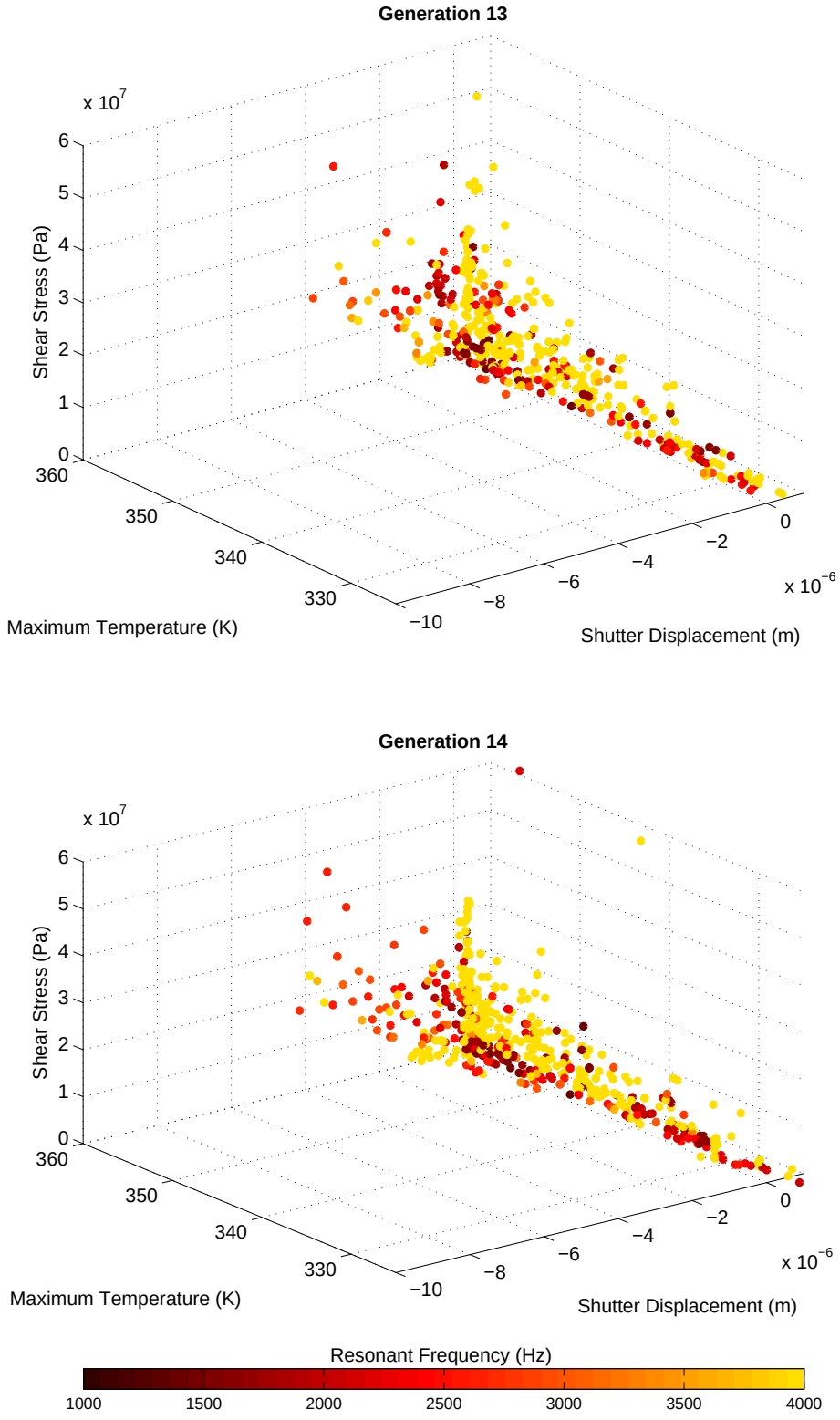

Generation 7
x 10^7
6
5
4
3
2
1
0
Shear Stress (Pa)
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Generation 8
x 10^7
6
5
4
3
2
1
0
Shear Stress (Pa)
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

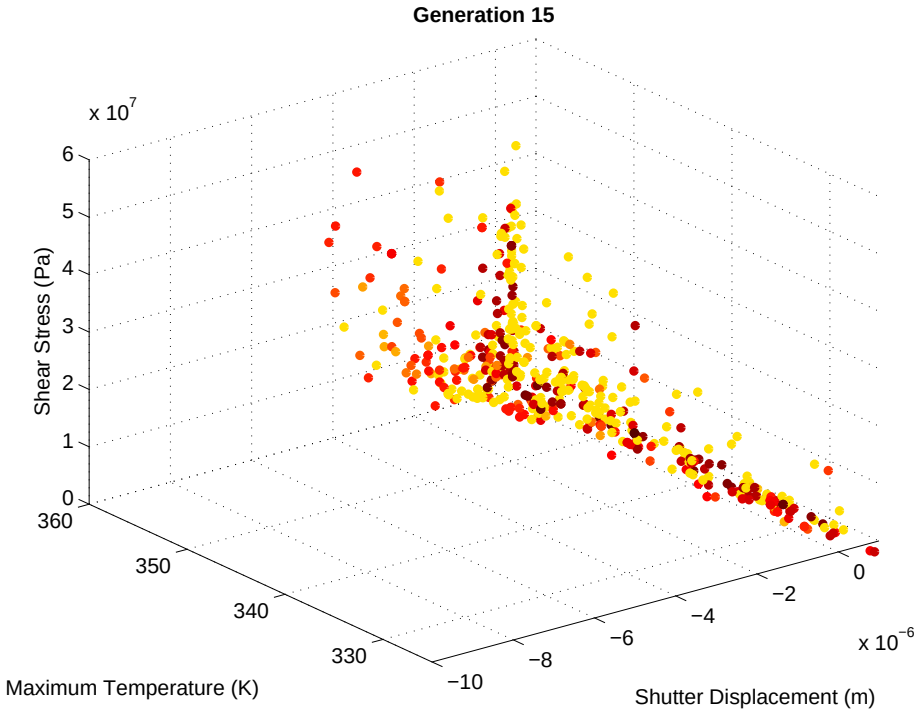

Generation 9
x 10^7
6
5
4
3
2
1
0
Shear Stress (Pa)
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)

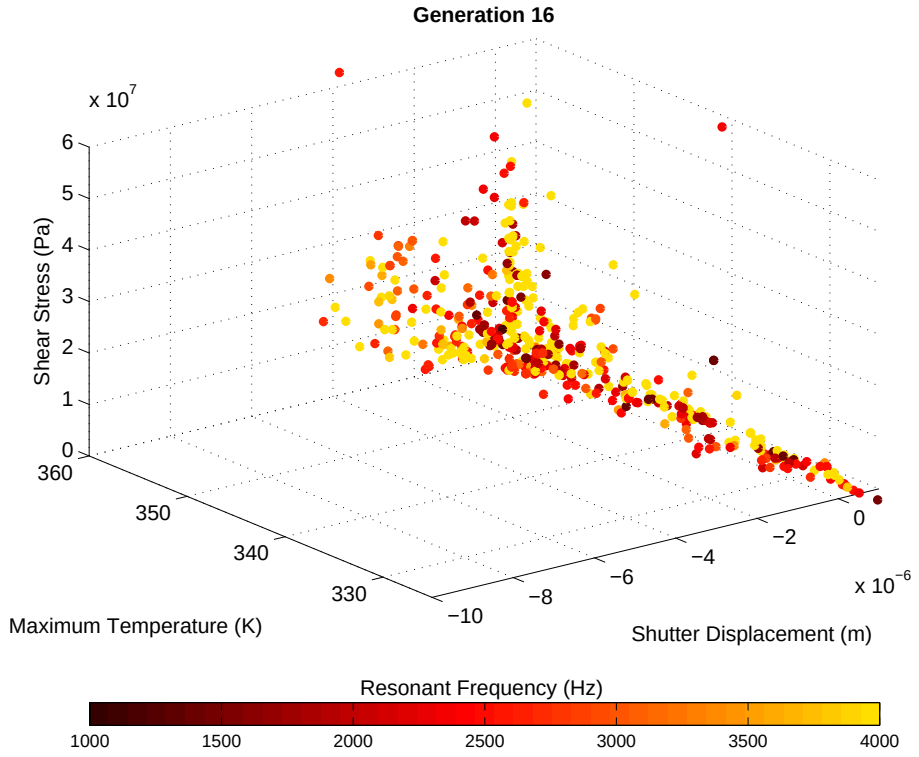

Generation 10
x 10^7
6
5
4
3
2
1
0
Shear Stress (Pa)
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

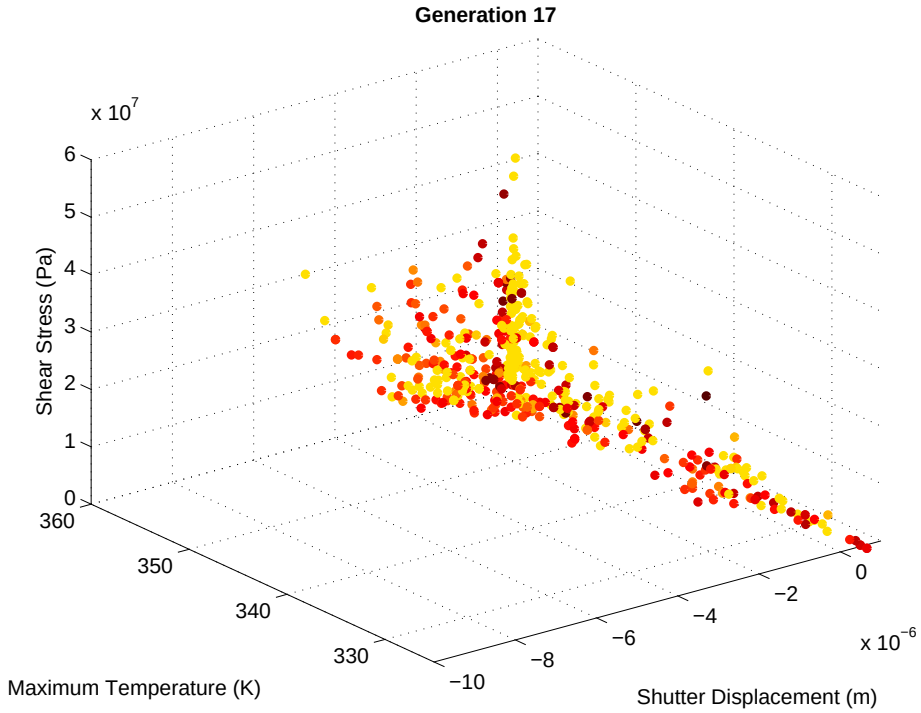

Generation 11
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6

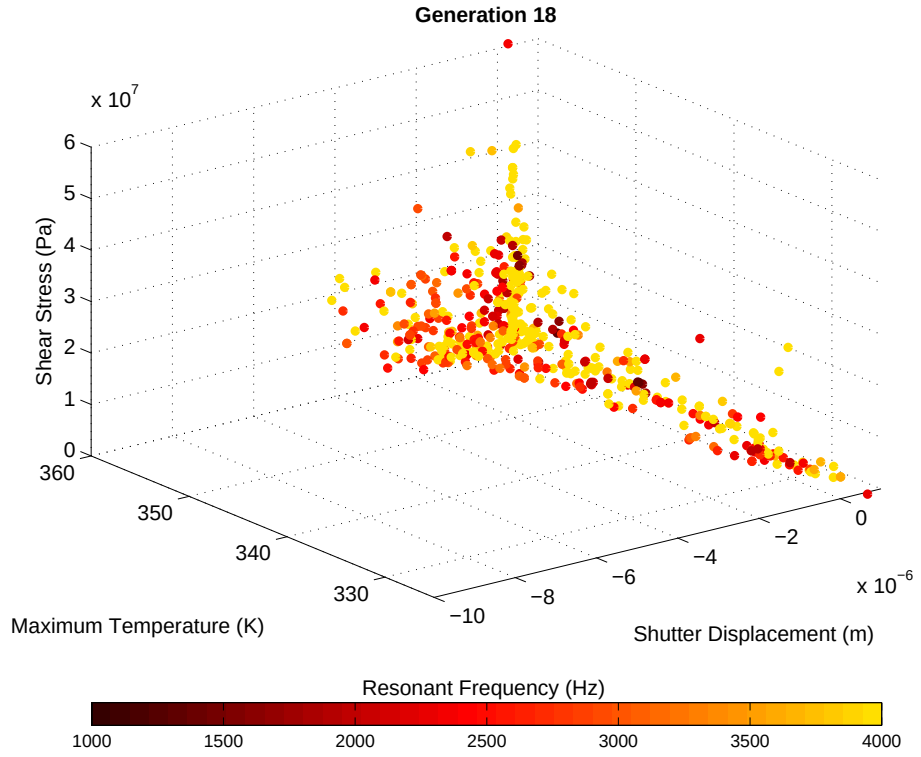

Generation 12
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Resonant Frequency (Hz)

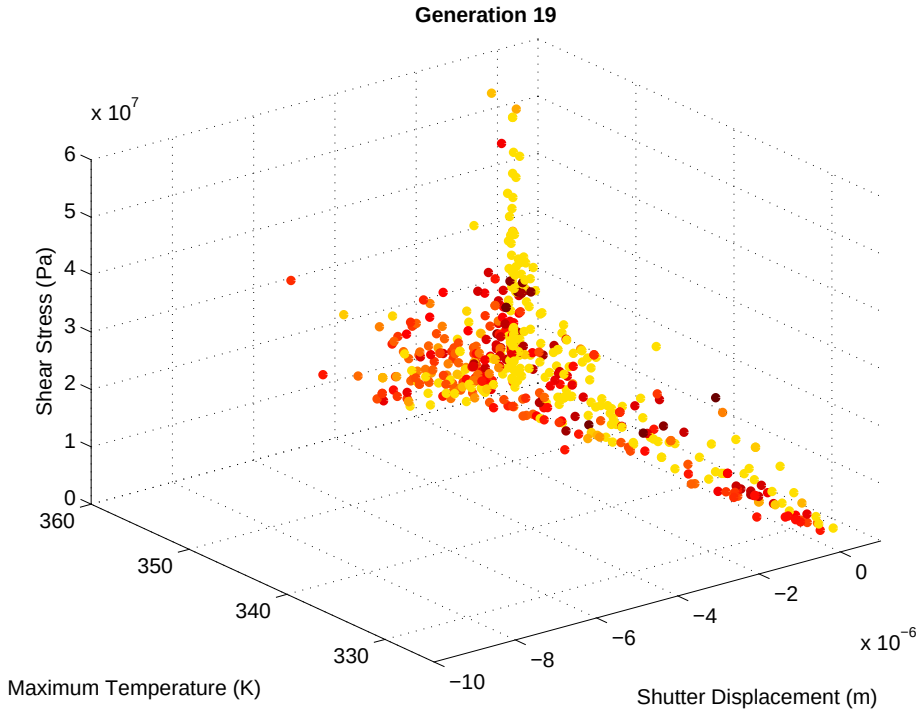

Generation 13
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Generation 14
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

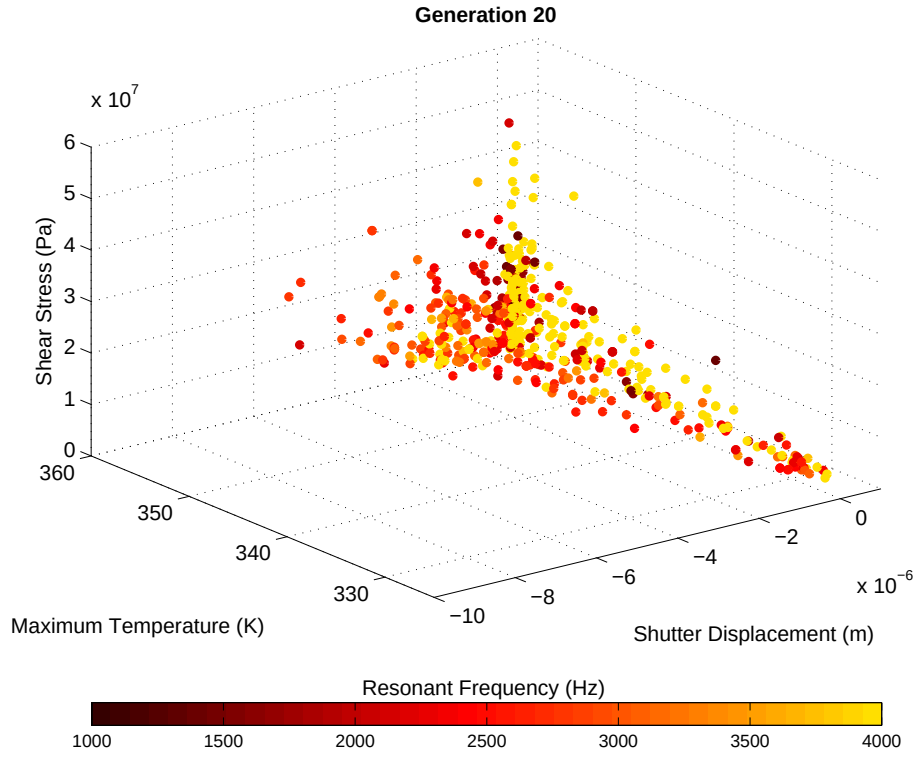

Generation 15
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6

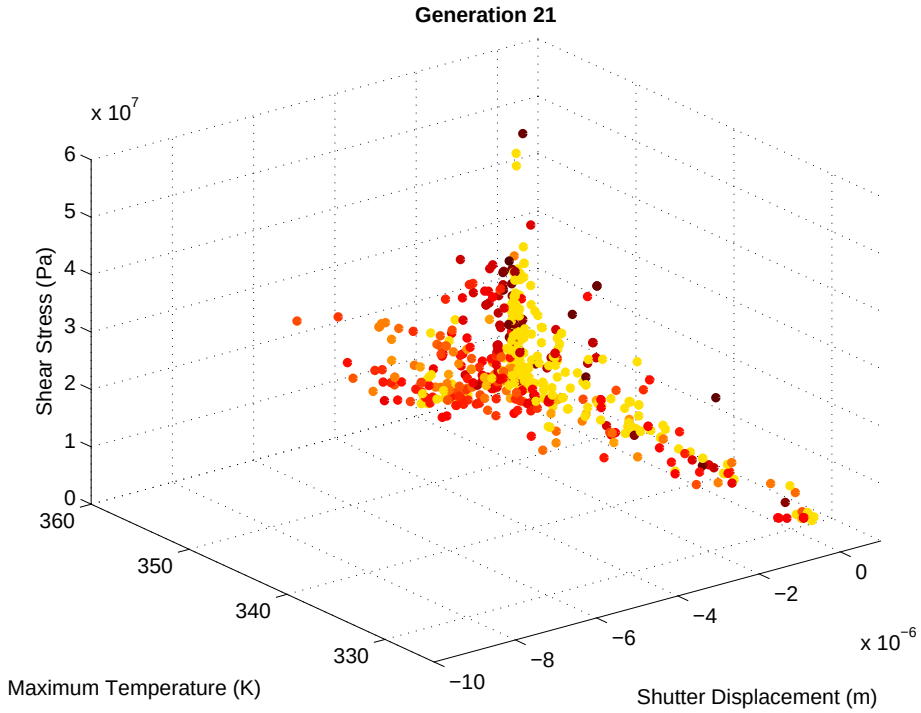

Generation 16
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Resonant Frequency (Hz)

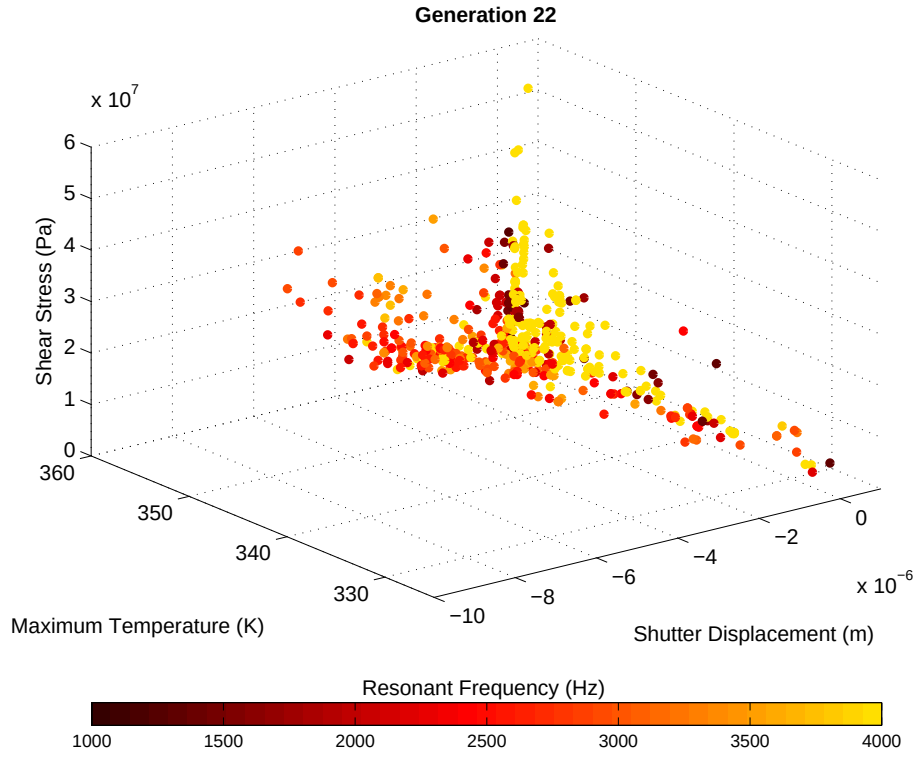

Generation 17
x 10^7
6
5
4
3
2
1
0
Shear Stress (Pa)
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)

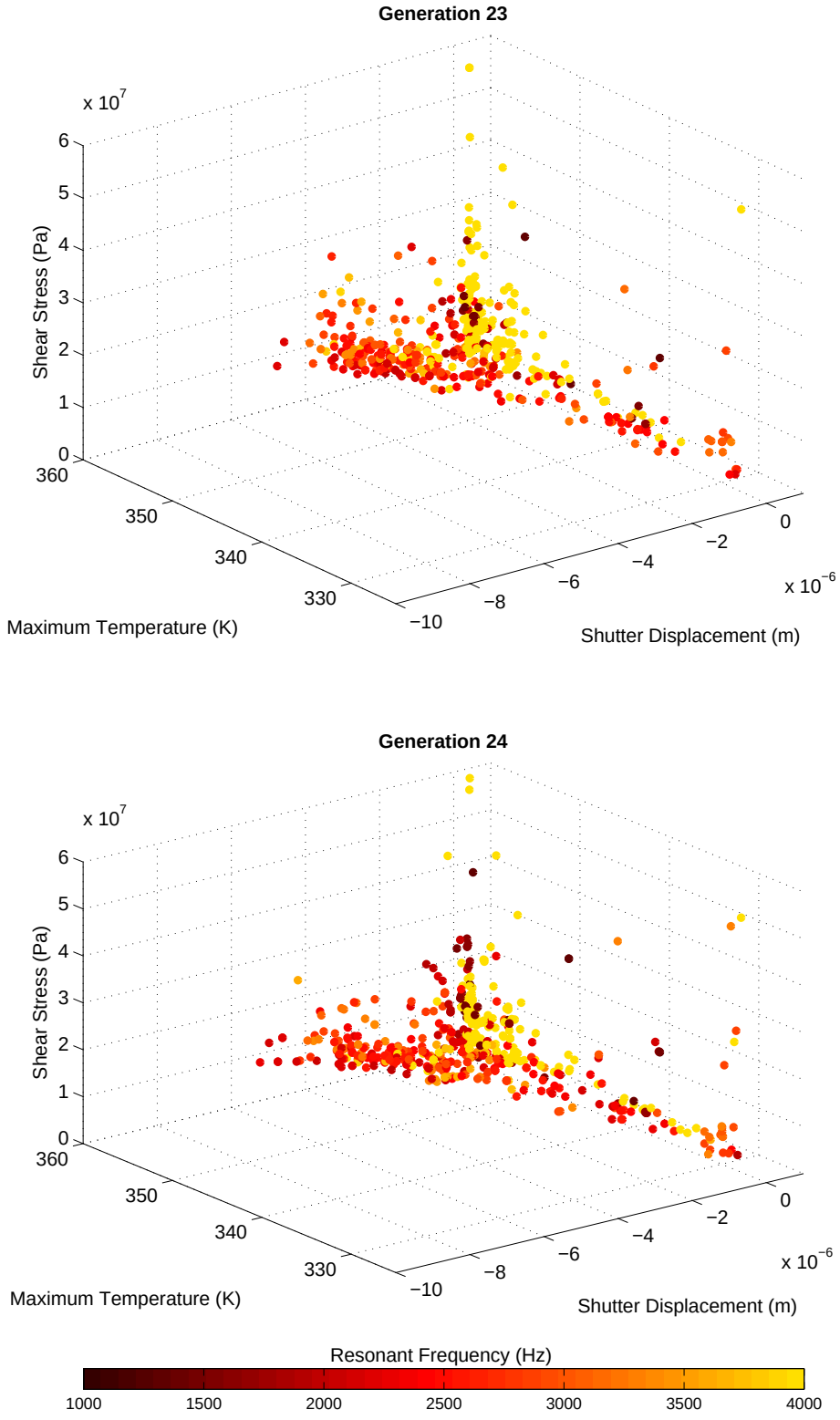

Generation 18
x 10^7
6
5
4
3
2
1
0
Shear Stress (Pa)
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

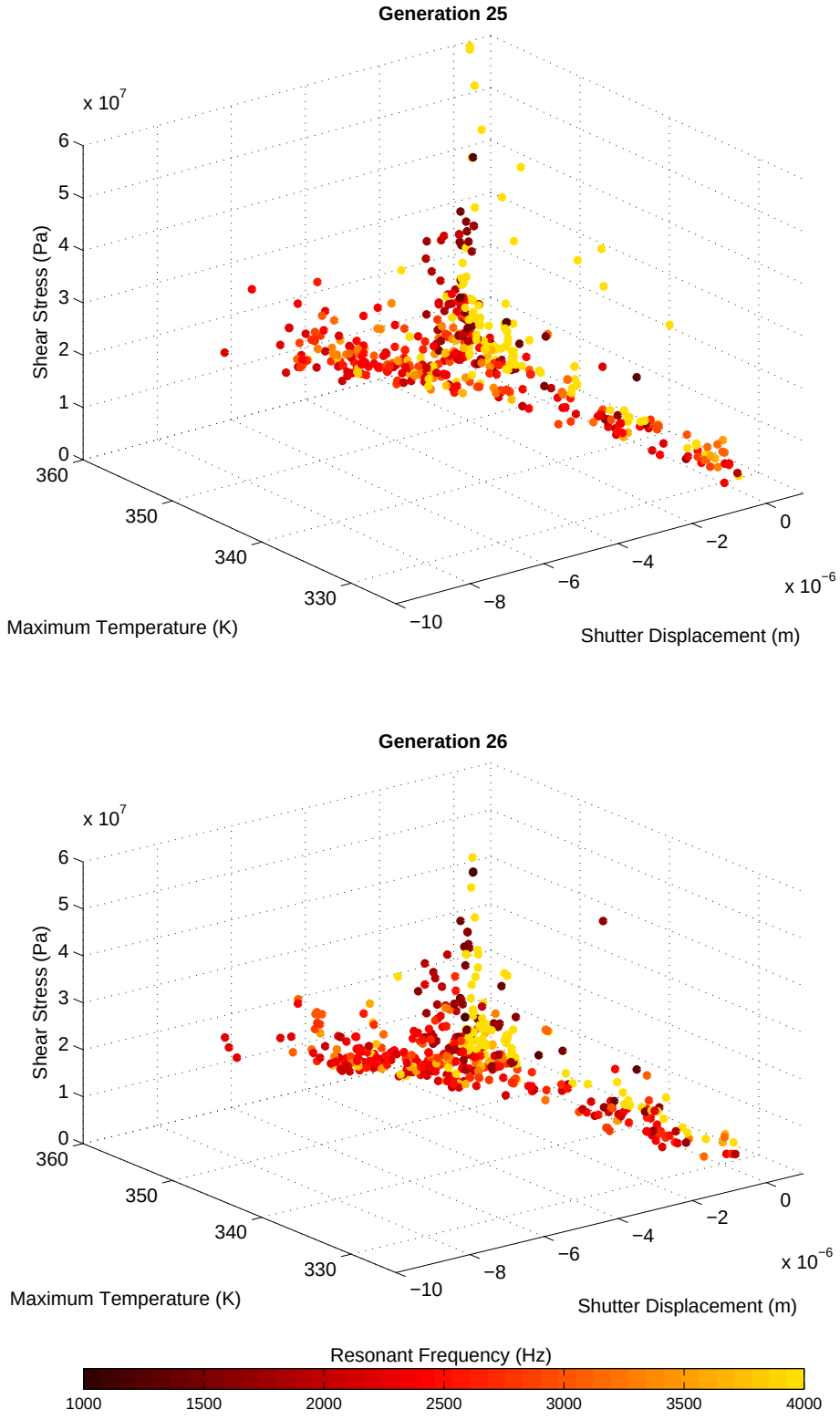

Generation 19
x 10^7
Shear Stress (Pa)
6
5
4
3
2
1
0
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)

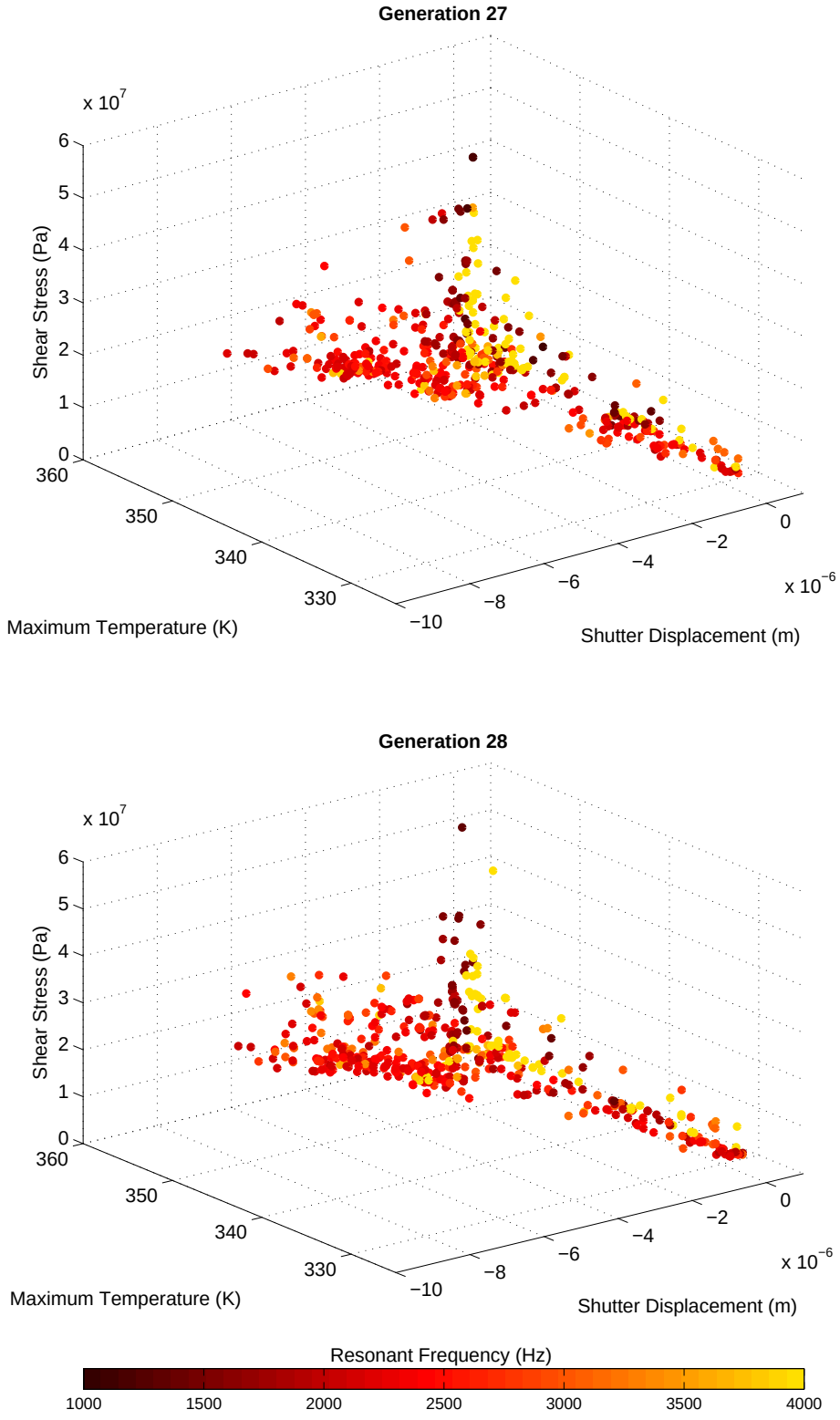

Generation 20
x 10^7
Shear Stress (Pa)
6
5
4
3
2
1
0
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

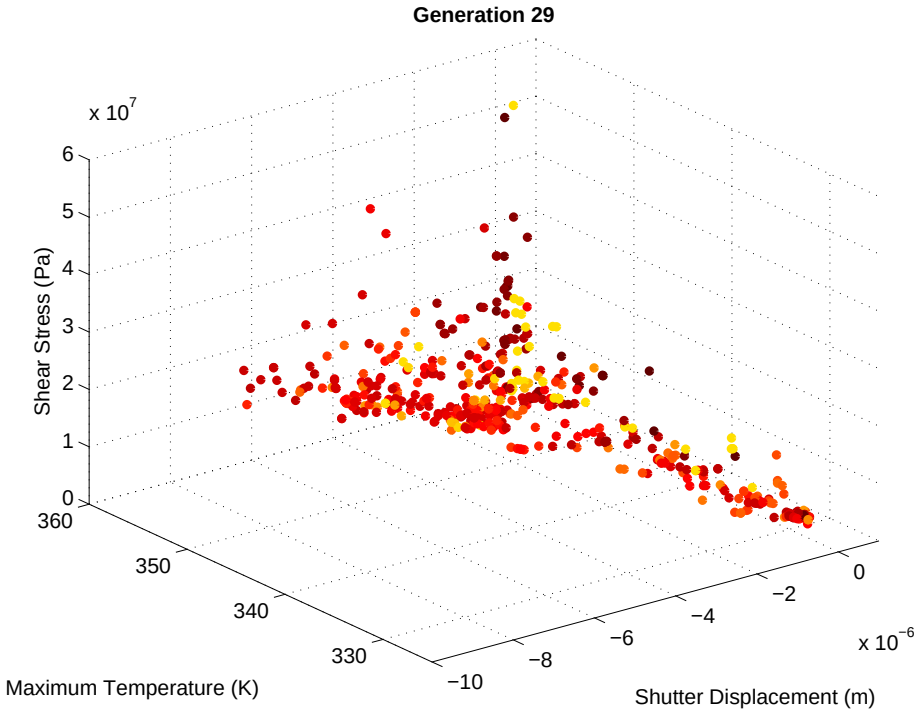

Generation 21
x 10^7
6
5
4
3
2
1
0
Shear Stress (Pa)
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)

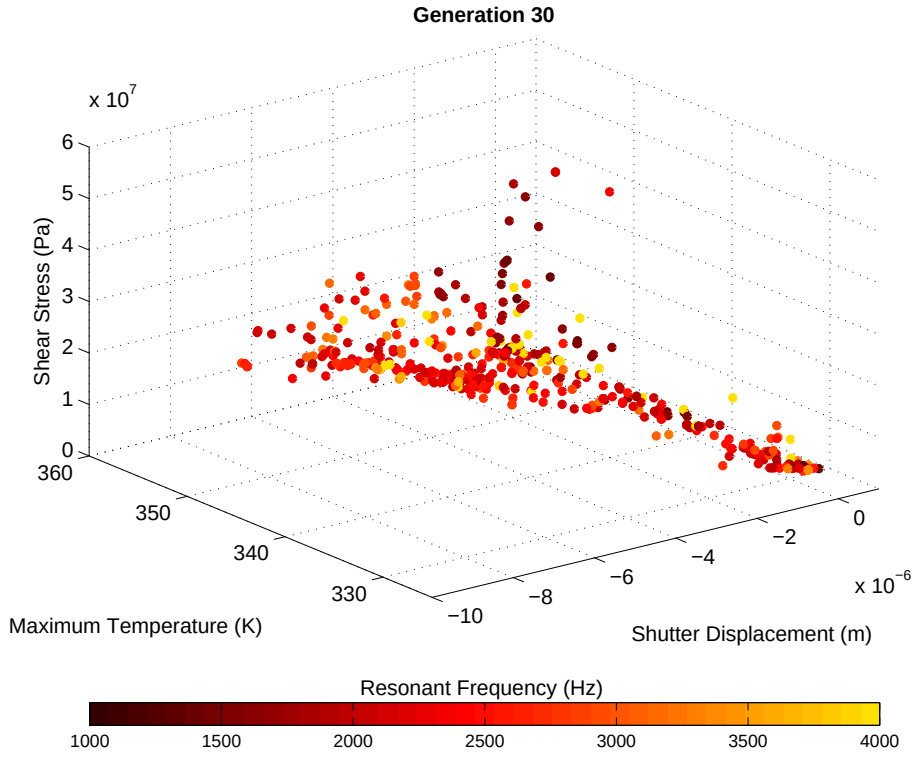

Generation 22
x 10^7
6
5
4
3
2
1
0
Shear Stress (Pa)
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

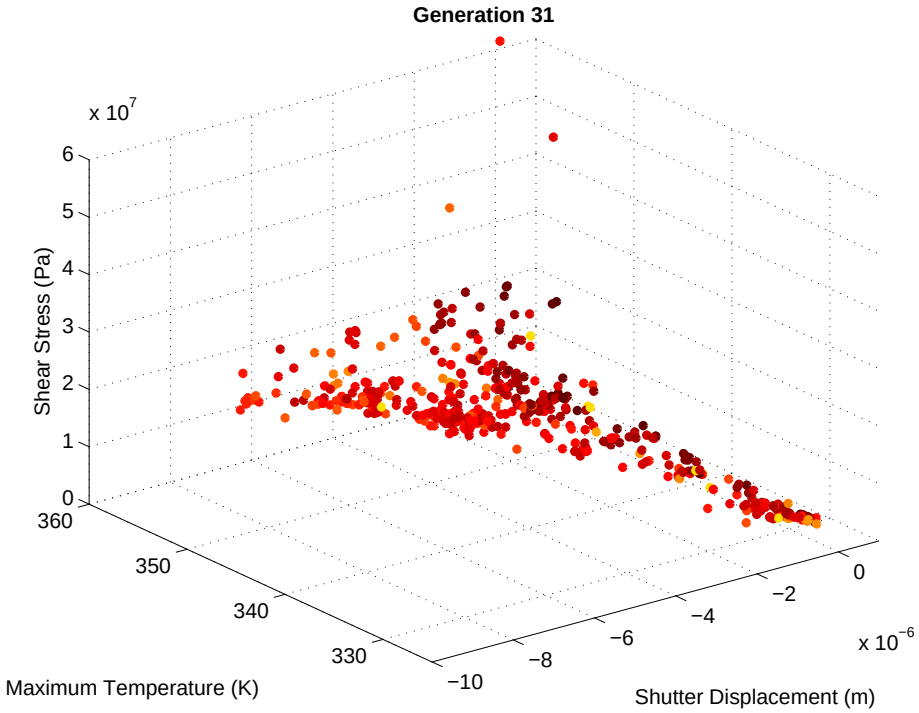

Generation 23
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Generation 24
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

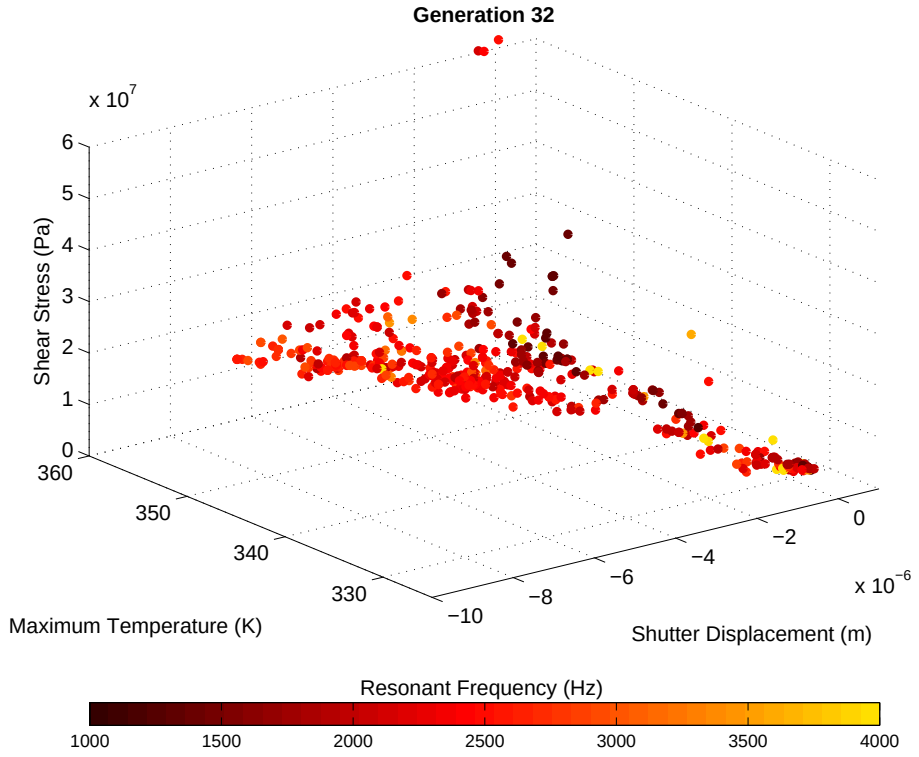

Generation 25
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Generation 26
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

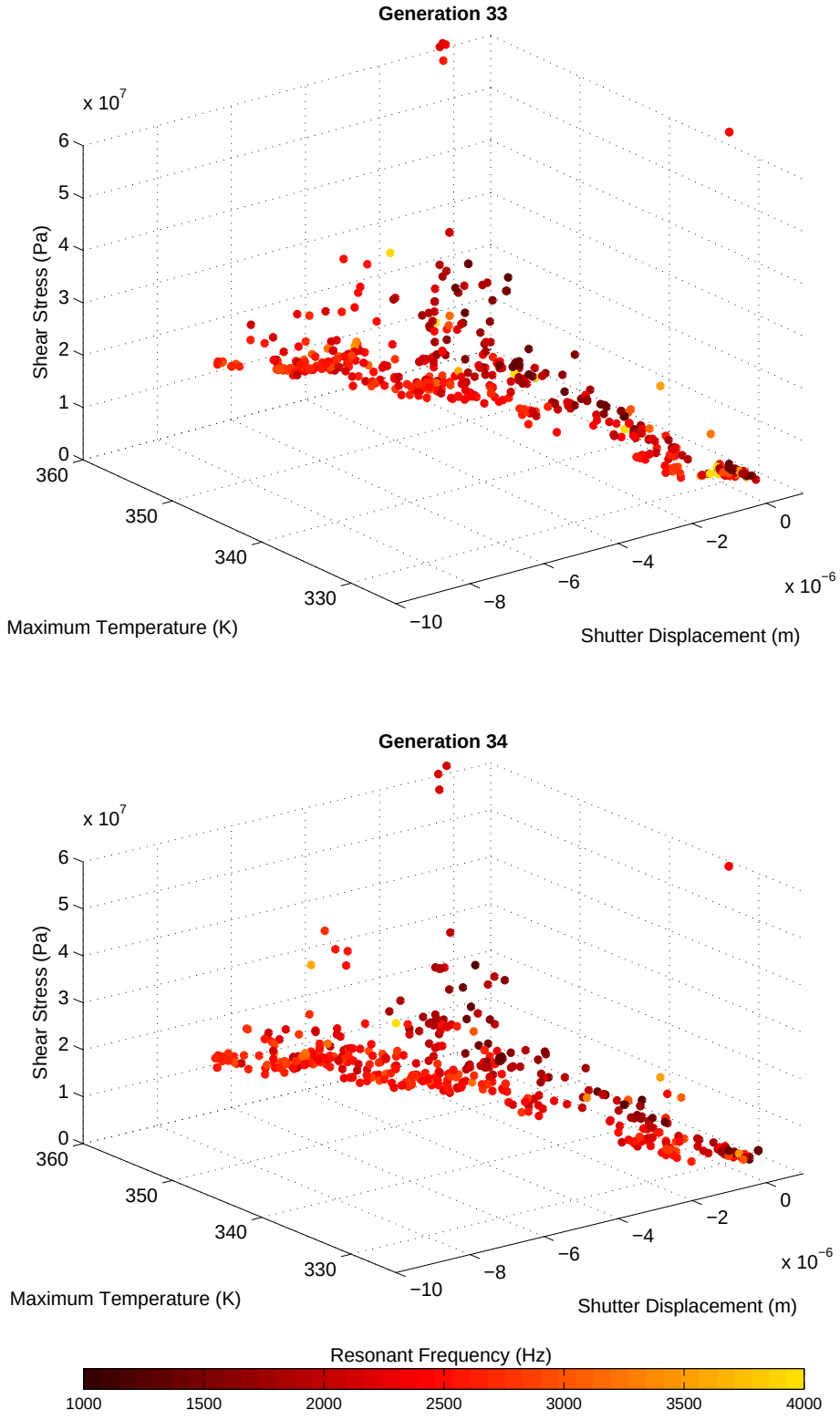

Generation 27
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Generation 28
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

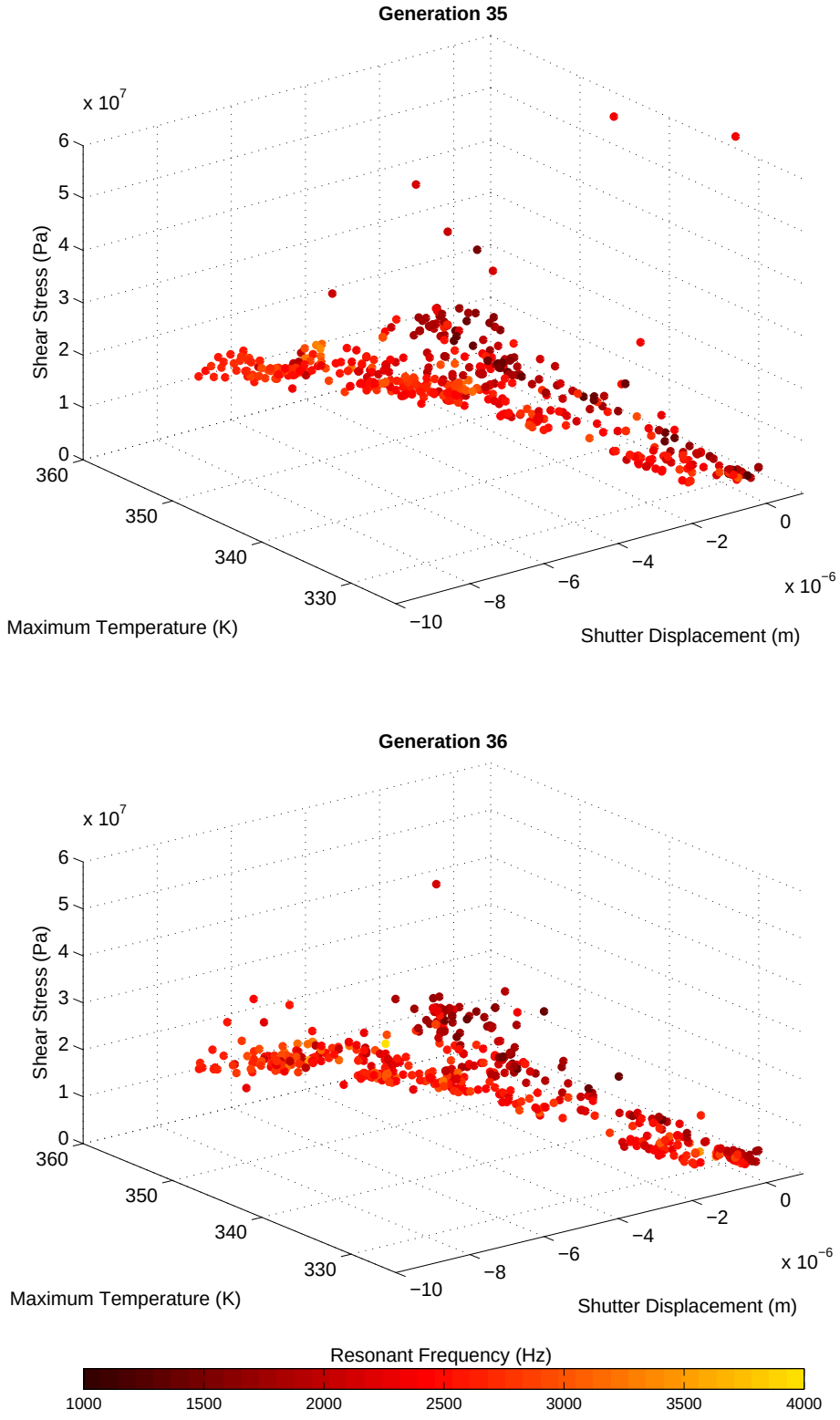

Generation 29
x 10^7
Shear Stress (Pa)
0
1
2
3
4
5
6
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)

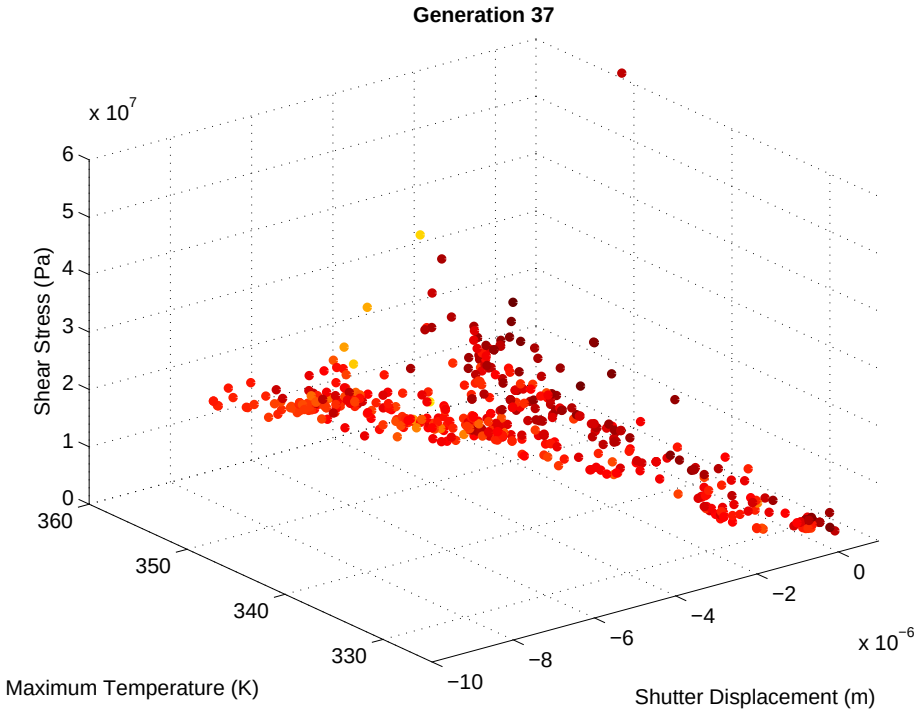

Generation 30
x 10^7
Shear Stress (Pa)
0
1
2
3
4
5
6
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

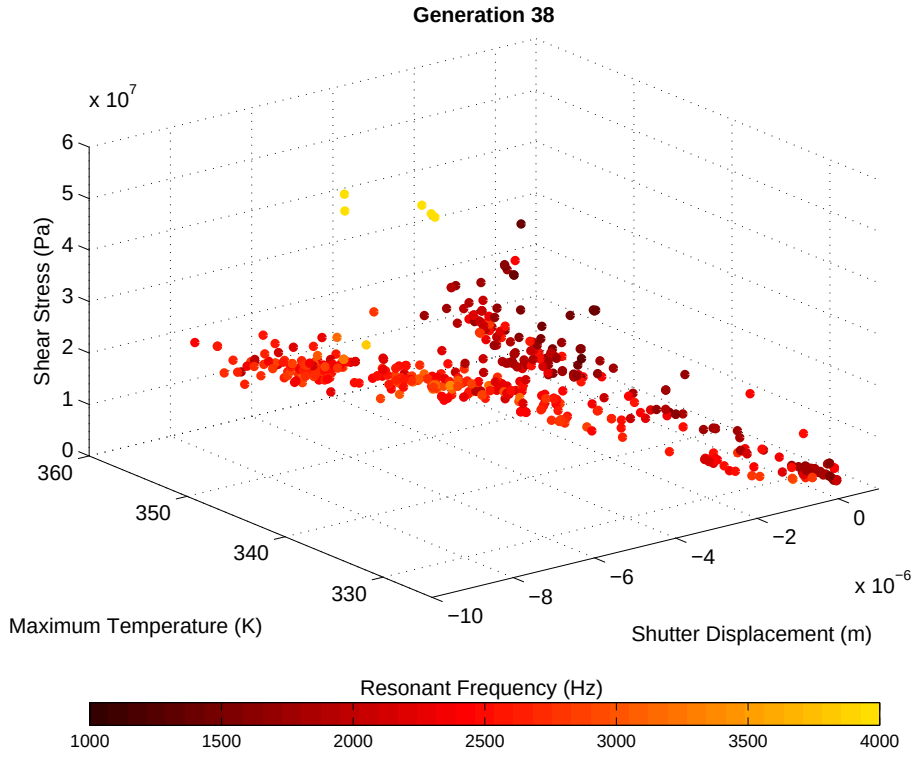

Generation 31
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6

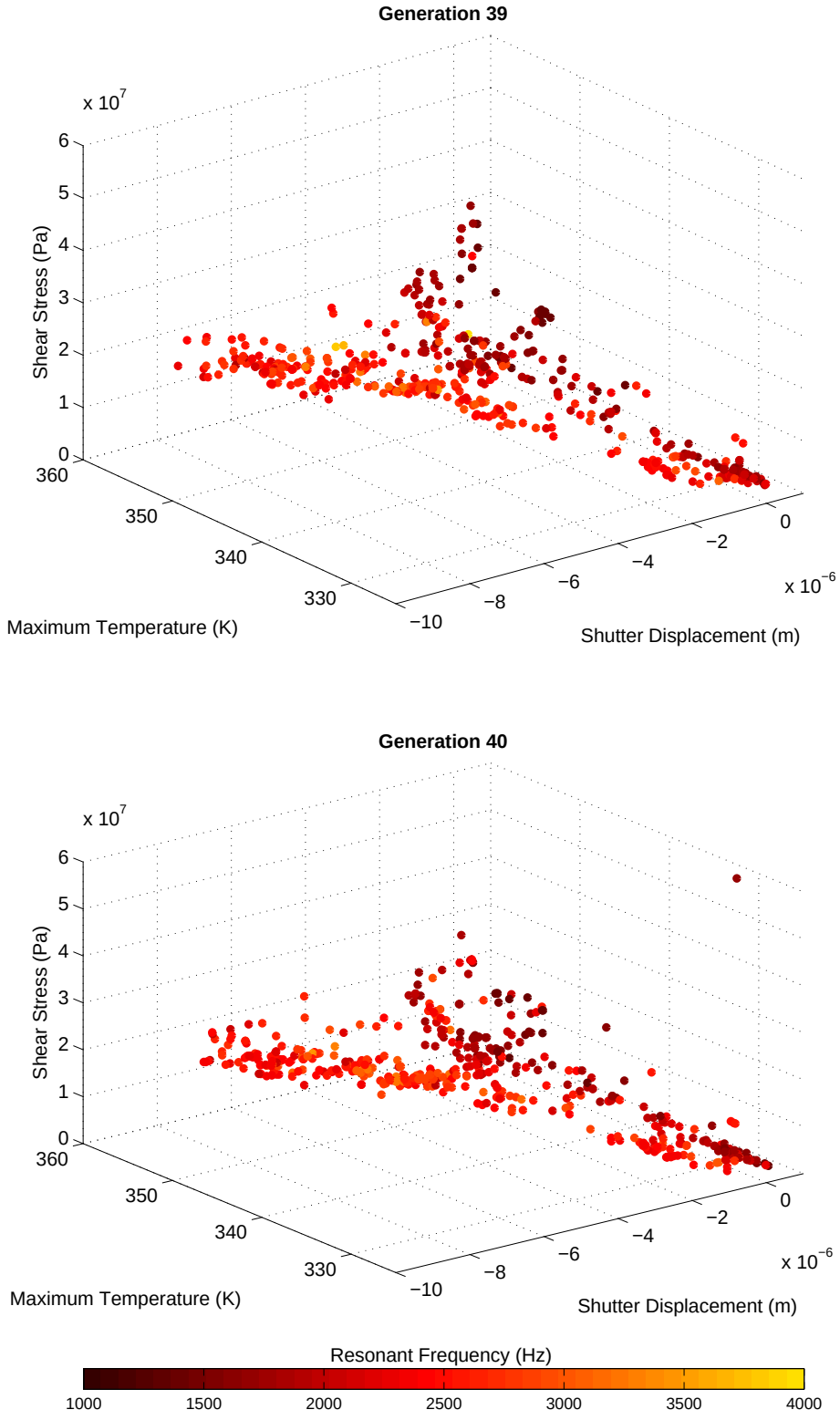

Generation 32
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

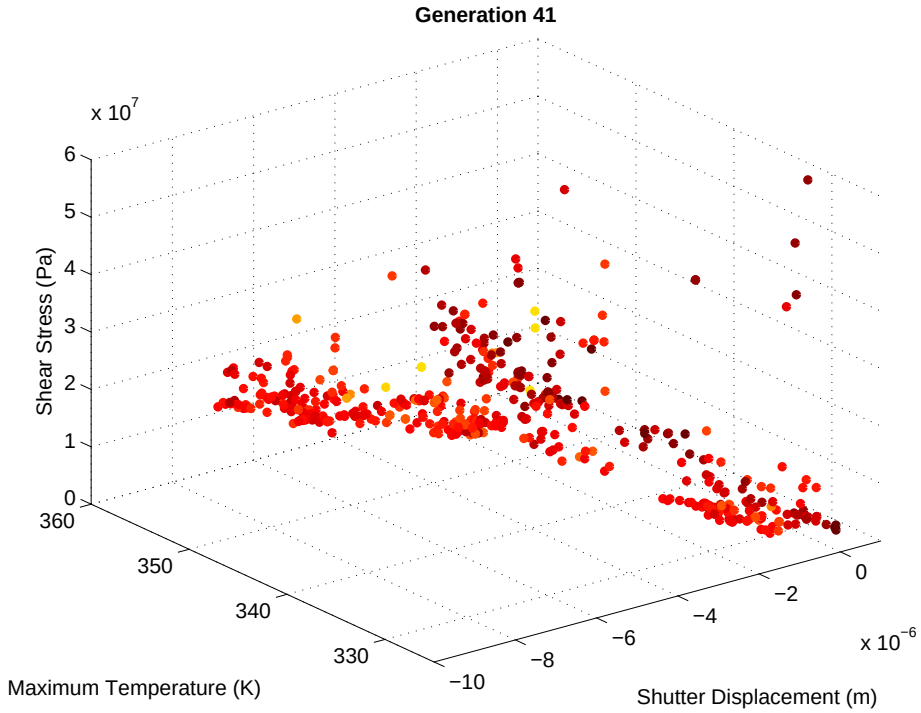

Generation 33
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Generation 34
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

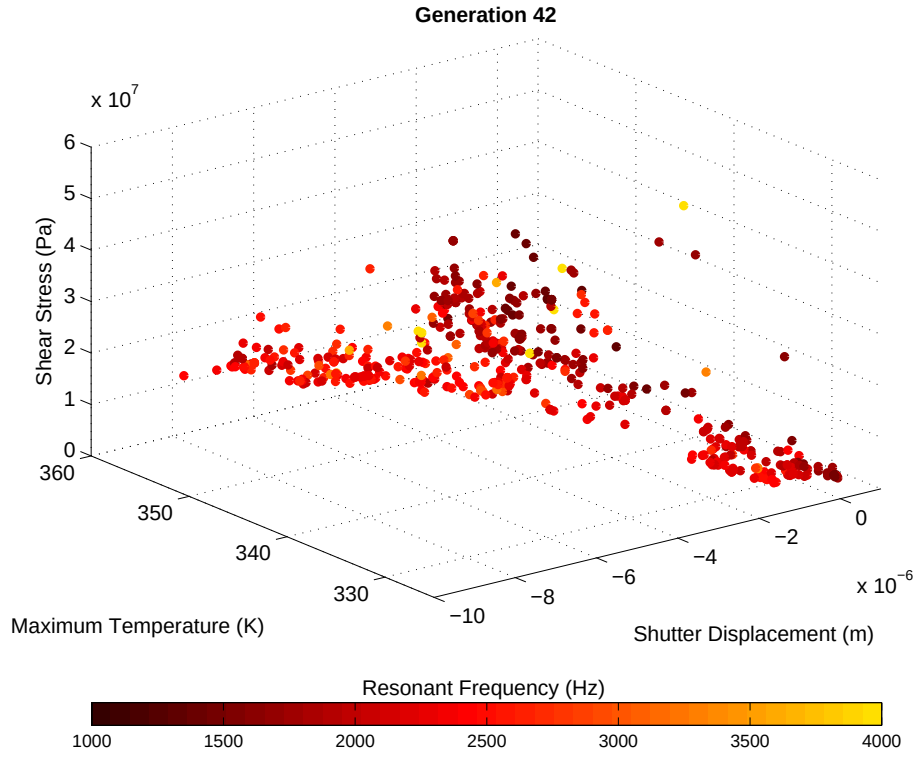

Generation 35
x 10^7
Shear Stress (Pa)
6
5
4
3
2
1
0
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Generation 36
x 10^7
Shear Stress (Pa)
6
5
4
3
2
1
0
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

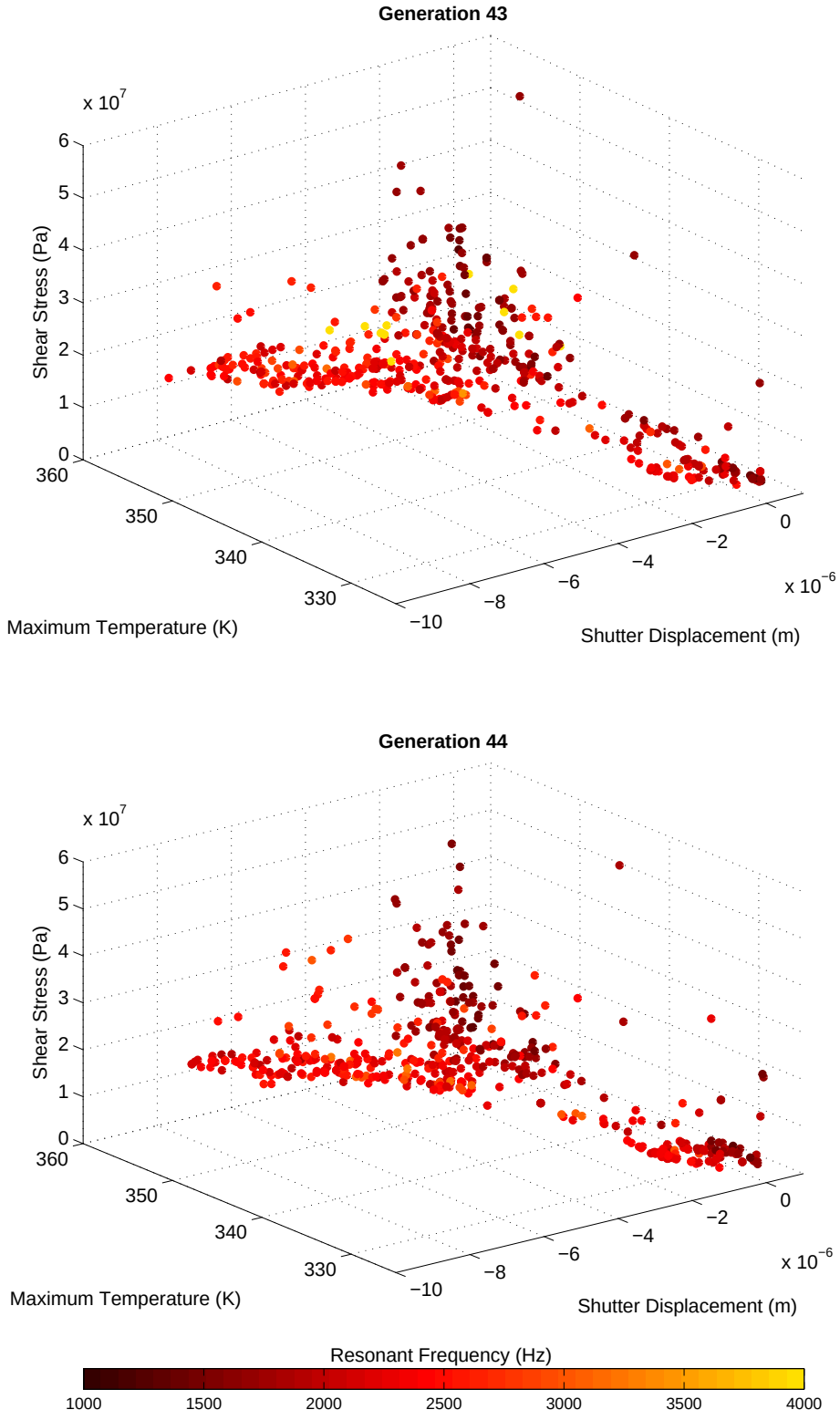

Generation 37
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6

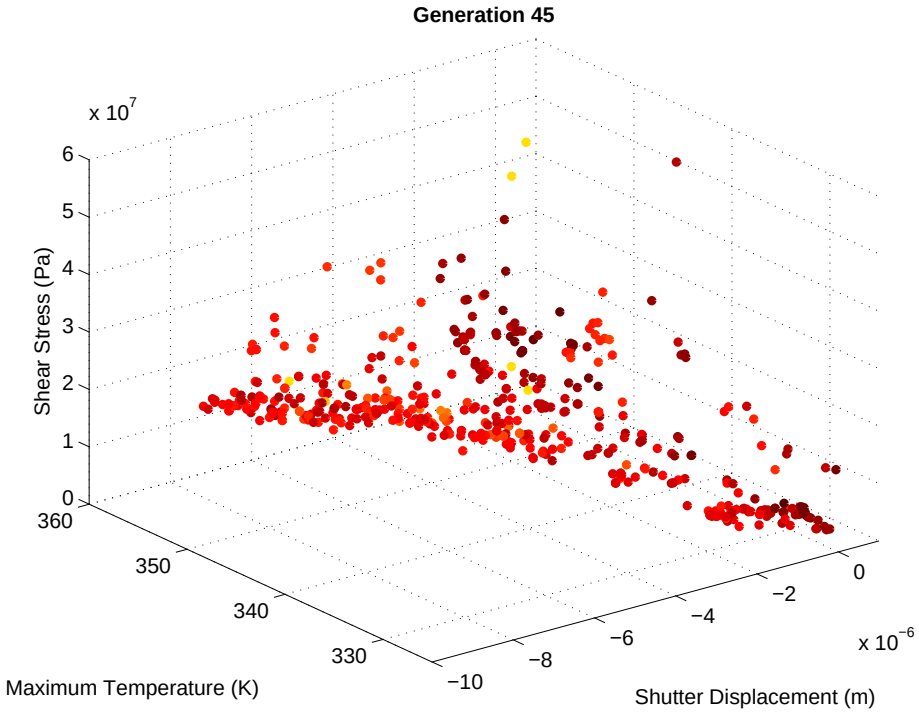

Generation 38
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

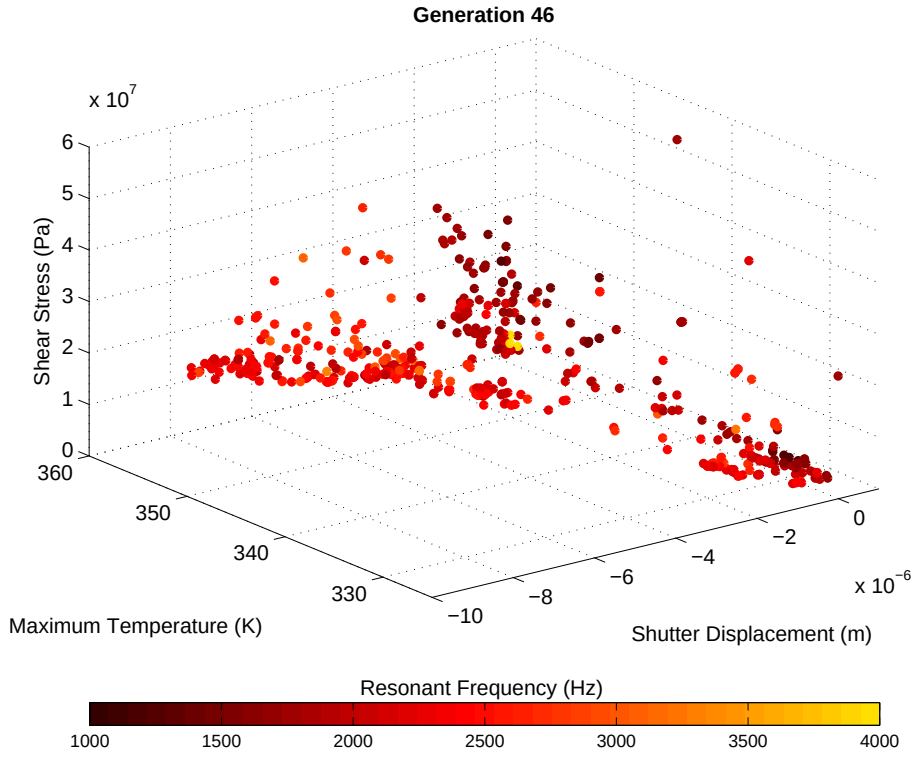

Generation 39
x 10^7
Shear Stress (Pa)
6
5
4
3
2
1
0
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Generation 40
x 10^7
Shear Stress (Pa)
6
5
4
3
2
1
0
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

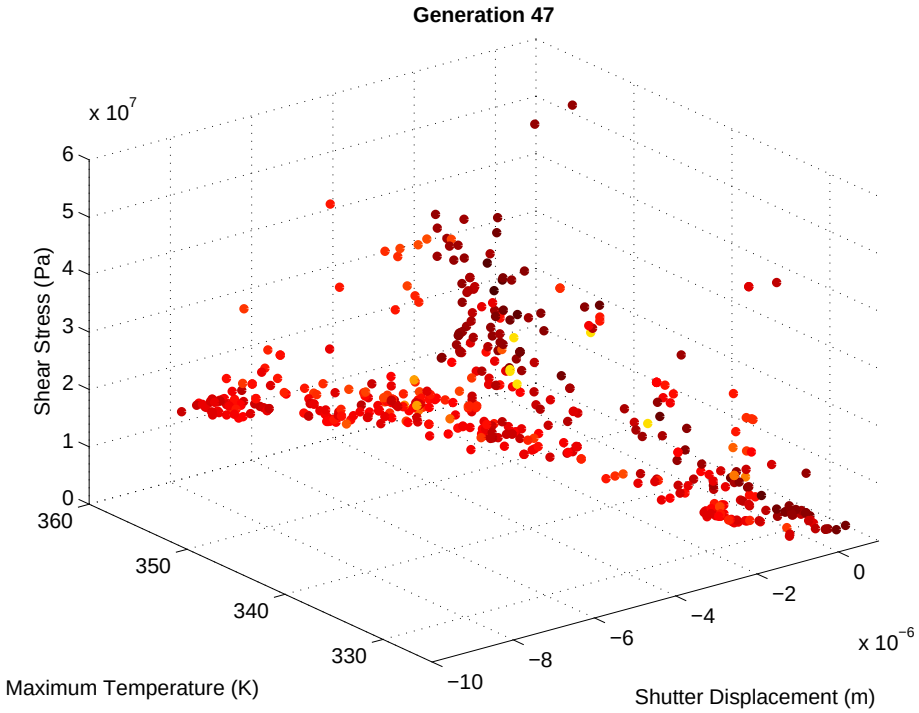

Generation 41
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6

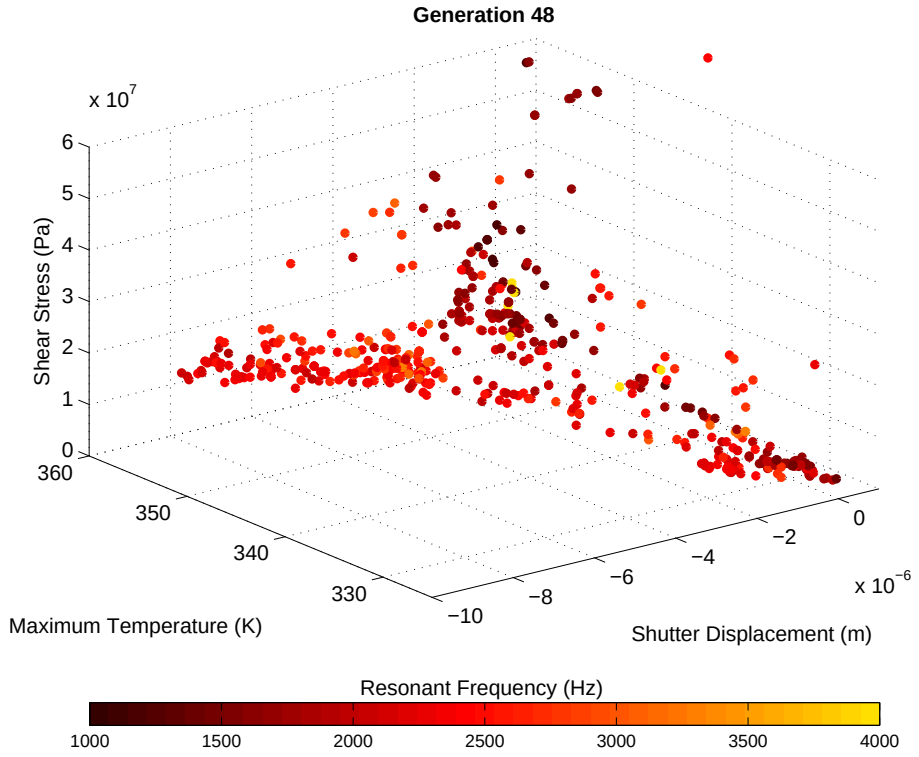

Generation 42
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Resonant Frequency (Hz)

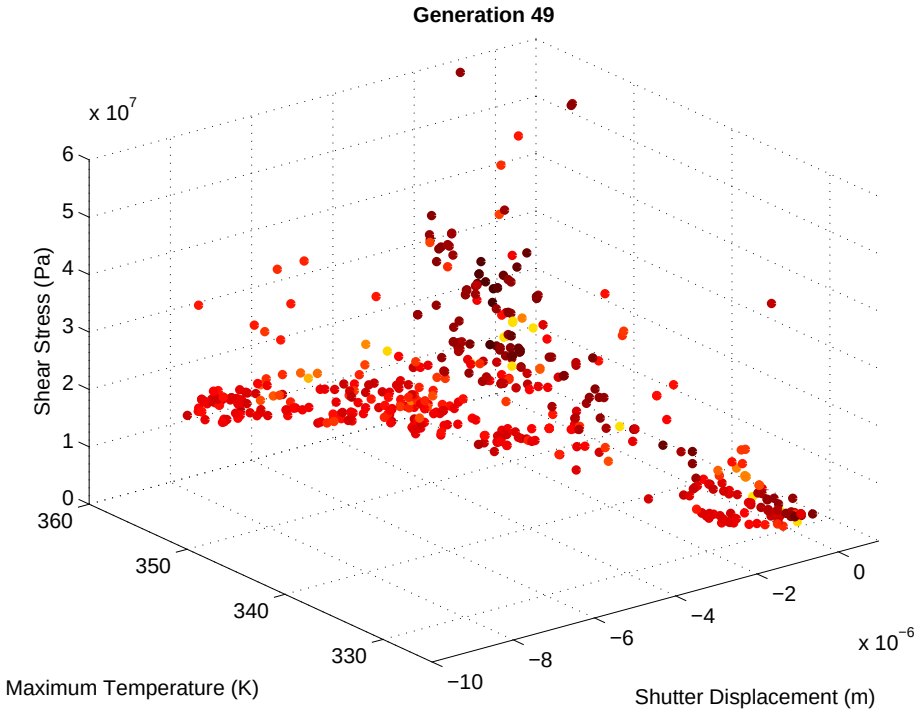

Generation 43
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Generation 44
x 10^7
Shear Stress (Pa)
Maximum Temperature (K)
Shutter Displacement (m)
x 10^-6
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

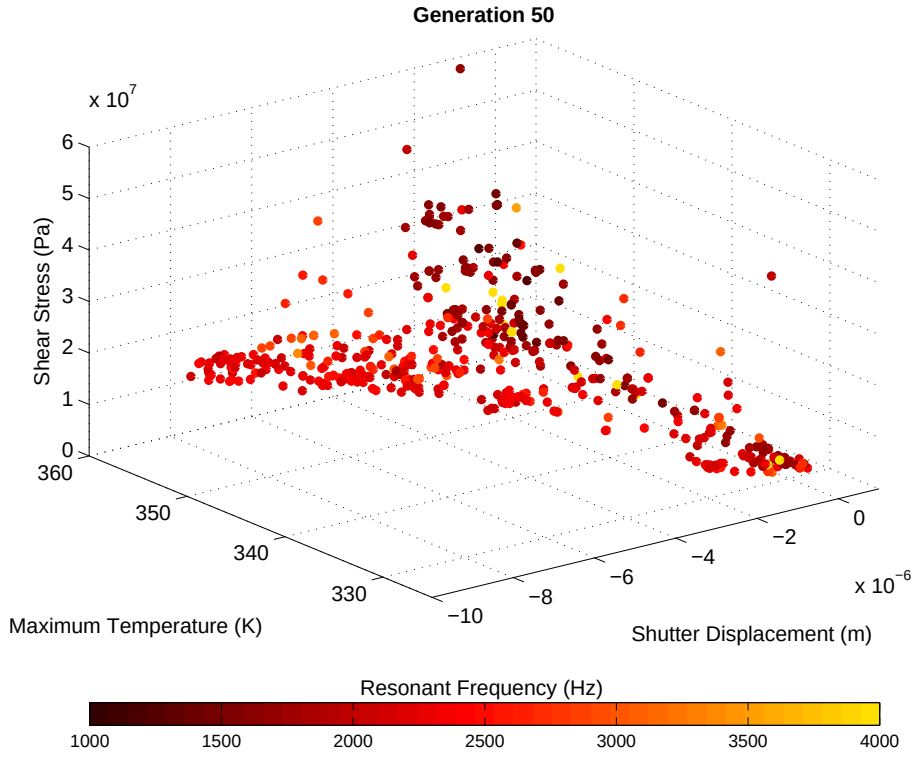

Generation 45
x 10^7
6
5
4
3
2
1
0
Shear Stress (Pa)
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)


Generation 46
x 10^7
6
5
4
3
2
1
0
Shear Stress (Pa)
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000


Generation 47
x 10^7
Shear Stress (Pa)
6
5
4
3
2
1
0
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)


Generation 48
x 10^7
Shear Stress (Pa)
6
5
4
3
2
1
0
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000


Generation 49
x 10^7
6
5
4
3
2
1
0
Shear Stress (Pa)
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)


Generation 50
x 10^7
6
5
4
3
2
1
0
Shear Stress (Pa)
360
350
340
330
Maximum Temperature (K)
−10
−8
−6
−4
−2
0
x 10^−6
Shutter Displacement (m)
Resonant Frequency (Hz)
1000
1500
2000
2500
3000
3500
4000

Appendix B: MATLAB Code for GA

sensor_genome.mat

```
classdef sensorGenome < hgsetget
    properties (Constant = true, GetAccess = private)

        voltage_range = 0.05:.01:3;

        % actuator parameter ranges
        thickness_range = 3:20;
        act_bm_range=1:30;
        act_bm_spc_range=5:20;
        act_bm_l_range = 50:300;
        act_bm_w_range = 3:6;
        act_bm_ang_range = 1:0.1:10;
        act_sh_w_range = 5:20;

        % lever parameter ranges
        lev_cbm_w_range = 3:10;
        lev_cbm_l_range = 75:200;
        lev_sbm_w_range = 3:10;
        lev_sbm_l_range = 10:200;
        lev_lbm_w_range = 3:20;
        lev_lbm_l_range = 100:2000;
        lev_cspr_tw_range = 7:30;
        lev_cspr_th_range = 15:200;
        lev_cpart_w_range = 1:20;
        lev_cpart_h_range = 3:20;

        % spring parameter ranges
        spr_no_range = 1:3;
        spr_th_range = 4:10;
        spr_tl_range = 200:1400;
        spr_h_w_range = 3:50;
        spr_end_th_range = 3:50;
        spr_bm_w_range = 3:50;
        spr_shut_l_range = 5:50;
        spr_spr_l_range = 5:50;
        spr_anc_l_range = 5:50;
    end

    properties (GetAccess = private)
        id;
        fem;

        shutter_displacement;
        max_shear_stress;
        eigenfreq;
        max_temp;
        max_shutter_temp;
```

```
48
49          voltage=0.5;
50      end
51
52      properties
53          % actuator parameters
54          thickness;
55          act_bm;
56          act_bm_spc;
57          act_bm_l;
58          act_bm_w;
59          act_bm_ang;
60          act_sh_w;
61
62          % lever parameters
63          lev_cbm_w;
64          lev_cbm_l;
65          lev_sbm_w;
66          lev_sbm_l;
67          lev_lbm_w;
68          lev_lbm_l;
69          lev_cspr_tw;
70          lev_cspr_th;
71          lev_cpart_w;
72          lev_cpart_h;
73
74          % spring parameters
75          spr_no;
76          spr_th;
77          spr_tl;
78          spr_h_w;
79          spr_end_th;
80          spr_bm_w;
81          spr_shut_l;
82          spr_spr_l;
83          spr_anc_l;
84      end
85
86      methods
87          function obj = sensorGenome(i)
88              obj.id = i;
89
90              obj.thickness = sensorGenome.thickness_range(...
91                                  floor(rand()*length(sensorGenome.thickness_range))+1);
92              obj.act_bm = sensorGenome.act_bm_range(...
93                                  floor(rand()*length(sensorGenome.act_bm_range))+1);
94              obj.act_bm_spc = sensorGenome.act_bm_spc_range(...
95                                  floor(rand()*length(sensorGenome.act_bm_spc_range))+1);
96              obj.act_bm_l = sensorGenome.act_bm_l_range(...
97                                  floor(rand()*length(sensorGenome.act_bm_l_range))+1);
98              obj.act_bm_w = sensorGenome.act_bm_w_range(...
99                                  floor(rand()*length(sensorGenome.act_bm_w_range))+1);
100             obj.act_bm_ang = sensorGenome.act_bm_ang_range(...
101                                 floor(rand()*length(sensorGenome.act_bm_ang_range))+1);
```

```
102             obj.act_sh_w = sensorGenome.act_sh_w_range(...
103                                 floor(rand()*length(sensorGenome.act_sh_w_range))+1);
104
105             obj.lev_cbm_w = sensorGenome.lev_cbm_w_range(...
106                                 floor(rand()*length(sensorGenome.lev_cbm_w_range))+1);
107             obj.lev_cbm_l = sensorGenome.lev_cbm_l_range(...
108                                 floor(rand()*length(sensorGenome.lev_cbm_l_range))+1);
109             obj.lev_sbm_w = sensorGenome.lev_sbm_w_range(...
110                                 floor(rand()*length(sensorGenome.lev_sbm_w_range))+1);
111             obj.lev_sbm_l = sensorGenome.lev_sbm_l_range(...
112                                 floor(rand()*length(sensorGenome.lev_sbm_l_range))+1);
113             obj.lev_lbm_w = sensorGenome.lev_lbm_w_range(...
114                                 floor(rand()*length(sensorGenome.lev_lbm_w_range))+1);
115             obj.lev_lbm_l = sensorGenome.lev_lbm_l_range(...
116                                 floor(rand()*length(sensorGenome.lev_lbm_l_range))+1);
117             obj.lev_cspr_tw = sensorGenome.lev_cspr_tw_range(...
118                                 floor(rand()*length(sensorGenome.lev_cspr_tw_range))+1);
119             obj.lev_cspr_th = sensorGenome.lev_cspr_th_range(...
120                                 floor(rand()*length(sensorGenome.lev_cspr_th_range))+1);
121             obj.lev_cpart_w = sensorGenome.lev_cpart_w_range(...
122                                 floor(rand()*length(sensorGenome.lev_cpart_w_range))+1);
123             obj.lev_cpart_h = sensorGenome.lev_cpart_h_range(...
124                                 floor(rand()*length(sensorGenome.lev_cpart_h_range))+1);
125
126             % spring parameters
127             obj.spr_no = sensorGenome.spr_no_range(...
128                                 floor(rand()*length(sensorGenome.spr_no_range))+1);
129             obj.spr_th = sensorGenome.spr_th_range(...
130                                 floor(rand()*length(sensorGenome.spr_th_range))+1);
131             obj.spr_tl = sensorGenome.spr_tl_range(...
132                                 floor(rand()*length(sensorGenome.spr_tl_range))+1);
133             obj.spr_h_w = sensorGenome.spr_h_w_range(...
134                                 floor(rand()*length(sensorGenome.spr_h_w_range))+1);
135             obj.spr_end_th = sensorGenome.spr_end_th_range(...
136                                 floor(rand()*length(sensorGenome.spr_end_th_range))+1);
137             obj.spr_bm_w = sensorGenome.spr_bm_w_range(...
138                                 floor(rand()*length(sensorGenome.spr_bm_w_range))+1);
139             obj.spr_shut_l = sensorGenome.spr_shut_l_range(...
140                                 floor(rand()*length(sensorGenome.spr_shut_l_range))+1);
141             obj.spr_spr_l = sensorGenome.spr_spr_l_range(...
142                                 floor(rand()*length(sensorGenome.spr_spr_l_range))+1);
143             obj.spr_anc_l = sensorGenome.spr_anc_l_range(...
144                                 floor(rand()*length(sensorGenome.spr_anc_l_range))+1);
145
146             while (obj.act_sh_w/2 + obj.act_bm_l + 50 > obj.act_sh_w/2...
147                         - obj.lev_cbm_w/2 + obj.lev_lbm_l)
148                 obj.act_bm_l = sensorGenome.act_bm_l_range(...
149                                         floor(rand()*length(sensorGenome.act_bm_l_range))+1);
150                 obj.lev_lbm_l = sensorGenome.lev_lbm_l_range(...
151                                         floor(rand()*length(sensorGenome.lev_lbm_l_range))+1);
152                 fprintf('Genome %d actuator overlaps shutter. Randomizing actuator beam
153                         length and lever long beam lengths.\n', obj.id);
154             end
155         end
```

```
156
157         function shutter_displacement = getShutterDisp(obj)
158             shutter_displacement = obj.shutter_displacement;
159         end
160
161         function max_shutter_temp = getMaxShutterTemp(obj)
162             max_shutter_temp = obj.max_shutter_temp;
163         end
164
165         function max_temp = getMaxTemp(obj)
166             max_temp = obj.max_temp;
167         end
168
169         function max_stress = getMaxShearStress(obj)
170             max_stress = obj.max_shear_stress;
171         end
172
173         function eigenfreq = getEigenfreq(obj)
174             eigenfreq = obj.eigenfreq;
175         end
176
177         function fem = getFem(obj)
178             fem = obj.fem;
179         end
180
181         function obj = setFem(obj,fem)
182             obj.fem=fem;
183         end
184
185         function obj = evalFitness(obj)
186            [fem0, eigenfreq, max_temp, max_shutter_temp, shutter_displacement, ...
187                  max_shear_stress] = matlab_sensor_sym(obj);
188            obj.fem = fem0;
189            obj.eigenfreq = eigenfreq;
190            obj.max_temp = max_temp;
191            obj.max_shutter_temp = max_shutter_temp;
192            obj.shutter_displacement = shutter_displacement;
193            obj.max_shear_stress = max_shear_stress;
194
195            fprintf('Evaluating fitness of genome %d. Disp: %d Maxtemp: %d Voltage: %d\n', ...
196                 obj.id, obj.shutter_displacement, obj.max_temp, obj.voltage);
197         end
198
199         function obj = mutate(obj, prop, mutateFactor)
200             range = strcat(prop,'_range');
201             range = obj.get(range);
202             range= range{1};
203
204             val=obj.get(prop);
205             val=val{1};
206
207             set=0; newVal= 0 ;
208             while (set == 0 || newVal < min(range) || newVal > max(range))
209                 newVal = val+mutateFactor*(rand()-0.5)*(max(range) - min(range));
```

```
210                 set=1;
211             end
212             obj=obj.set(prop,{newVal});
213         end
214
215         function [childA childB] = crossover(obj, mate, cxPt)
216             childA = sensorGenome(0);
217             childB = sensorGenome(0);
218             props= properties('sensorGenome');
219
220             for i=1:length(props)
221                 if (i < cxPt)
222                     childA.set(props(i),obj.get(props(i)));
223                     childB.set(props(i),mate.get(props(i)));
224                 else
225                     childA.set(props(i),mate.get(props(i)));
226                     childB.set(props(i),obj.get(props(i)));
227                 end
228             end
229         end
230
231         function obj = setId(obj, id)
232             obj.id=id;
233         end
234
235         function ret = gt(a,b)
236            ret= (a.eigenfreq > b.eigenfreq) && ...
237                 (a.max_temp < b.max_temp) && ...
238                 (a.max_shutter_temp < b.max_shutter_temp) && ...
239                 (a.shutter_displacement > b.shutter_displacement) && ...
240                 (a.max_shear_stress < b.max_shear_stress);
241         end
242
243         function ret = lt(a,b)
244            ret= (a.eigenfreq < b.eigenfreq) && ...
245                 (a.max_temp > b.max_temp) && ...
246                 (a.max_shutter_temp > b.max_shutter_temp) && ...
247                 (a.shutter_displacement < b.shutter_displacement) && ...
248                 (a.max_shear_stress > b.max_shear_stress);
249         end
250     end
251 end
```

population.mat

```
classdef population
    properties

        eigenfreq_max = 5000;
        eigenfreq_min = 500;

        maxtemp_max = 1000;
        maxtemp_min = 293;

        shuttertemp_max = 800;
        shuttertemp_min = 293;

        maxstress_max = 4e9;
        maxstress_min = 1e6;

        displacement_max = 25e-6;
        displacement_min = 0;

        pop;
    end

    methods
        function obj = population(n)
            clear pop;;
            for i=1:n
                pop(i)=sensorGenome(i);
            end
            obj.pop=pop;
        end

             function attributes = getAttributes(obj,i)
             attributes = [ (-obj.pop(i).getShutterDisp() - obj.displacement_min) /...
            (obj.displacement_max - obj.displacement_min) ...
            (obj.pop(i).getShutterTemp() - obj.shuttertemp_min) /...
            (obj.shuttertemp_max - obj.shuttertemp_min) ...
            (obj.pop(i).getMaxTemp() - obj.maxtemp_min) /...
            (obj.maxtemp_max - obj.maxtemp_min) ...
            (obj.pop(i).getEigenfreq() - obj.eigenfreq_min)/ ...
            (obj.eigenfreq_max - obj.eigenfreq_min) ...
            (obj.pop(i).getMaxShearStress() - obj.maxstress_min) / ...
            (obj.maxstress_max - obj.maxstress_min) ] ;
        end

        function count = nicheCount(obj,i, shareRadius)
            a = obj.getAttributes(i);
            count = 0;
            for b=1:length(obj.pop)
                if sqrt(sum((a-obj.getAttributes(b)).^2)) < shareRadius
                    count = count + 1;
                end
            end
        end
```

```
53
54          function winner = paretoTournament(obj,tdom,shareRadius)
55              rand_index = randperm(length(obj.pop));
56              candidate_1 = obj.pop(rand_index(1));
57              candidate_2 = obj.pop(rand_index(2));
58              candidate_1_dominated = false;
59              candidate_2_dominated = false;
60
61              for i=3:3+tdom
62                  if (obj.pop(i) > candidate_1)
63                      candidate_1_dominated = true;
64                  end
65                  if (obj.pop(i) > candidate_2)
66                      candidate_2_dominated = true;
67                  end
68              end
69
70              if (candidate_1_dominated && ~candidate_2_dominated)
71                  winner = candidate_2;
72              elseif (~candidate_1_dominated && candidate_2_dominated)
73                  winner = candidate_1;
74              elseif ( obj.nicheCount(rand_index(1),shareRadius) < ...
75                   obj.nicheCount(rand_index(2),shareRadius) )
76                  winner = candidate_1;
77              else
78                  winner = candidate_2;
79              end
80          end
81
82          function obj = crossover(obj)
83              n=length(obj.pop);
84              props = properties('sensorGenome');
85
86              clear selectedPop newPop
87              j=1; k=1;
88               % select from population members to crossover.
89             % Selected genomes are also cloned into the new population.
90              for i=1:n
91                  if (rand() < normFitness(i) / sum(normFitness(1:i)) && ...
92                  obj.pop(i).getShutterDisp() ~= 0)
93                      selectedPop(j)=obj.pop(i);
94                      newPop(k)=obj.pop(i);
95                      k=k+1;
96                      j=j+1;
97                  end
98              end
99
100             j=length(selectedPop);  % crossover until we have replenished the population
101             while (k <= n)
102                 i=round(rand()*(j-1))+1;
103                 mate = round(rand()*(j-1))+1;
104                 cxPt = round(rand()*(length(props)-1))+1;
105                 [childA childB] = selectedPop(i).crossover(selectedPop(mate),cxPt);
106
```

```
107                 newPop(k)=childA;
108                 newPop(k+1)=childB;
109                 k=k+2;
110             end
111
112             newPop=newPop(1:n);
113
114             for i=1:n
115                 newPop(i).setId(i);
116             end
117             obj.pop=newPop;
118         end
119
120         function obj = mutate(obj,mutateFactor,mutateOdds)
121             n=length(obj.pop);
122             props = properties('sensorGenome');
123             for i=1:n
124                 for j=1:length(props)
125                     if (rand() < mutateOdds)
126                         obj.pop(i).mutate(props(j),mutateFactor);
127                     end
128                 end
129             end
130         end
131
132         function obj = eval(obj)
133             obj=obj.fixGeomErrors();
134             pop_size = length(obj.pop);
135             for i=1:pop_size;
136                 obj.pop(i)=obj.pop(i).evalFitness();
137             end
138             obj=obj.sort();
139         end
140
141         function obj = fixGeomErrors(obj)
142             n=length(obj.pop);
143             for i=1:n
144                 while (obj.pop(i).act_sh_w/2 + obj.pop(i).act_bm_l + 50 > ...
145                         obj.pop(i).act_sh_w/2 - obj.pop(i).lev_cbm_w/2 + obj.pop(i).lev_lbm_l)
146                     obj.pop(i)=sensorGenome(i);
147                 end
148             end
149         end
150     end
151 end
```

A simple example implementing the above classes to run a genetic algorithm optimization:

```
for i = 1:10  % Run for generations 1 - 10;
    if i>1
        pop(i)=pop(i-1).crossover();
        pop(i)=pop(i).mutate(0.4,0.3);
    else
        pop(i)=population(100);
    end
    pop(i)=pop(i).eval();
end
```

Appendix C: MATLAB/COMSOL Code for Simulation of Field Mill

matlab_sensor_sym.mat

```
function [fem0, eigenfreq, max_temp, max_shutter_temp, shutter_displacement, max_shear_stress] =
         matlab_sensor_sym(obj)

%% simulation parameters
actuator_voltage = 0.39; % (volts)

au_si_resistivity = 1/1.275e-5; % (resistivity of gold on silicon)
si_resistivity = 1/5e-1; % (resistivity of silicon only)
si_conductivity = 150; % W/mk
si_density = 2330;

youngs_modulus = 1.295e11;
poissons_ratio = 0.22;
thermal_expansion_coef = 2.9e-6;
thickness = obj.thickness*1e-6;

temp_ref = 298;

%% Actuator Parameters
act_bm        = round(obj.act_bm); %5  % beam count
act_bm_spc    = obj.act_bm_spc*1e-6; %12e-6;   % beam spacing
act_bm_l      = obj.act_bm_l*1e-6; %190e-6; % beam length
act_bm_w      = obj.act_bm_w*1e-6; %6e-6;   % beam width
act_bm_ang    = obj.act_bm_ang*pi/180; %4.5*pi/180;    % beam angle (rad)
act_sh_w      = obj.act_sh_w*1e-6; %20e-6; %shuttle width
act_anc_w     = 50e-6;

%% Lever Parameters
lev_cbm_w =  obj.lev_cbm_w*1e-6; %5e-6; % connecting beam width
lev_cbm_l =  obj.lev_cbm_l*1e-6; %95e-6; % connecting beam length

lev_sbm_w =  obj.lev_sbm_w*1e-6; %3e-6; % short beam width
lev_sbm_l =  obj.lev_sbm_l*1e-6; %20e-6; % short beam length

lev_lbm_w =  obj.lev_lbm_w*1e-6; %4e-6; %long beam width
lev_lbm_l =  obj.lev_lbm_l*1e-6; %375e-6; %long beam length

lev_cspr_tw = obj.lev_cspr_tw*1e-6; %10e-6; %connecting spring total width
lev_cspr_th = obj.lev_cspr_th*1e-6; %60e-6;%connecting spring total height
lev_cspr_w  = 3e-6;%connecting spring width

lev_cpart_w = obj.lev_cpart_w*1e-6;% 10e-6; %connecting part width (after spring)
lev_cpart_h = obj.lev_cpart_h*1e-6; %9e-6; %connecting part length (after spring)

lev_anc_size =  100e-6; % size of anchor (square)

```

```
%% Shutter Parameters
shut_tw   = 2000e-6;  % shutter total width
shut_th   = 1000e-6;  % shutter total height

shut_h_w  = 975e-6; % shutter hole width
shut_h_h  = 14e-6;  % shutter hole height
shut_h_spc = 14e-6;  % shutter hole spacing

%% Spring Parameters
spr_no =  round(obj.spr_no); %2; % number of springs per corner

spr_tw = (obj.spr_h_w+2*obj.spr_th)*1e-6; %21e-6; % spring total width
spr_tl = obj.spr_tl*1e-6; %560e-6; % spring total length
spr_h_w = obj.spr_h_w*1e-6;  %15e-6; % spring hole width
spr_h_l = (obj.spr_tl - 2*obj.spr_end_th)*1e-6; %520e-6; % spring hole length

spr_bm_w = obj.spr_bm_w*1e-6; %20e-6; % spring bm connector width
spr_shut_l = obj.spr_shut_l*1e-6; %20e-6; % spring-shutter connector length
spr_spr_l =  obj.spr_spr_l*1e-6; %15e-6; % spring-spring connector length
spr_anc_l =  obj.spr_anc_l*1e-6; %25e-6;% spring-anchor connector length

%% Build Actuators
act_sh_anc = cos(act_bm_ang)*act_bm_l; % seperatation between shuttle and anchor
act_bm_hyp = act_bm_w / cos(act_bm_ang); % beam hypotenuse
act_sh_l = act_bm_spc+act_bm*(act_bm_spc+act_bm_hyp);  % length of shuttle

actuator_anchors_l = rect2(act_anc_w, (act_bm+1)*(act_bm_spc+act_bm_w),'base','corner','pos',...
                                [-act_sh_anc-act_anc_w, sin(act_bm_ang)*act_bm_l]);
actuator_anchors_r = rect2(act_anc_w, (act_bm+1)*(act_bm_spc+act_bm_w),'base','corner','pos',...
                                [act_sh_anc+act_sh_w,  sin(act_bm_ang)*act_bm_l]);
shuttle = rect2(act_sh_w, act_sh_l,'base','corner','pos',[0, 0]);

act_beam = line2( [0,0,-act_sh_anc,-act_sh_anc,0], [act_bm_spc, act_bm_spc + ...
                  act_bm_hyp, act_bm_spc + act_bm_hyp...
                  +sin(act_bm_ang)*act_bm_l, act_bm_spc...
                  + sin(act_bm_ang)*act_bm_l, act_bm_spc]);

act_beams = geomcomp(geomarrayr(act_beam, 0, act_bm_spc + act_bm_hyp, 1, act_bm));
act_beams = act_beams + mirror(act_beams, [act_sh_w / 2, 0], [1, 0]);
actuator = geomdel(act_beams+shuttle);

%% Build Levers / Connecting Arms
lev_cbm = rect2(lev_cbm_w,lev_cbm_l,'base','corner','pos',[act_sh_w/2 - lev_cbm_w / 2, act_sh_l])
lev_sbm = rect2(lev_sbm_l,lev_sbm_w,'base','corner','pos',[act_sh_w/2 - lev_cbm_w / 2 ...
               - lev_sbm_l, act_sh_l + lev_cbm_l + lev_lbm_w/2 - lev_sbm_w/2]);

lev_anchor = rect2(lev_anc_size,lev_anc_size,'base','corner','pos',[act_sh_w/2 ...
                  - lev_cbm_w /2 - lev_sbm_l - lev_anc_size, act_sh_l + lev_cbm_l ...
                  + lev_lbm_w/2 + lev_cbm_w/2 - lev_anc_size/2]);

lev_lbm = rect2(lev_lbm_l,lev_lbm_w,'base','corner','pos',[act_sh_w/2...
               - lev_cbm_w /2, act_sh_l + lev_cbm_l]);

lev_cspr = rect2(lev_cspr_tw, lev_cspr_th,'base','corner','pos',[act_sh_w/2 ...
```

```
                - lev_cbm_w/2 + lev_lbm_l, act_sh_l + lev_cbm_l + lev_lbm_w/2 - lev_cspr_th/2 ]);

lev_cspr_hole =   rect2(lev_cspr_tw-2*lev_cspr_w, lev_cspr_th-2*lev_cspr_w, ...
                        'base','corner','pos',[act_sh_w/2 - lev_cbm_w/2 + lev_lbm_l ...
                        + lev_cspr_w, act_sh_l + lev_cbm_l + lev_lbm_w/2 ...
                        - lev_cspr_th/2 + lev_cspr_w ]);

lev_cpart = rect2(lev_cpart_w,lev_cpart_h,'base','corner','pos',[act_sh_w/2 ...
                 - lev_cbm_w/2 + lev_lbm_l + lev_cspr_tw, act_sh_l + lev_cbm_l ...
                 + lev_lbm_w/2 - lev_cpart_h/2]);

lever = geomdel(lev_cbm + lev_sbm + lev_lbm + lev_cspr - lev_cspr_hole + lev_cpart);

lever_x = act_sh_w/2 - lev_cbm_w/2 + lev_lbm_l + lev_cspr_tw + lev_cpart_w;
lever_y = act_sh_l + lev_cbm_l + lev_lbm_w/2;

%% Build Shutter
shut_num_hole = floor(shut_th / (shut_h_h+shut_h_spc));
shut_h_yoffset = (shut_th - shut_num_hole*(shut_h_h + shut_h_spc)-shut_h_spc)/2;

shutter = rect2(shut_tw/2,shut_th,'base','corner','pos',[lever_x,lever_y-shut_th/2]);

hole = rect2(shut_h_w,shut_h_h,'base','corner','pos',[lever_x+shut_tw/4 ...
          - shut_h_w/2,lever_y-shut_th/2+shut_h_spc+shut_h_yoffset]);

holes = geomcomp(geomarrayr(hole, shut_tw/2, shut_h_spc + shut_h_h, 2, shut_num_hole));
shutter = shutter - holes;

%% Build Springs
spr_shut = rect2(spr_bm_w,spr_shut_l,'base','corner','pos',[lever_x,lever_y-shut_th/2-spr_shut_l
spr = rect2(spr_tl,spr_tw,'base','corner','pos',[lever_x+spr_bm_w/2-spr_tl/2,...
        lever_y-shut_th/2-spr_shut_l-spr_tw]);

spr = spr - rect2(spr_h_l,spr_h_w,'base','corner','pos',[lever_x+spr_bm_w/2...
                -spr_tl/2+(spr_tl-spr_h_l)/2, lever_y-shut_th/2-spr_shut_l...
                -(spr_tw-spr_h_w)/2-spr_h_w]);

spr = geomcomp(geomarrayr(spr, 0, -spr_tw-spr_spr_l, 1, spr_no));

spr_spr = rect2(spr_bm_w,spr_spr_l,'base','corner','pos',[lever_x,lever_y-shut_th/2...
              -spr_shut_l-spr_tw-spr_spr_l]);

spr_anc = rect2(spr_bm_w,spr_anc_l,'base','corner','pos',[lever_x,lever_y-shut_th/2...
               -spr_shut_l-spr_no*(spr_tw+spr_spr_l) + spr_spr_l - spr_anc_l ]);

spr_anchor = rect2(lev_anc_size,lev_anc_size,'base','corner','pos',[lever_x...
                    + spr_bm_w/2 - lev_anc_size/2 ,lever_y-shut_th/2-spr_shut_l...
                    -spr_no*(spr_tw+spr_spr_l) + spr_spr_l - spr_anc_l - lev_anc_size]);

if (spr_no > 1)
     spr_spr = geomcomp(geomarrayr(spr_spr,0,-spr_tw-spr_spr_l, 1, spr_no-1));
     springs = geomdel(spr_shut+spr+spr_spr+spr_anc);
else
     springs = geomdel(spr_shut+spr+spr_anc);
```

```
end

springs = springs + mirror(springs, [0, lever_y], [0, 1]);
spring_anchors = spr_anchor + mirror(spr_anchor, [0, lever_y], [0, 1]);


[g,st] = geomcsg({actuator_anchors_l, actuator_anchors_r,actuator,lev_anchor,...
                          lever,shutter,springs,spring_anchors});

ctx = geomcsg({actuator_anchors_l, actuator_anchors_r,actuator,...
                       lev_anchor,lever,shutter,springs,spring_anchors},'Out','ctx');

fem.geom = g;
fem.mesh = meshinit(fem,'methodsub','tri','Hmax',0.4);

%% find subdomains and boundaries
[subdomains,objid] = find(st);

sd_obj_ind = ones(length(objid),1);
clear sd_obj;
for j=1:size(subdomains)
    sd_obj(objid(j),sd_obj_ind(objid(j)))=subdomains(j);
    sd_obj_ind(objid(j))=sd_obj_ind(objid(j))+1;
end

act_ancl_sd = sd_obj(1,1:sd_obj_ind(1)-1);
[act_ancl_bd,junk] = find(ctx{1});

act_ancr_sd = sd_obj(2,1:sd_obj_ind(2)-1);
[act_ancr_bd,junk] = find(ctx{2});

act_sd = sd_obj(3,1:sd_obj_ind(3)-1);
[act_bd,junk] = find(ctx{3});

lev_anc_sd = sd_obj(4,1:sd_obj_ind(4)-1);
[lev_anc_bd,junk] = find(ctx{4});

lev_sd = sd_obj(5,1:sd_obj_ind(5)-1);
[lev_bd,junk] = find(ctx{5});

shut_sd = sd_obj(6,1:sd_obj_ind(6)-1);
[shut_bd,junk] = find(ctx{6});

symmetry_bd = max(shut_bd);
shut_bd = shut_bd(1:length(shut_bd)-1);

spr_sd = sd_obj(7,1:sd_obj_ind(7)-1);
[spr_bd,junk] = find(ctx{7});

spr_anc_sd = sd_obj(8,1:sd_obj_ind(8)-1);
[spr_anc_bd,junk]=find(ctx{8});

tmp = geomcsg({fem.geom},'Out','ctx');
[allboundaries,junk]=find(tmp{1});
```

```
num_bd = max(allboundaries);

% find interior boundaries.
% int. boundaries are defined as boundaries belonging to one or more
% objects.
bd_freq = zeros(1,num_bd);
num_int =0;
for i=1:length(ctx)
    [bd,junk]= find(ctx{i});
    for j=1:length(bd)
        if (bd_freq(bd(j)) == 1)
            num_int = num_int+1;
        end
        bd_freq(bd(j))=bd_freq(bd(j))+1;
    end
end
int_bd = zeros(1,num_int);
k=1;
for i=1:num_bd
    if (bd_freq(i) > 1)
        int_bd(k)=i;
        k=k+1;
    end
end

all_sd = subdomains;
clear allboundaries tmp ctx num_int i k junk sd_obj sd_obj_ind j subdomains objid;

%% Application Mode 1: Conductive DC Media
clear appl;
appl.mode.class = 'EmConductiveMediaDC';
appl.module = 'ACDC';
appl.assign_siffix = '_emdc';
clear prop;
clear weakconstr;
weakconstr.value = 'off';
weakconstr.dim = {'lm4'};
prop.weakconstr = weakconstr;
appl.prop = prop;

clear bnd

% keep interior boundaries on the actuator anchors but not in int_bd
tmp = zeros(1,length(int_bd));
k=0;
for i=1:length(int_bd)
    include = true;
    for j=1:length(act_ancl_bd)
       if (act_ancl_bd(j) == int_bd(i))
           include = false;
       end
    end
    for j=1:length(act_ancr_bd)
        if (act_ancr_bd(j) == int_bd(i))
```

```
263             include = false;
264         end
265     end
266     if (include)
267         k=k+1;
268         tmp(k)=int_bd(i);
269     end
270 end
271 int_keep_bd = tmp(1:k);
272
273 % find all boundaries not included in actuator boundaries or interior
274 % boundaries
275 tmp = ones(1,num_bd);
276 for i=1:length(act_ancl_bd)
277     tmp(act_ancl_bd(i)) = 0;
278 end
279 for i=1:length(act_ancr_bd)
280     tmp(act_ancr_bd(i)) = 0;
281 end
282 for i=1:length(int_keep_bd)
283     tmp(int_keep_bd(i)) = 0;
284 end
285 other_bd = zeros(1,num_bd-length(act_ancl_bd) - length(act_ancr_bd) - length(int_keep_bd));
286 k=1;
287 for i=1:num_bd
288     if (tmp(i) == 1)
289         other_bd(k) = i;
290         k=k+1;
291     end
292 end
293 clear k i tmp;
294
295 %boundary settings.  boundaries grouped by actuator ancors (left and
296 %right), interior boundaries (continuous) and all other boundaries
297 %(electric insulator)
298 bnd.V = {actuator_voltage,-actuator_voltage,0,0};
299 bnd.type = {'V','V','nJ0','cont'};
300 bnd.ind = {act_ancl_bd,act_ancr_bd,other_bd,int_keep_bd};
301 appl.bnd = bnd;
302
303 clear equ;
304 %group subdomains by actuator left anchors (where +voltage is applied),
305 %actuator right anchors (where -voltage is applied),
306 %gold-on-silicon structure, and silicon only structure.
307 equ.ind = { act_ancl_sd, act_ancr_sd, cat(2, act_sd, shut_sd ), ...
308     cat(2, lev_anc_sd, lev_sd, spr_anc_sd, spr_sd) };
309 equ.init = {actuator_voltage,-actuator_voltage,0, 0};
310 equ.sigma = { au_si_resistivity, au_si_resistivity, au_si_resistivity, si_resistivity };
311 equ.d = thickness;
312 appl.equ = equ;
313 equ.T = 'T';
314 fem.appl{2} = appl;
315
316 %% Application Mode 2: Joule Heating
```

```
clear appl
appl.mode.class = 'GeneralHeat';
appl.module = 'HT';
appl.shape = {'shlag(1,''J'')','shlag(2,''T'')'};
appl.assignsuffix = '_htgh';
clear prop
prop.analysis='static';
appl.prop = prop;

%subdomain settings
clear equ
equ.k = si_conductivity;
equ.rho = si_density;
equ.Q = 'Q_emdc';
equ.shape = 2;
equ.name = 'default';
equ.ind = {all_sd};
appl.equ = equ;

%boundary settings
% keep interior boundaries on the actuator anchors but not in int_bd
tmp = zeros(1,length(int_bd));
k=0;
for i=1:length(int_bd)
    include = true;
    for j=1:length(act_ancl_bd)
       if (act_ancl_bd(j) == int_bd(i))
           include = false;
       end
    end
    for j=1:length(act_ancr_bd)
        if (act_ancr_bd(j) == int_bd(i))
            include = false;
        end
    end
      for j=1:length(lev_anc_bd)
        if (lev_anc_bd(j) == int_bd(i))
            include = false;
        end
      end
       for j=1:length(spr_anc_bd)
        if (spr_anc_bd(j) == int_bd(i))
            include = false;
        end
    end
    if (include)
        k=k+1;
        tmp(k)=int_bd(i);
    end
end
int_keep_bd = tmp(1:k);

% find all boundaries not included in actuator boundaries or interior
% boundaries or symmetry boundary
```

```
tmp = ones(1,num_bd);
tmp(symmetry_bd) = 0;
for i=1:length(act_ancl_bd)
    tmp(act_ancl_bd(i)) = 0;
end
for i=1:length(act_ancr_bd)
    tmp(act_ancr_bd(i)) = 0;
end
for i=1:length(lev_anc_bd)
    tmp(lev_anc_bd(i)) = 0;
end
for i=1:length(spr_anc_bd)
    tmp(spr_anc_bd(i)) = 0;
end
for i=1:length(int_keep_bd)
    tmp(int_keep_bd(i)) = 0;
end
other_bd = zeros(1,num_bd-length(act_ancl_bd) - length(act_ancr_bd)...
                  - length(lev_anc_bd) - length(spr_anc_bd) - length(int_keep_bd)-1);
k=1;
for i=1:num_bd
    if (tmp(i) == 1)
        other_bd(k) = i;
        k=k+1;
    end
end

clear bnd
bnd.shape = 1;
bnd.T0 = { temp_ref, temp_ref, temp_ref, temp_ref, temp_ref, temp_ref,  temp_ref};
bnd.type = {'T','T','T','T','cont','q0','q0'};
bnd.ind = { act_ancl_bd,act_ancr_bd, lev_anc_bd, spr_anc_bd, int_keep_bd, other_bd, symmetry_bd
appl.bnd = bnd;
fem.appl{1} = appl;

%% Application Mode 3: Plane Strain
clear appl
appl.mode.class = 'SmePlaneStrain';
appl.module = 'MEMS';
appl.gporder = 4;
appl.cporder = 2;
appl.assignsuffix = '_smpn';

clear prop
prop.largedef='on';
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm3','lm4'};
prop.weakconstr = weakconstr;
appl.prop = prop;

%boundary settings. actuators and ancors = fixed, everything else = free.
clear bnd

```

```
% keep interior boundaries on the anchors but not in int_bd
tmp = zeros(1,length(int_bd));
k=0;
for i=1:length(int_bd)
    include = true;
    for j=1:length(act_ancl_bd)
       if (act_ancl_bd(j) == int_bd(i))
           include = false;
       end
    end
    for j=1:length(act_ancr_bd)
        if (act_ancr_bd(j) == int_bd(i))
            include = false;
        end
    end
    for j=1:length(lev_anc_bd)
        if (lev_anc_bd(j) == int_bd(i))
            include = false;
        end
    end
    for j=1:length(spr_anc_bd)
        if (spr_anc_bd(j) == int_bd(i))
            include = false;
        end
    end
    if (include)
        k=k+1;
        tmp(k)=int_bd(i);
    end
end
int_keep_bd = tmp(1:k);

% find all boundaries not included in anchor boundaries or interior
% boundaries or symmetry boundary
tmp = ones(1,num_bd);
tmp(symmetry_bd) = 0;
for i=1:length(act_ancl_bd)
    tmp(act_ancl_bd(i)) = 0;
end
for i=1:length(act_ancr_bd)
    tmp(act_ancr_bd(i)) = 0;
end
for i=1:length(lev_anc_bd)
    tmp(lev_anc_bd(i)) = 0;
end
for i=1:length(spr_anc_bd)
    tmp(spr_anc_bd(i)) = 0;
end
for i=1:length(int_keep_bd)
    tmp(int_keep_bd(i)) = 0;
end
other_bd = zeros(1,num_bd-length(act_ancl_bd) - length(act_ancr_bd)...
                  - length(lev_anc_bd) - length(spr_anc_bd) - length(int_keep_bd) -1);
k=1;
```

```
for i=1:num_bd
    if (tmp(i) == 1)
        other_bd(k) = i;
        k=k+1;
    end
end

clear k i tmp;

bnd.constrcond = {'fixed','fixed','fixed','fixed','free','free','sym'};
bnd.ind = {act_ancl_bd, act_ancr_bd, lev_anc_bd, spr_anc_bd, other_bd, int_keep_bd, symmetry_bd}
appl.bnd = bnd;

clear equ
equ.Tflag = 1;
equ.Tempref = temp_ref;
equ.Temp = 'T';
equ.rho = si_density;
equ.alpha = thermal_expansion_coef;
equ.constrcond = {'fixed','free'};
equ.nu = poissons_ratio;
equ.thickness = thickness;
equ.E = youngs_modulus;
equ.constrcond = {'fixed','fixed','fixed','fixed','free','free','free','free'};
equ.ind = {act_ancl_sd, act_ancr_sd, lev_anc_sd, spr_anc_sd, act_sd, lev_sd, shut_sd, spr_sd} ;
appl.equ = equ;
fem.appl{3} = appl;

fem = multiphysics(fem);
fem.xmesh=meshextend(fem);

% Solve problem
[fem.sol, stop] =femstatic(fem, ...
                  'solcomp',{'u','T','V','v'}, ...
                  'outcomp',{'u','T','V','v'}, ...
                  'rowscale','off','Out',{'sol','stop'});
if stop == 0

    pd=posteval(fem,'v','Dl',shut_bd(1), 'Edim',1);
    shutter_displacement = min(pd.d);
    pd=posteval(fem,'T','Dl',shut_bd(1), 'Edim', 1);
    max_shutter_temp = max(pd.d);
    max_temp =postmax(fem,'T');
    max_shear_stress = max( abs(postmax(fem,'sxy_smpn')), abs(postmin(fem,'sxy_smpn')));

else
    shutter_displacement = 0;
    max_temp = 10000;
    max_shutter_temp = 10000;
    max_shear_stress =-10e9;
end

fem0 = fem;

```

```
%% Application Mode 1: Plane Strain
clear appl
appl.mode.class = 'SmePlaneStrain';
appl.module = 'MEMS';
appl.gporder = 4;
appl.cporder = 2;
appl.assignsuffix = '_smpn';

clear prop
prop.analysis='eigen';
prop.largedef='on';
clear weakconstr
weakconstr.value = 'off';
weakconstr.dim = {'lm3','lm4'};
prop.weakconstr = weakconstr;
appl.prop = prop;

%boundary settings. actuators and ancors = fixed, everything else = free.
clear bnd
bnd.constrcond = {'fixed','fixed','fixed','fixed','free','free','sym'};
bnd.ind = {act_ancl_bd, act_ancr_bd, lev_anc_bd, spr_anc_bd, other_bd, int_keep_bd, symmetry_bd
appl.bnd = bnd;

clear equ
equ.Tempref = temp_ref;
equ.Temp = 'T';
equ.rho = si_density;
equ.alpha = thermal_expansion_coef;
equ.nu = poissons_ratio;
equ.thickness = thickness;
equ.E = youngs_modulus;
equ.constrcond = {'fixed','fixed','fixed','fixed','free','free','free','free'};
equ.ind = {act_ancl_sd, act_ancr_sd, lev_anc_sd, spr_anc_sd, act_sd, lev_sd, shut_sd, spr_sd} ;
appl.equ = equ;
fem.appl{1} = appl;
fem.appl(2:end)=[];
fem.border = 1;

% Multiphysics
fem=multiphysics(fem);
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femeig(fem, ...
               'solcomp',{'v','u'}, ...
               'outcomp',{'v','u'}, ...
               'blocksize','auto');

eigenfreq = abs(fem.sol.lambda(1))/(2*pi);

end
```

Appendix D: Heater Controller

As described earlier, the heater controller is used to regulate the temperature of the bottom electrode in the anodic bonding apparatus. The micro-controller uses its built in analog to digital converter (ADC) to read both the current bottom electrode temperature as well as the ambient temperature. The ADC is only read when the main TRIAC is turned off. The current draw from the zero-cross optically isolated TRIAC and ac noise cause error on readings taken by the ADC when the TRIAC is enabled. Since the time response is quite slow it does not seem to affect control. The op-amp gain is calculated by the simple non-inverting gain equation:

$$G = 1 + R2/R1 \tag{1}$$

where $R1$ and $R2$ is measured with an ohm-meter to be $220\,\Omega$ and $56.6\,\mathrm{k}\Omega$ respectively yielding a gain $G = 258.273$. When the ADC channel of the thermocouple is read, it is converted into a voltage by multiplying it by the ADC reference voltage (measured to be $4.98\,\mathrm{V}$), divided by the maximum scale of the 10-bit ADC, 1024, and then divided by the op-amp gain to recover the original thermocouple voltage, V_t:

$$V_t = \frac{ADC_{read} \times 4.98}{1024 \times G} \tag{2}$$

The temperature of the thermocouple is then determined by interpolating within a stored table of type-k thermocouple values [56]. The portion of the table used in the controller software includes values $5\,^{\circ}\mathrm{C}$ apart from $0 - 600\,^{\circ}\mathrm{C}$. Finally, the temperature reading of the ambient temperature sensor is added to the thermocouple reading giving the current thermocouple temperature. The ambient temperature sensor (TC1047A) is accurate to within $\pm 0.5^{o}C$ [57] and the minimum resolution of the ADC on the thermocouple channel is approximately $\pm 0.5\,^{\circ}\mathrm{C}$. This gives an estimated accuracy of approximately $\pm 1\,^{\circ}\mathrm{C}$. The ambient temperature sensor was calibrated using the specifications provided in the TB051 application note [58] from Microchop Inc. and verified by comparing measured ambient temperature readings to a calibrated Fluke 52 thermometer.

A 90mm fan is setup to blow across the ambient temperature sensor and then the heatsink of the main TRIAC on the controller board. This helps to prevent board heating from affecting the temperature readings as well as to dissipate any heat generated by the TRIAC during operation. The controller board is powered from a dc power supply from $6 - 9\,\mathrm{V}$ which is fed through a low dropout regulator to provide the main $5\,\mathrm{V}$ supply for the controller. The heater is connected to the controller with the ac hot line wired through a fuse to the first terminal of the main TRIAC. The fuse used is an $8\,\mathrm{A}$ fast blowing fuse, sized to match the $600\,\mathrm{W}$ power rating of the heating element at $120\,\mathrm{V_{ac}}$ which is approximately $7\,\mathrm{A_{RMS}}$. The maximum rating of the main TRIAC is $12\,\mathrm{A}$ which can not be exceeded by any heating element used. The element itself is connected on the neutral line and through the centre terminal of the TRIAC. It is important that the neutral be connected to the centre terminal of

the TRIAC which is electrically connected to the heatsink so that when the heater is not activated by the TRIAC but still plugged in to ac, it minimizes the risk of shock or short circuit through the heatsink.

There is also a single tactile switch mounted on the controller. Momentarily pushing the switch will toggle the controller from an idle state to a state in which the heater is activated. When idle, all control values are set to zero. The LED indicates the current status of the controller; when lit it indicates that the heater is active. Pressing and holding the tactile switch will enter a configuration menu which allows the PID gain values to be adjusted. When this menu is entered, it will first display the current proportional gain value. Using the set-point potentiometer the value can be adjusted; setting a value of zero will cause it to keep the current value unchanged. When the new value has been adjusted, momentarily pressing the tactile switch allows the integral and differential gain values to be adjusted similarly. After updating the differential gain value, the controller will return to an idle state.

All tasks of the micro-controller are timed using a 16-bit timer that is set to overflow and cause an interrupt every 5 ms. The C source code of the controller is provided in Appendix 7.2. The main tasks of the micro-controller are listed here:

1. Every 5ms update the status of the TRIAC based on the control signal. The control signal ranges from zero to 230 and represents the number of 5ms ticks that the TRIAC will remain activated out of 256. The maximum duty cycle is thus 230/256, or 89.84%, which has the TRIAC activated for 1.15 s and off for 130 ms.

2. Every100 ms the current temperature reading and control values are updated if possible.

3. The LCD animation is updated every 150 ms.

4. Every two seconds the current measurements and control values are reported to USB. The values are listed in a comma separated list where each line represents a two second interval. The values provided are current temperature, set-point temperature, ambient temperature, proportional error, integral error, differential error, proportional gain constant, integral gain constant, differential gain constant, and control value respectively. When connected by USB, the controller appears as a standard character device class (CDC) USB device with readily available drivers through the USB framework provide by Microchip Inc. [59]. Although possible to power the controller through USB, it is not recommended since the reference voltage provided is not reliable across different USB hosts and thus temperature readings when powered only from USB may be inaccurate.

5. The LCD display is refreshed every 5 seconds, however, the display may be updated sooner if the set-point is changed by more than 0.5 °C.

6. The LCD line-graph is updated every 30 seconds.

7. The ambient temperature reading is updated every 30 seconds.

Proportional-integral-differential (PID) control is employed to control the heating element. PID control derives the control signal of the heating element using three gain constants multiplied by the proportional, integral, and differential errors respectively. The book "Modern Control Engineering" by K. Ogata [60] provides a good overview of PID control theory. The proportional term is calculated by multiplying the difference between the set-point and the current temperature by the proportional gain value. The integral component is a bit more complex and operates by maintaining a counter that adds the proportional error multiplied by 100mS every 100mS. The counter is then multiplied by the integral gain constant to make up the integral component of the control signal. Finally, the differential term is calculated by taking an average of 5 thermocouple readings, waiting 10 seconds, averaging another 5 thermocouple readings, and then taking the difference between the two averages multiplied by the differential gain constant. This averaged differential was used since the heater is quite slow and taking an instantaneous difference produces little effect on control. The proportional, integral, and differential components are then summed to make the total control signal. The control signal is limited to a maximum of 230 and a minimum of 0 and corresponds to the number of 5ms ticks out of 256 that the element is activated. Non-linearities are introduced because the control signal is limited to a specified range, however, it has been shown to remain stable in this application.

PID values were manually tuned experimentally by varying values and running a test recording temperature and control values when set to 250 °C. 250 °C was chosen as a typical set-point value that is in the mid-range of the heaters capabilities. It is possible to control the temperature from approximately 50 °C to 400 °C using the current controller and the 600 W heating element. Figure 1 shows two thermal response curves of the heater set to 250 °C with two sets PID gain constants.

The red curve of Figure 1 has a higher integral gain which causes it to overshoot and then undershoot the desired value. The blue curve, however, has less overshoot and is more stable with no oscillations after the initial overshoot. In this system, the integral control has the most effect on the response. The differential gain has very little effect on the response until the temperature has reached its target temperature where differential gain works to maintain the temperature when small variations occur. For this system, it is desirable to have as little overshoot as possible because it is much faster to heat the ground plate than it is to cool, thus the PID gain constants of the blue curve in Figure 1 were chosen. With further experimentation or analysis, it is likely that better PID gain constants could be found to reduce the settling time.

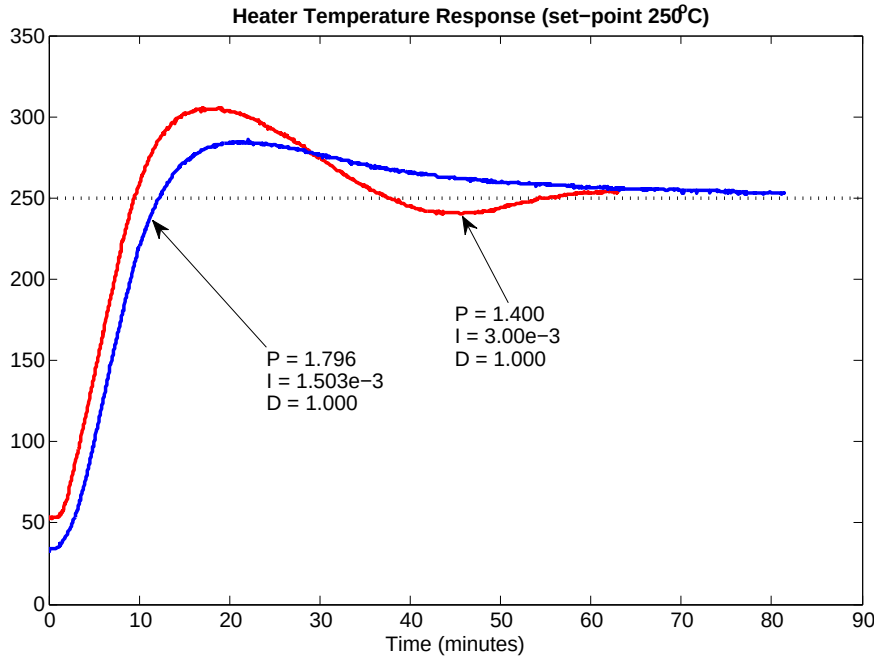
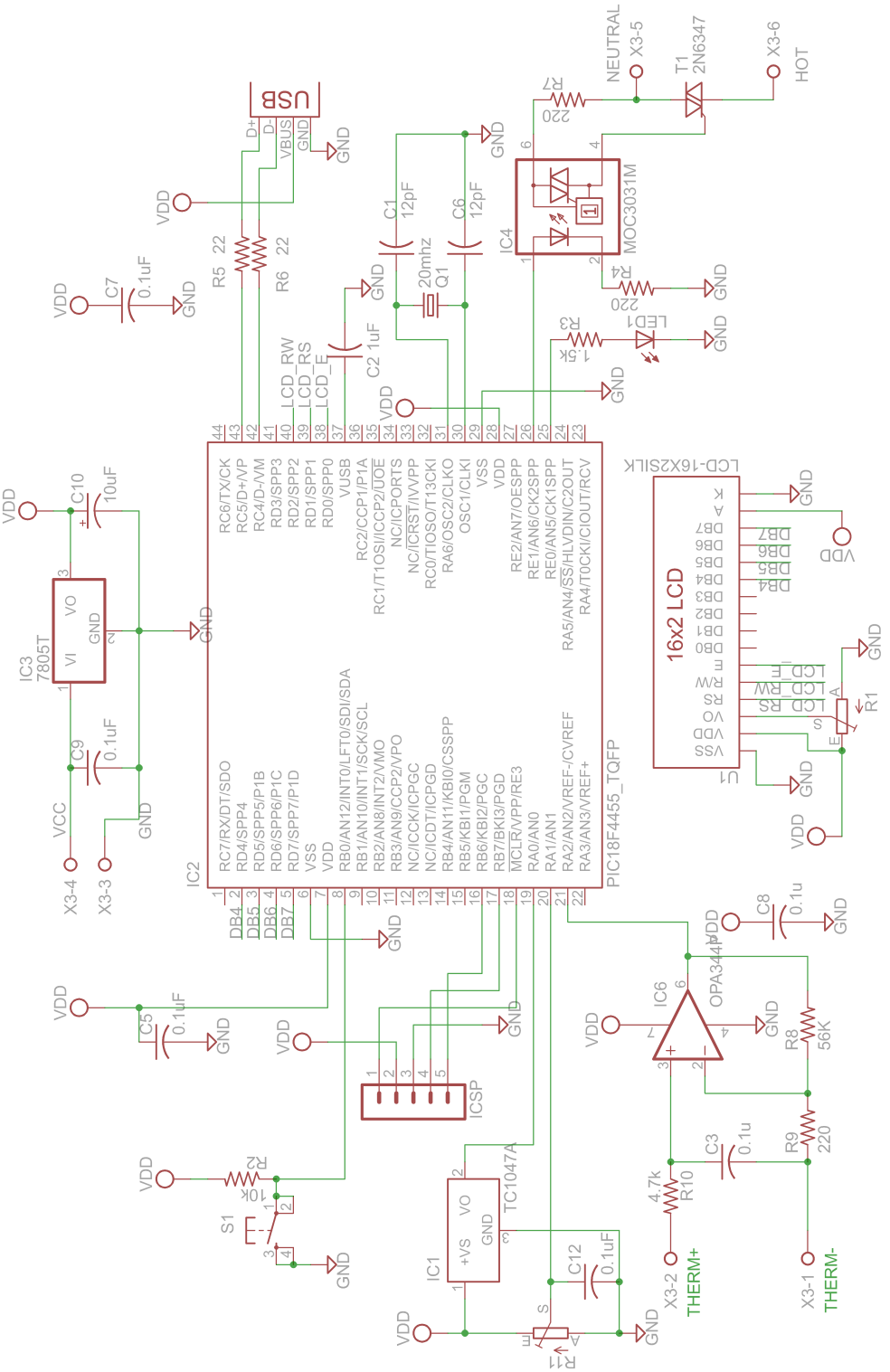


Figure 1: *Heater temperature response when set to* $250\,^{\circ}\mathrm{C}$ *in a rough vacuum. Each line is labelled with the corresponding PID constants used in the controller.*

Appendix E: Heater Controller Schematic



Appendix F: Heater Controller C Code

```
#include  <18F4455.h>
#device ADC=10
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NOMCLR,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
#use delay(clock=48000000)

#define __USB_PIC_PERIF__ 1

#rom int 0xf00000={0,0,0x07,0x08} // KP
#rom int 0xf00010={0,0,0x03,0xE8} // KD
#rom int 0xf00020={0,0,0x0F,0xA0} // KI

#include <usb_cdc.h>
#include <lcd.c>
#include <stdlib.h>

int16 CONST K_TABLE[125] = { 0, 198, 397, 597, 798, 1000, 1203, 1407, 1612, 1817, 2023, 2230, 243
                             2644, 2851, 3059, 3267, 3474, 3682, 3889, 4096, 4303, 4509, 4715,
                             4920, 5124, 5328, 5532, 5735, 5937, 6138, 6339, 6540, 6741, 6941,
                             7140, 7340, 7540, 7739, 7899, 8138, 8338, 8539, 8739, 8940, 9141,
                             9343, 9545, 9747, 9950, 10153, 10357, 10561, 10766, 10971, 11176,
                             11382, 11588, 11795, 12001, 12209, 12416, 12624, 12831, 13040,
                             13248, 13457, 13665, 13874, 14084, 14293, 14503, 14713, 14923,
                             15133, 15343, 15554, 15764, 15975, 16186, 16397, 16608, 16820,
                             17031, 17243, 17455, 17667, 17879, 18091, 18303, 18516, 18728,
                             18941, 19154, 19366, 19579, 19792, 20005, 20218, 20431, 20644,
                             20857, 22350, 22563, 22776, 22990, 23203, 23416, 23629, 23842,
                             24055, 24267, 24480, 24693, 24905  } ;


#define LED PIN_E0
#define TRIAC PIN_E1

#define SET_TEMP_ADC 0.341796875 // 400/1024
#define AMB_TEMP_ADC  0.486328125 // adjusted for 4.98V VREF.
#define AMB_TEMP_OFFSET 50
//THERMO_GAIN =  4.98 (measured LDO VDD) /0 x3FF / 258.2727273 * 1000000 (opamp gain)
// opamp gain calculated by 1 + R2/R1 where R2 = 56.6K (measured) and R1 = 220 ohm (measured)
#define THERMO_GAIN   18.83002243729

#define EEPROM_KC 0x00
#define EEPROM_KD 0x10
#define EEPROM_KI 0x20

long Kc = 0;
long Ki = 0;
long Kd = 0;

unsigned int graph[20] = { 0,0,0,0,0,
                           0,0,0,0,0,
                           0,0,0,0,0,
```

```
                            0,0,0,0,0,};

float ctrl = 0, lastTemp=0, error=0, error_d=0, error_i =0, d_tmp = 0;
unsigned int d_count = 0, d_count2 = 0;
unsigned int cycle_count = 0;

boolean on = 0;
boolean output = 0;
boolean updateLCD = TRUE;
boolean updateUSB = FALSE;
int k=0;
float setpoint=0;
float curtemp=0.0;
float ambtemp=0;
BYTE i, j, address, value;

int anim_hold = 40;
int animation_step = 0;
char msg[40];
int msg_l;

unsigned long ticks=0;

void lcd_load_graph_chars(void)
{
    int8 i,j,k,offset;
    // Set address counter pointing to CGRAM address 0.
    lcd_send_byte(0, 0x40);

    for(i = 0; i < 8; i++)  // chars
    {
        if (i > 3)
        {
            offset = (i-4)*5;
            for (k=0; k < 8; k++) // lines
            {
                j=8-k;
                lcd_send_byte(1, ((graph[offset]-8==j)<<4) + ((graph[offset+1]-8==j)<<3) +
                ((graph[offset+2]-8==j)<<2) + ((graph[offset+3]-8==j)<<1) + (graph[offset+4]-8==
            }
        } else
        {
            offset = i*5;
            for (k=0; k < 8; k++) // lines
            {
                j=8-k;
                lcd_send_byte(1, ((graph[offset]==j)<<4) + ((graph[offset+1]==j)<<3) +
                ((graph[offset+2]==j)<<2) + ((graph[offset+3]==j)<<1) + (graph[offset+4]==j));
            }
        }
    }

    // Set address counter pointing back to the DDRAM.
    lcd_send_byte(0, 0x80);
```

```
105 }
106
107 #SEPARATE
108 float getTemp(long voltage) {
109     float ret=0;
110     int i;
111
112     for (i=0; i < 120; i++) {
113         if (voltage <= K_TABLE[i])
114             break;
115     }
116
117     if (i==61) {
118         ret = 600.0;
119     } else if (i == 0) {
120         ret = 0.0;
121     } else
122     {
123         ret = 5.0*(i-1) + 5.0/(K_TABLE[i] - K_TABLE[i-1])*(voltage - K_TABLE[i-1]);
124     }
125
126     return ret + ambtemp;
127 }
128
129 #SEPARATE
130 void write_eeprom_settings(int offset,  long val)
131 {
132     write_eeprom(offset+0x02, val>>8);
133     write_eeprom(offset+0x03, val);
134 }
135
136 #SEPARATE
137 long read_eeprom_settings(int offset)
138 {
139     long val = 0;
140     val += read_eeprom(offset+0x02) * 256;
141     val += read_eeprom(offset+0x03);
142     return val;
143 }
144
145 // timer overflows every 5 mS.  initialize with 5536 and overflows every 5ms.
146 #INT_TIMER0
147 void tick()
148 {
149     float temp = 0;
150     float tmp,tmp2;
151     int i,j;
152     long blah;
153     disable_interrupts(INT_TIMER0);
154     ticks++;
155
156     if (usb_cdc_connected())
157         usb_task();
158
```

```
159     cycle_count++;
160     if (cycle_count < ctrl & output == TRUE) {
161         on = true;
162         output_bit(TRIAC,1);
163     } else
164     {
165         on = false;
166         output_bit(TRIAC,0);
167         delay_ms(1);
168     }
169 
170     // update ambient temp & graph every 30 seconds. only updates when TRIAC is turned off.
171     if (on == FALSE && ticks % 6000 == 0)
172     {
173         output_bit(TRIAC,0);
174         set_adc_channel(0);
175         delay_ms(1);
176         ambtemp = read_adc()*AMB_TEMP_ADC;
177         ambtemp -= AMB_TEMP_OFFSET;
178 
179         graph[k++]= (int)(curtemp/25.0);
180         lcd_load_graph_chars();
181         if (k==20) k = 0;
182 
183         lcd_gotoxy(1,1);
184         lcd_putc("\004\005\006\007\n\t\001\002\003");
185     }
186 
187     if (ticks % 30 == 0) // update animation every 150ms
188     {
189         lcd_gotoxy(6,2);
190 
191         j=0;
192         if (anim_hold == 0 || anim_hold == 40)
193             for (i=0;i<40 && j<12;i++)
194                 if (animation_step <= i)
195                 {
196                     lcd_putc(msg[i]);
197                     j++;
198                 }
199 
200         if (anim_hold == 0)
201         {
202             animation_step++;
203 
204             if (animation_step == (23 +(setpoint>100?1:0)+(ambtemp>100?1:0)) )
205                 animation_step = 0;
206 
207             if (animation_step == 0 || animation_step == (11 +(setpoint>100?1:0)))
208                 anim_hold = 40;
209         } else
210             anim_hold--;
211     }
212 
```

```
213     if (ticks % 400 == 0 && usb_enumerated()) // update usb every 2 second
214     {
215         updateUSB=TRUE;
216     }
217 
218     if (ticks % 1000 == 0) // update display every five second (if it hasnt already)
219     {
220         updateLCD=true;
221     }
222 
223     if (ticks % 20 == 0  ) // update control every 100 ms
224     {
225         if (!on) { // dont read ADC when the triac is on. Voltage dip causes bad readings.
226             set_adc_channel(2);
227             delay_ms(1);
228             blah=read_adc();
229             tmp2 = blah*THERMO_GAIN;
230             tmp2 = getTemp((long)tmp2);
231             tmp = curtemp - tmp2;
232             if (tmp < -1.0 || tmp > 1.0)
233                 updateLCD = TRUE;
234             curtemp=tmp2;
235         }
236 
237         if (output)
238         {
239             error = (setPoint - curtemp);
240             error_i += (error*0.1);
241             d_count++;
242             d_tmp+=curtemp;
243             if (d_count == 10)
244             {
245                 d_tmp = d_tmp / d_count;
246                 d_count2++;
247                 if (d_count2 == 10)
248                 {
249                     error_d = d_tmp - lastTemp;
250                     lastTemp=d_tmp;
251                     d_count2=0;
252                 }
253                 d_count = 0;
254             }
255             ctrl =  (Kc/1000.0*error + Ki/1000000.0*error_i - Kd/1000.0*error_d);
256             if (ctrl > 230.0) { ctrl = 230.0; }  // maximum output =~ 90% duty
257             else if (ctrl < 0.0) { ctrl = 0.0; }
258         }
259 
260         set_adc_channel(1);
261         delay_ms(1);
262         temp = read_adc()*SET_TEMP_ADC;
263         tmp = (temp-setpoint);
264 
265         if (tmp < -1.5 || tmp > 1.5 )
266         {
```

```
267             animation_step = 0;
268             anim_hold = 40;
269         }
270         if ( tmp < -.5 || tmp > .5 )
271         {
272             setpoint=temp;
273             updateLCD = TRUE;
274         }
275 
276         if (updateLCD)
277         {
278             printf(lcd_putc,"\f\004\005\006\007 cur:%3.1f\337C\n\t\001\002\003 ",curtemp);
279             sprintf(msg,"set:%3.1f\337C amb:%3.1f\337C set:%3.1f\337C",setpoint,ambtemp,setpoint)
280             j=0;
281             for (i=0;i<40 && j<12;i++)
282                 if (animation_step <= i)
283                 {
284                     lcd_putc(msg[i]);
285                     j++;
286                 }
287             updateLCD= FALSE;
288         }
289     }
290 
291     set_timer0(5536);
292     enable_interrupts(INT_TIMER0);
293 }
294 
295 #INT_EXT
296 void button()
297 {
298     unsigned long i;
299     long  param_last=0, param;
300     long tmp;
301 
302     disable_interrupts(INT_EXT);
303 
304     for (i=0;i<1000 && (input_b()&0x01) == 0x00; i++)
305         delay_ms(1);
306 
307     if (i >= 1000)
308     {
309         set_adc_channel(1);
310         param_last = -100;
311         param = (long)(read_adc()*9.765625);
312         tmp = (long)((param-param_last)*100);
313         if ( tmp < -20 | tmp > 20 )
314         {
315             printf(lcd_putc,"\fcur P=%lu\n",  Kc);
316             printf(lcd_putc,"new P=%lu", param);
317             param_last=param;
318         }
319 
320         while ((input_b()&0x01) == 0x00)
```

```
321                 delay_ms(1);
322         do
323         {
324             usb_task();
325             param = (long)(read_adc()*9.765625);
326             delay_ms(10);
327             tmp = (long)((param-param_last)*100);
328             if ( tmp < -20 | tmp > 20 )
329             {
330                 printf(lcd_putc,"\fcur P=%lu\n", Kc);
331                 printf(lcd_putc,"new P=%lu",  param);
332                 param_last=param;
333             }
334         } while ( (input_b()&0x01) == 0x01 );
335         if (param > 0)
336         {
337             Kc=param;
338             write_eeprom_settings(EEPROM_KC,Kc);
339         }
340 
341         while ((input_b()&0x01) == 0x00)
342             delay_ms(1);
343         param_last=-100;
344         do
345         {
346             usb_task();
347             param = (long)(read_adc()*9.765625);
348             delay_ms(10);
349             tmp = (long)((param-param_last)*100);
350             if ( tmp < -15 | tmp > 15 )
351             {
352                 printf(lcd_putc,"\fcur I=%lu\n", Ki);
353                 printf(lcd_putc,"new I=%lu",  param);
354                 param_last=param;
355             }
356         } while ( (input_b()&0x01) == 0x01 ) ;
357         if (param > 0)
358         {
359             Ki=param;
360             write_eeprom_settings(EEPROM_KI,Ki);
361         }
362 
363         while ((input_b()&0x01) == 0x00)
364                 delay_ms(1);
365         param_last=-100;
366         do
367         {
368             param = (long)(read_adc()*9.765625);
369             delay_ms(10);
370             tmp = (long)((param-param_last)*100);
371             if ( tmp < -15 | tmp > 15 )
372             {
373                 printf(lcd_putc,"\fcur D=%lu\n", Kd);
374                 printf(lcd_putc,"new D=%lu",  param);
```

```
375                 param_last=param;
376             }
377
378
379         } while ( (input_b()&0x01) == 0x01 ) ;
380         if (param > 0) {
381             Kd=param;
382             write_eeprom_settings(EEPROM_KD,Kd);
383         }
384
385         while ((input_b()&0x01) == 0x00)
386                 delay_ms(1);
387     } else
388     {
389         output = ~output;
390         output_bit(LED,output);
391         if (!output)
392             error=0; error_d=0; error_i=0; ctrl=0.0;
393     }
394
395     updateLCD = TRUE;
396     enable_interrupts(INT_EXT);
397
398 }
399
400 void main()
401 {
402     usb_cdc_init();
403     usb_init();
404     lcd_init();
405     lcd_load_graph_chars();
406
407     set_tris_b(0b00000001);
408     setup_adc(ADC_CLOCK_INTERNAL);
409     setup_adc_ports(AN0_TO_AN2 );
410
411     set_adc_channel(0);
412     delay_ms(1);
413     ambtemp = read_adc()*AMB_TEMP_ADC;
414     ambtemp -= AMB_TEMP_OFFSET;
415
416     setup_timer_0(RTCC_INTERNAL | RTCC_DIV_1);
417     set_timer0(5563);
418
419     Kc = read_eeprom_settings(EEPROM_KC);
420     Ki = read_eeprom_settings(EEPROM_KI);
421     Kd = read_eeprom_settings(EEPROM_KD);
422
423     ext_int_edge( H_TO_L );
424     enable_interrupts(INT_TIMER0);
425     enable_interrupts(INT_EXT);
426     enable_interrupts(GLOBAL);
427
428     output_bit(TRIAC,0);
```

```
429     output= FALSE;
430     output_bit(LED,output);
431 
432      while (TRUE)
433     {
434         if (updateUSB && usb_enumerated())
435         {
436             printf(usb_cdc_putc, "%3.2f,",curtemp); usb_task();delay_ms(5);
437             printf(usb_cdc_putc, "%3.2f,",setpoint);usb_task();delay_ms(5);
438             printf(usb_cdc_putc, "%3.2f,",ambtemp);usb_task();delay_ms(5);
439             printf(usb_cdc_putc, "%3.1f,", error);usb_task();delay_ms(5);
440             printf(usb_cdc_putc, "%3.1f,", error_i);usb_task();delay_ms(5);
441             printf(usb_cdc_putc, "%3.1f,", error_d);usb_task();delay_ms(5);
442             printf(usb_cdc_putc, "%lu,", kc);usb_task();delay_ms(5);
443             printf(usb_cdc_putc, "%lu,", ki);usb_task();delay_ms(5);
444             printf(usb_cdc_putc, "%lu,", kd);usb_task();delay_ms(5);
445             printf(usb_cdc_putc, "%3.1f\r\n", ctrl);usb_task();delay_ms(5);
446             usb_task();delay_ms(5);
447             usb_task();delay_ms(5);
448             updateUSB=FALSE;
449         }
450 
451     }
452 }
```