Fall 1-31-2011

Modeling of magnetic field driven simultaneous assembly

Rene David Rivero
New Jersey Institute of Technology

ABSTRACT

MODELING OF MAGNETIC FIELD DRIVEN SIMULTANEOUS ASSEMBLY

by
Rene David Rivero

The Magnetic Field Driven Simultaneous Assembly (MFDSA) is a method that offers a non-statistical and deterministic solution to the problem of assembly via batch processing; a hybrid of serial and parallel processing. The technique requires the use of electromagnets as well as soft and hard magnetic materials that are applied to devices and recesses respectively. The MFDSA approach offers the ability to check and correct errors in real-time and is capable of scalable, versatile, and high-yield integration.

Devices, coated with a layer of soft magnetic material, are moved from initial to final positions along predetermined pathways through the action of an array of electromagnets. Various devices, of arbitrary geometries, with different physical and functional properties, are manipulated simultaneously toward specific desired locations and then dropped onto a template under the influence of gravity by weakening the local applied field. Locations on the template correspond to sites on a substrate that contain recesses. When a number of devices have been dropped onto the template, a substrate is pressed onto it and the soft magnetic layers on the devices adhere to the hard magnetic strips in the recesses, completing integration in a single step.

The objectives of this dissertation are the following: to present the MFDSA method; comparing and contrasting it with other extant techniques employed by the semiconductor industry; to discuss key aspects of this solution with respect to the problem of assembly, and to model the calculations involved with determining both device pathways and field interactions that are required to implement the approach. The Fourier Series technique will be used to describe the force of attraction between the device's soft magnetic layer and the recess's hard magnetic strips. Methodology from finite element analysis will be employed to calculate the force exerted on a device by an array of electromagnets. The Swarm Algorithm, which was developed in this work to calculate device pathways, will be presented as a stable, well-defined solution.

Other concepts, such as the magnetic retention factor and the collision cross-section area, will be presented and developed. The solution to the problem of assembly, via the Swarm Algorithm, will be compared and contrasted with other analogous problems found in the literature. The results of these models, including software implementation, will be presented.

MODELING OF MAGNETIC FIELD DRIVEN SIMULTANEOUS ASSEMBLY

by
Rene David Rivero

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
and Rutgers, The State University of New Jersey – Newark
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Applied Physics

Federated Department of Physics

January 2011

.

APPROVAL PAGE

MODELING OF MAGNETIC FIELD DRIVEN SIMULTANEOUS ASSEMBLY

Rene David Rivero

Dr. Nuggehalli M. Ravindra, Dissertation Advisor — Date
Professor and Chair of Physics, NJIT

Dr. Michael R. Booty, Committee Member — Date
Professor of Mathematics, NJIT

Dr. Anthony T. Fiory, Committee Member — Date
Research Professor of Physics, NJIT

Dr. Martin Schaden, Committee Member — Date
Associate Professor of Physics, Rutgers, The State University of New Jersey – Newark

---

Dr. Vitaly Shneidman, Committee Member — Date
Senior University Lecturer of Physics, NJIT

BIOGRAPHICAL SKETCH

Author: Rene David Rivero

Degree: Doctor of Philosophy

Date: January 2011

Undergraduate and Graduate Education:

- Doctor of Philosophy in Applied Physics,
  New Jersey Institute of Technology, Newark, NJ, 2011

- Master of Science in Applied Physics,
  New Jersey Institute of Technology, Newark, NJ, 2007

- Bachelor of Science in Applied Physics,
  New Jersey Institute of Technology, Newark, NJ, 2001

Major: Applied Physics

Presentations and Publications:

R. D. Rivero, S. Shet, A. T. Fiory, M. R. Booty and N. M. Ravindra, "Magnetic Field Simultaneous Assembly: An Overview", *Proceedings of the 2010 MS&T Conference and Exhibition: Dielectric Ceramic Materials and Electronic Devices* October 2010.

R. D. Rivero, S. Shet, A. T. Fiory, M. R. Booty, M. P. Lepselter and N. M. Ravindra, "Magnetic Assembly of High Performance Solar-Cell Panels", *Proceedings of the 2010 Materials Research Society Workshop* October 2010.

R. D. Rivero, I. Padron, M. R. Booty, A. T. Fiory and N. M. Ravindra, "Indirect Template Method of Magnetic Field Assisted Assembly", *Advanced Materials Research* **89-91**, pp. 431-436 (2010).

R. D. Rivero, M. R. Booty, A. T. Fiory and N. M. Ravindra "Device Integration Using Magnetically Driven Simultaneous Assembly", *Proceedings of the 2010 TMS Annual Meeting and Exhibition: Coatings for Structural, Biological, and Electronic Applications* February 2010.

R. D. Rivero, M. R. Booty, A. T. Fiory and N.M. Ravindra, "Intermediate Template Magnetic Field Assisted Assembly", CT Volume 221, MS&T (2009).

N. M. Ravindra, R. D. Rivero, A. T. Fiory and M. R. Booty, "An Intermediate Template Method for Magnetic Field Assisted Assembly" US Provisional Patent Application 61/152,502 (2009).

R. D. Rivero, M. R. Booty, A. T. Fiory and M. M. Ravindra, "Indirect Template Model for Magnetic Field Assisted Assembly", *Proceedings of the 2009 TMS Annual Meeting and Exhibition: Recent Advances in Thin Films* February 2009.

R. D. Rivero, G. Devrani, M. R. Booty, A. T. Fiory and N. M. Ravindra, "A Two Dimensional Model for Magnetic Field Assisted Assembly", *Proceedings of the 2008 TMS Annual Meeting and Exhibition: Mechanics and Kinetics of Interfaces in Multi-Component Materials Systems* March 2008.

R. D. Rivero, S. Shet, M. R. Booty, A. T. Fiory and N. M. Ravindra, "Modeling of Magnetic Field Assisted Assembly of Semiconductor Devices", *J. Electron. Mater.* **37**, pp. 374-378 (2008).

C. Li, B. L. Sopori, R. D. Rivero, P. Rupnowski, A. T. Fiory and N. M. Ravindra, "Role of the Damage Layer in Bulk and Surface Passivation of Silicon Solar Cells by SiN:H", *Proceedings of the17 Annual Workshop on Crystalline Silicon Solar Cells and Modules: Materials and Processes*, editor: B. L. Sopori, **NREL/BK-520-41973**, pp. 303-308 (2007).

S. Shet, R. D. Rivero, M. R. Booty, A. T. Fiory, M. P. Lepselter and N. M. Ravindra, "Microassembly Techniques: A Review", *Materials Science & Technology* **1** 451-474 (2006).

To my parents, Rene and Alina Rivero, for taking me out of Cuba.
To my sister, Michelle Nott.
To my advisor, Dr. N. M. Ravindra, for his wisdom and belief in me.
And to those who came before, those who come after.

ACKNOWLEDGMENT

My work in the fields of magnetism and assembly started c. 2005 when I joined the group led by Dr. N. M. Ravindra. Special thanks are given to the other members of the committee: Dr. Michael R. Booty, Dr. Anthony T. Fiory, Dr. Martin Schaden, and Dr. Vitaly Shneidman.

I want to thank my professors at NJIT, especially Dr. John C. Hensel, Dr. Moses Fayngold, and the late Dr. V. A. Goldberg; I wish to be as great an educator to others as they were to me.

I extend my deepest and sincerest thanks to Dr. Sudhakar Shet, the originator of Magnetic Field Assisted Assembly, as well as Dr. Ivan Padron and Vijay Kasisomayajula for their insights.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

1.1 Abstract

The Magnetic Field Driven Simultaneous Assembly (MFDSA) is proposed as a method to integrate devices utilizing a combination of electromagnetic and gravitational fields. It is a non-statistical, fully deterministic and controllable parallel assembly technique with error-correction. It is a versatile system capable of scaling and high-yield integration.

Devices of arbitrary geometries, with different physical properties and functionalities, are coated with a layer of soft magnetic material. They are moved from initial to final positions through the action of an array of electromagnets. They are advanced, simultaneously, toward specific desired locations and then populated onto the template under the influence of gravity. Specific desired locations on the template correspond to recesses on the substrate where devices are intended to be placed and that already contain strips of hard magnetic material.

An error-correction algorithm is invoked to check the placements of devices that have been populated on top of the template prior to insertion. The substrate is pressed onto the template and devices are inserted into recesses. Devices are secured within recesses by the magnetic attraction between the devices' soft magnetic layer and the recess' hard magnetic strips [1].

1.2 Background

The current state of device integration technology is able to produce solid state devices, circuits and systems from components that are made of a variety of materials. The standard toolkit for device integration consists mainly of bulk and surface silicon micromachining, laser-micromachining, and other lithographic techniques. As an example, compound semiconductor devices tend to be created monolithically from the substrate [2].

The ability to integrate components into systems is valued by industry due to its significant applications. Trends indicate that future generations of Micro Electro Mechanical Systems (MEMS), Lab on a Chip (LOC), System on a Chip (SOC), XYZ on a Chip, Systems-in-Package (SIP), sensors, and actuators will be integrated along with other components onto wafers to form powerful and complex systems [3]. Enormous interest lies in integrating components with CMOS technologies in order to increase the number of functions on-wafer and, ultimately, to reduce power requirements, costs, sizes, and weights of systems [2].

Toward that end, methods need to be developed to enable the assembly of components into systems with dissimilar materials or even materials with incompatible physical properties. However, combining materials brings with it difficulties, among which are mismatches between lattice and thermodynamic properties, such as, for example, the very large differences in thermal expansion coefficients of silicon and III-V compounds used with optics [4]. The development of heterogeneous, small and large scale, and room-temperature parallel integration techniques is critical in order to realize low-cost, high-density systems [2].

1.2.1 Serial Assembly

A standard integration strategy, in the industry, is the 'pick and place' serial assembly approach (see Figure 1.1). The method encounters immediate and insurmountable constraints with respect to speed and cost. It is slow to use in situations that involve the assembly of a large number of components with high precision tolerances. It is unable to deal with situations where devices adhere onto the mechanism of assembly. A variety of parallel assembly techniques are being investigated and introduced to combat these limitations [5].

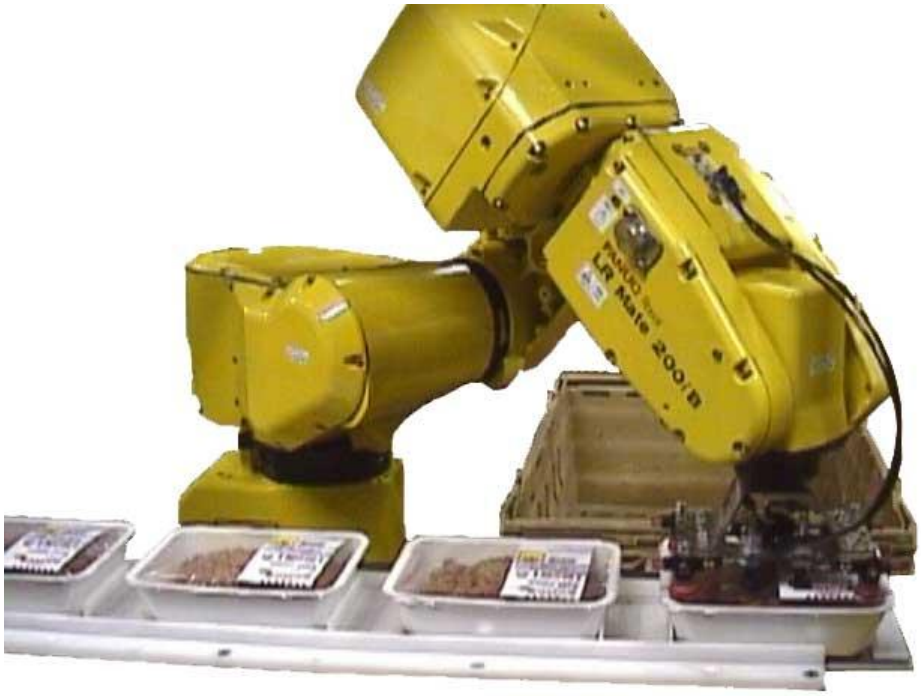

Figure 1.1 An illustration of a typical pick and place robot. A pick and place system needs to be customized prior to use. First, it is required to fit the environment where it will be used. Second, it is specialized to perform the task it is intended to do [6].

1.2.2 Parallel Assembly - General Manipulation

As the dimensions of micro-electrical, micro-optical, and micro-mechanical components and systems decrease, there is a need for technologies that simplify the effective processing of assembly. Several approaches have been proposed for such assembly. They include selective area growth, wafer bonding, and epitaxial lift-off. All of these approaches have inherent drawbacks and issues that limit their applicability.

An alternative, non-assembly method of integration is selective area growth. The technique involves the growth of GaAs or InP devices directly onto silicon. It is limited by lattice and thermal mismatches between GaAs or InP devices and silicon; further, devices grown on silicon are not comparable in functionalities or even integrity to devices grown on a lattice and thermally matched substrate. Additionally, growing GaAs or InP onto silicon is inherently difficult and costly and is limited to small areas [7-10].

The method of wafer bonding involves the transfer of a primary layer onto a secondary wafer (see Figure 1.2). The primary layer and secondary wafer are bonded together and processed into devices. The technique's major drawback is the thermal expansion coefficient mismatches when the layer and the wafer are comprised of different kinds of materials [11-13].

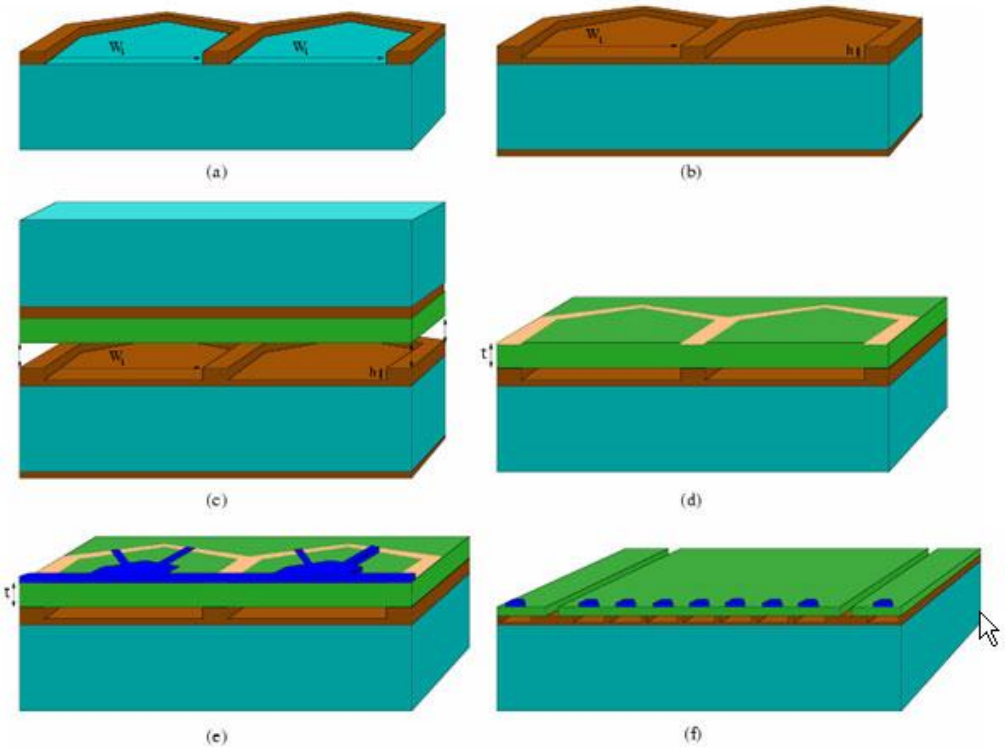

Figure 1.2 An illustration of the wafer bonding technique to fabricate Capacitive Micromachined Ultrasonic Transducers (CMUTs). (a) First thermal oxidation step and cavity definition with photolithography. (b) Second thermal oxidation to create the insulation. (c) Silicon direct bonding of the patterned prime wafer to the un-patterned Silicon on Insulator (SOI) wafer. (d) Removal of the handle and the Buried Oxide (BOX) of the SOI wafer to release the membranes. (e) Ground contact definition, electrode deposition and patterning. (f) Element definition by photolithography [14].

Another method is epitaxial lift-off (see Figure 1.3). An epitaxial layer is released out of the substrate. The epitaxial layer, supported by a membrane, is bonded onto the substrate by van der Waals forces. Devices can be processed either before or after the transfer of the layer depending on the requirements of the process. The technique suffers from various disadvantages, including the handling of potentially extremely thin layers, which is difficult, and the alignment of the devices onto circuitries, which is tedious [15-18].

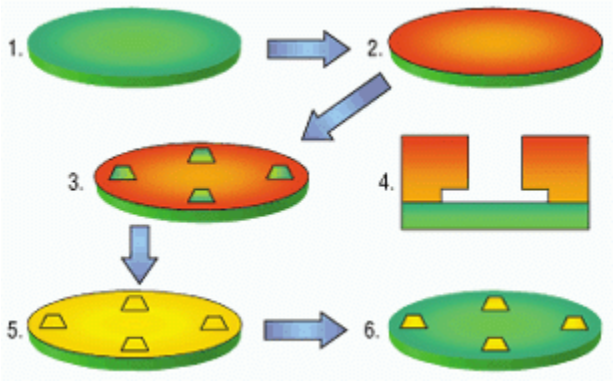

Figure 1.3 An illustration of the epitaxial lift-off technique. (1) GaAs epitaxial wafer. (2) Photoresist is spun-on. (3) Photoresist is patterned and developed. (4) Cross section showing the re-entrant sidewalls of resist windows. (5) Metal deposition. (6) Photoresist lift-off leaves metal behind [19].

1.2.3 Parallel Assembly - Mass Manipulation

An approach to parallel assembly is to integrate components without individual, device-by-device manipulation. Systems that follow this paradigm include vector potential parts manipulation, DNA and electrophoresis assisted assembly, and fluidic self-assembly.

The vector potential parts manipulation method permits the alignment of devices by using electromagnets to direct and insert units. Components must be charged in order to use electromagnets effectively. Such charges may damage devices and substrates. The DNA and electrophoresis assisted assembly uses two sets of matching DNA-like polymer films. Films are formed onto devices and deposited into recesses. As a result, devices adhere into recesses only if the films' DNA patterns match. However, the polymers are fragile and the process is costly and ineffective with respect to time [20-21].

The fluidic self-assembly method substitutes geometric patterns for DNA patterns (see Figure 1.4). Devices, etched as trapezoids, fit into recesses with matching physical geometry. Separate individual devices are aligned and inserted into the substrate passively through the aid of a fluid without individual device-by-device manipulation. The technique requires devices to be formed with a specific type of shape, which may be costly to achieve. The process of assembly itself is random; therefore, it is not guaranteed to yield a 100% complete and accurate assembly within a single iteration [4].

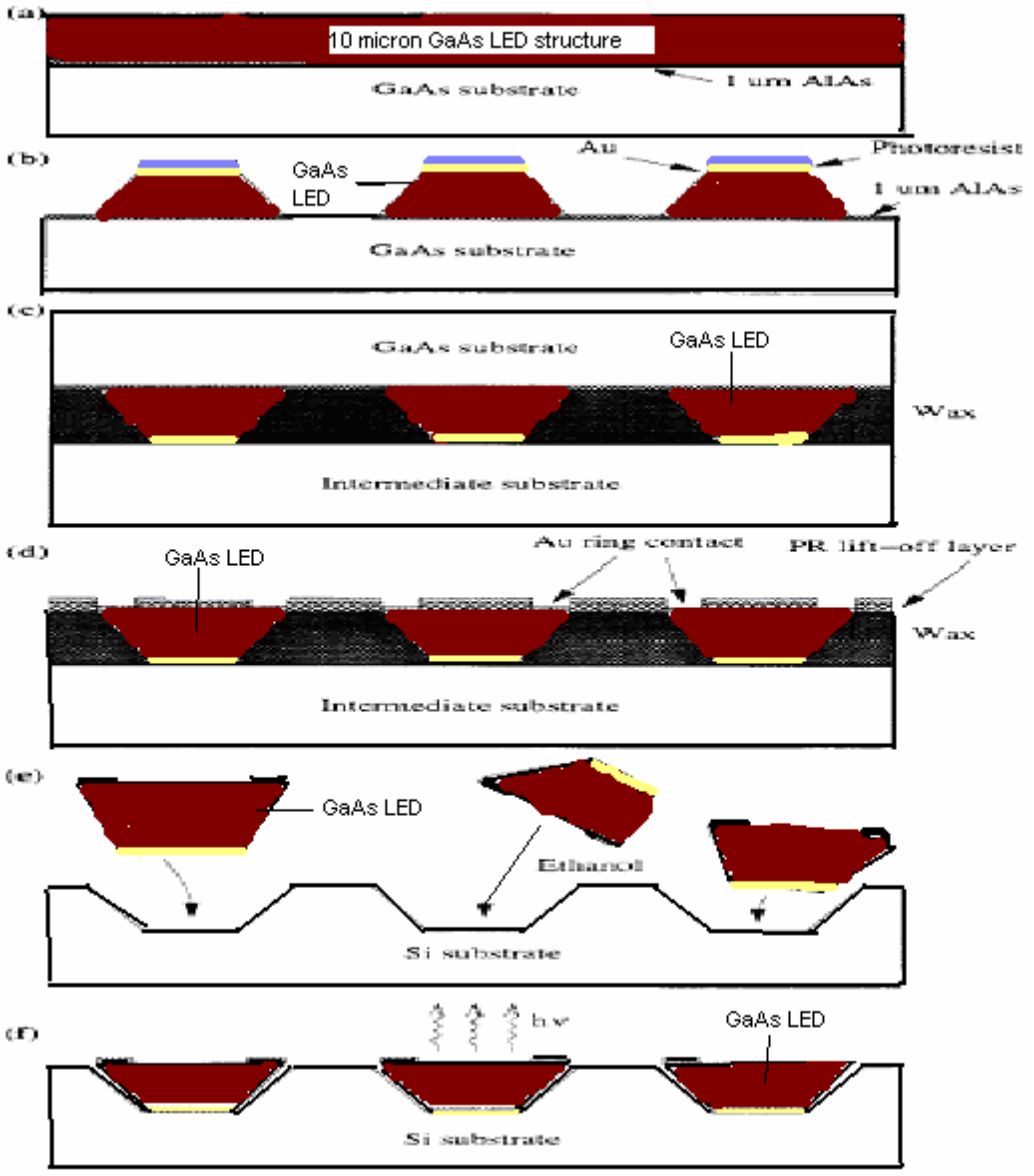

Figure 1.4 An illustration of the fluidic self-assembly technique. (a) Molecular Beam Epitaxy (MBE) growth structure with 1µm AlAs etch-stop layer. (b) Trapezoidal GaAs mesa definition. (c) Bonding to intermediate substrate with wax. (d) Top-side ring contact metallization. (e) Solution containing the GaAs blocks dispensed over patterned Si substrate. (f) Si substrate with GaAs, light-emitting diodes (LED) integrated by fluidic self-assembly [4].

1.2.4 Magnetism and Assembly

Other methods utilize magnetism to moderate the randomness of fluidic self-assembly.

The Magnetically Assisted Statistical Assembly (MASA), developed at the Massachusetts Institute of Technology (Cambridge, Massachusetts), adds layers of magnetic materials deposited onto devices and into recesses (see Figure 1.5). The fluid carries devices over recesses and the interactions between the layers cause devices to adhere into recesses [23].

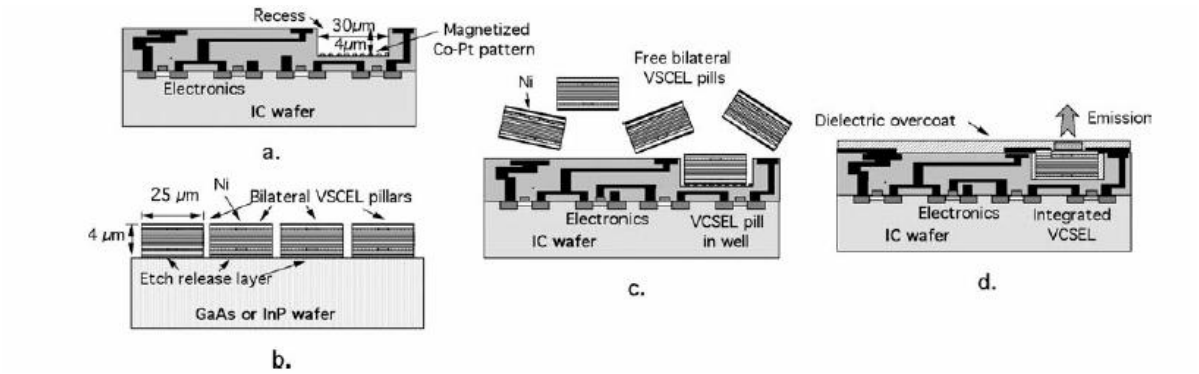

Figure 1.5 An illustration of the Magnetically Assisted Statistical Assembly (MASA) technique. (a) The processed integrated circuit (IC) wafer with prepared recesses. (b) The p-side down vertical-cavity surface-emitting laser (VCSEL) wafer with pillars etched in a close-packed array. (c) Assembly of freed nanopills into the recesses on the IC wafer. (d) After completion of device processing and integration [24].

The Magnetic Field Assisted Assembly (MFAA), proposed by the team at the New Jersey Institute of Technology (Newark, New Jersey), removes the fluid of MASA and inserts an external magnetic field to help devices to reach recesses (see Figure 1.6) [2, 25].

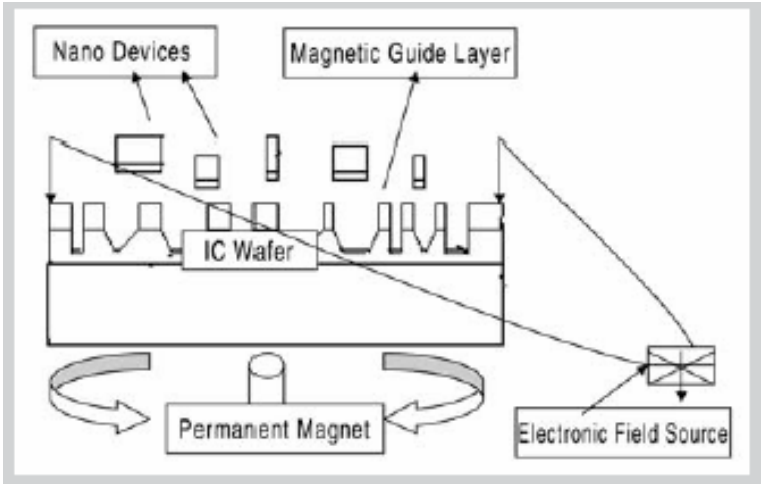

Figure 1.6 An illustration of the Magnetic Field Assisted Assembly (MFAA) technique to integrate components into an IC wafer [2].

A third method pursued at the Institute of Microelectronics (Singapore) involves an array of magnets placed below the substrate. The array drives devices (which have been coated with a layer of soft magnetic material) toward recesses. A vibration is given to the substrate to help finalize the assembly (see Figure 1.7) [26].

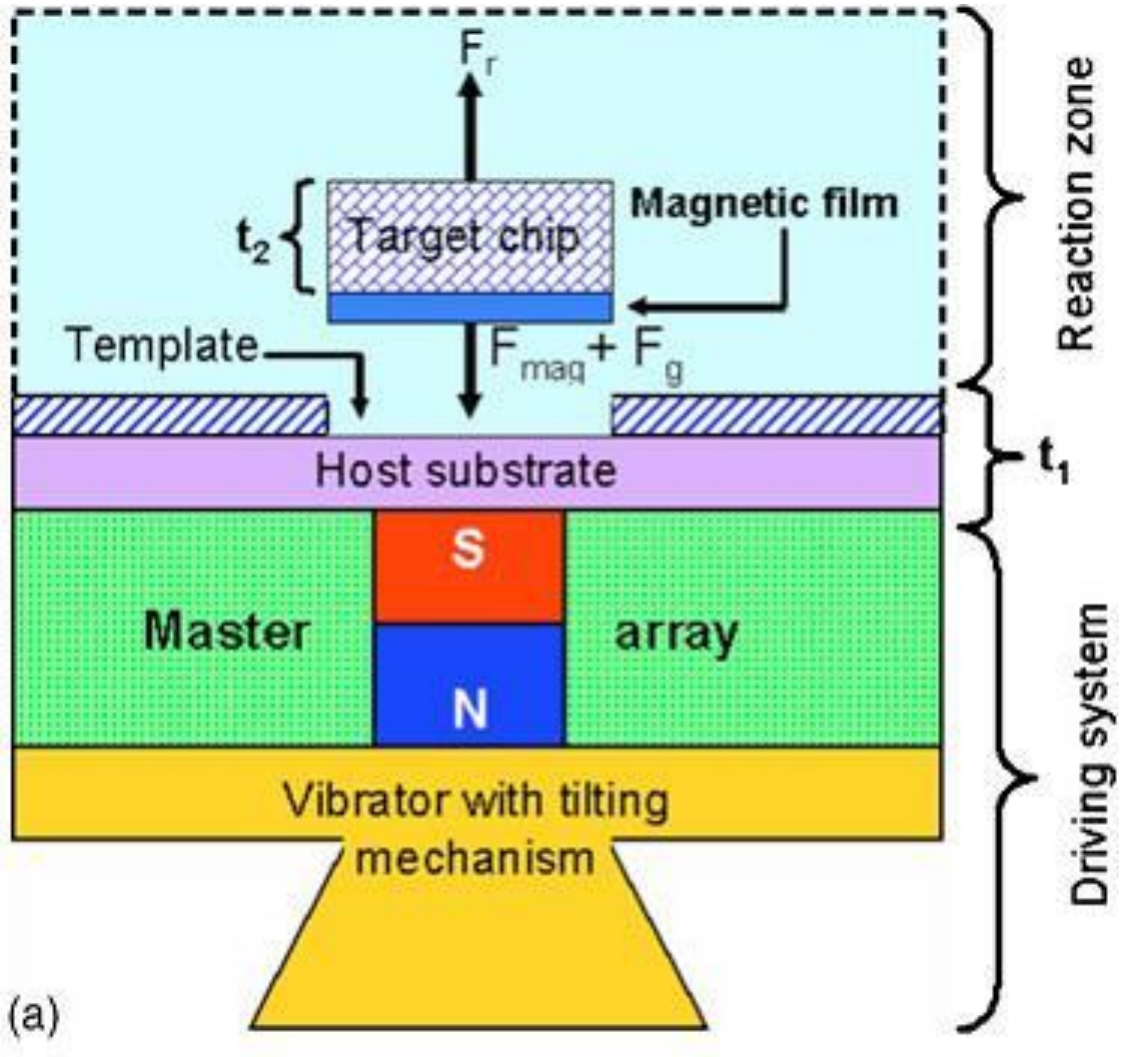

Figure 1.7 An illustration of the method investigated at Singapore consists of a combination of self-assembly via external magnetic array and vibration [26].

1.2.5 Areas of Improvements

Areas of research and development involve a fine-tune of the fluidic self-assembly methods to increase their yields. Techniques, investigated by the Alien Technology Corporation, alter standard fluidic self-assembly method by introducing asymmetric device/recess geometry. The effect of asymmetry is that devices tend to correct their orientations as they fall into recesses [27].

The work of Zheng et al. employs special auxiliary sites along the substrate to reorient devices as the fluid carries them into recesses [28]. The work of Lin et al. combines asymmetric device/recess geometry and surface tension effects to drive a self-correcting, self-assembly type of integration between devices and recesses [29].

These refinements are not free of important and limiting issues. The standard fluidic self-assembly method and its variants, including MASA and the method investigated at Singapore, are statistical and do not guarantee a 100% yield after a single iteration. Additionally, fluid and non-fluid based methods such as MFAA suffer from issues with respect to frustration, which involves devices in competition with each other to reach recesses. The insertion of devices into recesses is subject to other random effects; for example, components may enter at various angles, which may be impossible to correct without further assembly steps [30].

Table 1.1 Summary of Parallel Assembly Techniques with Gravitational Force

Authors	Demonstration	Results	References
Cohn, et al.	1000 hexagons into lattice	not reported	31
Yeh and Smith // Fonstad	GaAs LED's, GaAs/AlA's RTD's & VCSEL's	100% yield in 2.5min w. 1mm x 1.2mm x 235μm size blocks & 90% in 15min w. 150μm x 150μm x 35μm size blocks	4, 24, 25
Sangjun and Bohringer	2D & 3D dry assembly w. orientation uniqueness	100% yield in 5min (2 x redundant parts & 10% packing density) & 81% yield (1.5 x redundant parts & 40% packing density)	34
	2D & 3D dry assembly w. orientation uniqueness	95% yield w. rotational orientation error of 17deg & translational error of ±5μm	35
Baskaran, et al.	catalyst enhanced dry assembly process: parts - 800μm x 800μm x 50μm & catalysts - 2mm x 2mm x 0.5mm	20 - 50% reduction in acceleration & up to 4 x increase in number of activated parts	36

Source: Adapted from [37]. w. = with

Table 1.2 Summary of Parallel Assembly Techniques with Capillary, Type I Forces

Authors	Demonstration	Results	References
Tien, et al.	3D mm-scale circuit boards	not reported	38
Gracias, et al.	mm-sized polyhedra into helical aggregates w. 1-4 isolated electrical circuits	not reported	39
Jacobs, et al.	cylindrical display - 113 GaAlA's LED's - 280μm x 280μm x 200μm	assembly of 1500 chips w. 98% yield in 3min (5000 redundant parts)	40
Srinivasan, et al.	hexagonal micromirrors (464μm dia & 200μm thick) onto microactuators - binding site 200μm dia	fill factor of 95% (7 binding sites)	41, 42
	Si parts onto Si and quartz substrates - 150μm x 150μm x 15μm - 400μm x 400μm x 50μm	100% yield in 1min (array of 98 parts), w. 0.3deg rotational misalignment	43
Scott, et al.	helical & toroidal inductors (450μm x 950μm) on CMOS wafers	90% yield	44
Xiaorong, et al.	surface mount LED's	not reported	45
Fang and Bohringer // Jiandong, et al.	PZT's (4mm square) on pump chamber on 4" substrate	not reported	46, 47
Sheng-Hsiung, et al.	released DRIE comb drives on SOI wafers (1mm x 1mm x 200μm), substrate - 4x4 array	93% yield in 30s w. 100 components	48
Lee, et al.	3D assembly of 20μm - 100μm parts	not reported	49
Morris and Parviz	limitations on molten allowable size	97% yield for 100μm sized components & 80% yield for 40μm size components & 15% yield for 20μm size components	50, 51
Onoe, et al.	selective bonding - 3D sequential micro self assembly of 10μm components & microchain in two steps	1st step - 70% yield in 60min, 2nd step - 10% yield in 720min; max length of microchain - 6 units	52-54
Zheng and Jacobs // Kneel, et al.	selective bonding - assembly of 300μm sized LED's (36 red, green & yellow) & Si dies w. 72 interconnects	not reported	55, 56

Source: Adapted from [37]. w. = with; Type I = no shape recognition

Table 1.3 Summary of Parallel Assembly Techniques with Capillary, Type II Forces

Authors	Demonstration	Results	References
Zheng and Jacobs	AlGaN/GaN LED's (380μm x 330μm)	95% yield in 2min	55
	heterogeneous assembly of 3 non-identical chips - GaAs, Si, & GaP (200μm - 500μm)	not reported	57
Wei, et al. // Zheng, et al.	sequential assembly of 3 components - 600 LED's of 200μm size onto carries & encapsulation units onto carries	100% yield in 2min for LED's & 97% yield for encapsulation units	58-60
Zheng, et al.	angular & lateral orientation - parts 500μm - 2mm	angular orientation - 3deg, lateral orientation - 19μm	60
Knuesel, et al.	assembly of ultra small chips (20μm in length) & angular orientation using alignment pedestals	not reported	61
Fang and Bohringer	Si parts (790μm x 790μm x 330μm)	99% yield in 2min for 1000 receptor sites	62, 63
Fang and Bohringer // Jiandong and Bohringer // Fang, et al.	semi dry self assembly process (semi DUO-SPASS) w. orientation uniqueness (1-2mm square parts)	95% - 99% yield in 3min translational & rotational misalignment - 0.25mm & 18deg respectively	64-66
Jiandong and Bohringer // Fang and Bohringer	completely dry self assembly process (DUO-SPASS) w. orientation uniqueness (102mm square parts)	98% yield in 10min w. 50% redundant parts translational & rotational misalignment - 20μm & 2deg respectively	65, 67
Saeedi, et al. // Stauth and Parviz	heterogeneous assembly of FET's, diffusion resistors (100μm - 300μm)	97% yield in 3min for FET's & diffusion resistors	68, 69
Saeedi, et al	micro display - LED's (320μm)	65% yield for LED display	68, 70
Kim, et al. // Hosokawa, et al.	fluorescence detection unites (3x3 array)	not reported	71, 72

Source: Adapted from [37]. w. = with; Type II = shape recognition

Table 1.4 Summary of Parallel Assembly Techniques with Surface Tension Forces

Authors	Demonstration	Results	References
Fang and Bohringer	2D assembly of 100 components made of polyimide & polysilicon of 400μm size	not reported	62
Syms and Yeatman	3D assembly of hinged microstructures	97% yield in 1min	73
Green, et al. // Syms	3D assembly of hingeless microstructures	97% yield in 1min	74, 75
Syms	micro-otpomechanical torsion mirror scanner	not reported	76-78
	refractive collimating microlens arrays	not reported	79
Dahlman and Yeatman // Dahlman, et al.	integration of high Q inductors on IC's	99% yield in 5min	80-83

Source: Adapted from [37]. w. = with

Table 1.5 Summary of Parallel Assembly Techniques with Electric and Magnetic Forces

Authors	Demonstration	Results	References
Tien, et al. // Grzybowski, et al.	10um size gold disks on Si substrate, 2types of spheres in an ordered lattice	not reported	84, 85
Bohringer, et al.	surface mount capacitors & diodes (0.75mm - 2mm)	4 surface mount capacitors assembled in 30sec	86, 87
Nakakubo and Shimoyama	3D assembly of Si microstructures - 100μm concave & convex cubes	60% yield in 5min with 300 parts of each kind	88
Iwase and Shimoyama // Iwase, et al.	3D assembly of hinged microstructures, 3D assembly of 600μm x 800μm x 4.5um plates & 800μm long rectangular tetrahedrons	not reported	89-91
Grzybowski, et al. // Grzybowski and Whitesides	mm-scale magnetic disks	not reported	92, 93
Boncheva and Whitesides	assembly of planar elastomeric sheets into 3D objects and electrical circuit with LED's	3min for folding of sheets into an electrical circuit sphere	94
Fonstad // Rumpler, et al.	integration of semiconductor devices w. IC's	not reported	95, 96
Nichol, et al. // Anthony, et al.	folding of membranes pattered w. soft magnetic arrays using external magnetic field	translation alignment - 200nm	97, 98
Shet, et al.	assembly of GaAs or InP devices on semi-process or processes wafers w. integrated circuits	not reported	2, 99
Ramadan, et al.	parts of 1mm x 1mm x 0.5mm size w. electroplated CoNiP (1μm)	not reported	26

Source: Adapted from [37].

1.3 Magnetic Field Driven Simultaneous Assembly

The Magnetic Field Driven Simultaneous Assembly (MFDSA) contrasts all proposed parallel assembly techniques by seeking a non-statistical, deterministic solution to the problem of assembly.

MFDSA is a dry and not a wet process. It employs an array of electromagnets to drive the assembly process. It removes restrictions involving geometry, size and shape, including issues regarding orientation and symmetry/asymmetry. It is a room temperature process and therefore materials with different lattice and thermal properties can be integrated without physical damage. It uses programmed rather than random pathways and, therefore, is able to achieve a 100% yield after a single iteration. The only special preparation required to use MFDSA involves adding layers and strips of soft and hard magnetic material to devices and recesses respectively.

MFDSA adds soft and hard magnetic material on devices and in recesses respectively. The hard magnetic strips retain devices within recesses. The soft magnetic layers allow an array of electromagnets to direct the motions of devices.

An array of electromagnets suspends devices against the underside of a membrane and carries them toward their destinations. At the appropriate final locations, fields are weakened and devices are populated onto the template where they are kept temporarily. The template allows the method to correct errors if they arise. To complete the process, the substrate is pressed onto the template; devices adhere into recesses through the attraction between soft and hard magnetic materials.

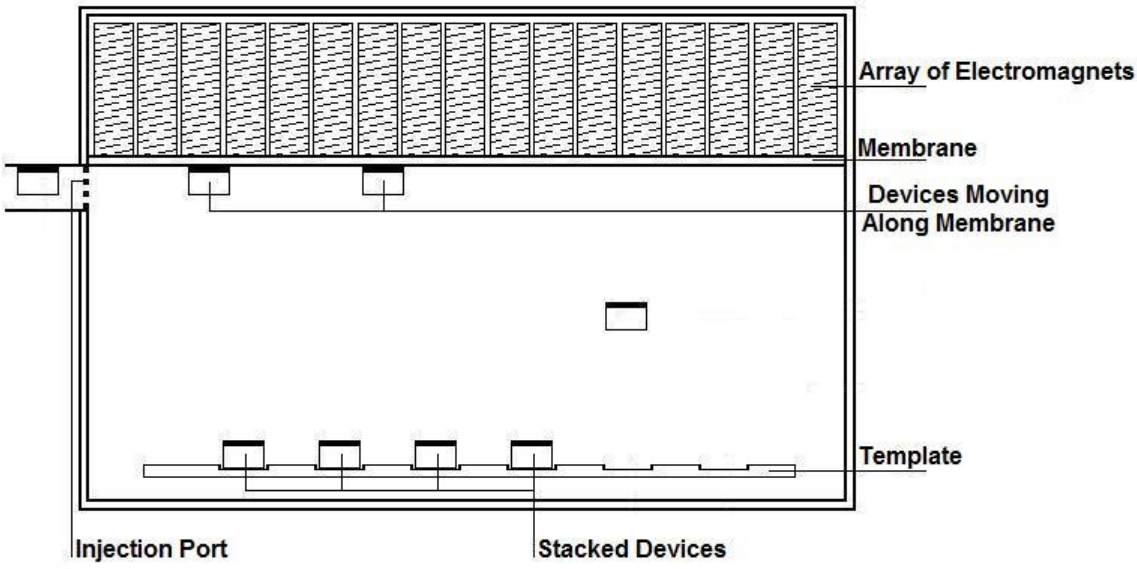

Figure 1.8 An illustration of the Magnetic Field Driven Simultaneous Assembly (MFDSA) technique showing a cross-section of the approach [1].

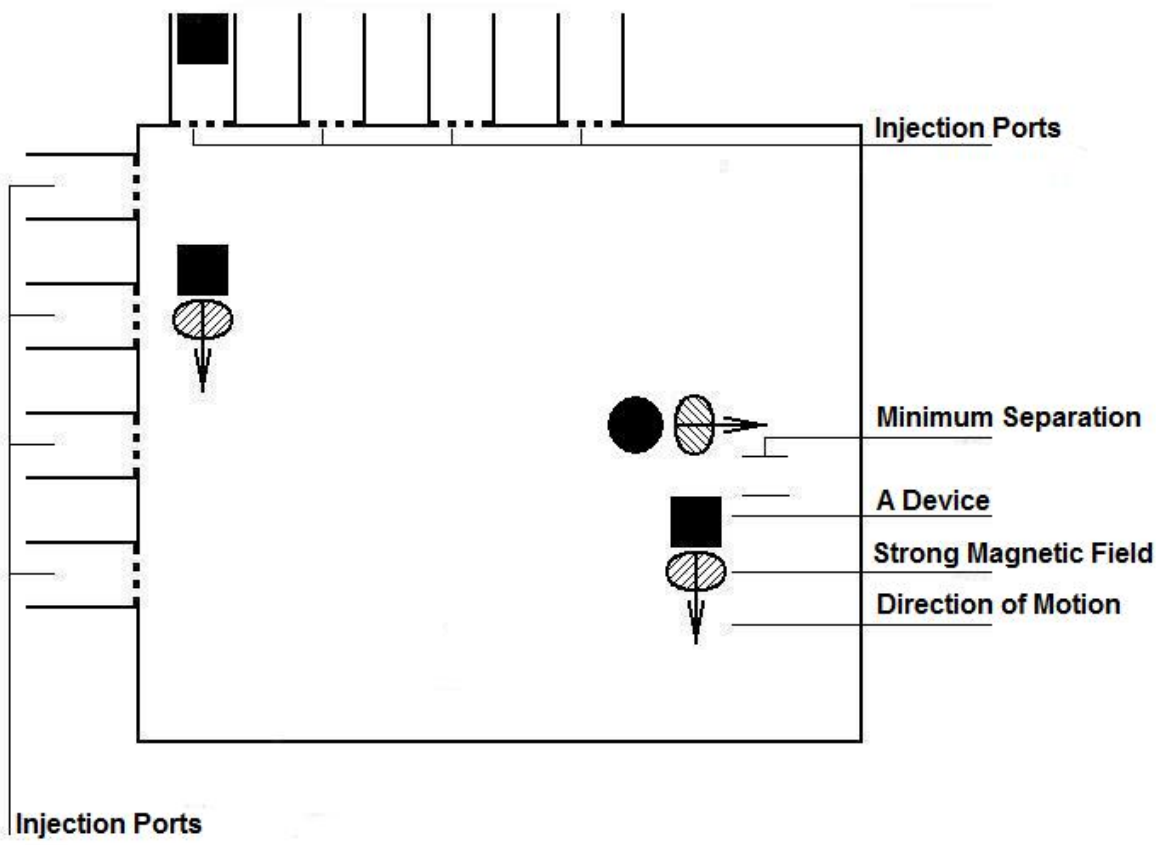

Figure 1.9 A top-down view of the Magnetic Field Driven Simultaneous Assembly (MFDSA) that illustrates the simultaneity of the technique [1].

1.4 Discussion of Magnetic Field Driven Simultaneous Assembly

1.4.1 Enclosure and Lower/Upper Chambers

The process of assembly is controlled within the enclosure (see Figure 1.8). The enclosure is encircled with the injection ports that are situated at intervals about its perimeter and through which devices enter (see Figure 1.9). The enclosure is divided by the membrane into two chambers. The lower chamber contains devices, substrate, and template and should be in vacuum. The upper chamber contains an array of electromagnets. The membrane itself is kept rigid between these chambers by a suitable construction or framework.

The upper chamber contains an array of electromagnets along with a function to cool its elements, such as a heat sink or a heat bath. It contains various other leads that connect the equipment to an external control unit, which programs each and every element of the array of electromagnets. The array produces localized magnetic fields that can be varied in magnitude, direction, and forces and can be used to manipulate the positions of devices along the underside of the membrane. Optionally, the elements of the electromagnetic array may terminate with materials shaped into geometries that intensify and localize their magnetic fields. Also, the elements of the electromagnetic array may contain internal auxiliary mechanisms to aid the dislodging of devices away from the membrane toward the template.

The membrane separates the upper and lower chambers of the enclosure. The purpose of the membrane is two-fold; first, to protect devices from damage that may be caused through direct physical contact with the electromagnetic array; second, to provide a surface across which devices are moved.

The lower chamber is where devices are injected, positioned, and inserted into the substrate. Although not depicted in the figures, the lower chamber contains various electronic components, which are connected to and controlled by an external control unit. Such components would be the following: accuracy control sensors and mechanisms, pressure and temperature regulators, and other real-time sensing feedback equipment required to facilitate assembly.

1.4.2 Simultaneous Assembly of Devices

Devices are coated with a layer of soft magnetic material; the permeability of the material and the thickness of the layer are to be such that it allows an array of electromagnets to suspend the component against gravity while minimizing its contribution to its weight as a whole.

Devices enter the enclosure through the injection ports and are held against the underside of the membrane by an array of electromagnets. The array generates localized magnetic fields that engulf each and every device at the membrane. These fields are generated at certain rates, at certain paths and draw devices from initial to final positions immediately above matched recess sites at the template. The process of assembly consists of manipulation of multiple devices independently provided such that the localized magnetic fields are short-ranged with respect to dimensions of devices and that devices are separated beyond distances referred to as Collision Cross-Section Areas (CCAs) (see Figure 1.9).

Devices are advanced toward their final desired locations and are then disengaged from the membrane by weakening the localized magnetic field below a threshold. Each and every device requires a minimum strength of the localized magnetic field to keep it suspended against gravity and if that field is weakened below that threshold, they fall. In the absence of an atmosphere, they land without deflection onto the template (see Figure 1.8). Note that sticktion is only an issue with devices of very light masses and sizes; modifications to the apparatus would be required to overcome sticktion if it arises.

1.4.3 Template

The template is a magnetically passive construction upon which devices are populated temporarily. Locations on the template correspond to matched recess sites along the substrate. Essentially, they serve as a negative of the substrate. The orientation of the template can be altered and the distance between the membrane and the template can be varied. The template can be a sheet, a strip, or a collection attached to a mechanism that switches among different templates. The template can be a single piece of material or composed of interchangeable and/or interlocked parts.

1.4.4 Injection Ports

The injection ports are located at intervals about the perimeter of the enclosure. They can be operated either mechanically or electromagnetically. They connect the enclosure to the bins (not depicted by the figures) which contain devices prior to assembly. Note that the bins may contain either one type of device or a known and/or controllable pattern of devices and are to be kept evacuated to preserve the vacuum of the enclosure. The interface between the injection ports and the enclosure is a partition that maintains the integrity of the vacuum within the enclosure.

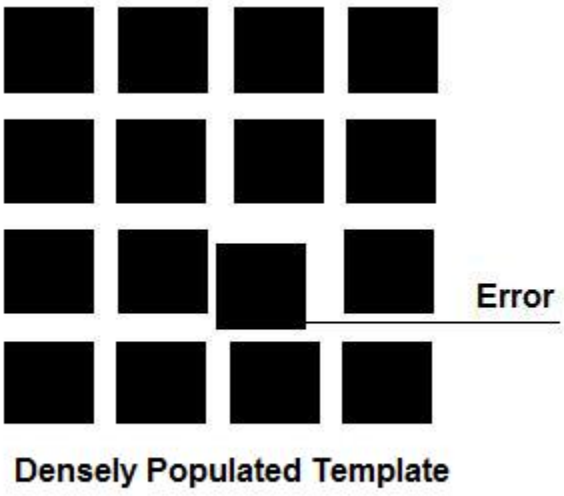

Figure 1.10 A schematic view showing the template with a device placed incorrectly. An error within a densely populated template will be difficult to fix because there are many devices around the error that would be affected [100].

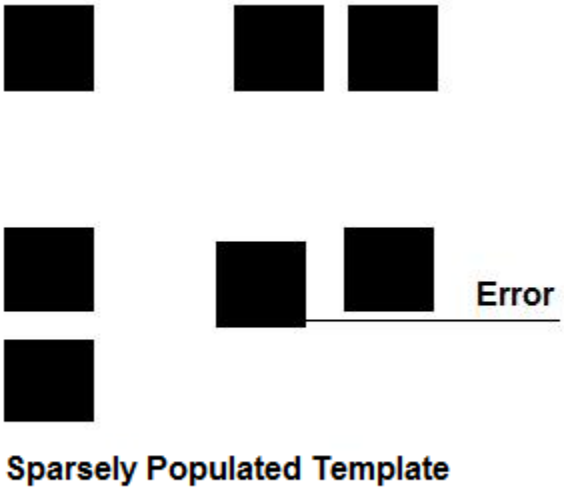

Figure 1.11 A schematic view showing the template with a device placed incorrectly. An error within a sparsely populated template will not be difficult to fix because there are few devices around the error that could be affected [100].

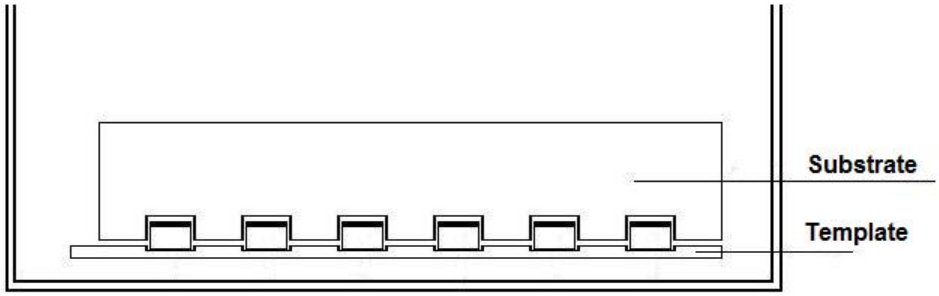

Figure 1.12 A cross-section view of the Magnetic Field Driven Simultaneous Assembly (MFDSA) that illustrates the integration of devices into recesses in the substrate [1].

1.4.5 Error-Correction, Dense/Sparse Population, and Integration

If devices are not populated onto the template within a certain allowable tolerance, then an error-correction algorithm is activated (see Figures 1.10 and 1.11). The template is raised toward the membrane and the array activates above those devices that were not properly populated. These devices reattach onto the membrane and may be repositioned within tolerance.

To complete assembly, the template may be densely or sparsely populated by devices (see Figures 1.10 and 1.11). In the case where the template is densely or fully populated, then integration is completed through a single insertion step. In the case where the template is sparsely or partially populated, then integration is accomplished through stages. Whether the assembly is accomplished by densely or sparsely populated variations (controlled by the fraction of devices which have been dropped onto the template), the integration requires that the substrate is placed in contact with the template; for example, by pressing or rolling (see Figure 1.12).

On the substrate, devices are physically secured into recesses by the attraction of soft magnetic layers and hard magnetic strips between devices and recesses respectively.

1.5 Dissertation Outline

Chapter 1 is an introduction to serial and parallel assembly methods including their history, status, and trends. It thoroughly presents and discusses an overview of the Magnetic Field Driven Simultaneous Assembly technique.

Chapter 2 discusses previously published work involving the development of Magnetic Field Driven Simultaneous Assembly.

Chapter 3 outlines advanced, unpublished work about the magnetic interaction model and the Swarm Algorithm, covering many core concepts including the Collision Cross-Section Area, the role of friction and the Magnetic Retention Factor.

Chapter 4 presents an in-depth development of the magnetic interaction model as well as a summary of comparative results.

Chapter 5 presents the Swarm Algorithm, which is used to calculate the pathways that are required by the devices as they move from initial to final positions, along with the results of several cases.

Chapter 6 is devoted to the conclusions of the study and recommendations for future work.

CHAPTER 2

BASIC MODELING

2.1 Abstract

The Magnetic Field Driven Simultaneous Assembly (MFDSA) is a parallel assembly technique that emerged out of advances in research and development regarding the standard fluidic self-assembly method and its variants. It is a synthesis of such approaches as the Magnetically Assisted Statistical Assembly (MASA), developed at the Massachusetts Institute of Technology (Cambridge, Massachusetts), and the Magnetic Field Assisted Assembly (MFAA), proposed by the team at the New Jersey Institute of Technology (Newark, New Jersey). The feature that sets the MFDSA technique separate and apart from its predecessors is an emphasis on deterministic assembly as opposed to a reliance on statistical assembly.

In a deterministic process, the pathways that devices follow as they move from initial to final positions, including the rate at which they are injected, populated, and inserted, need to be calculated. In a statistical process, these device-by-device, move-by-move details are ignored because randomness dictates the act of assembly as devices seek recesses. For example, in fluidic self-assembly, devices are carried by the fluid; the pathways they take, their success or failure to locate a recess, and whether they enter it correctly or incorrectly is random (strong frustration). Further, they may be clumped together or they may be scattered away; in either case, they do not participate in assembly (weak frustration).

The ostensible advantage of statistical techniques over deterministic methods is that they are 'quick and dirty'; therefore, they do not require solutions to the problem of assembly. Its ease-of-use is invalidated, however, by such factors as device/recess requirements, strong and weak frustration, and the fact that a single iteration of the approach is not enough to achieve high-yield integration. Yet, while a deterministic method does require solutions to the problem of assembly that a statistical technique is free to ignore, that aspect of its implementation is ultimately the cause of its flexibility, scalability, and high-yield gain.

2.2 The Limitations of Conventional Parallel Processing

2.2.1 Areas of Refinement

All refinements to fluidic self-assembly have been attempts to moderate its randomness. Two models have been developed to achieve that goal. First, to control the way that devices enter recesses (strong frustration). Second, to increase the probability that devices enter recesses (weak frustration).

Techniques, investigated at the Alien Technology Corporation, alter the standard fluidic self-assembly method through the introduction of asymmetric device/recess geometry. The effect of the asymmetry is to correct the orientations of devices while they fall into recesses [27]. The work of Zheng et al. adds special auxiliary sites about the surface of the wafer to allow the fluid to correct the orientations of devices while they advance toward recesses [28]. The work of Lin et al. combines asymmetric device/recess geometry and surface tension effects to drive a self-correcting, self-assembly type of integration of devices into recesses [29].

Another refinement is the MASA method of Cheng et al. [30]. The objective of the MASA approach is to increase the probability of assembly by aiding devices along their otherwise random search of recesses. It is accomplished by adding soft and hard magnetic materials onto devices and into recesses respectively. As they are carried by the fluid, the magnetic attractions between those soft and hard magnetic layers drive devices in the direction of recesses. The attraction retains devices inside recesses thus counteracting a tendency of the fluid to dislodge already assembled components.

2.2.2 Geometric Restrictions

The inability to integrate various devices of different types, geometries, and sizes onto a single substrate is an issue that remains intractable with respect to the standard fluidic self-assembly method and its variants. The conventional parallel assembly methodology is useful if and only if all of the devices, to be integrated, are identical. This restriction is fundamental and is due to the fact that the act of assembly is random, i.e., a square device would be just as likely to reach a square recess as a circular recess. If that restriction is not enacted, then, clumps of devices of various geometries could be competing for a recess even if it is incompatible; further, smaller devices could be entering larger recesses, which creates potentially irreversible defects especially if an approach like MASA is used.

2.2.3 Strong and Weak Frustration

The examples above represent the two facets of frustration.

Strong frustration arises when there is a mismatch between device and recess, including errors caused by a misaligned (or distorted) insertion. It is 'strong' because these errors are 'permanent', i.e., difficult if not impractical to correct without destroying the assembly already completed.

Weak frustration manifests when there is a competition among a number of devices (of similar or dissimilar types) for access to a single recess. It is 'weak' because it is an error of the process, not of the assembly. It is temporary and due to the medium carrying the devices; clumps of devices may appear and then scatter as the fluid carries them across the surface of the substrate.

The conventional parallel assembly paradigm addresses the strong side of frustration by prohibiting variation in type, geometry, and size. Essentially, all devices are identical, all recesses are identical, and are built to match. For example, fluidic self-assembly demands that devices and recesses are built with matching trapezoidal geometry (see Figure 1.4); devices enter recesses if and only if they approach with a certain orientation, any other orientation and those misalignments are carried away by the fluid [4].

2.2.4 Moderating the Effect of Frustration

The MFAA technique, as proposed by Shet et al. [2], exemplifies an attempt to circumvent the issue of frustration and remove the device type, geometry, and size restrictions imposed by the other parallel assembly methods. It employs ideas from MASA, specifically, the soft magnetic layer on devices and the hard magnetic strips in recesses. It adds a novelty that transforms it into a different, more deterministic than statistical approach, namely, an external magnetic field to direct devices into recesses as they move along the surface of the substrate.

The external magnetic field represents a fundamental conceptual advancement over the standard fluidic self-assembly method and its variants. It is the tool through which the nature of assembly shifts from statistical to deterministic; however, it fails to address the issue of frustration thoroughly. It does improve strong frustration, as the field is able to direct the pathways devices take to reach recesses. It does not improve weak frustration, as devices are able to clump about recesses.

The MFAA technique is able to resolve the limitation imposed on device type, geometry, and size. To achieve that practicality requires the assembly of the largest devices first, the next largest devices second, and so forth until the smallest devices are last. It requires several iterations to achieve a complete assembly; each type of device would be its own iteration.

Table 2.1 Parallel Processing Techniques and Strong/Weak Frustration

Technique	Frustration	Comments
Fluidic Self-Assembly	strong and weak	devices and recesses must be symmetric [4]
Zheng, Lin Refinement	weak	uses sites/forces to address strong frustration [28, 29]
MASA	strong and weak	magnetism is used to retain devices into recesses [24]
MFAA	weak	magnetism is used to drive assembly [25]

2.3 A Solution to Frustration

The MFDSA technique, developed by Rivero et al. [5], addresses frustration and other fundamental limitations of extant parallel assembly methods. It emphasizes total and indiscriminant simultaneity as it is capable of assembling a wide variety of devices at any given time. The template, upon which devices are populated and then inserted into the substrate, divides the assembly into stages; therefore, it is a batch processing approach, a hybrid of serial and parallel assembly. While this may appear to be a regression, however, the advantage of the template is manifold, including the ability to check and correct errors in placement before the actual integration of devices into recesses occurs.

The MFDSA technique eliminates the issue of frustration. The standard fluidic self-assembly method and its variants do not control the act of assembly; where the devices enter and exit, the directions they take and the recesses they seek are examples of its unpredictability; thus, its randomness is the cause of its frustration. The determinism of MFDSA is the opposite of the statistics of fluidic self-assembly; therefore, it does not suffer the effect of frustration.

Implementing a non-statistical regime of assembly is the key to eliminate all forms of frustration. It can be accomplished if and only if there is a plan for where devices enter and exit, the order of devices, and the pathways they take toward recesses. It requires another level of preparation embodied by the solution to the problem of assembly itself, which will be presented as the Swarm Algorithm (SA).

The batch processing approach is a trade-off that allows a high-yield with a single iteration of the system, a throughput matched only by 'pick and place'. Figure 2.1 illustrates a comparison of pure serial, pure parallel and hybrid batch assembly techniques. It is based on an attempt to assemble a hundred components onto a grid of ten by ten squares where each component and each square is identical. Pure parallel assembly attempts to integrate all components simultaneously. Pure serial assembly integrates each and every component individually. Hybrid assembly integrates a number of devices on a per-batch basis until all of the devices have been integrated.

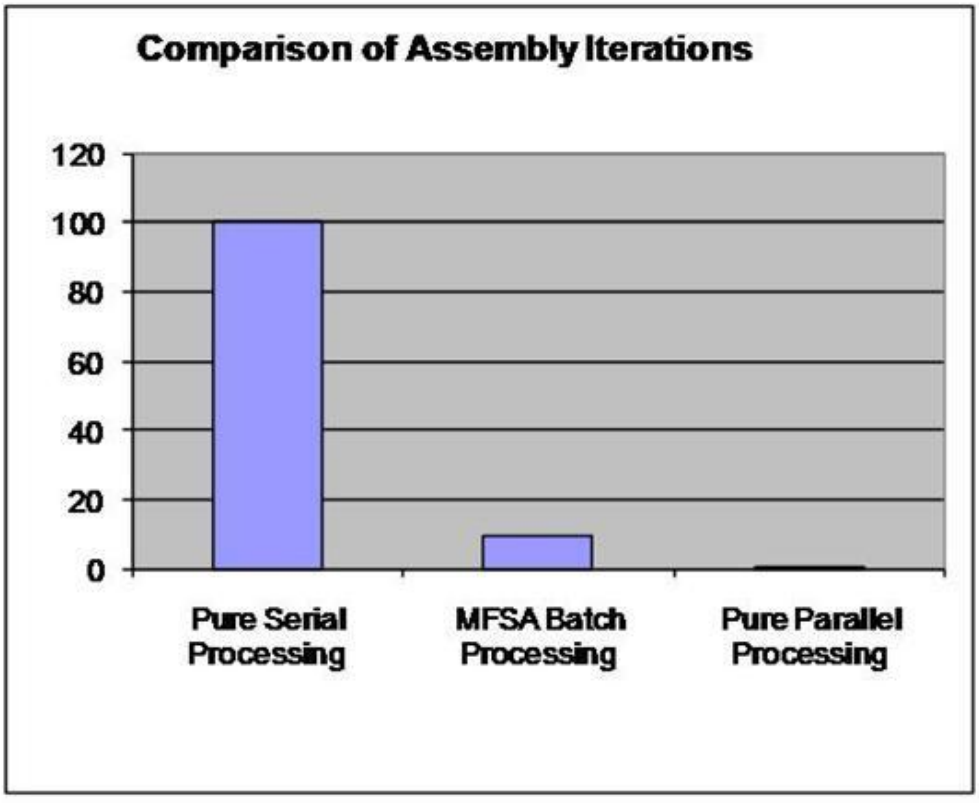

Figure 2.1 Comparison of Assembly Iterations. The figure is based on the assembly of a hundred components into a grid of ten by ten boxes. Pure Serial Processing requires a hundred iterations while Pure Parallel processing requires a single iteration. Batch Processing, a hybrid of serial and parallel methods, requires ten iterations of ten components per insertion [100].

2.4 Basic Soft/Hard Magnetic Field Modeling

The point of intersection between MFAA, MASA, and MFDSA involves the retention force between devices and recesses given by the interaction of soft and hard magnetic materials. A detailed discussion of that interaction is required to understand the force involved and how it can be maximized.

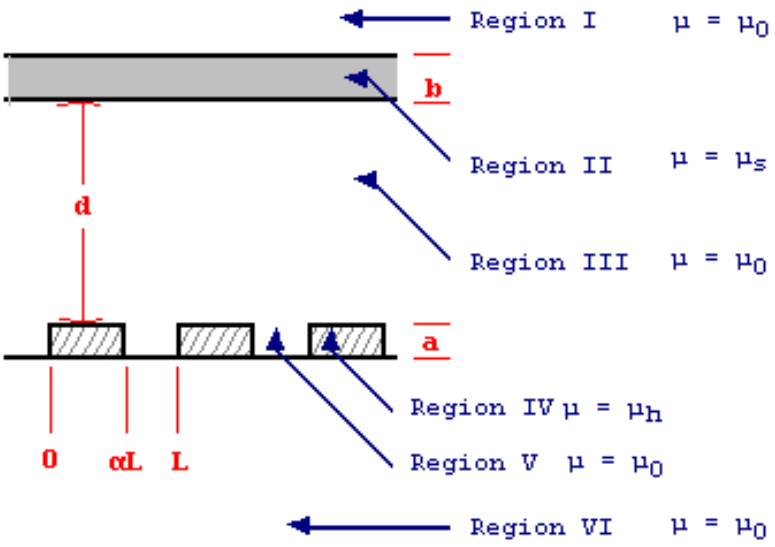

Figure 2.2 A schematic of the system's physical parameters. The regions are as follows: Region I is the device (of permeability μ_o), Region II is the soft magnetic layer (of permeability μ_s and thickness b), Region III is the 'air' between the device and the recess (of permeability μ_o and variable displacement d), Region IV is the hard magnetic strip (of permeability μ_h, thickness a, and width αL), Region V is the 'air' gap between the strips (of permeability μ_o, thickness a, and width (1-α)L), and Region VI is the substrate (of permeability μ_o).

The parameters of the device/recess system are summarized by Figure 2.2. The soft magnetic layer is uniform and coats the bottom of the device. The thickness of the layer is b and is of permeability μ_s (Region II). The hard magnetic strips are patterned evenly with a periodic size scale at the bottom of the recess. The thickness of the strip is a and is of permeability μ_h (Regions IV and V). L is the periodic size scale of the pattern, where αL is the width of the strip, and (1-α)L is the width of the gap; α is a parameter with a range of $0 \leq \alpha \leq 1$. The variable displacement between the layer and the strips is d in the y-axis (Region III). Note that the simplification of the problem is that the devices and recess extend indefinitely from end to end in the x-axis; also Region I represents the body of the device and Region VI represents the wafer.

An approach to the model that gives an estimate of the attraction between the layer and the strips will be developed below. Since there are no current sources and no electric fields or charges, Maxwell's equations reduce to:

$$\boldsymbol{\nabla} \times \boldsymbol{H} = 0 \; and \; \boldsymbol{\nabla} \cdot \boldsymbol{B} = 0 \tag{2.1}$$

Therefore, there exists a magnetostatic potential, φ, such that **H** = -**grad** φ everywhere. Considering linear, isotropic relations between the magnetic field **H** and the magnetic induction **B**, then:

$$\boldsymbol{B} = \mu_0 \boldsymbol{H} \tag{2.2}$$

$$\boldsymbol{B} = \boldsymbol{\mu_s} \boldsymbol{H} \tag{2.3}$$

$$\boldsymbol{B} = \boldsymbol{\mu_0} (\boldsymbol{H} + \boldsymbol{M_0}) \tag{2.4}$$

where, μ_s is the permeability of the soft magnetic layer and $\mathbf{M_0}$ is the known permanent magnetization of the strips. Equation 2.2 is valid in air, Equation 2.3 is valid in the soft magnetic layer, and Equation 2.4 is valid in the hard magnetic strips.

Generally, φ satisfies the Poisson's equation, with source term **div** $\mathbf{M_0}$. The magnetization, $\mathbf{M_0} = M_0$ **y**, is considered constant, with **y** as the unit normal vector in the y-axis, so that the source tem is zero and Poisson's equation becomes Laplace's equation.

$$\nabla^2\phi = 0 \tag{2.5}$$

The boundary conditions at the interfaces between the air and either the soft or the hard magnetic material follows from Equation 2.1, which implies that, at an interface, the tangential components of **H** and the normal components of **B** are continuous.

In terms of the potential, φ, using Equations 2.2-4, we have the condition that φ is continuous at the interfaces y = a + d (the bottom surface of the soft magnetic layer) and y = a + b + d (the top surface of the soft magnetic layer):

$$\mu_0\partial_y\phi_{|A} = \mu_0\partial_y\phi_{|S} \tag{2.6}$$

At the vertical sides, x = 0, x = αL and x = (1-α)L of the hard magnetic strips:

$$\partial_x\phi_{|A} = \partial_x\phi_{|H} \tag{2.7}$$

At the horizontal sides, 0 < x < αL, y = 0 and y = a of the hard magnetic strips:

$$\partial_y\phi_{|A} - \partial_y\phi_{|H} = -M_0 \tag{2.8}$$

where, the A, S, H denote the evaluation at the air, soft or hard interfaces.

The only forcing or inhomogeneity term in the model to calculate φ comes from the boundary condition at Equation 2.8.

The problem is periodic, with period L in the x-axis, and can be solved by constructing a Fourier Series in each of the regions shown in Figure 2.2 and then applying the continuity and boundary conditions at the interfaces with the extra condition that φ is constant as y goes to ±∞. These conditions lead to a linear algebraic system for the coefficients that can be found in closed form.

An expression for the force acting on the soft magnetic layer in response to the magnetization of the hard magnetic strips follows by evaluating the integral of the Maxwell stress-tensor over the layer's top and bottom surfaces. The general expression of the force is:

$$\boldsymbol{F} = \mu \int_{\partial\Omega} \boldsymbol{H}(\boldsymbol{H}\cdot\hat{\boldsymbol{n}}) - \frac{1}{2}(\boldsymbol{H}\cdot\boldsymbol{H})\hat{\boldsymbol{n}}\, dS \tag{2.9}$$

where, ∂Ω is a surface immediately outside of the region of interest, **n** is the outward unit normal vector and μ is the local permeability. Then, in terms of φ,

$$\boldsymbol{F} = \frac{\mu_0}{2}\left(\int_{top} \partial_y\phi^2 - \partial_x\phi^2 dx - \int_{bottom} \partial_y\phi^2 - \partial_x\phi^2 dx\right)\hat{\boldsymbol{y}} \tag{2.10}$$

where the integrals over the top and bottom surfaces are evaluated at y = a + d and y = a + b + d respectively with $0 \leq x \leq L$.

When the constructed Fourier Series is substituted into φ of Equation 2.10, the expression for force is given by:

$$\boldsymbol{F} = -\frac{\mu_0 L}{2}\sum_{n=1}^{\infty}\frac{M_0^2}{\pi^2 n^2}\left(1-\cos\left(2\pi n\alpha\right)\right)e^{-4\pi n\frac{d}{L}}\left(1-e^{-2\pi m\frac{a}{L}}\right)^2\frac{sinh\left(2\pi n\frac{b}{L}\right)}{sinh\left(2\pi n\frac{b}{L}-ln\left(\frac{\mu_S+\mu_0}{\mu_S-\mu_0}\right)\right)}\hat{\boldsymbol{y}} \tag{2.11}$$

The terms with n > 1 drop rapidly to zero leaving the most significant term as the n = 1 term.

$$\boldsymbol{F} = -\frac{\mu_0 L}{2}\frac{M_0^2}{\pi^2}\left(1-cos\left(2\pi\alpha\right)\right)e^{-4\pi\frac{d}{L}}\left(1-e^{-2\pi\frac{a}{L}}\right)^2\frac{sinh\left(2\pi\frac{b}{L}\right)}{sinh\left(2\pi\frac{b}{L}-ln\left(\frac{\mu_S+\mu_0}{\mu_S-\mu_0}\right)\right)}\hat{\boldsymbol{y}} \tag{2.12}$$

The force is strong short-ranged and weak long-ranged. It falls to zero rapidly beyond a distance of d/L > 0.5, which is reflected by its use in MASA as a bond to keep devices and recesses together. While the device and the recess are assembled mechanically, when the magnetic materials make contact, they are secured in place as long as $|\mathbf{F}| \geq |\mathbf{W}|$, where **W** is the weight of the device.

2.5 Discussion

The force expressed through Equation 2.12 is maximized when α = 0.5; it follows after a derivative, dF/dα, is set to zero. The extrema within the range of α are 0, 0.5 and 1; however, α = 0 and α = 1 lead to a force equal to zero. When α = 0, F is zero, which follows because α = 0 is equivalent to a hard magnetic strip without a width. When α = 1, F is zero, again, that is because α = 1 implies a uniform hard magnetic layer and a uniform potential; as force is related to the gradients of potentials, a constant potential yields a zero force.

The ratio of hyperbolic sines, which cannot explode to infinity, yields a condition that relates the soft magnetic layer's thickness and permeability:

$$\mu_S > \mu_0 \frac{e^{2\pi\frac{b}{L}} + 1}{e^{2\pi\frac{b}{L}} - 1} \tag{2.13}$$

or:

$$\frac{b}{L} > \frac{1}{2\pi} ln\left(\frac{\mu_s + \mu_0}{\mu_s - \mu_0}\right) \tag{2.14}$$

The relationships expressed by Equations 2.13-14 limit the lowest values of thickness and permeability, which lead to a physically valid solution and will be important to the Magnetic Retention Factor (MRF) developed later. Two behaviors should be noted. First, when μ_S is significantly larger than μ_0, then the logarithm term is effectively zero and the hyperbolic sine ratio is unity. Second, when b/L is significantly larger than the logarithm term, then, again, the hyperbolic sine ratio is unity. In either case, neither the thickness nor the permeability of the soft magnetic layer contributes to the force, a fact that is reflected by Figure 2.3.

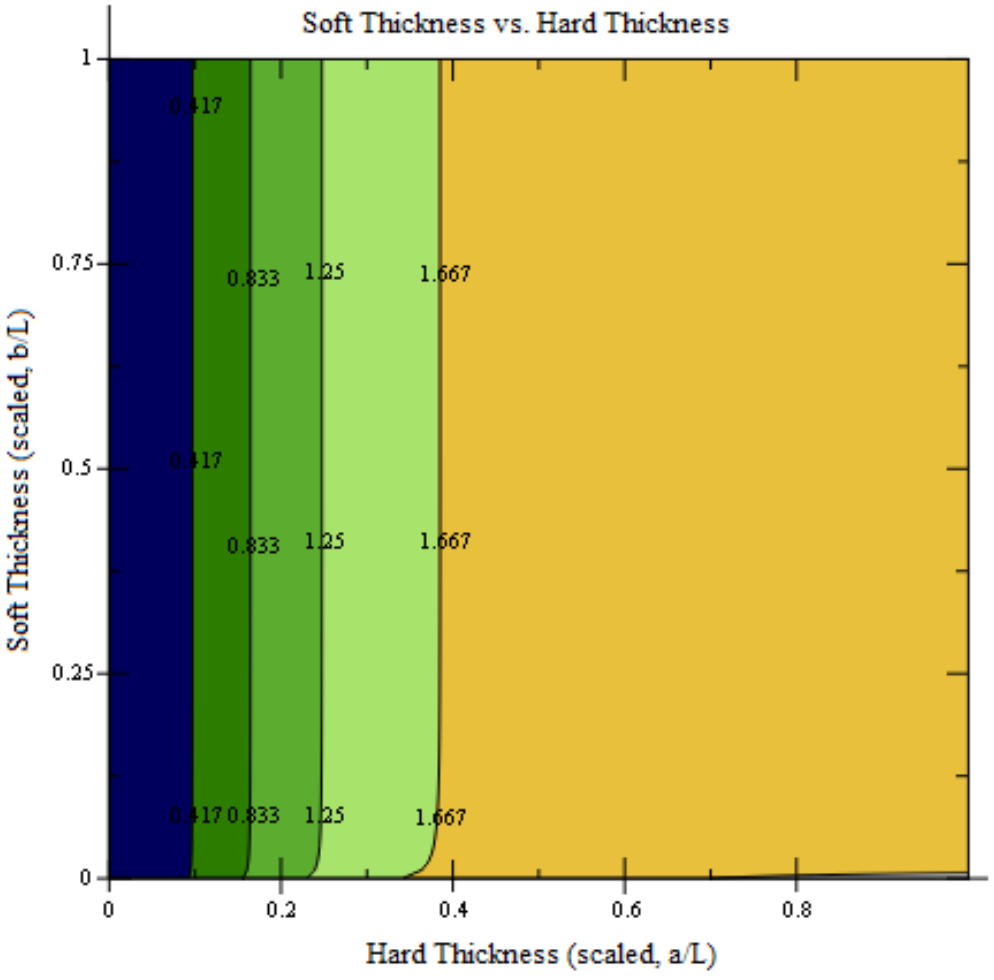

Figure 2.3 The effect of soft thickness and hard thickness on the force at contact (d/L = 0). This plot was found for a situation where $\alpha = 0.5$ and the relative permeability of the soft magnetic layer is 500. The scaled soft thickness (b/L, y-axis) and hard thickness (a/L, x-axis) varies from 0 to 1. The plot shows the scaled force (numbers within the plot) remaining virtually constant at any given scaled hard thickness through a range of scaled soft thickness values. (Note blue region is minimum, green to yellow is increasing force values.)

With respect to total scaled force output, the scaled soft magnetic thickness (b/L) is not as important a factor as the scaled hard magnetic thickness (a/L). As depicted by Figures 2.4 and 2.5, however, upon contact, i.e., d/L = 0, the scaled force is maximized at a/L = 0.8; a thickness greater than that value does not increase the force. That is because the force is stronger at the edges and weaker at the flat (or uniform) areas. Increasing the thickness leads to a situation where there is more flat area than edge area; eventually, only the edges on the top contribute, as the edges on the bottom will be too far away. Cutting the hard magnetic layer into a pattern maximizes the force of attraction by increasing the number of edges where the force is strong.

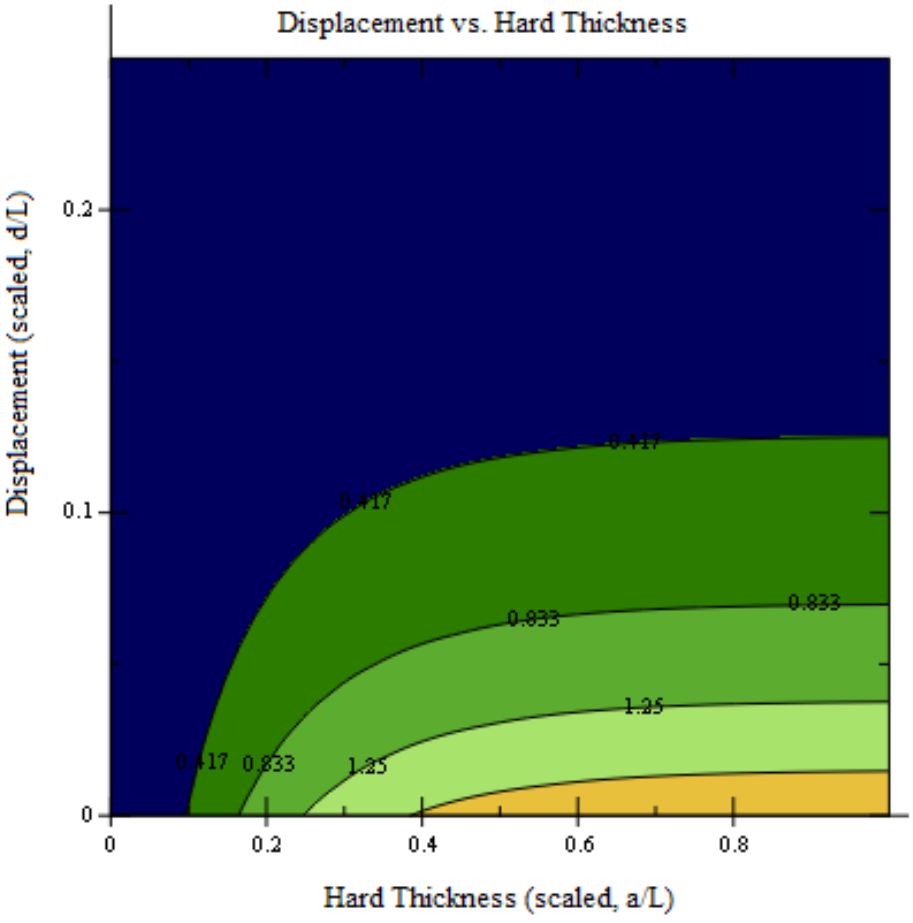

Figure 2.4 The effect of displacement and hard thickness on the force. This plot was found for a situation where $\alpha = 0.5$, the scaled soft thickness (b/L) is 1, and the relative permeability of the soft magnetic layer is 500. The scaled displacement (d/L, y-axis) varies from 0 to 0.25. The scaled hard thickness (a/L, x-axis) varies from 0 to 1. The plot shows that the scaled force (numbers within the plot) at contact (d/L = 0) saturates as a/L increases. (Note blue region is minimum, green to yellow is increasing force values.)

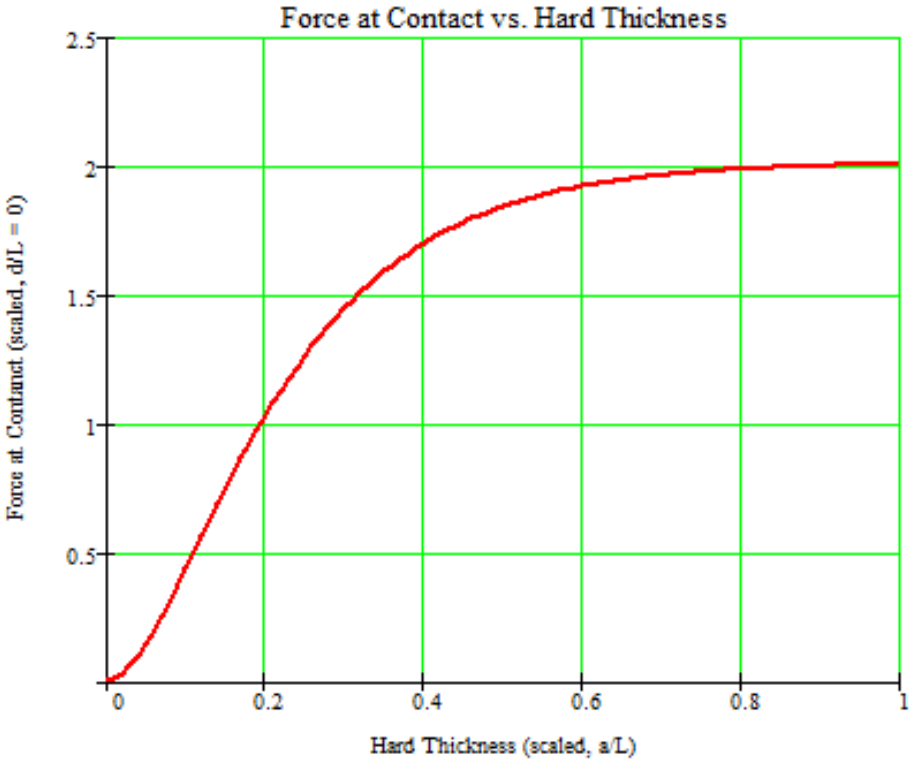

Figure 2.5 A plot of force at contact vs. hard thickness. This plot was found for a situation where $\alpha = 0.5$, the scaled soft thickness (b/L) is 1, and the relative permeability of the soft magnetic layer is 500. The scaled hard thickness (a/L, x-axis) varies from 0 to 1. The plot shows that the scaled force (y-axis) at contact (d/L = 0) saturates at about a/L = 0.8.

CHAPTER 3

ADVANCED MODELING

3.1 Abstract

The Magnetic Field Driven Simultaneous Assembly (MFDSA) is a complex problem to study due to its various layers of abstraction. MFDSA is divided into two general areas in order to model the topic efficiently and effectively. The first area is the magnetic field interaction between the array of electromagnets and devices. The second area is the calculation of pathways that devices are required to follow from initial to final positions.

The array/device interaction is the mechanism that drives the process of assembly. The magnetic field suspends devices below the membrane and above the template; it moves them from initial to final positions. The magnetic field interaction model calculates the forces involved with suspension and motion. Friction is a component of the motion and is factored into the algorithm. Additionally, such concepts as the Collision Cross-Section Area (CCA) and the rules of assembly are developed as prerequisites to achieve a successful hybrid assembly solution.

The pathways are determined by the Swarm Algorithm (SA) coined, developed, and encoded by the author. SA is an abstraction that separates the array/device interaction from the motion; however, certain features are connected to real, physical aspects like space, time, and the CCA. The goal is to extract the solution of the problem of assembly as a sequence of predetermined, precalculated 'quantized' (discrete and step-wise) movements which would be translated into real, physical movements.

Finally, MFDSA itself is compared and contrasted with topics from other branches of science.

3.2 The Rules of Assembly

The rules of assembly must be followed to assure that the MFDSA method does not fail. They are logical, as opposed to empirical, and follow from a consideration for the physical act of assembly. The magnetic field interaction model and SA as developed incorporate these rules within their algorithms.

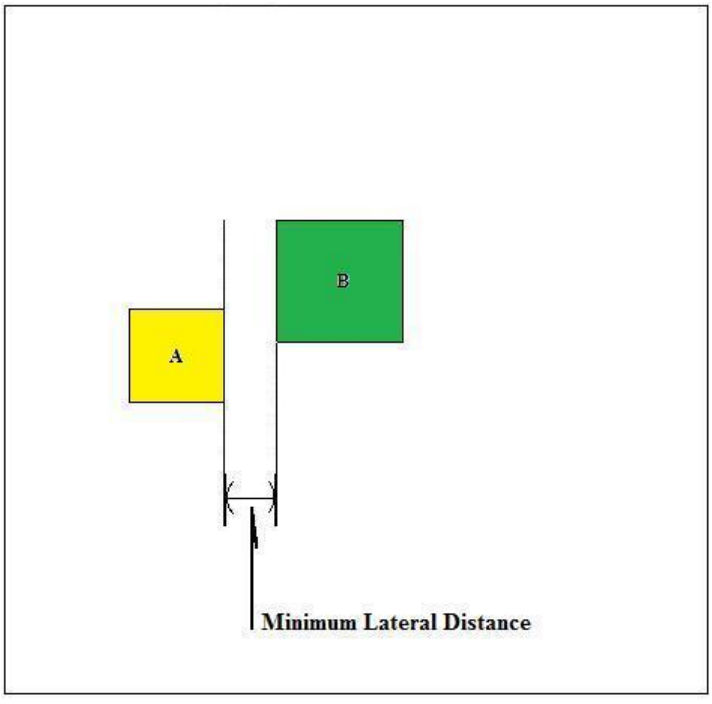

Figure 3.1 A diagram showing two devices, A and B, which are moving along the membrane simultaneously. To conform to the first rule of assembly, devices A and B must be kept a certain minimum lateral distance apart.

3.2.1 First Rule

Devices, as they move along the membrane, cannot collide physically with each other. Following this rule prevents weak frustration. The effect of this rule is that MFDSA maintains a low device density (defined as the number of devices within an area of the membrane). Devices are kept at a minimum lateral distance to avoid collision and to yield freedom of movement (see Figure 3.1); this combination of factors prevent the conditions that give rise to weak frustration. The minimum lateral distance is given by the CCA calculation of the magnetic field interaction model. The low device density is implemented by SA.

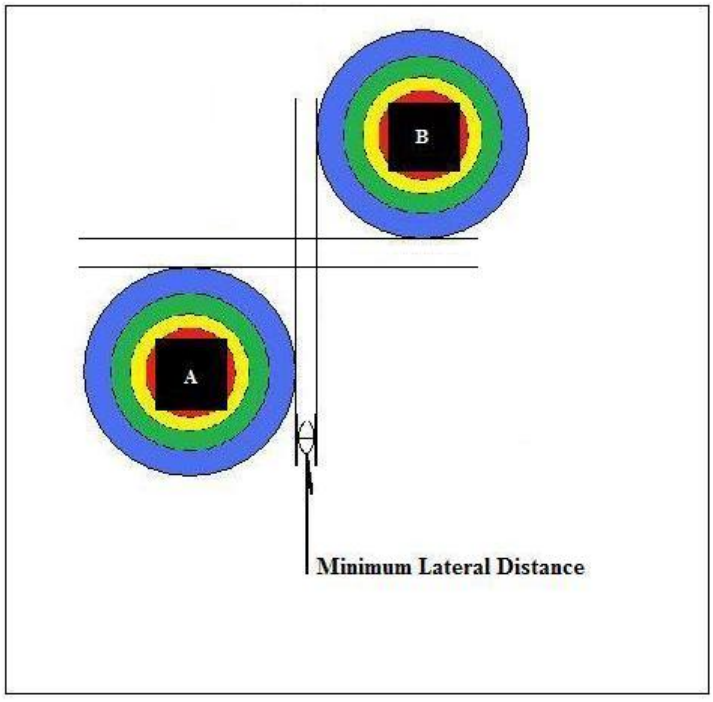

Figure 3.2 A diagram showing two devices, A and B, which are moving along the membrane simultaneously. The concentric colored circles around devices are a representation of the magnitude of the fields; red is the strongest region, blue is the weakest region. To conform to the second rule of assembly, the fields manipulating devices A and B must be kept a certain minimum lateral distance apart.

3.2.2 Second Rule

Fields, that suspend and move a device, cannot interact with any other device (see Figure 3.2). If there were to be a 'collision' between fields, then the result would be non-linear and uncorrectable interference among devices. The MFDSA approach is deterministic, therefore, requires detailed, step-by-step control over the process of assembly. Allowing fields that act on a device to effect other nearby devices would be equivalent to the introduction of chaos. The CCA is calculated to account for the extents of the device and the field together.

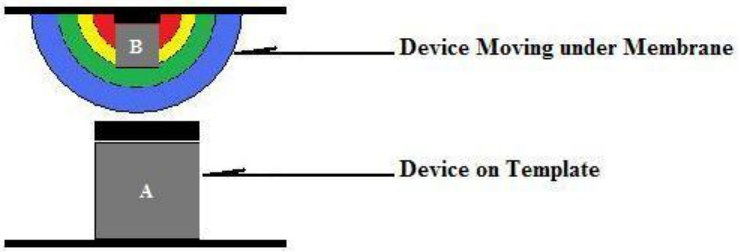

Figure 3.3 A diagram showing two devices, A and B; A is on the template while B is on the membrane. The concentric colored half-circles around device B are a representation of the magnitude of the fields; red is the strongest region, blue is the weakest region. To conform to the third rule of assembly, the fields manipulating device B must be kept a certain minimum lateral distance apart from device A.

3.2.3 Third Rule

Fields, suspending and moving a device, cannot interact with devices already populated at the template (see Figure 3.3). Following this rule prevents strong frustration. If a device is misaligned on the template, then it will be misaligned in the recess as they reflect each other. Collisions, whether they are physical or magnetic, represent an encroachment of chaos that MFDSA seeks to eliminate. The CCA is applied vertically as well as horizontally with respect to the plane of the device.

3.3 Magnetic Field Interaction Modeling

The task of the model is to calculate the interactions between the array of electromagnets and the soft magnetic layer of the device. The field, induced within the layer by the array, is used to calculate energy and force. The mathematics employed are numerical integration via the Composite Simpson's Rule and finite element analysis.

The array of electromagnets are modeled as a two dimensional lattice (any regular, repeating arrangement) with a constant period length termed the Basic Size Unit (BSU); see Figure 3.4. The elements of the array occupy the positive z region of space above the xy-plane of the membrane; the ends of the elements, which are considered to be air core solenoids, adjoin the xy-plane. The soft magnetic layer, and the device attached onto it, occupy the negative z region of space below the xy-plane of the membrane; the surface of the device adjoins the xy-plane. The membrane is the xy-plane and serves only to restrict the z component of motion as the device is suspended and moved. (See Figure 3.5.) The membrane is considered to be thin and inflexible (idealization) and does not contribute to magnetism in either the elements or the devices.

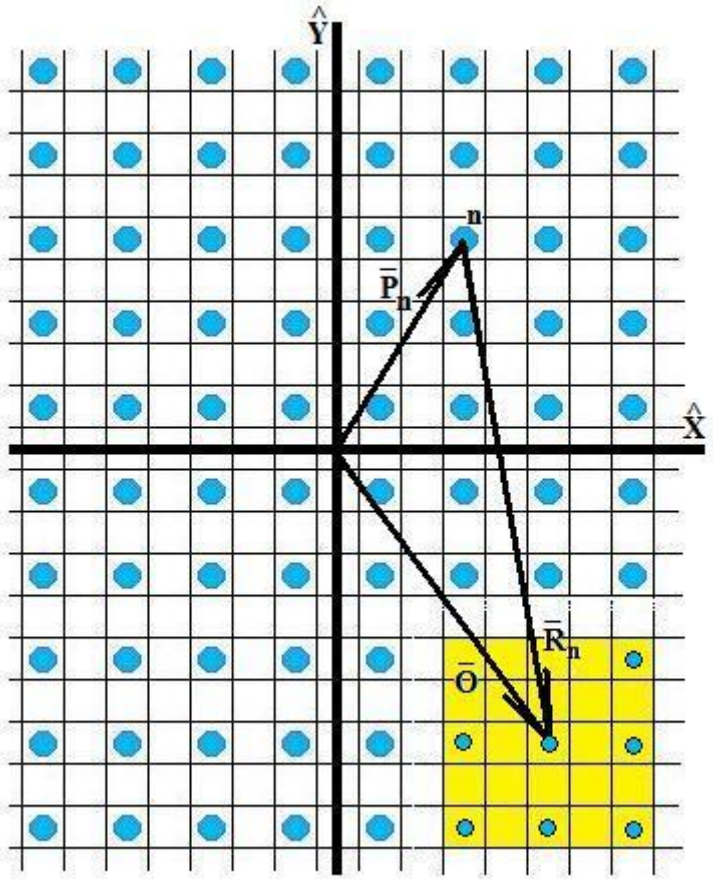

Figure 3.4 A top-down view of the array of electromagnets, the membrane, and the device. The array is depicted by a square lattice of circles (blue) with constant period length. (The circles are the elements and reflect their relative size.) The device is represented by a square (yellow). $\mathbf{P}_n$ is the vector from the origin to the n^{th} element of the array. **O** is the vector from the origin to the center of the layer at the top of the device. $\mathbf{R}_n = \mathbf{O} - \mathbf{P}_n$.

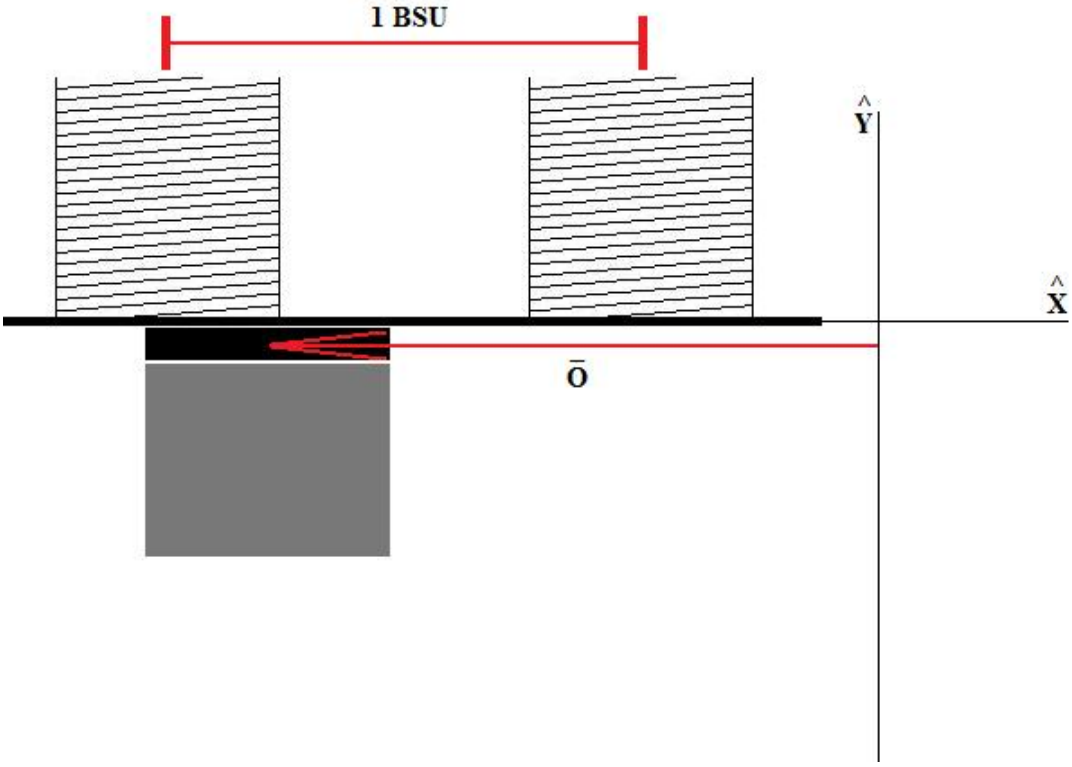

Figure 3.5 A side view of the array of electromagnets, the membrane, and the device. The elements of the array are solenoids that occupy the positive z region of space and are separated by a constant period length, 1 BSU, from center to center. The device occupies the negative z region of space and; **O** is the vector from the origin to the center of the layer (black) at the top of the device (gray).

3.3.1 The Array

The model is designed to admit cases where the array is uniform, where each and every element is unique with respect to its individual physical properties, and any mixture of those two extremes. The model only calculates field, energy, and force generated at points beyond the location of the elements; therefore, to achieve simplicity without loss of generality, they will be air core solenoids. The properties required to define a single element of the array are: $\mathbf{R_n}$, the position vector of the element (pointing from the origin to the center of its circular cross-section area), L, the length, N, the number of turns, I the current applied, I_{max}, the maximum value of current the element allows, and, a, the radius.

3.3.2 The Device

The model considers the soft magnetic layer to be rectangular. The size of the layer is independent of the size of the array; further, it is free to be located anywhere at the negative z region of the xy-plane, even at locations beyond the array. The parameters that define the layer are: **O**, the vector from the origin to the center of the top of the layer, x_{max}, y_{max}, and z_{max}, which define the extent of the layer, N, the number of segments that x, y, and z will be partitioned into, and χ_m, the susceptibility of the layer; additionally, the friction and weight of the device are required.

3.3.3 Other Parameters

The device's soft magnetic layer volume is divided into N^3 cubes of differential volume Δx, Δy, and Δz, where:

$$\Delta x = \frac{x_{max}}{N} \tag{3.1}$$

$$\Delta y = \frac{y_{max}}{N} \tag{3.2}$$

$$\Delta z = \frac{z_{max}}{N} \tag{3.3}$$

Each individual cube is parameterized by the *ijk* indexes: *i* for x-space, *j* for y-space, *k* for z-space; see Figures 3.6-7. The model employs the center of the cube when calculating field, energy, and force at that cube. The *ijk* indexes can be converted to x_i, y_j, and z_k coordinates via:

$$x_i = \left(x_0 - \frac{x_{max}}{2}\right) + \Delta x\left(i + \frac{1}{2}\right) \tag{3.4}$$

$$y_j = \left(y_0 - \frac{y_{max}}{2}\right) + \Delta y\left(j + \frac{1}{2}\right) \tag{3.5}$$

$$z_k = -z_{max} + \Delta z\left(k + \frac{1}{2}\right) \tag{3.6}$$

where, x_0 and y_0 (and $z_0 = 0$) represent the components of the **O** vector. It is seen that x ranges from $x_0 - x_{max}$ to $x_0 + x_{max}$, y ranges from $y_0 - y_{max}$ to $y_0 + y_{max}$, and z ranges from 0 to $-z_{max}$. The uniqueness of the z range reflects the fact that the device abuts the negative side of the xy-pane and extends into the negative z region of space.

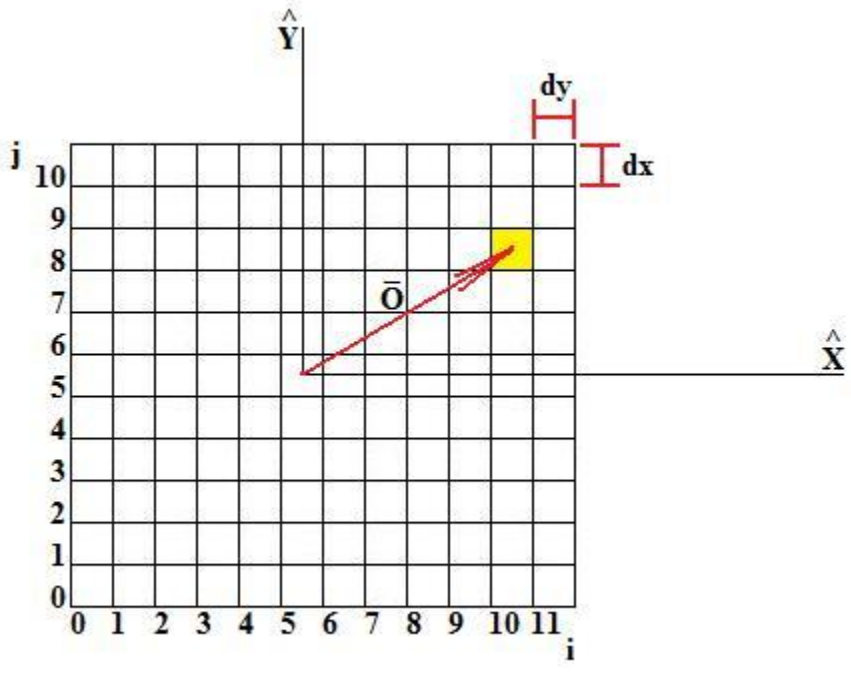

Figure 3.6 A top-down view of a layer at the xy-plane, divided into 12 segments in x and 11 segments in y. The dx and dy indicate the width of the segments. The yellow square represents the 11,8,0 (at the xy-plane k = 0) differential cube. **O** is a vector from the origin to the center of the 11,8,0 differential cube.

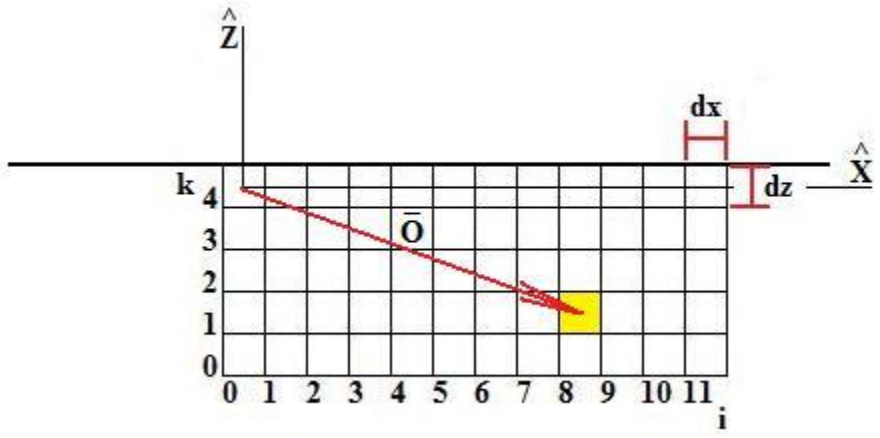

Figure 3.7 A side view of a layer at the xz-plane, divided into 12 segments in x and 5 segments in z. The dx and dz indicate the width of the segments. The yellow square represents the 8,0,1 (at the xz-plane j = 0) differential cube. **O** is a vector from the origin to the center of the 8,0,1 differential cube.

3.3.4 The Basic Size Unit

The BSU is relevant only when the array is a regular, periodic lattice, although it may be defined for a non-regular array. The BSU is the period length of the array, which is defined as the center-to-center distance between elements (see Figure 3.5). Alternatively, if the array is irregular, i.e., the elements of the array are arranged in a non-repeating pattern, the square of the BSU is the smallest possible constant value of area that encompasses all of the elements without overlaps. If the array is packed, i.e., the separation between elements of the array tends to zero, then the BSU approaches the threshold theoretical limit of 2a, where a is the radius of the element.

The BSU, which must be given to the magnetic field interaction model as a parameter, is used to calculate the size of the CCA. It is also the direct physical link between the SA's abstract view of space and the real physical space.

3.4 Main Calculators

The Biot-Savart law is used to calculate the field of the array at the layer. As depicted by Figures 3.8 and 3.9, the elements are solenoids of length L, turns N, and radius a, located at the positive z region of space. The field is evaluated at the point of interest x_i, y_j, and z_k, at the center of the *ijk* differential cube. The field's x, y, and z components are solved numerically by the Composite Simpson's Rule applied to a two variable system; the variables are η and φ, where η represents the turns and ranges from 0 to N and φ represents the angle and ranges from 0 to 2π.

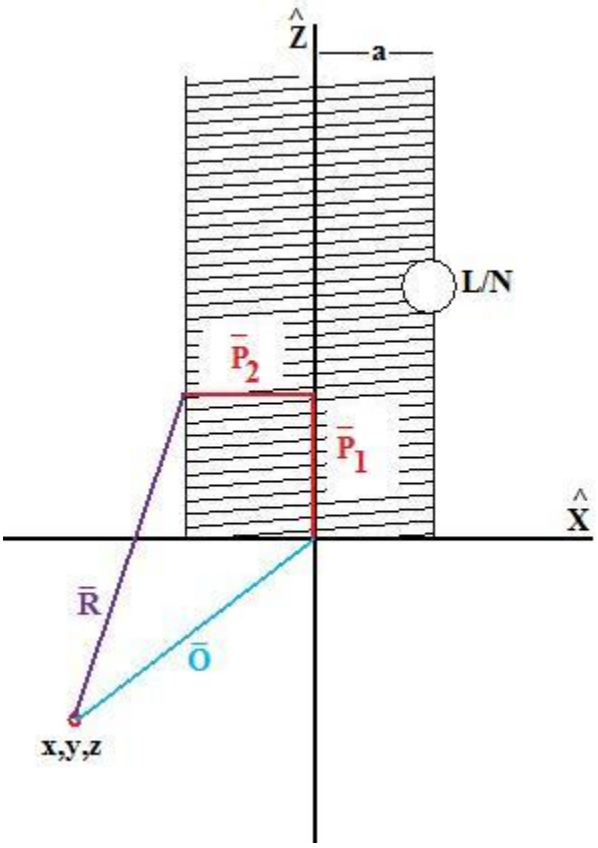

Figure 3.8 The setup of the Biot-Savart law for the solenoid field at point x,y,z. The solenoid occupies the positive z region of space. The point x,y,z is at the negative z regions of space. a is the radius of the solenoid. L/N is the density of length per turn. $\mathbf{P}_1 = (L/N)\eta\mathbf{z}$, $\mathbf{P}_2 = a\cos(\varphi)\mathbf{x} + a\sin(\varphi)\mathbf{y}$, $\mathbf{O} = x\mathbf{x} + y\mathbf{y} + z\mathbf{z}$, and $\mathbf{R} = \mathbf{O} - \mathbf{P}_1 - \mathbf{P}_2$.

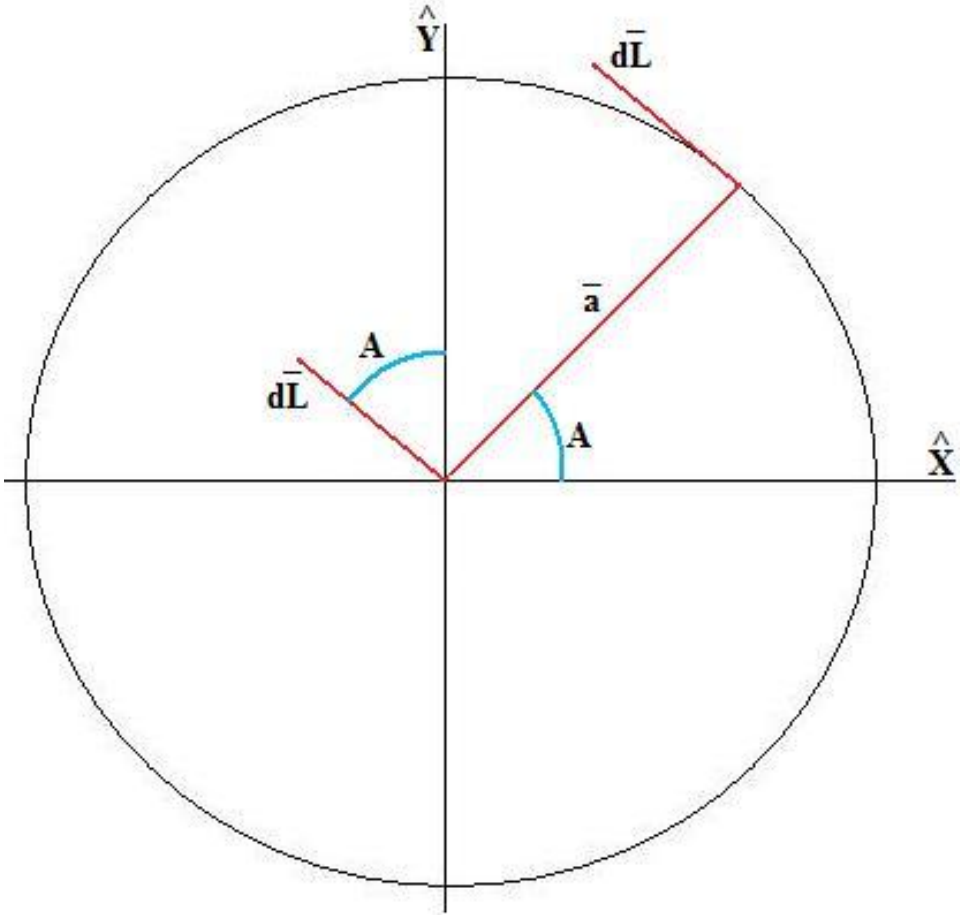

Figure 3.9 An illustration of the d**L** vector. The current flows through the turn along the length of the solenoid. The radius is a, the angle is A, and dA is the differential change in angle. d**L** = -a sin(A)dA**x** + a cos(A)dA**y**

3.4.1 Biot-Savart Law

The Biot-Savart Law, specialized for the solenoid, is [101]:

$$\bar{\boldsymbol{B}} = \frac{\mu}{4\pi} I \int \int \frac{d\bar{\boldsymbol{L}} \times \bar{\boldsymbol{R}}}{|\bar{\boldsymbol{R}}|^3} d\eta \tag{3.7}$$

where, μ is the permeability at the point of interest, I is the current in the element, **R** is the distance vector between the element and the point of interest (see Figure 3.8) and d**L** is a segment of arc (see Figure 3.9).

The x, y, and z components of the field are [101]:

$$B_x = \frac{\mu}{4\pi} Ia \int \int \frac{(z - \Delta\eta)cos\phi}{\left(x^2 + y^2 + z^2 + a^2 + (\Delta\eta)^2 - 2(axcos\phi + aysin\phi + \Delta\eta z)\right)^{\frac{3}{2}}} d\phi d\eta \tag{3.8}$$

$$B_y = \frac{\mu}{4\pi} Ia \int \int \frac{(z - \Delta\eta)sin\phi}{\left(x^2 + y^2 + z^2 + a^2 + (\Delta\eta)^2 - 2(axcos\phi + aysin\phi + \Delta\eta z)\right)^{\frac{3}{2}}} d\phi d\eta \tag{3.9}$$

$$B_z = \frac{\mu}{4\pi} Ia \int \int \frac{a - xcos\phi - ysin\phi}{\left(x^2 + y^2 + z^2 + a^2 + (\Delta\eta)^2 - 2(axcos\phi + aysin\phi + \Delta\eta z)\right)^{\frac{3}{2}}} d\phi d\eta \tag{3.10}$$

where, x, y, and z are the components of **R**, a is the radius of the element, and Δ is the ratio of length, L, per turn, N. The limits of integration for φ is 0 to 2π and for η is 0 to N.

Let f be a function of variables A and B, where A ranges from 0 to A_{max} and B ranges from 0 to B_{max}. The step-sizes are h_A and h_B, with the indexes i and j, ranging from 0 to n_A and 0 to n_B; therefore, $A_i = 0 + h_A\, i$ and $B_j = 0 + h_B\, j$.

The Simpson's Rule approach is applied to f on A [102]:

$$F(B) = \frac{h_A}{3}\left(f(0,B) + 2\sum_{i=1}^{\frac{n_A}{2}-1} f(A_{2i},B) + 4\sum_{i=1}^{\frac{n_A}{2}} f(A_{2i-1},B) + f(A_{max},B) \right) \tag{3.11}$$

then applied to F_0 on B:

$$G = \frac{h_B}{3}\left(F(0) + 2\sum_{j=1}^{\frac{n_B}{2}-1} F(B_{2j}) + 4\sum_{j=1}^{\frac{n_B}{2}} F(B_{2j-1}) + F(B_{max}) \right) \tag{3.12}$$

thus:

$$\int_0^{A_{max}} \int_0^{B_{max}} f(A,B)\,dA\,dB \approx G \tag{3.13}$$

To find the field within the region occupied by the soft magnetic layer is a two-step process. First, the layer is divided into N^3 differential cubes indexed by *ijk*. Second, the field contributions from each and every element of the array is calculated at each and every *ijk* differential cube.

3.4.2 Induced Magnetic Field

The magnetization, **m**, of the soft magnetic layer due to the field is given by [101]:

$$\bar{\boldsymbol{m}} = \frac{1}{\mu_0}\frac{\chi_m}{1+\chi_m} v\bar{\boldsymbol{B}} \tag{3.14}$$

where, χ_m is the susceptibility of the layer and v is the volume within which the field acts. This expression is parameterized for the *ijk* differential cube as:

$$\bar{\boldsymbol{m}}_{ijk} = \frac{1}{\mu_0}\frac{\chi_m}{1+\chi_m} \Delta x \Delta y \Delta z \bar{\boldsymbol{B}}_{ijk} \tag{3.15}$$

3.4.3 Energy

The energy of the field acting on the layer is given by [101]:

$$U = -\bar{\boldsymbol{m}} \cdot \bar{\boldsymbol{B}} \tag{3.16}$$

or, parameterized for the *ijk* differential cube:

$$U_{ijk} = -\frac{1}{\mu_0}\frac{\chi_m}{1+\chi_m} \Delta x \Delta y \Delta z \left|\bar{\boldsymbol{B}}_{ijk}\right|^2 \tag{3.17}$$

3.4.4 Forces

The force of the field acting on the layer is given by [101]:

$$\overline{\boldsymbol{F}} = \nabla(\overline{\boldsymbol{m}} \cdot \overline{\boldsymbol{B}}) \tag{3.18}$$

or, parameterized for the *ijk* differential cube:

$$\overline{\boldsymbol{F}}_{ijk} = \frac{1}{\mu_0}\frac{\chi_m}{1+\chi_m}\Delta x \Delta y \Delta z \nabla \left|\overline{\boldsymbol{B}}_{ijk}\right|^2 \tag{3.19}$$

3.4.5 Friction

Device weight and friction need to be considered as they affect the acceleration, therefore, the motion. The z-component of the force exerted at the layer by the array must be such that $|\mathbf{F}_z| > |\mathbf{W}|$, where **W** is the weight; otherwise, the device falls off of the membrane. Also, for the device to move in either the x or y direction, $|\mathbf{F}_x| > |\mathbf{F}_z|\mu$ or $|\mathbf{F}_y| > |\mathbf{F}_z|\mu$ respectively, where μ is the coefficient of friction.

Let **F** be the applied force at the layer by the array and $|\mathbf{F}_z|\mu$ be the force of friction; acceleration is:

$$a = \frac{g}{W}(F \pm |F_z|\mu) \tag{3.20}$$

3.5 Collision Cross-Section Area Calculator

The first phase is to determine the least amount of field required to suspend a weight of **W** η, where **W** is the weight of the soft magnetic layer and η is a factor related to the effect of friction ($\eta \geq 1$) and is a parameter of the calculation. The second phase is to seek the points where the magnitude of the field is B_{max} σ, where B_{max} is the maximum value of the field within the soft magnetic layer and σ is a parameter of the calculation ($0 \leq \sigma \leq 1$). The distance from the center of the soft magnetic layer to these points is used to calculate the CCA.

First, the positions and the constructions of the soft magnetic layer and the array are altered respectively. The center of the layer is placed at the origin. The array is built symmetrically about the origin with the zeroth element at the origin above the center of the soft magnetic layer. The array's period length is still the BSU.

Second, the parameter Δ, which controls the number of elements that are activated, is set to zero. All of the elements about the origin whose distance from the origin is equal to or lesser than Δ·BSU are ramped up to I_{max} α where I_{max} is the element's maximum current tolerance and α is a parameter of the calculation set between 0 and 1. The field within the soft magnetic layer is calculated; then the force is calculated. If $|\mathbf{F}_z| < |\mathbf{W}|$ η, then, Δ is incremented by 1 and the next ring of elements are activated. The loop stops as soon as $|\mathbf{F}_z| > |\mathbf{W}|$ η.

Third, the currents applied to the elements are reduced by a factor of 1/α, until the least amount of field required to suspend **W** η is achieved. This step is important for two reasons: first, it reduces the CCA size to a reasonable and manageable value and, second, by reducing the field and the force it exerts, it also reduces unwanted contributions due to friction (which is due to the normal force $|\mathbf{F}_z|\mu$) and prevents the elements from employing currents that are too strong for too long.

Fourth, the maximum magnitude of the field is found within the soft magnetic layer. The field in the air around the device on the xy-plane is sampled about the x and y axis. Starting at the edge of the layer, the process samples the field and stops when it finds the point on the x and y axis where the magnitude of the field is equal to or lesser than B_{max} σ. The largest value of length, either from the x or y axis, is chosen; it is converted to BSU and rounded to the next odd integer; it is the delta value (delta is not Δ, delta has the dimensions of length, Δ is related to the number of activated elements) through which the CCA is calculated as (see Figure 3.10):

$$CCA = 2\,delta + 1 \tag{3.21}$$

While the CCA is the actual size, in BSU, of the area reserved for the device, delta is the parameter that is given to SA. The CCA is always overestimated by 1 BSU to allow the fine turning of position that may be required to accurately place a device on the template (and in the substrate).

The effect of hysteresis is not considered by this algorithm; it is a static model and does not consider dynamic or second order effects. The CCA as considered by the algorithm admits both static and kinetic friction; however, as static friction is stronger than kinetic friction, it is advisable to use static friction and not kinetic friction when formulating the value of η.

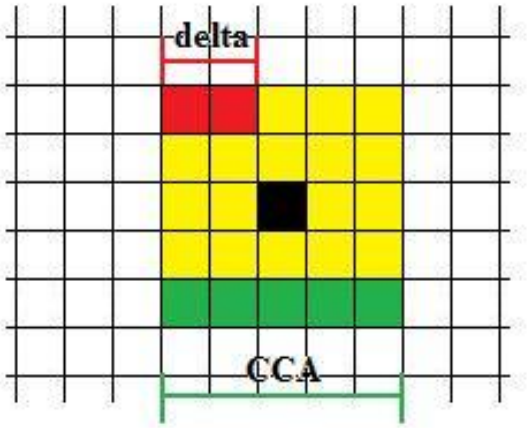

Figure 3.10 An illustration depicting the difference between the delta parameter and the CCA value. The CCA is the length of the square around the device. The delta is the distance between the center and the edge. (The scale of the grid is 1 BSU). The relationship between CCA and delta is CCA = 2 delta + 1.

3.6 Magnetic Retention Factor

Because $|\mathbf{F}_z|$ must be equal to or greater than the weight, it follows that the force of friction $|\mathbf{F}_z|\mu$ is also proportional to the weight. A way to minimize $|\mathbf{F}_z|$, and by extension, friction, and to reduce the size of the CCA (which is also proportional to weight) is to minimize the mass of the device. MFDSA is a passive tool to integrate devices into recesses; it cannot mandate how device/recess pairs are to be designed. The only avenue left to optimize the system is to reduce the soft magnetic layer's contribution to the weight.

The concept called the Magnetic Retention Factor (MRF) is proposed as a method to characterize the material used to fabricate the soft magnetic layer in order to chose the material that maximizes the magnetization while reducing its contribution to weight.

The addition of hard and soft magnetic material is a feature of MFDSA that is necessary to achieve assembly, yet thereafter, it represents a permanent passive component retained within the system. The goal is to minimize the impact of those materials. Aside from increasing friction, the layer increases the total weight of the system, especially the device itself. The heavier the device, the stronger the applied magnetic field needs to be both when the array suspends and moves it and when the recess retains it. Lighter devices reduce the effect of friction, the CCA, and require weaker or fewer amounts of hard magnetic materials.

A concise and abstract consideration of the MRF follows. The test electromagnet is a solenoid with an air core of radius r, length L, and turns N; a current of I flows through it. The test soft magnetic layer, at the top of the device, is a cylinder with a radius r, thickness t, volume v, and susceptibility χ_m. The solenoid and the device are on-axis without separation.

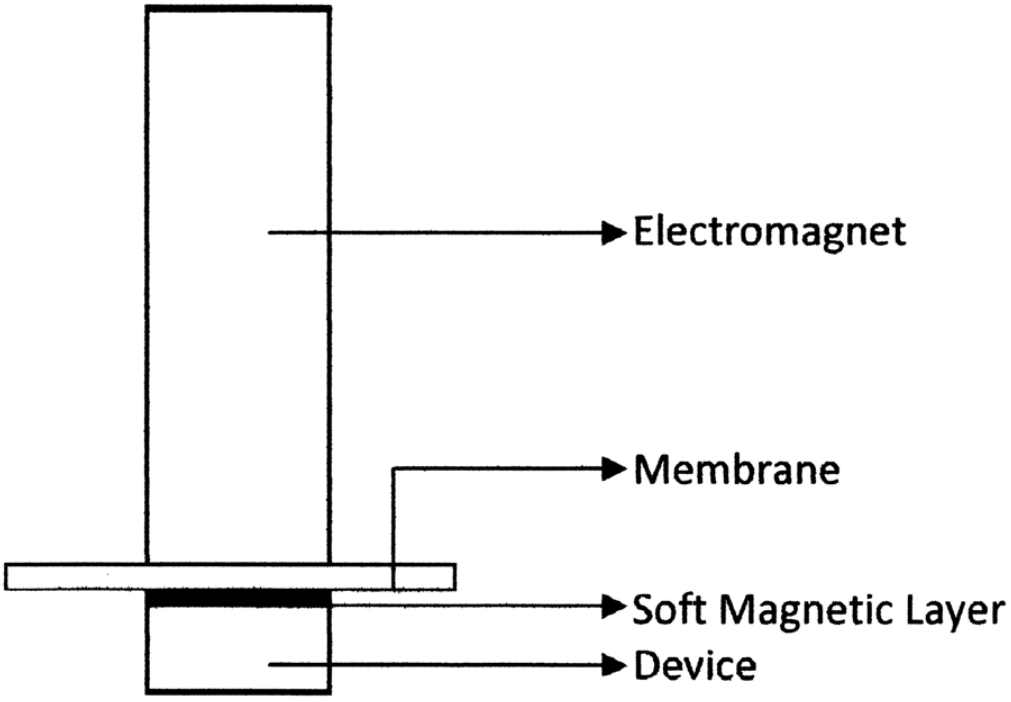

Figure 3.11 A schematic demonstrating the test model of interest. A device (with a layer of soft magnetic material) is suspended against gravity by the core of the electromagnet. A membrane of negligible thickness separates device and core. The device and the core are shown with equal radius.

Basic electromagnetic theory gives an approximate expression for force given at a situation as found in Figure 3.11 as [101]:

$$F = 2\frac{1}{\mu_0}\frac{\chi_m}{1+\chi_m}\frac{v}{r}B^2 \tag{3.22}$$

where, B is the field induced in the soft magnetic layer; secondary interactions between the solenoid and the device are ignored.

The force must be equal to the weight, W, of the device at the very least. The weight itself is split into two components: W_d is the weight of the bare device without a layer and W_l is the weight of the layer alone.

$$B = \sqrt{(W_d + W_l)\frac{1}{2}\frac{r}{v}\mu_0\frac{1+\chi_m}{\chi_m}} \tag{3.23}$$

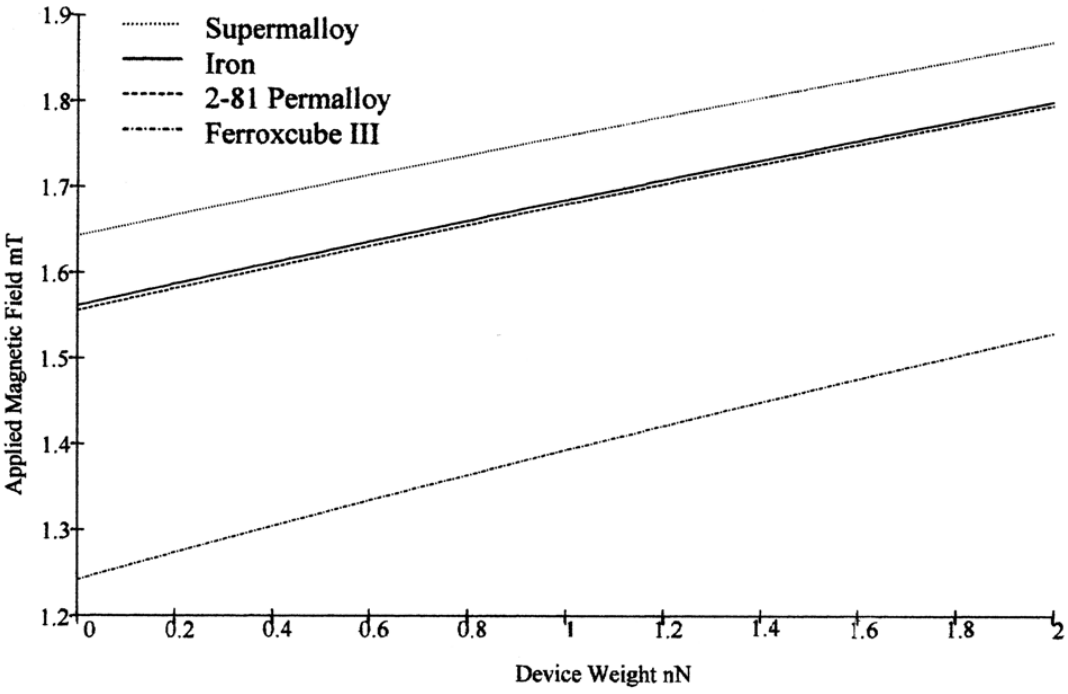

Figure 3.12 Applied Magnetic Field vs. Device Weight (a graph of Equation. 3.23) showing its application to four types of materials (Ferroxcube III, 2-81 Permalloy, iron, and Supermalloy) modeled as layers 10μm thick with a radius of 50μm. The device weight varies from 0 to 2nN. The calculated applied magnetic fields range from 1.2 to 1.9mT.

Figure 3.12 summarizes the results of Equation 3.23 for a soft magnetic layer with thickness 10μm and radius 50μm for each of the following materials: Ferroxcube III, 2-81 Permalloy, iron, and Supermalloy. Note that the abscissa is only the bare device weight, W_d, and when it is equal to zero, there is still a W_l which requires an induced magnetic field to suspend. Given the parameters of Table 3.1, Ferroxcube III is the material that requires the least magnetic field to support the total weight of the device.

The MRF is introduced as a way to determine those materials that generate the greatest magnetization while contributing the least weight to the system. MRF is defined as:

$$MRF = \frac{1}{\rho}\frac{1}{\mu_0}\frac{\chi_m}{1+\chi_m} \tag{3.24}$$

where, ρ is the material's mass density. Since W_l = ρvg, Equation 3.24 follows from Equation 3.23 where the soft magnetic layer volume v and radius r are fixed. MRF is simply a function of material properties; for the materials considered in Table 3.1 the susceptibly, χ_m, is larger than unity; therefore, MRF is dependent on mass density.

Table 3.1 gives values of MRF of several magnetic materials. It further demonstrates what is shown in Figure 3.12 and confirms the interpretation given for MRF. The Ferroxcube III has the largest value of MRF while iron yields the smallest value of MRF.

A further connection can be made between the permeability of the soft magnetic layer and the thickness required to adhere to the hard magnetic strips as found in Equations 2.13-14. A relationship developed between MRF and b/L, the scaled soft magnetic layer thickness, is:

$$\frac{b}{L} > \frac{1}{2\pi} ln\left(\frac{2}{\rho\mu_0 MRF} - 1\right) \tag{3.25}$$

Table 3.1 A Table of Magnetic Materials and Properties.

Material	**μ_r** at 20Gs	**ρ** kg/m^3	**MRF** $m^2/ s^2/T^2$	**b/L**	**T_C** °C
Ferroxcube III	1000	5000	159.0	0.0005	~135
2-81 Permalloy	125	7800	101.2	0.0027	460
Iron	200	7880	100.5	0.0017	622
Supermalloy	100000	8770	90.7	0.0003	400

Source: Adapted from [103-107].

Note: the Curie Temperature (T_C) imposes a limit with respect to further thermal processing unless the wafer is secured with a protective layer or film; beyond T_C, the materials are not magnetic and the integration could be destroyed.

3.7 Outline of the Swarm Algorithm

The Swarm Algorithm (SA) represents the other half of the MFDSA modeling program. SA is an iterative solution to the problem of assembly. It calculates the pathways devices need to follow as they travel from initial to final positions. SA is invoked prior to assembly; its output is the complete process reduced to step-by-step, 'quantized' movements.

The output is fed into the external control unit, along with such data as device mass, friction, size, and delta, which is used to convert the abstract solution into a real-time solution. The external control unit translates the movements into instructions that operate the array and that manipulate the field accordingly thereby moving devices. The actual methodology of translation is a problem with respect to engineering that further research and development is intended to address.

It is important to reiterate that SA is an abstraction of the problem.

3.7.1 Space and Time Abstractions

SA attempts to solve the simultaneous hybrid assembly process by imposing a sequential approach to calculate movement. The apparent sizes of devices is in fact the CCA, and, the grid-size is equal to 1 BSU. The 'space', as in distances from grid point to grid point, is connected to a true distance via the BSU.

The most important abstraction, however, cannot be easily reconciled and that is SA's notion of time. The analog of time is the step (STP), which is not a unique temporal duration. There does not exist a factor to convert steps into seconds like the BSU converts grid-size into meters. To SA, the step is an interval of iteration within which a set of movement occurs simultaneously. Although SA only moves devices 1 BSU length per step, devices of various inertias and frictions require different accelerations to displace that 1 BSU length. Ultimately, they require different times to displace their 1 BSU movement. It is possible to convert steps into seconds only on a step-per-step basis; it is found by calculating the time required for the slowest device with the weakest acceleration to move a distance of 1 BSU.

At a step, all of the movement occurs simultaneously; however, the fastest devices and the slowest devices complete their 1 BSU movements at different durations. The fastest devices actually spend a fraction of a step at rest delayed until the slowest devices complete their motion. The external control unit is required to wait until all devices complete their motion in order to advance. The effect is that the simultaneity of the assembly is not achieved by continuous motion but by grainy, quantized motion; it must be accomplished that way to keep control of the process and prevent occurrences of collision and frustration events.

3.7.2 Final Position Offsets

Another layer of abstraction includes the exact position of devices. The important rule of thumb is that the actual geometric center of a device is kept at the center of the CCA. When SA prescribes to move a device from grid to grid, it shifts the location of the CCA and the device by 1 BSU; before and after the motion, the centers of the device and the CCA coincide. To verify the positions of devices to ensure that they indeed always move by the 1 BSU distance, sensors are employed on the membrane to measure pressure and other effects which reveal the location of a device; algorithms within the external control unit would be able to follow the devices by keeping track of their sensors.

Related to the step-by-step motions of devices is the issue of the final position. The final position on the template represents the final position in the substrate. It may appear that the centers of the grids need to correspond with or align to the recess, which would be a subtle way that MFDSA constraints the design of wafers.

This difficulty is eliminated with the inclusion of offsets. Offsets are a way to fine-tune, if necessary, the final positions of devices. Offsets are fractional values of BSU and may be either positive or negative denoting the direction for the position to be corrected prior to dropping the device onto the template (see Figure 3.13). The actual offset value that can be reached is limited by the dexterity of the array; however, it simply needs to be within the allowable placement tolerance and is subject to the error checking/correction algorithm of MFDSA. When a device is brought into its final position, an additional set of motion in invoked, controlled by the offset values, which change the final position to its true desired value. The CCA is deliberately over-estimated to allow for these offset corrections without violating the rules of assembly.

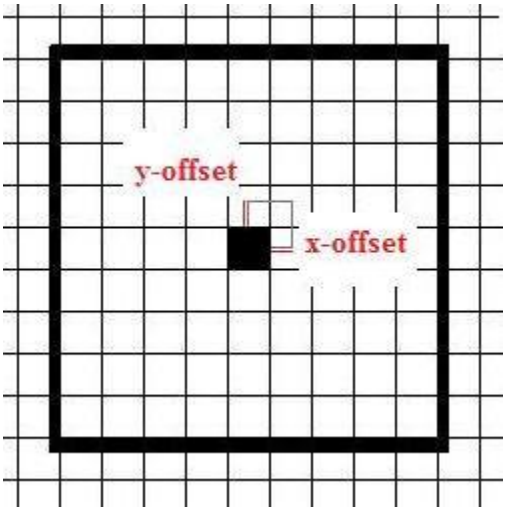

Figure 3.13 An illustration of the final position offsets. The CCA is outlined by the large square. The device is given by the small square. When the device reaches its final position it may be necessary to fine-tune that location by fractional BSU motions in either the x or y directions. The offsets allow the device to be placed at a location that corresponds to the recess.

3.7.3 Boundary Conditions, Driving Functions, and Other Parameters

The order of devices as they enter the enclosure functions in a way analogous to a driving force. The initial and final positions of devices are equivalent to the boundary conditions. The number of devices that actually enter from step to step is controlled by a pair of parameters: the maximum device and the trigger device values. The maximum device value controls the maximum number of devices that are moved on the grid at any given step. The trigger device value controls when an insertion is invoked; essentially, after a certain number of devices have been populated on top of the template, the insertion will be triggered. When the number of active devices left on the grid plus the populated devices on the template is equal to the trigger value, then no new devices are added to the enclosure until insertion is triggered.

3.7.4 The P and Q Tables

SA determines how to move devices simultaneously by constructing the pathways iteratively from step to step. The active devices on the grid are placed in a cue and are processed, sequentially, from first to last as prescribed by order of entry. For each device at a step, SA generates the P and Q Tables. These tables contain answers to logical yes/no questions out of which motion will be decided. The P-Table is first. The Q-Table is second if and only if the P-Table fails to yield a movement.

The P-Table is concerned with the long range motion and avoids frustration by choosing pathways that are not blocked. The Q-Table is mostly a short-sided view and circumvents frustration by avoiding motions into areas with high device density.

The pathways created by SA are composed of five discrete or 'quantized' 1 BSU movements whose units are: up, down (along the horizontal plane's y-direction), left, right (along the horizontal plane's x-direction), and null (diagonal movement is not allowed). Null is a special kind of movement indicating no movement; it is a safety valve designed to keep the method from failing. If all devices yield null motion, then the method is said to fail. As with an ordinary differential equation, the problem of assembly may be ill-posed and SA will not converge to solution. Typically, this occurs if there are more larger devices than smaller devices present at a given step or if devices are attempting to move through or around crowded areas to reach their destinations. These failures can be resolved by changing maximum and trigger device values and altering the order of device entry.

3.8 Discussion

In conjunction with the problem of assembly tackled by SA are a class of problems called 'parking lot' problems. The literature that exists regarding the parking lot problem is mainly concerned with optimizing the distribution of spaces given a parking lot of fixed shape and area [108]. Further, no formulation of the parking lot problem, or the solution to that problem, attempts to describe how exactly the cars move to obtain their spot [109-110]. The details of the placement of cars into spaces to fill a parking lot (the pathways) is ignored. Such details, if discussed, are left as random or based on the first-come, first-served paradigm. This is justified because, in general, all cars and all spaces are interchangeable [109].

The MFDSA is a tool for assembly. It does not impose any conditions regarding how devices and recess are distributed about a wafer; the pattern of the distribution is arbitrary. The initial and final positions are fixed boundary conditions that must be specified and not random. MFDSA is not a first-come, first-served method where devices are scattered onto the nearest recesses that will accept them. The MFDSA problem is about the minutiae of assembly itself. It is concerned with the choices of motion the devices are required to follow to attain their final desired positions.

MFDSA is also not analogous to the problem of obtaining travel directions with a global positioning system (GPS) navigator.

When a GPS determines a route, it does not take into account the way that cars need to maneuver about immediate real-time traffic or how to deal with other roadway obstacles that suddenly appear. Instead it compiles a series of 'turn here, turn there' directions which may be somewhat analogous to the output of SA but there are two areas of difference.

First, there are fixed immovable roadways with fixed directional flows; there are no fixed pathways in SA; the entire grid is open to motion in all directions. Second, there are no reverses in GPS navigation, the car is always on drive, moving forward and forward, even as it makes left or right turns; in SA, devices can go backward if required. The GPS is very much a reduced 1D problem whereas the MFDSA solves the full 2D problem.

CHAPTER 4

MAGNETIC FIELD INTERACTION MODEL

4.1 Overview

A numerical, magnetostatic approach is developed to model the magnetic field interaction between the array of electromagnets and the soft magnetic layer. The model contains functions that calculate magnetic field, energy, force field, and net force. A combination of array and layer definitions are required to complete the calculations.

The model requires two major inputs, the definitions of the array of electromagnets and the soft magnetic layer. It does not impose restrictions with respect to size and location of the array and the layer. The only constraints are that the array is above the xy-plane and the layer is below the xy-plane. The feedback between the elements of the array is ignored as that effect is concerned with torques and repulsive/attractive forces within the array that do not act on the layer. The material of the layer is assumed to be linear.

The xy-plane at z = 0 represents the location of the membrane, which separates the array of electromagnets and the soft magnetic layer. While the actual, physical membrane is finite, the model does not impose any limit with respect to the size of the xy-plane. A further set of simplifications are that its thickness is negligible and its susceptibility is zero.

The model is free to deal with any condition regarding the structure of the array of electromagnets. It may be homogenous, inhomogeneous, and a mixture of those extrema. If the array is homogenous then it is a regular, repeating 2D lattice, with a constant period length called the Basic Size Unit (BSU). If the array is inhomogeneous, then it is more amorphous than crystalline; the square of the BSU is the smallest constant area that encompasses each and every element without overlap (there could be gaps). The BSU of a dense array approaches the limit of D, the diameter of the element. (Note: the model allows non-homogenous array distributions as part of its overall design generality and does not reflect what would be ordinarily employed by the MFDSA technique.)

The array is considered to be an air core solenoid; it can be adjusted to admit a core; however, it is omitted for simplicity without too great a loss to generality. The properties that define the n^{th} element are: $\mathbf{R}_n$, the vector of position pointing from the origin to the center of its bottom circular cross-section area, L, the length, N, the number of turns, I the current supplied, I_{max}, the maximum value of current the element allows, and, a, the radius.

The layer is considered to be a rectangle. The properties that define the layer are: **O**, the vector of position pointing form the origin to the center of its top area, x_{max}, y_{max}, and z_{max}, which specify the size of the layer, N, the partitions it will be broken into along the x, y, and z axis, and χ_m, the susceptibility.

4.2 Parameters

The x_{max}, y_{max}, and z_{max} are divided by the N to obtain the step-sizes:

$$\Delta x = \frac{x_{max}}{N} \tag{4.1}$$

$$\Delta y = \frac{y_{max}}{N} \tag{4.2}$$

$$\Delta z = \frac{z_{max}}{N} \tag{4.3}$$

The Δx, Δy, and Δz define the volume of the differential cube. A differential cube is indexed by *ijk*, where *i*, *j*, and *k* range from 0 to N - 1 such that there will be a total of N^3 cubes within the layer.

O is the vector from the origin to the center of the top of the layer; however, the model requires $\mathbf{O}'_{ijk}$, the position vector from the origin to the center of the ijk differential cube. If $\mathbf{O} = x_0\mathbf{x} + y_0\mathbf{y} + 0\mathbf{z}$, then:

$$O'_{ijk-x} = \left(x_0 - \frac{x_{max}}{2}\right) + \Delta x\left(i + \frac{1}{2}\right) \tag{4.4}$$

$$O'_{ijk-y} = \left(y_0 - \frac{y_{max}}{2}\right) + \Delta y\left(j + \frac{1}{2}\right) \tag{4.5}$$

$$O'_{ijk-z} = -z_{max} + \Delta z\left(k + \frac{1}{2}\right) \tag{4.4}$$

where, the 1/2 factor reflects the fact that the terminus of the $\mathbf{O}'_{ijk}$ vector is the center of the *ijk* differential cube. Any point within the *ijk* differential cube could be chosen as the point of interest; for symmetry and to avoid vertexes and surfaces, the center is chosen. The components of the $\mathbf{O}'_{ijk}$ vector range in x from $x_0 - x_{max} / 2$ to $x_0 + x_{max} / 2$, in y from $y_0 - y_{max} / 2$ to $y_0 + y_{max} / 2$, and in z from 0 to $-z_{max}$.

$\mathbf{R}_{n,ijk}$ is the vector from the center of the n^{th} element to the center of the *ijk* differential cube; it is defined as $\mathbf{R}_{n,ijk} = \mathbf{O}'_{ijk} - \mathbf{R}_n$. If $\mathbf{R}_n = x_n\mathbf{x} + y_n\mathbf{y} + 0\mathbf{z}$, then:

$$R_{n,ijk-x} = \left(x_0 - \frac{x_{max}}{2}\right) + \Delta x\left(i + \frac{1}{2}\right) - x_n \equiv x \tag{4.7}$$

$$R_{n,ijk-y} = \left(y_0 - \frac{y_{max}}{2}\right) + \Delta y\left(j + \frac{1}{2}\right) - y_n \equiv y \tag{4.8}$$

$$R_{n,ijk-z} = -z_{max} + \Delta z\left(k + \frac{1}{2}\right) \equiv z \tag{4.9}$$

$\mathbf{P}_n$ is the vector from the origin to a segment of current within the n^{th} element. It is a combination of two vectors. First, a vertical vector that describes the height with respect to the number of turns. Second, a polar vector that describes a segment of current.

$$\boldsymbol{P}_n = a\cos\varphi\hat{\boldsymbol{x}} + a\sin\varphi\hat{\boldsymbol{y}} + \Delta\eta\hat{\boldsymbol{z}} \tag{4.10}$$

where, φ is from 0 to 2π, Δ is the ratio of length per turn, and η is from 0 to N.

$\mathbf{R}'_{n,ijk}$ is the displacement vector from the segment of current within the n^{th} element to the center of the *ijk* differential cube; it is required to compute the magnetic field vector at the *ijk* differential cube.

$$\boldsymbol{R}'_{n,ijk} = (x - a\cos\varphi)\hat{\boldsymbol{x}} + (y - a\sin\varphi)\hat{\boldsymbol{y}} + (\mathrm{z} - \Delta\eta)\hat{\boldsymbol{z}} \tag{4.11}$$

dL is the vector that defines the arc of current along a loop within the solenoid:

$$d\boldsymbol{L} = -a\sin\varphi d\varphi\hat{\boldsymbol{x}} + a\cos\varphi d\varphi\hat{\boldsymbol{y}} \tag{4.12}$$

4.3 Composite Simpson's Rule

At the center of the magnetic field interaction model is the Biot-Savart law. The field's components, in x, y, and z, are to be calculated with respect to the n^{th} element and the *ijk* differential cube. The net field at an *ijk* differential cube will be found by adding the contributions of the elements onto that *ijk* differential cube. The vector field will be found through the volume of the layer, i.e., a total of N^3 centers of *ijk* differential cubes.

The Biot-Savart law, specialized to fit a solenoid, is [101]:

$$\bar{\boldsymbol{B}} = \frac{\mu}{4\pi} I \int \int \frac{d\bar{\boldsymbol{L}} \times \bar{\boldsymbol{R}}}{|\bar{\boldsymbol{R}}|^3} d\eta \tag{4.13}$$

where, **R** is $\mathbf{R}'_{n,ijk}$ (Equation 4.11) and d**L** is as defined by Equation 4.12.

The components of the field are [101]:

$$B_x = \frac{\mu}{4\pi} Ia \int \int \frac{(z - \Delta\eta)cos\phi}{\left(x^2 + y^2 + z^2 + a^2 + (\Delta\eta)^2 - 2(axcos\phi + aysin\phi + \Delta\eta z)\right)^{\frac{3}{2}}} d\phi d\eta \tag{4.14}$$

$$B_y = \frac{\mu}{4\pi} Ia \int \int \frac{(z - \Delta\eta)sin\phi}{\left(x^2 + y^2 + z^2 + a^2 + (\Delta\eta)^2 - 2(axcos\phi + aysin\phi + \Delta\eta z)\right)^{\frac{3}{2}}} d\phi d\eta \tag{4.15}$$

$$B_z = \frac{\mu}{4\pi} Ia \int \int \frac{a - xcos\phi - ysin\phi}{\left(x^2 + y^2 + z^2 + a^2 + (\Delta\eta)^2 - 2(axcos\phi + aysin\phi + \Delta\eta z)\right)^{\frac{3}{2}}} d\phi d\eta \tag{4.16}$$

The solutions of Equations 4.14-16 are analytical only at a few, specific cases. The model must be able to calculate fields at any given point; therefore, a numerical quadrature scheme must be invoked to proceed further. The Composite Simpson's Rule is developed as that scheme.

Let f be a function of variables A (from 0 to A_{max}) and B (from 0 to B_{max}), with step-sizes h_A and h_B such that $A_i = 0 + h_A\, i$ (i from 0 to n) and $B_j = 0 + h_B\, j$ (j from 0 to m). The integral of f over A and B is [102]:

$$\int\int f(A,B)dAdB = \frac{h_A}{3}\frac{h_B}{3} \tag{4.17}$$

$$\times\Bigg[f(0,0) + f(A_{max},0) + f(0,B_{max}) + f(A_{max},B_{max})$$

$$+ 2\sum_{i=1}^{\frac{n}{2}-1}\big(f(A_{2i},0) + f(A_{2i},B_{max})\big) + 2\sum_{j=1}^{\frac{m}{2}-1}\Big(f\big(0,B_{2j}\big) + f\big(A_{max},B_{2j}\big)\Big)$$

$$+ 4\sum_{i=1}^{\frac{n}{2}}\big(f(A_{2i-1},0) + f(A_{2i-1},B_{max})\big) + 4\sum_{j=1}^{\frac{m}{2}}\Big(f\big(0,B_{2j-1}\big) + f\big(A_{max},B_{2j-1}\big)\Big)$$

$$+ 4\sum_{i=1}^{\frac{n}{2}-1}\sum_{j=1}^{\frac{m}{2}-1} f\big(A_{2i},B_{2j}\big) + 8\sum_{i=1}^{\frac{n}{2}-1}\sum_{j=1}^{\frac{m}{2}} f\big(A_{2i},B_{2j-1}\big) + 8\sum_{i=1}^{\frac{n}{2}}\sum_{j=1}^{\frac{m}{2}-1} f\big(A_{2i-1},B_{2j}\big)$$

$$+ 16\sum_{i=1}^{\frac{n}{2}}\sum_{j=1}^{\frac{m}{2}} f\big(A_{2i-1},B_{2j-1}\big)\Bigg]$$

The functions of Equations 4.14-16 are substituted in place of f.

4.4 Algorithms

4.4.1 Magnetic Field Calculations

Due to the size of the array of electromagnets, and the volume of the soft magnetic layer, the magnetic field calculator is slow to converge computationally. The function evaluates the net magnetic field at the *ijk* differential cube as a result of the entire array. It is general to regions occupied by the layer (where $\chi_m \neq 0$) and the vacuum (where $\chi_m = 0$). The convergence of the calculation is controlled by the SR2 parameter; SR2 determines the number of partitions used by the numerical integration scheme, hence, its tolerance and error.

The algorithm is summarized as:

for i, j, k = 0 to N - 1

find the $\mathbf{O}'_{ijk}$ vector

set the $\mathbf{B}ijk$ vector to zero

for n = 1 to M

find the $\mathbf{R}_{n,ijk}$ vector

calculate the $\mathbf{B}_{n,ijk}$ vector

update $\mathbf{B}_{ijk}$ by $\mathbf{B}_{n,ijk}$

loop n

loop i, j, k

4.4.2 Energy Calculations

The energy between a soft magnetic material and an applied field is given by [101]:

$$U = -\bar{\boldsymbol{m}} \cdot \bar{\boldsymbol{B}} \tag{4.18}$$

where, **m** is the induced magnetization at the layer and **B** is the field.

Assuming that the material is linear and that the field is weaker than stronger, then the magnetization of the layer is a function of **B**:

$$\bar{\boldsymbol{m}} = \frac{\chi_m}{\mu_0(1+\chi_m)} v\bar{\boldsymbol{B}} \tag{4.19}$$

where, v is the volume of the layer.

Substituting Equation 4.19 into Equation 4.18 and expanding **B**:

$$U = -\frac{\chi_m}{\mu_0(1+\chi_m)} v\left(B_x^2 + B_y^2 + B_z^2\right) \tag{4.20}$$

or, parameterized for the *ijk* differential cube:

$$U_{ijk} = -\frac{\chi_m}{\mu_0(1+\chi_m)} \Delta x \Delta y \Delta z\left(B_{ijk-x}^2 + B_{ijk-y}^2 + B_{ijk-z}^2\right) \tag{4.21}$$

The algorithm is summarized as:

for i, j, k = 0 to N - 1

calculate the U_{ijk} from $\mathbf{B}_{ijk}$ and parameters

loop

4.4.3 Force Field Calculations

The force between a soft magnetic material and an applied field is given by [101]:

$$\boldsymbol{F} = \nabla(\bar{\boldsymbol{m}} \cdot \bar{\boldsymbol{B}}) \tag{4.22}$$

where, **m** is the induced magnetization at the layer and **B** is the field.

Substituting Equation 4.19 into 4.22 and expanding **B**:

$$\boldsymbol{F} = \nabla\left(\frac{\chi_m}{\mu_0(1+\chi_m)} v\left(B_x^2 + B_y^2 + B_z^2\right)\right) \tag{4.23}$$

which is expanded into the x, y, and z components as:

$$F_x = \frac{\chi_m}{\mu_0(1+\chi_m)} v \frac{d}{dx}\left(B_x^2 + B_y^2 + B_z^2\right) \tag{4.24}$$

$$F_y = \frac{\chi_m}{\mu_0(1+\chi_m)} v \frac{d}{dy}\left(B_x^2 + B_y^2 + B_z^2\right) \tag{4.25}$$

$$F_z = \frac{\chi_m}{\mu_0(1+\chi_m)} v \frac{d}{dz}\left(B_x^2 + B_y^2 + B_z^2\right) \tag{4.26}$$

or, parameterized for the *ijk* differential cube:

$$F_{ijk-x} = 2\frac{\chi_m}{\mu_0(1+\chi_m)} \Delta x \Delta y \Delta z \left(B_{ijk-x}\frac{d}{dx}B_{ijk-x} + B_{ijk-y}\frac{d}{dx}B_{ijk-y} + B_{ijk-z}\frac{d}{dx}B_{ijk-z}\right) \tag{4.27}$$

$$F_{ijk-y} = 2\frac{\chi_m}{\mu_0(1+\chi_m)} \Delta x \Delta y \Delta z \left(B_{ijk-x}\frac{d}{dy}B_{ijk-x} + B_{ijk-y}\frac{d}{dy}B_{ijk-y} + B_{ijk-z}\frac{d}{dy}B_{ijk-z}\right) \tag{4.28}$$

$$F_{ijk-z} = 2\frac{\chi_m}{\mu_0(1+\chi_m)}\Delta x\Delta y\Delta z\left(B_{ijk-x}\frac{d}{dz}B_{ijk-x} + B_{ijk-y}\frac{d}{dz}B_{ijk-y} + B_{ijk-z}\frac{d}{dz}B_{ijk-z}\right) \quad (4.29)$$

A backward in space scheme is employed to represent the gradient operator:

$$F_{ijk-x} = 2\frac{\chi_m}{\mu_0(1+\chi_m)}\Delta y\Delta z\left(B_{ijk-x}\left(B_{ijk-x} - B_{i-1jk-x}\right) + B_{ijk-y}\left(B_{ijk-y} - B_{i-1jk-y}\right) + B_{ijk-z}\left(B_{ijk-z} - B_{i-1jk-z}\right)\right) \quad (4.30)$$

$$F_{ijk-y} = 2\frac{\chi_m}{\mu_0(1+\chi_m)}\Delta x\Delta z\left(B_{ijk-x}\left(B_{ijk-x} - B_{ij-1k-x}\right) + B_{ijk-y}\left(B_{ijk-y} - B_{ij-1k-y}\right) + B_{ijk-z}\left(B_{ijk-z} - B_{ij-1k-z}\right)\right) \quad (4.31)$$

$$F_{ijk-z} = 2\frac{\chi_m}{\mu_0(1+\chi_m)}\Delta x\Delta y\left(B_{ijk-x}\left(B_{ijk-x} - B_{ijk-1-x}\right) + B_{ijk-y}\left(B_{ijk-y} - B_{ijk-1-y}\right) + B_{ijk-z}\left(B_{ijk-z} - B_{ijk-1-z}\right)\right) \quad (4.32)$$

where, again, the *ijk* range from 0 to N - 1.

With respect to the x and y components of the force, the boundary conditions will be cyclical, i.e., i, j = -1 is i, j = N - 2. With respect to the z component, however, the force is clamped to zero at k = -1. A cyclic type of boundary condition is not imposed at the z component of the force due to the fact that the force is significantly different between the end-points of the k-range. The k = N - 1 is closest to the array and yields the strongest value of field and force while the k = 0 is farthest to the array and yields the weakest value of field and force. Without clamping the force at k = -1 to zero, the calculation returns an artificially strong value of force which skews the net force calculations.

The algorithm for the force at the *ijk* differential cube is summarized as:

for i, j, k = 0 to N - 1

calculate the $\mathbf{F}_{ijk}$ from the $\mathbf{B}_{ijk}$ and parameters

if k = 0 then set $\mathbf{F}_{ijk}$ to zero

loop

The algorithm for the net force is summarized as:

set the F vector to zero

for i, j, k = 0 to N - 1

update **F** by $\mathbf{F}_{ijk}$

loop

4.5 The Collision Cross-Section Area Calculations

The Collision Cross-Section Area (CCA) is calculated by a function of the model. This function requires the weight and friction of the layer. Additionally, it requires the array of electromagnets to be built symmetrically about the origin, with a zeroth element at the origin. The definition of the soft magnetic layer does not need to be altered; rather, the function shifts its position such that its center is at the origin.

The CCA calculation is a sequence that involves computing fields and adjusting forces. The CCA value is the buffer that surrounds the layer, which is implemented in order to enforce the three rules of assembly and to allow the offset final position. Beyond weight and friction, several other parameters specific to the calculation are required: η, the weight and friction factor ($\eta \approx 1 + \mu$, where μ is friction), α, the current ramp up/down factor ($0 < \alpha \leq 1$), and σ, the maximum field tolerance ($0 < \sigma \leq 1$).

The algorithm is divided into three phases:

First Phase, Δ parameter estimate

1. set the array symmetrically about the origin
2. set the layer at the origin
3. set the Δ parameter to zero
4. loop:

    set the elements Δ x BSU from the origin to a current of $I_{max}\ \alpha$

    calculate the $\mathbf{B}_{ijk}$, $\mathbf{F}_{ijk}$, and $\mathbf{F}$ of the layer

    if $|\mathbf{F}_z| < |\mathbf{W}|\ \eta$ then increment the delta parameter by 1

    if $|\mathbf{F}_z| \geq |\mathbf{W}|\ \eta$ then exit loop

Second Phase, delta parameter fine-tune

1. loop:

    reduce the currents of the elements by a factor of α

    calculate the $\mathbf{B}_{ijk}$, $\mathbf{F}_{ijk}$, and $\mathbf{F}$ of the layer

    if $|\mathbf{F}_z| < |\mathbf{W}| \eta$ then exit loop

2. increase the currents of the elements by a factor of α

Third Phase, CCA value

1. scan the $\mathbf{B}_{ijk}$ at the surface of the layer (k = N - 1)

2. find the maximum field magnitude, B_{max}

3. at the positive x edge:

    move outward along the axis

    sample the field magnitude until the field is B_{max} σ (distance is x')

4. at the positive y edge:

    move outward along the axis

    sample the field magnitude until the field is B_{max} σ (distance is y')

5. select the larger of the x', y' distances

6. convert distance to BSU, round to the next largest integer

7. set distance as the value of the delta parameter

8. the CCA value is 2 x delta + 1

The first phase is an estimate of the Δ parameter. The elements are activated with a maximum value of current. The field and force are calculated to ensure that the layer is supported by that configuration. If it is not supported, then another batch of elements are activated. If it is supported, then it moves onto the second phase.

The second phase is a fine-tune of the Δ parameter. The currents of the elements are reduced systematically until the force it generates is just enough to support the layer. The reduction of current leads to the reduction of force and field. The result is two-fold. First, the current will be minimized therefore minimizing the heat generated by the array. Second, the overall delta and CCA values will be shortened into a value that is feasible.

The third phase actually calculates the true delta parameter and CCA value. The field is sampled at locations beyond the edges at the x and y axis until the value of field is below a threshold. The larger of the x and y distances is chosen as the basis of the new delta parameter. The CCA value is calculated out of the delta parameter. The CCA value is always larger by at least 1 BSU to ensure that neither physical nor magnetic collisions occur and to allow the offset final position.

4.6 Friction

The z component of the net force, $\mathbf{F_z}$, is the action that keeps the layer attached to the membrane. It is also the normal force of contact between the layer and the membrane; therefore, the force of friction is:

$$f = |\mathbf{F}_z|\mu \tag{4.33}$$

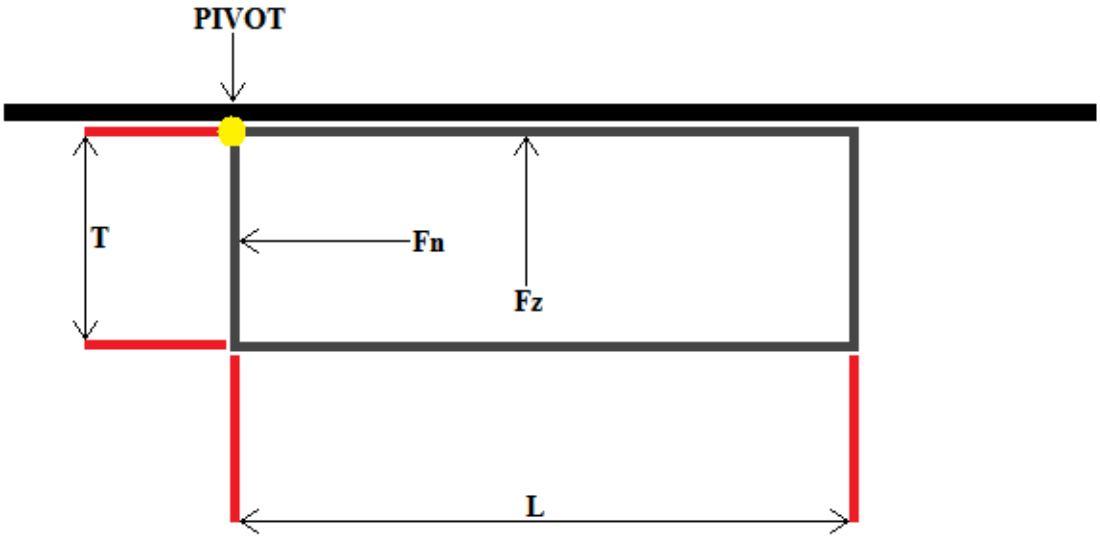

Figure 4.1 A schematic of the forces and torques acting on the layer (gray box). The layer's length is L and thickness is T. $\mathbf{F}_n$ is the force acting to move the layer toward the left. $\mathbf{F}_z$ is the normal contact force between the layer and the membrane (black line). The pivot is the point (yellow circle) at the edge toward the direction of motion. $\mathbf{F}_n$ and $\mathbf{F}_z$ generate torques on the pivot which may or may not flip the layer.

$\mathbf{F}_z$ should be greater than the weight, **W**, to factor the effect of friction. If $\mathbf{F}_z$ is not greater than **W**, then the device could be flipped. A layer intended to move along the direction n encounters friction at the edge closest to the direction n. Let L be the length of the layer, from edge to edge, along the direction n, T be the thickness of the layer, and $\mathbf{F}_n$ be the force moving the layer toward the direction n: then, $\boldsymbol{F}_z \frac{L}{2}$ is the torque acting at the edge rotating the layer toward the membrane and $\boldsymbol{F}_n \frac{D}{2}$ is the torque acting at the edge flipping the layer (see Figure 4.1).

The device will not flip if:

$$|\mathbf{F}_z| > |\mathbf{F}_n|\frac{T}{L} \tag{4.34}$$

If $\mathbf{F}_n$ is $\mathbf{F}_{x,y}$ - $|\mathbf{F}_z|\mu$, where $\mathbf{F}_{x,y}$ is the field generated force in x or y and $|\mathbf{F}_z|\mu$ is the friction, then:

$$|\mathbf{F}_z| > |\mathbf{F}_{x,y}|\frac{1}{\frac{L}{T}+\mu} \tag{4.35}$$

If $\mathbf{F}_z$ is $\mathbf{W}$ σ, then:

$$\sigma > \frac{|\mathbf{F}_{x,y}|}{|\mathbf{W}|}\frac{1}{\frac{L}{T}+\mu} \tag{4.36}$$

The $\mathbf{F}_{x,y}$ represents a force that acts to move the device in the x or y direction. The Swarm Algorithm (SA) returns 'quantized' motion, up, down, left, and/or right, diagonal motion is not allowed. The $\mathbf{F}_{x,y}$ is moderated by friction, **f**, into $\mathbf{F}_n$, which represents the actual, physical acceleration that displaces the device.

To move the device either in the x or y direction, the opposite component force must be zero. The left/right motion (along the x axis) requires $\mathbf{F}_y$ to be zero. The up/down motion (along the y axis) requires $\mathbf{F}_x$ to be zero. Directed motion of the kind is possible through a symmetric adjustment of the currents in the elements within the device's CCA and the elements at the destination.

To stop the device, the z component of the field is increased so that the friction eliminates the motion across a given distance:

$$|\mathbf{F}_z| = \frac{1}{2}\frac{|\mathbf{W}|}{g}\frac{V^2}{\mu\, d} \tag{4.37}$$

where, V is the velocity of the device at the moment where the deceleration is to be triggered and d is the distance across which the deceleration is to act.

The acceleration, in general, is:

$$a = \frac{g}{|\mathbf{W}|}\left(|\mathbf{F}_{x,y}| - |\mathbf{F}_z|\mu\right) \tag{4.38}$$

4.7 Discussion

The magnetic field interaction model is calibrated with respect to a known, analytical solution involving the field and force along the axis of a solenoid.

The magnetic field along the z-axis of a solenoid is:

$$B = \frac{\mu}{2} I \frac{N}{L}\left(\frac{z - L}{\sqrt{a^2 + (z - L)^2}} - \frac{z}{\sqrt{a^2 + z^2}}\right) \tag{4.39}$$

where, I is the current, N is the number of turns, L is the length and a is the radius.

The magnetic force along the z-axis of a solenoid is:

$$F = 2\frac{\chi_m}{\mu_0(1 + \chi_m)} V \left(\frac{\mu}{2} I \frac{N}{L}\right)^2 \left(\frac{z - L}{\sqrt{a^2 + (z - L)^2}} - \frac{z}{\sqrt{a^2 + z^2}}\right)\left(\frac{1}{(a^2 + (z - L)^2)^{\frac{3}{2}}} - \frac{1}{(a^2 + z^2)^{\frac{3}{2}}}\right) \tag{4.40}$$

where, V is the volume of the layer.

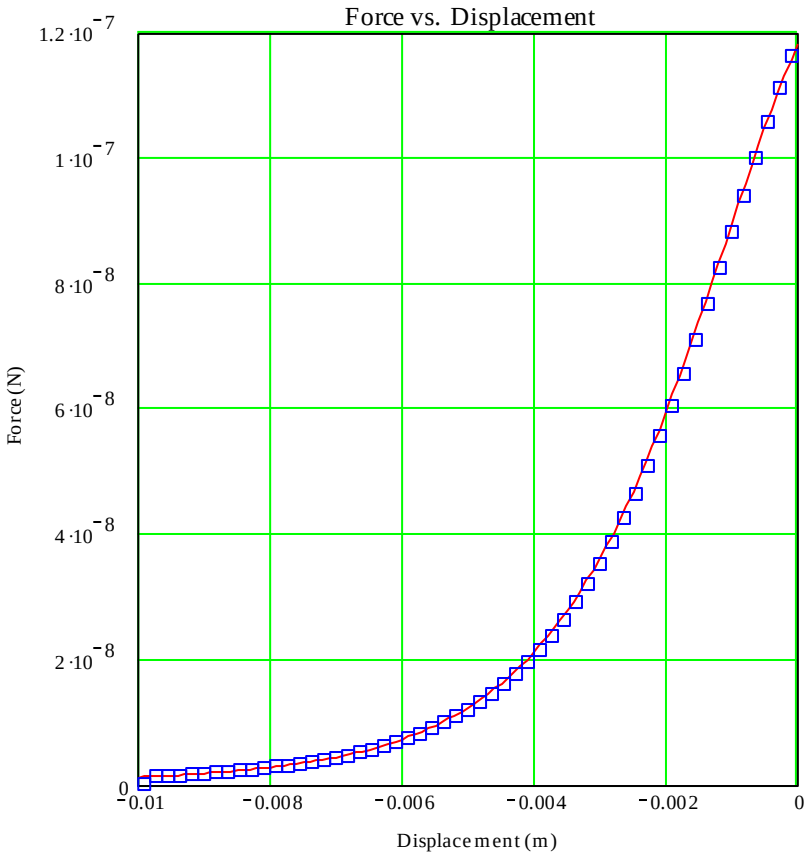

Figure 4.2 A comparison of analytical (red line) versus calculated (blue box) fields at various points along the negative z-axis. The solenoid's parameters are: L, length, 0.20m, N, turns, 50, I, current, 0.005mA. The SR2 MagStat parameter was 50 (SR2 controls the sizes of the partitions used with the Composite Simpson's Rule).

As depicted in Figure 4.2, the analytical and calculated solutions agree.

The CCA calculator of the magnetic field interaction model yields the exact numerical value; however, it is a tedious operation to perform with devices that are either too large or too heavy and arrays that are weaker rather than stronger. A CCA approximation is developed through a ratio that combines device and array properties. First, the device contribution is the product of the total weight, friction factor, and layer area. Second, the array contribution is the product of the maximum contact force, the square of the CCA value, and the BSU area. An additional factor is included to account for the ratio of layer to BSU areas.

$$R = F_0 A_{BSU} CCA^2 f^2 \tag{4.41}$$

where, F_0 is the maximum contact force of the solenoid, A_{BSU} is the area of the BSU, and f^2 is a factor that accounts for the ratio of device to BSU area. The product of F_0 and the square of the CCA value is approximately equal to the net force of attraction of the activated array elements. (The square of the CCA value represents the total number of activated array elements.) The square of the f value is unity when the layer area is less than the BSU area or BSU area divided by layer area when device layer is greater than the BSU area. An f value less than unity is balanced by a larger CCA value in order to keep the ratio balanced.

$$R = WA_{Layer}\sigma \tag{4.42}$$

where, W is the total weight of the device and layer, A_{Layer} is the area of the layer and σ is the friction factor.

Equations 4.41-42 are combined to yield the expression for the CCA value:

$$CCA \approx \frac{1}{f}\sqrt{\frac{W\sigma}{F_0}\frac{A_{Layer}}{A_{BSU}}} \tag{4.43}$$

The CCA value given by Equation 4.43 is to be rounded to the next, odd integer and increased by two. See Figures 4.3-6 for the results of Equation 4.43.

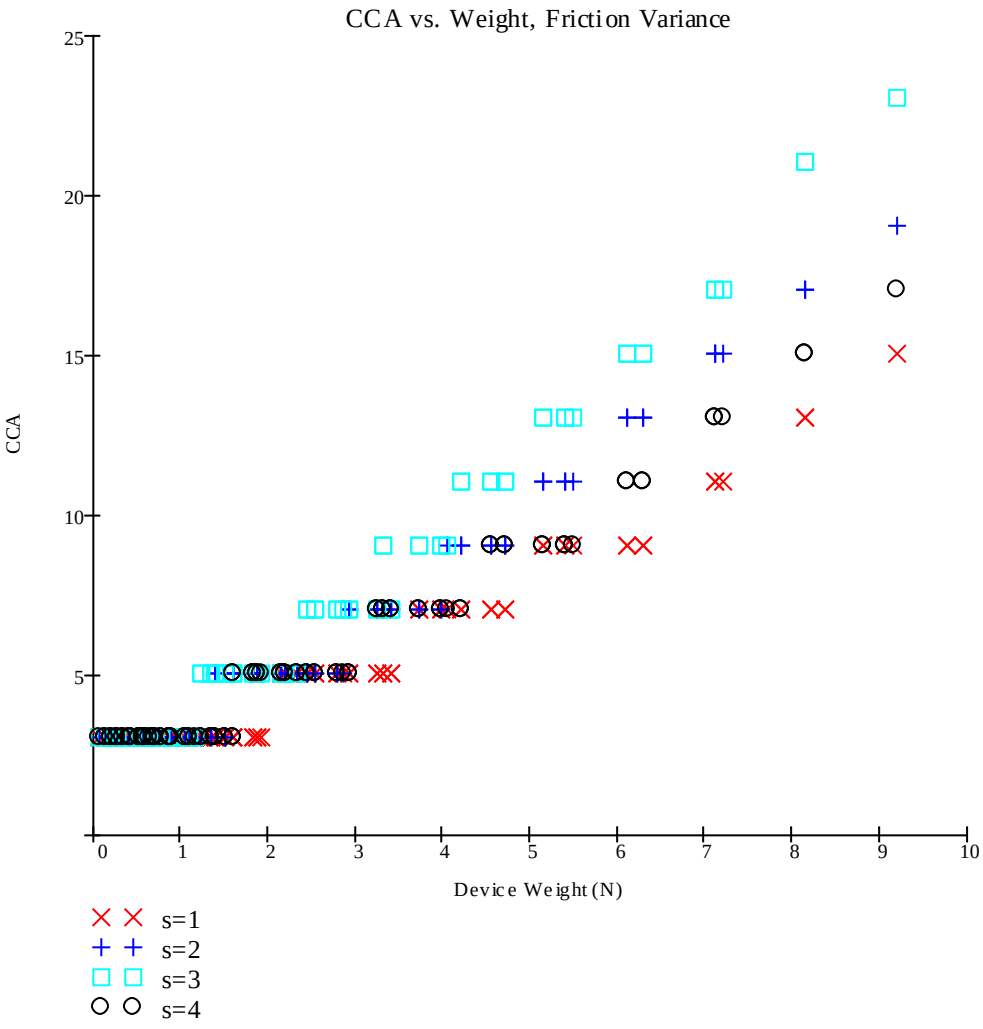

Figure 4.3 CCA versus Weight with friction factor variation. The friction factor varies from 1 to 4. Array parameters are: L = 0.05m, N = 250, I = 250mA, and BSU = 0.01m. The devices are solids of various areas and thickness; device is silicon and layer is Ferroxcube III. As the friction factor increases, the CCA value increases; it is because more and more elements are required to produce the required attraction. Note that, for devices of weights below 5N, the CCA values are 3, 5, 7, and 9, in agreement with the CCA calculator.

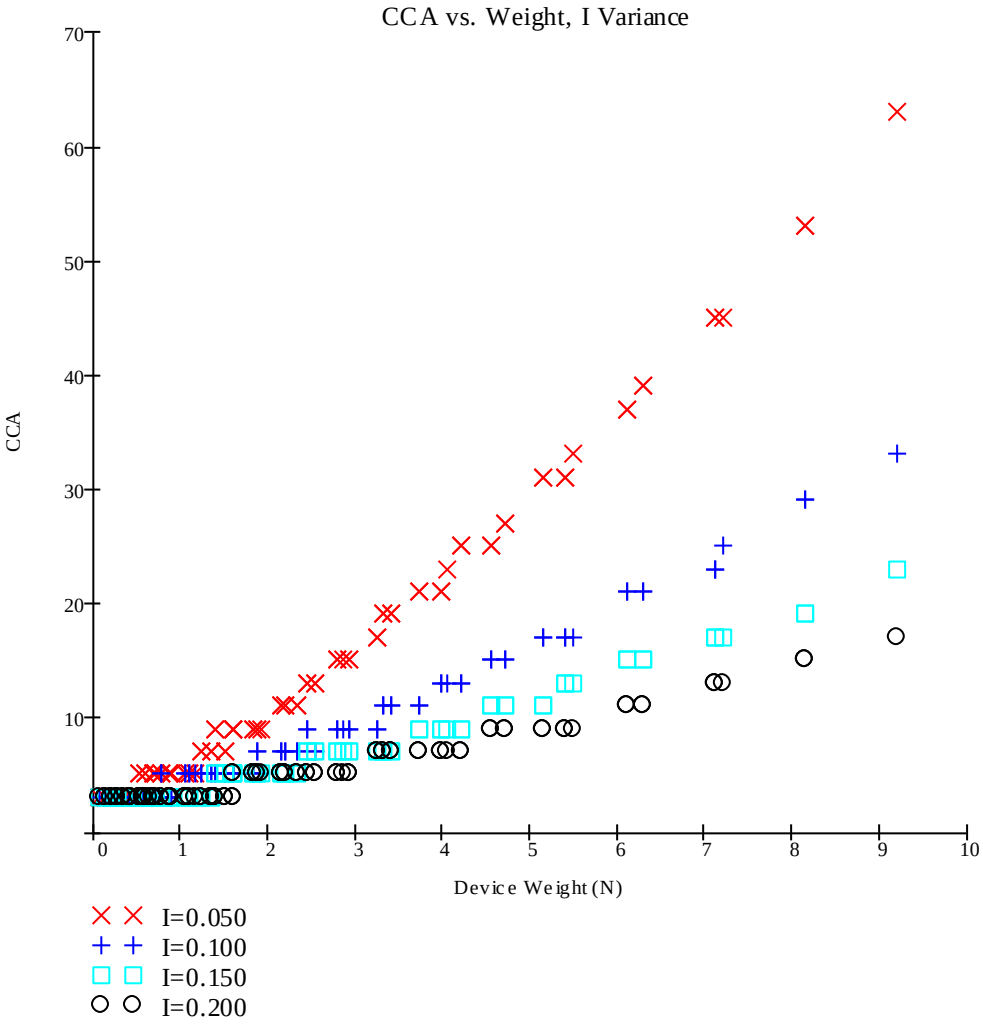

Figure 4.4 CCA versus Weight with current variation. The current varies from 50mA to 200mA. Array parameters are: L = 0.05m, N = 250, and BSU = 0.01m. The devices are solids of various areas and thickness; device is silicon and layer is Ferroxcube III; the friction factor is unity. As the current increases, the CCA value decreases; it is because, as the elements become stronger, fewer are needed to produce the attraction.

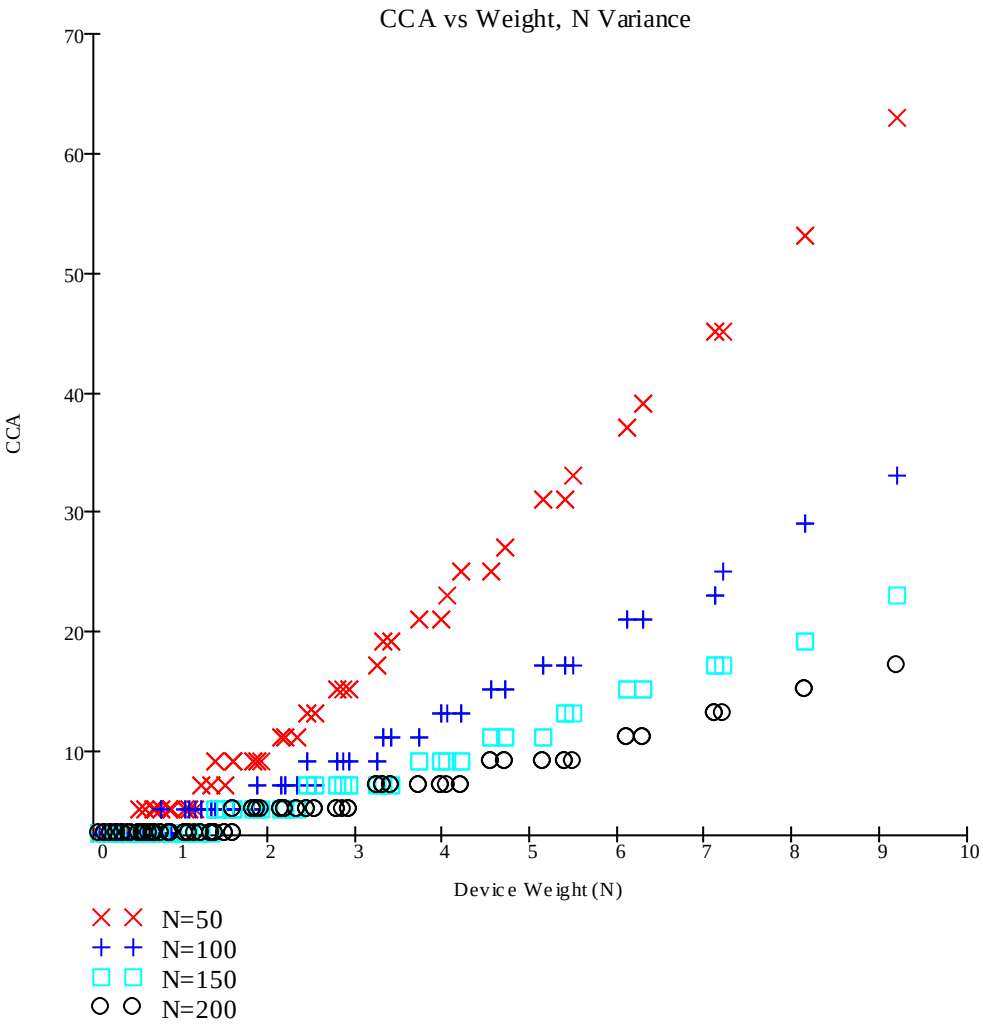

Figure 4.5 CCA versus Weight with turn variation. The turn varies from 50 to 200. Array parameters are: L = 0.05m, N = 250, I = 250mA, and BSU = 0.01m. The devices are solids of various areas and thickness; device is silicon and layer is Ferroxcube III; the friction factor is unity. As the turn increases, the CCA value decreases; it is because, as the elements become stronger, fewer are needed to produce the attraction.

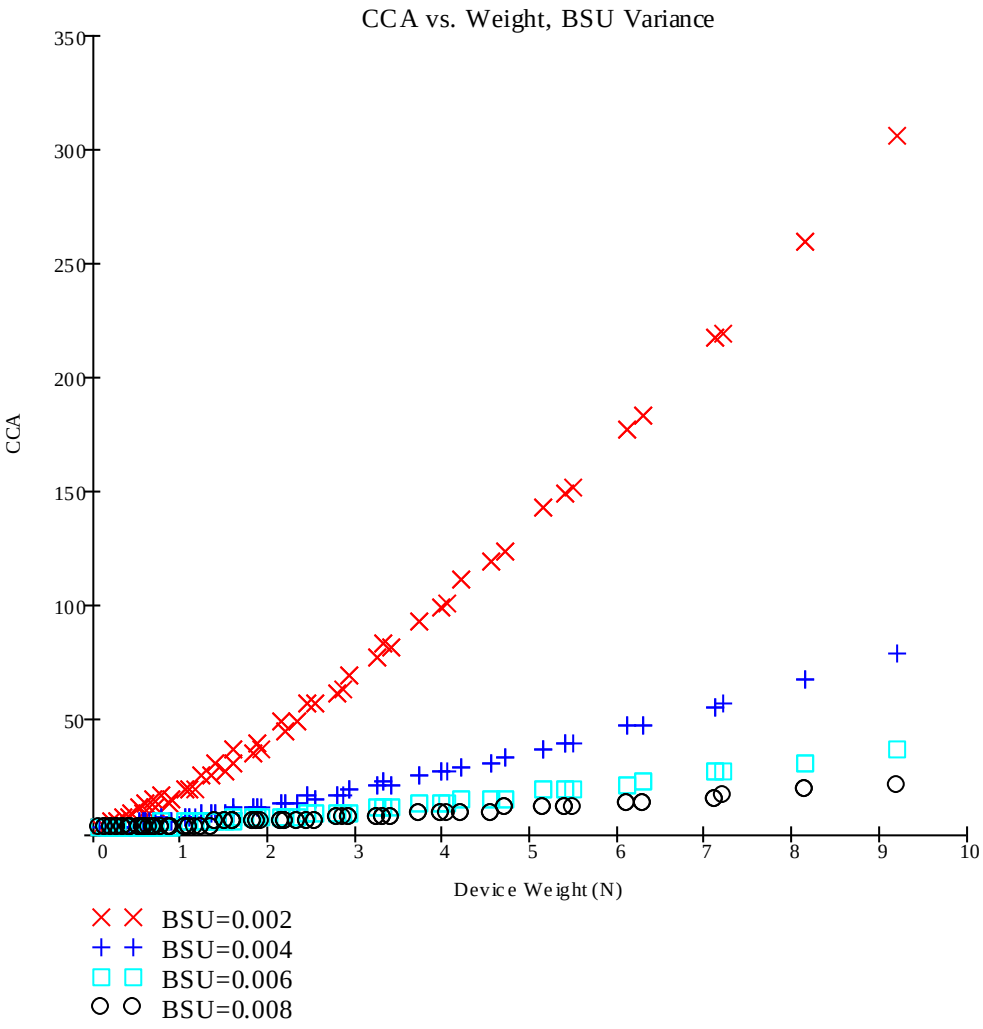

Figure 4.6 CCA versus Weight with BSU variation. The BSU varies from 0.002m to 0.008m. Array parameters are: L = 0.05m, N = 250, and I = 250mA. The devices are solids of various areas and thickness; device is silicon and layer is Ferroxcube III; the friction factor is unity. As the BSU increases, the CCA value decreases and vice-versa; it is because the CCA value is measured in units of BSU and as the BSU size varies, the CCA value is affected accordingly (and linearly).

CHAPTER 5

THE SWARM ALGORITHM

5.1 Overview

The Swarm Algorithm (SA) is a method to calculate pathways of devices, which are constrained to fit within the area of a grid, given known initial and final positions. The technique prevents collisions among devices by constructing a buffer called the Collision Cross-Section Area (CCA). The CCA of a device is not allowed to overlap the CCA of any other device. SA is iterative and, therefore, employs a device-by-device, move-by-move paradigm to extract a solution. It rejects shortest paths in favor of shortest steps required to achieve assembly.

SA yields the solution to the problem of assembly as a sequence of movement, which is to be enacted, simultaneously, at each and every step (STP) of the process. The 'quantized' values of movements are: left/right along the x-axis, up/down along the y-axis, and null. Null is a directive to keep the device static. The movements are displacements of, exactly, a Basic Size Unit (BSU).

A device travels either left/right or up/down (on the horizontal xy plane of the grid) and displaces a distance of a BSU as measured against the initial and the final positions of the center. To achieve a fine-tuned absolute final position, i.e., the ultimate alignment with the recess of the substrate, the device may be moved by an offset. The magnitudes of offsets are to be of values less than a BSU. The sizes of CCAs are inflated to allow devices the ability to adjust within their buffer.

The grid, i.e., the representation of the membrane, is composed of squares 1 BSU x 1 BSU. The BSU relates SA's concept of length to real, physical length. The BSU is the period length of the array of electromagnets.

SA computes movements through a device-by-device, move-by-move iteration, which emphasizes configurations and surroundings. Pathways are calculated by taking into account the device's current and final positions. SA's goal is a parallel solution found through a serial operation. To converge to a solution, the boundary conditions (specified by the device's initial and final positions) and the driving force (specified by device's order of entry into the system) need to be well-posed. Informal, empirical rules are demonstrated to assist with the design of a well-posed problem.

It is important to note that SA is not intended to be used 'live' during the act of assembly. Such a use of SA could be impractical as a set of boundary conditions and driving forces may not yield a solution. Instead, it is to be invoked as a phase of preprocessing.

The output of SA is a list of movement organized by step and then by device. It is designed to be loaded into the memory of an external control unit. The list of movement is to be translated into commands that control the array as required. Devices are moved simultaneously; the process of assembly consists of the conversion from abstract, iterative calculations to real, physical motions.

The step is analogous to a time. The physical temporal duration of a step varies throughout the assembly. The duration of a step is set by the time required of the slowest device to displace a BSU. To SA, all movement within a step is considered to be simultaneous even though the fastest devices will not be moving during the entirety of step. Only the slowest devices move continuously as the fastest devices wait intermittently.

5.2 Physical and Abstract Parameters

Aside from the boundary conditions and the driving forces, three other parameters characterize the SA problem: n, the size of the grid (n x n), maximum, the maximum device, and trigger, the trigger device. The actual, physical size of a unit is a BSU, where the BSU is the period length of the array of electromagnets. Maximum limits the number of devices that are active on the grid. Trigger limits the number of devices that are passive on the template.

Let "total" be the number of devices that are to be assembled. The ratio of maximum to total determines the assembly factor. An assembly factor of unity indicates pure parallel processing where all of the devices are assembled together. A value that tends to zero (as total tends to infinity) indicates pure serial processing, where assembly is on a device-by-device basis. SA, as implemented, will always converge to a solution given an assembly factor of zero. Its ability to solve a hybrid assembly process depends on how well-posed the boundary conditions and driving forces are stated.

Note that the grid is finite at n x n and that the device is inflated via its CCA. Therefore, geometry imposes a limit regarding how many devices may be entered onto the grid in spite of the values of the parameters. The SA method, as implemented, contains a check that prevents injecting devices if there is not enough space available.

The maximum and trigger values control the rate at which the process halts to do an error check/correction and insertion (of devices into recesses). The grid is to be empty to allow the error check/correction algorithm to act as the array will be used to fix misalignments if they emerge. The maximum and trigger parameters act in unison to free the grid of devices prior to the error check/correction and insertion. SA, as implemented, prevents a device from entering the system if the active (on grid) devices (controlled by maximum) plus the inactive (on template) devices (controlled by trigger) is equal to trigger. The effect is that when the number of inactive devices approaches trigger, devices do not enter the system and those already at the grid are processed and dropped, leaving it empty.

5.3 Boundary Conditions and Driving Forces Analogies

The combination of boundary conditions and driving forces are critical inputs to the SA method and take the form of a list: the device and the cue list.

The device list enumerates the initial and final positions as well as the offsets and other properties such as mass, friction, size, and CCA. SA is abstract, except with respect to CCA; mass, friction, and size are included only to assist the external control unit's translation from movement to action. The initial and final positions, size, and CCA are in units of BSU and must be positive. The offset, used to fine-tune position, are fractions of BSU and may be either positive or negative (along x and y) depending on the intended direction of adjustment.

The cue list determines the orders that the devices are injected into the system; it controls the iteration of SA and is not circumventable or adjustable during the calculations.

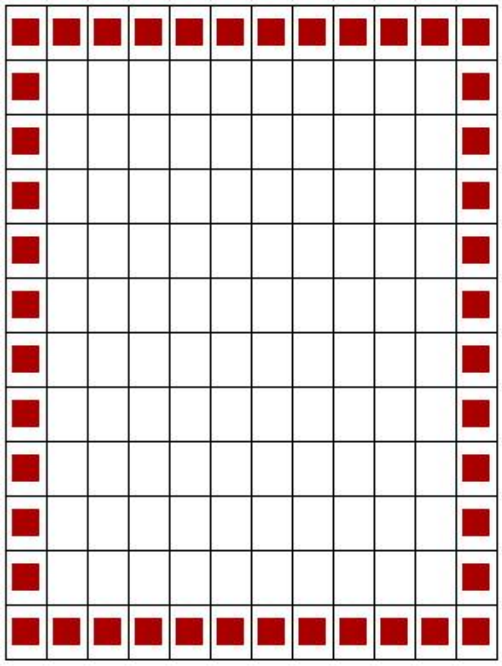

Figure 5.1 A basic 10 x 10 grid. The grid is composed of two areas: the perimeter (red boxes) which is the forbidden zone and the field (white boxes) which is where the devices are free to move.

5.4 The Grid/Membrane

To SA, the grid is the membrane where the size of a unit is defined to be a BSU, the period length of the array of electromagnets.

The grid is a mirror of the template and the substrate: the final position at the grid corresponds to a location on the template and a recess in the substrate.

Only certain parts of the grid are free to admit a device. The interior of the grid, from coordinates (1, 1) to (n, n), is referred to as the field. The perimeter of the grid is referred to as the forbidden zone. A device enters the enclosure and reaches the forbidden zone. There, it waits until the final insertion into the field. SA, as implemented, restricts backward (or any other kind of motion) that leads a device in the field to the forbidden zone. The forbidden zone must be free to admit devices. See Figure 5.1.

5.5 Decision Tables

The SA approach issues a movement based on the results of the P and Q Tables.

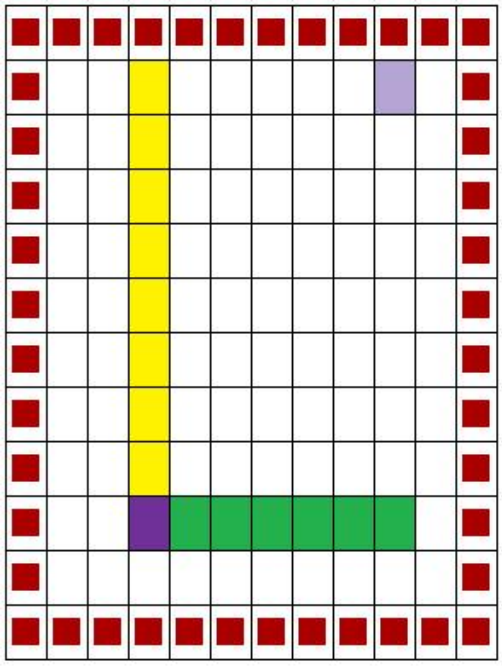

Figure 5.2 A view of the P Table. The device is dark purple. The exit is light purple. The P Table constructs two test pathways: a left/right (green) and an up/down (yellow). The lengths of these paths are determined by the displacements between current and desired positions.

5.5.1 P Table

The P Table examines two orthogonal pathways: a single up/down and a single left/right track. Note: it evaluates only up/down *or* left/right pathways. The P Table is far-sighted yet does not plan ahead.

The P Table is compiled for up/down and left/right pathways as determined by a device's current location and final position (see Figure 5.2). It contains the vector displacement value of the track, which will be used to opt between up or down, left or right movement. It adds the answers to two yes/no questions. First, is the displacement equal to zero? Second, is the pathway blocked?

SA chooses the shortest, non-zero pathway that is not blocked. If two or more pathways are available, then the precedent is up, down, left, and right. If all pathways fail, then the P Table issues a null movement and SA shifts onto the Q Table.

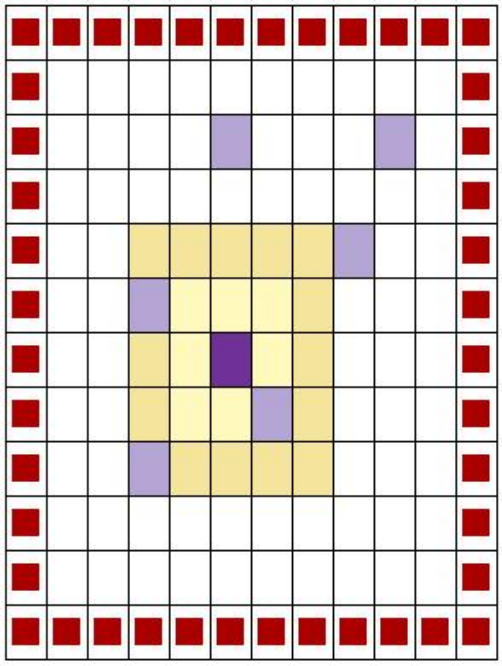

Figure 5.3 A view of the Q Table. The device is dark purple. Various other devices are light purple. The Q Table examines a region immediately around the device (light and dark tan). The light tan are the surroundings of the device's current immediate position, the dark tan is the future position of the four possible movements: up, down, left, and right.

5.5.2 Q Table

The Q Table examines the perimeter of the device. It evaluates up, down, left, and right. The Q Table is short-sighted yet does not negate a movement.

The Q Table is compiled for up, down, left, and right as determined by the device's current location; final position is ignored (see Figure 5.3). It examines the density of a pathway. It includes the answers to three yes/no questions. First, is a direction blocked? Second, is a direction forbidden? Third, does the direction negate the last, known movement?

SA chooses the direction that is not blocked and forbidden, that does not undo the last, known movement, and that leads to the lowest density. If two or more directions are available, then the precedent is up, down, left, and right. If all directions fail then the Q Table issues a null movement and SA does not alter the device's position.

5.5.3 Null Movement

The goal of SA is to move devices from step to step as they seek their final destinations. If the boundary conditions and driving forces are not well-posed, then frustration will be allowed to enter the system. If a device is stuck due to frustration, then SA issues that device a null movement to indicate that it will not be moved at that step. SA fails when all of the devices are issued a null movement. SA passes when all of the devices are inserted.

5.6 Safety Valves

SA, as implemented, incorporates a set of safety valves the goal of which is to ensure that the boundary conditions and driving forces are well-posed and that it yields a solution.

The device list is checked against four criteria. First, the initial position must be at the perimeter of the field excluding its corners. Second, the final position must be within the field and must be unique. Third, the magnitude of the offset must be fractional units of BSU. Fourth, device mass, friction, size, and CCA must be equal to or greater than zero; additionally, the CCA is bounded by:

$$\mathrm{CCA} \leq \sqrt{\mathrm{n}} \tag{5.1}$$

where, n is the size of the grid.

The cue list is checked to ensure that its contents refer to devices with properly. inputted positions, offsets, and parameters.

Additionally, devices enter the grid if and only if the full area they occupy is free. They may move into the field. They may not move into the forbidden zone. To prevent frustration at the field/forbidden interface, the density of the forbidden zone is twice that of the field zone as a bias (see Figure 5.4).

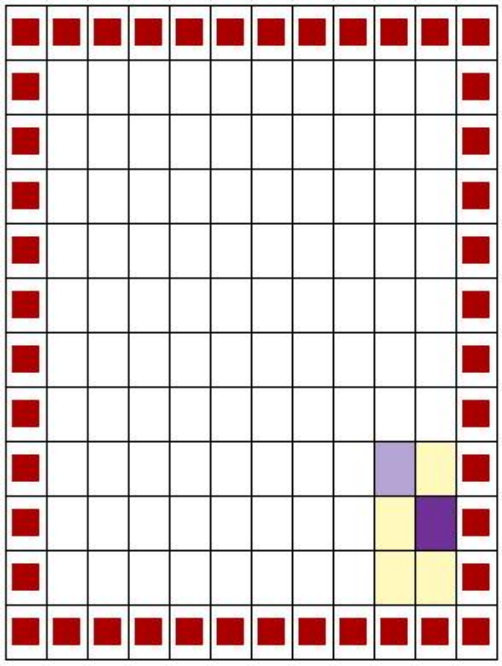

Figure 5.4 Density is a measure of local population/occupation. A device (dark purple) is surrounded by: three empty field grids (tan), occupied device grids (light purple), and three forbidden grids (red). The density is 7; the empty field grids are 0 per grid, the occupied device grids is 1 per grid, and the forbidden grids are 2 per grid. The inflation of the forbidden grids is designed to keep devices away from that area of the grid.

SA is not able to detect all errors. Care must be placed on the design of the device and cue list as well as the max and trig parameters. In general, the fewer the number of devices, the greater the likelihood of a solution. It is preferable to have more small and fewer large devices in terms of CCA at any given step. Setting trigger greater than maximum assures a fast solution. Also, the length that a device needs to travel from initial to final positions should not be larger than half the length of the grid; therefore, devices that enter at a side of the grid should be taken only to those spaces closest to that side (see Figure 5.5).

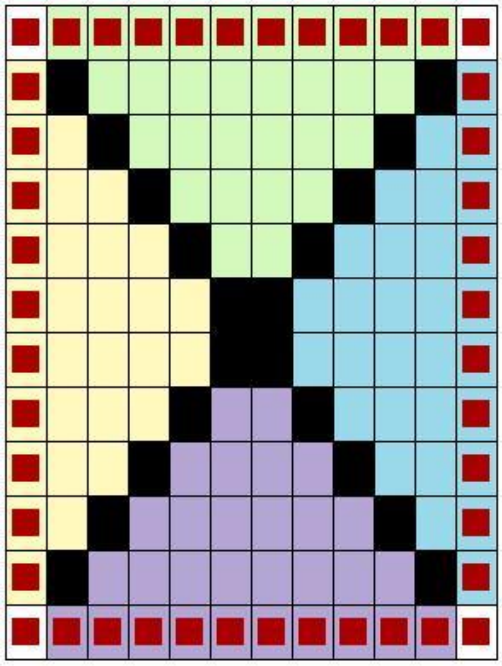

Figure 5.5 A graphic representation of the optimized path-length configuration. The grid is divided by two main diagonals into four areas (tan, green, blue, and purples). Devices that enter through an X-colored area of the grid ought to be given final positions within that X-colored area; for example, green to green, purple to purple, etc.

5.7 Algorithm

A simplified version of the SA algorithm follows; various internal steps are omitted.

set variables t_count and w_count to zero (t_count is the total number of devices to be assembled; w_count is the total number of devices already assembled)

loop:

A. if possible, inject devices

B. set variables active to injected and killed to zero (active is the number of devices waiting to be moved on the grid; killed is the number of devices that were not able to be moved)

C. for each active/injected device:

1. create P and Q Tables

2. if P Table is not null, issue that movement

3. if P Table is null:

a. if Q Table is not null, issue that movement

b. if Q Table is null:

1. issue that movement

2. decrement active by unity

3. increment killed by unity

4. if device reached final position:

a. issue drop directive

b. increment t_count by unity

c. decrement active by unity

d. decrement killed to zero

D. if killed is equal to or greater than active, SA failed, end

E. if w_count is equal to total, SA passed, end

F. if t_count is equal to trig:

1. issue error/insert directive

2. increment w_count by t_count

3. decrement t_count to zero

5.8 Discussion

The modeling of two systems will be compared; first, an ill-posed problem, second, a well-posed problem. Both systems involve twenty devices with identical initial positions. For the ill-posed system, shown in Figure 5.6, the final positions are random. For the well-posed system, shown in Figure 5.7, the final positions are within the areas indicated by Figure 5.5.

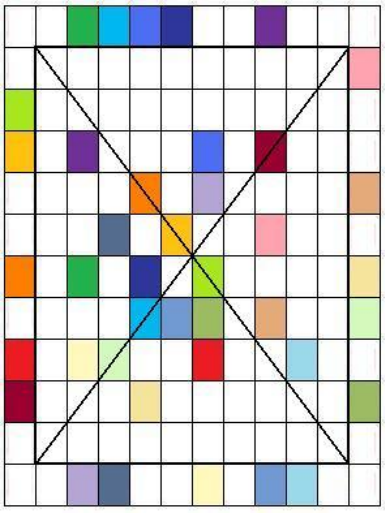

Figure 5.6 The ill-posed system. A group of twenty devices (represented by colors where each device is a color) start at the perimeter and move toward their final positions within the field. The system is ill-posed because the final positions are random and their displacements (distance from initial to final positions) are on average greater than half the length of the grid.

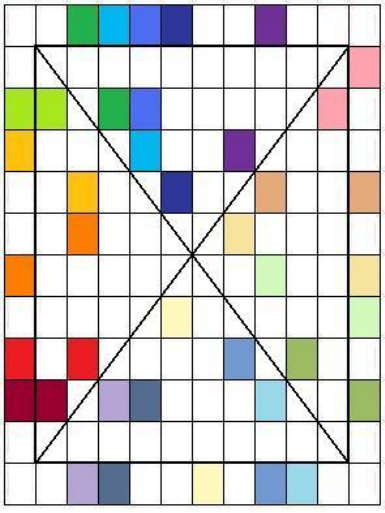

Figure 5.7 The well-posed system. A group of twenty devices (represented by colors where each device is a color) start at the perimeter and move toward their in-area final positions within the field. The system is well-posed because the final positions are within their immediate initial regions and their displacements (distance from initial to final positions) are on average equal to or lesser than half the length of the grid.

The systems were compiled under various conditions. First, the trigger device value was alternated between 10 and 20; trigger device controls the rate of insertions. Second, the maximum device value was changed, continuously, from 1 to 20; maximum device controls how many devices are active, simultaneously, on the grid. Figures 5.8 and 5.9 represent the result of the Swarm Algorithm given those systems and its conditions.

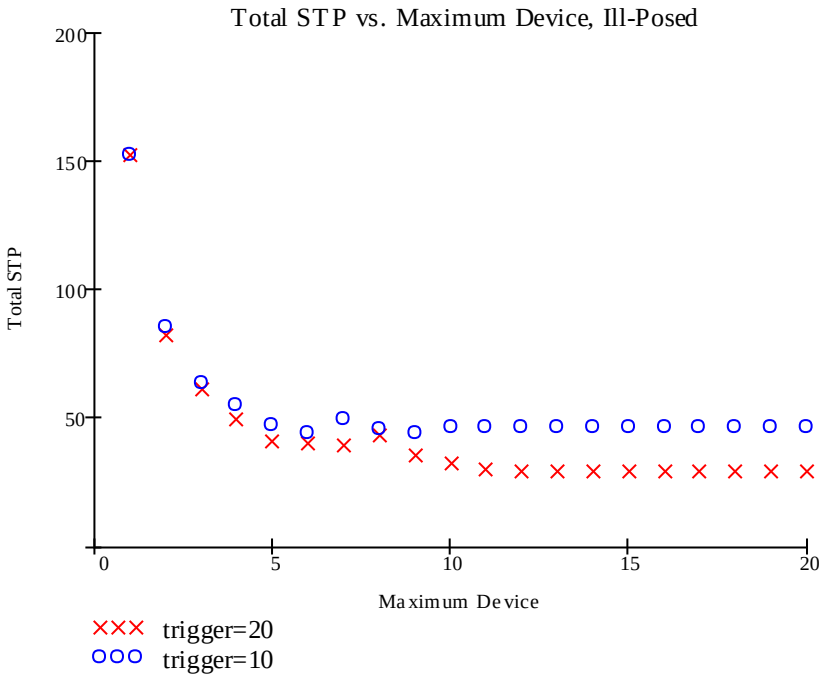

Figure 5.8 The ill-posed system plotted for trigger device values of 10 and 20. The total STP value, the number of steps required to complete assembly, varied from 152 for maximum device equal to 1 to 46 (for trigger = 10) and 29 (for trigger = 20) for maximum device equal to 20. At the serial process limit, the STP is equal to 152. At the parallel process limit, the STP is 46 for trigger = 10 and 29 for trigger = 20.

The ill-posed system (see Figures 5.6 and 5.8) contains a bottle-neck caused by the random mixture of different sized devices and scattered final positions; the bottle-neck is a condition where the current placement of a device forbids the movement of other, nearby devices already on the grid. After a maximum device value of 10, the STP is fairly constant for both trigger device values; an STP, or step, is an iteration of the method. The bottle-neck must be overcome in order to complete assembly; thereafter, assembly continues more or less identically.

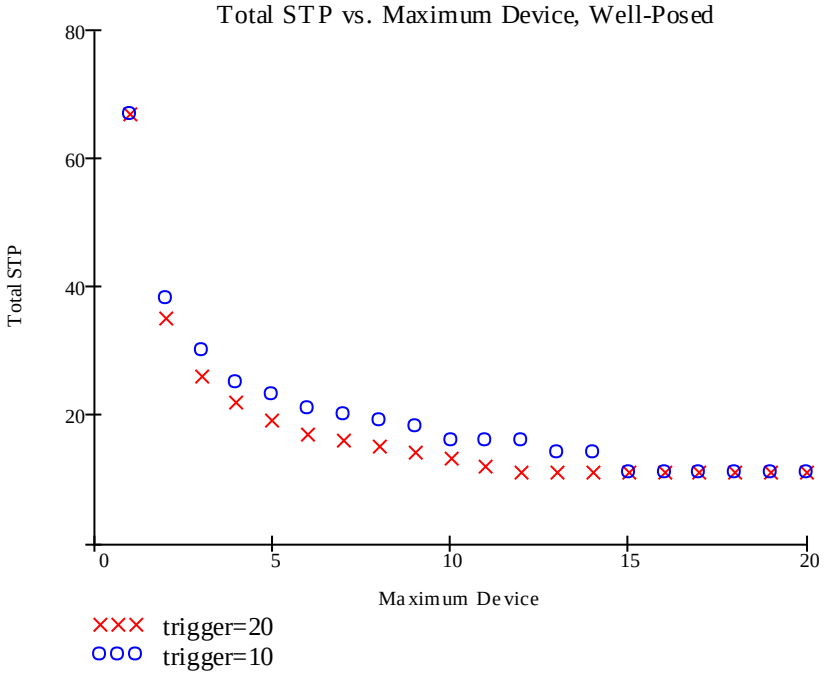

Figure 5.9 The well-posed system plotted for trigger device values of 10 and 20. The total STP, the number of steps required to complete assembly, varied from 67 for maximum device equal to 1 to 11 for maximum device equal to 20. At the serial process limit, the STP is equal to 67. At the parallel process limit, the STP is 11.

The well-posed system (see Figures 5.7 and 5.9), however, is not bottle-necked; the placement of a device does not prevent the movement of other, nearby devices. The serial processing and parallel processing limit are identical irrespective of the trigger value, indicating that a well-posed system is scalable. The amount of steps (or total STP) required to complete assembly is also consistently lower at both processing limits.

Note that the order of entry (driving force) and the initial positions (the first part of the boundary condition) are identical with the ill-posed and well-posed systems. The key differences are the final positions (the second part of the boundary condition). For the ill-posed system, the final positions were random throughout the field. For the well-posed system, the final positions were constrained to be within the corresponding perimeter to area sectors as depicted by Figure 5.5.

CHAPTER 6

CONCLUSIONS AND FUTURE DIRECTIONS

6.1 Conclusions

The goals of the models presented by this dissertation were: first, to calculate pathways, second, to calculate magnetic field interaction, third, to calculate the attraction between devices and recesses, and, fourth, to maximize that attraction while minimizing its contribution to weight.

The Swarm Algorithm method, introduced, developed, and coded in this thesis, demonstrated that deterministic, simultaneous assembly is viable. An informal set of rules were developed to assist the creation of well-posed combinations of boundary conditions and driving forces. A software toolkit was created to implement the Swarm Algorithm technique and it was used to model a variety of systems.

The magnetic field interaction model was presented and compared to a system with a known, analytic solution. The method was used to demonstrate the fields and forces required to manipulate various example devices and arrays. Additionally, the Collision Cross-Section Area was calculated for those examples.

The device/recess force was derived. It was used to model the attraction between soft and hard magnetic materials that retain devices and recesses respectively. It showed that the force of attraction was weak beyond a certain, critical distance; therefore, inclusion of such magnetic material will not affect the functioning of the substrate.

The Magnetic Retention Factor was a useful tool in conjunction with the engineering of the Magnetic Field Assisted Assembly.

6.2 Future Directions

The study of the Magnetic Field Driven Simultaneous Assembly method is incomplete without a significant amount of research and development with respect to its implementation.

The magnetic field interaction model introduced by this dissertation needs to be generalized to admit the dynamics of field and device responses. The device/recess force needs to be extended from one to two dimensions, including arbitrary geometries of devices and recesses.

It should be noted that the Swarm Algorithm is already dynamic. Improvements that could be implemented include: the ability to adjust boundary conditions and driving forces to prevent the failure of the algorithm and the addition of other auxiliary tables to calculate true, deep-sighted movement. Although not an improvement that would be useful to the problem of assembly, the Swarm Algorithm is extendable to n dimensional space and to any set of movement.

The important area of research and development is the engineering of the method.

First, the method of injection must be developed. A device may be contained either in bins or on tapes and then fed into the system. The act of injection is required to maintain the integrity of vacuum in the enclosure. Injection, whether through bins or tapes, ought to be efficient; therefore, the work to implement bins or tapes may not impose too great a burden to time and cost.

Second, the function of the array of electromagnets must be explored. The methodology to translate 'movement' into real, physical manipulation is to be developed. Various other aspects of the array, regarding its construction and arrangement, including the way it would be controlled must be explored as it effects the act of assembly.

Additionally, work involving the other magnetic field based methods proposed by the team at the New Jersey Institute of Technology should be pursued. It includes the Magnetic Field Assisted Assembly (MFAA) and the Method of Assembly Using An Array of Programmable Magnets (see Figure 6.1) [111].

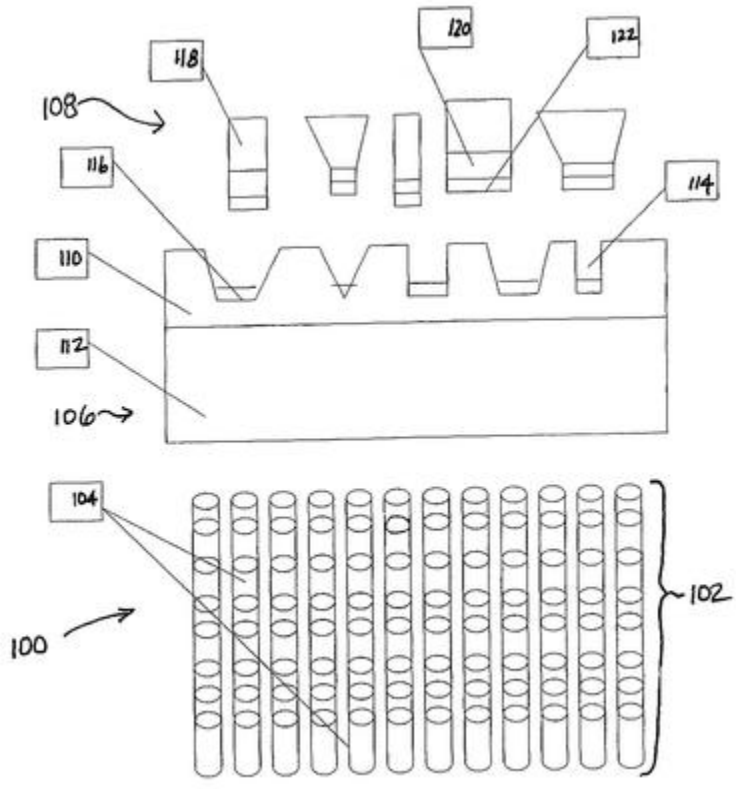

Figure 6.1 A schematic showing the method of assembly using programmable magnets. A hybrid of Magnetic Field Assisted Assembly and Magnetic Field Driven Simultaneous Assembly, it remains a work in progress [111].

APPENDIX A

DEVICE/RECESS FORCE DERIVATION

The following appendix is a derivation of Equation 2.11; see Figure A.1 for explanation of terms.

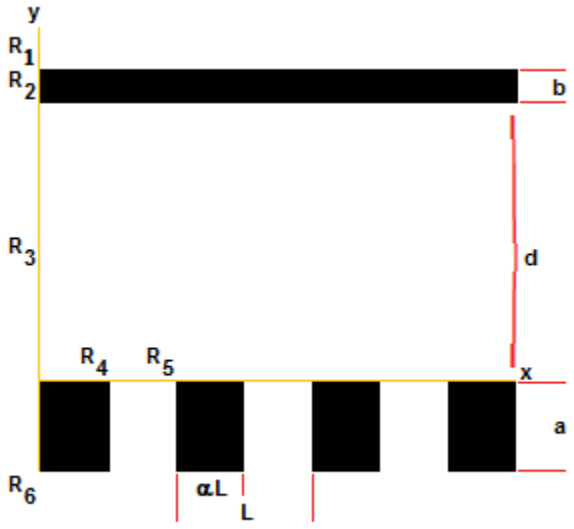

Figure A.1 A cross-section of the system involving soft magnetic layers and hard magnetic strips. R_1 is the device, R_2 is the soft magnetic layer, R_3 is air-gap, R_4 and R_5 are the hard magnetic strip and the gap, and R_6 is the substrate. The soft magnetic layer's thickness is b. The hard magnetic strip's thickness is a. The air-gap distance is d. The period length of the strip/gap pattern is L where αL is the width of the hard magnetic strip.

The expression for the force is:

$$\overline{\boldsymbol{F}} = \mu_0 \int_{\substack{0 \\ \varphi \to \varphi_1 \\ y=d+b}}^{L} \varphi_x \varphi_y \hat{\boldsymbol{x}} + \frac{1}{2}\left(\varphi_y^2 - \varphi_x^2\right)\hat{\boldsymbol{y}}\, dx - \mu_0 \int_{\substack{0 \\ \varphi \to \varphi_3 \\ y=a}}^{L} \varphi_x \varphi_y \hat{\boldsymbol{x}} + \frac{1}{2}\left(\varphi_y^2 - \varphi_x^2\right)\hat{\boldsymbol{y}}\, dx \quad \text{(A.1)}$$

The following are the simplified magnetic potentials:

$$\varphi_1 = \sum_{n=1}^{\infty} e^{-\Delta_n(y-(d+b))}(a_n cos\Delta_n x + b_n sin\Delta_n x) \tag{A.2}$$

$$\varphi_2 = \sum_{n=1}^{\infty} \left(\alpha_n^1 cosh\Delta_n(y-d) + \alpha_n^2 sinh\Delta_n(y-d)\right) cos\Delta_n x + \sum_{n=1}^{\infty} \left(\beta_n^1 cosh\Delta_n(y-d) + \beta_n^2 sinh\Delta_n(y-d)\right) sin\Delta_n x \tag{A.3}$$

$$\varphi_3 = \sum_{n=1}^{\infty} (\gamma_n^1 cosh\Delta_n y + \gamma_n^2 sinh\Delta_n y)\, cos\Delta_n x + \sum_{n=1}^{\infty} (\delta_n^1 cosh\Delta_n y + \delta_n^2 sinh\Delta_n y) sin\Delta_n x \tag{A.4}$$

$$\varphi_4 = \varphi_5 = \sum_{n=1}^{\infty} (\rho_n^1 cosh\Delta_n y + \rho_n^2 sinh\Delta_n y)\, cos\Delta_n x + \sum_{n=1}^{\infty} (\omega_n^1 cosh\Delta_n y + \omega_n^2 sinh\Delta_n y) sin\Delta_n x \tag{A.5}$$

$$\varphi_6 = \sum_{n=1}^{\infty} e^{\Delta_n(y+a)}(c_n cos\Delta_n x + d_n sin\Delta_n x) \tag{A.6}$$

where:

$$\Delta_n = \frac{2\pi}{L} n \tag{A.7}$$

Through Equations A.2-6 (and derivatives), with the boundary condition that $-\hat{\boldsymbol{n}} \cdot \nabla\varphi|_0 = -\hat{\boldsymbol{n}} \cdot \nabla\varphi|_1 + \hat{\boldsymbol{n}} \cdot \bar{\boldsymbol{M}}_{\boldsymbol{0}}$, relations are found among the constants.

$$\alpha_n^1 cosh\Delta_n b + \alpha_n^2 sinh\Delta_n b = a_n \tag{A.8}$$

$$\beta_n^1 cosh\Delta_n b + \beta_n^2 sinh\Delta_n b = b_n \tag{A.9}$$

$$\alpha_n^1 sinh\Delta_n b + \alpha_n^2 cosh\Delta_n b = -\frac{1}{\varepsilon} a_n \tag{A.10}$$

$$\beta_n^1 sinh\Delta_n b + \beta_n^2 cosh\Delta_n b = -\frac{1}{\varepsilon} b_n \tag{A.11}$$

$$\gamma_n^1 cosh\Delta_n b + \gamma_n^2 sinh\Delta_n b = \alpha_n^1 \tag{A.12}$$

$$\delta_n^1 cosh\Delta_n b + \delta_n^2 sinh\Delta_n b = \beta_n^1 \tag{A.13}$$

$$\gamma_n^1 sinh\Delta_n b + \gamma_n^2 cosh\Delta_n b = \varepsilon\alpha_n^2 \tag{A.14}$$

$$\delta_n^1 sinh\Delta_n b + \delta_n^2 cosh\Delta_n b = \varepsilon\beta_n^2 \tag{A.15}$$

$$\gamma_n^1 = \rho_n^1 \tag{A.16}$$

$$\delta_n^1 = \omega_n^1 \tag{A.17}$$

$$\gamma_n^2 - \rho_n^2 = \frac{\bar{a}_n}{\Delta_n} \tag{A.18}$$

$$\delta_n^2 - \omega_n^2 = \frac{\bar{b}_n}{\Delta_n} \tag{A.19}$$

$$\rho_n^1 cosh\Delta_n a - \rho_n^2 sinh\Delta_n a = c_n \tag{A.20}$$

$$\omega_n^1 cosh\Delta_n a - \omega_n^2 sinh\Delta_n a = d_n \tag{A.21}$$

$$\rho_n^1 sinh\Delta_n a - \rho_n^2 cosh\Delta_n a + c_n = \frac{\bar{a}_n}{\Delta_n} \tag{A.22}$$

$$\omega_n^1 sinh\Delta_n a - \omega_n^2 cosh\Delta_n a + d_n = \frac{\bar{b}_n}{\Delta_n} \tag{A.23}$$

where:

$$\varepsilon = \frac{\mu}{\mu_0} \tag{A.24}$$

The constants of Equations A.8-23 must be solved in terms of $\bar{a}_n, \bar{b}_n$ which will be determined by the system. Equations A.8-23 represent simple, linear systems that are solved algebraically as:

$$\alpha_n^1 = a_n\left(cosh\Delta_n b + \frac{1}{\varepsilon}sinh\Delta_n b\right) \tag{A.25}$$

$$\alpha_n^2 = -a_n\left(\frac{1}{\varepsilon}cosh\Delta_n b + sinh\Delta_n b\right) \tag{A.26}$$

$$\beta_n^1 = b_n\left(cosh\Delta_n b + \frac{1}{\varepsilon}sinh\Delta_n b\right) \tag{A.27}$$

$$\beta_n^2 = -b_n\left(\frac{1}{\varepsilon}cosh\Delta_n b + sinh\Delta_n b\right) \tag{A.28}$$

$$\gamma_n^1 = \alpha_n^1 cosh\Delta_n d - \varepsilon\alpha_n^2 sinh\Delta_n d \tag{A.29}$$

$$\gamma_n^2 = -\alpha_n^1 sinh\Delta_n d + \varepsilon\alpha_n^2 cosh\Delta_n d \tag{A.30}$$

$$\delta_n^1 = \beta_n^1 cosh\Delta_n d - \varepsilon\beta_n^2 sinh\Delta_n d \tag{A.31}$$

$$\delta_n^2 = -\beta_n^1 sinh\Delta_n d + \varepsilon\beta_n^2 cosh\Delta_n d \tag{A.32}$$

$$\rho_n^1 = \gamma_n^1 \tag{A.33}$$

$$\rho_n^2 = \gamma_n^2 - \frac{\bar{a}_n}{\Delta_n} \tag{A.34}$$

$$\omega_n^1 = \delta_n^1 \tag{A.35}$$

$$\omega_n^2 = \delta_n^2 - \frac{\bar{b}_n}{\Delta_n} \tag{A.36}$$

Adding Equations A.20 and A.22, simplifying the cosh/sinh terms:

$$(\rho_n^1 - \rho_n^2)e^{\Delta_n a} = \frac{\bar{a}_n}{\Delta_n} \tag{A.37}$$

Substituting Equations A.33-34, rearranging terms:

$$(\gamma_n^1 - \gamma_n^2)e^{\Delta_n a} = \frac{\bar{a}_n}{\Delta_n}(e^{\Delta_n a} - 1) \tag{A.38}$$

Substituting Equations A.29-30, simplifying and rearranging terms:

$$\alpha_n^1 - \varepsilon\alpha_n^2 = \frac{\bar{a}_n}{\Delta_n}e^{-\Delta_n d}(e^{\Delta_n a} - 1) \tag{A.39}$$

Substituting Equations A.25-26, simplifying and rearranging terms:

$$a_n\left(2cosh\Delta_n b + \left(\frac{1}{\varepsilon} + \varepsilon\right)sinh\Delta_n b\right) = \frac{\bar{a}_n}{\Delta_n}e^{-\Delta_n d}(e^{\Delta_n a} - 1) \tag{A.40}$$

Adding Equations A.21 and A.23, simplifying the cosh/sinh terms:

$$(\omega_n^1 - \omega_n^2)e^{\Delta_n a} = \frac{\bar{b}_n}{\Delta_n} \tag{A.41}$$

Substituting Equations A.35-36, rearranging terms:

$$(\delta_n^1 - \delta_n^2)e^{\Delta_n a} = \frac{\bar{b}_n}{\Delta_n}(e^{\Delta_n a} - 1) \tag{A.42}$$

Substituting Equations A.31-32, simplifying and rearranging terms:

$$\beta_n^1 - \varepsilon\beta_n^2 = \frac{\bar{b}_n}{\Delta_n}e^{-\Delta_n d}(e^{\Delta_n a} - 1) \tag{A.43}$$

Substituting Equations A.27-28, simplifying and rearranging terms:

$$b_n\left(2cosh\Delta_n b + \left(\frac{1}{\varepsilon} + \varepsilon\right)sinh\Delta_n b\right) = \frac{\bar{b}_n}{\Delta_n}e^{-\Delta_n d}(e^{\Delta_n a} - 1) \tag{A.44}$$

Equations A.40 and A.44 are simplified as:

$$a_n = \bar{a}_n\sigma \tag{A.45}$$

$$b_n = \bar{b}_n\sigma \tag{A.46}$$

where:

$$\sigma = \frac{1}{\eta}\frac{1}{\Delta_n}e^{-\Delta_n d}(e^{-\Delta_n a} - 1) \tag{A.47}$$

$$\eta = 2cosh\Delta_n b + \left(\frac{1}{\varepsilon} + \varepsilon\right)sinh\Delta_n b \tag{A.48}$$

With Equations A.45-46, Equations A.25-32 are rewritten as:

$$\alpha_n^1 = \bar{a}_n\sigma\left(cosh\Delta_n b + \frac{1}{\varepsilon}sinh\Delta_n b\right) \tag{A.49}$$

$$\alpha_n^2 = -\bar{a}_n\sigma\left(\frac{1}{\varepsilon}cosh\Delta_n b + sinh\Delta_n b\right) \tag{A.50}$$

$$\beta_n^1 = \bar{b}_n\sigma\left(cosh\Delta_n b + \frac{1}{\varepsilon}sinh\Delta_n b\right) \tag{A.51}$$

$$\beta_n^2 = -\bar{b}_n\sigma\left(\frac{1}{\varepsilon}cosh\Delta_n b + sinh\Delta_n b\right) \tag{A.52}$$

$$\gamma_n^1 = \bar{a}_n\sigma\left(e^{\Delta_n d}cosh\Delta_n b + \left(\frac{1}{\varepsilon}cosh\Delta_n d + \varepsilon sinh\Delta_n d\right)sinh\Delta_n b\right) \tag{A.53}$$

$$\gamma_n^2 = -\bar{a}_n\sigma\left(e^{\Delta_n d}cosh\Delta_n b + \left(\varepsilon cosh\Delta_n d + \frac{1}{\varepsilon}sinh\Delta_n d\right)sinh\Delta_n b\right) \tag{A.54}$$

$$\delta_n^1 = \bar{b}_n \sigma \left(e^{\Delta_n d} cosh\Delta_n b + \left(\frac{1}{\varepsilon} cosh\Delta_n d + \varepsilon sinh\Delta_n d \right) sinh\Delta_n b \right) \tag{A.55}$$

$$\delta_n^2 = -\bar{b}_n \sigma \left(e^{\Delta_n d} cosh\Delta_n b + \left(\varepsilon cosh\Delta_n d + \frac{1}{\varepsilon} sinh\Delta_n d \right) sinh\Delta_n b \right) \tag{A.56}$$

Let R and S be defined as:

$$R = e^{\Delta_n d} cosh\Delta_n b + \left(\frac{1}{\varepsilon} cosh\Delta_n d + \varepsilon sinh\Delta_n d \right) sinh\Delta_n b \tag{A.57}$$

$$S = e^{\Delta_n d} cosh\Delta_n b + \left(\varepsilon cosh\Delta_n d + \frac{1}{\epsilon} sinh\Delta_n d \right) sinh\Delta_n b \tag{A.58}$$

then:

$$S^2 - R^2 = \left(\varepsilon - \frac{1}{\varepsilon} \right) \left(2cosh\Delta_n b + \left(\frac{1}{\varepsilon} + \varepsilon \right) sinh\Delta_n b \right) sinh\Delta_n b \tag{A.59}$$

$$\frac{1}{\eta^2} (S^2 - R^2) = \frac{sinh\Delta_n b}{sinh\left(\Delta_n b + ln\left(\frac{\varepsilon + 1}{\varepsilon - 1} \right) \right)} \tag{A.60}$$

also:

$$a_n^2 + b_n^2 = \left(\bar{a}_n^2 + \bar{b}_n^2 \right) \frac{1}{\eta^2} \frac{1}{\Delta_n^2} e^{-2\Delta_n d} (1 - e^{-\Delta_n a})^2 \tag{A.61}$$

The $\bar{a}_n, \bar{b}_n$ constants are found by a Fourier Series of the magnetization of the hard magnetic strip, whose function is:

$$M(x) = \begin{cases} M_0 & 0 \le x \le \alpha L \\ 0 & x > \alpha L \end{cases} \tag{A.62}$$

$$\bar{a}_0 = \frac{1}{L}\int_0^L M(x)dx = \frac{1}{L}\int_0^{\alpha L} M_0 dx = \alpha M_0 \tag{A.63}$$

$$\bar{a}_n = \frac{2}{L}\int_0^L M(x)cos\Delta_n x dx = \frac{2}{L}\int_0^{\alpha L} M_0 cos\Delta_n x dx = \frac{1}{n\pi} M_0 sin2\pi\alpha n \tag{A.64}$$

$$\bar{b}_n = \frac{2}{L}\int_0^L M(x)sin\Delta_n x dx = \frac{2}{L}\int_0^{\alpha L} M_0 sin\Delta_n x dx = \frac{1}{n\pi} M_0(1 - cos2\pi\alpha n) \tag{A.65}$$

$$\bar{a}_n^2 + \bar{b}_n^2 = \frac{2}{n^2\pi^2} M_0^2(1 - cos2\pi\alpha n) \tag{A.66}$$

Substituting Equation A.66 into Equation A.61:

$$a_n^2 + b_n^2 = \frac{2}{n^2\pi^2} M_0^2(1 - cos2\pi\alpha n)\frac{1}{\eta^2}\frac{1}{\Delta_n^2} e^{-2\Delta_n d}(1 - e^{-\Delta_n a})^2 \tag{A.67}$$

The general form of the potential is:

$$\varphi = \sum_{n=1}^{\infty} (P_n cos\Delta_n x + Q_n sin\Delta_n x) \tag{A.68}$$

$$\partial_x \varphi = \sum_{n=1}^{\infty} \Delta_n (-P_n sin\Delta_n x + Q_n cos\Delta_n x) \tag{A.69}$$

$$\partial_y \varphi = \sum_{n=1}^{\infty} (P_n' cos\Delta_n x + Q_n' sin\Delta_n x) \tag{A.70}$$

The general form of the force is:

$$\overline{\boldsymbol{F}} = F_x \hat{\boldsymbol{x}} + F_y \hat{\boldsymbol{y}} \tag{A.71}$$

$$F_x = \pm\mu_0 \int_0^L \partial_x \varphi \partial_y \varphi dx = \pm \frac{\mu_0}{2} L \sum_{n=1}^{\infty} \Delta_n (Q_n P_n' - P_n Q_n') \tag{A.72}$$

$$F_y = \pm\mu_0 \int_0^L \partial_y \varphi^2 - \partial_x \varphi^2 dx = \pm \frac{\mu_0}{4} L \sum_{n=1}^{\infty} (P_n'^2 + Q_n'^2 - \Delta_n^2 P_n^2 - \Delta_n^2 Q_n^2) \tag{A.73}$$

$$\overline{\boldsymbol{F}} = \pm \frac{\mu_0}{2} L \sum_{n=1}^{\infty} \left(\Delta_n (Q_n P_n' - P_n Q_n') \hat{\boldsymbol{x}} + \frac{1}{2} (P_n'^2 + Q_n'^2 - \Delta_n^2 P_n^2 - \Delta_n^2 Q_n^2) \hat{\boldsymbol{y}} \right) \tag{A.74}$$

The force on the soft magnetic layer (R_2) due to the hard magnetic strips (R_4 and R_5) is the difference between the force at R_3 at the lower boundary of the layer and the force at R_1 at the upper boundary of the layer.

$$\overline{\boldsymbol{F}}_2 = \overline{\boldsymbol{F}}_3|_{\boldsymbol{y=d}} - \overline{\boldsymbol{F}}_1|_{\boldsymbol{y=d+b}} \tag{A.75}$$

At R_1, the P_n and Q_n constants are:

$$P_n = a_n e^{-\Delta_n(y-(d+b))} \tag{A.76}$$

$$P_n' = -\Delta_n a_n e^{-\Delta_n(y-(d+b))} \tag{A.77}$$

$$Q_n = b_n e^{-\Delta_n(y-(d+b))} \tag{A.78}$$

$$Q_n' = -\Delta_n b_n e^{-\Delta_n(y-(d+b))} \tag{A.79}$$

Equations A.76-79 are evaluated at the boundary, y = d + b, as:

$$P_n = a_n \tag{A.80}$$

$$P_n' = -\Delta_n a_n \tag{A.81}$$

$$Q_n = b_n \tag{A.82}$$

$$Q_n' = -\Delta_n b_n \tag{A.83}$$

$$\overline{\boldsymbol{F}}_{\mathbf{1}} = \frac{\mu_0}{2} L \sum_{n=1}^{\infty} \left(\Delta_n(-b_n a_n + a_n b_n)\hat{\boldsymbol{x}} + \frac{1}{2}(\Delta_n^2 a_n^2 + \Delta_n^2 b_n^2 - \Delta_n^2 a_n^2 - \Delta_n^2 b_n^2)\hat{\boldsymbol{y}} \right) = 0 \tag{A.84}$$

At R_3, the P_n and Q_n constants are:

$$P_n = \gamma_n^1 cosh\Delta_n a + \gamma_n^2 sinh\Delta_n a \tag{A.85}$$

$$P_n' = \Delta_n(\gamma_n^1 sinh\Delta_n a + \gamma_n^2 cosh\Delta_n a) \tag{A.86}$$

$$Q_n = \delta_n^1 cosh\Delta_n a + \delta_n^2 sinh\Delta_n a \tag{A.87}$$

$$Q_n' = \Delta_n(\delta_n^1 sinh\Delta_n a + \delta_n^2 cosh\Delta_n a) \tag{A.88}$$

$$Q_n P_n' - P_n Q_n' = \Delta_n(\delta_n^1 \gamma_n^2 - \delta_n^2 \gamma_n^1) \tag{A.89}$$

Substituting Equations A.57-58 into Equation A.53-56:

$$\gamma_n^1 = \bar{a}_n \sigma R \tag{A.90}$$

$$\gamma_n^2 = -\bar{a}_n \sigma S \tag{A.91}$$

$$\delta_n^1 = \bar{b}_n \sigma R \tag{A.92}$$

$$\delta_n^2 = -\bar{b}_n \sigma S \tag{A.93}$$

Substituting Equations A.90-93 into Equation A.89:

$$Q_n P_n' - P_n Q_n' = \Delta_n\left(-\bar{b}_n \sigma \bar{a}_n \sigma S + \bar{b}_n \sigma S \bar{a}_n \sigma R\right) = 0 \tag{A.94}$$

$$P_n'^2 + Q_n'^2 - \Delta_n^2 P_n^2 - \Delta_n^2 Q_n^2 = \Delta_n^2\left(-{\gamma_n^1}^2 + {\gamma_n^2}^2 - {\delta_n^1}^2 + {\delta_n^2}^2\right) \tag{A.95}$$

Substituting Equations A.90-93 into Equation A.95:

$$P_n'^2 + Q_n'^2 - \Delta_n^2 P_n^2 - \Delta_n^2 Q_n^2 = \Delta_n^2 \sigma^2\left(\bar{a}_n^2 + \bar{b}_n^2\right)(S^2 - R^2) \tag{A.96}$$

$$\bar{\boldsymbol{F}}_{\boldsymbol{3}} = -\frac{\mu_0}{2} L \sum_{n=1}^{\infty}\left(0\hat{\boldsymbol{x}} + \frac{1}{2}\Delta_n^2 \sigma^2\left(\bar{a}_n^2 + \bar{b}_n^2\right)\left(S^2 - R^2\right)\hat{\boldsymbol{y}}\right) \tag{A.97}$$

Substituting Equation A.47 into Equation A.97:

$$\bar{\boldsymbol{F}}_{\boldsymbol{3}} = -\frac{\mu_0}{4} L \sum_{n=1}^{\infty}\left(\Delta_n^2 \frac{1}{\eta^2}\frac{1}{\Delta_n^2} e^{-2\Delta_n d}\left(1 - e^{-\Delta_n a}\right)^2\left(\bar{a}_n^2 + \bar{b}_n^2\right)\left(S^2 - R^2\right)\hat{\boldsymbol{y}}\right) \tag{A.98}$$

Substituting Equation A.60 into Equation A.98:

$$\bar{\boldsymbol{F}}_{\boldsymbol{3}} = -\frac{\mu_0}{4} L \sum_{n=1}^{\infty}\left(e^{-2\Delta_n d}\left(1 - e^{-\Delta_n a}\right)^2\left(\bar{a}_n^2 + \bar{b}_n^2\right)\frac{sinh\Delta_n b}{sinh\left(\Delta_n b + ln\left(\frac{\varepsilon + 1}{\varepsilon - 1}\right)\right)}\hat{\boldsymbol{y}}\right) \tag{A.99}$$

Substituting Equation A.66 into Equation A.99:

$$\overline{\boldsymbol{F}}_{\boldsymbol{3}} = -\frac{\mu_0 M_0^2}{2\pi^2} L \sum_{n=1}^{\infty} \left(\frac{1}{n^2} (1 - cos2\pi\alpha n) e^{-2\Delta_n d} (1 - e^{-\Delta_n a})^2 \frac{sinh\Delta_n b}{sinh\left(\Delta_n b + ln\left(\frac{\varepsilon+1}{\varepsilon-1}\right)\right)} \hat{\boldsymbol{y}} \right) \tag{A.100}$$

Substituting Equations A.84 and A.100 into Equation A.75:

$$\overline{\boldsymbol{F}}_{\boldsymbol{2}} = -\frac{\mu_0 M_0^2}{2\pi^2} L \sum_{n=1}^{\infty} \left(\frac{1}{n^2} (1 - cos2\pi\alpha n) e^{-2\Delta_n d} (1 - e^{-\Delta_n a})^2 \frac{sinh\Delta_n b}{sinh\left(\Delta_n b + ln\left(\frac{\varepsilon+1}{\varepsilon-1}\right)\right)} \hat{\boldsymbol{y}} \right) \tag{A.101}$$

The Equation A.101 is maximized when α = 0.5; at that α the terms of the summation are 0 when n is even and 2 when n is odd.

$$\overline{\boldsymbol{F}}_{\boldsymbol{2}} = -\frac{\mu_0 M_0^2}{\pi^2} L \sum_{\substack{n=1 \\ n\ odd}}^{\infty} \left(\frac{1}{n^2} e^{-2\Delta_n d} (1 - e^{-\Delta_n a})^2 \frac{sinh\Delta_n b}{sinh\left(\Delta_n b + ln\left(\frac{\varepsilon+1}{\varepsilon-1}\right)\right)} \hat{\boldsymbol{y}} \right) \tag{A.102}$$

The n = 1 term dominates the summation; therefore:

$$\overline{\boldsymbol{F}}_{\boldsymbol{2}} = -\frac{\mu_0 M_0^2}{\pi^2} L e^{-2\Delta_n d} (1 - e^{-\Delta_n a})^2 \frac{sinh\Delta_n b}{sinh\left(\Delta_n b + ln\left(\frac{\varepsilon+1}{\varepsilon-1}\right)\right)} \tag{A.103}$$

APPENDIX B

SWARM APPLICATION CODE

The complete Swarm Algorithm program written in freeBASIC; [112] the application is divided into modules MAIN, DECLIB, APPLIB, CMPLIB, GUILIB, MODLIB, ZCOMPILE, ZEIPORT, ZLISTANB, ZPROJECT, and ZRECORD.

MAIN *is the central function that controls all other functions; links to all other modules*

DECLIB *contains structure definitions, variable definitions, and sub/function prototypes*

APPLIB *contains functions that define and maintain the program's file system and databases; used by all modules*

build_idxA	builds list of projects, 1st phase
build_idxB	builds list of projects, 2nd phase
build_dlp	builds list of project data
load	loads records from file
save	saves records to file
sift	sifts records based on sift keys
sort	sorts records based on sort keys
free	returns next free record number
mkcda	converts A-code into displayable data
mkcdz	converts Z-code into displayable data
mkdrp	adds * to device symbol if dropped
mkgnw	toggles between grid and window mode
pack	initializes record data

CMPLIB *contains functions that define the Swarm Algorithm; used by the ZPROJECT module*

build_ptable	constructs the P Table
build_qtable	constructs the Q Table
grid_000	resets the grid array
grid_add	inserts device to grid array
grid_sub	removes device from grid array
grid_upd	updates the grid array
stack_add	inserts device to stack array
stack_sub	removes device from stack array
dense	calculates density
scan_ptable	obtains decision from the P Table
scan_qtable	obtains decision from the Q Table
switch	resolves Up/Down, Left/Right from P Table
test	tests if a device can be added to stack
text$	converts device number to letter

GUILIB *contains functions that define and maintain the program's graphical user interface; used by all modules*

prompt_list_A	defines the input list screen
prompt_list_B	defines the output list screen
prompt_main	defines the main screen
prompt_project	defines the project screen
record_list_A1	displays input/lst record data
record_list_A2	displays input/dev record data
record_list_B	displays output/prc record data
record_main1	displays main project list data
record_main2	displays main project file data
record_project1	displays project grid data
record_project2	displays project file data
record_project3	displays project device data

MODLIB *contains various low-level functions used by the program's high-level functions; used by all modules*

export_devl	exports the dev input list
export_lstl	exports the lst input list
export_prcl	exports the prc output list
export_raw	exports input dev/lst data to raw format
export_stat	exports statistics
import_dev	imports dev data from raw format
import_lst	imports lst data from raw format
check_dev	integrity - if lst item is valid
check_fin	integrity - if final location is valid
check_gns	integrity - if lst item is unique
check_ini	integrity - if initial location is valid
check_mfsd	integrity - if properties are valid
check_off	integrity - if offset location is valid
check_sng	integrity - if final location is unique

ZCOMPILE *is the main body that controls the Swarm Algorithm; used by the ZPROJECT module*

ZEIPORT *maintains the data import and export functions; used by the ZPROJECT and ZLISTANB modules*

zexport	the export function
zimport	the import function
zprntscrn	the print screen function

ZLISTANB *maintains the input and output lists; used by the ZPROJECT module*

zlist_A	function that displays input data
zlist_B	function that displays output data

ZPROJECT *maintains the project's main functions; links to the ZLISTANB, ZEIPORT, and ZCOMPLIE modules*

proj_grid	sets up device data
proj_trace	sets up trace function
proj_stack	sets up stack data

ZRECORD *maintains the program's database (the project list and the input/output lists); used by the ZPROJECT, ZCOMPILE, and MAIN modules*

zrecord_dnl	controls the input data records
zrecord_mfdsa	controls the project records
zrecord_prc	controls the output data records
zreset	resets the project input/output records

MAIN Module:

```
defshort a-z
#include "zbasic.bi"
#include "_dec_lib_.bi"
#include "_app_lib_.bi"
#include "_gui_lib_.bi"
#include "_mod_lib_.bi"
#include "_cmp_lib_.bi"
#include "_zlistAnB_.bi"
#include "_zeiport_.bi"
#include "_zrecord_.bi"
#include "_zcompile_.bi"
#include "_zproject_.bi"

RGBA0(MAIN,1)=0: RGBA0(MAIN,2)=15
RGBA0(FRAM,1)=0: RGBA0(FRAM,2)=15
RGBA0(ACT1,1)=14: RGBA0(ACT1,2)=0
RGBA0(ACT2,1)=0: RGBA0(ACT2,2)=14
RGBA0(MENU,1)=0: RGBA0(MENU,2)=15
RGBA0(ERRL,1)=15: RGBA0(ERRL,2)=8
RGBA0(WIRE,1)=0: RGBA0(WIRE,2)=15
RGBA0(DOSX,1)=15: RGBA0(DOSX,2)=0

ZCMD 1,"MFDA-v3.IDX","B",PASS
if lof(1)=0 then gosub initialize
load 1,0,"IDXA"
Zi

gosub main_start
gosub main_fresh
gosub main_prompt
gosub main_record1
gosub main_record2

do
  CTRL$=ZHEAD$("MENU:   Add-A   Edit-E   Del-D   Mode-
   [C/Z]   Reset-R   [ENT]   [ESC]",Z_ARU$+Z_ARD$+Z_ARL$+
   Z_ARR$+Z_ARH$+Z_ARE$+Z_PGU$+Z_PGD$+Z_SPC$+"AED"+
  Z_UND$+"R"+Z_ENT$+Z_ESC$)
  select case CTRL$
  case Z_ARU$,Z_ARD$,Z_ARL$,Z_ARR$,Z_ARH$,Z_ARE$,Z_PGU$,
   Z_PGD$:
    ZSYS f0,f1,f2,36,IDX(0,0),CTRL$
    gosub main_record1
    gosub main_record2
  case Z_SPC$:
    gosub main_start
    gosub main_fresh
    gosub main_prompt
    gosub main_record1
    gosub main_record2
```

```
  case "A":
    zrecord_mfdsa 1,0,"A"
    gosub main_start
    gosub main_fresh
    gosub main_prompt
    gosub main_record1
    gosub main_record2
  case "E","D",Z_UND$,"R",Z_ENT$:
    if IDX(0,0)<>0 then
      select case CTRL$
      case "E":
        zrecord_MFDSA 1,IDX(0,f0),"E"
        gosub main_record1
        gosub main_record2
      case "D":
        zrecord_MFDSA 1,IDX(0,f0),"D"
        gosub main_start
        gosub main_fresh
        gosub main_prompt
        gosub main_record1
        gosub main_record2
      case Z_UND$:
        zrecord_MFDSA 1,IDX(0,f0),Z_UND$
        gosub main_record1
        gosub main_record2
      case "R":
        zreset IDX(0,f0)
        gosub main_record1
        gosub main_record2
      case Z_ENT$:
        zproject IDX(0,f0)
        gosub main_prompt
        gosub main_record1
        gosub main_record2
      end select
    end if
  case Z_ESC$:
    exit do
  end select
loop
Zo
save 1,0,"IDXA"
ZCMD 1,"","",FAIL
end

initialize:
idxA.a_record=0
idxA.r_record=0
idxA.m_record=32767
save 1,0,"IDXA"
return

main_start:
build_idxA 1,0
build_idxB 1,0
sort 0
return
```

```
main_fresh:
ZSYS f0,f1,f2,36,IDX(0,0),Z_ARH$
return

main_prompt:
prompt_main "Index"
return

main_record1:
record_main1 f0,f1,f2,0
return

main_record2:
record_main2 IDX(0,f0)
return
```

DECLIB Module:

```
type idxASpace
  a_record as short
  r_record as short
  m_record as short
end type
dim shared idxA as idxASpace

type idxBSpace
  fmfdsa as string*9
  fname  as string*32
  cntrl  as string*1
end type
dim shared idxB(32767) as idxBSpace

type mfdsaSpace
  acode as short
  zcode as short
  n     as short
  m     as short
  t     as short
  t_dev as short
  t_lst as short
  t_prc as short
  t_stp as short
  c_dte as string*10
  c_tme as string*8
end type
dim shared mfdsa(32767) as mfdsaSpace

type devSpace
  dev_n as short
  ini_x as short
  ini_y as short
  fin_x as short
  fin_y as short
  off_x as single
  off_y as single
  mass  as single
  fric  as single
  size  as single
  delta as short
  cntrl as string*1
end type
dim shared dev(32767) as devSpace

type lstSpace
  lst_n as short
  dev_n as short
  cntrl as string*1
end type
dim shared lst(32767) as lstSpace
```

```
type prcSpace
  prc_n as short
  dev_n as short
  stp_n as short
  text  as string*8
  move  as short
  ini_x as short
  ini_y as short
  fin_x as short
  fin_y as short
  t_cnt as short
  w_cnt as short
end type
dim shared prc(32767) as prcSpace

type stackSpace
  dev_n as short
  dev_s as string*1
  delta as short
  move  as short
  cur_x as short
  cur_y as short
  fin_x as short
  fin_y as short
  cntrl as short
end type
dim shared stack(32767) as stackSpace

type ptableSpace
  null as short
  blck as short
  leng as short
end type
dim shared ptable(2) as ptableSpace

type qtableSpace
  forb  as short
  blck  as short
  dense as short
  move  as short
  upd_x as short
  upd_y as short
end type
dim shared qtable(4) as qtableSpace

dim shared grid(1024,1024) as short,IDX(8,32767) as short,g as
short,max_dev as short,trg_dev as short,cur_dev as short,tot_dev
as short,cur_prc as short,tot_prc as short

const GF=0
const GB=1
const XK=2
const XF=2
const XB=-2
```

```
const LR=1
const UD=2
const LD=3

const NL=0
const UP=1
const DN=2
const LT=3
const RT=4

declare sub build_idxA(...)
declare sub build_idxB(...)
declare sub build_dlp(...)
declare sub load(...)
declare sub save(...)
declare sub sift(...)
declare sub sort(...)

declare function free(...)
declare function mkcda$(...)
declare function mkcdz$(...)
declare function mkdrp$(...)
declare function mkgnw$(...)
declare function pack(...)

declare sub prompt_list_A(...)
declare sub prompt_list_B(...)
declare sub prompt_main(...)
declare sub prompt_project(...)
declare sub record_list_A1(...)
declare sub record_list_A2(...)
declare sub record_list_B(...)
declare sub record_main1(...)
declare sub record_main2(...)
declare sub record_project1(...)
declare sub record_project2(...)
declare sub record_project3(...)

declare sub export_devl(...)
declare sub export_lstl(...)
declare sub export_prcl(...)
declare sub export_raw(...)
declare sub export_stat(...)
declare sub import_dev(...)
declare sub import_lst(...)

declare function check_dev(...)
declare function check_fin(...)
declare function check_gns(...)
declare function check_ini(...)
declare function check_mfsd(...)
declare function check_off(...)
declare function check_sng(...)
```

```
declare sub build_ptable(...)
declare sub build_qtable(...)
declare sub grid_000(...)
declare sub grid_add(...)
declare sub grid_sub(...)
declare sub grid_upd(...)
declare sub stack_add(...)
declare sub stack_sub(...)

declare function dense(...)
declare function scan_ptable(...)
declare function scan_qtable(...)
declare function switch(...)
declare function symbol$(...)
declare function test(...)
declare function text$(...)

declare sub proj_grid(...)
declare sub proj_trace(...)
declare function proj_stack(...)

declare sub zlist_A(...)
declare sub zlist_B(...)

declare sub zexport(...)
declare sub zimport(...)
declare sub zprntscrn(...)

declare sub zrecord_dnl(...)
declare sub zrecord_mfdsa(...)
declare sub zrecord_prc(...)
declare sub zreset(...)

declare sub zcompile(...)

declare sub zproject(...)
```

APPLIB Module:

```
sub build_idxA(n,u)
  IDX(u,0)=0
  for z0=1 to idxA.a_record
    load n,z0,"IDXB"
    if idxB(z0).cntrl<>"*" then
      IDX(u,0)=IDX(u,0)+1
      IDX(u,IDX(u,0))=z0
    end if
    save n,z0,"IDXB"
  next z0
end sub

sub build_idxB(n,u)
  for z0=1 to IDX(u,0)
    ZCMD 100,idxB(IDX(u,z0)).fmfdsa,"B",PASS
    load 100,IDX(u,z0),"MFDSA"
    save 100,IDX(u,z0),"MFDSA"
    ZCMD 100,"","",FAIL
  next z0
end sub

sub build_dlp(n,u,zmax,CTRL$)
  IDX(u,0)=0
  for z0=1 to zmax
    load n,z0,CTRL$
    if (CTRL$<>"DEV" ) or (CTRL$="DEV"  and dev(z0).cntrl<>
     "*" ) then
      if (CTRL$<>"LST" ) or (CTRL$="LST"  and lst(z0).cntrl
      <>"*" ) then
        IDX(u,0)=IDX(u,0)+1
        IDX(u,IDX(u,0))=z0
      end if
    end if
  next z0
end sub

sub load(n,k,CTRL$)
  select case CTRL$
  case "IDXA":
    seek #n,1
    ZGET n,tpx$,2: idxA.a_record=cvshort(tpx$)
    ZGET n,tpx$,2: idxA.r_record=cvshort(tpx$)
    ZGET n,tpx$,2: idxA.m_record=cvshort(tpx$)
  case "IDXB":
    seek #n,42&*(k-1)+10
    ZGET n,idxB(k).fmfdsa,9
    ZGET n,idxB(k).fname,32
    ZGET n,idxB(k).cntrl,1
```

```
  case "MFDSA":
    seek #n,1
    ZGET n,tpx$,2: mfdsa(k).acode=cvshort(tpx$)
    ZGET n,tpx$,2: mfdsa(k).zcode=cvshort(tpx$)
    ZGET n,tpx$,2: mfdsa(k).n   =cvshort(tpx$)
    ZGET n,tpx$,2: mfdsa(k).m   =cvshort(tpx$)
    ZGET n,tpx$,2: mfdsa(k).t   =cvshort(tpx$)
    ZGET n,tpx$,2: mfdsa(k).t_dev=cvshort(tpx$)
    ZGET n,tpx$,2: mfdsa(k).t_lst=cvshort(tpx$)
    ZGET n,tpx$,2: mfdsa(k).t_prc=cvshort(tpx$)
    ZGET n,tpx$,2: mfdsa(k).t_stp=cvshort(tpx$)
    ZGET n,tpx$,10: mfdsa(k).c_dte=tpx$
    ZGET n,tpx$,8: mfdsa(k).c_tme=tpx$
  case "DEV":
    seek #n,66&*(k-1)+1+36
    ZGET n,tpx$,2: dev(k).dev_n=cvshort(tpx$)
    ZGET n,tpx$,2: dev(k).ini_x=cvshort(tpx$)
    ZGET n,tpx$,2: dev(k).ini_y=cvshort(tpx$)
    ZGET n,tpx$,2: dev(k).fin_x=cvshort(tpx$)
    ZGET n,tpx$,2: dev(k).fin_y=cvshort(tpx$)
    ZGET n,tpx$,4: dev(k).off_x=cvs(tpx$)
    ZGET n,tpx$,4: dev(k).off_y=cvs(tpx$)
    ZGET n,tpx$,4: dev(k).mass =cvs(tpx$)
    ZGET n,tpx$,4: dev(k).fric =cvs(tpx$)
    ZGET n,tpx$,4: dev(k).size =cvs(tpx$)
    ZGET n,tpx$,2: dev(k).delta=cvshort(tpx$)
    ZGET n,tpx$,1: dev(k).cntrl=tpx$
  case "LST":
    seek #n,66&*(k-1)+34+36
    ZGET n,tpx$,2: lst(k).lst_n=cvshort(tpx$)
    ZGET n,tpx$,2: lst(k).dev_n=cvshort(tpx$)
    ZGET n,tpx$,1: lst(k).cntrl=tpx$
  case "PRC":
    seek #n,66&*(k-1)+39+36
    ZGET n,tpx$,2: prc(k).prc_n=cvshort(tpx$)
    ZGET n,tpx$,2: prc(k).dev_n=cvshort(tpx$)
    ZGET n,tpx$,2: prc(k).stp_n=cvshort(tpx$)
    ZGET n,tpx$,8: prc(k).text =tpx$
    ZGET n,tpx$,2: prc(k).move =cvshort(tpx$)
    ZGET n,tpx$,2: prc(k).ini_x=cvshort(tpx$)
    ZGET n,tpx$,2: prc(k).ini_y=cvshort(tpx$)
    ZGET n,tpx$,2: prc(k).fin_x=cvshort(tpx$)
    ZGET n,tpx$,2: prc(k).fin_y=cvshort(tpx$)
    ZGET n,tpx$,2: prc(k).t_cnt=cvshort(tpx$)
    ZGET n,tpx$,2: prc(k).w_cnt=cvshort(tpx$)
  end select
end sub

sub save(n,k,CTRL$)
  select case CTRL$
  case "IDXA":
    seek #n,1
    ZPUT n,mkshort$(idxA.a_record),2
    ZPUT n,mkshort$(idxA.r_record),2
    ZPUT n,mkshort$(idxA.m_record),2
```

```
  case "IDXB":
    seek #n,42&*(k-1)+10
    ZPUT n,idxB(k).fmfdsa,9
    ZPUT n,idxB(k).fname,32
    ZPUT n,idxB(k).cntrl,1
  case "MFDSA":
    seek #n,1
    ZPUT n,mkshort$(mfdsa(k).acode),2
    ZPUT n,mkshort$(mfdsa(k).zcode),2
    ZPUT n,mkshort$(mfdsa(k).n),    2
    ZPUT n,mkshort$(mfdsa(k).m),    2
    ZPUT n,mkshort$(mfdsa(k).t),    2
    ZPUT n,mkshort$(mfdsa(k).t_dev),2
    ZPUT n,mkshort$(mfdsa(k).t_lst),2
    ZPUT n,mkshort$(mfdsa(k).t_prc),2
    ZPUT n,mkshort$(mfdsa(k).t_stp),2
    ZPUT n,mfdsa(k).c_dte,         10
    ZPUT n,mfdsa(k).c_tme,          8
  case "DEV":
    seek #n,66&*(k-1)+1+36
    ZPUT n,mkshort$(dev(k).dev_n), 2
    ZPUT n,mkshort$(dev(k).ini_x), 2
    ZPUT n,mkshort$(dev(k).ini_y), 2
    ZPUT n,mkshort$(dev(k).fin_x), 2
    ZPUT n,mkshort$(dev(k).fin_y), 2
    ZPUT n,mks$(dev(k).off_x),     4
    ZPUT n,mks$(dev(k).off_y),     4
    ZPUT n,mks$(dev(k).mass),      4
    ZPUT n,mks$(dev(k).fric),      4
    ZPUT n,mks$(dev(k).size),      4
    ZPUT n,mkshort$(dev(k).delta), 2
    ZPUT n,dev(k).cntrl,           1
  case "LST":
    seek #n,66&*(k-1)+34+36
    ZPUT n,mkshort$(lst(k).lst_n), 2
    ZPUT n,mkshort$(lst(k).dev_n), 2
    ZPUT n,lst(k).cntrl,           1
  case "PRC":
    seek #n,66&*(k-1)+39+36
    ZPUT n,mkshort$(prc(k).prc_n), 2
    ZPUT n,mkshort$(prc(k).dev_n), 2
    ZPUT n,mkshort$(prc(k).stp_n), 2
    ZPUT n,prc(k).text,            8
    ZPUT n,mkshort$(prc(k).move),  2
    ZPUT n,mkshort$(prc(k).ini_x), 2
    ZPUT n,mkshort$(prc(k).ini_y), 2
    ZPUT n,mkshort$(prc(k).fin_x), 2
    ZPUT n,mkshort$(prc(k).fin_y), 2
    ZPUT n,mkshort$(prc(k).t_cnt), 2
    ZPUT n,mkshort$(prc(k).w_cnt), 2
  end select
end sub
```

```
sub sift(u1,u2)
  IDX(u2,0)=0
  for z0=1 to IDX(u1,0)
    if (HASH0$( 0)="") or (HASH0$( 0)<>"" and prc(IDX(u1,
     z0)).prc_n=cvshort(HASH0$( 0))) then
      if (HASH0$( 1)="") or (HASH0$( 1)<>"" and prc(IDX(u1,
       z0)).dev_n=cvshort(HASH0$( 1))) then
        if (HASH0$( 2)="") or (HASH0$( 2)<>"" and prc(IDX(u1,
         z0)).stp_n=cvshort(HASH0$( 2))) then
          if (HASH0$( 3)="") or (HASH0$( 3)<>"" and
           prc(IDX(u1,z0)).text=HASH0$( 3)) then
            if (HASH0$( 4)="") or (HASH0$( 4)<>"" and
             prc(IDX(u1,z0)).move=cvshort(HASH0$( 4))) then
              if (HASH0$( 5)="") or (HASH0$( 5)<>"" and
               prc(IDX(u1,z0)).ini_x=cvshort(HASH0$( 5))) then
                if (HASH0$( 6)="") or (HASH0$( 6)<>"" and
                 prc(IDX(u1,z0)).ini_y=cvshort(HASH0$( 6)))
                 then
                  if (HASH0$( 7)="") or (HASH0$( 7)<>"" and
                   prc(IDX(u1,z0)).fin_x=cvshort(HASH0$( 7)))
                   then
                    if (HASH0$( 8)="") or (HASH0$( 8)<>"" and
                     prc(IDX(u1,z0)).fin_y=cvshort(HASH0$(8)))
                     then
                      if (HASH0$( 9)="") or (HASH0$( 9)<>""
                       and prc(IDX(u1,z0)).t_cnt=
                       cvshort(HASH0$( 9))) then
                        if (HASH0$(10)="") or (HASH0$(10)<>""
                         and prc(IDX(u1,z0)).w_cnt=
                         cvshort(HASH0$(10))) then
                          IDX(u2,0)=IDX(u2,0)+1
                          IDX(u2,IDX(u2,0))=IDX(u1,z0)
                        end if
                      end if
                    end if
                  end if
                end if
              end if
            end if
          end if
        end if
      end if
    end if
  next z0
end sub

sub sort(u)
  zgap=IDX(u,0) \ 2
  do
    CTRL=FAIL
    zmax=IDX(u,0)
    do
      CTRL=PASS
      for z0=1 to zmax-zgap
        tpa$=ZSORT$(idxB(IDX(u,z0)).fname)+
         ZSORT$(idxB(IDX(u,z0)).fmfdsa)
```

```
        tpb$=ZSORT$(idxB(IDX(u,z0+zgap)).fname)+
         ZSORT$(idxB(IDX(u,z0+zgap)).fmfdsa)
        if tpa$>tpb$ then
          CTRL=FAIL
          zmin=z0
          swap IDX(u,z0),IDX(u,z0+zgap)
        end if
      next z0
      zmax=zmin
    loop until CTRL<>FAIL
    zgap=zgap \ 2
  loop until zgap<=0
end sub

function free()
  if idxA.a_record<>idxA.r_record then
    for z0=1 to idxA.a_record
      if idxB(z0).cntrl="*" then
        idxA.a_record=idxA.a_record+0
        idxA.r_record=idxA.r_record+1
        return z0
      end if
    next z0
  else
    if idxA.a_record<>idxA.m_record then
      idxA.a_record=idxA.a_record+1
      idxA.r_record=idxA.r_record+1
      return idxA.a_record
    end if
  end if
  return 0
end function

function mkcda$(CTRL)
  select case CTRL
  case FAIL: return "READONLY"
  case PASS: return " INSERT "
  end select
end function

function mkcdz$(CTRL)
  select case CTRL
  case FAIL: return "FAIL"
  case PASS: return "PASS"
  end select
end function

function mkdrp$(CTRL)
  select case CTRL
  case FAIL: return "Y"
  case PASS: return "N"
  end select
end function
```

```
function mkgnw$(CTRL)
  select case CTRL
  case 1: return "WINDOW"
  case 2: return " GRID "
  end select
end function

function pack(n,tpx$,zmax,CTRL$)
  select case CTRL$
  case "MFDSA":
    tpx=free
    if tpx<>0 then
      idxB(tpx).fmfdsa="MFDSA"+ZTEXT$(str$(tpx),"0",5,"R")
      idxB(tpx).fname=tpx$
      idxB(tpx).cntrl="#"
      save n,tpx,"IDXB"
      mfdsa(tpx).acode=PASS
      mfdsa(tpx).zcode=FAIL
      mfdsa(tpx).n   =10
      mfdsa(tpx).m   =0
      mfdsa(tpx).t   =0
      mfdsa(tpx).t_dev=0
      mfdsa(tpx).t_lst=0
      mfdsa(tpx).t_prc=0
      mfdsa(tpx).t_stp=0
      mfdsa(tpx).c_dte=date$
      mfdsa(tpx).c_tme=time$
      ZCMD 100,idxB(tpx).fmfdsa,"B",PASS
      save 100,tpx,"MFDSA"
      ZCMD 100,"","",FAIL
    end if
  case "DEV","LST","PRC":
    tpx=ZCOMP(zmax<>32767,zmax+1,0)
    if tpx<>0 then
      select case CTRL$
      case "DEV":
        dev(tpx).dev_n=tpx
        dev(tpx).ini_x=0
        dev(tpx).ini_y=0
        dev(tpx).fin_x=0
        dev(tpx).fin_y=0
        dev(tpx).off_x=0
        dev(tpx).off_y=0
        dev(tpx).mass=0
        dev(tpx).fric=0
        dev(tpx).size=0
        dev(tpx).delta=0
        dev(tpx).cntrl="*"
        save n,tpx,"DEV"
        zmax=tpx
      case "LST":
        lst(tpx).lst_n=tpx
        lst(tpx).dev_n=0
        lst(tpx).cntrl="*"
        save n,tpx,"LST"
        zmax=tpx
```

```
      case "PRC":
        prc(tpx).prc_n=tpx
        prc(tpx).dev_n=0
        prc(tpx).stp_n=0
        prc(tpx).text=tpx$
        prc(tpx).move=0
        prc(tpx).ini_x=0
        prc(tpx).ini_y=0
        prc(tpx).fin_x=0
        prc(tpx).fin_y=0
        prc(tpx).t_cnt=0
        prc(tpx).w_cnt=0
        save n,tpx,"PRC"
        zmax=tpx
      end select
    end if
  end select
  return tpx
end function
```

CMPLIB Module:

```
sub build_ptable(j)
  ptable(LR).leng=stack(j).fin_x-stack(j).cur_x
  if ptable(LR).leng=0 then ptable(LR).null=PASS else
   ptable(LR).null=FAIL
  if (stack(j).cur_y=0) or (stack(j).cur_y=g+1) then
    ptable(LR).blck=PASS
  else
    ptable(LR).blck=FAIL
    for y0=stack(j).cur_y-stack(j).delta to stack(j).cur_y+
     stack(j).delta
      for x0=stack(j).cur_x+sgn(ptable(LR).leng)*
       (stack(j).delta+1) to stack(j).fin_x step
       sgn(ptable(LR).leng)
        if grid(x0,y0)<>GF then ptable(LR).blck=PASS: exit
         for
      next x0
    next y0
  end if

  ptable(UD).leng=stack(j).fin_y-stack(j).cur_y
  if ptable(UD).leng=0 then ptable(UD).null=PASS else
   ptable(UD).null=FAIL
  if (stack(j).cur_x=0) or (stack(j).cur_x=g+1) then
    ptable(UD).blck=PASS
  else
    ptable(UD).blck=FAIL
    for x0=stack(j).cur_x-stack(j).delta to stack(j).cur_x+
     stack(j).delta
      for y0=stack(j).cur_y+sgn(ptable(UD).leng)*
       (stack(j).delta+1) to stack(j).fin_y step
       sgn(ptable(UD).leng)
        if grid(x0,y0)<>GF then ptable(UD).blck=PASS: exit
         for
      next y0
    next x0
  end if
end sub

sub build_qtable(j)
  qtable(UP).upd_x=stack(j).cur_x
  qtable(UP).upd_y=stack(j).cur_y+1
  if (abs(grid(qtable(UP).upd_x,qtable(UP).upd_y))=XK) or
   (qtable(UP).upd_y>g+1) then qtable(UP).forb=PASS else
   qtable(UP).forb=FAIL
  qtable(UP).blck=FAIL
  for z0=qtable(UP).upd_x-stack(j).delta to qtable(UP).upd_x
  +stack(j).delta
    if (abs(grid(z0,qtable(UP).upd_y))=XK) or (grid(z0,
    qtable(UP).upd_y)=GB) then qtable(UP).blck=PASS: exit for
  next z0
  if stack(j).move=DN then qtable(UP).move=PASS else
   qtable(UP).move=FAIL
  qtable(UP).dense=dense(j,qtable(UP).upd_x,qtable(UP).upd_y)
```

```
  qtable(DN).upd_x=stack(j).cur_x
  qtable(DN).upd_y=stack(j).cur_y-1
  if (abs(grid(qtable(DN).upd_x,qtable(DN).upd_y))=XK) or
   (qtable(DN).upd_y<0) then qtable(DN).forb=PASS else
   qtable(DN).forb=FAIL
  qtable(DN).blck=FAIL
  for z0=qtable(DN).upd_x-stack(j).delta to qtable(DN).upd_x
  +stack(j).delta
    if (abs(grid(z0,qtable(DN).upd_y))=XK) or (grid(z0,
     qtable(DN).upd_y)=GB) then qtable(DN).blck=PASS: exit
     for
  next z0
  if stack(j).move=UP then qtable(DN).move=PASS else
   qtable(DN).move=FAIL
  qtable(DN).dense=dense(j,qtable(DN).upd_x,qtable(DN).upd_y)

  qtable(LT).upd_x=stack(j).cur_x-1
  qtable(LT).upd_y=stack(j).cur_y
  if (abs(grid(qtable(LT).upd_x,qtable(LT).upd_y))=XK) or
   (qtable(LT).upd_x<0) then qtable(LT).forb=PASS else
   qtable(LT).forb=FAIL
  qtable(LT).blck=FAIL
  for z0=qtable(LT).upd_y-stack(j).delta to qtable(LT).upd_y
  +stack(j).delta
    if (abs(grid(qtable(LT).upd_x,z0))=XK) or
     (grid(qtable(LT).upd_x,z0)=GB) then qtable(LT).blck=
     PASS: exit for
  next z0
  if stack(j).move=RT then qtable(LT).move=PASS else
   qtable(LT).move=FAIL
  qtable(LT).dense=dense(j,qtable(LT).upd_x,qtable(LT).upd_y)

  qtable(RT).upd_x=stack(j).cur_x+1
  qtable(RT).upd_y=stack(j).cur_y
  if (abs(grid(qtable(RT).upd_x,qtable(RT).upd_y))=XK) or
   (qtable(RT).upd_x>g+1) then qtable(RT).forb=PASS else
  qtable(RT).forb=FAIL
  qtable(RT).blck=FAIL
  for z0=qtable(RT).upd_y-stack(j).delta to qtable(RT).upd_y
  +stack(j).delta
    if (abs(grid(qtable(RT).upd_x,z0))=XK) or
     (grid(qtable(RT).upd_x,z0)=GB) then qtable(RT).blck=
     PASS: exit for
  next z0
  if stack(j).move=LT then qtable(RT).move=PASS else
   qtable(RT).move=FAIL
  qtable(RT).dense=dense(j,qtable(RT).upd_x,qtable(RT).upd_y)
end sub
```

```
sub grid_000()
  erase grid
  for x0=0 to g+1
    for y0=0 to g+1
      grid(x0,y0)=XF
      if x0>=1 and x0<=g then
        if y0>=1 and y0<=g then
          grid(x0,y0)=GF
        end if
      end if
    next y0
  next x0
end sub

sub grid_add(j)
  for x0=stack(j).cur_x-stack(j).delta to stack(j).cur_x+
   stack(j).delta
    for y0=stack(j).cur_y-stack(j).delta to stack(j).cur_y+
     stack(j).delta
      if x0>=0 and x0<=g+1 then
        if y0>=0 and y0<=g+1 then
          if grid(x0,y0)=XF then grid(x0,y0)=XB
          if grid(x0,y0)=GF then grid(x0,y0)=GB
        end if
      end if
    next y0
  next x0
end sub

sub grid_sub(j)
  for x0=stack(j).cur_x-stack(j).delta to stack(j).cur_x+
   stack(j).delta
    for y0=stack(j).cur_y-stack(j).delta to stack(j).cur_y+
     stack(j).delta
      if x0>=0 and x0<=g+1 then
        if y0>=0 and y0<=g+1 then
          if grid(x0,y0)=XB then grid(x0,y0)=XF
          if grid(x0,y0)=GB then grid(x0,y0)=GF
        end if
      end if
    next y0
  next x0
end sub

sub grid_upd()
  grid_000
  for j=1 to max_dev
    if stack(j).cntrl=PASS then
      grid_add j
    end if
  next j
end sub
```

```
sub stack_add(n,stp,t_cnt,w_cnt)
  for j=1 to max_dev
    if stack(j).cntrl<>PASS then
      if test(lst(IDX(2,cur_dev+1)).dev_n)<>FAIL then
        if cur_dev<tot_dev then
          cur_dev=cur_dev+1
          stack(j).dev_n=lst(IDX(2,cur_dev)).dev_n
          stack(j).dev_s=symbol$(lst(IDX(2,cur_dev)).dev_n)
          stack(j).delta=dev(lst(IDX(2,cur_dev)).dev_n).delta
          stack(j).move=NL
          stack(j).cur_x=dev(lst(IDX(2,cur_dev)).dev_n).ini_x
          stack(j).cur_y=dev(lst(IDX(2,cur_dev)).dev_n).ini_y
          stack(j).fin_x=dev(lst(IDX(2,cur_dev)).dev_n).fin_x
          stack(j).fin_y=dev(lst(IDX(2,cur_dev)).dev_n).fin_y
          stack(j).cntrl=PASS
          zrecord_prc n,stp,j,"*!INJT!*",t_cnt,w_cnt
          exit for
        end if
      end if
    end if
  next j
end sub

sub stack_sub(j)
  stack(j).dev_n=0
  stack(j).dev_s=""
  stack(j).delta=0
  stack(j).move=NL
  stack(j).cur_x=0
  stack(j).cur_y=0
  stack(j).fin_x=0
  stack(j).fin_y=0
  stack(j).cntrl=FAIL
end sub

function dense(j,cur_x,cur_y)
  tpx=0
  for x0=cur_x-(stack(j).delta+1) to cur_x+
   (stack(j).delta+1)
    for y0=cur_y-(stack(j).delta+1) to cur_y+
     (stack(j).delta+1)
      if x0>=0 and x0<=g+1 then
        if y0>=0 and y0<=g+1 then
          tpx=tpx+grid(x0,y0) ^ 2
        end if
      end if
    next y0
  next x0
  return tpx
end function
```

```
function scan_ptable()
  tpx=NL
  for z0=LR to UD
    if ptable(z0).null=FAIL then
      if ptable(z0).blck=FAIL then
        tpx=tpx+z0
      end if
    end if
  next z0
  if tpx=LD then if abs(ptable(LR).leng)<abs(ptable(UD).leng)
   then tpx=LR else tpx=UD
  return tpx
end function

function scan_qtable()
  tpx=NL
  for z0=UP to RT
    if qtable(z0).forb=FAIL then
      if qtable(z0).blck=FAIL then
        if qtable(z0).move=FAIL then
          if (tpx=NL) or (tpx<>NL and qtable(z0).dense<
           qtable(tpx).dense) then tpx=z0
        end if
      end if
    end if
  next z0
  return tpx
end function

function switch(CTRL)
  tpx=NL
  select case CTRL
  case LR: if ptable(CTRL).leng<0 then tpx=LT else tpx=RT
  case UD: if ptable(CTRL).leng<0 then tpx=DN else tpx=UP
  end select
  return tpx
end function

function symbol$(CTRL)
  return chr$(asc("A")+((CTRL mod 26)-1))
end function
```

```
function test(CTRL)
  tpx=PASS
  for x0=dev(CTRL).ini_x-dev(CTRL).delta to dev(CTRL).ini_x+
   dev(CTRL).delta
    for y0=dev(CTRL).ini_y-dev(CTRL).delta to dev(CTRL).ini_y
    +dev(CTRL).delta
      if x0>=0 and x0<=g+1 then
        if y0>=0 and y0<=g+1 then
          if (grid(x0,y0)=XB) or (grid(x0,y0)=GB) then tpx
         =FAIL: exit for
        end if
      end if
    next y0
  next x0
  return tpx
end function

function text$(CTRL)
  tpx$="NL"
  select case CTRL
  case UP: tpx$="UP"
  case DN: tpx$="DN"
  case LT: tpx$="LT"
  case RT: tpx$="RT"
  end select
  return tpx$
end function

sub proj_grid(CTRL)
  erase grid
  for j=1 to CTRL
    for x0=stack(j).cur_x-stack(j).delta to stack(j).cur_x+
     stack(j).delta
      for y0=stack(j).cur_y-stack(j).delta to stack(j).cur_y
      +stack(j).delta
        grid(x0,y0)=j
      next y0
    next x0
  next j
end sub

sub proj_trace(k,x0,y0)
  if grid(x0,y0)<>0 then
    for z0=1 to mfdsa(k).t_prc
      if prc(z0).dev_n=stack(grid(x0,y0)).dev_n then
        if grid(prc(z0).fin_x,prc(z0).fin_y)=0 then
          grid(prc(z0).fin_x,prc(z0).fin_y)=grid(x0,y0)
        end if
      end if
    next z0
  end if
end sub
```

```
function proj_stack(k,CTRL)
  tpx=0
  for z0=1 to mfdsa(k).t_prc
    if prc(z0).stp_n=CTRL then
      select case prc(z0).text
      case "MOVEMENT":
        tpx=tpx+1
        stack(tpx).dev_n=prc(z0).dev_n
        stack(tpx).dev_s=symbol$(prc(z0).dev_n)
        stack(tpx).delta=dev(prc(z0).dev_n).delta
        stack(tpx).move=prc(z0).move
        stack(tpx).cur_x=prc(z0).fin_x
        stack(tpx).cur_y=prc(z0).fin_y
        stack(tpx).fin_x=dev(prc(z0).dev_n).fin_x
        stack(tpx).fin_y=dev(prc(z0).dev_n).fin_y
        stack(tpx).cntrl=PASS
      case "*!DROP!*":
        stack(tpx).cntrl=FAIL
      end select
    end if
  next z0
  return tpx
end function
```

GUILIB Module:

```
sub prompt_list_A(tpx$,tpa,tpb)
  ZWINDOWM "MFDSA-Swarm Algorithm,v3   //   List: "+tpx$,"",
   "" '"Error/Message"
  ZWINDOWS "Lst #",5,3,40,10,tpa
  ZWINDOWS "Dev #   (ini-X,ini-Y)<=>(fin-X,fin-Y)/(off-X,
   off-Y)         Mass         Fric         Size     Delta",5,
   14,40,113,tpb
end sub

sub prompt_list_B(tpx$)
  ZWINDOWM "MFDSA-Swarm Algorithm,v3   //   List: "+tpx$,"",
   "" '"Error/Message"
  ZWINDOWS "  Prc #      Dev #      Stp #      Instruct      MV
   (ini-X,ini-Y)  <=>  (fin-X,fin-Y)      T-Count     W-
   Count",5,3,40,124,WIRE
end sub

sub prompt_main(tpx$)
  ZWINDOWM "MFDSA-Swarm Algorithm,v3   //   Main: "+tpx$,"",
   "" '"Error/Message"
  ZWINDOWS "Project(s)",5,3,40,36,ACT1
  ZWINDOWS "Summary",   5,40,40,87,ACT2
  ZPRINT "  Project  Information ",9,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT "======================",10,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT " File MFDSA-         ",12,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT " File Name-          ",13,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT " File Mode-          ",14,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT " Comp Date-          ",16,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT " Comp Time-          ",17,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT " Comp Stat-          ",18,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT " Simulator Information ",22,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT "======================",23,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT "  Grid nxn-          ",25,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT "   Max Dev-          ",26,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT "   Trg Dev-          ",27,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT "   Misc.   Information ",31,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT "======================",32,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT " Total Dev-          ",34,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
```

```
  ZPRINT " Total Lst-          ",35,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT " Total Prc-          ",36,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
  ZPRINT " Total Stp-          ",37,42,RGBA0(WIRE,2),
   RGBA0(WIRE,1)
end sub

sub prompt_project(tpx$)
  ZWINDOWM "MFDSA-Swarm Algorithm,v3   //   Project: "+tpx$,
   "","" '"Error/Message"
  ZWINDOWS " Project -Grid/View",5,3,40,40,ACT1
  ZWINDOWS "Detail(s)-Simulator",5,44,20,83,ACT2
  ZWINDOWS "Detail(s)-  Misc.  ",25,44,20,83,ACT2
  ZWINDOW_G 8,32,4,40
  ZPRINT "Grid Size (n,n)  [    ,   ]",34,6,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT "Cur. Pos. (X,Y)  [    ,   ]",36,6,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT "Cur. Stp.        [    /    ]",38,6,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT "Display Mode:                ",41,6,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT "  Project  Information       ",9,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT "======================      ",10,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT " File MFDSA-                ",11,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT " File Name-                 ",12,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT " File Mode-                 ",13,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT " Comp Date-                 ",14,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT " Comp Time-                 ",15,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT " Comp Stat-                 ",16,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT " Simulator Information       ",18,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT "======================      ",19,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT "  Grid nxn-                 ",20,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT "   Max Dev-                 ",21,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT "   Trg Dev-                 ",22,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT "    Device-      [Symbol: ] ",29,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT " Cur. Pos.-[    ,    ]   ",31,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT " Ini. Pos.-[    ,    ]   ",32,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT " Fin. Pos.-[    ,    ]   ",33,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
```

```
  ZPRINT " Off. Pos.-[    ,    ]   ",34,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT "      Mass-                ",36,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT "      Fric-                ",37,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT "      Size-                ",38,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT "     Delta-                ",39,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
  ZPRINT "  Dropped?-                ",41,46,RGBA0(WIRE,
   2),RGBA0(WIRE,1)
end sub

sub record_list_A1(f0,f1,f2)
  for z0=f1 to f2
    if z0=f0 then color RGBA0(MAIN,2),RGBA0(MAIN,1)
    locate 8+(z0-f1),4
    print using " #####! "; lst(z0).dev_n; lst(z0).cntrl
    if z0=f0 then color RGBA0(MAIN,1),RGBA0(MAIN,2)
  next z0
end sub

sub record_list_A2(f0,f1,f2)
  for z0=f1 to f2
    if z0=f0 then color RGBA0(MAIN,2),RGBA0(MAIN,1)
    locate 8+(z0-f1),15
    print using " #####!    ####_,####        ####_,####
     #.###_,#.###   ##.####^^^^  ##.####^^^^  ##.####^^^^
     ####   "; dev(z0).dev_n; dev(z0).cntrl; dev(z0).ini_x;
     dev(z0).ini_y; dev(z0).fin_x; dev(z0).fin_y; dev(z0).off_x;
     dev(z0).off_y; dev(z0).mass; dev(z0).fric; dev(z0).size;
     dev(z0).delta
    if z0=f0 then color RGBA0(MAIN,1),RGBA0(MAIN,2)
  next z0
end sub

sub record_list_B(f0,f1,f2,u)
  for z0=f1 to f2
    if z0=f0 then color RGBA0(MAIN,2),RGBA0(MAIN,1)
    locate 8+(z0-f1),4
    print using "   #####      #####      #####      \      \
     \ \       ####_,####            ####_,####         #####
     #####  "; prc(IDX(u,z0)).prc_n; prc(IDX(u,z0)).dev_n;
     prc(IDX(u,z0)).stp_n; prc(IDX(u,z0)).text;
     text$(prc(IDX(u,z0)).move); prc(IDX(u,z0)).ini_x;
     prc(IDX(u,z0)).ini_y; prc(IDX(u,z0)).fin_x; prc(IDX(u,
     z0)).fin_y; prc(IDX(u,z0)).t_cnt; prc(IDX(u,z0)).w_cnt
    if z0=f0 then color RGBA0(MAIN,1),RGBA0(MAIN,2)
  next z0
end sub
```

```
sub record_main1(f0,f1,f2,u)
  for z0=f1 to f2
    if z0=f0 then color RGBA0(MAIN,2),RGBA0(MAIN,1)
    locate 8+(z0-f1),4
    print using " \                              \ "; idxB(IDX(u,
     z0)).fname
    if z0=f0 then color RGBA0(MAIN,1),RGBA0(MAIN,2)
  next z0
end sub

sub record_main2(k)
  locate 12,57
  print using "\       \"; idxB(k).fmfdsa
  locate 13,57
  print using "\                                \"; idxB(k).fname
  locate 14,57
  print using "\      \"; mkcda$(mfdsa(k).acode)
  locate 16,57
  print using "\        \"; mfdsa(k).c_dte
  locate 17,57
  print using "\      \"; mfdsa(k).c_tme
  locate 18,57
  print using "\  \";  mkcdz$(mfdsa(k).zcode)
  locate 25,57
  print using "#####"; mfdsa(k).n
  locate 26,57
  print using "#####"; mfdsa(k).m
  locate 27,57
  print using "#####"; mfdsa(k).t
  locate 34,57
  print using "#####"; mfdsa(k).t_dev
  locate 35,57
  print using "#####"; mfdsa(k).t_lst
  locate 36,57
  print using "#####"; mfdsa(k).t_prc
  locate 37,57
  print using "#####"; mfdsa(k).t_stp
end sub

sub record_project1(k,t0,f0,f1,f2,g0,g1,g2,CTRL)
  color RGBA0(MAIN,1),RGBA0(MAIN,2)
  for z0=f1 to f2
    for a0=g1 to g2
      select case grid(z0,a0)
      case is<>0:
        color 0,10
        if z0=f0 and a0=g0 then color 0,14
        locate (31-(a0-g1)*2),(5+(z0-f1)*3)
        print stack(grid(z0,a0)).dev_s+chr$(asc("*")*
         (stack(grid(z0,a0)).cntrl+1));
        if z0=f0 and a0=g0 then color 0,10
```

```
      case is=0:
        color 0,15
        if z0=f0 and a0=g0 then color 0,14
        locate (31-(a0-g1)*2),(5+(z0-f1)*3)
        print "  ";
        if z0=f0 and a0=g0 then color 0,15
      end select
      if ((z0=0 or z0=mfdsa(k).n+1) and (a0>=0 and a0<=
       mfdsa(k).n+1)) or ((a0=0 or a0=mfdsa(k).n+1) and (z0
      >=0 and z0<=mfdsa(k).n+1)) then
        color 4,4
        if z0=f0 and a0=g0 then color 0,14
        locate (31-(a0-g1)*2),(5+(z0-f1)*3)
        print "  ";
        if z0=f0 and a0=g0 then color 4,4
      end if
    next a0
  next z0
  color RGBA0(MAIN,1),RGBA0(MAIN,2)
  locate 34,24
  print using "#####"; mfdsa(k).n
  locate 34,30
  print using "#####"; mfdsa(k).n
  locate 36,24
  print using "#####"; f0
  locate 36,30
  print using "#####"; g0
  locate 38,24
  print using "#####"; t0
  locate 38,30
  print using "#####"; mfdsa(k).t_stp
  locate 41,20
  print using "\    \"; mkgnw$(CTRL)
end sub

sub record_project2(k)
  locate 11,59
  print using "\       \"; idxB(k).fmfdsa
  locate 12,59
  print using "\                                \"; idxB(k).fname
  locate 13,59
  print using "\      \"; mkcda$(mfdsa(k).acode)
  locate 14,59
  print using "\        \"; mfdsa(k).c_dte
  locate 15,59
  print using "\      \"; mfdsa(k).c_tme
  locate 16,59
  print using "\  \";  mkcdz$(mfdsa(k).zcode)
  locate 20,59
  print using "#####"; mfdsa(k).n
  locate 21,59
  print using "#####"; mfdsa(k).m
  locate 22,59
  print using "#####"; mfdsa(k).t
end sub
```

```
sub record_project3(k)
  locate 29,59
  print using "#####"; stack(k).dev_n
  locate 29,73
  print using "!"; stack(k).dev_s
  locate 31,60
  print using "#####"; stack(k).cur_x
  locate 31,67
  print using "#####"; stack(k).cur_y
  locate 32,60
  print using "#####"; dev(stack(k).dev_n).ini_x
  locate 32,67
  print using "#####"; dev(stack(k).dev_n).ini_y
  locate 33,60
  print using "#####"; stack(k).fin_x
  locate 33,67
  print using "#####"; stack(k).fin_y
  locate 34,60
  print using "#.###"; dev(stack(k).dev_n).off_x
  locate 34,67
  print using "#.###"; dev(stack(k).dev_n).off_y
  locate 36,59
  print using "##.####^^^^"; dev(stack(k).dev_n).mass
  locate 37,59
  print using "##.####^^^^"; dev(stack(k).dev_n).fric
  locate 38,59
  print using "##.####^^^^"; dev(stack(k).dev_n).size
  locate 39,59
  print using "#####"; stack(k).delta
  locate 41,59
  print using "!"; mkdrp$(stack(k).cntrl)
end sub
```

MODLIB Module:

```
sub export_devl(n,u)
  print #n,""
  print #n," Dev #  (ini-X,ini-Y)<=>(fin-X,fin-Y)/(off-X,
   off-Y)         Mass         Fric         Size     Delta "
  print #n,"-----------------------------------------------------
   -----------------------------------------------------"
  for z0=1 to IDX(u,0)
    print #n,using " #####    ####_,####        ####_,####
     #.###_,#.###   ##.####^^^^  ##.####^^^^  ##.####^^^^
     ####   "; dev(IDX(u,z0)).dev_n; dev(IDX(u,z0)).ini_x;
     dev(IDX(u,z0)).ini_y; dev(IDX(u,z0)).fin_x; dev(IDX(u,
     z0)).fin_y; dev(IDX(u,z0)).off_x; dev(IDX(u,z0)).off_y;
     dev(IDX(u,z0)).mass; dev(IDX(u,z0)).fric; dev(IDX(u,
     z0)).size; dev(IDX(u,z0)).delta
  next z0
  print #n,"-----------------------------------------------------
   -----------------------------------------------------"
  print #n,""
end sub

sub export_lstl(n,u)
  print #n,""
  print #n," Lst #   Dev # "
  print #n,"---------------"
  for z0=1 to IDX(u,0)
    print #n,using " #####   ##### "; lst(IDX(u,z0)).lst_n;
     lst(IDX(u,z0)).dev_n
  next z0
  print #n,"---------------"
  print #n,""
end sub

sub export_prcl(n,u)
  print #n,""
  print #n,"  Prc #      Dev #      Stp #      Instruct      MV
   (ini-X,ini-Y)  <=>  (fin-X,fin-Y)      T-Count     W-
   Count "
  print #n,"-----------------------------------------------------
   -------------------------------------------------------------
   ------"
  for z0=1 to IDX(u,0)
    print #n,using "  #####      #####      #####      \      \
     \ \       ####_,####            ####_,####         #####
     ##### "; prc(IDX(u,z0)).prc_n; prc(IDX(u,z0)).dev_n;
     prc(IDX(u,z0)).stp_n; prc(IDX(u,z0)).text;
     text$(prc(IDX(u,z0)).move); prc(IDX(u,z0)).ini_x;
     prc(IDX(u,z0)).ini_y; prc(IDX(u,z0)).fin_x; prc(IDX(u,
     z0)).fin_y; prc(IDX(u,z0)).t_cnt; prc(IDX(u,z0)).w_cnt
  next z0
  print #n,"-----------------------------------------------------
   -------------------------------------------------------------
   ------"
  print #n,""
end sub
```

```
sub export_raw(n,k,u1,u2,u3)
  write #n,mfdsa(k).n,mfdsa(k).m,mfdsa(k).t,IDX(u1,0),IDX(u2,0)
  for z0=1 to IDX(u1,0)
    print #n,dev(IDX(u1,z0)).ini_x,dev(IDX(u1,z0)).ini_y,
     dev(IDX(u1,z0)).fin_x,dev(IDX(u1,z0)).fin_y,dev(IDX(u1,
     z0)).off_x,dev(IDX(u1,z0)).off_y,dev(IDX(u1,z0)).mass,
     dev(IDX(u1,z0)).fric,dev(IDX(u1,z0)).size,dev(IDX(u1,
     z0)).delta
  next z0
  for z0=1 to IDX(u2,0)
    print #n,lst(IDX(u2,z0)).dev_n
  next z0
end sub

sub export_stat(n,k,u1,u2,u3)
  print #n,""
  print #n," Dev #   N.Lng   C.Lng   C.Stp   C/N Ratio   n^2
   Ratio   MAX Ratio      Vel.      Drg.      Wait "
  print #n,"-------------------------------------------------------
   ------------------------------------------"
  tot_n_lng=0
  tot_c_lng=0
  tot_c_stp=0
  tot_c_nll=0
  for z0=1 to IDX(u1,0)
    n_lng=abs(dev(IDX(u1,z0)).fin_x-dev(IDX(u1,z0)).ini_x)
    +abs(dev(IDX(u1,z0)).fin_y-dev(IDX(u1,z0)).ini_y)
    c_lng=0
    c_stp=0
    c_nll=0
    for a0=1 to IDX(u3,0)
      if prc(IDX(u3,a0)).dev_n=dev(IDX(u1,z0)).dev_n then
        select case prc(IDX(u3,a0)).text
        case "MOVEMENT":
          select case prc(IDX(u3,a0)).move
          case is<>NL: c_lng=c_lng+1
          case is=NL: c_nll=c_nll+1
          end select
          c_stp=c_stp+1
        case "*!DROP!*":
          exit for
        end select
      end if
    next a0
    tot_n_lng=tot_n_lng+n_lng
    tot_c_lng=tot_c_lng+c_lng
    tot_c_stp=tot_c_stp+c_stp
    tot_c_nll=tot_c_nll+c_nll
    ratio_1!=c_lng/n_lng
    ratio_n!=ratio_1!/mfdsa(k).n/mfdsa(k).n
    ratio_m!=ratio_1!/mfdsa(k).m
    vel!=c_lng/c_stp
    drg!=c_nll/c_stp
```

```
    print #n,using " #####   #####   #####   #####     ##.####
     ##.####     ##.####    ##.###    ##.###     ##### ";
     dev(IDX(u1,z0)).dev_n; n_lng; c_lng; c_stp; ratio_1!;
     ratio_n!; ratio_m!; vel!; drg!; c_nll
  next z0
  print #n,"-------------------------------------------------------
   ------------------------------------------"
  ratio_1!=tot_c_lng/tot_n_lng
  ratio_n!=ratio_1!/mfdsa(k).n/mfdsa(k).n
  ratio_m!=ratio_1!/mfdsa(k).m
  vel!=tot_c_lng/tot_c_stp
  drg!=tot_c_nll/tot_c_stp
  print #n,using "         #####   #####   #####     ##.####
   ##.####     ##.####    ##.###    ##.###     ##### ";
   tot_n_lng; tot_c_lng; tot_c_stp; ratio_1!; ratio_n!; ratio_m!;
   vel!; drg!; tot_c_nll
  print #n,""
  print #n,using " Proc Step=>######"; mfdsa(k).t_prc
  print #n,using " Time Step=>######"; mfdsa(k).t_stp
  print #n,using " Proc/Time=>##.###"; mfdsa(k).t_prc/
   mfdsa(k).t_stp
  print #n,""
  print #n,using "  Grid nxn=>  ####"; mfdsa(k).n
  print #n,using "   Max Dev=> #####"; mfdsa(k).m
  print #n,using "   Trg Dev=> #####"; mfdsa(k).t
  print #n,""
end sub

sub import_dev(n1,n2,k,zmax)
  for z0=1 to zmax
    input #n2,ini_x,ini_y,fin_x,fin_y,off_x!,off_y!,mass!,
     fric!,size!,delta
    tpx=pack(n1,"",mfdsa(k).t_dev,"DEV")
    if tpx<>0 then
      dev(tpx).dev_n=tpx
      dev(tpx).ini_x=ini_x
      dev(tpx).ini_y=ini_y
      dev(tpx).fin_x=fin_x
      dev(tpx).fin_y=fin_y
      dev(tpx).off_x=off_x!
      dev(tpx).off_y=off_y!
      dev(tpx).mass=mass!
      dev(tpx).fric=fric!
      dev(tpx).size=size!
      dev(tpx).delta=delta
      dev(tpx).cntrl="*"
      save n1,tpx,"DEV"
    end if
  next z0
end sub
```

```
sub import_lst(n1,n2,k,zmax)
  for z0=1 to zmax
    input #n2,dev_n
    tpx=pack(n1,"",mfdsa(k).t_lst,"LST")
    if tpx<>0 then
      lst(tpx).lst_n=tpx
      lst(tpx).dev_n=dev_n
      lst(tpx).cntrl="*"
      save n1,tpx,"LST"
    end if
  next z0
end sub

function check_dev(k,dev_n)
  tpx=FAIL
  if dev_n>0 then
    if dev_n<=mfdsa(k).t_dev then
      if dev(dev_n).cntrl<>"*" then
        tpx=PASS
      end if
    end if
  end if
  return tpx
end function

function check_fin(k,cur_x,cur_y)
  tpx=FAIL
  if cur_x>0 and cur_x<mfdsa(k).n+1 then
    if cur_y>0 and cur_y<mfdsa(k).n+1 then
      tpx=PASS
    end if
  end if
  return tpx
end function

function check_gns(k1,k2,dev_n)
  tpx=PASS
  for z0=1 to mfdsa(k1).t_lst
    if z0<>k2 then
      if lst(z0).dev_n=dev_n then
        tpx=FAIL
        exit for
      end if
    end if
  next z0
  return tpx
end function
```

```
function check_ini(k,cur_x,cur_y)
  tpx=FAIL
  if cur_x=0             and (cur_y>0 and cur_y<
   mfdsa(k).n+1) then tpx=PASS
  if cur_x=mfdsa(k).n+1 and (cur_y>0 and cur_y<
   mfdsa(k).n+1) then tpx=PASS
  if cur_y=0             and (cur_x>0 and cur_x<
   mfdsa(k).n+1) then tpx=PASS
  if cur_y=mfdsa(k).n+1 and (cur_x>0 and cur_x<
   mfdsa(k).n+1) then tpx=PASS
  return tpx
end function

function check_mfsd(k,mass!,fric!,size!,delta)
  tpx=FAIL
  if mass!>=0 then
    if fric!>=0 then
      if size!>=0 then
        if delta>=0 and delta<=(mfdsa(k).n ^ (1/2)-1)/2 then
          tpx=PASS
        end if
      end if
    end if
  end if
  return tpx
end function

function check_off(off_x!,off_y!)
  tpx=FAIL
  if abs(off_x!)<1 then
    if abs(off_y!)<1 then
      tpx=PASS
    end if
  end if
  return tpx
end function

function check_sng(k1,k2,cur_x,cur_y)
  tpx=PASS
  for z0=1 to mfdsa(k1).t_dev
    if z0<>k2 then
      if dev(z0).cntrl<>"*" then
        if dev(z0).fin_x=cur_x then
          if dev(z0).fin_y=cur_y then
            tpx=FAIL
            exit for
          end if
        end if
      end if
    end if
  next z0
  return tpx
end function
```

ZCOMPILE Module:

```
sub zcompile(n,k)
  if mfdsa(k).acode<>FAIL then
    if mfdsa(k).n<>0 then
      if mfdsa(k).m<>0 then
        if mfdsa(k).t<>0 then
          if IDX(1,0)<>0 then
            if IDX(2,0)<>0 then
              if IDX(3,0)=0 then
                g     =mfdsa(k).n
                max_dev=mfdsa(k).m
                trg_dev=mfdsa(k).t
                stp   =1
                cur_dev=0
                tot_dev=IDX(2,0)
                cur_prc=0
                tot_prc=0
                t_cnt =0
                w_cnt =0
                erase stack
                for j=1 to max_dev
                  stack_add n,stp,t_cnt,w_cnt
                next j
                grid_upd
                do
                  active_dev=0
                  killed_dev=0
                  for j=1 to max_dev
                    if stack(j).cntrl=PASS then
                      active_dev=active_dev+1
                      killed_dev=killed_dev+0
                      build_ptable j
                      CTRL=scan_ptable
                      select case CTRL
                      case LR,UD:
                        stack(j).move=switch(CTRL)
                        zrecord_prc n,stp,j,"MOVEMENT",t_cnt,
                         w_cnt
                      case NL:
                        build_qtable j
                        CTRL=scan_qtable
                        select case CTRL
                        case NL,UP,DN,LT,RT:
                          stack(j).move=CTRL
                          zrecord_prc n,stp,j,"MOVEMENT",
                           t_cnt,w_cnt
                          if CTRL=NL then
                            active_dev=active_dev+0
                            killed_dev=killed_dev+1
                          end if
                        end select
                      end select
```

```
                        if stack(j).cur_x=stack(j).fin_x then
                          if stack(j).cur_y=stack(j).fin_y then
                            active_dev=active_dev-1
                            killed_dev=killed_dev*0
                            w_cnt=w_cnt+0
                            t_cnt=t_cnt+1
                            stack(j).move=NL
                            zrecord_prc n,stp,j,"*!DROP!*",
                             t_cnt,w_cnt
                            stack_sub j
                          end if
                        end if
                      end if
                    next j
                    if killed_dev=active_dev and active_dev<>0 then
                      zrecord_prc n,stp,0,"*!FAIL!*",t_cnt,
                       w_cnt
                      mfdsa(k).zcode=FAIL
                      exit do
                    end if
                    if w_cnt=tot_dev and tot_dev<>0 then
                      zrecord_prc n,stp,0,"*!PASS!*",t_cnt,
                       w_cnt
                      mfdsa(k).zcode=PASS
                      exit do
                    end if
                    if active_dev=0 then
                      w_cnt=w_cnt+t_cnt
                      t_cnt=t_cnt*0
                      zrecord_prc n,stp,-1,"*!ERR.!*",t_cnt,
                       w_cnt
                      zrecord_prc n,stp,-1,"*!INS.!*",t_cnt,
                       w_cnt
                      for j=1 to max_dev
                        stack_add n,stp,t_cnt,w_cnt
                      next j
                    end if
                    if (active_dev+t_cnt)<trg_dev then
                      stack_add n,stp,t_cnt,w_cnt
                    end if
                    grid_upd
                    stp=stp+1
                  loop
                  mfdsa(k).t_prc=tot_prc
                  mfdsa(k).t_stp=stp
                  mfdsa(k).c_dte=date$
                  mfdsa(k).c_tme=time$
                end if
              end if
            end if
          end if
        end if
      end if
    end if
  end if
end sub
```

ZEIPORT Module:

```
sub zexport(k,u1,u2,u3)
  CTRL$=ZHEAD$("Select Export Type:   Dev-List-1   Lst-list-
   2   Prc-List-3   Statistic-4   RAW-5   [ESC]","12345"+
   Z_ESC$)
  if CTRL$<>Z_ESC$ then
    ZGFILE fl$
    if fl$<>"" then
      ZCMD 200,fl$,"O",PASS
      select case CTRL$
      case "1": export_devl 200,u1
      case "2": export_lstl 200,u2
      case "3": export_prcl 200,u3
      case "4": export_stat 200,k,u1,u2,u3
      case "5": export_raw  200,k,u1,u2,u3
      end select
      ZCMD 200,"","",FAIL
    end if
  end if
end sub

sub zimport(n,k)
  load n,k,"MFDSA"
  if mfdsa(k).t_dev=0 then
    if mfdsa(k).t_lst=0 then
      if mfdsa(k).t_prc=0 then
        ZGFILE fl$
        if fl$<>"" then
          ZCMD 200,fl$,"I",PASS
          input #200,mfdsa(k).n,mfdsa(k).m,mfdsa(k).t,zmax1,
           zmax2
          if zmax1*zmax2<>0 then
            import_dev n,200,k,zmax1
            import_lst n,200,k,zmax2
          end if
          ZCMD 200,"","",FAIL
        end if
      end if
    end if
  end if
  save n,k,"MFDSA"
end sub

sub zprntscrn()
  ZGFILE fl$
  if fl$<>"" then bsave fl$+".BMP",0
end sub
```

ZLISTANB Module:

```
sub zlist_A(n,k)
  gosub zlist_A_start
  gosub zlist_A_fresh1
  gosub zlist_A_fresh2
  gosub zlist_A_prompt
  gosub zlist_A_record1
  gosub zlist_A_record2
  do
    CTRL$=ZHEAD$("MENU:   Add-A   Edit-E   Del-D   Mode-
     [C/Z]   Export-X   PrntScrn-P   [ESC]",Z_ARU$+Z_ARD$
    +Z_ARL$+Z_ARR$+Z_ARH$+Z_ARE$+Z_PGU$+Z_PGD$+
     Z_SPC$+Z_TAB$+"AED"+Z_UND$+"XP"+Z_ESC$)
    select case CTRL$
    case Z_ARU$,Z_ARD$,Z_ARL$,Z_ARR$,Z_ARH$,Z_ARE$,Z_PGU$,
     Z_PGD$:
      select case CTRL
      case 1: ZSYS f0,f1,f2,36,mfdsa(k).t_lst,CTRL$
      case 2: ZSYS g0,g1,g2,36,mfdsa(k).t_dev,CTRL$
      end select
      gosub zlist_A_record1
      gosub zlist_A_record2
    case Z_SPC$:
      gosub zlist_A_start
      gosub zlist_A_fresh1
      gosub zlist_A_fresh2
      gosub zlist_A_prompt
      gosub zlist_A_record1
      gosub zlist_A_record2
    case Z_TAB$:
      select case CTRL
      case 1: CTRL=2: tpa=ACT2: tpb=ACT1
      case 2: CTRL=1: tpa=ACT1: tpb=ACT2
      end select
      gosub zlist_A_prompt
      gosub zlist_A_record1
      gosub zlist_A_record2
    case "A":
      zrecord_dnl n,k,0,0,0,0,"A"
      gosub zlist_A_fresh1
      gosub zlist_A_fresh2
      gosub zlist_A_prompt
      gosub zlist_A_record1
      gosub zlist_A_record2
    case "X":
      zexport k,1,2,4
    case "P":
      zprntscrn
```

```
    case "E","D",Z_UND$:
      if (CTRL=1 and mfdsa(k).t_lst<>0) or (CTRL=2 and
       mfdsa(k).t_dev<>0) then
        select case CTRL$
        case "E":
          select case CTRL
          case 1: zrecord_dnl n,k,f0,f1,f2,1,"E"
          case 2: zrecord_dnl n,k,g0,g1,g2,2,"E"
          end select
          gosub zlist_A_record1
          gosub zlist_A_record2
        case "D":
          select case CTRL
          case 1: zrecord_dnl n,k,f0,0,0,1,"D"
          case 2: zrecord_dnl n,k,g0,0,0,2,"D"
          end select
          gosub zlist_A_record1
          gosub zlist_A_record2
        case Z_UND$:
          select case CTRL
          case 1: zrecord_dnl n,k,f0,0,0,1,Z_UND$
          case 2: zrecord_dnl n,k,g0,0,0,2,Z_UND$
          end select
          gosub zlist_A_record1
          gosub zlist_A_record2
        end select
      end if
    case Z_ESC$:
      exit do
    end select
  loop
  exit sub

zlist_A_start:
CTRL=1: tpa=ACT1: tpb=ACT2
return

zlist_A_fresh1:
ZSYS f0,f1,f2,36,mfdsa(k).t_lst,Z_ARH$
return

zlist_A_fresh2:
ZSYS g0,g1,g2,36,mfdsa(k).t_dev,Z_ARH$
return

zlist_A_prompt:
prompt_list_A "D&L Input "+"["+idxB(k).fmfdsa+"]",tpa,tpb
return

zlist_A_record1:
record_list_A1 f0,f1,f2
return

zlist_A_record2:
record_list_A2 g0,g1,g2
return
end sub
```

```
sub zlist_B(n,k)
  ZCLEAR
  gosub zlist_B_start
  gosub zlist_B_fresh1
  gosub zlist_B_fresh2
  gosub zlist_B_prompt
  gosub zlist_B_record
  do
    CTRL$=ZHEAD$("MENU:   Export-X   PrntScrn-P   [ESC]",
     Z_ARU$+Z_ARD$+Z_ARL$+Z_ARR$+Z_ARH$+Z_ARE$+Z_PGU$
    +Z_PGD$+Z_SPC$+"XP"+Z_ESC$)
    select case CTRL$
    case Z_PGU$,Z_PGD$:
      ZSYS f0,f1,f2,mfdsa(k).t_dev+1,mfdsa(k).t_dev+1,
       CTRL$
      gosub zlist_B_fresh2
      gosub zlist_B_prompt
      gosub zlist_B_record
    case Z_ARU$,Z_ARD$,Z_ARL$,Z_ARR$,Z_ARH$,Z_ARE$:
      ZSYS g0,g1,g2,36,IDX(8,u),CTRL$
      gosub zlist_B_record
    case Z_SPC$:
      gosub zlist_B_start
      gosub zlist_B_fresh1
      gosub zlist_B_fresh2
      gosub zlist_B_prompt
      gosub zlist_B_record
    case "X":
      zexport k,1,2,8
    case "P":
      zprntscrn
    case Z_ESC$:
      exit do
    end select
  loop
  ZCLEAR
  exit sub

zlist_B_start:
return

zlist_B_fresh1:
ZSYS f0,f1,f2,mfdsa(k).t_dev+1,mfdsa(k).t_dev+1,Z_ARH$
return

zlist_B_fresh2:
if f0-1<>0 then HASH0$(1)=mkshort$(f0-1) else ZCLEAR
sift 4,8
ZSYS g0,g1,g2,36,IDX(8,u),Z_ARH$
return

zlist_B_prompt:
prompt_list_B "*P* Output "+"["+idxB(k).fmfdsa+"]"+"
D:"+ZTEXT$(str$(f0-1),"0",5,"R")
return
```

```
zlist_B_record:
record_list_B g0,g1,g2,8
return
end sub
```

ZPROJECT Module:

```
sub zproject(k)
  ZCMD 100,idxB(k).fmfdsa,"B",PASS
  load 100,k,"MFDSA"
  gosub zproject_start
  gosub zproject_fresh1
  gosub zproject_fresh2
  gosub zproject_prompt
  gosub zproject_record1
  gosub zproject_record2
  gosub zproject_record3
  do
    CTRL$=ZHEAD$("MENU:   D&L Input-A   *P* Output -B
     Compile-C   Export-X   Import-M   Trace-T   PrntScrn
    -P   [ESC]",Z_ARU$+Z_ARD$+Z_ARL$+Z_ARR$+Z_ARH$+
     Z_ARE$+Z_PGU$+Z_PGD$+Z_SPC$+Z_TAB$+"ABCXMTP"+
     Z_ESC$)
    select case CTRL$
    case Z_ARU$,Z_ARD$,Z_ARL$,Z_ARR$:
      select case CTRL
      case 1: gosub zproject_ctrl_space1
      case 2: gosub zproject_ctrl_space2
      end select
      gosub zproject_prompt
      gosub zproject_record1
      gosub zproject_record2
      gosub zproject_record3
    case Z_ARH$,Z_ARE$,Z_PGU$,Z_PGD$:
      gosub zproject_ctrl_time
      gosub zproject_fresh2
      gosub zproject_prompt
      gosub zproject_record1
      gosub zproject_record2
      gosub zproject_record3
    case Z_SPC$:
      gosub zproject_start
      gosub zproject_fresh1
      gosub zproject_fresh2
      gosub zproject_prompt
      gosub zproject_record1
      gosub zproject_record2
      gosub zproject_record3
    case Z_TAB$:
      select case CTRL
      case 1: CTRL=2
      case 2: CTRL=1
      end select
      gosub zproject_record1
      gosub zproject_record2
      gosub zproject_record3
```

```
    case "A":
      zlist_A 100,k
      gosub zproject_prompt
      gosub zproject_record1
      gosub zproject_record2
      gosub zproject_record3
    case "B":
      zlist_B 100,k
      gosub zproject_prompt
      gosub zproject_record1
      gosub zproject_record2
      gosub zproject_record3
    case "C":
      zcompile 100,k
      gosub zproject_start
      gosub zproject_fresh1
      gosub zproject_fresh2
      gosub zproject_prompt
      gosub zproject_record1
      gosub zproject_record2
      gosub zproject_record3
    case "X":
      zexport k,1,2,4
    case "M":
      zimport 100,k
    case "T":
      proj_trace k,f0,g0
      gosub zproject_prompt
      gosub zproject_record1
      gosub zproject_record2
      gosub zproject_record3
    case "P":
      zprntscrn
    case Z_ESC$:
      exit do
    end select
  loop
  save 100,k,"MFDSA"
  ZCMD 100,"","",FAIL
  exit sub

zproject_start:
build_dlp 100,1,mfdsa(k).t_dev,"DEV"
build_dlp 100,2,mfdsa(k).t_lst,"LST"
build_dlp 100,4,mfdsa(k).t_prc,"PRC"
CTRL=1
return

zproject_fresh1:
t0=1: t1=1: t2=mfdsa(k).t_stp: tsize=1: tleng=t2
f0=0: f1=0: f2=mfdsa(k).n+1: fsize=12: fleng=f2: if f2>fsize then
f2=fsize-1
g0=0: g1=0: g2=mfdsa(k).n+1: gsize=12: gleng=g2: if g2>gsize then
g2=gsize-1
return
```

```
zproject_fresh2:
proj_grid proj_stack(k,t0)
return

zproject_prompt:
prompt_project "["+idxB(k).fmfdsa+"]"
return

zproject_record1:
record_project1 k,t0,f0,f1,f2,g0,g1,g2,CTRL
return

zproject_record2:
record_project2 k
return

zproject_record3:
record_project3 grid(f0,g0)
return

zproject_ctrl_time:
select case CTRL$
case Z_ARH$: t0=t1
case Z_ARE$: t0=t2
case Z_PGU$: if t0<t2 then t0=t0+1
case Z_PGD$: if t0>t1 then t0=t0-1
end select
return

zproject_ctrl_space1:
select case CTRL$
case Z_ARU$: if g0<g2 then g0=g0+1
case Z_ARD$: if g0>g1 then g0=g0-1
case Z_ARL$: if f0>f1 then f0=f0-1
case Z_ARR$: if f0<f2 then f0=f0+1
end select
return

zproject_ctrl_space2:
select case CTRL$
case Z_ARU$: if g2<gleng then g2=g2+1: g1=(g2-gsize)+1: if g0<g1
then g0=g1
case Z_ARD$: if g1>0     then g1=g1-1: g2=(g1+gsize)-1: if g0>g2
then g0=g2
case Z_ARL$: if f1>0     then f1=f1-1: f2=(f1+fsize)-1: if f0>f2
then f0=f2
case Z_ARR$: if f2<fleng then f2=f2+1: f1=(f2-fsize)+1: if f0<f1
then f0=f1
end select
return
end sub
```

ZRECORD Module:

```
sub zrecord_dnl(n,k,f0,f1,f2,CTRL,CTRL$)
  if mfdsa(k).acode<>FAIL then
    select case CTRL$
    case "A":
      f0=pack(n,"",mfdsa(k).t_lst,"LST")
      f0=pack(n,"",mfdsa(k).t_dev,"DEV")
    case "E":
      select case CTRL
      case 1:
        load n,f0,"LST"
        dev_n=val(ZINPUT$(str$(lst(f0).dev_n),5,5,8+(f0
        -f1),5,RGBA0(MAIN,2),RGBA0(MAIN,1)))
        if check_dev(k,dev_n)<>FAIL then
          if check_gns(k,f0,dev_n)<>FAIL then
            lst(f0).dev_n=dev_n
          end if
        end if
        save n,f0,"LST"
      case 2:
        CTRL$=ZHEAD$("Select Edit Type:   Ini-X,Ini-Y-1
         Fin-X,Fin-Y-2   Off-X,Off-Y-3
         Mass/Fric/Size/Delta-4   [ESC]","1234"+Z_ESC$)
        if CTRL$<>Z_ESC$ then
          load n,f0,"DEV"
          select case CTRL$
          case "1":
            ini_x=val(ZINPUT$(str$(dev(f0).ini_x),4,4,8+
             (f0-f1),26,RGBA0(MAIN,2),RGBA0(MAIN,1)))
            ini_y=val(ZINPUT$(str$(dev(f0).ini_y),4,4,8+
             (f0-f1),33,RGBA0(MAIN,2),RGBA0(MAIN,1)))
            if check_ini(k,ini_x,ini_y)<>FAIL then
              dev(f0).ini_x=ini_x
              dev(f0).ini_y=ini_y
            end if
          case "2":
            fin_x=val(ZINPUT$(str$(dev(f0).fin_x),4,4,8+
             (f0-f1),45,RGBA0(MAIN,2),RGBA0(MAIN,1)))
            fin_y=val(ZINPUT$(str$(dev(f0).fin_y),4,4,8+
             (f0-f1),52,RGBA0(MAIN,2),RGBA0(MAIN,1)))
            if check_fin(k,fin_x,fin_y)<>FAIL then
              if check_sng(k,f0,fin_x,fin_y)<>FAIL then
                dev(f0).fin_x=fin_x
                dev(f0).fin_y=fin_y
              end if
            end if
          case "3":
            off_x!=val(ZINPUT$(str$(dev(f0).off_x),5,5,8+
             (f0-f1),61,RGBA0(MAIN,2),RGBA0(MAIN,1)))
            off_y!=val(ZINPUT$(str$(dev(f0).off_y),5,5,8+
             (f0-f1),68,RGBA0(MAIN,2),RGBA0(MAIN,1)))
            if check_off(off_x!,off_y!)<>FAIL then
              dev(f0).off_x=off_x!
              dev(f0).off_y=off_y!
            end if
```

```
        case "4":
          mass!=val(ZINPUT$(str$(dev(f0).mass),11,11,8+
           (f0-f1),76,RGBA0(MAIN,2),RGBA0(MAIN,1)))
          fric!=val(ZINPUT$(str$(dev(f0).fric),11,11,8+
           (f0-f1),89,RGBA0(MAIN,2),RGBA0(MAIN,1)))
          size!=val(ZINPUT$(str$(dev(f0).size),11,11,8+
           (f0-f1),102,RGBA0(MAIN,2),RGBA0(MAIN,1)))
          delta=val(ZINPUT$(str$(dev(f0).delta),4,4,8+
           (f0-f1),119,RGBA0(MAIN,2),RGBA0(MAIN,1)))
          if check_mfsd(k,mass!,fric!,size!,delta)<>FAIL
          then
            dev(f0).mass=mass!
            dev(f0).fric=fric!
            dev(f0).size=size!
            dev(f0).delta=delta
          end if
        end select
        save n,f0,"DEV"
      end if
    end select
  case "D":
    select case CTRL
    case 1:
      load n,f0,"LST"
      lst(f0).lst_n=lst(f0).lst_n
      lst(f0).dev_n=0
      lst(f0).cntrl="*"
      save n,f0,"LST"
    case 2:
      load n,f0,"DEV"
      dev(f0).dev_n=dev(f0).dev_n
      dev(f0).ini_x=0
      dev(f0).ini_y=0
      dev(f0).fin_x=0
      dev(f0).fin_y=0
      dev(f0).off_x=0
      dev(f0).off_y=0
      dev(f0).mass=0
      dev(f0).fric=0
      dev(f0).size=0
      dev(f0).delta=0
      dev(f0).cntrl="*"
      save n,f0,"DEV"
      for z0=1 to mfdsa(k).t_lst
        if lst(z0).dev_n=dev(f0).dev_n then
          zrecord_dnl n,k,z0,0,0,1,"D"
        end if
      next z0
    end select
```

```
    case Z_UND$:
      select case CTRL
      case 1:
        load n,f0,"LST"
        if dev(lst(f0).dev_n).cntrl="#" then
          select case lst(f0).cntrl
          case "*": lst(f0).cntrl="#"
          case "#": lst(f0).cntrl="*"
          end select
        end if
        save n,f0,"LST"
      case 2:
        if check_ini(k,dev(f0).ini_x,dev(f0).ini_y)<>FAIL
         then
          if check_fin(k,dev(f0).fin_x,dev(f0).fin_y)<>FAIL
           then
            if check_sng(k,f0,dev(f0).fin_x,dev(f0).fin_y)<>
             FAIL then
              if check_off(dev(f0).off_x,dev(f0).off_y)<>FAIL
               then
                if check_mfsd(k,dev(f0).mass,dev(f0).fric,
                 dev(f0).size,dev(f0).delta)<>FAIL then
                  load n,f0,"DEV"
                  select case dev(f0).cntrl
                  case "*":
                    dev(f0).cntrl="#"
                  case "#":
                    dev(f0).cntrl="*"
                    for z0=1 to mfdsa(k).t_lst
                      if lst(z0).dev_n=dev(f0).dev_n then
                        load n,z0,"LST"
                        lst(z0).cntrl=dev(f0).cntrl
                        save n,z0,"LST"
                      end if
                    next z0
                  end select
                  save n,f0,"DEV"
                end if
              end if
            end if
          end if
        end if
      end select
    end select
  end if
end sub
```

```
sub zrecord_mfdsa(n,k,CTRL$)
  if k<>0 then load n,k,"IDXB"
  select case CTRL$
  case "A":
    k=pack(n,"MFDSA00000",idxA.a_record,"MFDSA")
    if k<>0 then
      prompt_main "Add Project"
      record_main2 k
      tpx$=ZINPUT$(idxB(k).fname,32,32,13,57,RGBA0(MAIN,
       2),RGBA0(MAIN,1))
      if trim$(tpx$)<>"" then idxB(k).fname=tpx$
    end if
  case "E":
    if mfdsa(k).acode<>FAIL then
      prompt_main "Edit Project"
      record_main2 k
      CTRL$=ZHEAD$("Select Edit Type:   File Name-1   Grid
       nxn-2   Max Dev-3   Trg Dev-4   [ESC]","1234"+
       Z_ESC$)
      if CTRL$<>Z_ESC$ then
        ZCMD 100,idxB(k).fmfdsa,"B",PASS
        load 100,k,"MFDSA"
        select case CTRL$
        case "1":
          tpx$=ZINPUT$(idxB(k).fname,32,32,13,57,
           RGBA0(MAIN,2),RGBA0(MAIN,1))
          if trim$(tpx$)<>"" then idxB(k).fname=tpx$
        case "2":
          if mfdsa(k).t_dev=0 then
            if mfdsa(k).t_lst=0 then
              tpx=val(ZINPUT$(str$(mfdsa(k).n),5,5,25,57,
               RGBA0(MAIN,2),RGBA0(MAIN,1)))
              if tpx>=10 and tpx<=1024 then mfdsa(k).n=tpx
            end if
          end if
        case "3":
          if mfdsa(k).t_prc=0 then
            tpx=val(ZINPUT$(str$(mfdsa(k).m),5,5,26,57,
             RGBA0(MAIN,2),RGBA0(MAIN,1)))
            if tpx>=0 then mfdsa(k).m=tpx
          end if
        case "4":
          if mfdsa(k).t_prc=0 then
            tpx=val(ZINPUT$(str$(mfdsa(k).t),5,5,27,57,
             RGBA0(MAIN,2),RGBA0(MAIN,1)))
            if tpx>=0 then mfdsa(k).t=tpx
          end if
        end select
        save 100,k,"MFDSA"
        ZCMD 100,"","",FAIL
      end if
    end if
```

```
  case "D":
    if mfdsa(k).acode<>FAIL then
      prompt_main "Del Project"
      record_main2 k
      CTRL$=ZHEAD$("WARNING!!   DEL CANNOT BE REVERSED!!
       CONTINUE?   Y-1   N-2   [ESC]","12"+Z_ESC$)
      if CTRL$="1" then
        idxA.a_record=idxA.a_record-0
        idxA.r_record=idxA.r_record-1
        idxB(k).fmfdsa=""
        idxB(k).fname=""
        idxB(k).cntrl="*"
        kill "MFDSA"+ZTEXT$(str$(tpx),"0",5,"R")
      end if
    end if
  case Z_UND$:
    ZCMD 100,idxB(k).fmfdsa,"B",PASS
    load 100,k,"MFDSA"
    select case mfdsa(k).acode
    case FAIL: mfdsa(k).acode=PASS
    case PASS: mfdsa(k).acode=FAIL
    end select
    save 100,k,"MFDSA"
    ZCMD 100,"","",FAIL
  end select
  if k<>0 then save n,k,"IDXB"
end sub

sub zrecord_prc(n,stp,j,tpx$,t_cnt,w_cnt)
  cur_prc=pack(n,tpx$,tot_prc,"PRC")
  select case j
  case is<=0:
    prc(cur_prc).dev_n=-1
    prc(cur_prc).stp_n=stp
    prc(cur_prc).move=NL
  case is> 0:
    prc(cur_prc).dev_n=stack(j).dev_n
    prc(cur_prc).stp_n=stp
    prc(cur_prc).move=stack(j).move
  end select
  if j>0 then
    prc(cur_prc).ini_x=stack(j).cur_x
    prc(cur_prc).ini_y=stack(j).cur_y
    grid_sub j
    select case stack(j).move
    case UP: stack(j).cur_y=stack(j).cur_y+1
    case DN: stack(j).cur_y=stack(j).cur_y-1
    case LT: stack(j).cur_x=stack(j).cur_x-1
    case RT: stack(j).cur_x=stack(j).cur_x+1
    end select
    grid_add j
    prc(cur_prc).fin_x=stack(j).cur_x
    prc(cur_prc).fin_y=stack(j).cur_y
  end if
```

```
  prc(cur_prc).t_cnt=t_cnt
  prc(cur_prc).w_cnt=w_cnt
  save n,cur_prc,"PRC"
end sub

sub zreset(k)
  if mfdsa(k).acode<>FAIL then
    CTRL$=ZHEAD$("WARNING!!   RESET CANNOT BE REVERSED!!
     CONTINUE?   Y-1   N-2   [ESC]","12"+Z_ESC$)
    if CTRL$="1" then
      CTRL$=ZHEAD$("Select Reset Type:   D&L Input-1   *P*
       Output-2   [ESC]","12"+Z_ESC$)
      if CTRL$<>Z_ESC$ then
        ZCMD 100,idxB(k).fmfdsa,"B",PASS
        ZCMD 200,"MFDSA00000","B",PASS
        load 100,k,"MFDSA"
        select case CTRL$
        case "1":
          mfdsa(k).acode=PASS
          mfdsa(k).zcode=FAIL
          mfdsa(k).t_dev=0
          mfdsa(k).t_lst=0
          mfdsa(k).t_prc=0
          mfdsa(k).t_stp=0
          mfdsa(k).c_dte=date$
          mfdsa(k).c_tme=time$
        case "2":
          mfdsa(k).acode=PASS
          mfdsa(k).zcode=FAIL
          mfdsa(k).t_dev=mfdsa(k).t_dev
          mfdsa(k).t_lst=mfdsa(k).t_lst
          mfdsa(k).t_prc=0
          mfdsa(k).t_stp=0
          mfdsa(k).c_dte=date$
          mfdsa(k).c_tme=time$
        end select
        for z0=1 to mfdsa(k).t_dev
          load 100,z0,"DEV"
          save 200,z0,"DEV"
        next z0
        for z0=1 to mfdsa(k).t_lst
          load 100,z0,"LST"
          save 200,z0,"LST"
        next z0
        save 200,k,"MFDSA"
        ZCMD 200,"","",FAIL
        ZCMD 100,"","",FAIL
        kill idxB(k).fmfdsa
        name "MFDSA00000",idxB(k).fmfdsa
      end if
    end if
  end if
end sub
```

APPENDIX C

MAGSTAT APPLICATION CODE

The complete MagStat program written in freeBASIC [112]; the application is divided into modules MAIN, TYPE, VECT, FILE, CALB, CALU, CALF, CALS, and CALCCA.

MAIN *controls all functions and modules*

command_break	decodes the inputted line
command_parse	executes the inputted line
command_ready	obtains the inputted line

TYPE *contains data type struture, variable, and function definitions*

VECT *contains functions that define and maintain the vector data structure and its functions*

xVect	initializes a vector
aVect	adds two vectors
sVect	subtracts two vectors
mVect	multiplies a vector by a scalar
dVect	divides a vector by a scalar
dProd	the dot product of two vectors
xProd	the cross product of two vectors
magRt	returns the magnitude of a vector
angRt	returns the angle of a vector
magSt	sets the magnitude of a vector
nrmSt	normalizes a vectors

FILE *contains the functions that define the file system's input and output functions; used by the MAIN module*

load_file_ema	loads the array data file
load_file_sml	loads the layer data file
out1d_r	outputs "y v. x" plot, scalar input
out1d_v	outputs "y v. x" plot, vector input
out2d_r	outputs "surface" plot, scalar input
out2d_v	outputs "surface" plot, vector input
read_data_r	reads scalar data from file
read_data_v	reads vector data from file
save_file_ema	saves the array data file
save_file_sml	saves the layer data file
write_data_r	writes scalar data to file
write_data_v	writes vector data to file

CALB *contains the functions that calculate the magnetic field; used by the CALCCA and MAIN modules*

f_D	calculates the denominator factor
f_N	calculates the numerator factor
f_X	calculates the magnetic field x-component
f_Y	calculates the magnetic field y-component
f_Z	calculates the magnetic field z-component
B_INT	integrates the magnetic field components
B_CAL	calculates the magnetic field
f_B	magnetic field calculator

CALU *contains the functions that calculate the energy; used by the MAIN module*

U_CAL	calculates the magnetic energy
f_U	magnetic energy calculator

CALF *contains the functions that calculate the magnetic force; used by the CALCCA and MAIN modules*

F_CAL	calculates the magnetic force
f_F	magnetic force calculator

CALS *contains the functions that calculate the total force; used by the CALCCA and MAIN modules*

S_CAL	calculates the total force
f_S	total force calculator

CALCCA *contains the functions that calculate the exact CCA; used by the MAIN module*

BF_CAL	calculates the magnetic field and force
I_rampD	decreases the element current
I_rampU	increases the element current
I_setup	initializes the element current
swap_e	sets up the array data
swap_s	sets up the layer data
f_CCA	CCA calculator

MAIN Module:

```
defint a-z
#define declarefunc declare function
#define declarevoid declare sub
#define func function
#define void sub
#define global dim shared
#define cvr cvd
#define mkr mkd
#define cvn cvi
#define mkn mki
#define real double
#define nmbr integer
#define char string
option base 1

#include "type.bi"
#include "vect.bi"
#include "file.bi"
#include "calB.bi"
#include "calU.bi"
#include "calF.bi"
#include "calS.bi"
#include "calCCA.bi"

global AX$, BX$(0 to 10)

cls
print "=================================="
print "     ***  MFDSA - MagStat  ***    "
print "  Build Ver. #3.0 (in freeBASIC)  "
print "      Completed - 2010-10-18      "
print "   Program by Rene David Rivero   "
print "=================================="
print
print

do
  command_ready tp0$
  command_break tp0$, AX$, BX$()
  command_parse AX$, BX$()
  print
loop until AX$ = "Q"
```

```
void command_break(tp0$, AX$, BX$())
  tp0 = instr(tp0$, " ")
  if tp0 = 0 then tp0 = len(tp0$) + 1
  AX$ = trim$(mid$(tp0$, 1, tp0 - 1))
  tp0$ = trim$(mid$(tp0$, tp0 + 1))
  if tp0$ <> "" then
    BX$(0) = mkn(0)
    do
      BX$(0) = mkn(cvn(BX$(0)) + 1)
      tp0 = instr(tp0$, ",")
      if tp0 = 0 then tp0 = len(tp0$) + 1
      BX$(cvn(BX$(0))) = trim$(mid$(tp0$, 1, tp0 - 1))
      tp0$ = trim$(mid$(tp0$, tp0 + 1))
    loop until tp0$ = ""
  end if
end void

void command_parse(AX$, BX$())
  select case AX$
  case "NEW":
    erase ema, sml
    eMax = 0
    sMax = 0
    TOL = 10e-12
    sigma1 = 0.95
    sigma2 = 0.05
    omega0 = 0.50
    alpha0 = 1
    sr2 = 50
  case "LOADEMA":
    if cvn(BX$(0)) = 1 then load_file_ema BX$(1)
  case "SAVEEMA":
    if cvn(BX$(0)) = 1 then save_file_ema BX$(1)
  case "EMA":
    if cvn(BX$(0)) = 9 then
      ema(val(BX$(1))).R0   = nullV
      ema(val(BX$(1))).R1.x = val(BX$(2))
      ema(val(BX$(1))).R1.y = val(BX$(3))
      ema(val(BX$(1))).R1.z = val(BX$(4))
      ema(val(BX$(1))).L    = val(BX$(5))
      ema(val(BX$(1))).N    = val(BX$(6))
      ema(val(BX$(1))).LN   = ema(val(BX$(1))).L /
       ema(val(BX$(1))).N
      ema(val(BX$(1))).I0   = 0
      ema(val(BX$(1))).I1   = val(BX$(7))
      ema(val(BX$(1))).Im   = val(BX$(8))
      ema(val(BX$(1))).a    = val(BX$(9))
      magSt ema(val(BX$(1))).R0
      magSt ema(val(BX$(1))).R1
    end if
```

```
case "EMA?":
  if cvn(BX$(0)) = 2 then
    print "EMA      R1.x      R1.y      R1.z        L   N
     I1       Im        a "
    print "-------------------------------------------------
     ----------------------"
    for Z = val(BX$(1)) to val(BX$(2))
      print using "### +#.##^^^^ +#.##^^^^ +#.##^^^^ #.##^^^^
       #### #.##^^^^ #.##^^^^ #.##^^^^"; Z; ema(Z).R1.x;
       ema(Z).R1.y; ema(Z).R1.z; ema(Z).L; ema(Z).N; ema(Z).I1;
       ema(Z).Im; ema(Z).a
    next Z
    print "-------------------------------------------------
     ----------------------"
  end if
case "EMAX":
  if cvn(BX$(0)) = 1 then eMAX = val(BX$(1))
case "EMAX?":
  print eMAX
case "EBSU":
  if cvn(BX$(0)) = 1 then eBSU = val(BX$(1))
case "EBSU?":
  print eBSU
case "LOADSML":
  if cvn(BX$(0)) = 1 then load_file_sml BX$(1)
case "SAVESML":
  if cvn(BX$(0)) = 1 then save_file_sml BX$(1)
case "SML":
  if cvn(BX$(0)) = 10 then
    sml(val(BX$(1))).R0 = nullV
    sml(val(BX$(1))).R1.x = val(BX$(2))
    sml(val(BX$(1))).R1.y = val(BX$(3))
    sml(val(BX$(1))).R1.z = val(BX$(4))
    sml(val(BX$(1))).M.x  = val(BX$(5))
    sml(val(BX$(1))).M.y  = val(BX$(6))
    sml(val(BX$(1))).M.z  = val(BX$(7))
    sml(val(BX$(1))).N    = val(BX$(8))
    sml(val(BX$(1))).MN   = dVect(sml(val(BX$(1))).M,
     sml(val(BX$(1))).N)
    sml(val(BX$(1))).W    = val(BX$(9))
    sml(val(BX$(1))).x0   = 0
    sml(val(BX$(1))).x1   = val(BX$(10))
    magSt sml(val(BX$(1))).R0
    magSt sml(val(BX$(1))).R1
    magSt sml(val(BX$(1))).M
    magSt sml(val(BX$(1))).MN
  end if
case "SML?":
  if cvn(BX$(0)) = 2 then
    print "SML     R1.x     R1.y     R1.z     M.x     M.y
     M.z    N       W      x1 "
    print "-------------------------------------------------
     ---------------------------"
```

```
    for Z = val(BX$(1)) to val(BX$(2))
      print using "### +#.##^^^^ +#.##^^^^ +#.##^^^^ #.#^^^^
       #.#^^^^ #.#^^^^ #### #.#^^^^ #.#^^^^"; Z; sml(Z).R1.x;
       sml(Z).R1.y; sml(Z).R1.z; sml(Z).M.x; sml(Z).M.y;
       sml(Z).M.z; sml(Z).N; sml(Z).W; sml(Z).x1
    next Z
    print "-----------------------------------------------------
     ---------------------------"
  end if
case "SMAX":
  if cvn(BX$(0)) = 1 then sMAX = val(BX$(1))
case "SMAX?":
  print sMAX
case "SBSU":
  if cvn(BX$(0)) = 1 then sBSU = val(BX$(1))
case "SBSU?":
  print sBSU
case "TOL":
  if cvn(BX$(0)) = 1 then TOL = val(BX$(1))
case "TOL?":
  print TOL
case "SR2":
  if cvn(BX$(0)) = 1 then sr2 = val(BX$(1))
case "SR2?":
  print sr2
case "SIGMA1":
  if cvn(BX$(0)) = 1 then sigma1 = val(BX$(1))
case "SIGMA1?":
  print sigma1
case "SIGMA2":
  if cvn(BX$(0)) = 1 then sigma2 = val(BX$(1))
case "SIGMA2?":
  print simga2
case "OMEGA0":
  if cvn(BX$(0)) = 1 then omega0 = val(BX$(1))
case "OMEGA0?":
  print omega0
case "ALPHA0":
  if cvn(BX$(0)) = 1 then alpha0 = val(BX$(1))
case "ALPHA0?":
  print alpha0
case "B":
  if cvn(BX$(0)) = 2 then f_B BX$(1), sml(val(BX$(2)))
case "U":
  if cvn(BX$(0)) = 3 then f_U BX$(1), BX$(2), sml(val(BX$(3)))
case "F":
  if cvn(BX$(0)) = 3 then f_F BX$(1), BX$(2), sml(val(BX$(3)))
case "S":
  if cvn(BX$(0)) = 3 then f_S BX$(1), BX$(2), sml(val(BX$(3)))
case "CCA":
  if cvn(BX$(0)) = 2 then f_CCA BX$(1), sml(val(BX$(2)))
case "1DR":
  if cvn(BX$(0)) = 6 then out1d_r BX$(1), BX$(2), BX$(3),
   val(BX$(4)), val(BX$(5)), sml(val(BX$(6)))
```

```
  case "1DV":
    if cvn(BX$(0)) = 7 then out1d_v BX$(1), BX$(2), BX$(3),
     BX$(4), val(BX$(5)), val(BX$(6)), sml(val(BX$(7)))
  case "2DR":
    if cvn(BX$(0)) = 5 then out2d_r BX$(1), BX$(2), BX$(3),
     val(BX$(4)), sml(val(BX$(5)))
  case "2DV":
    if cvn(BX$(0)) = 6 then out2d_v BX$(1), BX$(2), BX$(3),
     BX$(4), val(BX$(5)), sml(val(BX$(6)))
  case "FEED":
    if cvn(BX$(0)) = 1 then
      open BX$(1) for input as #100
      do
        line input #100, tp0$
        command_break tp0$, AX$, BX$()
        if AX$ <> "FEED" then command_parse AX$, BX$()
      loop until AX$ = "END" or EOF(100)
      close #100
    end if
  case "END":
  case "Q":
  end select
end void

void command_ready(tp0$)
  print "Ready : ";
  line input tp0$
  tp0$ = ucase$(trim$(tp0$))
end void
```

TYPE Module:

```
type ema_space
  R0 as vect
  R1 as vect
  L  as real
  N  as real
  LN as real
  I0 as real
  I1 as real
  Im as real
  a  as real
end type
global ema(32767) as ema_space, eMAX as nmbr, eBSU as real
type sml_space
  R0 as vect
  R1 as vect
  M  as vect
  N  as real
  MN as vect
  W  as real
  x0 as real
  x1 as real
end type
global sml(32767) as sml_space, sMAX as nmbr, sBSU as real

global PI as real, u0 as real, TOL as real, sigma1 as real,
sigma2 as real, omega0 as real, alpha0 as real, sr2 as nmbr
PI = 3.14159
u0 = 1.25663 * 10^(-6)
TOL = 10e-12
sigma1 = 0.95
sigma2 = 0.05
omega0 = 0.50
alpha0 = 1
sr2 = 50

declarevoid load_file_ema(...)
declarevoid load_file_sml(...)
declarevoid out1d_r(...)
declarevoid out1d_v(...)
declarevoid out2d_r(...)
declarevoid out2d_v(...)
declarevoid read_data_r(...)
declarevoid read_data_v(...)
declarevoid save_file_ema(...)
declarevoid save_file_sml(...)
declarevoid write_data_r(...)
declarevoid write_data_v(...)
```

```
declarefunc f_D(...) as real
declarefunc f_N(...) as real
declarefunc f_X(...) as real
declarefunc f_Y(...) as real
declarefunc f_Z(...) as real
declarefunc B_INT(...) as vect
declarefunc B_CAL(...) as vect
declarevoid f_B(...)

declarefunc U_CAL(...) as real
declarevoid f_U(...)

declarefunc F_CAL(...) as vect
declarevoid f_F(...)

declarefunc S_CAL(...) as vect
declarevoid f_S(...)

declarefunc BF_CAL(...) as vect
declarevoid I_rampD(...)
declarevoid I_rampU(...)
declarevoid I_setup(...)
declarevoid swap_e(...)
declarevoid swap_s(...)
declarevoid f_CCA(...)

declarevoid command_break(...)
declarevoid command_parse(...)
declarevoid command_ready(...)
```

VECT Module:

```
type vect
  x as real
  y as real
  z as real
  m as real
end type

declarefunc xVect(...) as vect
declarefunc aVect(...) as vect
declarefunc sVect(...) as vect
declarefunc mVect(...) as vect
declarefunc dVect(...) as vect
declarefunc dProd(...) as real
declarefunc xProd(...) as vect
declarefunc magRt(...) as real
declarefunc angRt(...) as vect
declarevoid magSt(...)
declarevoid nrmSt(...)

global unitI as vect, unitJ as vect, unitK as vect, nullV as vect
unitI = xVect(1, 0, 0)
unitJ = xVect(0, 1, 0)
unitK = xVect(0, 0, 1)
nullV = xVect(0, 0, 0)

global lengR as nmbr, lengN as nmbr, lengV as nmbr
lengR = len(real)
lengN = len(nmbr)
lengV = len(vect)

func xVect(x as real, y as real, z as real) as vect
  redim tp0 as vect
  tp0.x = x
  tp0.y = y
  tp0.z = z
  magSt tp0
  return tp0
end func

func aVect(A as vect, B as vect) as vect
  redim tp0 as vect
  tp0.x = A.x + B.x
  tp0.y = A.y + B.y
  tp0.z = A.z + B.z
  magSt tp0
  return tp0
end func
```

```
func sVect(A as vect, B as vect) as vect
  redim tp0 as vect
  tp0.x = A.x - B.x
  tp0.y = A.y - B.y
  tp0.z = A.z - B.z
  magSt tp0
  return tp0
end func

func mVect(A as vect, B as real) as vect
  redim tp0 as vect
  tp0.x = A.x * B
  tp0.y = A.y * B
  tp0.z = A.z * B
  magSt tp0
  return tp0
end func

func dVect(A as vect, B as real) as vect
  redim tp0 as vect
  tp0.x = A.x / B
  tp0.y = A.y / B
  tp0.z = A.z / B
  magSt tp0
  return tp0
end func

func dProd(A as vect, B as vect) as real
  redim tp0 as real
  tp0 = A.x * B.x + A.y * B.y + A.z * B.z
  return tp0
end func

func xProd(A as vect, B as vect) as vect
  redim tp0 as vect
  tp0.x = A.y * B.z - A.z * B.y
  tp0.y = A.z * B.x - A.x * B.z
  tp0.z = A.x * B.y - A.y * B.x
  magSt tp0
  return tp0
end func

func magRt(A as vect) as real
  return dProd(A, A) ^  (1/2)
end func

func angRt(A as vect) as vect
  return xVect(atn(A.y / A.x), atn(A.z / A.y), atn(A.x / A.z))
end func

void magSt(A as vect)
  A.m = dProd(A, A) ^ (1/2)
end void

void nrmSt(A as vect)
  A = dVect(A, magRt(A))
end void
```

FILE Module:

```
void load_file_ema(file$)
  open file$ for input as #1
  input #1, eMAX
  input #1, eBSU
  for Z = 1 to eMAX
    input #1, ema(Z).R1.x, ema(Z).R1.y, ema(Z).R1.z, ema(Z).L,
     ema(Z).N, ema(Z).I1, ema(Z).Im, ema(Z).a
    ema(Z).R0 = nullV
    ema(Z).LN = ema(Z).L / ema(Z).N
    ema(Z).I0 = 0
    magSt ema(Z).R0
    magSt ema(Z).R1
  next Z
  close #1
end void

void load_file_sml(file$)
  open file$ for input as #1
  input #1, sMAX
  input #1, sBSU
  for Z = 1 to sMAX
    input #1, sml(Z).R1.x, sml(Z).R1.y, sml(Z).R1.z, sml(Z).M.x,
     sml(Z).M.y, sml(Z).M.z, sml(Z).N, sml(Z).W, sml(Z).x1
    sml(Z).R0 = nullV
    sml(Z).MN = dVect(sml(Z).M, sml(Z).N)
    sml(Z).x0 = 0
    magSt sml(Z).R0
    magSt sml(Z).R1
    magSt sml(Z).M
    magSt sml(Z).MN
  next Z
  close #1
end void

void out1d_r(fileI$, fileO$, l$, arg1 as nmbr, arg2 as nmbr, s as
sml_space)
  redim R as real
  open fileI$ for binary as #1
  open fileO$ for output as #2
  select case l$
  case "X":
    for i = 0 to s.N - 1
      read_data_r R, 1, s.N, i, arg1, arg2
      print #2, i, R
    next i
  case "Y":
    for j = 0 to s.N - 1
      read_data_r R, 1, s.N, arg1, j, arg2
      print #2, j, R
    next j
```

```
  case "Z":
    for k = 0 to s.N - 1
      read_data_r R, 1, s.N, arg1, arg2, k
      print #2, k, R
    next k
  end select
  close #2
  close #1
end void

void out1d_v(fileI$, fileO$, l$, mode$, arg1 as nmbr, arg2 as
nmbr, s as sml_space)
  redim V as vect
  open fileI$ for binary as #1
  open fileO$ for output as #2
  select case l$
  case "X":
    for i = 0 to s.N - 1
      read_data_v V, 1, s.N, i, arg1, arg2
      select case mode$
      case "X": print #2, i, V.x
      case "Y": print #2, i, V.y
      case "Z": print #2, i, V.z
      case "M": print #2, i, V.m
      end select
    next i
  case "Y":
    for j = 0 to s.N - 1
      read_data_v V, 1, s.N, arg1, j, arg2
      select case mode$
      case "X": print #2, j, V.x
      case "Y": print #2, j, V.y
      case "Z": print #2, j, V.z
      case "M": print #2, j, V.m
      end select
    next j
  case "Z":
    for k = 0 to s.N - 1
      read_data_v V, 1, s.N, arg1, arg2, k
      select case mode$
      case "X": print #2, k, V.x
      case "Y": print #2, k, V.y
      case "Z": print #2, k, V.z
      case "M": print #2, k, V.m
      end select
    next k
    end select
  close #2
  close #1
end void
```

```
void out2d_r(fileI$, fileO$, p$, arg as nmbr, s as sml_space)
  redim R as real
  open fileI$ for binary as #1
  open fileO$ for output as #2
  select case p$
  case "XY":
    for i = 0 to s.N - 1
      for j = 0 to s.N - 1
        read_data_r R, 1, s.N, i, j, arg
        print #2, R,
      next j
      print #2, ""
    next i
  case "XZ":
    for i = 0 to s.N - 1
      for k = 0 to s.N - 1
        read_data_r R, 1, s.N, i, arg, k
        print #2, R,
      next k
      print #2, ""
    next i
  case "YZ":
    for j = 0 to s.N - 1
      for k = 0 to s.N - 1
        read_data_r R, 1, s.N, arg, j, k
        print #2, R,
      next k
      print #2, ""
    next j
  end select
  close #2
  close #1
end void

void out2d_v(fileI$, fileO$, p$, mode$, arg as nmbr, s as
sml_space)
  redim V as vect
  open fileI$ for binary as #1
  open fileO$ for output as #2
  select case p$
  case "XY":
    for i = 0 to s.N - 1
      for j = 0 to s.N - 1
        read_data_v V, 1, s.N, i, j, arg
        select case mode$
        case "X": print #2, V.x,
        case "Y": print #2, V.y,
        case "Z": print #2, V.z,
        case "M": print #2, V.m,
        end select
      next j
      print #2, ""
    next i
```

```
  case "XZ":
    for i = 0 to s.N - 1
      for k = 0 to s.N - 1
        read_data_v V, 1, s.N, i, arg, k
        select case mode$
        case "X": print #2, V.x,
        case "Y": print #2, V.y,
        case "Z": print #2, V.z,
        case "M": print #2, V.m,
        end select
      next k
      print #2, ""
    next i
  case "YZ":
    for j = 0 to s.N - 1
      for k = 0 to s.N - 1
        read_data_v V, 1, s.N, arg, j, k
        select case mode$
        case "X": print #2, V.x,
        case "Y": print #2, V.y,
        case "Z": print #2, V.z,
        case "M": print #2, V.m,
        end select
      next k
      print #2, ""
    next j
  end select
  close #2
  close #1
end void

void read_data_r(R as real, n as nmbr, size as real, i as nmbr, j
as nmbr, k as nmbr)
  seek #n, 1 + lengN + i * lengR * (size ^ 0) + j * lengR * (size
   ^ 1) + k * lengR * (size ^ 2)
  R = cvr(input$(lengR, #n))
end void

void read_data_v(V as vect, n as nmbr, size as real, i as nmbr, j
as nmbr, k as nmbr)
  seek #n, 1 + lengN + i * lengV * (size ^ 0) + j * lengV * (size
   ^ 1) + k * lengV * (size ^ 2)
  V.x = cvr(input$(lengR, #n))
  V.y = cvr(input$(lengR, #n))
  V.z = cvr(input$(lengR, #n))
  V.m = cvr(input$(lengR, #n))
end void
```

```
void save_file_ema(file$)
  open file$ for output as #1
  print #1, eMAX
  print #1, eBSU
  for Z = 1 to eMAX
    print #1, ema(Z).R1.x, ema(Z).R1.y, ema(Z).R1.z, ema(Z).L,
     ema(Z).N, ema(Z).I1, ema(Z).Im, ema(Z).a
  next Z
  close #1
end void

void save_file_sml(file$)
  open file$ for output as #1
  print #1, sMAX
  print #1, sBSU
  for Z = 1 to sMAX
    print #1, sml(Z).R1.x, sml(Z).R1.y, sml(Z).R1.z, sml(Z).M.x,
     sml(Z).M.y, sml(Z).M.z, sml(Z).N, sml(Z).W, sml(Z).x1
  next Z
  close #1
end void

void write_data_r(R as real, n as nmbr, size as real, i as nmbr,
j as nmbr, k as nmbr)
  seek #n, 1 + lengN + i * lengR * (size ^ 0) + j * lengR * (size
   ^ 1) + k * lengR * (size ^ 2)
  tp0$ = mkr(R): put #n, , tp0$
end void

void write_data_v(V as vect, n as nmbr, size as real, i as nmbr,
j as nmbr, k as nmbr)
  seek #n, 1 + lengN + i * lengV * (size ^ 0) + j * lengV * (size
   ^ 1) + k * lengV * (size ^ 2)
  tp0$ = mkr(V.x): put #n, , tp0$
  tp0$ = mkr(V.y): put #n, , tp0$
  tp0$ = mkr(V.z): put #n, , tp0$
  tp0$ = mkr(V.m): put #n, , tp0$
end void
```

CALB Module:

```
func f_D(V as vect, a as real, b as real, e as ema_space, s as
sml_space) as real
  redim R as real
  R = ((e.a ^ (2)) + (V.m ^ (2)) + ((e.LN * b) ^ (2)) - 2 * (e.a
   * V.x * cos(a) + e.a * V.y * sin(a) + e.LN * V.z * b)) ^ (3/2)
  return R
end func

func f_N(V as vect, a as real, b as real, e as ema_space, s as
sml_space) as real
  redim R as real
  R = u0 * (1 + s.x1) * e.I1 * e.a / 4 / PI
  return R
end func

func f_X(V as vect, a as real, b as real, e as ema_space, s as
sml_space) as real
  redim R as real
  R = f_N(V, a, b, e, s) / f_D(V, a, b, e, s) * (V.z - e.LN * b)
   * cos(a)
  return R
end func

func f_Y(V as vect, a as real, b as real, e as ema_space, s as
sml_space) as real
  redim R as real
  R = f_N(V, a, b, e, s) / f_D(V, a, b, e, s) * (V.z - e.LN * b)
   * sin(a)
  return R
end func

func f_Z(V as vect, a as real, b as real, e as ema_space, s as
sml_space) as real
  redim R as real
  R = f_N(V, a, b, e, s) / f_D(V, a, b, e, s) * (e.a - V.x *
   cos(a) - V.y * sin(a))
  return R
end func

func B_INT(R as vect, e as ema_space, s as sml_space) as vect
  redim V as vect, g as real, h as real
  g = 2 * PI / sr2
  h = 1 / sr2
  V = xVect(0, 0, 0)
  V.x = f_X(R, 0, 0, e, s) + f_X(R, 2 * PI, 0, e, s) + f_X(R, 0,
   e.N, e, s) + f_X(R, 2 * PI, e.N, e, s)
  for j = 1 to sr2 / 2 - 1
    V.x = V.x + 2 * (f_X(R, (2 * j) * g, 0, e, s) + f_X(R, (2 *
   j) * g, e.N, e, s))
  next j
  for k = 1 to sr2 * e.N / 2 - 1
    V.x = V.x + 2 * (f_X(R, 0, (2 * k) * h, e, s) + f_X(R, 2 *
     PI, (2 * k) * h, e, s))
  next k
```

```
for j = 1 to sr2 / 2
  V.x = V.x + 4 * (f_X(R, (2 * j - 1) * g, 0, e, s) + f_X(R, (2
   * j - 1) * g, e.N, e, s))
next j
for k = 1 to sr2 * e.N / 2
  V.x = V.x + 4 * (f_X(R, 0, (2 * k - 1) * h, e, s) + f_X(R, 2
   * PI, (2 * k - 1) * h, e, s))
next k
for j = 1 to sr2 / 2 - 1
  for k = 1 to sr2 * e.N / 2 - 1
    V.x = V.x + 4 * f_X(R, (2 * j) * g, (2 * k) * h, e, s)
  next k
next j
for j = 1 to sr2 / 2 - 1
  for k = 1 to sr2 * e.N / 2
    V.x = V.x + 8 * f_X(R, (2 * j) * g, (2 * k - 1) * h, e, s)
  next k
next j
for j = 1 to sr2 / 2
  for k = 1 to sr2 * e.N / 2 - 1
    V.x = V.x + 8 * f_X(R, (2 * j - 1) * g, (2 * k) * h, e, s)
  next k
next j
for j = 1 to sr2 / 2
  for k = 1 to sr2 * e.N / 2
    V.x = V.x + 16 * f_X(R, (2 * j - 1) * g, (2 * k - 1) * h,
     e, s)
  next k
next j
V.x = g * h / 9 * V.x
if abs(V.x) <= TOL then V.x = 0
V.y = f_Y(R, 0, 0, e, s) + f_Y(R, 2 * PI, 0, e, s) + f_Y(R, 0,
e.N, e, s) + f_Y(R, 2 * PI, e.N, e, s)
for j = 1 to sr2 / 2 - 1
  V.y = V.y + 2 * (f_Y(R, (2 * j) * g, 0, e, s) + f_Y(R, (2 *
   j) * g, e.N, e, s))
next j
for k = 1 to sr2 * e.N / 2 - 1
  V.y = V.y + 2 * (f_Y(R, 0, (2 * k) * h, e, s) + f_Y(R, 2 *
   PI, (2 * k) * h, e, s))
next k
for j = 1 to sr2 / 2
  V.y = V.y + 4 * (f_Y(R, (2 * j - 1) * g, 0, e, s) + f_Y(R, (2
   * j - 1) * g, e.N, e, s))
next j
for k = 1 to sr2 * e.N / 2
  V.y = V.y + 4 * (f_Y(R, 0, (2 * k - 1) * h, e, s) + f_Y(R, 2
   * PI, (2 * k - 1) * h, e, s))
next k
for j = 1 to sr2 / 2 - 1
  for k = 1 to sr2 * e.N / 2 - 1
    V.y = V.y + 4 * f_Y(R, (2 * j) * g, (2 * k) * h, e, s)
  next k
next j
```

```
for j = 1 to sr2 / 2 - 1
  for k = 1 to sr2 * e.N / 2
    V.y = V.y + 8 * f_Y(R, (2 * j) * g, (2 * k - 1) * h, e, s)
  next k
next j
for j = 1 to sr2 / 2
  for k = 1 to sr2 * e.N / 2 - 1
    V.y = V.y + 8 * f_Y(R, (2 * j - 1) * g, (2 * k) * h, e, s)
  next k
next j
for j = 1 to sr2 / 2
  for k = 1 to sr2 * e.N / 2
    V.y = V.y + 16 * f_Y(R, (2 * j - 1) * g, (2 * k - 1) * h,
     e, s)
  next k
next j
V.y = g * h / 9 * V.y
if abs(V.y) <= TOL then V.y = 0
V.z = f_Z(R, 0, 0, e, s) + f_Z(R, 2 * PI, 0, e, s) + f_Z(R, 0,
e.N, e, s) + f_Z(R, 2 * PI, e.N, e, s)
for j = 1 to sr2 / 2 - 1
  V.z = V.z + 2 * (f_Z(R, (2 * j) * g, 0, e, s) + f_Z(R, (2 *
   j) * g, e.N, e, s))
next j
for k = 1 to sr2 * e.N / 2 - 1
  V.z = V.z + 2 * (f_Z(R, 0, (2 * k) * h, e, s) + f_Z(R, 2 *
   PI, (2 * k) * h, e, s))
next k
for j = 1 to sr2 / 2
  V.z = V.z + 4 * (f_Z(R, (2 * j - 1) * g, 0, e, s) + f_Z(R, (2
   * j - 1) * g, e.N, e, s))
next j
for k = 1 to sr2 * e.N / 2
  V.z = V.z + 4 * (f_Z(R, 0, (2 * k - 1) * h, e, s) + f_Z(R, 2
   * PI, (2 * k - 1) * h, e, s))
next k
for j = 1 to sr2 / 2 - 1
  for k = 1 to sr2 * e.N / 2 - 1
    V.z = V.z + 4 * f_Z(R, (2 * j) * g, (2 * k) * h, e, s)
  next k
next j
for j = 1 to sr2 / 2 - 1
  for k = 1 to sr2 * e.N / 2
    V.z = V.z + 8 * f_Z(R, (2 * j) * g, (2 * k - 1) * h, e, s)
  next k
next j
for j = 1 to sr2 / 2
  for k = 1 to sr2 * e.N / 2 - 1
    V.z = V.z + 8 * f_Z(R, (2 * j - 1) * g, (2 * k) * h, e, s)
  next k
next j
for j = 1 to sr2 / 2
  for k = 1 to sr2 * e.N / 2
    V.z = V.z + 16 * f_Z(R, (2 * j - 1) * g, (2 * k - 1) * h,
     e, s)
  next k
next j
```

```
  V.z = g * h / 9 * V.z
  if abs(V.z) <= TOL then V.z = 0
  magSt V
  return V
end func

func B_CAL(R as vect, s as sml_space) as vect
  redim V as vect
  V = xVect(0, 0, 0)
  for Z = 1 to eMAX
    V = aVect(V, B_INT(sVect(R, ema(Z).R1), ema(Z), s))
  next Z
  magSt V
  return V
end func

void f_B(file$, s as sml_space)
  redim V as vect
  open file$ for binary as #1
  seek #1, 1: tp$ = mkn(s.N): put #1, , tp$
  for i = 0 to s.N - 1
    for j = 0 to s.N - 1
      for k = 0 to s.N - 1
        V = xVect(0, 0, 0)
        V.x = (s.R1.x - (s.M.x / 2)) + s.MN.x * (i + 0.5)
        V.y = (s.R1.y - (s.M.y / 2)) + s.MN.y * (j + 0.5)
        V.z = (s.R1.z - (s.M.z / 1)) + s.MN.z * (k + 0.5)
        magSt V
        write_data_v B_CAL(V, s), 1, s.N, i, j, k
      next k
    next j
  next i
  close #1
end void
```

CALU Module:

```
func U_CAL(V as vect, s as sml_space) as real
  redim R as real
  R = -s.x1 / (u0 * (1 + s.x1)) * s.MN.x * s.MN.y * s.MN.z * V.m
   * V.m
  return R
end func

void f_U(fileI$, fileO$, s as sml_space)
  redim V as vect
  open fileI$ for binary as #1
  open fileO$ for binary as #2
  seek #2, 1: tp$ = mkn(s.N): put #2, , tp$
  for i = 0 to s.N - 1
    for j = 0 to s.N - 1
      for k = 0 to s.N - 1
        read_data_v V, 1, s.N, i, j, k
        write_data_r U_CAL(V, s), 2, s.N, i, j, k
      next k
    next j
  next i
  close #2
  close #1
end void
```

CALF Module:

```
func F_CAL(R as vect, A as vect, B as vect, C as vect, s as
sml_space) as vect
  redim V as vect
  V = xVect(0, 0, 0)
  V.x = 2 * s.x1 / (u0 * (1 + s.x1)) * s.MN.y * s.MN.z * (R.x *
   (R.x - A.x) + R.y * (R.y - A.y) + R.z * (R.z - A.z))
  if abs(V.x) <= TOL then V.x = 0
  V.y = 2 * s.x1 / (u0 * (1 + s.x1)) * s.MN.x * s.MN.z * (R.x *
   (R.x - B.x) + R.y * (R.y - B.y) + R.z * (R.z - B.z))
  if abs(V.y) <= TOL then V.y = 0
  V.z = 2 * s.x1 / (u0 * (1 + s.x1)) * s.MN.x * s.MN.y * (R.x *
   (R.x - C.x) + R.y * (R.y - C.y) + R.z * (R.z - C.z))
  if abs(V.z) <= TOL then V.z = 0
  magSt V
  return V
end func

void f_F(fileI$, fileO$, s as sml_space)
  redim V as vect, A as vect, B as vect, C as vect
  open fileI$ for binary as #1
  open fileO$ for binary as #2
  seek #2, 1: tp$ = mkn(s.N): put #2, , tp$
  for i = 0 to s.N - 1
    for j = 0 to s.N - 1
      for k = 0 to s.N - 1
        read_data_v V, 1, s.N, i, j, k
        if i = 0 then read_data_v A, 1, s.N, s.N - 2, j, k else
         read_data_v A, 1, s.N, i - 1, j, k
        if j = 0 then read_data_v B, 1, s.N, i, s.N - 2, k else
         read_data_v B, 1, s.N, i, j - 1, k
        if k = 0 then read_data_v C, 1, s.N, i, j, 0       else
         read_data_v C, 1, s.N, i, j, k - 1
        write_data_v F_CAL(V, A, B, C, s), 2, s.N, i, j, k
      next k
    next j
  next i
  close #2
  close #1
end void
```

CALS Module:

```
func S_CAL(n as nmbr, s as sml_space) as vect
  redim V as vect, R as vect
  V = xVect(0, 0, 0)
  for i = 0 to s.N - 1
    for j = 0 to s.N - 1
      for k = 0 to s.N - 1
        read_data_v R, n, s.N, i, j, k
        V.x = V.x + R.x
        if abs(V.x) <= TOL then V.x = 0
        V.y = V.y + R.y
        if abs(V.y) <= TOL then V.y = 0
        V.z = V.z + R.z
        if abs(V.z) <= TOL then V.z = 0
      next k
    next j
  next i
  magSt V
  return V
end func

void f_S(fileI$, fileO$, s as sml_space)
  redim V as vect
  open fileI$ for binary as #1
  open fileO$ for output as #2
  V = S_CAL(1, s)
  print #2, "** Net Force Calculator **"
  print #2, "--------------------------"
  print #2, "Fx  -> "; V.x
  print #2, "Fy  -> "; V.y
  print #2, "Fz  -> "; V.z
  print #2, "|F| -> "; V.m
  close #2
  close #1
end sub
```

CALCCA Module:

```
func BF_CAL(n1 as nmbr, n2 as nmbr, s as sml_space) as vect
  redim V as vect, A as vect, B as vect, C as vect
  seek #n1, 1: tp$ = mkn(s.N): put #n1, , tp$
  for i = 0 to s.N - 1
    for j = 0 to s.N - 1
      for k = 0 to s.N - 1
        V = xVect(0, 0, 0)
        V.x = (s.R1.x - (s.M.x / 2)) + s.MN.x * (i + 0.5)
        V.y = (s.R1.y - (s.M.y / 2)) + s.MN.y * (j + 0.5)
        V.z = (s.R1.z - (s.M.z / 1)) + s.MN.z * (k + 0.5)
        magSt V
        write_data_v B_CAL(V, s), n1, s.N, i, j, k
      next k
    next j
  next i
  seek #n2, 1: tp$ = mkn(s.N): put #n2, , tp$
  for i = 0 to s.N - 1
    for j = 0 to s.N - 1
      for k = 0 to s.N - 1
        read_data_v V, n1, s.N, i, j, k
        if i = 0 then read_data_v A, 1, s.N, s.N - 2, j, k else
         read_data_v A, 1, s.N, i - 1, j, k
       if j = 0 then read_data_v B, 1, s.N, i, s.N - 2, k else
         read_data_v B, 1, s.N, i, j - 1, k
       if k = 0 then read_data_v C, 1, s.N, i, j, 0       else
         read_data_v C, 1, s.N, i, j, k - 1
        write_data_v F_CAL(V, A, B, C, s), n2, s.N, i, j, k
      next k
    next j
  next i
  V = S_CAL(n2, s)
  return V
end func

void I_rampD(dlt as nmbr, s as sml_space)
  for Z = 1 to eMAX
    if (abs(ema(Z).R1.x - s.R1.x) / eBSU <= dlt) or
    (abs(ema(Z).R1.y - s.R1.y) / eBSU <= dlt) then
      ema(Z).I1 = ema(Z).I1 * omega0
    else
      ema(Z).I1 = 0
    end if
  next Z
end void
```

```
void I_rampU(dlt as nmbr, s as sml_space)
  for Z = 1 to eMAX
    if (abs(ema(Z).R1.x - s.R1.x) / eBSU <= dlt) or
     (abs(ema(Z).R1.y - s.R1.y) / eBSU <= dlt) then
      ema(Z).I1 = ema(Z).I1 / omega0
    else
      ema(Z).I1 = 0
    end if
  next Z
end void

void I_setup(dlt as nmbr, s as sml_space)
  for Z = 1 to eMAX
    if (abs(ema(Z).R1.x - s.R1.x) / eBSU <= dlt) or
     (abs(ema(Z).R1.y - s.R1.y) / eBSU <= dlt) then
      ema(Z).I1 = ema(Z).Im * sigma1
    else
      ema(Z).I1 = 0
    end if
  next Z
end void

void swap_e()
  for Z = 1 to eMAX
    swap ema(Z).I0, ema(Z).I1
  next Z
end void

void swap_s(s as sml_space)
  swap s.R0.x, s.R1.x
  swap s.R0.y, s.R1.y
  swap s.R0.z, s.R1.z
  swap s.R0.m, s.R1.m
end void

void f_CCA(file$, s as sml_space)
  redim V as vect, xMAX as real, yMAX as real, cnt as nmbr, dlt
   as nmbr, CCA as nmbr
  cnt = -1
  dlt = -1
  CCA = 0
  swap_e
  swap_s s
  open "CCA-B.txt" for binary as #1
  open "CCA-F.txt" for binary as #2
  open file$ for output as #3
  do
    dlt = dlt + 1
    I_setup dlt, s
    V = BF_CAL(1, 2, s)
  loop until (abs(V.z)>= s.W * alpha0 or dlt = 1000)
  I_setup dlt, s
  do
    cnt = cnt + 1
    I_rampD dlt, s
    V = BF_CAL(1, 2, s)
  loop until (abs(V.z) < s.W * alpha0 or cnt = 1000)
```

```
  I_rampU dlt, s
  sBSU = 0
  for i = 0 to s.N - 1
    for j = 0 to s.N - 1
      for k = 0 to s.N - 1
        V = xVect(0, 0, 0)
        V.x = (s.R1.x - (s.M.x / 2)) + s.MN.x * (i + 0.5)
        V.y = (s.R1.y - (s.M.y / 2)) + s.MN.y * (j + 0.5)
        V.z = (s.R1.z - (s.M.z / 1)) + s.MN.z * (k + 0.5)
        magSt V
        V = B_CAL(V, s)
        if abs(V.m)> abs(sBSU) then sBSU = V.m
      next k
    next j
  next i
  swap s.x0, s.x1
  i = -1: j = -1: k = 0
  do
    i = i + 1
    V = xVect(0, 0, 0)
    V.x = (s.M.x / 2) + s.MN.x * (i + 0.5)
    V.y = 0
    V.z = -s.MN.z * 0.5
    magSt V
    V = B_CAL(V, s)
  loop until (abs(V.m) < abs(sBSU) * sigma2)
  xMAX = abs((s.M.x / 2) + s.MN.x * (i + 0.5))
  do
    j = j + 1
    V = xVect(0, 0, 0)
    V.x = 0
    V.y = (s.M.y / 2) + s.MN.y * (j + 0.5)
    V.z = -s.MN.z * 0.5
    magSt V
    V = B_CAL(V, s)
  loop until (abs(V.m) < abs(sBSU) * sigma2)
  yMAX = abs((s.M.y / 2) + s.MN.y * (j + 0.5))
  swap s.x0, s.x1
  if xMAX > yMAX then sBSU = xMAX else sBSU = yMAX
  dlt = sBSU / eBSU
  if dlt / 2 = int(dlt / 2) then dlt = dlt + 1
  CCA = 2 * dlt + 1
  print #3, "** CCA Calculator **"
  print #3, "---------------------"
  print #3, "delta -> "; dlt
  print #3, " CCA  -> "; CCA
  print #3, "";
  print #3, "count -> "; cnt - 1
  close #3
  close #2
  close #1
  kill "CCA-B.txt"
  kill "CCA-F.txt"
  swap_e
  swap_s s
end void
```

REFERENCES

1. N. M. Ravindra, R. D. Rivero, A. T. Fiory, M. R. Booty, "An Intermediate Template Method for Magnetic Field Assisted Assembly", US Patent Provisional Application 61/152,502 (2009).
2. S. Shet, V. R. Mehta, A. T. Fiory, M. P. Lepselter, N. M. Ravindra, *J. Miner. Met. Mater. Soc.* **56** 32-34 (2004).
3. F. Roozeoom, W. Dekkers, Y. Lamy, J. H. Klootwijk, E. van Grunsven, H.-D. Kim, *Solid State Technology* 38 (2008).
4. H.-J. J. Yeh, J. S. Smith, *IEEE Photon. Tech. Lett.* **6** 706 (1994).
5. R. D. Rivero, S. Shet, M. R. Booty, A. T. Fiory, N. M. Ravindra, *J. Electron. Mater.* **37** 374-378 (2008).
6. "Pacepacker Pick & Place Solutions", http://www.pacepacker-services.co.uk / products / pick-and-place-details.htm; Internet; accessed 1 December 2010.
7. R. Fischer, *Electronic Letters* **20** 945–947 (1985).
8. H. P. Lee, X. Liu, S. Wang, *Appl. Phys. Lett.* **56** 1014–1016 (1990).
9. D. G. Deppe, *Appl. Phys. Lett.* **56** 740–742 (1990).
10. S. F. Fang, *J. Appl. Phys.* **68**(**7**) R31–R58 (1990).
11. J. Salmi, J. Salonen, *Workshop on Bonding and Die Attach Technologies*, CERN (2003).
12. R. Pu, *IEEE Photon. Tech. Lett.* **9**(**12**) 1622-1624 (1997).
13. A. Singh, *International Conference on Solid-State Sensors and Actuators* 265–268 (1997).
14. "Sacrificial-Etch Fabrication Process for CMUTs", http://www-kyg.stanford.edu / khuriyakub / opencms / en / research / fabrication_and_packaging / Bulk_Micromachining / index.html; Internet; accessed 1 December 2010.
15. D. L. Mathine, R. Droopad, G. N. Maracas, *IEEE Photon. Tech. Lett.* **9** 869–871 (1997).
16. C. Camperi-Ginestet, Y. W. Kim, N. M. Jokerst, M. G. Allen, M. A. Brooke, "Three Dimensional Integrated Circuits: Epitaxial Lift Off GaAs Photodetectors Integrated Directly On Top Of Silicon Circuits", http://www.ee.duke.edu / ~mbrooke / papers / 1999 / 00697487.pdf; Internet; accessed 1 December 2010.
17. E. Yablonovitch, *Appl. Phys. Lett.* **51**(**26**) 2222–2224 (1987).
18. M. C. Hargis, *IEEE Photon. Tech. Lett.* **5**(**10**) 1210–1212 (1993).

19. A. Bavin, "Cluster Tool Sputtering for Compound Semiconductors", http://www.electroiq.com / index / display / semiconductors-article-display / 107519 / articles / solid-state-technology / volume-44 / issue-7 / features / materials / cluster-tool-sputtering-for-compound-semiconductors.html; Internet; accessed 1 December 2010.

20. S. C. Esener, D. Hartmann, M. J. Heller, J. M. Cable, *Critical Reviews of Optical Engineering*, edited by Anis Husain and Mahmoud Fallahi (San Jose, CA: SPIE Optical Engineering Press) **CR70** 113-140 (1998).

21. C. Edman, *Electronic Pick and Place Technology for Molecular Electronics* (2000)

22. "ELI Tech Group EPOCH Biosciences", http://www.nanogen.com; Internet; accessed 1 December 2010.

23. J. M. Perkins, "Magnetically Assisted Statistical Assembly of III-V Heterostructures on Silicon: Initial Process and Technology Development" Master's thesis, Massachusetts Institute of Technology, Cambridge, MA (2002).

24. C. G. Fonstad, "Magnetically-Assisted Statistical Assembly – A New Heterogeneous Integration Technique", http://dspace.mit.edu / bitstream / handle / 1721.1 / 3978 / AMMNS013.pdf

25. C. G. Fonstad, *Heterogeneous Integration: Critical Reviews of Optical Engineering* edited by Elias Tone (Bellingham, WA: SPIE Optical Engineering Press) **R76** (2000).

26. Q. Ramadan, Y. S. Uk, K. Vaidyanathan, *APL* **90** 172502 (2007).

27. M. Mastrangeli, S. Abbasi, C. Varel, C. van Hoof, J-P. Celis, K. F. Bohringer, *J. Micromech. Microeng.* **19** 5-14 (2009).

28. W. Zheng, O. H. Jacobs, *Adv. Mater.* **18** 1387-138792 (2006).

29. C. Lin, F.-G. Tsent, C-C. Chieng, *J. Micromech. Microeng.* **19** 2 (2009).

30. D. I. Cheng, J. J. Rumpler II, J. M. Perkins, M. Zahn, C. G. Fonstad, *J. Appl. Phys.* **105** 2 (2009).

31. M. B. Cohn, C. J. Kim, A. P. Pisano, *IEEE International Conference on Solid-State Sensors and Actuators, TRANSDUCERS* 490-493 (1991).

32. A. K. Verma, M. A. Hadley, H. J. Yeh, J. S. Smith, *Electronic Components and Technology* 1263-1268 (1995).

33. J. J. Talghader, J. K. Tu, J. S. Smith, *IEEE Photon. Tech. Lett.* **7** 1321-1323 (1995).

34. P. Sangjun, K. F. Bohringer, *IEEE International Conference on Solid-State Sensors, Actuators, and Microsystems, TRANSDUCERS* 2079-2082 (2007).

35. P. Sangjun, K. F. Bohringer, *IEEE International Conference on Micro Electro Mechanical Systems, MEMS* 1077-1080 (2008).

36. R. Baskaran, H. Ji Hao, C. Bowen, K. F. Bohringer, *IEEE International Conference on Micro Electro Mechanical Systems,* MEMS 1069-1072 (2008).

37. S. B. Shetye, "Magnetic Self-Assembly of Small Parts" PhD thesis, University of Florida, Gainesville, FL (2009).

38. J. Tien, T. L. Breen, G. M. Whitesides, *Journal of the American Chemical Society* **120** 12670-12671 (1998).

39. D. H. Gracias, J. Tien, T. L. Breen, C. Hsu, G. M. Whitesides, *Science* **289** 1170-1172 (2000).

40. H. O. Jacobs, A. R. Tao, A. Schwartz, D. H. Gracias, G. M. Whitesides, *Science* **296** 323-325 (2002).

41. U. Srinivasan, M. A. Helmbrecht, C. Rembe, R. S. Muller, R. T. Howe, *IEEE/LEOS International Conference on Optical MEMS* 59-60 (2000).

42. U. Srinivasan, M. A. Helmbrecht, C. Rembe, R. S. Muller, R. T. Howe, *IEEE Journal of Selected Topics in Quantum Electronics* **8** 4-11 (2002).

43. U. Srinivasan, D. Liepmann, R. T. Howe, *Journal of Microelectromechanical Systems* **10** 17-24 (2001).

44. K. L. Scott, T. Hirano, H. Yang, H. Singh, R. T. Howe, A. M. Niknejad, *Journal of Microelectromechanical Systems* **13** 300-309 (2004).

45. X. Xiaorong, Y. Hanein, F. Jiandong, Y. Wang, W. Wang, D. T. Schwartz, K. F. Bohringer, *Journal of Microelectromechanical Systems* **12** 117-127 (2003).

46. J. Fang, K. F. Bohringer, *Tech. Dig. Solid-State Sensor, Actuator, and Microsystems* 208-211 (2004).

47. F. Jiandong, W. Kerwin, K. F. Bohringer, *Journal of Microelectromechanical Systems* **15** 871-878 (2006).

48. L. Sheng-Hsiung, K. Wang, K. F. Bohringer, *IEEE International Conference on Micro Electro Mechanical Systems, MEMS* 592-595 (2005).

49. Y. C. Lee, B. A. Parviz, J. A. Chiou, A. S. Chen, *IEEE Transactions on Advanced Packaging* **26** 217-226 (2003).

50. C. J. Morris, B. A. Parviz, *IEEE International Conference on Solid-State Sensors, Actuators, and Microsystems, TRANSDUCERS* 411-414 (2007).

51. C. J. Morris, B. A. Parviz, *J. Micromech. Microeng.* **18** 015022 (2008).

52. H. Onoe, K. Matsumoto, I. Shimoyama, *IEEE International Conference on Micro Electro Mechanical Systems, MEMS* 5-8 (2004).

53. H. Onoe, K. Matsumoto, I. Shimoyama, *Journal of Microelectromechanical Systems,* **13** 603-611 (2004).

54. H. Onoe, K. Matsumoto, I. Shimoyama, *Small* **3** 1383-1389 (2007).

55. W. Zheng, H. O. Jacobs, *Appl. Phys. Lett.* **85** 3635-3637 (2004).

56. R. Kneel, S. Bose, W. Zheng, H. O. Jacobs, *Materials Research Society Symposium* 313-318 (2007).

57. H. O. Jacobs, W. Zheng, *Advanced Functional Materials* **15** 732-738 (2005).

58. Z. Wei, C. Jaehoon, H. O. Jacobs, *Journal of Microelectromechanical Systems* **15** 864-870 (2006).

59. Z. Wei, C. Jaehoon, H. O. Jacobs, *IEEE International Conference on Micro Electro Mechanical Systems, MEMS* 8-11 (2005).

60. W. Zheng, P. Buhlmann, H. O. Jacobs, *Proceedings of the National Academy of Sciences* **101** 12814-12817 (2004).

61. R. Knuesel, S. Bose, W. Zheng, H. O. Jacobs, *Proceedings of NSTI-Nanotech Conference* (2007).

62. J. Fang, K. F. Bohringer, *J. Micromech. Microeng.* **16** 721-730 (2006).

63. J. Fang, K. F. Bohringer, *IEEE International Conference on Micro Electro Mechanical Systems, MEMS* 250-253 (2006).

64. J. Fang, K. F. Bohringer, *IEEE International Conference on Micro Electro Mechanical Systems, MEMS* 12-15 (2005).

65. F. Jiandong, K. F. Bohringer, *Journal of Microelectromechanical Systems* **15** 531-540 (2006).

66. J. Fang, S. Liang, K. Wang, X. Xiong, K. F. Bohringer, *Conference on Foundations of Nanoscience: Selfassembled Architectures and Devices* (2005).

67. J. Fang, K. F. Bohringer, *IEEE International Conference on Solid-State Sensors, Actuators, and Microsystems, TRANSDUCERS* 956-959 (2005).

68. E. Saeedi, S. Kim, J. R. Etzkorn, D. R. Meldrum, B. A. Parviz, *IEEE Conference on Automation Science and Engineering* 375-380 (2007).

69. S. A. Stauth, B. A. Parviz, *Proceedings of the National Academy of Sciences* **103** 13922-13927 (2006).

70. E. Saeedi, S. Kim, B. A. Parviz, *IEEE International Conference on Micro Electro Mechanical Systems, MEMS* 755-758 (2007).

71. S. S. Kim, E. Saeedi, D. R. Meldrum, B. A. Parviz, *IEEE International Conference on Nano/Micro Engineered and Molecular Systems, NEMS* 927-931 (2007).

72. K. Hosokawa, I. Shimoyama, H. Miura, *Artificial Life* **1** 413-427 (1994).

73. R. A. Syms, E. M. Yeatman, *Electronics Letters* **29** 662-664 (1993).

74. P. W. Green, R. A. Syms, E. M. Yeatman, *Journal of Microelectromechanical Systems* **4** 170-176 (1995).

75. R. A. Syms, *Journal of Microelectromechanical Systems* **4** 177-184 (1995).

76. R. A. Syms, *Journal of Microelectromechanical Systems* **8** 448-455 (1999).

77. R. A. Syms, *Electronics Letters* **35** 1157-1158 (1999).

78. R. A. Syms, *IEEE Photon. Tech. Lett.* **12** 1519-1521 (2000).

79. R. A. Syms, *IEEE Photon. Tech. Lett.* **12** 1507-1509 (2000).

80. G. W. Dahlmann, E. M. Yeatman, *IEEE International Symposium on High Performance Electron Devices for Microwave and Optoelectronic Applications* 128-133 (2000).

81. G. W. Dahlmann, E. M. Yeatman, *Electronics Letters* **36** 1707-1708 (2000).

82. G. W. Dahlmann, E. M. Yeatman, P. Young, I. D. Robertson, S. Lucyszyn, *Sensors and Actuators* **97-98** 215-220 (2002).

83. G. W. Dahlmann, E. M. Yeatman, P. R. Young, I. D. Robertson, S. Lucyszyn, *Proceedings of Microwave Symposium Digest, 2001 IEEE MTT-S International* **1** 329-332 (2001).

84. J. Tien, A. Terfort, G. M. Whitesides, "Microfabrication Through Electrostatic Self-Assembly", http://sws1.bu.edu / jtien / Tien_Langmuir1997_13_5349.pdf; Internet; accessed 1 December 2010.

85. B. A. Grzybowski, A. Winkleman, J. A. Wiles, Y. Brumer, G. M. Whitesides, *Nat Mater*. **2** 241-245 (2003).

86. K. F. Bohringer, M. Cohn, K. Goldberg, R. Howe, A. Pisano, *Proceedings of AVS National Symposium* (1997).

87. K. F. Bohringer, K. Goldberg, M. Cohn, R. Howe, A. Pisano, *IEEE International Conference on Robotics and Automation* 1204-1211 (1998).

88. T. Nakakubo, I. Shimoyama, *Sensors and Actuators* **83** 161-166 (2000).

89. E. Iwase, I. Shimoyama, *Journal of Microelectromechanical Systems* **14** 1265-1271 (2005).

90. E. Iwase, I. Shimoyama, *IEEE International Conference on Micro Electro Mechanical Systems, MEMS* 588-591 (2005).

91. E. Iwase, S. Takeuchi, I. Shimoyama, *IEEE International Conference on Micro Electro Mechanical Systems, MEMS* 188-191 (2002).

92. B. A. Grzybowski, H. A. Stone, G. M. Whitesides, *PNAS* **99** 4147-4151 (2002).

93. B. A. Grzybowski, G. M. Whitesides, *J. Phys. Chem.***106(6)** 1188-1194 (2002).

94. M. Boncheva, G. M. Whitesides, *PNAS* **102** 3924-3929 (2005).

95. C. G. Fonstad, *Advanced Materials for Micro- and Nano-Systems* (2002).

96. J. Rumpler, J. M. Perkins, C. G. Fonstad, *Conference on Lasers and Electro-Optics* **2** 2 (2004).

97. A. J. Nichol, W. J. Arora, G. Barbastathis, *J. Vac. Sci. Technol.* **24** 3128-3132 (2006).

98. J. N. Anthony, S. S. Paul, J. A. William, B. George, *Microelectron. Eng*. **84** 1168-1171 (2007).

99. S. Shet, V. R. Mehta, R. D. Rivero, M. R. Booty, A.T. Fiory, M. P. Lepselter, N. M. Ravindra, *Materials Science And Technology - Association For Iron And Steel Technology* **1** 451-474 (2006).

100. R. D. Rivero, I. Padron, M. R. Booty, A. T. Fiory, N. M. Ravindra, *Advanced Materials Research* **89-91** 431-436 (2010).

101. D. J. Griffiths, *Introduction to Electrodynamics* (Upper Sadder River, New Jersey: Prentice Hall) pp. 196-270 (1989).

102. J. D. Faires, R. Burden, *Numerical Methods* (United States: Brooks/Cole) pp. 150-161 (2003).

103. *CRC Handbook of Chemistry and Physics, 68th edition* edited by R.C. Weast (Boca Raton, Florida: CRC Press) E-118 (1988).

104. "Curie Temperature", http://en.wikipedia.org / wiki / Curie_temperature; Internet; accessed 1 December 2010.

105. Bieniosek, O. Kurnaev, A. Cherepakhin, J. Dinkel, "Development of a Beam Sweeping System for the Fermilab Antiproton Source Target", http://accelconf.web.cern.ch / accelconf / pac97 / papers / pdf / 8P010.PDF; Internet; accessed 1 December 2010.

106. E. Watkinson, "Design of Radio Frequency Inductors", http://www.ax84.com / static / rdh4 / chapte11.pdf; Internet; accessed 1 December 2010.

107. "Large Impeder Cores for Inductive Pipe Welding", http://www.ferroxcube.com / appl / info / impeders.pdf; Internet; accessed 1 December 2010.

108. R. T. Fumble, "A Computer Solution of the Parking Lot Problem", http://www.eric.ed.gov / PDFS / ED046067.pdf; Internet; accessed 1 December 2010.

109. M. Arbatskaya, K. Mukhopadhaya, E. Rasmusen, "The Parking Lot Problem", http://www.rasmusen.org / papers / parking-rasmusen.pdf; Internet; accessed 1 December 2010.

110. "Activities in Algebra", http://www.math.wichita.edu / history / activities / algebra-act.html; Internet; accessed 1 December 2010.

111. US Patent 7,737,515.

112. "FreeBASIC", http://www.freebasic.net/; Internet; accessed 1 December 2010.