ABSTRACT

MAGNETIC FIELD ASSISTED MILLI-SCALE ROBOTIC ASSEMBLY MACHINE: AN APPROACH TO MASSIVELY PARALLEL SWARM ROBOTIC AUTOMATION SYSTEMS

by
Yan Liu

The vision of highly parallel, automated manufacturing systems that can build macroscopic products by heterogeneous assembly of many small devices will have a major impact in manufacturing. In this study, a novel milli-scale robotic assembly machine with parallel capabilities, assisted with programmable magnetic field, is developed. The machine prototype consists of a 16x16 array of electromagnets. The dimensions of the electromagnets are 5mm high with an inner diameter of 1.1mm and outer diameter of 2.5mm. All the electromagnets are driven by a 16x16 array of H-Bridges, and an Arduino microcontroller is used to control and program the arrays.

Using the machine to manipulate up to nine milli-scale robots simultaneously is demonstrated. The robot is designed with a 3x3 electromagnets array to operate and it consists of two parts: a polycarbonate chassis and five grade N42 NdFeB permanent magnets located at four corners and center of the chassis.

The capability of pick-and-place millimeter size devices, such as SMD (Surface Mounted Device) LEDs (Light Emitting Diodes), specifically 0805 LEDs, is demonstrated by using the prototype machine. A milli-scale tweezer is designed using AutoCAD Fusion 360 and simulated with COMSOL Multiphysics. The milli-scale tweezer is fabricated using a home-built Computer Numerical Control (CNC) machine. The tweezer is subsequently mounted to the robots. For proof-of-concept, simultaneous operation for pick-and-place two LEDs is carried out by two milli robots.

Furthermore, an 8x8 LED array is assembled by operating a single robot, which proves the potential capability of assembling an LED screen with the presented technology.

The problems and challenges as well as the future outlook are discussed in the last chapter.

MAGNETIC FIELD ASSISTED MILLI-SCALE ROBOTIC ASSEMBLY MACHINE: AN APPROACH TO MASSIVELY PARALLEL ROBOTIC AUTOMATION SYSTEMS

by
Yan Liu

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Materials Science and Engineering

Interdisciplinary Program in Materials Science and Engineering

December 2018

APPROVAL PAGE

MAGNETIC FIELD ASSISTED MILLI-SCALE ROBOTIC ASSEMBLY MACHINE: AN APPROACH TO MASSIVELY PARALLEL SWARM ROBOTIC AUTOMATION SYSTEMS

Yan Liu

Dr. Nuggehalli M. Ravindra, Dissertation Advisor Professor of Physics, NJIT	Date
Dr. Siva Nadimpalli, Committee Member Associate Professor of Mechanical & Industrial Engineering, NJIT	Date
Dr. Eon Soo Lee, Committee Member Assistant Professor of Mechanical & Industrial Engineering, NJIT	Date
Dr. Michael Jaffe, Committee Member Research Professor of Biomedical Engineering, NJIT	Date
Dr. Anthony T. Fiory, Committee Member Research Professor (Retired) of Physics, NJIT	Date

BIOGRAPHIC SKETCH

Author:	Yan Liu
Degree:	Doctor of Philosophy
Date:	December 2018

Undergraduate and Graduate Education:

- Doctor of Philosophy in Materials Science and Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2018

- Master of Applied Physics,
  New Jersey Institute of Technology, Newark, NJ, 2012

- Bachelor of Applied Physics,
  Huazhong University of Science and Technology, Wuhan, China, 2010

Major: Materials Science and Engineering

Presentations:

Yan Liu and Nuggehalli M. Ravindra, "Magnetic Field Assisted Assembly - Modeling, Design and Implementation", Symposium on Recent Developments in Biological, Structural and Functional Thin Films and Coatings, The Minerals, Metals & Materials Society (TMS） Annual Meeting & Exhibition, San Diego, February 2017.

Yan Liu and Nuggehalli M. Ravindra, "A Magnetic Field Assisted Assembly Machine: Design and Modeling", Symposium on Advanced Manufacturing Technologies, Materials Science & Technology (MS&T) Annual Meeting, Salt Lake City, October 2016.

Yan Liu and Nuggehalli M. Ravindra, "Magnetic Field Assisted Assembly Machine", Symposium on Recent Developments in Biological, Structural and Functional Thin Films and Coatings, The Minerals, Metals & Materials Society (TMS） Annual Meeting & Exhibition, Nashville, February 2016.

Publications:

Yan Liu and Nuggehalli M. Ravindra, "A Magnetic-Field-Assisted Milli-Scale Robotic Assembly Machine: An Approach to Parallel Robotic Automation Systems", Micromachines, 9(4), 144, 2018.

Yan Liu and Nuggehalli M. Ravindra, "Magnetic Field Assisted Assembly Machine: Design and Implementation", Materials Science & Technology (MS&T), Proceedings, 2016.

This Dissertation and all my academic achievements are dedicated to my beloved wife, Xiqoqing. People think getting a PhD is hard. But compared to raising a child (and now I have two), a PhD is just a joke.

ACKNOWLEDGEMENT

I am eternally grateful for the help, guidance and support of my dissertation advisor, Dr. Nuggehalli M. Ravindra, who treated me as his son, and used his personal financial resources to partially support this work.

I thank my dissertation committee members, Dr. Siva Nadimpalli, Dr. Eon Soo Lee, Dr. Michael Jaffe and Dr. Anthony T Fiory.

During my seven years at NJIT, many friends provided help, especially Sita Rajyalaxmi Marthi and Asahel Banobre, as we always worked together and traveled together.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

Self-assembly, as a physical process and topic of research, has its origin in organic chemistry. It is already a widely-applied strategy in synthesis and fabrication. There are several reasons for interest in self-assembly. Two of the reasons are as follows: self-assembly is one of the few practical strategies for making ensembles of nanostructures so that they will be an essential part of nanotechnology; and the manufacturing and robotics will benefit from applications of self-assembly [1].

A difficulty in using self-assembly in designing systems is that equilibration is usually required to attain ordered structures. The components must be able to move with respect to others and reach their steady state positions to balance all types of interactions among them which are determined by characteristics of components, such as shape, surface properties, charge, magnetic dipole, mass etc. Molecular self-assembly usually involves weak covalent interactions, electrostatic, hydrophobic interactions and hydrogen bonds. For large scale components, self-assembly, mesoscopic or macroscopic devices, they are often needed to be assisted by external forces to make components mobile, such as gravitational force, electric fields, magnetic fields, capillary, vibration and lasers. Various assisted self-assembly or assisted assembly techniques have been proposed in the last decade, including template assisted assembly, fluid assisted assembly, electric field assisted assembly, microwave assisted assembly, laser assisted assembly and ultrasonic assisted assembly.

However, all these assisted assembly techniques depend on a statistical process, which cannot provide precise individual component assembly. The state of the art precision assembly technology, used in Micro Electro Mechanical System (MEMS) fabrication and manufacturing, relies on robotic micro assembly which utilize several types of micro-grippers to perform pick-and-place [2]. Though the latest high-yield robotic micro assembly technology has high precision, it is still a serial pick-and-place process, and cannot assemble large number of devices simultaneously compared to self-assembly [3].

Thus, we propose a new Magnetic Field Assisted Micro Robotic Assembly technique to address both the problems: precision assisted assembly and simultaneous device assembly. We combine magnetic field assisted assembly with micro robotic assembly. Magnetic Field Assisted Assembly (MFAA) is a technique for the integration of microstructures onto silicon or other semiconductor wafers [4]. It is proposed as a low-cost, efficient, and reliable technique which does not rely on statistical randomness.

CHAPTER 2

LITERATURE SURVEY, JUSTIFICATION

2.1 Self-Assembly

Self-assembly is the ultimate dream of a lazy scientist: just mix the components, and the forces of nature will assemble them into a desired structure. With this vision, countless physicists, chemists, and engineers have spent decades trying to build self-assembly systems at all scales. There are basically two kinds of self-assembly: static/equilibrium self-assembly and dynamic self-assembly [5].

Static self-assembly systems correspond to a global or local thermodynamic equilibrium during the assembly process. The static assembly process can be driven by entropy or enthalpy, or by both entropy and enthalpy. Depending on the nature of the components, the ability to self-assemble can be simplified as a process of balance of interactions at the molecular level. Table 2.1 summarizes some of the most common interactions and their scaling properties at the molecular level.

Table 2.1 Common Interactions and their Scaling Properties Used in Designing Molecular Self-Assembly [5]

Interaction	Range	Scaling relations	References
Ionic (*attractive/repulsive*)	No scale	$U \propto \pm 1/r$ (Coulomb energy). The strength is ≈ 500 kJ mol^{-1} (for two isolated ions at contact)	Ionic crystals[165]
Screened Ionic (*attractive/repulsive*)	1 nm–1μm	$U \propto \pm e^{\kappa r}/r$ where $\kappa^{-1} = (2e^2 c_s / k_B T \varepsilon_0 \varepsilon)^{-1/2}$ is the screening length ($k_B T$, thermal energy; $\varepsilon_0 \varepsilon$, dielectric permittivity of solvent; e, fundamental charge; c_s, salt concentration)	Polyelectrolytes multilayers,[166] global conformation of hammerhead ribozyme[167]
van der Waals (*attractive*)[a]	1 nm–10 nm	$U \propto -1/r^6$ (London dispersion energy). The strength is ≈ 10 kJ mol^{-1} for two alkane molecules (*e.g.*, CH_4, C_6H_6 or C_6H_{12}) in water	Peptide nanofibers,[11] dimeric surfactants[62]
Dipole dipole (*attractive/repulsive*)	0.1 nm–1 nm	$U \propto -1/r^3$ (fixed) and $U \propto -1/r^6$ (rotating, Keesom energy). The strength is ≈ 10 kJ mol^{-1} for two dipoles of strength 1D separated by 0.2 nm	Helical beta-peptides,[168] liquid crystals[169]
Hydrogen-bond (*attractive*)	0.1 nm–1 nm	$U \propto -1/r^2$ (roughly). The strength of most hydrogen-bonds is between 10 and 40 kJ mol^{-1}	Supramolecular architecture,[164] helical chains[170]
Aromatic (π–π) (*attractive*)	0.1 nm–1 nm	Arise from overlapping of p-orbitals in π-conjugated systems. Magnitude scales with the number of π-electrons. Typically the length scale of interaction is ≈ 3.4 Å	Mushroom-shaped architecture,[171] aromatic rods[172]

[a] Repulsive van der Waals interactions are possible only for systems of three materials (molecules of type A can attract a molecule of type B in medium M if the refractive index of the medium, n_M, is between those of the molecules—*e.g.*, $n_A > n_M > n_B$).

Dynamic self-assembly or non-equilibrium self-assembly depends on external forces or energy supply. Especially on the scale larger than molecules, with multiple different components, the theory of dynamic self-assembly cannot be formulated by extending equilibrium thermodynamics to non-equilibrium thermodynamics. Since the fundamental law of entropy maximization is not valid in the non-equilibrium system, the self-assembled systems can only reside and organize themselves in a low entropy state only by dissipating energy [6]. In order to design a dynamic self-assembly system, one can be guided by the following rules, based on common knowledge: by identifying suitable interactions, the system must be regulated by external delivered energy; choosing "competing" interactions to balance the attractions and repulsions to introduce selectivity into the system; choosing a proper length scale such that the attractive and repulsive forces are in similar magnitude and therefore balance each other; synthesis by bottom-up, since dynamic self-assembly systems are usually complicated processes; thus, the dynamic self-assembly system should be built from the simplest set of components and component systems.[5] The synthetic self-assembly system, assisted by external forces or energy approach, offers better ability to design various methods of dynamic self-assembly. We will illustrate various assisted self-assembly methodologies in the next section.

2.2 Assisted Self-Assembly

As we showed earlier in Table 2.1, various interactions can be involved in the static self-assembly process. However, these interactions are usually limited at the molecular level due to the nature of their interaction range. Thus, when designing a self-assembly system for larger scale, such as macroscopic or millimeter scale, those molecular level interactions will not be sufficient. One or more interactions which will work at the macroscopic scale are needed to supply energy to the self-assembly system. And those interactions usually can involve external forces; therefore, these self-assembly systems can be dynamic. Since the dynamic self-assembly systems are “Assisted” by external forces, which are not pure self-assembly, we call them Assisted self-assembly systems. Table 2.2 gives some examples of dynamic forces for assisted self-assembly.

Table 2.2 Examples of Dynamic Forces for Assisted Self-Assembly [5]

Interaction	**Addressed By**	**Particle Size**
Induced magnetic dipole-dipole	Induced by external magnetic field	0.1–100 μm
Optical binding	Intense optical fields induce long-range forces between dielectric particles	0.1–100 μm
2D vortex-vortex	Induced by rotating in a fluid create a repulsive force perpendicular to the rotational axis	μm–mm
3D vortex columns	Objects rotating above/below one another at two fluid-fluid interfaces	μm–mm
Dynamic capillary	Surface tension gradient – Marangoni effect	μm–mm
Induced electric dipole-dipole	DC and AC voltages up to 100 kHz	0.1–100 μm
Light-switch aggregation of magnetic colloids	Recently observed effect - unknown	10 nm

2.3 Magnetic Field Assisted Assembly

Amongst all the external forces, which can be used to assist a dynamic self-assembly system, magnetic fields are the most fascinating and most complex. This is because magnetic fields have a broad range of scale for manipulation from atomic size to macroscopic level and can be generated and manipulated by permanent magnets or electric circuits. Magnetic fields have tremendous potential to build assisted self-assembly systems.

Despite the fact that magnetic fields can be manipulated, it is still not easy to implement and control them. In recent years, only a few research groups have tried to build a magnetic field assisted assembly system, especially in relation to the heterogenous integration of high-performance electronic devices, microelectromechanical structures (MEMS), and optoelectronic devices onto the same substrate [7].

Qasem Ramadan et al. [8] reported a method for assembly and alignment of microcomponents on large substrate using a unique magnetic driving structure for creating an array of magnetic potential wells. This structure was called “master array” because it applies and focuses an external magnetic field onto the receptor sites on the substrate and can be fabricated for parallel assembly of many substrates.

Figure. 2.1a shows a schematic view of the magnetic assembly setup with one magnet and one chip. The assembly system comprises the following: a master array in which high aspect ratio neodymium iron boron (NdFeB) magnets are embedded (Figure 2.1b); this array has the same pattern of the template on the host substrate; host substrate on which the target chips are assembled, this substrate has physical recess array with identical pattern to that of the master array. The target chips are coated with 1 µm thick soft magnetic material film (CoNiP) using electroless plating method.

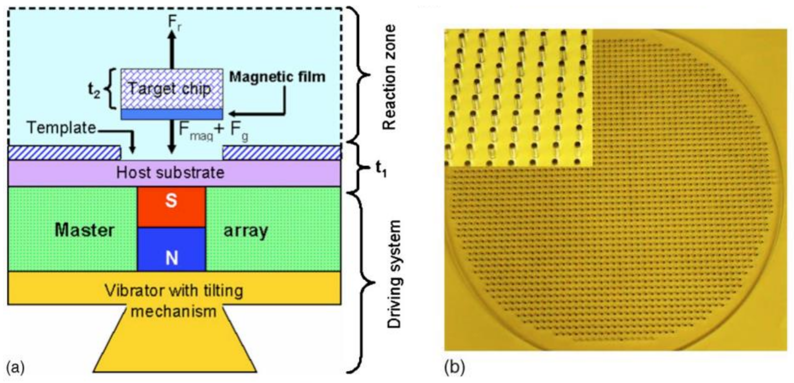


Figure 2.1 (a) Close schematic cross-sectional view of the assembly setup. (b) Magnet array of 2500 NdFeB magnets 14 mm^2 embedded in acrylic substrate 8 in. in diameter [8].

The assembly process takes place in four steps as shown in Figure 2.2: (1) Host substrate is aligned to the master magnetic array; (2) a large number of chips are randomly seeded on top of the host substrate; (3) applying vibration to the system, each individual chip tends to keep fluctuating until it is stochastically located in a close proximity to the physical cavity of the host substrate and trapped in the cavity due to the magnetic force generated by the corresponding magnet located beneath it; (4) after the final alignment of the chips is completed, the master array can be removed. A 97% assembly was achieved using the system described above. However, this magnetic assembly process, developed by Qasem Ramadan et al., rely on a vibration generator, which is a pure statistical process. Although it can assemble thousands of devices at the same time, it cannot identify/address an individual chip. All the chips need to be of the same dimensions etc.

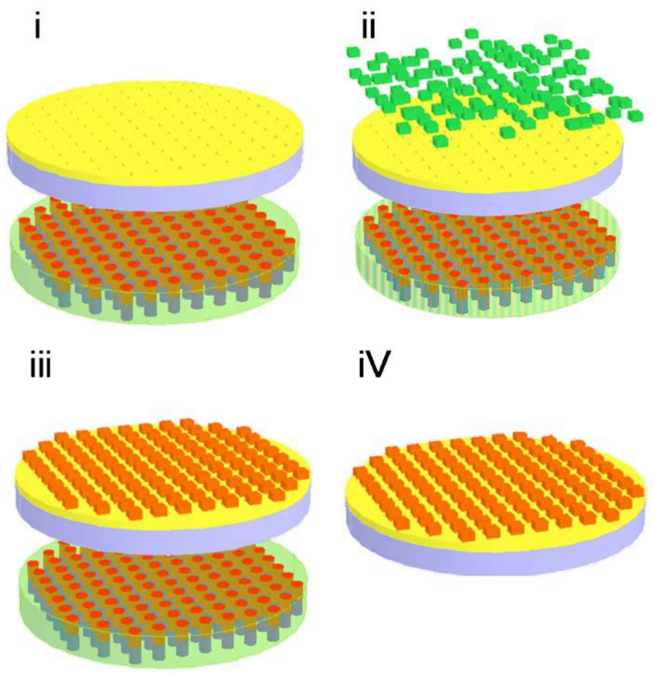


Figure 2.2 (i) Host substrate aligned to the master magnetic array (ii) chip seeding (iii) applying vibration, and (iv) final alignment of chip and master array removal [8].

E. E. Kuran and colleagues introduced a novel self-assembly method for integration of Ultra-Thin Chips on flexible polymer foils [9]. The miniaturization trend in consumer electronics has made the semiconductor industry to deliver ever smaller ICs with minimal power requirements and high accuracy. Ultra-thin chip (UTC) packaging technology enables integration of such ICs into low cost flexible substrates. However, high volume and cost-effective integration of UTCs is challenging due to their fragility: at micro-scale thickness, adhesion forces in contact manipulation causes damage to the chips and slows down the assembly process with conventional pick-and-place tools. In E. E. Kuran et al.'s work, their method uses magnetic interactions between a magnetic alignment unit placed underneath the polymer substrate and UTCs with Ni-Au bumps as shown in Figure 2.3. This aligns the chip with the applied field and drives it to the desired position. A die-attach adhesive is used as a fluid medium between the chip and the foil for mobilizing the chip and bonding it after alignment.

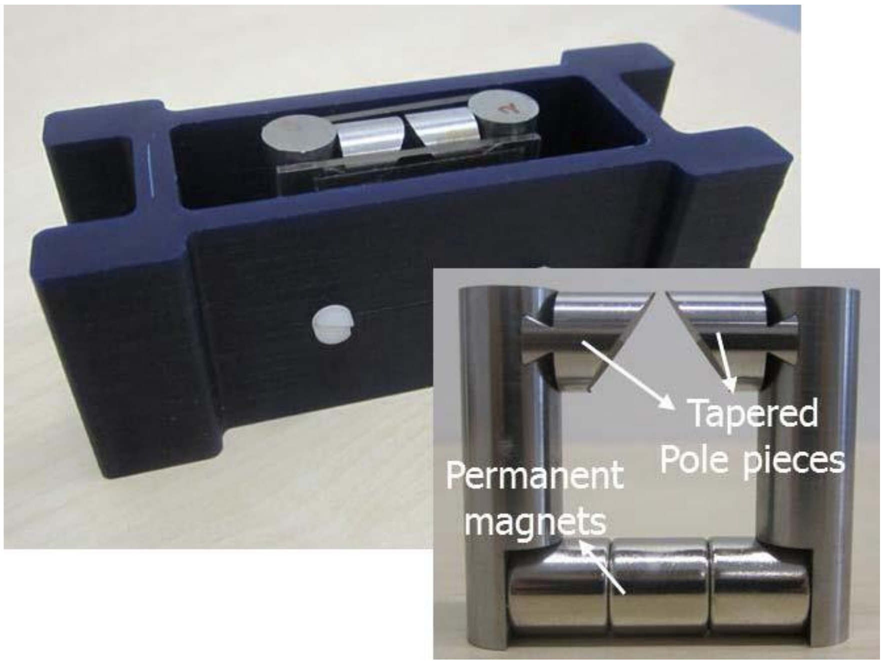


Figure 2.3 3D printed plastic holder. Inset shows the detail of the alignment unit [9].

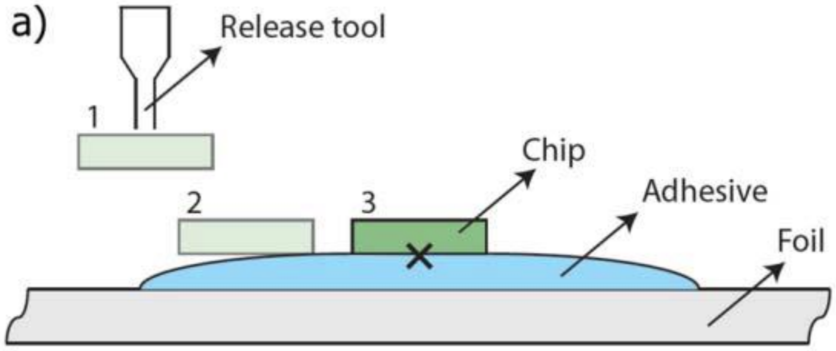


Figure 2.4 Assembly steps: Between 1-2 chip is falling and between 2-3 it is moving on the adhesive [9].

The chip assembly process take place in two steps as described in Figure 2.4. During the chip’s free fall, i.e., after the release and before the touchdown on the adhesive, the chip faces two forces: gravitational force and magnetic force. While the gravitational force and the vertical component of the magnetic force are pulling down the chip, the lateral magnetic forces dive it closer to the target position. Once the chip touches the die-attach adhesive, it also starts to experience fluidic forces. The vertical component of the surface tension and buoyancy helps to float the chip, and the shear force created by the motion of the chip on the adhesive opposes the lateral magnetic forces. Figure 2.5 shows stills from a video taken during one of the experiments using the NFC chip (STMicroelectronics M24LR64) which has a thickness of 140μm. Video taken during the experiments were used to determine the repeatability and alignment duration. The best cycle time values are 0.3 seconds for M24LR64 NFC chip. The translational alignment precision ended up with accuracies within ±100μm in x and y directions. It is worth noting that this method does not rely on a statistical process, but it also does not provide large volume assembly. Besides, it also needs a magnetic material (Ni-Au) bonded to the chip.

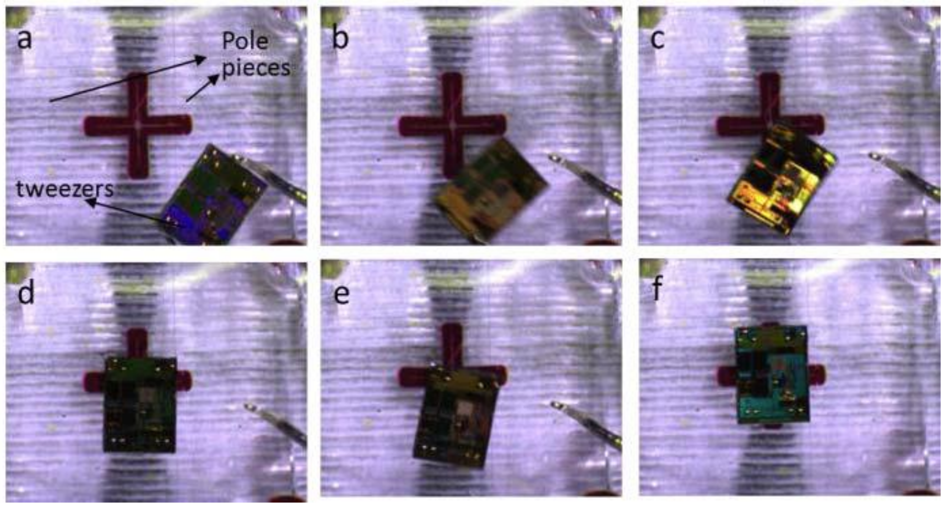


Figure 2.5 Stills from a video taken during the experiments using the NFC chip [9].

Another method of Magnetically-Assisted Statistical Assembly (MASA) was proposed by Clifton G. Fonstad Jr. for the heterogeneous integration of compound semiconductor devices (laser diodes, for example) with silicon integrated circuits. Their approach, called magnetically-assisted statistical assembly, combines statistical self-assembly with magnetic retention to locate compound semiconductor device heterostructures in shallow recesses patterned into the surface of an integrated circuit wafer. The importance of integrating different materials and different device functions, a process generally termed heterogeneous integration, is widely recognized. So too are the problems inherent in combing different materials. Principal amongst these problems is that of thermal expansion coefficient differences because the thermal expansion mismatch between silicon, the primary material of interest for large-scale high-density integrated circuits, and III-V compounds, the materials of interest for optoelectronic and microwave devices and circuits, is very large. For the most part, heterogeneous integration today is done by using some variation of flip-chip solder-ball bonding to attach modest sized arrays.

The MASA process begins with the preparation of the substrate and of the nanopills. The entire assembly process is shown schematically in Figure 2.6. Shallow recesses are patterned into the thick dielectric layers covering the wafer surface, as shown in Figure 2.6a. The depth of the recesses matches the thickness of the nanopills. A high coercivity magnetic layer is then deposited on the wafer and patterned in the bottom of the recesses. After the film is patterned, it is magnetized normal to the wafer surface, and the wafer is ready for the statistical assembly step. Formation of the nanopills begins with an epitaxial wafer. The heterostructure from which the devices being integrated are to be fabricated under optimal conditions on the optimum substrate. The heterostructures will contain an etch-free layer which can be selectively etched away to free the device heterostructures from the substrate. This epitaxial wafer is patterned into a close-packed array of cylindrical mesas, as shown in Figure 2.6b. These mesas are then etched free from their original substrate using a selective etch to form individual heterostructure device nanopills, as shown in Figure 2.6c. At some point in this processing, a thin layer of nickel is also deposited on both sides of the nanopills. During statistical assembly, the surface of a wafer, prepared as described in the first paragraph of this section, will be flooded with several orders of magnitude more nanopills than are needed to fill its recesses, as shown in Figure 2.6c. The result will be that the probability that a given recess is filled will be very nearly one, as illustrated in Figure 2.6d. The strong short-range magnetic attractive force, which will come into play when a pill settles into a recess, will keep the pill from being removed from the recess by gravity or by another nanopill or by the fluid used to flood the surface with nanopills. The process can be favorably compared to carrier trapping by deep levels in semiconductors.

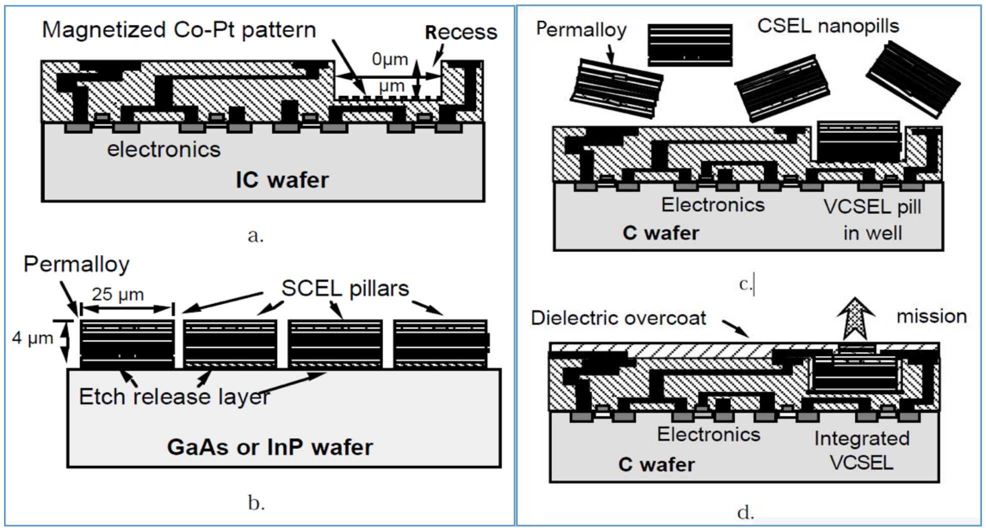


Figure 2.6 The MASA process: (a) the processed IC wafer with the recesses prepared, and (b) the p-side down device wafer (in this case VCSELs) with pillars etched in a close packed array; (c) statistical assembly of the freed nanopills in the recesses on the IC wafer; and (d) after completing device processing and integration.

The key to this MASA process is that large number of pills will mean that there are many pills near each of the recesses, and the symmetric nature of the pills will result in a high probability that a pill near a recess will fall into it. Thus, it is a purely statistical process. MASA also requires a thin layer of soft magnetic material to be deposited on the nanopill devices.

Magnetic-Field-Assisted Assembly (MFAA) is a technique that is similar to Magnetic Field Assisted Statistical Assembly (MASA) for the integration of microstructures onto silicon or other semiconductor wafers [10]. It is proposed as a low-cost, efficient, and reliable technique. The experimental approach is shown schematically in Figure 2.7. MFAA begins with the separate preparation of the substrate and micro-components. The substrate can be made from various materials, including glass, plastic, silicon, etc., depending on the desired application. For the integration of optoelectronics and MEMS devices with silicon integrated circuits, the starting substrate is an insulator, a semi-processed wafer, or a final wafer that contains the required integrated circuitry. In all cases, recesses are patterned either into the dielectric layer covering the wafer surface or into the surface of the insulator, as shown in Figure 2.7.

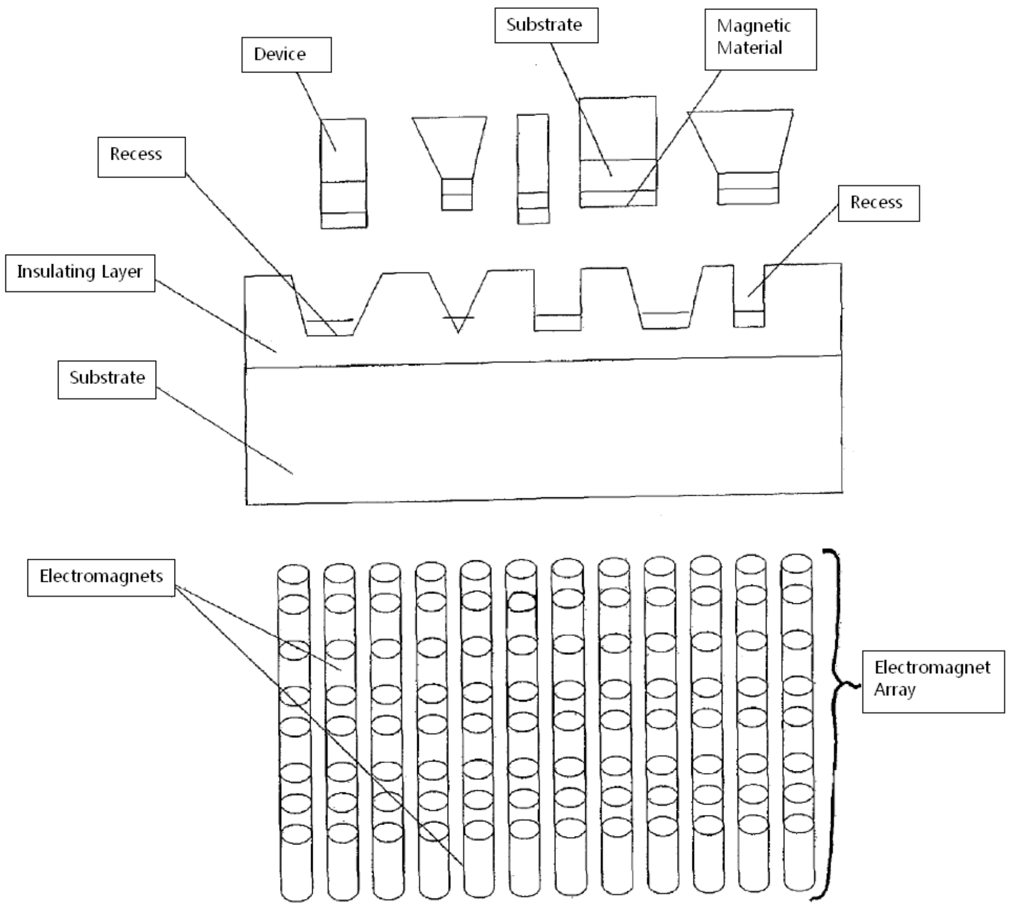


Figure 2.7. A schematic of the magnetic-field-assisted assembly method of integrating micro-components and integrated circuits [10].

The recesses are formed on the surface of the substrate in such a way that the shape and depth of the recesses match the shape and thickness of the micro-components. A highly coercive ferromagnetic material, such as cobalt or nickel, cobalt-palladium or a cobalt-platinum alloy, is deposited on the insulator substrate or wafer. The layer is patterned to form either simple or complex features at the bottom of the recesses and is subsequently magnetized to act as a host for the micro-components. During assembly, a moving magnetic field is applied on the back of the substrate; then the micro components are fixed in place as show in Figure 2.8.

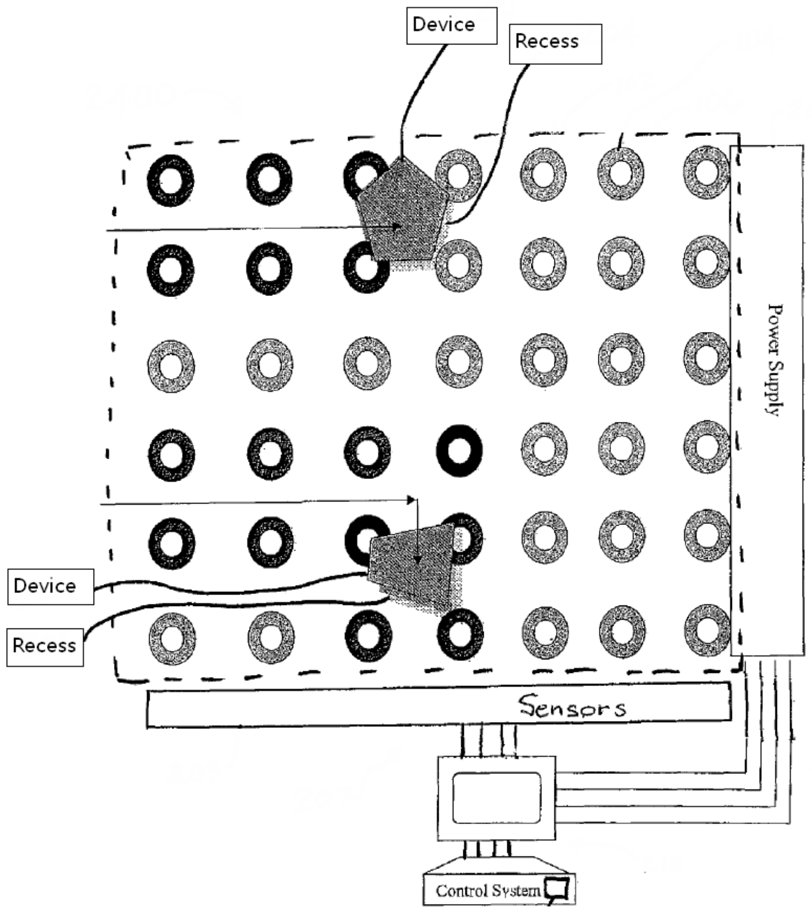


Figure 2.8 During assembly, a moving magnetic field is applied on the back of the substrate [4].

The direct magnetic-field-assisted assembly method described above does not rely on statistical randomness [11]; for example, pathways for assembling micro components can be rendered deterministic by application of moving magnetic fields. Its desirable attributes, when compared to statistical assembly, are scalability to rapid assembly of a plurality of micro components onto a host substrate and the avoidance of frustration effects that lead to assembly errors. Unfortunately, like all other magnetic field assisted assembly methods discussed above, MFAA still requires a magnetic material to be deposited or bonded to the devices. It is the main drawback of these technologies, because current industrial practices which rely on pick-and-place technique do not require devices to be magnetic. This will, therefore, lead to incompatibility of all these magnetic field assisted assembly techniques with the current technology.

2.4 Summary

We have discussed several assisted assembly methods in this chapter; each of them has some advantages and drawbacks; however, comparing them to our “dream machine”, none of them are as great as a static self-assembly system – such as life. As shown in Table 2.3, none of these magnetic assisted assembly methods can deal with non-magnetic devices; they all require devices to be magnetically activated somehow, and some of them also rely on a statistical process which may cause frustration. The current micro robotic assembly technology, which relies on pick-and-place, is a mature and reliable process; however, to attain high process yield, one must utilize several robotic assembly machines together to perform the same task, which in turn costs significant space, time, and energy.

In this dissertation, we attempt to build a “dream machine” which can perform high yield micro assembly at low costs, with precision, and does not rely on statistical randomness, and of course, does not need the devices to be magnetic.

Table 2.3 Summary of Assisted Assembly Methods

Method	**Large Scale /High Yield**	**Precision**	**Non-Statistical**	**Non-Magnetic Devices**
Static Self-assembly	√	√	√	√
Ramadan, et.al., "Largescale microcomponents assembly using an external magnetic array	√	√		
Kuran, E. E., M. Tichem, and U. Staufer. "Magnetic force driven self-assembly of ultra-thin chips."		√	√	
Fonstad Jr, Clifton G. "Magnetically-Assisted Statistical Assembly."	√	√		
Yan, Ravindra, Magnetic Field Assisted Assembly	√	√	√	
Robotic Assembly		√	√	√

CHAPTER 3

INTRODUCTION TO MAGNETIC FIELD ASSISTED MILI ROBOTIC ASSEMBLY

The vision of highly parallel, automated manufacturing systems that can build macroscopic products by heterogeneous assembly of many small devices will have a major impact in several areas of manufacturing and might potentially revolutionize the way to manufacture products.

Perhaps, the most commercially available micro assembly technique is the pick-and-place die bonding, which is widely used in printed circuit board assembly, also known as Surface Mount Technology (SMT). However, the typical size of the Surface Mount Devices (SMD), which are handled by the SMT machine, is ~ one millimeter, while the size of SMT robots itself is usually one meter. In addition to the size difference of the pick-and-place machine and the micro device, there is an inherent problem in this technology: it is a serial process. One macroscopic machine can only assemble one micro device at a time. This has significant implications on costs, space, time and energy, if there are millions, and not billions of small parts that need to be assembled. By contrast, nature provides numerous examples of parallel micro assembly, such as termite mounds, which can grow to more than 7m [12].

Various attempts have been made in the literature to build a parallel micro assembly system, with two different methodologies. One of the methodologies involves the use of external forces to deliver and/or anchor micro devices directly to receptor locations. Based on the applied external forces, these assembly techniques can be classified into four categories: (1) fluidic shape-directed self-assembly [13]; (2) capillary-driven self-assembly

[14-19]; (3) electrostatically driven self-assembly [20-21]; (4) magnetically assisted self-assembly [22]. These assisted self-assembly techniques usually have specific requirements for the micro components, i.e. to be submerged in liquid, uniquely shaped or magnetically active. Therefore, such assembly techniques have limited applications.

Another methodology is solely to overcome these limitations, which is to build robot swarms to heterogeneously assemble massive number of small devices [23-24]. In this approach, challenges are that when scaling down robots, the mechanics of microrobots are dominated by microscale physics, primarily due to their increasing surface-to-volume ratio, surface properties, forces as well as chemistry that becomes significant. These microscale forces have several different characteristics compared to that at the macroscale [25]. In addition, besides mobility and actuation, a micro robot may need power, sensors and communication devices onboard to maximize robot capabilities; these factors make it difficult to fabricate such robots; this poses a major challenge. While the capabilities of single robot are still very limited, microrobots may need to cooperate in a group, or simply need to avoid collisions between each other, which brings another challenge to organize the behavior of “swarm microrobots”, or “swarm intelligence” [26].

Various works have been reported in the literature to address these challenges. Ronald S. Fearing at UC Berkeley developed a planar milli-robot system using air bearing to levitate robot and electromagnet coil array for robot movement [27]. Bruce R. Donald at Duke University and coworkers reported a parallel microrobotic assembly scheme using MEMS microrobots; electrostatic force was used as single global control. An average docking accuracy of 5μm and final assemblies with shape match of 96% by area was reported [28]. Chytra Pawashe and coworkers at Carnegie Mellon University presented another multiple magnetic microrobot control that is achieved by an array of addressable electrostatic anchoring; magnetic robots were driven by pulsed external magnetic fields. Yet, device assembly by robots was not reported in their paper.

Perhaps, the most successful parallel robotic assembly system, involving continuous work on automated 2D micro assembly, using diamagnetically levitated milli-robots, at SRI International, has been led by Ronald E. Pelrine [29-33]. The machine, DiaMagnetic Micro Manipulator (DM3) system, is used to control many small robots. The robots are diamagnetically levitated and driven by traces in printed circuit boards. The DM3 system uses multilayer traces and one layer of diamagnetic graphite to move the robot in a manner of a linear stepper motor. The DM3 robots are made by an array of millimeter size of neodymium-iron-boron (NdFeB) magnets. Pelrine and coworkers have reported exceptional open loop repeatability of motion (200nm rms) and relative speeds of 37.5cm/s. A system using 130 micro robots, as small as 1.7mm, with densities up to 12.5 robots/cm^2 has been demonstrated. A 29 cm long cubic truss has been built using their DM3 system [34].

Despite the exceptional work at SRI and other research laboratories, robots with active arms of tweezers, while keeping the robots to be untethered, using magnetic fields, have not been demonstrated in the literature. In addition, all the robots discussed earlier, either levitated or not, were manipulated by some kind of surface and were not capable of motion under the surface or under the roof, or upside down. This is particularly important since, most of the time, small devices are needed to be assembled onto a substrate and the substrate cannot be used to manipulate robots at the same time.

As we have discussed in this chapter and earlier in the last chapter, the concept of MFAA offers many advantages, such as non-statistical process and potentially large scale parallel assembly capability; however, it has a major problem – it relies on devices that are handled by MFAA to be magnetically active. And the solution is, as discussed earlier, is to create another "layer" between the devices and MFAA – a swarm of magnetically active robots.

In this study, we present a novel robotic assembly machine with parallel control of milli-scale robots running under the ceiling with active tweezers.

CHAPTER 4

PROPOSED APPROACH TO MAGNETIC FIELD ASSISTED MILI ROBOTIC ASSEMBLY

In this chapter, we propose a new technique which can perform micro assembly with high yield at low cost, with precision, does not rely on statistical randomness, and does not need the devices to be magnetic. We combine Magnetic Field Assisted Assembly (MFAA) and Robotic Assembly technologies, together to build a Magnetic Field Assisted Micro Robotic Assembly Machine, which will also combine all the benefits of MFAA and Robotic Assembly.

The structure of the machine is briefly described in Figure 4.1 through 4.3. The key to this proposed assembly method is to use a magnetic active micro robot to carry a device, and the magnetic active robot is driven by an electromagnet array, as shown in Figure 4.1. In MFAA, as we discussed in the last chapter, devices are driven directly by the electromagnet array which in turn require devices to be magnetic active. Now if the devices are carried by magnetic active micro robot, there is no need for magnetic films or layer to be deposited on the devices. This is very important since most of the semiconductor devices have very low magnetic susceptibility and cannot be driven directly by magnetic fields, such as silicon based devices.

In order to be able to carry a device, a micro vacuum pump can be integrated with the micro robot. During the assembly process, a micro robot will be driven to the initial device location, then the micro vacuum pump will be activated to pick up the device, as shown in Figure 4.1. Then the micro robot will be driven to the designated final location, as shown in Figure 4.2. The driving route and moving speed can be pre-calculated by computation to minimize the driving time and avoid frustrations. When the robot arrives at its final location, the micro vacuum pump will be deactivated to release the device, then the robot will be able to carry the next device, as shown in Figure 4.3.

Since the electromagnet arrays are fully saleable and many micro robots can be driven at the same time, the proposed machine will be able to drive many devices simultaneously. Furthermore, the devices are driven indirectly and held to the micro robot using vacuum, which is an industrial standard for pick-and-place machines; thus, our proposed machine will be fully compatible with the current technology.

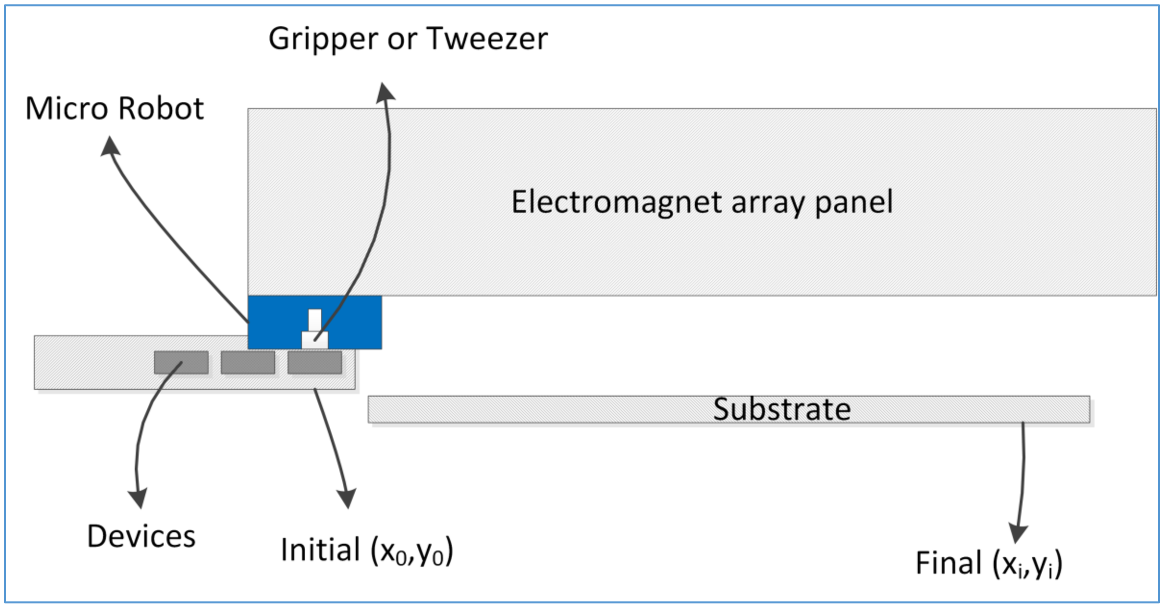


Figure 4.1 Robot picking up a device at its initial location.

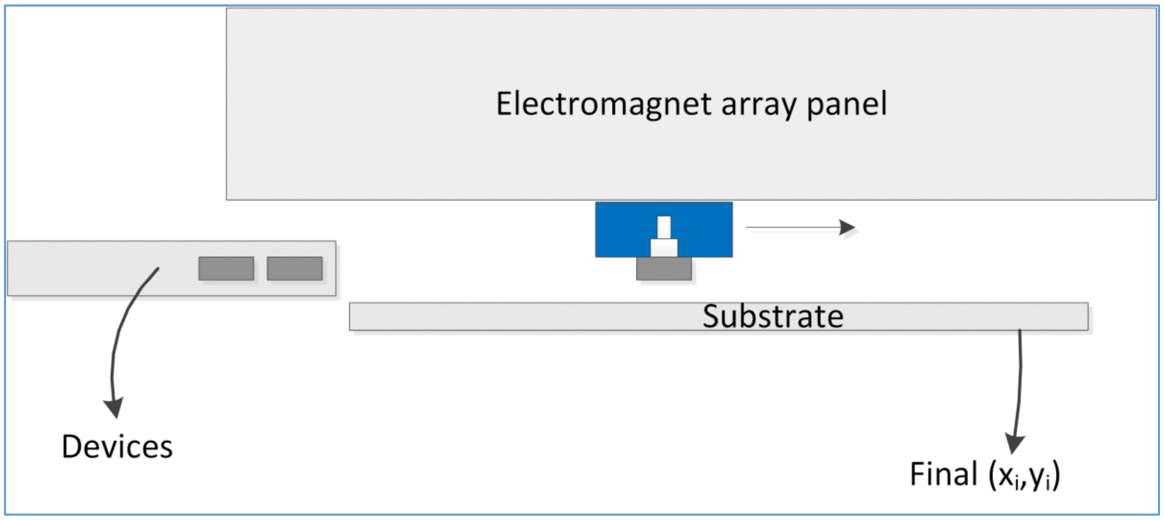


Figure 4.2 Robot carrying the device and moving to the designated location.

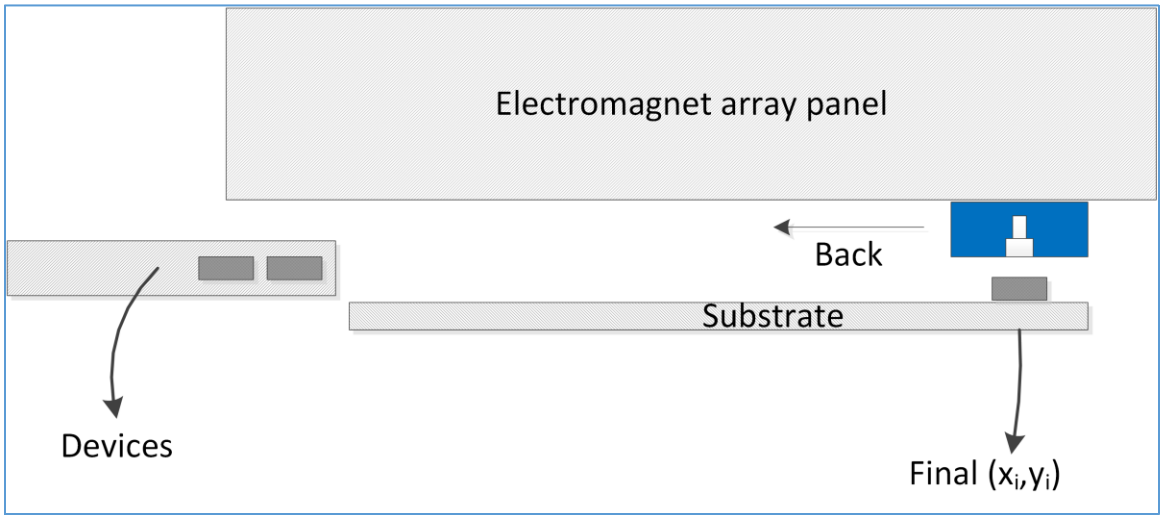


Figure 4.3 Robot arrives at the designated location and places the device onto designated location, then the robot will move on to next task.

CHAPTER 5

DESIGN AND FABRICATION OF ROBOT DRIVE SYSTEM

The design and fabrication of the robot drive system is discussed in this chapter. As we discussed earlier, the robots will be driven by an electromagnet array (16x16, with dimension of 48mm x 48mm) and are controlled individually. Further, having 256 independent small elements within a small area could potentially have random defects or problems; therefore, it is critical to properly test the system.

5.1 Robot Drive System Overview

The system architecture is shown in Figure 5.1. The drive system mainly consists of three parts: an aluminum panel with 16x16=256 electromagnets embedded in it; an array of 256 H-Bridge driver ICs which are used to power every single electromagnet; and an array of shift registers connected with an Arduino MCU to control every single electromagnet independently through the array of H-Bridge.

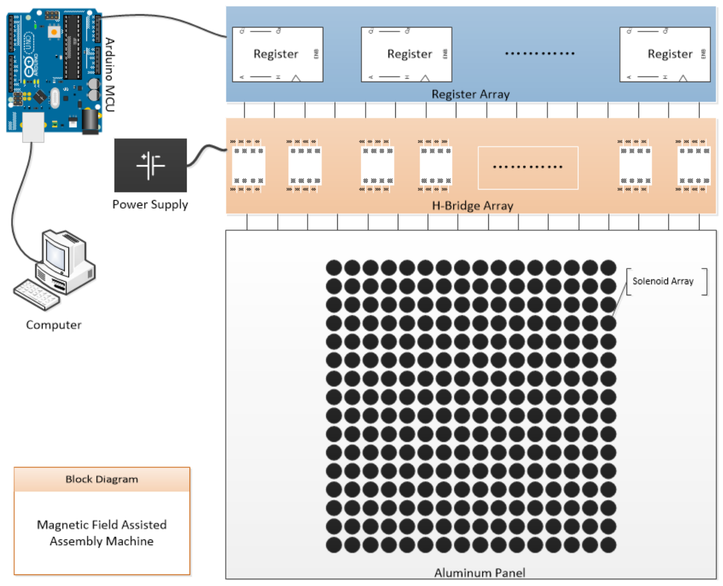


Figure 5.1. System architecture

5.2 Introduction to Arduino

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards can read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. Commands can be sent to the board by sending a set of instructions to the microcontroller on the board. This requires the use of the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Arduino was created at the Ivrea Interaction Design Institute as an easy tool for fast prototyping; it was aimed at students who do not have a background in electronics and programming. All Arduino boards are completely open-source, empowering users to implement it in a variety of applications independently and eventually adapt the boards to their needs. The software is open-source and it is growing through the contributions of users worldwide [35].

In this research, an Arduino Uno R3 board was used to control the electromagnet array. The Arduino board is equipped with an ATMEGA328p microcontroller, as shown in Figure 5.2.

Figure 5.2. Arduino UNO R3 board.

5.3 Control System and PCB Layout

The electromagnets are driven by L9110h [36] H-Bridges which is controlled by the Arduino board. 64 units of SN74HC595N shift registers [37] are cascaded and connected to the Arduino board through Serial Peripheral Interface (SPI) [38] to provide 512 output ports that are used to control all 256 H-bridges. An H-bridge is an electronic circuit that enables a voltage to be applied across a load in either direction. These circuits are often used in robotics and other applications to allow DC motors to run forwards and backwards [39]. H bridges are available as integrated circuits (IC) or can be built from discrete components. In this design, we have used DIP-8 L9110h IC with 800mA continuous current output capability. Each H-Bridge IC drives one solenoid of the 16x16 electromagnet array; therefore, we are able to switch the current direction through the solenoid. Hence switching the polarity of the magnetic field. Using the microcontroller, we are able to fully program the electromagnet array to generate the desired magnetic field distribution and change the field distribution very quickly.

The layout of the Printed Circuit Board (PCB) has been designed using Fritzing. Fritzing is an open-source hardware initiative that makes electronics accessible as a creative material for anyone. Fritzing offers a software tool, a community website and services in the spirit of Processing and Arduino, fostering a creative ecosystem that allows users to document their prototypes, share them with others, teach electronics in a classroom, and layout and manufacture professional PCBs [40].

Figure 5.3 shows the PCB layout of our system. The dimension of the board is 30cm x 20cm. Each of the boards is designed to host 32 shift registers and 128 H-Bridges; thus, with two of these boards cascaded together, we will be able to control all 256 electromagnets. The PCB is fabricated with our home-built CNC machine which can be found in Appendix A.

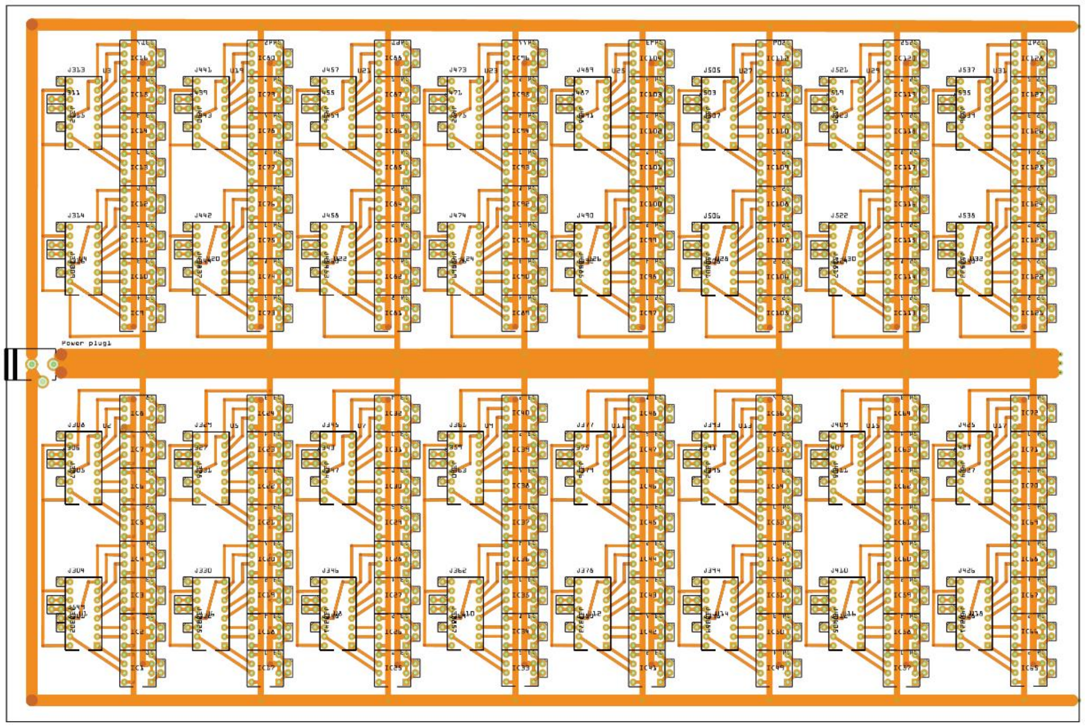


Figure 5.3 PCB layout of H-Bridge and control system.

5.4 Design and Fabrication of Electromagnets Array

The electromagnets have been designed by winding seven layers of 0.1mm diameter insulated copper wire; each layer consists of 40 turns, resulting in a total of 280 turns; the size of the electromagnets is 5mm high with an inner diameter of 1.1mm and outer diameter of 2.5mm. This is shown in Figure 5.4.

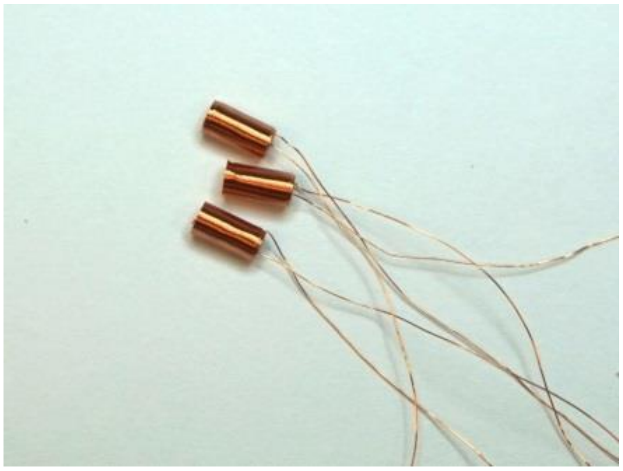


Figure 5.4 Samples of electromagnet solenoid, the size of electromagnets is 5mm high, with an inner diameter of 1.1mm and outer diameter of 2.5mm.

In order to hold all these small electromagnets into an array, an aluminum panel with 16x16 holes has been designed using Autodesk Fusion 360. It is a cloud-based CAD/CAM/CAE tool for collaborative product development. Fusion 360 combines fast and easy organic modeling with precise solid modeling, allowing the user to make designs that are manufacturable[41]. The design of the aluminum panel is shown in Figure 5.5 The dimension of the panel is 120mm x 80mm x 6.35mm, with 16x16 holes that are drilled in the center of the panel to hold the electromagnets. The diameter of the holes is 2.55mm.

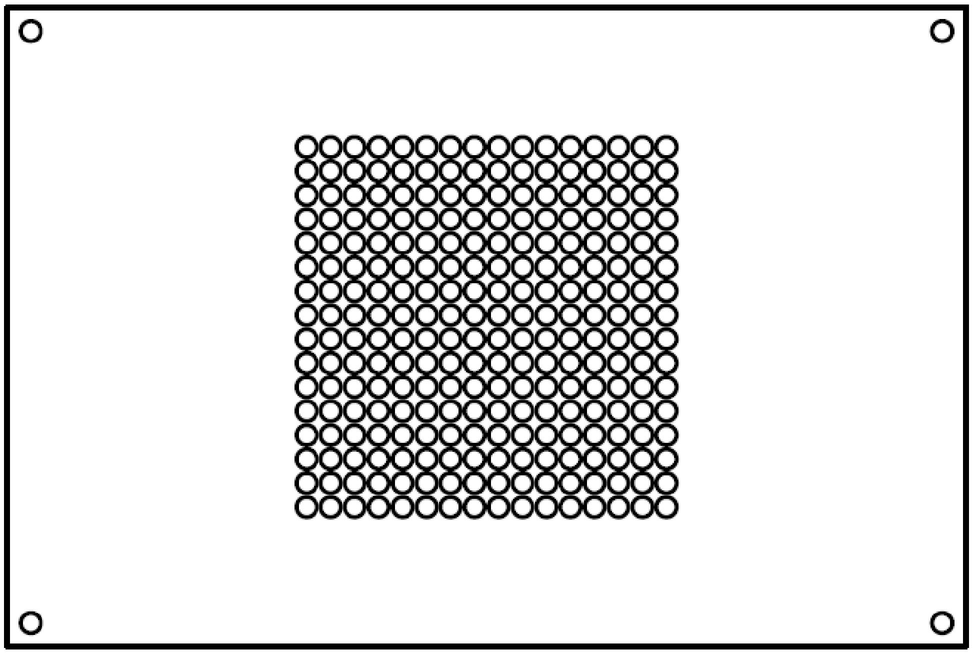


Figure 5.5 The design of the aluminum panel.

The panel is made from a 6.35mm thick T6061 aluminum sheet, with dimension of 120mmx80mm. Aluminum, as a paramagnetic material, is usually considered as a non-magnetic material due to its low magnetic susceptibility of 1.65x10^{-5}cm^{3}/mole [42]; it is very easy to manufacture and is machined subsequently. An array of 16x16 holes are milled on the aluminum panel by a CNC machine; all the holes are 2.55mm in diameter, pitch between the holes is set to be 3.0mm. Thus, an array of 256 holes, with a dimension of 48mm by 48mm, is created to hold 256 small electromagnets. The panel is machined by our home-built CNC, is shown in Figure 5.6.

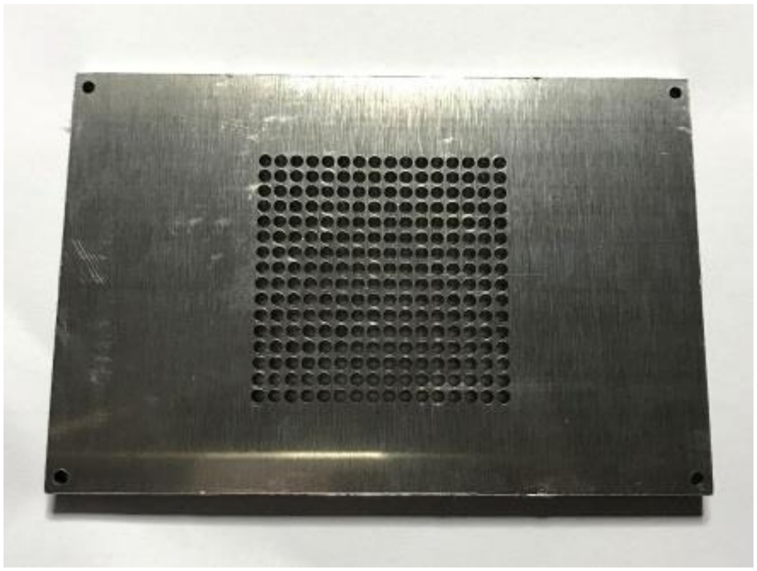


Figure 5.6 An Aluminum panel milled by a CNC machine to hold small electromagnet solenoids.

The electromagnets are formed by winding seven layers of 0.1mm diameter insulated copper wire; each layer consists of 40 turns, resulting in a total of 280 turns; the size of the electromagnets is 5mm high with an inner diameter of 1.1mm and outer diameter of 2.5mm. Then, all the 256 electromagnets are carefully assembled into the aluminum panel and glued by Loctite super glue. The assembly process is shown in Figure 5.7.

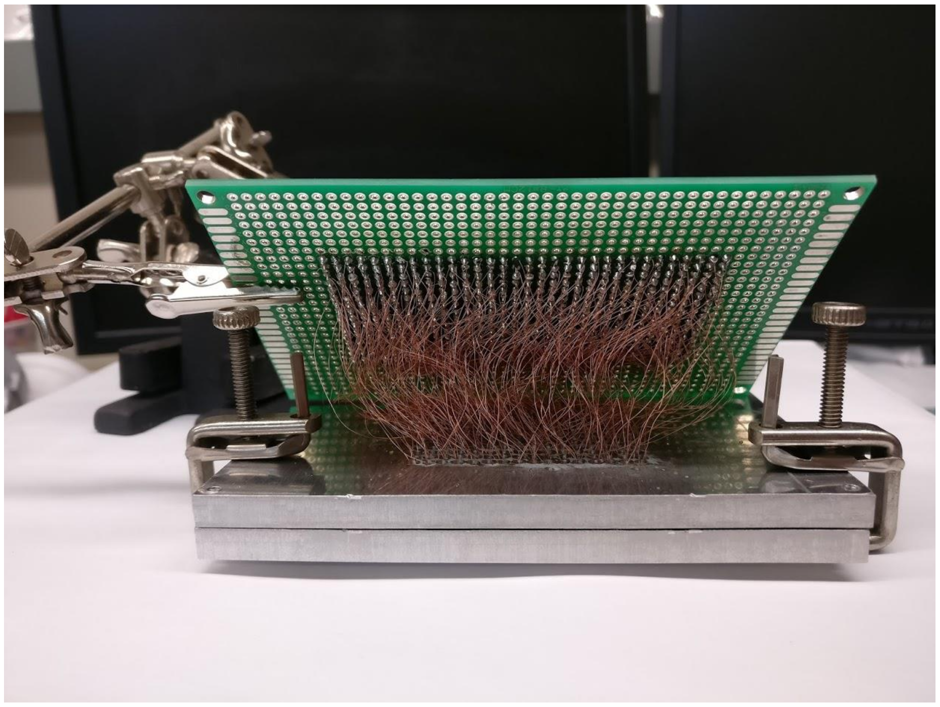


Figure 5.7 The assembly process of electromagnetic panel.

The assembled electromagnet array is as shown in Figure 5.8. A thin layer of polyester tape (green color) with 50µm (25µm polyester and 25 µm silicone adhesive) thickness is applied on top of the panel to protect from wear and reduce friction between the panel and the robots.

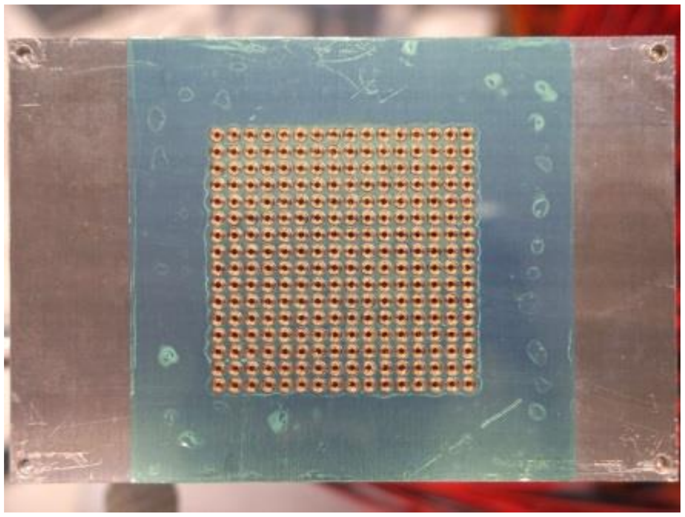


Figure 5.8 Assembled electromagnet array panel. A thin layer of polyester tape (green color) is applied on top of the panel.

5.5 Design and Fabrication of Milli-Robot

The robot is designed to use 3x3 electromagnet array to operate and consists of two parts: a polycarbonate chassis and 5 grade N42 NdFeB permanent magnets located at four corners and center of the chassis. The chassis, designed using Autodesk Fusion 360, is as shown in Figure 5.9. The dimension of the robot is designed to match the dimension of the electromagnet array panel with 3x3 matrix, with a pitch across corner of 9mm, with a thickness of 1/16 inch. The diameter of the holes is 2.5mm which is supposed to hold cylindrical NdFeB magnets with dimension of (1/10) inch diameter and (1/16) inch thickness. The robots are fabricated using a home-built CNC machine from Lexan 9034 polycarbonate sheet with a thickness of (1/16) inch [43].

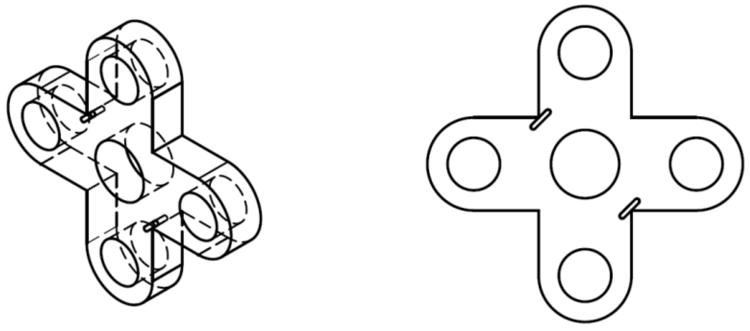


Figure 5.9 Robot chassis sketch designed using Autodesk Fusion 360.

Then 5 NdFeB permanent magnets are mounted into the holes of the polycarbonate chassis; the magnetic south is marked in black in Figure 5.10.

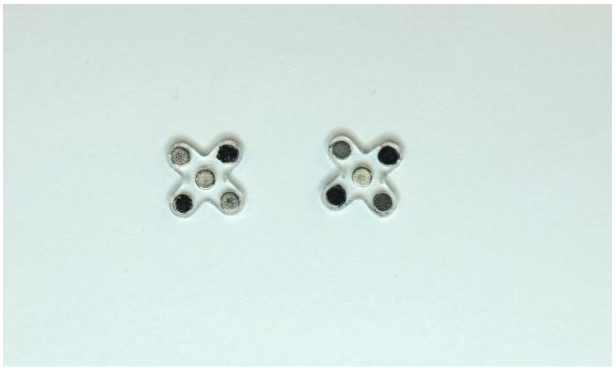


Figure 5.10 Fabricated milli-robot, the magnetic south is marked in black.

5.6 Testing Robot Drive System

The fully assembled system is shown in Figure 5.11.

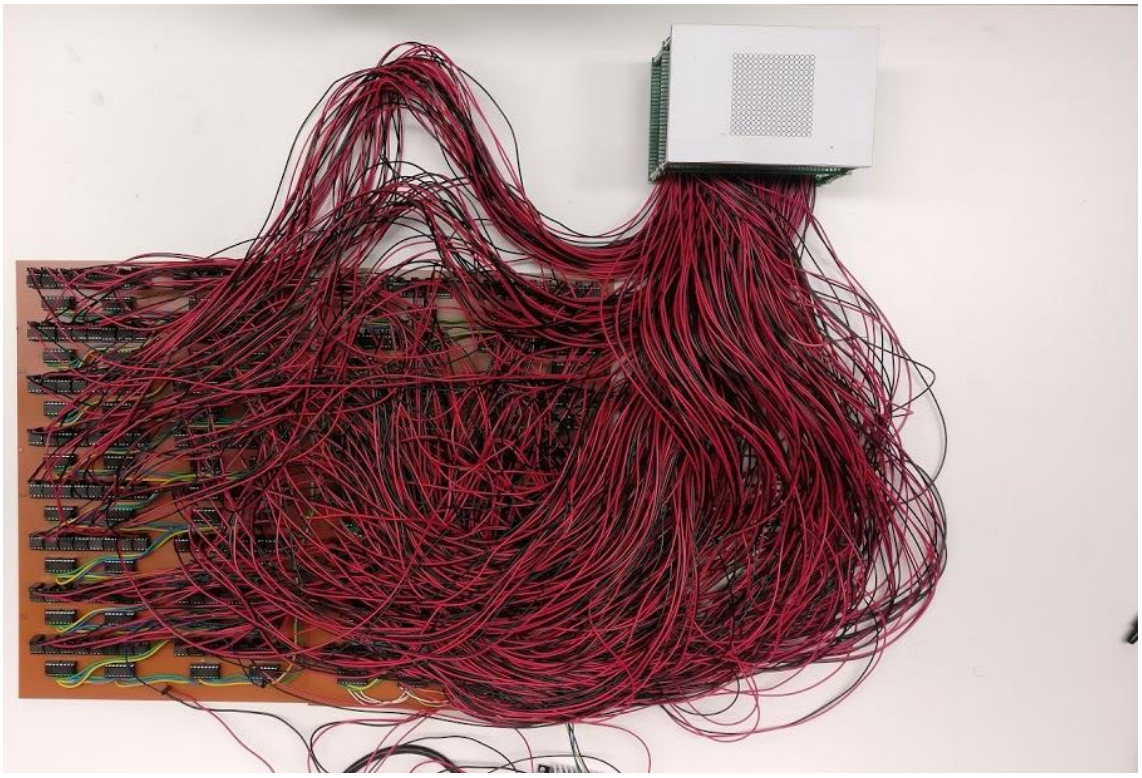


Figure 5.11 The fully assembled system.

Since there are 256 electromagnets and 512 wires to drive all of them, potential mistakes and defects could be present in the system, especially since all the assembly work has been performed by bare hands. It is critical to test the entire system before implementing it in the MFAA device.

Each H-Bridge IC drives one solenoid of the 16x16 electromagnet array; therefore, we are able to switch the direction of current through the solenoid hence switching the polarity of the magnetic field. Using the microcontroller, we are able to fully program the electromagnet array to generate the desired magnetic field distribution and change the field distribution very quickly. In Figure 5.12, the measurement of the frequency to switch the magnetic field polarity of one solenoid, using an Oscilloscope, is presented. We are able to switch the field direction at frequencies of more than 2 kHz with SPI [44] bus running at 4 MHz.

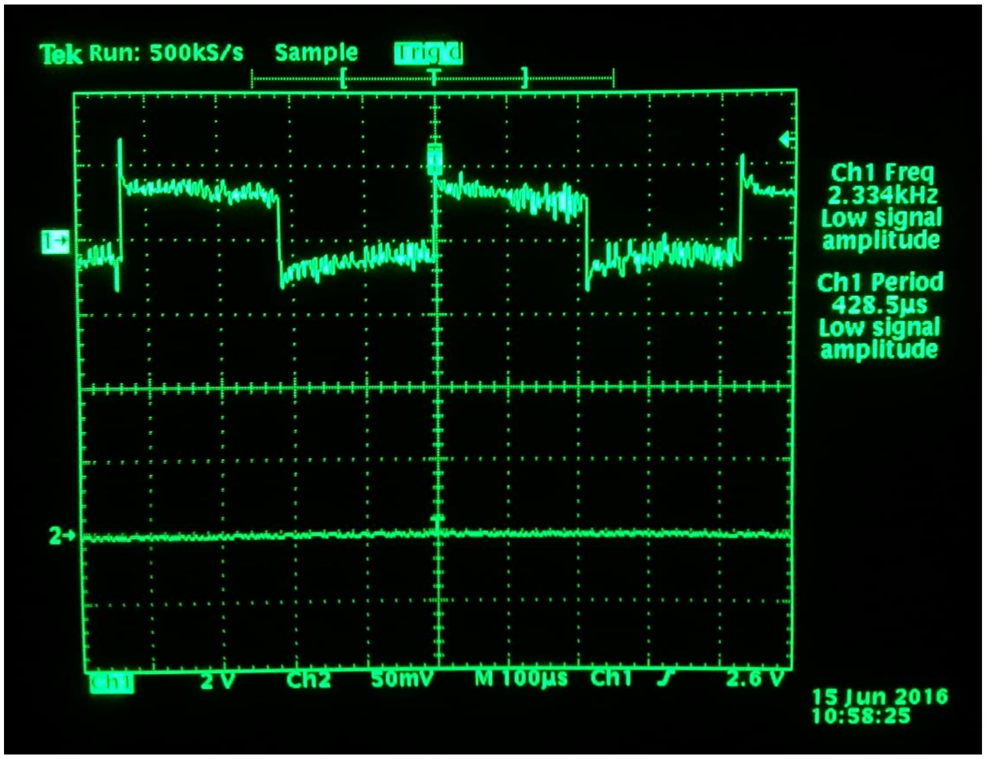


Figure 5.12 Testing Control System by measuring of the frequency to switch the magnetic field polarity of one solenoid.

For a coil with current I, ideally, the magnetic field can be obtained using Biot-Savart Law:

$$\vec{B}(\vec{r}) = \mu_0 H = \frac{\mu_0 I}{4\pi} \int_l \frac{d\vec{l} \times \vec{r}}{|\vec{r}|^3} \tag{5.1}$$

where $\mu_0 = 4\pi \times 10^{-7} N/A^2$ is the permeability of free space, and $\vec{r}$ is the vector from the current element I$d\vec{l}$ to the calculation point. For a permanent magnet with constant magnetization $\vec{M} = M_0\vec{z},$ where $\vec{z}$ is the direction out of the plane of the electromagnet array and perpendicular to the array. The magnetic force per unit volume between magnet and coil can be written as [45]:

$$d\vec{F} = \nabla(\vec{M} \cdot \vec{B}) = M_0 \nabla B \tag{5.2}$$

where B_z is the z component of the magnetic field of the coil. The total force is obtained from:

$$\vec{F} = \int_{V_0} d\vec{F} dV = \int_{V_0} M_0 \nabla B dV \tag{5.3}$$

where V_0 is the volume of the permanent magnet. The vertical force F_z will hold/anchor the permanent magnet onto the coil thus preventing it from falling down even during turning the entire machine upside down if the force is big enough; when switching the current direction, the force could push the magnet away and can be used for actuation.

The horizontal force can be obtained by:

$$F_x = \int_{V_0} \mu_0 M_0 \frac{\partial H}{\partial x} dV \tag{5.4}$$

The vertical force can be obtained by:

$$F_z = \int_{V_0} \mu_0 M_0 \frac{\partial H}{\partial z} dV \tag{5.5}$$

Given the cylindrical coil and magnet, which are symmetrical around the z direction, the magnet will be self-aligned to the coil to minimize the horizontal force. If the magnet is not aligned with the coil, for example, if the magnet is at neighboring coil location, it will be dragged towards the current coil and gets aligned with the coil. Therefore, with proper handling and manipulating the electromagnet array, a robot built with a permanent magnet array could be dragged to another location or be rotated along a certain point.

In order to test every single electromagnet properly, a program has been written to operate one single magnet (with south pole face up) to sequentially go through every point of the electromagnet array. In case of the presence of any defects in the system, the magnet should stop moving or flip the direction of the magnetic field during motion. The program to perform this task can be found in Appendix B1. Figure 5.13 is a snapshot of the frame from the test process video.

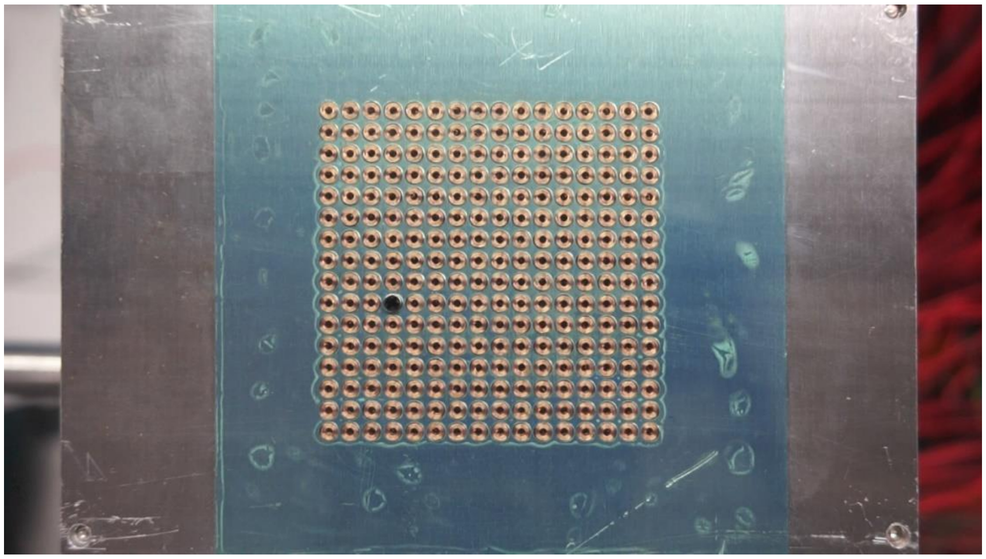
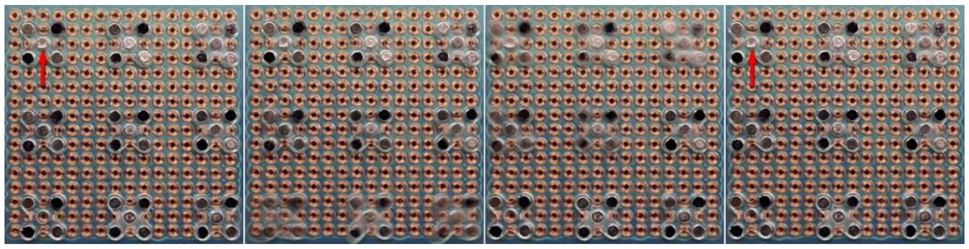


Figure 5.13 Testing the EM array by driving a single magnet.

The Magnetic Field Assisted Assembly process requires two fundamental movements of a device: linear motion and rotation, in total, three degrees of freedom. Here, we demonstrate our prototype machine to be able to move and rotate nine robots simultaneously as shown in Figure 5.14a and 5.14b. It is to be noted that we are demonstrating all nine robots to have the same movement; it does not mean that they must move the same way. The program to perform this task can be found in Appendix B2. As discussed earlier, we used the same number of H-Bridges to control all the electromagnets; so, each of the electromagnet can be individually addressed independently, which gives us the capability to independently control each robot. The bottom three robots always react to the change in magnetic field first while the top three robots are always delayed relative to the magnetic fields; this is due to the control circuit design and the control signal always gets first to the bottom electromagnets. The pictures in Figure 5.6.4 are frames taken from a video. The rotating magnetic vector can be generated using an $n \times n$ electromagnet matrix. An $n \times n$ matrix will have n^2 elements with $n^2 - (n-2)^2 = 4n - 4$ elements in the outline of the matrix. With the magnetic field vector having $4n - 4$ directions, the resolution of the rotational angle will be $\frac{360}{4n-4}$. Using a 3x3 solenoid matrix will yield an angle resolution of 45 degrees.


(**a**)

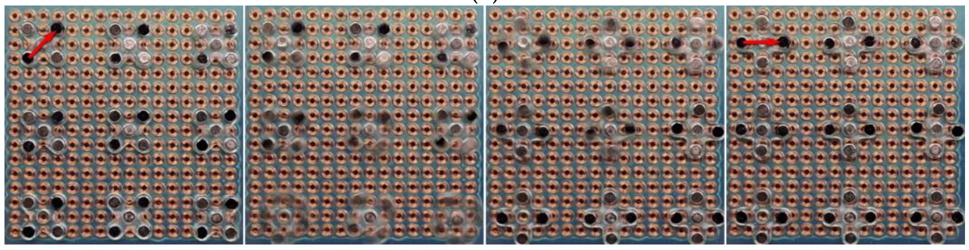

(**b**)

Figure 5.14 Movement and Rotation of nine robots simultaneously (**a**) Frames from video of moving nine robots to the next (left) locations. (**b**) Frames from video of rotating (45°) nine robots (indicated by arrows). Tweezers were not yet added to the robots.

CHAPTER 6

TWEEZER DESIGN AND FABRICATION

In this study, we aim to design and construct robots that can pick-and-place millimeter size devices, such as SMDs used in printed circuit boards, specifically the 0805 LEDs; the 0805 LED is one of the most commonly used industrial standard LED, with the dimension of 2.0mm x 1.25mm x 0.8mm (L x W x H). In order to perform the task of LED assembly, we need to create an active robot instead of the passive robots that have been tested earlier, specifically, give an active "hand" to the robot to enable it to perform the task of pick-and-place. Therefore, we need to design a magnetic field operated tweezer for the robots.

6.1 Magnetic Force Measurement in z Direction

The magnetic field utilized for robot navigation and rotation are generated by solenoids. The magnetic field acting on the NdFeB magnets will generate forces in all the directions as discussed earlier; however, in this study, we focus on the z-direction. This is because of the fact that we could utilize the force in the z-direction to actuate the tweezer by alternating the current through the solenoid. Also, the force in the z-direction is responsible for holding the robot from falling down when it is moving under the ceiling of the electromagnet array. Therefore, it is critical to determine the magnitude of the force in the z-direction acting on the NdFeB magnets at different values of current. Hence, we set up an apparatus to experimentally test the magnetic force in the z-direction, as shown in Figure 6.1. The results of these measurements are shown in Figure 6.2. Tests have been performed for three different currents (300mA, 400mA and 500mA) for distances ranging from 25µm to 600 µm between the permanent magnet and the electromagnet.

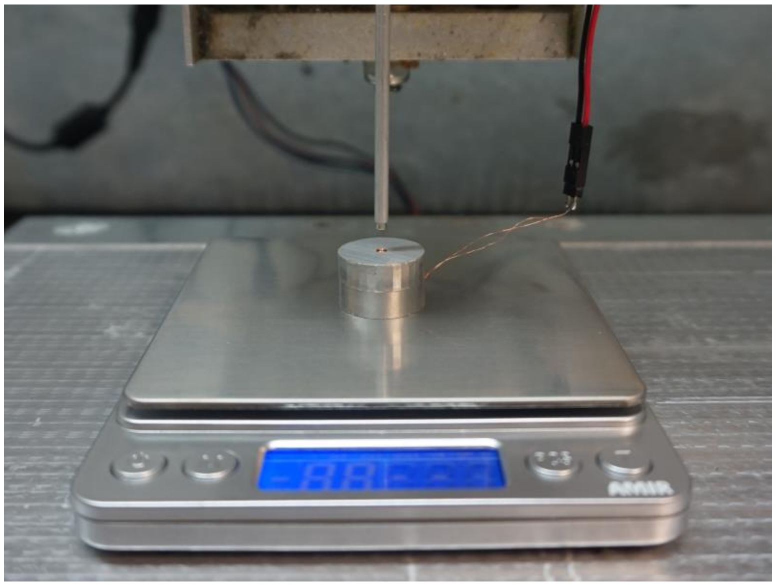


Figure 6.1 Apparatus to measure the magnetic force vs current of electromagnet coil.

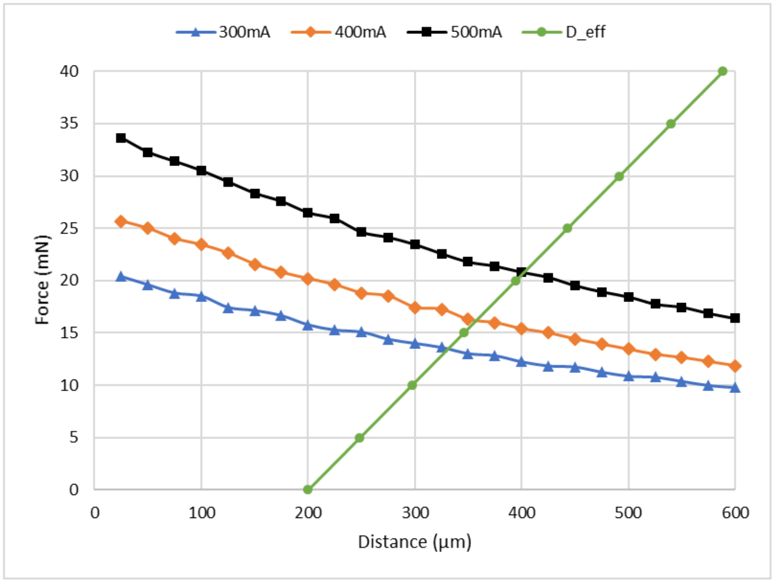


Figure 6.2. Measured results of the magnetic force vs different currents (300mA, 400mA and 500mA) of electromagnet coil. D_eff is the simulated curve of force load vs effector bar displacement (with 200µm initial distance offset).

6.2 Tweezer Design and Simulations

A model of a gripper has been developed using computer aided design (CAD) software (Autodesk Fusion 360) based on a Four bar linkage model as shown in Figure 6.3. The objective is to establish the effective motion of the Jaws. The Denavit-Hartenberg convention is applied for kinematics assuming perfect rigidity of each link and free rotation of every joint around the single degree of freedom, as illustrated in Figure 6.3.

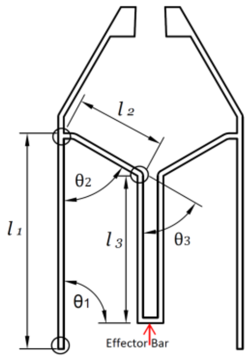


Figure 6.3. Kinematic analysis of tweezer model using Denavit-Hartenberg convention. The arrow indicates the effector bar of the tweezer which will be subjected to load force.

The homogeneous transformation matrix for the effector bar is:

$$\begin{pmatrix} c(\alpha_1) & -s(\alpha_1) & 0 & l_1c(\theta_1)+l_2c(\alpha_2)+l_3s(\alpha_1) \\ s(\alpha_1) & c(\alpha_1) & 0 & l_1s(\theta_1)+l_2s(\alpha_2)+l_3c(\alpha_1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.1}$$

where $\alpha_1 = \theta_1 + \theta_2 - \theta_3 - \pi$ and $\alpha_2 = \theta_1 + \theta_2 - \pi$ and $\theta_3 = \theta_1 + \theta_2 - \frac{\pi}{2}$. The vertical displacement of effector bar becomes:

$$y_{eff} = l_1 \sin(\theta_1) - l_2 \sin(\theta_1 + \theta_2) - l_3 \tag{6.2}$$

For vertical link l_1, i.e. $\theta_1 = 90°$, and the tweezer is perpendicular to the robot chassis, the system becomes stiff and the displacement of the effector bar becomes:

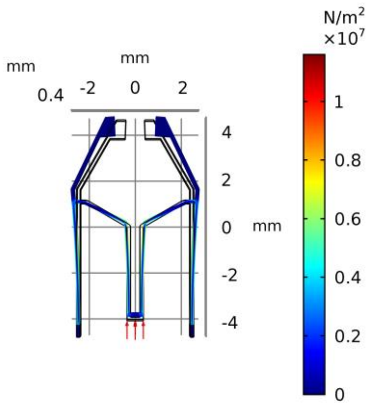
$$y_{eff} = l_1 - l_2 \cos(\theta_2) - l_3 \tag{6.3}$$

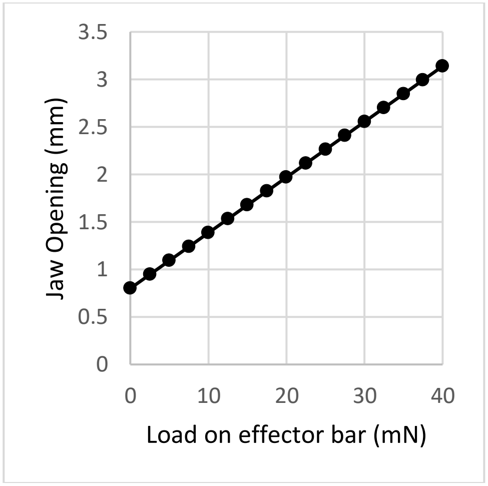
It can be observed that the vertical displacement of the effector bar is very much dependent on the angle θ_2, and a small value of θ_2 is desired to increase the opening range of the tweezer. This is because, as the distance between the effector and the electromagnet coil increases, the force will decrease rapidly as can be seen in Figure 6.2.

Since the width of the LED is about 1.25mm, the physical parameters of the tweezer need to be tuned so that a jaw opening of more than the width of the LED is required to grip the LED. Simulations of the tweezer have been performed using finite element analysis software COMSOL Multiphysics 5.3. COMSOL Multiphysics® is a simulation platform that encompasses all of the steps in the modeling workflow — from defining geometries, material properties, and the physics that describe specific phenomena to solving and postprocessing models for producing accurate and trustworthy results.

In this study, Lexan 9034 polycarbonate sheet, with a thickness of (1/32) inch has been chosen; it is a clear and strong thermoplastic material with typical tensile modulus of 238MPa, yield strength of 62MPa and Poisson ratio of 0.37. Perfect elasticity is assumed throughout the simulation with the parameters of the tweezer as follows: $l_1 = 5mm$, $l_2 = 2mm$, $l_3 = 4mm$, $\theta_2 = 60°$ and an initial jaw opening of 0.8mm. The thickness of the tweezer is (1/32 inch) which is the same as the thickness of the chosen material. Various wall thicknesses of tweezer have been attempted and the final tweezer thickness has been set to be equal to 150μm. The results of the simulation are shown in Figure 6.4. As can be seen in Figure 6.4, with 40mN force acting at the effector bar, the jaw will have an opening of more than 3mm, and the maximum von Mises stress is about ~10MPa, which is much lower than the yield strength; thus, no irreversible plastic deformation should be expected and the jaw opening vs loading force is nearly linear.

As a jaw opening of more than 1.25mm is required, a minimum loading force of about ~10mN could cause an effector bar displacement of around 100μm as can be seen in Figure 6.3 - curve D_eff (the bar is 200μm initial distance away from the electromagnet at zero force). According to our simulation result of jaw opening vs effector displacement, as shown in Figure 6.5, a 100μm displacement of the effector bar should generate a jaw opening of more than 1.25mm. Furthermore, as the typical mass of the robot is about 0.4 gram, in order to manipulate these robots upside down, a minimum force of 4mN is desired so that even one electromagnet could hold the robot from falling down. From Figure 6.2, it can be observed that a current of 300mA should be able to hold the robot; however, considering anisotropy in the assembly process of all the electromagnets and robots, the current was set to be 400mA. By setting the current to 400mA, at a distance of 350μm between the solenoid and the magnet (which is connected with tweezer effector bar), the loading force is about 15mN; this will create a jaw opening of around 1.6mm (jaw has an initial opening of 0.8mm).





(**a**) (**b**)

Figure 6.4. Simulation result of Jaw opening vs effector displacement (**a**) Displacement field of the tweezer under force load at effector (red arrows) and von Mises stress (N/m^2) distribution. (**b**) Simulation results of jaw opening vs different load forces applied at the tweezer bottom effector.

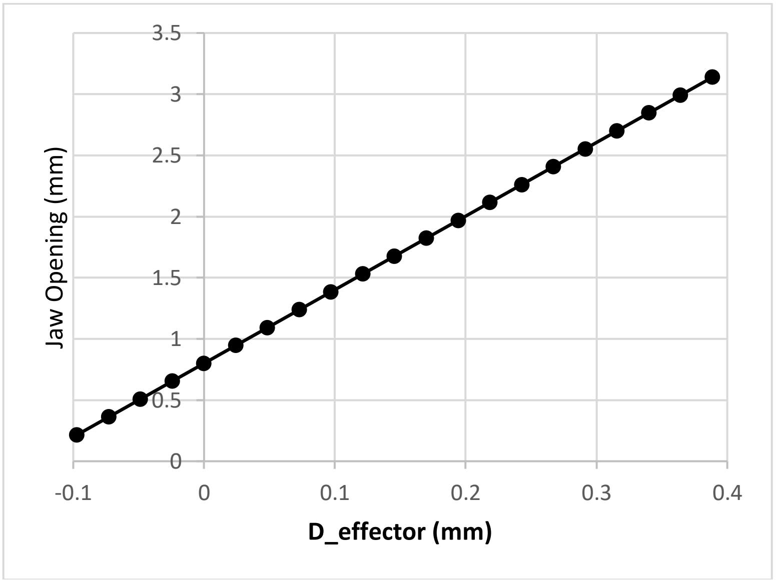
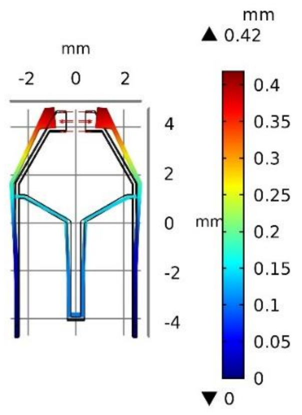
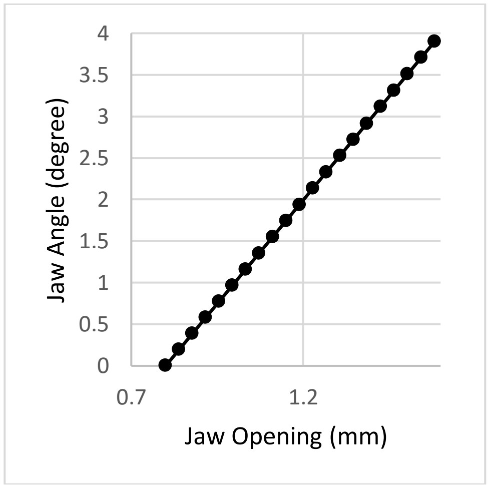


Figure 6.5. Simulation result of Jaw opening vs effector displacement. At rest, i.e. no displacement in the effector, the jaw has an initial opening of 0.8mm.

Since the initial jaw opening without external force is 0.8mm, which is less than the width of the LED, i.e., 1.25mm, a jaw displacement of 213µm or more is needed to open to hold the LED. However, the displacement at the top and the bottom of the jaw are different, as shown in Figure 6.6a. In order to hold the LED properly, it is important to cut a portion of the jaw so that, at 1.25mm total opening, the left and right jaw of the tweezer are parallel to create a uniform stress on the LED. The angle of the jaw, corresponding to y axis, has been simulated as a function of jaw displacement to determine the angle that needs to be cut, as shown in Figure 6.6b. The final angle is set to be 1.9 degrees.





(**a**) (**b**)

Figure 6.6. (**a**) Displacement field distribution of the tweezer under loading force (red arrows) at both sides of jaws. (**b**) Simulation results of jaw opening angle vs different load forces applied at both sides of tweezer jaw.

6.3 Tweezer Fabrication

The tweezers have been fabricated by cutting a 0.8mm thickness polycarbonate (PC) sheet using our home-built CNC machine with 0.016-inch flat end miniature mill, as shown in Figure 6.7. Tweezers are mounted to robot chassis and tested on the electromagnetic panel.

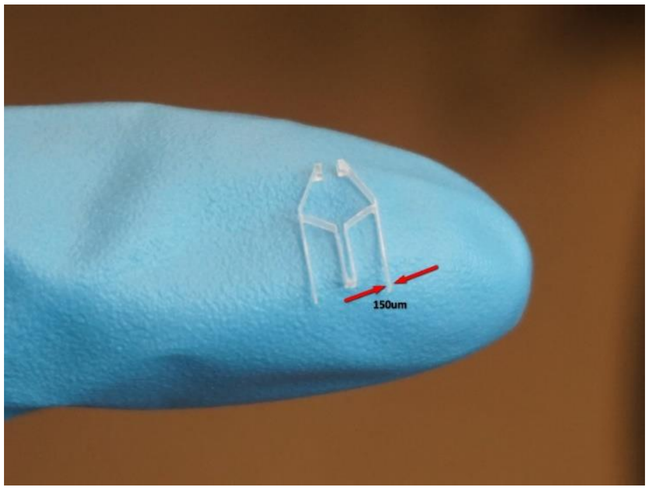


Figure 6.7. Sample tweezer fabricated using CNC machine. The thickness of the tweezer is 0.8mm, and the wall thickness is 150µm.

6.4 Tweezer Testing

Tweezers are mounted to robot chassis and tested on the electromagnetic panel. As can be seen in Figure 6.8, we have been able to close and open the tweezer by switching the direction of current (400mA) in the electromagnet.

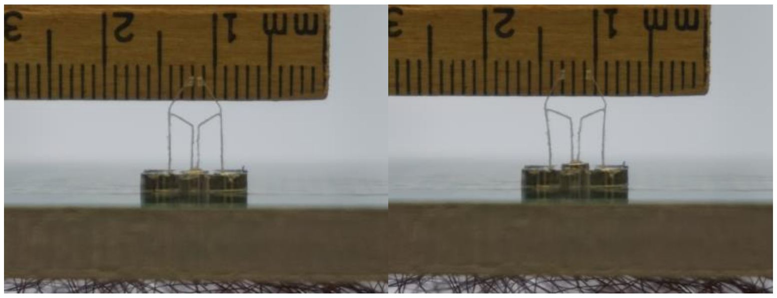


Figure 6.8 Testing the robot with tweezer on our prototype machine. Tweezer is closed on left side of the figure and open on the right side of the figure.

CHAPTER 7

RESULTS OF ASSEMBLING LEDS

A transparent box has been fabricated to hold the electromagnetic panel and in addition, to hold an LED cartridge. The cartridge is fabricated using our CNC machine from 1/4 inch thick aluminum plate, the size of the cartridge is 42mm x 15mm with round corners. The cartridge can hold up to 5x14=70 LEDs. The picture of the cartridge is shown in Figure 7.1.

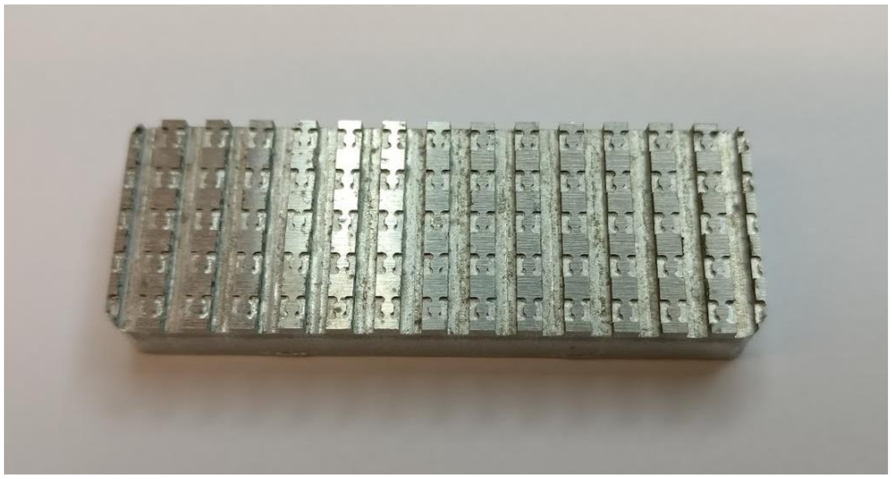


Figure 7.1 Aluminum LED cartridge fabricated using CNC machine.

7.1 Parallel Assembly Using Two Robots

In order to demonstrate the ability of parallel assembly of the prototype machine, two LEDs have been put inside the cartridge. A 2.4mm x 2.4mm blue sticker representing an assembly substrate is placed next to the cartridge. The sticker has double sided tape applied on top to simulate the soldering gel used in SMT. A linear actuator below the cartridge is also controlled by Arduino MCU to lift and drop (up and down) the cartridge, so that the LED can be inserted into the tweezer jaw when it is open, as shown in Figure 7.2. The main purpose of the box stand is to hold the electromagnetic array above it by using the four bolts in the corner of this box. Since the robots are running upside down (tweezer towards ground) between the electromagnetic array ceiling and the substrate, there need to be some precise space between the ceiling and the substrate, and the box can be used to hold the ceiling and the bolts can be used to adjust the space in between. To demonstrate the proof-of-concept, we have used two robots to simultaneously pick up the LEDs and then place them at the desired locations. The corresponding coordinates viewed via the microcontroller are from points (11,5) and (11,12) to location (3,5) and (3,12) of the electromagnetic array; so each of the robots need to move eight steps right to pick up LED and then eight steps left to drop the LED; with each step set to 300ms, the total process takes about 6s to complete. Figure 7.3 are frames taken from the demonstration assembly video. Again, as we have discussed earlier, the two robots with tweezer are independently operated, they do not necessarily do the same work at the same time.

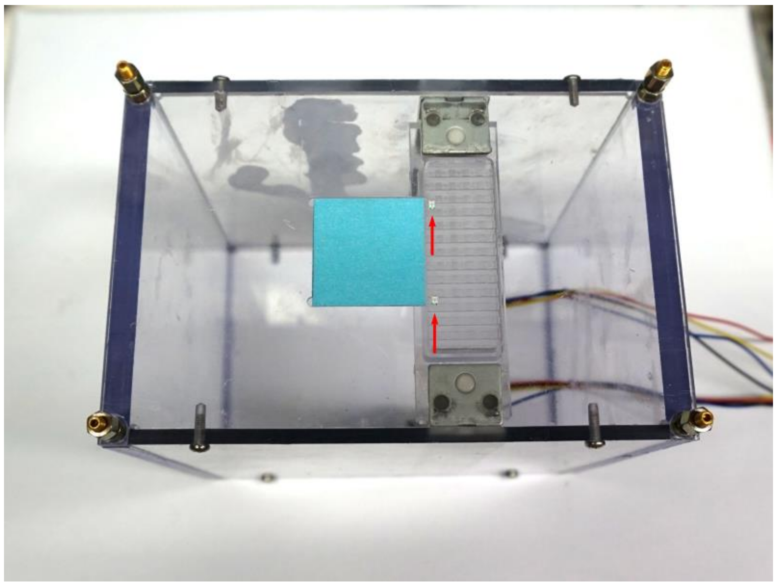


Figure 7.2 Transparent box fabricated to hold the electromagnetic panel and LED cartridge. The dimension of the box is 120mm x 80mm x 120mm (L x W x H) Linear cartridge actuator is inside the box under the LED cartridge. Two LEDs (indicated by red arrows) are put into the cartridge. The box is used to hold the electromagnetic array above it by using the four bolts in the corner, and the bolts can be used to adjust the space in between.

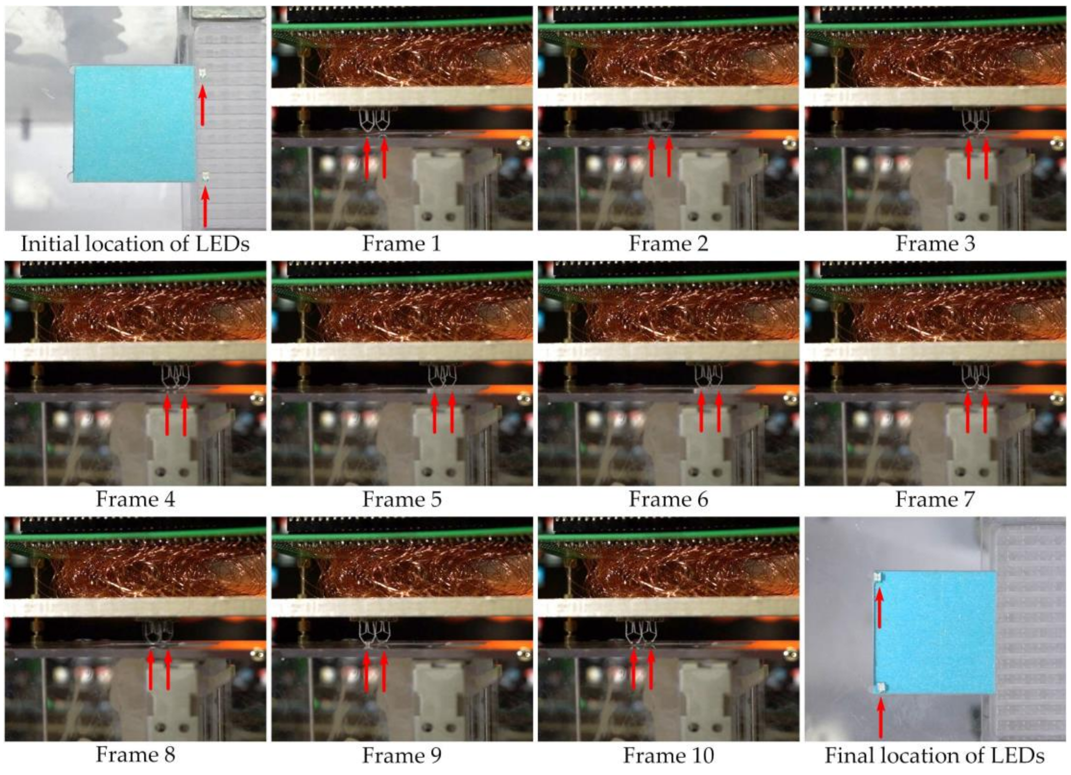


Figure 7.3 Simultaneous assembly of two LEDs by our prototype machine. The LEDs are picked-up from their initial location in the cartridge (top left) to their desired location on blue sticker (bottom right). Others are frames taken from the demonstration assembly video 2. The location of the tweezers is marked in red arrows. Referring to this figure, from left to right and from the top row to the bottom row - Initially, the LEDs are residing in their initial location of the cartridge; In Frame 1, the Robots are at rest; In Frame 2, Robots are moving right towards the initial location of the LEDs in the cartridge ; In Frame 3, the Robots have arrived at the initial location of LEDs; In Frame 4, the tweezer jaws are open; In Frame 5, the cartridge moves up and the LEDs are inserted into the opening jaws; In Frame 6, the tweezer jaws are closed to hold the LEDs; In Frame 7, the cartridge moves down and the LEDs are now held by the tweezers; In Frame 8, the Robots are moving left towards the desired final location of the LEDs; In Frame 9, the Robots have arrived at the desired location of LEDs; In Frame 10, the tweezer jaws open again to drop the LEDs ; In the end, the LEDs are placed at their desired location.

7.2 Sequential Assembly of 64 LEDs Using Single Robot

One of the potential applications of our proposed assembly technology is the capability to assemble LED screens. Here, we demonstrate sequential assembly of an 8x8 LED matrix using our prototype machine. To begin with, 64 0805 LEDs have been placed into the cartridge, as shown in Figure 7.4.

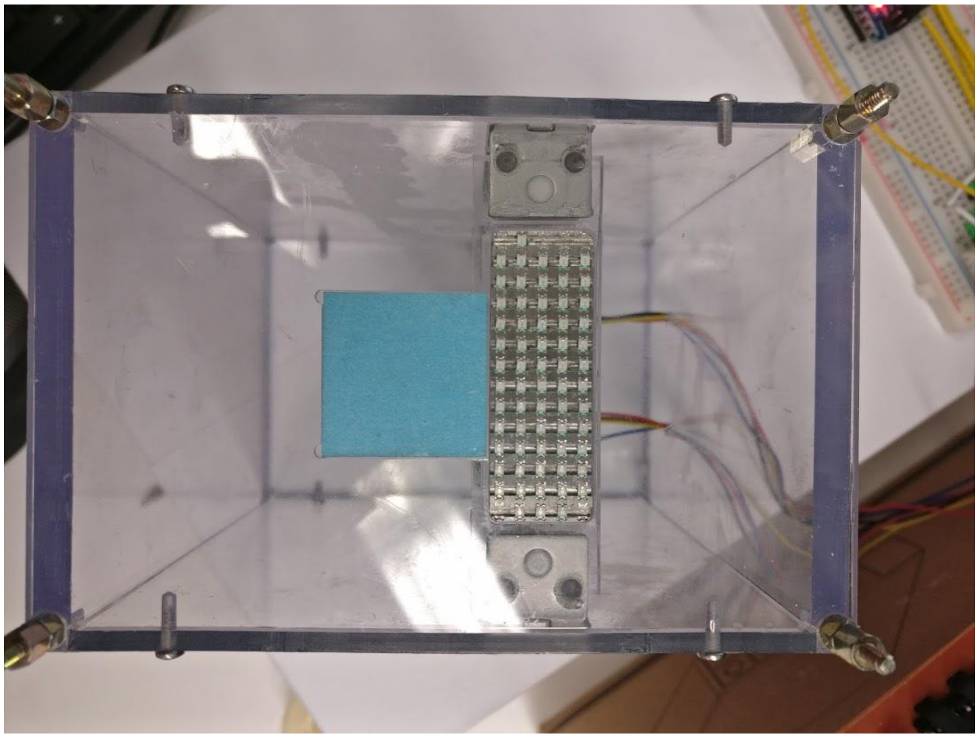


Figure 7.4 64 0805 LEDs have been placed into the cartridge.

After placing our machine in the box, the robot moves all the LEDs from the cartridge to their target location (blue sticker) on the left. Figure 7.5 is a snapshot frame from the assembly video showing the assembly process. The entire process takes about 10 minutes to complete.

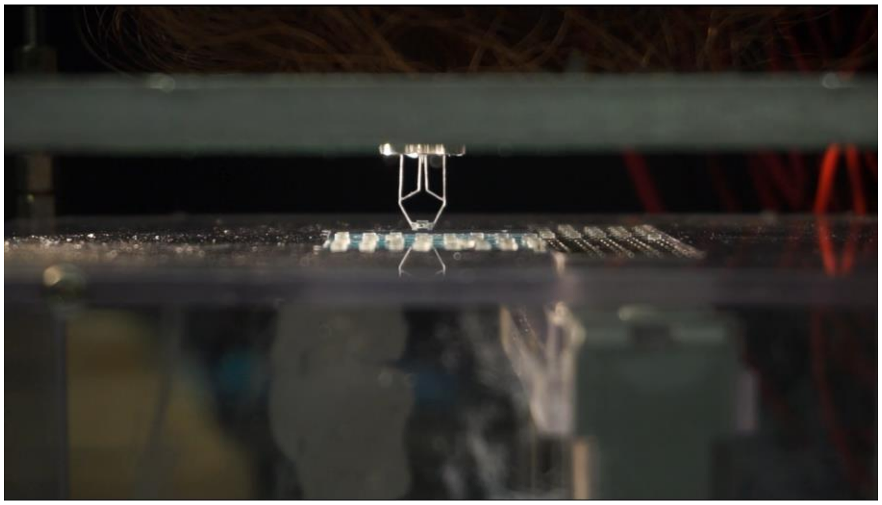


Figure 7.5 Snapshot frame from the assembly video showing the assembly process.

The final results of the assembly of 64 0805 LEDs are shown in Figure 7.6. The program to perform this task can be found in Appendix B3. As can be seen in the figure, our machine successfully assembled 64 LEDs into an 8x8 matrix; the entire process took about 10 minutes to complete, resulting in assembly speed of 6.4 LEDs per minute. The minor defects which are shown in the picture is due to the fact that our machine is handmade and lacks precision.

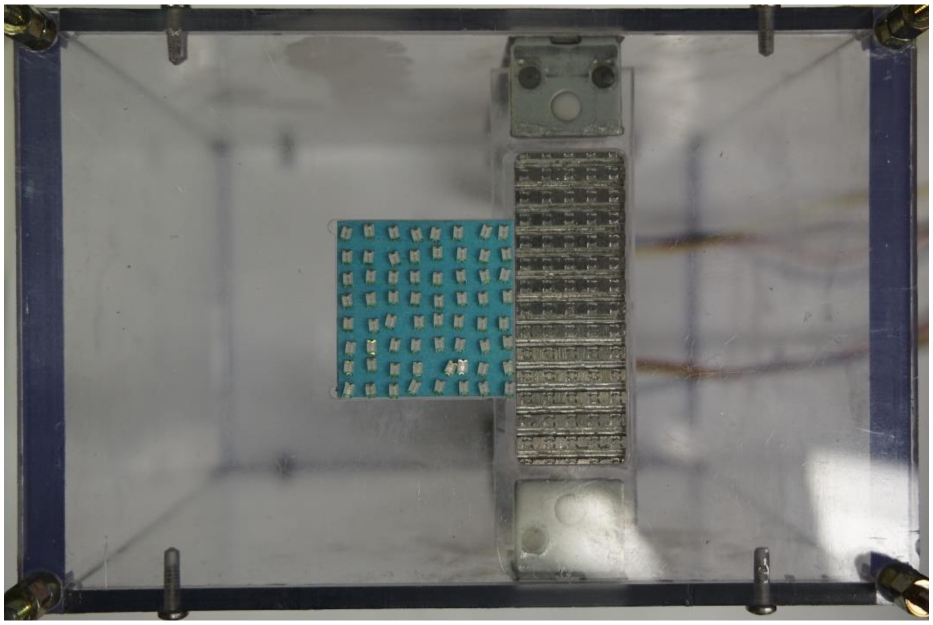


Figure 7.6 64 0805 LEDs has been assembled into an 8x8 matrix.

CHAPTER 8

DISCUSSION AND OUTLOOK

8.1 Summary of Results

We have demonstrated a parallel assembly technique based on the controlled manipulation of magnetic field assisted robots. By using a 16x16 electromagnet array, we have been able to control 9 robots, about 3906 robots/m^2; all of them can be equipped with active tweezer. We have presented parallel assembly by operating two robots simultaneously, and serial assembly of 64 LEDs. It is worth noting that our MFAA system can be easily scaled up by using larger array of electromagnets to create a swarm robotic system. This system has the potential to assemble thousands of small devices simultaneously while keeping the machine at a conventional size (~ 1m).

8.2 Discussion

We have successfully demonstrated our concept of Magnetic Field Assisted Micro Robotic Assembly; however, it is still in its very early stage, and consequently, there are many questions or limitations that need to be addressed in the future.

1. Compatibility

   The first question that needs to be addressed is the following - is this technology compatible with the current Surface Mount technology? The surface mount relies on vacuum tweezers to perform pick-and-place of standardized surface mount devices (SMD). The size of these SMDs fit out technology very well being that the LEDs used in this work are exactly one of the most commonly used SMDs. The main difficulty to make our technology compatible to current SMT is that we need to develop a vacuum tweezer which can be integrated with our robot and be able to be operated by our system.

2. Scalability

   Scalability is the least problem of concern for applications of this technology; even the electromagnetic array that was used in this research is 16 x 16; there is no foreseeable limits to prevent us to fabricate a 1600 x 1600 array.

3. Resolution

   Resolution could be the biggest challenge. This is because of the following: in our current design of the MFAA system, the resolution of the target location is completely depending on the pitch of the electromagnet array. As the pitch is 3mm, the LEDs must be dropped with a minimum distance of 3mm; in addition, it must be dropped at the location of an electromagnet. One of the solutions to overcome this problem is to change the size and pitch of the electromagnet array; however, it may also require changes to the design of robots and would be very hard to shrink the diameter of the electromagnet size down to micron level. A reasonable solution would be to develop more advanced robots and permit the robots to handle the resolution between the pitch. More specifically, integration of a micro xyz moving stage with a range of half pitch into the robot is a possible solution. While the robot moves at a step size of the array pitch, the xyz stage will move the tweezer within the range of the pitch precisely to be able to reach any point in the plane. In this case, the resolution of the system will be the resolution of the xyz stage. Specially, if the tweezer within the xyz stage is the vacuum tweezer discussed earlier, our machine will be able to perform pick-and-place assembly with any desired resolution.

4. Capability

   Although the purpose of our research is focused on assembly, the capability of the robot should not be limited to assembly. A robot, depending on the kinds of tools it is equipped with, can perform a variety of tasks. For example, if the robot, developed in this research, is equipped with a drill bit, the robot could be used as a drilling robot. Alternately, if the tweezer is replaced with a dispenser, the machine could be used as a printer.

5. Advanced robots with wireless power supply

   For the design of advanced robots, which can be integrated into the present technology, one must consider the robots to be equipped with built-in power supply, sensors and communication module. Since the system that has been demonstrated in the present study uses a large number of electromagnets, potentially, we could use these electromagnets to supply power to the robots at any location by wireless means, or even better, with wireless power supply at the desired location on demand.

6. Dream Machine

   By scaling the MFAA system to 3-D, the integration of 3-D devices can be performed at will. As was discussed at the beginning of this thesis, the "Dream Machine" is the machine that will perform any task all by itself, just like self-assembly. Using the proposed technology, the transformation from assembly in 2-D (for example, circle or square) to 3-D (for example, cylinder or cube) is feasible.

8.3 Outlook to Future

While there are several steps that can be implemented in the future to improve this technology, potentially with little improvement to the robot, we should be able to make a fully working color LED display module using our prototype machine. As we discussed in this research, we have already assembled an 8x8 matrix of mono color LEDs. However, a color LED display requires a mix of 3 color (RGB) matrices into one; thus, the total number of LEDs to be assembled increases to 192. Furthermore, each pixel of the color LED module will need 3 sub-pixels (RGB) to be put together closely, while the distance between the pixels is relatively bigger. This poses a challenge with our current design of the machine; as discussed earlier, the LEDs can only be dropped at the location of each electromagnet. Of course, this problem can be easily solved if we have advanced robot design with micro xyz moving stage.

But, since the structure of the LED sub-pixels are identical at any pixel and the sub-pixels must be very close, we could assemble the 3 sub-pixels just by adding another displaced tweezer, as shown in Figure 8.1. The tweezer in the center is the normal tweezer (we call it G-Tweezer) being used in this study; however, the tweezer on the left (we can call it R-Tweezer) is modified with displaced jaws. When assembling LEDs, the displaced jaw will drop the LED on the left to the center with a displacement so that it will not fall onto the position of G. The tweezer on the right (we call it R'-Tweezer) will be the same tweezer as G-Tweezer but rotated 180˚ along z-axis. With such simple design, we could use two different tweezer designs to assemble three different color (RGB) LEDs, which will make it possible to fabricate a color LED display module by using our current machine.

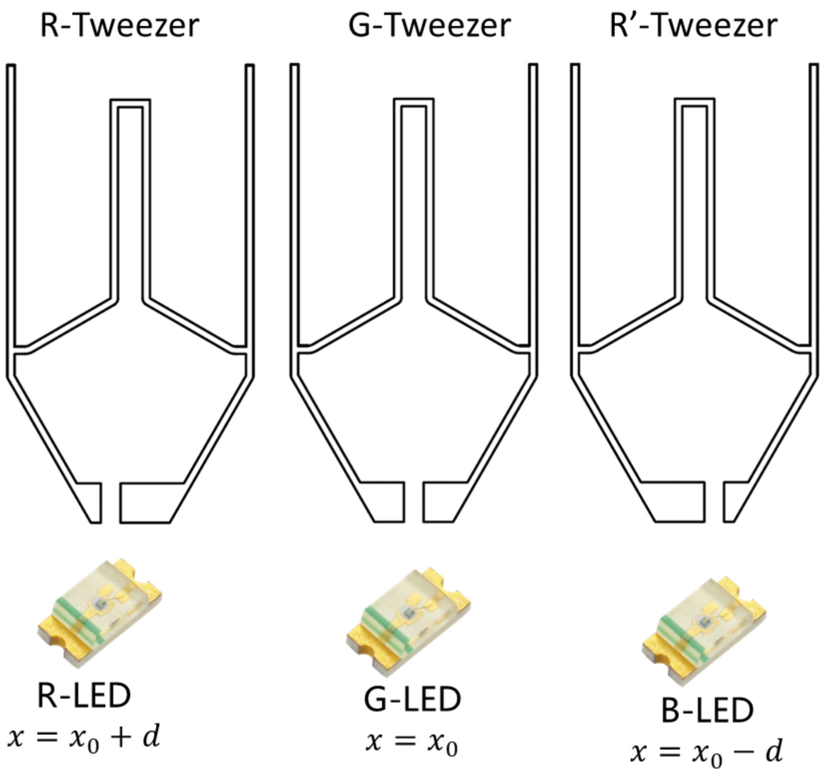


Figure 8.1 Two different tweezer designs to assemble three different color LEDs.

APPENDIX A

HOME-BUILT CNC MACHINE

Figure A.1 shows the home-built CNC machine that was used in this project.

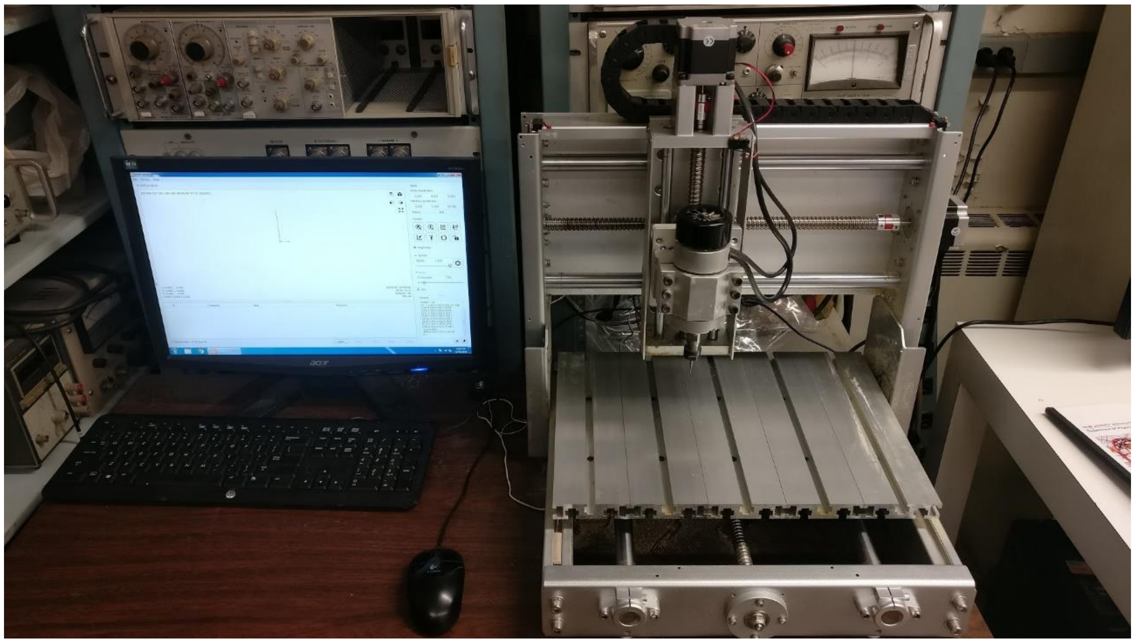


Figure A.1 Home-built CNC machine.

CNC machine tool chain used in this project:

CAD: Autodesk Fusion 360

CAM: Candle 1.1.7 [46]

CNC Controller: grbl v1.1f [47]

APPENDIX B

ARDUINO SOURCE CODE FOR MACHINE CONTROL

B.1 Source Code Used to Test the EM Array

The following was written in Arduino IDE 1.8.1. The purpose of this code is to operate a small NdFeB magnet run through every point of the EM array to perform a test. In case of any defects or error in the EM array, the magnet should stop or flip at the defect point.

```
#include <SPI.h>
#define LATCH     10
#define StepTime  300
#include <Stepper.h>
const int stepsPerRevolution = 20;  // change this to fit the number of steps per revolution for your motor
// initialize the stepper library on pins 5 through 8:
Stepper myStepper(stepsPerRevolution, 5,6,7,8);

int Robot[3][3] = { {1,0,-1},
                    {0,-1,0},
                    {-1,0,1},};

int LocationX=2;
int LocationY=3;

byte MagArray[16][4] = {{B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},};

void setup() {
  pinMode(12,INPUT_PULLUP);
  pinMode(LATCH,OUTPUT);
  SPI.begin ();

  // set the stepper speed at 600 rpm:
  myStepper.setSpeed(600);
  SetRobot(2,3);
  Refresh();
}

void loop() {
  while(digitalRead(12)==HIGH){
    Refresh();
    delay(10);
  }
  TestPanel();
}

void TestPanel() {
  int i=1;
  int j=1;
  while(j<17){
    if(i==1){
      while(i<17){
        OpenRobotTweezer(i,j);
        Refresh();
        delay(300);
        UnSetRobotTweezer(i,j);
        Refresh();
        i++;
      }
    }
    else if (i==17){

        while(i>1){
          i--;
          OpenRobotTweezer(i,j);
          Refresh();
          delay(300);
          UnSetRobotTweezer(i,j);
          Refresh();

        }
      }
  j++;
  }
}
void OpenRobotTweezer(int x, int y){
  int i;
  int j;
  int Remainder;

  j = x/4;
  i = y-1;
  Remainder = x%4;
  if(Remainder==0){
    j--;
  }
  bitClear(MagArray[i][j],FindBitN(Remainder));
  bitSet(MagArray[i][j],FindBitS(Remainder));
}

void CloseRobotTweezer(int x, int y){
  int i;
  int j;
  int Remainder;

         j = x/4;
```

```
          i = y-1;
          Remainder = x%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitS(Remainder));
          bitSet(MagArray[i][j],FindBitN(Remainder));
        }

        void UnSetRobotTweezer(int x, int y){
          int i;
          int j;
          int Remainder;

          j = x/4;
          i = y-1;
          Remainder = x%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitN(Remainder));
          bitClear(MagArray[i][j],FindBitS(Remainder));
        }

        int FindBitN(int Remainder){
          if(Remainder==0){
            return 1;
          }
          else if(Remainder==1){
            return 7;
          }
          else if(Remainder==2){
            return 5;
          }
          else if(Remainder==3){
            return 3;
          }
        }
        int FindBitS(int Remainder){
          if(Remainder==0){
            return 0;
          }
          else if(Remainder==1){
            return 6;
          }
          else if(Remainder==2){
            return 4;
          }
          else if(Remainder==3){
            return 2;
          }
        }

        void UnSetRobot(int x, int y){
          int i;
          int j;
          int Remainder;

          j = (x-1)/4;
          i = y-2;
          Remainder = (x-1)%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitN(Remainder));

          j =(x+1)/4;
          i = y-2;
          Remainder = (x+1)%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitS(Remainder));

          j = (x-1)/4;
          i = y;
          Remainder = (x-1)%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitS(Remainder));

          j = (x+1)/4;
          i = y;
          Remainder = (x+1)%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitN(Remainder));
        }

        void SetRobot(int x, int y){
          int i;
          int j;
          int Remainder;

          j = (x-1)/4;
          i = y-2;
          Remainder = (x-1)%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitN(Remainder));

          j =(x+1)/4;
          i = y-2;
          Remainder = (x+1)%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitS(Remainder));

          j = x/4;
          i = y-1;
          Remainder = x%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitN(Remainder));

          j = (x-1)/4;
          i = y;
          Remainder = (x-1)%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitS(Remainder));

          j = (x+1)/4;
          i = y;
          Remainder = (x+1)%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitN(Remainder));
        }

        void MoveTo (int x, int y) {
          while(LocationX < x){
              UnSetRobot(LocationX,LocationY);
              LocationX++;
              SetRobot(LocationX,LocationY);
              Refresh();
              delay(StepTime);
              LocationX--;
              UnSetRobotTweezer(LocationX,LocationY);
              Refresh();
              LocationX++;
              /*delay(100);*/
          }
          while(LocationX > x){

              UnSetRobot(LocationX,LocationY);
              LocationX--;
              SetRobot(LocationX,LocationY);
              Refresh();
              delay(StepTime);
              LocationX++;
              UnSetRobotTweezer(LocationX,LocationY);
              Refresh();
              LocationX--;
              /*delay(100);*/
          }
          while(LocationY < y){
              UnSetRobot(LocationX,LocationY);
              LocationY++;
              SetRobot(LocationX,LocationY);
              Refresh();
              delay(StepTime);
              LocationY--;
              UnSetRobotTweezer(LocationX,LocationY);
              Refresh();
              LocationY++;
              /*delay(100);*/
          }
         while(LocationY > y){
              UnSetRobot(LocationX,LocationY);
              LocationY--;
              SetRobot(LocationX,LocationY);
              Refresh();
              delay(StepTime);
              LocationY++;
              UnSetRobotTweezer(LocationX,LocationY);
              Refresh();
              LocationY--;
              /*delay(100);*/
          }
        }

        void Refresh() {
          SPI.beginTransaction (SPISettings (4000000, LSBFIRST, SPI_MODE0));
          digitalWrite (LATCH, LOW);
          for(int i=15; i>=0; i--)
          {
            for(int j=3; j>=0; j--)
            {
              SPI.transfer(MagArray[i][j]);
            }
          }
          digitalWrite (LATCH, HIGH);
          SPI.endTransaction();
        }

        void ClearAll(){
          for(int i=0; i<=15; i++)
          {
            for(int j=0; j<=3; j++)
            {
              MagArray[i][j]=B00000000;
            }
          }
        }
```

B.2 Source Code Used for Movement and Rotation of Nine Robots

The following was written in Arduino IDE 1.8.1. The purpose of this code is to operate nine robots for movement and rotation.

```
#include <SPI.h>
#define LATCH     10
#define StepTime  300
#include <Stepper.h>
const int stepsPerRevolution = 20;  // change this to fit the number of steps per revolution for your motor
// initialize the stepper library on pins 5 through 8:
Stepper myStepper(stepsPerRevolution, 5,6,7,8);

// int Robot[3][3] = {  {1,0,-1},
//                        {0,-1,0},
//                        {-1,0,1},   };

int Robots[2][2] = {  {2,2},
                      {8,2},
                              };

int LocationX=2;
int LocationY=3;

byte MagArray[16][4] = {{B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},};


void setup() {
  pinMode(12,INPUT_PULLUP);
  pinMode(LATCH,OUTPUT);
  SPI.begin ();
   // set the speed at 600 rpm:
  myStepper.setSpeed(600);

  SetRobot(2,2);
  SetRobot(8,2);
  SetRobot(14,2);
  SetRobot(2,8);
  SetRobot(8,8);
  SetRobot(14,8);
  SetRobot(2,14);
  SetRobot(8,14);
  SetRobot(14,14);
  //LocationX=2;
 // LocationY=2;

Refresh();
}

void loop() {
  while(digitalRead(12)==HIGH){
    Refresh();
    delay(10);
  }

  while(digitalRead(12)==LOW){
    for(int i=0; i<2; i++){
    UnSetRobot(2,2);
    SetAngle45(2,2);
    UnSetRobot(2,8);
    SetAngle45(2,8);
    UnSetRobot(2,14);
    SetAngle45(2,14);
    UnSetRobot(8,2);
    SetAngle45(8,2);
    UnSetRobot(8,8);
    SetAngle45(8,8);
    UnSetRobot(8,14);
    SetAngle45(8,14);
    UnSetRobot(14,2);
    SetAngle45(14,2);
    UnSetRobot(14,8);
    SetAngle45(14,8);
    UnSetRobot(14,14);
    SetAngle45(14,14);
    Refresh();
    delay(200);

    UnSetAngle45(2,2);
    SetAngle90(2,2);
    UnSetAngle45(2,8);
    SetAngle90(2,8);
    UnSetAngle45(2,14);
    SetAngle90(2,14);
    UnSetAngle45(8,2);
    SetAngle90(8,2);
    UnSetAngle45(8,8);
    SetAngle90(8,8);
    UnSetAngle45(8,14);
            SetAngle90(8,14);
            UnSetAngle45(14,2);
            SetAngle90(14,2);
            UnSetAngle45(14,8);
            SetAngle90(14,8);
            UnSetAngle45(14,14);
            SetAngle90(14,14);
            Refresh();
            delay(200);

            UnSetAngle90(2,2);
            SetAngle135(2,2);
            UnSetAngle90(2,8);
            SetAngle135(2,8);
            UnSetAngle90(2,14);
            SetAngle135(2,14);
            UnSetAngle90(8,2);
            SetAngle135(8,2);
            UnSetAngle90(8,8);
            SetAngle135(8,8);
            UnSetAngle90(8,14);
            SetAngle135(8,14);
            UnSetAngle90(14,2);
            SetAngle135(14,2);
            UnSetAngle90(14,8);
            SetAngle135(14,8);
            UnSetAngle90(14,14);
            SetAngle135(14,14);
            Refresh();
            delay(200);

            UnSetAngle135(2,2);
            SetAngle180(2,2);
            UnSetAngle135(2,8);
            SetAngle180(2,8);
            UnSetAngle135(2,14);
            SetAngle180(2,14);
            UnSetAngle135(8,2);
            SetAngle180(8,2);
            UnSetAngle135(8,8);
            SetAngle180(8,8);
            UnSetAngle135(8,14);
            SetAngle180(8,14);
            UnSetAngle135(14,2);
            SetAngle180(14,2);
            UnSetAngle135(14,8);
            SetAngle180(14,8);
            UnSetAngle135(14,14);
            SetAngle180(14,14);
            Refresh();
            delay(200);
            }

            for(int i=0; i<2 ;i++){
            UnSetRobot(2,2);
            SetRobot(3,2);
            UnSetRobot(2,8);
            SetRobot(3,8);
            UnSetRobot(2,14);
            SetRobot(3,14);
            UnSetRobot(8,2);
            SetRobot(9,2);
            UnSetRobot(8,8);
            SetRobot(9,8);
            UnSetRobot(8,14);
            SetRobot(9,14);
            UnSetRobot(14,2);
            SetRobot(15,2);
            UnSetRobot(14,8);
            SetRobot(15,8);
            UnSetRobot(14,14);
            SetRobot(15,14);
            Refresh();
            delay(200);
```

```
          UnSetRobotTweezer(2,2);
          UnSetRobotTweezer(2,8);
          UnSetRobotTweezer(2,14);
          UnSetRobotTweezer(8,2);
          UnSetRobotTweezer(8,8);
          UnSetRobotTweezer(8,14);
          UnSetRobotTweezer(14,2);
          UnSetRobotTweezer(14,8);
          UnSetRobotTweezer(14,14);
          Refresh();

          UnSetRobot(3,2);
          SetRobot(2,2);
          UnSetRobot(3,8);
          SetRobot(2,8);
          UnSetRobot(3,14);
          SetRobot(2,14);
          UnSetRobot(9,2);
          SetRobot(8,2);
          UnSetRobot(9,8);
          SetRobot(8,8);
          UnSetRobot(9,14);
          SetRobot(8,14);
          UnSetRobot(15,2);
          SetRobot(14,2);
          UnSetRobot(15,8);
          SetRobot(14,8);
          UnSetRobot(15,14);
          SetRobot(14,14);
          Refresh();
          delay(200);
          UnSetRobotTweezer(3,2);
          UnSetRobotTweezer(3,8);
          UnSetRobotTweezer(3,14);
          UnSetRobotTweezer(9,2);
          UnSetRobotTweezer(9,8);
          UnSetRobotTweezer(9,14);
          UnSetRobotTweezer(15,2);
          UnSetRobotTweezer(15,8);
          UnSetRobotTweezer(15,14);
          Refresh();
          }
        }

        while(digitalRead(12)==HIGH){
          delay(10);
        }

       }

      void OpenRobotTweezer(int x, int y){
        int i;
        int j;
        int Remainder;

        j = x/4;
        i = y-1;
        Remainder = x%4;
        if(Remainder==0){
          j--;
        }
```

```
          bitClear(MagArray[i][j],FindBitN(Remainder));
          bitSet(MagArray[i][j],FindBitS(Remainder));
        }

        void CloseRobotTweezer(int x, int y){
          int i;
          int j;
          int Remainder;

          j = x/4;
          i = y-1;
          Remainder = x%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitS(Remainder));
          bitSet(MagArray[i][j],FindBitN(Remainder));
        }

        void UnSetRobotTweezer(int x, int y){
          int i;
          int j;
          int Remainder;

          j = x/4;
          i = y-1;
          Remainder = x%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitN(Remainder));
          bitClear(MagArray[i][j],FindBitS(Remainder));
        }

        int FindBitN(int Remainder){
          if(Remainder==0){
            return 1;
          }
          else if(Remainder==1){
            return 7;
          }
          else if(Remainder==2){
            return 5;
          }
          else if(Remainder==3){
            return 3;
          }
        }
        int FindBitS(int Remainder){
          if(Remainder==0){
            return 0;
          }
          else if(Remainder==1){
            return 6;
          }
          else if(Remainder==2){
            return 4;
          }
          else if(Remainder==3){
            return 2;
          }
        }

        void UnSetRobot(int x, int y){
          int i;
          int j;
          int Remainder;

          j = (x-1)/4;
          i = y-2;
          Remainder = (x-1)%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitN(Remainder));

          j =(x+1)/4;
          i = y-2;
          Remainder = (x+1)%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitS(Remainder));

          /*j = x/4;
          i = y-1;
          Remainder = x%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitN(Remainder));*/

          j = (x-1)/4;
          i = y;
          Remainder = (x-1)%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitS(Remainder));

          j = (x+1)/4;
          i = y;
          Remainder = (x+1)%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitN(Remainder));

        }

        void SetRobot(int x, int y){
          int i;
          int j;
          int Remainder;

          j = (x-1)/4;
          i = y-2;
          Remainder = (x-1)%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitN(Remainder));
```

```

    j =(x+1)/4;
    i = y-2;
    Remainder = (x+1)%4;
    if(Remainder==0){
      j--;
    }
    bitSet(MagArray[i][j],FindBitS(Remainder));

    j = x/4;
    i = y-1;
    Remainder = x%4;
    if(Remainder==0){
      j--;
    }
    bitSet(MagArray[i][j],FindBitN(Remainder));

    j = (x-1)/4;
    i = y;
    Remainder = (x-1)%4;
    if(Remainder==0){
      j--;
    }
    bitSet(MagArray[i][j],FindBitS(Remainder));

    j = (x+1)/4;
    i = y;
    Remainder = (x+1)%4;
    if(Remainder==0){
      j--;
    }
    bitSet(MagArray[i][j],FindBitN(Remainder));
  }

  void UnSetAngle180 (int x, int y){
    int i;
    int j;
    int Remainder;

    j = (x-1)/4;
    i = y-2;
    Remainder = (x-1)%4;
    if(Remainder==0){
      j--;
    }
    bitClear(MagArray[i][j],FindBitN(Remainder));

    j =(x+1)/4;
    i = y-2;
    Remainder = (x+1)%4;
    if(Remainder==0){
      j--;
    }
    bitClear(MagArray[i][j],FindBitS(Remainder));

    j = (x-1)/4;
    i = y;
    Remainder = (x-1)%4;
    if(Remainder==0){
      j--;
    }
```

```
          bitClear(MagArray[i][j],FindBitS(Remainder));

          j = (x+1)/4;
          i = y;
          Remainder = (x+1)%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitN(Remainder));
        }

        void SetAngle180 (int x, int y){
          int i;
          int j;
          int Remainder;

          j = (x-1)/4;
          i = y-2;
          Remainder = (x-1)%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitN(Remainder));

          j =(x+1)/4;
          i = y-2;
          Remainder = (x+1)%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitS(Remainder));

          j = (x-1)/4;
          i = y;
          Remainder = (x-1)%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitS(Remainder));

          j = (x+1)/4;
          i = y;
          Remainder = (x+1)%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitN(Remainder));
        }

        void UnSetAngle135 (int x, int y){
          int i;
          int j;
          int Remainder;

          j = x/4;
          i = y-2;
          Remainder = x%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitS(Remainder));

          j = (x-1)/4;
          i = y-1;
          Remainder = (x-1)%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitN(Remainder));

          j = (x+1)/4;
          i = y-1;
          Remainder = (x+1)%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitN(Remainder));

          j = x/4;
          i = y;
          Remainder = x%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitS(Remainder));
        }

        void SetAngle135 (int x, int y){
          int i;
          int j;
          int Remainder;

          j = x/4;
          i = y-2;
          Remainder = x%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitS(Remainder));

          j = (x-1)/4;
          i = y-1;
          Remainder = (x-1)%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitN(Remainder));

          j = (x+1)/4;
          i = y-1;
          Remainder = (x+1)%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitN(Remainder));

          j = x/4;
          i = y;
          Remainder = x%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitS(Remainder));
        }

        void UnSetAngle90 (int x, int y){
          int i;
          int j;
          int Remainder;

          j = (x-1)/4;
          i = y-2;
          Remainder = (x-1)%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitS(Remainder));

          j =(x+1)/4;
          i = y-2;
          Remainder = (x+1)%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitN(Remainder));

          j = (x-1)/4;
          i = y;
          Remainder = (x-1)%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitN(Remainder));

          j = (x+1)/4;
          i = y;
          Remainder = (x+1)%4;
          if(Remainder==0){
            j--;
          }
          bitClear(MagArray[i][j],FindBitS(Remainder));
        }
        void SetAngle90 (int x, int y){
          int i;
          int j;
          int Remainder;

          j = (x-1)/4;
          i = y-2;
          Remainder = (x-1)%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitS(Remainder));

          j =(x+1)/4;
          i = y-2;
          Remainder = (x+1)%4;
          if(Remainder==0){
            j--;
          }
          bitSet(MagArray[i][j],FindBitN(Remainder));

```

```
        j = (x-1)/4;
        i = y;
        Remainder = (x-1)%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitN(Remainder));

        j = (x+1)/4;
        i = y;
        Remainder = (x+1)%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitS(Remainder));

      }

      void UnSetAngle45 (int x, int y){
        int i;
        int j;
        int Remainder;

        j = x/4;
        i = y-2;
        Remainder = x%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitN(Remainder));

        j = (x-1)/4;
        i = y-1;
        Remainder = (x-1)%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitS(Remainder));

        j = (x+1)/4;
        i = y-1;
        Remainder = (x+1)%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitS(Remainder));

        j = x/4;
        i = y;
        Remainder = x%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitN(Remainder));
      }

      void SetAngle45 (int x, int y){
        int i;
        int j;
        int Remainder;

        j = x/4;
        i = y-2;
        Remainder = x%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitN(Remainder));

        j = (x-1)/4;
        i = y-1;
        Remainder = (x-1)%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitS(Remainder));

        j = (x+1)/4;
        i = y-1;
        Remainder = (x+1)%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitS(Remainder));

        j = x/4;
        i = y;
        Remainder = x%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitN(Remainder));
      }

      void Rotation(int x, int y, int Angle){


      }
      void ArrayMoveTo (int Destination[2][2]) {


      }

      void MoveTo (int x, int y) {
        while(LocationX < x){
            UnSetRobot(LocationX,LocationY);
            LocationX++;
            SetRobot(LocationX,LocationY);
            Refresh();
            delay(StepTime);
            LocationX--;
            UnSetRobotTweezer(LocationX,LocationY);
            Refresh();
            LocationX++;
            /*delay(100);*/
        }
        while(LocationX > x){

            UnSetRobot(LocationX,LocationY);
            LocationX--;
            SetRobot(LocationX,LocationY);
            Refresh();
            delay(StepTime);
            LocationX++;
            UnSetRobotTweezer(LocationX,LocationY);
            Refresh();
            LocationX--;
            /*delay(100);*/
        }
        while(LocationY < y){
            UnSetRobot(LocationX,LocationY);
            LocationY++;
            SetRobot(LocationX,LocationY);
            Refresh();
            delay(StepTime);
            LocationY--;
            UnSetRobotTweezer(LocationX,LocationY);
            Refresh();
            LocationY++;
            /*delay(100);*/
        }
       while(LocationY > y){
            UnSetRobot(LocationX,LocationY);
            LocationY--;
            SetRobot(LocationX,LocationY);
            Refresh();
            delay(StepTime);
            LocationY++;
            UnSetRobotTweezer(LocationX,LocationY);
            Refresh();
            LocationY--;
            /*delay(100);*/
        }
      }

      void Refresh() {
        SPI.beginTransaction (SPISettings (4000000, LSBFIRST, SPI_MODE0));
        digitalWrite (LATCH, LOW);
        for(int i=15; i>=0; i--)
        {
          for(int j=3; j>=0; j--)
          {
            SPI.transfer(MagArray[i][j]);
          }
        }
        digitalWrite (LATCH, HIGH);
        SPI.endTransaction();
      }

      void ClearAll(){
        for(int i=0; i<=15; i++)
        {
          for(int j=0; j<=3; j++)
          {
            MagArray[i][j]=B00000000;
          }
        }
      }
```

B.3 Source Code Used for Parallel Assembly

The following was written in Arduino IDE 1.8.1. The purpose of this code is to operate two robots simultaneously pick-and-place two 0805 LEDs into their designated location.

```
#include <SPI.h>
#define LATCH     10
#define StepTime  300
#include <Stepper.h>
const int stepsPerRevolution = 20;  // change this to fit the number of steps pe
r revolution for your motor
// initialize the stepper library on pins 5 through 8:
Stepper myStepper(stepsPerRevolution, 5,6,7,8);

int Robot[3][3] = { {1,0,-1},
                    {0,-1,0},
                    {-1,0,1},};

int LocationX=2;
int LocationY=5;

byte MagArray[16][4] = {{B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},};


void setup() {
  pinMode(12,INPUT_PULLUP);
  pinMode(LATCH,OUTPUT);
  SPI.begin ();

   // set the speed at 600 rpm:
  myStepper.setSpeed(900);

  SetRobot(3,5);
  SetRobot(3,12);
  Refresh();
}

void loop() {
  while(digitalRead(12)==HIGH){
    Refresh();
    delay(10);
  }

  while(digitalRead(12)==LOW){
   MoveTo(11,5);
    //UnSetRobot(2,2);
    //OpenRobotTweezer(2,2);
    //SetRobot(3,2);
    Refresh();
    delay(500);
  }

  while(digitalRead(12)==HIGH){
    //Refresh();
    delay(10);
  }

  while(digitalRead(12)==LOW){
    OpenRobotTweezer(11,5);
    OpenRobotTweezer(11,12);
    Refresh();
    delay(500);
  }

  while(digitalRead(12)==HIGH){
    //Refresh();
    delay(10);
  }

 while(digitalRead(12)==LOW){
    myStepper.step(64);
    Refresh();
    //delay(500);
  }

 while(digitalRead(12)==HIGH){
    //Refresh();
    delay(10);
  }

  while(digitalRead(12)==LOW){
    CloseRobotTweezer(11,5);
    CloseRobotTweezer(11,12);
    Refresh();
    delay(500);
  }

  while(digitalRead(12)==HIGH){
    //Refresh();
    delay(10);
        }

        while(digitalRead(12)==LOW){
          myStepper.step(-64);
          Refresh();
          //delay(500);
        }

        while(digitalRead(12)==HIGH){
          //Refresh();
          delay(10);
        }

        while(digitalRead(12)==LOW){
          MoveTo(3,5);
          Refresh();
          delay(500);
        }

        while(digitalRead(12)==HIGH){
          //Refresh();
          delay(10);
        }

        while(digitalRead(12)==LOW){
          UnSetRobotTweezer(3,5);
          UnSetRobotTweezer(3,12);
          Refresh();
          delay(200);
          OpenRobotTweezer(3,5);
          OpenRobotTweezer(3,12);
          Refresh();
          delay(400);
          CloseRobotTweezer(3,5);
          CloseRobotTweezer(3,12);
          Refresh();
        }

      }

      void OpenRobotTweezer(int x, int y){
        int i;
        int j;
        int Remainder;

        j = x/4;
        i = y-1;
        Remainder = x%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitN(Remainder));
        bitSet(MagArray[i][j],FindBitS(Remainder));
      }

      void CloseRobotTweezer(int x, int y){
        int i;
        int j;
        int Remainder;

        j = x/4;
        i = y-1;
        Remainder = x%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitS(Remainder));
        bitSet(MagArray[i][j],FindBitN(Remainder));
      }

      void UnSetRobotTweezer(int x, int y){
        int i;
        int j;
        int Remainder;
```

```

        j = x/4;
        i = y-1;
        Remainder = x%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitN(Remainder));
        bitClear(MagArray[i][j],FindBitS(Remainder));
      }

      int FindBitN(int Remainder){
        if(Remainder==0){
          return 1;
        }
        else if(Remainder==1){
          return 7;
        }
        else if(Remainder==2){
          return 5;
        }
        else if(Remainder==3){
          return 3;
        }
      }
      int FindBitS(int Remainder){
        if(Remainder==0){
          return 0;
        }
        else if(Remainder==1){
          return 6;
        }
        else if(Remainder==2){
          return 4;
        }
        else if(Remainder==3){
          return 2;
        }
      }

      void UnSetRobot(int x, int y){
        int i;
        int j;
        int Remainder;

        j = (x-1)/4;
        i = y-2;
        Remainder = (x-1)%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitN(Remainder));

        j =(x+1)/4;
        i = y-2;
        Remainder = (x+1)%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitS(Remainder));

        /*j = x/4;
        i = y-1;
        Remainder = x%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitN(Remainder));*/

        j = (x-1)/4;
        i = y;
        Remainder = (x-1)%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitS(Remainder));

        j = (x+1)/4;
        i = y;
        Remainder = (x+1)%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitN(Remainder));
      }

      void SetRobot(int x, int y){
        int i;
        int j;
        int Remainder;

        j = (x-1)/4;
        i = y-2;
        Remainder = (x-1)%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitN(Remainder));

        j =(x+1)/4;
        i = y-2;
        Remainder = (x+1)%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitS(Remainder));

        j = x/4;
        i = y-1;
        Remainder = x%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitN(Remainder));

        j = (x-1)/4;
        i = y;
        Remainder = (x-1)%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitS(Remainder));
```

```

        j = (x+1)/4;
        i = y;
        Remainder = (x+1)%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitN(Remainder));
      }

      void MoveTo (int x, int y) {
        while(LocationX < x){
            UnSetRobot(LocationX,LocationY);
            UnSetRobot(LocationX,LocationY+7);
            LocationX++;
            SetRobot(LocationX,LocationY);
            SetRobot(LocationX,LocationY+7);
            Refresh();
            delay(StepTime);
            LocationX--;
            UnSetRobotTweezer(LocationX,LocationY);
            UnSetRobotTweezer(LocationX,LocationY+7);
            Refresh();
            LocationX++;
            /*delay(100);*/
        }
        while(LocationX > x){
            UnSetRobot(LocationX,LocationY);
            UnSetRobot(LocationX,LocationY+7);
            LocationX--;
            SetRobot(LocationX,LocationY);
            SetRobot(LocationX,LocationY+7);
            Refresh();
            delay(StepTime);
            LocationX++;
            UnSetRobotTweezer(LocationX,LocationY);
            UnSetRobotTweezer(LocationX,LocationY+7);
            Refresh();
            LocationX--;
            /*delay(100);*/
        }
        while(LocationY < y){
            UnSetRobot(LocationX,LocationY);
            LocationY++;
            SetRobot(LocationX,LocationY);
            Refresh();
            delay(StepTime);
            LocationY--;
            UnSetRobotTweezer(LocationX,LocationY);
            Refresh();
            LocationY++;
            /*delay(100);*/
        }
       while(LocationY > y){
            UnSetRobot(LocationX,LocationY);
            LocationY--;
            SetRobot(LocationX,LocationY);
            Refresh();
            delay(StepTime);
            LocationY++;
            UnSetRobotTweezer(LocationX,LocationY);
```

```
              Refresh();
              LocationY--;
              /*delay(100);*/
          }
        }

        void Refresh() {
          SPI.beginTransaction (SPISettings (4000000, LSBFIRST, SPI_MODE0));
          digitalWrite (LATCH, LOW);
          for(int i=15; i>=0; i--)
          {
            for(int j=3; j>=0; j--)
            {
              SPI.transfer(MagArray[i][j]);
            }
          }
          digitalWrite (LATCH, HIGH);
          SPI.endTransaction();
        }

        void ClearAll(){
          for(int i=0; i<=15; i++)
          {
            for(int j=0; j<=3; j++)
            {
              MagArray[i][j]=B00000000;
            }
          }
        }
```

B.4 Source Code for Sequential Assembly of LED Matrix

The following was written in Arduino IDE 1.8.1. The purpose of this code is to operate a single robot for sequentially assemble 64 LEDs into an 8x8 matrix.

```
#include <SPI.h>
#define LATCH     5
#define button    11

#define StepTime  250
#include <Stepper.h>
const int stepsPerRevolution = 20;  // change this to fit the number of steps per revolution for your motor
// initialize the stepper library on pins 5 through 8:
Stepper myStepper(stepsPerRevolution,6,7,8,9);
int LocationX=0;
int LocationY=0;

byte MagArray[16][4] = {{B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},
                        {B00000000,B00000000,B00000000,B00000000},};

byte Cartridge[64][2] = {        {12,2},{13,2},{14,2},
                          {11,3},{12,3},{13,3},{14,3},{15,3},
                          {11,4},{12,4},{13,4},{14,4},{15,4},
                          {11,5},{12,5},{13,5},{14,5},{15,5},
                          {11,6},{12,6},{13,6},{14,6},{15,6},
                          {11,7},{12,7},{13,7},{14,7},{15,7},
                          {11,8},{12,8},{13,8},{14,8},{15,8},
                          {11,9},{12,9},{13,9},{14,9},{15,9},
                          {11,10},{12,10},{13,10},{14,10},{15,10},
                          {11,11},{12,11},{13,11},{14,11},{15,11},
                          {11,12},{12,12},{13,12},{14,12},{15,12},
                          {11,13},{12,13},{13,13},{14,13},{15,13},
                          {11,14},{12,14},{13,14},{14,14},{15,14},
                                 {12,15},
};

byte LED[64][2] = {     {3,5}, {4,5}, {5,5}, {6,5}, {7,5}, {8,5}, {9,5}, {10,5},

                        {3,6}, {4,6}, {5,6}, {6,6}, {7,6}, {8,6}, {9,6}, {10,6},

                        {3,7}, {4,7}, {5,7}, {6,7}, {7,7}, {8,7}, {9,7}, {10,7},
                        {3,8}, {4,8}, {5,8}, {6,8}, {7,8}, {8,8}, {9,8}, {10,8},
                        {3,9}, {4,9}, {5,9}, {6,9}, {7,9}, {8,9}, {9,9}, {10,9},
                        {3,10},{4,10},{5,10},{6,10},{7,10},{8,10},{9,10},{10,10},
                        {3,11},{4,11},{5,11},{6,11},{7,11},{8,11},{9,11},{10,11},
                        {3,12},{4,12},{5,12},{6,12},{7,12},{8,12},{9,12},{10,12},
};


void setup() {
  pinMode(10,INPUT_PULLUP);
  pinMode(button, INPUT_PULLUP);
  pinMode(LATCH,OUTPUT);
  attachInterrupt(digitalPinToInterrupt(10), pause, FALLING);
  SPI.begin ();
   // set the speed at 900 rpm:
  myStepper.setSpeed(900);
  Refresh();
}

void loop() {
  SetRobot(2,2);
  Refresh();
  while(digitalRead(button)==HIGH){
      Refresh();
      delay(10);
  }
  MoveTo(12,2);
  wait();
  MoveTo(12,14);
  wait();
  MoveTo(2,2);
  wait();
  for(int i=0; i<64; i++){
    PickAndPlace(Cartridge[i][0],Cartridge[i][1],LED[i][0],LED[i][1]);
  }
  MoveTo(2,2);
}

void PickAndPlace(int x0, int y0, int x, int y){
  MoveTo(x0,y0);
  delay(400);
  OpenRobotTweezer(x0,y0);
  Refresh();
  delay(300);
  myStepper.step(64);
  delay(300);
  CloseRobotTweezer(x0,y0);
  Refresh();
  delay(300);
  myStepper.step(-64);
  delay(300);
         MoveTo(x,y);
         delay(400);

         UnSetRobotTweezer(x,y);
         Refresh();
```

```
        delay(300);
        OpenRobotTweezer(x,y);
        Refresh();
        delay(400);
        CloseRobotTweezer(x,y);
        Refresh();
        delay(400);
      }

      void pause(){
        while(digitalRead(button)==HIGH){
        }
      }

      void wait(){
          while(digitalRead(button)==HIGH){
          //Refresh();
          delay(10);
         }
      }

      void OpenRobotTweezer(int x, int y){
        int i;
        int j;
        int Remainder;

        j = x/4;
        i = y-1;
        Remainder = x%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitN(Remainder));
        bitSet(MagArray[i][j],FindBitS(Remainder));
      }

      void CloseRobotTweezer(int x, int y){
        int i;
        int j;
        int Remainder;

        j = x/4;
        i = y-1;
        Remainder = x%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitS(Remainder));
        bitSet(MagArray[i][j],FindBitN(Remainder));
      }

      void UnSetRobotTweezer(int x, int y){
        int i;
        int j;
        int Remainder;

        j = x/4;
        i = y-1;
        Remainder = x%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitN(Remainder));
        bitClear(MagArray[i][j],FindBitS(Remainder));
      }


      int FindBitN(int Remainder){
        if(Remainder==0){
          return 1;
        }
        else if(Remainder==1){
          return 7;
        }
        else if(Remainder==2){
          return 5;
        }
        else if(Remainder==3){
          return 3;
        }
      }
      int FindBitS(int Remainder){
        if(Remainder==0){
          return 0;
        }
        else if(Remainder==1){
          return 6;
        }
        else if(Remainder==2){
          return 4;
        }
        else if(Remainder==3){
          return 2;
        }
      }

      void UnSetRobot(int x, int y){
        int i;
        int j;
        int Remainder;

        j = (x-1)/4;
        i = y-2;
        Remainder = (x-1)%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitN(Remainder));

        j =(x+1)/4;
        i = y-2;
        Remainder = (x+1)%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitS(Remainder));

        /*j = x/4;
        i = y-1;
        Remainder = x%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitN(Remainder));*/

        j = (x-1)/4;
        i = y;
        Remainder = (x-1)%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitS(Remainder));

        j = (x+1)/4;
        i = y;
        Remainder = (x+1)%4;
        if(Remainder==0){
          j--;
        }
        bitClear(MagArray[i][j],FindBitN(Remainder));
      }

      void SetRobot(int x, int y){
        LocationX=x;
        LocationY=y;
        int i;
        int j;
        int Remainder;

        j = (x-1)/4;
        i = y-2;
        Remainder = (x-1)%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitN(Remainder));

        j =(x+1)/4;
        i = y-2;
        Remainder = (x+1)%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitS(Remainder));

        j = x/4;
        i = y-1;
        Remainder = x%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitN(Remainder));

        j = (x-1)/4;
        i = y;
        Remainder = (x-1)%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitS(Remainder));

        j = (x+1)/4;
        i = y;
        Remainder = (x+1)%4;
        if(Remainder==0){
          j--;
        }
        bitSet(MagArray[i][j],FindBitN(Remainder));
      }

      void MoveTo (int x, int y) {
        while(LocationY < y){
            LocationY++;
            SetRobot(LocationX,LocationY);
            Refresh();
            LocationY--;
            UnSetRobot(LocationX,LocationY);
            Refresh();
            delay(StepTime);
            UnSetRobotTweezer(LocationX,LocationY);
            Refresh();
            LocationY++;
            delay(100);
        }
       while(LocationY > y){
            LocationY--;
            SetRobot(LocationX,LocationY);
            Refresh();
            LocationY++;
            UnSetRobot(LocationX,LocationY);
            Refresh();
            delay(StepTime);
            UnSetRobotTweezer(LocationX,LocationY);
            Refresh();
            LocationY--;
            delay(100);
        }
        while(LocationX < x){
            LocationX++;
            SetRobot(LocationX,LocationY);
            Refresh();
            LocationX--;
            UnSetRobot(LocationX,LocationY);
            Refresh();
            delay(StepTime);
            UnSetRobotTweezer(LocationX,LocationY);
            Refresh();
            LocationX++;
            delay(100);
        }
        while(LocationX > x){
            LocationX--;
            SetRobot(LocationX,LocationY);
            Refresh();
            LocationX++;
            UnSetRobot(LocationX,LocationY);
            Refresh();
            delay(StepTime);
            UnSetRobotTweezer(LocationX,LocationY);
            Refresh();
            LocationX--;
            delay(100);
        }
      }
```

```

void Refresh() {
  SPI.beginTransaction (SPISettings (2000000, LSBFIRST, SPI_MODE0));
  digitalWrite (LATCH, LOW);
  for(int i=15; i>=0; i--)
  {
    for(int j=3; j>=0; j--)
    {
      SPI.transfer(MagArray[i][j]);
    }
  }
  digitalWrite (LATCH, HIGH);
  SPI.endTransaction();
}

void ClearAll(){
  for(int i=0; i<=15; i++)
  {
    for(int j=0; j<=3; j++)
    {
      MagArray[i][j]=B00000000;
    }
  }
}
```

REFERENCES

[1] Whitesides, G. M., & Grzybowski, B. (2002). Self-assembly at all scales. Science, 295(5564), 2418-2421.

[2] Cecil, J., Powell, D., & Vasquez, D. (2007). Assembly and manipulation of micro devices—A state of the art survey. Robotics and Computer-Integrated Manufacturing, 23(5), 580-588.

[3] Gauthier, M., & Régnier, S. (Eds.). (2011). Robotic Microassembly. John Wiley & Sons.

[4] Ravindra, N.M., Fiory, A.T., and Shet, S., U.S. Patent 7,737,515 (15 Jun 2010).

[5] Grzybowski, B. A., Wilmer, C. E., Kim, J., Browne, K. P., & Bishop, K. J. (2009). Self-assembly: from crystals to cells. Soft Matter, 5(6), 1110-1128.

[6] Cross, M. C., & Hohenberg, P. C. (1993). Pattern formation outside of equilibrium. Reviews of Modern Physics, 65(3), 851.

[7] Shet, S., Revero, R. D., Booty, M. R., Fiory, A. T., Lepselter, M. P., & Ravindra, N. M. (2006). Microassembly Techniques: A Teview. Materials Science and Technology-Association for Iron and Steel Technology, 1, 451.

[8] Ramadan, Q., Uk, Y. S., & Vaidyanathan, K. (2007). Large scale microcomponents assembly using an external magnetic array. Applied Physics Letters, 90(17), 172502.

[9] Kuran, E. E., Tichem, M., & Staufer, U. (2012, September). Magnetic force driven self-assembly of ultra-thin chips. In Electronic System-Integration Technology Conference (ESTC), (pp. 1-6). IEEE.

[10] Shet, S., Mehta, V. R., Fiory, A. T., Ravindra, N. M., & Lepselter, M. P. (2004). The magnetic field-assisted assembly of nanoscale semiconductor devices: A new technique. JOM, 56(10), 32-34.

[11] Ravindra, N.M., Fiory, A.T., and Shet, S., U.S. Patent 7,217,592 (15 May 2007).

[12] Korb, J. (2003). Thermoregulation and ventilation of termite mounds. Naturwissenschaften, 90(5), 212-219.

[13] Yeh, H. J., & Smith, J. S. (1994). Fluidic self-assembly for the integration of GaAs light-emitting diodes on Si substrates. IEEE Photonics Technology Letters, 6(6), 706-708.

[14] Jacobs, H. O., Tao, A. R., Schwartz, A., Gracias, D. H., & Whitesides, G. M. (2002). Fabrication of a cylindrical display by patterned assembly. Science, 296(5566), 323-325.

[15] Srinivasan, U., Liepmann, D., & Howe, R. T. (2001). Microstructure to substrate self-assembly using capillary forces. Journal of Microelectromechanical Systems, 10(1), 17-24.

[16] Xiong, X., Hanein, Y., Fang, J., Wang, Y., Wang, W., Schwartz, D. T., & Bohringer, K. F. (2003). Controlled multibatch self-assembly of microdevices. Journal of Microelectromechanical Systems, 12(2), 117-127.

[17] Zheng, W., Chung, J. H., & Jacobs, H. O. (2005). Non-robotic fabrication of packaged microsystems by shape-and-solder-directed self-assembly. In Micro Electro Mechanical Systems, MEMS 2005. 18th IEEE International Conference on (pp. 8-11). IEEE.

[18] Fang, J., Wang, K., & Böhringer, K. F. (2004, June). Self-assembly of micro pumps with high uniformity in performance. In Solid State Sensor, Actuator, and Microsystems Workshop, Hilton Head Island, SC.

[19] Fang, J., & Böhringer, K. F. (2006). Parallel micro component-to-substrate assembly with controlled poses and high surface coverage. Journal of Micromechanics and Microengineering, 16(4), 721.

[20] Bohringer, K. F., Goldberg, K., Cohn, M., Howe, R., & Pisano, A. (1998, May). Parallel microassembly with electrostatic force fields. In Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on (Vol. 2, pp. 1204-1211). IEEE.

[21] Cohn, M. B., Howe, R. T., & Pisano, A. P. (1995, November). Self-assembly of microsystems using non-contact electrostatic traps. In ASME International Congress and Exposition, Symposium on Micromechanical Systems,(IC'95) (pp. 893-900).

[22] Perkins, J., Rumpler, J., & Fonstad, C. G. (2002). Magnetically assisted self-assembly—a new heterogeneous integration technique. MIT Technical Report.

[23] Valdastri, P., Corradi, P., Menciassi, A., Schmickl, T., Crailsheim, K., Seyfried, J., & Dario, P. (2006). Micromanipulation, communication and swarm intelligence issues in a swarm microrobotic platform. Robotics and Autonomous Systems, 54(10), 789-804.

[24] Sitti, M. (2007). Microscale and nanoscale robotics systems [grand challenges of robotics]. IEEE Robotics & Automation Magazine, 14(1), 53-60.

[25] Sitti, M., Micro-and nano-scale robotics. In American Control Conference, Proceedings of the 2004 (Vol. 1, pp. 1-8). IEEE.

[26] Şahin, E. (2004, July). Swarm robotics: From sources of inspiration to domains of application. In International workshop on swarm robotics (pp. 10-20). Springer, Berlin, Heidelberg.

[27] Fearing, R. S. (1996). A planar milli-robot system on an air bearing. In ROBOTICS RESEARCH-INTERNATIONAL SYMPOSIUM- (Vol. 7, pp. 570-581). MIT PRESS.

[28] Donald, B. R., Levey, C. G., & Paprotny, I. (2008). Planar microassembly by parallel actuation of MEMS microrobots. Journal of Microelectromechanical Systems, 17(4), 789-808.

[29] Pelrine, R. E. (1990, February). Room temperature, Open-Loop levitation of microdevices using diamagnetic materials. In Micro Electro Mechanical Systems, 1990. Proceedings, An Investigation of Micro Structures, Sensors, Actuators, Machines and Robots. IEEE (pp. 34-37). IEEE.

[30] Pelrine, R., Wong-Foy, A., McCoy, B., Holeman, D., Mahoney, R., Myers, G., & Low, T. (2012, May). Diamagnetically levitated robots: An approach to massively parallel robotic systems with unusual motion properties. In Robotics and Automation (ICRA), 2012 IEEE International Conference on (pp. 739-744). IEEE.

[31] Pelrine, R., Wong-Foy, A., Hsu, A., & McCoy, B. (2016, July). Self-assembly of milli-scale robotic manipulators: A path to highly adaptive, robust automation systems. In Manipulation, Automation and Robotics at Small Scales (MARSS), International Conference on (pp. 1-6). IEEE.

[32] Hsu, A., Cowan, C., Chu, W., McCoy, B., Wong-Foy, A., Pelrine, R., ... & Randall, J. (2017, July). Automated 2D micro-assembly using diamagnetically levitated milli-robots. In Manipulation, Automation and Robotics at Small Scales (MARSS), 2017 International Conference on (pp. 1-6). IEEE.

[33] Pelrine, R., Hsu, A., Cowan, C., & Wong-Foy, A. (2017, July). Multi-agent systems using diamagnetic micro manipulation—From floating swarms to mobile sensors. In Manipulation, Automation and Robotics at Small Scales (MARSS), 2017 International Conference on (pp. 1-6). IEEE.

[34] SRI International, "Magnetically actuated micro-robots for advanced manufacturing applications," 2015. https://www.youtube.com/watch?v=Bxb3-bT8uxk accessed on 1st October 2017.

[35] https://www.arduino.cc accessed 15 June 2016

[36] http://www.elecrow.com/download/datasheet-l9110.pdf accessed 15 June 2016.

[37] Texas Instruments, SCLS041I –DECEMBER 1982–REVISED SEPTEMBER 2015, http://www.ti.com/lit/ds/symlink/sn74hc595.pdf

[38] Leens, Frédéric. "An introduction to I 2 C and SPI protocols." Instrumentation and Measurement Magazine, IEEE 12.1 (2009): 8-13.

[39] Williams, A. (2002). Microcontroller projects using the Basic Stamp. CRC Press.

[40] http://fritzing.org/home/ accessed 15 June 2016

[41] https://www.autodesk.com/products/fusion-360/overview accessed 15 June 2016

[42] Haynes, W. M. (Ed.). (2016). CRC handbook of chemistry and physics, 97th edition. CRC press.

[43] http://sfs.sabic.eu/wp-content/uploads/resource_pdf/1448531609-74377051-LEXAN-9034-9030XXX-DATASHEET-US-2015.pdf accessed on 1st October 2017.

[44] Lee, C. Control and Manipulation of Magnetic Nanoparticles and Cold Atoms Using Micro-electromagnets, (2002).

[45] Griffiths, D. J. (1999). Introduction to electrodynamics, 3rd edition. Prentice Hall.

[46] https://github.com/Denvi/Candle accessed 15 June 2016

[47] https://github.com/grbl/grbl accessed 15 June 2016