INTELLIGENT CONTROL OF SHAPE MEMORY ALLOY ACTUATOR ARRAYS WITH ELECTRIC AND THERMOFLUIDIC INPUTS

by

Leslie James Flemming

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Mechanical Engineering

The University of Utah

August 2012

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Leslie James Flemming**

has been approved by the following supervisory committee members:

Stephen A. Mascaro	, Chair	**6/14/2012** Date Approved
Stacy M. Bamberg	, Member	**6/14/2012** Date Approved
Mark A. Minor	, Member	**6/14/2012** Date Approved
Eric R. Pardyjak	, Member	Date Approved
David E. Johnson	, Member	Date Approved

and by **Timothy Ameel** , Chair of

the Department of **Mechanical Engineering**

and by Charles A. Wight, Dean of The Graduate School.

ABSTRACT

Shape Memory Alloy (SMA) actuators are compact and have high force-to-weight ratios, making them strong candidates to actuate robots, exoskeletons, and prosthetics. However, these actuators are thermomechanical in nature and slow cooling rates can limit their performance. Electricity can resistively heat the SMA actuators very quickly to produce contraction. To improve the convective cooling, SMA wires have been embedded in vascular networks, allowing cold fluid to pass across the actuators and extend them faster. The vascular network can also deliver hot fluid to heat and contract the wire. To minimize the weight and size of the control hardware for the vascular and electrical networks, a scalable *N*x*N* architecture has been implemented that allows for 2*N* control devices to be shared amongst N^2 actuators. This Network Array Architecture (NAA) allows each actuator to be controlled individually or in discrete subarrays. However, this architecture does not allow all combinations of actuators to be activated simultaneously; therefore a sequence of control commands may be required to achieve the complete desired actuation.

This dissertation presents the development of an intelligent controller for large arrays of wet SMA actuators with electric and thermofluidic inputs. The controller uses graph theory to identify a sequence to control commands to optimize the performance of the actuators. By treating each actuator as binary (contracted / extended), the collected states of an actuator array can be represented as nodes of the graph and the discrete NAA control commands as the graph edges. By weighting the costs of the graph edges (actuation times, energy), graph theory algorithms can find a set of control commands to transition the array to the desired state with specific performance characteristics. NAA results in a multi-graph that has a large number of nodes (2^{NxN}) and is highly

interconnected, causing problems with scalability. The search algorithm has incorporated an expanding wavefront algorithm to construct only a small portion of the graph as needed. The computational cost to construct the graph has been minimized by using bitwise operations and the discrete nature of the array of binary actuators and the NAA control commands. The algorithm was implemented in MATLAB and it is able to identify the optimal solution for a 4x4 array with more than 14 million edges. By using an expanding wavefront, the algorithm, on average, explored less than 100 edges (<0.01%) in 0.03 seconds. A 6x6 array was optimized in 0.7 seconds, exploring just 2400 edges.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

My thanks go first to my wife, Sarah, for her love and support. I look forward to the adventures ahead of us. I would also like to thank my family for their support over this long journey.

Thanks to my advisor, Stephen Mascaro, for his support as I pursued my degree. I look forward to sharing the experiences and knowledge gained under his guidance with my future students. I also deeply appreciate the ideas and hard work contributed by my colleagues and instructors at the University of Utah.

This work is supported by the National Science Foundation: Award 1031848.

CHAPTER 1

INTRODUCTION

Robotic systems such as prosthetics, exoskeletons and haptic devices will be wearable and impact a large number of people on a personal level. Prosthetic limbs restore functionality to amputees [1] and facial prostheses could reproduce facial expressions to restore dignity to those with facial disfigurements [2]. As well as augmenting strength, exoskeletons can be used to manipulate joints to restore mobility to individuals with muscular and neurological problems [3, 4]. Wearable haptic devices will allow people to interact with virtual environments and teleoperate other robotic systems [5].

The wearability of robotic systems is enhanced when robot actuation closely matches the characteristics of human actuation according to metrics such as DOF, size, strength, speed, and range of motion [6, 7]. Safety from mechanical, electrical, thermal, chemical, and audible hazards/discomforts must also be considered in the design of wearable robotics [8]. Additionally, prosthetics and exoskeletons require a long-lasting untethered energy source [9].

Shape Memory Alloys (SMA) refer to a group of materials that have the ability to return to a predetermined shape when heated and this characteristic can be utilized to produce actuation. SMAs have been described as artificial muscles, where heating will make them contract and cooling will allow them to extend. SMA "muscles" are normally compact because they are extremely strong with a maximum stress of about 200MPa (800 times greater than human muscle [10]). Therefore a small amount of material can produce the force, but generally they have strains on the order 4% (1/5 of human muscle).

To overcome this limitation, mechanical advantage can amplify the displacement while sacrificing some of the force. Additionally, they operate silently without vibrations.

SMA actuators have been used in a variety of robotic applications such as robotic hands [11] and faces [12], prostheses [13-17], surgical devices [18], and tactile displays [19-21]. One of the greatest limitations of SMA actuators has been their relatively low bandwidth (typically 1 Hz or less). The impediment to high frequency operation has been the cooling time constant. For example, a 0.8 mm diameter nickel titanium (NiTi) SMA wire can be electrically activated in about 50 ms, while the cooling time using free convection in air is about 1 second [22]. It has been observed that the time constant of relaxation improves up to four times if subjected to forced convection [23]. If an effective cooling mechanism is designed to rapidly remove heat from the wire, a substantial improvement in the relaxation time can be achieved, allowing for higher cycling rates. Furthermore, if such a cooling system could be achieved without the addition of bulky fans or fluidic tanks, SMA actuators could be more effectively used for the applications mentioned above. Some researchers have designed actuators where the SMA itself is a fluid-filled tube [24] or the SMA is nestled between fluid-filled tubes [18]. However in these configurations, only part or none of the SMA surface is actually in direct contact with the fluid. Electrothermal cooling with Peltier modules has also been implemented [25], but this solution is inefficient. Another idea that has recently achieved publicity is a fuel-powered SMA actuator [26], where a catalyst coated SMA wire is heated by a local fuel-cell reaction; however this technique has yet to match the speed of electric heating and offers no solution to fast cooling.

In order to improve the convection rate, [27] developed *vasculated robotic flesh*, where a robot is endowed with a network of “blood vessels” for removing thermal energy to embedded SMA actuators. These wet actuators consist of compliant vessels containing Shape Memory Alloy (SMA) wires. In addition to cooling, the vascular network can

deliver hot fluid in conjunction with electricity to contract the SMA wires. These wet SMA actuators have a theoretical power-to-weight ratios on the order of 90 W/kg (100 times greater than DC motors), with bandwidths of at least 5 Hz using a combination of electric heating and fluidic cooling [28]. Fluidic heating and cooling has a theoretical speed and efficiency of 2.5 Hz and 0.18% respectively. Fluidic activation is less efficient than electricity, but a chemical energy source used to heat the fluid can have up to 100 times the energy density of batteries.

Since SMA actuators are so compact, creating SMA bundles [29] and arrays [30] is a natural extension to these artificial muscles. However, the control devices, especially the fluidic valves, can be orders of magnitude larger and heavier than the SMA wire. To overcome this problem, wet SMA actuator arrays have been designed and implemented by [31, 32], where networks of fluidic valves are used to direct hot and cold water to any actuator in the array, while networks of electrical switches control electric current to any actuator. The architecture allows an NxN array of actuators to be controlled with 2N fluidic valves and 2N electric switches. This architecture does place the constraints on controlling the array actuators. Energy can be delivered/removed to each actuator individually or to certain subsections of the array [33]. The number of subarrays is small, $(2^N\text{-}1)^2$, relative to the possible desired heat transfer processes combinations and it is unlikely that a single subsection will deliver/remove the energy to produce the desired actuation of the array. Therefore, it may take multiple control commands (subarrays) to produce the complete desired actuation.

This dissertation presents development of an intelligent controller for arrays of SMA actuators with multiple energy domain inputs. Chapter 2 examines and develops the discrete control of wet SMA actuator array using thermofluidic inputs. Chapter 3 uses computation fluid dynamics to characterize the performance of a wet SMA actuator with both electric and thermofluidic inputs. Chapter 4 presents an optimal control strategy

using graph theory (path planning) to maximize the performance to the actuator array using both electric and thermofluidic inputs. Finally, Chapter 5 will examine some of the intricacies of the developed intelligent controller and future work to address them.

1.2 References

[1] E. Schaffalitzky, P. Gallagher, M. Maclachlan and N. Ryall, "Understanding the benefits of prosthetic prescription: exploring the experiences of practitioners and lower limb prosthetic users," *Disability & Rehabilitation*, vol. 33, issue 15, pp. 1314-1323, 2000.

[2] J. Pacey-Lowrie, "Working with prosthetic," *Optician*, vol. 234, issue 6121, pp. 26-27, 2007.

[3] K. Kiguchi, R. Esaki, and T. Toshio, "Development of a wearable exoskeleton for daily forearm motion assist," *Advanced Robotics*, vol. 19, issue 7, pp. 751-771, 2005.

[4] M. Samer, Y Amirat, and H. Rifai, "Lower-Limb Movement Assistance through Wearable Robots: State of the Art and Challenges", Advanced Robotic, vol. 26, issue 1/2, pp. 1-22, 2012.

[5] L. Sangyoon and G. Kim, "Effects of haptic feedback, stereoscopy, and image resolution on performance and presence in remote navigation," *International Journal of Human-Computer Studies*, vol. 66, issue 10, pp. 701-717, 2008.

[6] L. Tommaso, N. Vitello, S. Rossi, A. Persichetti, F. Giovacchini, S. Roccela, F. Vecchi, and M. Carrozzo, "Measuring human–robot interaction on wearable robots: A distributed approach," *Mechatronics*, vol. 21, issue 6, pp. 1123-1121, 2011.

[7] S. Yong-Ho, J. Il-Woong and Y. Hyun, "Motion capture-based wearable interaction system and its application to a humanoid robot, AMIO," *Advanced Robotics*, vol. 21, issue 15, pp. 1725-1741, 2007.

[8] A. Ivorra, C. Daniels, and B. Rubinsky, "Minimally obtrusive wearable device for continuous interactive cognitive and neurological assessment," *Physiological Measurement*, vol. 29, issue 5, pp. 543–554, 2008

[9] K. Shorter, G. Kogler, E. Loth, W. Durfee, and E. Hsiao-Wecksler, "A portable powered ankle-foot orthosis for rehabilitation," *Journal of Rehabilitation Research & Development*, vol. 48, issue 4, pp. 459-472, 2011.

[10] I. W. Hunter and S. Lafontaine, "Comparison of muscle with artificial actuators," in *Proc. IEEE Solid-State Sensors and Actuator Workshop*, Hilton Head Island, SC, USA, pp. 178-185, 1992.

[11] F. Garcia-Cordova, J. Lopez-Coronado, and A. Guerrero-Gonzalez, "Design of an anthropomorphic finger using shape memory alloy springs," in *Proc. IEEE International Conference on Systems, Man and Cybernetics*, Tokyo, Japan, pp. 242-247, 1999.

[12] H. Kobayashi, H. Akasawa, and F. Hara, "Study on new face robot platform for robot-human communication," in *Proc. IEEE International Robot and Human Communication Workshop*, Pisa, Italy, pp. 242-247, 1999.

[13] C. Pfeiffer, K. DeLaurentis, and C. Mavroidis, "Shape memory alloy actuated robot prostheses: initial experiments," *IEEE International Conference on Robotics and Automation*, Detroit, MI, USA, vol. 3, pp. 2385-2391. 1999.

[14] C. Pfeiffer, C. Mavroidis, K. DeLaurentis, and M. Mosley, "Shape memory alloy actuated robot prostheses: initial prototypes," *American Society of Mechanical Engineers,Bioengineering Division*, Nashville, TN, USA, 1999.

[15] K. Andrianesis and A. Tzes, "Design of an anthropomorphic prosthetic hand driven by shape memory alloy actuators," in *Proc. 2nd Biennial IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics*, Scottsdale, AZ, USA, pp. 517-522, 2008.

[16] T. Dutta and T. Chau, "A feasibility study of Flexinol as the primary actuator in a prosthetic hand," in *Proc. Canadian Conference on Electrical and Computer Engineering*, Piscataway, NJ, USA, vol. 3, pp. 1449-1452, 2003.

[17] K. O'Toole, M. McGrath, and D. Hatchett, "Transient characterization and analysis of shape memory alloy wire Bundles for the actuation of finger joints in prosthesis design," *Mechanika*, vol. 68, pp. 65-69.

[18] K. Ikuta, M. Tsukamoto, and S. Hirose, "Shape memory alloy servo actuator system with electric resistance feedback and application for active endoscope," *in Proc. IEEE International Conference on Robotics and Automations*, Philadelphia, PA, USA, pp. 427-430, 1988.

[19] P. M. Taylor, A. Moser, and A. Creed, "The Design and control of a tactile display based on shape memory alloys," in *Proc. IEEE International Conference on Robotics and Automation*, Albuquerque, NM, USA, pp. 1318-1323, 1997.

[20] S. Tachi, M. Nakatani, H. Kajimoto, K. Vlack, D. Sekijuchi, and N. Kawakami, "Control method for a 3d form display with coil-type shape memory alloy " in *Proc. IEEE International Conference on Robotics and Automation*, 2005.

[21] M. Nakamura and L. Jones, "An actuator for the tactile vest - a torso-based haptic device," in *Proc. 11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, Los Alamitos, CA, USA, pp. 333-339, 2003.

[22] I. W. Hunter, S. Lafontaine, J. M. Hollerbach, and P. J. Hunter, "Fast reversible NiTi fibers for use in microrobotics," in *Proc. IEEE Conference on Micro Electro Mechanical Systems*, Nara, Japan, pp. 166-170, 1991.

[23] A. Shahin, P. Meckl, J. Jones, and M. Thrasher, "Enhanced cooling of shape memory alloy wires using semiconductor `heat pump' modules," *Journal of Intelligent Material Systems and Structures*, vol. 5, pp. 95-104, 1994.

[24] J. Boyd, C. Tesluk, and A. Duncan, "Shape memory alloy heat pipe," *Proc. SPIE Conference on Smart Materials and Structures*, San Diego, CA, USA, pp. 44-447, 1997.

[25] B. Selden, K. Cho, and H. Asada, "Segmented binary control of shape memory alloy actuator systems using the peltier effect," in *Proc. IEEE International Conference on Robotics and Automation*, New Orleans, LA, USA, pp. 4931-4936, 2004.

[26] V. Ebron, Z. Yang, D. Seyer, M. Kozlov, J. Oh, H. Xie, J. Razal, L. Hall, J. Ferraris, A. MacDiarmid, and R. Baughman, "Fuel-powered artificial muscles," *Science*, vol. 311, pp. 1580-1583, 2006.

[27] S. Mascaro and H. Asada, "Wet shape memory alloy actuators for active vasculated robotic flesh," in *Proc. IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, vol. 1, pp. 282-287, 2003.

[28] L. Flemming and S. Mascaro, "Analysis of hybrid electric/thermofluidic control for wet shape memory alloy actuators," in *Proc. ASME Dynamic Systems and Controls*, Hollywood, CA, pp. 129-135, 2009.

[29] K. De Laurentis, A. Fisch, J. Nikitczuk, and C. Mavroidis, "Optimal design of shape memory alloy wire bundle actuators," in *Proc. IEEE International Conference on Robotics and Automation*, Washington, DC, USA, pp. 2363-2368, 2002.

[30] R. Mukherjee, T. Christian, and R. Thiel, "An actuation system for the control of multiple shape memory alloy actuators," *Sensors and Actuators A: Physical*, vol. 55, pp. 367-382, 1996.

[31] S. Mascaro and H. Asada, "Vast DOF wet shape memory alloy actuators using matrix manifold and valve system," ASME International Mechanical Engineering Congress, Dynamic Systems and Control Division, Washington, DC., USA, pp. 577-582, 2003.

[32] L. Flemming and S. Mascaro, "Wet SMA actuator array with matrix vasoconstriction device," in *Proc. ASME International Mechanical Engineering Congress and Exposition*, Dynamic Systems and Control Division, Orlando, FL, USA, , pp. 1751-1758, 2005.

[33] L. Flemming and S. Mascaro, "Control of a scalable matrix vasoconstrictor device for wet actuator arrays," in *Proc. IEEE International Conference on Robotics and Automation*, Rome, Italy, pp. 648-653, 2007.

CHAPTER 2

DISCRETE CONTROL OF THERMOFLUIDIC INPUTS FOR WET SMA ACTUATOR ARRAYS

2.1 Abstract

Shape Memory Alloys (SMA) can be implemented as high force-to-weight ratio actuators. The linear actuation of SMA wire is thermomechanical in nature. The heating of SMA wire produces a contraction and can be completed in milliseconds with Joule heating. However, unforced air convection cooling, which results in extension, can be slower by multiple orders of magnitude. To increase the cooling rate, SMA wires have been embedded in a compact vascular network, which allows cold fluid to flow across the actuators. Not only can the vascular network remove thermal energy from the SMA wire, it can also deliver hot fluid to convectively heat the SMA. Fluidic heating is advantageous compared to electric heating because the wire and surrounding fluid can remain hot for an extended time, allowing the wire to remain contracted without further energy input.

In order to maintain the high force-to-weight ratio of the SMA actuators, a scalable Matrix Vasoconstriction Device (MVD) was developed to control the fluid flow to the vascular network. The MVD uses a network architecture that has $2N+1$ valves controlling the fluid flow to N^2 SMA actuators. However, the network architecture does come at a cost. The control of the actuators is now linked and not all of the actuators can be controlled simultaneously. The MVD has $2(2^N-1)^2$ control commands where each actuator can be operated individually or in certain subarrays simultaneously. The wet SMA actuators in the array will operate as independent binary actuators or with multiple

binary actuators connected mechanically in series to produce a range of discrete displacements.

This paper presents scalable algorithms to operate the MVD as it controls the hot and cold fluid flow to an *N*x*N* array. The wet SMA actuators in the array will operate as independent binary actuators or with multiple binary actuators connected mechanically in series to produce a range of discrete displacements. The series algorithms use a method of forecasting the intermediate states of the array to keep the algorithms scalable. Using these algorithms with array sizes up to 6x6, the MVD can continuously operate with an actuation rate of 3.2 times that of a single actuator. This work will serve as the foundation for future research that uses both electric and thermofluidic networks to optimize the speed and efficiency of wet actuator arrays.

2.1 Introduction

As robots, exoskeletons, and haptic devices become more complex with large numbers of degrees of freedom (DOF); there will be a need for compact actuators to power these DOF. Humanoid robots such as the Honda Asimo [1] and the Sony SDR-X [2] are operating with about 30 DOF, which is an order of magnitude less than the human body [3]. These robots are also significantly heavier than a human of comparable size due to heavy electric motors, gearing and batteries. When operating at their fastest rates, these robots have only enough power to operate for about 30 minutes. As well as augmenting strength, exoskeletons can be used to manipulate joints to restore mobility to individuals with muscular and neurological problems [4],[5]. Wearable haptic devices will allow people to interact with virtual environments and teleoperate other robotic systems [6]. In order for these robotic systems to approach the mobility, dexterity and power of humans, hundreds or even thousands of compact high power-to-weight ratio actuators may be needed.

Shape Memory Alloys (SMA) actuators have high force-to-weight ratios and have been

described as artificial muscles because they are able to contract and extend like biological muscles. SMAs are able to strain 4-8% and are capable of strengths of 200 MPa (800 times stronger than human muscles) [7]. SMA wires contract and extend as the crystalline structure changes, which is heavily dependent on temperature. Increasing the temperature of the SMA wire above its transformation temperature will produce a contraction, and allowing it to cool back down will result in extension. Contraction can be done quickly with resistive heating, but the extension resulting from cooling with free convection can be multiple orders of magnitude slower [8]. Cooling rates of SMA have been improved by forced convection, water baths, and conduction with Peltier modules [9]-[11].

Biological muscles are supported by a circulatory system that delivers energy, removes waste and regulates temperature. Using this as inspiration, a vascular network has been used [12]-[21] to deliver/remove heat from embedded SMA wires. These *wet SMA actuators* have improved cooling rates over free air convection and can be modulated based on flow rates and fluid temperature. Simulation of a wet SMA actuator using electricity to heat and 25 °C fluid to cool has produced cycle rates of 4.5 Hz and an efficiency (mechanical power out/electric power in for a complete contraction/extension cycle) of 0.5%. Thermofluidic heating with 90 °C fluid and 25 °C fluid to cool, at 5 mL/s, has a simulated cycle rate of 2.4 Hz and an efficiency of 0.15% [16]-[19]. These are an order of magnitude faster than unforced air convection cooling rates (5.5 s) that the manufacturer has documented [8]. Although the efficiency of the electrically heated system is ~3 times greater than that of the thermofluidic heated system, the energy density of the thermofluidic power source, such as propane, may be 100 times greater than the electrical power source (batteries). Therefore, thermofluidic heating may have up to 30 times more actuation per unit mass than the electrical system. The focus of this paper will be the control of the vascular system to heat and cool the wet SMA actuators with fluid.

The size and weight of the control hardware for the fluidic system is multiple of orders

larger than the electrical system. To overcome this, an electric network array architecture (NAA) has been implemented that shares power transistors amongst the actuators requiring attention at any given time [13],[14]. This scalable architecture allows N^2 actuators to be controlled with $2N$ transistors. In the fluidic domain, scalability is more critical because of the size and weight of the valves. NAA has been implemented fluidically using a Matrix Vasoconstriction Device (MVD) which controls the hot and cold fluid flow through the array [15]. NAA allows each actuator to be controlled individually or in certain subarrays simultaneously, but not all combinations are controllable.

SMA actuators have nonlinear characteristics and are difficult to control; therefore for this work they will be treated as binary actuators, completely contracted or extended. To achieve a range of desired displacements and forces, SMAs have been bundled together in arrays so they can be connected in series and/or parallel [16]-[24].

This paper will present the development of a controller that maximizes the actuation rate of an *N*x*N* wet SMA actuator array. The possible MVD control commands are described and their influence on the fluid propagation through the system is identified. Two methods of identifying the best control command will be defined to maximize the actuation rate. Finally, the actuators will be linked mechanically in series and an algorithm to identify the most time effective control commands will be presented.

2.2 Background

2.2.1 *Wet Shape Memory Alloy (SMA)*

SMA is a smart material that can be used to create high force-to-weight ratio actuators. The shape memory effect allows these alloys to recover a predefined shape after being deformed (strained). This work uses Dynalloy Flexinol© [8], a nickel – titanium (NiTi) alloy in straight wire form, which produces a linear actuation when activated. This alloy has two crystalline structures that occur at different temperature and stress levels. The

soft martensite structure occurs below the transformation temperature of approximately 55 ºC and can be strained to 4-8%. When the wire is heated above 75 ºC, the stronger predefined austenite crystal structure is recovered, producing a linear contraction of the wire. SMAs can be heated (contracted) very quickly with electricity, but the unforced convective cooling (extension) may take multiple orders of magnitude longer than the heating process. Controlling the exact temperature of the SMA wire, and therefore the strain of the wire, is extremely difficult. Thus for many applications the SMA actuators are treated as binary, either completely contracted (1) or extended (0).

In order to improve the cooling rate, SMA wires are embedded in compliant vessels (Figure 2.1), allowing cold fluid to pass over the wire while maintaining the mechanical and electrical connections. The fluid can also be used to convectively heat the wire above the transformation temperature.

2.2.2 *Bundling Large Number of Wet SMA Actuators*

Even with the additional mass of the tubing and fluid, the wet SMA actuators still have higher force-to-weight ratios than traditional actuators. However, they still have limited displacements. To overcome this limited displacement, it has been suggested [22]-[24]

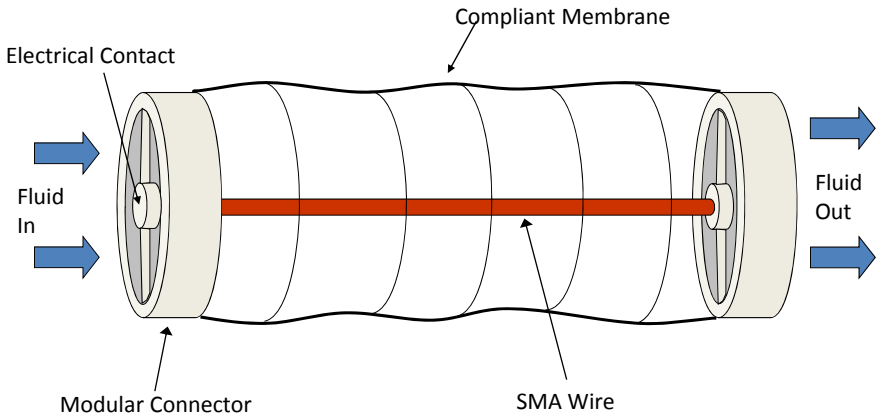


Figure 2.1: Wet SMA actuator. An SMA wire is in embedded in a compliant vessel with connections that allow for electricity and/or fluid to be used to heat the wire to produce contraction. Cold fluid can also be pumped through the vessel to cool the wire, resulting in extension.

that compact actuators can be bundled together in arrays, where the actuators are connected in series/parallel to produce the desired actuation. However, the valves and transistors necessary to control the fluid flow and electric current respectively are multiple orders heavier and larger than the wet SMA actuators themselves. Network Array Architecture (NAA) [11] assumes that on average, only a small percentage of the actuators must operate at a given time and that the transistors/valves could be shared by multiple actuators, allowing the system to be more scalable. With this architecture, N^2 actuators can be controlled by $2N$ control elements. NAA (Figure 2.2A) was implemented successfully using electricity to resistively heat the SMA actuators. However, the fluidic NAA implemented in the Matrix Manifold Valve (MMV) system [13] was unsuccessful, demonstrating parasitic effects because of the compliance of the tubing and high fluidic resistance of solenoid valves [15]. Figure 2.2B shows the Collocated NAA, a modified version of the original NAA, where single throw – multipole valves are arranged orthogonally and collocated on the source side of the actuators. This architecture was successfully implemented in the Matrix Vasoconstriction Device (MVD) [15]. In order to eliminate the fluidic resistance of the solenoid valve, a pneumatic constrictor valve was developed that crushes the wet SMA actuator vessel and when released, it does not contribute any fluidic resistance to the system. The constrictors were constructed to be single throw - multi pole “valves”, which allowed them to be collocated at the source side of the system, eliminating alternative paths through the system. A 4x4 MVD, shown in Figure 2.3, has been used to control 500 mm long wet SMA actuators that can contract and extend at a rate of 1 Hz. Since the MVD only controls the fluid distribution through the system, two additional vasoconstrictor valves are added to the system to select between the hot and cold water sources.

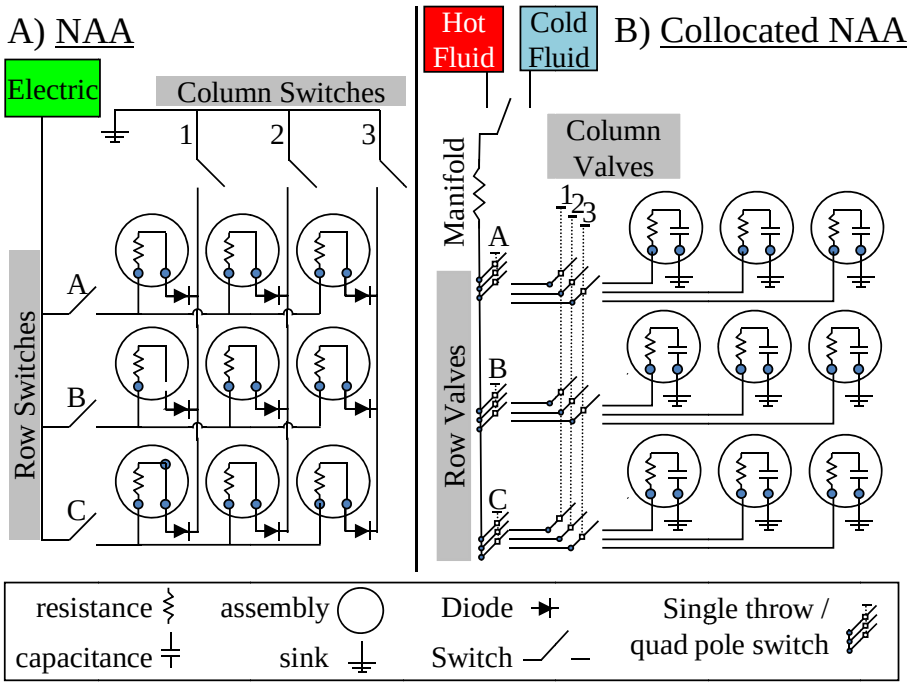


Figure 2.2: Network Array Architecture (NAA) shares the control hardware (switches/valves) to deliver/remove energy from the actuator array. A) Standard NAA for electric systems, B) Collocated NAA for fluid system.

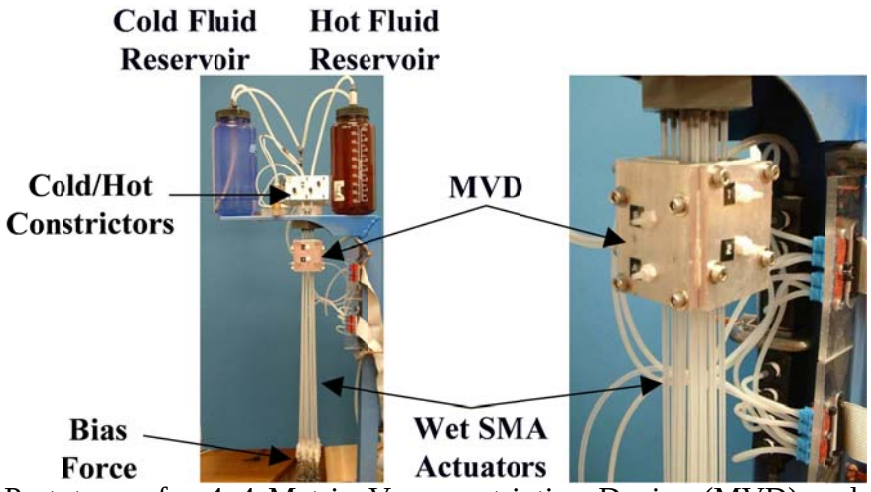


Figure 2.3: Prototype of a 4x4 Matrix Vasoconstriction Device (MVD) and wet SMA array. This prototype uses only hot and cold water to heat and cool the SMAs. The bias force for each wet SMA actuator is a 1 kg mass hanging vertically. The array has a total force capacity of 150 N.

2.2.3 *Binary Representation of SMA Actuator Array*

Since the actuators of the array are being treated as binary, the state of the array at any given time can be represented by N^2 digits or as a single N^2 bit binary number. Therefore the following 2x2 array, [1 0; 0 1], can be represented by the 4 bit binary number [1001]. By using a binary representation of the system, the computational cost of the control algorithm can be significantly reduced using bitwise operations. For example, if the state of each actuator of a 4x4 array were stored as an integer, it would take 16 individual comparisons (operations) of each actuator state. On the other hand, assuming that at least a 16 bit processor is being used, the bitwise comparison of the two 16 bit binary numbers would take only single math operation. For example, if the origin (starting state) of a 2x2 array is [1001] and the destination (state) is [0111], the actuators that need to change states can be identified by simply taking a bitwise OR of the two states. The result of is [1110], identifying that the first three actuators of the array need to change states. Additional bitwise algorithms will be discussed later in this paper to identify control commands to maximize performance. These bitwise operations are only between the state of the array and the control commands that the MVD can produces; this work does not implement any fluidic logic relative to fluid flow/pressures.

2.2.4 *Operating the Actuators Mechanically in Series*

NAA provides a scalable hardware solution to deliver/remove thermal energy to/from the array of wet SMA actuators, but each actuator can only produce a binary output, limiting the applications where it can be implemented. However, if a column of equal length actuators were joined mechanically in series, their combined displacement can take on N+1 discrete positions between 0 and N (Figure 2.4). Joining a column of actuators in series does reduce the number of output possibilities of the entire array to $N(N+1)$, but there may be multiple actuator combinations that can produce the desired output. If the desired output of an entire column is d (desired actuation) and the number of actuators in the column is N, then the number of possible combinations (C) of a single

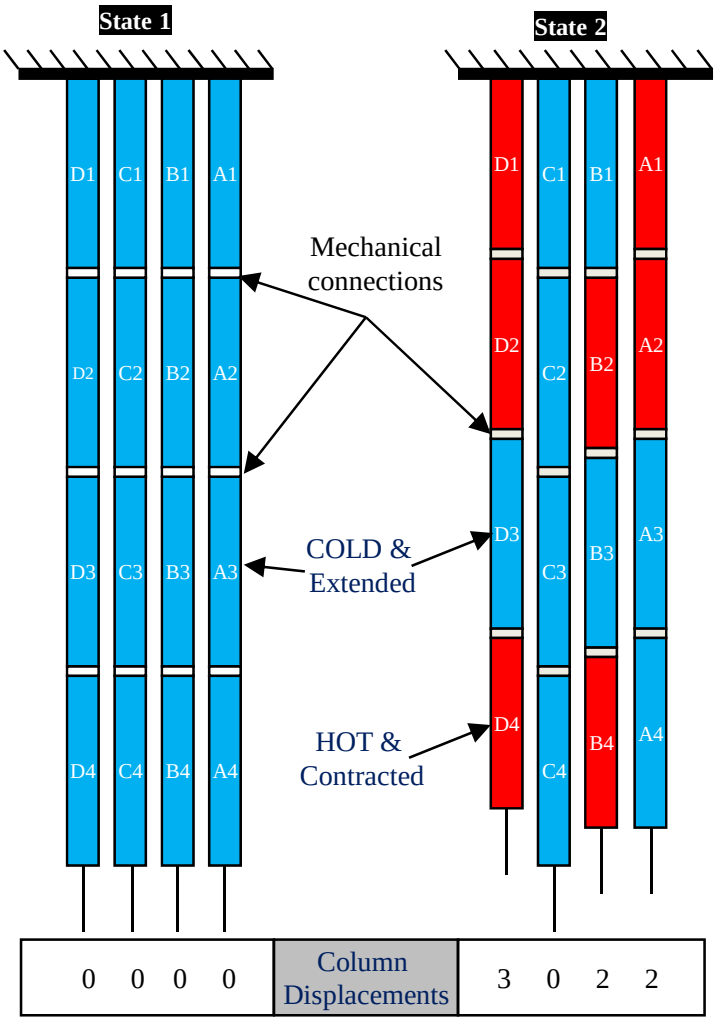


Figure 2.4: Two states of a 4x4 array of actuators where the columns of actuators are connected in series. State 1 has all actuators extended for a total displacement of [0 0 0 0] and State 2 has 7 actuators contracted for a displacement of [3 0 2 2]. Note that the connections are only mechanically in series; the electric and fluid connections are still made across rows and columns according to NAA.

column is described by:

$$ {}_N C_d = \frac{N!}{(N-d)!d!} \qquad (2.1) $$

For $N = 4$ and $d = 2$, the possible combinations are:

$$ \begin{bmatrix}1\\1\\0\\0\end{bmatrix}\begin{bmatrix}1\\0\\1\\0\end{bmatrix}\begin{bmatrix}1\\0\\0\\1\end{bmatrix}\begin{bmatrix}0\\1\\1\\0\end{bmatrix}\begin{bmatrix}0\\1\\0\\1\end{bmatrix}\begin{bmatrix}0\\0\\1\\1\end{bmatrix} $$

Accounting for the N columns of the array, the maximum number of possible

destinations, D_{max}, that can produce a *1xN* desired displacement command is defined by:

$$D_{\max} = \left(\frac{N!}{(N - floor(N/2))!\, floor(N/2)!} \right)^N \qquad (2.2)$$

where *floor(x)* is a function that results in the largest integer not greater than *x*.

For a 4x4 array, where the desired total displacement of each of the columns is 2, each column has 6 possible combinations and the array has a total of 6^4 or 1296 possible states resulting in the same desired displacement. Identifying the optimal state of the array becomes unmanageable when *N* is greater than 4, but an algorithm for operating the wet actuators in series has been developed based on forecasting how fluid flow will influence the system, and it will be presented in the following section.

2.3 Discrete Control Logic

2.3.1 *NAA Discrete Control*

Each valve of the NAA can be operated in saturation and can therefore be represented as a binary device that is either open (1) or closed (0). With NAA, *N* valves are arranged into sets of row and column valves. Each set of valves can take on 2^N configurations like an *N* bit number. Table 2.1 shows the configurations that a set of 4 rows and column switches can be in. When there are zero valves open, 0000, there is no complete circuit between the source and the sink, resulting in no energy flow through the array. Therefore 0000 would not produce any change in the system and leave 2^N-1 valid configurations. Since the NAA arranges the row and column valves orthogonally to one another, at least one row and one column valve must be open in order to complete a circuit between the source and sink. Therefore, there are $(2^N\text{-}1)^2$ discrete configurations of open valves that can complete the circuit and deliver/remove energy to various subsections of the array. Table 2.2 shows the size and the number of occurrences for similarly size subarrays. The NAA subarrays/control commands can also be tabled as N^2 bit binary numbers. The

Table 2.1: Binary number representation of valves (open=1, closed=0).

Number of Closed Valves	Column Configurations					
0	0000					
1	1000	0100	0010	0001		
2	1100	1010	1001	0110	0101	0010
3	1110	1101	1011	0111		
4	1111					

Table 2.2: Size and number of submatrix control commands of a 4x4 NAA.

Submatrix	1x1	1x2 2x1	1x3 3x1	1x4 4x1	2x2	2x3 3x2	2x4 4x2	3x3	3x4 4x3	4x4	Total
Occurrences	16	24 24	16 16	4 4	36	24 24	6 6	16	4 4	1	225

fluidic NAA can deliver either hot or cold fluid for a total of $2(2^N-1)^2$ discrete control commands that can deliver or remove thermal energy to/from the SMA wires.

Since the MVD control commands can only deliver or remove thermal energy to/from certain subsections of the array, it may take a sequence of control commands to produce the required heat transfer, as seen in Figure 2.5. The best solution will be based on material explored in the following section.

2.3.2 *Fluidic Resistance / Actuation Rates / Time Constants*

The convective heating and cooling of the SMA wire is a function of the fluid temperature along the length of the wire. In order for the fluid surrounding the wire to change temperature, the original volume of fluid must be displaced by the new volume of fluid, which takes a discrete amount of time based on the volumetric flow rates. Figure 2.6 shows a schematic of the MVD wet actuator array assembly and an equivalent electrical circuit model. Since the actuators are the primary source of fluidic resistance and are operating in parallel, a constant pressure source will deliver the most effective fluid flow. The manifold resistance is lumped into the actuator resistances. The constrictors are shown as single-throw single-pole switches for convenience, but the

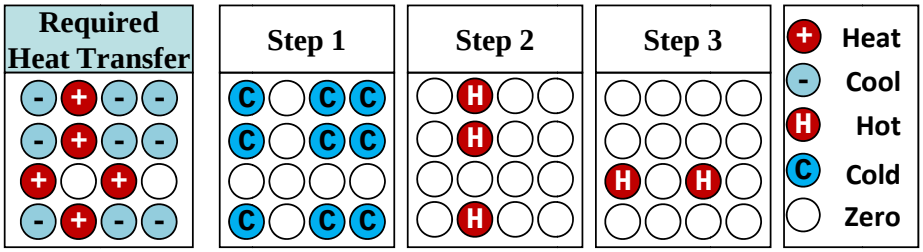


Figure 2.5: Required heat transfer process vs. output of MVD. Due to the shared control hardware of the MVD, a sequence of three control commands will need to be completed to achieve the required heat transfer process.

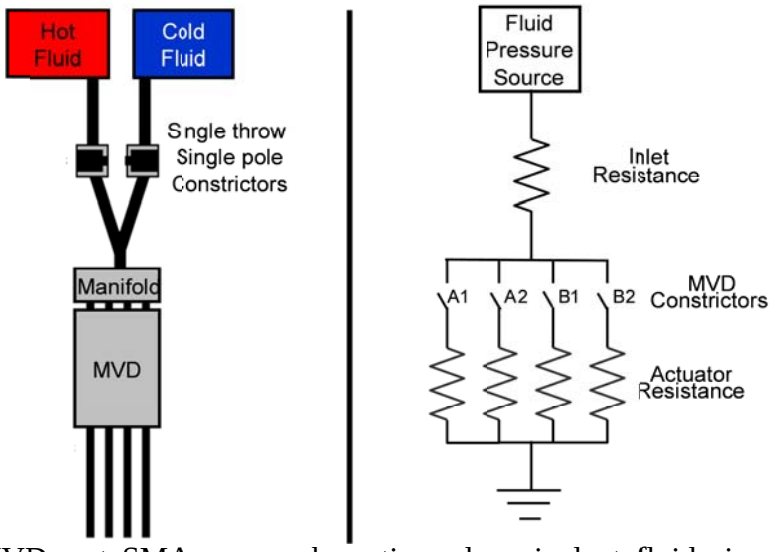


Figure 2.6: MVD wet SMA array schematic and equivalent fluid circuit: The single-throw single-pole constrictors and the MVD introduce no fluid resistance to the system, such that the tubing and manifold are the only components that introduce fluidic resistance to the system. The fluidic resistance of the manifold is lumped together with that of each of the actuators.

letters and numbers refer to the row and column valves. Both must be released simultaneously to complete the circuit. For example, in order to have flow through B1 and B2, the row constrictor B and column constrictors 1 and 2 must be released.

The total equivalent fluidic resistance (R_{eq}) of the system can be defined by the fluidic resistance of the inlet piping (R_i) plus the lumped resistance of a single actuator and manifold path (R_a), divided by the number of actuators being driven (d).

$$R_{eq} = R_i + \frac{R_a}{d} \qquad (2.3)$$

R_{eq} decreases as the number of driven actuator increases and therefore the total fluid flow rate increases. Because of the multiple connectors used to connect the tubing, the values of R_a and R_i were determined experimentally to have a linear relationship between pressure and flow rate, with a value of 1.5×10^{10} N·s/m^5 and 5.3×10^9 N·s/m^5 respectively:

$$P = QR \qquad (2.4)$$

where P (N/m^2) is pressure, Q (m^3/s) is flow rate and R (N·s/m^5) is the lumped fluidic resistance (tubing and connectors) of the actuator. Therefore with 100 kPa pressure source, the flow rate for a single actuator is 5 mL/s.

Equation (2.5) models (a_d) the number of actuations per actuation time of a single actuator (τ_1) for the MVD array,

$$a_d = \frac{d}{\tau_1} = \frac{\left(\frac{R_i}{R_a}+1\right)}{\tau_1\left(\frac{R_i}{R_a}+\frac{1}{d}\right)} \qquad (2.5)$$

where τ_1 is a function of the actuator dimensions, fluid properties, SMA properties, pressure source and inlet resistance.

Figure 2.7A shows how the ratio of R_i to R_a influences the actuation rate of a single actuator (a_1) in the MVD system with respect to $a_{1,0}$, the actuation rate of a single actuator where R_i is equal to zero. As the ratio of R_i to R_a decreases, the actuation rate increases, and the fastest actuation rate occurs when $R_i = 0$ (the actuator is connected directly to the pressure source.) Figure 2.7B shows how a_d, the actuation rate of the array with respect to actuation of a single actuator in the array (a_1), increases as the number of driven actuator (d) increases. As d approaches infinity, a_d approaches an asymptotic value of $(R_a/R_i+1)/\tau_1$. If $R_i = 0$, then $a_d = d/\tau_1$ and the actuation time for any number of driven actuators would remain constant (τ_1). For the prototype MVD, the R_i/R_a is 0.35, resulting in $a_1 = 1/\tau_1$ and $a_{16} = 3.26/\tau_1$. Since control commands that deliver fluid to larger subarrays have the higher actuation rates, identifying and controlling the largest

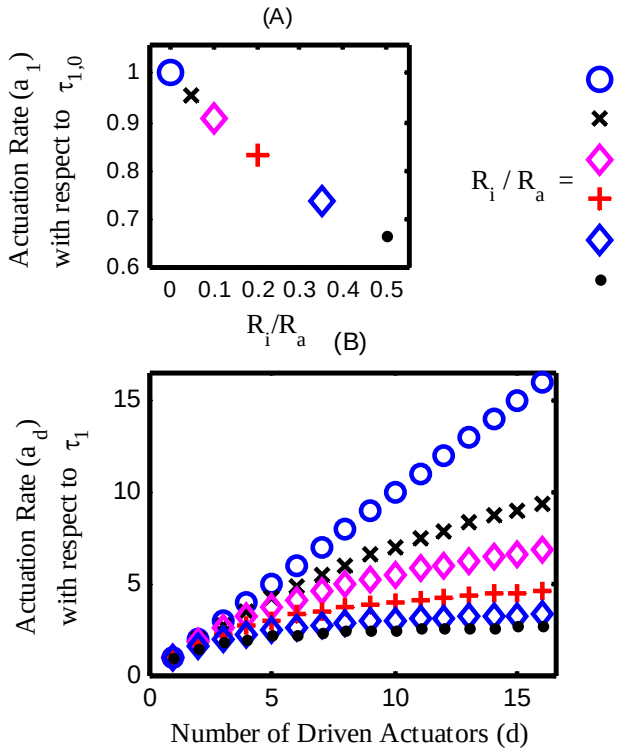


Figure 2.7: A) Actuation rate of a single actuator (a_1) vs. R_i/R_a. B) Actuation rate of the array (a_d) vs. number of driven actuators (d). The actuation rate a_1 is relative to the actuation time ($\tau_{1,0}$) of a single actuator attached to a manifold with zero fluidic resistance. The actuation rates a_d is relative to the actuation time (τ_1) of a single actuator in the array with a manifold that has fluidic resistance.

submatrices will maximizes the actuation rate of the SMA array. Re-examining Figure 2.5, the proposed solution uses an algorithm of identifying the largest submatrix and then working towards the smallest. The following section outlines the algorithm to identify the best MVD control commands.

2.3.3 *Control Command Forecasting*

As presented earlier and shown in Figure 2.5, there may not be a complete solution using a single control command from the origin to the destination, but a sequence of control commands will produce the actuations required to arrive at the destination. To identify the sequence of control commands for fastest actuation, a forecasting method has been implemented. From any given state, there are only $2(2^N\text{-}1)^2$ control commands that

can be applied. Some of these operations will produce zero actuations (complete superfluous flow), others will produce a partial actuation, and one may produce the complete actuation to the destination. A forecasting method is used to determine what would happen if a particular control command were applied to the origin or intermediate state. If the complete actuation is not achieved with a single control command, then from the conclusions in the previous section, the control command corresponding to the largest subarray would be identified in order to maximize the actuation rate. This control command would be applied to get to an intermediate state and the process would repeat until the destination was achieved.

Since both the states of the actuators and the control commands can be represented by binary numbers, bitwise operations can be used to forecast the potential intermediate states of the system that can be achieved using each control command. Three bitwise operators are used in the forecasting process: *AND*, *OR*, *NOT*.

2.3.4 *Standard Control Algorithm*

Since the MVD system has a single fluidic inlet, the best control command can only provide either hot or cold fluid; therefore both positive and negative heat transfer processes must be evaluated and compared to identify the largest control command. Figure 2.8 outlines the standard algorithm that identifies valid control commands that only deliver fluid to actuators that require their state to change. Any control command that would result in any fluid flow that does not produce an actuation is termed superfluous and is not valid for this algorithm. For example, if hot fluid were commanded to a subarray of actuators, one or more of which were already contracted, some or all of the flow would be superfluous, and the command would be invalid for this algorithm. Examining the HOT fluid side of the flow chart, the first bitwise AND ensures that there is no superfluous flow (valid if all zeros). If there is no superfluous flow, the second bitwise AND determines if the flow will move the system towards the destination (valid

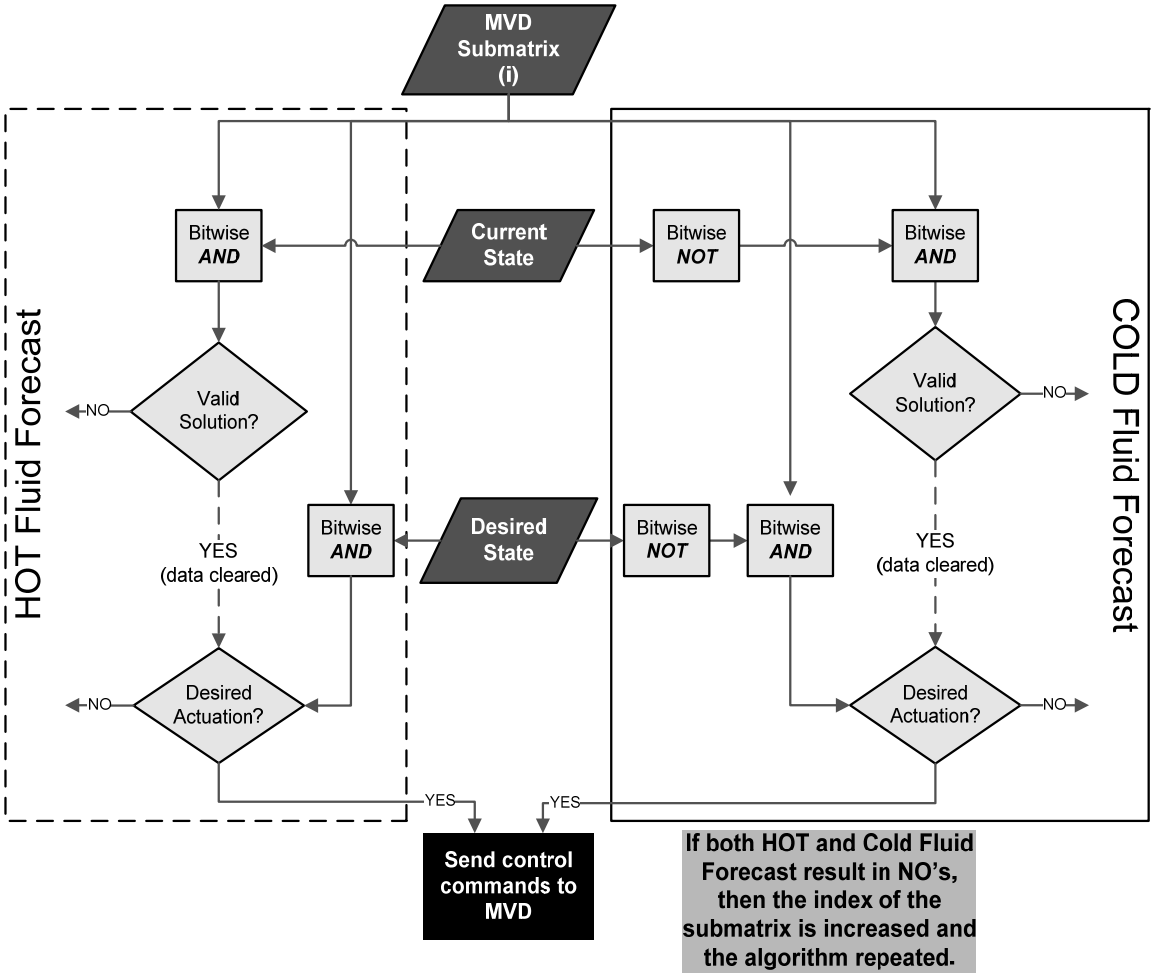


Figure 2.8: Standard control algorithm for MVD. This algorithm uses bitwise operations to identify the largest control command without superfluous flow that drives the array toward the destination. The control command submatrix list is sorted in a descending order, so that the algorithm can stop once a valid control command is found. This results in the fastest actuation rate.

if answer equals MVD submatrix(*i*)). If the forecasted state is not closer to the destination, then the algorithm iterates and another control command is tested. For all the search algorithms, the MVD submatrices have been presorted from the largest to the smallest, therefore once a valid control command has been identified, the control command is guaranteed to have the fastest actuation rate.

However, there may be other control commands of similar size that are also valid and may be preferred. In Figure 2.9, there are two solutions presented that produce the required heat transfer process. Both solutions have identified a 2x2 heating control

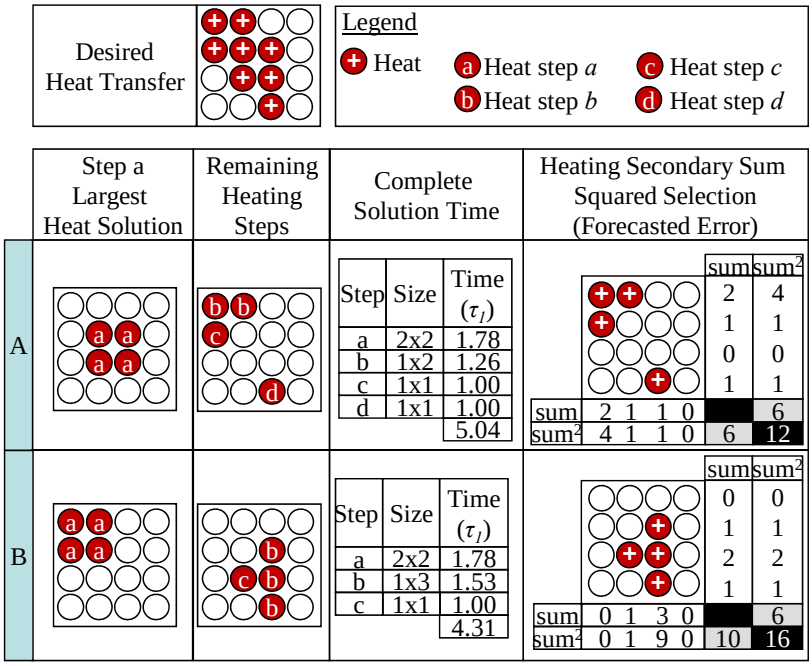


Figure 2.9: Secondary sum squared selection process for submatrices of similar size. Both control commands A & B have the same actuation rate, however the remaining heat transfer process for B can be achieved faster than for A, because of the larger remaining subarray with higher flow rates.

command as the first command of the solution. However, the total actuation time for A takes $0.73\tau_l$ longer than solution B. To identify the control command that will minimize the complete actuation sequence, a secondary selection process needs to be completed for each step when multiple solutions of similar size are found (1x4 vs. 2x2). A forecasted error (remaining required heat transfer processes) is simulated by applying the valid solution to the required heat transfer process. From this forecasted error, a weight is calculated by summing the rows and columns individually, then squaring these values and adding them together for a total weight. By selecting the solution with the maximum total sum squared weight, the contracted actuators and extended actuators will be grouped into larger subarrays. These larger subarrays will allow faster control commands to be identified as the system progresses to the destination. Solution B has the largest sum squared total weight and the option for a 1x3 control command in the remaining steps, while solution A is limited to 1x2 or 2x1 control commands in the remaining steps. If both positive and negative heat transfer is required and two control commands of the

similar size are identified, then the sum squared weight is calculated for both the positive and negative heat transfer processes independently and again the control command with the largest summed squared value is selected.

The HOT and COLD fluid forecasting of each control command is done simultaneously with a slight variation to the COLD forecast. The NOT of the origin/intermediate state is taken to identify the actuators that are extended. For example, if the system is represented by [1100], the first two actuators are contracted. By taking the NOT of the number, [0011], the last two actuators can be identified as extended.

2.3.5 *Superfluous Control Algorithm (SCA)*

Superfluous fluid flow to a single actuator results in no actuation, and a waste of thermal energy. However, when the actuator is part of an array, certain control commands would result in a combination of superfluous and nonsuperfluous flows. In these cases, superfluous flow can be tolerated (and even beneficial) because it allows for a larger subarray to be controlled, increasing the total flow rate and improving the average actuation rate. The Superfluous Control Algorithm (SCA), Figure 2.10, identifies control commands that address larger subarrays, and these control commands will on average have higher actuation rates than the standard method. Examples A and B in Figure 2.11 shows how superfluous flow would involve fewer steps and the task is completed faster than the non-superfluous actuation of the standard algorithm. Example C cannot use superfluous flow, because the first actuator would be heated and result in undesired actuation and the SCA would revert to the standard control algorithm. Simulation of both algorithms will be examined in the following section.

2.3.6 *Series Algorithm*

When a row of actuators are joined mechanically in series, the reference command for the system will be a 1xN array specifying the desired total displacement of each column of actuators. Since the number of combinations that will produce the desired

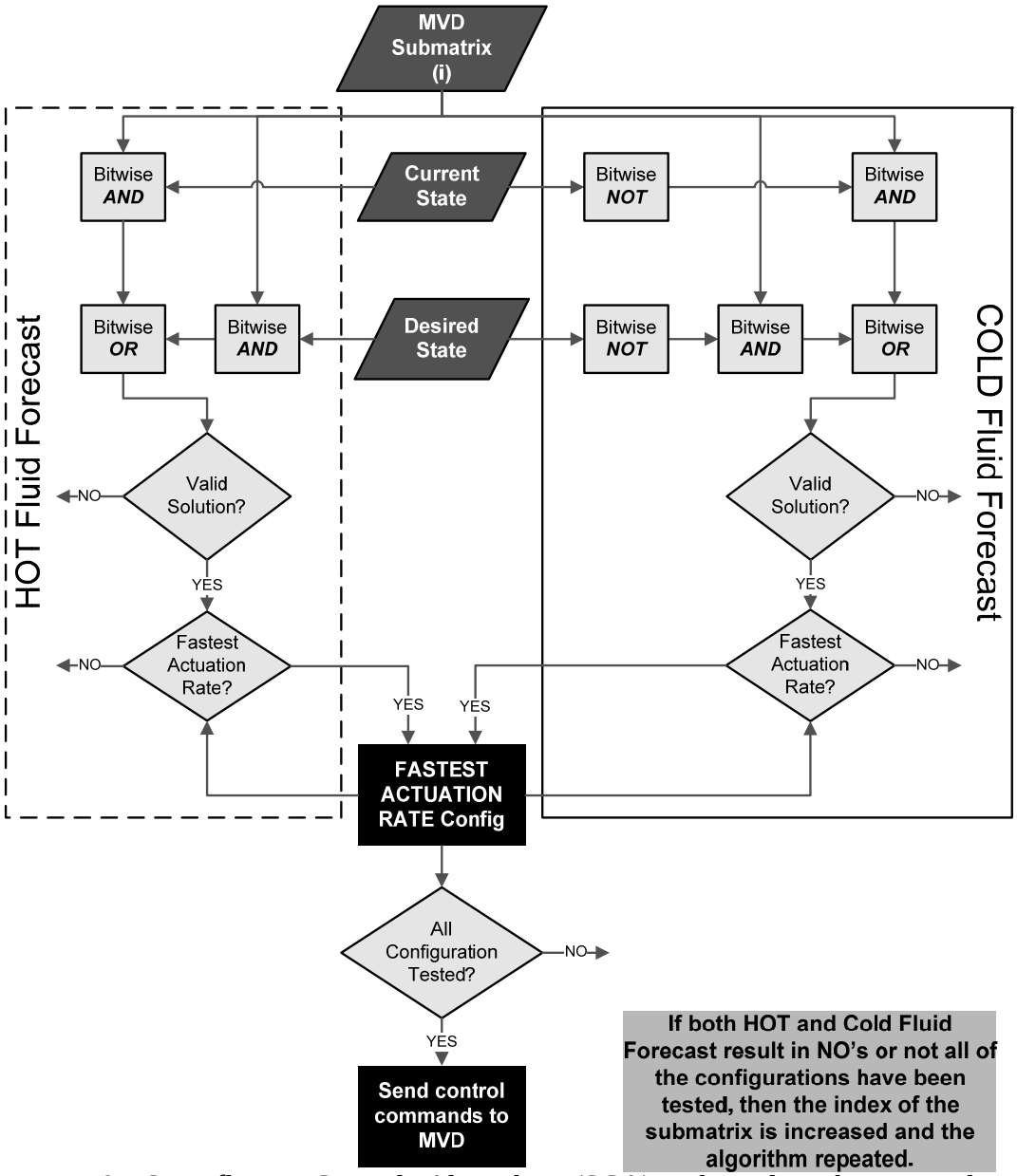


Figure 2.10: Superfluous Control Algorithm (SCA). This algorithm uses bitwise operations to identify the largest control command that drives the array toward the destination. This algorithm will allow superfluous flow, which is faster but uses more thermal energy. The control command submatrix list is sorted in a descending order, so that the algorithm can stop once a valid control command is found.

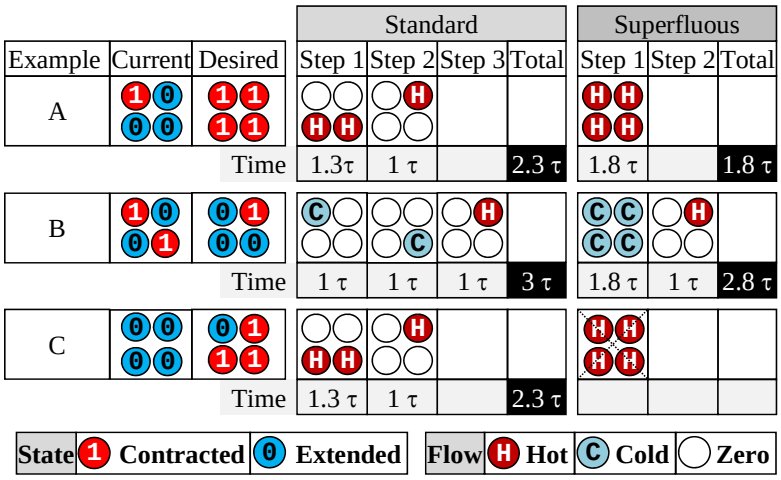


Figure 2.11: Comparison of standard and superfluous control algorithms for the MVD. Examples A & B show examples where superfluous flow results in faster actuation times. Example C shows a case where superfluous flow is not valid because it would result in the upper left actuator changing to an undesired state.

displacements may be very large, it is not feasible to test all of them. Unlike operating the actuators individually, where both origin and destination are known, only the total displacement of each column is known. For example, the column may have a desired displacement equal to 2; any combinations of contracted actuators, (A1, A2, A3 and A4) that add up to a displacement of 2 is a valid destination. Figure 2.12 will be used to visually work through an example that consists of 2 columns with 4 actuators each. The columns have a current displacement of {1 1} and it is desired to transition to a displacement of {3 3}. There are three combinations of control commands for each column where hot water is driven across the actuators that will produce the desired displacements. However, if a combination of commands *a* & *d* were to be applied, it would take at least a sequence of two 2x1 control commands to accomplish this task because they cannot be performed simultaneously. For this example, there are 9 different combinations, 3(column 1) x 3(column 2), that will produce the desired displacement. However, all but the combination *c* & *f* require a sequence of control commands. If *c* & *f* are used, it would take a single 2x2 control command to produce the desired displacements. These combinations can be used to generate possible destinations for the

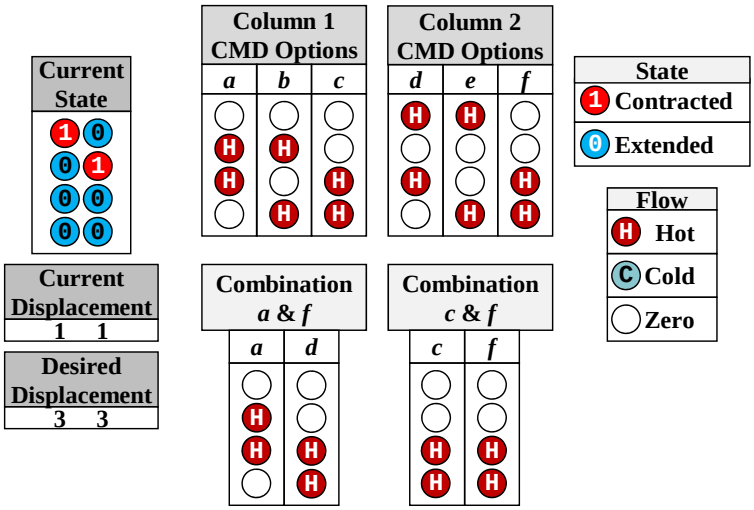


Figure 2.12 Optimizing the control algorithm for actuator connected in series. A combination/sequence of CMD options will transition the system from the current displacement to the desired displacement. Only options c & f can be done simultaneously in a single control command.

array, and then the standard or the superfluous control algorithms can be used to determine an optimal sequence of commands. This method becomes unmanageable because each combination would need to be tested against all of the possible control commands.

To overcome the large number of combinations for series operation, a subset of control commands can be identified and used to minimize the computational cost of the algorithm. Instead of identifying all of the control commands that can produce the desired displacements and their associated operational cost, a method of forecasting possible destinations will be used. The array of actuators can only be changed by applying one of $2(2^N\text{-}1)^2$ control commands. Therefore, by forecasting the response of the array to all of the control commands, the intermediate states of the array can be determined. From these intermediate states, the intermediate displacement can be calculated. The control command that results in the smallest total displacement error is the optimal control command, and the forecasting process is repeated until the desired displacement is achieved. Figure 2.13 shows how forecasting works with the system from Figure 2.12. Five of the possible MVD control commands are shown, along with the forecasted state

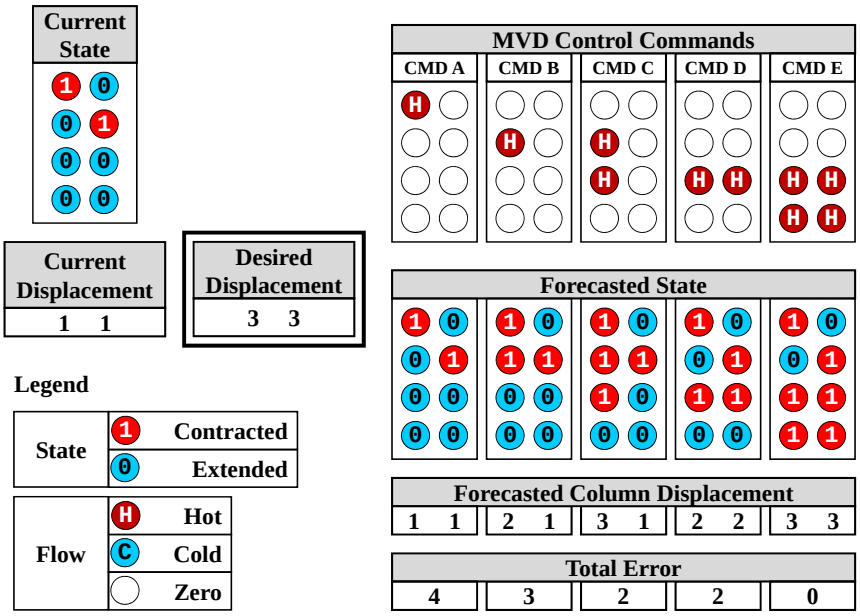


Figure 2.13: Forecasting possible destinations to optimize actuation rates and minimize search algorithm. The forecasted state is the predicted state of the array when the control commands are applied to the current state. The control command resulting in the forecasted state with the smallest total error is the one that will maximize the performance of the array.

of the array if each command were applied. CMD A produces zero actuation and the error to desired destination is unchanged, but the remaining four control commands result in actuation and reduce the error to the desired displacement. CMD E produces a complete solution and achieves the desired displacements. This algorithm uses bitwise operations and a solution can be found with at most 2N iterations.

Multiple MVD control commands may advance the system towards the desired displacement by the same amount, and like the case presented in Figure 2.9, one of the control commands may result in a preferred forecasted state. However, unlike the previous case where the exact required heat transfer process was known, there may be multiple solutions that result in the desired displacement. Instead of using the heat transfer processes the intermediate state of the array will be used to identify the preferred state. Figure 2.14 shows the origin, current displacements and the desired displacements. From this information, the algorithm can identity two 2x2 hot MVD control commands

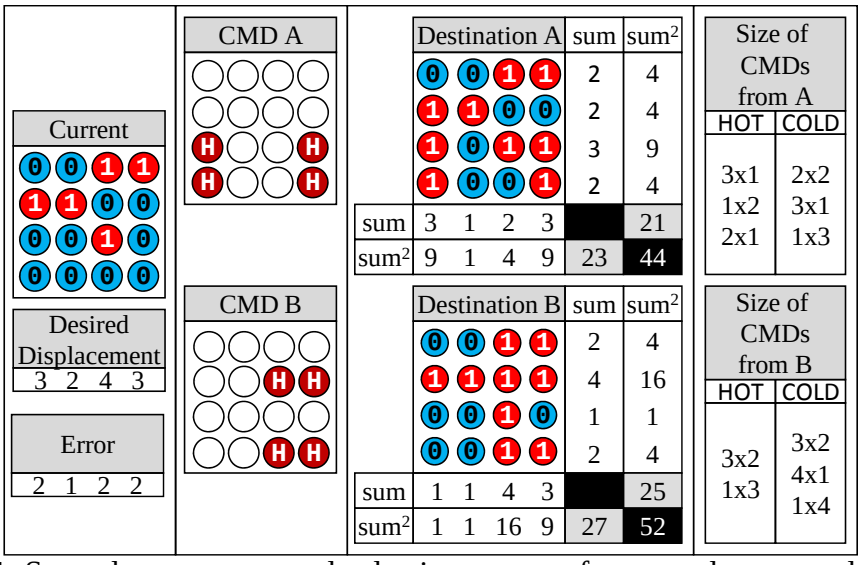


Figure 2.14: Secondary sum squared selection process for control commands of similar size. Both CMD A & B are the same size, but the destination (intermediate) state of B allows for larger control commands to be completed from this new intermediate state, which may result in faster actuation from this intermediate state.

that can transition the array towards the desired displacements the fastest. To compare these two intermediate states and identify a preferred one, the sum of each row and column are calculated and then squared. These values are then summed for a total system weighting for each possible intermediate state. The maximum sum squared value identifies the preferred destination. Although destination B has only a slightly higher weighting (51) than A (44), the contracted and extended actuators of B are arranged in larger subarrays and this allows larger control commands to be applied from this state.

This sum squared weighting works well when comparing multiple hot or multiple cold control commands, but it will always select the hot command when comparing two similar sized hot and cold commands. To properly identify the best intermediate state, the sum squared weighting of both the intermediated state and the NOT of the intermediate state must be used to identify the optimal control command as seen in Figure 2.15. In this example, command C would be preferred because it allows either of the actuators on the right side of the array to be heated to achieve the desired displacement.

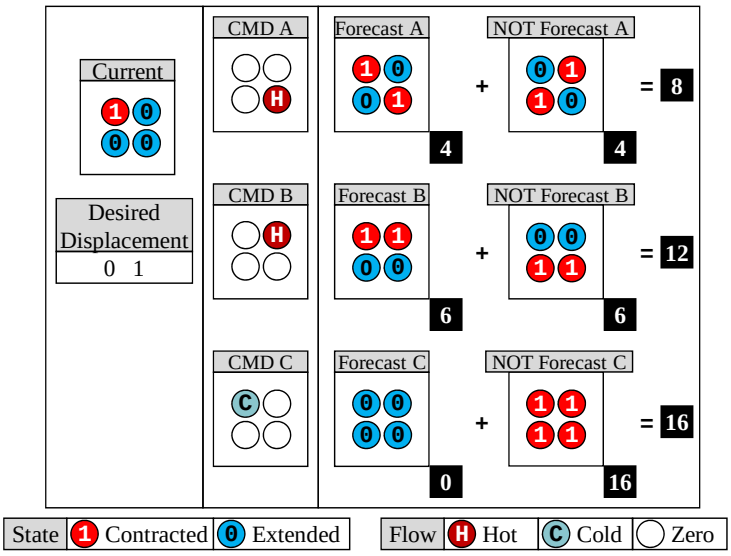


Figure 2.15: Evaluating similar hot and cold operations. If only the forecasted state with the largest sum squared value is used, the hot control command would always be preferred. By including the NOT of the forecasted state, CMD C can be identified as the preferred control command, because it offers more options from the forecasted (intermediate) state.

2.4 Simulations

To examine the performance of the MVD control algorithms, randomly generated *NxN* desired actuations are used as reference commands to the algorithm, which then determines the control commands that will maximize the actuation rate. Since the MVD may not be able to produce the complete *NxN* desired actuation in a single step, two methods of managing the reference commands will be used. 1) For each reference command, the system will be allowed to apply a sequence of control commands until the desired actuation is complete. A new reference command will be not be generated until the previous desired actuation is complete. 2) Reference commands will be generated on regular time intervals, regardless of whether the previous desired actuation is complete or not. A queue will be used to store the desired actuations (of individual actuators) that are not completed prior to the next interval. The algorithm examines all of the desired actuations in the queue in order to identify a control command that maximizes actuation

rates. The queue does not place preference on when the desired actuations were issued. The 4x4 prototype actuator array will be the basis for simulation, unless noted otherwise.

To examine the maximum theoretical actuation rate, a_d, the second method will be used. An *NxN* desired actuation will be added to the queue on a regular interval of τ_1, the time for a single actuator in MVD system to be activated individually. The simulation will vary the average number of actuators changing states per reference command.

Figure 2.16 shows the simulated response of the prototype MVD system, using the standard algorithm. When the average number of actuators changing states-per-unit time (τ_1) is 2.8 or 3.0, the average number of reference commands added to the queue is equal to the number of control commands sent to the MVD, therefore the system is considered stable. At a value of 3.2, the system appears to be marginally stable, having regions of slow increase and decrease in the number of commands in the queue. When the number of reference commands per unit time (τ_1) averages 3.4, the number of commands in the queue continues to increase and this is defined as unstable. This does match up well with the maximum theoretical value of a_d, 3.27 (a_{16}/a_1 = 2.42/0.74). The large number of

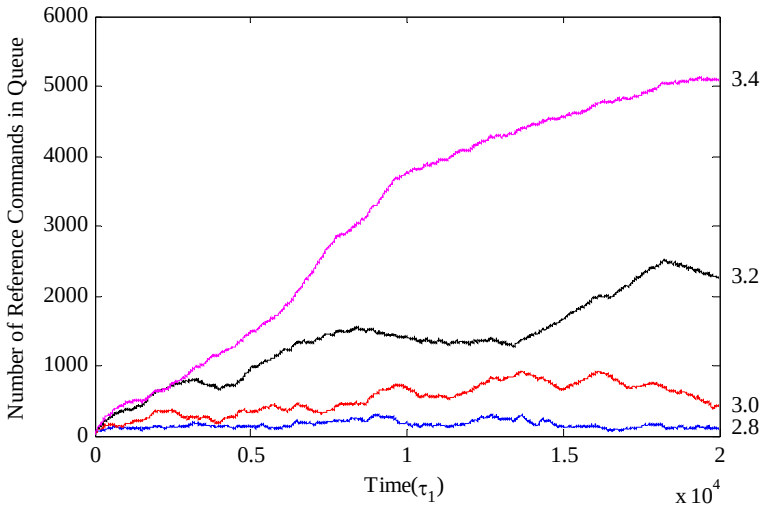


Figure 2.16: Number of Reference Commands in Queue vs. Time (τ_1). The numbers on right hand side of the graph are the average number of actuators being driven per reference command occurring every τ_1, a_d. When the number of reference commands remains constant, the actuator array is able meet the desired performance.

simulations in Figures 2.16, as well as 2.17 and 2.18, are used to validate stability trends. 2.17 shows how varying size of the MVD affects the response of the system. As the size of the array increases, there is an improvement of the performance. This is due to the increase in the number of possible MVD control commands with larger a_d and the increase in fluid flow rate for control commands that address larger subarrays.

This simulation was also used to compare the standard and superfluous control algorithms. Figure 2.18 shows that the standard algorithm is shown to be marginally stable at 3.2 desired actuations per unit time, where the performance of the SCA is stable and has completed 1766 more control commands, a 2.8% increase in actuation rate. At 3.4 reference commands per unit time, the SCA has completed 1920 more reference commands than the standard method, for a 6.6% increase in actuation rate. The series algorithm is used in conjunction with the standard algorithm and the SCA to identify a

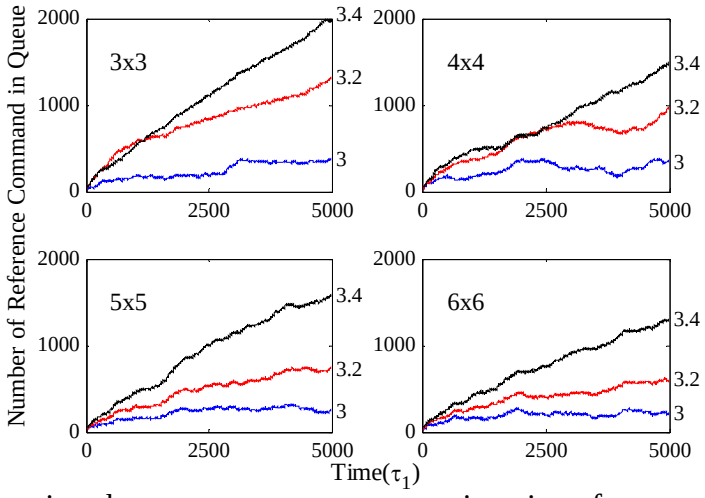


Figure 2.17: Comparing the system response to varying size of arrays. The numbers on right hand side of the graphs are the average number of actuators being driven per reference command occurring every τ_1 (a_d). As the size of the array increases, a_d also increases because of the larger number and size of the subarrays that the MVD can produce.

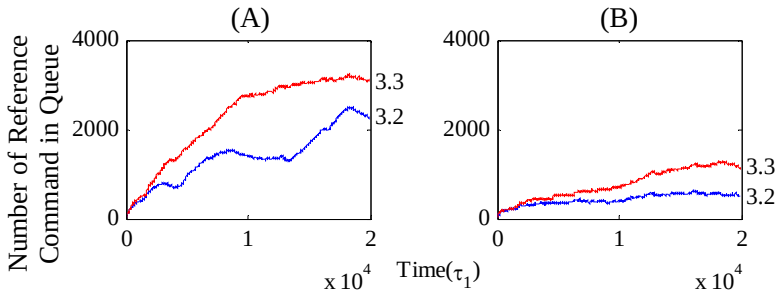


Figure 2.18: System response to A) standard control algorithm B) superfluous control algorithm*)*. The numbers on the right hand side of the graphs are the average number of actuators being driven per reference command (occurring every τ_1), a_d. The superfluous control algorithm results in higher actuation rates, but also uses more net thermal energy per actuation.

destination with the desired displacement, and therefore has similar results to these simulations. Figure 2.19 shows how the secondary sum squared selection process improves the performance of the system. Subplots A-C show the control commands to three reference commands using the maximum (best) and minimum (worst) sum squared values. The optimal solution eliminates at least 1 control command from each of these sequences. The optimal control commands are 14%, 12% and 33% faster than the worst solution, respectively. However, in about 87% of cases there is only a single solution to the reference command. Figure 2.19D shows 7 out of 50 control commands (marked by the black arrow heads) that can be optimized, which results in a 2% decrease in total actuation times for a 4x4 array.

2.5 Conclusions

This paper presents methods for controlling an array of wet SMA actuators utilizing a scalable Matrix Vasoconstriction Device (MVD) to control hot and cold fluid flow to N^2 wet SMA actuators with only $2N+1$ control valves. An algorithm was developed to search through the $2(2^N-1)^2$ possible control commands to identify those which maximize the actuation rate. Furthermore, a Superfluous Control Algorithm (SCA) was developed,

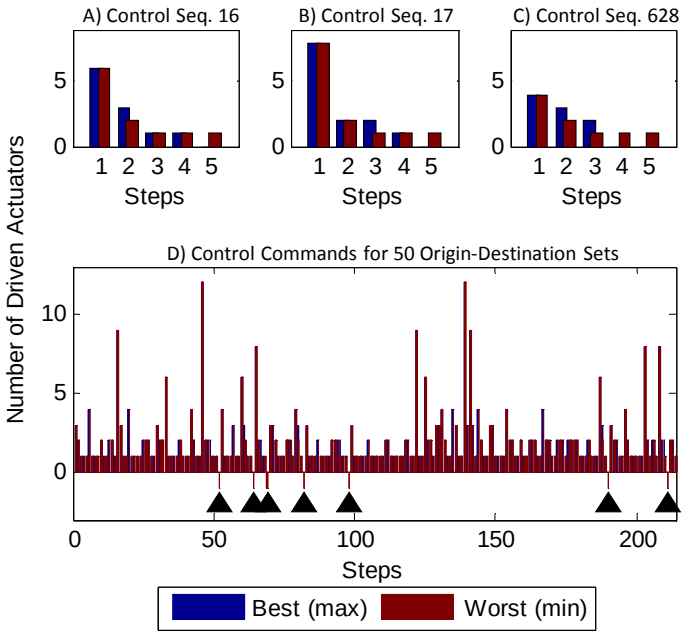


Figure 2.19: Secondary sum squared selection. A-C) shows the best (max) and worst (min) complete solution for three different origin-destination sets. The first control command for both the best and worst control commands are the same size, but at least one of the proceeding best control commands is larger than the worst control commands. Therefore, by using the best control command, the actuation time can be minimized. D) shows how often secondary sum squared selection occurs.

allowing for an even faster actuation rate by identifying control commands that deliver fluid (thermal energy) to larger subarray resulting in a faster actuation rate. However, these larger subarrays will have superfluous fluid flow across at least one of the SMA actuators that is already in the desired state. Therefore the SCA improves the average actuation rate of the entire array at the expense of reducing the average energy efficiency of the entire array. Using the SCA with array sizes up to 6x6, the MVD can continuously operate with an actuation rate of 3.2 times that of a single actuator.

Finally, the algorithms were adapted for the case in which a whole column of actuators are connected mechanically in series to allow for incremental actuation of a robotic joint. In this case, there are potentially a large number of destinations that would all result in the desired displacement. The algorithms were modified to use the intermediate state as a method of forecasting. For a 5x5 array (N=5), there could be up to 10^5, unique destinations that could result in the same displacement for actuators connected in series.

Rather than working backward from the 10^5 possible destinations, the forecasting method works forward from the $2(2^2\text{-}1)^2 = 1922$ control commands to optimize the actuation rate.

Future work will seek to frame the control of the wet actuator array as an optimal control problem where a weighted cost function of time and energy will be used to manage both electrical and thermofluidic inputs to the wet SMA actuators. If thermofluidic heating is used, chemical energy sources, such as propane, can be used to heat the fluid, and offer a greater number of actuations per unit mass of energy stored than an electrical energy source [16]. However electricity can be used to heat the SMA wire in a fraction of the time required for fluidic heating. The speed and efficiency of electric vs. fluidic activation are quantitatively characterized in [18] through dynamic modeling. Therefore [21] has implemented fluidic and electric control networks concurrently, allowing the controller to optimize the performance of the array based on a weighted combination of time and energy costs. Performance can also be improved by delivering electricity to one subarray while concurrently delivering fluid to another, which requires additional intelligent algorithms to coordinate. While this work has been motivated by research with SMA actuator arrays, it is anticipated that these control algorithms could also be applied to a more general class of actuators arrays with networked inputs.

2.6 References

[1] M. Hirose and K. Ogawa, "Honda humanoid robots development," *Philosophical Transactions of the Royal Society London, Series A (Mathematical, Physical and Engineering Sciences)*, vol. 365, pp. 11-19, 2007.

[2] T. Ishida, Y. Kuroki, J. Yamaguchi, M. Fujita, and T. T. Doi, "Motion entertainment by a small humanoid robot based on OPEN-R," *in Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1079-1086,* 2001.

[3] L. Sherwood, *Human physiology 7th ed.*, Belmont, CA, Cengage Learning, 2008.

[4] K. Kiguchi, R. Esaki, and T. Toshio, “Development of a wearable exoskeleton for daily forearm motion assist,” *Advanced Robotics*, vol. 19, issue 7, pp. 751-771, 2005.

[5] M. Samer, Y Amirat, and H. Rifai, "Lower-Limb Movement Assistance through Wearable Robots: State of the Art and Challenges”, *Advanced Robotics,* vol. 26, issue 1/2, pp.1-22, 2012.

[6] L. Sangyoon and G. Kim, “Effects of haptic feedback, stereoscopy, and image resolution on performance and presence in remote navigation,” *International Journal of Human-Computer Studies*, vol. 66, issue 10, pp 701-717, 2008.

[7] I. Hunter and S. Lafontaine, "A comparison of muscle with artificial actuators," in *Proc. IEEE Solid-State Sensors and Actuator Workshop,* Hilton Head, SC, USA, pp. 175-185, 1992.

[8] "Technical Characteristics of Flexinol Actuator Wires," Dynalloy Inc., Available: http://www.dynalloy.com/pdfs/TCF1140.pdf

[9] M. Nakatani, H. Kajimoto, K. Vlack, D. Sekiguchi, N. Kawakami, and S. Tachi, "Pop up!: 3D form display with coil-type shape memory alloy," *Journal of the Institute of Image Information and Television Engineers*, vol. 60, pp. 183-191, 2006.

[10] A. Shahin, P. Meckl, J. Jones, and M. Thrasher, "Enhanced cooling of shape memory alloy wires using semiconductor 'heat pump' modules," *Journal of Intelligent Material Systems and Structures,* vol. 5, pp. 95-104, 1994.

[11] B. Selden, K. Cho, and H. Asada, "Segmented binary control of shape memory alloy actuator systems using the peltier effect," in *Proc. IEEE International Conference on Robotics and Automation*, New Orleans, LA, USA, pp. 4931-4936, 2004.

[12] S. Mascaro and H. Asada, "Wet shape memory alloy actuators for active vasculated robotic flesh," in *Proc. IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, pp. 282-287, 2003.

[13] S. Mascaro and H. Asada, "Vast DOF wet shape memory alloy actuators using matrix manifold and valve system," in *Proc. ASME International Mechanical Engineering Congress*, Washington, DC, USA, pp. 1992-1997, 2003.

[14] S. Mascaro, K. Cho, and H. Asada, "Design and control of vast DOF wet SMA array actuators," in *Proc. of the IEEE International Conference on Intelligent Robots and Systems*, Las Vegas, NV, USA, pp. 577-582, 2003.

[15] L. Flemming and S. Mascaro, "Wet SMA actuator array with matrix vasoconstriction device," in *Proc. ASME International Mechanical Engineering Congress and Exposition*, Orlando, FL, USA, pp. 1751-1758, 2005.

[16] L. Flemming and S. Mascaro, "Control of a scalable matrix vasoconstrictor device for wet actuator arrays," in *Proc. IEEE International Conference on Robotics and Automation,* Rome, Italy, pp. 638-653, 2007.

[17] J. Ertel and S. Mascaro, “Thermomechanical modeling of a wet shape memory alloy actuator,” in Proc. ASME International Mechanical Engineering Congress and Exposition, Chicago, IL, USA, pp. 1317-1324, 2006.

[18] L. Flemming and S. Mascaro, "Analysis of hybrid electric/thermofluidic control for wet shape memory alloy actuators," in *Proc. ASME Dynamic Systems and Controls*, Hollywood, CA, USA, pp. 1041-1047, 2009.

[19] J. Ertel and S. Mascaro, “Dynamic thermomechanical modeling of a wet shape memory alloy actuator,” *Transactions of the ASME Journal of Dynamic Systems, Measurements and Control*, vol. 132, no. 4, pp. 1-9, 2010.

[20] L. Flemming, D. Johnson and S. Mascaro, “Optimal control of multi-input SMA actuator arrays using graph theory,” *in Proc. IEEE International Conference on Robotics and Automation*, Shanghai, China, pp. 780-785, 2011.

[21] L. Flemming, D. Johnson and S. Mascaro, “Optimal control of multi-input SMA actuator arrays using graph theory: expanding wavefront & simultaneous operation,” in *Proc. IEEE/RSJ International Conference on Robots and Systems*, San Francisco, CA, USA, pp. 780-785, 2011.

[22] R. Mukherjee, T. Christian, and R. Thiel, "An actuation system for the control of multiple shape memory alloy actuators," *Sensors and Actuators A: Physical*, vol. 55, pp. 367-382 1996.

[23] K. De Laurentis, A. Fisch, J. Nikitczuk, and C. Mavroidis, "Optimal design of shape memory alloy wire bundle actuators," in *Proc. IEEE International Conference on Robotics and Automation*, Washington, DC, USA, pp. 2363-2368, 2002.

[24] C. Lee and C. Mavroidis, "Analytical dynamic model and experimental robust and optimal control of shape-memory-alloy bundle actuators," in *Proc. ASME International Mechanical Engineering Congress and Exposition*, New Orleans, LA, USA, pp. 491-498, 2002.

CHAPTER 3

ANALYSIS OF HYBRID ELECTRIC / THERMOFLUIDIC INPUTS FOR WET SHAPE MEMORY ALLOY ACTUATORS

3.1 Abstract

A wet Shape Memory Alloy (SMA) actuator is characterized by an SMA wire embedded within a compliant fluid-filled tube. Heating and cooling of the SMA wire produces a linear contraction and extension of the wire. Thermal energy can be transferred to and from the wire using combinations of resistive heating and free/forced convection. This paper analyzes the speed and efficiency of a simulated wet SMA actuator using a variety of control strategies involving different combinations of electrical and thermofluidic inputs. A computational fluid dynamics (CFD) model is used in conjunction with a temperature-strain model of the SMA wire to simulate the thermal response of the wire and compute strains, contraction/extension times and efficiency. The simulations produce cycle rates of up to 5 Hz for electrical heating and fluidic cooling, and up to 2 Hz for fluidic heating and cooling. The simulated results demonstrate efficiencies up to 0.5% for electric heating and up to 0.2% for fluidic heating. Using both electric and fluidic inputs concurrently improves the speed and efficiency of the actuator and allows for the actuator to remain contracted without continually delivering energy to the actuator, because of the thermal capacitance of the hot fluid. The characterized speeds and efficiencies are key requirements for implementing broader research efforts involving the intelligent control of electric and thermofluidic networks to optimize the speed and efficiency of wet actuator arrays.

3.2 Introduction

In order for robots and exoskeletons to be able to mimic humans and other creatures, they must have similar degrees of freedom (DOF) allowing them to produce the desired range of motion. These DOF must also be manipulated with a wide range of forces to accomplish specific tasks. Many of the current humanoid robots such as the Honda Asimo [1] and the Sony SDR-X [2] operate with about 30 DOF, which is still an order of magnitude less than that of the human body [3]. Heavy electric motors, gears and batteries make these robots significantly heavier than a human of comparable size and when operating at the fastest rates, these robots have only enough power to operate for about 30 minutes. To emulate human motion and strength/power capabilities, these robots will need to incorporate hundreds or even thousands of high power-to-weight ratio actuators. Possible solutions include Shape Memory Alloy (SMA) wires [4] and electro-active polymers [5], which have been described as artificial muscles because they are able to contract and extend when activated. SMA wires contract and extend as their crystalline structure changes due to temperature. They are capable of strengths 800 times stronger than human muscle (200 MPa) and strain between 4 and 8% [4]. Resistive heating can contract the wire extremely fast, but unforced convective cooling results in slow extension. The cooling rate can be improved by a factor 100 if the SMA actuator is operated in a water (with glycol) bath [6]. However, this adds mass to the actuator and requires additional electrical energy to heat the wire, decreasing the efficiency of the actuator. Heat conduction with Peltier modules has also resulted in improved bandwidths [7],[8], but is also inefficient. Fluidic heating and cooling has produced 1 Hz cycle rate and has a theoretical efficiency of 3% [9].

Biological muscles are supported by a circulatory system that delivers energy, removes waste and regulates temperature. Using this as inspiration, robotic vascular networks [10]-[19] have been implemented to deliver/remove heat from SMA actuators. The thermofluidic heating of these wet SMA actuators has been modeled using lumped

parameters [15],[17]. This model does not account for two major factors. First, the developing flow is not modeled as the fluid is cycled on and off. Second, the lumped parameters do account for the boundary layers that develop in the fluid flow.

This paper will examine the performance of a wet SMA actuator that is operated by electric and fluidic inputs. The unsteady heat transfer process will be modeled using computational fluid dynamics (CFD). The simulated temperatures of the SMA wire are then used as inputs to a thermomechanical model to simulate the mechanical actuation. Electric current (heat generation), flow rates, and fluid temperature will be varied to examine their effects on the contraction/extension times and energy efficiency.

3.3 Background

3.3.1 *Wet SMA Actuators*

SMAs are smart materials that can be implemented as high force-to-weight ratio actuators. The shape memory effect allows these alloys to recover a predefined shape (contraction) after being deformed (strained/extension). This research uses Dynalloy Flexinol® [6], a nickel – titanium (NiTi) alloy, in straight wire form. When the SMA is below its transformation temperature (<70C), it can be strained by 4-5% with external stress of less than 70 kPa. As the SMA is heated above its transformation, it will recover its predefined shape and can exert an external force per cross sectional area of wire of greater than 175kPa. SMAs can be heated very quickly with electricity to produce the contraction, leading to very high power-densities for contraction alone. However, the cooling rate is a function of heat transfer coefficients and may be multiple orders of magnitude slower than the heating process. Thus the power density is typically much smaller when the cooling time is taken into consideration. Another challenge in actuating SMAs is that they are nonlinear. Controlling the exact the strain of the SMA wire is extremely difficult; thus for many applications the controlled state of the wire is treated as binary, either completely contracted (1) or extended (0). With this constraint on the

operation of the actuators, the primary concern is to deliver / remove enough thermal energy to achieve the complete transition from contracted to extend or vice versa.

Water baths have been used to improve the cooling rates of SMAs; however this increases the overall mass of the system and requires more heat to contract the SMA [20]. Another solution to this problem is the wet SMA actuator (Figure 3.1), where a SMA wire is embedded in a compliant fluid-filled vessel [10]. This assembly improves the cooling rates while minimizing the volume of the water. Additionally, hot fluid can be used to contract the actuator instead of or in addition to electricity. While many of the technical challenges in implementing both electric and fluidic controls have been overcome, there has not yet been any thorough characterization of wet SMA actuators when various ranges and combinations of electric and thermofluidic inputs are used.

For the simulations in this paper, the wet SMA actuator is modeled as 254 mm long and is constructed of 0.254 mm diameter Flexinol® wire and 1.6 mm ID x 3.2 mm OD silicone tubing. The actuator is arranged vertically with a 0.45 kg mass hanging from it as a constant bias force (4.14 N) to stretch out the wire during the cooling process. The remaining material properties are presented in Table 3.1.

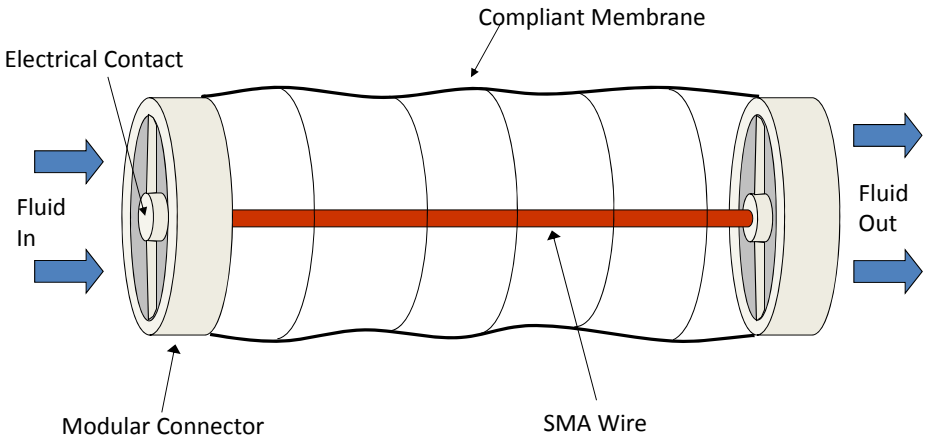


Figure 3.1: Wet SMA actuator – an SMA wire is in embedded in a compliant vessel with connections that allow for electricity and fluid to be used to heat the wire to produce contraction. Cold fluid can also be pumped through the vessel to cool the wire, resulting in extension.

Table 3.1: Wet SMA actuator characteristics

Property	Wire	Fluid	Tube	Air	Units
Outer Diameter	0.254	1.588	3.175		mm
Inner Diameter		0.254	1.588	3.175	mm
Length	254	254	254		mm
Density	6450	988	1400	1.225	kg/m^3
Specific Heat	837	4182	692	1006	J/kg-K
Thermal Conductivity	17.99	0.6	1.26	0.0242	J

Flexinol® wires properties from [6], Silicone tube properties from Matweb.com [21], fluid (water) and air default properties from Fluent 6.3® [22].

3.4 Wet SMA Actuator Modeling

3.4.1 *Computational Fluid Dynamic (CFD) Modeling*

The heating/cooling simulations of the wet SMA actuator have been done in Fluent®6.3 [22], a computational fluid dynamics (CFD) package by ANSYS. The ideal wet SMA actuator is a concentric annulus that is axi-symmetric along the center of wire. Therefore a two-dimensional axi-symmetric mesh was used, as shown in Figure 3.2, which greatly reduces the number of elements in the CFD model. The model was axially (longitudinally) divided into 1000 equal segments of 0.254 mm each. The radial direction was divided into 30 segments, with a finer mesh along the surfaces of the materials to

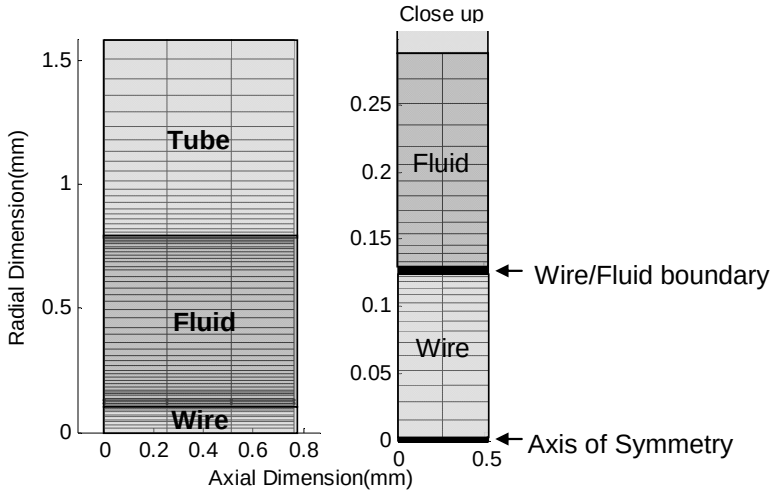


Figure 3.2: A section of the mesh of wet SMA actuator used in Fluent 6.3©. The axial (longitudinal) dimension is 254 mm and was divided into 1000 segments. The radial dimensions, wire, fluid and tube, were divided into 30 segments, with a finer mesh along the boundary between materials.

allow for detailed information about the thermal and fluidic boundary layers. With the lowest fluid flow rate of 2 mL/s, the Reynolds (Re) number is 1780, while all other larger flow rates will have a greater Re value. Therefore, in Fluent the k-epsilon viscous modeling option was turned on to model turbulent flow (Re > 2000). The energy modeling and unsteady (transient) options were also turned on to simulate the changes in temperature in the model. The resistive heating was modeled with uniform heat generation in the volume of the wire. The fluid was modeled with a constant pressure source and uniform velocity at the inlet of the tube. The fluid flow in the concentric annulus model develops a parabolic velocity profile 10 mm into the tubing and continues along the length of the tubing. The surface between external surface of the tube and air was model was held constant at 25 ºC. All other Fluent 6.3 parameters were unchanged from their default values. After each iteration of the CFD model, the temperature of each element is exported for use in the temperature–strain model.

3.4.2 *Temperature–Strain Model of SMAs*

Using the temperature-strain relationship developed by [15],[17], which is shown in equations 1 and 2, the strain of individual elements of SMA wire from the mesh can be calculated. Equation 3.1 calculates the martensite fraction (R_m) of the SMA based on the temperature of the element (T), the average transformation temperature (T_{avg}) and the standard deviation of T_{avg} (σ_T). This equation applies to both the heating and cooling, which each have different transformation parameters (shown in Table 3.2), accounting for the hysteresis of the transformation effect that occurs between heating and cooling. Equation 3.2 takes the martensite fraction and calculates the strain of each element of the wire. The components of equation 2 account for the nonlinearity of the relationship between martensite fraction, stress, and strain.

$$R_m = \frac{1}{2}\left[1 - \mathrm{erf}\left(\frac{T - T_{avg}}{\sigma_T \sqrt{2}}\right)\right] \tag{3.1}$$

$$\varepsilon = \begin{cases} \dfrac{\sigma}{E_a + R_m(E_m - E_a)} & \varepsilon \le \varepsilon_m^y \\ \dfrac{\sigma - Rm(e_m^y(E_m - E_t)}{E_a + R_m(E_t - E_a)} & \varepsilon_m^y < \varepsilon \le \varepsilon_m^d \\ \dfrac{\sigma - Rm(e_m^y(E_m - E_t) + e_m^d(E_t - E_d))}{E_a + R_m(E_d - E_a)} & \varepsilon > \varepsilon_m^d \end{cases} \tag{3.2}$$

Figure 3.3 plots the strain vs. temperature relationship using equations (1) and (2) and the parameters in Table 3.2, which were empirically experimentally determined in [15],[17]. During the heating phase, the wire begins to contract at 50 ºC and fully contracts at 90 ºC. As the SMA is then cooled, it begins to extend at 60 ºC and fully extends at 38 ºC.

3.5 Simulations

For the simulations in this paper, the 254 mm long actuator will be attached to a simulated mass of 0.45 kg (4.41 N), whose weight will act as the bias force to extend the wire. In Figure 3.3, the strain of the SMA wire is simulated as the temperature is uniformly increased from 25 to 90°C and cooled back down to 25 °C. At 90°C, the

Table 3.2: Austenite and Martensite characteristics
for temperature-martensite fraction equation

Austenite Phase			**Units**
Modulus of Elasticity	E_a	37.5	GPa
Mean Transformation Temperature	$T_{a\text{-}avg}$	78	ºC
Stdev. of T_a	σ_a	9	ºC
Martensite Phase			
Modulus of Elasticity	E_m	12.2	GPa
Mean Transformation Temperature	$T_{m\text{-}avg}$	55	ºC
Stdev of T_m	σ_m	6	ºC
Yeild Strain – Fully Twinned	ε^d_m	0.0417	
Minimum Strain – Detwinned	ε^y_m	0.0040	

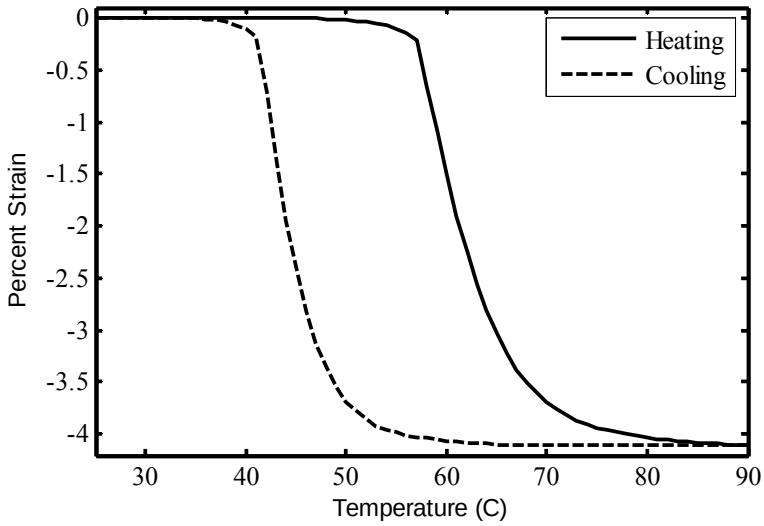


Figure 3.3: Temperature vs. strain of a uniformly-heated SMA wire. The temperature-strain model is able to account for the hysteresis of the SMA wire and the bias force used to extend the wire during cooling.

simulation shows that the increase in temperature results in a 4.1% strain (~10 mm). For the rest of this paper, full contraction will be defined as the point at which 95% of the strain has occurred. For all case studies, the wet SMA actuator is filled with water and all components have an initial temperature of 25°C.

3.5.1 *Case Study 1 – Electric Heating, Varied Power Input*

This case study varies the electrical power applied to the SMA wire to generate heat within the wire. The resistance for the 0.254 mm diameter SMA wire is 18.5 Ω/m and for the 254 mm long actuator, the total resistance is 4.4 Ω. In Fluent, the resistive heating was modeled as uniform heat generation, which for the following simulations ranged from 4 to 10 GW/m^3, corresponding to a range of 3.4 to 5.4 Amps of current. The heating process is shown in Figure 3.4 and contraction times range from 0.06 to 0.28 seconds. For this case study, the electricity is kept on until the displacement comes to a steady state as defined by 95% contraction. In reality, if the electricity remains on, the wire continues to be heated and if the temperature becomes too high, the wire may be damaged and stop demonstrating the shape memory effect. Figure 3.5 shows the average

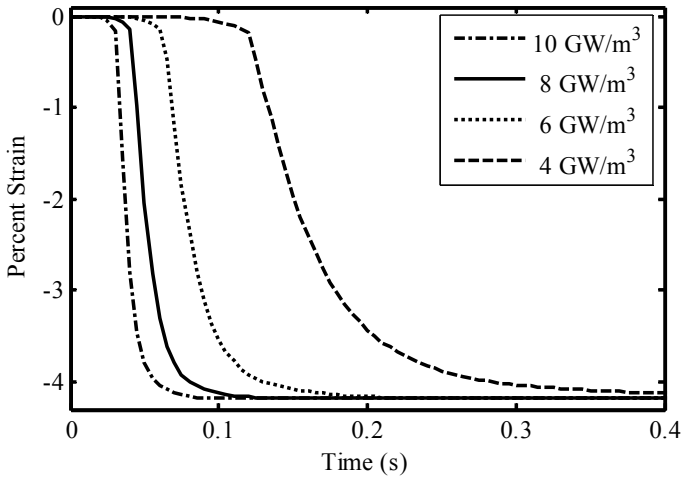


Figure 3.4: Strain vs. time for varying electric heat generation. Electric current (3.4 to 5.4 Amps) is used to resistively heat (4 to 10 GW/m^3) the wire and produce a strain of 4.1%. This strain corresponds to approximately 90 °C, and if the electric current remains on, the wire will continue to heat without additional strain. Overheating the wire may damage the crystalline structure and shape memory effect.

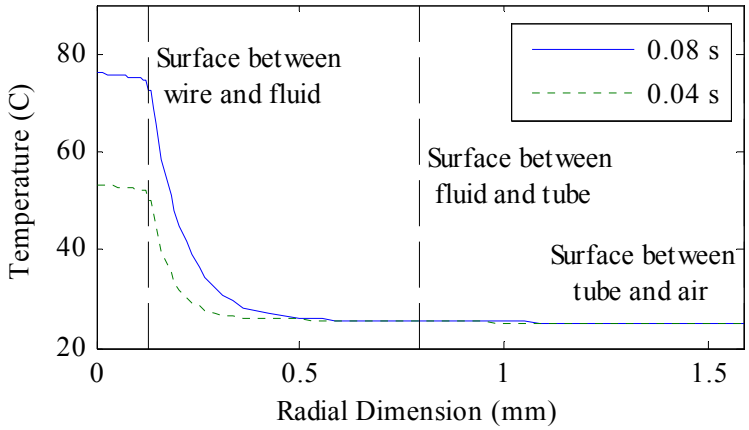


Figure 3.5. Average Temperature profile, at the middle of the wire, in the radial direction, for electric heating (5 Amps, zero fluid flow). The boundary layer demonstrates that a lumped parameter model is not sufficient to model the wet SMA actuator.

temperature profile of the wet SMA actuator in the radial direction at times of 0.04 and 0.08 seconds. Due to the rate of the heating, a thermal boundary layer is formed in the fluid and it is evident that a lumped parameter model (i.e. a model where the cross section is treated as having uniform temperature) will not accurately model a wet SMA actuator that is heated electrically. In Figure 3.6, both the heating and cooling processes are

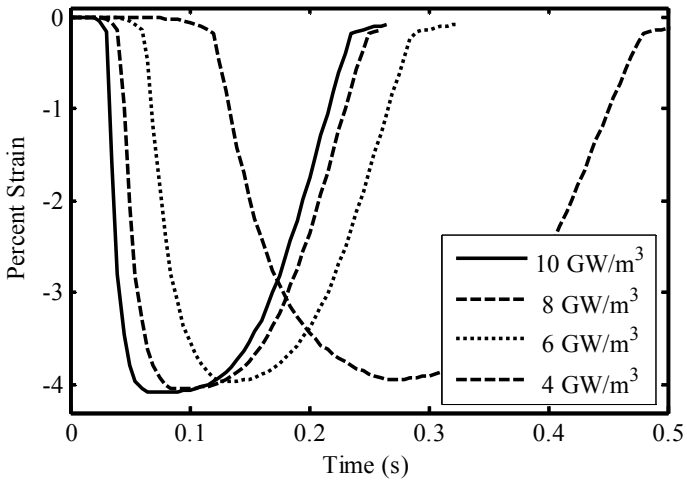


Figure 3.6: Strain vs. time for electric heat generation and passive convective cooling (zero fluid flow). For these simulations, the SMA wire was heated to 4% strain, and then allowed to cool back down and extend back to its original length of 254 mm with a bias force of 4.4 N.

shown. For this simulation, there is zero fluid flow (25 °C) during the heating and cooling processes, and the electricity is simply turned on for heating and off for cooling. The cooling process is slower than the heating and will depend on the temperature of the fluid. This method of heating and cooling cannot be sustained, because the fluid would continue to heat and no longer be able to remove thermal energy from the wire. In Figure 3.7, the electrical energy input and the change in internal energy of the SMA wire are shown. The difference between these two is the amount energy that is transferred to the fluid. Because higher electrical power inputs result in faster heating, less energy is actually transferred to the fluid during the contraction. Therefore the fluid stays cooler and results in a faster extension as well.

3.5.2 *Case Study 2 – Electric Heating, Varied Fluid Flow*

The electric power input for this case is held constant at 10 GW/m^3 (5.4 amps) and the fluid flow is varied between zero flow, intermittent flow and continuous flow (5 mL/s). The intermittent fluid flow is defined as off during the heating process and then turned on (5 mL/s) for the cooling process. Figure 3.8 shows the simulated results, with the cooling

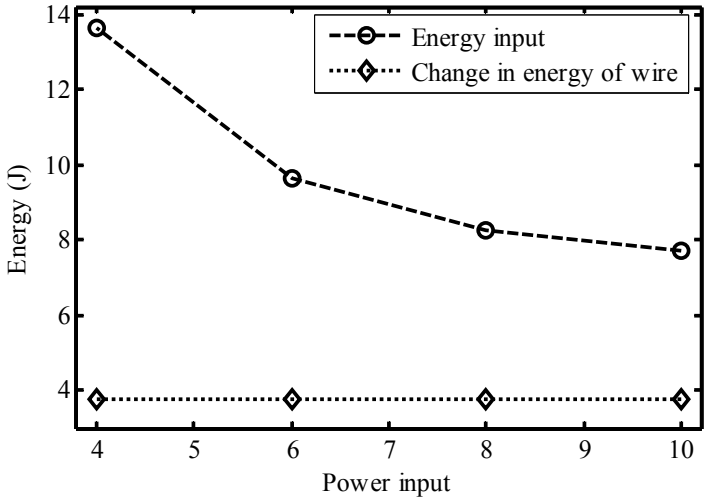


Figure 3.7: Energy vs. power input for electric heating and zero fluid flow. During the heating of the wire to produce contraction, the internal energy of the wire increases by 3.8 J. The wire is also convectively heating the water surrounding it. When the wire is heated at a slower rate, with less power input, more energy is lost to the fluid, making the actuator less efficient.

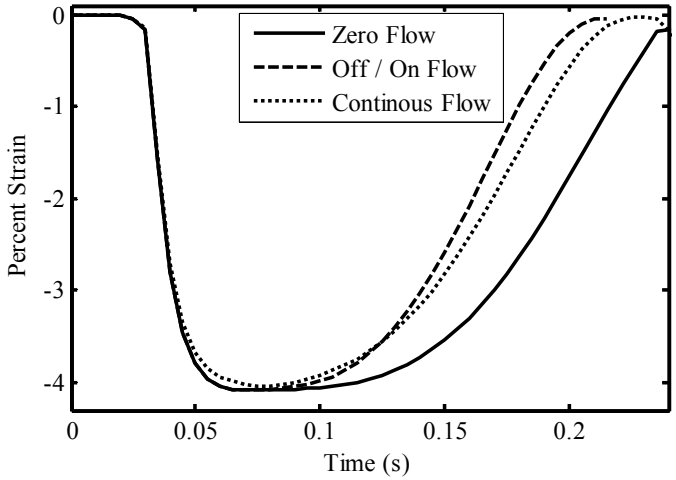


Figure 3.8: Strain vs. time for electric heating (10 GW/m^3) with varied cold fluid (25 °C) flow. The complete actuation cycle was modeled for three cases. Case 1 is zero fluid flow during the entire cycle, Case 2 is zero fluid flow during the heating process and 5 mL/s flow during the cooling process, and Case 3 is 5 mL/s flow during the entire cycle.

times ranging from about 0.15s to 0.20s. These are ~30 times faster than the published cooling rate of 5.5s in air [6]. As stated above, the zero flow case is not a viable long term operating process because the heat lost by the wire during cooling is absorbed by the fluid, increasing the fluid temperature. Therefore the fluid will eventually need to be replaced. The continuous flow requires more time/energy to produce the contraction, but appears to be a viable solution in cases where the fluid flow cannot be controlled or turned on and off at the desired rate.

3.5.3 *Case Study 3 – Variation of Fluid Flow Rates*

Case 2 has shown that the wet SMA actuator has improved cooling rates compared to dry SMAs operated in air, simply by having cold liquid in the compliant tubing. The compliant tubing also allows for hot fluid to be used to deliver thermal energy to the actuator. For this case study, the pressure source is varied from 25kPa to 6.25kPa and is delivering 90 ºC fluid to the actuator. The simulated fluid flow rates for these pressures vary from 5 mL/s down to 2 mL/s. Unlike the electrical heating process, where there is uniform heating along the length of the wire, the hot fluid must propagate through the tube before heating occurs. The approximate time for the fluid to propagate through the entire length of the actuator varies from 0.1 to 0.25 seconds. This propagation time has been defined as the time necessary to replace the volume of the fluid (volume of fluid/ volumetric flow rate). Because the wet SMA actuator is a concentric annulus, the fluid flow has a parabolic shape with zero velocity along the walls. This fluidic boundary layer induces a thermal boundary layer (Figure 3.9) and results in slower heat transfer. Figure 3.10 shows the temperature profile along the longitudinal dimension of the actuator. The temperature along the fluid-wire edge is approximately 7 ºC lower than the centerline temperature. This is the primary reason a lumped parameter model in [15],[17] is invalid. The lumped parameter model segmented the wire, fluid and tube longitudinally, but in the radial direction each component was lumped together in a single cross section. These

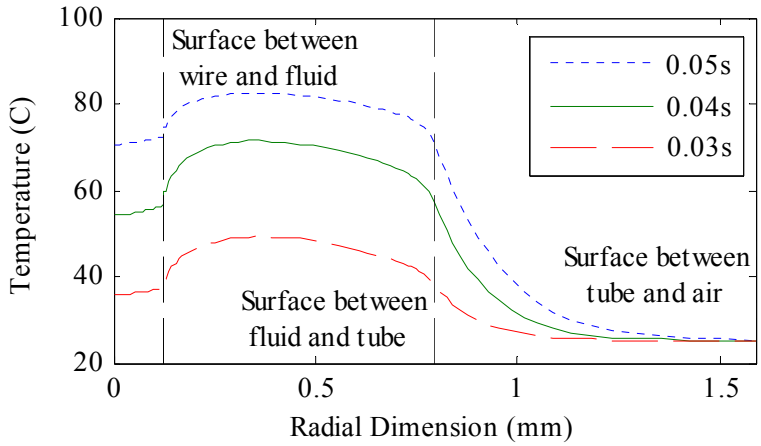


Figure 3.9: Temperature profile in the radial direction for fluidic heating (90 ºC at 5 mL/s) at 30 mm from the inlet of the actuator. The fluidic and thermal boundary layers that develop in the wet SMA actuator (concentric annulus) limit the heat transfer process between the fluid and the wire.

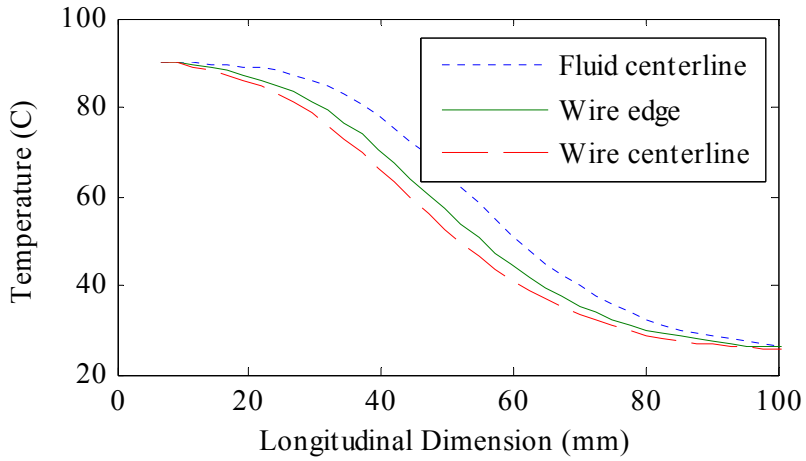


Figure 3.10: Temperature profile in the longitudinal direction for fluidic heating (90 ºC, ~5 mL/s) for the actuator. A 7 degree Celsius temperature differential is observed between the fluid centerline and the wire edge.

lumped volumes of the actuator had a uniform temperature and therefore the model was not able to correctly simulate the heat transfer process along surfaces. The strain vs. time plots for the various flow rates are shown in Figure 3.11. As expected, lower flow rates result in longer contraction times. In general, the contraction times for the 90 ºC fluid are approximately twice the time it takes for the fluid to propagate through the actuator. Figure 3.11 shows the complete heating and cooling cycle producing cycle times from 0.4 to 0.8 seconds.

3.5.4 *Case Study 4 – Variation of Fluid Temperature*

For this case study, the temperature of the fluid is varied from 75 ºC to 90 ºC to the temperature effects on the contraction times and strain. The results from this simulation are shown in Figure 3.12. As expected, the contraction times increase as the temperature of the fluid decreases, however, the strains only decrease by 0.2%. It is not until the inlet fluid temperature is less than 75 ºC that the strain is significantly reduced.

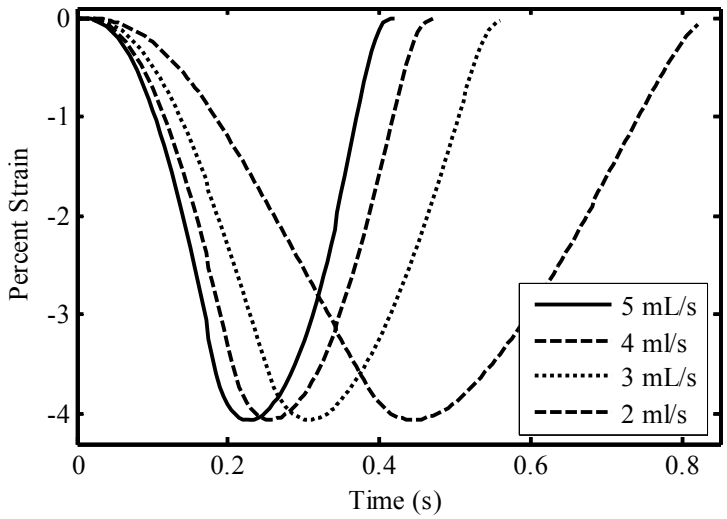


Figure 3.11: Strain vs. time for fluidic heating (90 ºC) and cooling (25 ºC). Hot fluid is delivered to the wet SMA actuator using a constant pressure source, resulting in flow rates between 2-5 mL/s. When the actuator is fully contracted, the source is switched over to cold fluid.

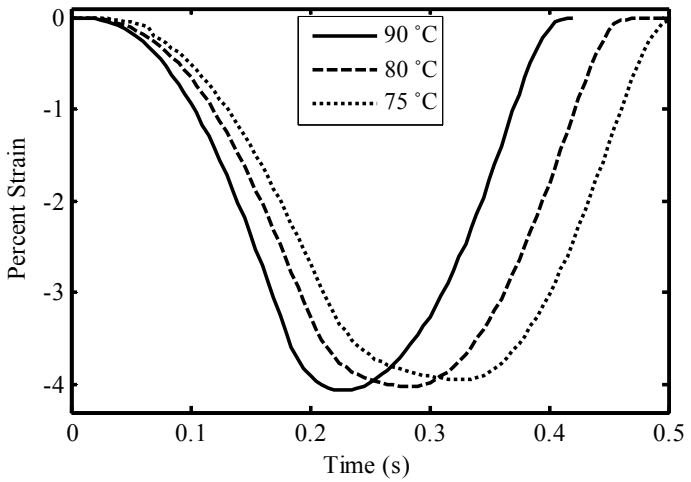


Figure 3.12: Strain vs. time for varied fluid temperature (5 mL/s).). Hot fluid is delivered to the wet SMA actuator using a constant pressure source of 25 kPa, resulting in a flow rate of 5 mL/s. When the actuator is fully contracted, 95% of steady state, the source is switched over to cold fluid. The 90 °C continues to heat the wire even after the cold fluid begins to flow due to the fluid propagation time and the hysteresis of the SMA wire.

3.5.5 *Case Study 5 – Heating Combinations – Zero Flow Cooling.*

In previous case studies, the wet SMA actuator was heated either electrically or fluidically to achieve the contraction. However, both electricity and hot fluid can beapplied simultaneously. Figure 3.13 shows three possible scenarios for actuation: 1.electric heating only (10 GW/m^3) with zero fluid flow, 2. fluidic heating only (90 °C at 5 mL/s) followed by zero flow, and 3. simultaneous electric and fluidic heating (4 GW/m^3, 90 °C at 5 mL/s) followed by zero flow. For the simultaneous operation, the electric power input was reduced so that total contraction time is equal to the fluid propagation time. Once the contraction is complete, all of the inputs to the actuator are turned off and the system is allowed to return to a steady state. The electrically heated actuator relaxes to its initial length as heat is transferred to the fluid, while the fluidically heated actuator remains contracted. The simultaneous fluid/electric heated actuator has a faster contraction time than the fluid only actuator, and remains contracted. Although heat transfer continues to occur between the water and wire, the temperature of the wire does not drop below T_m (martensite transformation temperature) in the time shown, and

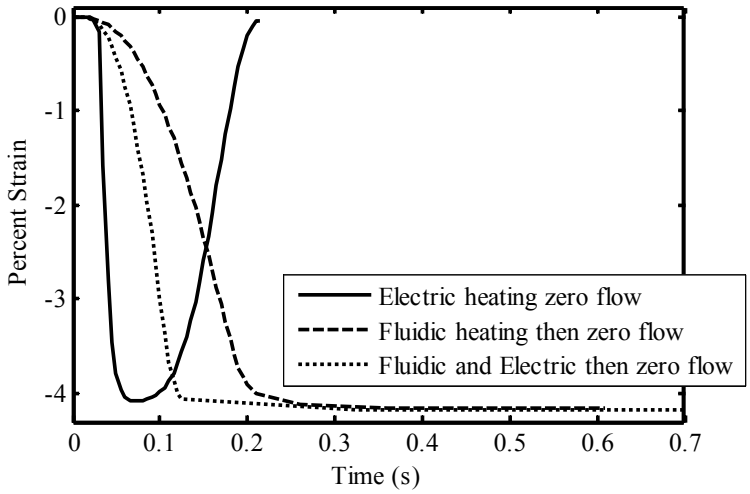
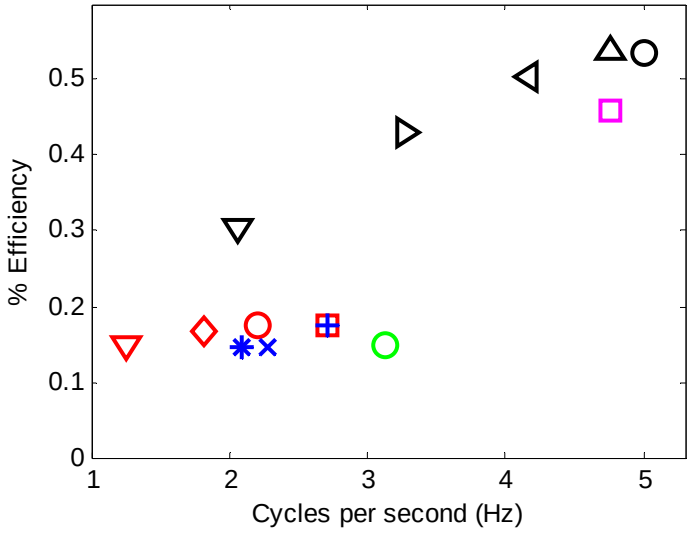


Figure 3.13: Strain vs. time for varied heating methods followed by zero flow. Without continuous energy input, the electrically driven actuator will cool and return to it extended state. However, the fluidically driven actuator will remain contracted because of the thermal capacitance of the hot fluid.

the wire remains contracted. Experimental results show that actuator with hot fluid will remain contracted for 20 seconds without external inputs.

3.5.6 Efficiency vs. Speed Analysis of Heating and Cooling Processes.

For each of the processes examined above, there are thermal energy losses. In the case of the electric heating, some of the energy goes to heating the cold fluid. In the case of fluidic heating, much of the energy in the hot fluid can be recovered by recycling the hot fluid. However some of the energy in the hot fluid goes into heating the wire and tube. During the cooling process all of the thermal energy is transferred back to the cold fluid or the environment and is unrecoverable. In Figure 3.14, the efficiency vs. cycle rates are shown for the various processes. The efficiency of the actuator is defined as the mechanical work out divided by the thermal energy input for the complete contraction-extension cycle. The electrically driven actuator has up to three times higher efficiencies than that of the fluidically driven actuator, which is a somewhat surprising result The thermofluidic heating has a lower efficiency because more thermal energy is transferred to the tube and environment during the fluidic heating process as compared to the energy



Legend

P_e= 10 GW/m^3, Q= 0	Case 1
P_e= 8 GW/m^3, Q= 0	
P_e= 6 GW/m^3, Q= 0	
P_e= 4 GW/m^3, Q= 0	
Q= 5mL/s @ T_f= 90C	Case 3
Q= 4mL/s @ T_f= 90C	
Q= 3mL/s @ T_f= 90C	
Q =2mL/s @ T_f= 90C	
Q= 5mL/s @ T_f= 90C	Case 4
Q= 5mL/s @ T_f= 80C	
Q= 5 mL/s @ T_f= 75C	
Q= 5mL/s @ T_f= 90C and P_e= 4e9 GW/m^3	Case 5
Q= 5mL/s @ T_f=25C and P_e= 10e9 GW/m^3	
P_e= 10e9 GW/m^3, Q= 0 then Q= 5mL/s	

Figure 3.14: Efficiency vs. cycle rate (legend shown in Table 4.3). (P_e) is the electric power delivered during Joule heating of the SMA actuator. Q and T_f are the fluid flow rate and fluid temperature respectively. The electrically-heated actuation is up to 2 times as fast and 2.5 times as efficient as the fluidically-heated actuation.

to the fluid during the longer contraction time. Therefore, the faster the energy is lost to the fluid during electrical heating process. This is primarily because of the increased contraction time during fluidic heating. The efficiency of the electrically driven actuator decreases with smaller power inputs, because more thermal energy is transferred delivered to the SMA wire, the more efficient the actuator is.

With the electric heating and varied flow (case study 2), the intermittent flow method has improved speed over the zero flow method because of the improved convection during the cooling process. This is more energy efficient than the continuous flow method because it has a lower convection rate during the heating process.

3.6 Conclusions

This paper has presented the simulated dynamic performance of a wet SMA actuator in response to various electrical and thermofluidic control inputs. The various heating processes were able to increase the temperature of the wire to produce ~4% contraction in the simulated 254 mm long actuator. Electrical heating of 10 GW/m^3 followed by fluidic cooling (25 ºC, 5 mL/s) was characterized by a maximum efficiency of 0.5% and maximum cycle rate of 5 Hz. Fluidic heating (90 ºC, 5 mL/s) followed by fluidic cooling was slower and less efficient, characterized by a maximum efficiency of 0.2% and a maximum cycle rate of 2 Hz. While the electric heating was expected to be faster than fluidic heating, it was somewhat surprising that it was also more efficient. While the electric heating does waste energy by heating the surrounding fluid, it turns out that fluidic heating is less efficient due to the heat lost to the surrounding tube/environment. Although the efficiency for the fluidic heating is about 40% of the efficiency for electrical heating, a chemical energy source (e.g., propane) used to heat the fluid would have 100 times the energy density of electric batteries, which would still give a clear advantage to fluidic heating in terms of the mass of energy storage required to operate the system for a fixed number of actuations.

During simulations, both fluidic and thermal boundary layers have been observed in the CFD model. These boundary layers limit the thermal interaction of the fluid with the wire and the tube. Future actuator designs may address these boundary layers that limit the performance of the wet SMA actuator by careful consideration of factors such as wire and tube dimensions, surface roughness, and fluid viscosity. Additional work will need to examine if a nondimensionalized model can be created and optimized.

These wet SMA actuators have been bundled together in arrays to create large degree-of-freedom actuation systems. Actuators in the array can be connected in series and parallel to produce larger displacements and forces [11]-[14],[18],[23]-[25]. The array implementing wet SMA actuators is controlled by a scalable network array architecture that shares $2N$ control devices for N^2 actuators. This architecture minimizes the total weight of the array in order to maintain the high force-to-weight ratio advantage of the actuators; however it does place the constraint on the array that only certain combinations of actuators in the array can be simultaneously addressed. For this reason, a series of sequential operations may be required to produce the desire actuation for the entire array. Given the constraints of the network control architecture and the speed/efficiency characterization presented in this paper, an optimal control algorithm has been developed that can maximize the performance of the array using a cost function that represents a weighted combination of actuation time and energy. The algorithm uses graph theory (.i.e. path planning) to identify an optimal sequence of control commands to produce the desired actuation with minimized time and energy [18],[19].

3.7 References

[1] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, “The development of Honda humanoid robot,” in *Proc. IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1321-1326, 1998.

[2] T. Ishida, Y. Kuroki, and J. Yamaguchi, “Mechanical system of a small biped entertainment robot,” in Proc. IEEE/RSJ Intl. Conference on Intelligent Robots and Systems, vol. 2, pp. 1129-1134, 2003.

[3] L. Sherwood, Human physiology 7th ed., Belmont, CA, Cengage Learning, 2008.

[4] I. Hunter and S. Lafontaine, “A comparison of muscle with artificial actuators,” in *Proc. IEEE Solid-State Sensor and Actuator Workshop*, pp. 178-185, 1992.

[5] A Ramaratnam and N. Jalili, “Feasibility study of actuators and sensors using electro-active polymers reinforced with carbon nanotubes,” in *Proc. SPIE Conference on Smart Structures and Materials*, pp. 349-356, 2004.

[6] “Technical Characteristics of Flexinol Actuator Wires,” Dynalloy Inc., Available: http://www.dynalloy.com/pdfs/TCF1140RevD.pdf

[7] A. Shahin, P. Meckl, J. Jones and M. Thrasher ”Enhanced cooling of shape memory alloy wires using semiconductor 'heat pump' modules,” *Journal of Intelligent Material Systems and Structures*, vol. 5, pp. 95-104, 1994.

[8] B. Selden,, K. Cho, and H. Asada, “Segmented binary control of shape memory ally actuator systems using the Peltier Effect,” in *Proc. IEEE International Conference on Robotics and Automation*, pp. 4931-4936, 2004.

[9] O. Rediniotis, D. Lagoudas-Dimitris, H. Jun, and R. Allen, “Fuel-powered compact SMA actuator,” in *Proc. SPIE Conference on Smart Structures and Materials*, vol. 4698, pp. 441-453, 2002.

[10] S. Mascaro and H. Asada, “Wet shape memory alloy actuators for active vasculated robotic flesh,” in *Proc. IEEE International Conference on Robotics and Automation*, vol. 1, pp. 282-287, 2003.

[11] S. Mascaro and H. Asada, "Vast DOF wet shape memory alloy actuators using matrix manifold and valve system," in *Proc. ASME International Mechanical Engineering Congress*, Washington, DC, USA, pp. 1992-1997, 2003.

[12] S. Mascaro, K. Cho and H. Asada, “Design and control of vast DOF wet SMA array actuators,” in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, pp. 1992-1997, 2003.

[13] L. Flemming and S. Mascaro, “Wet SMA actuator array with matrix vasoconstriction device,” in *Proc. ASME International Mechanical Engineering Congress and Exposition*, pp. 1751-1758, 2005.

[14] L. Flemming and S. Mascaro, "Control of a scalable matrix vasoconstrictor device for wet actuator arrays," in *Proc. IEEE International Conference on Robotics and Automation, Rome, Italy*, pp. 648–653, 2007.

[15] J. Ertel and S. Mascaro, “Thermomechanical modeling of a wet shape memory alloy actuator,” in *Proc. ASME International Mechanical Engineering Congress and Exposition*, Chicago, IL, pp. 1317-1324, 2006.

[16] L. Flemming and S. Mascaro, "Analysis of hybrid electric/thermofluidic control for wet shape memory alloy actuators," in *Proc. ASME Dynamic Systems and Controls*, Hollywood, CA, USA, pp. 1041-1047, 2009.

[17] J. Ertel and S. Mascaro, “Dynamic thermomechanical modeling of a wet shape memory alloy actuator,” *ASME Journal of Dynamic Systems, Measurements and Control*, pp.1-9, 2010.

[18] L. Flemming, D. Johnson, and S. Mascaro, “Optimal control of multi-input SMA actuator arrays using graph theory,” in *Proc. IEEE International Conference on Robotics and Automation*, Shanghai, China, pp. 6109–6114, 2011.

[19] L. Flemming, D. Johnson, and S. Mascaro, “Optimal control of multi-input SMA actuator arrays using graph theory: expanding wavefront & simultaneous operations,” in *Proc. IEEE International Conference on Robots and Systems, San Francisco*, CA, USA, pp. 780-785, 2011.

[20] M Nakatani, H Kajimoto, K. Vlack, D. Sekiguchi, N Kawakami, and S Tachi ”Pop up!: 3D form display with coil-type shape memory alloy,” Journal of the Institute of Image Information and Television Engineers, vol. 60, pp. 183-191, 2006.

[21] “Matweb silicone rubber data sheet”, Matweb.com, Available: http://www.matweb.com/search/DataSheet.aspx?MatGUID=cbe7a469897a47eda563816c86a73520&ckck=1

[22] Ansys Fluent 6.5 Installation guide, Ansys Inc., Available: http://www1.ansys.com/customer/content/documentation/120/ai_instl.pdf

[23] R. Mukherjee, T. Christian, and R. Thiel, "An actuation system for the control of multiple shape memory alloy actuators," Sensors and Actuators A: Physical, vol. 55, pp. 367-382 1996.

[24] K. De Laurentis, A. Fisch, J. Nikitczuk, and C. Mavroidis, "Optimal design of shape memory alloy wire bundle actuators," in *Proc. IEEE International Conference on Robotics and Automation*, Washington, DC, USA, pp. 2363-2368, 2002.

[25] C. Lee and C. Mavroidis, "Analytical dynamic model and experimental robust and optimal control of shape-memory-alloy bundle actuators," in *Proc. ASME International Mechanical Engineering Congress and Exposition*, New Orleans, LA, USA, pp. 491-498, 2002.

CHAPTER 4

OPTIMAL CONTROL OF WET SHAPE MEMORY ALLOY ACTUATOR ARRAYS USING GRAPH THEORY

4.1 Abstract

Shape Memory Alloy (SMA) actuators are compact and have high force-to-weight ratios, making them strong candidates to actuate robots, exoskeletons, and prosthetics. However, the dynamics of these actuators are thermomechanical in nature and slow cooling rates can limit their performance. To improve the convective cooling, SMA wires have been embedded in vascular networks, allowing cold fluid to pass across the actuators and extend them faster. The vascular network can also deliver hot fluid to heat and contract the wire. In addition to the fluidic network, an electrical network operating in parallel can resistively heat the SMA to produce contraction. To minimize the weight and size of the vascular and electrical networks, a scalable NxN architecture has been implemented that allows for 2N control devices (valves, transistors) to be shared amongst N^2 actuators. This Network Array Architecture (NAA) allows each actuator to be controlled individually or in discrete subarrays. However, this architecture does not allow all combinations of actuators to be activated simultaneously; therefore a sequence of control commands may be required to achieve the complete desired actuation.

In order to find an optimal sequence of control commands, graph theory algorithms have been implemented. By treating each actuator's state as binary (fully contracted or extended), the collected states of the actuator array can be represented as nodes of the graph and the control commands as the graph edges. By properly weighting the time and energy costs of the graph edges, search algorithms can be used to find an optimal set of

control commands for desired state changes. NAA results in a multigraph that has $2^{N \times N}$ nodes and is highly interconnected. This article formulates the control of NAA actuators systems as a scalable graph theory problem, and characterizes the ability of search algorithms to optimize a weighted combination of speed and energy usage, while minimizing computational search cost. The algorithm was implemented in MATLAB and it is able to identify the optimal solution for a 4x4 array with more than 14 million edges. By using an expanding wavefront, the algorithm, on average, explored less than 100 edges (<0.01%) in 0.03 seconds. A 6x6 array was optimized in 0.7 seconds, exploring just 2400 edges.

4.2 Introduction

A primary challenge to maximize the wearability of robotics [1],[2], prosthetics[3],[4] and exoskeletons [5], is the need to actuate many degrees of freedom (DOF) with minimal bulk. Wearability is enhanced when robotic actuation closely matches the characteristics of human actuation according to metrics such as strength, speed, range of motion, power density, degrees-of-freedom, and use of a long-lasting untethered energy source.

A prime example would be the human hand, which has 21 degrees-of-freedom in the fingers and thumb alone [6]. It is extremely difficult to design a robotic/prosthetic hand to mimic the range of motion of a human hand. Another engineering challenge is to actuate the mechanical hand with similar power density of skeletal muscles (50 to 100 W/kg) [7]. Although DC motors can produce a wide range of motion and torques, their typical power density is approximately 10 W/kg. Therefore, in order for a mechanical hand to match the capabilities of its biological counterpart, it would need to be 5 to 10 times heavier using DC motors. Pneumatic and hydraulic actuators can match or exceed the power density of human muscles, but they involve the hazards of high operating pressures and require a tethered source of pressurized fluid or a heavy/noisy compressor.

Piezoelectric actuators also have a power density comparable to human muscle [8], but they require potentially hazardous operating voltages and create only microscale displacements, which would be impractical to leverage into useful motion for a prosthetic or exoskeleton.

A novel solution to advancing wearable robotics is to implement large networks of high force-to-weight wet Shape Memory Alloys (SMA) "muscle" actuators. The principal dynamics of a SMA actuator are thermomechanical, and a biologically inspired vascular network has been developed to deliver/remove thermal energy by thermofluidic and electric methods [9]. The system dynamics of wet SMA actuators are highly nonlinear and therefore difficult to control with classical control methods [10]-[19]. However, by using a discrete control logic approach, the nonlinear dynamics can be neglected and the actuators can be treated as binary (fully contracted/extended). Also, by networking large numbers of these binary actuators mechanically in series or parallel, a more realistic means of actuation can be achieved [20]-[30].

The weight of transistors and valves necessary to control the electric and fluid flow to the wet SMA actuators can be multiple orders larger than the weight of the SMA itself. Therefore, a scalable control architecture that has $2N$ control devices controlling the flow to an array of N^2 actuators has been implemented [25]. This reduction in control hardware allows actuators to be extended or contracted individually or in certain combinations simultaneously. However, it may take multiple steps of discrete control commands to produce the complete desired actuation for the entire array [28].

Graph theory has been shown to be an effective way of optimally controlling the discrete nature of an array of actuators and control hardware [16]. However, the complete graph scales poorly because it has a large number of states and is highly interconnected [29],[30]. Previously, a 4x4 array was the largest array that the graph theory algorithms could be tested on.

The objective of this article is to present optimal control algorithms for controlling arrays of wet actuators by applying graph theory. Additional algorithms will be introduced that address the scalability of the NAA graph, allowing for larger arrays of actuators to be controlled. In Section 4.2, the characteristics of wet SMA actuators arrays will be presented. In Section 4.3, the control of NAA will be formulated as a graph theory problem, which will be used to search for sequences of control commands that optimize the actuation time and energy usage. To improve the total actuation speed, the algorithms will also include the operation of the fluidic and electric network in parallel. In Section 4.4, the issue of scalability of the NAA graph will be addressed by implementing an expanding wavefront which uses lazy evaluation to construct only the portions of the graph that are being searched. Finally, in Section 4.5, the performance of the new graph theory algorithms will be examined through simulation.

4.3 Background

4.3.1 *Wet SMA Actuator*

SMAs are a class of smart materials that are able to return to a predefined shape after being deformed [31]. This deformation and recovery of strain is achieved by a thermomechanical process, where the SMA is strained while it is cold (e.g., <~70 ºC) and then restored by heating it above its transformation temperature (e.g., ~70 ºC). These characteristics allow SMA to be implemented as compact, high force-to-weight ratio actuators. Because the thermomechanical process is nonlinear, they are difficult to model and control. This research will treat the actuators as binary (fully contracted (1) or extended (0)) and the control strategy will be concerned only with delivering/removing the energy necessary to transition between these two states. In order to achieve higher resolution displacements/forces, the actuators can be bundled together and operated in series/parallel respectively.

Joule heating of SMA wire can produce very quick contractions (milliseconds);

however the extension process using unforced air convection can take multiple orders of magnitude longer [31]. Forced convection and water baths have been used to improve the cooling process [32], but these add a significant amount of size and mass to the actuator. To maintain a high force-to-weight ratio for the actuators, SMA wire has been embedded in a compliant vessel [9], allowing a small amount of fluid to flow over the wire. Cold fluid is used to improve the convection cooling, while hot fluid can convectively heat the wire. Figure 4.1 shows a robotic hand that is actuated by these wet SMA actuators operating in series to produce discrete displacements of the fingers.

4.3.2 *Network Array Architecture (NAA)*

While the wet SMA actuator assembly does resolve the issue of cooling rate; the control devices (valves, transistors) are still many times heavier than the wet SMA actuator itself. Arranging the actuators in a Network Array Architecture (NAA) allows for an *N*x*N* array of actuators to be controlled by 2*N* control devices, reducing the total weight of the system. Figure 4.2A shows a schematic of the electrical network. The actuators in a common row are connected to a constant voltage source by a switch. The actuators in a common column are also connected together, where each column can be

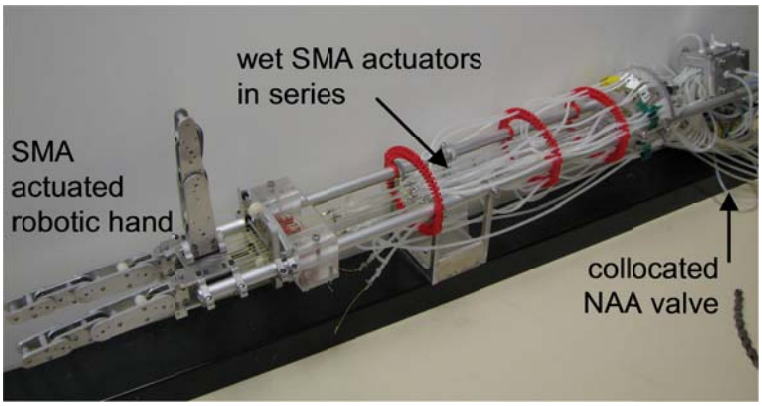


Figure 4.1: Robotic hand actuated by wet SMA actuators connected in series. The wet SMA actuators have an SMA wire embedded in a compliant tube with connectors allowing electricity and fluid to deliver/remove thermal energy.

connected to the sink (ground) by another switch. In order to send energy to a specific actuator, the intersecting row and column switches must be closed to complete the circuit. The diodes in the circuit ensure that the current cannot take an undesired route to ground. Each actuator can be activated individually, or a subarray of actuators can be activated simultaneously by closing multiple rows and/or columns. However, actuators on the diagonal (e.g., A1 & B2) cannot be activated simultaneously without permitting energy to flow into adjoining actuators as well (e.g., A2 & B1). Therefore, it may take a sequence of control commands to achieve the complete desired actuation of the array. The fluidic network, shown in Figure 4.2B, is controlled in a similar manner to the electrical network, with an additional valve to switch between the hot and cold fluid sources. However, to prevent parasitic effects due to the capacitance of the fluidic vessels and

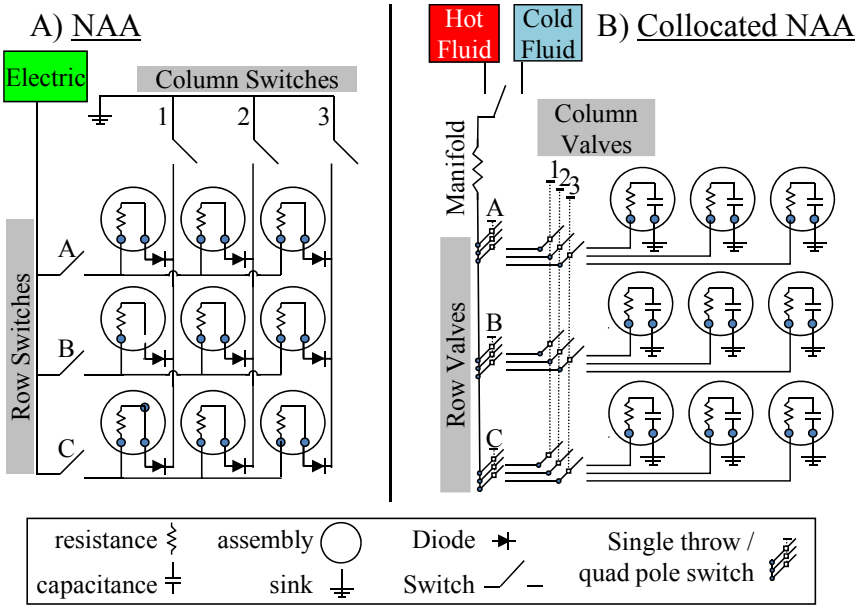


Figure 4.2: Network Array Architecture (NAA): A) Electric B) Fluidic. The electric NAA uses power transistors as switches to complete the circuit from source to ground. The fluidic NAA uses single-throw multiple-pole constrictor valves to control fluid flow through the array. The constrictors are used instead of solenoid valves because they can be collocated at one end of the array and do not introduce any fluid resistance when open. The collocation of the valves on the inlet side of the array eliminates parasitic fluidic dynamics associated with the basic NAA.

high fluid resistance of solenoid valves, the fluidic network must be constructed using a collocated NAA [27].

4.3.3 *Control Logic for NAA*

The control logic for NAA is described in detail in [28]. Each control device of NAA is treated as a binary device which is either connected (1) or disconnected (0). The N switches (valves) on the source and the sink side of the array can be in any of 2^N states (e.g., 0000, 1000, 0100, 1100, 1111). When the set of switches are all disconnected (0000), there is no flow (electric/fluidic) through the system from source to ground, which will not produce any change in the system. Therefore, there are 2^N-1 configurations for each set of switches. In order for energy to flow from the source through the actuator(s) to ground, at least one switch on the source side and one switch on the sink side must be connected (closed). Since the two sets of control devices are orthogonal to one another, there are $(2^N-1)^2$ configurations of the control devices that will connect a subarray (e.g., 1x1, 1x2, 2x1, 2x2, …, NxN) of actuators from the source to sink. Figure 4.3 shows a few examples of control command configurations for both the fluidic and electric NAA. Due to the architecture of NAA, not all the actuators can be controlled simultaneously and therefore it may take a sequence of control commands to

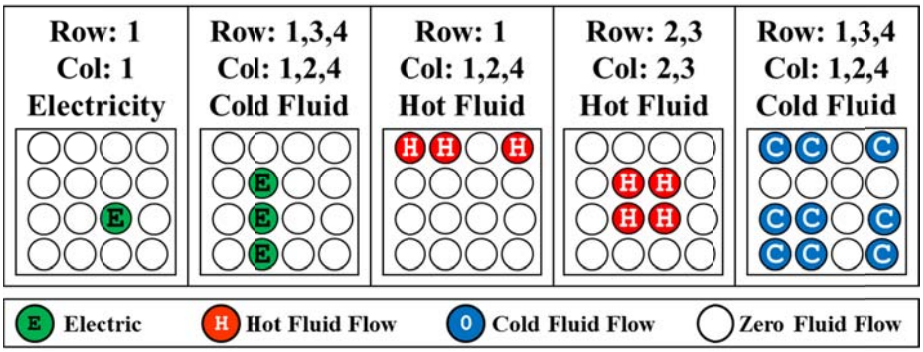


Figure 4.3: Control commands examples. There are $(2^N-1)^2$ combinations of intersecting rows and columns switches that can be closed to allow thermal energy to be delivered to various subarrays.

produce the desired actuation.

The fluidic network allows either hot or cold fluid to be delivered to the actuator array at any point in time. The fluidic and electrical networks are independent of one another and can operate in parallel. The algorithm which allows for this simultaneous operation will be discussed in section 4.5.

4.4 Optimization of the Wet SMA Actuator Array

Previous work [33]-[35] has characterized the performance of wet SMA actuators using both fluid and electricity to deliver thermal energy to produce actuation. Figure 4.4 shows the characteristics (speed, efficiency) of a single wet SMA actuator with different rates of energy input. The results show that the electrical network (P_e) is up to 2.5 times more efficient than and twice as fast as the fluidic network (Q, T_f). However, the energy source to heat the fluid (e.g., propane) can have up to 100 times the energy density than that of an electric battery. Although the fluidic heating is not as fast or efficient, approximately

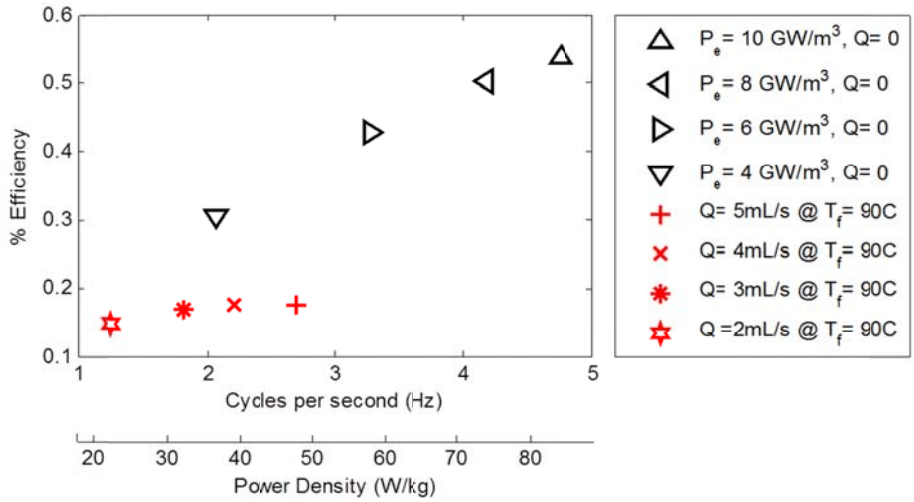


Figure 4.4: Simulated efficiency vs. speed of electrically and fluidically heated prototype actuator [34]. (P_e) is the electric power delivered for Joule heating of the SMA actuator. Q and T_f are the fluid flow rate and fluid temperature respectively. The electrically heated actuator is up to 2 times as fast and 2.5 times as efficient as the fluidically heated actuator.

30 times more total actuations can be completed per unit mass of energy storage. The NAA controller must use these characteristics to identify the optimal control command sequence. The previous work only modeled a single actuator and did not account for any energy losses associated with heating the fluid or delivering the hot fluid to the actuator manifold. Therefore, for the simulations in this paper, it will be assumed that fluidic heating has a 3:1 advantage (rather than a 30:1 advantage) over electric heating in terms of the energy cost. This will offer a worst case perspective of the algorithms that follow.

Figure 4.5 shows the fluid flow rate, electric current, the actuator efficiency, the total actuation energy cost, and the total actuation time for the prototype. The total energy cost and actuation times are calculated based on the flow rate through the each actuator, and are expressed with respect to the actuation time and energy cost of a single wet SMA actuator driven fluidically (τ_1, ε_1). Therefore, the cost to actuate a single actuator electrically will be $0.4\tau_1$ and $3\varepsilon_1$, which is 2.5 times faster and 3 times the energy cost of a fluidically driven actuator.

The cold fluid reservoir is assumed to maintain a constant temperature of 25℃ by free convection with the atmosphere, and therefore no energy is used to maintain this temperature. However, in the following algorithms, a pseudo-energy will be assigned to the cold fluid control command that is equal to that of the hot fluid control command. By doing so, the algorithm will not preference the cold control command over the hot fluid command of similar size when the objective of the algorithm is to minimize energy.

4.4.1 *Graph Theory Structure of NAA*

In the previous sections, the SMA actuators and NAA have been defined by discrete states and control commands. Because of these discrete components, Graph Theory [36] can be used to identify an optimal sequence of control commands to transition the wet SMA actuator from one state to another. NAA arranges the binary actuators (contracted (1) and extended (0)) in an NxN array that can be in one of $2^{N\text{x}N}$ states. These states will

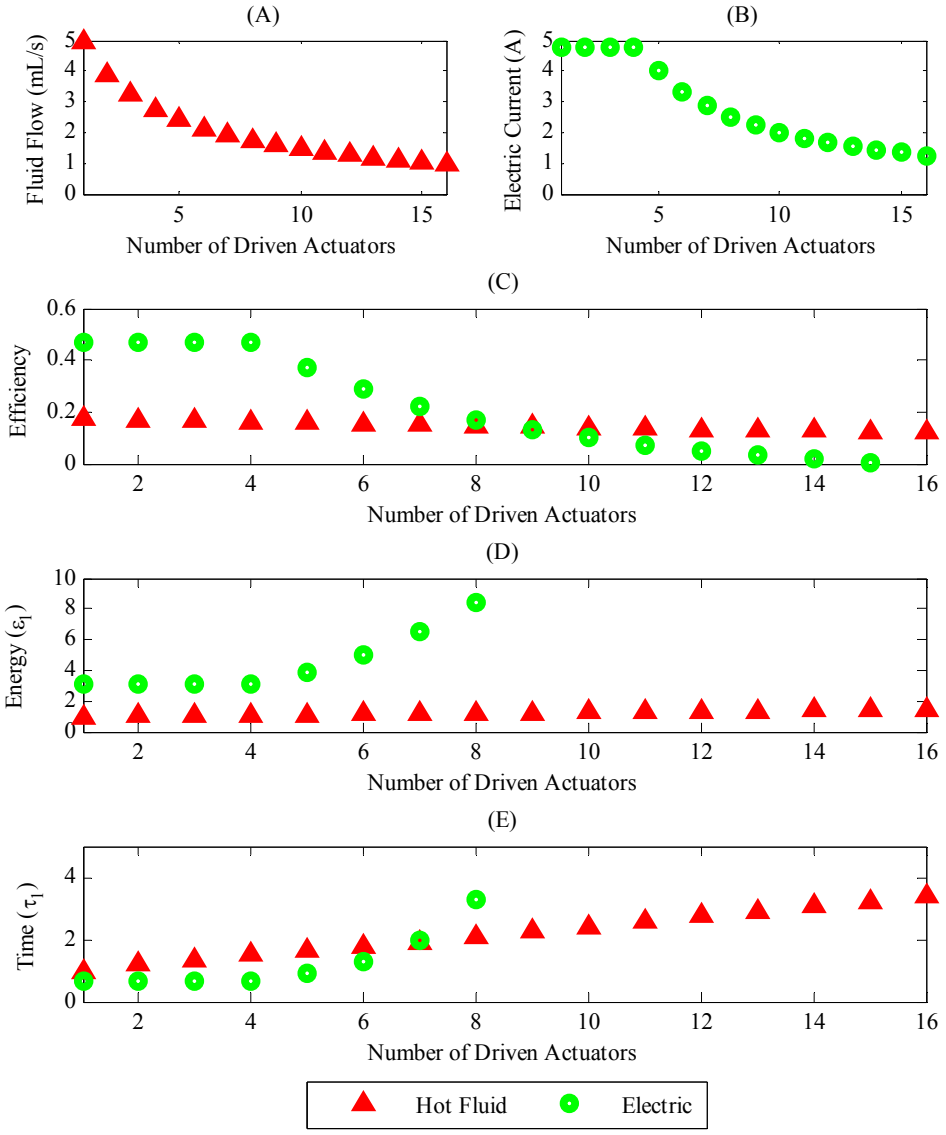


Figure 4.5: Characteristics of actuator array. A) Fluid flow rate with respect to the number of actuators driven of the MVD prototype with constant pressure sources. B) The electric current with respect to the number of driven actuators of the NAA prototype with a constant voltage source. C-E) The efficiency, energy cost and actuation times are defined with respect to the actuation energy cost and actuations times of single fluidic actuator, ε_1 and τ_1 respectively. The total current from the voltage source was limited to 20 Amps based on practical limits of the prototype system. When nine or more actuators are driven, the current is not sufficient to heat the wire above the transition temperature.

be treated as nodes in the graph. NAA defines $(2^N\text{-}1)^2$ discrete control command configurations that can address certain subarrays of actuators. These control commands are potential edges between nodes. Each of these control commands can be applied to any node of the graph; however some control commands will not produce any actuation. Commands that produce at least one actuation will be defined as edges of the graph. The NAA graph is defined as a *multigraph* because there can be multiple edges between two specific nodes. The simplest case accounts for the three input modes (hot fluid, cold fluid, or electricity). Therefore, there are $3(2^N\text{-}1)^2$ control commands that can be executed from each node.

The $2^{N\text{x}N}$ nodes and $3(2^N\text{-}1)^2$ control commands can be represented by N^2 bit numbers. In this paper the nodes and control commands/edges will be presented as decimal numbers, with the first bit representing the upper left corner and the bits will be numbered from left to right, top to bottom.

Edges of the graph can be easily identified by applying bitwise operations between the nodes and control commands. A complete graph for the NAA can be constructed with the following algorithm:

- Hot fluid or electrical heating control commands:
    - *h_test* = *OR*(current node, control command)
    - if *h_test* ~= current node, then control command = edge
- Cold fluid control command:
    - *c_test* = ~ *OR*(~current node,control command)
    - if *c_test* ~= current node, then control command = edge

The above algorithm identifies all the edges that cause at least 1 actuator to change states. If the control command, forming the edge, delivers energy to an actuator of a subarray, and there is no resulting actuation (e.g., hot fluid is delivered to a subarray of actuators, one of which is already hot and contracted), then this energy flow is termed *superfluous*.

As long as at least one actuation occurs, the edge is still valid, but not as energy efficient as one resulting in full actuation (e.g., 4 out of 4). Additionally, some edges are redundant because they produce the same actuation (e.g., a 1x2 control command with superfluous flow can be completed by a 1x1 control command). Even with just valid non-redundant edges, the complete NAA graph scales poorly. Table 4.1 shows the size of a preprocessed NAA graph for different size arrays. Even when $N = 4$, the size of the graph becomes marginally unmanageable [29]. In Section III, expanding wavefront algorithms will be presented to address this scalability issue.

When a fluidic or electric circuit is completed, the actuators are connected together in parallel to the source and sink. Therefore, the equivalent fluidic/electric resistance of the array will decrease as the number driven actuators increases. With this decrease in resistance for larger subarrays, the constant voltage/pressure source will deliver energy at faster rate than smaller control commands [28]. Also, with this increased energy flow rate, an edge (control command) may sacrifice energy with superfluous flow to produce the desired actuation in a shorter amount of time.

Figure 4.6 shows a small portion of the 2x2 graph where only a portion of the nodes and edges are shown. Figure 4.7 shows a couple of control command sequences that will transition the actuator array from Node 1 to Node 15. Path ABC has no superfluous flow and an accumulated time and energy cost of $[2.8\tau_1, 3\varepsilon_1]$. Path AC has an accumulated time and energy cost of $[2.3\tau_1, 4\varepsilon_1]$; where the desired actuation is completed faster with superfluous flow at the penalty of being less efficient. To express a preference for energy

Table 4.1: Size of Preprocessed Graph

N	Nodes 2^{NxN}	Control CMDs $3(2^N-1)^2$	# of Total Edges	Data Size for Preprocessed Graph
2	16	27	165	0.33 kb
3	512	147	25,419	50.8 kb
4	65536	675	14,910,429	29.8 Mb

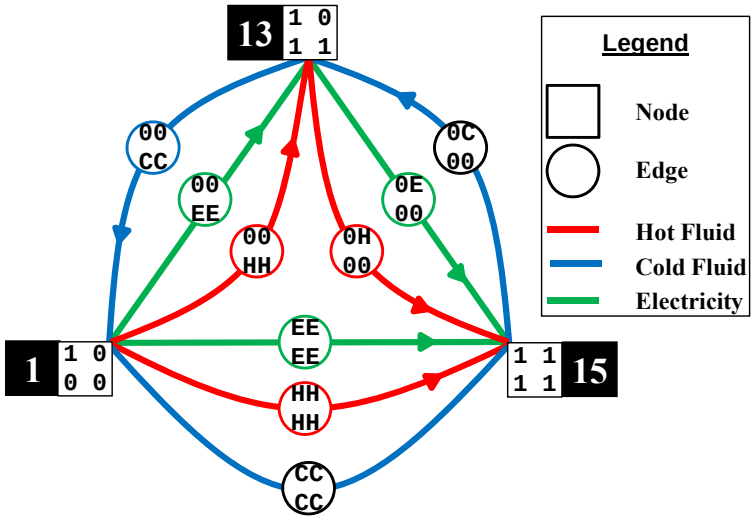


Figure 4.6: Partial graph of a 2x2 actuator array. The control commands (hot fluid, cold fluid, electricity) are edges that transition the array between Nodes 1, 13 and 15.

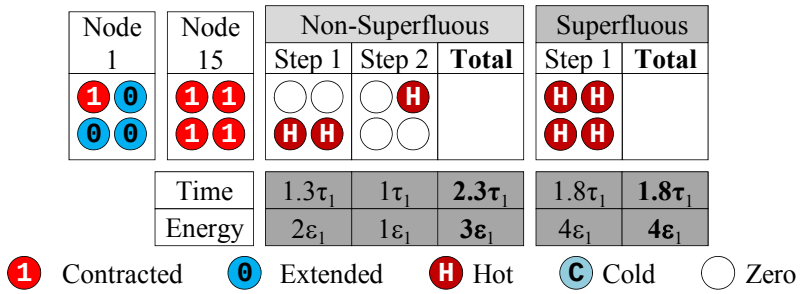


Figure 4.7: Two control command sequences are shown that will transition the actuator array from Node 1 to Node 15. The nonsuperfluous sequence produces the desired actuation with a sequence of two control commands with the minimum energy usage. The single control command with superfluous flow results in faster actuation, but uses more energy because hot fluid is delivered to actuator that is already contracted.

or time, a weighted time-energy cost (C_{te}) function with a performance weighting factor (w_{te}) has been defined as:

$$C_{te} = w_{te} * Time + (1 - w_{te}) * Energy \qquad (4.1)$$

where *Time* is the time necessary to delivery/remove enough thermal energy from the array and *Energy* is the thermal energy to necessary to produce the complete actuation. Both of these values are respectively normalized to the time and energy required to fluidically activate a single actuator. When $w_{te} = 1$, the preference is to minimize the total actuation time and when $w_{te} = 0$, the preference is to minimize the total actuation energy.

Although *Time* and *Energy* are normalized with respect to the performance of a single actuator, they are not scaled with respect to one another. Therefore, a w_{te} value of 0.5 may not represent an equal balance between the use of fluidic and electric heating.

4.4.2 *Graph Theory Algorithms*

Now that NAA has been formulated as a graph, three basic graph theory search algorithms [36] will be used to evaluate the performance of the actuator array and the computational cost to identify an optimal solution. All three algorithms use the basic principle of identifying a node with the minimum C_{te} and using this node to continue the search.

- **Best First Search** (BFS): is a greedy search technique which follows nodes with minimum estimated heuristic (cost-to-goal, C_{te}). While the path is not optimal, BFS tends to have a low computational cost.
- **Dijkstra's**: is a search technique that is guaranteed to identify a path with minimum path cost. However, Dijkstra's tends to be computationally costly, because it explores the graph with an uninformed perspective of the destination.
- **A*** (pronounced "A star"): is a search algorithm that can identify a minimum cost (C_{te}) path like Dijkstra's but it includes an estimated cost-to-goal heuristic like BFS to take a directed approach of exploring the graph. This results in lower computational cost than Dijkstra's and a more optimal path than BFS.

In order to find the optimal solution with the minimum amount of computational cost, the A^* search algorithm has been implemented with an estimated total cost-to-goal (C_{total}) of:

$$C_{total} = w_h * h + \sum C_{te} \tag{4.2}$$

where, $\sum C_{te}$ is the accumulated C_{te} to an intermediate node. The heuristic (h) or cost-to-

goal is estimated to be the minimum C_{te} for the number of actuators that remain to change states (flipped). This value may be associated with the fluid or electric domain depending on the value of w_{te}. Because the NAA graph is highly connected, having parallel paths between start and goal nodes, an additional heuristic weighting (w_h) has been added to the A*. When $w_h = 0$, then A* is equivalent to Dijkstra's and an optimal path will always be found. If h is admissible, then setting $w_h = 1$ will result in an optimal solution with the average computational cost being much less than Dijkstra's. If the computational cost, due to the NAA graph having a large number of parallel edges, is too large, then increasing w_h above 1 will decrease the computational cost. However, this may potentially sacrifice finding an optimal solution.

When multiple nodes have equal C_{total}, then an additional metric will be used to help identify the preferred node. In [28], a sum squared metric was used to identify nodes that had larger future control commands available. This metric showed to have up to a 32% improvement in complete sequence actuation times with a controller that operated randomly similar to BFS.

4.5 Scalability and Simultaneous Operation

4.5.1 *Expanding Wavefront*

In order to overcome the data size of the complete graph, a partial graph will be constructed as needed in real time for each origin and destination node set. The two primary reasons that this approach works is that the $3(2^N-1)^2$ control commands are scalable and the fact that simple binary operations can be used to identify the adjacent neighbors.

Although there are $3(2^N-1)^2$ control commands for each node, there will be at most $(2^N-1)^2$ neighbors to a single node. Figure 4.8 will be used to visualize the control commands applied to Node 1 that result in either some actuation, zero actuation or are redundant to a smaller control command. First, the hot fluid and electric commands produce the same

A)

Current Node	Node 1		Binary Format	1000	Decimal Format	1

B)

Index	CMD	bin	dec	Neighbor	bin	dec	Zero Actuation	Redundant Edge
H1		1000	1		1000	1	✓	
H2		0100	2		1100	3		
H4		0010	4		1010	5		
H8		0001	8		1001	9		
H3		1100	3		1100	3		✓
H5		1010	5		1010	5		✓
H10		0101	10		1101	11		
H12		0011	12		1101	13		
H15		1111	15		1111	15		

C)

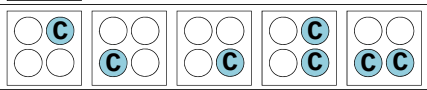
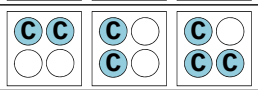
	Cold Control Commands	Neighbor
Actuation		
Zero Actuation		
Redundant CMDs		

Figure 4.8: Example of expanding wavefront algorithm. A) Current node in binary and decimal format. B) Propagation of wavefront from current node to neighbor using hot control commands. Control commands H1, H3, and H5 can be removed from the control command list because they do not produce actuation or are redundant to a smaller control command. C) A similar process can be used with the cold commands. From Node 1 the control command list is reduced from 9 hot & 9 cold control commands to 6 hot & 1 cold control commands.

actuation and will have common start and end nodes. Secondly, if a cooling control command is valid, the corresponding heating command is not valid because it does not produce any actuation and vice versa. Control command C1 is valid going from Node 1 to Node 0 and H1 results in no actuation. Finally, when some control commands (e.g., commands H2 & H5) are applied, they result in a common neighbor and the command addressing the larger submatrix is redundant to a smaller submatrix. For Node 1 of a 2x2 array, there are 27 possible control commands, but only 7 unique neighbors and 15 connecting edges. The A^* algorithm examines the minimum cost between the nodes and will identify whether electricity or hot fluid will be used to form the edge between the nodes, eliminating one half of the heating edges.

4.5.2 *Control Command Reduction*

The expanding wave front and A* algorithm can identify the optimal solution for arrays larger than 4x4, but at every node the wave expands, it must still examine $3(2^N-1)^2$ commands. Figure 4.8 shows that a large portion of the commands can be ignored while expanding the wavefront from a single node.

The redundant and zero actuation control commands can be filtered prior to expanding from the start node. The filtering process is accomplished by determining the desired heat transfer process between the start and destination nodes. From the required positive and negative heat transfer, control commands that result in zero actuation or that are redundant can be filtered out of the wavefront command list.

The algorithm used to identify the reduced control command list is shown in Table 4.2. The complete NAA command list is also used in this algorithm, [1000, 0100, 0010, 0001, 1100, 1010, 0101, 0011, 1111]. A numerical example is provided to the right of the algorithm and Figure 4.9 shows a graphical representation of the same solution. For this example, six heating control commands and one cooling control command will remain in the reduced control command list and these will be used in the expanding wavefront

Table 4.2: Reduced Command List Algorithm

	Results
any_heat_transfer = BITXOR(origin_node, destination_node)	1111
neg_heat_transfer = BITAND(origin_node, any_heat_transfer)	1000
neg_cmd_test = BITAND(neg_heat_transfer, NAA_command_list)	[1000,0000,0000, 0000,1000,0000, 1000,0000,1000]
neg_cmd_test_index = UNIQUE*(neg_cmd_test)	[1 2]
neg_cmd_index = REMOVE_INDEX**(neg_cmd_test == 0000,pos_cmd_test_index)	[1]
reduced_neg_cmd_list = NAA_cmd_list (neg_cmd_index)	[1000]
pos_heat_transfer = BITAND(destination_node, any_heat_transfer)	0111
pos_cmd_test = BITAND(pos_heat_transfer, NAA_command_list)	[0000,0100,0010, 0001,0100,0010, 0101,0011,0111]
pos_cmd_test_index = UNIQUE*(pos_cmd_test)	[1 2 3 4 7 8 9]
pos_cmd_index = REMOVE_INDEX**(pos_cmd_test == 0000,pos_cmd_test_index)	[2 3 4 7 8 9]
reduced_pos_cmd_list = NAA_cmd_list(pos_cmd_index)	[0100,0010,0001, 0101,0011,1111]

*The UNIQUE function identifies the index of the first unique elements of the array. ** The REMOVE_INDEX function identifies and removes the index of the control commands that result in zero actuation.

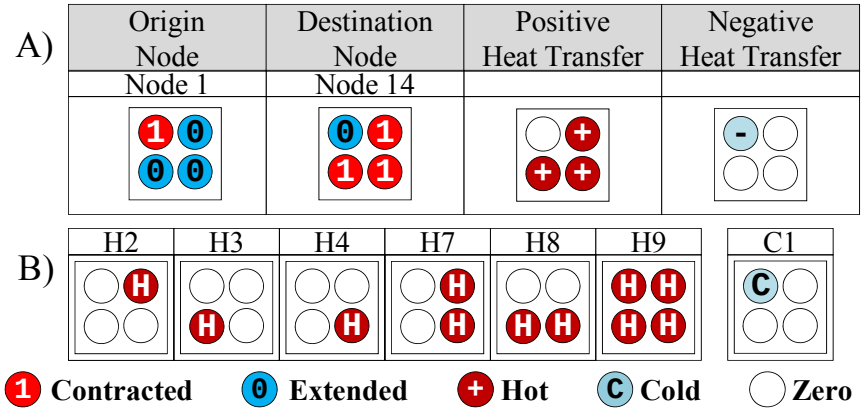


Figure 4.9: A) The required heat transfer processes. B) Reduced control commands list. This is a visual representation of the example presented in Table II. From the origin and destination nodes, the required heat transfer can be calculated, and the reduced control command list can then be determined with bitwise operations.

process. The original NAA graph constructed using the complete control command list has no disconnections between the start and destination nodes. However, with the reduced command list, the search algorithm may find a node with no neighbors. This is because a command with superfluous flow changes the state of an actuator that did not need to change states. At this point in the algorithm, the eliminated commands are temporarily reinstated to allow the wavefront to expand towards the destination. The path that includes this unnecessary actuation may be desirable because it is faster than working around this single actuator. For example, if the start node is 0 of 16 actuators contracted and the destination state is 15 out of 16 actuators contracted, then flushing the entire 4x4 array with hot fluid and then sending a 1x1 cold command is faster ($3.3\,\tau_1 + 1\,\tau_1 = 4.3\,\tau_1$) than sending a sequence of 3x4 and 1x3 hot commands ($3.1\,\tau_1 + 1.5\,\tau_1 = 4.6\,\tau_1$).

4.5.3 *Simultaneous Operation of Fluid and Electric Inputs*

As discussed earlier, the architecture of NAA limits how energy can be delivered to the actuator array and it may take multiple control commands to achieve the desired actuation. However, the fluidic and electrical networks are independent and can operate in parallel, reducing the total operational time. In Figure 4.10, case A is the optimal solution to minimize energy ($3\varepsilon_1$) while case B minimizes the time ($1.8\,\tau_1$) using a sequence of commands that do not overlap. However in case C, the electrical command, E2 ($0.4\,\tau_1$, $3\varepsilon_1$), can be executed simultaneously with C2 ($1\,\tau_1$, $1\varepsilon_1$) at time zero and then E4 ($0.4\,\tau$, $3\varepsilon_1$) can be immediately executed after E2, for accumulated cost of ($1\,\tau_1$, $7\varepsilon_1$).

Equation 3 defines the *Time* cost for simultaneous operation, where T_{ex} is the execution time of the control command and T_s is the time that the potential control command can operate simultaneously with the control commands from the other domain. Therefore, the accumulated execution time to each node must be tracked for both the fluidic and electrical domains. For case C, command C1 does not overlap with an electric command, so $T_s = 0$. After C1 has been completed, the accumulated execution time is $1\tau_1$ for the

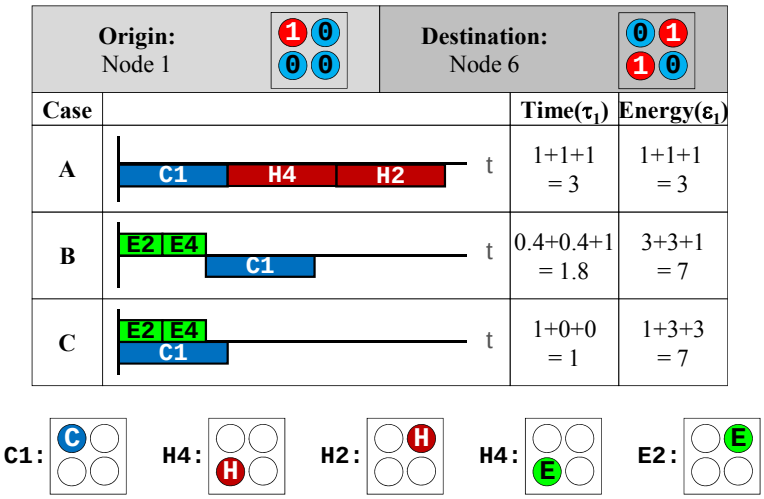


Figure 4.10: Comparison of sequential and simultaneous operations. The hot and cold fluids use the same vascular network to deliver/remove thermal energy and therefore can only operate sequentially. The electric and fluidic networks are independent of one another and can operate simultaneously for improved speed.

fluidic domain and $0\tau_1$ for the electrical domain. From this new intermediate node, the *Time* cost for E2 is $0\tau_1$ because this electrical command can be executed at the same time as C1 with complete overlap. From this second intermediate node, the *Time* cost for E4 is again $0\tau_1$ because this second electrical command can be executed immediately after E4 and at the same time as C1 with overlap. Figure 4.11 shows a case where simultaneous operations cannot be used. If steps 1 and 2 are done simultaneously, the state of the first actuator cannot be guaranteed.

$$Time = \begin{cases} T_{ex}, T_s \leq 0 \\ T_{ex} - T_s, 0 < T_s < T_{ex} \\ 0, T_s \geq T_{ex} \end{cases} \qquad (4.3)$$

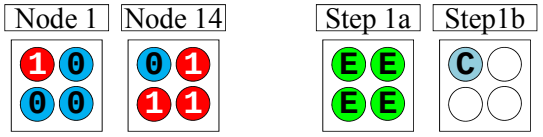


Figure 4.11: A case where simultaneous operation is not valid. Both steps 1a and 1b are viable control commands but when applied simultaneously, the state of the upper right actuator cannot be guaranteed.

4.6 Simulations and Analysis

4.6.1 *Analysis of Expanding Wavefront and Command Reduction*

The expanding wavefront and command reduction algorithms proposed above have been implemented in MATLAB. Figure 4.12 shows the performance of A^* using the expanding wave front algorithm with both the complete and reduced command list. Plots A and B show the average number of edges explored by the A^* algorithm using both the complete and reduced command list respectively. Although both lists produce the same solution (plot C) the number of edges explored by the reduced list is only about 5% of the edges explored for the complete list when $w_h = 0$ (Dijkstra's). The value of w_h may vary depending on the performance characteristics of the wet SMA actuator array, the choice of performance weighting (w_{te}) and/or the size of the array.

Figure 4.13 shows the characteristics of a 4x4 array of actuators where the control

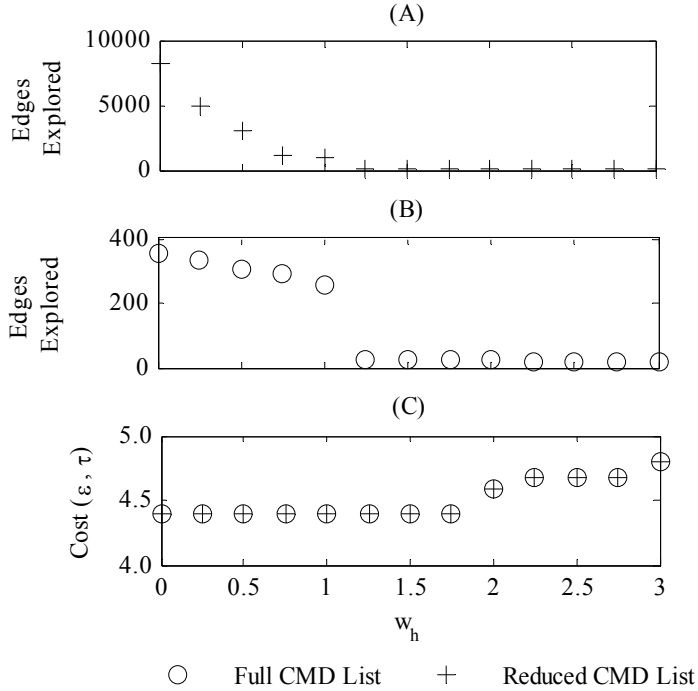


Figure 4.12: Comparison of full and reduced commands for a 4x4 array using the *A** expanding wavefront algorithm. Plots A & B show the number of edges explored for the full and reduced lists, respectively, as w_h is varied. Plot C shows that both lists result in the same solution.

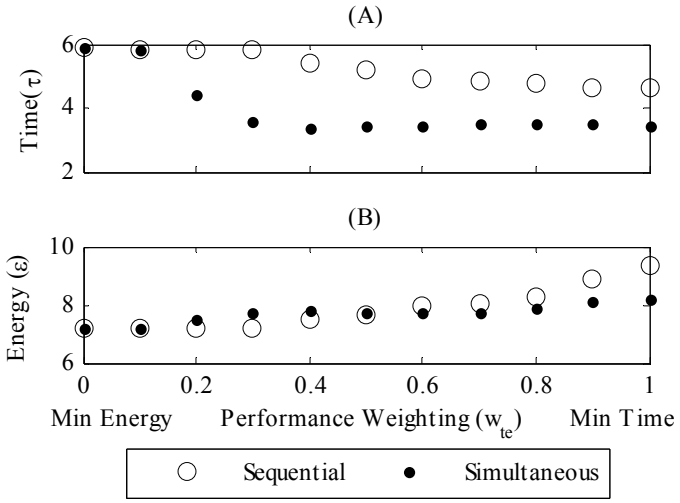


Figure 4.13: Performance comparison of a 4x4 actuator array using sequential and simultaneous control commands. The simultaneous mode uses more electrical energy in order to improve the speed of the array. At a performance weighting of greater than 0.5, the simultaneous mode becomes more energy efficient than the sequential mode. This is because the sequential mode is using only electricity, which has a higher energy cost.

inputs are done sequentially or simultaneously, where both methods sacrifice energy for faster operation. Figure 4.14 shows the type of energy being delivered to the actuators to achieve the desired performance. When the desired performance is to minimize energy ($w_{te} < 0.2$), both the sequential and simultaneous operations use the same commands to deliver the low energy cost fluid to the array. As the desired speed of the array becomes more important, the simultaneous mode is the first to introduce electricity. The sequential mode completely swaps out hot fluid for electricity to achieve lower total actuation times, while the simultaneous mode continues to use hot fluid with electricity. Both modes incorporate superfluous flow to decrease the actuation time as seen by the increase in the cold fluid energy (pseudo-energy). The superfluous flow increases the speed of the array, but delivers energy to actuators that do not change states (zero work), and lowers the total efficiency of the array. When time is the critical component of operating cost, the simultaneous mode saves approximately 15% on energy and 25% on time compared to the sequential mode.

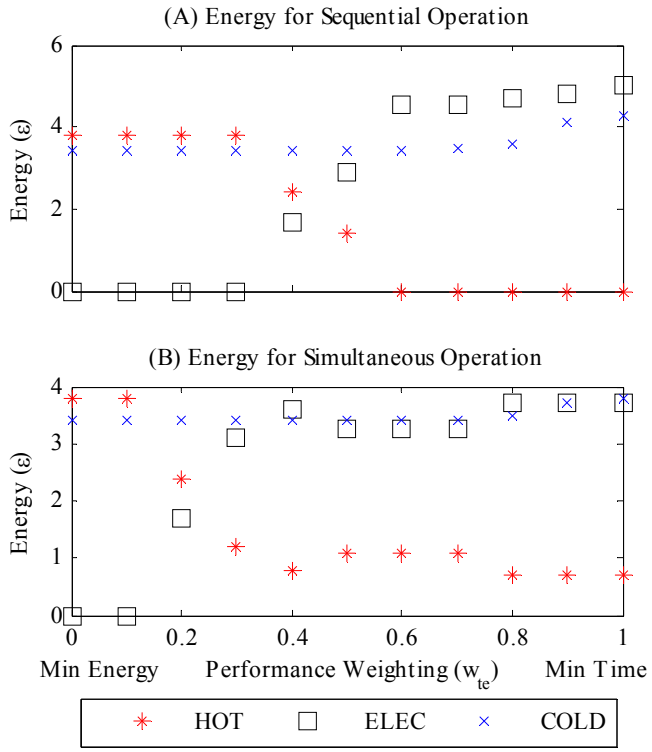


Figure 4.14: Comparison of energy inputs to the array for (A) sequential operation and (B) simultaneous operation. When it is desired to have the fastest operation, the sequential mode only uses the higher energy-cost electricity to heat the wire ($5.5\varepsilon_1$), while the simultaneous mode uses both hot fluid and electricity ($4.2\varepsilon_1$) to heat the wire.

Table 4.3 shows the relative performance of the expanding wavefront A* algorithm using the two command lists with w_h = 1.75. If w_h is set to 1, it can take A* multiple orders longer to explore the graph because there can be a large number of nodes that have the same estimated cost-to-goal. The value of w_h may vary depending on the performance characteristics of the wet SMA actuator array, performance weighting (w_{te}) or the size of the array. For the 5x5 array, the reduced command list explores only about 10% of the edges explored by the complete command list and it takes about 12% of the computational time. For the 6x6 array, only simple solutions could be calculated when the complete command list was used and therefore no average computational cost could be simulated.

Table 4.3: Performance Characteristics of A* with expanding wave front using $1.75 \cdot H$.

	Complete CMD list		Reduced CMD List	
N	Mean Edges Explored	Mean Comp Time(s)	Mean Edges Explored	Mean Comp Time(s)
3	103	0.029	21	0.016
4	734	0.167	98	0.031
5	4623	1.132	469	0.116
6	-	-	2424	0.640

4.7 Conclusion

This paper presents the implementation of graph theory to optimally control an array of previous challenge of the scalability of identifying an optimal path by constructing the SMA actuators using thermofluidic and electric inputs. This work overcomes the graph on the fly instead of using a preprocessed graph that scaled poorly. The wavefront algorithm examines the required heat transfer process and removes control commands that result in no actuation or redundant control commands. This reduction of commands can improve the search algorithm by as much as 90% while continuing to find an optimal solution. The expanding wavefront and control command reduction has allowed for an optimal solution to be found for a 6x6 array, which has over one million more states than a 4x4 array of previous work. Allowing the fluidic and electric networks to operate simultaneously results in both faster and more efficient operation. While this work has been motivated by research with SMA actuator arrays, it is anticipated that these control methods could also be applied to a more general class of actuators with multiple inputs, where the inputs have differing time and energy costs.

Additional work will need to examine the operation of wet SMA arrays that have rows of actuators connected in series or parallel for greater resolution of the displacements. The challenge with this hardware configuration is that there are multiple combinations that result in the same total displacements (e.g., [1100], [1010], [0011]). For this reason, there would not be a single destination node and the A* algorithm would need to be

modified to identify the destination node as well as the path with minimum weighted cost.

4.8 References

[1] I. Kossyk, J. Dorr, and K. Kondak, “Design and evaluation of a wearable haptic interface for large workspaces,” in *Proc. IEEE International Conference on Robots and Systems*, Taipei, Taiwan, pp. 4674-4679, 2010.

[2] P. Taylor, A. Moser, and A. Creed, “Design and control of a tactile display based on shape memory alloys,” in Proc. IEEE International Conference on Robotics and Automation, Albuquerque, NM, pp. 1318-1323, 1997.

[3] Younkoo Jeong, Dongjoon Lee, Kyunghwan Kim, and Jong Oh Park, “A wearable robotic arm with high force-reflection capability,” in *Proc. IEEE International Workshop on Robot and Human Interactive Communication,* Osaka, Japan, pp. 411-416*,* 2000.

[4] E. Biddiss and Tom Chau, “Dielectric elastomers as actuators for upper limb prosthetics: Challenges and opportunities,” *Medical Engineering & Physics*, vol. 30, issue 4, pp. 403-418, 2008.

[5] C. Moreno, F. Brunetti, J. Pons, J Baydal, and R. Barbera, “Rationale for multiple compensation of muscle weakness walking with a wearable robotic orthosis,” in *Proc. IEEE International Conference on Robotics and Automation*, Arganda del Rey, Spain, pp. 1914, 2005.

[6] L. Sherwood, *Human physiology 7th ed.*, Belmont, CA, Cengage Learning, 2008.

[7] I. Hunter and S. Lafontaine, "Comparison of muscle with artificial actuators," in *Proc. IEEE Solid-State Sensors and Actuator Workshop*, Hilton Head Island, SC, USA, pp. 178-185, 1992.

[8] E. Steltz and R. Fearing, “Dynamometer Power Output Measurements of Piezoelectric Actuators,” in Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3980-3986, 2007.

[9] S. Mascaro and H. Asada, "Wet shape memory alloy actuators for active vasculated robotic flesh," in *Proc. IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, pp. 282-287, 2003.

[10] S. Dutta and F. Ghorbel, "Differential hysteresis modeling of a shape memory alloy wire actuator," *IEEE/ASME Transactions on Mechatronics*, vol. 10, pp. 189-197, 2005.

[11] S. Govindjee and E. Kasper, "Computational aspects of one-dimensional shape memory alloy modeling with phase diagrams," *Computer Methods in Applied Mechanics and Engineering*, vol. 171, pp. 309-326, 1999.

[12] K. Ikuta, M. Tsukamoto, and S. Hirose, "Mathematical model and experimental verification of shape memory alloy for designing micro actuator," in *Proc. IEEE Micro Electro Mechanical Systems*, Nara, Japan, pp. 103-108, 1991.

[13] K. Kuribayashi, S. Shimizu, M. Yoshitake, and S. Ogawa, "Mechanical properties and control of shape memory alloy thin film actuator," *Journal of the Japan Society for Precision Engineering*, vol. 64, pp. 413-417, 1998.

[14] C. Liang and C. Rogers, "One-dimensional thermo-mechanical constitutive relations for shape memory materials," *Journal of Intelligent Material Systems and Structures*, vol. 8, pp. 285-302, 1997.

[15] Y. Liu, D. Favier, and L. Orgeas, "Mechanistic simulation of thermomechanical behaviour of thermoelastic martensitic transformations in polycrystalline shape memory alloys," in *Proc. 7th European Mechanics of Materials Conference on Adaptive Systems and Materials: Constitutive Materials and Hybrid Structures*, Frejus, France, pp. 37-45, 2004.

[16] A. Bhattacharyya and D. Lagoudas, "Stochastic thermodynamic model for the gradual thermal transformation of SMA polycrystals," *Smart Materials and Structures*, vol. 6, pp. 235-250, 1997.

[17] K. O'Toole, M. McGrath, and D. Hatchett, "Transient characterization and analysis of shape memory alloy wire bundles for the actuation of finger joints in prosthesis design," *Mechanika*, vol. 68, pp. 65-69, 2007.

[18] M. Elahinia and H. Ashrafiuon, "Nonlinear control of a shape memory alloy actuated manipulator," *Journal of Vibration and Acoustics*, Transactions of the ASME, vol. 124, pp. 566-575, 2002.

[19] J. Jayender and R. Patel, "Modeling and Gain Scheduled Control of Shape Memory Alloy Actuators," in *Proc. IEEE Conference on Control Applications, Toronto*, Canada, pp. 767-772, 2005.

[20] B. Selden, K. Cho, and H. Asada, "Segmented binary control of shape memory alloy actuator systems using the peltier effect," in *Proc. IEEE International Conference on Robotics and Automation*, New Orleans, LA, USA, pp. 4931-4936, 2004.

[21] K. Cho, B. Selden, and H. Asada, "Segmented binary control of shape memory alloy actuator systems," *Smart Structures and Materials*, pp. 314-322, 2005.

[22] K. Cho, J. Rosmarin, and H. Asada, "SBC hand: a lightweight robotic hand with an SMA actuator array implementing C-segmentation," in *Proc. IEEE International Conference on Robotics and Automation*, Rome, Italy, pp. 921-926, 2007.

[23] K. De Laurentis, A. Fisch, J. Nikitczuk, and C. Mavroidis, "Optimal design of shape memory alloy wire bundle actuators," in *Proc. IEEE International Conference on Robotics and Automation*, Washington, DC, USA, pp. 2363-2368, 2002.

[24] R. Mukherjee, T. Christian, and R. Thiel, "An actuation system for the control of multiple shape memory alloy actuators," *Sensors and Actuators A: Physical*, vol. 55, pp. 367-382, 1996.

[25] Mascaro and H. Asada, "Vast DOF wet shape memory alloy actuators using matrix manifold and valve system," in *Proc. ASME International Mechanical Engineering Congress, Dynamic Systems and Control Division*, Washington, DC, USA, pp.577-582, 2003.

[26] S. Mascaro, K. Cho, and H. Asada, "Design and control of vast DOF wet SMA array actuators," in *Proc. of the IEEE International Conference on Intelligent Robots and Systems*, Las Vegas, NV, USA, pp. 577-582, 2003.

[27] L. Flemming and S. Mascaro, "Wet SMA actuator array with matrix vasoconstriction device," *in Proc. ASME International Mechanical Engineering Congress and Exposition, Dynamic Systems and Control Division*, Orlando, FL, USA, pp. 1751-1758, 2005.

[28] L. Flemming and S. Mascaro, "Control of a scalable matrix vasoconstrictor device for wet actuator arrays," in Proc. IEEE International Conference on Robotics and Automation, Rome, Italy, pp. 638-653, 2007.

[29] L. Flemming, D. Johnson and S. Mascaro, "Optimal control of multi-input SMA actuator arrays using graph theory," in *Proc. IEEE International Conference on Robotics*, Shanghai, China, pp. 6109–6114, 2011.

[30] L. Flemming, D. Johnson and S. Mascaro, "Optimal control of multi-input SMA actuator arrays using graph theory: expanding wavefront & simultaneous operation," in *Proc. IEEE/RSJ International Conference on Robots and Systems*, San Francisco, CA, USA, pp. 780-785, 2011.

[31] "Technical Characteristics of Flexinol Actuator Wires" Dynalloy Inc., Available: www.dynalloy.com.

[32] M. Nakatani, H. Kajimoto, K. Vlack, D. Sekiguchi, N. Kawakami, and S. Tachi, "Pop up!: 3D form display with coil-type shape memory alloy," *Journal of the Institute of Image Information and Television Engineers*, vol. 60, pp. 183-191, 2006.

[33] J. Ertel and S. Mascaro, “Thermomechanical modeling of a wet shape memory alloy actuator,” in Proc. ASME International Mechanical Engineering Congress and Exposition, Chicago, IL, USA, pp. 1317-1324, 2006.

[34] L. Flemming and S. Mascaro, "Analysis of hybrid electric/thermofluidic control for wet shape memory alloy actuators," *in Proc. ASME Dynamic Systems and Controls Conference*, Hollywood, CA, USA, pp. 1041-1047, 2009.

[35] J. Ertel and S. Mascaro, “Dynamic thermomechanical modeling of a wet shape memory alloy actuator,” *Transactions of the ASME Journal of Dynamic Systems, Measurements and Control*, vol. 132, no. 4, pp. 1-9, 2010.

[36] S. Russell and P. Norvig, *Artificial intelligence – a modern approach*, 3rd ed., Upper Saddle River, New Jersey, USA: Pearson Education Inc., 2010.

CHAPTER 5

CONCLUSIONS

The work presented in this dissertation has shown that large arrays of wet SMA actuators can be controlled intelligently based on the performance characteristics of the actuators and graph theory algorithms. NAA not only addresses the hardware scalability of wet SMA actuators, but the discrete nature of NAA has been shown to be beneficial with the scalability of the algorithms. The discrete NAA control commands allow the NAA graph to be constructed using an expanding wavefront and simple bitwise operations. By filtering the complete control command list based on the desired actuation, the A^* algorithm, in conjunction with the expanding wave front algorithm, explores only a small percentage of the NAA graph, less than 0.01% for a 4x4, to identify an optimized sequence of control commands. This intelligent control theory will allow other actuators such as Electroactive Polymers (EAPs), to be integrated into large arrays and be controlled by multiple inputs.

Although the A^* algorithm and the expanding wavefront provide an effective method of identifying an optimal sequence of control commands for large arrays, it has not completely resolved the issue of scalability. When the columns of actuators are connected in series or parallel, there may be a large number of nodes (array configurations) that can produce the desired displacements (destination nodes). The current A^* algorithm can be adapted to search for a minimum cost solution to multiple nodes; however, it is not viable

to search all these nodes to guarantee an optimal solution. For example, when a 5x5 array has a desired displacement of [2 2 2 2 2], there are 100000 nodes that can produce this displacement. The solution presented in Chapter 2, Section F appears to be a partial solution to this problem. This forecasting process ensures that selected destination node has the largest subarrays available for the proceeding destination. Although this may not be an optimal node for the current desired displacements, the selected node is believed to the best node with minimal cost to all future desired displacements.

In this work, all of the actuators had equal lengths, and when connected in series, there are N+1 discrete displacements that can be produced. If it is desired to have finer resolution with equal length actuators, the number of actuators would have to increase. Another option would be to have unequal actuator lengths. For example, if there are two actuators with lengths of 1/3 and 2/3, respectively, then there are 2^N-1 (3) combinations of contracted actuators that produce unique displacements. For this method, there would be a single destination node that can produce the desired displacement (each column would have a single solution). One challenge of this solution is that each actuator would need to be controlled for different amounts of time. Additionally, if the desired displacements was somewhere between two combinations of wire segments, there could be multiple destination to examine for an optimal solution.

The solution presented in this dissertation only examined the cost to transition the array from one state to another. However, there could be additional energy cost incurred by the array to maintain the actuators in contracted state. If an actuator is electrically heated, it must be continuously heated to remain contracted. Whereas if it had been driven fluidically, the fluid flow could be turned off and the actuator would remain contracted

for a longer period of time. Therefore the cost function will need to be modified to manage desired duration of the contractions.

APPENDIX

NAA GRAPH THEORY ALGORITHMS

The algorithms presented in this dissertation were implemented in MATLAB. astar_naa.m is the primary function that integrates the components of Chapters 2 and 4. run_Astar.m is a script that runs the A* algorithm with different origin and destination and performance characteristics (energy and time). The build_cost_index.m and build_time_index.m are functions that are used in the run_Astar script to format the performance characteristics in to the correct format for the astar_naa. build_N_submatrix.m is a script that can construct a table or control command for different size arrays.

A.1 astar_naa.m

This function determines that optimal control command sequence between the origin and destination nodes. There are 13 nested functions in this function that are able to share data without the need to passing the data in and out of separate functions. This function also uses persistent variables which allow the memory locations to remain active after the function is completed (closed) and therefore these memory location do not have to be re-established the next time the function is called.

```
function [V_output,V_count] =
astar_naa(ORIGIN,DESTINATION,WH,COST,SIMULTANEOUS,REDUCE)
global SUBMATRIX  N2 N_CMDS

%%%%%
% The persistent function variables that are local to the function in which
% they are declared yet their values are retained in memory between calls
% to the function. The goal of using the persistent function it prevent the
% need to allocate memory locations every time the function is called.
%%%%%

%%%%%
% Naming Convention
%
% All terms with all CAPS are inputs to this function
% All terms beginning with 'I' represent an index in the explored table
% and will never change in the function
%
% All terms beginning with 'V' are variables that will updated during the function
%
% All terms that are lower case are functions.  Those functions starting
```

```
% with 'nf' are nested / embedded functions with in this 'astar_naa.m'_


%%%%%
% Some lines of code are too long for printing.  A series of three dots
% (...) allow for the line of code to be continued on to the next page.
%%%%%

persistent I_hot  I_elec  I_cold
persistent I_node  I_node_index  I_prev_node  I_prev_node_index
persistent I_accum_path_cost  I_estimated_total_cost  I_cost2goal
persistent I_complete_hot  I_complete_elec  I_complete_cold
persistent I_time2node_fluid  I_time2node_elec  I_cmd_execute_time
persistent I_cmd2node  I_cmd2node_index  I_cmdtype2node  I_edge_cost
persistent V_explored  V_explored_size  V_part_explored
persistent V_openset_index  V_active_set
persistent V_edges2neighbors_heat_index  V_cost_of_edge
persistent V_edges2neighbors_cool_index  V_explored_neighbor_index  V_current_cmd
persistent V_current_cmd_index  V_neighbors2current_cold  V_neighbors2current_heat
persistent V_cost_hot  V_test_cost_hot  V_test_cost V_cmd_type V_prev_node_index
persistent V_cost_elec  V_test_cost_elec
persistent V_sub_h_cmd  V_sub_h_index  V_sub_h_cmd_not  V_sub_h_index_not
persistent V_sub_c_cmd  V_sub_c_index  V_sub_c_cmd_not  V_sub_c_index_not
persistent V_time2node_hot  V_min_est_index_temp
persistent V_time2node_elec  V_current_explored_node_index V_current_node
V_time2node
persistent V_execute_time_elec  V_execute_time_hot  V_execute_time   V_neighbor

nf_initialize;
nf_submatrix_reduced;
nf_astar;
nf_create_directions;

   function nf_astar
      while ~isempty(V_openset_index)
        V_part_explored = V_explored(:,V_openset_index);
        [~, V_min_est_index_temp] = min(V_part_explored(I_estimated_total_cost,:));
        V_current_node = V_part_explored(I_node,V_min_est_index_temp);
        V_current_explored_node_index =...
           V_part_explored(I_node_index,V_min_est_index_temp);
        if V_current_node == DESTINATION;
           break % this will get out of this for while loop
        end
        nf_move_open2closed
        nf_expand_wavefront
        for ii = length(V_neighbors2current_heat):-1:1;
           V_count = V_count+1;
           V_neighbor = V_neighbors2current_heat(ii);

           V_explored_neighbor_index = find(V_neighbor == V_active_set);
           nf_is_in_explored
           V_current_cmd_index = V_edges2neighbors_heat_index(ii);
           V_current_cmd = SUBMATRIX.dec(V_current_cmd_index);
           nf_calculate_hot_cost
           nf_calculate_elec_cost
           if V_test_cost_hot < V_test_cost_elec
              V_test_cost = V_test_cost_hot;
              V_cost_of_edge = V_cost_hot;
              V_cmd_type = I_hot;
              V_execute_time = V_execute_time_hot;
              V_time2node = V_time2node_hot;
           else
              V_test_cost = V_test_cost_elec;
              V_cost_of_edge = V_cost_elec;
              V_cmd_type = I_elec;
              V_execute_time = V_execute_time_elec;
```

```
            V_time2node = V_time2node_elec;
         end
         % this is not <=, becuase there is no need to nf_update path if solution
         % is already optimal
         if V_test_cost < V_explored(I_accum_path_cost,V_explored_neighbor_index);
            nf_update
         end
      end
      for ii = length(V_neighbors2current_cold):-1:1;
         V_count = V_count+1;
         V_neighbor = V_neighbors2current_cold(ii);
         V_explored_neighbor_index = find(V_neighbor == V_active_set);
         nf_is_in_explored;
         V_current_cmd_index = V_edges2neighbors_cool_index(ii);
         V_current_cmd = SUBMATRIX.dec(V_current_cmd_index);
         nf_calculate_cold_cost
         % this is not <=, becuase there is no need to nf_update path if
         % solution is already optimal
         if V_test_cost < V_explored(I_accum_path_cost,V_explored_neighbor_index);
            V_cmd_type = I_cold;
            nf_update
         end
      end
   end %%%%% close nf_astar
end


function nf_initialize
   if isempty(V_explored)
     I_hot = 1;
     I_elec = 2;
     I_cold = 3;
     I_node = 1;
     I_node_index = 2;
     I_prev_node = 3;
     I_prev_node_index = 4;
     I_accum_path_cost = 5;
     I_estimated_total_cost = 6;
     I_edge_cost =7;
     I_time2node_fluid = 8;
     I_time2node_elec = 9;
     I_cmd_execute_time = 10;
     I_cmd2node = 11;
     I_cmd2node_index = 12;
     I_cmdtype2node = 13;
     I_cost2goal = 14; % heuristic value # bits to be flipped
     I_complete_hot = 15;
     I_complete_elec = 16;
     I_complete_cold = 17;
     V_explored = zeros(I_complete_cold,5000);
   end
   V_count=0;
   V_explored= V_explored*0;
   V_explored_size = 1;
   V_explored_index = V_explored_size;
   % this set of I_nodes have a path, but the path cost to I_node has
   % not been proven to be minimum
   V_explored(I_node,V_explored_index) = ORIGIN;
   V_explored(I_node_index,V_explored_index) = V_explored_size;
   V_explored(I_cost2goal,V_explored_index) = nf_weighted_cost2goal(ORIGIN);
   V_explored(I_estimated_total_cost,V_explored_index) =...
     V_explored(I_cost2goal,V_explored_index);
   V_openset_index= 1;
   V_active_set = ORIGIN;
   V_current_node = ORIGIN;
   V_current_explored_node_index = 1;
```

```
end %%close nf_intialize


function nf_calculate_hot_cost
   if SIMULTANEOUS
     V_overlap =...
        bitand(V_explored(I_complete_elec,V_current_explored_node_index)...
        ,V_current_cmd);
     if V_overlap == 0
        V_execute_time_hot =...
           V_explored(I_time2node_fluid,V_current_explored_node_index);
     else
        V_execute_time_hot =...
           V_explored(I_time2node_elec,V_current_explored_node_index);
        V_segment_overlap =...
           bitand(V_explored(I_cmd2node,V_current_explored_node_index)...
           ,V_current_cmd);
        V_prev_node_index = V_explored(I_prev_node_index...
           ,V_current_explored_node_index);
        while ((V_segment_overlap == 0) && (V_prev_node_index > 1))
           V_execute_time_hot = V_explored(I_time2node_elec,V_prev_node_index);
           V_segment_overlap =...
             bitand(V_explored(I_cmd2node,V_prev_node_index),V_current_cmd);
           V_prev_node_index = V_explored(I_prev_node_index,V_prev_node_index);
        end
        if V_execute_time_hot <...
             V_explored(I_time2node_fluid,V_current_explored_node_index)
           V_execute_time_hot =...
             V_explored(I_time2node_fluid,V_current_explored_node_index);
        end
     end
     V_time2node_hot = V_execute_time_hot + COST.time(I_hot,V_current_cmd_index);
     % Complete V_overlap
     if V_time2node_hot <=...
           V_explored(I_time2node_elec,V_current_explored_node_index)
        V_cost_hot = COST.w_energy(I_hot,V_current_cmd_index);
        V_test_cost_hot =...
           V_explored(I_accum_path_cost,V_current_explored_node_index)...
           + COST.w_energy(I_hot,V_current_cmd_index) +  0 ;
        % No V_overlap
     elseif V_execute_time_hot >=...
           V_explored(I_time2node_elec,V_current_explored_node_index)
        V_cost_hot = COST.w_energy(I_hot,V_current_cmd_index)...
           + COST.w_time(I_hot,V_current_cmd_index);
        V_test_cost_hot =...
           V_explored(I_accum_path_cost,V_current_explored_node_index)...
           + COST.w_energy(I_hot,V_current_cmd_index)...
           + COST.w_time(I_hot,V_current_cmd_index) ;
        % partial V_overlap
     else
        V_cost_hot = + COST.w_energy(I_hot,V_current_cmd_index)...
           + (V_time2node_hot -...
           V_explored(I_time2node_elec,V_current_explored_node_index))...
           / COST.time(I_hot,V_current_cmd_index)...
           *COST.w_time(I_hot,V_current_cmd_index);
        V_test_cost_hot =...
           V_explored(I_accum_path_cost,V_current_explored_node_index)...
           + COST.w_energy(I_hot,V_current_cmd_index)...
           + (V_time2node_hot...
           -V_explored(I_time2node_elec,V_current_explored_node_index))/...
           COST.w_time(I_hot,V_current_cmd_index)...
           *COST.weighted_t(V_current_cmd_index);
     end
   else %NOT SIMULTANEOUSLY
     V_execute_time_hot =...
        max(V_explored(I_time2node_fluid:I_time2node_elec,...
```

```
        V_current_explored_node_index));
     V_time2node_hot = V_execute_time_hot + COST.time(I_hot,V_current_cmd_index);
     V_test_cost_hot =...
        V_explored(I_accum_path_cost,V_current_explored_node_index)...
        + COST.w_cost(I_hot,V_current_cmd_index);
     V_cost_hot =  COST.w_cost(I_hot,V_current_cmd_index);
   end
end %%%%% close nf_calculate_hot_cost


function nf_calculate_elec_cost
   if SIMULTANEOUS
     V_overlap =...
        bitand(V_explored(I_complete_cold,V_current_explored_node_index),...
        V_current_cmd);
     if V_overlap == 0;
        V_execute_time_elec =...
           V_explored(I_time2node_elec,V_current_explored_node_index);
     else
        V_execute_time_elec =...
           V_explored(I_time2node_fluid,V_current_explored_node_index);
        V_segment_overlap =...
           bitand(V_explored(I_cmd2node,V_current_explored_node_index),...
           V_current_cmd);
        V_prev_node_index = V_explored(I_prev_node_index,...
           V_current_explored_node_index);
        while (((V_segment_overlap == 0) && (V_prev_node_index > 1))...
             && ~(V_execute_time_elec < V_explored(I_time2node_elec,...
             V_current_explored_node_index)))
           V_execute_time_elec= V_explored(I_time2node_fluid,V_prev_node_index);
           V_segment_overlap =...
             bitand(V_explored(I_cmd2node,V_prev_node_index),V_current_cmd);
           V_prev_node_index = V_explored(I_prev_node_index,V_prev_node_index);
        end
        if V_execute_time_elec <...
             V_explored(I_time2node_elec,V_current_explored_node_index)
           V_execute_time_elec =...
             V_explored(I_time2node_elec,V_current_explored_node_index);
        end
     end

     V_time2node_elec = V_execute_time_elec...
        + COST.time(I_elec,V_current_cmd_index);
     % Complete V_overlap
     if V_time2node_elec <= V_explored(I_time2node_fluid,...
           V_current_explored_node_index)
        V_cost_elec =COST.w_energy(I_elec,V_current_cmd_index) +  0;
        V_test_cost_elec = V_explored(I_accum_path_cost,...
           V_current_explored_node_index)...
           + COST.w_energy(I_elec,V_current_cmd_index) +  0;
        % No V_overlap
     elseif V_execute_time_elec >=...
           V_explored(I_time2node_elec,V_current_explored_node_index)
        V_cost_elec = COST.w_energy(I_elec,V_current_cmd_index)...
           + COST.w_time(I_elec,V_current_cmd_index);
        V_test_cost_elec =...
           V_explored(I_accum_path_cost,V_current_explored_node_index)...
           + COST.w_energy(I_elec,V_current_cmd_index)...
           + COST.w_time(I_elec,V_current_cmd_index);
        % partial V_overlap
     else
        V_cost_elec = COST.w_energy(I_elec,V_current_cmd_index)...
           + (V_time2node_elec...
           -V_explored(I_time2node_fluid,V_current_explored_node_index))...
           / COST.time(I_elec,V_current_cmd_index)...
           *COST.w_time(I_elec,V_current_cmd_index);
```

```
        V_test_cost_elec = V_explored(I_accum_path_cost,...
           V_current_explored_node_index)...
           + COST.w_energy(I_elec,V_current_cmd_index)...
           + (V_time2node_elec...
           -V_explored(I_time2node_fluid,V_current_explored_node_index))...
           / COST.w_time(I_elec,V_current_cmd_index)...
           *COST.weighted_t(V_current_cmd_index);
      end

   else
      V_execute_time_elec =...
         max(V_explored(I_time2node_fluid:I_time2node_elec,...
         V_current_explored_node_index));
      V_time2node_elec = V_execute_time_elec...
         + COST.time(I_elec,V_current_cmd_index);
      V_test_cost_elec =...
         V_explored(I_accum_path_cost,V_current_explored_node_index)...
         + COST.w_cost(I_elec,V_current_cmd_index);%(time & energy)
      V_cost_elec = COST.w_cost(I_elec,V_current_cmd_index);%(time & energy)
   end
end %%%%% close nf_calculate_elec_cost


function nf_calculate_cold_cost
   if SIMULTANEOUS
      V_overlap =...
         bitand(V_explored(I_complete_elec,V_current_explored_node_index),...
         V_current_cmd);
      if V_overlap == 0
         V_execute_time =...
            V_explored(I_time2node_fluid,V_current_explored_node_index);
      else
         V_execute_time =...
            V_explored(I_time2node_elec,V_current_explored_node_index);
         V_segment_overlap =...
            bitand(V_explored(I_cmd2node,V_current_explored_node_index),...
            V_current_cmd);
         V_prev_node_index = V_explored(I_prev_node_index,...
            V_current_explored_node_index);
         while ((V_segment_overlap ==0) &&  (V_prev_node_index > 1))
            V_execute_time = V_explored(I_time2node_elec,V_prev_node_index);
            V_segment_overlap =...
              bitand(V_explored(I_cmd2node,V_prev_node_index),V_current_cmd);
            V_prev_node_index = V_explored(I_prev_node_index,V_prev_node_index);
         end
         if V_execute_time <...
              V_explored(I_time2node_elec,V_current_explored_node_index)
            V_execute_time =....
              V_explored(I_time2node_elec,V_current_explored_node_index);
         end
      end
      V_time2node = V_execute_time + COST.time(I_cold,V_current_cmd_index);
      % Complete V_overlap
      if V_time2node <= V_explored(I_time2node_elec,V_current_explored_node_index)
         V_cost_of_edge = COST.w_energy(I_cold,V_current_cmd_index) +  0 ;
         V_test_cost =...
            V_explored(I_accum_path_cost,V_current_explored_node_index)...
            + COST.w_energy(I_cold,V_current_cmd_index) +  0 ;
         % No V_overlap
      elseif V_execute_time >=...
            V_explored(I_time2node_elec,V_current_explored_node_index)
         V_cost_of_edge = COST.w_energy(I_cold,V_current_cmd_index)...
            + COST.w_time(I_cold,V_current_cmd_index) ;
         V_test_cost =...
            V_explored(I_accum_path_cost,V_current_explored_node_index)...
            + COST.w_energy(I_cold,V_current_cmd_index)...
```

```
            + COST.w_time(I_cold,V_current_cmd_index) ;
         % partial V_overlap
      else
         V_cost_of_edge = COST.w_energy(I_cold,V_current_cmd_index)...
            + (V_time2node-V_explored(I_time2node_elec,...
            V_current_explored_node_index))...
            / COST.time(I_cold,V_current_cmd_index)...
            *COST.w_time(I_cold,V_current_cmd_index);
         V_test_cost =...
            V_explored(I_accum_path_cost,V_current_explored_node_index)...
            + COST.w_energy(I_cold,V_current_cmd_index)...
            + (V_time2node...
            -V_explored(I_time2node_elec,V_current_explored_node_index))...
            / COST.time(I_cold,V_current_cmd_index)...
            *COST.w_time(I_cold,V_current_cmd_index);
      end
   else
      V_execute_time =...
         max(V_explored(I_time2node_fluid:I_time2node_elec,...
         V_current_explored_node_index));
      V_time2node = V_execute_time + COST.time(I_cold,V_current_cmd_index);
      V_test_cost = V_explored(I_accum_path_cost,V_current_explored_node_index)...
         + COST.w_cost(I_cold,V_current_cmd_index);
      V_cost_of_edge = COST.w_cost(I_cold,V_current_cmd_index);
   end
end %%%%% close nf_calculate_cold_cost


function nf_update
   V_explored(I_accum_path_cost,V_explored_neighbor_index) = V_test_cost;
   V_explored(I_edge_cost,V_explored_neighbor_index) = V_cost_of_edge;
   V_explored(I_prev_node,V_explored_neighbor_index) = V_current_node;
   V_explored(I_prev_node_index,V_explored_neighbor_index) =...
     V_current_explored_node_index;
   V_explored(I_estimated_total_cost,V_explored_neighbor_index) =...
     V_explored(I_accum_path_cost,V_explored_neighbor_index)...
     +  V_explored(I_cost2goal,V_explored_neighbor_index);
   V_explored(I_cmd_execute_time,V_explored_neighbor_index) = V_execute_time;
   V_explored(I_cmd2node,V_explored_neighbor_index) = V_current_cmd;
   V_explored(I_cmd2node_index,V_explored_neighbor_index) = V_current_cmd_index;
   V_explored(I_cmdtype2node,V_explored_neighbor_index) = V_cmd_type;
   switch V_cmd_type
     case I_hot
        % no action taken in I_electric domain, therefore no time added
        V_explored(I_time2node_fluid,V_explored_neighbor_index) = V_time2node;
        V_explored(I_time2node_elec,V_explored_neighbor_index) =...
           V_explored(I_time2node_elec,V_current_explored_node_index);
        V_explored(I_complete_hot,V_explored_neighbor_index) =...
           bitor(V_explored(I_complete_hot,V_current_explored_node_index),...
           V_current_cmd);
        V_explored(I_complete_elec,V_explored_neighbor_index) =...
           V_explored(I_complete_elec,V_current_explored_node_index);
        V_explored(I_complete_cold,V_explored_neighbor_index) =...
           V_explored(I_complete_cold,V_current_explored_node_index);
     case I_elec
        % no action taken in I_electric domain, therefore no time added
        V_explored(I_time2node_fluid,V_explored_neighbor_index) =...
           V_explored(I_time2node_fluid,V_current_explored_node_index);
        V_explored(I_time2node_elec,V_explored_neighbor_index) = V_time2node;
        V_explored(I_complete_hot,V_explored_neighbor_index) =...
           V_explored(I_complete_hot,V_current_explored_node_index);
        V_explored(I_complete_elec,V_explored_neighbor_index) =...
           bitor(V_explored(I_complete_elec,V_current_explored_node_index),...
           V_current_cmd);
        V_explored(I_complete_cold,V_explored_neighbor_index) =...
           V_explored(I_complete_cold,V_current_explored_node_index);
```

```
        case I_cold
           % no action taken in I_electric domain, therefore no time added
           V_explored(I_time2node_fluid,V_explored_neighbor_index) = V_time2node;
           V_explored(I_time2node_elec,V_explored_neighbor_index) =...
              V_explored(I_time2node_elec,V_current_explored_node_index);
           V_explored(I_complete_hot,V_explored_neighbor_index) =...
              V_explored(I_complete_hot,V_current_explored_node_index);
           V_explored(I_complete_elec,V_explored_neighbor_index) =...
              V_explored(I_complete_elec,V_current_explored_node_index);
           V_explored(I_complete_cold,V_explored_neighbor_index) =...
              bitor(V_explored(I_complete_cold,V_current_explored_node_index),...
              V_current_cmd);
      end
   end %%%%% nf_update


   function h = nf_weighted_cost2goal(I_node2evaluate) %Heuristic
      % this identifies the actuators that need to change states (the absolute
V_error)
      dx = double(bitxor(I_node2evaluate,DESTINATION));
      [~,e]=log2(max(dx));
      bits2flip =sum(rem(floor(dx*pow2(1-max(1,e):0)),2));
      h = WH*COST.w_h(bits2flip+1);
      h = WH*bits2flip;
      h = COST.w_h(bits2flip+1);
   end


   function nf_expand_wavefront
      % Heat nf_expand_wavefront
      V_neighbors2current_heat = bitor(V_current_node,V_sub_h_cmd);
      if ~isempty(V_neighbors2current_heat)
        [V_neighbors2current_heat,index_unique_V_neighbors_h] =...
           unique(V_neighbors2current_heat,'first');
        index_unique_V_neighbors_h(V_neighbors2current_heat == V_current_node) = [];
        V_neighbors2current_heat(V_neighbors2current_heat == V_current_node) = [];
        V_edges2neighbors_heat_index = V_sub_h_index(index_unique_V_neighbors_h);

      else% isempty(V_neighbors2current_heat)
        V_neighbors2current_heat = bitor(V_current_node,V_sub_h_cmd_not);
        [V_neighbors2current_heat,index_unique_V_neighbors_h] =...
           unique(V_neighbors2current_heat,'first');
        index_unique_V_neighbors_h(V_neighbors2current_heat == V_current_node) = [];
        V_neighbors2current_heat(V_neighbors2current_heat == V_current_node) = [];
        V_edges2neighbors_heat_index =
V_sub_h_index_not(index_unique_V_neighbors_h);
      end

      % cool nf_expand_wavefront
      V_neighbors2current_cold =...
        bitcmp(bitor(bitcmp(V_current_node,N2),V_sub_c_cmd),N2);
      if ~isempty(V_neighbors2current_cold);
        [V_neighbors2current_cold,index_unique_V_neighbors_c] =...
           unique(V_neighbors2current_cold,'first');
        index_unique_V_neighbors_c(V_neighbors2current_cold == V_current_node) = [];
        V_neighbors2current_cold(V_neighbors2current_cold == V_current_node) = [];
        V_edges2neighbors_cool_index = V_sub_c_index(index_unique_V_neighbors_c);
      else% isempty(V_neighbors2current_cold)
        V_neighbors2current_cold = bitcmp(bitor(bitcmp(V_current_node,N2),...
           V_sub_c_cmd_not),N2);
        [V_neighbors2current_cold,index_unique_V_neighbors_c] =...
           unique(V_neighbors2current_cold,'first');
        index_unique_V_neighbors_c(V_neighbors2current_cold == V_current_node) = [];
        V_neighbors2current_cold(V_neighbors2current_cold == V_current_node) = [];
        V_edges2neighbors_cool_index =
V_sub_c_index_not(index_unique_V_neighbors_c);
```

```
      end
   end %%%%% nf_expand_wavefront


   function nf_is_in_explored
      if isempty(V_explored_neighbor_index)
        V_active_set = [V_active_set V_neighbor];
        V_explored_size = V_explored_size+1;
        V_explored(I_node,V_explored_size) = V_neighbor;
        V_explored(I_node_index,V_explored_size) = V_explored_size;
        V_explored(I_prev_node,V_explored_size) = V_current_node;
        V_openset_index = [V_openset_index V_explored_size];
        V_explored(I_accum_path_cost,V_explored_size)= inf;
        V_explored(I_cost2goal,V_explored_size) =
nf_weighted_cost2goal(V_neighbor);
        V_explored_neighbor_index = V_explored_size;
      end
   end


   function nf_move_open2closed
      remove_index = (V_openset_index == V_current_explored_node_index);
      V_openset_index(remove_index) = [];
   end %%%%% nf_move_open2closed


   function nf_submatrix_reduced
      if REDUCE
        V_error = bitxor(ORIGIN,DESTINATION);
        V_error_h = bitand(DESTINATION,V_error);
        V_error_c = bitand(ORIGIN,V_error);

        [sub_h, V_sub_h_index, ~] = unique(bitand(V_error_h,SUBMATRIX.dec),'first');
        V_sub_h_index(sub_h == 0) =[];
        V_sub_h_cmd = SUBMATRIX.dec(V_sub_h_index);

        V_sub_h_index_not = 1:N_CMDS;
        V_sub_h_index_not(V_sub_h_index) = [];
        V_sub_h_cmd_not = SUBMATRIX.dec;
        V_sub_h_cmd_not(V_sub_h_index) = [];

        [sub_c, V_sub_c_index, ~] = unique(bitand(V_error_c,SUBMATRIX.dec),'first');
        V_sub_c_index(sub_c == 0) =[];
        V_sub_c_cmd = SUBMATRIX.dec(V_sub_c_index);

        V_sub_c_index_not = 1:N_CMDS;
        V_sub_c_index_not(V_sub_c_index) = [];
        V_sub_c_cmd_not = SUBMATRIX.dec;
        V_sub_c_cmd_not(V_sub_c_index) = [];

      else %full control command list
        V_sub_h_cmd = SUBMATRIX.dec;
        V_sub_c_cmd = SUBMATRIX.dec;
        V_sub_h_index = 1:N_CMDS;
        V_sub_c_index = 1:N_CMDS;
      end
   end %%%%% nf_submatrix_reduced


   function nf_create_directions
      V_backnode_index = V_current_explored_node_index;
      V_output.I_nodes = [];
      V_output.cmd =[];
      V_output.type = [];
      V_output.energy = [];
      V_output.time2 = [];
```

```
      V_output.timeE = [];

      while V_backnode_index ~= 1
        V_output.I_nodes = [V_explored(I_node,V_backnode_index),V_output.I_nodes];
        V_output.cmd = [V_explored(I_cmd2node,V_backnode_index),V_output.cmd];
        V_output.type = [V_explored(I_cmdtype2node,V_backnode_index),V_output.type];
        V_output.time2 =...
           [V_explored(I_time2node_fluid:I_time2node_elec,V_backnode_index),...
           V_output.time2];
        V_output.timeE =...
           [V_explored(I_cmd_execute_time,V_backnode_index),V_output.timeE];
        if V_explored(I_cmdtype2node,V_backnode_index) == I_hot
           V_output.energy =...
              [[COST.energy(I_hot,...
              V_explored(I_cmd2node_index,V_backnode_index));0;0],V_output.energy];
        elseif V_explored(I_cmdtype2node,V_backnode_index) == I_elec
           V_output.energy =...
              [[0;COST.energy(I_elec,...
              V_explored(I_cmd2node_index,V_backnode_index));0],V_output.energy];
        else
           V_output.energy =...
              [[0;0;COST.energy(I_cold,...
              V_explored(I_cmd2node_index,V_backnode_index))],V_output.energy];
        end
        V_backnode_index = V_explored(I_prev_node_index,V_backnode_index);
      end
      V_output.I_nodes =[ORIGIN,V_output.I_nodes ];
      V_output.V_explored_size = V_explored_size;
      V_output.n_sub_cmd = size(V_sub_h_cmd,1) + size(V_sub_c_cmd,1);
      V_output.cost = V_part_explored(I_accum_path_cost,V_min_est_index_temp);
   end %%%%% nf_create_directions
end %% close of astar_naa
```

A.2 build_n_submatrix.m

```
function submatrix=build_N_submatrix(size_array)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This MATLAB function creates a mat file with a list the NAA control
% commands or an NxN array. size_array can be a vector of integers

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 for N= size_array; %N is the size of the array
   N2 = N^2;  %N squared
   n_nodes = 2^N2; % Number of nodes
   n_cmds = (2^N-1)^2; % Number of NAA CMDS
   s_list_index=0;  % This is a counter for the submatrix list
   submatrix_1d_list = false(n_cmds,N2); %a N^2 x (2^N-1)^2 array that
   submatrix_flow_list = zeros(1,N2);
   for ii=1:N
      % create a list of all combination with ii out of N row switches on
      row = combntns(1:N,ii);
      for jj=1:N
        % create a list of all combination with jj out of N column switches on
        col=combntns(1:N,jj);
        for rr=1:size(row,1)
           for cc=1:size(col,1)
              s_list_index=1+s_list_index; %index
              smatrix_2d = zeros(N,N); %reset all elements to zero
              % set all intersecting elements of on switches to
              % on(1,flow)
              smatrix_2d(row(rr,:),col(cc,:)) = 1;
              submatrix_1d = reshape(smatrix_2d,1,N2);
              submatrix_1d_list(s_list_index,:)=submatrix_1d;
              rtable(row(rr,:),s_list_index)=1;
```

```
            ctable(s_list_index,col(cc,:))=1;
            submatrix_flow_size(s_list_index)=sum(submatrix_1d);
         end
      end
   end
end

% Sort the submatrix based on size of control command
[submatrix_flow_list,submatrix_flow_size_sorted_index] =...
   sort(submatrix_flow_size,'ascend');
sorted_smatrix_1d_list = submatrix_1d_list(submatrix_flow_size_sorted_index,:);
rtable=rtable(:,submatrix_flow_size_sorted_index);
ctable=ctable(submatrix_flow_size_sorted_index,:);

submatrix.bin = sorted_smatrix_1d_list;
submatrix.rtable.bin = rtable;
submatrix.ctable.bin = ctable;
submatrix.smod = submatrix_flow_size(submatrix_flow_size_sorted_index);

% Convert to submatrix from binary from to decimal form
% It is easier to visually identify identical binary number in decimal form.

%%%% Convert to submatrix.bin to decimal form
% Convert submatrix.bin to string
submatrix_bin_string = num2str(submatrix.bin);

% Remove spaces from string
submatrix_bin_string =
submatrix_bin_string(:,[1:3:size(submatrix_bin_string,2)]);

% Convert submatrix_bin_string to decminal form.  It is easier to identify
% identical binary number in decimal form.
submatrix.dec = bin2dec(num2str(submatrix_bin_string));

rtable_string = num2str(rtable');
rtable_string = rtable_string(:,[1:3:size(rtable_string,2)]');
submatrix.rtable.dec = bin2dec(num2str(rtable_string));

ctable_string = num2str(ctable);
ctable_string = ctable_string(:,[1:3:size(ctable_string,2)]');
submatrix.ctable.dec = bin2dec(num2str(ctable_string));
submatrix.index = 1:n_cmds;

% This data is used to for the series algorithm
single_col_combinations = 1:(2^N-1);
size_cmd_col_combinations = zeros(1,(2^N-1));
for ii = 1:2^N-1
   % this identifies the actuators that need to change states (the
   % absolute error)
   dx = double(single_col_combinations(ii));
   [~,e]=log2(max(dx));
   size_cmd_col_combinations(ii) =sum(rem(floor(dx*pow2(1-max(1,e):0)),2));
end
[size_cmd, size_ind] = sort(size_cmd_col_combinations);

single_all_columns.dec = zeros(N,(2^N-1));
for jj = 1:N
   single_all_columns.dec(N-jj+1,:) = bitshift(size_ind,N*(jj-1));
end

single_all_columns.sumbmatrix_dec_index = zeros(size(single_all_columns));
for ii = 1:size(single_all_columns.dec,1)
   for jj = 1:size(single_all_columns.dec,2)
     single_all_columns.sumbmatrix_dec_index(ii,jj) =...
        find(single_all_columns.dec(ii,jj) == submatrix.dec);
   end
```

```
    end

    single_all_columns.smod = ones(N,1)*size_cmd;
    single_all_columns.rtable.dec = ones(N,1)*single_col_combinations;
    single_all_columns.ctable.dec = [1:N]'*ones(1,(2^N-1));

    % Delete all variables that are temporarly used
    clear c col ctable ctable_string dx e ii jj r row rtable rtable_string
    clear s_list_index single_col_combinations size_cmd
    clear size_cmd_col_combinations size_ind smatrix_2d
    clear sorted_smatrix_1d_list submatrix_1d submatrix_1d_list
    clear submatrix_bin_string submatrix_flow_list submatrix_flow_size
    clear submatrix_flow_size_sorted_index

    % Save variables as a mat file
    eval(['save submatrix_N',num2str(N),'_07_2011']);
end
```

A.3 run_Astar.m

```
clear all
clear fun
close all
clc

global  HOT_INDEX ELEC_INDEX COLD_INDEX N2 N SUBMATRIX N_CMDS
HOT_INDEX = 1;
ELEC_INDEX = 2;
COLD_INDEX = 3;

for N = 3 % size of the array
    N2 = N^2; %number of actuators in array
    N_CMDS = (2^N-1)^2;  % number of control commands


    % Load a 2x10000 list of randomly generated nodes, between 0-2^N2, to
    % be use orgin and destinations for repeatability
    eval(['load origin_destination_n',num2str(N),'.mat'])


    % Load the NAA control commands for use in algorithms
    eval(['load submatrix_N',num2str(N),'_07_2011'])
    SUBMATRIX =submatrix;


    % time for contraction using hot_fluid
    time = 1*[0  1.0000   1.5876  1.9744  2.2482  2.4522  2.6102  2.7360...
           2.8387  2.9241  2.9961  3.0578  3.1111  3.1577  3.1988...
           3.2353   3.2679];
    time=[time ones(1,25)*time(end)];


    % time for contraction using electricity
    time(ELEC_INDEX,:) = .25*time(1,:);


    % time for extention using cold fluid
    time(COLD_INDEX,:) = 1*time(1,:);


    % energy for contraction using hot fluid
    energy = 1*submatrix.smod;
```

```
    % energy for contraction using electricity
    energy(ELEC_INDEX,:) = 3*energy(HOT_INDEX,:);


    % energy for extention using cold fluid
    energy(COLD_INDEX,:) = 1*energy(HOT_INDEX,:);


    % reshape time and erergy to work in Astar algorithm
    time_f_cmd = build_time_index(time,submatrix.smod);
    energy_f_cmd = energy ;


    % initilize the parameters of how Astar will operate.
    reduce = 1;
    sim = 1;


    % Heuristic weighting, defines the performance of the Astar Algorithm,
    % w_h = 0 is Dikjstra , w_h = if is Best First Search.  w_h = 1 should
    % be make the heuristic value admissiable
    w_h_array =[1.5];


    % The performance weighting to define performance of wet SMA array.
    % w_te = 0 minimizes energy, w_te = 1 minimizes time
    w_te_array = [.99];


    % selection determine what elements of the origin-destination array for
    % simulations
    selection = 1:10;


    % This section allows for multiple values of w_h and w_te, with
    % multiple origin
    for kk = 1:length(w_h_array)
       for jj = 1:length(w_te_array)
         w1 =w_te_array(jj)*ones(1,N2);
         cost = build_cost_index(time_f_cmd,energy_f_cmd,w1);
         for ii = selection  %reduced
            start = origin_destination(ii,1);
            destination = origin_destination(ii,2);
            [output_path] =...
               astar_naa(start,destination,w_h_array(kk),cost,sim,reduce)
         end
       end

    end
end


% output_path =
%          % Nodes that the path will take starting from start to destination
%          nodes: [4850127 22693327 31344079 31344607 30689247 30689246]
%          % Control commands that will take the array through nodes
%           cmd1: [17843200 8650752 528 655360 1]
%          % Type of control commands (1-Hot, 2-Elec, 3-cold)
%           type: [1 1 1 3 3]
%          % Energy used from each control command
%          energy: [3x5 double]
%          % Operation time for each control command for the fluidic
%          time2: [2x5 double]
%          % When the control commands are executed
%          timeE: [0 2.2482 3.8358 5.4234 7.0110]
%          % Number of nodes explored
```

```
%          V_explored_size: 163
%          % Number of edges explored
%          count: 195
%          % Total weighted cost
%           cost: 11
%          % Number of reduced control commands
%          n_sub_cmd: 88
```

A.4 build_cost_index.m

```
function cost = build_cost_index(time_f_cmd,energy_f_cmd,w1_1d)
global  HOT_INDEX ELEC_INDEX COLD_INDEX N2  SUBMATRIX N_CMDS

x = ones(N_CMDS,1);
w1_2d = x*w1_1d;


% the complimentary for w1
w1_2d_comp=1-w1_2d;

weighted_t = dot(w1_2d',SUBMATRIX.bin')./SUBMATRIX.smod;
weighted_e = dot(w1_2d_comp',SUBMATRIX.bin')./SUBMATRIX.smod;


%%%% This is no longer used in simultaneous operations
% %the weighted cmd time
% % weighted time for hot fluid
weighted_time(HOT_INDEX,:) = weighted_t.*time_f_cmd(HOT_INDEX,:);


% % weighted time for cold fluid
weighted_time(ELEC_INDEX,:) = weighted_t.*time_f_cmd(ELEC_INDEX,:);


% % weighted time for electricity
weighted_time(COLD_INDEX,:) = weighted_t.*time_f_cmd(COLD_INDEX,:);


%the weighted cmd energy
% weighted energy for hot fluid
% weighted_energy = 3,n_nodes-1);
weighted_energy(HOT_INDEX,:) = weighted_e.*energy_f_cmd(HOT_INDEX,:);


% weighted energy for cold fluid
weighted_energy(ELEC_INDEX,:) = weighted_e.*energy_f_cmd(ELEC_INDEX,:);


% weighted energy for electricity
weighted_energy(COLD_INDEX,:) = weighted_e.*energy_f_cmd(COLD_INDEX,:);


% format into desired form at
cost.time = time_f_cmd;
cost.w_time = weighted_time;
cost.energy = energy_f_cmd;
cost.w_energy = weighted_energy;
cost.w_cost = cost.w_time+cost.w_energy;
cost.cost = cost.time+cost.energy ;


%zero is added to allow for the destination (h =0) to be identified in
%Astar_heuritic cost.h = cost.w_cost(h+1)
cost.w_h = [0 spline(unique(SUBMATRIX.smod),unique(min(cost.w_cost)),1:N2)];
```

```
cost.weighted_t = weighted_t;
cost.weighted_e = weighted_e;
```

A.5. build_time_index.m

```
function cmd_time = build_time_index(time,smod)
% reshapes the time variable from [hot time; cold time; electric time] as
% function of smod to a 1d [hot time, cold time, electric time] to a
% function of cmd
cmd_time =time(:,smod+1);
```

A.6 build_origin_destination

```
function origin_destination = build_origin_destination(size_array,num_nodes)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This MATLAB function creates a mat file with a list of origin and
% and destination nodes.  This set of nodes is used as a data set for repeatability
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for N= size_array;
    N2 = N^2
    origin_destination = floor((2^N2)*rand(num_nodes,2));
    eval(['save origin_destination_N',num2str(N)']);
end
```

A.7 combntns.m

```
function out=combntns(choicevec,choose);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This MATLAB function was downloaded from Mathworks Central Exchange Website.
% This function is not built into MATLAB.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%COMBNTNS  Computes all combinations of a given set of values
%
%  c = COMBNTNS(choicevec,choose) returns all combinations of the
%  values of the input choice vector.  The size of the combinations
%  are given by the second input.  For example, if choicevec
%  is [1 2 3 4 5], and choose is 2, the output is a matrix
%  containing all distinct pairs of the choicevec set.
%  The output matrix has "choose" columns and the combinatorial
%  "length(choicevec)-choose-'choose'" rows.  The function does not
%  account for repeated values, treating each entry as distinct.
%  As in all combinatorial counting, an entry is not paired with
%  itself, and changed order does not constitute a new pairing.
%  This function is recursive.

%  Copyright 1996-2003 The MathWorks, Inc.
%  Written by:  E. Brown, E. Byrns
%   $Revision: 1.11.4.1 $    $Date: 2003/08/01 18:15:39 $

if nargin ~= 2;  error('Incorrect number of arguments');  end

%  Input dimension tests

if min(size(choicevec)) ~= 1 | ndims(choicevec) > 2
    error('Input choices must be a vector')

elseif max(size(choose)) ~= 1
    error('Input choose must be a scalar')
```

```
else
    choicevec = choicevec(:);       %  Enforce a column vector
end

%  Ensure real inputs

if any([~isreal(choicevec) ~isreal(choose)])
    warning('Imaginary parts of complex arguments ignored')
    choicevec = real(choicevec);    choose = real(choose);
end

%  Cannot choose more than are available

choices=length(choicevec);
if choices<choose(1)
    error('Not enough choices to choose that many')
end


%  Choose(1) ensures that a scalar is used.  To test the
%  size of choices upon input results in systems errors on
%  the Macintosh.  Maybe somehow related to recursive nature of program.

%  If the number of choices and the number to choose
%  are the same, choicevec is the only output.

if choices==choose(1)
    out=choicevec';

%  If being chosen one at a time, return each element of
%  choicevec as its own row

elseif choose(1)==1
    out=choicevec;

%  Otherwise, recur down to the level at which one such
%  condition is met, and pack up the output as you come out of
%  recursion.

else
    out = [];
    for i=1:choices-choose(1)+1
        tempout=combntns(choicevec(i+1:choices),choose(1)-1);
        out=[out; choicevec(i)*ones(size(tempout,1),1)  tempout];
    end
end
```