

MORPHOLOGY FOR PLANAR HEXAGONAL
MODULAR SELF-RECONFIGURABLE ROBOTIC SYSTEMS

A THESIS IN MECHATRONICS

Presented to the faculty of the American University of Sharjah
School of Engineering
in partial fulfillment of
the requirement of the degree

MASTER OF SCIENCE

by

HOSSEIN SADJADI

Sharjah, U.A.E

March 2009

© 2009

HOSSEIN SADJADI

ALL RIGHTS RESERVED

We approve the thesis of Hossein Sadjadi	Date of signature
<hr/> Dr. Mohammad Amin Al-Jarrah Professor, Mechanical Engineering Thesis Advisor	<hr/>
<hr/> Dr. Khaled Assaleh Associate Professor, Electrical Engineering Thesis Co-Advisor	<hr/>
<hr/> Dr. Taha Landolsi Assistant Professor, Computer Science and Engineering Graduate Committee	<hr/>
<hr/> Dr. Aydin Yesildirek Associate Professor, Electrical Engineering Graduate Committee	<hr/>
<hr/> Dr. Rached Dhaouadi Associate Professor, Electrical Engineering Coordinator, Mechatronics Graduate Program	<hr/>
<hr/> Dr. Hany El-Kadi Associate Dean, College of Engineering	<hr/>
<hr/> Dr. Yousef Al-Assaf Dean, College of Engineering	<hr/>
<hr/> Mr. Kevin Lewis Mitchell Director, Graduate & Undergraduate Programs	<hr/>

MORPHOLOGY FOR PLANAR HEXAGONAL MODULAR SELF-RECONFIGURABLE ROBOTIC SYSTEMS

Hossein Sadjadi, Candidate for the Master of Science Degree

American University of Sharjah, 2009

ABSTRACT

This thesis primarily covers the design and implementation of a planar hexagonal Modular Self-Reconfigurable Robotic System (MSRRS) along with the construction of its reconfiguration path planner and control algorithm. Both platform and algorithm are designed based on a multilayer approach where each layer is dedicated to perform a specific task; in other words, the design itself is considered to be modular. In the first part a universal module is carefully designed to maintain certain criteria that seem to be in line with the common goals of this promising field including homogeneity, cost-effectiveness, fast actuation and quick and strong connections. In the second part, a reconfiguration path planner and a control algorithm is developed to determine the required sequence of individual module movements that transforms the shape of the system from an arbitrary initial configuration to a desired goal configuration in an optimal manner while enforcing several constraints and taking into account the kinematic model of the system.

The shape of the physical platform was inspired from natural structures such as bees' nest and crystal molecules, where homogeneous hexagonal modules are capable of forming variety of structures. Electromagnets installed on six sides serve as the required actuating force providing fast and cost effective motion for the module. In this case, each module is not able to perform any motion alone; however, a combination of two or more modules makes the motion possible. Moreover, pull type solenoids located on six corners of the module provide quick and strong inter-module connections. Although the imple-

mented working prototype is both large and restricted to a planar geometry, it is designed such that its hardware and software can be scaled up in the number of units and down in unit size; similarly, the platform has the potential to be extended for 3D applications. The software infrastructure of this platform is designed in a way that different hierarchies for distributed control and communication can be implemented.

The path planner is designed to minimize the number of module movements during reconfiguration while enforcing collision avoidance and connectivity constraints. The algorithm is based on a hierarchical multilayer approach, where upper layers decompose the problem into sub-problems solvable by lower layers. The core of the algorithm relies on a heuristic function and a Markov Decision Process (MDP) optimization to generate a centralized near-optimal reconfiguration path planner and a control algorithm for a lattice, homogenous, rigid, planar hexagonal MSRRS. In this approach the connectivity test and MDP formulation require a centralized stage, yet the scalability issues required to move towards a truly decentralized approach are discussed in this thesis as well.

Among several novel approaches incorporated in this system, multilayer nature of both hardware and software design provides openness, flexibility and ease of modification or adaptation for other platforms. In this approach each layer is dedicated to perform a specific task and can be modified or enhanced separately while keeping the remaining layers untouched.

Thesis Supervisor: Dr. Mohammad Ameen Al-Jarrah

Professor and Department Head

Mechanical Engineering Program, AUS

Thesis Supervisor: Dr. Khaled Assaleh

Associate Professor

Electrical Engineering Program, AUS

Contents

ABSTRACT	III
LIST OF FIGURES	XI
LIST OF TABLES	XV
ACKNOWLEDGMENT	XVI
1 INTRODUCTION	1
1.1 PROBLEM STATEMENT	1
1.2 BACKGROUND	2
1.2.1 <i>Modular Self-Reconfigurable Robotic Systems (MSRRSs)</i>	2
1.2.2 <i>Research Motivation and Challenges</i>	5
1.3 CONTRIBUTION	12
1.4 THESIS OUTLINE	13
2 RELATED WORK	14
2.1 HISTORY	15
2.2 CHOICES FOR PHYSICAL PLATFORM	15
2.2.1 <i>Architectural Topology</i>	15
2.2.2 <i>Homogeneity</i>	18
2.2.3 <i>Rigidity</i>	19
2.2.4 <i>Shape</i>	20
2.3 RANGE OF CONTROL ALGORITHMS	22
2.3.1 <i>Reconfiguration vs. Locomotion</i>	22
2.3.2 <i>Deterministic vs. Stochastic</i>	24

2.3.3	<i>Optimal vs. Near Optimal</i>	25
2.3.4	<i>Centralized vs. Distributed</i>	25
2.3.5	<i>Serial vs. Parallel</i>	27
2.4	MODULAR SELF-RECONFIGURING ROBOTS	27
2.4.1	<i>ATRON</i>	27
2.4.2	<i>Catom</i>	28
2.4.3	<i>CEBOT</i>	29
2.4.4	<i>CONRO</i>	30
2.4.5	<i>Crystalline</i>	30
2.4.6	<i>Fracta / 3D Fracta</i>	31
2.4.7	<i>I-Cubes</i>	32
2.4.8	<i>Metamorphic</i>	32
2.4.9	<i>Miche</i>	33
2.4.10	<i>Micro Unit</i>	33
2.4.11	<i>Molecube</i>	34
2.4.12	<i>Molecule</i>	35
2.4.13	<i>MTRAN</i>	35
2.4.14	<i>Polypod / PolyBot</i>	35
2.4.15	<i>Programmable Parts</i>	37
2.4.16	<i>RIKEN Vertical</i>	38
2.4.17	<i>Stochastic (2D/3D)</i>	38
2.4.18	<i>SuperBot</i>	39
2.4.19	<i>Telecube</i>	39
3	HEXBOT: PHYSICAL PLATFORM	41
3.1	DESIGN CRITERIA	41
3.2	MECHANICAL DESIGN	42
3.2.1	<i>Universal Module</i>	42
3.2.2	<i>Actuators</i>	44
3.2.3	<i>Inter-Module Connections</i>	45
3.2.4	<i>Motion through Rotation</i>	46
3.3	ELECTRICAL SYSTEM	50
3.3.1	<i>Design</i>	51
3.3.2	<i>Power Base</i>	54
3.3.3	<i>Layer 1 – Power Connection & Mechanical Support</i>	55
3.3.4	<i>Layer 2 – Power Unit</i>	57

3.3.5	<i>Layer 3 – Drive Circuit</i>	58
3.3.6	<i>Layer 4 – Control Board</i>	60
3.3.7	<i>Layer 5 – Communication</i>	60
3.4	PROCESSING UNIT.....	63
3.4.1	<i>Microcontrollers</i>	63
3.4.2	<i>Internal Module Connections</i>	67
3.4.3	<i>IR Transceivers</i>	68
3.4.4	<i>Handshaking</i>	69
3.4.5	<i>Testing and Debugging</i>	70
3.4.6	<i>Reprogramming</i>	70
3.5	GRAPHICAL USER INTERFACE.....	71
3.5.1	<i>Communication with the MSRRS</i>	71
3.5.2	<i>Interface</i>	73
3.5.3	<i>Functions</i>	74
3.6	SUMMARY.....	78
4	RECONFIGURATION PLANNING	79
4.1	PRELIMINARIES.....	80
4.1.1	<i>Environment</i>	80
4.1.2	<i>Reinforcement Learning</i>	81
4.1.3	<i>Markov Property</i>	82
4.1.4	<i>Markov Decision Process (MDP)</i>	83
4.1.5	<i>Partially Observable MDP (POMDP)</i>	84
4.1.6	<i>Multi-Agent MDP (MMDP)</i>	85
4.2	CONSTRAINTS AND ASSUMPTIONS.....	85
4.2.1	<i>Constraints</i>	86
4.2.2	<i>Markov Assumptions</i>	87
4.2.3	<i>Kinematic Model Assumptions</i>	88
4.2.4	<i>Initial and Goal Configuration Assumptions</i>	88
4.3	PROBLEM FORMULATION.....	89
4.3.1	<i>Reference Frame and Coordinate system</i>	90
4.3.2	<i>MSRRS Representation</i>	91
4.3.3	<i>Hierarchical Multilayer Approach</i>	92
4.3.4	<i>Layer 1 – Obtain Initial and Goal States</i>	94
4.3.5	<i>Layer 2 – Potential Voids and Mobile Electrons</i>	98
4.3.6	<i>Layer 3 – Void Propagation</i>	105

4.3.7	<i>Layer 4 – Mobile Electron Path Planning</i>	106
4.3.8	<i>Layer 5 – Mobile Electron Motion</i>	117
4.3.9	<i>Simulation</i>	119
4.4	KEY ISSUES.....	119
4.4.1	<i>Scalability</i>	119
4.4.2	<i>Energy Consumption</i>	120
4.4.3	<i>Discussion</i>	121
4.5	SUMMARY	122
5	EVALUATION	123
5.1	EXPERIMENTAL SETUP.....	123
5.2	EXAMPLES AND EXPERIMENTS	124
5.2.1	<i>Physical Platform Performance</i>	124
5.2.2	<i>Reconfiguration Algorithm Performance</i>	128
5.3	DISCUSSIONS.....	140
5.3.1	<i>HexBot Evaluation Criteria and Performance</i>	141
6	CONCLUDING REMARKS.....	143
6.1	SUMMARY	143
6.2	CONCLUSIONS	144
6.3	LIMITATIONS AND DIRECTIONS FOR FUTURE RESEARCH.....	144
	BIBLIOGRAPHY	146
	APPENDICES.....	161
A.	MICROCONTROLLER CODES	161
	<i>Control Board Program</i>	162
	<i>Communication Board Program</i>	168
	<i>Wireless SPI Code</i>	179
B.	VISUAL BASIC CODE	184
	<i>Port Configuration</i>	185
	<i>Main Window</i>	186
C.	MATLAB FUNCTIONS	192
	<i>Function 1: FindVE</i>	193
	<i>Function 2: Hex</i>	193
	<i>Function 3: HexActuate</i>	194
	<i>Function 4: HexAll_P</i>	196

Function 5:	HexAll_S	197
Function 6:	HexCCW.....	198
Function 7:	HexCG.....	198
Function 8:	HexCGT.....	199
Function 9:	HexCW.....	200
Function 10:	Hexgrid	200
Function 11:	HexIMS	201
Function 12:	Hexmove.....	201
Function 13:	HexNe	203
Function 14:	HexNeM.....	203
Function 15:	Hexp	203
Function 16:	Hexplot.....	204
Function 17:	Hexpm	205
Function 18:	HexVP	205
Function 19:	HexVPT.....	206
Function 20:	HexZP.....	208
Function 21:	Jointplot.....	208
Function 22:	MDP_Action	208
Function 23:	MDP_NRL	209
Function 24:	MDP_Reward	211
Function 25:	MDP_State	211
Function 26:	MDP_VI	213
Function 27:	Mobile	214
Function 28:	PV.....	214
Function 29:	Sideplot.....	215
Test 1:	TestAll_PO1	216
Test 2:	Test_All_PO2	217
Test 3:	Test_All_S01.....	218
Test 4:	Test_All_S02.....	219
Test 5:	Test_CG.....	220
Test 6:	Test_Constraints1	220
Test 7:	Test_Constraints2	221
Test 8:	Test_Immobile	221
Test 9:	Test_IR_Simulation	222
Test 10:	Test_Layer5.....	222

<i>Test 11:</i>	<i>Test_Localization</i>	222
<i>Test 12:</i>	<i>Test_MDP</i>	223
<i>Test 13:</i>	<i>Test_MDP_Convergence</i>	223
<i>Test 14:</i>	<i>Test_Simulator</i>	224
<i>Test 15:</i>	<i>Test_VP</i>	225
VITA		227

LIST OF FIGURES

FIGURE 1-1 ARTIST'S RENDITION OF A SPACE APPLICATION	4
FIGURE 1-2 SELF-ASSEMBLY AND SELF-REPAIR	6
FIGURE 2-1 BASIC CHARACTERISTICS OF PHYSICAL PLATFORMS	16
FIGURE 2-2 PRIMARILY ARCHITECTURAL TOPOLOGIES	17
FIGURE 2-3 RIGID, DEFORMABLE AND COMPRESSIBLE MODULES	19
FIGURE 2-4 TWO DIMENSIONAL TRIANGULAR, CUBICAL, HEXAGONAL AND CIRCULAR PLATFORMS	21
FIGURE 2-5 THREE DIMENSIONAL CUBICAL, RHOMBIC DODECAHEDRON AND SPHERICAL PLATFORMS	22
FIGURE 2-6 CONTROLLER DESIGN HIERARCHY	23
FIGURE 2-7 ATRON (CHRISTENSEN & STOY, 2006)	28
FIGURE 2-8 CATOM (GOLDSTEIN, MOWRY, GIBBONS, PILLAI, RISTER, & LEE, 2006)	29
FIGURE 2-9 CEBOT (FUKUDA & KAWAKUCHI, 1990)	29
FIGURE 2-10 CONRO (CASTANO, BEHAR, & WILL, 2002)	30
FIGURE 2-11 CRYSTALLINE (RUS & VONA, 2001)	31
FIGURE 2-12 FRACTA (MURATA, KUROKAWA, & KOKAJI, 1994)	31
FIGURE 2-13 3D FRACTA (MURATA, KUROKAWA, YOSHIDA, TOMITA, & KOKAJI, 1998)	31
FIGURE 2-14 I-CUBES (UNSAI & KHOSLA, 2000)	32
FIGURE 2-15 METAMORPHIC (CHIRIKJIAN G. , 1994)	32
FIGURE 2-16 MICHE (GILPIN, KOTAY, & RUS, 2007)	33
FIGURE 2-17 MICRO UNIT (YOSHIDA E. , MURATA, KOKAJI, KAMIMURA, TOMITA, & KUROKAWA, 2002)	34
FIGURE 2-18 MOLECUBE (ZYKOV, MYTILINAIOS, ADAMS, & LIPSON, 2005)	34
FIGURE 2-19 MOLECULE (KOTAY & RUS, 2005)	35
FIGURE 2-20 MTRAN (KAMIMURA A. , KUROKAWA, YOSHIDA, TOMITA, MURATA, & KOLAJI, 2003), (KUROKAWA, KAMIMURA, YOSHIDA, TOMITA, KOKAJI, & MURATA, 2003)	36
FIGURE 2-21 POLYPOD(YIM M. , 1993)	36
FIGURE 2-22 POLYBOT (YIM, DUFF, & ROUFAS, 2000)	37
FIGURE 2-23 PROGRAMMABLE PARTS(KLAVINS, BURDEN, & NAPP, 2006)	37
FIGURE 2-24 RIKEN VERTICAL (HOSOKAWA, ET AL., 1998)	38
FIGURE 2-25 STOCHASTIC (WHITE, KOPANSKI, & LIPSON, 2004), (WHITE, ZYKOV, BONGARD, & LIPSON, 2005)	38
FIGURE 2-26 SUPERBOT (SHEN, KRIVOKON, CHIU, EVERIST, RUBENSTEIN, & VENKATESH, 2006)	39
FIGURE 2-27 TELECUBE (SUH, HOMANS, & YIM, 2002)	40
FIGURE 3-1 NATURAL HEXAGONAL STRUCTURES	43
FIGURE 3-2 HEXBOT STRUCTURE	43
FIGURE 3-3 SELF-ALIGNING MECHANISM	44

FIGURE 3-4 HOLD FORCE VS. INPUT POWER.....	44
FIGURE 3-5 HEXBOT ACTUATORS.....	45
FIGURE 3-6 BALL TRANSFER UNITS.....	45
FIGURE 3-7 INTER-MODULE CONNECTIONS	46
FIGURE 3-8 HEXBOT MOTION	49
FIGURE 3-9 MULTILAYER ELECTRICAL SYSTEM DESIGN FOR HEXBOT	50
FIGURE 3-10 POWER CIRCUIT SCHEMATIC	51
FIGURE 3-11 DRIVE CIRCUIT SCHEMATIC	52
FIGURE 3-12 MICROCONTROLLER DESIGN SCHEMATIC.....	52
FIGURE 3-13 IR COMMUNICATION SCHEMATIC	53
FIGURE 3-14 BREADBOARD VERIFICATION	53
FIGURE 3-15 POWER BASE	54
FIGURE 3-16 GRIDS AND COORDINATE SYSTEM	55
FIGURE 3-17 BASE LAYER DESIGN	55
FIGURE 3-18 MANUFACTURED BASE LAYER	56
FIGURE 3-19 POWER TRANSFER TO THE MODULE	56
FIGURE 3-20 POWER CIRCUIT DESIGN	57
FIGURE 3-21 MANUFACTURED POWER UNIT	58
FIGURE 3-22 DRIVE CIRCUIT DESIGN	59
FIGURE 3-23 MANUFACTURED DRIVE CIRCUIT	59
FIGURE 3-24 CONTROL BOARD DESIGN.....	60
FIGURE 3-25 MANUFACTURED CONTROL BOARD.....	61
FIGURE 3-26 COMMUNICATION BOARD DESIGN	62
FIGURE 3-27 MANUFACTURED COMMUNICATION BOARD	62
FIGURE 3-28 MULTILAYER PCBs	63
FIGURE 3-29 INTERNAL MODULE CONNECTIONS.....	67
FIGURE 3-30 MODULATED IR SIGNAL	68
FIGURE 3-31 SETTING THE PROGRAMMING JUMPER.....	71
FIGURE 3-32 PC-MSRRS COMMUNICATION COMPONENTS	72
FIGURE 3-33 PC-MSRRS COMMUNICATION	72
FIGURE 3-34 GRAPHICAL USER INTERFACE.....	73
FIGURE 3-35 PORT CONFIGURATION WINDOW	74
FIGURE 4-1 REINFORCEMENT LEARNING MODEL.....	82
FIGURE 4-2 SEQUENCE OF ACTIONS IN REINFORCEMENT LEARNING	82
FIGURE 4-3 PARTIALLY OBSERVABLE MDP MODEL	84

FIGURE 4-4 CONNECTIVITY CONSTRAINT.....	87
FIGURE 4-5 COLLISION AVOIDANCE CONSTRAINT	87
FIGURE 4-6 IMMOBILE CONFIGURATION.....	89
FIGURE 4-7 TRANSFORMATION FROM AN INITIAL CONFIGURATION TO A GOAL CONFIGURATION	90
FIGURE 4-8 COORDINATE SYSTEM.....	91
FIGURE 4-9 REFERENCE JOINT AND SIDE NUMBERING	91
FIGURE 4-10 SYSTEM REPRESENTATION	92
FIGURE 4-11 HIERARCHICAL MULTILAYER APPROACH	93
FIGURE 4-12 MSRRS TASK HANDLING.....	94
FIGURE 4-13 LOCALIZATION PROCESS (0).....	95
FIGURE 4-14 LOCALIZATION PROCESS (1).....	96
FIGURE 4-15 LOCALIZATION PROCESS (2, 3)	97
FIGURE 4-16 ELECTRONS AND VOIDS.....	99
FIGURE 4-17 MOBILE ELECTRONS AND POTENTIAL VOIDS.....	100
FIGURE 4-18 EXAMPLE OF ME AND PV	101
FIGURE 4-19 GRAPH REPRESENTATION	102
FIGURE 4-20 CONNECTIVITY GRAPH	102
FIGURE 4-21 ARTICULATING NODES	102
FIGURE 4-22 LAPLACIAN MATRIX REPRESENTATION	104
FIGURE 4-23 EXAMPLE OF A DISCONNECTED CONFIGURATION.....	104
FIGURE 4-24 VOID PROPAGATION.....	106
FIGURE 4-25 STATES OF MDP	108
FIGURE 4-26 AVAILABLE ACTIONS OF AN STATE.....	109
FIGURE 4-27 STATES REWARD FUNCTION.....	110
FIGURE 4-28 VALUE FUNCTION OF STATES	113
FIGURE 4-29 CONVERGENCE OF VI.....	115
FIGURE 4-30 MOTION OF A MOBILE ELECTRON	117
FIGURE 4-31 MOBILE ELECTRON ACTUATION	119
FIGURE 5-1 EXPERIMENTAL SETUP	123
FIGURE 5-2 PORT CONFIGURATION	124
FIGURE 5-3 HANDSHAKING.....	125
FIGURE 5-4 LOCALIZATION TEST.....	126
FIGURE 5-5 REFERENCE MODULE.....	126
FIGURE 5-6 LOCALIZATION	127
FIGURE 5-7 ROTATION TEST.....	128

FIGURE 5-8 SIMULATION TEST 1	131
FIGURE 5-9 SIMULATION TEST 2	135
FIGURE 5-10 SIMULATION TEST 3	137
FIGURE 5-11 SIMULATION TEST 4	140

LIST OF TABLES

TABLE 3-1 REQUIRED ROTATION STEPS (ONLY 2 MODULES WITH NO OTHER NEIGHBORS)	47
TABLE 3-2 REQUIRED ROTATION STEPS (2 MODULES WITH OTHER NEIGHBORS).....	48
TABLE 3-3 PORT CONFIGURATION (COMMUNICATION MICROCONTROLLER).....	64
TABLE 3-4 PORT CONFIGURATION (CONTROL MICROCONTROLLER)	66
TABLE 3-5 IR CHANNEL SELECTOR.....	69
TABLE 3-6 MICROCONTROLLERS FUSE BITS SETTINGS.....	70
TABLE 3-7 HEXBOT FUNCTIONS	75
TABLE 3-8 MESSAGE FORMAT	77
TABLE 3-9 WEIGHT OF HEXBOT COMPONENTS.....	78
TABLE 4-1 RELATIVE COORDINATES AND ORIENTATIONS.....	96
TABLE 4-2 DISCOUNT FACTOR AND NUMBER OF ITERATIONS	116
TABLE 4-3 MODULE ACTUATION.....	118

ACKNOWLEDGMENT

First and foremost, I would like to thank God for giving me the opportunity to be raised in my lovely family which in turns paved the way for me to be educated in my wonderful university and accomplish this work.

My dept to my parents and my family is undoubtedly beyond measure and I will be forever grateful for what they have sacrificed for me to get to this point, thank you so much.

To our dean, Dr. Assaf, the founder of Mechatronics center Dr. Jarrah and our program coordinator Dr. Dhaouadi, I cannot adequately express how deeply indebted I am to all you for the supports and facilities you have provided at AUS.

To my always busy advisors, Dr. Jarrah for his everlasting supports and motivations he gave me to continue manufacturing my little modules and Dr. Khaled for introducing me to the Markovian word, thank you so much for all your help and support and I am always grateful to you.

I would not be standing here without the unyielding support of my best friend Omid for his help in the Mechanical design; even though, I really had to push him so hard. I will never forget the support I got from Mr. Ashraf, Mr. Narayanan and Mr. Ricardo who helped me out during the manufacturing process.

I am also, of course, very thankful to Roozbeh for his brilliant design ideas and Behrooz for his advice to reduce my circuit noise.

I can no way forget my excellent friend Younes and all our chats during our evening Starbucks coffee breaks; you are the real “*Bogh Al-Havij*” by all sense of meaning, you are a great person, thank you.

I am grateful to many dear friends especially, Amer Al-radaideh, Laith Sahawneh “*Tanak Al-Zeytoon*”, Ali Ghaz and Amir Jafari for the unforgettable times we spent together, thank you so much.

This work will remain as a great honor of my life and my heart is filled with nothing but gratitude to all of you, thank you.

DEDICATION

It is my biggest joy and privilege to dedicate this work to my thoughtful, caring and compassionate parents.

Chapter 1

Introduction

This thesis primarily aims to explain the physical implementation of *HexBot*, a two dimensional Modular Self-Reconfigurable Robotic System (MSRRS) built in the AUS Mechatronics Center, along with its developed reconfiguration planning and control algorithm. The idea behind the proposed design both for the platform and the algorithm will be clearly addressed in this thesis.

1.1 Problem Statement

The work includes the construction of a centralized near-optimal reconfiguration path planner and a control algorithm for a lattice, homogenous, rigid, planar hexagonal MSRRS in addition to the implementation of *HexBot* as a universal module for such an algorithm.

To clarify and explain more details about the above statement regarding the proposed algorithm and the implemented platform, this chapter will be continued by a brief background about MSRRS. Moreover, to position this work, next chapter will thoroughly explore similar related works.

1.2 Background

1.2.1 *Modular Self-Reconfigurable Robotic Systems (MSRRSs)*

A conventional robot with a fixed architecture is usually designed for a limited task and works in a particular environment. Therefore, it can no more perform well when either its task or environment is changed. On the contrary, MSRRSs gained popularity since their functionalities were no longer restricted to a specific task or a certain environment.

MSRRSs are aimed to mimic the infrastructural principle of life and matter, where every element is composed of its fundamental components such as cells or atoms. Although each fundamental component is quite simple in its shape, intelligence and etc, a huge combination of them can form a powerful and complex living creature or object. There are several examples of such formations in our real life. One given by (Bishop, et al., 2005), is the ribosome that seem to be built out of large numbers of simple components. “This seems to occur when simple components self-organize via local interactions into more complex aggregates which, in turn, self-organize into larger aggregates and processes”.

Likewise, MSRRS implement the same concept by being composed of several simple robotic modules where each module has the ability to move around its neighbor modules to change its location using its primitive actuators, sensors, processors and communication. Therefore, the complete system would be capable of changing its shape autonomously and transforming into other shapes by moving its modules. Having such a feature, MSRRSs can reconfigure themselves to form structures that best fit the operating environment as well as the required functionality.

Overall similarities of a living biological organism and a MSRRS are summarized by (Murata & Haruhisa, 2007), as follow:

- Both consist of small components, living cells for the former and robotic modules for the latter
- Communication among the components is achieved by the diffusion of chemical substances in the former and by the exchange of digital information through module-to-module communication in the latter

- In both cases, the components cooperate with one another to adjust their configuration to the environment
- The mechanism of cooperation is embedded as the genomic information in the living cell and as a distributed program for each processor in the module

In MSRRSs the main concept that needs to be heeded is the “global formation” of the entire system out of “local interactions” among its individual modules which requires the coordination of a very large number of modules with a high degree of freedom.

As can be seen, the key feature of MSRRSs is versatility. For example, such a system composed of numerous small and simple modules can reconfigure itself to:

- form a locomotion gait that best fits the existing terrain
- avoid obstacles in highly constrained and unstructured environments
- envelope objects for protection or recovery
- form a manipulator that best fits the object needs to be handled
- optimize the measurement of sensory information
- grow structures such as bridges in times of emergency

Significant technological advances are expected from MSRRSs. An illustrative example of a potential application of such a versatile system is as follow: “A modular platform could carry a collection of self-reconfiguring modules to a site. The modules could then grow into a tower, enter the site through a small opening (such as a window) and reconfigure to survey the site. The modules could carry different sensors and collaborate to deploy these within their environment” (Butler & Rus, 2003).

The main difference in the design of a MSRRS and a conventional robot is explained by (Murata & Haruhisa, 2007). Many successful conventional robots are designed by mimicking the dynamical function of living creatures; however, MSRRSs are designed by mimicking the structural formation of living creatures. Conventional robots are designed top-down, i.e. based on the required performance and functionality, fundamental components are selected. In contrast, MSRRSs are designed down-top, i.e. initial module specifications will eventually determine the potential functionality of the complete system. As a result, modules should be designed in a way that the overall system would deliver the desired functionality. The point is that a MSRRS “is not a robot de-

signed to perform a specific task but a system that develops into various types of robots and executes a variety of tasks” (Murata & Haruhisa, 2007).

Figure 1-1 illustrates an artist’s rendition of a space application of a MSRRS re-configured in various morphologies for a variety of tasks (Zykov, Mytilinaios, Desnoyer, & Lipson, 2007).

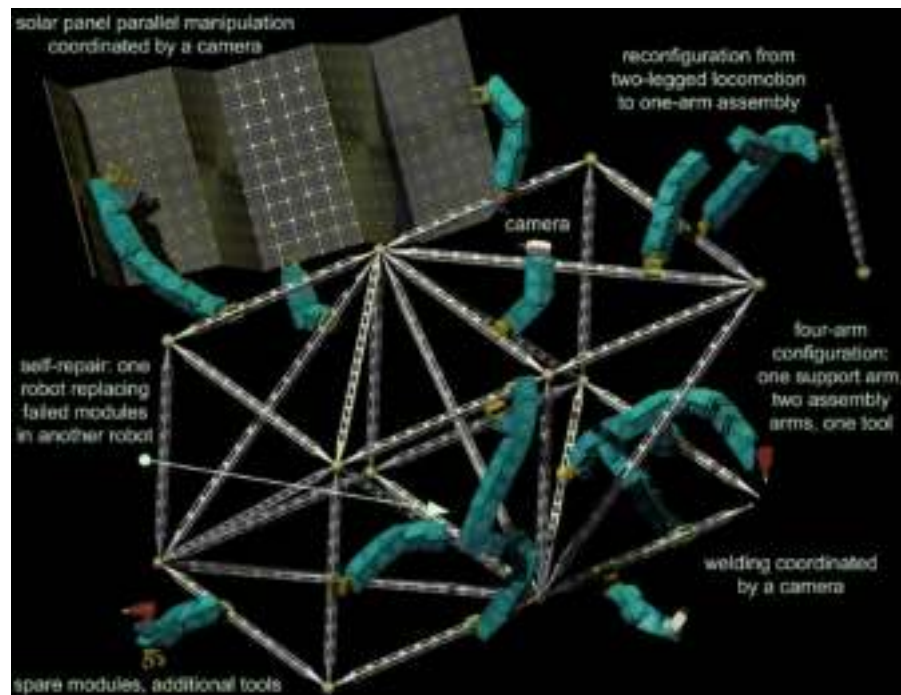


Figure 1-1 Artist's rendition of a space application

However, as discussed by (Yim, Duff, & Roufas, 2000) and (Murata & Haruhisa, 2007), it should be noted that the need of such versatile MSRRSs will be felt more in places where their unique functionalities are required; otherwise, fixed-architecture conventional robots designed for a particular task always perform better than MSRRSs for the same task and are also more cost effective and efficient.

As mentioned by (Yim, et al., 2007), there are few other types of robotic systems that share similar design and control challenges, but they are not MSRRSs. For example:

- Modular Robots: Built from modules, but cannot reconfigure themselves
- Self-assembling systems: Composed of multiple modules and can take different configuration, but cannot dynamically control or reconfigure their target shape

- Tensegrity robots: Composed of multiple interchangeable modules, but cannot self-reconfigure
- Multi-robot cooperating systems such as swarm robots: Composed of multiple units, but do not typically connect to form more complex physical structures
- Industrial robots with tool changers: Can be considered modular, but the degree to which they self-reconfigure is very limited

Unfortunately, conventional robotics methods are not able to address the field of MSRRSs and realization of the potential applications for MSRRSs based on the latest mechatronics designs is still beyond the current state-of-the-art.

Consequently, the growing field of MSRRS poses a distinct set of engineering challenges including: design of a universal module, implementation of the module in a relatively small scale, inter-module connections, communications, motion planning and control.

These challenges are mainly classified into two categories:

- Physical Implementation
- Algorithmic Development

Physical implementation focuses more on providing modules with a relatively small size, enough strength, robust inter-module connections, reliable communication, proper actuators, required processing power and of course a cost-effective solution which enables mass production of potentially numerous modules.

Apart from physical implementation difficulties, algorithmic issues are even more challenging to deal with. Path planning for a large number of modules in a potentially parallel and distributed manner considering the limited functionality of each simple individual module in terms of its computational and communicational power seems to be extremely demanding.

1.2.2 Research Motivation and Challenges

This section will address research motivation, potential application and challenges of the field of MSRRSs.

Motivation

The unique self-reconfiguration capability of MSRRSs provides this new field of robotics with a bright and promising future. The main incentives behind the development and design of MSRRSs are summarized below and an example (Inou, Minami, & Koskei, 2003) is illustrated in Figure 1-2:

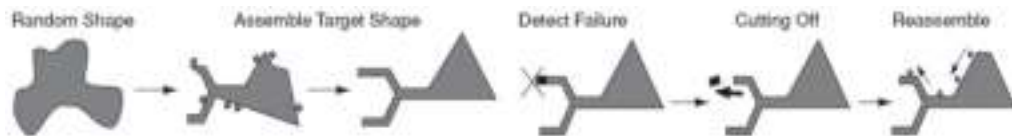


Figure 1-2 Self-Assembly and Self-Repair

- **Versatility** – (Through: Morphology, Shape changing / shifting, Self-assembling / organizing):

As discussed so far the diverse behavior of MSRRSs has been a fundamental motivation factor behind the ongoing research.

The self-reconfiguration capability enables multi functionality for these robots by making them potentially more adaptive to new tasks and thus performing a wide range of activities.

Not only that, but such a robotic system can also dynamically reconfigure itself and continue its mission in a changing, unstructured or even uncertain environment. A MSRRS can switch between different locomotion gaits such as: crawling, walking, rolling, climbing, jumping, shrinking, climbing stairs and etc.

Examples of such versatility can be seen in several works done in this field such as: (Fukuda & Nakagawa, 1987), (Murata, Kurokawa, Yoshida, Tomita, & Kokaji, 1998), (Yim, Duff, & Roufas, 2000), (Castano, Shen, & Will, 2000) and (Murata & Haruhisa, 2007).

- **Robustness** – (Through: Self-Repair):

Since the complete robotic system is basically composed of a large number of interchangeable modules, a malfunctioning module can be autonomously replaced with another one.

Self-repair was studied in several works such as: (Murata, Kurokawa, & Kokaji, 1994), (Yoshida E. , Murata, Tomita, Kurokawa, & Kokaji, 1998), (Tomita, Murata,

Kurokawa, Yoshida, & Kokaji, 1999), (Fitch, Rus, & Vona, 2000) and (Murata, Yoshida, Kurokawa, Tomita, & Kokaji, 2001). An illustration is also depicted in Figure 1-2.

- **Self-Reproduction:**

The modularity of MSRRSs transforms them to potential candidates for self-reproduction. Self-reproduction refers to the process where a system has the ability to autonomously produce another functional system as explained by (Zykov, Mytilinaios, Adams, & Lipson, 2005).

If the produced system is identical to the original system, this process is called self-replication. In self-replication, the produced copy can as well make a copy of itself and such a process can proceed till enough number of the required system is available. Such a process tends to duplicate the fundamental nature of biological life.

As explained by (Zykov, Mytilinaios, Desnoyer, & Lipson, 2007) conventional robots achieve long-term sustainability and adaptation by the use of durable hardware and adaptive controllers which is totally in contrast with the way biological systems behave. In biological systems, long-term sustainability and evolutionary adaptation are provided through the process of self-repair and, ultimately, self-reproduction.

- **Scalability:**

Another interesting feature of MSRRs is scalability. Having modular building blocks, these robotic systems can be scaled relatively easily compared to conventional robots. Note that it is generally difficult to realize complete scalability in robotic systems, i.e. scalable mechanical platform, scalable distributed controller, scalable communication and etc. In practice there is normally a tradeoff between scalability and response time of systems; the more the number of modules, the less the speed of response. This issue is addressed in (White & Yim, 2007).

- **Cost-Effectiveness:**

MSRRS are eventually cost effective. Once a single module or few types of modules are designed, the overall system can be formed by mass-production of the same modules. Not only that, but once modules are mass produced, huge variety of robots can be built from them as well.

As explained earlier in the in 1.2.1, it should be always noted that the need of such versatile MSRRSs will be felt more in places where their unique functionalities are

required; otherwise, fixed-architecture conventional robots designed for a particular task always perform better than MSRRSs for the same task and are also more cost effective and efficient. Moreover, the current mechatronics state-of-the-art is still not capable of realizing all these features and more research needs to be done in this growing field.

Application

The current ideas behind the applications for MSRRSs rely primarily on versatility and fault tolerance rather than quick or optimal response. Potential applications for such robotic systems are classified into three main categories: locomotion, manipulation and structure formation.

- **Locomotion:**

MSRRSs can form different locomotion gaits and move in various environments. The importance of such locomotion gaits becomes extremely attractive when the environment is unstructured, uncertain or dynamically changing.

Typical application in this category would be: Search and rescue missions (fire fighting, earthquake, etc.), Sea or space exploration (undersea mining, planetary search and data collections, etc.), Hazardous or remote environment operations (chemical plants, desert sites, etc.), Surveillance (monitoring, etc.), General locomotion (climbing stairs, obstacle avoidance, etc.) and Sensory network monitoring (Optimize the measurement of a sensory network, etc.)

As an illustrative example, let's look at "*Space Exploration*", where perhaps the most attention is paid to MSRRS. (Yim, et al., 2007) describe the situation as follow: "Long-term space missions require a self-sustaining robotic ecology that can handle unforeseen situations and may require self-repair. Self-reconfigurable systems are better able to handle tasks that are not known a priori, especially compared to fixed-configuration systems. In addition, space missions are highly volume and mass constrained. Sending a robot system that can reconfigure to achieve many tasks saves shipping mass and volume as compared to sending many robots that each can accomplish one task."

- **Manipulation:**

As MSRRS are modular, they can change their shape to form the most suitable manipulator needed to handle objects. Such a feature enables forming several different manipulators from a collection of simple modules.

- **Structure Formation:**

There are also interests in using MSRRSs to form structures. As an example, a MSRRS can be utilized to form a physical 3D simulation of a part or a graph. Other examples are growing structures to form bridges or buttresses in times of emergency as explained by (Walter, Tsai, & Amato, 2005) or envelopment of objects, (such as satellites in space) for protection or recovery purposes.

Challenges

The fast growing field of MSRRS with unique capabilities poses many fundamental engineering challenges. Few of those challenges have been addressed and especially during the last decade, there has been a huge advancement in this field. Nevertheless, there still exist many more challenges to be overcome in future.

These challenges are essentially classified in three categories. First is to design a physical platform composed of universal modules to achieve the goal of MSRRS in terms of small size, fast actuation, enough strength and etc, all at a reasonable cost. The second challenge refers to algorithmic issues regarding reconfiguration, locomotion, manipulation, intelligence, control and etc, all in an optimal manner. The last category is to find suitable application for such systems where such capabilities can be utilized considering the overall performance of the system. These three challenges are explained briefly below:

- **Physical Implementation**

Performance of a MSRRS depends directly on the design of its individual modules and the performance of each individual module in turn depends on a number of factors such as: space filling geometric shape, robustness to failure, physical strength, light weight, scalable in size and quantity, fast actuation, quick and reliable connection, low power consumption and cost-effectiveness.

Consider designing a *universal module*, where the task and environment of a versatile system composed of a huge number of it, are not known before hand during the design stage while all the above factors are meant to be taken into consideration.

So far, there have been several designs proposed by different researchers in this field and each tried to address some of the above factors. Though, there has not been a single design addressing all those factors efficiently. Those fundamental challenges are briefed as follow:

- **Shape:** Different geometrical shapes have been proposed and tested which includes, spherical, cubical and hexagonal modules. The main criterion for the physical shape is to be able to fill the required structure densely with minimum gaps. However, in some designs these gaps are utilized to provide the complete system with a degree of flexibility.
- **Robustness:** Modules shall be designed such that the overall system shall be able to recover itself from any kind of failure (mechanical, electrical, control, communication, power, etc). In other words, ideally modules shall be able to understand if there is a failure in them or their neighbors and the system shall be able to remove the faulty module when it is not responding.
- **Strength:** Modules shall be made from materials that have enough strength to provide the system with the ability of performing different tasks. The down side of this is usually either the weight or the cost if a composite material is chosen. In general it is preferred to use material with high strength to weight ratio.
- **Weight:** To increase the overall performance and efficiency of the system, every single part of the module shall be designed to be as light as possible so that the overall module can move with the minimum amount of energy. However, sometimes for stability purposes this condition is overlooked.
- **Scalability: (Scale down the size, scale up the quantity)** Size: the smaller the modules the finer the resolution of the complete system. Some researchers are now thinking of MEMs to manufacture smaller modules. Quantity: Most of current available systems are made of less than 100 modules which is just enough for the research purposes and demonstration of the conceptual design.
- **Actuation:** It's preferred to have fast actuation; however, most designs lack this factor which is considered as one of the most important properties of a module. Once the quantity of modules increased, the need of fast actuation will be felt more.

- **Connection:** Inter-module connection plays an important rule as well. Ideal connections shall be strong and shall not require precise alignment between two modules. Moreover, the connection shall be performed quickly with minimum power consumption. Preferably, once modules are connected no more power shall be consumed to maintain the connection.
- **Power Consumption:** Clearly, it is preferred to keep the system working for a long period and the overall power consumption of the system depends directly on the power consumption of its modules. Therefore, ideally, it is proffered to have very high efficiency for module actuators.
- **Cost-effectiveness:** Since the goal is to make huge collection of modules, each single module should be implemented with a minimum cost. Mass production would hopefully reduce the overall cost of these systems, once an ideal universal module is designed.
- **Algorithmic Issues**

The unique nature of MSRRS requires the coordination of a large number of modules in an optimal manner in terms of time or energy. This leads to the development of challenging algorithms. The ultimate goal for these algorithms is to be decentralized (acting in a distributed manner), unsupervised (no global feedback) and architecture independent (can be applied to different systems). Furthermore, they shall incorporate parallel actuation and cooperative actions. The primarily requirements for such algorithms are summarized below:

- **Intelligence:** A high level algorithm is required to specify the optimal configuration or locomotion gait, based on the current state of the environment and assigned task.
- **Robustness:** A high level algorithm is required to detect failure (in modules or the overall system) and rearrange to recover from failure and continue the mission.
- **Reconfiguration:** A low level algorithm is required to plan paths for modules to transform the shape of the system from one configuration into another configuration.

- **Locomotion:** A low level algorithm is required to provide motion for the complete system. Reconfiguration algorithm may be used for this purpose while enforcing constraints for the intermediate configurations.

Several methods proposed have not been very successful since they are computationally very expensive for the current state-of-the-art. Therefore, hierarchical multilayer approaches (like the one in this work) are preferred to distribute the load into different levels.

- **Application**

Apart from the mentioned engineering challenges, there still one uncommon challenge remains which deals with an appropriate application for MSRRS. Many researchers in this field such as (Yim, et al., 2007) agree that identifying a truly demanding application for MSRRSs where all capabilities and advantages are required is still a key challenge.

1.3 Contribution

The thesis in general contributes to the growing field of modular self-reconfigurable robotic systems. Specific contributions of this thesis are mainly those addressing the first two sets of challenges: physical implementation and algorithmic development. Listed below are the detailed contributions:

- Platform
 1. Design of a universal two dimensional hexagonal module towards the promising goals of the field of MSRRS
 2. Homogeneity in all aspects, such as actuation, connection, computation and etc
 3. Fast actuation and motion with no moving parts through magnetic fields, in line with the ultimate goals and possible scalability issues in terms of size reduction and cost effectiveness.
 4. Quick and strong connection without requiring precise alignment
 5. Multilayered electronics circuits and possibility of replacement, modification or improvement for every individual layer separately

6. Experimental performance evaluation for the functionality of the entire platform
- Algorithm
 1. Development of a hierarchical multilayer framework for lattice based modular systems
 2. Optimization and path planning for minimum module movement
 3. Problem formulation as a Markov Decision Process (MDP) that can be easily adopted for other platforms
 4. Implementation of an optimal policy search method enforcing the required constraints using dynamic programming in MDP
 5. Multilayered nature of the framework, providing openness, flexibility and ease of modification and improvement for each individual layer to move towards the essential goals
 6. Modeling, testing and simulation of the complete algorithm with access to every single function using Matlab

1.4 Thesis Outline

Having introduced the topic in this chapter, we will continue with a literature review to look at related works in this field in chapter 2. This chapter will briefly review literature in terms of available physical and algorithmic platforms for MSRRS. Next we move on to chapter 3 where we explain about our universal module implementation and our design criteria are explained. In chapter 4, the control algorithm will be introduced in details using a simple example that is followed throughout the chapter. Chapter 5 is dedicated to several examples to illustrate and evaluate the performance of the proposed system and continues with results and discussions. Finally, chapter 6 concludes the work by a summary along with the limitations and directions for future research.

Chapter 2

Related Work

There has been great advancement in the field of MSRRS over the past decade. Achieving the ultimate goal in this promising field has pushed researchers forward to attempt overcoming the challenges by proposing several hardware implementations along with algorithmic developments.

The overall performance of a MSRRS is a combination of hardware as well as software functionalities. Furthermore, the hardware and software performances are directly linked together; in other words, the physical platform imposes its possible actualizations and limitations to the algorithm and at the same time the algorithm imposes its computational and communicational needs to the platform. As a result optimization of the overall system depends heavily on the development and functionality of both hardware and software.

The new field of MSRRS has been attracting significant attention during the recent years and there has been incredible improvement in this field. Therefore, this chapter is dedicated to review the literature in order to position and motivate this work. The chapter will continue by introducing a brief history and the overall infrastructure of MSRRSs in terms of different methods implemented in hardware and software. Majority of suc-

successful implemented platforms will be introduced and several developed control algorithms will be presented.

2.1 History

According to (Murata & Haruhisa, 2007), the basic idea of having systems composed of homogeneous building blocks dates back to 1966 by (Neumann, 1966). However, as a new trend in robotics, the idea of MSRRS was first introduced in 1987 by (Fukuda & Nakagawa, 1987) with CEBOT (cellular robot) where each module was in fact a complete mobile robot that could work individually and independent from other modules. The design was based on heterogeneous modules (locomotion module, rotation joint module, prismatic joint module and end-effector module) and it was demonstrated that a system composed of all these individual modules is capable of performing several tasks.

Later on the concept was geared towards simulating the behavior of living biological organisms and modules became simpler (like atoms and cells) which could perform tasks only in groups and eventually leading towards MSRRSs.

2.2 Choices for Physical Platform

Clearly performance of the overall MSRRS depends directly on the design of its individual modules and so far there have several types of physical platforms proposed by different researchers in this field. Figure 2-1 illustrates the main characteristics of any physical platform and each of these categories is briefly explained.

2.2.1 Architectural Topology

MSRRSs are classified into four categories based on their architectural topologies: mobile, lattice, chain, and hybrid. Figure 2-2 illustrates the main topologies.

In mobile architecture, modules have the ability to move in the environment independently from other modules. A universal module in this category shall be equipped with all necessary components required for motion which is considered to be a disadvantage. Modules can disconnect, move around and reconnect to each other and form a chain or lattice type of configuration. The best example for this class is the work done by

(Fukuda & Nakagawa, 1988) for CEBOT. Not all researchers agree that this type can be considered as a MSRRS since modules will be disconnected during the reconfiguration and therefore other categories gained more popularity.

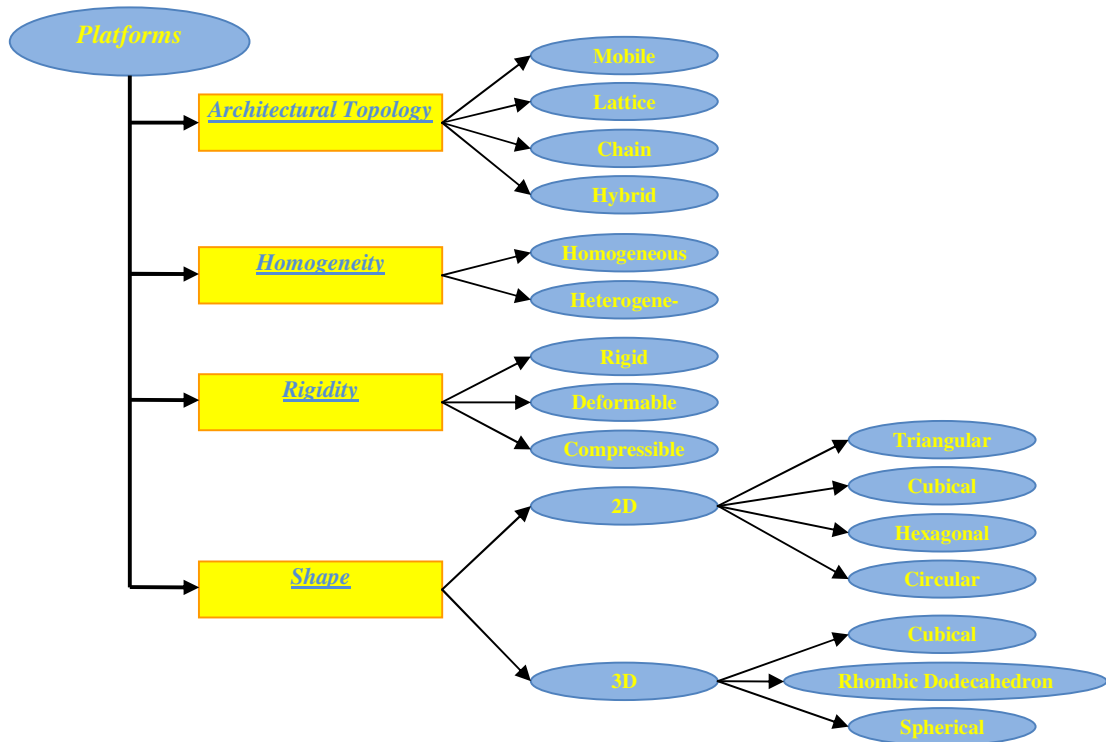


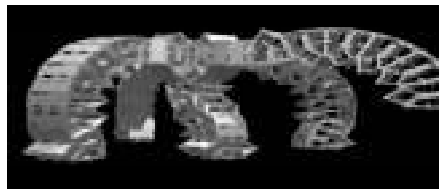
Figure 2-1 Basic Characteristics of Physical Platforms

In lattice architecture modules are more similar to biological cells. In this configuration, modules can fill discrete positions in a grid structure. This architecture offers: parallel and open-loop control, simple reconfiguration, high potential for scalability, and relatively easy collision detection (only locally). In general, the lattice configuration performs very well for reconfiguration but not for locomotion. Many platforms have been designed based on lattice architecture, such as: (Murata, Kurokawa, Yoshida, Tomita, & Kokaji, 1998) for 3D Fracta robots, (Christensen & Stoy, 2006) for ATRON robots, (Goldstein, Mowry, Gibbons, Pillai, Rister, & Lee, 2006) for Catom robots, (Rus & Vona, 2001) for Crystalline robots, (Yoshida E., Murata, Kurokawa, Tomita, & Kokaji, 1998) for Fracta robots, (Unsal, Kiliccote, & Khosla, 1999) for I-Cube robots, (Gilpin, Kotay, & Rus, 2007) for Miche robots, (Yoshida E., Murata, Kokaji, Kamimura, Tomita,

& Kurokawa, 2002) for Mico Unit robots, (Kotay K. , Rus, Vona, & McGray, 1998) for Molecule robots, (Inou, Minami, & Koskei, 2003) for Pneumatic robots, (Klavins, Burden, & Napp, 2006) for Programmable Parts robots, (Hosokawa, et al., 1998) for RI-KEN Vertical robots and (Suh, Homans, & Yim, 2002) for Telecube robots. Moreover, some researchers developed lattice based algorithms such as: (Walter, Tsai, & Amato, 2002) for Hexagonal modules, (Fitch, Rus, & Vona, 2000) for Cubical modules, (Pamecha, Ebert-Uphoff, & Chirikjian, 1997) for Metamorphic hexagonal modules.



a) Lattice (Gilpin, Kotay, & Rus, 2007)



b) Chain (Yim M. , 1994)



c) Hybrid (Kurokawa, Kamimura, Yoshida, Tomita, Kokaji, & Murata, 2003)

Figure 2-2 Primarily Architectural Topologies

In chain architecture, modules are connected to each other in a serial manner forming tree (open) or loop (closed) structures. Chain architecture is primarily successful for motion generation and locomotion. Although modules can also fold and form space filling structures, self-reconfiguration is computationally very demanding and cannot be easily performed as it is done in lattice architectures. The chain architecture usually requires a closed loop control for locomotion and is considered to be more versatile as it can reach continuous locations in the space. Collision detection is relatively more difficult in this category since global collision detection technique is required, compared to local collision detection methods used in lattice architecture. The first work in chain architecture started by (Yim, Lamping, Mao, & Chase, 1997) for Polypod robots although it was not considered to be a totally MSRRS since inter-module connections were not automatic but the newer version (Yim, Goldberg, & Casal, 2000) for PolyBot robots had automatic connections between modules. Other works in this area include: (Hamlin & Sanderson, 1998) for Tetrobot modules, (Bojinov, Casal, & Hoag, 2000) for Proteo ro-

bots, (Stoy, Shen, & Will, 2002) for CONRO robots, (Mytilinaios, Desnoyer, Marcus, & Lipson, 2004) for Molecubes robots and (Casal & Yim, 1999) for self-reconfiguration planning.

In hybrid architecture, designs are combining both lattice and chain configurations to take advantage of their potential benefits and eliminate their drawbacks. Therefore, hybrid systems are capable of performing remarkable motion generation needed for locomotion and are also capable to reconfigure into different configurations. The best examples of hybrid systems can be seen in the work of (Yoshida E. , Murata, Kamimura, Tomita, Kurokawa, & Kokaji, 2003) for MTRAN modules and (Salemi, Moll, & Shen, 2006) for SuperBot robots.

2.2.2 Homogeneity

MSRRSs are classified into two major categories: homogeneous and heterogeneous as described below.

Homogeneous systems (also known as metamorphic) are composed of identical modules where all modules have the exact same physical shape, computational power and communicational capabilities. The main advantage in these systems is referred to as “module interchangeability” that increases the overall robustness of the system by the means of redundancy that helps self-repair and self-replication. Another advantage in this category refers to simplified control algorithms as it is not necessary to move a specific module to a specific location as long as all modules are the same. Homogeneous systems are given a lot of attention in this field such as the work of (Murata, Kurokawa, & Kokaji, 1994) for Fracta robots, (Pamecha, Ebert-Uphoff, & Chirikjian, 1997) for Metamorphic robots, (Yim, Lamping, Mao, & Chase, 1997) for Polypod robots, (Kotay & Rus, 1998) for Molecule robots, (Hosokawa, et al., 1998) for RIKEN Vertical robots, (Walter, Welch, & Amato, 2000) for hexagonal modules, (Yoshida, Kokaji, & Murata, 2000) for Micro Unit robots, (Rus & Vona, 2001) for Crystalline robots, (Castano, Behar, & Will, 2002) for CONRO modules, (Murata, Yoshida, Kamimura, Kurokawa, Tomita, & Kokaji, 2002) for MTRAN robots, (Suh, Homans, & Yim, 2002) for Telecube robots, (Mytilinaios, Desnoyer, Marcus, & Lipson, 2004) for Molecubes robots and (Klavins, Burden, & Napp, 2006) for Programmable Parts robots.

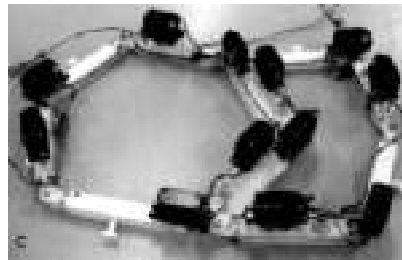
Heterogeneous systems on the other hand provide the freedom of having different types of modules in the system. The primary advantage of these systems relies on specific modules functionality. For example, there can be sensor modules, communication modules, locomotion modules, power modules and etc. in the same system which reduces the cost of each module as long as all modules are not supposed to be equipped with all these features. The main drawback in these systems is the loss of redundancy and also more complex software requirements. Less work has been done on this category which including the work of (Fukuda & Ueyama, 1994) for CEBOT robots, (Fitch, Butler, & Rus, 2003) and (Fitch, butler, & Rus, 2005) addressing the reconfiguration planning.

2.2.3 Rigidity

In terms of rigidity there have been three types modules proposed: rigid, deformable and compressible as shown in Figure 2-3.



a) Rigid (Zykov, Mytilinaios, Desnoyer, & Lipson, 2007)



b) Deformable (Pamecha, Chiang, Stein, & Chirikjian, 1996)



c) Compressible (Butler, Fitch, & Rus, 2002)

Figure 2-3 Rigid, Deformable and Compressible Modules

Most systems developed in this field are based on rigid modules such as the work of (Fukuda, Ueyama, & Kawauchi, 1990) for CEBOT robots, (Murata, Kurokawa, & Kokaji, 1994) for Fracta robots, (Hosokawa, et al., 1998) for RIKEN Vertical robots, (Walter, Welch, & Amato, 2000) for hexagonal modules, (Bojinov, Casal, & Hoag, 2000) for Proteo robots, (Yim, Duff, & Roufas, 2002) for PolyBot robots, (White, Kopanski, & Lipson, 2004) for 2D stochastic robots and (White, Zykov, Bongard, & Lipson, 2005) for 3D stochastic robots.

Deformable and compressible modules make the reconfiguration planning easier by allowing modules to change their shape or size and help other modules to pass among them and therefore reduce the collision constraint. Work on these modules include, (Pamecha, Chiang, Stein, & Chirikjian, 1996) for deformable Metamorphic robots, (Rus & Vona, 2001) for compressible Crystalline robots and (Suh, Homans, & Yim, 2002) for compressible Telecube robots.

2.2.4 Shape

Two Dimensional Platforms

Work was primarily started with two dimensional modules such as: (Hosokawa, et al., 1998) for RIKEN Vertical robots, (Casal & Yim, 1999) for reconfiguration planning, (Tomita, Murata, Kurokawa, Yoshida, & Kokaji, 1999) for Fracta robots, (Chiang & Chirikjian, 2001) for Metamorphic robots, (Rus & Vona, 2001) for Crystalline robots, (Inou, Kobayashi, & Koseki, 2002) for Pneumatic robots, (Walter, Welch, & Amato, 2002) for reconfiguration planning of hexagonal modules, (Yoshida E. , Murata, Kokaji, Kamimura, Tomita, & Kurokawa, 2002) for Micro Unit robots, (White, Kopanski, & Lipson, 2004) for 2D stochastic robots, (Goldstein, Mowry, Gibbons, Pillai, Rister, & Lee, 2006) for Catom robots and (Klavins, Burden, & Napp, 2006) for Programmable Parts robots.

These robots were mainly built in triangular, cubical, hexagonal or circular shapes as shown in Figure 2-4.

Three Dimensional Platforms

MSRRSs are not limited only to two dimensional planar robots since several three dimensional robots were proposed afterwards such as the work of (Yim, Lamping, Mao, & Chase, 1997) for Polypod robots, (Murata, Kurokawa, Yoshida, Tomita, & Kokaji, 1998) for 3D Fracta robots, (Hamlin & Sanderson, 1998) for Tetrobot robots, (Kotay & Rus, 1999) for Molecule robots, (Kurokawa, Murata, Yoshida, Tomita, & Kokaji, 2000) for MTRAN robots, (Castano & Will, 2000) for CONRO robots, (Unsal & Khosla, 2000) for I-Cubes robots, (Yoshida, Kokaji, & Murata, 2000) for Micro Unit robots, (Yim, Zhang, Lamping, & Mao, 2001) for Proteo robots, (Suh, Homans, & Yim, 2002) for Te-

lecube robots, (Yim, Roufas, Duff, Zhang, Eldershaw, & Homans, 2003) for PolyBot robots, (Jorgensen, Ostergaard, & Lund, 2004) for ATRON robots, (Zykov, Mytilinaios, Adams, & Lipson, 2005) for Molecubes robots, (White, Zykov, Bongard, & Lipson, 2005) for 3D Stochastic robots, (Salemi, Moll, & Shen, 2006) for SuperBot robots and (Gilpin, Kotay, & Rus, 2007) for Miche robots.



a) Programmable Parts (Bishop, et al., 2005)



b) Crystalline (Butler, Fitch, & Rus, 2002)



c) Fracta (Murata, Kurokawa, & Kokaji, 1994)



d) Catom (Goldstein, Mowry, Gibbons, Pillai, Rister, & Lee, 2006)

Figure 2-4 Two Dimensional Triangular, Cubical, Hexagonal and Circular Platforms

These robots were mainly built in cubical, rhombic dodecahedron and spherical shapes as shown in Figure 2-5.

Cubical platforms are constructed for MTRAN (Murata, Yoshida, Kurokawa, Tomita, & Kokaji, 2001), Molecubes (Mytilinaios, Desnoyer, Marcus, & Lipson, 2004) and 3D stochastic (White, Zykov, Bongard, & Lipson, 2005). Rhombic dodecahedron shape is used in Polypod (Yim, Lamping, Mao, & Chase, 1997) and Proteo (Bojinov, Casal, & Hoag, 2000). Finally spherical platforms are built for Molecule (Kotay K. , Rus, Vona, & McGray, 1998) and ATRON (Jorgensen, Ostergaard, & Lund, 2004).

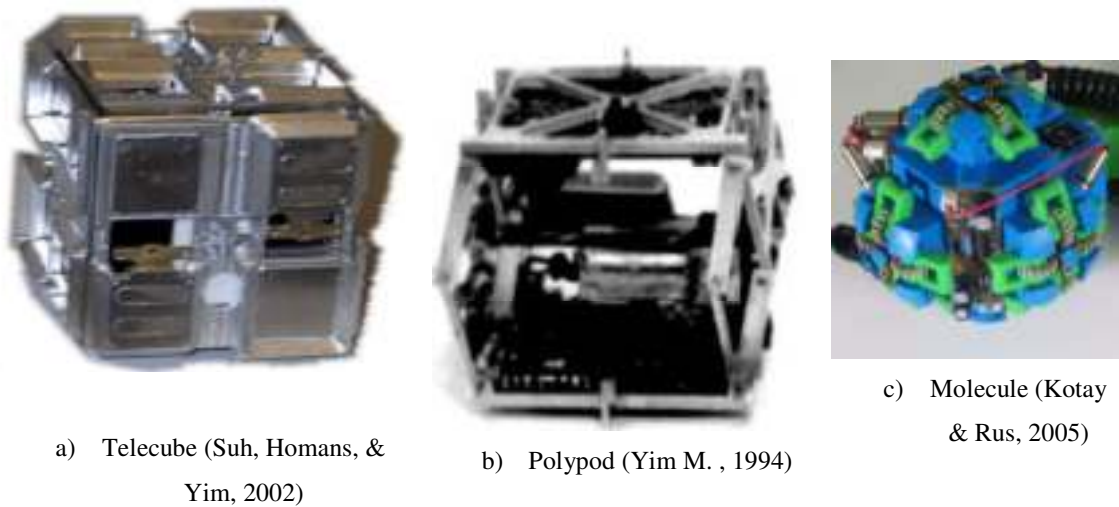


Figure 2-5 Three Dimensional Cubical, Rhombic Dodecahedron and Spherical Platforms

2.3 Range of Control Algorithms

Effective use of MSRRSs requires a method powerful enough to deal with a possibly large number of modules so that the combination of the modules will perform the required action. This is generally referred to as a path planner and controller design and considered to be one of the key challenges of this field.

The controller can be generally designed for different purposes in different ways and can have several properties as illustrated in Figure 1-2.

2.3.1 *Reconfiguration vs. Locomotion*

The planning algorithms are mainly designed for either reconfiguration or locomotion.

- Reconfiguration (also referred to as morphology) algorithms generate a sequence of module movements that transforms the overall shape of the system from an arbitrarily initial configuration to a desired goal configuration. In other words, these algorithms specify how modules shall rearrange themselves to form different structures.
- On the other hand, locomotion algorithms are primarily designed to generate motion for the overall system through different locomotion gaits.

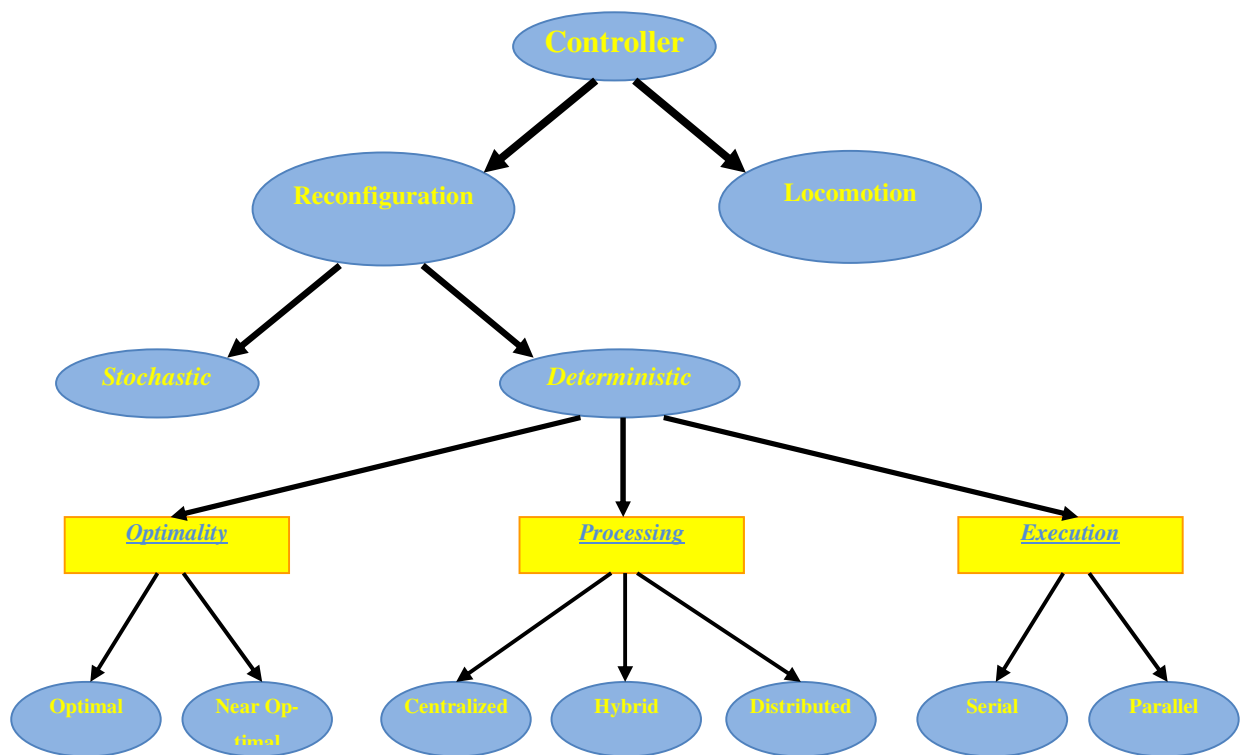


Figure 2-6 Controller Design Hierarchy

Generally speaking locomotion may seem to be a subset of reconfiguration when a group of module reconfigure themselves into different intermediate configurations to move from a starting location towards a goal location. However, it should be noted that reconfiguration algorithms are only interested in the initial configuration and the goal configuration; therefore, the shape of the system during reconfiguration is unspecified and not restricted to any boundaries. While locomotion algorithms take into account the initial, intermediate and goal configuration of the overall system.

Reconfiguration problem is addressed in several works such as: (Chirikjian, Pamecha, & Ebert-Uphoff, 1996) for Metamorphic robots, (Murata, Kurokawa, Yoshida, Tomita, & Kokaji, 1998) for 3D Fracta robots, (Casal & Yim, 1999) for chain modules, (Shen, Will, & Castano, 1999) for CONRO robots, (Bojinov, Casal, & Hoag, 2000) for Proteo robots, (Kotay & Rus, 2000) for Molecule robots, (Walter, Welch, & Amato, 2002) for hexagonal modules and (Vassilvitskii, Kubica, & Rieffel, 2002) for Telecube robots.

Locomotion algorithms are also presented in several works such as: (Yim M. , 1994) for Polypod robots, (Kotay & Rus, 1999) for Molecule robots, (Stoy, Shen, & Will, 2002) for CONRO robots, (Yoshida E. , Murata, Kamimura, Tomita, Kurokawa, & Kokaji, 2003) for MTRAN robots, (Kamimura A. , Kurokawa, Yoshida, Tomita, Kokaji, & Murata, 2004) for MTRAN robots, (Kamimura A. , Kurokawa, Yoshida, Murata, Tomita, & Kokaji, 2005) for MTRAN robots and (Shen, Krivokon, Chiu, Everist, Rubenstein, & Venkatesh, 2006) for SuperBot robots.

2.3.2 Deterministic vs. Stochastic

Deterministic reconfiguration algorithms require path planning for every individual module to be moved from one location into another. Therefore, control algorithm dictates a sequence of module movements. In this type of reconfiguration, the exact locations of modules need to be known for the algorithm and the time takes for the reconfiguration can be guaranteed.

On the contrary, in stochastic reconfiguration, modules are moved randomly in the environment and collide with each other. The task of the control algorithm in this case is to determine whether the collided modules should stay connected after the collision or not. In this type of reconfiguration, the locations of modules are unknown unless they collide with the substrate and the reconfiguration time can be statistically guaranteed. It should be noted that stochastic reconfiguration can help reduce the size of modules since the required movement actuation can be provided externally from the environment and does not need to be implemented in the modules.

Deterministic planners are the common controllers designed for MSRRSs and extensively studied in literature, such as the work of: (Fukuda & Nakagawa, 1988) for CEBOT robots, (Fukuda, Buss, Hosokai, & Kawauchi, 1991) for CEBOT robots, (Yim M. , 1994) for Polypod robots, (Yoshida E. , Murata, Kurokawa, Tomita, & Kokaji, 1998) for Fracta robots, (Casal & Yim, 1999) for chain robots, (Rus & Vona, 1999) for Crystalline robots, (Kotay & Rus, 2000) for Molecule robots, (Salemi, Shen, & Will, 2004) for CONRO robots, (Unsal & Khosla, 2001) for I-Cube robots, (Butler, Fitch, & Rus, 2002) for Crystalline robots, (Walter, Tsai, & Amato, 2002) for hexagonal modules, (Yim, Zhang, & Duff, 2002) for PolyBot robots, (Vassilvitskii, Kubica, & Rieffel, 2002) for Telecube

robots(Christensen, Ostergaard, & Lund, 2004) for ATRON robots and (Yoshida, Kurokawa, Kamimura, Tomita, Kokaji, & Murata, 2004) for MTRAN robots.

Stochastic reconfiguration was addressed in few works such as: (White, Kopanski, & Lipson, 2004) for 2D stochastic robots, (White, Zykov, Bongard, & Lipson, 2005) for 3D stochastic robots and (Klavins, Burden, & Napp, 2006) for Programmable Parts robots.

2.3.3 Optimal vs. Near Optimal

Optimal sequence of moves from an arbitrary initial configuration to a desired goal configuration can be found in theory by searching the graph of all possible configurations; however, such search approaches become intractable due to the fact that the number of possible configurations grows exponentially with the number of modules in the configuration. Therefore, techniques relying on such a representation require space and computation proportional to the number of modules, leading to intractable space (storage) and time complexity.

As a result, near optimal reconfiguration algorithms are gaining more popularity where heuristics and optimization come together to perform the required reconfiguration.

2.3.4 Centralized vs. Distributed

In the centralized control method, modules are given instructions from a central unit. This unit needs to have complete global information of all modules and decide on what motion is required for each module to perform the reconfiguration. This method can be suitable for MSRRSs that have small number of modules and more optimal reconfigurations paths can be computed since global information is available. The main problem with this type of controller is the single point of failure; once the controller fails all modules will fail. Another problem is that this method requires huge communication bandwidth once the system is scaled up and in most cases modules require to have fixed ID to communicate with the central unit. Another problem in this method is that all the computation is done in one unit and therefore the processing takes more time. Such controllers are addressed in (Yim M. , 1994) for Polypod robots, (Kotay K. , Rus, Vona, & McGray, 1998) for Molecule robots, (Yoshida E. , Murata, Kaminura, Tomita, Korokawa, &

Kokaji, 2000) for MTRAN robots, (Chiang & Chirikjian, 2001) for Metamorphic robots and (Nguyen, Guibas, & Yim, 2001) for PolyBot robots.

In the distributed control method, the global reconfiguration of the system is achieved through the decision and motion of each individual module based on its local neighborhood information. Distributed controllers are more robust since they do not rely on a centralized unit and they can be easily scaled up. Moreover, parallel processing makes the reconfiguration faster in these systems. The absence of a centralized controller eliminates the need of very high bandwidth communication and the ID's can be assigned dynamically. All these come at the cost of near-optimal solutions where the reconfiguration cannot be performed as optimal as it's done in the centralized controllers. There are many researchers in this field trying to address the distributed controllers design such as: (Fukuda, Ueyama, & Sekiyama, 1995) for CEBOT robots, (Yoshida E. , Murata, Tomita, Kurokawa, & Kokaji, 1998) for Fracta robots, (Hosokawa, et al., 1998) for RIKEN Vertical robots, (Casal & Yim, 1999) for chain robots, (Walter, Welch, & Amato, 2000) for hexagonal modules, (Butler, Byrnes, & Rus, 2001) for Crystalline robots, (Yim, Zhang, Lamping, & Mao, 2001) for Proteo robots, (Kubica, Casal, & Hogg, 2001), (Lee & Sanderson, 2001) for Tetrabot robots, (Vassilvitskii, Yim, & Suh, 2002) for Telecube robots, (Butler, Kotay, Rus, & Tomita, 2002) for Crystalline robots, (Shen, Salemi, & Will, 2002) for CONRO robots, (Yim, Zhang, & Duff, 2002) for PolyBot robots, (Payne, Salemi, Will, & Shen, 2004) for CONRO robots and (Rosa, Goldstein, Lee, Campbell, & Pillai, 2006) for lattice based robots.

Many successful controllers designed are considered to be hybrid. In other words, to provide an optimal reconfiguration plan they are composed of both a centralized part and a distributed part. These algorithms are usually multiphase or multilayer. In an initial processing phase they require a global knowledge of the system and the remaining processing can be done in a distributed manner. For example (Yoshida E. , Murata, Kamimura, Tomita, Kurokawa, & Kokaji, 2001) proposed a two-layered motion planning for MTRAN. The first layer is a global flow planner to provide the possible paths and motion orders and the second layer is a local motion scheme selector based on a rule database to make the flow. Another example is (Prevas, Unsal, Efe, & Khosla, 2002) where

a hierarchical motion planning strategy for a distributed bipartite robotic system, I-Cubes is presented.

2.3.5 Serial vs. Parallel

Depending on the design of the algorithm either single module can move at a time (serial motion) or several modules can move at a time (parallel motion). When the reconfiguration time is being minimized by the algorithm, parallel motion has definitely an advantage over serial motion. On the contrast, when total energy consumption or numbers of moves are being minimized, parallel motion do not usually have an advantage over serial motion.

2.4 Modular Self-Reconfiguring Robots

2.4.1 ATRON

ATRON is shown in Figure 2-7 and is developed by (Christensen & Stoy, 2006). This platform is based on a lattice, spherical, 3D, one degree of freedom module. The modules are composed of two hemispheres where one can rotate with respect to the other one. This module weighs around 850g and has a diameter of 11cm.

Each module can connect to other modules through four strong actuated connectors positioned at four sides of the module. The power can also be transmitted through the same connectors. There are also four infrared communication channels below each of these connectors to allow local communication among modules.

There are 100 ATRON modules constructed mainly to explore the idea of using meta-modules to perform reconfiguration.

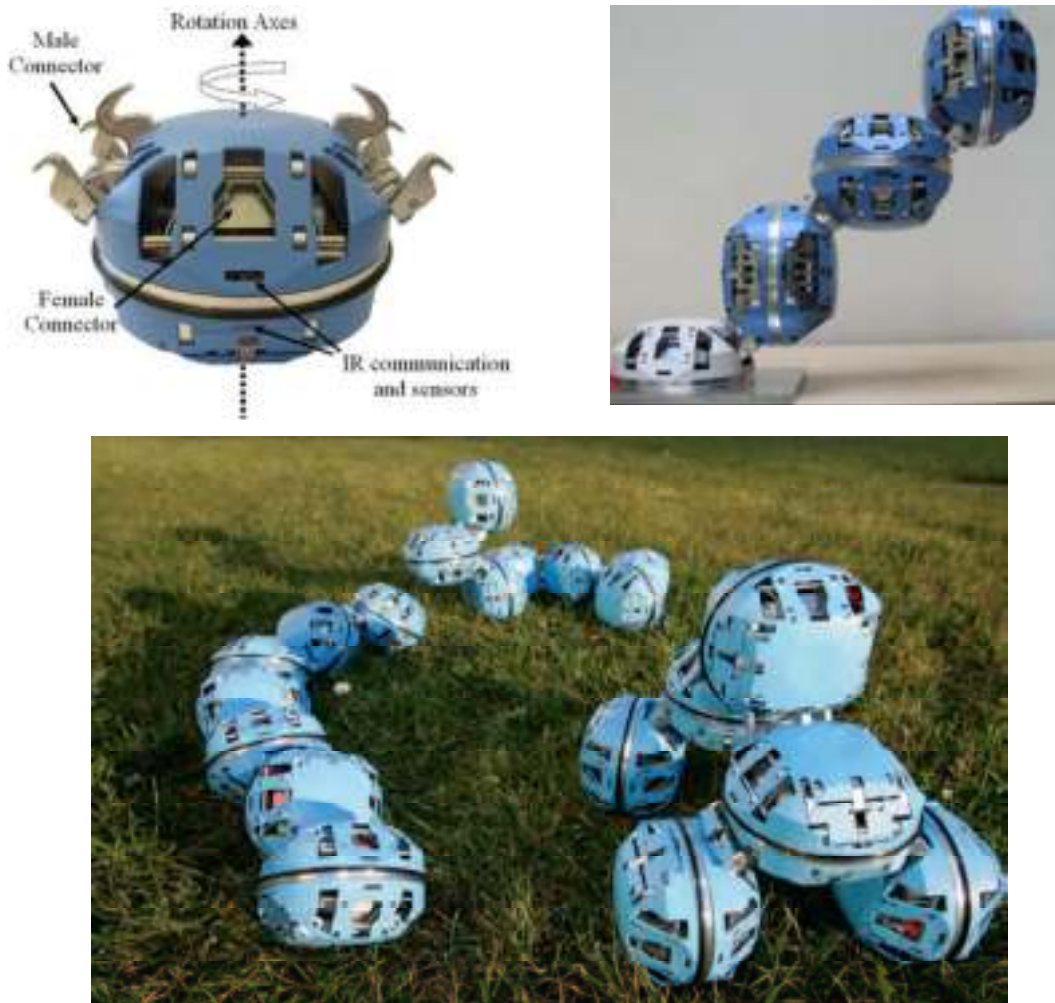


Figure 2-7 ATRON (Christensen & Stoy, 2006)

2.4.2 *Catom*

Catom is shown in Figure 2-8 and is developed by (Goldstein, Mowry, Gibbons, Pillai, Rister, & Lee, 2006). This platform is based on a lattice, circular 2D modules. In this platform gravity holds the individual modules to a surface and there is no connectors connecting the modules together. The actuation force is provided by electromagnets. Catom weighs 105g (50g for the magnets, 55g for everything else) and has a diameter of 44mm and a height of 60mm.

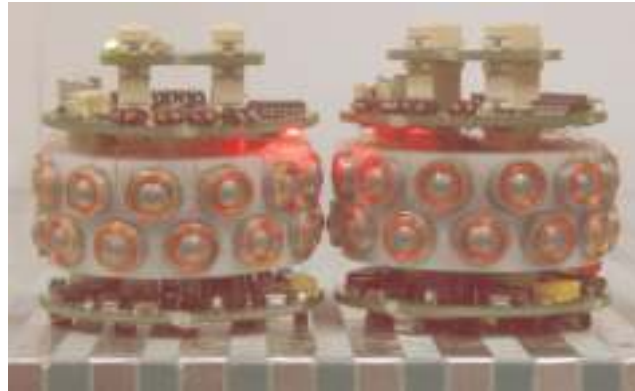


Figure 2-8 Catom (Goldstein, Mowry, Gibbons, Pillai, Rister, & Lee, 2006)

2.4.3 CEBOT

CEBOT stands for Cellular Robot is shown in Figure 2-9 and developed by (Fukuda & Kawakuchi, 1990). CEBOT is a mobile platform composed of heterogeneous modules and works in a 2D plane. Each module is designed for a simple function such as move, bend, rotate and slide.

Several prototypes were developed for this project. In the first prototype Mark I the modules were cubical and had passive connectors on opposite sides. Latching was performed using shape memory alloy (SMA) actuators. In the other prototypes Mark II and Mark IV, the connection was performed using a mechanical hook and also a cone shape mechanical alignment mechanism was added to modules. In Mark III, modules were hexagonal shaped and each side had a connector.

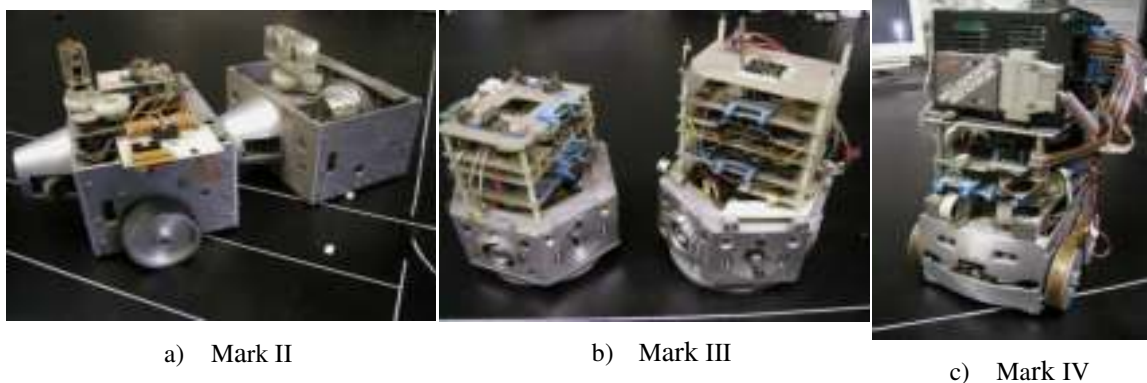


Figure 2-9 CEBOT (Fukuda & Kawakuchi, 1990)

2.4.4 CONRO

The CONRO (CONfigurable RObot) is shown in Figure 2-10 and is developed by (Castano, Behar, & Will, 2002). This platform is based on chain, homogeneous, 3D modules.

Each module is equipped with a female connector on one side and three male connectors on the other side. Shape memory alloys are used to latch the connectors. Each of these connectors has also an infrared transceiver used for local communication. These connectors can rotate with respect to the body in two orthogonal orientations.

The rectangular bounding box of each module is around 10cm x 4.5cm x 4.5cm and the module weighs around 100grams.

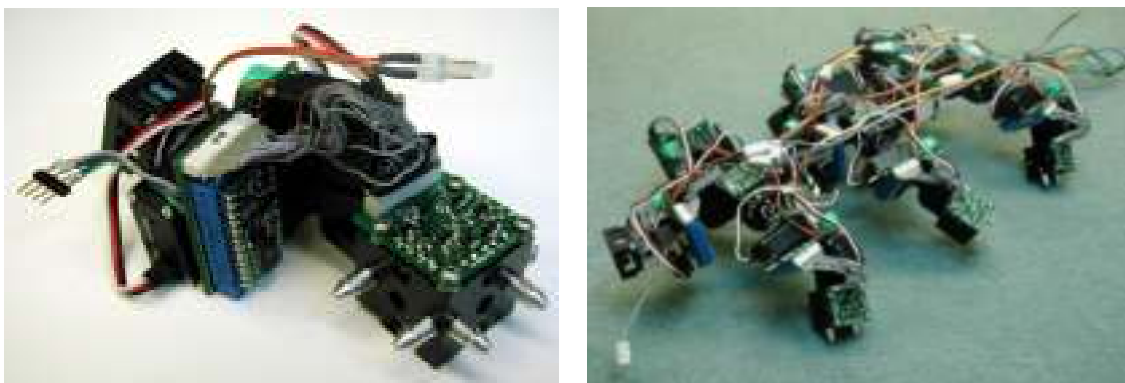


Figure 2-10 CONRO (Castano, Behar, & Will, 2002)

2.4.5 Crystalline

Crystalline robots are shown in Figure 2-11 and developed by (Rus & Vona, 2001). This platform is based on lattice, homogeneous, compressible, cubical, 2D modules.

The main advantage in this platform is that modules are not restricted to move only on the surface of the platform but they can also move through the structure. This is done using an inch-worm method when two neighboring modules scrunch into a single grid space and leave a free space behind for the moving module to pass through it. This mechanism has led to a Melt-Grow type of algorithm where the modules will first Melt to provide space for the moving modules and then Grow.

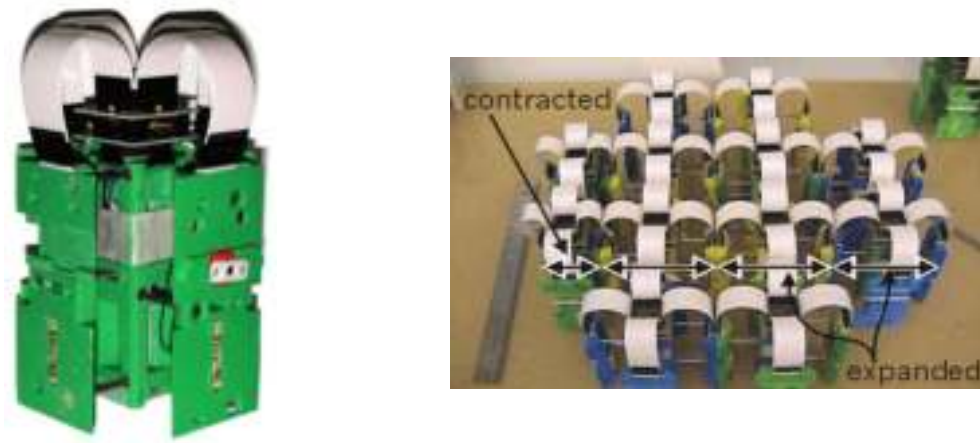


Figure 2-11 Crystalline (Rus & Vona, 2001)

2.4.6 *Fracta / 3D Fracta*

Fracta is shown in Figure 2-12 and developed by (Murata, Kurokawa, & Kokaji, 1994). This platform is based on lattice, homogeneous, rigid, hexagonal, 2D modules.



Figure 2-12 Fracta (Murata, Kurokawa, & Kokaji, 1994)

The platform was later modified to work in a three dimensional environment as shown in Figure 2-13.

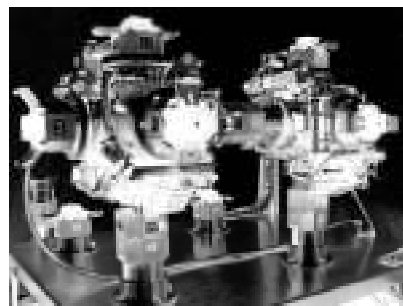


Figure 2-13 3D Fracta (Murata, Kurokawa, Yoshida, Tomita, & Kokaji, 1998)

2.4.7 I-Cubes

I-Cubes platform is shown in Figure 2-14 and developed by (Unsal & Khosla, 2000). This platform is based on three dimensional lattice modules and consists of passive cubes and active links. The links have two male connectors that can connect to cubes and relocate them.



Figure 2-14 I-Cubes (Unsal & Khosla, 2000)

2.4.8 Metamorphic

Metamorphic robots are shown in Figure 2-15 and developed by (Chirikjian G. , 1994). The platform is based on lattice, homogeneous, deformable, hexagonal, 2D modules.

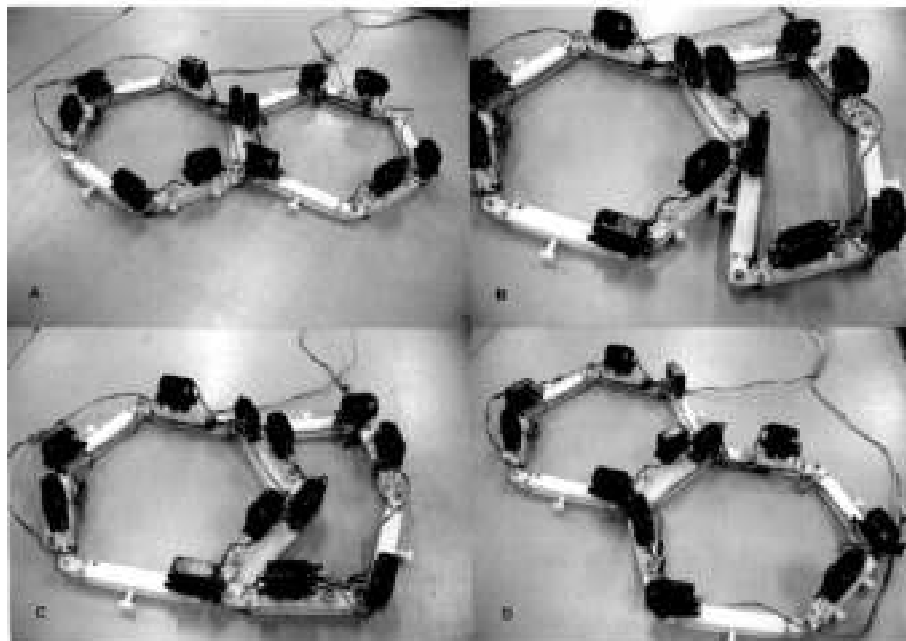


Figure 2-15 Metamorphic (Chirikjian G. , 1994)

2.4.9 Miche

Miche is shown in Figure 2-16 and developed by (Gilpin, Kotay, & Rus, 2007). This platform is based on lattice 3D modules and has demonstrated robust performance for over hundreds of experiments for self-assembly and disassembly. Modules are equipped with magnetic switch connectors and can communicate locally via infrared.



Figure 2-16 Miche (Gilpin, Kotay, & Rus, 2007)

2.4.10 Micro Unit

Micro unit is shown in Figure 2-17 and developed by (Yoshida E. , Murata, Kokaji, Kamimura, Tomita, & Kurokawa, 2002). This platform is based on lattice, homogeneous, 2D modules. The main concept of the design is the miniaturization using shape memory alloy for both actuations and connections. A 3D module can also be implemented combining two of the 2D modules.

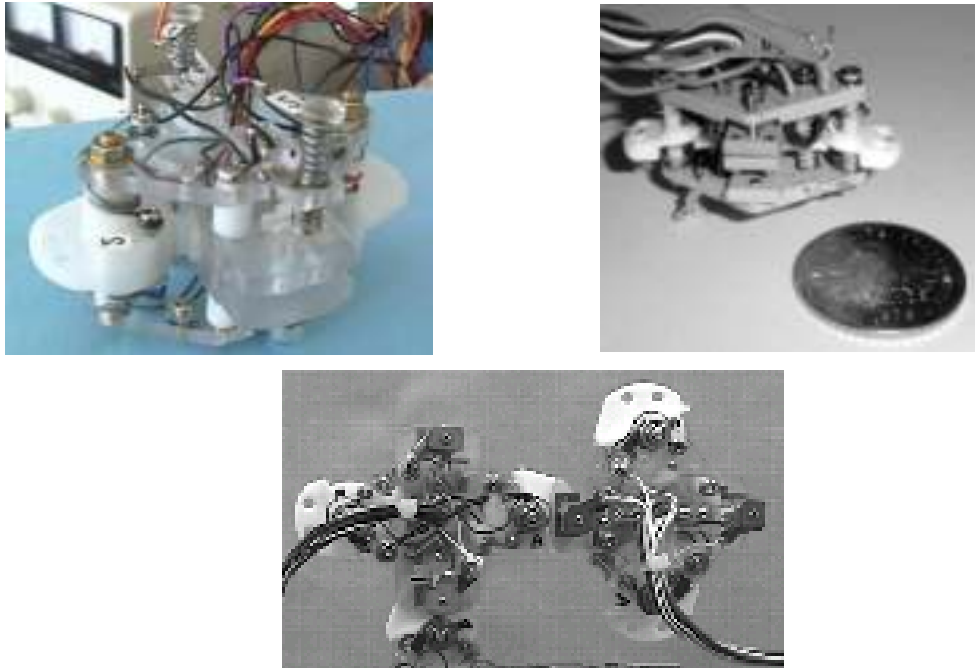


Figure 2-17 Micro Unit (Yoshida E. , Murata, Kokaji, Kamimura, Tomita, & Kurokawa, 2002)

2.4.11 Molecube

Molecube is shown in Figure 2-18 and developed by (Zykov, Mytilinaios, Adams, & Lipson, 2005). This platform is based on chain, homogeneous, cubical, 3D modules. Each module has one degree of freedom normal to its longest diagonal. The inter-module connection is based on electromagnets. Main power is supplied from a power base and it is passed through the face of modules. Each module is around 650g and 10cm long edge. The platform was primarily tested for self-replication algorithms.



Figure 2-18 Molecube (Zykov, Mytilinaios, Adams, & Lipson, 2005)

2.4.12 Molecule

The Molecule is shown in Figure 2-19 and developed by (Kotay K. , Rus, Vona, & McGray, 1998). This platform is based on lattice, homogeneous, spherical, 3D modules. Each Molecule is composed of two spherical atoms linked by a rigid bond connection. Atoms can rotate 180 degrees relative to the bond.

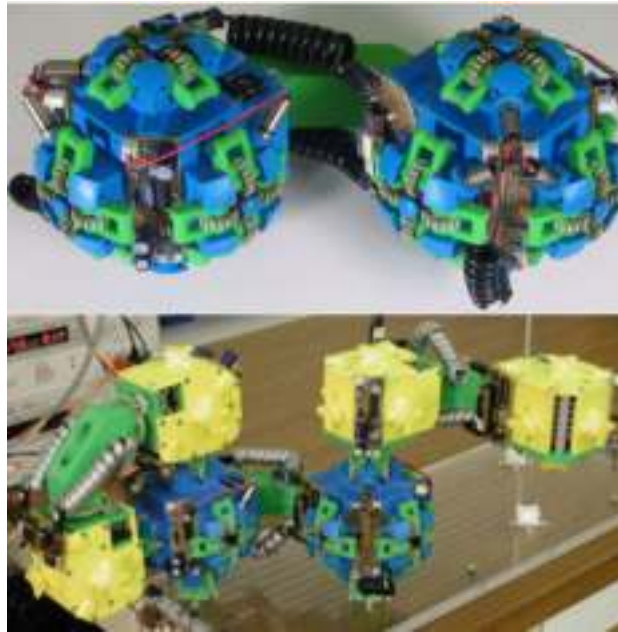


Figure 2-19 Molecule (Kotay & Rus, 2005)

2.4.13 MTRAN

MTRAN (Modular Transformer) is shown in Figure 2-20 and developed by (Murata, Yoshida, Kamimura, Kurokawa, Tomita, & Kokaji, 2002). This platform is based on hybrid, homogeneous, cubical, 3D modules. There are both local and global communication capabilities available in these modules.

2.4.14 Polypod / PolyBot

Polypod is shown in Figure 2-21 and developed by (Yim M. , 1993). The platform is based on chain, rigid, rhombic dodecahedron, 3D modules. There are basically two types of modules in the system: segment and node. Segments have two connections and two degrees of freedom while nodes have six connections and no degrees of freedom.

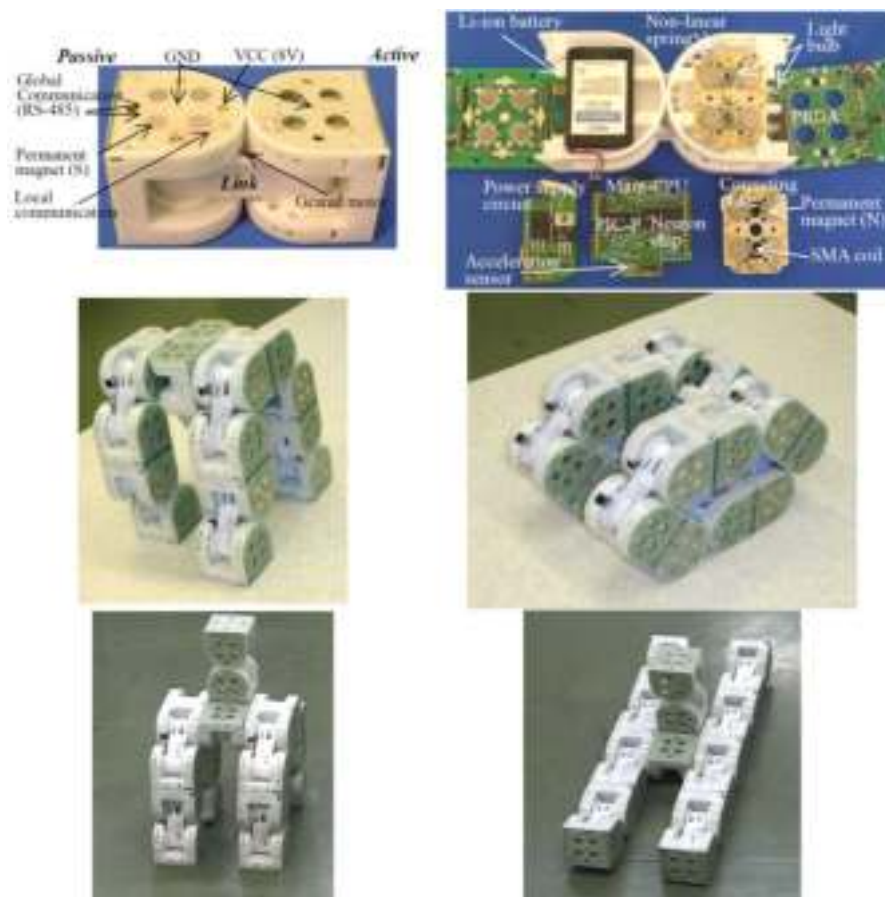


Figure 2-20 MTRAN (Kamimura A. , Kurokawa, Yoshida, Tomita, Murata, & Kolaji, 2003), (Kurokawa, Kamimura, Yoshida, Tomita, Kokaji, & Murata, 2003)

The designed was not considered completely a MSRRS since the connections were not automated and therefore it was followed by its successor PolyBot shown in Figure 2-22 developed by (Yim, Duff, & Roufas, 2000).

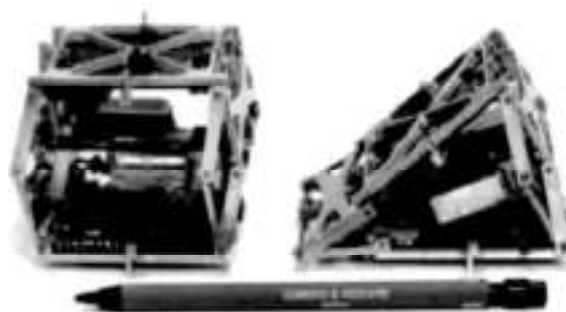


Figure 2-21 Polypod(Yim M. , 1993)

This platform is based on chain, rigid, 3D modules. Each module has one rotational degree of freedom and the inter-module connections are performed using shape memory alloy. Communication is done through IR transceivers. This platform was very successful in demonstrating several locomotion gaits including a rolling type with a 1.6m/s speed.



Figure 2-22 PolyBot (Yim, Duff, & Roufas, 2000)

2.4.15 Programmable Parts

Programmable Parts platform is based on lattice, homogeneous, triangular, 2D modules and is shown in Figure 2-23. This platform is developed by (Klavins, Burden, & Napp, 2006) to basically understand how to program stochastic self-assembly. Modules are moved randomly on an air-table and when they collide with each other they decide whether to stay connected or not based on their local communication and decisions.



Figure 2-23 Programmable Parts(Klavins, Burden, & Napp, 2006)

2.4.16 RIKEN Vertical

RIKEN Vertical is shown in Figure 2-24 and developed by (Hosokawa, et al., 1998). This platform is based on lattice, homogeneous, rigid, cubical, 2D modules. Each module has two degrees of freedom one prismatic and one revolute and the bonding sides are covered with magnetic sheets.

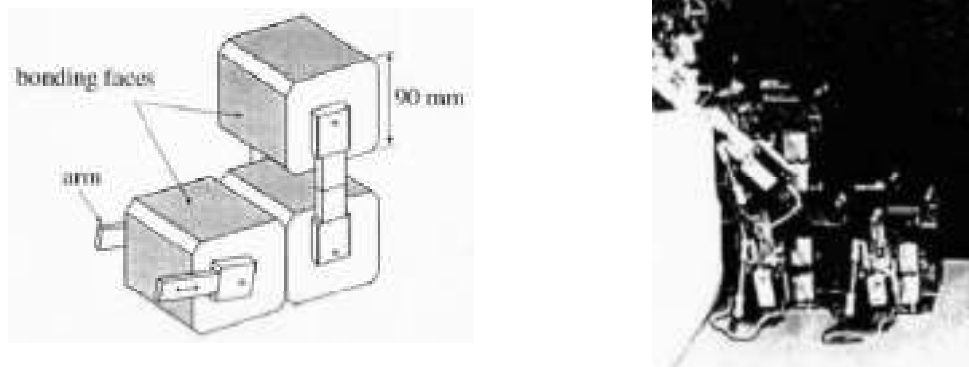


Figure 2-24 RIKEN Vertical (Hosokawa, et al., 1998)

2.4.17 Stochastic (2D/3D)

Stochastic platforms are shown in Figure 2-25 and developed by (White, Kopanski, & Lipson, 2004) for two dimensional and (White, Zykov, Bongard, & Lipson, 2005) for three dimensional. These platforms are based on rigid, triangular/cubical, 2/3D modules.



Figure 2-25 Stochastic (White, Kopanski, & Lipson, 2004), (White, Zykov, Bongard, & Lipson, 2005)

2.4.18 SuperBot

SuperBot is shown in Figure 2-26 and developed by (Shen, Krivokon, Chiu, Everist, Rubenstein, & Venkatesh, 2006). This platform is based on hybrid, homogeneous, 3D modules. The modules have three degree of freedom (pitch, yaw and roll) and the inter-module connection is based on six identical dock connectors which are also used for communication. This platform is primarily developed for real world applications and demonstrated an excellent performance.

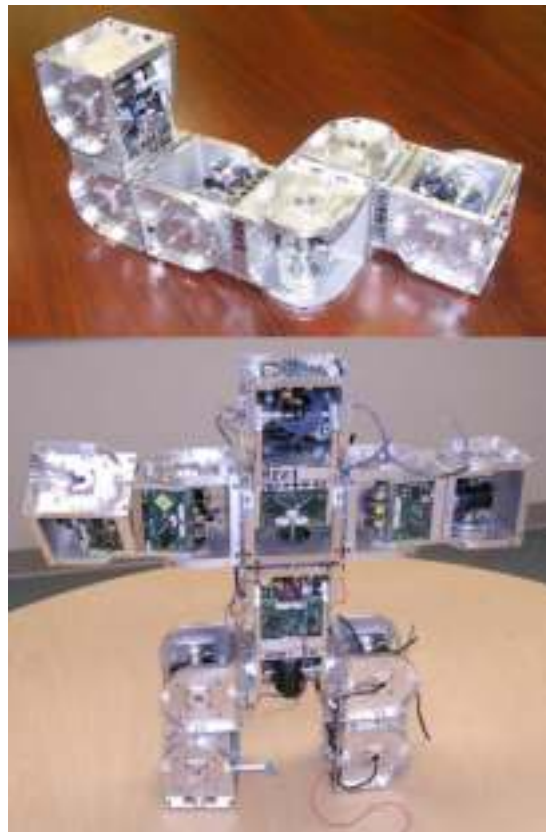


Figure 2-26 SuperBot (Shen, Krivokon, Chiu, Everist, Rubenstein, & Venkatesh, 2006)

2.4.19 Telecube

Telecube is shown in Figure 2-27 and developed by (Suh, Homans, & Yim, 2002). This platform is based on lattice, homogeneous, cubical, 3D modules. It is considered as the 3D version of the Crystalline platform and it uses permanent magnets for connections.



Figure 2-27 Telecube (Suh, Homans, & Yim, 2002)

Chapter 3

HexBot: Physical Platform

Designing a universal module for a MSRRS requires a deep understanding of the ultimate goals in the research area. In 3.4 physical implementation challenges were briefly introduced. In my point of view the main implementation challenge to be overcome is the scalability which in turn enforces other challenges as well. There are primarily two scalability issues 1) reduce the size 2) increase the quantity. Eventually the size of each module should be scaled down to a level that a complete system composed of such modules can form structures with a reasonable resolution. Meanwhile, the number of modules in the system shall be scaled up to fill the required structure.

3.1 Design Criteria

Considering scalability issues and imagining a huge collection of tiny modules working together, the following design criteria were considered:

- 1) Extremely fast actuation for modules so that a large number of them can be moved in an acceptable amount of time
- 2) Quick and strong inter-module connection
- 3) Mass production of the universal module shall be extremely cost effective

- 4) Power consumption shall be as low as possible to keep the system working for a long period of time. Ideally no motion for a module should result in no power required for its actuators or connectors to maintain its status
- 5) Suitable shape to form an arbitrary structure with minimum gaps
- 6) Homogeneity offers module interchangeability which is preferred for self-repair and helps reconfiguration algorithms to perform faster

Taking the above criteria into considerations immediately eliminates the idea of having a regular robot with DC motors and wheels as an individual module to move around its neighbors. Perhaps, ultimately each module should look like a charged particle that can be moved quickly in a magnetic field to fulfill the above requirements.

3.2 Mechanical Design

The module is primarily designed to satisfy the above requirements and the system is developed for two dimensional environments.

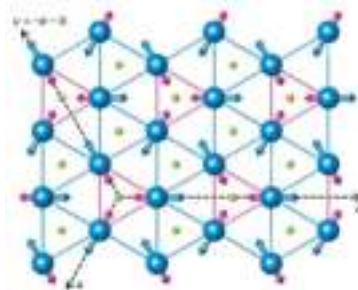
3.2.1 Universal Module

Inspiring from natural structures such as bees nest, crystal molecule structures and etc, it was decided to have hexagonal shaped modules. The main reason behind having hexagons as building blocks of the complete MSRRS is their ability of densely filling structures as shown in Figure 3-1.

Furthermore, the system is designed to be homogenous allowing module interchangeability to make reconfiguration or auto repair faster and easier. Figure 3-2 illustrates the designed and the implemented *HexBot* module.



a) Bees nest



b) Crystal structure of hexagonal RMnO_3

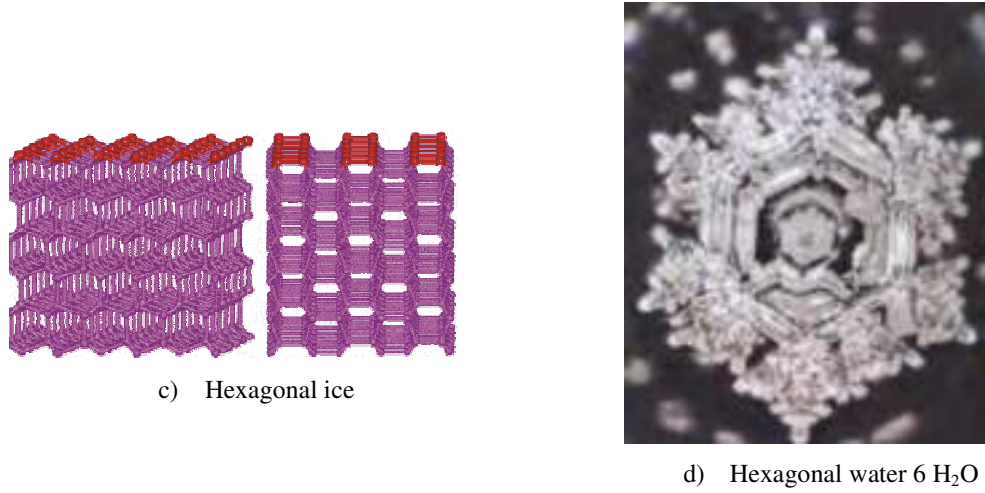


Figure 3-1 Natural Hexagonal Structures

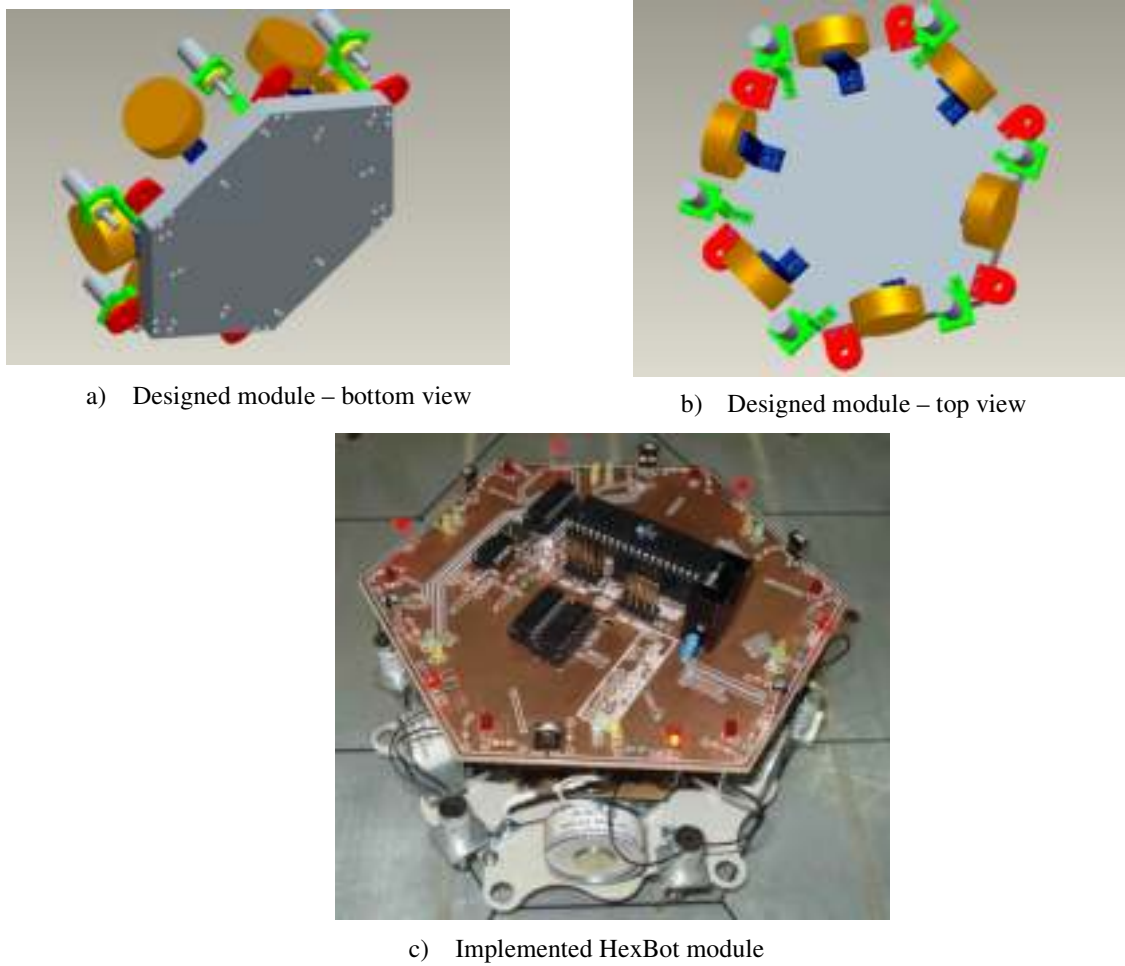
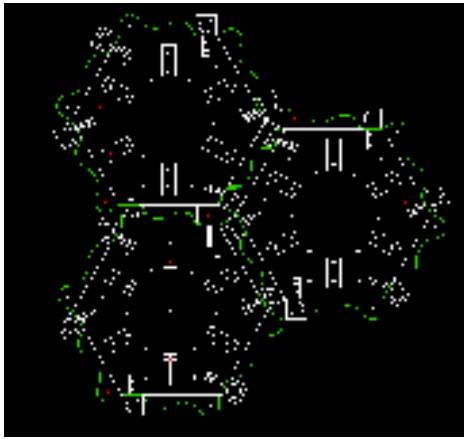


Figure 3-2 HexBot Structure

As shown in Figure 3-3 the edges of the *HexBot* modules were further modified to provide a mechanical self-aligning mechanism.



a) Designed sine-shape curve



b) Implemented mechanical self-aligning mechanism

Figure 3-3 Self-Aligning Mechanism

3.2.2 Actuators

Electromagnets are chosen to provide the required actuation for the modules. This choice provides quick actuation at a relatively cost effective manner.

As shown in Figure 3-3, each side of the module is equipped with an electromagnet E-05-125 from *Magnetic Sensor Systems* with a holding force plotted as a function of input power in Figure 3-4:

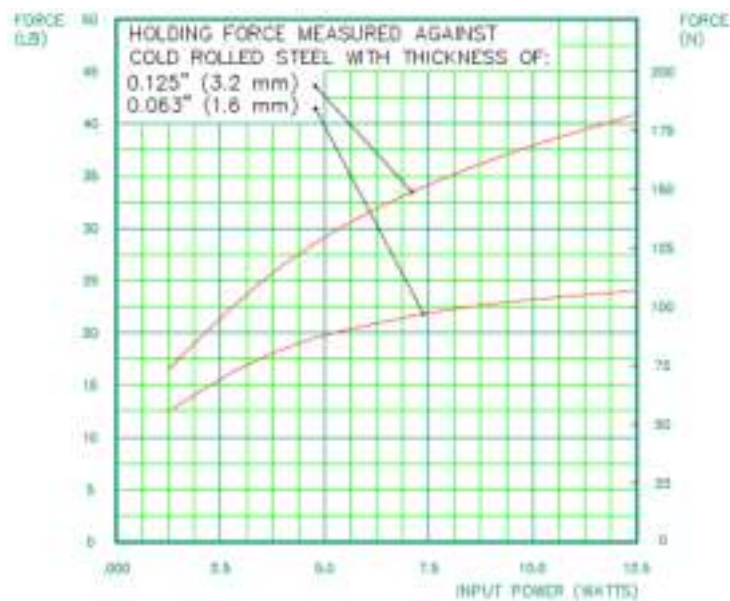


Figure 3-4 Hold Force vs. Input Power

One module by itself is not able to perform any motion; however, a combination of two modules with their magnetic forces makes the motion possible. In order to perform any rotation there will be an initial repulsion between two adjacent sides of the modules, followed by an attraction between the other two sides of the modules as shown in Figure 3-5.

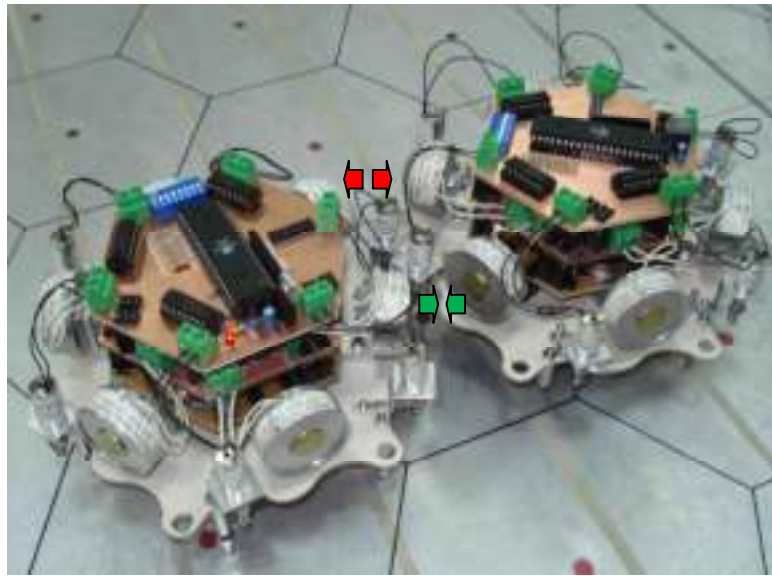


Figure 3-5 HexBot Actuators

As shown in Figure 3-6, each module is equipped with three ball transfer units (passive elements) providing an omni-directional motion the module.



a) Ball transfer units on the module



b) Ball transfer units – bottom view

Figure 3-6 Ball Transfer Units

3.2.3 Inter-Module Connections

One of the most limiting factors for fast module movements is related to inter-module connections. Most designs are suffering from either slow or weak connection

mechanisms. In HexBot, in order to provide a strong as well as quick connection between two modules, pull type solenoids are utilized. Each edge of the module is equipped with a solenoid S-69-38 (active actuated male connector) from *Magnetic Sensor Systems* in addition to a passive female connector as shown in Figure 3-7



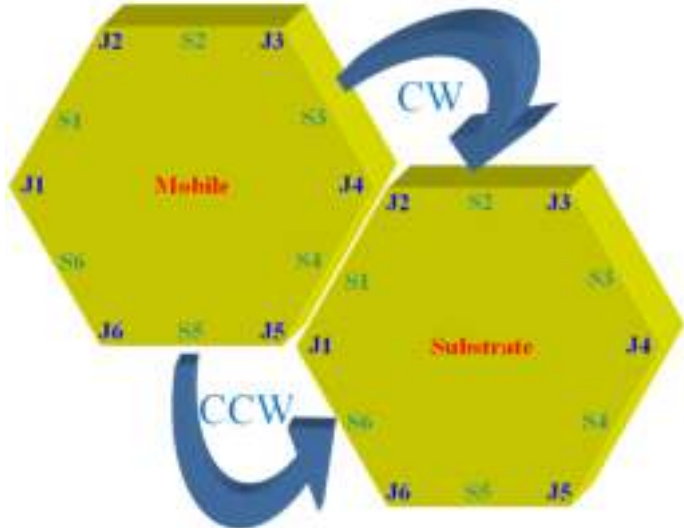
Figure 3-7 Inter-module Connections

3.2.4 Motion through Rotation

The relocation of a module is provided by its rotation around its neighboring modules. As mentioned before, one module by itself is not able to perform any motion; however, a combination of two modules with their magnetic forces makes the motion possible. Moreover, there should be some precise delays and timings for the actuation of the electromagnets and solenoids in order to perform a consistent and complete rotation. The embedded microcontroller of HexBot specifies these delays and controls the actuators accordingly. These timings are sent manually to the modules and were calculated based on the dynamic model of the system.

Table 3-1 specifies the required steps for the mobile module to rotate around its substrate when there are no other modules in their neighborhood. In this platform one complete rotation takes nearly 220 ms with an angular speed of 9.52 rad/s. This speed is considered to be relatively fast compared to other platforms.

Table 3-1 Required Rotation Steps (Only 2 modules with no other neighbors)

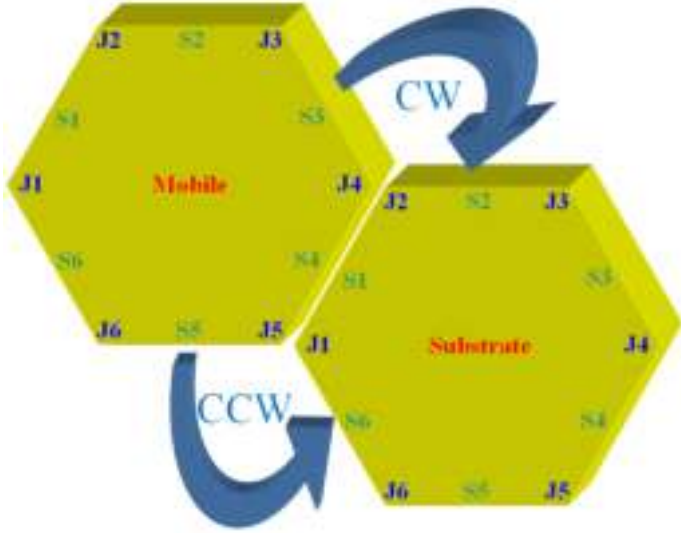


Steps	Rotation CW (around joint $i=4$)		Rotation CCW (around joint $i=5$)		No Neighbors	No Neighbors
	Mobile	Substrate	Mobile	Substrate		
-	All J_x Close	All J_x Close	All J_x Close	All J_x Close		
-	All S_x Off	All S_x Off	All S_x Off	All S_x Off		
1	J_{i+1} Open	J_{i-3}, J_{i-2} Open	J_i, J_{i-1} Open	-		
2	Delay $t_1 = 80$ ms	Delay $t_1 = 80$ ms	Delay $t_1 = 80$ ms	Delay $t_1 = 80$ ms		
3	S_{i+1}, S_i -ve	S_{i-3} -ve, S_{i-2} +ve	S_{i-1}, S_i -ve	S_{i+2} -ve, S_{i+1} +ve		
4	Delay $t_2 = 45$ ms	Delay $t_2 = 40$ ms	Delay $t_2 = 40$ ms	Delay $t_2 = 45$ ms		
5	J_i Open	All J_x Close	J_i Close	J_{i+1}, J_{i+2} Open		
6	Delay $t_3 = 90$ ms	Delay $t_3 = 95$ ms	Delay $t_3 = 95$ ms	Delay $t_3 = 90$ ms		
7	All J_x Close	-	All J_x Close	All J_x Close		
8	All S_x Off	All S_x Off	All S_x Off	All S_x Off		

Note that before any module movements all actuators (sides and joints) are off and after completing the rotation they will be switched off as well. This will ensure minimum power consumption for modules when the system is not changing. The delay timings (t_1 , t_2 and t_3) are set such that the required actuations take place precisely. These timings are sent manually to the modules as will be explained at the end of this chapter.

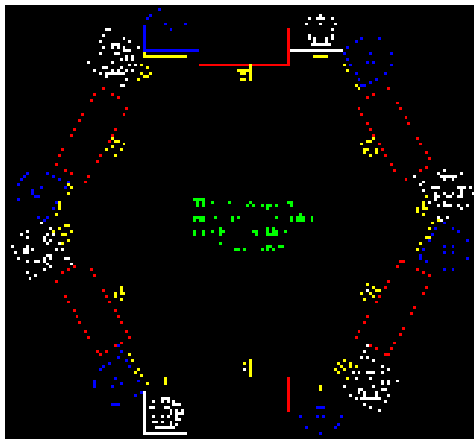
Table 3-2 specifies the same rotation steps required to be taken if there exist other modules in the neighborhood apart from the substrate.

Table 3-2 Required Rotation Steps (2 modules with other neighbors)

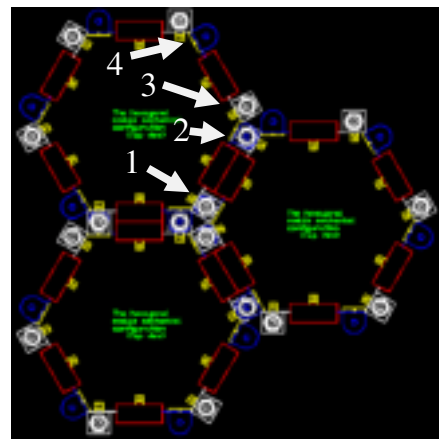


Steps	Rotation CW (around joint $i=4$)			Rotation CCW (around joint $i=5$)		
	Mobile	Substrate	Other Neighbors	Mobile	Substrate	Other Neighbors
-	All J_x Close	All J_x Close	All J_x Close	All J_x Close	All J_x Close	All J_x Close
-	All S_x Off	All S_x Off	All S_x Off	All S_x Off	All S_x Off	All S_x Off
1	All J_x except J_i Open	J_{i-3}, J_{i-2} Open	J_n Open	All J_x Open	-	J_n Open
2	Delay $t_1 = 80$ ms	Delay $t_1 = 80$ ms	-	Delay $t_1 = 80$ ms	Delay $t_1 = 80$ ms	-
3	S_{i-1}, S_i -ve	S_{i-3} -ve, S_{i-2} +ve	-	S_{i-1}, S_i -ve	S_{i+2} -ve, S_{i+1} +ve	-
4	Delay $t_2 = 45$ ms	Delay $t_2 = 40$ ms	-	Delay $t_2 = 40$ ms	Delay $t_2 = 45$ ms	-
5	J_i Open	All J_x Close	-	J_i Close	J_{i+1}, J_{i+2} Open	-
6	Delay $t_3 = 90$ ms	Delay $t_3 = 95$ ms	-	Delay $t_3 = 95$ ms	Delay $t_3 = 90$ ms	-
7	All J_x Close	-	J_n Close	All J_x Close	All J_x Close	J_n Close
8	All S_x Off	All S_x Off	-	All S_x Off	All S_x Off	-

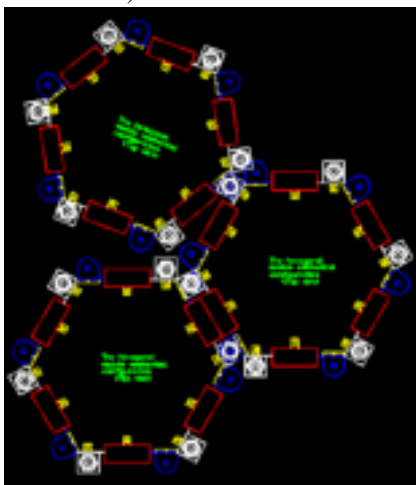
Figure 3-8 illustrates these different steps required to perform a single module rotation. As can be seen in this figure, during each rotation both solenoids (mobile and substrate) are utilized.



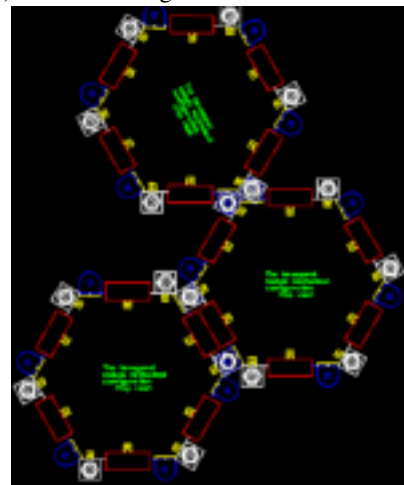
a) Individual module



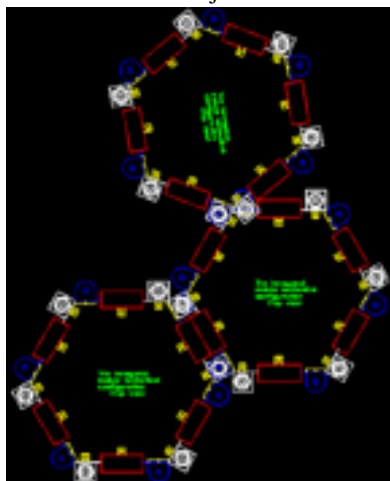
b) Initial configuration – Joints 3 and 4 open



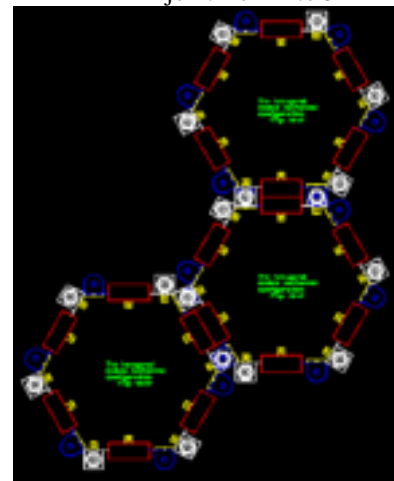
c) Joints 1, 3 and 4 open – Rotation around joint 2



d) Joints 1 and 4 open – Switching the pivot joint from 2 to 3



e) Joints 1, 2 and 4 open – Rotation around joint 3



f) Joints 1 and 2 open – Fixing the module by joints 3 and 4

Figure 3-8 HexBot Motion

3.3 Electrical System

The electrical system is designed based on the mechanical platform requirements. This system is basically responsible to get the power from a power base and switch on and off the required actuator or joint based on the control algorithm. The design is based on a modular multilayer approach as shown in Figure 3-9. Such an approach is extremely open and flexible, since each module (or layer) can be modified individually. At the same time, when there is a failure in the system, only the faulty module needs to be replaced.

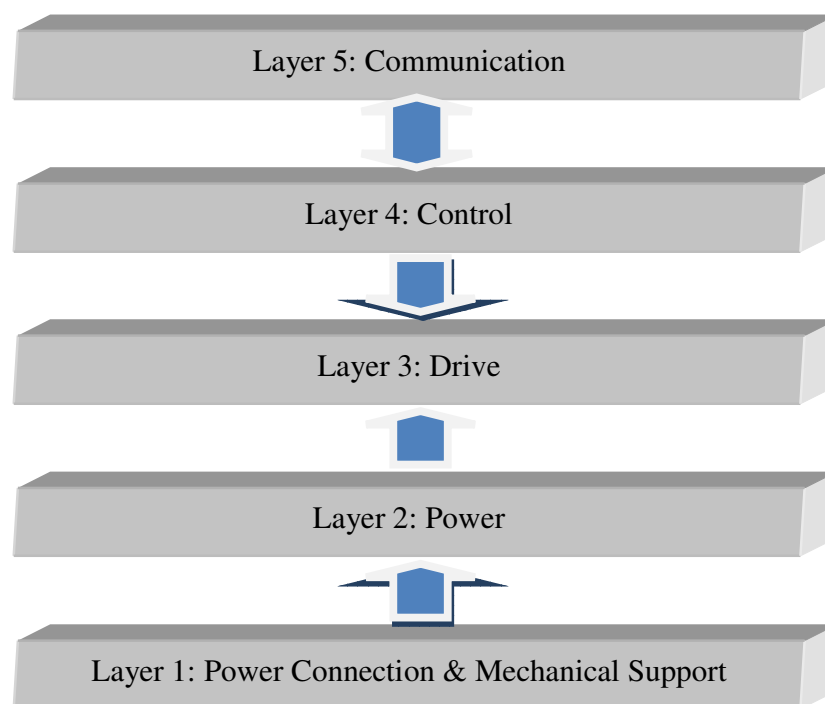


Figure 3-9 Multilayer Electrical System Design for HexBot

The first layer is primarily responsible to provide the module with the main power supply, and it also acts as a mechanical support for joints and actuators. The second layer receives the main power from layer 1 and regulates it into different voltages required for the system. Upon receiving a control signal from layer 4, layer 3 will drive required actuators and solenoids. Layer 5 is mainly dedicated for both inter-module and centralized user-module communication. Each of these layers is briefly discussed below.

3.3.1 Design

The required circuits to be designed were the followings:

- a) Power
- b) Drive
- c) Control
- d) Communication

A proper power circuit design plays an important role in the electrical system since all actuators are inductive loads and therefore generate huge noise during transitions. If the noise is not filtered properly it can damage other components of the system and it may also cause the complete system fail by restarting the microcontrollers during the transitions. The circuit was designed as illustrated in Figure 3-10.

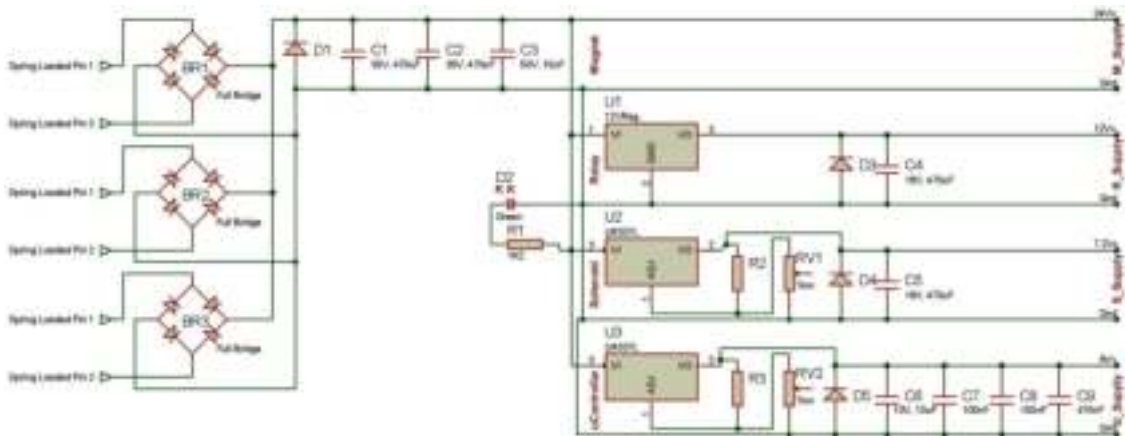


Figure 3-10 Power Circuit Schematic

Next a simple drive circuit was designed as shown in Figure 3-11. As can be seen, electromagnets are switched on and off with different polarities using relays and the coils of the relays are actuated through transistor arrays. Solenoids are switched on and off through the use of a transistor array directly. Each solenoid uses three channels of a transistor array to obtain sufficient current for actuation.

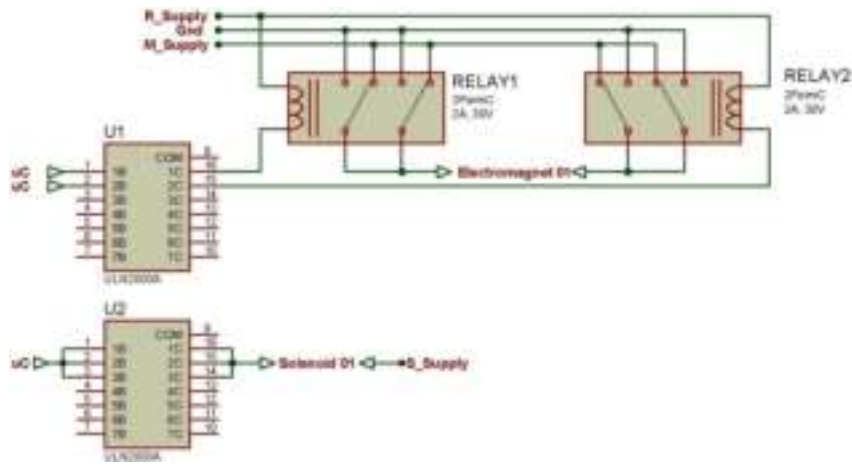


Figure 3-11 Drive Circuit Schematic

The control and communication microcontrollers were designed as shown in Figure 3-12.

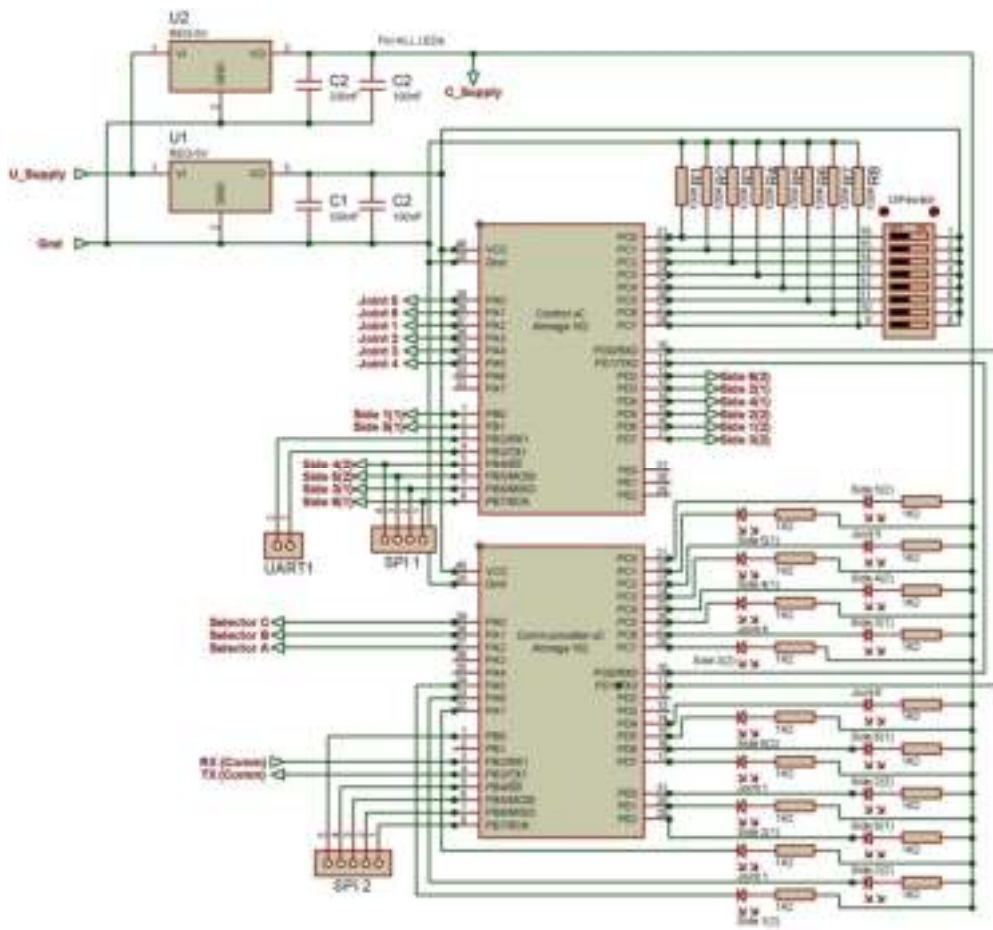


Figure 3-12 Microcontroller Design Schematic

More details about the microcontrollers are provided in the 3.4.

Finally the communication board (based on infrared) was designed as illustrated in Figure 3-13.

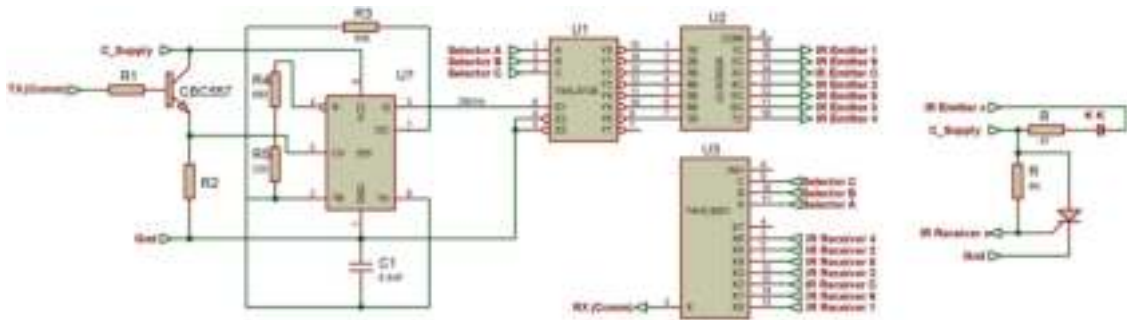


Figure 3-13 IR Communication Schematic

In this design a single serial port of the communication microcontroller with a transmit (TX) and a receive (RX) pins is connected to seven infrared transceivers using a multiplexer and a demultiplexer. The signals are modulated (38kHz) to eliminate the environment noise.

All these designs were tested on bread boards as shown in Figure 3-14 before proceeding to the printed circuit board (PCB) design.

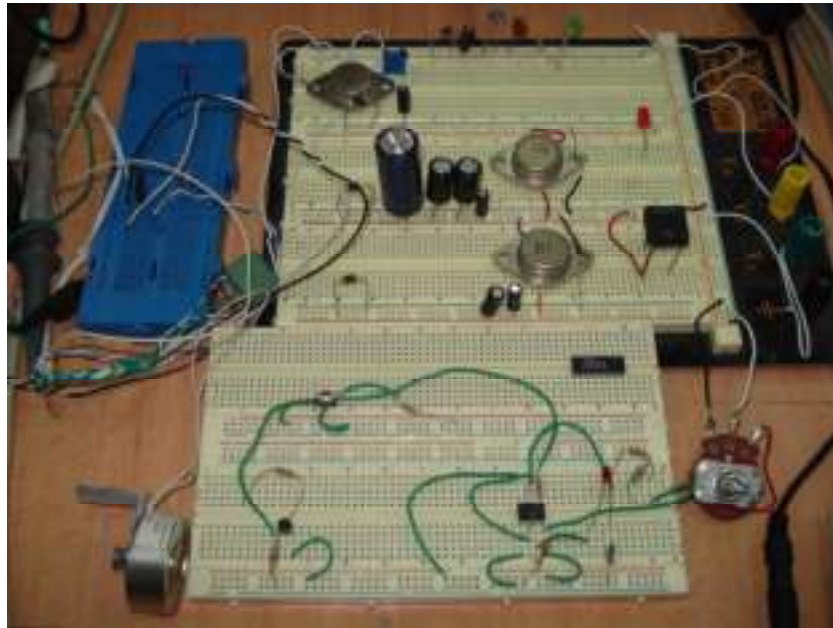


Figure 3-14 Breadboard Verification

In the remaining of this chapter all different components of the electrical system are explained in details.

3.3.2 Power Base

In order to provide power to the modules a power base is designed as shown in Figure 3-15.

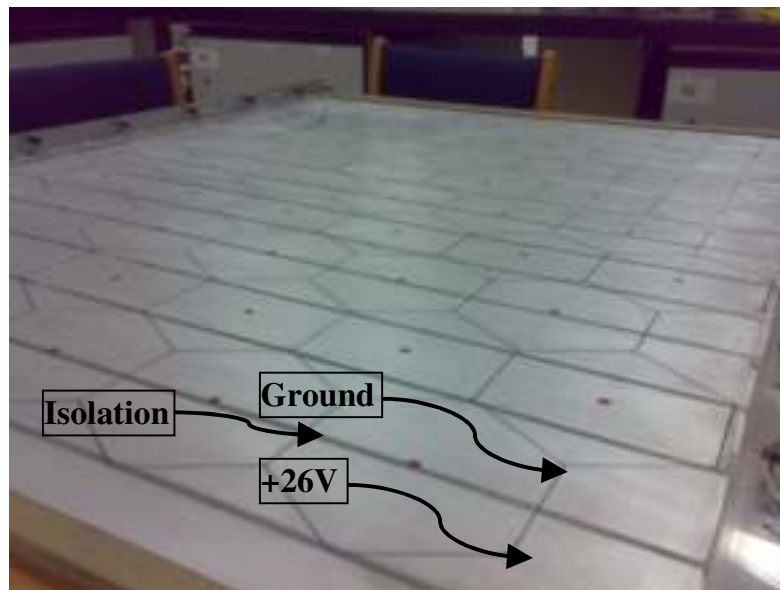


Figure 3-15 Power Base

The based is composed of strips of aluminum connected alternatively to positive and negative of the main supply. Modules can freely move on the base using their ball transfer units and get the required power from the base. Providing power form the base helps the modules to be lighter by not carrying a separate battery. Moreover, the base can also be used to illustrate the coordinate system in addition to location and orientation of modules as shown in Figure 3-16.

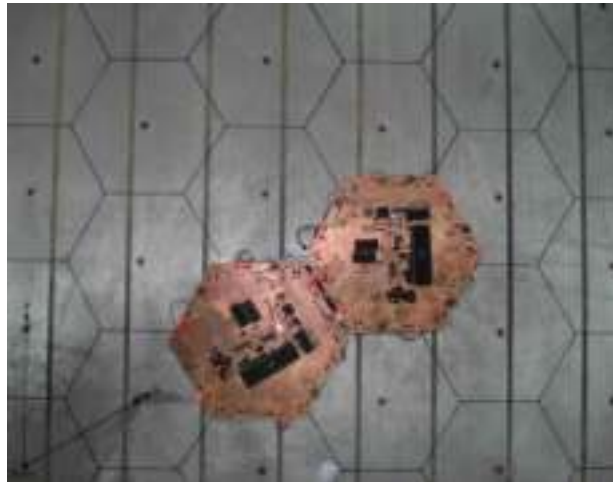


Figure 3-16 Grids and Coordinate System

3.3.3 Layer 1 – Power Connection & Mechanical Support

This layer is a single sided printed circuit board (PCB) dedicated to transfer the main power supply from the base to the power unit. The shape of this PCB is also designed to match the physical module shape to act as a mechanical support for module components. This layer is designed in AutoCAD as illustrated in Figure 3-17.

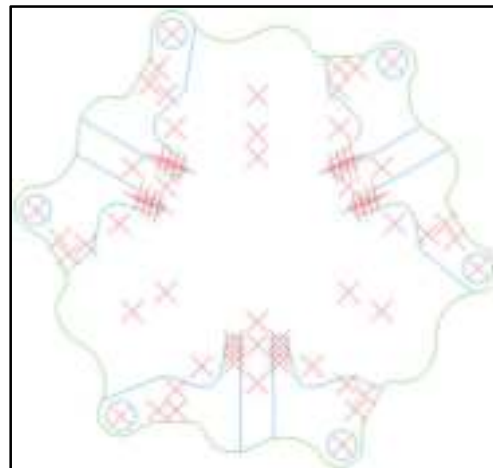


Figure 3-17 Base Layer Design

As can be seen in Figure 3-17, the outer shape of the PCB forms the base of the module and the holes (specified by crosses) are drilled to mount the components such as ball transfer units, electromagnets and solenoids. The six holes located at the edges of the board are also the passive female connectors for solenoids as can be seen in Figure 3-18.

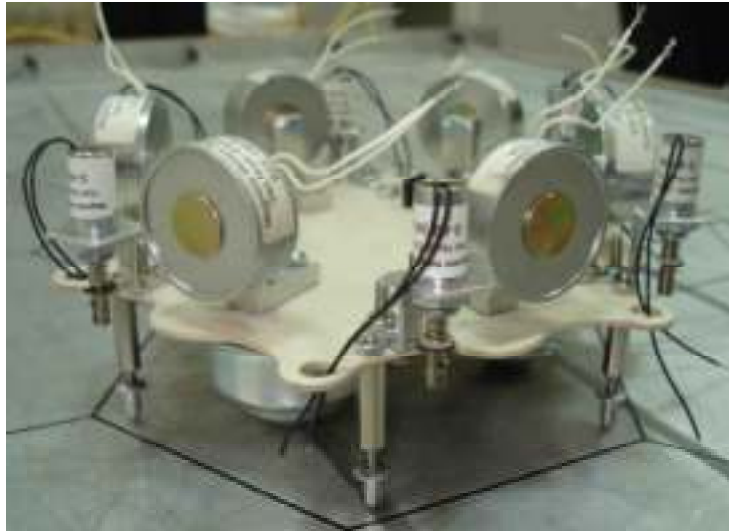


Figure 3-18 Manufactured Base Layer

Figure 3-19 illustrates the spring-loaded pins located at the edges of the module transferring the main power supply to the power unit.



Figure 3-19 Power Transfer to the Module

The width of the power strips on the ground and their spacing is designed in a way to ensure that at each location and orientation at least one of the six spring-loaded pins contacts the negative supply and at least another one contacts the positive supply.

3.3.4 Layer 2 – Power Unit

Each spring-loaded pin is either connected to the positive supply or negative supply or neither of them (if connected to the isolating material). The power unit is equipped with three full bridges connected to these pins and provides a positive and negative supply. This supply is further divided for the following resources:

- 1) Electromagnet Supply: 24V, 4A
- 2) Solenoid Supply: 6V, 500mA
- 3) Drive Circuit Supply: 12V, 500mA
- 4) Microcontroller and Communication Supply: 9V, 1A

Only the drive circuit supply is permanently fixed to 12V. Other supplies are designed such that they can be manually adjusted if required using trimmers. The supply for the electromagnets is the main supply minus the power dissipation of the full bridges (around 1.7V) and the supplies for the solenoid and microcontroller can be adjusted using two trim potentiometers.

The PCB is designed as shown in Figure 3-20.

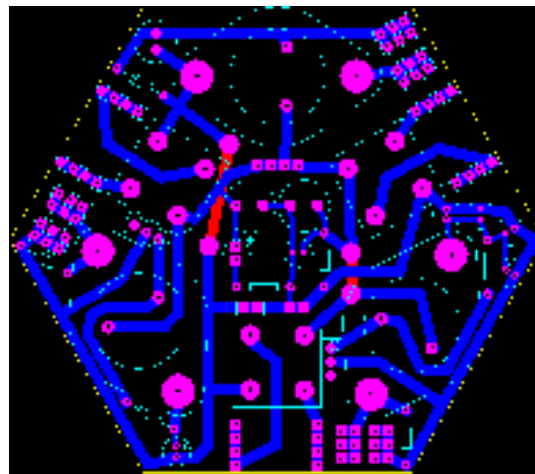


Figure 3-20 Power Circuit Design

The circuit is manufactured as shown in Figure 3-21. All supplies are perfectly filtered because of the inductive nature of the actuators and their corresponding transition noises.



a) Power unit



b) Power unit installed on the module

Figure 3-21 Manufactured Power Unit

3.3.5 Layer 3 – Drive Circuit

In order to switch on each electromagnet that requires high current, a pair of relays each with two parallel form C contacts is used. The control signals coming from the control board are switching the relays through a transistor array. The PCB is designed as shown in Figure 3-22.

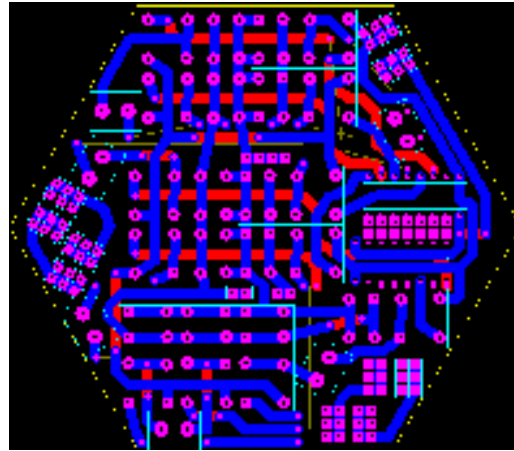


Figure 3-22 Drive Circuit Design

Note that the drive circuit is designed to drive only the electromagnets not the solenoids. The solenoids do not require high power and can be switched directly from the microcontroller through a transistor array. The PCB is manufactured in a double sided board as shown in Figure 3-23.



a) Drive circuit



b) Drive circuit installed on the power unit



c) Drive circuit installed on the module

Figure 3-23 Manufactured Drive Circuit

3.3.6 Layer 4 – Control Board

ATMEGA 162 of the AVR family from Atmel is chosen to send control signals to the drive circuit and actuate the required solenoids and magnets. This microcontroller comes with two serial ports, one port is linked to the other microcontroller on the communication board to receive and execute the commanded functions. The other port is preserved for testing and debugging purposes. The PCB was designed as shown in Figure 3-24.

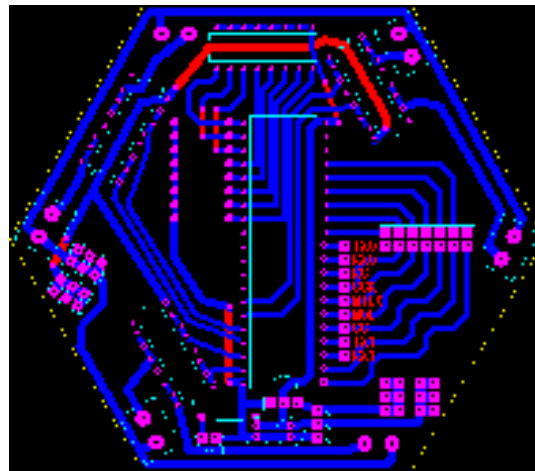
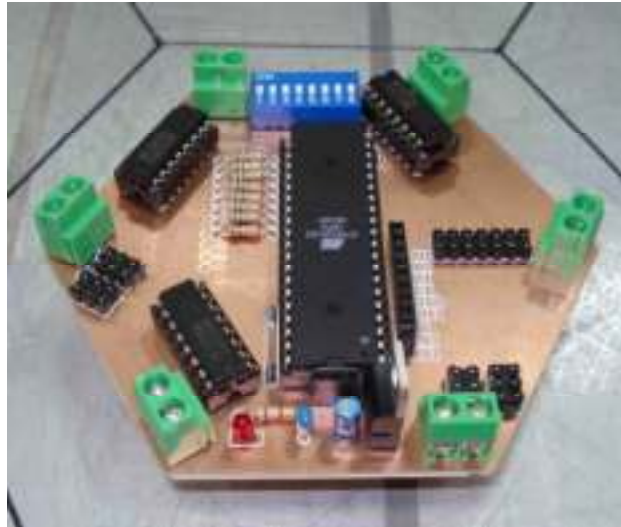


Figure 3-24 Control Board Design

The PCB is manufactured as shown in Figure 3-25. There are eight configuration micro switches incorporated into this board that can be used to configure different functionality of the board such as timings, operation modes and etc.

3.3.7 Layer 5 – Communication

The last layer is dedicated to provide both inter-module communication and central communication to the PC. This layer is equipped with seven infrared transceivers; six of them are located on the sides of the hexagonal module and one is located in the center of the module for central communication. One serial port of the ATMEGA 162 microcontroller on this board is connected to all the seven infrared transceivers through a set of multiplexer and demultiplexer. The other serial port of the microcontroller is linked to the control board microcontroller to send the required control commands.



a) Control board



b) Control board installed on drive circuit



c) Control board installed on the module

Figure 3-25 Manufactured Control Board

The communication board comes also with two Serial Peripheral Interface (SPI) ports; one for the control board microcontroller and the other for the communication board microcontroller. Both microcontrollers can be easily reprogrammed by In-System Programming (ISP) feature through these ports. One of the SPI ports can also be utilized for wireless communication which is planned to be added in the next stage of the project. The board was designed as shown in Figure 3-26.

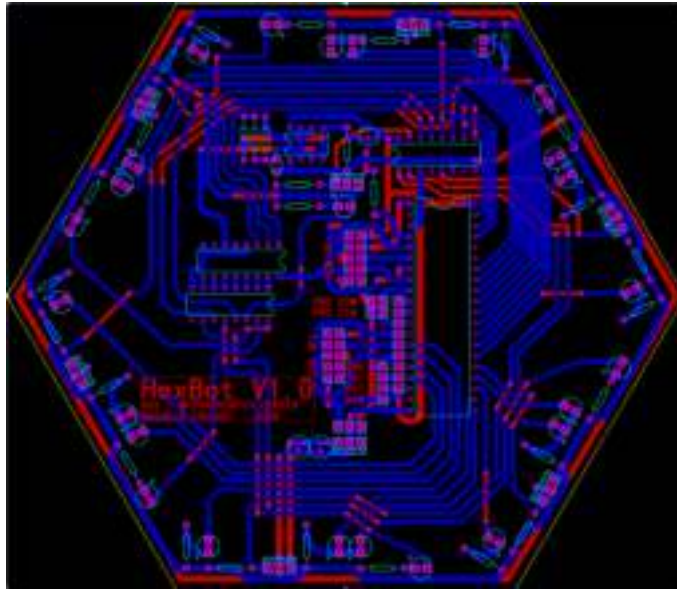


Figure 3-26 Communication Board Design

The PCB is manufactured as shown in Figure 3-27.



a) Communication board



Communication board installed

Figure 3-27 Manufactured Communication Board

As can be noticed the communication board is sized to be exactly the same size as the power base grids.

As shown in Figure 3-28 all these different layers are connected through male headers and can easily be disconnected or replaced for maintenance and future modifications.

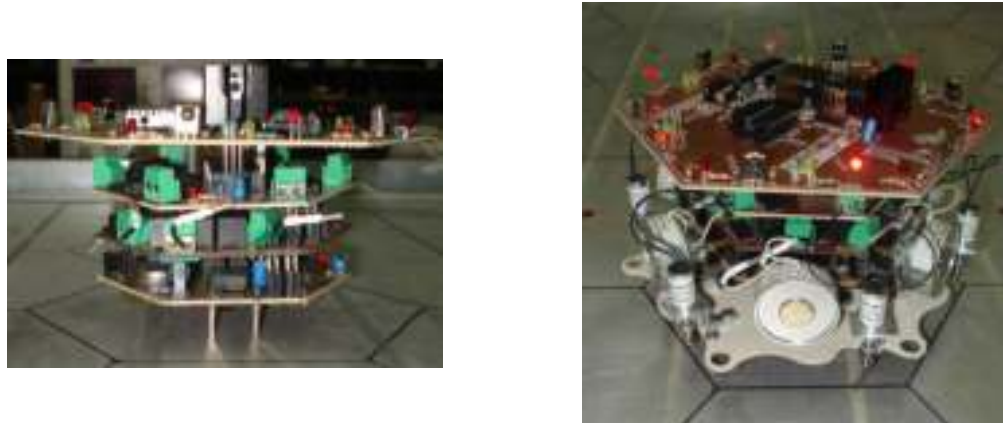


Figure 3-28 Multilayer PCBs

3.4 Processing Unit

The processing unit in the HexBot is primarily responsible to provide:

1. Inter-module communication
2. Centralized communication with the PC
3. Activate the solenoids to open or close the joints
4. Actuate the magnets to provide motion
5. Indicate the status of the module using several LEDs

3.4.1 Microcontrollers

Two ATMEGA 162 microcontrollers from Atmel form the core for control and communication of HexBot. One is located in the control board and the other one in the communication board. These microcontrollers are responsible for actuation and communication (inter-module and centralized) and have the following features:

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
 - 131 Powerful Instructions – Most Single-clock Cycle Execution
 - 32 x 8 General Purpose Working Registers
 - Fully Static Operation
 - Up to 16 MIPS Throughput at 16 MHz
- High Endurance Non-volatile Memory segments
 - 16K Bytes of In-System Self-programmable Flash program memory

- 512 Bytes EEPROM
- 1K Bytes Internal SRAM
- In-System Programming by On-chip Boot Program
- JTAG (IEEE std. 1149.1 Compliant) Interface
- Peripheral Features
 - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
 - Two 16-bit Timer/Counters with Separate Prescalers, Compare Modes, and Capture Modes
 - Real Time Counter with Separate Oscillator
 - Six PWM Channels
 - Dual Programmable Serial USARTs
 - Master/Slave SPI Serial Interface
- Special Microcontroller Features
 - Power-on Reset and Programmable Brown-out Detection
 - Internal Calibrated RC Oscillator
- I/O and Packages
 - 35 Programmable I/O Lines

More details about Atmega162 microcontrollers are available in the datasheets available on Atmel website.

Table 3-3 shows the port configuration for the communication microcontroller. The first column is the microcontroller pin number and second column corresponds to the port number. Column three indicates connections between this layer (communication layer) and other layers which in this case is only the control layer. The last column specifies the assigned functionality of each pin.

Table 3-3 Port Configuration (Communication Microcontroller)

Microcontroller		Communication Board	
		Connections	Functions
Pin	Port	Control	
1	PB0		COM SPI (4)
2	PB1		
3	PB2 (RX1)		IR - RX

4	PB3 (TX1)		IR - TX
5	PB4 (SS)		COM SPI (8)
6	PB5 (MOSI)		COM SPI (3)
7	PB6 (MISO)		COM SPI (9)
8	PB7 (SCK)		COM SPI (2)
9	- (RST)		COM RST
10	PD0 (RX0)	Control 9	UART - CONTROL BOARD
11	PD1 (TX 0)	Control 8	UART - CONTROL BOARD
12	PD2		
13	PD3		
14	PD4		Joint 6 LED
15	PD5		Side 6 (2) LED
16	PD6		Side 6 (1) LED
17	PD7		Joint 1 LED
18	- (XTAL1)		
19	-(XTAL2)		
20	-(GND)		
21	PC0		Side 5 (2) LED
22	PC1		Side 5 (1) LED
23	PC2		Joint 5 LED
24	PC3		Side 4 (1) LED
25	PC4		Side 4 (2) LED
26	PC5		Joint 4 LED
27	PC6		Side 3 (1) LED
28	PC7		Side 3 (2) LED
29	PE2		Joint 3 LED
30	PE1		Side 2 (1) LED
31	PE0		Side 2 (2) LED
32	PA7		Joint 2 LED
33	PA6		Side 1 (1) LED
34	PA5		Side 1 (2) LED
35	PA4		
36	PA3		
37	PA2		Selector A
38	PA1		Selector B
39	PA0		Selector C

40	-(VCC)		
----	--------	--	--

Similarly Table 3-4 shows the port configuration for the control microcontroller. Note that this layer is interfaced to both drive and communication layers.

Table 3-4 Port Configuration (Control Microcontroller)

Microcontroller		Control Board		
		Connections		Functions
Pin	Port	Communication	Drive	
1	PB0		Drive 7	Side 1 (1)
2	PB1		Drive 6	Side 5 (1)
3	PB2 (RX1)	Comm 1		UART RX
4	PB3 (TX1)	Comm 2		UART TX
5	PB4 (SS)	Comm 3	Drive 5	Side 4 (2)
6	PB5 (MOSI)	Comm 4	Drive 4	Side 5 (2) - CTR SPI (3) - MOSI
7	PB6 (MISO)	Comm 5	Drive 3	Side 3 (1) - CTR SPI (9) - MISO
8	PB7 (SCK)	Comm 6	Drive 2	Side 6 (1) - CTR SPI (2) - SCK
9	-(RST)	Comm 7		CTR RST
10	PD0 (RX0)	Comm 8		UART - COMM BOARD
11	PD1 (TX 0)	Comm 9		UART - COMM BOARD
12	PD2		Drive 14	Side 6 (2)
13	PD3		Drive 13	Side 2 (1)
14	PD4		Drive 12	Side 4 (1)
15	PD5		Drive 11	Side 2 (2)
16	PD6		Drive 10	Side 1 (2)
17	PD7		Drive 9	Side 3 (2)
18	-(XTAL1)			
19	-(XTAL2)			
20	-(GND)			
21	PC0			DIP 1
22	PC1			DIP 2
23	PC2			DIP 3
24	PC3			DIP 4
25	PC4			DIP 5
26	PC5			DIP 6
27	PC6			DIP 7

28	PC7			DIP 8
29	PE2			
30	PE1			
31	PE0			
32	PA7			
33	PA6			
34	PA5			Joint 3
35	PA4			Joint 4
36	PA3			Joint 2
37	PA2			Joint 1
38	PA1			Joint 6
39	PA0			Joint 5
40	-(VCC)			

3.4.2 Internal Module Connections

The primary communication functionalities of each module are listed below and illustrated in Figure 3-29.

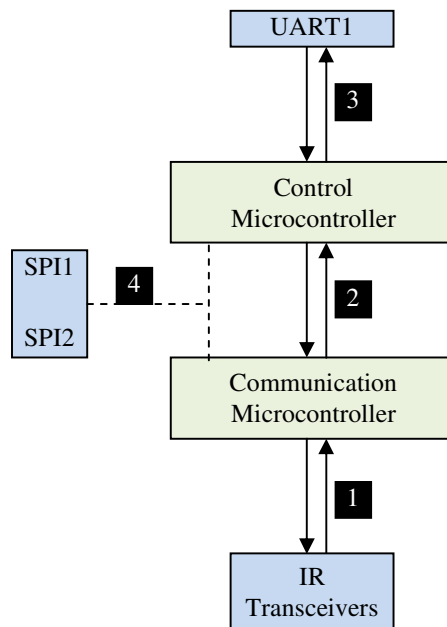


Figure 3-29 Internal Module Connections

1. IR transceivers
2. Communication between the two microcontrollers

3. UART1 connection
4. SPI connection

3.4.3 IR Transceivers

As mentioned earlier in layer 5, infrared (IR) transceivers are used for both inter-module and centralized communication. The infrared emitter is transmitting a 38 kHz modulated signal to reduce the effect of the environmental noise. Similarly the receiver is equipped with band pass filter centered at 38 kHz to reject any other unfavorable signals as illustrated in Figure 3-30.

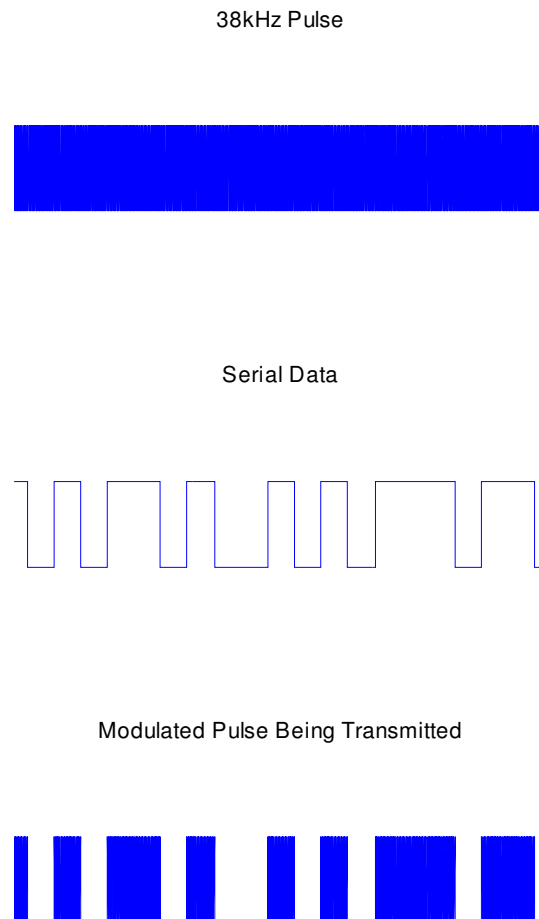


Figure 3-30 Modulated IR Signal

The serial port of the communication microcontroller is connected through a set of multiplexer and demultiplexer to the seven IR transceivers as shown in Table 3-5

Table 3-5 IR Channel Selector

IR TRANSCEIVER									
Selector			RX			TX			
C	B	A	MUX	PIN	SIDE	DEMUX	PIN	TA (PIN)	SIDE
L	L	L	Y0	13	1	Y0	15	16	1
L	L	H	Y1	14	6	Y1	14	15	6
L	H	L	Y2	15	Center	Y2	13	14	Center
L	H	H	Y3	12	2	Y3	12	13	2
H	L	L	Y4	1	5	Y4	11	12	5
H	L	H	Y5	5	3	Y5	10	11	3
H	H	L	Y6	2	4	Y6	9	10	4

In this table the first three columns are the outputs from the microcontroller to specify which channel needs to be selected. The channel shall be either one of the six sides or the center. It is clear that it is not possible to connect the serial port of the microcontroller to more than one channel at a time in this configuration. The next three columns represent the seven IR receivers connection to the microcontroller RX pin through the multiplexer which its pin numbers are listed. The last four columns correspond to the connection of the TX pin of the microcontroller to the demultiplexer which is further connected to the seven IR emitters through a transistor array (TA).

3.4.4 Handshaking

The serial communication between the two microcontrollers is required to provide the transmission of data from the communication layer to the control board to achieve the required actuation. To ensure that this connection is available and active, once the module is switched on, both microcontrollers start the process of handshaking by transmitting specific packets to each other and respond accordingly. If the process is successful the user will find a new module on the graphical user interface mentioning that the module's microcontrollers are ready.

3.4.5 Testing and Debugging

UART1 provides direct access to control board for testing and debugging purposes. It can also be used to send control commands directly to the module without the use of the communication layer.

3.4.6 Reprogramming

SPI1 and SPI2 are used to reprogram the microcontrollers through the In-System Programming (ISP) feature and they can also be used for wireless communication planned to be added in the next stage.

The fuse bits of both microcontrollers shall be set as shown in Table 3-6 before programming. Note that JTAG is not enabled in this configuration since the JTAG pins are used for other purposes.

Table 3-6 Microcontrollers Fuse Bits Settings

Fuse bits: 0XFF, 0XD9, 0X62	
Brown-out detection disabled	BODLEVEL=111
Boot flash size = 1024 words and start address = \$1C00	BOOTSZ=00
Divide clock by 8 internally	CKDIV8=0
Int RC Osc. Start up time: 6CK + 65ms	CKSEL=0010 SUT=10

Reprogramming of the microcontrollers can be done following these steps:

1. Switch off the module by placing it outside the power base
2. Set the programming jumper for the corresponding microcontroller (Figure 3-31 a)
3. Connect the ISP cable from the STK500 to the corresponding SPI port of the module
4. Connect the power cable from the STK500 to the UART1 port of the module
5. Turn on STK500 board
6. Download the program (Ctr+F9)
7. Turn off STK500 board
8. Set the programming jumper back (Figure 3-31 b)

9. Switch on the module by placing it in the power base

Figure 3-31 illustrates an example of how to set the programming jumper to re-program the control microcontroller.



a) Set for programming the control board



b) Set for running the module

Figure 3-31 Setting the Programming Jumper

3.5 Graphical User Interface

A simple Graphical User Interface (GUI) is developed using Visual Basic (VB). This interface is basically used to send different messages to the modules and receive their status.

3.5.1 Communication with the MSRRS

All modules are interfaced to the computer through their IR transceivers as illustrated in Figure 3-29. For testing or debugging purposes one module can also be interfaced to the GUI using its UART1 connection.

From the computer side, two boards were designed to provide communication to modules. One is an IR transceiver (Figure 3-32 a) and the other one is an RS232 converter (Figure 3-32 b).

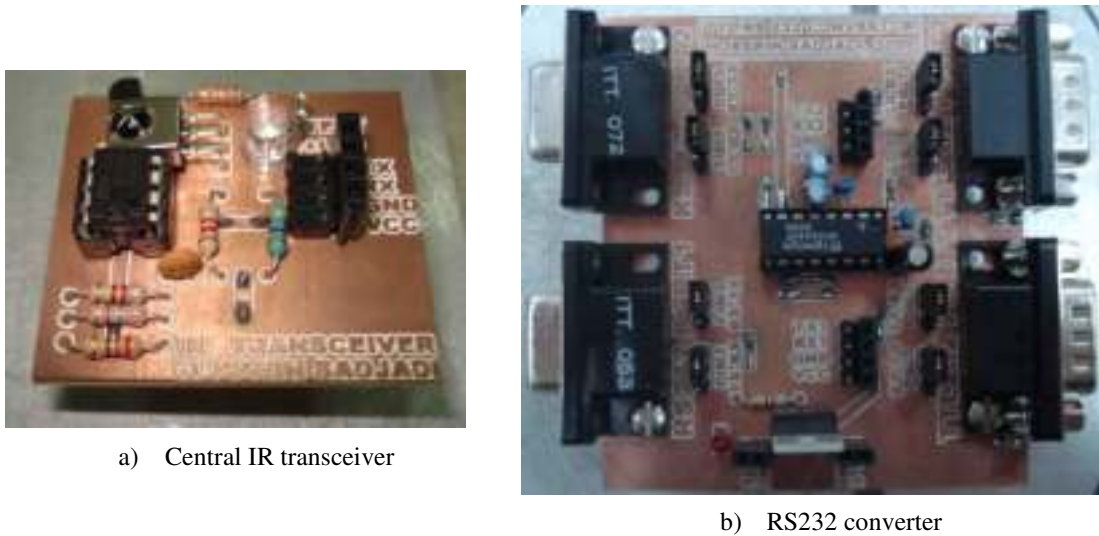


Figure 3-32 PC-MSRRS Communication Components

In order to have the centralized communication, modules shall first switch to their center IR transceivers based on Table 3-5 and then start communicating to the central IR transceiver. The central IR transceiver is also connected to the PC through the RS232 converter as illustrated in Figure 3-33.

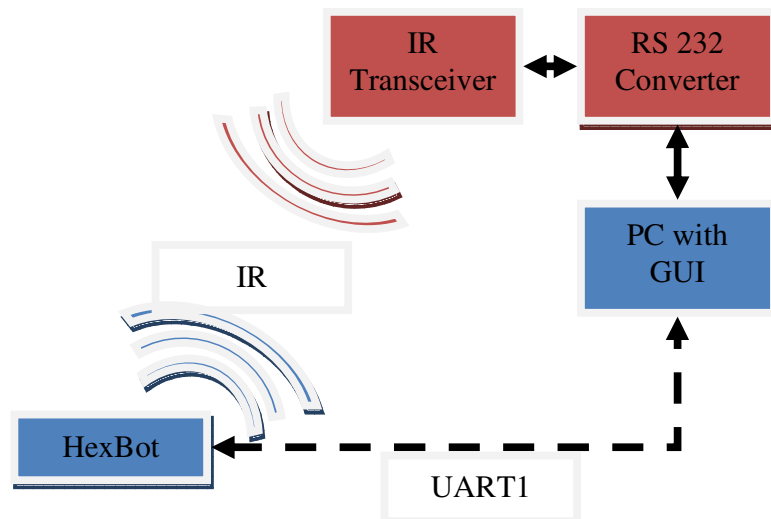


Figure 3-33 PC-MSRRS Communication

Any message sent to a module will receive an acknowledgement from the same module to ensure that the proper function was executed.

5. Function Code Table (down)

The communication frame provides a port configuration window for setting the ports as shown in Figure 3-35 along with connect / disconnect buttons.

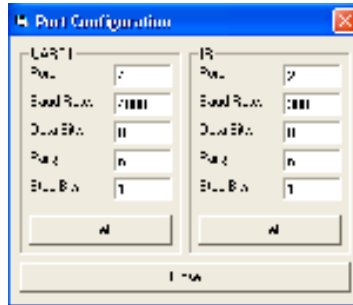


Figure 3-35 Port Configuration Window

Message transmission is used to send different commands to the modules. The command codes are listed in the “Function Code Table” at the bottom of the same window providing an easy access to all functions. These functions are listed and explained in the next section.

If UART1 is connected, the UART1 terminal displays all messages coming from the control board. All functions executed in the control layer with their corresponding parameters and acknowledgment codes are displayed in the same frame as well.

The IR terminal is the main terminal communicating with all modules. Any message sent from modules will be immediately displayed on this terminal with the module ID number. Also all executed functions will be acknowledged from the module by transmitting the executed function code.

3.5.3 Functions

There have been several useful functions implemented within the HexBot including configuration settings, testing, localization and rotation. These functions are listed in Table 3-7 below:

Table 3-7 HexBot Functions

Control Board Functions			
Code	Function	Arguments	Comment
1	JxOpen(i)	i: Joint Number (1 to 6)	Open a solenoid
2	JxClose(i)	i: Joint Number (1 to 6)	Close a solenoid
3	JxOpenAll()	-	Open all solenoids
4	JxCloseAll()	-	Close all solenoids
5	JTest(n,t)	n: number of test t:delay between actuations	Open all solenoids alternatively
6	-	-	Not Used
7	-	-	Not Used
8	-	-	Not Used
9	-	-	Not Used
10	-	-	Not Used
11	SxP(i)	i: Side Number (1 to 6)	Turn on a magnet (Positive pole)
12	SxN(i)	i: Side Number (1 to 6)	Turn on a magnet (Negative pole)
13	SxOff(i)	i: Side Number (1 to 6)	Turn off a magnet
14	SxOffAll()	-	Turn off all magnets
15	SPTest(n,t)	n: number of test t:delay between actuations	Turn on all magnets alternatively (Positive pole)
16	SNTTest(n,t)	n: number of test t:delay between actuations	Turn off all magnets alternatively (negative pole)
17	-	-	Not Used
18	-	-	Not Used
19	-	-	Not Used
20	TurnOffAll()	-	Turn off all actuators
21	MCW(i,t1,t2,t3)	i: rotation joint t1:delay before magnets start t2: delay before joints are exchanged t3: delay before magnets stop	Mobile module Clockwise rotation
22	MCCW(i,t1,t2,t3)	i: rotation joint t1:delay before magnets start t2: delay before joints are exchanged t3: delay before magnets stop	Mobile module Counterclockwise rotation
23	SCW(i,t1,t2,t3)	i: rotation joint t1:delay before magnets start t2: delay before joints are exchanged t3: delay before magnets stop	Substrate Clockwise support
24	SCCW(i,t1,t2,t3)	i: rotation joint t1:delay before magnets start t2: delay before joints are exchanged t3: delay before magnets stop	Substrate Counterclockwise support

Communication Board Functions			
Code	Function	Arguments	Comment
25	IrSel (i)	Side number (0-6), 0:center	UART1 (IR) Channel Selection
26	-	-	Not Used
27	-	-	Not Used
28	SideOnP (i)	Side number (1-6)	Positive Magnet On LED
29	SideOffP (i)	Side number (1-6)	Positive Magnet Off LED
30	SideOnAllP ()	-	All Positive On LEDs
31	SideOffAllP ()	-	All Positive Off LEDs
32	SideOnN (i)	Side number (1-6)	Negative Magnet On LED
33	SideOffN (i)	Side number (1-6)	Negative Magnet Off LED
34	SideOnAllN ()	-	All Negative On LEDs
35	SideOffAllN ()	-	All Negative Off LEDs
36	SidePTest(n,t)	n: number of test t:delay between actuations	Turn on all magnets alternatively (Positive pole)
37	SideNTest(n,t)	n: number of test t:delay between actuations	Turn off all magnets alternatively (negative pole)
38	JointOn (i)	Joint number (1-6)	Joint On LED
39	JointOff (i)	Joint number (1-6)	Joint Off LED
40	JointOnAll ()	-	All Joint On LEDs
41	JointOffAll ()	-	All Joint Off LEDs
42	RESERVED	"*" Beginning of MSG	-
43	JointTest(n,t)	n: number of test t:delay between actuations	Open all solenoids alternatively
44	LOn ()		All LEDs On
45	LOff ()		All LEDs Off
46	LCircle (n,t)	n: number of test t:delay between lights	Rotating LEDs
47	LColor (n,t)	n: number of test t:delay between lights	Rotating LEDs
48	LFlash1 (n,t)	n: number of test t:delay between lights	Flashing LEDs
49	LFlash2 (n,t)	n: number of test t:delay between lights	Flashing LEDs
50	LYref (i)	i: CCW=1, CW=-1	Ref. Indicating LEDs
51	-	-	Not Used
52	-	-	Not Used
53	-	-	Not Used
54	-	-	Not Used
55	-	-	Not Used
56	LMCW(i,t1,t2,t3)	i: rotation joint t1:delay before magnets start t2: delay before joints are exchanged t3: delay before magnets stop	Mobile module Clockwise rotation LEDs
57	LMCCW(i,t1,t2,t3)	i: rotation joint t1:delay before magnets start t2: delay before joints are exchanged t3: delay before magnets stop	Mobile module Counterclockwise rotation LEDs

58	LSCW(i,t1,t2,t3)	i: rotation joint t1:delay before magnets start t2: delay before joints are exchanged t3: delay before magnets stop	Substrate Clockwise support LEDs
59	LSCCW(i,t1,t2,t3)	i: rotation joint t1:delay before magnets start t2: delay before joints are exchanged t3: delay before magnets stop	Substrate Counterclockwise support LEDs
60	-	-	Not Used
61	-	-	Not Used
62	-	-	Not Used
63	-	-	Not Used
64	-	-	Not Used
65	Ref()	-	Set the module as the reference
66	Localize()	-	Start a localization process
67	CW(si,st1,st2,sti,mi,mt1,mt2,mt3)	Similar to CW both for Substrate and Mobile	One message for both modules
68	CCW(si,st1,st2,sti,mi,mt1,mt2,mt3)	Similar to CCW both for Substrate and Mobile	One message for both modules
69	-	-	Not Used
70	-	-	Not Used
71	-	-	Not Used
72	-	-	Not Used
73	-	-	Not Used
74	-	-	Not Used
75	-	-	Not Used
126	RESERVED	"~" UART Receive time exceeded time out indication	-

The message format to call and execute a function is shown in Table 3-8.

Table 3-8 Message Format

*	B1	B2	B3	B4	B5	B6	B7	B8	B9	#
Arguments										
B1	Message Code Number									
B2	si or i or n									
B3	st1 or t1 or t									
B4	st2 or t2									
B5	st3 or t3									
B6	mi									
B7	mt1									
B8	mt2									
B9	mt3									

Each message starts with “*” and ends with “#”. B1 is the code number of the message and B2 to B9 are the arguments of the functions.

3.6 Summary

The implementation of HexBot was discussed in details in this chapter. Mechanical design, electrical system, processing unit and the graphical user interface were explained. Physical specification of the implemented module is as follow:

Each HexBot is a hexagonal module with a side of 78mm and a height of 110mm. The weight of each part and the module is measured and tabulated in Table 3-9.

Table 3-9 Weight of HexBot Components

Item	Qty	Weight (g)	Total (g)
Magnet + Support	6	56	336
Solenoid + Support	6	23	138
Power Pins	6	2	12
Ball Transfer Units	3	42	126
Mechanical Layer	1	4	4
Power Board	1	79	79
Drive Board	1	99	99
Control Board	1	53	53
Communication Board	1	100	100
Total			947

Chapter 4

Reconfiguration Planning

This chapter details the control algorithm used for the transformation of the global shape of the system from an arbitrary initial configuration to a desired goal configuration.

Although each module of the system may have very limited motion, a huge collection of these modules causes the overall system to have numerous degrees of freedom. Consequently, to achieve the unique potential of MSRRS, the distinct and complicated challenge of path planning for a large number of independent modules shall be overcome.

The reconfiguration planning should determine the sequence of individual module movements that transforms the shape of the system from an initial configuration to a desired goal configuration in a preferably optimal manner while enforcing several constraints and considering the kinematic model of the modules.

1. Optimality in MSRRS may be:
 - Minimizing the number of module movements
 - Minimizing the overall reconfiguration time
 - Minimizing the energy consumption during reconfiguration

In this work the effort has been done to minimize the number of module movements and as will be explained in section 4.4.2, approximately, this choice will as well minimize the total energy consumption during reconfiguration.

2. The following primary constraints are considered for the reconfiguration planning which are further explained in section 4.2.1:
 - Avoid collision
 - Maintain connectivity
3. Assumptions of the kinematic model are also specified in section 4.2.3.

In this work the reconfiguration planning problem is considered for a limited number of modules as the algorithm incorporates a centralized path planner; however, the scalability issues and techniques to move towards a decentralized path planning are explained and discussed in section 4.4.1.

4.1 Preliminaries

4.1.1 Environment

In order to plan the paths for modules to move and reconfigure the shape of the system, the environment in which this motion is taking place has to be studied. The main environmental properties are as follow:

- Discrete vs. Continuous
- Static vs. Dynamic
- Deterministic vs. Stochastic
- Fully observable vs. Partially observable
- Episodic vs. Sequential

Let's briefly examine the environmental properties of MSRRS.

Discrete or continuous: Finite number of actions and discrete states leads to a discrete environment.

Static or dynamic: Parallel actuation of several modules at a time means a dynamic environment; in other words, the environment will not remain unchanged till a specific module takes an action. However, it is possible to have serial execution for module

movements and consider the environment static which comes at a cost of more reconfiguration time.

Deterministic or stochastic: The environment can only be considered deterministic, if there are no uncertainties, i.e. full control over the environment, no failures during movement and etc. which is not the case in MSRRS.

Fully or partially observable: Since this work addresses reconfiguration planning for limited number of modules, the environment can be considered fully observable. However in general case of MSRRS the environment is partially observable. Thus section 4.1.5 demonstrates an extension of the utilized tool that overcomes this limitation.

Episodic or sequential: Module movements have long term effects and each movement depends on movements performed earlier; therefore, the environment is sequential.

4.1.2 Reinforcement Learning

There are generally three types of learning:

- **Unsupervised Learning:** Decision making based on observations only with no feedback
- **Reinforcement Learning:** Decision making based on observation and only a scalar evaluative feedback such as a reward function
- **Supervised Learning:** Decision making based on observation and a detailed feedback specifying the exact error

Considering the environmental properties of MSRRS, reinforcement learning is chosen to act as the basic platform for the reconfiguration planning. For a rich and excellent introduction to reinforcement learning, the reader is referred to the textbook (Sutton & Barto, 1998). The main ideas and key features are summarized below.

- Reinforcement learning tries to optimize the performance of the system in uncertain environments by decision making based on direct interaction with environment, without relying on external supervision
- The learner is not told which actions to take, but instead must discover which actions yield the most (long term) reward

- In the most interesting and challenging cases such as MSRRS, actions affect not only the immediate reward but also all subsequent rewards

Looking at MSRRS in the context of reinforcement learning, each module is considered as an *agent* who is planning to reach its goal in an optimal manner. Therefore, the agent and its environment interact over a sequence of discrete time steps as illustrated in Figure 4-1.

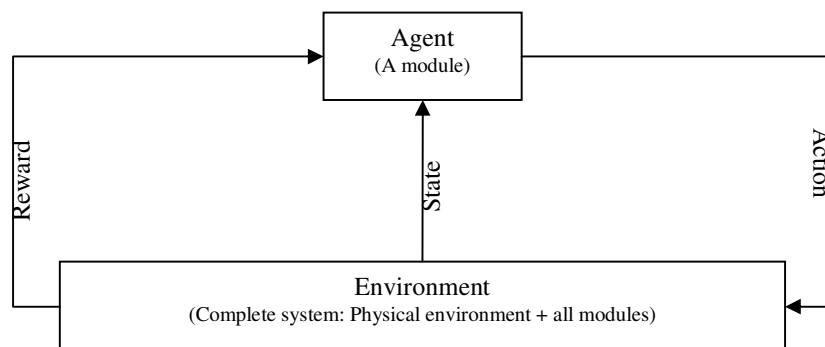


Figure 4-1 Reinforcement Learning Model

At each discrete time step t the agent interacts with the environment based on the followings:

1. Observe the environment *state: s_t*
2. Make a decision and execute an action *action: a_t*
3. Receive the resulting reinforcement *reward: r_{t+1}*
4. Observe the resulting state *state: s_{t+1}*

Consequently, the overall sequence of discrete actions will look as in Figure 4-2.

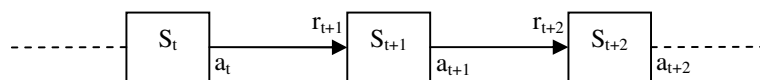


Figure 4-2 Sequence of Actions in Reinforcement Learning

4.1.3 Markov Property

In causal systems the response of the system at time step t can depend on all the actions that has taken place up to time step $t-1$; therefore, the state of the system can only be represented if the complete history of actions is available.

On the other hand, if the state could somehow retain all previous actions, it is called a *Markov* and its environment has a *Markov Property*; therefore in a *Markovian World*, the response of the system at time step t depends only on the state and action at time step $t-1$.

This means that knowing the current state and action in an environment with a Markov property will enable predication of the next state and its expected reward. As a result of iteration, in a Markovian world, all future states and expected rewards can also be predicted by knowing only the current state and actions which are being taken.

The Markov property is a key concept in reinforcement learning since decisions are assumed to be a function of the current state only.

4.1.4 Markov Decision Process (MDP)

MDP is considered as the main framework for studying planning under uncertainty. It can be applied to fully observable environments with Markov properties which are history-independent and stationary; in other words, they have what are so called *Markovian Dynamics* to be modeled as MDPs.

Now, considering a transition model $T(s, a, s')$ which specifies the outcome probabilities for each action at each state (i.e. probability of reaching state s' if action a is taken at state s), an MDP will be formulated as 4-tuple $\langle S, A, T, R \rangle$ with:

- A set of states: S
- A set of legal actions at each state: $A(s)$
- A set of transition probabilities: $T(s, a, s')$
- A set of expected rewards: $R(s)$

Given an MDP, an agent objective is to learn a policy $\pi(s)$ which specifies what action shall be taken at state s that maximizes the expected reward. Note that in general case for MDPs, policies can also be considered non-deterministic, i.e. $\pi(s, a)$ meaning the probability of choosing action a in state s .

4.1.5 Partially Observable MDP (POMDP)

The main difference between MDP and POMDO is whether the current state is fully observable or not. The reader is referred to (Cassandra, Littman, & Kaelbling, 2003) for a comprehensive introduction to POMDPs and their solution techniques.

MDP requires the state to be completely known. However, in some cases, like general MSRRS, each agent can observe only part of the state which is available in its local neighborhood. In such cases, full observability of the state is not a valid assumption and each agent should learn to behave in a partially observable environment.

In POMDPs we will need to introduce a *Belief State* that represents the world state with uncertainty due to partial or imperfect information, in addition to an *observation model* which specifies the probability of each observation at each state:

- Observation model: $O(s', a, o)$
- Belief state: $B(s)$

In this case the fully observable world state is replaced with an observation model and a belief state; however, since the world state is still no longer available, the entire history of the process needs to be stored. Nevertheless, it can be shown that maintaining a probability distribution over all of the states provides the same information as maintaining the complete history.

Therefore, the agent will initially hold an internal belief state b , once an action a is taken and an observation o is made, the agent will use a state estimator (SE) to update the belief state from b to b' as shown in Figure 4-3.

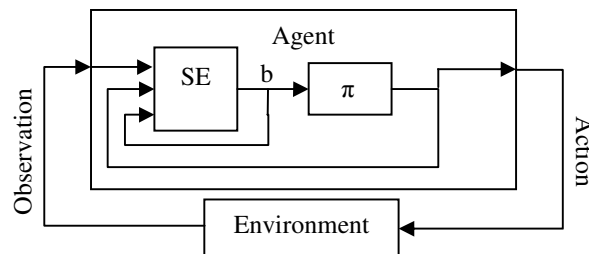


Figure 4-3 Partially Observable MDP Model

4.1.6 Multi-Agent MDP (MMDP)

As can be implied from the name, MMDP refers to environments in which more than one agent is trying to make decisions and such environments are usually modeled as *multi-agent MDPs*.

In MMDPs, single-agent MDP techniques can no longer be applied since the decision of other agents and their consequent actions are directly affecting the environment state and therefore, such an environment cannot be considered static. In effect, the actions taken by other agents influence the decision and action taken by the agent.

In a single-agent MDP, the environment is normally assumed to be fully observable and the goal is to find an optimal policy that maximizes the agent expected rewards.

However, in a multi-agent MDP, the environment is usually partially observable to each agent and the goal is to find an optimal policy that maximizes the expected global rewards for the agent team.

The idea of a single-agent MDP can be simply (but inefficiently) extended to the multi-agent MDPs as follow:

MMDP will be formulated as 5-tuple $\langle N, S, A, T, R \rangle$ where:

- A set of Agents: N
- A set of states: S
- A set of legal joint actions at each state: A
- A set of transition probabilities: T
- A set of expected rewards: R

The main drawback with this formulation is that the number of the states and actions are dramatically increasing as the number of agents in the system increases.

4.2 Constraints and Assumptions

In order to proceed with the problem formulation and explain how the algorithm works a number of constraints and assumptions which are considered shall be addressed.

The enforced constraints are the primarily constraints required for any reconfiguration planning algorithm to generate paths for the mobile modules in our MSRRS. How-

ever, the assumptions are made based on the technique used in this work to attack the problem of the reconfiguration planning.

Note again that in this work the reconfiguration planning problem is addressed for only a limited number of modules and scalability issues and techniques are discussed in section 4.4.1.

4.2.1 Constraints

Constraints which are mainly due to the physical model of the MSRRS need to be considered for the motion of each module in the system and include the following:

1. Maintain Connectivity
2. Avoid Collision

The first constraint prevents global disconnection which is imposed based on the concept of MSSRS and ensures that every module remains connected to at least one other module during its motion.

Besides, there exists one module physically fixed to ground to specify the initial coordinates and orientation of the complete system. The importance of this fix module will become more obvious in section 4.3.

Figure 4-4 illustrates an example for connectivity constraint. In this figure the *blue* module is fixed permanently to ground and the motion of the *green* module can violate the connectivity constraint.

The second constraint ensures that the motion of a module will not cause any collision with other modules. This constraint can also be extended to avoid collision with obstacles in the environment as explained in 4.3.

Figure 4-5 illustrates an example for the collision constraint. As can be seen from this figure, motion of the *green* module to M' will cause collision with M . In fact, rotation of the *green* module around point O , requires mapping of vertices $[a b c d]$ to new locations $[a' b' c' d']$ which violates the collision constraint since during this motion vertex c will collide with vertex d' of M .

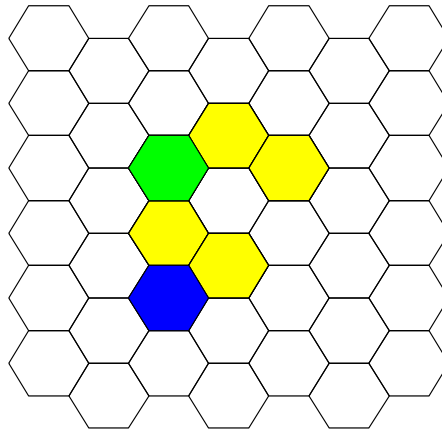


Figure 4-4 Connectivity Constraint

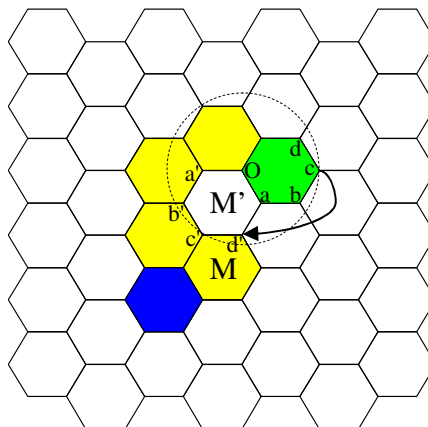


Figure 4-5 Collision Avoidance Constraint

4.2.2 Markov Assumptions

In order to use MDP as the main framework to formulate our problem the following assumptions about the nature of the environment shall be made:

- Stationary Environment

It is assumed that the environment does not change over time

- Fully Observability

It is assumed that the agent has complete information about the state

- Markovian Transition

It is assumed that transitions are Markovian, i.e. the next state of the environment depends only on the current state and the current action being taken by the agent

4.2.3 Kinematic Model Assumptions

Modules kinematic assumptions are quite similar to those of the other researchers who have considered hexagonal modules as the base of their algorithms, such as (Jennifer E. Walter, 2002):

- Each module is a rigid hexagon of the same size as the cells of the plane and always occupies exactly one of the cells.
- One module is connected to the fixed base.
- Each module can move maximum of one lattice space per time step.
- Modules can move into spaces which are not occupied by other modules.
- One module can lift only itself and cannot carry other modules.
- The only motion allowed for a module is the rigid rotation around a vertex it shares with some immobile substrate in the configuration.
- Modules do not fail during rotation even though a pre-specified rotation timing is not guaranteed.
- After initial localization, each module would know at all times: its location, its orientation and also which of its neighboring cells are occupied by other modules.

4.2.4 Initial and Goal Configuration Assumptions

There are basically four main assumptions about the initial and goal configurations:

1. Each module in the configurations is at least connected to one other module.
2. At least one module is connected to the fix base.
3. Common modules in both configurations do not need to move.
4. Configurations are not immobile.

Assumptions 1 and 2 are pretty clear from the concept of MSRRS. Assumption 3 indicates that there is no need for path planning for common modules. Assumption 4 is explained by (Nguyen, Guibas, & Yim, 2001) as following:

A configuration is said to be *immobile* when no module is able to move without violating the motion constraints. Therefore, if the initial or final configuration is immobile, it will be impossible to find a motion plan between the two configurations. Figure 4-6, illustrates an immobile configuration where any module movement will violate the constraints.

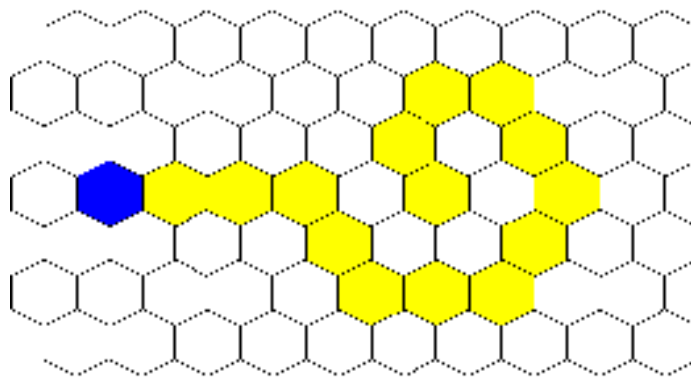


Figure 4-6 Immobile Configuration

4.3 Problem Formulation

This section explains the details of the reconfiguration planner algorithm. Since we are dealing with a homogeneous MSRRS the algorithm therefore rely on module interchangeability.

Before moving into details, let's review the problem once again:

The reconfiguration planning algorithm requires to determine the sequence of individual module movements that transforms the shape of the system from an initial configuration to an arbitrary desired goal configuration. Figure 4-7 illustrates an example of such a problem.

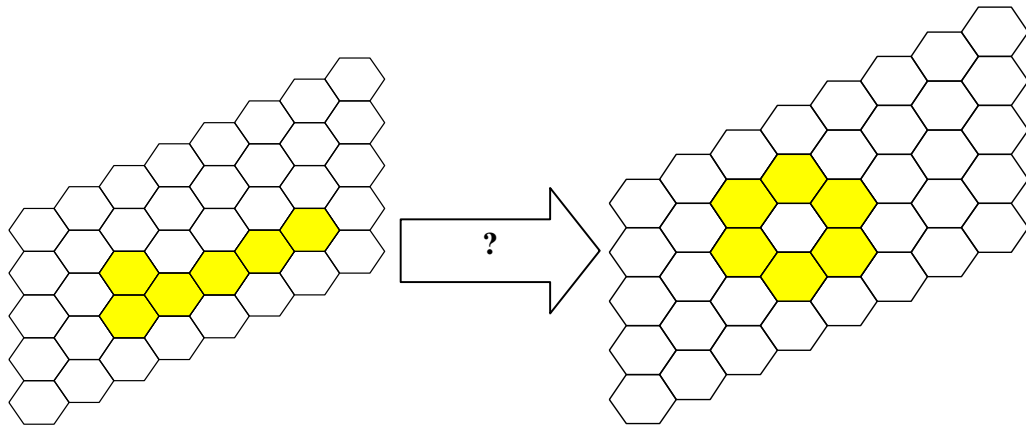


Figure 4-7 Transformation from an Initial Configuration to a Goal Configuration

This problem could be formulated directly using an MMDP approach. However such a formulation would be an extremely inefficient method since the number of states and actions increases tremendously with the number of modules.

Therefore in this work an alternative method is presented which is based on a hierarchical multilayer approach.

4.3.1 Reference Frame and Coordinate system

Let's first establish the coordinate systems and the reference frame for an individual universal module.

As shown in Figure 4-8, the plane is divided into equal hexagons and the coordinate system is chosen to be similar to that of other researchers working with hexagonal modules such as (Chirikjian G. , 1994).

Figure 4-9 illustrates the labeling considered for each individual universal module. After each rotation, modules update their status to keep the same orientation and labeling for consistency and ease of programming.

The reference numbering shown in Figure 4-9 is referred to in all relevant codes used in either in Matlab or Microcontroller attached in appendices.

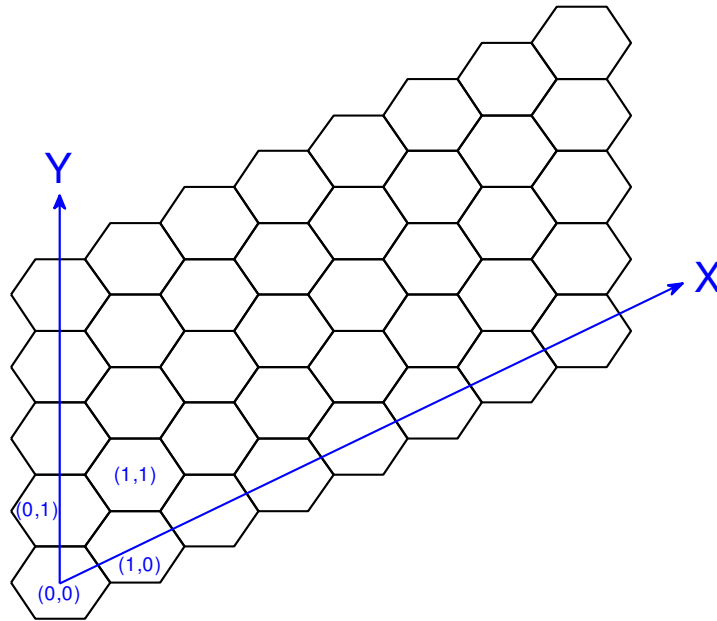


Figure 4-8 Coordinate System

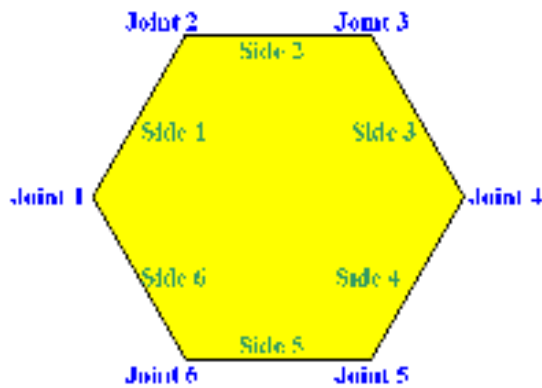


Figure 4-9 Reference Joint and Side Numbering

4.3.2 MSRRS Representation

A complete MSRRS can be represented as a set where each module in the system corresponds to an element of this set. In each configuration, coordinates of modules specify those elements. For example, Figure 4-10 illustrates a system consisting of four modules.

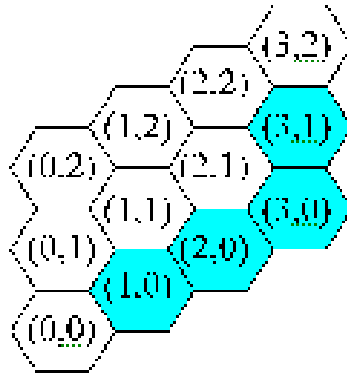


Figure 4-10 System Representation

This configuration can be represented as: $C1 = \{(1,0), (2,0), (3,0), (3,1)\}$. Using this representation, set theory can be utilized to provide useful information about different configurations. For example, $C1 \cap C2$ can specify common modules in two configurations.

The example in Figure 4-10 can also be represented in an array format as:

$$C1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

where in this case modules are represented by one and free locations are the zero elements of the array.

The array format representation is very similar to the set representation and can provide the same information. In this work array representation was proffered since it was easier to manipulate arrays in Matlab.

4.3.3 Hierarchical Multilayer Approach

The ground has now been paved to proceed with the algorithm. The algorithm is designed in five layers as depicted in Figure 4-11. In this approach, upper layers decompose the problem into sub-problems solvable by lower layers. The functionality of each layer is briefly explained as follow:

- *Layer 1* is responsible to obtain the initial configuration of the modules through localization. It is also responsible to obtain the desired goal configuration from higher level controllers which in this work is provided manually through user

- *Layer 2* is responsible to find which gaps shall be first covered and which modules shall be first moved
- *Layer 3* is based on a heuristic method to find the corresponding module for each gap
- *Layer 4* is responsible to plan the path for each individual module
- *Layer 5* is responsible to transfer the required motion into available module actuations

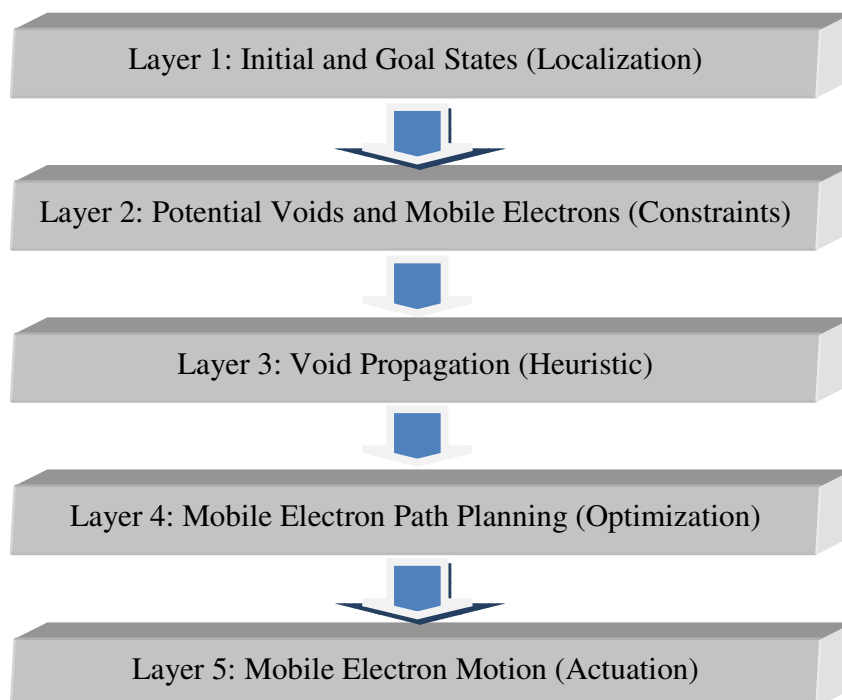


Figure 4-11 Hierarchical Multilayer Approach

Layers 1 to 3 are considered as *Global Flow Planners* since they are dealing with the complete system, while layers 4 and 5 are considered to be *Local Motion Planners* as they are dealing with only an individual module.

Note that the combination of the *heuristic* method used in layer four in addition to the *optimization* method used in layer five yields a *Near-Optimal* solution for the reconfiguration problem.

4.3.4 Layer 1 – Obtain Initial and Goal States

Let's start with the first layer. Figure 4-12 illustrates the importance of the task assigned to this layer.

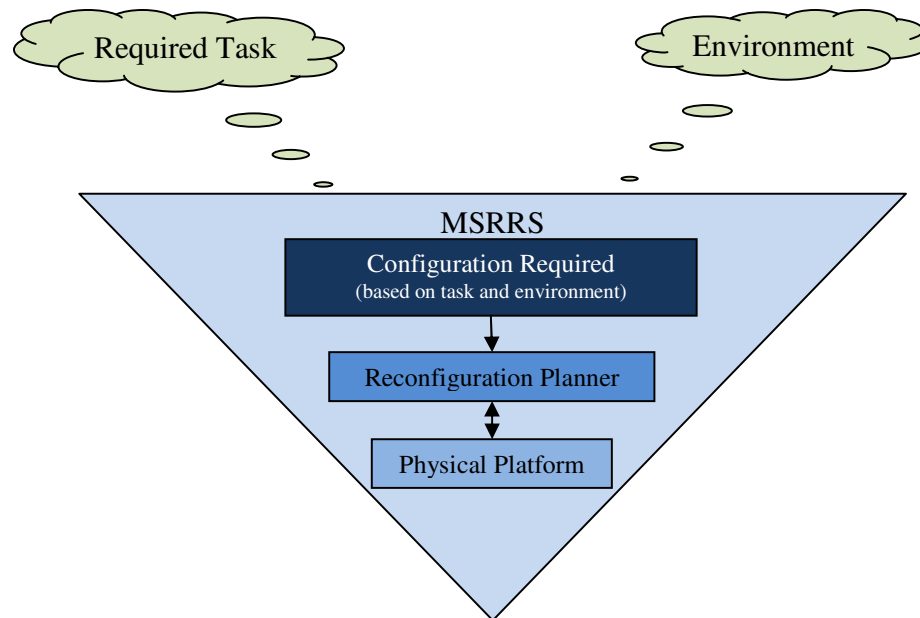


Figure 4-12 MSRRS Task Handling

As can be observed from the general picture in Figure 4-12, this layer interacts with the higher level controller to get the desired goal configuration which is feed manually in our case. At the same time this layer interacts with the physical platform to obtain the current status of the modules in the system. This requires all modules to be localized and ready to send their location to this layer.

Note that for the reconfiguration planner, orientation of the modules does not matter since all modules are symmetrical and following the reference orientation as shown earlier in Figure 4-9. However, orientation of each individual module is important for the module itself in order to know its corresponding sides and joints to actuate during its motion.

As a result a *localization* routine is essentially considered necessary for all modules to know the following at all times:

1. Their location for the reconfiguration planner

2. Their orientation for the actuation required during their motion

Localization

When the system initially starts, no module knows its location and orientation. Therefore a localization routine is executed to provide modules with this information.

The process starts with the module fixed to the base, which is the only module in the configuration knowing its location and orientation as shown in Figure 4-13.

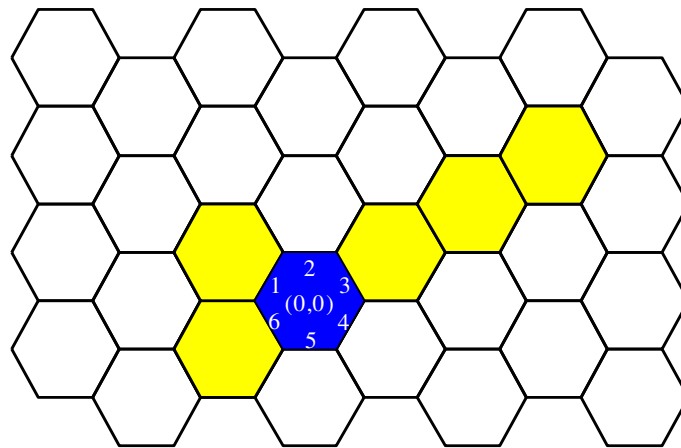


Figure 4-13 Localization Process (0)

This module tries to localize its neighbor modules. Once neighbor modules are localized, they similarly try to localize their neighbor modules and this process continues till all the modules in the configuration are localized. For example, the configuration shown in Figure 4-13 it takes 3 steps for all the modules to be localized as shown in Figure 4-14 and Figure 4-15

Now, let's look at more localization details. The following two main features of the physical platform are required for the localization purpose:

1. Optical Switches (installed on all sides of the modules)
2. IR (infrared) Communication with Multiplexer (communicating to all sides)

For example in Figure 4-13, the localize module at location $(0,0)$ would know the existence of non-localized neighbors at sides 1, 3 and 6 through the use of its optical switches; therefore, it tries to multiplex the serial communication between these sides and

informs them about their locations and orientations by sending them messages in the form of *MSG1*. *MSG1* is an 8 byte message that looks like the following:

MSG1: * ±X X ±Y Y S

The message starts with a “*” and sends the X coordinates, Y coordinates and orientation of the neighbor module (S). Note that in this message format the coordinate system is limited to a 199 by 199 cells and can be easily expanded if required.

This process is done through refereeing to the look-up table shown in Table 4-1.

Table 4-1 Relative Coordinates and Orientations

Module (x,y) Side s	Neighbor: X	Neighbor: Y	Neighbor Side: S
1	X = x - 1	Y = y+1	S = 4 (s+3)
2	X = x	Y = y+1	S = 5 (s+3)
3	X = x+1	Y = y	S = 6 (s+3)
4	X = x+1	Y = y-1	S = 1 (s-3)
5	X = x	Y = y-1	S = 2 (s-3)
6	X = x-1	Y = y	S = 3 (s-3)

For example in Figure 4-13, the localized module at (0,0) will process as follow:

- Switch to side 1 and send: * - 0 1 + 0 1 4
- Switch to side 3 and send: * + 0 1 + 0 0 6
- Switch to side 6 and send: * - 0 1 + 0 0 3

After this stage the localization status will become as depicted in Figure 4-14.

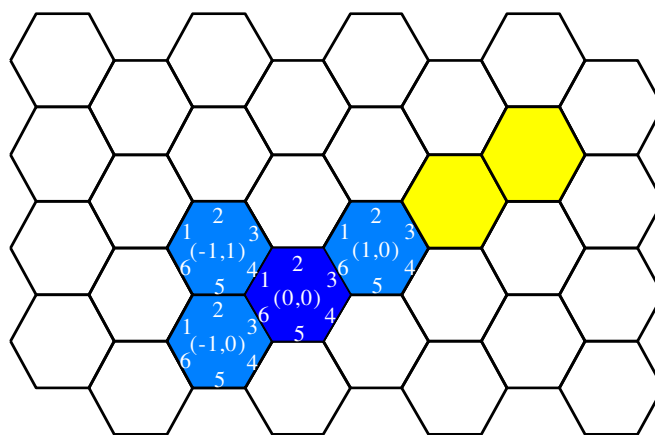


Figure 4-14 Localization Process (1)

Using optical switches at this stage, modules $(-1,1)$ and $(-1,0)$ and $(0,0)$ will find their neighbors fully localized and therefore will not send any messages. However, module $(1,0)$ will find a non-localized module at side 3 and tries to localized it. The next two steps of localization are shown in Figure 4-15.

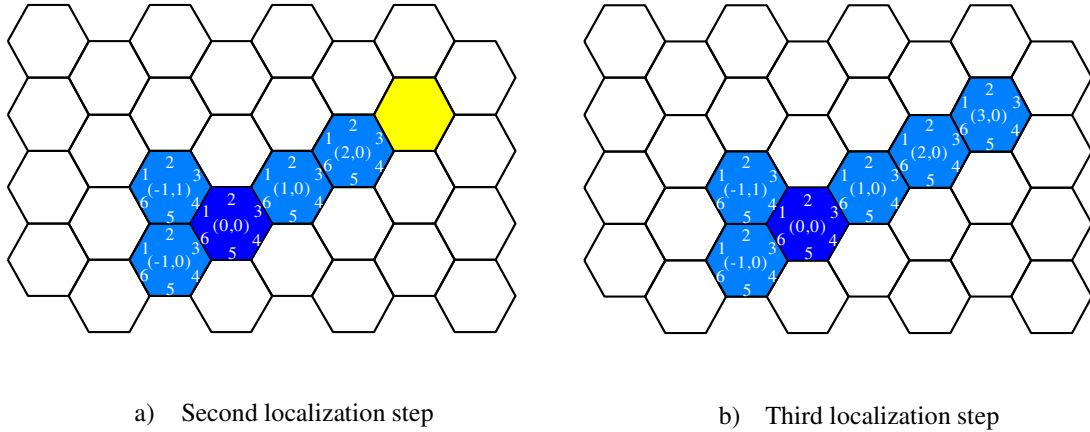


Figure 4-15 Localization Process (2, 3)

Let's now look at *localization timing* in our platform. MSG1 consists of 8 bytes and each byte requires 10 bits in UART (1 start bit, 8 data bits, 1 stop bit). Therefore MSG1 requires 80 bits to be transferred.

Using the internal 1MHz clock of the Atmega162, the circuit allows a maximum transmission rate of 62500 b/s. Therefore the transmission of each MSG1 through each side would take:

$$t_{msg1_side} = \frac{80}{62500} = 0.00128s = 1.28ms \quad (4.1)$$

Let's allow a gap of 0.38ms after each MSG1 for safely switching the multiplexer to the next channel. Therefore the transmission of MSG1 to all six sides of the module would take:

$$t_{msg1_module} = 6(1.28 + 0.38) = 10ms \quad (4.2)$$

Note that the delay of 0.38ms is mainly due to the limitation imposed by the multiplexer (CD4051) requiring 240ns for switching while the microcontroller requires only 2 clock cycles = $2 \cdot 8 = 16ns$

As can be seen from (4.2) it will take each localized module maximum of 10ms to send MSG1 to all its neighbors.

Non-localized modules should listen to each of their existing neighbors for at least 10ms to ensure the reception of the message. Similarly, a gap of 0.38ms is allocated for the multiplexer to switch to the next channel after each reception is over. Therefore, for non-localized module with a localized neighbor, it will take a maximum of:

$$t_{msg1_receive} = 6(10 + 0.38) = 62.3ms \quad (4.3)$$

As can be seen from (4.3) it will take each non-localized module maximum of 62.3ms to receive MSG1 from any of its neighbors.

For example the maximum localization timing for Figure 4-13 can be calculated as follow:

- Time step 1: 62.3ms for modules $(-1,1)$, $(1,0)$ and $(-1,0)$
- Time step 2: 62.3ms for module $(2,0)$
- Time step 3: 62.3ms for module $(3,0)$

As can be seen the configuration requires 3 time steps which is nearly 187ms to get localized.

4.3.5 Layer 2 – Potential Voids and Mobile Electrons

Now that the desired goal configuration is obtained and the location of all modules and therefore the current configuration is known, it is possible to proceed to the next layer.

Electrons are modules which exist in the current configuration but do not belong to the goal configuration. In contrast, *voids* are modules which exist in the goal configuration but are not available in the current configuration. Figure 4-16 illustrates these definitions.

Electrons and voids can be easily found by subtracting the two (current and goal) configurations as done in code “*FindVE*” attached in appendices. Now the task is to fill the voids with electrons.

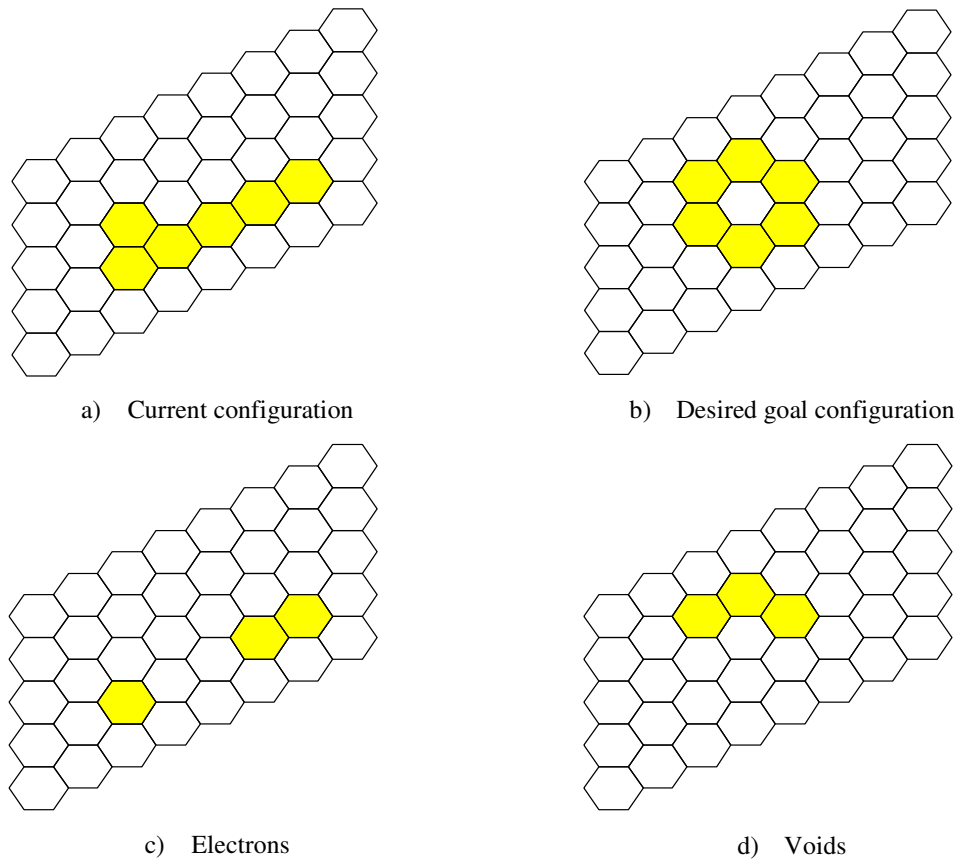


Figure 4-16 Electrons and Voids

However, at this point not all electrons can move since they may violate connectivity constraint. At the same time, not all voids can be filled since they may not be connected to the current configuration. Therefore, we have to find out which electrons should be first moved and which voids should be first filled.

- *Mobile electrons* are electrons which can move without violating the connectivity constraint (disconnecting the structure) or collision avoidance constraint.
- *Potential voids* are voids that can be filled immediately with mobile electrons of the current configuration.

Figure 4-17 illustrates mobile electrons (left) and potential voids (right) in our example.

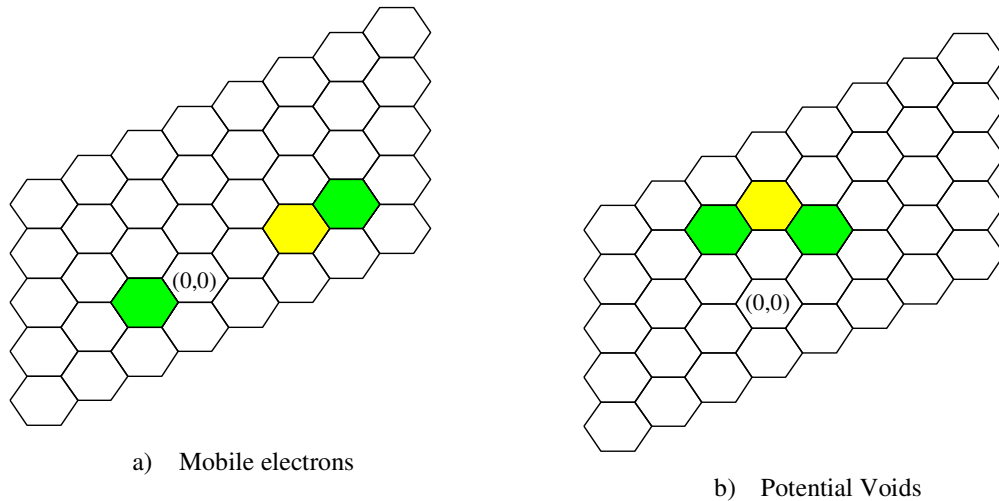


Figure 4-17 Mobile Electrons and Potential Voids

In Figure 4-17 the left figure displays mobile electrons in green at locations $(-1,0)$ and $(3,0)$ and an immobile electron in yellow at location $(2,0)$. It's clear that the yellow electron in this configuration cannot move since by moving it the module at location $(3,0)$ will be disconnected from the configuration which violates the connectivity constraint.

Likewise, in Figure 4-17 the right figure displays potential voids (in green) at locations $(-1,2)$ and $(1,2)$. It's also clear that the yellow void at location $(0,2)$ in this configuration cannot be filled directly before a potential void is filled.

The following point shall be noted at this stage:

- The fix module to the base ensures overlapping of the two configurations (current with the goal). Therefore, it is not possible to have all the modules as electrons.
- If the two configurations do not overlap and the algorithm is used for locomotion planning instead of reconfiguration planning (where there is no possibility of having a fixed module), the locomotion between two configurations can be divided into intermediate stages where configurations actually overlap with each other.
- It is not always the case that the number of potential voids and mobile electrons are the same. For example if we switch current and goal configurations in our example, there will be three mobile electrons and two potential voids as shown in Figure 4-18.

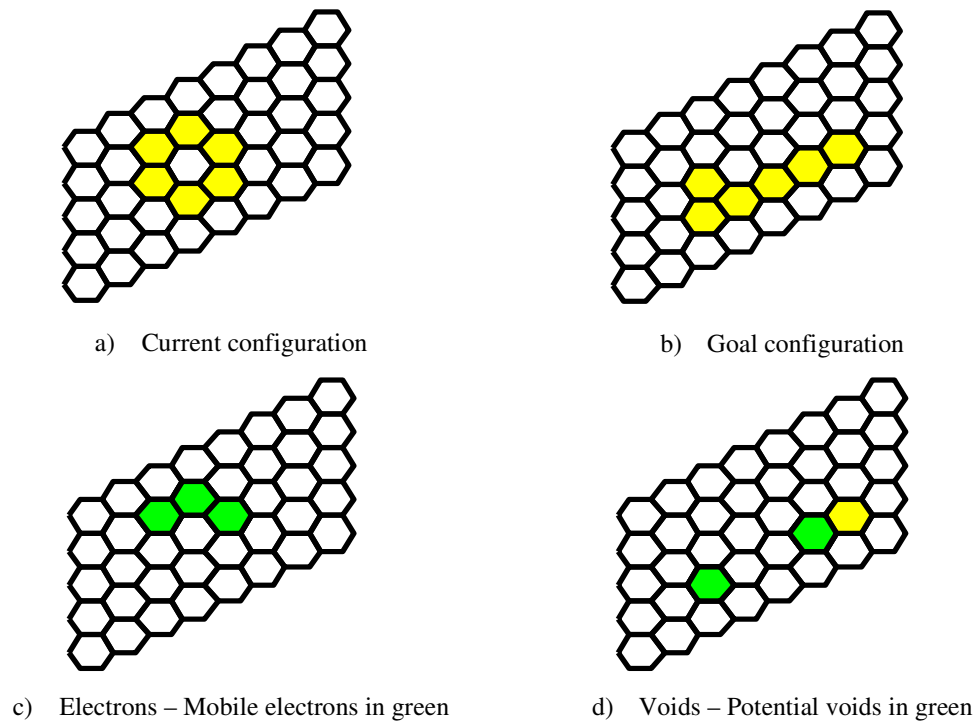


Figure 4-18 Example of ME and PV

Mobile Electrons

For an electron to be mobile two conditions shall be satisfied:

- 1) There shall be a free location in its neighborhood that the electron can move without collision
- 2) The motion of the electron shall not cause disconnection in the whole system

As explained in 4.2.1, free locations with more than three neighboring modules cannot be filled without violating the collision avoidance constraint. Therefore, the first criterion can be easily checked accordingly.

The second criterion is thoroughly examined below. In fact, this part will provide us with a tool to check the connectivity constraint.

Mobility of an electron can be checked through graph operations while moving one module at a time, i.e. the configuration is fixed elsewhere. The connectivity graph for MSRRS can be constructed by dedicating a node for each module and an edge for each of its neighbors. Once the graph is constructed, all electrons can be considered mobile unless they are *articulation points* in the connectivity graph.

Articulation points (or cut vertices), are nodes that their removal will cause their neighbors to be no longer connected. In other words, the removal of articulation point in a graph will increase the number of connected components by breaking the graph into separate parts. In MSRRS, articulating points correspond to immobile electrons. A simple illustration is provided in Figure 4-19, Figure 4-21 and Figure 4-20.



Figure 4-19 Graph Representation

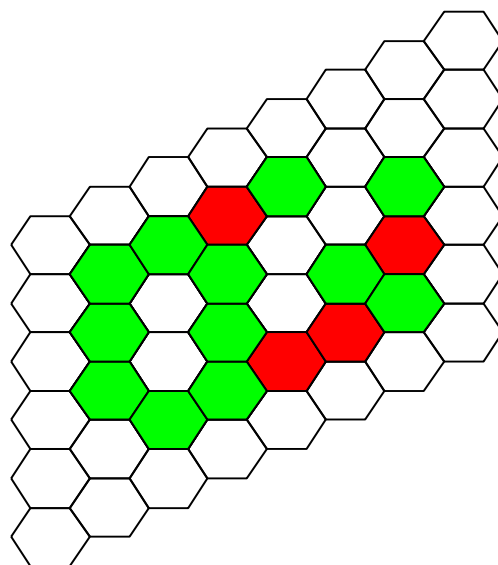


Figure 4-21 Articulating Nodes

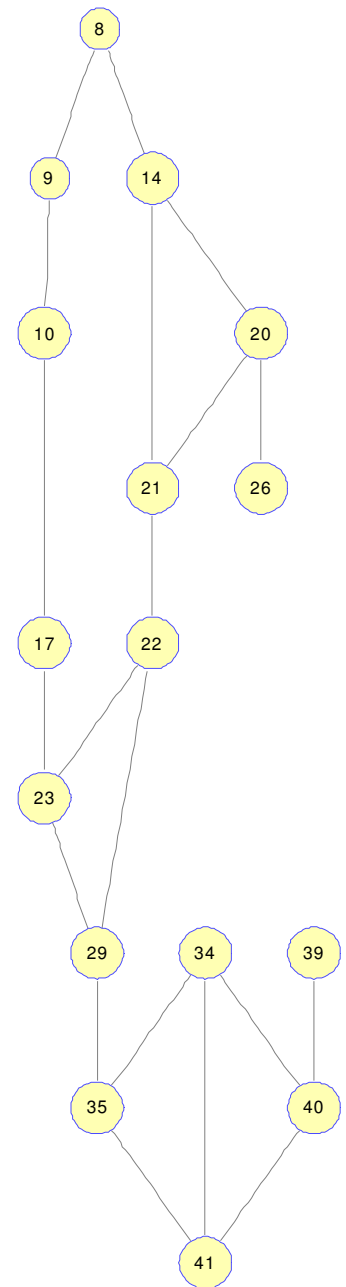


Figure 4-20 Connectivity Graph

In Figure 4-19, the nodes' IDs are assigned in a linear indexing format starting from 1 the upper left module and ending with 48 on the lower right module. As can be seen in Figure 4-20, each module is assigned to a corresponding node in the connectivity graph and each neighbor is considered as an edge.

As can be seen in Figure 4-21, in this configuration, all modules are mobile except those red modules at articulating nodes 20, 29, 35 and 40 which are immobile. Note that the codes *HexCG* and *HexCGT* attached in the appendices are used to construct and draw the connectivity graph in addition to finding the articulating nodes. For a good introduction to graphs theory, the reader is referred to (Kocay & Kreher, 2005) and (Buckley & Lewinter, 2003). However, for the sake of completeness, a simple method of finding articulating nodes is briefly presented below.

The algorithm to find articulating nodes is divided into two stages:

- a) Given a graph, find the number of connected components
- b) Given a graph, find articulated nodes.

Part a - Number of connected components in the graph

The number of connected components in a graph is equal to the number of zero eigenvalues of the Laplacian matrix of the graph. Once again for additional information the reader is referred to (Kocay & Kreher, 2005).

Laplacian (or Kirochhoff or Admittance) matrix L for a graph G with n nodes is a square $n*n$ matrix representation of G that with indices $l_{i,j}$ as follow:

$$l_{i,j} = \begin{cases} \text{deg}(n_i) & ; i = j \\ -1 & ; i \neq j \text{ and } n_i \text{ is adjacent to } n_j \\ 0 & ; \text{Otherwise} \end{cases} \quad (4.4)$$

Figure 4-22 presents a simple demonstration for Laplacian matrix. As can be seen in this figure the eigenvalue of L has only one zero, meaning that there is 1 connected component in this system.

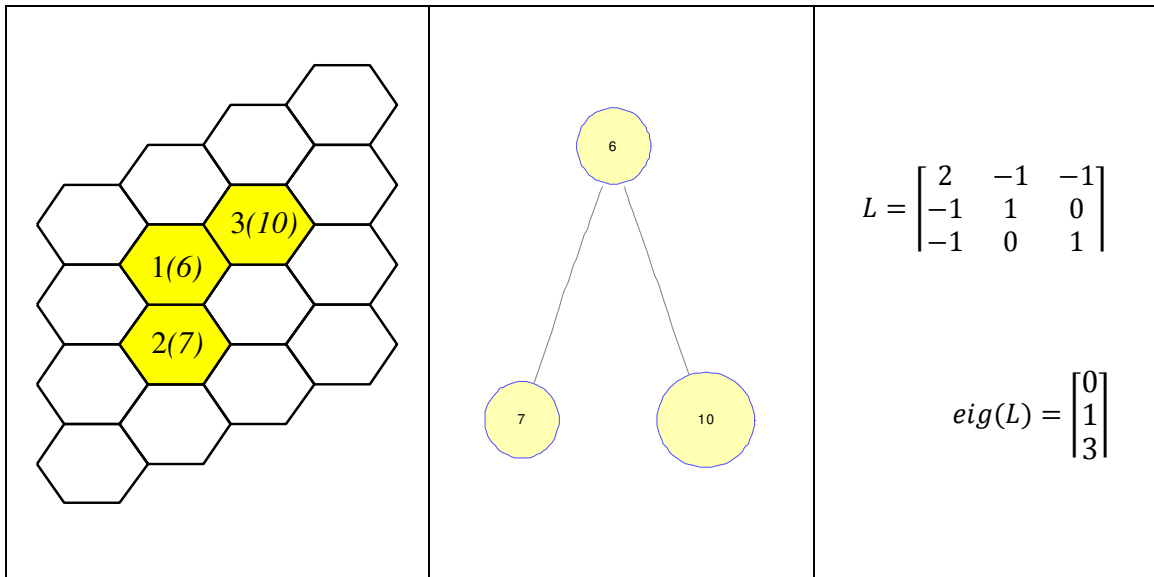


Figure 4-22 Laplacian Matrix Representation

Part b – Find articulated nodes of a graph

This is done simply using *Part a*. First a node is chosen, and then there is a check to see if the node removal increases the number of connected components, i.e. the node is articulated.

Just to illustrate a combinational example for *Part a* and *Part b*, Figure 4-23 displays the same graph discussed in Figure 4-19 but an immobile electron at location 29 is removed. It can be seen that the number of connected components is now increased from 1 to 2. A code *Mobile* used to mark electrons as mobile is also attached in appendices.

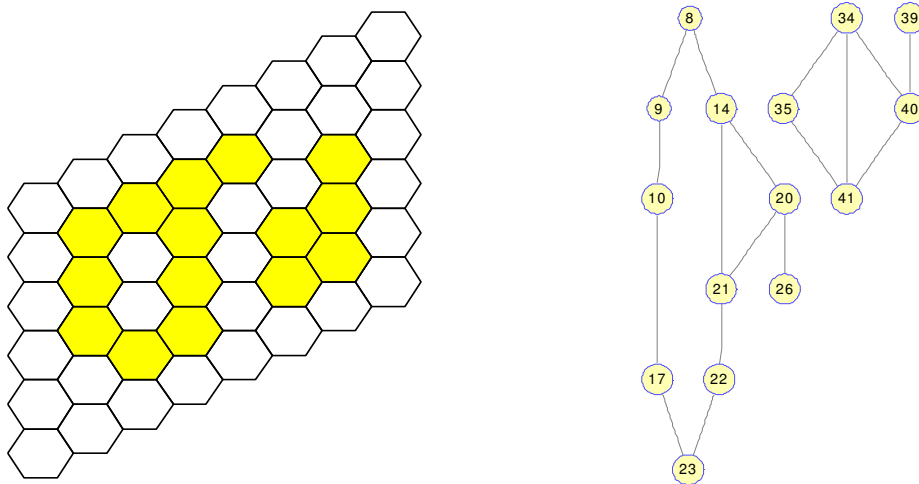


Figure 4-23 Example of a Disconnected Configuration

Potential Voids

Potential voids are much easier to find. In fact, in order to investigate wheatear a void is a potential void or not, all its six sides are checked. If there exists at least one immobile module in the neighborhood of the void module, then the void is marked as a potential void and can therefore be filled immediately. The code *PV* that finds potential voids is also attached in appendices and Figure 4-18 illustrates an example of potential voids.

4.3.6 Layer 3 – Void Propagation

Since the MSRRS is considered to be homogeneous, it is not really clear which mobile electron should fill which potential void. Therefore, void propagation provides a simple heuristic technique to find a corresponding mobile electron for each void.

Void propagation technique works as follow:

- 1) Mark voids which can be filled as potential voids (PV) in the goal configuration
- 2) Mark electrons that can move as mobile electrons (ME) in the initial configuration
- 3) Each potential void (i) in the configuration transmits $MSG2(i)$ to all its neighbors
- 4) At each time step of reception of $MSG2(i)$, each location transmits $MSG2(i)$ to all its neighbors, if the location is empty, has an immobile substrate (IS), has not received $MSG2(i)$ earlier, and it is not a potential void
- 5) $MSG2(i)$ will be erased from all locations when reaches a mobile electron

Note that $MSG2$ includes the potential void ID (assigned dynamically to modules and is the x and y coordinates of the module) in addition to the time step (id,t). The technique is applied to the previous example in Figure 4-16, and the messages are displayed in Figure 4-24.

As can be seen in Figure 4-24, the potential void at location 12, i.e. $PV(12)$ forms a message $(12,0)$ and in the first time step increments the time of the message and transmits the message $(12,1)$ to all its immediate neighbors; however only two of them:

- Are empty locations
- Are not potential voids
- Have immobile substrates

- Have not received a message from PV(12) before

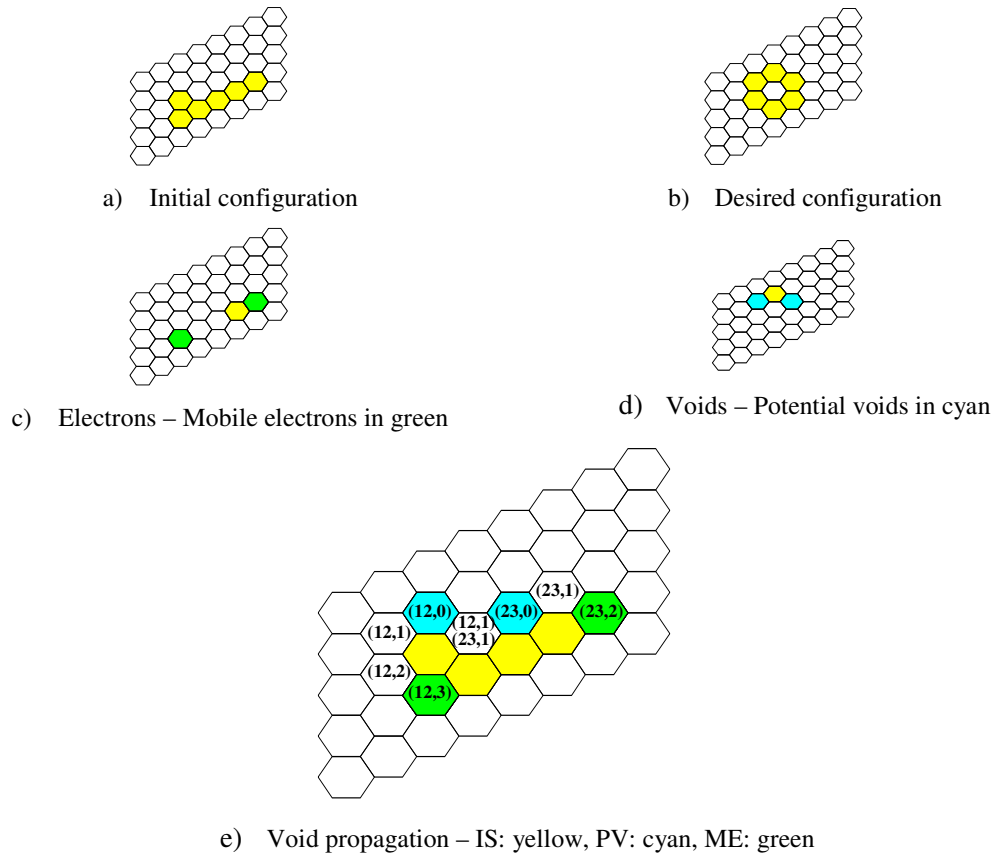


Figure 4-24 Void Propagation

Therefore those two increment the time step in the message and transmit the message $(12,2)$ to all their neighbors. This process continues till the message $(12,3)$ reached the mobile electron at location 14, i.e. ME(14). At this time all messages with $i^{\wedge}=12$ will be erased from all locations since PV(12) has found its ME(14). Similarly, PV(23) finds its corresponding ME(34).

HexVP is a function written for void propagation and is available in appendices. Now the problem is reduced a motion planning for a ME to a PV and is forwarded to the next layer.

4.3.7 Layer 4 – Mobile Electron Path Planning

This layer is based on an MDP that can be used for trajectory planning of a single module (ME) to find a traversable path form an initial location to a goal location (PV).

The complete process mainly consists of two parts:

- Formulating the problem as an MDP
- Implementing a policy search

MDP Formulation

As explained earlier in 4.1.4, MDP is considered as the main framework for studying planning for *sequential decision processes* under uncertainty and can be applied to fully observable environments with Markov properties.

MDP is formulated as 4-tuple $\langle S, A, T, R \rangle$ and therefore the followings are required:

- a) A set of states: $s \in S$
- b) A set of legal actions at each state: $a \in A(s)$
- c) A set of transition probabilities: $T(s, a, s')$
- d) A set of expected rewards: $r \in R(s)$

Let's proceed with finding S, A, T and R for the MSRRS presented in this work. Note that in this layer, it is assumed that the environment is fully observable. For scalability issues refer to 4.4.1.

a – Set of states

Each mobile electron or an empty location with at least one and not more than three immobile substrates is considered as a state (s) of the current configuration. Mobile electrons are therefore, allowed to be located only in the states. At this stage, fixed physical structures in the environment such as walls can also be eliminated from the set of states to avoid collisions.

In order to find available state at each configuration (C), the neighbors of each module are checked. If the neighbors are not currently occupied and do not have more than three substrates, they will be marked as states. Figure 4-25 illustrates the states (in magenta) of a given configuration (C) with mobile electrons (in green). A simple code (MDP_State) to find the state of an arbitrary configuration is also attached in the appendices.

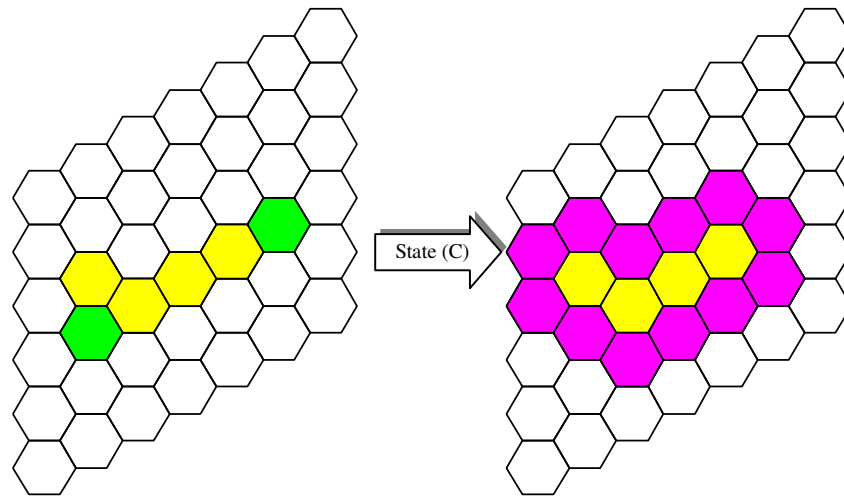


Figure 4-25 States of MDP

b – Set of legal actions at each state

For each arbitrary $s \in S$, there are mainly three possible actions in our platform:

- Stay in its location
- Rotate clockwise
- Rotate counter clockwise

The first action is pretty clear and the resulting location will be the same as the initial location. However the resulting locations of the rotations depend on the center of rotations which is the pivot point or the joint of the module.

In order to find the resulting locations of the rotations, first the neighboring modules are considered and those of which that are unoccupied (avoid collision) and have immobile substrate (maintain connectivity) will be chosen as the resulting locations.

Figure 4-26 illustrates all available actions (in black) of the mobile module located in states (s). The centers of rotations are marked with red dots and the blue dot is an example of a wrong center of rotation that violates the constraints.

As can be noticed the three actions of a mobile module located in state (s) are simply the rotations in addition to staying in its location. A simple code *MDP_Action* to find possible actions of a state is provided in appendices.

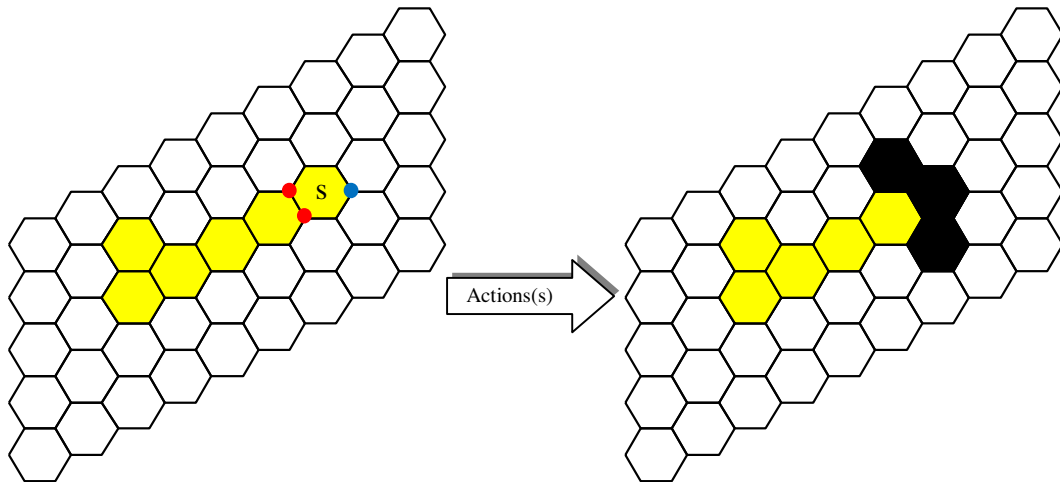


Figure 4-26 Available Actions of an State

d – Set of transition probabilities

Transition model $T(s, a, s')$ specifies the outcome probabilities for each action at each state (i.e. probability of reaching state s' if action a is taken at state s). This work assumes no module failure and therefore considers all probabilities 1.

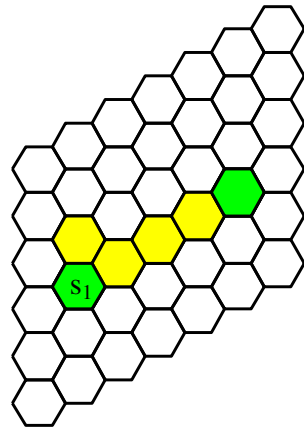
d – Set of expected rewards

Each state in the configuration should be correlated to a reward value. In this work the following rewards criteria is considered:

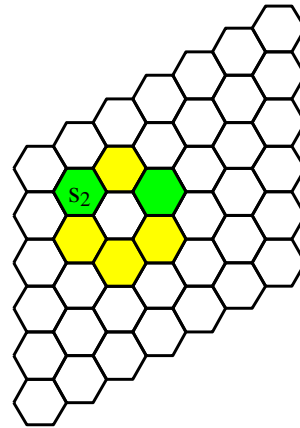
- a) Reward of 1 for the potential void (goal location)
- b) Reward of -1 for other mobile electrons (avoid collision)
- c) Reward of -0.04 for all other states (penalty for not moving)

Note that the second criterion can also be used to avoid collisions with dynamically moving obstacles in the environment

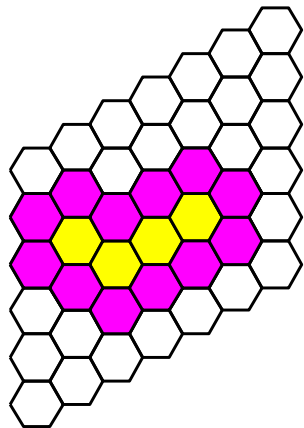
A simple code *MDP_Reward* is attached in appendices that can form the reward function for each state. Figure 4-27 illustrates the reward function for moving a mobile electron from (s_1) to the potential void (s_2). As can be seen the reward function has a value associated to each state.



a) Current configuration – ME in green



b) Goal configuration – PV in green



c) Immobile substrate – States in magenta

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.04 & 1 & 0 & 0 & 0 & 0 & 0 \\ -0.04 & 0 & -0.04 & -0.04 & -0.04 & 0 & 0 \\ 0 & -0.04 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & -0.04 & -0.04 & -0.04 & -0.04 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

d) Formulated reward function

Figure 4-27 States Reward Function

Policy Search

Policy is a solution of an MDP problem, i.e. given an MDP, each module objective is to learn a policy $\pi(s)$ which specifies what action shall be taken at each state s . In other words, a policy associates a decision with every location that the module may reach.

An optimal policy $\pi^*(s)$ is a policy that its execution will maximize the expected sum of future rewards during the sequence.

Reader is referred to (Russell & Norving, 2003) for an elaborative explanation and different techniques used to find optimal policies. Below, only the concepts needed toward the requirement of this work are summarized.

Utility of a sequence

Let's say a mobile electron chooses a path to reach its corresponding potential void that passes through the following states sequence: $\{s_0, s_1, s_2, \dots, s_n\}$. There could be another path that will also take the mobile electron from its location to its goal location and passes through other states $\{s'_0, s'_1, s'_2, \dots, s'_n\}$. Which path shall the mobile electron take?

In order to answer such a question a performance measurement tool for a sequence of actions needs to be defined. Let's define the utility of a sequence of states as follow:

$$U_{seq} = (\{s_0, s_1, s_2, \dots, s_n\}) = R(s_0) + R(s_1) + R(s_2) + \dots + R(s_n) \quad (4.5)$$

As can be seen in equation (4.5), the utility of the sequence is the sum of all its states' rewards. Also it can be noted that in such a definition the performance of the module is considered to be stationary and has no time dimension.

The fact that the utility of the sequence is simply the *additive rewards* of each state of the sequence can be problematic in many cases. Especially the problem arises when the number of states increases and U_{seq} tends to grow rapidly and may even tend to infinity if the policy is improper i.e. does not guarantee reaching the goal state. Therefore a better representation for those cases is to use *discounted rewards* as follow:

$$U_{seq} = (\{s_0, s_1, s_2, \dots\}) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots \quad (4.6)$$

where γ is the discount factor and it is a real number in the range of $[0, 1]$. In this case, the utility of the sequence will be bounded as shown below:

$$U_{seq} = (\{s_0, s_1, s_2, \dots\}) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \frac{R_{max}}{1 - \gamma} \quad (4.7)$$

where R_{max} is the maximum reward value of the states. Note that the discount factor γ comes with some other interesting advantages, similar to the interest rate $\left(\frac{1}{\gamma} - 1\right)$ as follow:

- a) It specifies the diminishing value of the future rewards; in other words, it weighs more the current state and in weights less the future states
- b) Similarly it can sometimes be used to determine how far the module is willing to see and value the future rewards
- c) It is also similar to the diminishing preferences of human and animal overtime

Optimal policy

Given the utility of the sequence definition and its equation (4.7), the optimal policy should satisfy the following:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} E \left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi \right] \quad (4.8)$$

This can be read as follow: Optimal policy is a policy that its execution will maximize the expected sum of discounted rewards of its sequence.

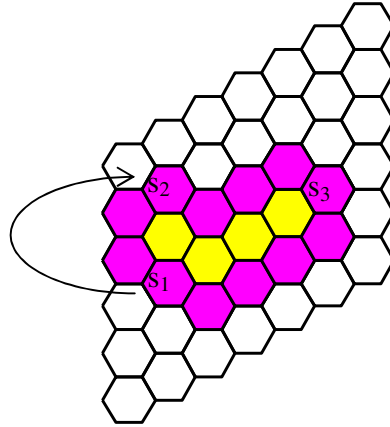
Value function

Value function (also called utility function or cost-to-go function) is associated with each policy and its value at each state is the expected utility of the state sequence encountered when a policy is executed starting from that state. Therefore the value function definition is based on the utility of state sequences as shown below:

$$U^\pi(s) = E \left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid s_0 = s \right] \quad (4.9)$$

Note that in this definition $U(s)$ is actually the *long-term* total reward from s onwards. This is in opposition to $R(s)$ which is the *short-term* reward for being in state s .

Just to illustrate the concept, let's refer back to our example in Figure 4-28 where the mobile electron in state s_1 is planning its path towards its corresponding potential void in state s_2 and it should not collide with the other mobile electron in state s_3 acting in the same environment. The normalized value function is calculated for 0.7 discount rate using the value iteration technique which will be explained shortly and the result is displayed in Figure 4-28.



$$R(s) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.04 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.04 & 0 & -0.04 & -0.04 & -0.04 & 0 & 0 & 0 \\ 0 & -0.04 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -0.04 & -0.04 & -0.04 & -0.04 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad U(s) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.68 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.47 & 0 & 0.68 & 0.47 & 0.32 & 0 & 0 & 0 \\ 0 & 0.32 & 0 & 0 & 0 & -0.08 & 0 & 0 \\ 0 & 0 & 0.21 & 0.13 & 0.08 & 0.05 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4-28 Value Function of States

Note that the more the states are close to s_2 and far from s_3 , the cost-to-go value is less and therefore the value function is more. Having the value function, the agent can select an action that maximizes its state value function. In other words:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, a, s') U(s') \quad (4.10)$$

The definition of the value function presented in equation (4.9), and the optimal policy presented in equation (4.10), will imply the following: if a module is following an optimal policy, then the value function at its state is the immediate reward for that state in addition to the maximum expected (discounted) value of the next state. This is known as the *Bellman equation* as follow:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s') \quad (4.11)$$

Value iteration

Several techniques can be used to compute an optimal policy given an MDP problem. Value iteration as one of dynamic programming (DP) techniques is used in this work

to find the optimal policy for each module. This algorithm simply works based on solving the Bellman equation in an iterative manner. Note that the Bellman equation is a non-linear equation. For n states, there will be n equations and n unknown to be solved for.

In other words, the value iteration algorithm iteratively solves equations relating the utilities of each state to that of its neighbors simultaneously, as follow:

- a) Star with an arbitrary initial values for the states
- b) Calculate the right hand side of the equation
- c) Update the left hand side
- d) Repeat b and c till the error is minimized to a threshold

The code for the algorithm (*MDP_VI*) is also attached in the appendices.

Convergence of value iteration

One main question remains to be answered is weather this algorithm will converge to the unique solutions of the Bellman equation or not. The proof of convergence is based on the concept of *contraction*. The detailed proof is provided in (Russell & Norving, 2003) and the basic idea is as follow.

A contraction function is basically a function that when applied to two inputs, it makes their difference less.

Let's go back to the value iteration algorithm. The i^{th} iteration would be:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_i(s') \quad (4.12)$$

Consider the Bellman update as an operator B ($U_{i+1} \leftarrow BU_i$) and max norm, as a function to measure the distances between utility vectors i.e. the maximum distance between corresponding elements ($\|U\| = \max |U(s)|$), therefore,

$$\|BU_i - BU'_i\| \leq \gamma \|U_i - U'_i\| \quad (4.13)$$

As can be seen from (4.13), the Bellman update is a contraction function by the factor of γ and therefore the value iteration always converges to the unique solution of the Bellman equations. It can be seen that the error also converges exponentially to zero.

It can be shown that N iteration is required to reach an error of ϵ :

$$N = \log \frac{2R_{max}}{\varepsilon(1-\gamma)} / \log \frac{1}{\gamma} \quad (4.14)$$

The equation (4.14) is plotted in Figure 4-29 for different values of $c = \varepsilon / R_{max}$. The Y-axis is the number of iteration required to achieve c and it is plotted in log scale to magnify the differences between the graphs. The X-axis has different values of the discount factor plotted in linear scale.

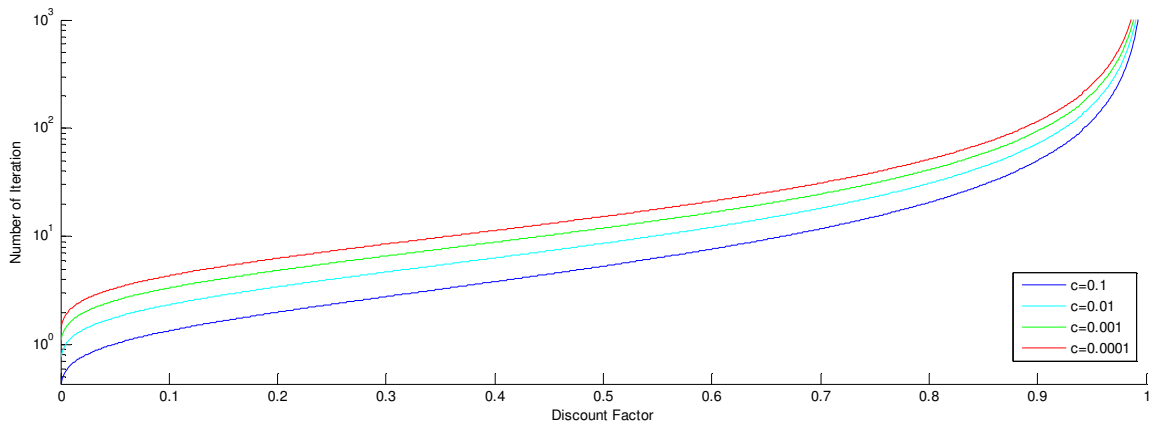


Figure 4-29 Convergence of VI

Important observations from the graph in Figure 4-29:

- Number of iterations does not depend heavily on the value of c because of the exponential nature of convergence
- For small values of discount factor very few iterations are required; however the module will have very limited horizon and will miss the long-term rewards
- For large values of discount factor N grows rapidly and so many iterations are required

Using this algorithm, Table 4-2 shows the value function for our earlier example in Figure 4-27 for different discount factors.

As can be seen for small discount factor the algorithm works very quickly and the number of iterations is low. However, the modules in states which are far from their goal are reluctant to move into a better state since they have short-horizon and do not value the long-term rewards.

Table 4-2 Discount Factor and Number of Iterations

$\gamma=0.1, \epsilon=0.001$	
$U(s) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.06 & 1 & 0 & 0 & 0 & 0 & 0 \\ -0.03 & 0 & 0.06 & -0.03 & -0.04 & 0 & 0 \\ 0 & -0.04 & 0 & 0 & 0 & -0.9 & 0 \\ 0 & 0 & -0.04 & -0.04 & -0.04 & -0.04 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	
$\gamma=0.5, \epsilon=0.001$	
$U(s) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.48 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0.22 & 0 & 0.48 & 0.22 & 0.09 & 0 & 0 \\ 0 & 0.09 & 0 & 0 & 0 & -0.46 & 0 \\ 0 & 0 & 0.02 & -0.01 & -0.02 & -0.03 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	
$\gamma=0.9, \epsilon=0.001$	
$U(s) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.90 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0.80 & 0 & 0.90 & 0.80 & 0.72 & 0 & 0 \\ 0 & 0.72 & 0 & 0 & 0 & 0.55 & 0 \\ 0 & 0 & 0.64 & 0.57 & 0.51 & 0.49 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	

On the other hand for large discount factor the algorithm requires more number of iterations and modules will value long-term rewards. For example, at 0.9 discount factor, the module with state value of 0.49 is even willing to collide with the other mobile module at state s_3 , just because it can see that in that path it will reach its goal state faster. It can be noted that there is a trade-off in choosing the discount factor. The code (*MDP_NRL*) attached in appendices uses 0.7 discount factor that is found suitable for this work and based on that this code recommends the next move for each module.

4.3.8 Layer 5 – Mobile Electron Motion

This layer is based on a predefined feed-forward controller and provides the motion for the mobile electron from one location into another neighborhood location by transforming the movement into the 6-axial rotations of the platform.

In addition to that, before actuating any of the magnets, this layer will also ensure that there is no collision during the rotations.

Considering the immobile configurations and collision avoidance constraints discussed earlier in this section, this layer will

- a) Ensure that there are no collisions during the movement (empty states required)
- b) Ensure there is an immobile substrate to perform each rotation
- c) Transform the required motion to 6-axial rotations of the platform

Figure 4-30 illustrates a mobile module and a possible motion to one of its immediate neighbor location (*A, B, C, D, E, F*).

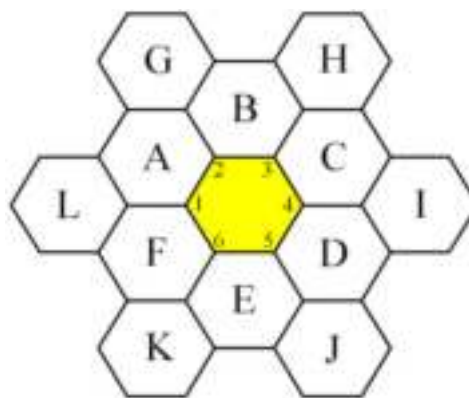


Figure 4-30 Motion of a Mobile Electron

The required look-up table is formed and tabulated in Table 4-3. For example, it can be seen in this table that there are two moves that end in location A. One by rotating around joint 2 clockwise and the other by rotating around joint 1 counterclockwise. However, only one will be possible and this is checked by the immobile substrates and empty locations around the module. After this check the solenoids of the joints and magnets will be energized accordingly.

Table 4-3 Module Actuation

Joint	Direction	Immobile Substrate	Empty Locations			Goal
			D	E	K	
1	CW	A	D	E	K	F
2	CW	B	E	F	L	A
3	CW	C	F	A	G	B
4	CW	D	A	B	H	C
5	CW	E	B	C	I	D
6	CW	F	C	D	J	E
1	CCW	F	C	B	G	A
2	CCW	A	D	C	H	B
3	CCW	B	E	D	I	C
4	CCW	C	F	E	J	D
5	CCW	D	A	F	K	E
6	CCW	E	B	A	L	F

Note that the mobile electron always has negative fields and the immobile substrate will have a positive and a negative field to provide the rotations as shown in Figure 4-31. There will be repulsion between the two negative (green) fields and attraction between the negative (green) and the positive (red) field and the rotation will be performed around the fixed joint (red).

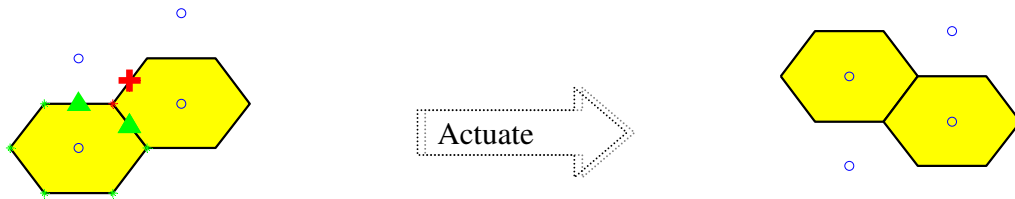


Figure 4-31 Mobile Electron Actuation

4.3.9 Simulation

The code *HexMove* attached in appendices will simulate the motion of the mobile electrons by providing the joint and side forces. This function is extremely useful since the exact same inputs going from the microcontroller into the physical platform (Actuation of magnets and solenoids) are also the inputs to this function and the resulting configuration is displayed.

4.4 Key Issues

4.4.1 Scalability

The first and perhaps most important question that comes to mind when evaluating a MSRRS algorithm is whether it is scalable and can be applied to systems with large number of modules.

As discussed by several researchers such as (Fitch & Butler, 2007), for large MSRRS any algorithm that requires linear space per module or linear time computation is undesirable. One of the main advantages of dynamic programming used in this work is their efficiency. It can be shown that the computational requirement for them is approximately a polynomial in the number of actions and states.

Moreover, algorithms are generally considered to be scalable if they are running in a decentralized manner on each module. In other words, decision making of each module running the algorithm does not require a complete knowledge of all other modules including their locations, actions, and etc. Accordingly, decision making in these systems is done through only local information and not the global knowledge of the system.

However, as a starting point, the approach presented in this work does require a centralized stage (global knowledge of the complete system) especially for layers 2 and 4. Other layers can be easily performed in a decentralized manner.

In layer 2 the connectivity test requires a centralized knowledge of the system and to the best of my knowledge there is still no decentralized method to overcome this problem. In layer 4 the motion planning for each module requires a centralized knowledge of the system for the MDP solver. This can be overcome using POMDP; although, learning techniques in POMDPs, such as Q-learning are no longer guaranteed to converge to a solution and therefore, more caution should be taken in to considerations with the solver of POMDPs. Another method proposed to overcome the limitation of the centralized stage in layer 4 is function approximation that is based on supervised learning (Sutton & Barto, 1998). Other techniques such as Gradient Ascent in Policy Space (GAPS) or Distributed GAPS are also used to provide a decentralized approach (Varshavskaya, 2007) to address the same issue.

4.4.2 Energy Consumption

Referring to Figure 4-31, the total energy required for each single rotation can be roughly calculated as follow:

- Four magnets (2 on each module) and each magnet requires a 24^V , 2^A and stays on for nearly 100^{ms} .
- Eight solenoids (5 on the mobile module and 3 on the three neighbors) and each solenoid requires 5^V , 0.5^A and stays on an average for nearly 100^{ms} .
- Drive circuit, microcontroller and communication require 5^V , 3^A for also 100^{ms} . (Microcontroller and communication will always stay on but their power consumption is neglected at this stage)

Therefore, the total power / energy required for a single motion is approximately:

$$P = 4*(24*2)+8*(5*0.5)+(5*3) = 227 \text{ W} \text{ and } E = 227*0.1 = 22.7 \text{ J}$$

Note the calculated E is actually the maximum energy required if all neighbor modules are available; however, since the major part of the energy is related to the Mag-

nets the overall energy consumption will not depend a lot on the neighboring modules and will be very close to the maximum E .

Therefore, since the approach presented has minimized the number of moves to transform the shape of the system from an initial configuration to a goal configuration, the total energy consumed will be very close to the multiple of maximum energy that could be required for the reconfiguration.

4.4.3 Discussion

As explained by (Nguyen, Guibas, & Yim, 2001) the optimal policy can be found in theory by searching the graph of all possible configurations; however, such search approaches are not practical since the number of possible configurations grows exponentially with the number of modules in the configuration.

As showed by (Chirikjian G. , 1994) in a simplified case dealt with hexagonal modules, the number of available configurations (N) is asymptotic to:

$$N = \frac{(2n - 1)!}{(n - 1)!(n + 1)} \left(\frac{5\sqrt{5}}{4}\right) \quad (4.15)$$

where n is the number of modules in the system. If each configuration is considered as a state to search for the optimal path, as the number of modules (n) increases, the total number of available configurations (N) grows even faster and therefore the approach would not be practical for large n .

Therefore, techniques relying on such a representation such as MMDPs require space and computation proportional to the product of the size of all the state and action variables, leading to intractable space (storage) and time complexity as discussed by (Fitch, Hengst, Suc, Calbert, & Scholz, 2005).

Conclusively, optimal policies for motion planning cannot be achieved by simple algorithmic search methods for a large number of modules (Chirikjian G. , 1994) that come with extreme computational cost and other scalable and decentralized methods like the one explained in this work shall be considered. The algorithm presented in this work is designed for limited number of modules to start with and to move towards a totally decentralized path planning for MSRRS in a near optimal manner for large number of modules later on.

4.5 Summary

Several works addressed the reconfiguration planning for MSRRS and still a solution of a totally decentralized technique is not found since most algorithms require a centralized stage. Also finding an optimal sequence of moves from I to G becomes intractable due to the fact that large number of modules comes with a large possible number of configurations which requires a computation time that is exponential in the number of modules.

This chapter described the algorithm used for transformation of the global shape of a MSRRS by determining the sequence of individual module movements that morphs the shape of the system from an initial configuration to a desired goal configuration.

The problem was formulated assuming module interchangeability and a hierarchical multilayer approach was recommended to solve the problem.

In the first layer, the initial configuration was obtained through localization and the goal configuration was assumed to be entered manually by the user. Second layer could separate mobile electrons and potential voids from the rest of the modules and each potential void was linked to a mobile electron in the third layer. Forth layer planned the path for electrons to fill the voids and finally last and fifth layer transformed the required motion into actuation for each module

It was shown that iterations converge all the time and the number of iterations does not depend heavily on the resolution but depends heavily on the value of the discount factor. It was also shown that large discount factors provide longer horizon for modules but requires more computations as oppose to small discount factors providing short horizon and quick computations.

Chapter 5

Evaluation

This chapter is aimed to evaluate the performance of the physical platform and the reconfiguration algorithm through several experiments and simulations.

5.1 Experimental Setup

The experimental setup is shown in Figure 5-1.

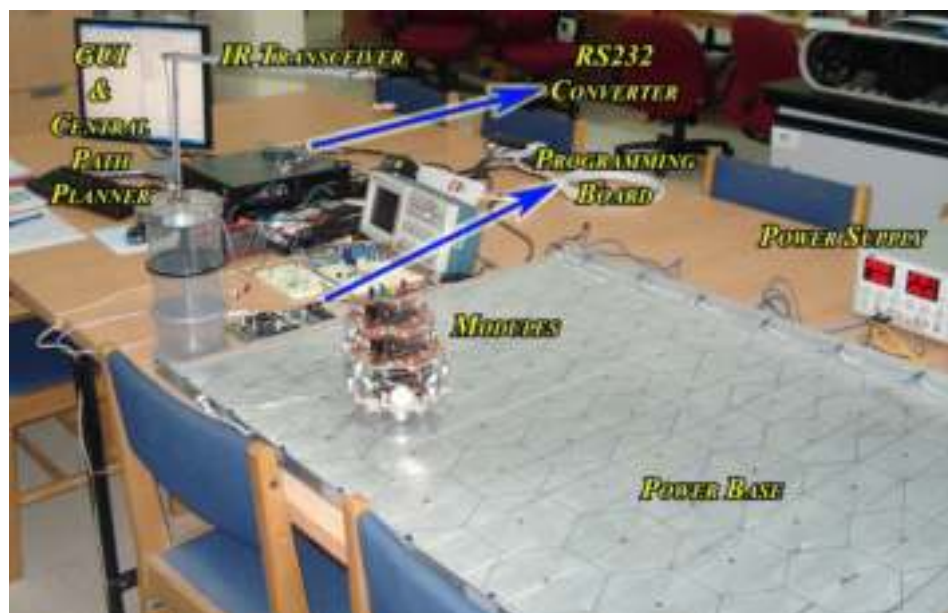


Figure 5-1 Experimental Setup

As illustrated in Figure 5-1, the system is equipped with a Graphical User Interface (GUI) communicating with modules through serial port of the computer using an RS232 Converter and an IR Transceiver. The modules can be quickly and easily reprogrammed if necessary without removing the microcontrollers using In-System-Programming (ISP) feature through STK500 Programming Board.

5.2 Examples and Experiments

Availability of only two physical modules at the time restricted the experiments to be more on the simulation side rather than the practical one; however, the practical experiments could sufficiently prove the basic concept and the design of the universal modules and the system is planned to be expanded to work with several modules in future.

5.2.1 Physical Platform Performance

This section aims to validate the two main requirements of the designed modules: Localization and Rotation.

First all boards including power, drive, control and communication were tested and their proper functionalities were ensured. Then control and communication microcontrollers were programmed with all required functions (program codes and functions are attached in appendices). Communication ports were also configured as shown in Figure 5-2.



Figure 5-2 Port Configuration

Note that UART1 is a temporary connection providing direct access to control board of any given module and is designed for testing purposes only. IR (InfraRed) on the other hand is the main communication channel providing direct access to all modules' functionalities through their communication boards.

Each module was turn on to check the handshaking status between the control and communication microcontrollers and the result of one test is shown in Figure 5-3.

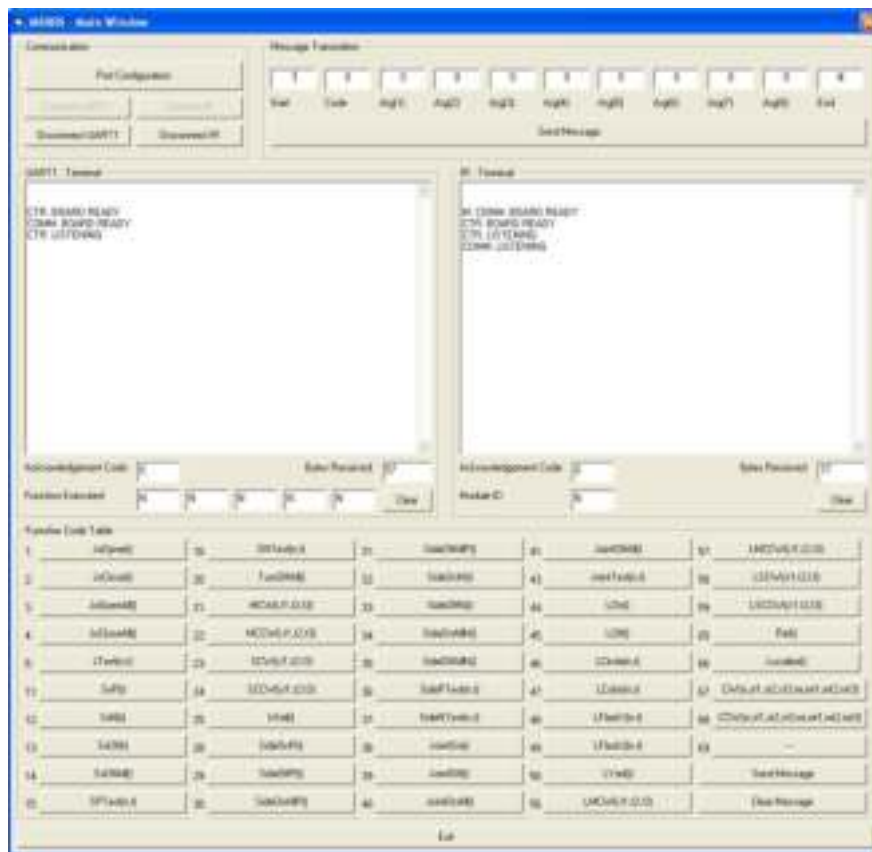


Figure 5-3 Handshaking

All joints (solenoids), sides (magnets), LEDs, IRs and all functions were tested for each module.

Once both modules were fully tested and all hardware and software components proved to be functional, they were placed on the power base as shown in Figure 5-4 to test localization and motion functions.



Figure 5-4 Localization Test

Before conducting the localization experiment, one module was chosen to be the reference module (localized with known coordinates and orientation) by running the *Ref()* function as shown in Figure 5-5.

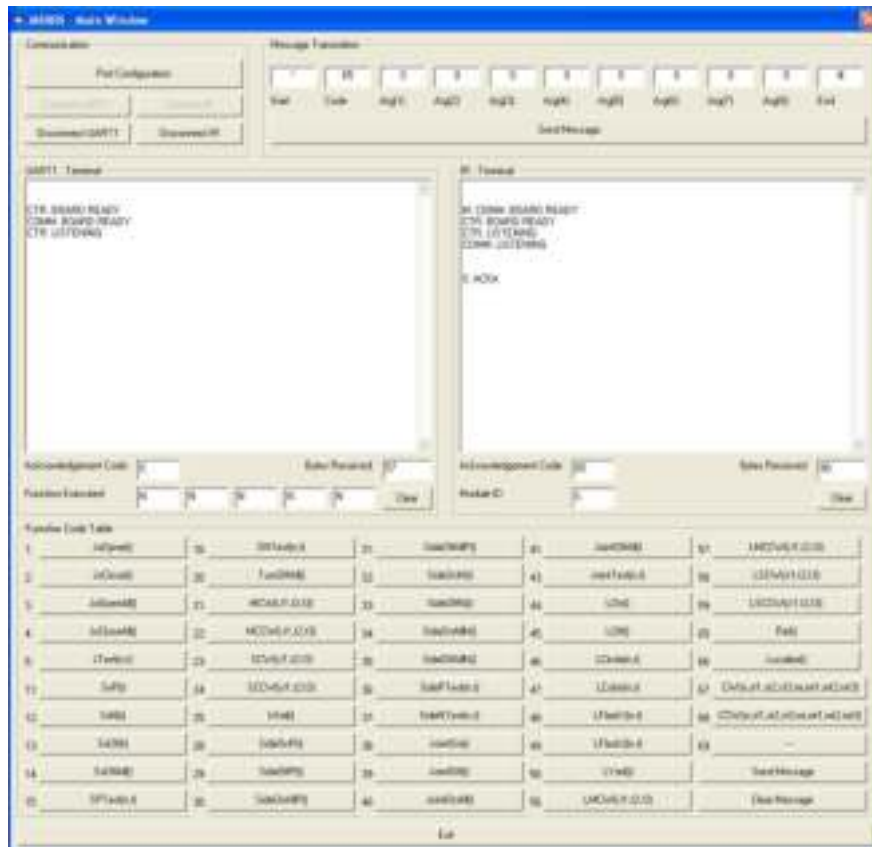


Figure 5-5 Reference Module

This function returned the correct acknowledgement code of 65 and set the module as a substrate (S) module as well.

In order to localize the other module, the *Localize()* function was called and executed. At this time the localized module sent the localization message (MSG1) discussed in Chapter 4 to the other module and the other module was localized accordingly. The acknowledgment codes were sent from both modules as shown in Figure 5-6.

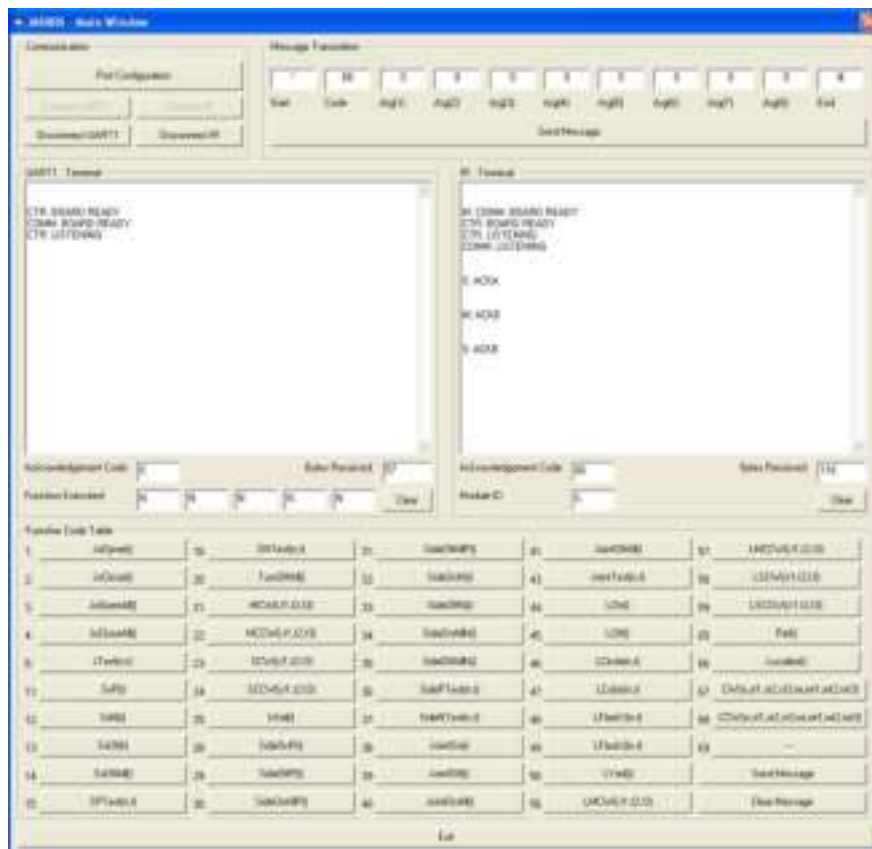


Figure 5-6 Localization

Similarly the Clockwise and Counterclockwise functions were tested by calling *CW()* and *CCW()* functions and steps are shown in Figure 5-7.

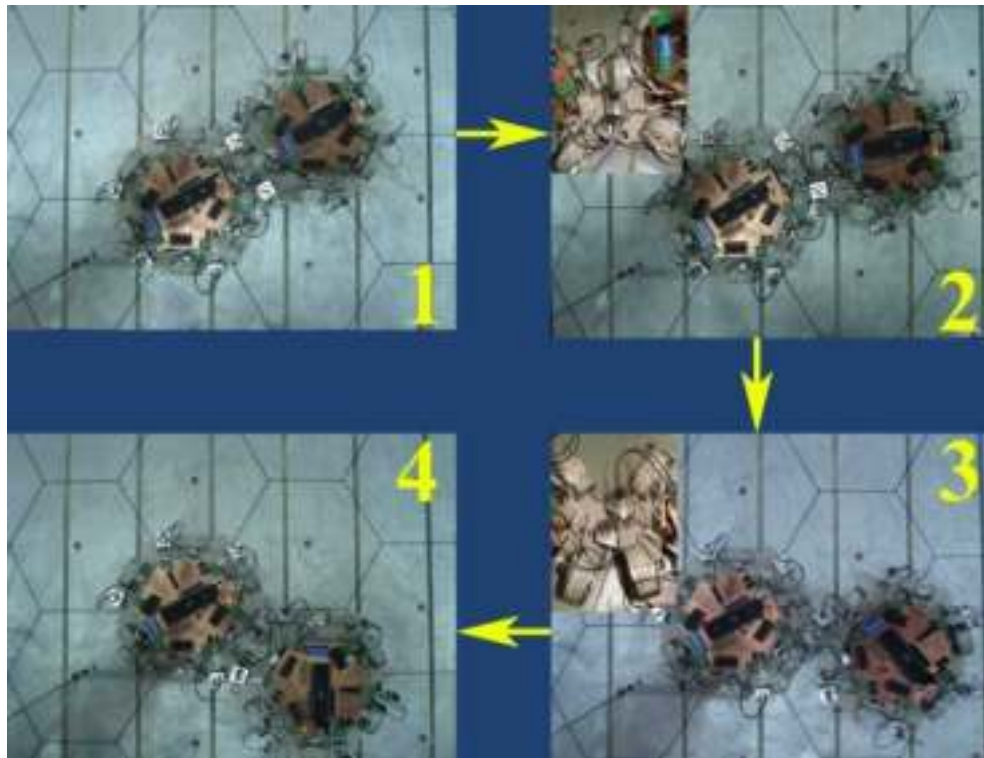


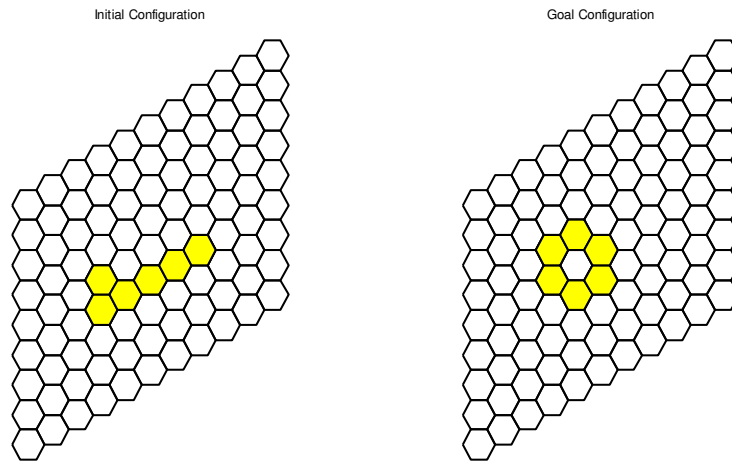
Figure 5-7 Rotation Test

5.2.2 Reconfiguration Algorithm Performance

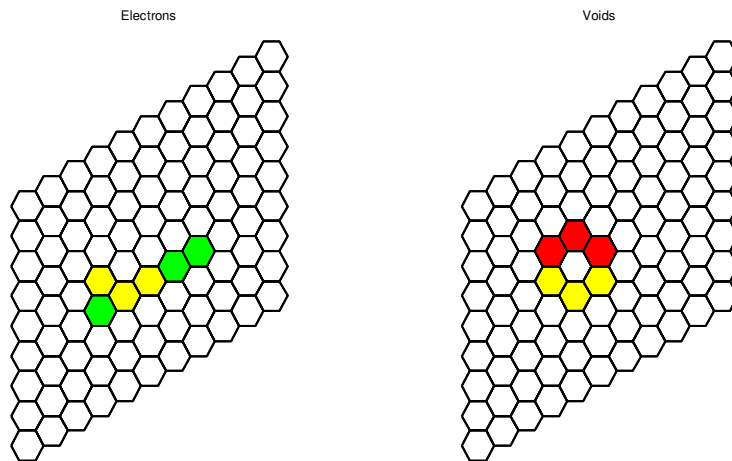
The physical platform performance test was followed by the reconfiguration algorithm performance test. This test was no longer limited to two modules and the simulator could provide an excellent chance to test the algorithm.

The two main functions used for the simulations were $HexAll_P()$ and $HexAll_S()$ attached in appendices. These functions require only the initial and goal configurations as their input arguments. By combining all layers of the reconfiguration algorithm, these functions directly provide the actuation required both for side magnets and joint solenoids at each time step. $HexAll_P()$ is designed to provide *parallel* motion for more than one module and $HexAll_S()$ is designed to provide a *serial* motion for only one module at a time.

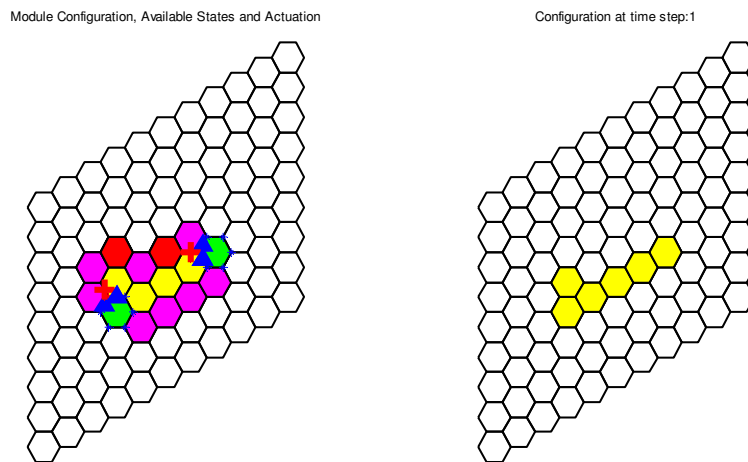
Let's start examining the algorithm by running the familiar example discussed throughout Chapter 4. The code for this example is attached in appendices as $Test_All_P01$ and the results are shown in Figure 5-8.



a) Initial and goal configurations

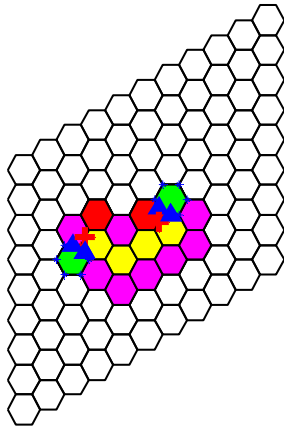


b) Electrons and voids

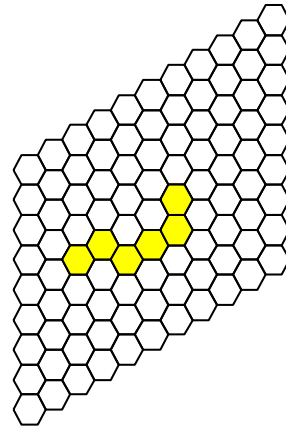


c) Time step 1

Module Configuration, Available States and Actuation

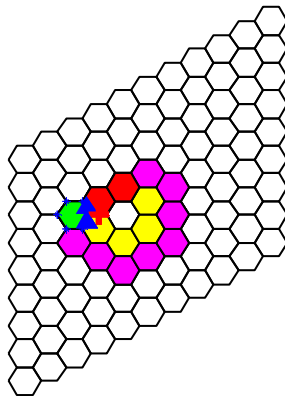


Configuration at time step:2

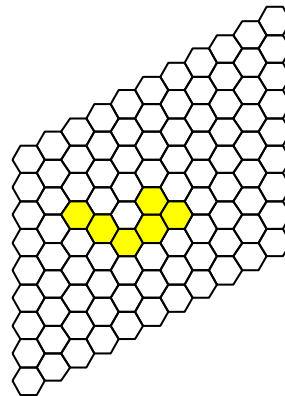


d) Time step 2

Module Configuration, Available States and Actuation

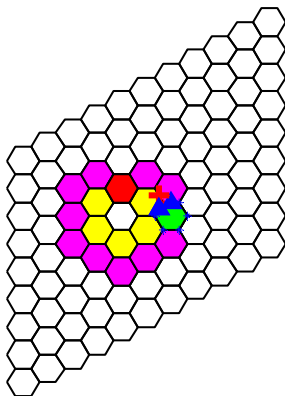


Configuration at time step:3

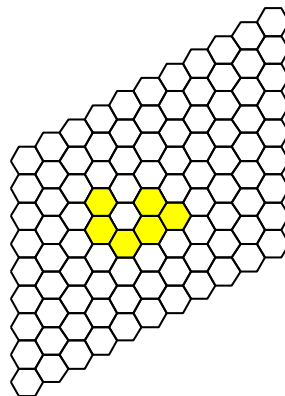


e) Time step 3

Module Configuration, Available States and Actuation

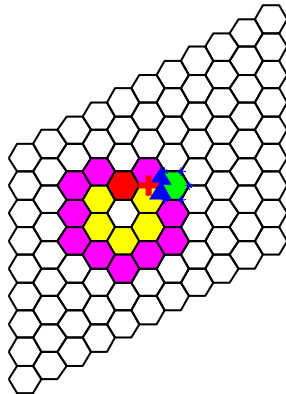


Configuration at time step:4

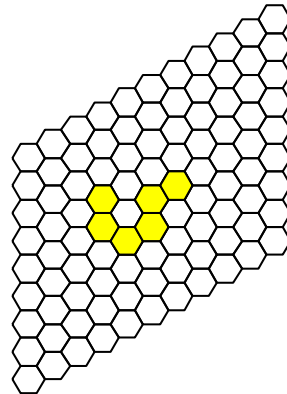


f) Time step 4

Module Configuration, Available States and Actuation

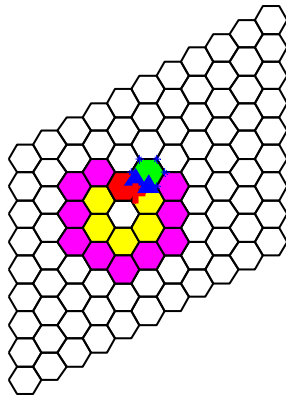


Configuration at time step:5

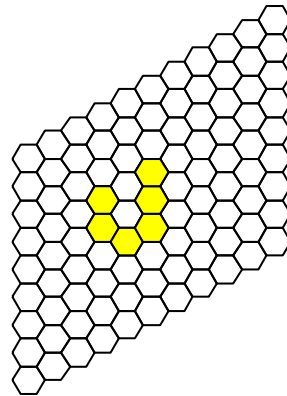


g) Time step 5

Module Configuration, Available States and Actuation

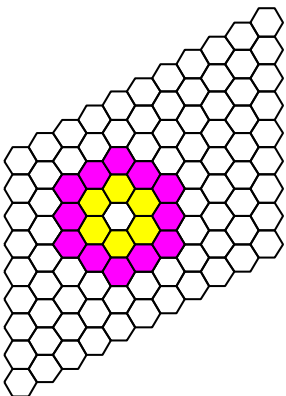


Configuration at time step:6

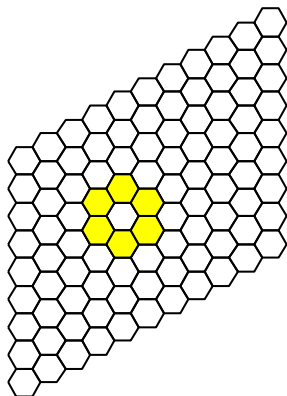


h) Time step 6

Module Configuration, Available States



Configuration at time step:7

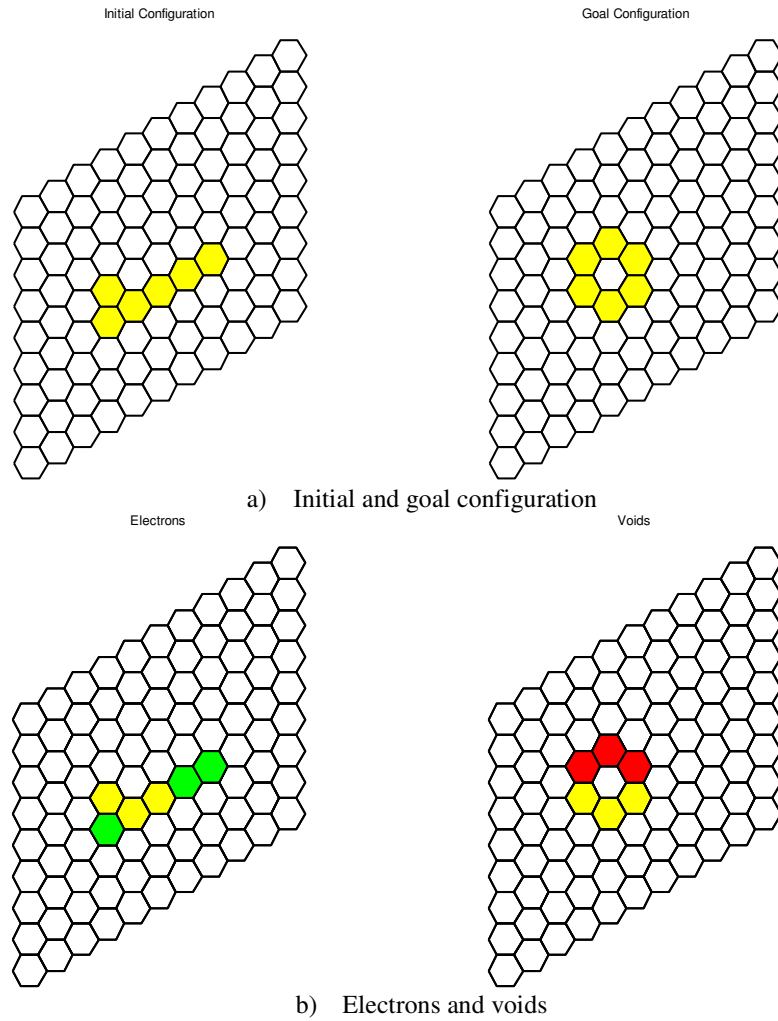


i) Time step 7

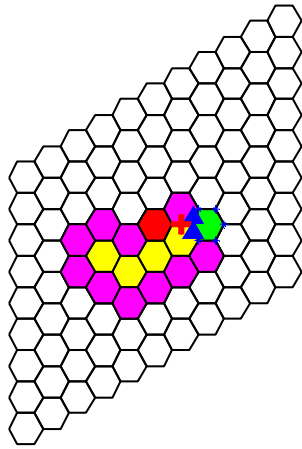
Figure 5-8 Simulation Test 1

As can be seen in this simple example, the initial configuration in time step 1 was transformed into the desired goal configuration in six time steps using parallel module motions.

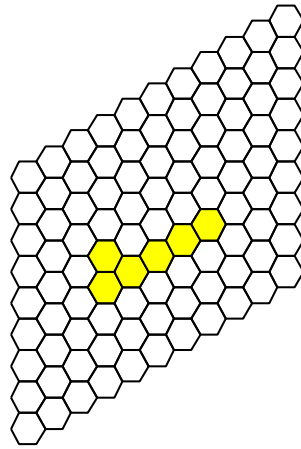
The same example was tested by *Test_All_S01* (code attached in appendices) for serial motion execution and the results are shown in Figure 5-9.



Module Configuration, Available States and Actuation

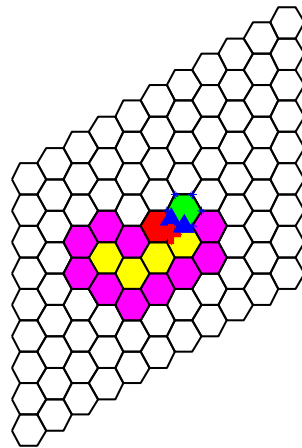


Configuration at time step:1

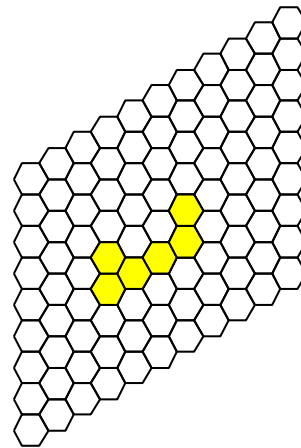


c) Time step 1

Module Configuration, Available States and Actuation

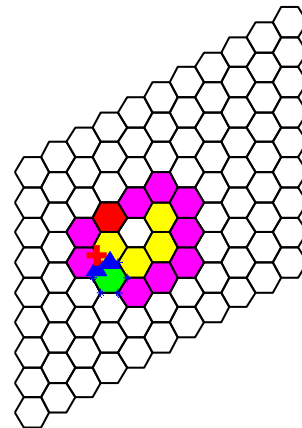


Configuration at time step:2

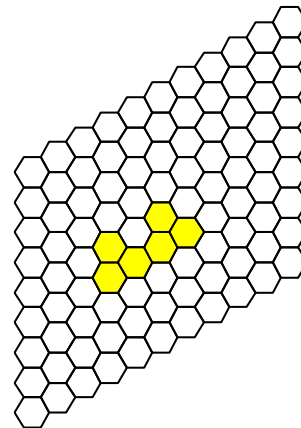


d) Time step 2

Module Configuration, Available States and Actuation

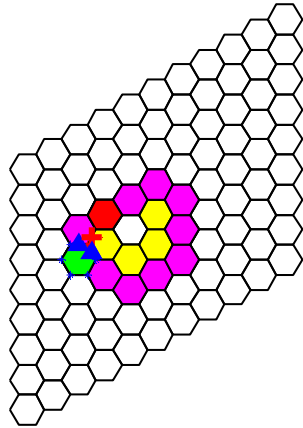


Configuration at time step:3

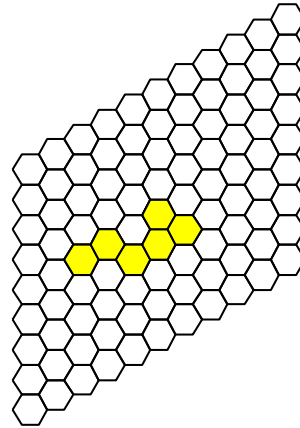


e) Time step 3

Module Configuration, Available States and Actuation

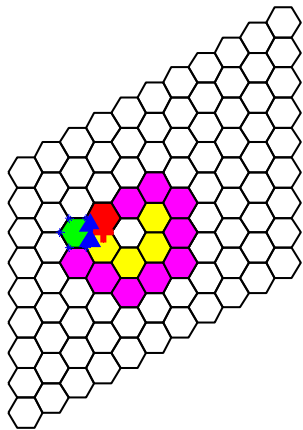


Configuration at time step:4

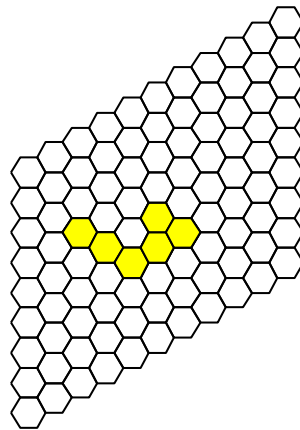


f) Time step 4

Module Configuration, Available States and Actuation

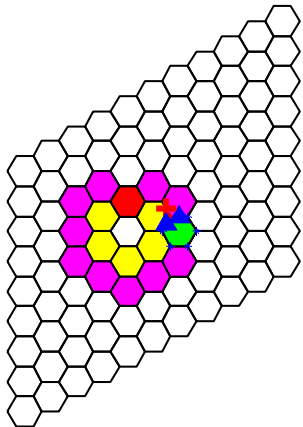


Configuration at time step:5

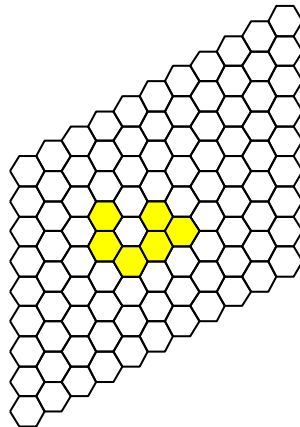


g) Time step 5

Module Configuration, Available States and Actuation

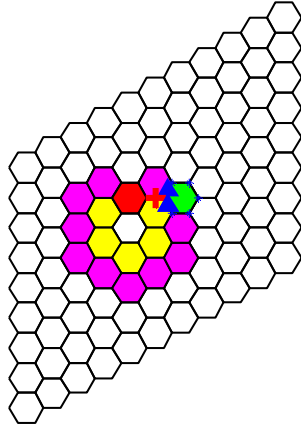


Configuration at time step:6

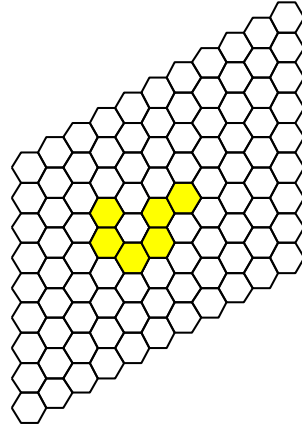


h) Time step 6

Module Configuration, Available States and Actuation

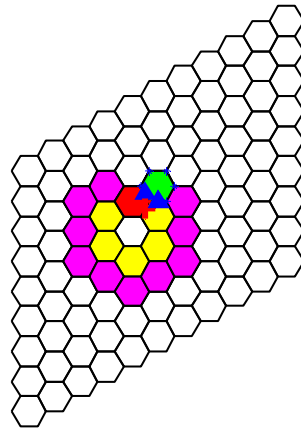


Configuration at time step:7

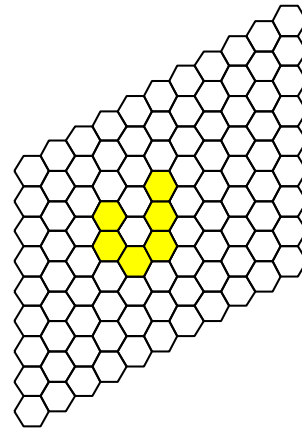


i) Time step 7

Module Configuration, Available States and Actuation

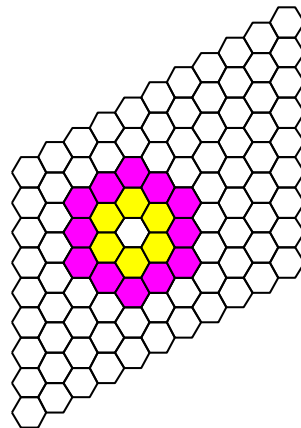


Configuration at time step:8

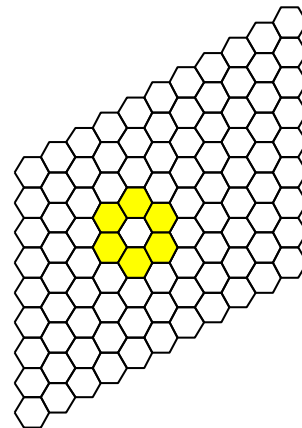


j) Time step 8

Module Configuration, Available States



Configuration at time step:9

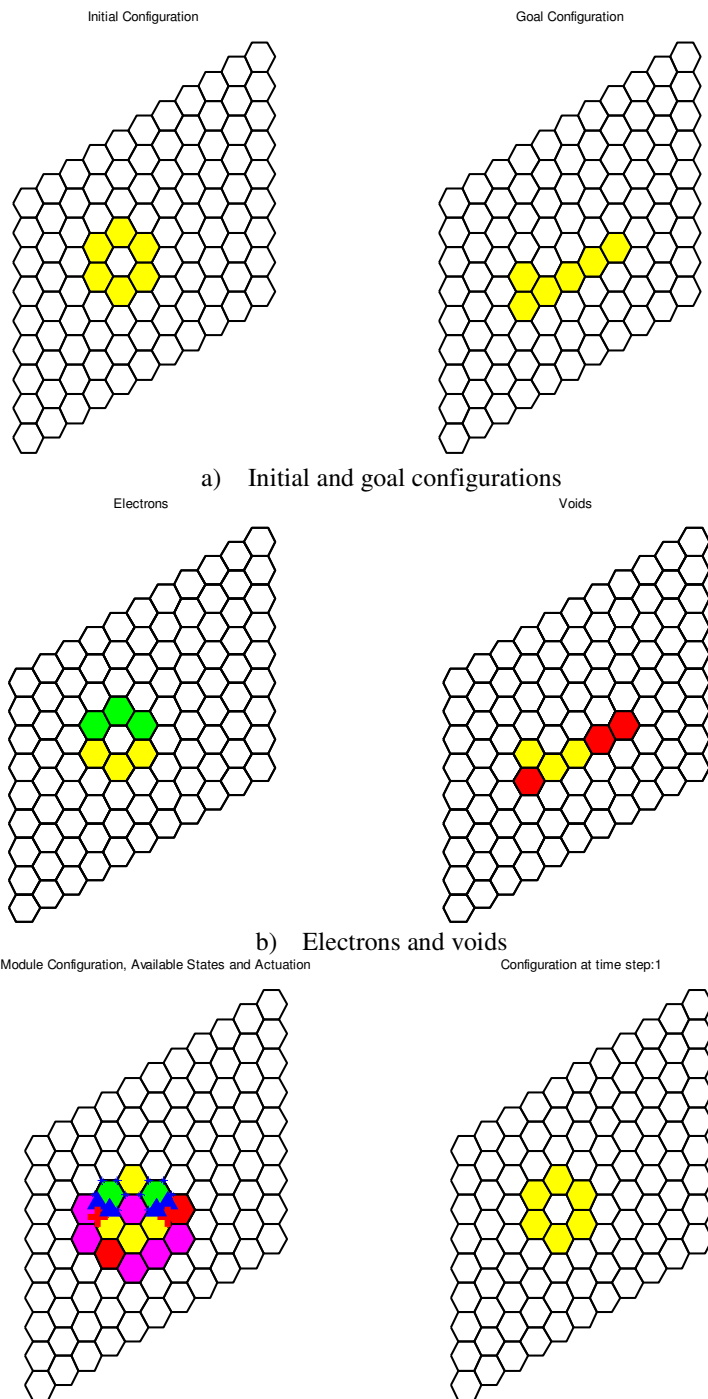


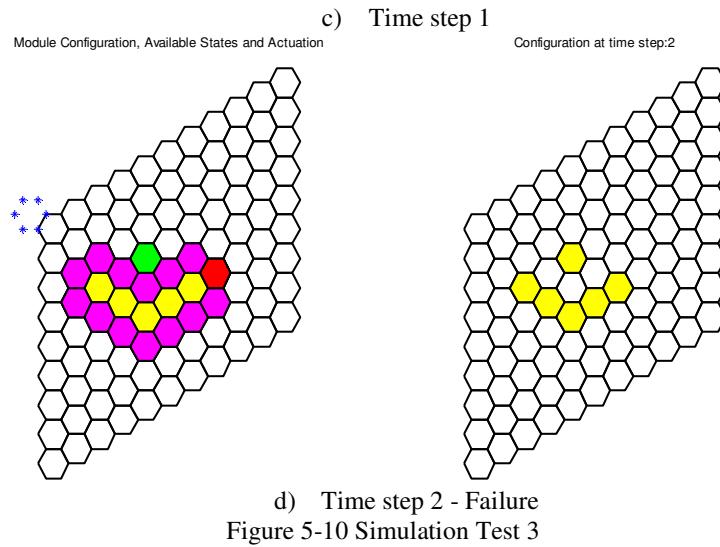
k) time step 9

Figure 5-9 Simulation Test 2

As expected the serial motion requires more time steps; however, number of module movements (which is considered to be minimized by the algorithm) is exactly the same as the parallel module movements.

The next example in Figure 5-10 illustrates a case where the parallel motion faces a problem and the reconfiguration can be performed through serial motion.

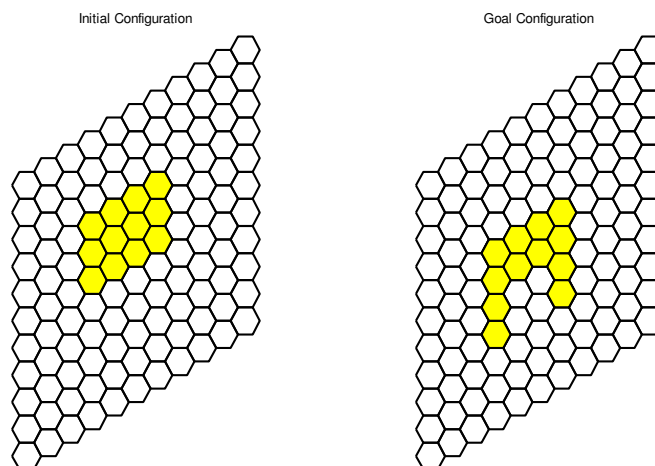




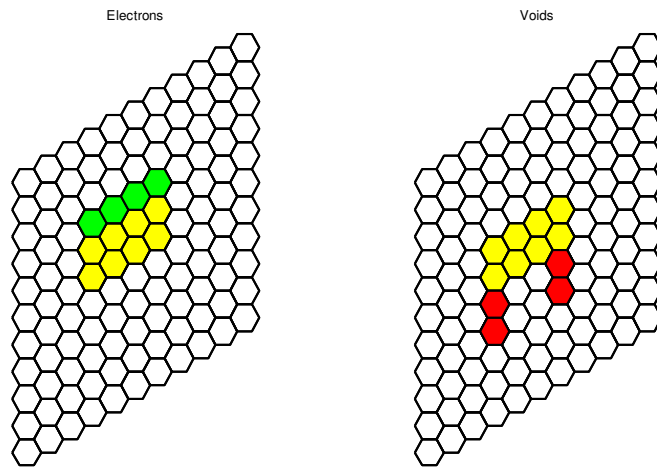
As can be seen, in the first time step, two electrons check the connectivity at the same time and find themselves as mobile electrons and start their motion. However, by doing so they leave a third module disconnected from the rest of the system.

This example illustrates a potential advantage of a serial motion planning compared to a parallel motion planning in the developed algorithm. However, since the emphasis was to minimize the number of module movements which is irrespective to parallel or serial motion execution, $HexAll_S()$ function is sufficient to plan such a reconfiguration.

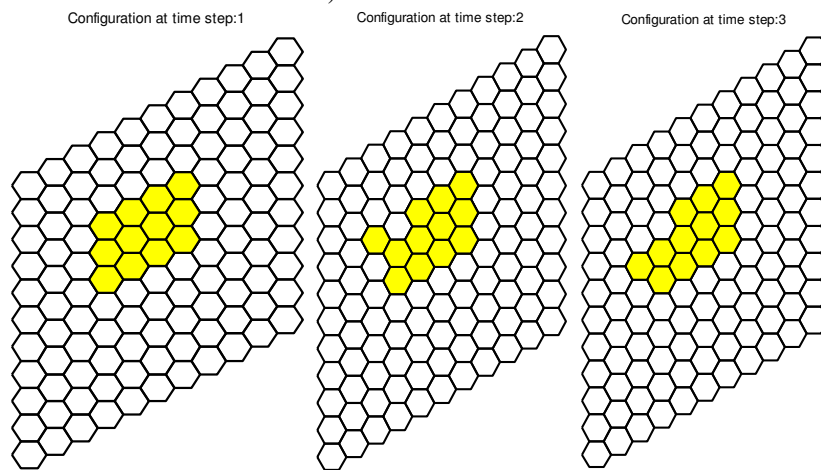
The last example is planned for a larger system consisting of 12 modules. In this example the central computer required much more time to process the reconfiguration path as the number of modules was increased to twice the previous examples.



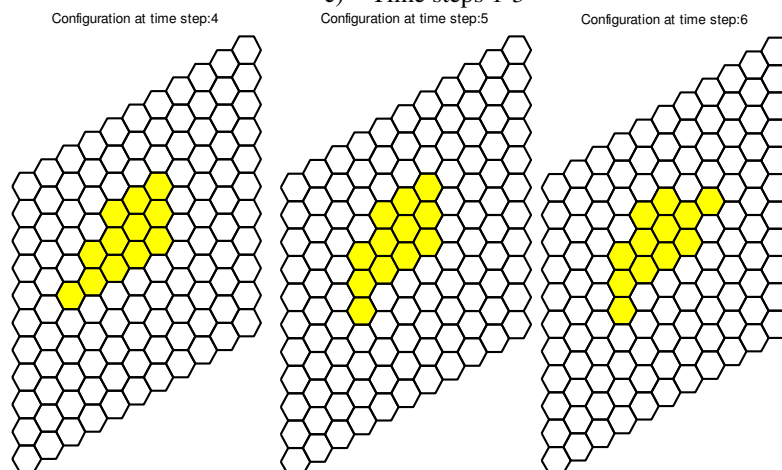
a) Initial and goal configurations



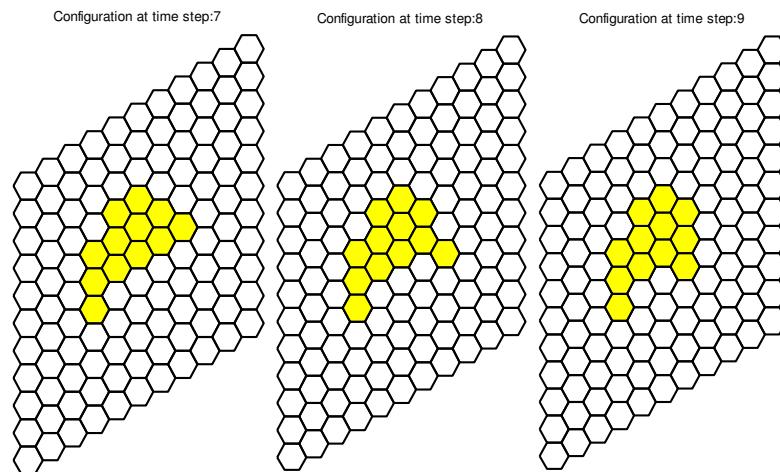
b) Electrons and voids



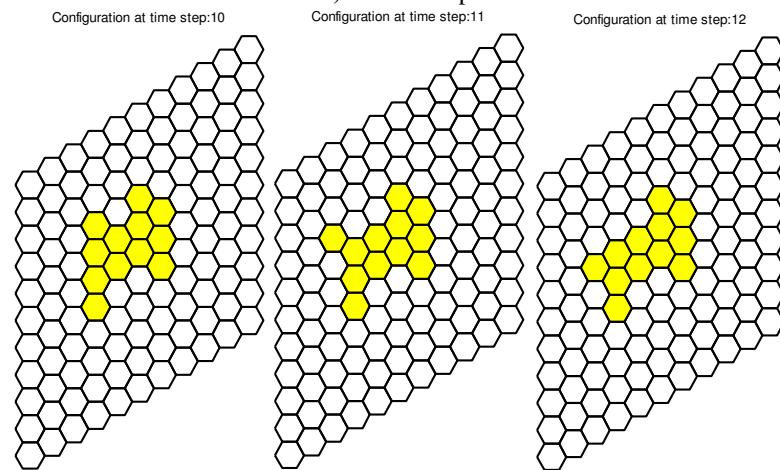
c) Time steps 1-3



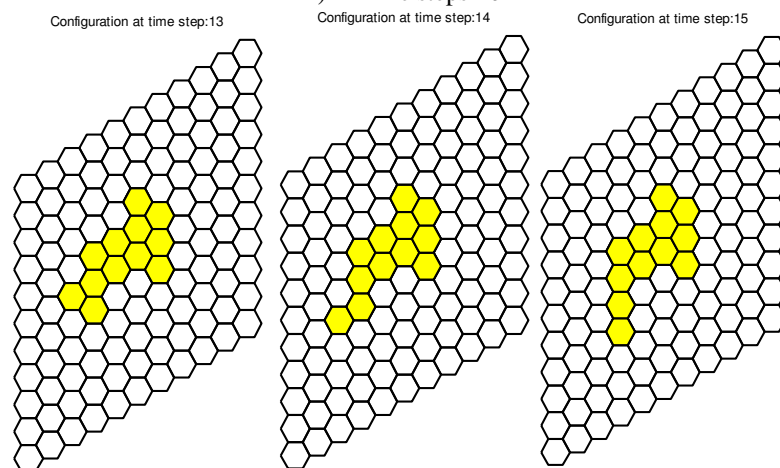
d) Time steps 4-6



e) Time steps 7-9



f) Time steps 10-12



g) Time steps 13-15

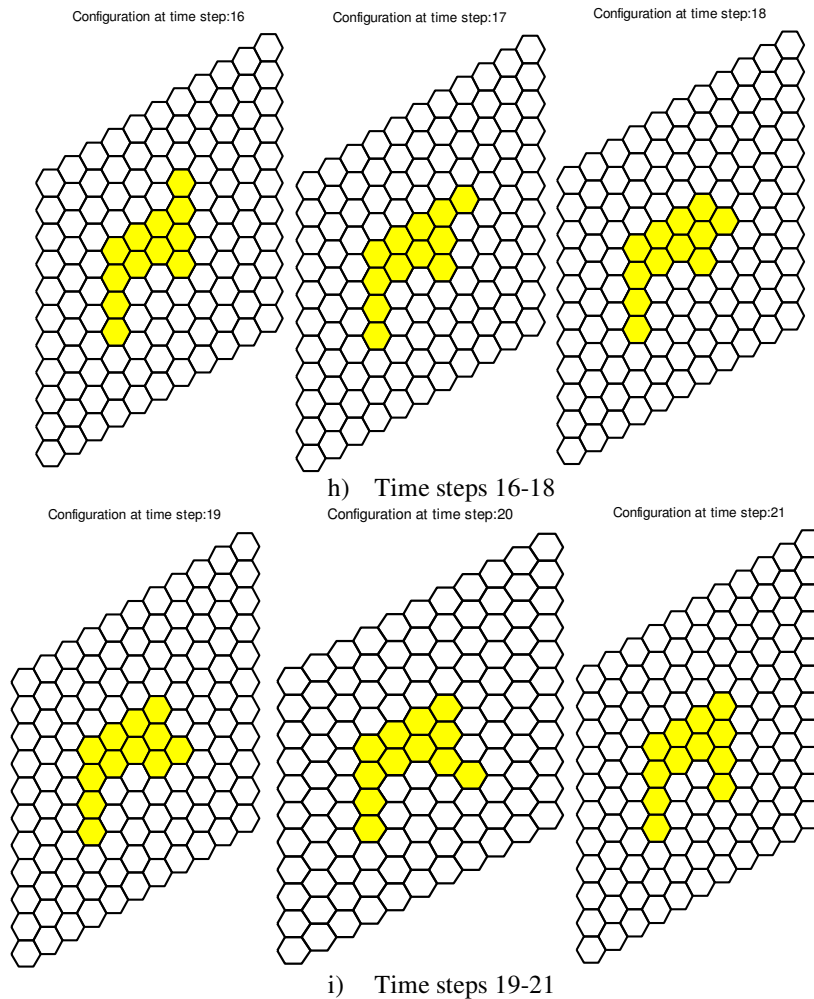


Figure 5-11 Simulation Test 4

As can be seen the reconfiguration took place in 20 time steps with a serial execution path planner. This example illustrated one of the main drawbacks of a centralized processing compared to a decentralized processing in terms of execution time.

5.3 Discussions

Evaluating the overall performance of a MSRRS by comparing to other MSRRSs is a relatively difficult challenge because of the following reasons.

Most systems designed so far are addressing only a specific area from the expected features of an ideal MSRRS. For example, some address locomotion or reconfiguration while others realize self-replication or self-repair. Clearly such systems would perform well for one area and poor in other areas. Still a complete system featuring all expected functionalities is yet to come.

Most algorithms developed are custom made for a specific class. For example some 2D algorithms cannot be easily expanded to 3D or some algorithms for rectangular modules cannot be realized for hexagonal modules and etc. To make the situation worse many developed algorithms rely heavily on specific platform features. For example, existence of push bottom switches on the side as an indication of the existence on a neighboring module.

To the best of my knowledge, there is still no well developed method or analysis that can be applied generally to all MSRRSs.

5.3.1 HexBot Evaluation Criteria and Performance

The following metrics and evaluation criteria are used in this section to summarize and address the performance evaluation of the HexBot.

5.3.1.1 HexBot Physical Platform

- **Scalability:** HexBot platform can be easily scaled down in size as the required actuation relies only on electromagnetic forces. The platform can also be extended to 3D using the similar actuators and connectors. In terms of quantity, HexBot can also be scaled up since the module can be mass-manufactured in an extremely cost effective manner as there are no expensive sensors or actuators involved.
- **Robustness:** Handshaking between the two microcontrollers of the modules and sending the status to the main processor (PC in this case) can be used as an indication of module failure; however, more work needs to be done to implement real time failure detection.
- **Speed of Response:** Magnetic actuators provide fast motion and do not require precise alignment between two modules since the alignment is done mechanically. Joint connectors are quick with low power consumption and they do not require power once the modules are connected
- **Power Consumption:** This is considered the main drawback in HexBot design. Quick and strong magnetic force comes at a cost of high power consumption.
- **Cost-effectiveness:** HexBot can be mass-produced relatively cheap since there are no expensive sensors or actuators in the design.

- **Physical:** In terms of geometric shape, HexBot modules have the advantage of filling any required structure densely with no gaps; however these modules are primarily developed for research and are not strong enough for real world applications.

5.3.1.2 HexBot Control Algorithm

- **Robustness:** The algorithm is not designed to detect failure in the module movements or perform a self-repair action. The only indication of a proper function execution is the acknowledgement message sent from the modules to the central controller. Moreover the overall system is not robust to work in different environment as there are no sensory feedback and technique implemented for obstacle avoidance or etc.
- **Scalability:** In terms of scalability to different platforms, the algorithm is perfectly scalable and requires only slight modification in specific layers; however the algorithm is not considered to be an architecture independent since all the modifications need to be done manually. Besides, the current version cannot be scaled up in quantity as well since it relies on a centralized controller with limiting factors: processing power, communication bandwidth and etc.
- **Stability:** Convergence property of MDP is well understood and therefore the algorithm will ensure converging to a solution and there are no dead locks.
- **Optimality:** The heuristic function used in layer 4 (void propagation) along with the optimization method performed in layer 5 (MDP) result a near optimum solution. Therefore the optimality of the overall algorithm would depend on the combination of these two factors. For example, in situations where heuristic function works well the solution will be very close to the global optimum solution and if the heuristic function does not perform well then the solution will not be very optimum.
- **Adaptability:** Reconfiguration algorithm developed can support adaptation to different tasks or working environment; however, more work needs to be done in this field.

Chapter 6

Concluding Remarks

Finally a MSRRS project was established in the AUS Mechatronics Center and hopefully this project would continue and contribute to the growing field of modular self-reconfigurable robotic system.

6.1 Summary

We have motivated, introduced and established the MSRRS project both from hardware and software aspects in this work. We have deeply reviewed, organized and presented the relevant literature along with the most successful platforms and algorithms.

We have demonstrated the main criteria behind HexBot as our universal module in terms of design and implementation. We have also provided a comprehensive background including preliminaries needed for the reconfiguration algorithm and explained in details how the algorithm was developed.

Finally we have evaluated the performance of both platform and algorithm through several examples and discussed the results.

6.2 Conclusions

HexBot was successfully implemented and a reconfiguration algorithm for a planar hexagonal MSRRS was developed.

The design of the universal module required a well understanding of the ultimate common goals in the area and we specifically focused on the two dimensional, homogeneous systems. We have developed an extremely fast actuation which is exceptionally competitive to other designs where other kinds of actuators are utilized. Inter-module connections are designed to be quick and powerful without requiring precise alignment. All electronics boards were designed in a multilayered manner where each board has specific task and can be modified or replaced separately.

The control algorithm was successfully designed and implemented to transform the global shape of the system from an arbitrary initial configuration to a desired goal configuration. The approach incorporated the development of a hierarchical multilayer framework for lattice based modular systems to optimize and plan paths for the minimum number of module movements. The overall problem was successfully formulated as a Markov Decision Process (MDP) that could be easily adopted for other platforms. In the policy search collision avoidance and connectivity constraints were implemented. Multilayered nature of the framework provides openness, flexibility and ease of modification and improvement for each individual layer.

6.3 Limitations and Directions for Future Research

Clearly this work was just planned to pave the way for others to continue and contribute more to this exciting field.

The physical platform can be improved in the following ways:

1. Friction with the environment shall be minimized through:
 - a. An air-table can be used to reduce the normal force
 - b. Ball transfer-table can be used to reduce the weight, since ball transfers will not be mounted on the modules
 - c. Battery instead of power based to eliminate the need and friction of the power pins

- d. Mechanical parts shall be designed with a more strength to weight ration to reduce the weight
2. Better magnets (in terms of size and magnetic force) can be replaced
3. Size can be reduced
4. Eventually we should also move toward a 3D environment

The control algorithm can also be improved in the following ways:

1. Parallel actuation problem shall be overcome
2. Reward function should be upgraded to allow motion for fixed modules which are blocking the way for other modules to move
3. MDP can be replaced with POMDP or other distributed ways of path planning for scalability
4. There should be a distributed way of testing the connectivity constraint to move towards a totally distributed controller

Bibliography

Bishop, J., Burden, S., Klavins, E., Kreisberg, R., Malone, W., Napp, N., et al. (2005). Programmable parts: a demonstration of the grammatical approach to self-organization. *International Conference on Intelligent Robots and Systems*, (pp. 3684-3691).

Bojinov, H., Casal, A., & Hoag, T. (2000). Emergent structures in modular. *Proceedings of IEEE International Conference on Robotics and Automation*, (pp. 1734-1741). San Francisco.

Buckley, F., & Lewinter, M. (2003). *A Friendly Introduction to Graph Theory*. New Jersey: Prentice Hall.

Butler, Z., & Rus, D. (2003). Distributed Planning and Control for Modular Robots with Unit-Compressible Modules. *The International Journal of Robotics Research*, 22 (9), 699-715.

Butler, Z., Byrnes, S., & Rus, D. (2001). Distributed motion planning for modular robots with unit-compressible modules. *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (pp. 790-796). Hawaii, USA.

Butler, Z., Fitch, R., & Rus, D. (2002). Distributed control for unit-compressible robots: Goal-recognition, locomotion and splitting. 7 (4), 418-430.

- Butler, Z., Kotay, K., Rus, D., & Tomita, K. (2002). Generic decentralized control for a class of self-reconfigurable robots. *Proceedings of IEEE ICRA02*.
- Casal, A., & Yim, M. (1999). Self-Reconfiguration Planning for a Class of Modular Robots. *Proceedings of SPIE, Sensor Fusion and Decentralized Control in Robotic Systems*, (pp. 246-257).
- Cassandra, T., Littman, M., & Kaelbling, L. (2003, November 6). *Partially Observable Markov Decision Process*. Retrieved September 12, 2008, from POMDP: <http://www.pomdp.org>
- Castano, A., & Will, P. (2000). Mechanical design of a module for autonomous reconfigurable robots. *Proceedings of IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, (pp. 2203–2209).
- Castano, A., Behar, A., & Will, P. M. (2002). The Conro modules for reconfigurable robots. *IEEE/ASME Transaction of Mechatronics* , 7 (4), 403-409.
- Castano, A., Chokkaingam, R., & Will, P. (2000). Autonomous and selfsufficient conro modules for reconfigurable robots. *Proceedings, 5th International Symposium on Distributed Autonomous Robotic Systems (DARS'00)*, (pp. 155-164). Knoxville, Texas, USA.
- Castano, A., Shen, W. M., & Will, P. (2000). CONRO: Towards Deployable Robots with Inter-Robots Metamorphic Capabilities. *Autonomous Robots* , 8 (3), 309-324.
- Chiang, C. H., & Chirikjian, G. (2001). Modular robot motion planning using similarity metrics. *Autonomous Robots* , 10 (1), 91-106.
- Chiang, C. J., & Chirikjian, G. (2001). Similarity metrics with applications to modular robot motion planning. *Autonomous Robots* , 10 (1), 91-106.
- Chirikjian, G. (1994). Kinematics of a metamorphic robot system. *IEEE International Conference of Robotics and Automation*, (pp. 449-455). San Diego.

- Chirikjian, G., Pamecha, A., & Ebert-Uphoff, I. (1996). Evaluating efficiency of self-reconfiguration in a class of modular robots. *Robot System*, 13 (5), 317-338.
- Christensen, D. J., & Stoy, K. (2006). Selecting a Meta-Module to Shape-Change the ATRON Self-Reconfigurable Robot. *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, (pp. 2534-2538). Orlando, Florida.
- Christensen, D. J., Ostergaard, E., & Lund, H. (2004). Metamodule control for the ATRON self-reconfigurable robotic system. *Proceedings of the The 8th Conference on Intelligent Autonomous Systems (IAS-8)*, (pp. 685-692).
- Dumitrescu, A., Suzuki, I., & Yamasashita, M. (2002). High speed formations of reconfigurable modular robotic systems. *Proceedings of IEEE Internatinal Conference of Robotics and Automation*, (pp. 123-128). Washington, DC.
- Fitch, R., & Butler, Z. (2007). Scalable Locomotion for Large Self-Reconfiguring Robots. *IEEE Inernational Conference on Robotics and Automation*, (pp. 10-14). Rome.
- Fitch, R., butler, Z., & Rus, D. (2005). Reconfiguration Planning Among Obstacles for Heterogeneous Self-Reconfiguring Robots. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, (pp. 117-124). Barcelona, Spain.
- Fitch, R., Butler, Z., & Rus, D. (2003). Reconfiguration Planning for Heterogeneous Self-Reconfiguring Robots. *Proceedings of the 2003 IEEWRSJ International Conference on Intelligent Robots and Systems*, (pp. 2460-2467). Las Vegas, Nevada.
- Fitch, R., Hengst, B., Suc, D., Calbert, G., & Scholz, J. (2005). Structural Abstraction Experiments in Reinforcement Learning. In *AI 2005: Advances in Artificial Intelligence* (Vol. 3809/2005, pp. 164 - 175). Sydney, Australia: Springer Berlin / Heidelberg.
- Fitch, R., Rus, D., & Vona, M. (2000). A basis for self-repair using crystalline modules. *Proceedings of Intelligent Autonomous Systems Conference (IAS-6)*, (pp. 903-909).
- Fukuda, T., & Kawakuchi, Y. (1990). Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator. In *Proceedings of IEEE Conference on Robotics and Automation (ICRA)*, (pp. 662-667).

- Fukuda, T., & Nakagawa, S. (1987). A Dynamically Reconfigurable Robotic System (concept of a system and optimal configurations). *Proceedings of IEEE International Conference on Industrial Electronics, Control, and Instrumentation* (pp. 588-595). Los Alamitos, CA: IEEE Computer Society Press.
- Fukuda, T., & Nakagawa, S. (1988). Approach to the Dynamically Reconfigurable Robotic System. *Journal of Intelligent and Robot Systems* , 55-72.
- Fukuda, T., & Nakagawa, S. (1988). Dynamically reconfigurable robotic system. *Proceedings of the 1988 IEEE International Conference on Robotics and Automation. 3*, pp. 1581–1586. Los Alamitos, CA: IEEE Computer Society Press.
- Fukuda, T., & Nakagawa, S. (1990). Method of autonomous approach, docking and detaching between cells for dynamically reconfigurable robotic system CEBOT. *JSME International J. III-VIB. C. , 33* (2), 263-268.
- Fukuda, T., & Ueyama, T. (1994). *Cellular robotics and micro robotic systems*. London: World Scientific Publishing.
- Fukuda, T., Buss, M., Hosokai, H., & Kawauchi, Y. (1991). Cell structured robotic system CEBOT: Control, planning and communication. *7* (2-3), 239-248.
- Fukuda, T., Nakagawa, S., Kawauchi, Y., & Buss, M. (1988). Self organizing robots based on cell structures—CEBOT. *Proceedings of the 1988 IEEE International Workshop on Intelligent Robots* (pp. 145-150). Los Alamitos, CA: IEEE Computer Society Press.
- Fukuda, T., Sekiyama, K., Ueyama, T., & Arai, F. (1993). Efficient communication method in the cellular robotic system. *Proceedings of the 1993 IEEE/RSJ Internatioanl Conference on Intelligent Robots and Systems. 2*, pp. 1091-1096. Los Alamitos, CA: IEEE Computer Society Press.
- Fukuda, T., Ueyama, T., & Kawauchi, Y. (1990). Self-organization in cellular robotic system (CEBOT) for space application with knowledge allocation method. *Proceedings*

of the 1990 International Symposium on Artificial Intelligence, Robotics and Automation in Space, (pp. 101-104). Kobe, Japan.

Fukuda, T., Ueyama, T., & Sekiyama, K. (1995). Distributed intelligent systems in cellular robotics. *Artificial Intelligence in Industrial Decision Making, Control and Automation* , 225-246.

Gilpin, K., Kotay, K., & Rus, D. (2007). Miche: Modular Shape Formation by Self-Dissassembly. *IEEE International Conference on Robotics and Automation (ICRA07)*, (pp. 2241-2247). Roma, Italy.

Goldstein, S., Mowry, T., Gibbons, P., Pillai, P., Rister, B., & Lee, P. (2006). Ensembles of Millions of Microbots. *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*. Orlando, FL.

Hamlin, G., & Sanderson, A. (1998). *A Modular Approach to Reconfigurable Parallel Robotics*. Boston: Kluwer Academic Publishers.

Hosokawa, K., Tsujimori, T., Fujii, T., Kaetsu, H., Asama, H., Kuroda, Y., et al. (1998). Self-organizing collective robots with morphogenesis in a vertical plane. *In Proceedings of IEEE International Conference on Robotics & Automation (ICRA'98)*, (pp. 2858-2863). Leuven, Belgium.

Inou, N., Kobayashi, H., & Koseki, M. (2002). Development of pneumatic cellular robots forming a mechanical structure. *Proceedings of International Conference of Control, Automation, Robotics and Vision*, (pp. 63-68).

Inou, N., Minami, K., & Koskei, M. (2003). Group robots forming a mechanical structure-development of slide motion mechanism and estimation of energy consumption of the structural formation. *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, (pp. 874-879).

Jennifer E. Walter, E. M. (2002). Choosing Good Paths for Fast Distributed Reconfiguration of Hexagonal Metamorphic Robots. *International Conference on Robotics & Automation*, (pp. 102 - 109). Washington.

- Jorgensen, M., Ostergaard, E., & Lund, H. (2004). Modular ATRON: Modules for a self-reconfigurable robot. *In proceedings of IEEE/RSJ International Conference on Intelligent Robots*, (pp. 2068–2073). Sendai, Japan.
- Kamimura, A., Kurokawa, H., Yoshida, E., Murata, S., Tomita, K., & Kokaji, S. (2005). Automatic locomotion design and experiments for a modular robotic system. *IEEE/ASME Transaction on Mechatronics* , 10 (3), 314-325.
- Kamimura, A., Kurokawa, H., Yoshida, E., Tomita, K., Kokaji, S., & Murata, S. (2004). Distributed Adaptive Locomotion by a Modular Robotic System, M-TRAN II From Local Adaptation to Global Coordinated Motion Using CPG Controllers. *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (pp. 2370-2377). Sendai, Japan.
- Kamimura, A., Kurokawa, H., Yoshida, E., Tomita, K., Murata, S., & Kolaji, S. (2003). Automatic Locomotion Pattern Generation for Modular Robots. *Proceedings of the 2003 IEEE International Conference on Robotics & Automation*, (pp. 714-720). Taipei, Taiwan.
- Khoshnevis, B., Kovac, B., Shen, W. M., & Will, P. (2001). Reconnectable joints for self-reconfigurable robots. *Proceedings, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'01)*, 1, pp. 584-589. Hawaii, USA.
- Klavins, E., Burden, S., & Napp, N. (2006). Optimal rules for programmed stochastic self-assembly. *Proceedings of Robotics Science Systems '06*.
- Klavins, E., Ghrist, R., & Lipsky, D. (2006). A Grammatical Approach to Self-Organizing Robotic Systems. *IEEE Transactions on Automatic Control* , 51 (6), 949-962.
- Kocay, W., & Kreher, D. L. (2005). *Graphs, Algorithms, and Optimization*. Florida: Chapman & Hall/CRC Press.
- Kotay, K. (2003). *Self-Reconfiguring Robots: Designs, Algorithms, and Applications*, Ph.D Thesis. Dartmouth College, Computer Science Department.

- Kotay, K., & Rus, D. (2000). Algorithms for selfreconfiguring molecule motion planning. *Proceedings of the International Conference on Intelligent Robots and Systems*, (pp. 2184-2193).
- Kotay, K., & Rus, D. (2005). Efficient Locomotion for a Self-Reconfiguring Robot. *Proceedings of the IEEE International Conference on Robotics and Automation*, (pp. 2963-2969).
- Kotay, K., & Rus, D. L. (1998). Motion Synthesis for the Self-Reconfiguring Robotic Molecule. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2, pp. 843-851.
- Kotay, K., & Rus, D. (1999). Locomotion versatility through selfreconfiguration. *Robotics and Autonomous Systems* , 26, 217-232.
- Kotay, K., & Rus, D. (2000). Scalable parallel algorithm for configuration planning for self-reconfiguring robots. *In Proceedings of the Society of Photo-Optical Instrumentation Engineers*. Boston.
- Kotay, K., Rus, D., Vona, M., & McGray, C. (1998). The self-reconfiguring robotic molecule. *In Proceedings of the IEEE International Conference on Robotics and Automation*, (pp. 424-431). Leuven, Belgium.
- Kotay, K., Rus, D., Vona, M., & McGray, C. (1998). The selfreconfiguring robotic Molecule: design and control algorithms. *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*. Houston,.
- Kubica, J., Casal, A., & Hogg, T. (2001). Complex behaviors from local rules in modular self-reconfigurable robots. *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, (pp. 360-367).
- Kurokawa, H., Kamimura, A., Yoshida, E., Tomita, K., Kokaji, S., & Murata, S. (2003). M-TRAN II: Metamorphosis from a Four-Legged Walker to a Caterpillar. *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, (pp. 2454-2459). Las Vegas, Nevada.

- Kurokawa, H., Murata, S., Yoshida, E., Tomita, K., & Kokaji, S. (2000). A three-dimensional self-reconfigurable system. *Adv. Robot* , 13 (6), 591-602.
- Lee, W. H., & Sanderson, A. (2001). Dynamic analysis and distributed control of the tetrabot modular reconfigurable robot system. *Autonomous Robots* , 10 (1), 67-82.
- Murata, S., & Haruhisa, K. (2007). Self-Reconfigurable Robots: Shape-Changing Cellular Robots Can Exceed Conventional Robot Flexibility. *IEEE Robotics & Automation Magazine* , 71-78.
- Murata, S., Kakomura, K., & Kurokawa, H. (2006). Docking experiments of a modular robot by visual feedback. *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 625-630). Los Alamitos, CA: IEEE Computer Society Press.
- Murata, S., Kamimura, A., Kurokawa, H., Yoshida, E., Tomita, K., & Kokaji, S. (2004). Self-Reconfigurable Robots: Platforms for Emerging Functionality. *Embodied Artificial Intelligence* , 312-330.
- Murata, S., Kurokawa, H., & Kokaji, S. (1994). Self-assembling machine. *IEEE International Conference on Robotics & Automation*, (pp. 441-448). San Diego.
- Murata, S., Kurokawa, H., Yoshida, E., Tomita, K., & Kokaji, S. (1998). A 3-D Self-Reconfigurable Structure. *IEEE International Conference on Robotics & Automation*, (pp. 432-439). Leuven.
- Murata, S., Yoshida, E., Kamimura, A., Kurokawa, H., Tomita, K., & Kokaji, S. (2002). M-TRAN: Selfreconfigurable modular robotic system. *IEEE/ASME Transactions on Mechatronics* , 7 (4), 431-441.
- Murata, S., Yoshida, E., Kurokawa, H., Tomita, K., & Kokaji, S. (2001). Concept of self-reconfigurable modular robotic system. *J. AI Eng* , 15 (4), 383-387.
- Murata, S., Yoshida, E., Kurokawa, H., Tomita, K., & Kokaji, S. (2001). Selfrepairing mechanical system. *Autonomous Robots* , 7-21.

- Murata, S., Yoshida, E., Tomita, K., Kurokawa, H., Kamimura, A., & Kokaji, S. (2000). Hardware design of modular robotic system. *Proceedings, IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'00)*, (pp. 2210-2217). Takamatsu, Japan.
- Mytilinaios, E., Desnoyer, M., Marcus, D., & Lipson, H. (2004). Designed and evolved blueprints for physical self-replicating machines. *Proceedings of the 9th International Conference on the Simulation and Synthesis of Living Systems (Artificial Life IX)* (pp. 15-20). Cambridge, MA: MIT Press.
- Neumann, J. V. (1966). *Theory of Self-Reproducing Automata*. Urbana: University of Illinois Press.
- Nguyen, A., Guibas, L., & Yim, M. (2001). Controlled module density helps reconfiguration planning. *Proceedings of the 4th International Workshop on Algorithmic Foundations of Robotics, New Directions in Algorithmic and Computational Robotics* (pp. 23-36). Hanover: A.K. Peters.
- Pamecha, A., Chiang, C. J., Stein, D., & Chirikjian, G. (1996). Design and implementation of metamorphic robots. *Design Engineering Technical Conf. and Computers in Engineering Conference*.
- Pamecha, A., Ebert-Uphoff, I., & Chirikjian, G. (1997). Useful metrics for modular robot motion planning. *IEEE Transactions on Robotics and Automation*, 13 (4), 531-545.
- Payne, K., Salemi, B., Will, P., & Shen, W. M. (2004). Sensor-based distributed control for chain-typed self-reconfiguration. *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2, pp. 2074-2080. Los Alamitos, CA: IEEE Computer Society Press.
- Prevas, K. C., Unsal, C., Efe, M. O., & Khosla, P. K. (2002). A Hierarchical Motion Planning Strategy for a Uniform Self-Reconfigurable Modular Robotic System. *Proceedings of the 2002 IEEE International Conference on Robotics & Automation*, (pp. 787-792). Washington, DC.

Rosa, M. D., Goldstein, S., Lee, P., Campbell, J., & Pillai, P. (2006). Scalable shape sculpting via hole motion: Motion planning in lattice-constrained modular robots. *Proceedings of IEEE International Conference on Robotics and Automation*.

Rubenstein, M., Payne, K., Will, P., & Shen, W. M. (2004). Docking among independent and autonomous CONRO self-reconfigurable robots. *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, 3, pp. 2877-2882. Los Alamitos, CA: IEEE Computer Society Press.

Rus, D., & Vona, M. (2001). Crystalline robots: Self-configuration with compressible unit modules. *Autonomous Robots*, 10 (1), 107-124.

Rus, D., & Vona, M. (1999). Self-reconfiguration planning with compressible unit modules. *Proceedings of IEEE International Conference Robotics and Automation*, (pp. 2513-2530).

Russell, S. J., & Norving, P. (2003). *Artificial Intelligence A Modern Approach*. New Jersey: Prentice Hall.

Salemi, B., Moll, M., & Shen, W. M. (2006). SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system. *Proceedings of 2006 IEEE/RSJ International Conference of Intelligent Robots Systems*, (pp. 3636-3641).

Salemi, B., Shen, W. M., & Will, P. (2004). Hormone-controlled metamorphic robots. *In Proceedings of IEEE ICRA*, (pp. 4194-4199).

Shen, W. M., & Will, P. (2001). Docking in Self-Reconfigurable Robots. *Proceedings of IEEE International Conference on Intelligent Robots and Systems*. Maui, USA.

Shen, W. M., Krivokon, M., Chiu, H., Everist, J., Rubenstein, M., & Venkatesh, J. (2006). Multimode locomotion via SuperBot reconfigurable robots. *Autonomous Robots*, 20, 165-177.

Shen, W. M., Will, P., & Castano, A. (1999). Robot modularity for self-reconfiguration. *In SPIE Conference on Sensor Fusion and Decentralized Control in Robotic Systems*.

- Shen, W., Salemi, B., & Will, P. (2002). Hormone-inspired adaptive communication and distributed control for CONRO self-reconfigurable robots. *IEEE Transaction on Robot. and Automat.* , 18, 700-712.
- Stoy, K., Shen, W. M., & Will, P. (2002). Global Locomotion from Local Interaction in Self-Reconfigurable Robots. *Proceedings of Intelligent Autonomous Systems*.
- Stoy, K., Shen, W., & Will, P. (2002). Using role based control to produce locomotion in chain-type self-reconfigurable robot. *IEEE/ASME Transaction on Mechatronics* , 7, 410-417.
- Suh, J., Homans, S., & Yim, M. (2002). Telecubes: Mechanical Design of a Module for Self-Reconfigurable Robotics. *Proceedings, IEEE International Conference on Robotics and Automation (ICRA'02)*, 4, pp. 4095-4101. Washington, DC, USA.
- Sutton, R., & Barto, A. (1998). *Reinforcement Learning: An Introduction*. Cambridge: MIT Press.
- Tomita, K., Murata, S., Kurokawa, H., Yoshida, E., & Kokaji, S. (1999). A selfassembly and self-repair method for a distributed mechanical system. *IEEE Transaction on Robotics and Automation* , 1035-1045.
- Unsal, C., & Khosla, P. K. (2001). A Multi-Layered Planner for Self-Reconfiguration of a Uniform Group of I-Cube. *Proceedings of 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Unsal, C., & Khosla, P. (2000). Mechatronic design of a modular self-reconfiguring robotic system. *Proceedings, IEEE International Conference on Robotics & Automation (ICRA'00)*, (pp. 1742-1747). San Francisco, USA.
- Unsal, C., & Khosla, P. (2000). Solutions for 3-d self-reconfiguration in a modular robotic system: Implementation and motion planning. *Proceedings, SPIE Sensor Fusion and Decentralized Control in Robotic Systems III*, (pp. 388-401).

- Unsal, C., Kiliccote, H., & Khosla, P. K. (1999). I(CES)-cubes: a Modular Self-Reconfigurable Bipartite Robotic System. *Proceedins SPIE, Sensor Fusion and Decentralized Control in Robotic Systems II*, (pp. 246-257).
- Varshavskaya, P. (2007). *Phd Thesis: Distributed Reinforcement Learning for Self-Reconfiguring Modular Robots*. Massachusetts: MIT University.
- Vassilvitskii, S., Kubica, J., & Rieffel, E. (2002). On the General Reconfiguratio Problem for Expanding Cube Style Modular Robots. *Proceedings of the 2002 IEEE International Conference on Robotics & Automation*, (pp. 801-808). Washington, DC.
- Vassilvitskii, S., Yim, M., & Suh, J. (2002). A complete, local and parallel reconfiguration algorithm for cube style modular. *In Proceedings of IEEE ICRA*, (pp. 117-122).
- Walter, J. E., Tsai, E. M., & Amato, N. M. (2005). Algorithms for Fast Concurrent Reconfiguration of Hexagonal Metamorphic Robots. *IEEE Transactions of Robotics* , 21 (4), 621-631.
- Walter, J. E., Tsai, E. M., & Amato, N. M. (2002). Choosing good paths for fast distributed reconfiguration of hexagonal metamorphic robots. *Proceedings of IEEE ICRA*, (pp. 102-109).
- Walter, J. E., Welch, J. L., & Amato, N. M. (2002). Concurrent metamorphosis of hexagonal robot chains into simple connected configurations. *IEEE Transactions on Robotics and Automation* , 18 (6), 945-956.
- Walter, J. E., Welch, J. L., & Amato, N. M. (2000). Distributed Reconfiguration of Hexagonal Metamorphic Robots in Two Dimensions. *Proc. SPIE, Sensor Fusion and Decentralized Control in Robotic Systems III*, (pp. 441-453).
- Walter, J., Welch, J., & Amato, N. (2000). Distributed reconfiguration of metamorphic robot chains. *Proceedins of ACM Symposium Principles of Distributed Computing*, (pp. 171-180). Portland, OR.

- White, P. J., & Yim, M. (2007). Scalable Modular Self-reconfigurable Robots Using External Actuation. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, (pp. 2773-2778). San Diego, CA, USA.
- White, P. J., Kopanski, K., & Lipson, H. (2004). Stochastic Self-Reconfigurable Cellular Robotics. *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, (pp. 2888-2893). Now Weans. LA.
- White, P., Zykov, V., Bongard, J., & Lipson, H. (2005). Three Dimensional Stochastic Reconfiguration of Modular Robots. *Robotics: Science and Systems* , 161-168.
- Yim, M. (1993). A reconfigurable modular robot with multiple modes of locomotion. *Proceedings of JSME International Conference of Advanced Mechatronics*, (pp. 283-288). Tokyo, Japan.
- Yim, M. (1994). *Locomotion with a unit-modular reconfigurable robot. PhD thesis.* Department of Mechanical Engineering, Stanford University.
- Yim, M. (1994). New locomotion gaits. *Proceedings, International Conference on Robotics & Automation (ICRA '94)*, (pp. 2508-2514). San Diego, California, USA.
- Yim, M., Duff, D. G., & Roufas, K. D. (2002). Walk on the wild side. *IEEE Robot. Automat.* , 9 (4), 49-53.
- Yim, M., Duff, D., & Roufas, K. (2000). Polybot: A modular reconfigurable robot. *IEEE International Conference on Robotics & Automation*, (pp. 514-520). San Francisco.
- Yim, M., Goldberg, D., & Casal, A. (2000). Connectivity Planning for Closed-Chain Reconfiguration. *Proceedings SPIE, Sensor Fusion and Decentralized Control in Robotic Systems III*, (pp. 402-412).
- Yim, M., Lamping, J., Mao, E., & Chase, J. G. (1997). *Rhombic dodecahedron shape for self-assembling robots.* Xerox PARC SPL, Tech. Rep.
- Yim, M., Roufas, K., Duff, D., Zhang, Y., Eldershaw, C., & Homans, S. B. (2003). Modular reconfigurable robots in space applications. *Auton. Robots* , 14 (2-3), 225-327.

- Yim, M., Shen, W.-M., Salemi, B., Rus, D., Moll, M., Lipson, H., et al. (2007). Modular Self-Reconfigurable Robot Systems - Challenges and Opportunities for the Future. *IEEE Robotics & Automation Magazine* , 43-52.
- Yim, M., Zhang, Y., & Duff, D. G. (2002). Modular Robots. *IEEE Spectr.* , 39 (2), 30-34.
- Yim, M., Zhang, Y., Lamping, J., & Mao, E. (2001). Distributed control for 3D metamorphosis. *Auton. Robot.* , 10 (1), 41-56.
- Yim, M., Zhang, Y., Roufas, K., Duff, D., & Eldershaw, C. (2002). Connecting and disconnecting for chain self-reconfiguration with Polybot. *Proceedings of IEEE/ASME Transaction of Mechatronics*, 7, pp. 442-451.
- Yoshida, E., Kokaji, S., & Murata, S. (2000). Miniaturization of Self-Reconfigurable Robotic System using Shape Memory Alloy. *Robotics and Mechatronics* , 1579-1585.
- Yoshida, E., Kurokawa, H., Kamimura, A., Tomita, K., Kokaji, S., & Murata, S. (2004). Planning Behaviors of a Modular Robot: an Approach Applying a Randomized Planner to Coherent Structure. *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (pp. 2056-2061). Sendai, Japan.
- Yoshida, E., Murata, S., Kamimura, A., Tomita, K., Kurokawa, H., & Kokaji, S. (2001). A Motion Planning Method for a Self-Reconfigurable Modular Robot. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2001)*.
- Yoshida, E., Murata, S., Kamimura, A., Tomita, K., Kurokawa, H., & Kokaji, S. (2003). A self-reconfigurable modular robot: reconfiguration planning and experiments. *Int. J. Robot. Res.* , 21 (10), 903-916.
- Yoshida, E., Murata, S., Kamimura, A., Tomita, K., Kurokawa, H., & Kokaji, S. (2003). Evolutionary Synthesis of Dynamic Motion and Reconfiguration Process for a Modular Robot M-TRAN. *Proceedings 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, (pp. 1004-1010). Kobe, Japan.

- Yoshida, E., Murata, S., Kaminura, A., Tomita, K., Korokawa, H., & Kokaji, S. (2000). Motion Planning of Selfreconfigurable Modular Robot. *Proceedings of the 7th International Symposium on Experimental Robotics*, (pp. 375-384).
- Yoshida, E., Murata, S., Kokaji, S., Kamimura, A., Tomita, K., & Kurokawa, H. (2002). Get back in shape! a hardware prototype self-reconfigurable modular. *IEEE Robotics & Automation Magazine* , 9 (4), 54-60.
- Yoshida, E., Murata, S., Kurokawa, H., Tomita, K., & Kokaji, S. (1998). A distributed reconfiguration method for 3D homogeneous structure. *Proceedings of IEEE International Conference on Intelligent Robots and Systems*, (pp. 852-859). Victoria, BC, Canada.
- Yoshida, E., Murata, S., Tomita, K., Kurokawa, H., & Kokaji, S. (1998). Experiment of self-repairing modular machine. *Distributed Autonomous Robotic Systems 3* , 119-128.
- Zhang, Y., Roufas, K., Eldershaw, C., Yim, M., & Duff, D. (2003). Sensor computations in modular self reconfigurable robots. *In Proceedings of the 8th International Symposium on Experimental Robotics, volume 5 of Springer Tracts in Advanced Robotics* (pp. 276-286). Berlin, Germany: Springer.
- Zykov, V., Mytilinaios, E., Adams, B., & Lipson, H. (2005). Self-reproducing machines. *Nature* , 163-164.
- Zykov, V., Mytilinaios, E., Desnoyer, M., & Lipson, H. (2007). Evolved and Designed Self-Reproducing Modular Robotics. *IEEE Transactions on Robotics* , 23 (2), 308-319.

APPENDIX A

Microcontroller Codes

These codes are developed for the HexBot modules based on two ATmega162 microcontrollers. The first code is written for the control board including 16 functions for actuators. The second code is developed for the communication board allowing inter-modules communication and central communication with the computer including 30 general purpose functions. The final code is developed for the wireless board based on SPI communication which is part of the communication board that is planned to be integrated in the next stage of the project.

Control Board Program

```
//***** MAIN PROGRAM FOR CONTROL BOARDS *****
//ICC-AVR application builder : 1/1/2009 12:48:09 AM
// Target : M162
// Crystal: 1.0000Mhz
#include <iom162v.h>
#include <macros.h>
int ts; //time step, increased every 10ms

//***** PORT INITIALIZATION *****
void port_init(void)
{
  PORTA = 0x00;
  PORTB = 0x00;
  PORTC = 0x00;
  PORTD = 0x00;
  PORTE = 0x00;
  DDRA = 0b11111111; //Joint Actuators
  DDRB = 0b1111011; //Side Actuators - UART1
  DDRC = 0b00000000; //DIP Switches
  DDRD = 0b1111110; //Communication Board UART0 - Side Actuators
  DDRE = 0b1111111; //Currently Not Used
}

//***** TIMER INITIALIZATION *****
//TIMER0 initialize - prescale:64
// WGM: Normal
// desired value: 10mSec
// actual value: 9.984mSec (0.2%)
void timer0_init(void)
{
  TCCR0= 0x00; //stop
  TCNT0= 0x64; //set count
  OCR0= 0x9C; //set compare value
  TCCR0= 0x03; //start timer
}
#pragma interrupt_handler timer0_ovf_isr:iv_TIMO_OVF
void timer0_ovf_isr(void)
{
  TCNT0= 0x64; //reload counter value
  ts++; //increase ts every 10ms
}
//Delay of multiple of ts (10ms)
void DelayXts (int i)
{
  ts=0;while (ts<i) {}
}

//***** UART INITIALIZATION *****
//UART0 initialize
```

```

// desired baud rate: 4800
// actual: baud rate:4808 (0.2%)
// char size: 8 bit
// parity: Disabled
void uart0_init(void)
{
    UCSROB = 0x00;           //disable while setting baud rate
    UCSROA = 0x00;           //disable while setting baud rate
    UBRR0L =0x0C;           //set baud rate
    UBRR0H = 0x00;
    UCSROC = BIT(URSEL0) | 0x06;
    UCSROA = 0x00;           //enable
    UCSROB = 0x18;           //enable
}
// Transmite0
void TransmitByte0( unsigned char data )
{while ( !(UCSR0A & (1<<UDRE0)) );UDR0 = data;}
// Receive0
unsigned char ReceiveByte0(void)
{while ( !(UCSR0A & (1<<RXC0)) ) {} return UDR0;}
// Receive0TL :time limited listening to the port (t: time stpes)
unsigned char ReceiveByte0TL(int t)
{
    ts=0;
    while (!(UCSR0A&(1<<RXC0))) && (ts<t){};
    if (ts>=t) {return '~';} else {return UDR0;} //~: time exceeded
}
//UART1 initialize
// desired baud rate:4800
// actual baud rate:4808 (0.2%)
// char size: 8 bit
// parity: Disabled
void uart1_init(void)
{
    UCSR1B = 0x00;           //disable while setting baud rate
    UCSR1A = 0x00;           //disable while setting baud rate
    UBRR1L =0x0C;           //set baud rate
    UBRR1H = 0x00;
    UCSR1C = BIT(URSEL1) | 0x06;
    UCSR1A = 0x00;           //enable
    UCSR1B = 0x18;           //enable
}
// Transmit1
void TransmitBytel( unsigned char data )
{while ( !(UCSR1A & (1<<UDRE1)) );UDR1 = data;DelayXts(2);} //Keep a delay after each
transmission
// Receive1
unsigned char ReceiveBytel(void)
{while ( !(UCSR1A & (1<<RXC1)) ) {} return UDR1;}
// Preamble
void PRE (void)
{
    unsigned char txt[20]={10,13,10,13,'#'};
    int i=0;
    while (txt[i]!='#') {TransmitBytel(txt[i]);i++;}
}
// Control Board Ready
void CTR_Ready (void)
{
    unsigned char txt[20]={'C','T','R','.',' ','B','O','A','R','D',' ','R','E','A','D','Y','#'};
    int i=0;
    while (txt[i]!='#') {TransmitBytel(txt[i]);i++;}
    TransmitBytel(10);TransmitBytel(13);
}
// Communication Board Ready
void COMM_Ready (void)
{
    unsigned char txt[20]={'C','O','M','M','.',' ','B','O','A','R','D',' ','R','E','A','D','Y','#'};
}

```

```

    int i=0;
    while (txt[i]!='#') {TransmitByte1(txt[i]);i++;}
    TransmitByte1(10);TransmitByte1(13);
}
// Control Board Listening
void CTR_LST (void)
{
    unsigned char txt[20]={'C','T','R','. ','',
    ',','L','I','S','T','E','N','I','N','G',10,13,'#'};
    int i=0;
    while (txt[i]!='#') {TransmitByte1(txt[i]);i++;}
}
// Communication Board Listening
void COMM_LST (void)
{
    unsigned char txt[20]={'C','O','M','M','. ','',
    ',','L','I','S','T','E','N','I','N','G',10,13,'#'};
    int i=0;
    while (txt[i]!='#') {TransmitByte1(txt[i]);i++;}
}
// Package Received
void RCV (void)
{
    unsigned char txt[20]={'R','E','C','E','I','V','E','D',10,13,'#'};
    int i=0;
    while (txt[i]!='#') {TransmitByte1(txt[i]);i++;}
}

//***** ALL DEVICES INITIALIZATION *****
//call this routine to initialize all peripherals
void init_devices(void)
{
    //stop errant interrupts until set up
    CLI(); //disable all interrupts
    port_init();
    timer0_init();
    uart0_init();
    uart1_init();

    MCUCR= 0x00;
    EMCUCR = 0x00;
    TIMSK= 0x02; //timer interrupt sources
    ETIMSK=0x00;
    GICR= 0x00;
    PCMSK0=0x00;
    PCMSK1=0x00;
    SEI(); //re-enable interrupts
    //all peripherals are now initialized
}

//***** JOINT ACTUATION FUNCTIONS *****
//Joint Actuation Functions (without direction - only on/off)
void JxOpen (int i)
{
    //Place i in the required range
    while (i<1) {i=i+6;}
    while (i>6) {i=i-6;}
    //Actuate accordingly
    if (i==1) {PORTA = PORTA|0b00000100;}
    if (i==2) {PORTA = PORTA|0b00001000;}
    if (i==3) {PORTA = PORTA|0b00100000;}
    if (i==4) {PORTA = PORTA|0b00010000;}
    if (i==5) {PORTA = PORTA|0b00000001;}
    if (i==6) {PORTA = PORTA|0b00000010;}
}
void JxClose (int i)
{
    //Place i in the required range
    while (i<1) {i=i+6;}

```

```

while (i>6) {i=i-6;}
//Actuate accordingly
if (i==1) {PORTA = PORTA&0b11111011;}
if (i==2) {PORTA = PORTA&0b11110111;}
if (i==3) {PORTA = PORTA&0b11011111;}
if (i==4) {PORTA = PORTA&0b11101111;}
if (i==5) {PORTA = PORTA&0b11111110;}
if (i==6) {PORTA = PORTA&0b11111101;}
}
void JxOpenAll (void)
{
PORTA = PORTA|0b00111111;
}
void JxCloseAll (void)
{
PORTA = PORTA&0b11000000;
}
//Joint Test
void JTest (int n,int t)
{
int i,j;
for (j=0;j<n;j++)
{
for (i=1;i<7;i++) {JxOpen(i);DelayXts(t);JxClose(i);DelayXts(t);}
}
}

//***** SIDE ACTUATION FUNCTIONS *****
//Side Actuation Functions (with direction - on(positive or negative)/off
void SxP (int i)
{
//Place i in the required range
while (i<1) {i=i+6;}
while (i>6) {i=i-6;}
//Actuate accordingly
if (i==1) {PORTB = PORTB|0b00000001;}
if (i==2) {PORTD = PORTD|0b00001000;}
if (i==3) {PORTB = PORTB|0b01000000;}
if (i==4) {PORTD = PORTD|0b00010000;}
if (i==5) {PORTB = PORTB|0b00000010;}
if (i==6) {PORTB = PORTB|0b10000000;}
}
void SxN (int i)
{
//Place i in the required range
while (i<1) {i=i+6;}
while (i>6) {i=i-6;}
//Actuate accordingly
if (i==1) {PORTD = PORTD|0b01000000;}
if (i==2) {PORTD = PORTD|0b00100000;}
if (i==3) {PORTD = PORTD|0b10000000;}
if (i==4) {PORTB = PORTB|0b00010000;}
if (i==5) {PORTB = PORTB|0b00100000;}
if (i==6) {PORTD = PORTD|0b00000100;}
}
void SxOff (int i)
{
//Place i in the required range
while (i<1) {i=i+6;}
while (i>6) {i=i-6;}
//Actuate accordingly
if (i==1) {PORTB = PORTB&0b11111110;PORTD = PORTD&0b10111111;}
if (i==2) {PORTD = PORTD&0b11110111;PORTD = PORTD&0b11011111;}
if (i==3) {PORTB = PORTB&0b10111111;PORTD = PORTD&0b01111111;}
if (i==4) {PORTD = PORTD&0b11101111;PORTB = PORTB&0b11101111;}
if (i==5) {PORTB = PORTB&0b11111011;PORTB = PORTB&0b11011111;}
if (i==6) {PORTB = PORTB&0b01111111;PORTD = PORTD&0b11111011;}
}
void SxOffAll (void)
{

```



```

    SxOffAll();          //8 - All Sx Off
}

//***** FUNCTIONS CALLED FROM COMMUNICATION BOARD *****
//Listening to the Communication Board for Actuation
void Listen (void)
{
    unsigned char R;      //Received Character
    int MSG[6];          //Received Message
    int i;

    while (ReceiveByte0() != '*') {;}          //Wait for the start of message
    for (i=1;i<6;i++) {MSG[i] = ReceiveByte0();} //Read the 5 bytes of the message

    //Confirm receiving instruction from communication board - Comm
    TransmitByte0(MSG[1]);
    //Command Received PC
    PRE();          //Preamble - PC
    // RCV();          //Received

    //Acknowledgement (with the command code going to be executed) - PC
    TransmitBytel('A');
    TransmitBytel('C');
    TransmitBytel('K');
    TransmitBytel(MSG[1]);
    TransmitBytel(10);
    TransmitBytel(13);
    //Message Received - PC
    TransmitBytel('M');

    TransmitBytel('S');
    TransmitBytel('G');
    TransmitBytel(MSG[1]);
    TransmitBytel(MSG[2]);
    TransmitBytel(MSG[3]);
    TransmitBytel(MSG[4]);
    TransmitBytel(MSG[5]);
    TransmitBytel(10);
    TransmitBytel(13);

    //Run the required funcaion based on "Control Board Functions Table"

    if (MSG[1]==1) {JxOpen(MSG[2]);}
    if (MSG[1]==2) {JxClose(MSG[2]);}
    if (MSG[1]==3) {JxOpenAll();}
    if (MSG[1]==4) {JxCloseAll();}
    if (MSG[1]==5) {JTest(MSG[2],MSG[3]);}

    if (MSG[1]==11) {SxP(MSG[2]);}
    if (MSG[1]==12) {SxN(MSG[2]);}
    if (MSG[1]==13) {SxOff(MSG[2]);}
    if (MSG[1]==14) {SxOffAll();}
    if (MSG[1]==15) {SPTest(MSG[2],MSG[3]);}
    if (MSG[1]==16) {SNTTest(MSG[2],MSG[3]);}

    if (MSG[1]==20) {TurnOffAll();}

    if (MSG[1]==21) {MCW(MSG[2],MSG[3],MSG[4],MSG[5]);}
    if (MSG[1]==22) {MCCW(MSG[2],MSG[3],MSG[4],MSG[5]);}
    if (MSG[1]==23) {SCW(MSG[2],MSG[3],MSG[4],MSG[5]);}
    if (MSG[1]==24) {SCCW(MSG[2],MSG[3],MSG[4],MSG[5]);}
}

//***** MAIN PROGRAM *****
void main(void)
{
    init_devices();
    DelayXts(10);          //wait for voltages to sta-
bilize
}

```

```

TurnOffAll();
JTest(1,5); //Joint test (indicating
ready)
PRE(); //Preamble - PC
CTR_Ready(); //Control Board Ready - PC
TransmitByte0('R'); //Control Board Ready - Com-
munication
while (ReceiveByte0TL(10)!='R') {TransmitByte0('R');} //Wait for the communication
board ready signal
COMM_Ready(); //Control Board Ready - PC
TransmitByte0('L'); //Listening
CTR_LST(); //Control Board Listening -
PC
while (1) {Listen();} //Execute the required func-
tion
}

```

Communication Board Program

```

//***** MAIN PROGRAM FOR COMMUNICATION BOARDS
*****
//ICC-AVR application builder : 1/1/2009 12:48:09 AM
// Target : M162
// Crystal: 1.0000Mhz
#include <iom162v.h>
#include <macros.h>
int ts; //time step, increased every 10ms
int M=1;S=0; //Assuming the module is Mobile not Substrate
int L=0; //Assuming the module is not initially localized
int X=0,Y=0; //Assuming the initial location of the module
int Yref=0; //Rotation Transformation (Difference between the body and
global frame)

//***** PORT INITIALIZATION *****
void port_init(void)
{
PORTA = 0x00;
PORTB = 0x00;
PORTC = 0x00;
PORTD = 0x00;
PORTE = 0x00;
DDRA = 0b11111111; //Selectors - Side LEDs
DDRB = 0b11111011; //IR Communication
DDRC = 0b11111111; //Side LEDs - Joint LEDs
 DDRD = 0b11111110; //Control Communication - Side LEDs - Joint LEDs
 DDRE = 0b11111111; //Side LEDs - Joint LEDs
}

//***** TIMER INITIALIZATION *****
//TIMER0 initialize - prescale:64
// WGM: Normal
// desired value: 10mSec
// actual value: 9.984mSec (0.2%)
void timer0_init(void)
{
TCCR0= 0x00; //stop
TCNT0= 0x64; //set count
OCR0= 0x9C; //set compare value
TCCR0= 0x03; //start timer
}
#pragma interrupt_handler timer0_ovf_isr:iv_TIM0_OVF
void timer0_ovf_isr(void)
{
TCNT0= 0x64; //reload counter value
ts++; //increase ts every 10ms
}
//Delay of multiple of ts (10ms)
void DelayXts (int i)

```

```

{
    ts=0;while (ts<i) {;}
}

//***** UART INITIALIZATION *****
//UART0 initialize
// desired baud rate: 4800
// actual: baud rate:4808 (0.2%)
// char size: 8 bit
// parity: Disabled
void uart0_init(void)
{
    UCSROB = 0x00;           //disable while setting baud rate
    UCSROA = 0x00;           //disable while setting baud rate
    UBRR0L =0x0C;            //set baud rate
    UBRR0H = 0x00;
    UCSROC = BIT(URSEL0) | 0x06;
    UCSROA = 0x00;           //enable
    UCSROB = 0x18;           //enable
}
// Transmit0
void TransmitByte0( unsigned char data )
{while ( !(UCSR0A & (1<<UDRE0)) );UDR0 = data;}
// Receive0
unsigned char ReceiveByte0(void)
{while ( !(UCSR0A & (1<<RXC0)) ) {;} return UDR0;}
// Receive0TL :time limited listening to the port (t: time stpes)
unsigned char ReceiveByte0TL(int t)
{
    ts=0;
    while (!(UCSR0A&(1<<RXC0)) && (ts<t)){;}
    if (ts>=t) {return '~';} else {return UDR0;} //'~': time exceeded
}
// Contorl Message Format
int MSG( int a,int b,int c,int d,int e)
{
    TransmitByte0('*');
    TransmitByte0(a);
    TransmitByte0(b);
    TransmitByte0(c);
    TransmitByte0(d);
    TransmitByte0(e);
    TransmitByte0('#');
    return ReceiveByte0TL(250); //Wait for acknowledgment (max of 2.5sec)
}
//UART1 initialize
// desired baud rate:300
// actual baud rate:300 (0.2%)
// char size: 8 bit
// parity: Disabled
void uart1_init(void)
{
    UCSR1B = 0x00;           //disable while setting baud rate
    UCSR1A = 0x00;           //disable while setting baud rate
    UBRR1L =0xCF;            //set baud rate
    UBRR1H = 0x00;
    UCSR1C = BIT(URSEL1) | 0x06;
    UCSR1A = 0x00;           //enable
    UCSR1B = 0x18;           //enable
}
// Transmit1
void TransmitByte1( unsigned char data )
{while ( !(UCSR1A & (1<<UDRE1)) );UDR1 = data;DelayXts(5);} //Keep a delay after
each transmission
// Receive1
unsigned char ReceiveByte1(void)
{while ( !(UCSR1A & (1<<RXC1)) ) {;} return UDR1;}
// Receive1TL :time limited listening to the port (t: time steps)
unsigned char ReceiveByte1TL(int t)
{

```

```

ts=0;
while (!(UCSR1A&(1<<RXCL)) && (ts<t)){;
if (ts>=t) {return '~';} else {return UDR1;} // '~': time exceeded
}
// Localization Message Format: +xx+yyS
void LOCMSG1( int a,int b,int c,int d,int e, int f, int g)
{
TransmitBytel('*');
TransmitBytel(a);
TransmitBytel(b);
TransmitBytel(c);
TransmitBytel(d);
TransmitBytel(e);
TransmitBytel(f);
TransmitBytel(g);
}
// Preamble
void PRE (void)
{
unsigned char txt[20]={10,13,10,13,'#'};
int i=0;
while (txt[i]!='#') {TransmitBytel(txt[i]);i++;}
if (S==1) {TransmitBytel('S');TransmitBytel(':');TransmitBytel(' ');}
if (M==1) {TransmitBytel('M');TransmitBytel(':');TransmitBytel(' ');}
}
// Control Board Ready
void CTR_Ready (void)
{
unsigned char txt[20]={'C','T','R','. ',' ','B','O','A','R','D',' ','R','E','A','D','Y','#'};
int i=0;
while (txt[i]!='#') {TransmitBytel(txt[i]);i++;}
TransmitBytel(10);TransmitBytel(13);
}
// Communication Board Ready
void COMM_Ready (void)
{
unsigned char txt[20]={'C','O','M','M','. ',' ','B','O','A','R','D',' ','R','E','A','D','Y','#'};
int i=0;
while (txt[i]!='#') {TransmitBytel(txt[i]);i++;}
TransmitBytel(10);TransmitBytel(13);
}
// Control Board Listening
void CTR_LST (void)
{
unsigned char txt[20]={'C','T','R','. ','L','I','S','T','E','N','I','N','G',10,13,'#'};
int i=0;
while (txt[i]!='#') {TransmitBytel(txt[i]);i++;}
}
// Communication Board Listening
void COMM_LST (void)
{
unsigned char txt[20]={'C','O','M','M','. ','L','I','S','T','E','N','I','N','G',10,13,'#'};
int i=0;
while (txt[i]!='#') {TransmitBytel(txt[i]);i++;}
}
// Localized
void LOC (void)
{
unsigned char txt[20]={'L','O','C','A','L','I','Z','E','D',10,13,'#'};
int i=0;
while (txt[i]!='#') {TransmitBytel(txt[i]);i++;}
}
// Package Received
void RCV (void)
{
unsigned char txt[20]={'R','E','C','E','I','V','E','D',10,13,'#'};

```

```

int i=0;
while (txt[i]!='#') {TransmitByte1(txt[i]);i++;}
}

//***** ALL DEVICES INITIALIZATION *****
//call this routine to initialize all peripherals
void init_devices(void)
{
//stop errant interrupts until set up
CLI(); //disable all interrupts
port_init();
timer0_init();
uart0_init();
uart1_init();

MCUCR= 0x00;
EMCUCR = 0x00;
TIMSK= 0x02; //timer interrupt sources
ETIMSK=0x00;
GICR= 0x00;
PCMSK0=0x00;
PCMSK1=0x00;
SEI(); //re-enable interrupts
//all peripherals are now initialized
}

//***** CONTROL FUNCTIONS *****
int JxOpen (int i) {int a; a=MSG(1,i,'N','N','N');return a;}
int JxClose (int i) {int a; a=MSG(2,i,'N','N','N');return a;}
int JxOpenAll (void) {int a; a=MSG(3,0,'N','N','N');return a;}
int JxCloseAll (void) {int a; a=MSG(4,0,'N','N','N');return a;}
int JTest (int n,int t) {int a; a=MSG(5,n,t,'N','N');return a;}
int SxP (int i) {int a; a=MSG(11,i,'N','N','N');return a;}
int SxN (int i) {int a; a=MSG(12,i,'N','N','N');return a;}
int SxOff (int i) {int a; a=MSG(13,i,'N','N','N');return a;}
int SxOffAll (void) {int a; a=MSG(14,'N','N','N','N');return a;}
int SPTest (int n,int t) {int a; a=MSG(15,n,t,'N','N');return a;}
int SNTTest (int n,int t) {int a; a=MSG(16,n,t,'N','N');return a;}
int TurnOffAll (void) {int a; a=MSG(20,'N','N','N','N');return a;}
int MCW (int i,int t1,int t2, int t3) {int a; a=MSG(21,i,t1,t2,t3);return a;}
int MCCW (int i,int t1,int t2, int t3) {int a; a=MSG(22,i,t1,t2,t3);return a;}
int SCW (int i,int t1,int t2, int t3) {int a; a=MSG(23,i,t1,t2,t3);return a;}
int SCCW (int i,int t1,int t2, int t3) {int a; a=MSG(24,i,t1,t2,t3);return a;}

//***** IR SELECTOR *****
void IrSel (int i)
{
// (MUX, DEMUX Selector (C,B,A)
if (i==1){PORTA = PORTA & 0b11111000;PORTA = PORTA | 0b00000000;} //Y0: SIDE 1
if (i==6){PORTA = PORTA & 0b11111100;PORTA = PORTA | 0b00000100;} //Y1: SIDE 6
if (i==0){PORTA = PORTA & 0b11111010;PORTA = PORTA | 0b00000010;} //Y2: CENTER
if (i==2){PORTA = PORTA & 0b11111110;PORTA = PORTA | 0b00000110;} //Y3: SIDE 2
if (i==5){PORTA = PORTA & 0b11111001;PORTA = PORTA | 0b00000001;} //Y4: SIDE 5
if (i==3){PORTA = PORTA & 0b11111101;PORTA = PORTA | 0b00000101;} //Y5: SIDE 3
if (i==4){PORTA = PORTA & 0b11111011;PORTA = PORTA | 0b00000011;} //Y6: SIDE 4
}
//***** SIDE LEDs *****
void SideOnP (int i)
{
//Place i in the required range
while (i<1) {i=i+6;}
while (i>6) {i=i-6;}
//Turn On LED
if (i==1){PORTA = PORTA & 0b10111111;}
if (i==2){PORTE = PORTE & 0b11111101;}
if (i==3){PORTC = PORTC & 0b10111111;}
if (i==4){PORTC = PORTC & 0b11110111;}
if (i==5){PORTC = PORTC & 0b11111101;}
if (i==6){PORTD = PORTD & 0b10111111;}
}

```

```
}
void SideOffP (int i)
{
    //Place i in the required range
    while (i<1) {i=i+6;}
    while (i>6) {i=i-6;}
    //Turn Off LED
    if (i==1){PORTA = PORTA | 0b01000000;}
    if (i==2){PORTE = PORTE | 0b00000010;}
    if (i==3){PORTC = PORTC | 0b01000000;}
    if (i==4){PORTC = PORTC | 0b00001000;}
    if (i==5){PORTC = PORTC | 0b00000010;}
    if (i==6){PORTD = PORTD | 0b01000000;}
}
void SideOnAllP (void)
{
    //Turn On LED
    PORTA = PORTA & 0b10111111;
    PORTC = PORTC & 0b10110101;
    PORTD = PORTD & 0b10111111;
    PORTE = PORTE & 0b11111101;
}
void SideOffAllP (void)
{
    //Turn Off LED
    PORTA = PORTA | 0b01000000;
    PORTE = PORTE | 0b00000010;
    PORTC = PORTC | 0b01001010;
    PORTD = PORTD | 0b01000000;
}
void SideOnN (int i)
{
    //Place i in the required range
    while (i<1) {i=i+6;}
    while (i>6) {i=i-6;}
    //Turn On LED
    if (i==1){PORTA = PORTA & 0b11011111;}
    if (i==2){PORTE = PORTE & 0b11111110;}
    if (i==3){PORTC = PORTC & 0b01111111;}
    if (i==4){PORTC = PORTC & 0b11101111;}
    if (i==5){PORTC = PORTC & 0b11111110;}
    if (i==6){PORTD = PORTD & 0b11011111;}
}
void SideOffN (int i)
{
    //Place i in the required range
    while (i<1) {i=i+6;}
    while (i>6) {i=i-6;}
    //Turn Off LED
    if (i==1){PORTA = PORTA | 0b00100000;}
    if (i==2){PORTE = PORTE | 0b00000001;}
    if (i==3){PORTC = PORTC | 0b10000000;}
    if (i==4){PORTC = PORTC | 0b00010000;}
    if (i==5){PORTC = PORTC | 0b00000001;}
    if (i==6){PORTD = PORTD | 0b00100000;}
}
void SideOnAllN (void)
{
    //Turn On LED
    PORTA = PORTA & 0b11011111;
    PORTC = PORTC & 0b01101110;
    PORTD = PORTD & 0b11011111;
    PORTE = PORTE & 0b11111110;
}
void SideOffAllN (void)
{
    //Turn Off LED
    PORTA = PORTA | 0b00100000;
    PORTC = PORTC | 0b10010001;
    PORTD = PORTD | 0b00100000;
}
```

```

    PORTE = PORTE | 0b00000001;
}
//Side (Magnet) Test - Positive
void SidePTest (int n,int t)
{
    int i,j;
    for (j=0;j<n;j++)
    { for (i=1;i<7;i++) {SideOnP(i);DelayXts(t);SideOffP(i);DelayXts(t);}}
}
//Side (Magnet) Test - Negative
void SideNTest (int n,int t)
{
    int i,j;
    for (j=0;j<n;j++)
    { for (i=1;i<7;i++) {SideOnN(i);DelayXts(t);SideOffN(i);DelayXts(t);}}
}

//***** JOINT LEDs *****
void JointOn (int i)
{
    //Place i in the required range
    while (i<1) {i=i+6;}
    while (i>6) {i=i-6;}
    //Turn On LED
    if (i==1){PORTD = PORTD & 0b01111111;}
    if (i==2){PORTA = PORTA & 0b01111111;}
    if (i==3){PORTE = PORTE & 0b11111011;}
    if (i==4){PORTC = PORTC & 0b11011111;}
    if (i==5){PORTC = PORTC & 0b11111011;}
    if (i==6){PORTD = PORTD & 0b11101111;}
}
void JointOff (int i)
{
    //Place i in the required range
    while (i<1) {i=i+6;}
    while (i>6) {i=i-6;}
    //Turn Off LED
    if (i==1){PORTD = PORTD | 0b10000000;}
    if (i==2){PORTA = PORTA | 0b10000000;}
    if (i==3){PORTE = PORTE | 0b00000100;}
    if (i==4){PORTC = PORTC | 0b00100000;}
    if (i==5){PORTC = PORTC | 0b00000100;}
    if (i==6){PORTD = PORTD | 0b00010000;}
}
void JointOnAll (void)
{
    //Turn On LED
    PORTA = PORTA & 0b01111111;
    PORTC = PORTC & 0b11011011;
    PORTD = PORTD & 0b01101111;
    PORTE = PORTE & 0b11111011;
}
void JointOffAll (void)
{
    //Turn Off LED
    PORTA = PORTA | 0b10000000;
    PORTC = PORTC | 0b00100100;
    PORTD = PORTD | 0b10010000;
    PORTE = PORTE | 0b00000100;
}

//Joint Test LED
void JointTest (int n,int t)
{
    int i,j;
    for (j=0;j<n;j++)
    {
        for (i=1;i<7;i++) {JointOn(i);DelayXts(t);JointOff(i);DelayXts(t);}
    }
}

```

```

//***** LED FUNCTIONS *****
//LED All On
void LOn (void)
{SideOnAllP();SideOnAllN();JointOnAll();}
//LED All Off
void LOff (void)
{SideOffAllP();SideOffAllN();JointOffAll();}
//LED Test (Circle)
void LCircle (int n,int t)
{
  int i,j;
  for (j=0;j<n;j++)
  {
    for (i=1;i<7;i++)
    {
      JointOn(i);DelayXts(t);JointOff(i);DelayXts(t);
      IrSel(i);TransmitBytel(255);DelayXts(t);TransmitBytel(0);DelayXts(t);
      SideOnN(i);DelayXts(t);SideOffN(i);DelayXts(t);
      SideOnP(i);DelayXts(t);SideOffP(i);DelayXts(t);
    }
  }
  IrSel(0);
}
//LED Test (Color)
void LColor (int n,int t)
{
  int i,j;
  for (j=0;j<n;j++)
  {
    JointOnAll();DelayXts(t);JointOffAll();DelayXts(t);
    SideOnAllN();DelayXts(t);SideOffAllN();DelayXts(t);
    SideOnAllP();DelayXts(t);SideOffAllP();DelayXts(t);
  }
}
//LED Test (Flash)
void LFlash1 (int n,int t)
{
  int i,j;
  for (j=0;j<n;j++)
  {
    SideOnAllN();SideOffAllP();DelayXts(t);
    SideOffAllN();SideOnAllP();DelayXts(t);
  }
  LOff();
}
//LED Test (Flash)
void LFlash2 (int n,int t)
{
  int i,j;
  for (j=0;j<n;j++)
  {
    LOn();DelayXts(t);LOff();DelayXts(t);
  }
}
//Y Ref. (Flash)
void LYref (int i)
{
  if (i==0) {Yref=Yref;} //Same Orientation: just indicate +Y
  if (i==1) {Yref++;} //CCW Rotation
  if (i==2) {Yref--;} //CW Rotation
  if (Yref>5) {Yref=Yref-6;}
  if (Yref<0) {Yref=Yref+6;}
  //indicate the + Y access (on side 2): global orientation
  SideOnP(2-Yref);SideOffN(2-Yref);

  DelayXts(10);
  SideOnN(2-Yref);SideOffP(2-Yref);
  DelayXts(10);
  SideOffN(2-Yref);
}

```



```

}
//Mobile Module Rotation (CW) LED
void LMCW (int i,int t1,int t2, int t3)
{
  JointOn(i);JointOn(i+1);          //1 - Ji, Ji+1 Open
  DelayXts(t1);                     //2 - Delay t1
  SideOnN(i);SideOnN(i-1);         //3 - Si, Si-1 Negative
  DelayXts(t2);                     //4 - Delay t2
  JointOffAll();                   //5 - All Jx Close
  DelayXts(t3);                     //6 - Delay t3
                                     //7 - No action
  SideOffAllP();SideOffAllN();     //8 - All Sx Off
}
//Mobile Module Rotation (CCW) LED
void LMCCW (int i,int t1,int t2, int t3)
{
  JointOn(i+1);                     //1 - Ji+1 Open
  DelayXts(t1);                     //2 - Delay t1
  SideOnN(i);SideOnN(i-1);         //3 - Si, Si-1 Negative
  DelayXts(t2);                     //4 - Delay t2
  JointOn(i);                       //5 - Ji Open
  DelayXts(t3);                     //6 - Delay t3
  JointOffAll();                   //7 - All Jx Close
  SideOffAllP();SideOffAllN();     //8 - All Sx Off
}
//Substrate (CW) LED
void LSCW (int i,int t1,int t2, int t3)
{
  JointOn(i-1);                     //1 - Ji-1 Open
  DelayXts(t1);                     //2 - Delay t1
  SideOnN(i-3);SideOnP(i-2);       //3 - Si-3 Negative, Si-2 Positive
  DelayXts(t2);                     //4 - Delay t2
  JointOn(i-2);                     //5 - Ji-2 Open
  DelayXts(t3);                     //6 - Delay t3
  JointOffAll();                   //7 - All Jx Close
  SideOffAllP();SideOffAllN();     //8 - All Sx Off
}
//Substrate (CCW) LED
void LSCCW (int i,int t1,int t2, int t3)
{
  JointOn(i-3);JointOn(i-4);       //1 - Ji-3, Ji-4 Open
  DelayXts(t1);                     //2 - Delay t1
  SideOnN(i-4);SideOnN(i-5);       //3 - Si-4 Negative, Si-5 Positive
  DelayXts(t2);                     //4 - Delay t2
  JointOffAll();                   //5 - All Jx Close
  DelayXts(t3);                     //6 - Delay t3
                                     //7 - No action
  SideOffAllP();SideOffAllN();     //8 - All Sx Off
}

//***** PRIMARY LOCALIZATION AND MOTION FUNCTIONS *****
//Set the module as the ref.
void Ref (void)
{
  M=0;                               //Not a Mobile module
  S=1;                               //Substrate module
  L=1;                               //Localized
  X=0;                               //At X=0
  Y=0;                               //At Y=0
  Yref=0;                            //Orientation
  LYref(0);                          //Indicate the orientation
  LYref(0);                          //Indicate the orientation
  LYref(0);                          //Indicate the orientation
}
//Localize Others (for ref. or localized modules)
void Localize (void)
{
  int MSG1[10];                      //Localization Message (ref. to thesis)
  int i,j;

```

```

//Localized (ref.) modules
if (L==1)

{
  for (i=0;i<6;i++)           //Send the MSG1 6 times
  {
    SideOnP(2);SideOnN(2);    //Indication on side 2
    IrSel(2);                 //Select side 2
    DelayXts(5);PRE();       //Prepare

    LOCMMSG1(0,0,0,0,0,1,5);  //Send MSG1
    DelayXts(5);PRE();       //Prepare

    LOCMMSG1(0,0,0,0,0,1,5);  //Send MSG1
    SideOffP(2);SideOffN(2);  //Indication off

    SideOnP(3);SideOnN(3);    //Indication on side 3
    IrSel(3);                 //Select side 3
    DelayXts(5);PRE();       //Prepare

    LOCMMSG1(0,0,1,0,0,0,6);  //Send MSG1
    DelayXts(5);PRE();       //Prepare

    LOCMMSG1(0,0,1,0,0,0,6);  //Send MSG1
    SideOffP(3);SideOffN(3);  //Indication off
  }
  IrSel(0);                   //Listen to PC
}
//Non localized modules
if (L==0)

{
  for (j=1;j<7;j++)           //Send the MSG1 6 times
  {
    IrSel(j);                 //Listen to side j
    SideOnP(j);SideOnN(j);    //Indication
    i=ReceiveByte1TL(150);
    while ((i!='*')&&(i!='~')) {i=ReceiveByte1TL(150);} //Wait for the start of mes-
    sage for 1 sec
    if (i=='*')                //If there is a message (Lo-
    calized)
    {
      for (i=1;i<8;i++) {MSG1[i] = ReceiveByte1();} //Read its 7 bytes
      X=MSG1[2];Y=MSG1[6];    //Update location
      Yref=MSG1[7]-j;        //Update orientation: amount
    of CW rotations (ref-body)
    //Place Yref in the required range [0-5]
    while (Yref<0) {Yref=Yref+6;}
    while (Yref>5) {Yref=Yref-6;}
    L=1;j=7;IrSel(0);       //Localized, quit the loop
    LOff();                  //Turn off the LEDs
  }
  SideOffP(j);SideOffN(j);  //Indication
}
if (L==1)                  //If localized
{
  IrSel(0);
  PRE();
  LOC();
  //Indicate from which side the module received the localization information
  TransmitByte1('S');TransmitByte1('i');TransmitByte1('d');TransmitByte1('e');
  TransmitByte1(':');TransmitByte1(' ');
  TransmitByte1(MSG1[7]+48);TransmitByte1(10);TransmitByte1(13);
  //Indicate amount of rotation
  TransmitByte1('Y');TransmitByte1('r');TransmitByte1('e');TransmitByte1('f');
  TransmitByte1(':');TransmitByte1(' ');
  TransmitByte1(Yref+48);TransmitByte1(10);TransmitByte1(13);
  //Indicate the orientation
  LYref(0);LYref(0);LYref(0);
}

```

```

    else {LCircle(1,5);}           //Could not be localized
}
IrSel(0);                         //Listen to PC
}
// CW rotation [substrate, mobile)
int CW (int si,int st1,int st2, int st3, int mi,int mt1,int mt2, int mt3)
{
    int a=0;
    if (M==1) {a=MCW(mi,mt1,mt2,mt3);LMCW(mi,mt1,mt2,mt3);}
    if (S==1) {a=SCW(si,st1,st2,st3);LSCW(si,st1,st2,st3);}
    return a;
}
// CCW rotation
int CCW (int si,int st1,int st2, int st3, int mi,int mt1,int mt2, int mt3)
{
    int a=0;
    if (M==1) {a=MCCW(mi,mt1,mt2,mt3);LMCCW(mi,mt1,mt2,mt3);}
    if (S==1) {a=SCCW(si,st1,st2,st3);LSCCW(si,st1,st2,st3);}
    return a;
}

//***** FUNCTIONS CALLED FROM THE PC
*****
//Listening to the PC
void Listen (void)
{
    unsigned char R=0;           //Received Character
    int MSG[6];                 //Received Message
    int i,a=78;                 //a: acknowledge (default "N")

    //while (R!='*') {TransmitBytel(R);R=ReceiveBytel();}           //temp test
    while (ReceiveBytel()!='*') {};                                 //Wait for the start
of message
    for (i=1;i<10;i++) {MSG[i] = ReceiveBytel();}                   //Read the message

    //Run the required funcaion based on the "Function Table"

    //Control board functions
    if (MSG[1]==1) {a=JxOpen(MSG[2]);JointOn(MSG[2]);}
    if (MSG[1]==2) {a=JxClose(MSG[2]);JointOff(MSG[2]);}
    if (MSG[1]==3) {a=JxOpenAll();JointOnAll();}
    if (MSG[1]==4) {a=JxCloseAll();JointOffAll();}
    if (MSG[1]==5) {a=JTest(MSG[2],MSG[3]);JointTest(MSG[2],MSG[3]);}

    if (MSG[1]==11) {a=SxP(MSG[2]);SideOnP(MSG[2]);}
    if (MSG[1]==12) {a=SxN(MSG[2]);SideOnN(MSG[2]);}
    if (MSG[1]==13) {a=SxOff(MSG[2]);SideOffP(MSG[2]);SideOffN(MSG[2]);}
    if (MSG[1]==14) {a=SxOffAll();SideOffAllP();SideOffAllN();}
    if (MSG[1]==15) {a=SPTest(MSG[2],MSG[3]);SidePTest(MSG[2],MSG[3]);}
    if (MSG[1]==16) {a=SNTest(MSG[2],MSG[3]);SideNTest(MSG[2],MSG[3]);}

    if (MSG[1]==20) {a=TurnOffAll();LOff();}

    if (MSG[1]==21) {a=MCW(MSG[2],MSG[3],MSG[4],MSG[5]);LMCW(MSG[2],MSG[3],MSG[4],MSG[5]);}
    if (MSG[1]==22)
{a=MCCW(MSG[2],MSG[3],MSG[4],MSG[5]);LMCCW(MSG[2],MSG[3],MSG[4],MSG[5]);}
    if (MSG[1]==23) {a=SCW(MSG[2],MSG[3],MSG[4],MSG[5]);LSCW(MSG[2],MSG[3],MSG[4],MSG[5]);}
    if (MSG[1]==24)
{a=SCCW(MSG[2],MSG[3],MSG[4],MSG[5]);LSCCW(MSG[2],MSG[3],MSG[4],MSG[5]);}

    //Communication board functions
    if (MSG[1]==25) {IrSel(MSG[2]);a=MSG[1];}

    if (MSG[1]==28) {SideOnP(MSG[2]);a=MSG[1];}
    if (MSG[1]==29) {SideOffP(MSG[2]);a=MSG[1];}
    if (MSG[1]==30) {SideOnAllP();a=MSG[1];}
    if (MSG[1]==31) {SideOffAllP();a=MSG[1];}
    if (MSG[1]==32) {SideOnN(MSG[2]);a=MSG[1];}

```

```

if (MSG[1]==33) {SideOffN(MSG[2]);a=MSG[1];}
if (MSG[1]==34) {SideOnAllN();a=MSG[1];}
if (MSG[1]==35) {SideOffAllN();a=MSG[1];}
if (MSG[1]==36) {SidePTest(MSG[2],MSG[3]);a=MSG[1];}
if (MSG[1]==37) {SideNTest(MSG[2],MSG[3]);a=MSG[1];}

if (MSG[1]==38) {JointOn(MSG[2]);a=MSG[1];}
if (MSG[1]==39) {JointOff(MSG[2]);a=MSG[1];}
if (MSG[1]==40) {JointOnAll();a=MSG[1];}
if (MSG[1]==41) {JointOffAll();a=MSG[1];}
if (MSG[1]==43) {JointTest(MSG[2],MSG[3]);a=MSG[1];}

if (MSG[1]==44) {LOn();a=MSG[1];}
if (MSG[1]==45) {LOff();a=MSG[1];}
if (MSG[1]==46) {LCircle(MSG[2],MSG[3]);a=MSG[1];}
if (MSG[1]==47) {LColor(MSG[2],MSG[3]);a=MSG[1];}
if (MSG[1]==48) {LFlash1(MSG[2],MSG[3]);a=MSG[1];}
if (MSG[1]==49) {LFlash2(MSG[2],MSG[3]);a=MSG[1];}
if (MSG[1]==50) {LYref(MSG[2]);a=MSG[1];}

if (MSG[1]==56) {LMCW(MSG[2],MSG[3],MSG[4],MSG[5]);a=MSG[1];}
if (MSG[1]==57) {LMCCW(MSG[2],MSG[3],MSG[4],MSG[5]);a=MSG[1];}
if (MSG[1]==58) {LSCW(MSG[2],MSG[3],MSG[4],MSG[5]);a=MSG[1];}
if (MSG[1]==59) {LSCCW(MSG[2],MSG[3],MSG[4],MSG[5]);a=MSG[1];}

if (MSG[1]==65) {Ref();a=MSG[1];}
if (MSG[1]==66) {Localize();a=MSG[1];}
if (MSG[1]==67) {a=CW(MSG[2],MSG[3],MSG[4],MSG[5],MSG[6],MSG[7],MSG[8],MSG[9]);}
if (MSG[1]==68) {a=CCW(MSG[2],MSG[3],MSG[4],MSG[5],MSG[6],MSG[7],MSG[8],MSG[9]);}

//Command Received
if (S==1) {DelayXts(100);} //If it's the substrate module wait for 1sec to
avoid interference
PRE(); //Preamble - IR
// RCV(); //Received

//Acknowledgement (with the command code executed)
TransmitByte1('A');
TransmitByte1('C');
TransmitByte1('K');
TransmitByte1(a);
TransmitByte1(10);
TransmitByte1(13);
//Message Received
// TransmitByte1('M');

// TransmitByte1('S');
// TransmitByte1('G');
// for (i=1;i<10;i++) {TransmitByte1(MSG[i]);}
// TransmitByte1(10);
// TransmitByte1(13);
}

//***** MAIN PROGRAM *****
void main(void)
{
// int i,a; //Acknowledge
// unsigned char R;
init_devices();
LOff(); //Turn off all LEDs
IrSel(0); //Select the central communication
PRE(); //Preamble - IR
COMM_Ready(); //Communication Board Ready - IR

//Hand shaking between the control and communication microcontrollers
while (ReceiveByte0()!='R') {}; //Wait for the control board ready signal
LFlash1(3,10);
CTR_Ready(); //Control Board Ready - IR

TransmitByte0('R'); //Communication Ready signal

```

```

while (ReceiveByte0TL(10)!='L') {TransmitByte0('R');} //Wait for the control board
to listen
LFlash2(3,10);
CTR_LST(); //Control Board Listening -
IR

//***** Write the localization routine: done manually *****
LCircle(1,5); //Localization
Yref=1; //assuming
IrSel(0); //Select the central communi-
cation
LOC(); //localized

//Execute the required function
IrSel(0); //Select the central communi-
cation
LColor(3,10); //Listening
COMM_LST(); //Communication Board Listen-
ing - IR
while(1) {Listen();} //Start listening to PC
}

```

Wireless SPI Code

```

//ICC-AVR application builder : 12/23/2008 12:41:31 PM
// Target : M162
// Crystal: 1.0000Mhz

#include <iom162v.h>
#include <macros.h>

int Counter=0; //increased every 10ms
int DataTX[]; //SPI TX Data
int DataRX[]; //SPI RX Data

// SPI PORT CONFIGURATION
//PB4 - SLE (SS)
void SEL_OUTPUT (void) {DDRB|=(1<<4);}
void HI_SEL (void) {PORTB|=(1<<4);}
void LOW_SEL (void) {PORTB&=~(1<<4);}
//PB5 - SDI (MOSI)
void SDI_OUTPUT (void) {DDRB|=(1<<5);}
void HI_SDI (void) {PORTB|=(1<<5);}
void LOW_SDI (void) {PORTB&=~(1<<5);}
//PB6 - SDO (MISO)
void SDO_INPUT (void) {DDRB&=~(1<<6);}
int HI_SDO(void) {return PINB&(1<<6);}
void LOW_SDO (void) {PORTB&=~(1<<6);}
//PB7 - SCK (SCK)
void SCK_OUTPUT (void) {DDRB|=(1<<7);}
void HI_SCK (void) {PORTB|=(1<<7);}
void LOW_SCK (void) {PORTB&=~(1<<7);}
//PD4 - DATA
void DATA_OUT (void) {DDRD|=(1<<4);}
void HI_DATA (void) {PORTD|=(1<<4);}
//PD5 - nIRQ (INT0)
void IRQ_IN (void) {DDRD&=~(1<<5);}
void WAIT_IRQ_LOW (void) {while (PIND&(1<<5)) {;}}
//PD6 - GREEN LED
void LEDG_OUTPUT (void) {DDRD|=(1<<6);}
void LEDG_ON (void) {PORTD|=(1<<6);}
void LEDG_OFF (void) {PORTD&=~(1<<6);}
//PD7 - RED LED
void LEDR_OUTPUT (void) {DDRD|=(1<<7);}
void LEDR_ON (void) {PORTD|=(1<<7);}
void LEDR_OFF (void) {PORTD&=~(1<<7);}

// RF12 PORT INITIALIZATION
void RF12_PORT_INIT (void)

```

```

{
  HI_SEL();
  HI_SDI();
  LOW_SCK();
  SEL_OUTPUT();
  SDI_OUTPUT();
  SDO_INPUT();
  SCK_OUTPUT();
  DATA_OUT();
  HI_DATA(); //SEt nFFS pin HI when using FIFO ,TX register
  IRQ_IN(); //PD5(INT0)
}

// RF12 WRITE COMMAND
unsigned int RF12_WRT_CMD (unsigned int aCmd)
{
  unsigned char i;
  unsigned int temp=0;
  LOW_SCK();
  LOW_SEL();
  for (i=0;i<16;i++)
  {
    if(aCmd&0x8000) {HI_SDI();} else {LOW_SDI();}
    HI_SCK();
    aCmd<<=1;

    temp<<=1;
    if(HI_SDO()) {temp|=0x0001;}
    LOW_SCK();
  }
  HI_SEL();
  return (temp);
}

void RF12_INIT(void)
{
  RF12_WRT_CMD(0x80D7); // EL, EF, 433band,12, 0pF
  RF12_WRT_CMD(0x8239); // !er, !ebb, ET, ES, EX, !eb, !ew, DC
  RF12_WRT_CMD(0xA640); // A140=430.8 MHz
  RF12_WRT_CMD(0xC647); // 4.8kbps
  RF12_WRT_CMD(0x94A0); // VDI, FAST, 134kHz, 0dBm, -103dBm
  RF12_WRT_CMD(0xC2AC); // AL, !m1, DIG, DQD4
  RF12_WRT_CMD(0xCA81); // FIFO8, SYNC, !ff, DR
  RF12_WRT_CMD(0xCED4); // SYNC=2DD4
  RF12_WRT_CMD(0xC483); // @PWR, NO RSTRIC, !st, !fi, 0E, EN
  RF12_WRT_CMD(0x9850); // !mp, 9810=30kHz, MAX, OUT
  RF12_WRT_CMD(0xCC67); // OB1, OB0, ! lpx, !ddy, DDIT, BW0
  RF12_WRT_CMD(0xE000); // NOT USE
  RF12_WRT_CMD(0xC800); // NOT USE
  RF12_WRT_CMD(0xC400); // 1.66MHz, 2.2V
}

void RF12_SEND(unsigned char aByte)
{
  while (PIND&(1<<5)); //wait for previously TX over
  RF12_WRT_CMD(0xB800+aByte);
}

unsigned char RF12_RECV(void)
{
  unsigned int FIFO_data;
  WAIT_IRQ_LOW();
  RF12_WRT_CMD(0x0000);
  FIFO_data=RF12_WRT_CMD(0xB0000);
  return (FIFO_data&0x00FF);
}

void LED_DELAY(void)
{
  Counter=0;while (Counter<20) {;} //delay of 200ms
}

```

```

}

void POWER_ON_LED (void)
{
    int i;
    LEDG_OFF ();
    LEDR_OFF ();
    LEDG_OUTPUT ();
    LEDR_OUTPUT ();
    for (i=0;i<3;i++)
    {
        LEDG_ON ();LEDR_OFF ();LED_DELAY ();
        LEDG_OFF ();LEDR_ON ();LED_DELAY ();
    }
    LEDG_OFF ();LEDR_OFF ();
}

void RF12_TX(void)
{
    unsigned int i;
    unsigned char ChkSum;

    LEDR_ON ();LED_DELAY ();LEDR_OFF ();

    RF12_WRT_CMD(0x0000);//read status register
    RF12_WRT_CMD(0x8239);//!er, !ebb, ET, ES, EX, !eb, !ew, DC

    ChkSum=0;
    RF12_SEND(0xAA);//PREAMBLE
    RF12_SEND(0xAA);//PREAMBLE
    RF12_SEND(0xAA);//PREAMBLE
    RF12_SEND(0x2D);//SYNC HI BYTE
    RF12_SEND(0xD4);//SYNC LOW BYTE
    for (i=1;i<17;i++) {RF12_SEND(DataTX[i]);ChkSum+=DataTX[i];}//Send Data
    RF12_SEND(ChkSum);//Send ChkSum
    RF12_SEND(0xAA);//DUMMY BYTE
    RF12_SEND(0xAA);//DUMMY BYTE
    RF12_SEND(0xAA);//DUMMY BYTE
}

void RF12_RX(void)
{
    unsigned int i;
    unsigned char ChkSum;

    LEDG_ON ();LED_DELAY ();LEDG_OFF ();

    RF12_WRT_CMD(0xCA81);//Initialize FIFO
    RF12_WRT_CMD(0xCA83);//Enable FIFO

    ChkSum=0;
    for (i=1;i<17;i++) {DataRX[i]=RF12_RECV();ChkSum+=DataRX[i];}//Receive Data
    i=RF12_RECV();//Receive ChkSum

    RF12_WRT_CMD(0xCA81);//Disable FIFO

    if(ChkSum==i) {DataRX[17]=1;} else {DataRX[17]=0;}// Package Check
}

//SPI initialize
// clock rate: 250000hz
void spi_init(void)
{
    SPCR= 0x00; //diabile spi
    SPSR= 0x01; //2X
    SPCR= 0x43; //setup SPI
}

void port_init(void)
{

```

```

DDRA = 0x00;
 DDRB = 0x00;
 DDRC = 0x00;
 DDRD = 0x00;
 DDRE = 0x00;
}

//TIMER0 initialize - prescale:256
// WGM: Normal
// desired value: 10mSec
// actual value: 9.984mSec (0.2%)
void timer0_init(void)
{
  TCCR0= 0x00; //stop
  TCNT0= 0xD9; //set count
  OCR0= 0x27; //set compare value
  TCCR0= 0x04; //start timer
}

#pragma interrupt_handler timer0_ovf_isr:iv_TIMO_OVF
void timer0_ovf_isr(void)
{
  TCNT0= 0xD9; //reload counter value
  Counter++; //increase the counter every 10ms
}

//UART0 initialize
// desired baud rate: 4800
// actual: baud rate:4808 (0.2%)
// char size: 8 bit
// parity: Disabled
void uart0_init(void)
{
  UCSROB = 0x00; //disable while setting baud rate
  UCSROA = 0x00; //disable while setting baud rate
  UBRR0L =0x0C; //set baud rate
  UBRR0H = 0x00;
  UCSROC = BIT(URSEL0) | 0x06;
  UCSROA = 0x00; //enable
  UCSROB = 0x18; //enable
}

// Transmite0
void TransmitByte0( unsigned char data )
{while ( !(UCSR0A & (1<<UDRE0)) );UDR0 = data;}

//UART1 initialize
// desired baud rate:4800
// actual baud rate:4808 (0.2%)
// char size: 8 bit
// parity: Disabled
void uart1_init(void)
{
  UCSR1B = 0x00; //disable while setting baud rate
  UCSR1A = 0x00; //disable while setting baud rate
  UBRR1L =0x0C; //set baud rate
  UBRR1H = 0x00;
  UCSR1C = BIT(URSEL1) | 0x06;
  UCSR1A = 0x00; //enable
  UCSR1B = 0x18; //enable
}

// Transmite1
void TransmitByte1( unsigned char data )
{while ( !(UCSR1A & (1<<UDRE1)) );UDR1 = data;}

//call this routine to initialize all peripherals
void init_devices(void)
{
  //stop errant interrupts until set up

```



```
CLI(); //disable all interrupts
spi_init();
port_init();
timer0_init();
uart0_init();
uart1_init();

RF12_PORT_INIT();
RF12_INIT();

MCUCR= 0x00;
EMUCR = 0x00;
//GIMSK= 0x00;
GICR = 0x00;
TIMSK= 0x02; //timer interrupt sources
ETIMSK=0x00;
GICR= 0x00;
PCMSK0=0x00;
PCMSK1=0x00;
SEI(); //re-enable interrupts
//all peripherals are now initialized
}
// TEST FUNCTIONS
void RX_TEST(void)
{
    init_devices();
    POWER_ON_LED();
    DataTX[1]=1;
    DataTX[2]=2;
    DataTX[3]=3;
    DataTX[4]=4;
    DataTX[5]=5;
    DataTX[6]=6;
    DataTX[7]=7;
    DataTX[8]=8;
    DataTX[9]=9;
    while (1)
    {
        RF12_TX();
        LED_DELAY();
        LED_DELAY();
    }
}

void TX_TEST(void)
{
    int i;
    init_devices();
    POWER_ON_LED();
    while (1)
    {
        RF12_RX();
        TransmitBytel('S');LED_DELAY();
        TransmitBytel('T');LED_DELAY();
        TransmitBytel('A');LED_DELAY();
        TransmitBytel('R');LED_DELAY();
        TransmitBytel('T');LED_DELAY();
        for (i=1;i<18;i++)
        {
            TransmitBytel(DataRX[i]);LED_DELAY();
        }
    }
}
```

APPENDIX B

Visual basic code

This program is primarily developed as a Graphical User Interface (GUI) for the user to communicate to modules.

Port Configuration

```

Private Sub ClosePC_Click()
Me.Hide
End Sub

Public Sub MSComm1_OnComm()
On Error Resume Next
readByte = PortConf.MSComm1.Input
' Fix the new line problem
If readByte = Chr(10) Then
Form1.Text1.Text = Form1.Text1.Text & vbCrLf
End If
If readByte = Chr(13) Then
Form1.Text1.Text = Form1.Text1.Text
Else
If readByte <> Chr(10) Then
Form1.Text1.Text = Form1.Text1.Text + readByte
End If
End If

' Ack code:
S = InStrRev(Form1.Text1.Text, "ACK")
L = Len(Form1.Text1.Text)
If S <> 0 Then
If L > S + 3 Then
Form1.Ack1.Text = Asc(MID(Form1.Text1.Text, S + 3, 1))
End If
End If

' Message Received:
S = InStrRev(Form1.Text1.Text, "MSG")
L = Len(Form1.Text1.Text)
If S <> 0 Then
If L > S + 8 Then
Form1.FE1.Text = Asc(MID(Form1.Text1.Text, S + 3, 1))
Form1.FE2.Text = Asc(MID(Form1.Text1.Text, S + 4, 1))
Form1.FE3.Text = Asc(MID(Form1.Text1.Text, S + 5, 1))
Form1.FE4.Text = Asc(MID(Form1.Text1.Text, S + 6, 1))
Form1.FE5.Text = Asc(MID(Form1.Text1.Text, S + 7, 1))
End If
End If

End Sub

Private Sub MSComm2_OnComm()
On Error Resume Next
readByte = PortConf.MSComm2.Input
' Fix the new line problem
If readByte = Chr(10) Then
Form1.Text2.Text = Form1.Text2.Text & vbCrLf
End If
If readByte = Chr(13) Then
Form1.Text2.Text = Form1.Text2.Text
Else
If readByte <> Chr(10) Then
Form1.Text2.Text = Form1.Text2.Text + readByte
End If
End If

' Ack code:
S = InStrRev(Form1.Text2.Text, "ACK")
L = Len(Form1.Text2.Text)

```

```
    If S <> 0 Then
        If L > S + 3 Then
            Form1.Ack2.Text = Asc(MID(Form1.Text2.Text, S + 3, 1))
            Form1.MID.Text = MID(Form1.Text2.Text, S - 3, 1)
        End If
    End If

End Sub

Private Sub SetIR_Click()
On Error Resume Next
MSComm2.PortOpen = False
MSComm2.CommPort = PN2.Text
MSComm2.Settings = BR2.Text + "," + P2.Text + "," + DB2 + "," + SB2
MSComm2.PortOpen = True
End Sub

Private Sub SetUART1_Click()
On Error Resume Next
MSComm1.PortOpen = False
MSComm1.CommPort = PN1.Text
MSComm1.Settings = BR1.Text + "," + P1.Text + "," + DB1 + "," + SB1
MSComm1.PortOpen = True
End Sub
```

Main Window

```
Private Sub Ack2_Change()
On Error Resume Next
If Chr(Ack2.Text) = "N" Then Ack2.Text = "N"
End Sub

Private Sub ClearIR_Click()
Text2.Text = ""
Ack2.Text = "0"
MID.Text = "N"
RB2.Text = "0"
End Sub

Private Sub ClearMSG_Click()
MSG1.Text = "*"
MSG2.Text = "0"
MSG3.Text = "0"
MSG4.Text = "0"
MSG5.Text = "0"
MSG6.Text = "0"
MSG7.Text = "0"
MSG8.Text = "0"
MSG9.Text = "0"
MSG10.Text = "0"
MSG11.Text = "#"
End Sub

Private Sub ClearUART1_Click()
Text1.Text = ""
Ack1.Text = "0"
FE1.Text = "N"
FE2.Text = "N"
FE3.Text = "N"
FE4.Text = "N"
FE5.Text = "N"
RB1.Text = "0"
End Sub

Private Sub CloseMain_Click()
Unload Form1
Unload PortConf
End Sub
```

```
Private Sub ConnectIR_Click()  
On Error Resume Next  
PortConf.MSComm2.PortOpen = True  
ConnectIR.Enabled = False  
DisconnectIR.Enabled = True  
End Sub  
  
Private Sub ConnectUART1_Click()  
On Error Resume Next  
PortConf.MSComm1.PortOpen = True  
ConnectUART1.Enabled = False  
DisconnectUART1.Enabled = True  
End Sub  
  
Private Sub DisconnectIR_Click()  
On Error Resume Next  
PortConf.MSComm2.PortOpen = False  
ConnectIR.Enabled = True  
DisconnectIR.Enabled = False  
End Sub  
  
Private Sub DisconnectUART1_Click()  
On Error Resume Next  
PortConf.MSComm1.PortOpen = False  
ConnectUART1.Enabled = True  
DisconnectUART1.Enabled = False  
End Sub  
  
Private Sub FE1_Change()  
On Error Resume Next  
If Chr(FE1.Text) = "N" Then FE1.Text = "N"  
End Sub  
  
Private Sub FE2_Change()  
On Error Resume Next  
If Chr(FE2.Text) = "N" Then FE2.Text = "N"  
End Sub  
  
Private Sub FE3_Change()  
On Error Resume Next  
If Chr(FE3.Text) = "N" Then FE3.Text = "N"  
End Sub  
  
Private Sub FE4_Change()  
On Error Resume Next  
If Chr(FE4.Text) = "N" Then FE4.Text = "N"  
End Sub  
  
Private Sub FE5_Change()  
On Error Resume Next  
If Chr(FE5.Text) = "N" Then FE5.Text = "N"  
End Sub  
  
Private Sub Form_Unload(Cancel As Integer)  
On Error Resume Next  
PortConf.MSComm1.PortOpen = False  
PortConf.MSComm2.PortOpen = False  
End Sub  
  
Private Sub Fun1_Click()  
MSG2.Text = 1  
SendMSG1() = True  
End Sub  
  
Private Sub Fun2_Click()  
MSG2.Text = 2  
SendMSG1() = True  
End Sub  
  
Private Sub Fun3_Click()
```

```
MSG2.Text = 3
SendMSG1() = True
End Sub

Private Sub Fun4_Click()
MSG2.Text = 4
SendMSG1() = True
End Sub

Private Sub Fun5_Click()
MSG2.Text = 5
SendMSG1() = True
End Sub

Private Sub Fun11_Click()
MSG2.Text = 11
SendMSG1() = True
End Sub

Private Sub Fun12_Click()
MSG2.Text = 12
SendMSG1() = True
End Sub

Private Sub Fun13_Click()
MSG2.Text = 13
SendMSG1() = True
End Sub

Private Sub Fun14_Click()
MSG2.Text = 14
SendMSG1() = True
End Sub

Private Sub Fun15_Click()
MSG2.Text = 15
SendMSG1() = True
End Sub

Private Sub Fun16_Click()
MSG2.Text = 16
SendMSG1() = True
End Sub

Private Sub Fun20_Click()
MSG2.Text = 20
SendMSG1() = True
End Sub

Private Sub Fun21_Click()
MSG2.Text = 21
SendMSG1() = True
End Sub

Private Sub Fun22_Click()
MSG2.Text = 22
SendMSG1() = True
End Sub

Private Sub Fun23_Click()
MSG2.Text = 23
SendMSG1() = True
End Sub

Private Sub Fun24_Click()
MSG2.Text = 24
SendMSG1() = True
End Sub

Private Sub Fun25_Click()
```

```
MSG2.Text = 25
SendMSG1() = True
End Sub

Private Sub Fun28_Click()
MSG2.Text = 28
SendMSG1() = True
End Sub

Private Sub Fun29_Click()
MSG2.Text = 29
SendMSG1() = True
End Sub

Private Sub Fun30_Click()
MSG2.Text = 30
SendMSG1() = True
End Sub

Private Sub Fun31_Click()
MSG2.Text = 31
SendMSG1() = True
End Sub

Private Sub Fun32_Click()
MSG2.Text = 32
SendMSG1() = True
End Sub

Private Sub Fun33_Click()
MSG2.Text = 33
SendMSG1() = True
End Sub

Private Sub Fun34_Click()
MSG2.Text = 34
SendMSG1() = True
End Sub

Private Sub Fun35_Click()
SendMSG1() = True
MSG2.Text = 35
End Sub

Private Sub Fun36_Click()
MSG2.Text = 36
SendMSG1() = True
End Sub

Private Sub Fun37_Click()
MSG2.Text = 37
SendMSG1() = True
End Sub

Private Sub Fun38_Click()
MSG2.Text = 38
SendMSG1() = True
End Sub

Private Sub Fun39_Click()
MSG2.Text = 39
SendMSG1() = True
End Sub

Private Sub Fun40_Click()
MSG2.Text = 40
SendMSG1() = True
End Sub

Private Sub Fun41_Click()
```

```
MSG2.Text = 41
SendMSG1() = True
End Sub

Private Sub Fun43_Click()
MSG2.Text = 43
SendMSG1() = True
End Sub

Private Sub Fun44_Click()
MSG2.Text = 44
SendMSG1() = True
End Sub

Private Sub Fun45_Click()
MSG2.Text = 45
SendMSG1() = True
End Sub

Private Sub Fun46_Click()
MSG2.Text = 46
SendMSG1() = True
End Sub

Private Sub Fun47_Click()
MSG2.Text = 47
SendMSG1() = True
End Sub

Private Sub Fun48_Click()
MSG2.Text = 48
SendMSG1() = True
End Sub

Private Sub Fun49_Click()
MSG2.Text = 49
SendMSG1() = True
End Sub

Private Sub Fun50_Click()
MSG2.Text = 50
SendMSG1() = True
End Sub

Private Sub Fun56_Click()
MSG2.Text = 56
SendMSG1() = True
End Sub

Private Sub Fun57_Click()
MSG2.Text = 57
SendMSG1() = True
End Sub

Private Sub Fun58_Click()
MSG2.Text = 58
SendMSG1() = True
End Sub

Private Sub Fun59_Click()
MSG2.Text = 59
SendMSG1() = True
End Sub

Private Sub Fun65_Click()
MSG2.Text = 65
SendMSG1() = True
End Sub

Private Sub Fun66_Click()
```



```
MSG2.Text = 66
SendMSG1() = True
End Sub

Private Sub Fun67_Click()
MSG2.Text = 67
SendMSG1() = True
End Sub

Private Sub Fun68_Click()
MSG2.Text = 68
SendMSG1() = True
End Sub

Private Sub Fun69_Click()
MSG2.Text = 69
SendMSG1() = True
End Sub

Private Sub PortConfig_Click()
On Error Resume Next
PortConf.Show
End Sub

Private Sub SendMSG1_Click()
On Error Resume Next
PortConf.MSComm2.Output = MSG1.Text
PortConf.MSComm2.Output = Chr(MSG2.Text)
PortConf.MSComm2.Output = Chr(MSG3.Text)
PortConf.MSComm2.Output = Chr(MSG4.Text)
PortConf.MSComm2.Output = Chr(MSG5.Text)
PortConf.MSComm2.Output = Chr(MSG6.Text)
PortConf.MSComm2.Output = Chr(MSG7.Text)
PortConf.MSComm2.Output = Chr(MSG8.Text)
PortConf.MSComm2.Output = Chr(MSG9.Text)
PortConf.MSComm2.Output = Chr(MSG10.Text)
PortConf.MSComm2.Output = MSG11.Text
End Sub

Private Sub SendMSG2_Click()
On Error Resume Next
PortConf.MSComm2.Output = MSG1.Text
PortConf.MSComm2.Output = Chr(MSG2.Text)
PortConf.MSComm2.Output = Chr(MSG3.Text)
PortConf.MSComm2.Output = Chr(MSG4.Text)
PortConf.MSComm2.Output = Chr(MSG5.Text)
PortConf.MSComm2.Output = Chr(MSG6.Text)
PortConf.MSComm2.Output = Chr(MSG7.Text)
PortConf.MSComm2.Output = Chr(MSG8.Text)
PortConf.MSComm2.Output = Chr(MSG9.Text)
PortConf.MSComm2.Output = Chr(MSG10.Text)
PortConf.MSComm2.Output = MSG11.Text
End Sub

Private Sub Text1_Change()
' Number of Bytes Received
RB1.Text = Len(Form1.Text1.Text)
End Sub

Private Sub Text2_Change()
' Number of Bytes Received
RB2.Text = Len(Form1.Text2.Text)
End Sub
```

APPENDIX C

Matlab Functions

These functions were primarily developed to be utilized for the algorithm and simulation in Matlab. Moreover there are few test codes attached illustrating the use of different functions and the complete algorithm as well.

Function 1: FindVE

```
% [V,E] = FindVE(I, G)
%
% Given
% I: initial state
% G: goal state
%
% Find
% V: voids
% E: electrons

function [V,E] = FindVE(I, G)
V = zeros(size(I));
E = zeros(size(I));
A = G - I;
X = find(A== -1);
E(X)=1;
X = find(A==1);
V(X)=1;
end
```

Function 2: Hex

```
% [Vx Vy] = Hex(C)
%
% Given
% C: center of a hexagon
%
% Find
% Vx: six x coordinates of vertices
% Vy: six y coordinates of vertices
%
% Notes:
% Using this function you can enter the center of the hexagon (in Hex
% Coordinates) and it will give you the vertex (in Cartesian Coordinates)
% C: [x y]
% Vx: [px1 px2 px3 px4 px5 px6]
% Vy: [py1 py2 py3 py4 py5 py6]

function [Vx Vy] = Hex(C)
a = 1/sin(pi/3);
b = tan(pi/6);

x = C(1,1);
y = C(1,2)*a + x*b;

a = 1/3;
b = 1/(3*tan(pi/6));

px1 = x - 2*a;
py1 = y;

px2 = x - a;
py2 = y + b;

px3 = x + a;
py3 = y + b;

px4 = x + 2*a;
```

```

py4 = y;

px5 = x + a;
py5 = y - b;

px6 = x - a;
py6 = y - b;

Vx = [px1 px2 px3 px4 px5 px6];
Vy = [py1 py2 py3 py4 py5 py6];
end

```

Function 3: HexActuate

```

% [Fj, Fs]=HexActuate(s1,s2,S)
%
% Given
% s1: initial location of the mobile electron
% s2: next location of the mobile electron
% S: current system configuration
%
% Find
% Fj: joint forces
% Fs: side forces
%
% Example
% S =
%      0      0      0      0      0      0      0      0      0
%      0      0      0      0      0      0      0      0      0
%      0      0      0      0      0      0      0      0      0
%      0      0      1      0      0      0      0      0      0
%      0      0      1      1      1      1      1      0      0
%      0      0      0      0      0      0      0      0      0
%      0      0      0      0      0      0      0      0      0
%      ];
% s1=[5,7];
% s2=[4,6];
% [Fj, Fs]=HexActuate(s1,s2,S);
% G = Hexmove(S,Fj,Fs);

function [Fj, Fs]=HexActuate(s1,s2,S)

% Refer to Fig 4-29 of thesis (Matlab Coordinates)
A = [-1, -1];
B = [-1, 0];
C = [0, 1];
D = [1, 1];
E = [1, 0];
F = [0, -1];
G = [-2, -1];
H = [-1, 1];
I = [1, 2];
J = [2, 1];
K = [1, -1];
L = [-1, -2];

% Relative relocation
s=s2-s1;

% Absolute locations
a=s1+A;
b=s1+B;
c=s1+C;
d=s1+D;
e=s1+E;
f=s1+F;
g=s1+G;
h=s1+H;
i=s1+I;

```

```

j=s1+J;
k=s1+K;
l=s1+L;

% Refer to Hexmove function
Fj=zeros(1,3);
Fs=zeros(2,4);

% Refer to Table 4-3 of thesis
m1=a;m2=d;m3=e;m4=k;m5=F;
if
(S(m1(1,1),m1(1,2))==1)&&(S(m2(1,1),m2(1,2))==0)&&(S(m3(1,1),m3(1,2))==0)&&(S(m4(1,1),m4(
1,2))==0)&&isequal(m5,s)
    Fj=[s1 1];
    Fs=[s1 1 6
        a 5 4];
    % disp('1')
end
m1=b;m2=e;m3=f;m4=l;m5=A;
if
(S(m1(1,1),m1(1,2))==1)&&(S(m2(1,1),m2(1,2))==0)&&(S(m3(1,1),m3(1,2))==0)&&(S(m4(1,1),m4(
1,2))==0)&&isequal(m5,s)
    Fj=[s1 2];
    Fs=[s1 2 1
        b 6 5];
    % disp('2')
end
m1=c;m2=f;m3=a;m4=g;m5=B;
if
(S(m1(1,1),m1(1,2))==1)&&(S(m2(1,1),m2(1,2))==0)&&(S(m3(1,1),m3(1,2))==0)&&(S(m4(1,1),m4(
1,2))==0)&&isequal(m5,s)
    Fj=[s1 3];
    Fs=[s1 3 2
        c 1 6];
    % disp('3')
end
m1=d;m2=a;m3=b;m4=h;m5=C;
if
(S(m1(1,1),m1(1,2))==1)&&(S(m2(1,1),m2(1,2))==0)&&(S(m3(1,1),m3(1,2))==0)&&(S(m4(1,1),m4(
1,2))==0)&&isequal(m5,s)
    Fj=[s1 4];
    Fs=[s1 4 3
        d 2 1];
    % disp('4')
end
m1=e;m2=b;m3=c;m4=i;m5=D;
if
(S(m1(1,1),m1(1,2))==1)&&(S(m2(1,1),m2(1,2))==0)&&(S(m3(1,1),m3(1,2))==0)&&(S(m4(1,1),m4(
1,2))==0)&&isequal(m5,s)
    Fj=[s1 5];
    Fs=[s1 5 4
        e 3 2];
    % disp('5')
end
m1=f;m2=c;m3=d;m4=j;m5=E;
if
(S(m1(1,1),m1(1,2))==1)&&(S(m2(1,1),m2(1,2))==0)&&(S(m3(1,1),m3(1,2))==0)&&(S(m4(1,1),m4(
1,2))==0)&&isequal(m5,s)
    Fj=[s1 6];
    Fs=[s1 6 5
        f 4 3];
    % disp('6')
end
m1=f;m2=c;m3=b;m4=g;m5=A;
if
(S(m1(1,1),m1(1,2))==1)&&(S(m2(1,1),m2(1,2))==0)&&(S(m3(1,1),m3(1,2))==0)&&(S(m4(1,1),m4(
1,2))==0)&&isequal(m5,s)
    Fj=[s1 1];
    Fs=[s1 1 6
        f 2 3];

```

```

% disp('11')
end
m1=a;m2=d;m3=c;m4=h;m5=B;
if
(S(m1(1,1),m1(1,2))==1)&&(S(m2(1,1),m2(1,2))==0)&&(S(m3(1,1),m3(1,2))==0)&&(S(m4(1,1),m4(
1,2))==0)&&isequal(m5,s)
    Fj=[s1 2];
    Fs=[s1 2 1
        a 3 4];
% disp('22')
end
m1=b;m2=e;m3=d;m4=i;m5=C;
if
(S(m1(1,1),m1(1,2))==1)&&(S(m2(1,1),m2(1,2))==0)&&(S(m3(1,1),m3(1,2))==0)&&(S(m4(1,1),m4(
1,2))==0)&&isequal(m5,s)
    Fj=[s1 3];
    Fs=[s1 3 2
        b 4 5];
% disp('33')
end
m1=c;m2=f;m3=e;m4=j;m5=D;
if
(S(m1(1,1),m1(1,2))==1)&&(S(m2(1,1),m2(1,2))==0)&&(S(m3(1,1),m3(1,2))==0)&&(S(m4(1,1),m4(
1,2))==0)&&isequal(m5,s)
    Fj=[s1 4];
    Fs=[s1 4 3
        c 5 6];
% disp('44')
end
m1=d;m2=a;m3=f;m4=k;m5=E;
if
(S(m1(1,1),m1(1,2))==1)&&(S(m2(1,1),m2(1,2))==0)&&(S(m3(1,1),m3(1,2))==0)&&(S(m4(1,1),m4(
1,2))==0)&&isequal(m5,s)
    Fj=[s1 5];
    Fs=[s1 5 4
        d 6 1];
% disp('55')
end
m1=e;m2=b;m3=a;m4=l;m5=F;
if
(S(m1(1,1),m1(1,2))==1)&&(S(m2(1,1),m2(1,2))==0)&&(S(m3(1,1),m3(1,2))==0)&&(S(m4(1,1),m4(
1,2))==0)&&isequal(m5,s)
    Fj=[s1 6];
    Fs=[s1 6 5
        e 1 2];
% disp('66')
end

% Conversion to Hex coordinates
s = size(S,1);

mx = Fj(1,2);my = Fj(1,1);
hx = mx-1;hy = s-my;
Fj(1,1)=hx;Fj(1,2)=hy;
Fs(1,1)=hx;Fs(1,2)=hy;

mx = Fs(2,2);my = Fs(2,1);
hx = mx-1;hy = s-my;
Fs(2,1)=hx;Fs(2,2)=hy;
end

```

Function 4: HexAll_P

```

% [Joint, Side, S]=HexAll_P(I,G)
%
% Given
% I: Initial Configuration (State)
% G: Goal Configuration (State)
%

```

```

% Find
% Joint: Required Joint Forces
% Side: Required Side Forces
% S: Parallel Motion, each row: [s1:current location of the mobile module, s2:NRL, s3:PV]

function [Joint, Side, S]=HexAll_P(I,G)
% ***** Layer 1: Initial and Goal States *****
%Entered Manually at this stage, Ex:
% I = [0 0 0 0 0
%      1 0 0 0 0
%      1 1 1 1 1];
%
% G = [1 1 0 0 0
%      1 0 1 0 0
%      0 1 1 0 0];
%Double Zero Padding
%I = HexZP(I);I = HexZP(I);
%G = HexZP(G);G = HexZP(G);

% ***** Layer 2: Mobile Electrons and Potential Voids *****
[V E] = FindVE(I,G); %Find all voids and electrons
PoV = PV(I,G); %Find potential voids
ME = Mobile(E,I); %Find mobile electrons
if (sum(sum(PoV))==0) || (sum(sum(ME))==0)
    disp('There is no more possible motion');return;
end

% ***** Layer 3: Void Propagation *****
PVME=HexVP(I,G);
s = size (PVME,1); %Number of parallel movements

% ***** Layer 4: MDP *****
for i=1:s
    s1=[PVME(i,3) PVME(i,4)]; %ME
    s2=[PVME(i,1) PVME(i,2)]; %PV
    NRL = MDP_NRL(I,G,s1,s2,0); %Next Recommended Location for ME
    Motion(i,:)= [s1 NRL s2];
end

% ***** Layer 5: Actuation *****
for i=1:s
    s1=[Motion(i,1) Motion(i,2)]; %Current Location
    s2=[Motion(i,3) Motion(i,4)]; %Next Location
    s3=[Motion(i,5) Motion(i,6)]; %Desired Location
    [Fj, Fs]=HexActuate(s1,s2,I); %Required Actuation
    Joint(i,:)=Fj; %Joint Forces
    Side(i,:)= [Fs(1,:), Fs(2,:)]; %Side Forces
    S(i,:)= [s1, s2, s3];
end
end

```

Function 5: HexAll_S

```

%[Joint, Side, S]=HexAll_S(I,G)
%
%Given
%I: Initial Configuration (State)
%G: Goal Configuration (State)
%
%Find
%Joint: Required Joint Forces
%Side: Required Side Forces
%S: Serial Motion (One module movement at a time): [s1:current location of the mobile module, s2:NRL, s3:PV]

function [Joint, Side, S]=HexAll_S(I,G)
% ***** Layer 1: Initial and Goal States *****
%Entered Manually at this stage, Ex:
% I = [0 0 0 0 0

```

```

%      1 0 0 0 0
%      1 1 1 1 1];
%
% G = [1 1 0 0 0
%      1 0 1 0 0
%      0 1 1 0 0];
%Double Zero Padding
%I = HexZP(I);I = HexZP(I);
%G = HexZP(G);G = HexZP(G);

% ***** Layer 2: Mobile Electrons and Potential Voids *****
[V E] = FindVE(I,G);      %Find all voids and electrons
PoV = PV(I,G);           %Find potential voids
ME = Mobile(E,I);        %Find mobile electrons
if (sum(sum(PoV))==0) || (sum(sum(ME))==0)
    disp('There is no more possible motion');return;
end

% ***** Layer 3: Void Propagation *****
PVME=HexVP(I,G);

% ***** Layer 4: MDP *****
s1=[PVME(1,3) PVME(1,4)]; %ME
s2=[PVME(1,1) PVME(1,2)]; %PV
NRL = MDP_NRL(I,G,s1,s2,0); %Next Recommended Location for ME
Motion(1,:)=s1 NRL s2];

% ***** Layer 5: Actuation *****
s1=[Motion(1,1) Motion(1,2)]; %Current Location
s2=[Motion(1,3) Motion(1,4)]; %Next Location
s3=[Motion(1,5) Motion(1,6)]; %Desired Location
[Fj, Fs]=HexActuate(s1,s2,I); %Required Actuation
Joint(1,:)=Fj; %Joint Forces
Side(1,:)=Fs(1,:), Fs(2,:)); %Side Forces
S(1,:)=s1, s2, s3];
end

```

Function 6: HexCCW

```

% C_new = HexCCW(C_old, Joint)
%
% Given
% C_old: current coordinates of the center of a module
% Joint: around which a CCW rotation is happening
%
% Find
% C_new: next coordinates of the center after the rotation
%
%Notes:
% Counter clockwise rotation of module around a joint
% Center (old and new) is a: [x, y]

function C_new = HexCCW(C_old, Joint)
CCW = [1 2 3 4 5 6;
       -1 0 +1 +1 0 -1;
       +1 +1 0 -1 -1 0];

C_new = [C_old(1,1)+CCW(2,Joint), C_old(1,2)+CCW(3,Joint)];
end

```

Function 7: HexCG

```

%HexCG(I)
%
%Given
% I: configuration matrix
%
% Find

```



```

% plot the connectivity graph
%
% Notes:
% note that all sides of the configuration should be zero padded
% to allow module movements on the sides (Use HexZP)

function HexCG(I)

F = find(I);[X Y] = find(I); %find all modules in the configuration
s = size(F,1); %number of modules in the configuration

for i=1:s
    IDs(1,i) = {num2str(F(i))};
end

%CM = zeros(s(1,1),s(1,1)); %connection matrix (square matrix,
number of nodes)
UG = sparse([],[],true,s,s); %Undirected Graph, same size of the
nodes

%To have undirected-like of graph
for i=1:s-1
    x = X(i);y = Y(i); %Start with one node, check if others
are connected to it
    N = HexNeM(x,y); %find neighbor modules
    for j=i+1:s %check the connection to the remaining
modules
        id = str2num(cell2mat(IDs(1,j))); %get the id of the node
        for k=1:6 %check if this id is in the neighbor-
hood
            nid = (N(k,1)+(N(k,2)-1)*size(I,1));
            if nid==id %if id is found to be in the neighbor-
hood
                UG(i,j) = true; %connect to that module
            end
        end
    end
end

bg = biograph(UG,IDs); %Construct the graph with no connection
set(bg.Nodes, 'Shape', 'Circle')
set(bg, 'ShowArrows', 'off')
Grid = ones(size(I));
figure,axis off;hold on;Hexpm(Grid);Hexplot(I,1);
bg.view;
end

% To display the descendants of a node
% i=1
% desNodes = getdescendants(bg.nodes(i),1);
% set(desNodes,'Color',[1 .7 .7]);

```

Function 8: HexCGT

```

% C = HexCGT(I)
%
% Given
% I: configuration matrix
%
% Find
% C = 1 if the graph is connected and returns C = 0 if graph is not connected
%
% Notes:
% note that all sides of the configuration should be zero padded
% to allow module movements on the sides (Use HexZ0)

function C = HexCGT(I)

F = find(I);[X Y] = find(I); %find all modules in the configuration

```

```

s = size(F,1); %number of modules in the configuration

for i=1:s
    IDs (1,i) = {num2str(F(i))};
end

UG = sparse([ ],[ ],true,s,s); %Undirected Graph, same size of the
nodes

for i=1:s % [1:S-1]
    x = X(i);y = Y(i); %Start with one node, check if others
    are connected to it
    N = HexNeM(x,y); %find neighbor modules
    for j=1:s % [I+1:S] %check the connection to the remaining
modules
        id = str2num(cell2mat(IDs(1,j))); %get the id of the node
        for k=1:6 %check if this id is in the neighbor-
hood
            nid = (N(k,1)+(N(k,2)-1)*size(I,1));
            if nid==id %if id is found to be in the neighbor-
hood
                UG(i,j) = true; %connect to that module
            end
        end
    end
end

P = graphallshortestpaths(UG, 'directed', false);

T = sum(sum(P));
if (T==Inf)
    C=0;
else
    C=1;
end

```

Function 9: HexCW

```

% C_new = HexCW(C_old, Joint)
%
% Given
% C_old: current coordinates of the center of a module
% Joint: around which a CW rotation is happening
%
% Find
% C_new: next coordinates of the center after the rotation
%
%Notes:
% Counter clockwise rotation of module around a joint
% Center (old and new) is a: [x, y]

function C_new = HexCW(C_old, Joint)
CW = [1 2 3 4 5 6;
      -1 -1 0 +1 +1 0;
        0 +1 +1 0 -1 -1];

C_new = [C_old(1,1)+CW(2,Joint), C_old(1,2)+CW(3,Joint)];
end

```

Function 10: Hexgrid

```

% Hexgrid(min, max)
%
% Given
% (min, max): ranges to grid
%
% Find
% Adds the Hex Axis Grids

```

```

function Hexgrid(min, max)

min = floor(min);
max = ceil(max);
a = 1/sin(pi/3);
b = tan(pi/6);

% find the grid points
hold on;
for i = min:max
    for j = min:max
        plot(i,j*a+i*b, 'bo');
    end
end
min = min*(a+b)-1;
max = max*(a+b)+1;
axis ([min max min max]);
shg;
end

```

Function 11: HexIMS

```

% x = HexIMS(I,s)
%
% Given
% I: a configuration matrix
% s: a location in Matlab Coordinates
%
% Find
% x=1 if s has an immobile substrate in I
%
% Example
% I = [0 0 0 0 0
%      1 0 0 0 0
%      1 1 1 1 1
%      0 0 0 0 0];
% s = [1 1];

function x = HexIMS(I,s)
xmax=size(I,1);ymax=size(I,2);
x=0;
N=HexNeM(s(1,1),s(1,2));%check the neighbors
for j=1:6
    xn=N(j,1);yn=N(j,2);
    if (xn>0&&xn<=xmax&&yn>0&&yn<=ymax) && (I(xn,yn)==1)
        x=1;
    end
end
end

```

Function 12: Hexmove

```

% G = Hexmove(I, Fj, Fs, p)
%
% Given
% I: Initial State of the modules
% Fj: Joint Forces (the only close joint for the mobile module)
%      [could be found from Fs automatically as well, but included to simulate the actual
%      microcontroller commands]
% Fs: Side Forces (Energized magnets: Mobile and neighbor)
% p: plotting (1:plot, 0:don't plot)
%
% Find
% G: Goal State, next state after the actuation happens
%
% Notes:
% Move from Initial state to Goal state based on Joint and Side forces

```

```

%
% Example
% I = [1 1 1;
%      1 1 0;
%      0 0 1]
% Fj = [2 0 2] %second joint of the mobile module (2,0)
% Fs = [2 0 1 2; %sides 1 and 2 of mobile (2,0) are -ve
%        1 1 3 4] %side 3 of neighbor (2,1) is +ve and 4 -ve
% G = [1 1 1;
%      1 1 1;
%      0 0 0]

function G = Hexmove(varargin)
I = varargin{1};
Fj = varargin{2};
Fs = varargin{3};
p=0;
if length(varargin) > 3
    p = cell2mat(varargin{4});
end

% FIND DIRECTION OF ROTATION
% (if fixed joint number and repulsion (-ve) side number are the same CW otherwise CCW)
% neighbor repulsion points (N)
[Vx Vy]=Hex([Fs(2,1),Fs(2,2)]);
a = Fs(2,4);b = Fs(2,4)+1;if (b==7) b=1; end
N = [Vx(a) Vy(a);
     Vx(b) Vy(b)];

% mobile module points (M)
[Vx Vy]=Hex([Fj(1,1),Fj(1,2)]);
a = Fj(1,3);b = Fj(1,3)+1;if (b==7) b=1; end
M = [Vx(a) Vy(a);
     Vx(b) Vy(b)];

%round the numbers so that "isequal" works fine
N=round(1000*N)/1000;
M=round(1000*M)/1000;

if isequal(N,M)
    C=HexCW([Fj(1,1), Fj(1,2)],Fj(1,3));
elseif isequal(N,flipdim(M,1))
    C=HexCW([Fj(1,1), Fj(1,2)],Fj(1,3));
else
    C=HexCCW([Fj(1,1), Fj(1,2)],Fj(1,3));
end

% FORM THE GOAL STATE
G = flipdim(I,1);
G(Fj(1,2)+1,Fj(1,1)+1)=0;
G(C(1,2)+1,C(1,1)+1)=1;
G = flipdim(G,1);

if (p==1)
    % PLOT THE GIVEN DATA
    P = 1; % Speed
    Grid = ones(size(I));
    scrsz = get(0,'ScreenSize');
    figure('Name','Simulation Window','NumberTitle','off','Position',[1 1 scrsz(3)
scrsz(4)]);hold on
    subplot(1,2,1);axis off;title('Current Module Configuration + Actua-
tion');Hexpm(Grid);Hexplot(I,1);Jointplot(Fj);Sideplot(Fs);
    subplot(1,2,2);axis off;title('Next Configuration');Hexpm(Grid);Hexplot(G,1);
    pause(P);%close ('Simulation Window')
end
end

```

Function 13: HexNe

```

% [N] = HexNe(x,y)
%
% Given
% (x,y) coordinates of a module
%
% Find
% N: locations of neighbor modules
%
% Notes:
% This function works with HEX COORDINATES

function [N] = HexNe(x,y)
N = [x-1, y+1; %side 1
     x, y+1; %side 2
     x+1,y; %side 3
     x+1,y-1; %side 4
     x, y-1; %side 5
     x-1, y]; %side 6
end

```

Function 14: HexNeM

```

% [N] = HexNeM(i, j)
%
% Given
% (i,j): coordinates of a module (Matlab indices of an array)
%
% Find
% N: locations of neighbor modules
%
% Notes:
% This function works with MATLAB COORDINATES

function [N] = HexNeM(i, j)
N = [i-1, j-1; %side 1
     i-1, j; %side 2
     i, j+1; %side 3
     i+1, j+1; %side 4
     i+1, j; %side 5
     i, j-1]; %side 6
end

```

Function 15: Hexp

```

% Hexp(C)
%
% Given
% C: Center of a hexagonal
%
% Find
% draws the hexagonal (used for plotting purposes only)

function Hexp(C)
a = 1/sin(pi/3);
b = tan(pi/6);

x = C(1,1);
y = C(1,2)*a + x*b;

a = 1/3;
b = 1/(3*tan(pi/6));

px1 = x - 2*a;
py1 = y;

```

```

px2 = x - a;
py2 = y + b;

px3 = x + a;
py3 = y + b;

px4 = x + 2*a;
py4 = y;

px5 = x + a;
py5 = y - b;

px6 = x - a;
py6 = y - b;

Vx = [px1 px2 px3 px4 px5 px6 px1];
Vy = [py1 py2 py3 py4 py5 py6 py1];

hold on
plot (Vx,Vy,'k','LineWidth',2)
end

```

Function 16: Hexplot

```

% Hexplot(A,g,c)
%
% Given
% A: configuration matrix
% g: don't add grid if g=1
% c: set color
%
% Find
% draws the hexagonal
%
% Notes:
% Plots Hex for nonzero elements of A (in Hex Coordinates)
% By default adds grid; does not add the grid if g is 1. c is the color
%
% Example
% | (0,2) (1,2) (2,2) |
% A | (0,1) (1,1) (2,2) |
% | (0,0) (1,0) (2,0) |

function Hexplot(varargin)

A = varargin{1};
g = 0;
c = 'y';

if length(varargin) > 1
    g = cell2mat(varargin(2));
end

if length(varargin) > 2
    c = cell2mat(varargin(3));
end

A = flipdim(A,1);
[b,a] = find(A);
a = a-1;b=b-1;

s = max(size(a));
hold on;
for i=1:s
    [Vx,Vy] = Hex([a(i),b(i)]);
    fill (Vx,Vy,c);
    Hexp([a(i),b(i)]);
end

```

```

if g ~= 1
    Hexgrid(0,max(size(A))-1)
end

```

Function 17: Hexpm

```

% Hexpm(A)
%
% Given
% A: configuration matrix
%
% Find
% plots background Hex for nonzero elements of A (in Hex Coordinates:)
%
% Example
% | (0,2) (1,2) (2,2) |
% A | (0,1) (1,1) (2,2) |
% | (0,0) (1,0) (2,0) |

function Hexpm(A)
A = flipdim(A,1);
[b,a] = find(A);
a = a-1;b=b-1;

s = max(size(a));
hold on;
for i=1:s
    Hexp([a(i),b(i)]);
end
end

```

Function 18: HexVP

```

% PVME=HexVP(I,G)
% Given
% I: initial configuration
% G: goal configuration
%
% Find
% PVME: all possible corresponding ME for PV
%
% Notes:
% this function also specifies how many time steps are required
% each row is: [Xpv Ypv Xme Yme T]
%
% Example
% clear all;clc;
% I = [0 0 0 0 0
%      1 0 0 0 0
%      1 1 1 1 1
%      0 0 0 0 0];
% G = [1 1 0 0 0
%      1 0 1 0 0
%      0 1 1 0 0
%      0 0 0 0 0];
% %Double Zero Padding
% I = HexZP(I);I = HexZP(I);
% G = HexZP(G);G = HexZP(G);
% PVME=HexVP(I,G);

function PVME=HexVP(I,G)

[V,E] = FindVE(I,G);
PoV = PV(I,G);
ME = Mobile(E,I);

nme = size(find(ME),1); %check number of available ME
npv = size(find(PoV),1); %check number of available PV

```

```

n = min(nme,npv);           %possible number of connections (ME==>PV)
if (n<1)                   %stop if no connection is possible
    disp('There must be at least one ME and one PV for HexVP function')
    return;
end

PVME = zeros(n,5);        %maximum possible relationship
a=1;
while (n>0)
    [X Y]=find(PoV);
    FME=zeros(size(X,1),3);
    for i=1:npv
        x=X(i);y=Y(i);
        T=HexVPT(I,G,[x,y]);
        FME(i,:)=T(1,:);
    end
    %find the first ME found
    FME3=FME(:,3);
    m=min(FME3);
    r=find (FME3<m+1,1);
    %correspond the PV to ME
    PVME (a,:)= [X(r), Y(r), FME(r,1), FME(r,2), m];
    %eliminate the addressed PV and ME
    PoV(X(r),Y(r))=0;ME(FME(r,1),FME(r,2))=0;
    %update
    nme = size(find(ME),1);
    npv = size(find(PoV),1);
    n = min(nme,npv);
    a=a+1;
end
end

```

Function 19: HexVPT

```

% T = HexVPT(I,G,s,p,ps)
%
% Given
% I: initial configuration (double zero padded)
% G: goal configuration (double zero padded)
% s: the required PV to propagate
% p: if 1 the progress will be plotted
% ps: the plotting speed
%
% Find
% T: the time step required to reach each ME, [x y t]
%
% Notes:
% given a PV, this function will specify the time steps required to reach MEs
%
% Example
% clear all;clc;
% I = [0 0 0 0 0
%      1 0 0 0 0
%      1 1 1 1 1
%      0 0 0 0 0];
%
% G = [1 1 0 0 0
%      1 0 1 0 0
%      0 1 1 0 0
%      0 0 0 0 0];
%
%
% %Double Zero Padding
% I = HexZP(I);I = HexZP(I);
% G = HexZP(G);G = HexZP(G);
% s = [3 3]; %required PV
%
% T = HexVPT(I,G,s,1,0.05)

```



```

function T = HexVPT(varargin)

    I = varargin{1};
    G = varargin{2};
    s = cell2mat(varargin(3));
    p=0;
    ps=0.1;
    if length(varargin) > 3
        p = cell2mat(varargin(4));
    end
    if length(varargin) > 4
        ps = cell2mat(varargin(5));
    end

    xmax=size(I,1);ymax=size(I,2);
    [V,E] = FindVE(I, G);
    ME = Mobile(E,I); %find all ME
    n = size(find(ME),1); %check number of available ME
    %Initiate the time step matrix
    T = zeros(n,3); %each row (x y t):each ME should be filled with a number of time
steps needed
    t=1;
    %stop if no connection is possible
    if (n<1)
        disp('There must be at least one ME and one PV for HexVP function')
        return;
    end

    I=I-ME; %assume MEs as allowable propagation locations

    %Initial propagation matrix (P)
    x=s(1,1);y=s(1,2);
    P = zeros(size(I));P(x,y)=1;
    Found = P; %Mark identified locations
    [XN YN] = find(P); %Locations needs to be propagated

    %just for plotting purposes
    if (p==1)
        figure(1);hold on;axis off;clf;Hexplot(I);Hexplot(P,1,'r');
    end

    %Continue till all time steps are found
    m=1; %the first mobile electron to find its time steps
    MEP = ME+P; %just a mark to check if the propagation is finished
    while (sum(sum(MEP-Found))>0)
        %propagate each node
        NP=P; %propagation matrix at next time step
        for i=1:size(XN,1)
            x=XN(i,1);y=YN(i,1); %set the propagation nodes
            N=HexNeM(x,y); %check the neighbors
            %check each of six neighbors for propagation possibility
            for j=1:6
                xn=N(j,1);yn=N(j,2);
                if (xn>0&&xn<=xmax&&yn>0&&yn<=ymax)
                    if (p==1)%plotting
                        TTT = ze-
ros(size(P));TTT(xn,yn)=1;pause(ps);clf;Hexplot(I);Hexplot(NP,1,'r');Hexplot(ME,1,'g');Hex-
xplot(Found,1,'m');Hexplot(TTT,1,'b');
                    end
                    if (I(xn,yn)==0) %is empty?
                        %has immobile substrate?
                        NP(xn,yn)=HexIMS(I,[xn,yn]);%mark as new propagation node
                        if (ME(xn,yn)==1)
                            T(m,1)=xn;T(m,2)=yn;T(m,3)=t;m=m+1;
                            Found(xn,yn)=1;
                        end
                    end
                end
            end
        end
    end
end
end
end
end

```

```

[XN YN]=find(NP-P); %Locations needs to be propagated
t=t+1;           %increment the time step
P=NP;           %update P
if (t>1000) return; end %just to be in the safe side !!!
end
end

```

Function 20: HexZP

```

% [CZ] = HexZP(C)
%
% Given
% C: a configuration matrix
%
% Find
% CZ: zero padded configuration
%
% Notes:
% It will add zeros to the outer layer of the configuration (C)

function Cz = HexZP(C)
V = zeros(size(C,1),1);
C = [V C V];
H = zeros(1,size(C,2));
Cz = [H;C;H];
end

```

Function 21: Jointplot

```

% Jointplot(A)
%
% Given
% A: Joint Forces (the only close joint for the mobile module)
% Note that all joints are normally closed except for the mobile
% module that has all joints opened except the pivot joint.
% Joint Numbering:
%      2 3
%      1 4
%      6 5
% ex, A = [1 0 2] %second joint of the mobile module (1,0)
%
% Find
% Plot the joints

function Jointplot(A)
hold on;

[Vx,Vy]=Hex([A(1,1),A(1,2)]);
c = A(1,3); %the only closed joint (pivot)
for i=1:6
    if (i==c)
        plot (Vx(i),Vy(i), 'r*');
    else
        plot (Vx(i),Vy(i), 'b*');
    end
end
end

```

Function 22: MDP_Action

```

% a=MDP_Action(I,G,s,p)
%
% Given
% I: initial configuration
% G: goal configuration
% s: state
% p: p = 1, the result will be plotted
%

```

```

% Find
% a: all possible actions
%
% Note:
% Configurations should be provided double zero padded on all sides allowing module move-
ments
%
% Example
% I = [0 0 0 0 0
%      1 0 0 0 0
%      1 1 1 1 1
%      0 0 0 0 0];
%
% G = [1 1 0 0 0
%      1 0 1 0 0
%      0 1 1 0 0
%      0 0 0 0 0];
%
% %Double Zero Padding
% I = HexZP(I); I = HexZP(I);
% G = HexZP(G); G = HexZP(G);
% s = [5,3];
% a = MDP_Action(I,G,s,1)

function a=MDP_Action(varargin)
I = varargin{1};
G = varargin{2};
s = varargin{3};
p = 0;
if length(varargin) > 3
    p = cell2mat(varargin(4));
end

a = zeros(2,2);           %assume no action is possible (neither CW nor CCW)
i=1;                     %look for the first action

S = MDP_State(I,G);      %look for all possible states of I
x = s(1,1);y=s(1,2);

N = HexNeM(x,y);         %find only neighbor modules (Connectivity Constraint)
for j=1:6
    nx = N(j,1);ny=N(j,2);
    if S(nx,ny)==1       %Collision Avoidance Constraint
        a(i,1)=nx;a(i,2)=ny;
        i=i+1;
    end
end

a(i,1)=x;a(i,2)=y;       %One possible action is to stay in its location

%Plotting the result
if p==1
    Grid = ones(size(I)); %display the grid
    S = zeros(size(I));S(x,y)=1; %display the chosen state
    A = zeros(size(I)); %display the available actions
    for l = 1:i-1
        A(a(l,1),a(l,2))=1;
    end
    figure;hold on
    subplot(1,2,1);axis off;title('Module Configuration');Hexpm(Grid);Hexplot(I,1);
    subplot(1,2,2);axis off;title('Available Ac-
tions');Hexpm(Grid);Hexplot(I,1);Hexplot(S,1,'b');Hexplot(A,1,'k');
end

```

Function 23: MDP_NRL

```

% NRL = MDP_NRL(I,G,s1,s2,p)
%

```

```

% Given:
% I: initial configuration
% G: goal configuration
% s1: mobile electron
% s2: potential void
% p: p=1, plotting the result
%
% Find
% NRL: Next Recommended Location
%
% Notes:
% that configurations should be provided double zero padded on all sides allowing module
movements
% if p is not entered the result will not be plotted
%
% Example
% I = [0 0 0 0 0
%      1 0 0 0 0
%      1 1 1 1 1
%      0 0 0 0 0];
% G = [1 1 0 0 0
%      1 0 1 0 0
%      0 1 1 0 0
%      0 0 0 0 0];
% %Double Zero Padding
% I = HexZP(I);I = HexZP(I);
% G = HexZP(G);G = HexZP(G);
% s1 = [5,3];
% s2 = [3,3];
% NRL = MDP_NRL(I,G,s1,s2,1)

function NRL = MDP_NRL(varargin)
I = varargin{1};
G = varargin{2};
s1 = varargin{3};
s2 = varargin{4};
p=0;

if length(varargin) > 4
    p = cell2mat(varargin(5));
end

U = MDP_VI(I,G,s1,s2);           %Find the utility function

% Look for the best action
x = s1(1,1);y=s1(1,2);
Umax = U(x,y);                   %Assume the best action is to stay
NRL = [x y];                      %Assume the best NRL is to stay

N = HexNeM(x,y);                 %find all the neighbors
for j=1:6
    nx = N(j,1);ny=N(j,2);
    if U(nx,ny)>Umax               %If a better action is found
        Umax = U(nx,ny);          %Update the action
        NRL = [nx ny];            %Recommend the next location for ME
    end
end

if p==1
    ME = zeros(size(I));ME(s1(1,1),s1(1,2))=1;
    PV = zeros(size(I));PV(s2(1,1),s2(1,2))=1;
    MA = zeros(size(I));MA(NRL(1,1),NRL(1,2))=1;
    S=MDP_State(I,G);
    Grid = ones(size(I));
    figure;hold on
    subplot(1,2,1);axis off;title('Module Configuration, ME in green, PV in
red');Hexpm(Grid);Hexplot(I,1);Hexplot(ME,1,'g');Hexplot(PV,1,'r');
    subplot(1,2,2);axis off;title('Available States, Next Recommended Location in
green');Hexpm(Grid);Hexplot(I,1);Hexplot(S,1,'m');Hexplot(MA,1,'g');
end

```

Function 24: MDP_Reward

```

% R=MDP_Reward(I,G,s1,s2)
%
% Given
% I: initial configuration
% G: goal configuration
% s1: ME, start point
% s2: PV, end point
%
% Find
% R: reward function
%
% Example
% I = [0 0 0 0 0
%      1 0 0 0 0
%      1 1 1 1 1
%      0 0 0 0 0];
% G = [1 1 0 0 0
%      1 0 1 0 0
%      0 1 1 0 0
%      0 0 0 0 0];
% %Double Zero Padding
% I = HexZP(I);%I = HexZP(I);
% G = HexZP(G);%G = HexZP(G);
% s1 = [4,2];
% s2 = [2,2];

function R=MDP_Reward(I,G,s1,s2)
R = zeros(size(I));
S = MDP_State(I,G);
[V,E] = FindVE(I,G);
ME = Mobile(E,I);

R = R-0.04*S;           %Penalty for staying at their states
R(s2(1,1),s2(1,2))=1; %Reward to reach the goal
ME(s1(1,1),s1(1,2))=0;%exclude this ME from the rest of MEs
R = R-(0.96.*ME);     %Penalty for going close to other mobile electrons
end

% Pot = PV(I,G);
% ME = mobile(E,I);
% Grid = ones(size(I));
% figure;hold on
% subplot(1,3,1);axis off;title('Current Configuration, ME in green');
% Hexpm(Grid);Hexplot(I,1);Hexplot(ME,1,'g');
% subplot(1,3,2);axis off;title('Goal Configuration, PV in green');
% Hexpm(Grid);Hexplot(G,1);Hexplot(Pot,1,'g');
% subplot(1,3,3);axis off;title('Immobile Modules, Available states in magenta');
% Hexpm(Grid);Hexplot(I,1);Hexplot(S,1,'m');

```

Function 25: MDP_State

```

% S=MDP_State(I,G,p)
%
% Given
% I: initial configuration
% G: goal configuration
% p: p=1 the result will be plotted
%
% Find
% all possible states
%
% Notes:
% configurations should be provided with zero padded on all sides allowing module move-
ments
%

```

```

% Example
% I = [0 0 0 0 0
%      1 0 0 0 0
%      1 1 1 1 1
%      0 0 0 0 0];
% G = [1 1 0 0 0
%      1 0 1 0 0
%      0 1 1 0 0
%      0 0 0 0 0];
% %Zero Padding
% I = HexZP(I);
% G = HexZP(G);
% S=MDP_State(I,G,p)

function S=MDP_State(varargin)
I = varargin{1};
G = varargin{2};
p = 0;
if length(varargin) > 2
    p = cell2mat(varargin(3));
end

% We have to consider the location of the current ME as available states
% Therefore, we will remove them from the current configuration
[V,E] = FindVE(I, G);
ME = Mobile(E,I);
I = I - ME;

[X Y] = find(I);           %find all modules in the configuration
s = size(X,1);           %number of modules in the configuration
S = zeros(size(I));      %initially assume there are no states

for i=1:s
    x = X(i);y = Y(i);    %Start with one node, check if others are connected to it
    N = HexNeM(x,y);     %find neighbor modules
    for k=1:6             %check if this neighbor can be a state
        nx = N(k,1);ny = N(k,2);
        if I(nx,ny)==0
            S(nx,ny)=1;
        end
    end
end
end

% Test if the location can be filled (Collision Avoidance)
% If there are more than 3 neighbors for an empty location
% that location can not be considered as a state since it
% can not be filled
[X Y] = find(S);
s=size(X,1);
for i = 1 : s
    [N] = HexNeM(X(i),Y(i)); %find the neighbors
    n=0; %number of occupied neighbors
    for j = 1:6
        if (I(N(j,1),N(j,2))==1)
            n=n+1;
        end
    end
    if n>3
        S(X(i),Y(i))=0;
    end
end
end

if p==1
    Grid = ones(size(I));
    figure;hold on
    subplot(1,2,1);axis off;title('Module Configuration, ME in
green');Hexpm(Grid);Hexplot(I,1);Hexplot(ME,1,'g');
    subplot(1,2,2);axis off;title('Available
States');Hexpm(Grid);Hexplot(I,1);Hexplot(S,1,'m');
end
end

```

end

Function 26: MDP_VI

```

% U=MDP_VI(I,G,s1,s2,g,p)
%
% Given
% I: Initial Configuration
% G: Goal Configuration
% s1: Mobile Electron
% s2: Potential Void
% g: Discount Factor
% p: Plotting The Result
%
% Find
% U: Utility Function, using Value Iteration (VI) technique
%
% Notes:
% configurations should be provided double zero padded on all sides allowing module move-
ments
% if g is not entered the default value of 0.7 is used
% if p is not entered the result will not be plotted
% if p = 1, the result will be plotted
%
% Example
% clear all;clc
% I = [0 0 0 0 0
%      1 0 0 0 0
%      1 1 1 1 1
%      0 0 0 0 0];
% G = [1 1 0 0 0
%      1 0 1 0 0
%      0 1 1 0 0
%      0 0 0 0 0];
% %Double Zero Padding
% I = HexZP(I);I = HexZP(I);
% G = HexZP(G);G = HexZP(G);
% ME = [5,3];
% PV = [3,3];
% U = MDP_VI(I,G,ME,PV,0.7,1)

function U=MDP_VI(varargin)
I = varargin{1};
G = varargin{2};
ME = varargin{3};
PV = varargin{4};
g = 0.7;
p = 0;
if length(varargin) > 4
    g = cell2mat(varargin(5));
end
if length(varargin) > 5
    p = cell2mat(varargin(6));
end

S = MDP_State(I,G);
R = MDP_Reward(I,G,ME,PV);
U = zeros(size(I));           %Initial Values
[X Y] = find(S);

i=1;D=1;
while (D(i)>0.001)
    U_Old = U;
    for j=1:size(X,1)
        x=X(j);y=Y(j);
        a = MDP_Action(I,G,[x y]);
        if (size(a,1)>1), A = [U(a(1,1),a(1,2)) U(a(2,1),a(2,2))]; end
        if (size(a,1)>2), A = [U(a(1,1),a(1,2)) U(a(2,1),a(2,2)) U(a(3,1),a(3,2))];
    end
end

```

```

        U(x,y)=R(x,y)+g*max(A);
    end

    i = i+1;
    D(i) = abs(max(max(U - U_Old))); %Number of Iterations %Converges
    if (i>500) %Stop After 500 Iteration, even if error is still
big
        D(i)=0;
    end
end

% Normalize the values
U = U/max(max(U));

if p==1
    %Remove the first element (Was set to 1 manually)
    D = D(2:size(D,2));
    plot(D);xlabel('Number of Iteration');ylabel('Maximum Error');title('State Conver-
gence');
end
end

```

Function 27: Mobile

```

% ME = Mobile(E,S)
%
% Given
% E: an electron
% S: a configuration state
%
% Find
% ME: mobile electrons

function ME = Mobile(E,S)
    ME = zeros(size(E));
    % Test Mobility (Collision Avoidance)
    [X Y] = find(E);
    s=size(X,1);
    for i = 1 : s
        [N] = HexNeM(X(i),Y(i)); %find the neighbors
        n=0; %number of occupied neighbors
        for j = 1:6
            if (S(N(j,1),N(j,2))==1)
                n=n+1;
            end
        end
        if n<4
            ME(X(i),Y(i))=1;
        end
    end

    % Test Connectivity
    [X Y] = find(ME);
    s=size(X,1);
    for i = 1 : s
        I_test = S;
        I_test(X(i),Y(i))=0;
        C = HexCGT(I_test);
        if C~=1
            ME(X(i),Y(i))=0;
        end
    end
end

```

Function 28: PV

```

% PV = PV(I,G)
%

```



```

% Given:
% I: Initial Configuration
% G: Goal Configuration
%
% Find
% PV: Potential voids

function PV = PV(I,G)
[V,E] = FindVE(I, G);
ME = Mobile(E,I);
C = I-ME;          %remove mobile electrons to find immobile modules

[X Y] = find(V);   %find all voids
PV = zeros(size(V)); %initially assume there is no PV

s = size(X,1);
for i = 1 : s      %check all voids
    N = HexNeM(X(i),Y(i));
    for j = 1:6    %check their 6 sides
        if C(N(j,1),N(j,2))==1
            PV(X(i),Y(i))=1;
        end
    end
end
end
end

```

Function 29: Sideplot

```

% Sideplot (A)
%
% Given
% A: location + side forces (Check below notes)
%
% Find
% Plots side forces
%
% Notes:
%   Sideplot(A), plots the side forces of A
%   A: Side Forces (Energized magnets: Mobile and neighbor)
%   Note that all magnets are normally off
%   Mobile modules have -ve field (both sides of the fixed joint, electron)
%   Neighbor module should have a +ve (attraction) and -ve (repulsion)
%   Side 1 is between joints 1,2 and so on.
%   ex,  A = [2 0 1 2; %sides 1 and 2 of mobile (2,0) are -ve
%            2 1 3 4] %side 3 of neighbor (2,1) is +ve and 4 -ve

function Sideplot(A)
hold on;

% Plot the sides
[Vx,Vy]=Hex([A(1,1),A(1,2)]);
a = A(1,3);b = A(1,3)+1;if (b==7) b=1; end
x = (Vx(a)+Vx(b))/2;y = (Vy(a)+Vy(b))/2;
plot (x,y, 'b^', 'MarkerSize',5, 'LineWidth',7);
a = A(1,4);b = A(1,4)+1;if (b==7) b=1; end
x = (Vx(a)+Vx(b))/2;y = (Vy(a)+Vy(b))/2;
plot (x,y, 'b^', 'MarkerSize',5, 'LineWidth',7);
[Vx,Vy]=Hex([A(2,1),A(2,2)]);
a = A(2,3);b = A(2,3)+1;if (b==7) b=1; end
x = (Vx(a)+Vx(b))/2;y = (Vy(a)+Vy(b))/2;
plot (x,y, 'r+', 'MarkerSize',15, 'LineWidth',5);
a = A(2,4);b = A(2,4)+1;if (b==7) b=1; end
x = (Vx(a)+Vx(b))/2;y = (Vy(a)+Vy(b))/2;
plot (x,y, 'b^', 'MarkerSize',5, 'LineWidth',7);
end

```

Test 1: TestAll_P01

```

% This code is an example to illustrate parallel reconfiguration planning

clear all;clc
% Initial Configuration
I = [0 0 0 0 0
     1 0 0 0 0
     1 1 1 1 1];

% Goal Configuration
G = [1 1 0 0 0
     1 0 1 0 0
     0 1 1 0 0];
%Double Zero Padding
I = HexZP(I);I = HexZP(I);I = HexZP(I);
G = HexZP(G);G = HexZP(G);G = HexZP(G);

%Simulation Figure
Grid = ones(size(I));
scrsz = get(0,'ScreenSize');
figure('Name','Simulation Window','NumberTitle','off','Position',[1 1 scrsz(3)
scrsz(4)]);hold on
subplot(1,2,1);axis off;title('Initial Configuration');Hexpm(Grid);Hexplot(I,1);
subplot(1,2,2);axis off;title('Goal Configuration');Hexpm(Grid);Hexplot(G,1);

pause();[V E] = FindVE(I,G); %Find all voids and electrons
subplot(1,2,1);axis off;title('Electrons');Hexpm(Grid);Hexplot(E,1,'g');
subplot(1,2,2);axis off;title('Voids');Hexpm(Grid);Hexplot(V,1,'r');

pause(1);
C = I; %initialize the current configuration
t=0;
while (isequal(C,G)<1)
    t=t+1;
    %Start Reconfiguration
    [Joint, Side, S]=HexAll_P(C,G); %Move from current to goal
    s = size (Joint,1); %Number of parallel movements

    %Plotting the motion
    St=MDP_State(C,G);
    clf;
    subplot(1,2,2);axis off;title(['Configuration at time step:', int2str(t)]);
    Hexpm(Grid);Hexplot(C,1);
    subplot(1,2,1);axis off;title('Module Configuration, Available States and Actua-
tion');
    Hexpm(Grid);Hexplot(C,1);Hexplot(St,1,'m');

    for i=1:s %Move the MEs
        Fj=Joint(i,:);
        Fs=[Side(i,[1 2 3 4]);Side(i,[5 6 7 8])];
        s1 = S(i,[1 2]); %current location
        s2 = S(i,[3 4]); %next location
        s3 = S(i,[5 6]); %desired location
        if (sum(sum(Fs))==0) || (sum(Fj)==0)
            disp('There is no more possible actuation');break;
        end
        %Plot the ME and its PV pluse forces
        ME = zeros(size(I));ME(s1(1,1),s1(1,2))=1;
        PoV = zeros(size(I));PoV(s3(1,1),s3(1,2))=1;
        Hexplot(ME,1,'g');Hexplot(PoV,1,'r');Jointplot(Fj);Sideplot(Fs);
        C = Hexmove(C,Fj,Fs,0); %Update the configuration
    end
    pause(0.5)
end
t=t+1;St=MDP_State(C,G);clf;
subplot(1,2,1);axis off;title('Module Configuration, Available States');
Hexpm(Grid);Hexplot(C,1);Hexplot(St,1,'m');
subplot(1,2,2);axis off;title(['Configuration at time step:', int2str(t)]);

```

```
Hexpm(Grid);Hexplot(C,1);
```

Test 2: Test_All_P02

```
% This code is an example to illustrate parallel reconfiguration planning

clear all;clc
% Initial Configuration
I = [1 1 0 0 0
     1 0 1 0 0
     0 1 1 0 0];

% Goal Configuration
G = [0 0 0 0 0
     1 0 0 0 0
     1 1 1 1 1];

%Double Zero Padding
I = HexZP(I);I = HexZP(I);I = HexZP(I);
G = HexZP(G);G = HexZP(G);G = HexZP(G);

%Simulation Figure
Grid = ones(size(I));
scrsz = get(0,'ScreenSize');
%figure('Name','Simulation Window','NumberTitle','off','Position',[1 1 scrsz(3)
scrsz(4)]);hold on
subplot(1,2,1);axis off;title('Initial Configuration');Hexpm(Grid);Hexplot(I,1);
subplot(1,2,2);axis off;title('Goal Configuration');Hexpm(Grid);Hexplot(G,1);

pause();[V E] = FindVE(I,G); %Find all voids and electrons
subplot(1,2,1);axis off;title('Electrons');Hexpm(Grid);Hexplot(E,1,'g');
subplot(1,2,2);axis off;title('Voids');Hexpm(Grid);Hexplot(V,1,'r');

pause(1);
C = I; %initialize the current configuration
t=0;
while (isequal(C,G)<1)
    t=t+1;
    %Start Reconfiguration
    [Joint, Side, S]=HexAll_P(C,G); %Move from current to goal
    s = size(Joint,1); %Number of parallel movements

    %Plotting the motion
    St=MDP_State(C,G);
    clf;
    subplot(1,2,2);axis off;title(['Configuration at time step:', int2str(t)]);
    Hexpm(Grid);Hexplot(C,1);
    subplot(1,2,1);axis off;title('Module Configuration, Available States and Actua-
tion');
    Hexpm(Grid);Hexplot(C,1);Hexplot(St,1,'m');

    for i=1:s %Move the MEs
        Fj=Joint(i,:);
        Fs=[Side(i,[1 2 3 4]);Side(i,[5 6 7 8])];
        s1 = S(i,[1 2]); %current location
        s2 = S(i,[3 4]); %next location
        s3 = S(i,[5 6]); %desired location
        if (sum(sum(Fs))==0) || (sum(Fj)==0)
            disp('There is no more possible actuation');break;
        end
        %Plot the ME and its PV pluse forces
        ME = zeros(size(I));ME(s1(1,1),s1(1,2))=1;
        PoV = zeros(size(I));PoV(s3(1,1),s3(1,2))=1;
        Hexplot(ME,1,'g');Hexplot(PoV,1,'r');Jointplot(Fj);Sideplot(Fs);
        C = Hexmove(C,Fj,Fs,0); %Update the configuration
    end
    pause(0.5)
end
t=t+1;St=MDP_State(C,G);clf;
```

```
subplot(1,2,1);axis off;title('Module Configuration, Available States');
Hexpm(Grid);Hexplot(C,1);Hexplot(St,1,'m');
subplot(1,2,2);axis off;title(['Configuration at time step:', int2str(t)]);
Hexpm(Grid);Hexplot(C,1);
```

Test 3: Test_All_S01

```
% This code is an example to illustrate serial reconfiguration planning

clear all;clc
% Initial Configuration
I = [0 0 0 0 0
     1 0 0 0 0
     1 1 1 1 1];

% Goal Configuration
G = [1 1 0 0 0
     1 0 1 0 0
     0 1 1 0 0];
%Double Zero Padding
I = HexZP(I);I = HexZP(I);I = HexZP(I);
G = HexZP(G);G = HexZP(G);G = HexZP(G);

%Simulation Figure
Grid = ones(size(I));
scrsz = get(0,'ScreenSize');
figure('Name','Simulation Window','NumberTitle','off','Position',[1 1 scrsz(3)
scrsz(4)]);hold on
subplot(1,2,1);axis off;title('Initial Configuration');Hexpm(Grid);Hexplot(I,1);
subplot(1,2,2);axis off;title('Goal Configuration');Hexpm(Grid);Hexplot(G,1);

pause();[V E] = FindVE(I,G); %Find all voids and electrons
subplot(1,2,1);axis off;title('Electrons');Hexpm(Grid);Hexplot(E,1,'g');
subplot(1,2,2);axis off;title('Voids');Hexpm(Grid);Hexplot(V,1,'r');

pause(1);
C = I; %initialize the current configuration
t=0;
while (isequal(C,G)<1)
    t=t+1;
    %Start Reconfiguration
    [Joint, Side, S]=HexAll_S(C,G); %Move from current to goal

    %Plotting the motion
    St=MDP_State(C,G);
    clf;
    subplot(1,2,2);axis off;title(['Configuration at time step:', int2str(t)]);
    Hexpm(Grid);Hexplot(C,1);
    subplot(1,2,1);axis off;title('Module Configuration, Available States and Actua-
tion');
    Hexpm(Grid);Hexplot(C,1);Hexplot(St,1,'m');

    Fj=Joint(1,:);
    Fs=[Side(1,[1 2 3 4]);Side(1,[5 6 7 8])];
    s1 = S(1,[1 2]); %current location
    s2 = S(1,[3 4]); %next location
    s3 = S(1,[5 6]); %desired location
    if (sum(sum(Fs))==0)|| (sum(Fj)==0)
        disp('There is no more possible actuation');break;
    end
    %Plot the ME and its PV pluse forces
    ME = zeros(size(I));ME(s1(1,1),s1(1,2))=1;
    PoV = zeros(size(I));PoV(s3(1,1),s3(1,2))=1;
    Hexplot(ME,1,'g');Hexplot(PoV,1,'r');Jointplot(Fj);Sideplot(Fs);
    C = Hexmove(C,Fj,Fs,0); %Update the configuration
    pause(10)
end
t=t+1;St=MDP_State(C,G);clf;
subplot(1,2,1);axis off;title('Module Configuration, Available States');
```

```
Hexpm(Grid);Hexplot(C,1);Hexplot(St,1,'m');
subplot(1,2,2);axis off;title(['Configuration at time step:', int2str(t)]);
Hexpm(Grid);Hexplot(C,1);
```

Test 4: Test_All_S02

```
% This code is an example to illustrate serial reconfiguration planning

clear all;clc
% Initial Configuration
I = [1 1 1 1 0
     1 1 1 1 0
     1 1 1 1 0
     0 0 0 0 0
     0 0 0 0 0];

% Goal Configuration
G = [0 0 0 0 0
     1 1 1 1 0
     1 1 1 1 0
     1 0 0 1 0
     1 0 0 1 0];

%Double Zero Padding
I = HexZP(I);I = HexZP(I);I = HexZP(I);
G = HexZP(G);G = HexZP(G);G = HexZP(G);

%Simulation Figure
Grid = ones(size(I));
scrsz = get(0,'ScreenSize');
figure('Name','Simulation Window','NumberTitle','off','Position',[1 1 scrsz(3)
scrsz(4)]);hold on
subplot(1,2,1);axis off;title('Initial Configuration');Hexpm(Grid);Hexplot(I,1);
subplot(1,2,2);axis off;title('Goal Configuration');Hexpm(Grid);Hexplot(G,1);

pause();[V E] = FindVE(I,G); %Find all voids and electrons
subplot(1,2,1);axis off;title('Electrons');Hexpm(Grid);Hexplot(E,1,'g');
subplot(1,2,2);axis off;title('Voids');Hexpm(Grid);Hexplot(V,1,'r');

pause(10);
C = I; %initialize the current configuration
t=0;
while (isequal(C,G)<1)
    t=t+1;
    %Start Reconfiguration
    [Joint, Side, S]=HexAll_S(C,G); %Move from current to goal

    %Plotting the motion
    St=MDP_State(C,G);
    clf;
    subplot(1,2,2);axis off;title(['Configuration at time step:', int2str(t)]);
    Hexpm(Grid);Hexplot(C,1);
    subplot(1,2,1);axis off;title('Module Configuration, Available States and Actua-
tion');
    Hexpm(Grid);Hexplot(C,1);Hexplot(St,1,'m');

    Fj=Joint(1,:);
    Fs=[Side(1,[1 2 3 4]);Side(1,[5 6 7 8])];
    s1 = S(1,[1 2]); %current location
    s2 = S(1,[3 4]); %next location
    s3 = S(1,[5 6]); %desired location
    if (sum(sum(Fs))==0) || (sum(Fj)==0)
        disp('There is no more possible actuation');break;
    end
    %Plot the ME and its PV pluse forces
    ME = zeros(size(I));ME(s1(1,1),s1(1,2))=1;
    PoV = zeros(size(I));PoV(s3(1,1),s3(1,2))=1;
    Hexplot(ME,1,'g');Hexplot(PoV,1,'r');Jointplot(Fj);Sideplot(Fs);
    C = Hexmove(C,Fj,Fs,0); %Update the configuration
```

```

    pause(1)
end
t=t+1;St=MDP_State(C,G);clf;
subplot(1,2,1);axis off;title('Module Configuration, Available States');
Hexpm(Grid);Hexplot(C,1);Hexplot(St,1,'m');
subplot(1,2,2);axis off;title(['Configuration at time step:', int2str(t)]);
Hexpm(Grid);Hexplot(C,1);

```

Test 5: Test_CG

```

% This code is an example to illustrate the connectivity constraint test

clear all;clc;close all;
% Always keep zeros around the matrices to allow module movement
I = [ 0 0 0 0 0 0 0 0
      0 1 1 1 1 0 0 0
      0 1 0 1 0 0 1 0
      0 1 0 1 0 1 1 0
      0 0 1 1 1 1 1 0
      0 0 0 0 0 0 0 0];

%Plot the connectivity graph for the current configuration
HexCG(I)
%Let's do the connectivity test for all modules (assuming all modules to be electrons)
E = I;
%Find mobile electrons
ME = Mobile(E,I);

Grid = ones(size(I));
figure;axis off;hold on;Hexpm(Grid);Hexplot(I,1,'r');Hexplot(ME,1,'g');

```

Test 6: Test_Constraints1

```

% Example of a collision avoidance constraint

% Background Grid
Grid = [1 0 0 0 0 0 0;
        1 1 1 0 0 0 0;
        1 1 1 1 1 0 0;
        1 1 1 1 1 1;
        1 1 1 1 1 1;
        1 1 1 1 1 1;
        0 1 1 1 1 1;
        0 0 0 1 1 1;
        0 0 0 0 0 1;
        0 0 0 0 0 0];
Hexpm(Grid);

% Immobile Config
IC = [0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 1 1 0 0 0;
      0 0 1 0 1 0 0;
      0 0 0 1 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0];
Hexplot(IC,1);

% Fixed Module
[Vx,Vy] = Hex([2,4]);fill (Vx,Vy,'b');

% Mobile Module
[Vx,Vy] = Hex([4,5]);fill (Vx,Vy,'g');

% Plot properties

```

```
axis off;
title('Example of a constrained motion');
```

Test 7: Test_Constraints2

```
% Example of a connectivity constraint

% Background Grid
Grid = [1 0 0 0 0 0 0;
        1 1 1 0 0 0 0;
        1 1 1 1 1 0 0;
        1 1 1 1 1 1 1;
        1 1 1 1 1 1 1;
        1 1 1 1 1 1 1;
        0 1 1 1 1 1 1;
        0 0 0 1 1 1 1;
        0 0 0 0 0 1 1;
        0 0 0 0 0 0 0;];
Hexpm(Grid);

% Immobile Config
IC = [0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 1 1 0 0 0;
      0 0 1 0 1 0 0;
      0 0 0 1 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;];
Hexplot(IC,1);

% Fixed Module
[Vx,Vy] = Hex([2,4]);fill (Vx,Vy,'b');

% Mobile Module
[Vx,Vy] = Hex([2,6]);fill (Vx,Vy,'g');

% Plot properties
axis off;
title('Example of a constrained motion');
```

Test 8: Test_Immobile

```
% Example of an Immobile Configuration

% Background Grid
Grid = [1 0 0 0 0 0 0;
        1 1 1 0 0 0 0;
        1 1 1 1 1 0 0;
        1 1 1 1 1 1 1;
        1 1 1 1 1 1 1;
        1 1 1 1 1 1 1;
        1 1 1 1 1 1 1;
        1 1 1 1 1 1 1;
        1 1 1 1 1 1 1;
        1 1 1 1 1 1 1;
        1 1 1 1 1 1 1;
        0 1 1 1 1 1 1;
        0 0 0 1 1 1 1;
        0 0 0 0 0 1 1;
        0 0 0 0 0 0 0;];
Hexpm(Grid);

% Immobile Config
IC = [0 0 0 0 0 0 0;
```

```

0 0 0 0 0 0 0;
0 0 0 0 0 0 0;
0 1 1 1 0 0 0;
0 1 0 0 1 0 0;
0 0 1 1 0 1 0;
0 0 0 0 0 1 0;
0 0 0 1 1 1 0;
0 0 0 1 0 0 0;
0 0 0 1 0 0 0;
0 0 0 1 0 0 0;
0 0 0 0 0 0 0;
0 0 0 0 0 0 0;
0 0 0 0 0 0 0;]
Hexplot(IC,1);

% Fixed Module
[Vx,Vy] = Hex([3,3]);fill (Vx,Vy,'b');

% Plot properties
axis off;
title('Example of an immobile configuration');

```

Test 9: Test_IR_Simulation

```

% Illustrate the modulated IR transceiver signal

clear all;clc
a=0;
data=[1 0 1 0 1 1 0 1 0 0 1 0 1 0 1 1 1 0 1 1 0];
for i=1:20001 %Move the MEs
    if a==0
        P(i)=0;a=1;
    else
        P(i)=1;a=0;
    end
    I = round(i/1000);
    D(i) = data(I+1);
    S(i) = P(i).*D(i);
end

figure;
subplot(3,1,1);plot(P);title('38kHz Pulse');axis([0 20000 -1 2]);axis off;
subplot(3,1,2);plot(D);title('Serial Data');axis([0 20000 -1 2]);axis off;
subplot(3,1,3);plot(S);title('Modulated Pulse Being Transmitted');axis([0 20000 -1 2]);axis off;

```

Test 10: Test_Layer5

```

% To draw a module an its neighbors required for actuation: layer 5

clear all;clc;
C = [0 1 0 0 0
     1 1 1 1 0
     0 1 1 1 0
     0 1 1 1 1
     0 0 0 1 0];
M = zeros(size(C));
M(3,3)=1;
figure;hold on;axis off;
Hexpm(C);Hexplot(M,1);

```

Test 11: Test_Localization

```

% Localization example discussed in thesis

```



```

% Background Grid
Grid = [1 0 0 0 0 0 0 0;
        1 1 1 0 0 0 0 0;
        1 1 1 1 1 0 0 0;
        1 1 1 1 1 1 1 0;
        0 1 1 1 1 1 1 1;
        0 0 0 1 1 1 1 1;
        0 0 0 0 0 1 1 1;
        0 0 0 0 0 0 1 1];
Hexpm(Grid);

I = [0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0;
     0 0 1 0 0 0 0 0;
     0 0 1 1 1 1 1 0;
     0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0];

Hexplot (I,1);
% Fixed Module
[Vx,Vy] = Hex([3,3]);fill (Vx,Vy,'b');

% Plot properties
axis off;
%title('Localization');

```

Test 12: Test_MDP

```
% Test of MDP: Next Recommended Location
```

```

clear all;clc
I = [0 0 0 0 0
     1 0 0 0 0
     1 1 1 1 1
     0 0 0 0 0];

G = [1 1 0 0 0
     1 0 1 0 0
     0 1 1 0 0
     0 0 0 0 0];
%Double Zero Padding
I = HexZP(I);I = HexZP(I);
G = HexZP(G);G = HexZP(G);

s1 = [5,3];
s2 = [3,3];

while sum(abs(s1-s2))>0
    s1 = MDP_NRL(I,G,s1,s2,1);
end

```

Test 13: Test_MDP_Convergence

```
% This code is developed to evaluate MDP Convergence
```

```

clear all;clc;
g = 0:0.001:1;
figure;hold on;

c = 0.1; %e/Rmax
N = log10(2./(c.*(1-g)))./log10(1./g);
plot (g,N,'b')

c = 0.01; %e/Rmax
N = log10(2./(c.*(1-g)))./log10(1./g);

```

```

plot (g,N,'c')

c = 0.001; %e/Rmax
N = log10(2./(c.*(1-g)))./log10(1./g);
plot (g,N,'g')

c = 0.0001; %e/Rmax
N = log10(2./(c.*(1-g)))./log10(1./g);
plot (g,N,'r')

axis([0 1 0 1000]);xlabel('Discount Factor');ylabel('Number of Iteration');

```

Test 14: Test_Simulator

```

% An example to test the simulator

clear all;clc;close all;
% Always keep zeros around the matrices to allow module movement
I = [ 0 0 0 0 0 0
      0 0 0 0 0 0
      1 1 1 1 1 1];

G = [0 0 0 1 1 0
      0 0 0 1 0 1
      0 0 0 0 1 1];

figure(1);clf;
Grid = ones(size(I));
subplot(1,2,1);axis off;hold on;title('Initial State (Configuration) of the Modules');Hexpm(Grid);Hexplot(I,1);
subplot(1,2,2);axis off;hold on;title('Desired Goal State (Configuration) of the Modules');Hexpm(Grid);Hexplot(G,1);pause();

% First Move
I = I;
Fj = [0 0 3];
Fs = [0 0 2 3;
      1 0 1 6];
G = Hexmove(I,Fj,Fs,1);
% Next Move
I = G;
Fj = [0 1 4];
Fs = [0 1 3 4;
      1 0 2 1];
G = Hexmove(I,Fj,Fs,1);
% Next Move
I = G;
Fj = [1 1 4];
Fs = [1 1 3 4;
      2 0 2 1];
G = Hexmove(I,Fj,Fs,1);
% Next Move
I = G;
Fj = [2 1 4];
Fs = [2 1 3 4;
      3 0 2 1];
G = Hexmove(I,Fj,Fs,1);
% Next Move
I = G;
Fj = [3 1 4];
Fs = [3 1 3 4;
      4 0 2 1];
G = Hexmove(I,Fj,Fs,1);
% Next Move
I = G;
Fj = [4 1 4];
Fs = [4 1 3 4;

```

```

        5 0 2 1];
G = Hexmove(I,Fj,Fs,1);
% Next Move
I = G;
Fj = [1 0 3];
Fs = [1 0 2 3;
      2 0 1 6];
G = Hexmove(I,Fj,Fs,1);
% Next Move
I = G;
Fj = [1 1 4];
Fs = [1 1 3 4;
      2 0 2 1];
G = Hexmove(I,Fj,Fs,1);
% Next Move
I = G;
Fj = [2 1 4];
Fs = [2 1 3 4;
      3 0 2 1];
G = Hexmove(I,Fj,Fs,1);
% Next Move
I = G;
Fj = [3 1 4];
Fs = [3 1 3 4;
      4 0 2 1];
G = Hexmove(I,Fj,Fs,1);
% Next Move
I = G;
Fj = [4 1 3];
Fs = [4 1 2 3;
      5 1 1 6];
G = Hexmove(I,Fj,Fs,1);
% Next Move
I = G;
Fj = [2 0 3];
Fs = [2 0 2 3;
      3 0 1 6];
G = Hexmove(I,Fj,Fs,1);
% Next Move
I = G;
Fj = [2 1 4];
Fs = [2 1 3 4;
      3 0 2 1];
G = Hexmove(I,Fj,Fs,1);
% Next Move
I = G;
Fj = [3 0 2];
Fs = [3 0 1 2;
      3 1 6 5];
G = Hexmove(I,Fj,Fs,1);
% Next Move
I = G;
Fj = [2 1 3];
Fs = [2 1 2 3;
      3 1 1 6];
G = Hexmove(I,Fj,Fs,1);
% Next Move
I = G;
Fj = [2 2 4];
Fs = [2 2 3 4;
      3 1 2 1];
G = Hexmove(I,Fj,Fs,1);

```

Test 15: Test_VP

```

% Void Propagation example discussed in thesis
% The actual VP algorithm is accomplished in the lower levels
% through the use of microcontrollers installed on the modules.

```

```
clear all;clc;
I = [0 0 0 0 0 0 0 0
     0 0 0 0 0 0 0 0
     0 0 1 0 0 0 0 0
     0 0 1 1 1 1 1 0
     0 0 0 0 0 0 0 0];

G = [0 0 0 0 0 0 0 0
     0 0 1 1 0 0 0 0
     0 0 1 0 1 0 0 0
     0 0 0 1 1 0 0 0
     0 0 0 0 0 0 0 0];

[V E] = FindVE(I,G);    %Find all voids and electrons
PV = PV(I,G);          %Find potential voids
ME = Mobile(E,I);      %Find mobile electrons

% plot the results
Grid = ones(size(I));
clf;
% subplot (2,2,1);axis off;Hexpm(Grid);Hexplot(I,1);title('Current Configuration');
% subplot (2,2,2);axis off;Hexpm(Grid);Hexplot(G,1);title('Goal Configuration');
% subplot (2,2,3);axis off;Hexpm(Grid);Hexplot(E,1);title('Electrons - Mobile electrons
in green');Hexplot(ME,1,'g');
% subplot (2,2,4);axis off;Hexpm(Grid);Hexplot(V,1);title('Voids - Potential Voids in
green');Hexplot(PV,1,'g');
% plot the connectivity graph
%HexCG(I);
clf;Hexpm(Grid);axis off;Hexplot(I,1);Hexplot(ME,1,'g');Hexplot(PV,1,'c');
```

VITA

Hossein Sadjadi was born on August 14, 1980, in Tehran, Iran. Having finished the primary and secondary schools in Iran, he moved to the United Arab Emirates (UAE) in 2001 where he started a Bachelor of Science degree in electrical and electronics engineering at the American University of Sharjah (AUS). Mr. Sadjadi was ranked the first in the final Comprehensive Assessment Examination (CAE) of the department and graduated with honor (Cum Laude) in 2005. He then joined Odasco Automation where he acted as a senior automation engineer for two years and managed to successfully implement several industrial projects.

Mr. Sadjadi began a Master of Science degree in Mechatronics at AUS in 2006 and awarded the Master of Science degree in 2009 with a 4.0 GPA. Since November, 2006 he has been acting as a senior Mechatronics laboratory specialist at AUS.