TESTBED FOR ROBOTIC TEAM-ASSISTED REPAIR:
HEXAGONAL DISTRIBUTED MODULAR ROBOT II

by

Yunuscan Sevimli

A masters essay submitted to Johns Hopkins University in conformity with the requirements for the degree of Master of Science

Baltimore, Maryland

October, 2015

© 2015 Yunuscan Sevimli
All Rights Reserved

ABSTRACT

Advancements in robotics have increasingly taken robots out of controlled environments to perform in hostile conditions that increase the chance of a system fault. Currently most solutions require human intervention or, in the case of homogenous modular systems, discarding the faulty agents which in return degrade the overall capabilities of the system. There is an emerging need for a reliable system that can perform its duties regardless of the changing mission requirements and effectively alleviate a range of possible fault states. This essay presents the second version of the Hexagonal Distributed Modular Robot (Hex-DMR II) capable of autonomous team-assisted repair to address this issue. The Hex-DMR II system is composed of reconfigurable modular agents that encapsulate distinct capabilities (and thus the related fault states) within modules and can replace these modules to meet the changing needs of the mission as well as to maintain functionality despite a faulty state. The essay introduces the design and capabilities of the system, presents the methods of team-assisted repair and discusses the results of module replacement maneuvers undertaken by the system. Overall Hex-DMR II demonstrates the working principles of robotic team-assisted repair through successful experimental trials.

Advisor: Dr. Gregory Chirikjian

Reader: Dr. Russell Taylor

PREFACE

I would like to thank,

Dr. Gregory Chirikjian, my research advisor and PI of the Hex-DMR project, for his generous financial support and guidance that I owe my expected MS degree in Mechanical Engineering,

Dr. Nathan Scott, my academic advisor and mentor, for always setting the bar higher for me as a student as well as a person and for being the engineer I one day aspire to be,

Joshua Davis and Kendal Ackerman, PhD candidates and fellow research team members, for their continuous support and guidance inside and outside the research laboratory,

Ada, Arzu and Kemal Sevimli, my family, whose resilience has inspired me in my every step as I strive to be worthy of having them. *İyi ki varsınız.*

TABLE OF CONTENTS

LIST OF FIGURES vi

1. INTRODUCTION 1
- **1.1.** COOPERATIVE MULTI-AGENT SYSTEMS (CMS) 1
- **1.2.** HOMOGENOUS AND HETEROGENOUS CMS 2
- **1.3.** TEAM-ASSISTED REPAIR ON HETEROGENOUS CMS 3

2. DESIGN 5
- **2.1.** MATERIAL SELECTION AND MANUFACTURING 5
- **2.2.** EVOLUTION OF DESIGN 7
- **2.3.** SYSTEM OVERVIEW 8
  - **2.3.1.** MODULARITY 9
  - **2.3.2.** MODULAR REDUNDANCY AND CUSTOMIZABILITY 10
  - **2.3.3.** SIX-FOLD ROTATIONAL SYMMETRY 11
  - **2.3.4.** DOCKING MECHANISM 13
  - **2.3.5.** MOBILITY 16
  - **2.3.6.** SENSING 16
  - **2.3.7.** COMMUNICATION 16
  - **2.3.8.** CONTROL 17
- **2.4.** MODULES 19
  - **2.4.1.** CENTRAL HUB (CH) 19
  - **2.4.2.** DRIVE MODULE (DRM) 21
  - **2.4.3.** MANIPULATOR MODULE (MAM) 22
  - **2.4.4.** POWER MODULE (POM) 24

2.4.5. CONTROL MODULE (COM) 25

2.4.6. CAMERA MODULE (CAM) 26

2.4.7. ELEVATOR MODULE (ELM) 28

3. METHODS 32

3.1. ROBOTS IN FAULT STATES (RIF) 32

3.2. STEPS OF TEAM-ASSISTED REPAIR 32

4. TEAM-ASSISTED REPAIR MANEUVER DEMONSTRATIONS 35

4.1. AUTONOMOUS MODULE EXTRACTION AND INSERTION 35

4.1.1. PROCEDURES 36

4.1.2. RESULTS 39

4.2. REMOTELY-ASSISTED, MULTI-LEVEL MODULE REPLACEMENT 40

4.2.1. PROCEDURE 40

4.2.2. RESULTS 43

5. CONCLUSIONS AND FUTURE WORK 43

APPENDICES 46

APPENDIX 1 - CODE FOR AUTONOMOUS MODULE EXTRACTION 46

APPENDIX 2 - SUPPORTING CODE FOR AUTONOMOUS OPERATION 53

APPENDIX 3 - CODE FOR AUTONOMOUS MODULE INSERTION 57

APPENDIX 4 - CODE FOR TELEOPERATED OPERATION 65

BIBLIOGRAPHY 69

CURRICULUM VITAE 72

LIST OF FIGURES

Figure 1 - Stages of Manufacturing of a Hex-DMR II Module 7

Figure 2 - Comparison of Hex-DMR and Hex-DMR II 8

Figure 3 - Isometric and Exploded Views of Hex-DMR II 11

Figure 4 - Exploded View of a Drive Module 12

Figure 5 - Module Insertion onto the Central Hub 14

Figure 6 - Detailed View of Module to Central Hub Connections 15

Figure 7 - Two Sides of the PIC Board 18

Figure 8 - Central Hub 20

Figure 9 - Central Hub PCB with Connections 21

Figure 10 - Drive Module 22

Figure 11 - Manipulator Module 24

Figure 12 - Power Module 25

Figure 13 - Control Module 26

Figure 14 - Camera Module 28

Figure 15 - Elevator Module 31

Figure 16 - Autonomous Module Extraction 37

Figure 17 - Autonomous Module Insertion 39

Figure 18 - Complete Teleoperated Module Repair 42

1. INTRODUCTION

1.1. COOPERATIVE MULTI-AGENT SYSTEMS (CMS)

With their constantly increasing capabilities, robots continue serving as humankind's reflection of intelligence in hostile and unknown environments such as the deepest trenches of our oceans [20] or on the uncharted craters of Mars [15]. Consequently, to mitigate risk and reduce variability in such environments, an increasing number of applications utilize cooperative multi-agent systems (CMS) [5] [10] [17].

A CMS consists of independent agents that work together towards a common goal. Such a distributed system offers many advantages over a conventional, single agent system including advanced capabilities without the additional complexity. The ability to distribute tasks within a team leads to individually less-complex (and less-costly) agents that can "multitask" effectively as a team. Furthermore, such a system can tolerate some of its agents becoming non-functional as their tasks would be taken over by the remaining functional agents thus making the whole system more robust [5]. Maybe the most iconic example of a CMS can be cited as the fictional T1000 liquid-metal robot from the Terminator series by James Cameron. This evil creation of machine intelligence can repair itself and change its form to outwit the pesky humans.

In reality, to create a more desirable future, many groups have worked to harness the peaceful benefits of CMSs. The first example of applying a connection mechanism to an entirely modular robot was realized in the CEllular roBOT (CEBOT) by Fukuda in late 1980's [8] [24]. CEBOT consisted of relatively large modules (each about 1.1 kg) that could communicate and interact with each other using independent processors and motors.

Within the next two decades several other projects explored various ways of locomotion in modular robotics. A prominent example is the Polypod [23]. Developed by Yim in 1994, the Polypod could demonstrate a variety of locomotion gaits such as the slinky, caterpillar or rolling track gait [24].

The complexity of the control problem due the high number of modules in a system soon became apparent. The lattice style configuration systems developed by Chirikjian and Murata [4] [14] allowed to represent the configurations easily and paved the way to realize more complex kinematics on modular robotics. As a result lattice systems soon became popular among computational roboticists [24].

Since the 2000s advances in computational and sensory techniques as well as the establishment of new fields such as swarm robotics resulted in the creation of even more capable multi-agent systems. Although still far from a cunning T1000, the abilities of organization [16] [22], assembly [11] [18] and reconfiguration [2] [13] have been demonstrated through different systems.

1.2. HOMOGENOUS AND HETEROGENOUS CMS

Depending on their specific morphologies, modular robotics can be split into two groups as homogenous and heterogeneous. Homogenous systems consist of identical modules that can be arranged into complex structures and mechanisms such as a truss or a six degree of freedom (dof) robotic arm [13] [9] [19]. On the other hand, heterogeneous modular systems often delegate tasks to different modules with distinct features that come together to make up a more capable system.

Not surprisingly, faulty recovery and repair have different implications for homogenous and heterogeneous CMSs. In the former case, as mentioned earlier, faults can be tolerated up to a degree by discarding the faulty components and distributing their tasks among the remaining agents [7]. In the latter case, fault recovery implies replacing the module in a fault state by a specific module. Unlike in homogenous fault recovery, a successful repair operation in a heterogeneous system does not degrade the overall capability of the system as long as a replacement module is available.

1.3. TEAM-ASSISTED REPAIR ON HETEROGENOUS CMS

Only recently heterogeneous modular robotic systems have been developed to demonstrate faulty module replacement and team-assisted repair. The first two examples are by Bererton and Khosla in the form of an agent with three replaceable subsystems and by Kutzer through a four-module, repairable, heterogeneous system. [3] [12].

The early research efforts on the topic also led to the development of theories on heterogeneous system repair design. Below are the series of constraints proposed by Bererton and Khosla (a-c) and later expanded by Ackerman and Chirikjian (d-f) that are necessary for a heterogeneous CMS to perform robust team repair [3] [1].

a) **Homogeneity and robustness of repair:** Every component of the system should be repairable using similar procedures and tools. The repair should be possible regardless of the configuration and the orientation of the system as well as the state of the environment.

b) **Completeness of repair:** In order to address the most number of failure modes, as many components as possible in the system should be replaceable.

c) **Resolution of repair:** The replaceable component should be as small as possible to mitigate the costs associated with the lost components during the replacement process. An ideal repair process is fine enough to replace only the faulty component.

d) Independence of repair: The repair process should not rely on any assistance from the agent under repair.

e) Ubiquity of repair capabilities: If not all, most of the members of the CMS should be capable of repairing (and being repaired by) the other agents.

f) Versatility of agents: Stressing that team repair is not the primary task of the CMS, the system should be capable of performing other tasks when a repair of an agent is not needed.

In addition to the six above, the author proposes that the following constraint must also be satisfied to ensure that a given CMS can realize self-repair with minimal assistance from its environment:

g) Self-sustainability of the system: The system should carry redundant components to supply enough spare parts for the maximum number of part replacements it can realize among the agents.

Following these (first six) design constraints Chirikjian and Ackerman worked on a testbed to demonstrate team-assisted repair through a heterogeneous CMS design resulting in the development of the first generation Hexagonal Distributed Modular Robot (Hex-DMR) [1]. This system, which provided the main source of inspiration for our work, successfully exhibited remotely-assisted team repair. Our work furthered this effort and led to the demonstration of autonomous team-assisted repair through a more capable system, Hex-DMR II [6] [5]. The two systems are compared under the 2.2. Evolution of Design section below and also in *Figure 2*.

This essay details on the design of Hex-DMR II using a top-down approach, presenting the capabilities of the entire system followed by the roles of each module. It also discusses the implications of our design choices and how certain design aspects fulfill the constraints listed above. The 3. Methods section describes the algorithms for the remotely-assisted and autonomous part replacement processes whose results are shared under the 4. Team-Assisted Repair Maneuver Demonstrations section. Finally, the essay offers an evaluation of the system with respect to its performance in the demonstrations and suggests further improvements to the Hex-DMR II.

2. DESIGN

2.1. MATERIAL SELECTION AND MANUFACTURING

The first design choice that had to be made in the creation of the Hex-DMR II system was the material selection. The base material which almost all mechanical components of the system would be made of had many implications on the rest of the design, including size, weight, resolution, tolerances, strength, manufacturing techniques, cost etc.

The candidates for the base materials included the four that are commonly used in field of robotics research: sheet metal, prefabricated robotic kits, 3D printed plastic and laser-cut acrylic. Ultimately, clear laser-cut acrylic of thickness 0.12” was selected due to the following reasons:

- easy accessibility
- low material cost
- high strength to weight ratio
- sufficiently high resolution
- easy access to CNC manufacturing facilities (laser-cutter)
- easy to machine manually if needed
- abundance of mechanical and chemical bonding techniques
- transparency (visible internal components help with debugging)

- insulator and chemically inert

The drawbacks of this choice of material include:

- 2D by nature and cannot be bent or folded
- brittleness
- limitations due to the laser cutter (i.e. feed rate of the laser beam directly affects the amount of material removed and thus the tolerances)

Almost all structural components of the system including the outer shells of the modules and the inner supports are made of the same thickness laser-cut acrylic. The remaining components include off-the-shelf electronics, wheels and motors as well as a few custom made parts such as molded two-part plastic screws, threaded inserts and coiled springs.

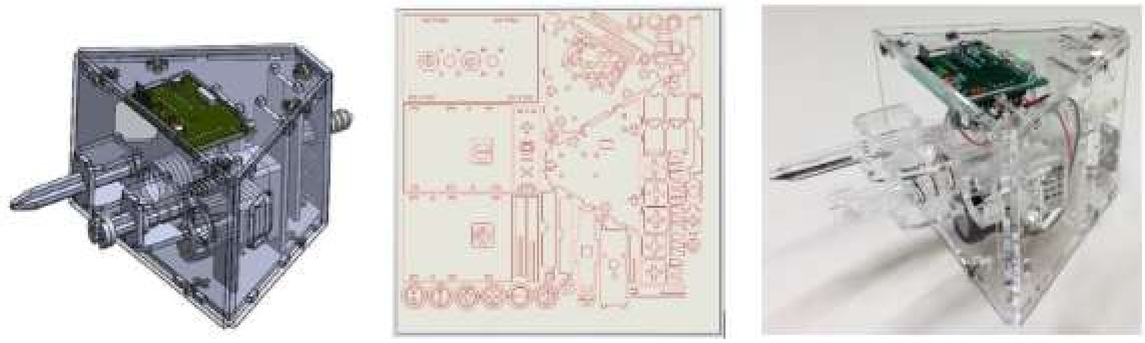
The components of the system are held together either by chemical adhesives (acrylic cement or two-part epoxy glue) or mechanical fasteners (nuts and machine screws). The choice depends on the desired permanence of the connection: components designed to make a permanent assembly (i.e. a shaft) are bonded together using chemical adhesives whereas parts like the outer shell of a module that is expected to be frequently opened to access the inside are held in place by screws. Size 4-40 standard machine screws and nuts are selected due to their compatible size to the thickness of the acrylic sheets and are exclusively used as mechanical fasteners throughout the system for practical reasons.

The manufacturing of the system consists of the following steps:

1. The system is modeled in a CAD software (SolidWorks) as a composition of parts and assemblies. All the acrylic parts are given a thickness of 0.12”.
2. A drawing file for each module containing the planar footprints of the parts within is generated.

3. The drawing file is loaded onto the laser cutter which in return translates the 2D drawings to g-code and cuts the 24”x12” acrylic sheets into the desired parts.
4. The acrylic parts and the other components are manually assembled into the modules of the Hex-DMR II system.

See *Figure 1* for the stages of the manufacturing process.



a) CAD Model *b) CAD Drawing* *c) Completed Module*

Figure 1 - Stages of Manufacturing of a Hex-DMR II Module

2.2. EVOLUTION OF DESIGN

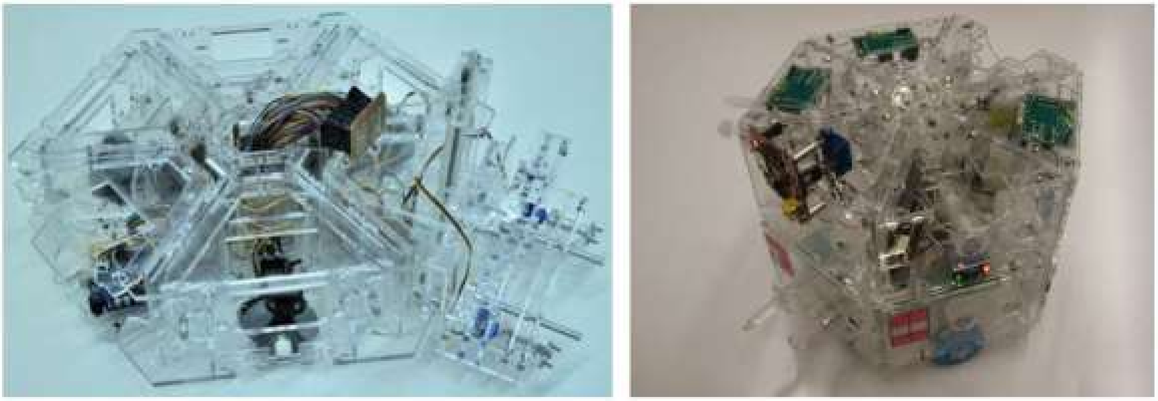
The first generation Hex-DMR has provided a baseline for the development of Hex-DMR II. While inheriting the overall geometry of the previous system, the new design offers the following improvements over its predecessor:

- autonomy due to sensing capabilities
- increased overall robustness
- redundancy and versatility

The Hex-DMR II agents can 'see' their environments through the use of the Camera Module (see 2.4.6. Camera Module) and distinguish other agents and modules through a computer vision algorithm calibrated for the specific visual trackers on the agents. The inclusion of such a sensing capability in the design of Hex-DMR II when paired with a decision making algorithm allows the system to perform autonomous team-assisted repair.

The increased robustness in the Hex-DMR II system is due to the improvements in the design of intra-modular mechanical and electrical connections. Hex-DMR II only has four electrical connections between modules compared to twenty in Hex-DMR making an electrical misconnection less likely. The forklift-like manipulator in the older system is replaced with a screw-like actuator that is more error tolerant during module replacement maneuvers while consuming less power.

Owing to its two-level structure, Hex-DMR II system can carry up to twelve modules – twice as many as the Hex-DMR. This allows the Hex-DMR II to hold spare modules not required for basic functionality. The extra modules add redundancy, can be used as spare parts during module replacement or can grant additional capabilities to the agent, making Hex-DMR II a highly versatile system. Refer to *Figure 2* for a visual comparison of the two systems.



a) Hex-DMR *b) Hex-DMR II*

Figure 2 - Comparison of Hex-DMR Systems

2.3 SYSTEM OVERVIEW

2.3.1 MODULARITY

The Hex-DMR systems exhibits a modular structure where every function of an agent is confined to one or more modules. This allows the problem on an agent to be fixed by replacing the module that contains the malfunctioning component. By making each module as specific to its task as possible, the resource cost of repair is mitigated.

The capabilities the Hex-DMR II system is envisioned to have gave rise to the design of the following six modules:

- Drive Module (DrM)
- Manipulator Module (MaM)
- Power Module (PoM)
- Control Module (CoM)
- Camera Module (CaM)
- Elevator Module (ElM)

A passive Central Hub (CH) serves as the backbone of the system, ensuring the mechanical and electrical connections between the modules. (See Section 2.4. Modules for a detailed description). A combination of a subset of all the modules is present in every agent.

In Hex-DMR II, the size and the extent of the contents of each module are selected to strike a balance between practical simplicity and resolution of repair [1] [3]. Admittedly, there is a lot of room for improvement in the resolution which would be achievable by further downsizing the modules. The author believes that by following the same design principles and defining finer modules, the next version of a modular CMS can be capable of replacing its end effectors and wheels independently of its actuators.

Another advantage of the modular configuration featured by the Hex-DMR systems is that it allows homogeneous repair [1] [3] across all agents regardless of the mode of failure. In almost all cases, a failure is contained within the physical boundaries of a single module which requires the same procedure and tools to replace as all the other modules.

2.3.2. MODULAR REDUNDANCY AND CUSTOMIZABILITY

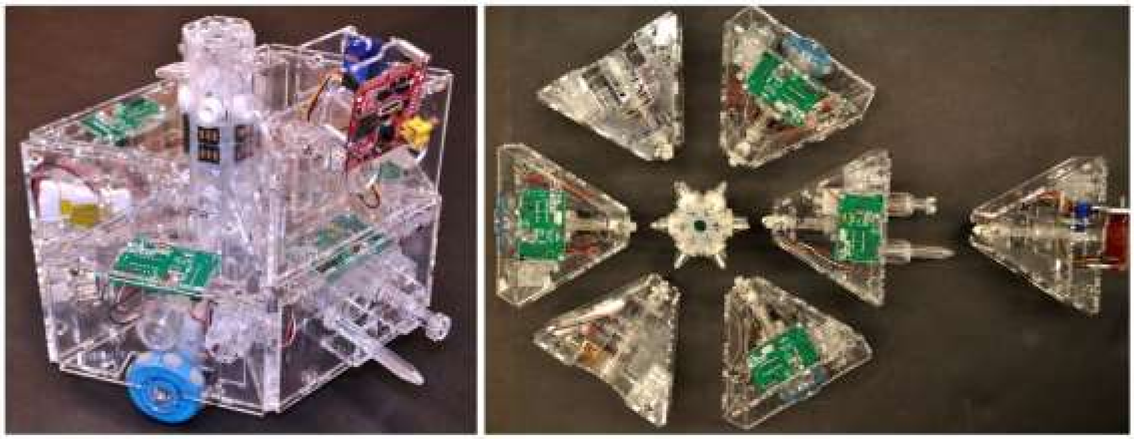
The double layered configuration of Hex-DMR II allows docking of twelve modules onto the CH while an agent only needs five for basic functionality. The four most simple functional agent configurations are

- **Basic Configuration:** three DrMs, one CoM, one PoM
- **Basic Repair Configuration:** Basic Configuration plus one MaM
- **Autonomous Repair Configuration:** Basic Repair Configuration plus one CaM
- **Support Configuration:** Autonomous Repair Configuration plus one ElM and/or another MaM on the other level.

Refer to *Figures 2* and *3* for an isometric and exploded view of a Hex-DMR II agent in autonomous repair configuration, respectively.

The empty spots on an agent in one of these configurations can be filled with spare modules to be used in a team-repair maneuver. The modular redundancy in Hex-DMR II fulfills the self-sustainability requirement of the system. It has been shown that the system allows for 9895 different functional agent configurations each featuring a different combination of spare modules [6].

The Hex-DMR II system also allows for the change of functionality of an agent during a mission by exchanging a module to meet the changing mission requirements resulting in a highly versatile dynamic system.



a) Isometric View *b) Exploded View*

Figure 3 - Isometric and Exploded Views of Hex-DMR II

2.3.3. SIX-FOLD ROTATIONAL SYMMETRY

The completeness of repair [1] [3] is ensured through the hexagonal placement of modules on every agent. Configuring the modules as two vertically stacked rings around the hub allows each to be accessed externally. The choice of having six modules on every level is shown to be the right balance between the number of modules and the ease at which they can be accessed [5].

The six-fold rotational symmetry is also propagated through the inner structure of the modules. Each module features a central shaft extending from the front face (facing outward) to the back face (closest to the CH). Inside every module vertical supports keep the central shaft aligned and control its axial sliding without restricting rotation. To the right side of the main shafts, each module also has a conformal friction mechanism, an assembly to provide a secondary point of contact to the MaM (see 2.4.3. Manipulator Module) during a replacement maneuver, facing forward. These features repeated in every module regardless of their function make the repair process to resolve any problem in a given agent highly homogenous. *Figure 4* shows an exploded view of a module containing these common features.

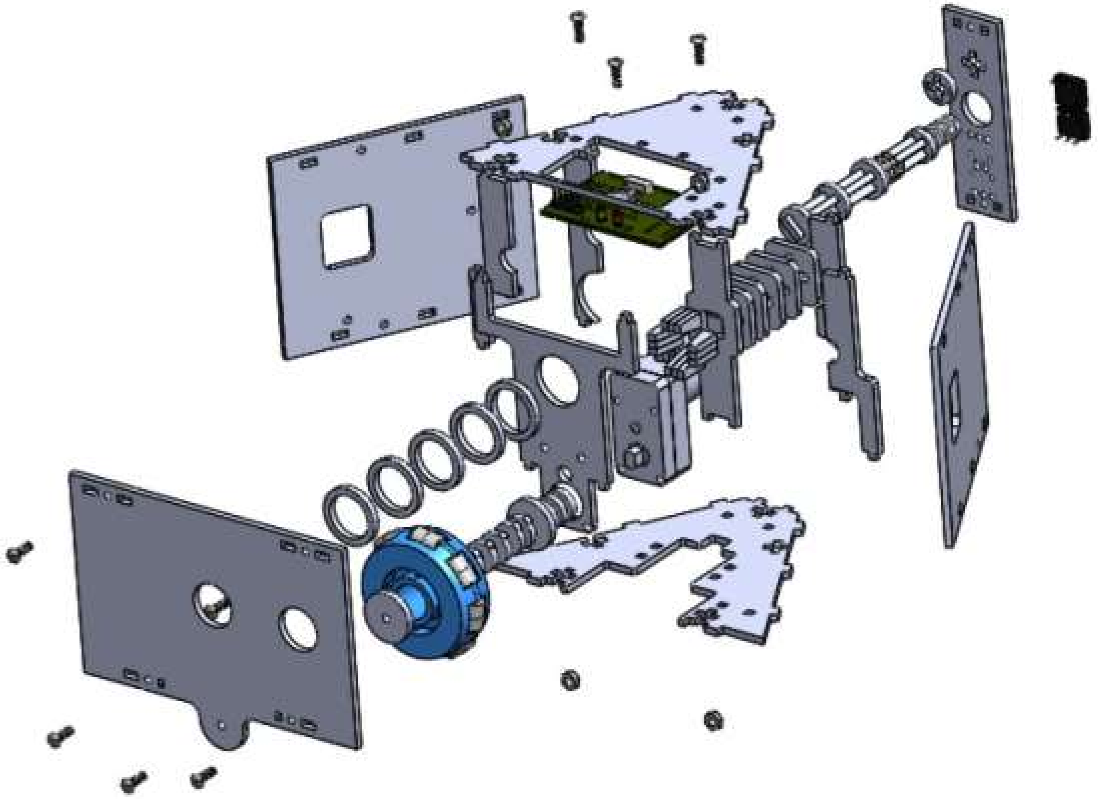


Figure 4 - Exploded View of a Drive Module

A byproduct of the six-fold rotational symmetry across each agent is the outer shape of the modules. Every module (excluding the CH) has the same trapezoidal footprint. The benefits of this shape include the following:

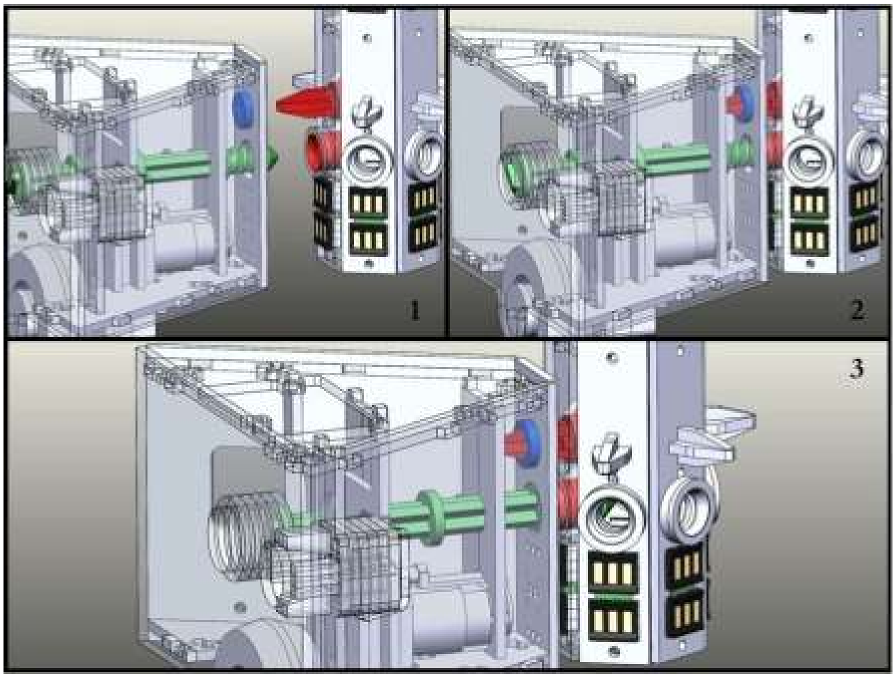
- During module insertion, the side faces of the two neighboring modules help guide the target module towards the CH.
- The side faces of each module are designed to be flush against each other forming a complete hexagonal ring. This configuration improves the overall stability of the agent and ensures robust electrical connections between the modules and the CH.

- The outer faces of each module can be easily removed to gain access to the internal components of the modules providing practical benefits.

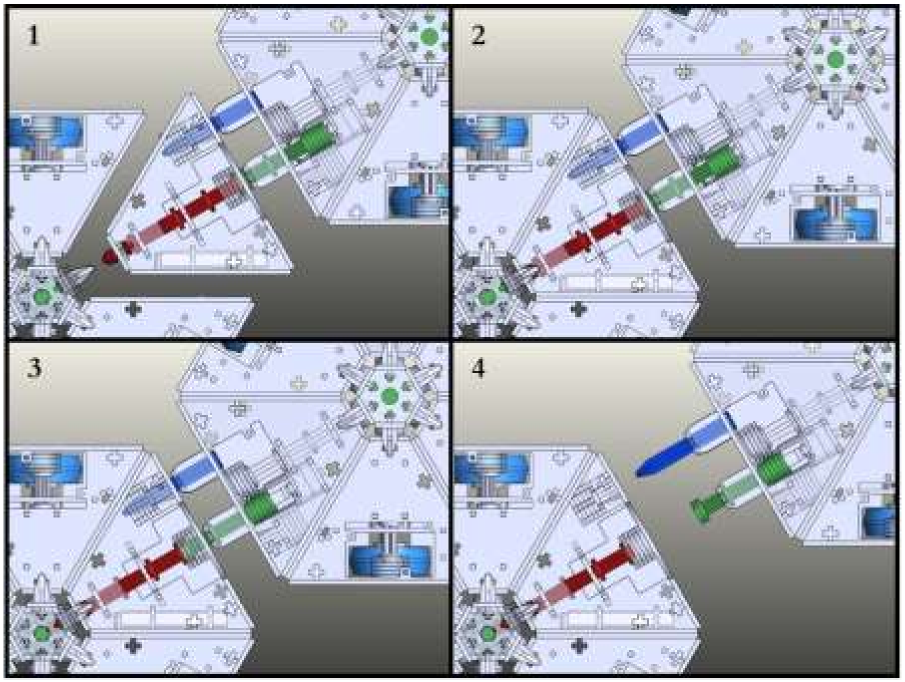
2.3.4. DOCKING MECHANISM

The mechanical connections between the modules and the CH (see 2.4.1. Central Hub) on Hex-DMR II are realized using a screw-like mechanism and passive alignment pins. Screws that are mounted on the back end of the central shafts on every module are designed to mate with the corresponding inserts with internal threads on every face of the CH. Once screwed in, the modules are prevented from arbitrarily rotating about the screws by the use of the alignment pins that extend from the CH into corresponding holes on the back of each module. The alignment pins serve a secondary role during docking as they help correct for small errors and ensure proper alignment of the screws to the corresponding inserts.

The MaM (see 2.4.3. Manipulator Module) features an active shaft that engages with the central shafts on every module and can screw the modules in and out of the CH. The rotating shafts on the MaMs as well as the passive central shafts on every module feature coil springs that keep the two shafts in contact as they slide along their central axis during docking. Additional alignment pins that extend out from the front faces of the MaMs further help with the docking by engaging with the respective conformal friction mechanisms on every module. *Figure 5* shows the procedure of module insertion onto the CH.



a) Isometric View of Module Insertion



b) Top View of Module Insertion

Figure 5 - Module Insertion onto the Central Hub

The screw and alignment pin mechanism featured in the newest version of the testbed is shown to greatly increase the robustness of the repair process compared to the linear actuation system used in the earlier version. The coarse, polyurethane screws (3.15 threads/cm) with tapered ends along with the alignment pins on the CH ease the docking process by allowing proper mating even when the module and CH are misaligned up to 2.54mm or 18° 46' 12" during module insertion and 16° 48' 36" during module extraction [5].

The mechanical docking of the modules onto the CH also ensures an electrical connection between them. The six male pins located at the back of each module, below the screws, align with the corresponding female pins on the CH (only four of the connections are active, see 2.3.8. Control). The compliant nature of the male pins (spring-loaded with 1.5mm of travel) ensure a robust electrical contact at all connections during normal operation of the system. See *Figure 5* for a detailed view of mechanical and electrical connections between the modules and the CH.

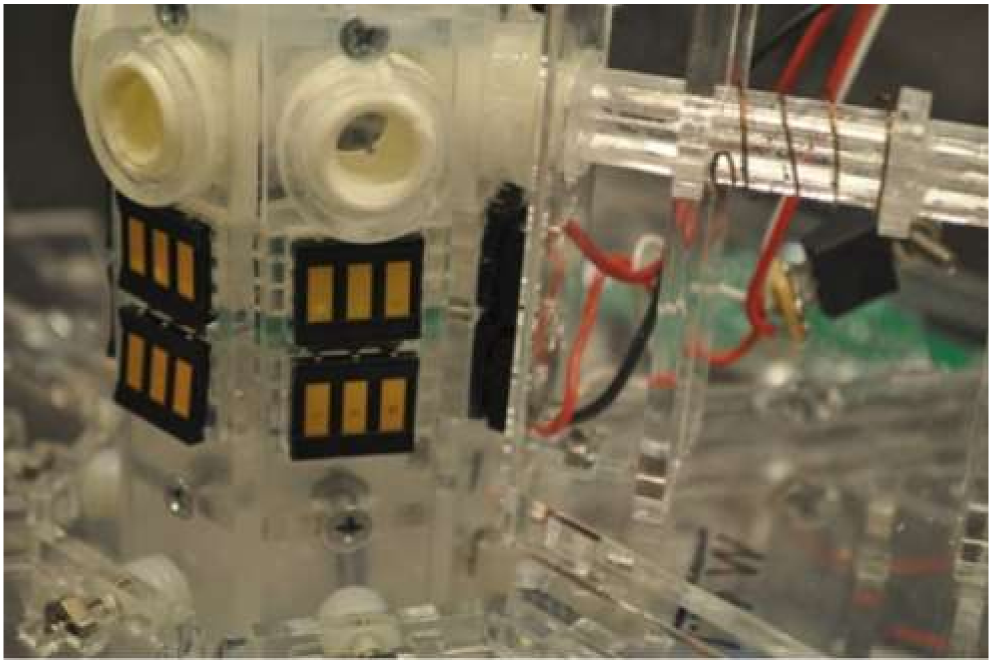


Figure 6 - Detailed View of Module to Central Hub Connections

2.3.5. MOBILITY

The docking procedure employed in the part replacement process in the Hex-DMR systems require the agents performing fine-tuned maneuvers. Previous work by Wolfe [21] has shown that non-holonomic systems require numerous corrections to achieve the desired small movements and can be problematic. Holonomic systems, on the other hand, can create instantaneous accelerations in any direction and orientation, an ability that makes fine docking maneuvers much more efficient.

The Hex-DMR systems feature three DrMs (see 2.4.2. Drive Module) each containing an omni-directional wheel arranged radially spaced 120° apart around every agent (in Hex-DMR II, the DrMs can be stored in the upper level but are only functional when in the lower level). This configuration, despite the wheels having fixed axes (non-steerable), only requires three bidirectional actuators for holonomic movement. The improved maneuverability owing to this drive scheme allows Hex-DMR systems to easy dock and correct for errors in their position/orientation.

2.3.6. SENSING

The Hex-DMR II features a CaM (see 2.4.6. Camera Module) for visual sensing of the environment and other agents for navigation. The onboard camera can also identify individual modules on other agents using a 4-bit barcode thus making autonomous closed feedback-controlled team-assisted repair feasible.

2.3.7. COMMUNICATION

The Hex-DMR II agents are capable of wireless data communication with other team members and a human operator depending on the mission. The communications are handled by the XBee wireless radio contained in the CoM (see 2.4.5. Control Module). In the teleoperated mode, the system receives commands from an operator and sends a confirmation back for every command received. For an autonomous mission, any agent can be programmed to communicate with other members in order to perform a coordinated repair maneuver involving multiple active agents. In addition, inter-agent communication can also be used as a part of team-assisted diagnosis, where the communication with an agent (or lack thereof) can indicate the type of faulty state.

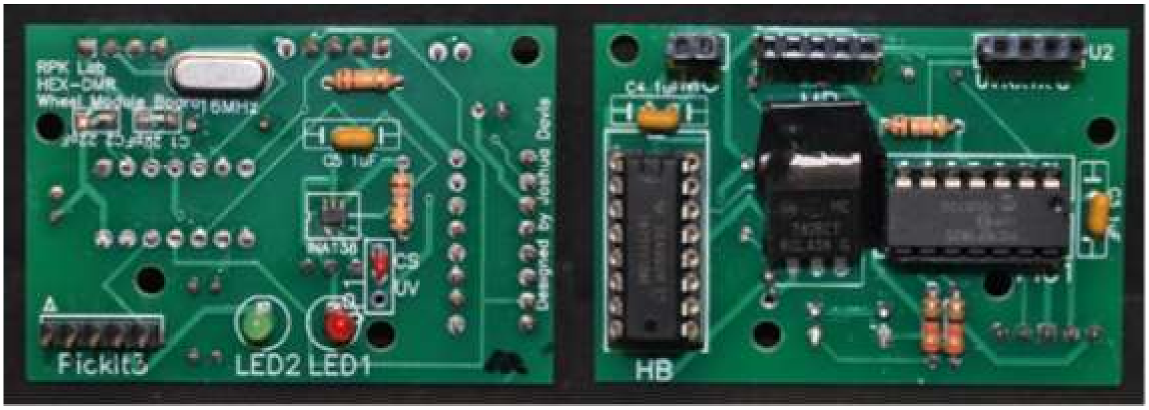
2.3.8. CONTROL

The data processing within an agent of Hex-DMR II is handled through the ATmega168-20PU microcontrollers contained inside the CoM as well as uniquely addressed PIC16F1825 microcontrollers installed on the PIC boards on every actuated module (DrM, MaM and ElM). Addressed data packets generated by the main microcontroller are sent through asynchronous serial communication to the PIC boards and are translated into motion at the actuators.

Other than the microcontroller, a PIC board also contains

- a voltage regulator to step the 7.4V supply voltage down to 5V for the microcontroller
- a 16MHz crystal oscillator to generate a clock signal
- a quadruple half-H driver (H-bridge) for bidirectional motor control (through a steady-state or pulse-width modulated signal depending on the application)
- a ceramic capacitor as a single-stage low-pass filter to reduce the electrical noise on the motor leads
- a pair of green and red light emitting diodes (LEDs) to indicate that the board has power (solid green), the board receives an invalid serial command (blinking green) and a byte is received over the serial communication (blinking red).

Figure 6 shows both sides of the board and its components. The board is placed on top of each actuated module for quick access to the serial programmer leads and good visibility.



a) Front *b) Back*

Figure 7 - Two Sides of the PIC Board

One advantage of the distributed control architecture of Hex-DMR II is that it allows the central processor to only perform the general tasks such as navigation and module exchange based on visual feedback. The movements required to achieve the general command sent by the CoM are processed individually at the lower level microcontrollers and are realized at the actuators dedicated to each microcontroller.

The use of a distributed control system also allows Hex-DMR II to only contain four electrical lines between every module and the CH, resulting in a system less prone to electrical misconnections. The electrical bus lines on Hex-DMR II are:

- Power (PWR)
- Ground (GND)
- Receiver (RX)
- Transmitter (TX)

The commands from the CoM are conveyed through the TX line to the actuated modules and the CaM which in return send information back on the RX line.

2.4. MODULES

2.4.1. CENTRAL HUB (CH)

The CH is a passive component responsible of the electrical and mechanical connections between the modules. Its design is a byproduct of the shape, size and number of the modules as well as the docking mechanism selected. The result is a tall hexagonal prism with twelve threaded inserts and alignment pins arranged as two rings along the long sides of the CH (*Figure 8*). The function of these mechanical components are discussed under section 2.3.4. Docking Mechanism.

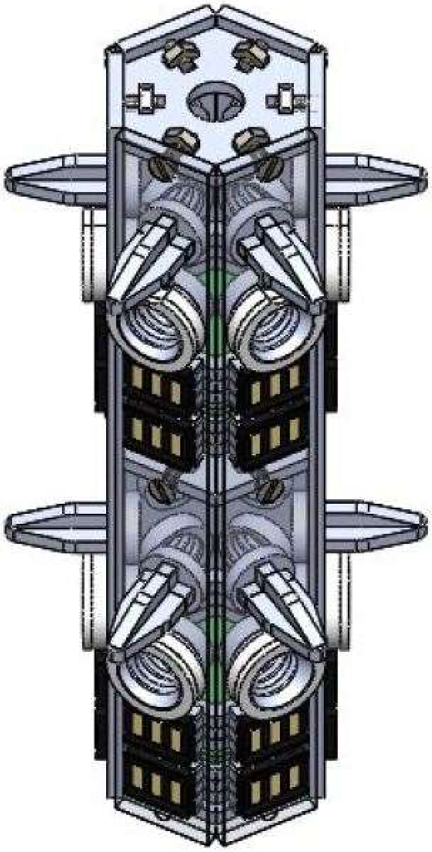


Figure 8 - Central Hub

The electrical connection between every module docked onto the CH is ensured by twelve sets of six female pins positioned across the faces of the hub to correspond to the compliant male pins on the back of each module. These female pins are augmented with brass c-channels facing outward to further increase the robustness of the electrical contacts. The CH also houses the custom-made PCB boards to which the passive pins are soldered (*Figure 9*). Each of these boards connect the thirty six female pins together and are responsible for the connections between one level of modules. The two PCBs that are required to accommodate a total of twelve modules are connected together using a ribbon cable that vertically transverses the CH.

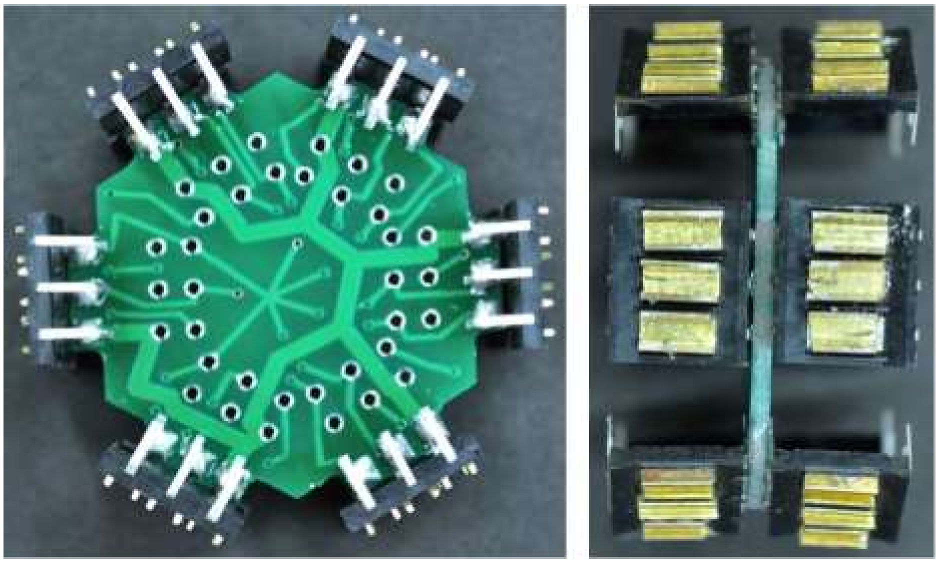

a) Top View *b) Side View*

Figure 9 - Central Hub PCB with Connections

2.4.2. DRIVE MODULE (DrM)

The DrMs give mobility to Hex-DMR system. Each agent requires at least three functional DrMs placed in a radially symmetric fashion on its lower level to move as intended (see 2.3.5. Mobility).

Each DrM contains a 49.2 mm diameter omni-directional wheels with eight cylindrical rubber rollers which allow for sliding along the axial direction as dictated by the holonomic drive scheme while maintaining the "no-slip" condition in the tangential direction. Each wheels is driven by a geared DC motor, attached through a keyed shaft (*Figure 10*).

The wheels and the motors make the DrMs the second heaviest module (after the double-layered ElM) with a mass of 262 grams. This property is harnessed to increase the overall stability of the agents as the center of mass of each agent is concentrated in the geometric center when they contain three evenly-spaced-apart DrMs.

The DrMs have the same the same trapezoidal outer dimensions as the every other module with the exception of the protruding wheel and motor on the bottom face. When placed on a flat surface, this provides 30 mm of ground clearance. Due to the downward protruding features, the DrMs can only be stored on the upper level directly above a PoM or CoM. These modules are not actuated and thus do not need a PIC board on their top faces. This has allowed their design to include a cut-out on their front and top faces to accommodate a DrM stored on top of them. In this configuration, the DrM needs to be removed before the module below it can be accessed.

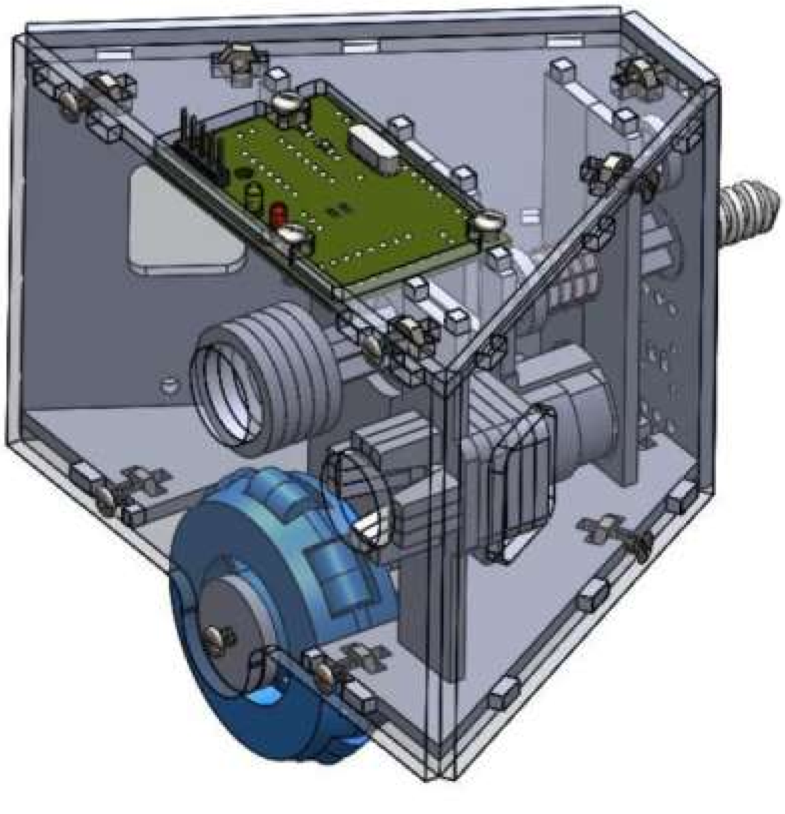


Figure 10 - Drive Module

2.4.3. MANIPULATOR MODULE (MaM)

The MaMs are the way the Hex-DMR II agents physically interact with each other during the part replacement process. Their distinguishing features include a protruding active shaft driven by a DC geared motor and an additional alignment pin extending outward from the front face of the module (*Figure 11*).

The alignment pin extends 39mm from the front face of the MaM. Its acts as the first point of contact between the MaM and the target module during the module extraction process. Its tapered end is designed to fix misalignments up to 2.54mm and 0.33 radians as the two modules engage [6]. Once fully engaged, the contact between the alignment pin and the conformal friction mechanism of the target module ensures the module to be safely attached to the MaM during transportation and until the module is inserted onto a new agent.

The actuated component of the MaM, the active shaft, is the component that realizes the mechanical attachment/detachment of the target module onto the CH. Its active end has a slotted head screw-like form designed to couple easily with the slotted screwdriver-like end of the central shaft of the target module. Similar to the center shafts on each module, the active MaM shafts feature a coiled spring wound around its body, pushing the shaft forward. This assembly, when coupled with a central shaft, ensures that the shafts remain in contact throughout the attachment/detachment procedure as the screw at the back end of the central shaft screws into or out of the threaded insert on the CH. The range of axial movement on both the active and central shafts is 11mm, which in return drives the male screw into the CH by the same amount during module docking.

Realizing that the MaM is a component with relatively high risk of a malfunction due to its high complexity, Hex-DMR II system allows every MaM to be manipulated and replaced using another MaM the same way as the other modules. Similar to the other modules, the MaM also houses a central shaft and a conformal friction mechanism that can engage with the active shaft and the alignment pin of another MaM, respectively. This ensures the homogeneity and completeness of repair in the system. [1]

The PIC board contained on top the MaM features an additional current sensor to control the actuation of the active shaft. During module insertion, the active shaft turns clockwise until a higher than nominal current is sensed on the PIC, caused by stalling of the motor due to the screw being fully engaged into the CH. Conversely, during module extraction, the PIC allows the active shaft to turn counterclockwise until the screw is completely out of the threaded insert, which is indicated by a lower than nominal current through the current sensor.

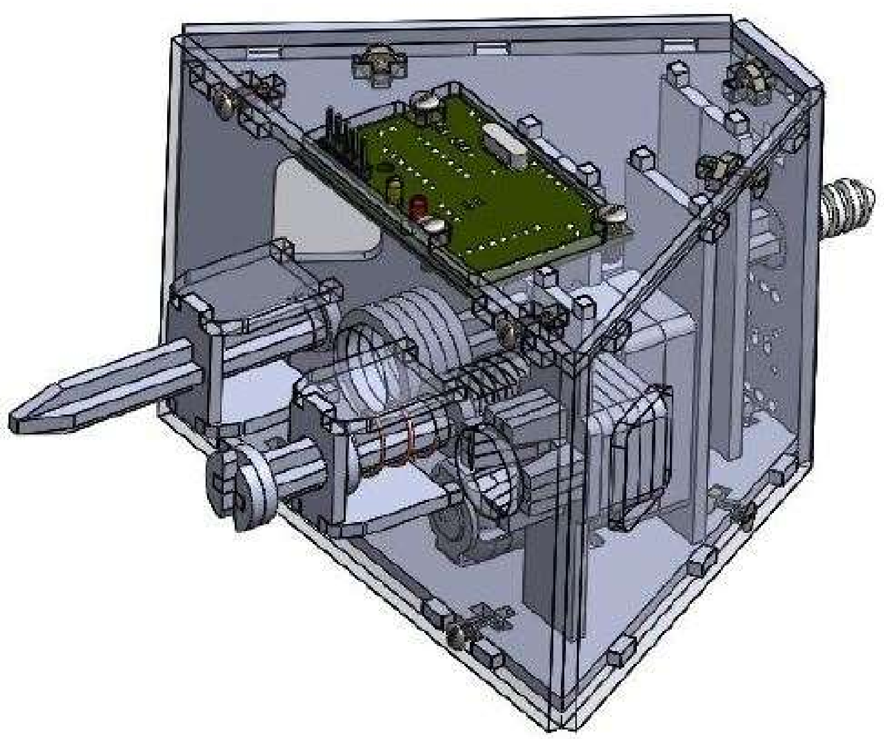


Figure 11 - Manipulator Module

2.4.4. POWER MODULE (PoM)

The PoM features the battery to supply power to all the other modules docked onto an agent (*Figure 12*). While at least one PoM is required on each agent for functionality, multiple PoMs can be docked onto an agent to prolong service life or simply for storage and transportation. The internal volume of the PoM houses an 800mAh, 7.4V Lithium-ion polymer battery connected to a power switch and a DC power jack for charging. Cutouts on the front panel provide easy access to the switch and the power jack. The connection between the battery and the male electrical pins on the back of the module includes a diode in order to prevent back-charging when multiple PoMs are docked on an agent.

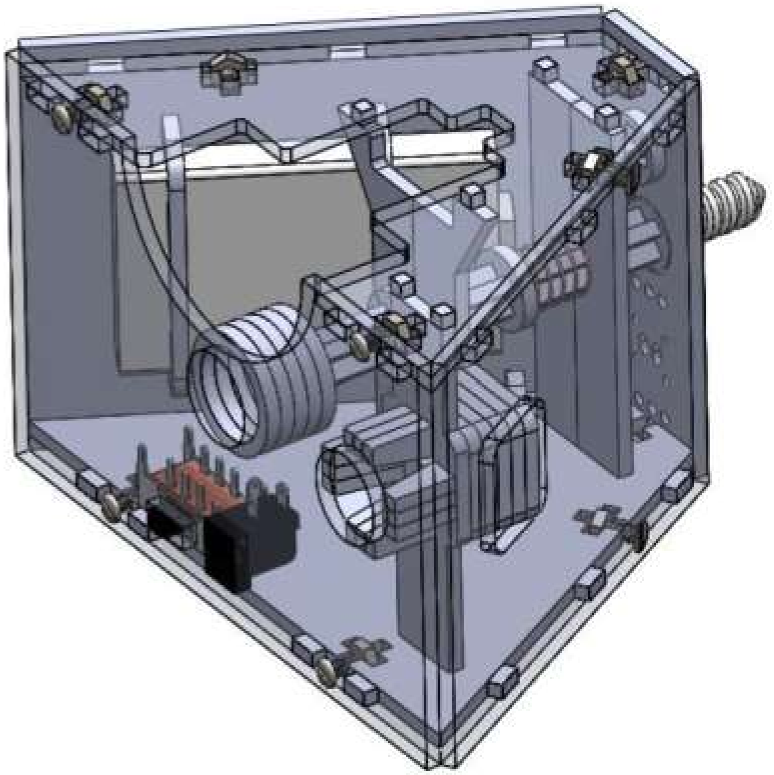


Figure 12 - Power Module

2.4.5. CONTROL MODULE (CoM)

The second passive module, in the Hex-DMR II system, the CoM, is responsible of handling the decision-making in each agent as well as inter-agent and agent-controller communications (*Figure 13*). It features an ATmega168-20PU microcontroller for serial communication and data processing. The CoM uses the TX line on the electrical bus to send commands to the actuated modules as well as the CaM and the RX line to receive information.

The microcontroller is also connected to an XBee wireless radio through which it can communicate with a controller as well as with other agents. The commands through the XBee are sent and received via a software serial protocol. The radio is positioned below the central shaft inside the CoM and can be easily accessed through a cutout on the front face of the module.

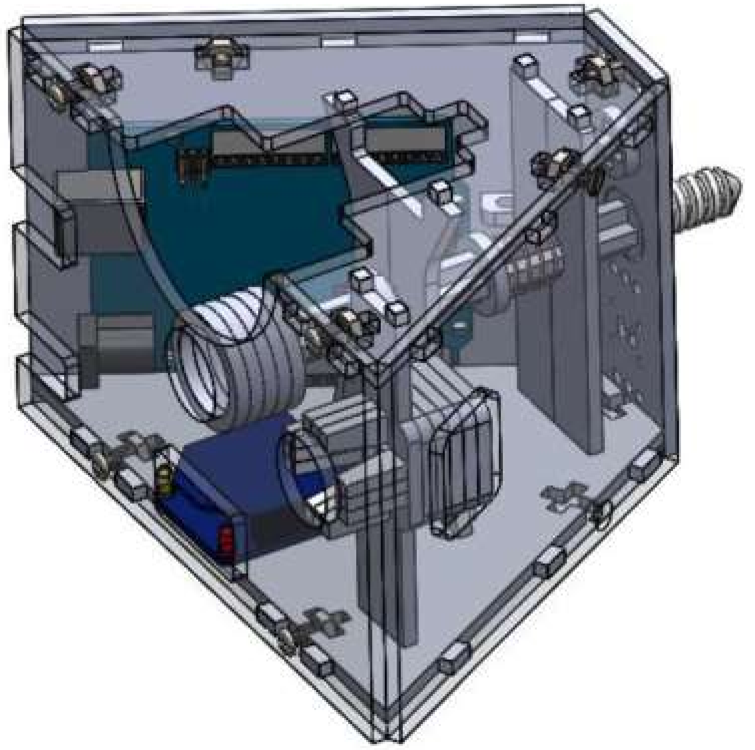


Figure 13 - Control Module

2.4.6. CAMERA MODULE (CaM)

The CaM is the 'eye' of the Hex-DMR II, granting the system its only sensing capability (*Figure 14*). The CMUcam4 was selected for this application due to its sufficient capabilities to realize the simple autonomous maneuvers the system was envisioned to perform, while being relatively simple to use. It interfaces with the microcontroller in the CoM over serial communication, similar to the PICs on the actuated modules. The CMUcam4 provides simple color tracking that is actively used to locate other agents and to differentiate between modules during the autonomous repair process. Theoretically, it can also be used for navigation and collision avoidance.

The camera on the CaM is placed on top of the central shaft, directed forward. This layout ensures an unobstructed view. The field of view is further augmented by the use of a servo motor that actuates the camera plate and can provide a tilt angle up to 0.79 radians. One drawback of this configuration is that the camera protruding upward from the top of the module restricts certain placements of the module. The CaM can freely be docked on the upper level of an agent, but no other module can be docked on top of it when it is placed on the lower level.

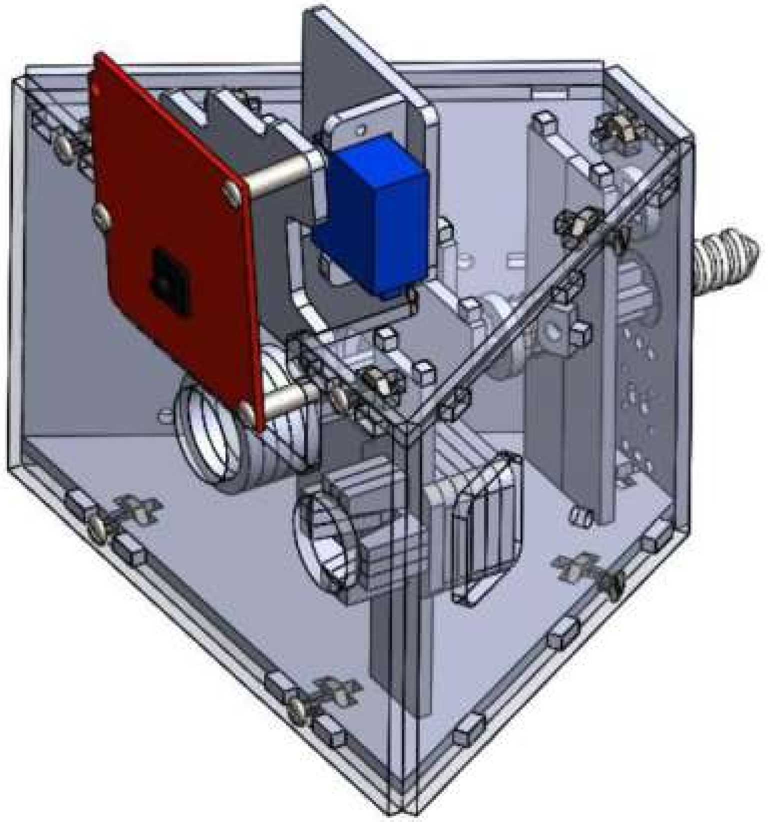


Figure 14 - Camera Module

2.4.7. ELEVATOR MODULE (ElM)

The ElM is a unique module that is developed to be able to fully benefit from the two-leveled architecture of the Hex-DMR II system (*Figure 15*). The design that spans both the levels of an agent is a byproduct of its function as an elevator that can vertically transport modules between the upper and lower levels. This ability increases the versatility of the system by providing access to all twelve spots on the CH for module docking. As a result, the ElM is roughly twice as large as every other module.

The vertical actuation inside the ElM is realized through a nut and threaded rod system. The 3/8" - 8 Acmed size stainless steel threaded rod, positioned to right of the central hub, vertically spans the entire module. It is coupled to a DC motor at the bottom of the module through a custom made gearbox. The use of the gearbox with a 1:1 overall gear ratio is required to meet the space requirements inside the module. A nut of the same thread size, screwed onto the threaded rod and restricted from rotating, moves along a vertical track in the direction determined by the direction of rotation of the motor. This assembly provides a mechanical advantage of 14.1, a value that is carefully chosen to ensure that the motor can lift even the heaviest module without making the whole process too time intensive. A vertical panel is fixed around the nut and includes the features for a module to dock onto, similar to a side panel on the CH. The ElM also features two IR sensors, each positioned on the opposite ends of the vertical track 'looking' towards each other. They are calibrated to trigger a signal to the PIC board when the sliding panel reaches the corresponding end of the vertical track and to consequently terminate the operation of the motor.

The transportation of a module between the two levels requires the target module to first be inserted onto the sliding plate on the ElM by using a MaM contained on another agent. This happens the same way as attaching a module onto a CH, through the mating of the screw on the back of the module onto the threaded insert on the ElM's sliding plate. Once the module is fully engaged, the sliding plate moves in the desired vertical direction until it triggers an IR sensor on the opposite side of the track, indicating that module is in terminal position. Finally, a MaM on the corresponding level of a different agent disengages the target module from the ElM and picks it up. One must note that this process requires at least two functional agents: one containing the ElM and the other(s) with the MaMs to get the target module to and from the ElM.

Despite its two-level design, the ElM does not reduce the homogeneity or completeness of the repair process since it can be manipulated in a similar fashion to the other modules [1]. The ElM contains a double set of the standard features of the other modules, i.e. two central shafts and two conformal friction mechanisms. These features are positioned on top of each other, making it possible for the ElM to connect to the CH on both levels. The double mechanical connection between the ElM and the CH is necessary to ensure the stability of the agent under the large downward torque that occurs when a module is attached to the ElM at the uppermost position. Consequently, manipulating the ElM is realized in a similar fashion to manipulating two other modules on top of each other; that is by using two MaMs, one on each level.

The size of the ElM and the large number of internal features it contains (the metal threaded rod being the most significant) makes the ElM also the heaviest of all the modules (537 grams). In practice, supporting the ElM simultaneously by two vertically placed MaMs makes the module replacement process less prone to errors that may otherwise happen due to the sagging of the ElM during transport.

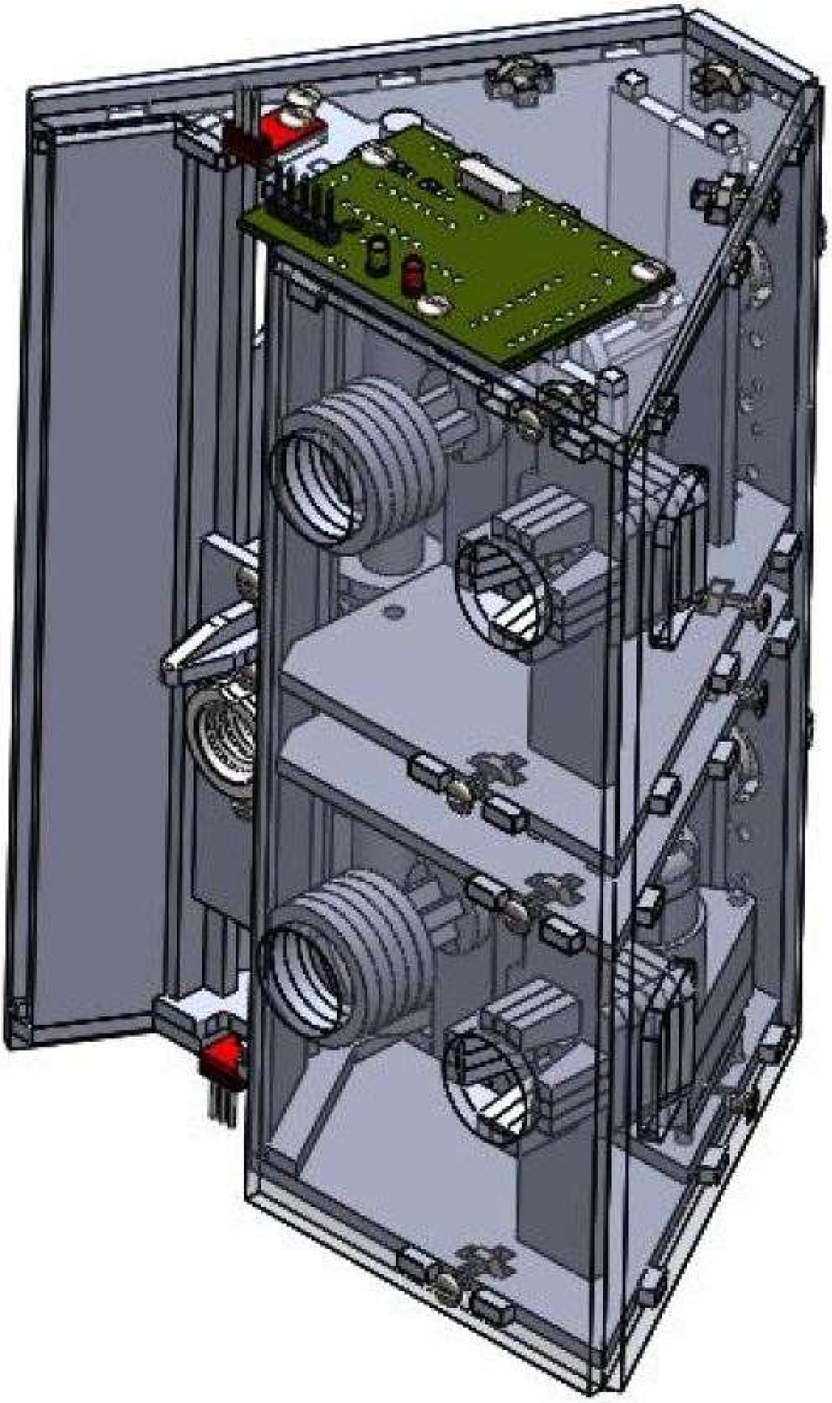


Figure 15 - Elevator Module

3. METHODS

3.1. ROBOTS IN FAULT STATES (RIF)

The Hex-DMR systems realize team-assisted repair either through the replacing a faulty module with a functional one or by simply adding a functional module onto an agent to meet a need. An agent that requires a repair is referred to as a "robot in fault state" (RIF). There are three reasons for a RIF:

Case 1: A malfunctioning component due to an unforeseen circumstance, i.e. a broken wheel

Case 2: A malfunctioning component due to the completion of its service time, i.e. a drained battery

Case 3: An underequipped agent due to changing mission requirements, i.e. need of a MaM on the upper level

While algorithms to determine the cause of a RIF - "team-assisted diagnosis" - have been proposed, this was not demonstrated and is left for future work. The next section discusses the steps of resolving a RIF assuming that the cause of the problem is known.

3.2. STEPS OF TEAM-ASSISTED REPAIR

The general algorithm that can be used to repair almost all RIFs regardless of the placement and the type of the faulty module (if any) on the RIF is presented below. One exception is a RIF caused by a broken wheel on the lower level, which requires an extra agent to support the RIF during the replacement of the target DrM. The algorithm assumes the following:

Assumption 1: The reason for the RIF is known

Assumption 2: All the agents involved are within the line of sight of each other. This is about 150cm in well-lit laboratory conditions.

Assumption 3: The functional agents involved in the repair process know their own configurations.

Assumption 4: The spare modules needed in the process are contained on a supporting agent. The same agent is also used to store the faulty modules, if there exists any.

The general algorithm is as follows:

1) A robot is assigned to conduct the repair (RCR) and another to provide support (named 'robot supporting repair' - RSR).

2) The RCR locates the RIF by searching for the alternating colored markers placed on top of the RIF's CH through the use of the CaM.

3) The RCR orients itself to center the colored marker in its field of view. It approaches the RIF up to a predetermined distance (usually two robot-lengths or about 45cm) while making corrections to its path to keep the colored marker centered.

4) In order to determine if it is facing the faulty module, the RCR compares the 4-bit barcode it sees on the face of the module to the barcode of the faulty module. This is repeated for two barcodes if the RIF has a modules on both levels.

5) If the barcode read by the CaM matches the target barcode, RCR proceeds to the next step. If not, the RCR orbits the RIF in the CCW direction until another colored marker is centered on its field of view. Step 4 is repeated.

6) Once the faulty module is located, the RCR turns a predetermined amount around its axis to orient its MaM towards the faulty module. This is necessary when the faulty module is on the upper level of the RIF. In cases where the faulty module is on the lower level of the RIF and the RCR has its MaM placed directly below the CaM this step can be omitted.

7) The RCR strafes accordingly to align its active shaft and alignment pin with the faulty module's central shaft and conformal friction mechanism, respectively.

8) The RCR drives forward to mate its components with the faulty module.

9) The active shaft on the MaM of the RCR rotates CCW until the current sensor reads a lower value indicating that the screw of the faulty module has successfully been released from the insert on the CH of the RIF.

10) The RCR drives backwards while still holding onto the released faulty module.

11) In order to store the faulty module on the RSR, the RCR repeats steps 2 and 3 on the RSR.

12)[*] Steps 4-6 are repeated on the RSR. This time instead of looking for the correct barcode, the RCR looks for the absence of a barcode indicating an open slot on the RSR that the faulty module can be stored in.

---

[*] Depending on the configuration of the agents, in some cases the target modules may need to be first moved between levels before they can be docked onto the RIF or the RSR. This requires the RSR to contain an ElM and the RCR to have a MaM on its both levels. In this case the RCR would first dock the target module onto the ElM, retrieve it from the other level and then dock it onto its target agent. An example of this procedure is provided in section 4.2 Remotely-Assisted, Mutli-Level Module Replacement.

13) Once in the right position, the RCR strafes to align the screw at the back of the faulty module it is carrying with the threaded insert on the CH of the RSR.

14) RCR drives forward pressing the faulty module to its corresponding storage spot on the RSR.

15) The active shaft on the MaM holding the faulty module rotates CW until the motor stalls, indicating that the mating on the faulty module onto the RSR is successful.

16) RCR drives backwards to disengage.

17) Steps 4-10 are repeated in order to retrieve the functional module from the RSR.

18) Steps 2-3 are repeated to relocalize the RIF.

19) Steps 4-6 are repeated, again to find the absence of the barcode.

20) Steps 13-16 are repeated to dock the functioning module onto the RIF

21) The system checks to see if the problem is alleviated. If so, all agents continue normal operation.

Note that if the RIF only requires the insertion of a new module instead of a module replacement (Case 3 in the previous section), the algorithm starts by locating the RSR and follows the steps 17-21.

4. TEAM-ASSISTED REPAIR MANEUVER DEMONSTRATIONS

Team-assisted repair on the Hex-DMR II system has been demonstrated both autonomously and teleoperated. The following sections describe the procedures.

4.1. AUTONOMOUS MODULE EXTRACTION AND INSERTION

In most cases a complete repair requires both of the module extraction and insertion maneuvers to be executed one after another, multiple times. In this demonstration, each is realized independently. (The video of the autonomous module insertion and extraction maneuvers can be found at https://www.youtube.com/watch?v=7E6h6ln7aEs&feature=youtu.be)

4.1.1. PROCEDURES

The setup for the autonomous module extraction maneuver is as follows:

a.e) The RCR is in Autonomous Repair Configuration and the RIF is in Basic Repair Configuration (see section 2.3.2. Modular Redundancy and Customizability). The CaM on the RCR is docked directly above the MaM.

b.e) The agents are placed 61 cm apart and the RCR is rotated 90° CCW from the RIF.

c.e) The faulty module (a PoM) on the RIF is located directly CW from the module closest to RCR.

d.e) "Mark3_Continuous_Extraction.ino" (see Appendix 1) and the supporting code (see Appendix 2) are uploaded onto the CoM of the RCR.

The autonomous module extraction procedure is as follows (*Figure 16*):

1.e) The RCR rotates CW until the RIF is located.

2.e) The RCR approaches the RIF and checks the barcode of the module it is facing.

3.e) The RCR orbits CW around RIF until it finds the faulty module.

4.e) The RCR drives forward, strafes right and drives forward again to dock onto the RIF. It unscrews the faulty module.

5.e) The RCR drives backwards and extracts the faulty module from the RIF.

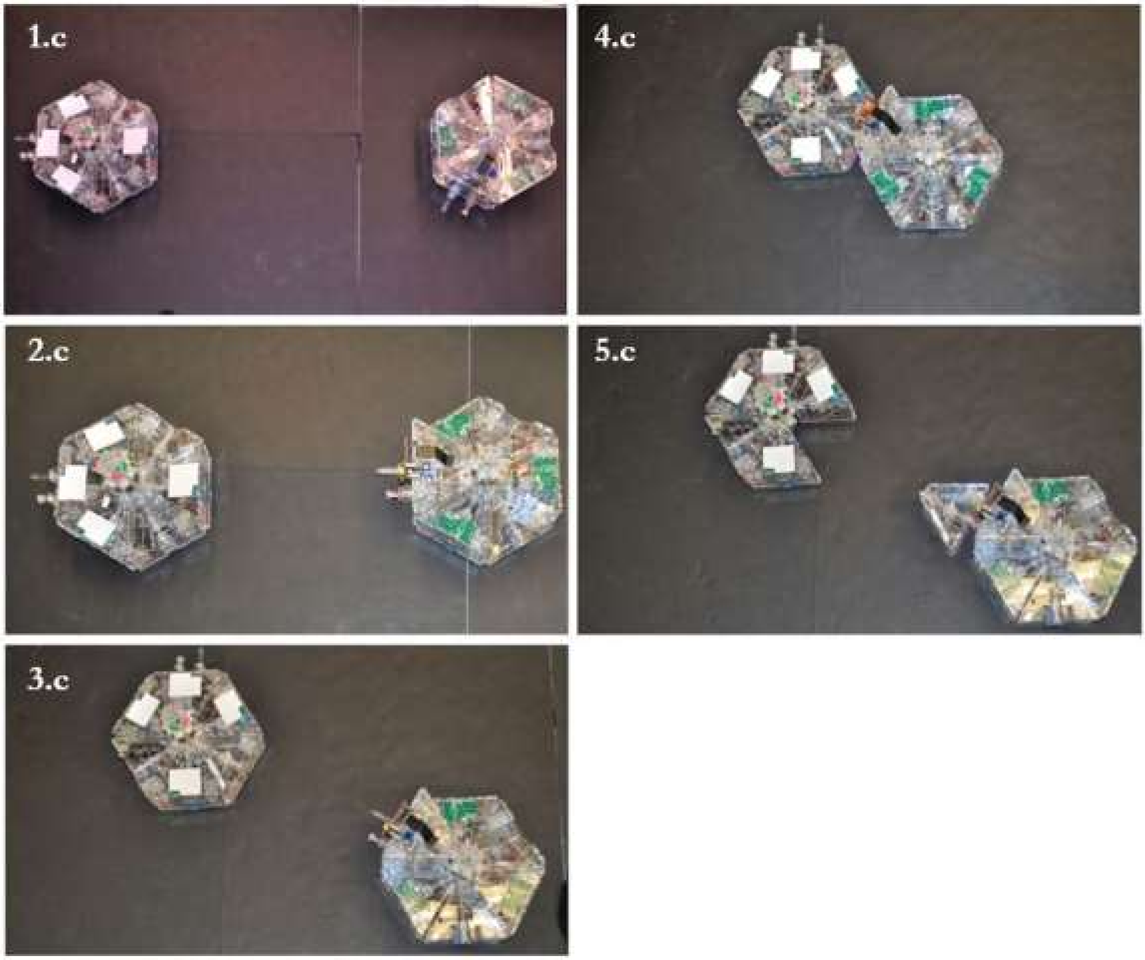


Figure 16 - Autonomous Module Extraction

Next, the autonomous module insertion maneuver is demonstrated. The setup is as follows:

a.i) The module configurations are the same as in (a.e) with the exception of the RIF missing a PoM on its lover level. This module is instead carried by the MaM of the RCR.

b.i) Same as (b.e)

c.i) The empty spot on the lower level of the RIF is located directly CCW from the module closest to RCR.

d.i) "Mark3_Continuous_Insertion.ino" (see Appendix 3) and the supporting code (see Appendix 2) are uploaded onto the CoM of the RCR.

The autonomous module insertion procedure is as follows (*Figure 17*):

1.i) The RCR rotates CW until the RIF is located.

2.i) The RCR approaches the RIF and check the barcode of the module it is facing.

3.i) The RCR orbits CCW around RIF until it finds the empty spot.

4.i) The RCR drives forward, strafes right and drives forward again to dock onto the module it is carrying onto the RIF. It screws the module onto the CH of the RIF.

5.i) The RCR drives backwards.

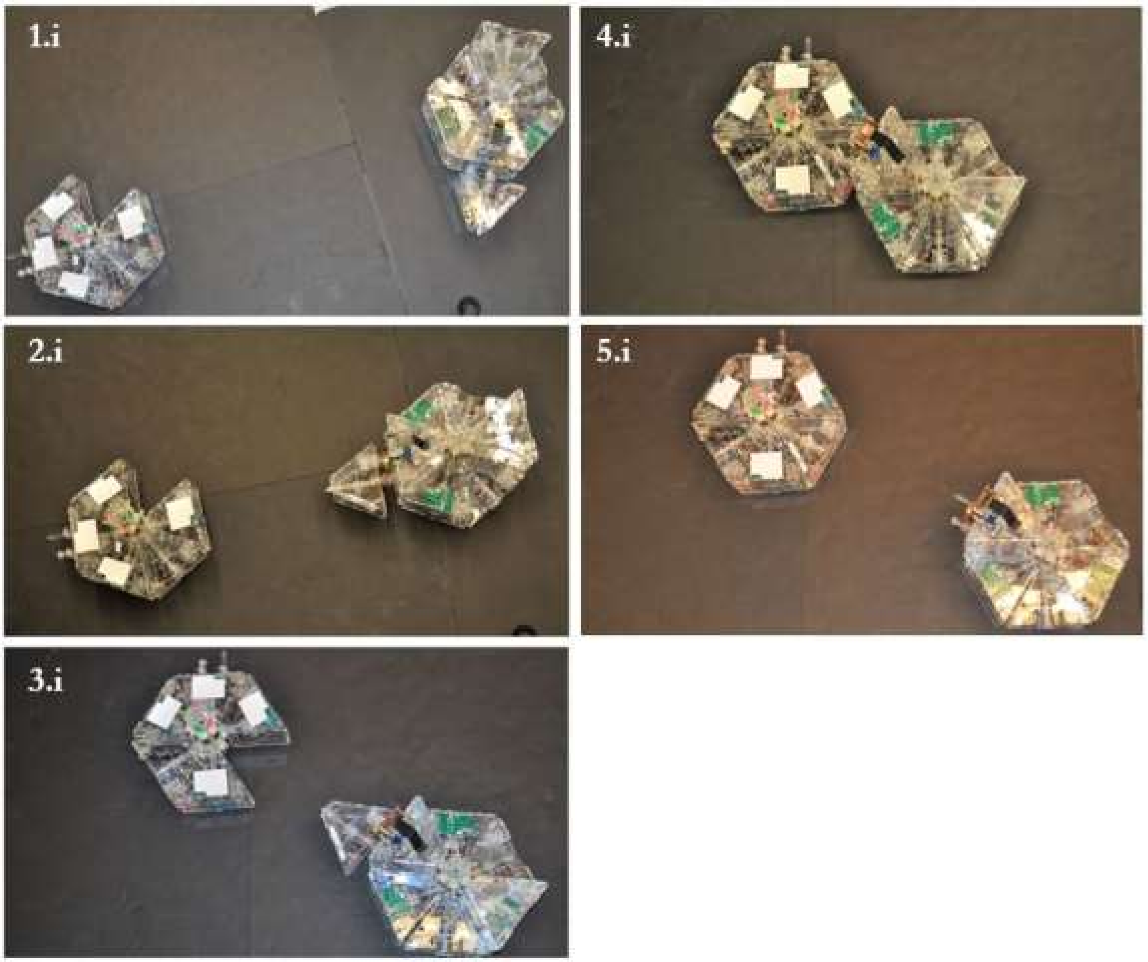


Figure 17 - Autonomous Module Insertion

4.1.2. RESULTS

Given the controlled conditions determined by the setups above, the Hex-DMR II system is shown to be able to successfully complete the autonomous module extraction and insertion maneuvers.

The success rate of the extraction maneuver is found to be higher than that of the insertion maneuver. This is due to the introduction of additional errors to the system during the insertion process when the MaM is holding another module. Practically, the slight horizontal or vertical (sagging) misalignment of the module being held with respect to the MaM can inhibit successful mating of the module to the RIF, compromising the whole maneuver. In the case of an extraction, the system is shown to be more error tolerant and robust.

4.2. REMOTELY-ASSISTED, MULTI-LEVEL MODULE REPLACEMENT

The following demonstration shows a more realistic team-assisted repair mission that requires two active agents and involves module transfers between upper and lower levels. This constitutes as a complete repair and, for now, is realized teleoperated.

4.2.1. PROCEDURE

The setup for the complete repair maneuver is as follows *(Figure 18)*:

a.c) The RCR is in Autonomous Repair Configuration with an addition of a secondary MaM on the upper level. The RIF is in Basic Repair Configuration but is missing a PoM. The RSR is in Support Configuration (contains an ElM and a spare PoM on the upper level)

b.c) The three agents are placed in arbitrary locations and orientations.

c.c) "RCControl.ino" (see Appendix 4) is uploaded onto the CoMs of all three modules.

The complete module repair procedure is as follows (*Figure 18*):

1.c) The RCR and RSR approach the RIF.

2.c) The RCR docks onto the spare PoM on the RSR.

3.c) The RCR extracts the PoM from the RSR.

4.c) The RCR aligns itself to the ElM on the RSR.

5.c) The RCR docks the PoM onto the ElM on the RSR.

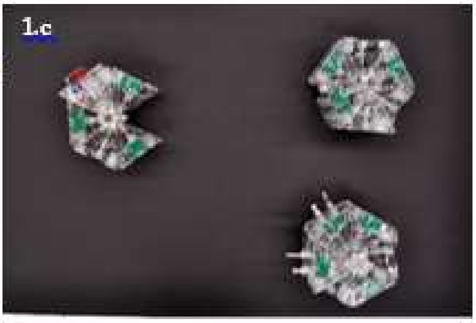
6.c) The RCR drives backwards while the ElM lowers the PoM to the lower level.

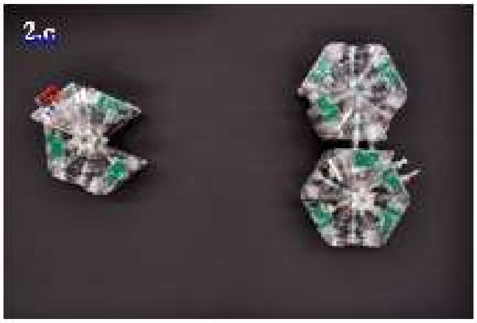
7.c) The RCR extracts the PoM from the ElM using its MaM on the lower level.

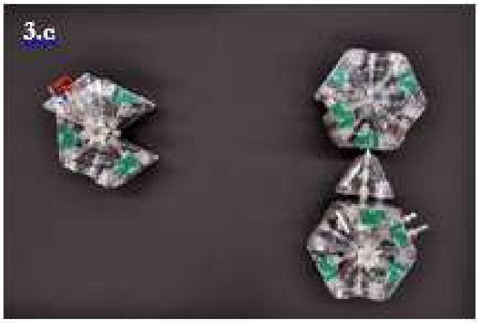
8.c) The RCR orbits around the RIF to align itself with the open spot.

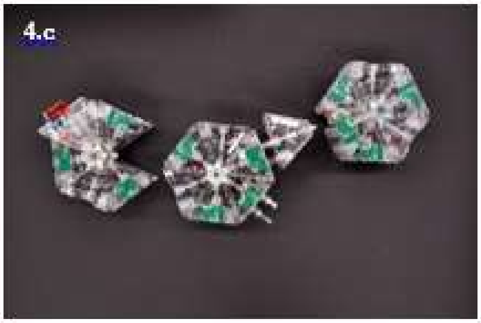
9.c) The RCR inserts the PoM onto the open spot on the RIF

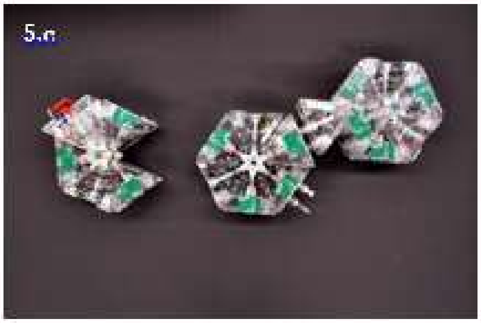
10.c) The RCR retracts and all agents resume normal operation.

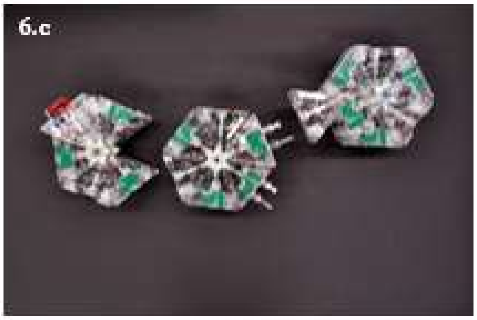

1.c

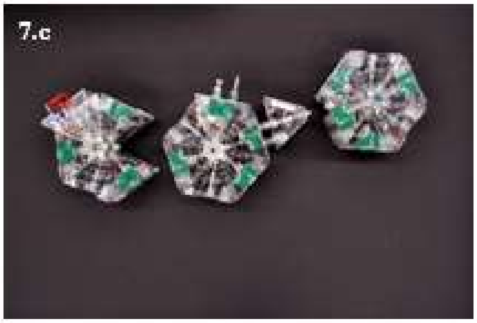

2.c

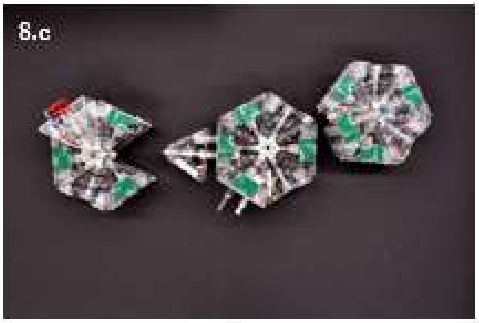

3.c

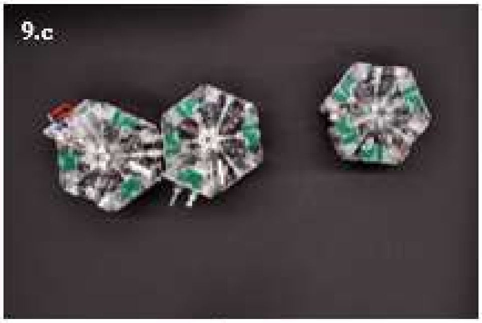

4.c


5.c


6.c


7.c


8.c


9.c

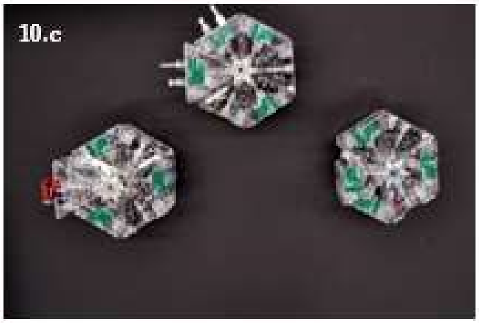

10.c

Figure 18 - Complete Teleoperated Module Repair

4.2.2. RESULTS

The execution of this maneuver involving a set of consecutive module extraction and insertion was shown repeatedly, with a success rate of 14/15. The human operator had to closely monitor the maneuver at every step to ensure that the misalignments between the modules were within the acceptable limits. In these closely controlled and monitored conditions, the design of the Hex-DMR II has been proven to be sufficiently robust to realize a relatively complex team-assisted repair maneuver.

Considering its complexity and length, the operation is prone to potential errors. Given that each step of the maneuver requires the successful completion of the previous one, failure at completing a certain step would result in the entire maneuver being unsuccessful. This was not a problem in the teleoperated case as the human operator would constantly check for and fix the errors. On the other hand, given the relatively low success rates of the previous autonomous module extraction and insertion maneuvers, the autonomous execution of this complex maneuver is very unlikely.

5. CONCLUSIONS AND FUTURE WORK

As the use of CMSs in more uncontrolled and hostile environments increase, it is necessary that they are adequately equipped to offer reliability and robustness regardless of the changing needs of the missions. One approach to meet this need is through the development of modular systems that can autonomously diagnose and alleviate the fault states that may occur. This work has presented the second version of the Hexagonal Distributed Modular Robot capable of autonomous team-assisted repair, developed to address this issue. The essay discussed the design choices that led to the development of the Hex-DMR II, introduced system components as well as their role in overall capabilities of the system, described three team-assisted maneuvers undertaken by the system and presented their results.

Hex-DMR II is shown to meet all the constraints of a heterogeneous modular self-repair system presented in the Introduction while surpassing the capabilities of its previous version on homogeneity, robustness, resolution, versatility and self-sustainability. Much of this is owed to effective mechanical design as the product of lengthy, iterative design process. Nevertheless, there is still room for improvement, especially in making the part replacement process more robust. From the perspective of mechanical design the following modifications are recommended

- Increase the error tolerance of the electrical contacts between modules and the CH through the use of a more compliant design.
- Modify the way the MaM supports a free module during the replacement process to reduce the chance of misalignment between the two modules.

One must note that, Hex-DMR II is not only a testbed for demonstrating team assisted repair, but also a platform for testing different designs capabilities a heterogeneous modular system can have. Although our work has not included such a comparison (except for a kinematic study on using different number of DrMs [6]), by switching only a part of the system, the platform can be used to test and compare different locomotion, manipulation, sensing and control systems. The author suggests the testing of more advanced sensing and control systems as the difficulties with the autonomous demonstrations can be attributed to shortcomings of the current versions of these systems. Specifically, the author recommends the following:

- Switching to a more capable camera with a larger field of view and resolution
- Adding an additional mode of sensing, such as IR or hall-effect sensors, to aid with the alignment of the agents during docking
- Switching to a more capable main processor to handle the data from these higher complexity components

These changes are also expected to assist in the demonstration of a complete team-assisted diagnosis, where the healthy agents deduce the mode of RIF through an algorithm involving communicating with and observing the RIF. When paired with an autonomous team-assisted repair process, the diagnostic process is likely to be a powerful tool making self-sustainable modular robotics feasible.

This work has presented the design as well as the results of the team-assisted repair demonstrations of the Hex-DMR II. The system is shown to be big step in the ongoing research towards demonstrating fully autonomous team-assisted robotic repair with a potential of demonstrating more advanced operations. The author believes that, despite being seemingly quite different, the principles developed through the agents of Hex-DMR II performing maneuvers in controlled laboratory conditions can one day contribute to the design of completely self-sustainable distributed systems to further extend our reach to the once uncharted environments.

APPENDICES

APPENDIX 1 - CODE FOR AUTONOMOUS MODULE EXTRACTION

```
// Mark3_Continuous_Extraction
// by Joshua Davis

#include <SoftwareSerial.h>
#include <CMUcam4.h>
#include <CMUcom4.h>
#include "variables.h"

void setup() {

  initialize_xbee(xbeeBaud);

  initialize_motors();

  initialize_camera();

  initialize_pics();

}


void loop() {
  cam.noiseFilter(2);
  COLORS[0] = COLORGREEN;
  COLORS[1] = COLORPINK;
  if (targetSearch) search();
  if (targetApproach) approach();
  if (targetOrbitInit) orbit(oCol);
  if (barcodeSearch) barcode();
  if (findNewBarcode) fnBarcode();
  if (targetDock) dock();
  if (targetActuate) actuate();
  if (Screw) manAct();
}

void search() {
  xbee.println("target search");
  cam.setTrackingWindow(0,1,159,60);
  if(dockColor) {

cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPINK
[5]);
    Color = COLORPINK;
  }
  else {

cam.trackColor(COLORGREEN[0],COLORGREEN[1],COLORGREEN[2],COLORGREEN[3],COLORGREEN[4],COLO
RGREEN[5]);
    Color = COLORGREEN;
  }

  //ROTATE UNTIL A GREEN TARGET IS OBSERVED
  cam.getTypeTDataPacket(&trackData);
  while(targetSearch){

    //IF A GREEN TARGET IS OBSERVED RECORD IN "targetFound" ORTHERWISE SPIN
    while(!targetFound) {
      xdot=0;
      ydot=0;
      tdot=1;
      driveSpd=105;
      robot_Move();
      while((trackData.y2-trackData.y1)<5) {
        cam.getTypeTDataPacket(&trackData);
      }
      driveSpd = 0;
      robot_Move();
      targetFound = true;
    }

    //IF TARGET IS FOUND ROTATE IN THE SAME DIRECTION AT A RATE PROPORTIONAL TO THE PIXEL
DISTANCE OF THE TARGET FROM THE CENTER OF THE IMAGE
    //IF THE TARGET IS CENTERED, THIS LOOP'S CONDITION IS FALSE AND THE TARGET APPROACH
LOOP'S CONDITION IS TRUE
    if (trackData.mx>round(imCenter-centerRange*imCenter) &&
trackData.mx<round(imCenter+centerRange*imCenter)){
      driveSpd = 0;
      robot_Move();
      targetSearch=false;
      targetApproach = true;

      if(skipApp) {
        if (skipDock) {
          targetDock = false;
        }
        else {
          targetDock = true;
        }
        targetApproach = false;
      }
      return;
    }
    xdot=0;
    ydot=0;
    tdot=-(imCenter-trackData.mx);
    driveSpd=round(map((long) abs(imCenter-trackData.mx), 0, (long) imCenter, 65, 75));
    robot_Move();
    cam.getTypeTDataPacket(&trackData);
    if(trackData.confidence == 0) targetFound = false;


  }
}

void approach() {
  xbee.println("target approach");
  cam.setTrackingWindow(0,0,159,60);
  if(dockColor) {

cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPINK
[5]);
    Color = COLORPINK;
  }
  else {

cam.trackColor(COLORGREEN[0],COLORGREEN[1],COLORGREEN[2],COLORGREEN[3],COLORGREEN[4],COLO
RGREEN[5]);
    Color = COLORGREEN;
  }
  //DRIVE TOWARDS TARGET UNTIL THE TARGET IS A CERTAIN DISTANCE AWAY
  /***********************************************************************/
  while(targetApproach){
    //cam.getTypeTDataPacket(&trackData);
    int height=trackData.y2-trackData.y1;
    //IF THE TARGET IS IN THE DESIRED DISTANCE RANGE, DEACTIVE THIS LOOP AND ACTIVE THE
ORBITING LOOP
    while(height<minHeight || height>maxHeight) {
      height=trackData.y2-trackData.y1;
      //IF THE TARGET BECOMES UNCENTERED DEACTIVE THIS LOOP AND REACTIVE SEARCH LOOP
      if(trackData.mx<round(imCenter-centerRange*imCenter) ||
trackData.mx>round(imCenter+centerRange*imCenter)){
        targetSearch=true;
        search();
//        targetApproach=false;
//        return;
      }
      //IF TARGET REMAINS CENTERED DRIVE ALONG THE Y AXIS WITH A SPEED AND DIRECTION
PROPORTIONAL TO THE HEIGHT OF THE TRACKED BOUNDING BOX
      xdot = 0;
      ydot = (height<minHeight)?1:-1;
      tdot = 0;
      driveSpd=round(map((long) ((height<minHeight)?abs(minHeight-height):abs(maxHeight-
height)), 1, (long) ((height<minHeight)?minHeight:(120-maxHeight)), 120, 200));
      robot_Move();
      cam.getTypeTDataPacket(&trackData);
    }

    driveSpd = 0;
    robot_Move();
    targetOrbitInit = true;
    targetSearch = false;
    targetApproach=false;
  }

}

void barcode() {
  //CHECK FOR MISSING BARCODE TO DETERMINE THE CORRECT INSERTION POINT
  xbee.println("barcode search");
  //targetApproach = true;
  approach();
  cam.setTrackingWindow(30, 70, 100, 119);

cam.trackColor(BARCODEPINK[0],BARCODEPINK[1],BARCODEPINK[2],BARCODEPINK[3],BARCODEPINK[4]
,BARCODEPINK[5]);
  cam.getTypeTDataPacket(&trackData);
  barcodeRead();
  if(arrComp(code)) {
    targetDock = true;
    barcodeSearch = false;
    findNewBarcode = false;
    cam.autoWhiteBalance(1);
    delay(5);
    cam.autoGainControl(1);
    delay(2000);
    cam.autoWhiteBalance(0);
    delay(5);
    cam.autoGainControl(0);
    delay(5);
    return;
  }
  dockColor = !dockColor;
  oCol = !oCol;
  barcodeSearch = false;
  findNewBarcode = true;
}

void fnBarcode() {
  //Add error checking
  xbee.println("find new barcode");
  BARCODEPINK[4] = 112;
  BARCODEPINK[5] = 160;
  COLORPINK[0] = 190;
  COLORPINK[2] = 182;
  COLORPINK[4] = 100;
  COLORPINK[5] = 170;
  while(findNewBarcode) {
    if (dockColor) {
      Color = COLORGREEN;
    }
    else {
      Color = COLORPINK;
    }
    xdot = -1;
    ydot = 0;
    tdot = -0.07;
    driveSpd = 190;
    robot_Move();
    apRat = 1.5;
    while(apRat<1.6 && apRat>=1.4) {
      cam.getTypeTDataPacket(&trackData);
      widthC = abs(trackData.x2-trackData.x1);
      heightC= abs(trackData.y2-trackData.y1);
      apRat = ((float)widthC)/heightC;
    }
    targetOrbit = true;
    orbit(!dockColor);
    targetSearch = true;
    skipApp = true;
    search();
    targetApproach = true;
    approach();
    findNewBarcode = false;
    barcodeSearch = true;

  }

}

void dock() {
  xbee.println("target dock");
  cam.setTrackingWindow(0,0,159,80);
  imCenter=55;
  centerRange = 0.08;
  if(!skipApp) center();
  if (dockColor == 0) {

cam.trackColor(COLORGREEN[0],COLORGREEN[1],COLORGREEN[2],COLORGREEN[3],COLORGREEN[4],COLO
RGREEN[5]);
    Color = COLORGREEN;
  }
  else {

cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPINK
[5]);
    Color = COLORPINK;
  }
  heightC=trackData.y2-trackData.y1;
  xdot=0;
  ydot=1;
  tdot=0;
  driveSpd=150;
  robot_Move();
  while(heightC<19) {
    if(abs(trackData.mx-imCenter) > round(centerRange*imCenter)){
      targetSearch=true;
      skipApp = true;
      skipDock = false;
      targetApproach=false;
      targetOrbit=false;
      targetDock=false;
      return;
    }
    cam.getTypeTDataPacket(&trackData);
    heightC=trackData.y2-trackData.y1;
  }
  driveSpd = 0;
  robot_Move();
  targetDock=false;
  targetActuate = true;

}

void actuate() {
  xbee.println("target actuate");
  cam.setTrackingWindow(0,1,159,80);
  if(dockColor == 0) {

cam.trackColor(COLORGREEN[0],COLORGREEN[1],COLORGREEN[2],COLORGREEN[3],COLORGREEN[4],COLO
RGREEN[5]);
    Color = COLORGREEN;
  }
  else {

cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPINK
[5]);
    Color = COLORPINK;
  }
  while(targetActuate) {
    heightC=trackData.y2-trackData.y1;
    imCenter = 49;
    centerRange = 0.02;
    if(abs(trackData.mx-imCenter)>3) {
      if(trackData.mx < imCenter) {
        xdot=1;
        ydot=0;
        tdot=0;
        driveSpd=160;
      }
      else if(trackData.mx > imCenter) {
        xdot=-1;
        ydot=0;
        tdot=0;
        driveSpd=160;
      }
    }
    else {
      if(heightC < 25){
        xdot=0;
        ydot=1;
        tdot=0;
        driveSpd=150;

      }
      else {
        int lastHeight = 0;
        while(heightC != lastHeight || heightC <39) {
          xdot=0;
          ydot=1;
          tdot=0;
          driveSpd=175;
          robot_Move();
          lastHeight = heightC;
          cam.getTypeTDataPacket(&trackData);
          heightC=trackData.y2-trackData.y1;
          xbee.println(heightC);

        }
        driveSpd = 0;
        robot_Move();
        targetActuate = false;
        Screw = true;
      }

    }
    //AT THE END OF THE LOOP TELL ROBOT TO EXECUTE LAST CHANGED VELOCITIES
    robot_Move();
    cam.getTypeTDataPacket(&trackData);
  }

}

void manAct() {
  cam.end();
  Serial.begin(19200);
  delay(10);
  while(Screw) {
    PICcall(manip,'u',0b11111111,1);
    delay(20);
    while(Serial.available()>0) {
      Serial.read();
    }
    while (Serial.available() == 0) {
    }
    while(Serial.available()>0) {
      char screwSig = Serial.read();
      xbee.println(screwSig);
      if(screwSig == 's') {
        xdot = 0;
        ydot = -1;
        tdot = 0;
        driveSpd = 200;
        robot_Move();
        delay(2500);
        driveSpd = 0;
        robot_Move();
        Screw = false;
        break;
      }
    }
  }

}

void orbit(int orbCol) {
  xbee.println("target orbit");
  cam.setTrackingWindow(0,0,159,50);

cam.trackColor(COLORS[orbCol][0],COLORS[orbCol][1],COLORS[orbCol][2],COLORS[orbCol][3],CO
LORS[orbCol][4],COLORS[orbCol][5]);
//  if (targetOrbitInit) {
//    Color = orbCol?COLORS[orbCol]:COLORS[!orbCol];
//  }
//  else{
//    Color = COLORS[orbCol];
//  }
  Color = COLORS[orbCol];
  cam.getTypeTDataPacket(&trackData);
  widthC = abs(trackData.x2-trackData.x1);
  while(targetOrbitInit||targetOrbit) {
    if(widthC<10) {

cam.trackColor(COLORS[orbCol][0],COLORS[orbCol][1],COLORS[orbCol][2],COLORS[orbCol][3],CO
LORS[orbCol][4],COLORS[orbCol][5]);
      cam.getTypeTDataPacket(&trackData);
      if (targetOrbit) {
        k_o = -1;
      }
      else {
        xbee.println(trackData.mx);
        k_o = ((imCenter-trackData.mx)<0)?1:-1;
      }
      xdot = k_o;
      ydot = 0;
      tdot = k_o*0.07;
      driveSpd = 200;
      robot_Move();
      //      while(widthC<10 && widthC<20) {
      //        cam.getTypeTDataPacket(&trackData);
      //        widthC = abs(trackData.x2-trackData.x1);
      //        heightC= abs(trackData.y2-trackData.y1);
      //        float ans= ((float)widthC)/heightC;
      //        xbee.println(ans);
      //        xbee.println(widthC);
      //      }
      while(apRat>=1.6 || apRat<1.4) {
        cam.getTypeTDataPacket(&trackData);
        widthC = abs(trackData.x2-trackData.x1);
        heightC= abs(trackData.y2-trackData.y1);
        apRat = ((float)widthC)/heightC;
      }
      apRat = 0.5;
      driveSpd = 0;
      robot_Move();
      if(targetOrbitInit) {
        targetOrbitInit = false;
        barcodeSearch = true;
      }
      else {
        targetOrbit = false;
        barcodeSearch = true;
      }

    }
    else {
      if(targetOrbitInit) {
        targetOrbitInit = false;
        barcodeSearch = true;
      }
      else {
        targetOrbit = false;
        barcodeSearch = true;
      }
    }
  }
}

boolean arrComp(int arr[]) {
  for(int i = 0; i<4; i++) {
    if(arr[i] != desArr[i]) {
      return false;
    }
  }
  return true;

}

void center(){
  if (dockColor == 0) {

cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPINK
[5]);
        Color = COLORPINK;
  }
  else {

cam.trackColor(COLORGREEN[0],COLORGREEN[1],COLORGREEN[2],COLORGREEN[3],COLORGREEN[4],COLO
RGREEN[5]);
    Color = COLORGREEN;
  }
  cam.getTypeTDataPacket(&trackData);
  k_o = ((imCenter-trackData.mx)<0)?1:-1;
  xdot = k_o;
  ydot = 0;
  tdot = k_o*0.07;
  driveSpd = 200;
  robot_Move();
  while(abs(imCenter-trackData.mx)>2) {
    cam.getTypeTDataPacket(&trackData);
  }
  driveSpd = 0;
  robot_Move();
}
```

APPENDIX 2 - SUPPORTING CODE FOR AUTONOMOUS OPERATION

```
// barcodeRead_DC
// by Joshua Davis

void barcodeRead(){

//cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPI
NK[5]);

cam.trackColor(BARCODEPINK[0],BARCODEPINK[1],BARCODEPINK[2],BARCODEPINK[3],BARCODEPINK[4]
,BARCODEPINK[5]);
  cam.getTypeTDataPacket(&trackData);
  int corners[4]={
      trackData.x1,trackData.y1,trackData.x2,trackData.y2};
  //xbee.println(trackData.confidence);
  while(trackData.confidence<=50) {
    corners[0] =corners[0]+2;
    corners[2] = corners[2]+2;
    cam.setTrackingWindow(corners[0],corners[1],corners[2],corners[3]);

cam.trackColor(BARCODEPINK[0],BARCODEPINK[1],BARCODEPINK[2],BARCODEPINK[3],BARCODEPINK[4]
,BARCODEPINK[5]);
    cam.getTypeTDataPacket(&trackData);
    int corners[4]={trackData.x1,trackData.y1,trackData.x2,trackData.y2};
  }
  if (trackData.confidence>50){
     int corners[4]={
      trackData.x1,trackData.y1,trackData.x2,trackData.y2};
    float lengthX= corners[2]-corners[0];
    float lengthY= corners[3]-corners[1];


    //BLOCK 1
    cam.setTrackingWindow(floor(corners[0]+0.6*lengthX),corners[1],corners[2],
floor(corners[1]+0.3*lengthY));

//cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPI
NK[5]);

cam.trackColor(BARCODEPINK[0],BARCODEPINK[1],BARCODEPINK[2],BARCODEPINK[3],BARCODEPINK[4]
,BARCODEPINK[5]);
    cam.getTypeTDataPacket(&blockData);
    if(blockData.pixels>80){
      code[0]=1;
    }
    else{
      code[0]=0;
    }
    delay(5);

    //BLOCK 2
    cam.setTrackingWindow(corners[0],floor(corners[1]+0.4*lengthY),
floor(corners[0]+0.4*lengthX), floor(corners[3]-0.4*lengthY));

//cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPI
NK[5]);

cam.trackColor(BARCODEPINK[0],BARCODEPINK[1],BARCODEPINK[2],BARCODEPINK[3],BARCODEPINK[4]
,BARCODEPINK[5]);
    cam.getTypeTDataPacket(&blockData);
    if(blockData.pixels>80){
      code[1]=1;
    }
    else{
      code[1]=0;
    }
    delay(5);

    //BLOCK 3
    cam.setTrackingWindow(floor(corners[0]+0.6*lengthX),floor(corners[1]+0.4*lengthY),
corners[2], floor(corners[3]-0.4*lengthY));

//cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPI
NK[5]);

cam.trackColor(BARCODEPINK[0],BARCODEPINK[1],BARCODEPINK[2],BARCODEPINK[3],BARCODEPINK[4]
,BARCODEPINK[5]);
    cam.getTypeTDataPacket(&blockData);
    if(blockData.pixels>80){
      code[2]=1;
    }
    else{
      code[2]=0;
    }
    delay(5);

    //BLOCK 4
    cam.setTrackingWindow(corners[0],floor(corners[3]-0.3*lengthY),
floor(corners[0]+0.4*lengthX), corners[3]);

//cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPI
NK[5]);

cam.trackColor(BARCODEPINK[0],BARCODEPINK[1],BARCODEPINK[2],BARCODEPINK[3],BARCODEPINK[4]
,BARCODEPINK[5]);
    cam.getTypeTDataPacket(&blockData);
    if(blockData.pixels>80){
      code[3]=1;
    }
    else{
      code[3]=0;
    }
    delay(5);

//    for(int i=0; i<4; i++){
//      Serial.print(code[i]);
//    }
//    delay(5);
//
//    Serial.println();
```

```
xbee.println("Code");
xbee.print(code[0]);
xbee.print(code[1]);
xbee.print(code[2]);
xbee.println(code[3]);

  }
}


// Hex_CMUcam4_DC
// by Joshua Davis

void initialize_camera(){
  xbee.println("Initializing camera...");
  cam.begin();
  cam.resetSystem();
  cam.horizontalMirror(1);
  cam.verticalFlip(1);
  cam.colorTracking(1);
  delay(5);
  cam.autoWhiteBalance(1);
  delay(5);
  cam.autoGainControl(1);
  delay(5000);
  cam.autoWhiteBalance(0);
  delay(5);
  cam.autoGainControl(0);
  delay(5);
  //cam.pollMode(1);
  //delay(5);
  //cam.lineMode(1);
  //delay(5);
}



// Hex_Drive_DC
// by Joshua Davis

void initialize_motors(){
  xbee.println("Initializing motors...");

  for(int i=0; i<3; i++){
    PICcall(wheel[i],'f', 0, 2);
  }
  PICcall(manip, 'f', 0, 2);
}

void robot_Move(){
  int L=1;
  float CF[]={
    c1, c2, c3    };

  //  if(xdot!=0 || ydot!=0 || tdot!=0){
  //    xbee.println();
  //    for(int i=0; i<sizeof(CF)/sizeof(float); i++){
  //      xbee.print(CF[i]);
  //      xbee.print(" ");
  //    }
  //  }


  float phi[3];
  //  phi[0]=(xdot+L*tdot)/(1+L);
  //  phi[1]=(-0.5*xdot+0.866*ydot+L*thetadot)/(0.5+0.866+L);
  //  phi[2]=(-0.5*xdot-0.866*ydot+L*thetadot)/(0.5+0.866+L);
  ////Hex1
  //  phi[0]=-(0.5*xdot-0.866*ydot-L*tdot);
  //  phi[1]=-(0.5*xdot+0.866*ydot-L*tdot);
  //  phi[2]=-(-xdot-L*tdot);

  //Hex2
  phi[2]=(0.5*xdot-0.866*ydot-L*tdot);
  phi[0]=(0.5*xdot+0.866*ydot-L*tdot);
  phi[1]=(-xdot-L*tdot);

  for(int i=0; i<3; i++){
    if(phi[i]>0){
      dir[i]='f';
    }
    else{
      dir[i]='r';
    }
  }

  float maxphi=0;
  for (int i=0; i<3; i++){
    if(abs(phi[i])>maxphi){
      maxphi=abs(phi[i]);
    }
  }

  float gain=driveSpd/maxphi;


  for(int i=0;i<3;i++){
    PICcall(wheel[i],dir[i], (byte) round(CF[i]*gain*abs(phi[i])),i);
  }

//  xbee.print((byte) round(CF[0]*gain*abs(phi[0])));
//  xbee.print("   ");
//  xbee.print((byte) round(CF[1]*gain*abs(phi[1])));
//  xbee.print("   ");
//  xbee.println((byte) round(CF[2]*gain*abs(phi[2])));
}





// Hex_PIC_DC
// by Joshua Davis

void initialize_pics(){
  Serial.begin(19200);
  xbee.println("Initializing PICs...");
  Serial.write(uni);
  delay(1);
  Serial.write('x');
  delay(1);
  cam.LEDOn(2);
}


void PICcall(byte address, char Direct, byte dutyCycle, int last) {
  if(last == 0) {
    Serial.write('\r');
    delay(2);
    Serial.write(address);
    Serial.write(Direct);
    Serial.write(dutyCycle);
    delay(4);
  }
  else if (last == 2) {
    Serial.write(address);
    Serial.write(Direct);
    Serial.write(dutyCycle);
    delay(2);
    if(targetActuate) {
      cam.setTrackingWindow();
    }
    else if(targetDock||findNewBarcode){
      cam.setTrackingWindow(0,1,159,80);
    }
    else {
      cam.setTrackingWindow(0,1,159,60);
    }
    cam.trackColor(Color[0],Color[1],Color[2],Color[3],Color[4],Color[5]);
  }
  else {
    Serial.write(address);
    Serial.write(Direct);
    Serial.write(dutyCycle);
    delay(4);
  }

}
```

APPENDIX 3 - CODE FOR AUTONOMOUS MODULE INSERTION

```
// Mark3_Continuous_Insertion
// by Joshua Davis

#include <SoftwareSerial.h>
#include <CMUcam4.h>
#include <CMUcom4.h>
#include "variables.h"

void setup() {

  initialize_xbee(xbeeBaud);

  initialize_motors();

  initialize_camera();

  initialize_pics();

}

//Look into why the tracking window isn't setting properly

void loop() {
  cam.noiseFilter(2);
  COLORS[0] = COLORGREEN;
  COLORS[1] = COLORPINK;
  if (targetSearch) search();
  if (targetApproach) approach();
  if (targetOrbitInit) orbit(oCol);
  if (barcodeSearch) barcode();
  if (findNewBarcode) fnBarcode();
  if (targetDock) dock();
  if (targetActuate) actuate();
  if (unScrew) manAct();
}

void search() {
  xbee.println("target search");
  cam.setTrackingWindow(0,1,159,60);
  if(dockColor) {

cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPINK
[5]);
    Color = COLORPINK;
  }
  else {

cam.trackColor(COLORGREEN[0],COLORGREEN[1],COLORGREEN[2],COLORGREEN[3],COLORGREEN[4],COLO
RGREEN[5]);
    Color = COLORGREEN;
  }

  //ROTATE UNTIL A GREEN TARGET IS OBSERVED
  cam.getTypeTDataPacket(&trackData);
  while(targetSearch){

    //IF A GREEN TARGET IS OBSERVED RECORD IN "targetFound" ORTHERWISE SPIN
    while(!targetFound) {
      xdot=0;
      ydot=0;
      tdot=1;
      driveSpd=105;
      robot_Move();
      while((trackData.y2-trackData.y1)<5) {
        cam.getTypeTDataPacket(&trackData);
      }
      driveSpd = 0;
      robot_Move();
      targetFound = true;
    }

    //IF TARGET IS FOUND ROTATE IN THE SAME DIRECTION AT A RATE PROPORTIONAL TO THE PIXEL
DISTANCE OF THE TARGET FROM THE CENTER OF THE IMAGE
    //IF THE TARGET IS CENTERED, THIS LOOP'S CONDITION IS FALSE AND THE TARGET APPROACH
LOOP'S CONDITION IS TRUE
    if (trackData.mx>round(imCenter-centerRange*imCenter) &&
trackData.mx<round(imCenter+centerRange*imCenter)){
      driveSpd = 0;
      robot_Move();
      targetSearch=false;
      targetApproach = true;

      if(skipApp) {
        if (skipDock) {
          targetDock = false;
        }
        else {
          targetDock = true;
        }
        targetApproach = false;
      }
      return;
    }
    xdot=0;
    ydot=0;
    tdot=-(imCenter-trackData.mx);
    driveSpd=round(map((long) abs(imCenter-trackData.mx), 0, (long) imCenter, 65, 75));
    robot_Move();
    cam.getTypeTDataPacket(&trackData);
    if(trackData.confidence == 0) targetFound = false;


  }
}

void approach() {
  xbee.println("target approach");
  cam.setTrackingWindow(0,1,159,60);
  if(dockColor) {

cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPINK
[5]);
    Color = COLORPINK;
  }
  else {

cam.trackColor(COLORGREEN[0],COLORGREEN[1],COLORGREEN[2],COLORGREEN[3],COLORGREEN[4],COLO
RGREEN[5]);
    Color = COLORGREEN;
  }
  //DRIVE TOWARDS TARGET UNTIL THE TARGET IS A CERTAIN DISTANCE AWAY
  /*****************************************************************************/
  while(targetApproach){
    //cam.getTypeTDataPacket(&trackData);
    int height=trackData.y2-trackData.y1;

    //IF THE TARGET IS IN THE DESIRED DISTANCE RANGE, DEACTIVE THIS LOOP AND ACTIVE THE
ORBITING LOOP
    while(height<minHeight || height>maxHeight) {
      height=trackData.y2-trackData.y1;
      //IF THE TARGET BECOMES UNCENTERED DEACTIVE THIS LOOP AND REACTIVE SEARCH LOOP
      if(trackData.mx<round(imCenter-centerRange*imCenter) ||
trackData.mx>round(imCenter+centerRange*imCenter)){
        targetSearch=true;
        search();
//        targetApproach=false;
//        return;
      }
      //IF TARGET REMAINS CENTERED DRIVE ALONG THE Y AXIS WITH A SPEED AND DIRECTION
PROPORTIONAL TO THE HEIGHT OF THE TRACKED BOUNDING BOX
      xdot = 0;
      ydot = (height<minHeight)?1:-1;
      tdot = 0;
      driveSpd=round(map((long) ((height<minHeight)?abs(minHeight-height):abs(maxHeight-
height)), 1, (long) ((height<minHeight)?minHeight:(120-maxHeight)), 120, 200));
      robot_Move();
      cam.getTypeTDataPacket(&trackData);
    }

    driveSpd = 0;
    robot_Move();
    targetOrbitInit = true;
    targetSearch = false;
    targetApproach=false;
  }

}

void barcode() {
  //CHECK FOR MISSING BARCODE TO DETERMINE THE CORRECT INSERTION POINT
  xbee.println("barcode search");
  //targetApproach = true;
  approach();
  cam.setTrackingWindow(40, 70, 90, 119);

cam.trackColor(BARCODEPINK[0],BARCODEPINK[1],BARCODEPINK[2],BARCODEPINK[3],BARCODEPINK[4]
,BARCODEPINK[5]);
  cam.getTypeTDataPacket(&trackData);
  if(trackData.confidence == 0) {
    targetDock = true;
    barcodeSearch = false;
    findNewBarcode = false;
    cam.autoWhiteBalance(1);
    delay(5);
    cam.autoGainControl(1);
    delay(2000);
    cam.autoWhiteBalance(0);
    delay(5);
    cam.autoGainControl(0);
    delay(5);
    return;
    //    cam.setTrackingWindow(0,0,159,60);
    //
cam.trackColor(COLORGREEN[0],COLORGREEN[1],COLORGREEN[2],COLORGREEN[3],COLORGREEN[4],COLO
RGREEN[5]);
    //    cam.getTypeTDataPacket(&trackData);
    //    int mxG = trackData.mx;
    //
cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPINK
[5]);
    //    cam.getTypeTDataPacket(&trackData);
    //    int mxP = trackData.mx;
    //    dockColor = ((abs(mxP-imCenter) > abs(mxG-imCenter))?0:1);
    //    xbee.print("dc = ");
    //    xbee.println(dockColor);
  }
  //  cam.setTrackingWindow(0,0,159,60);
  //
cam.trackColor(COLORGREEN[0],COLORGREEN[1],COLORGREEN[2],COLORGREEN[3],COLORGREEN[4],COLO
RGREEN[5]);
  //  cam.getTypeTDataPacket(&trackData);
  //  int mxG = trackData.mx;
  //  xbee.println(mxG);
  //
cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPINK
[5]);
  //  cam.getTypeTDataPacket(&trackData);
  //  int mxP = trackData.mx;
  //  xbee.println(mxP);
  //  dockColor = ((abs(mxP-imCenter) > abs(mxG-imCenter))?0:1);
  //  xbee.print("dc = ");
  //  xbee.println(dockColor);
  dockColor = !dockColor;
  oCol = !oCol;
  barcodeSearch = false;
  findNewBarcode = true;
}

void fnBarcode() {
  //Add error checking
  xbee.println("find new barcode");
  BARCODEPINK[4] = 112;
  BARCODEPINK[5] = 160;
  COLORPINK[0] = 190;
  COLORPINK[2] = 182;
  COLORPINK[4] = 100;
  COLORPINK[5] = 170;
  while(findNewBarcode) {
    if (dockColor) {
      Color = COLORGREEN;
    }
    else {
      Color = COLORPINK;
    }
    xdot = -1;
    ydot = 0;
    tdot = -0.065;
    driveSpd = 210; //190 without module
    robot_Move();
    apRat = 1.5;
    while(apRat<1.6 && apRat>=1.4) {
      cam.getTypeTDataPacket(&trackData);
      widthC = abs(trackData.x2-trackData.x1);
      heightC= abs(trackData.y2-trackData.y1);
      apRat = ((float)widthC)/heightC;
    }
    targetOrbit = true;
    orbit(!dockColor);
    targetSearch = true;
    skipApp = true;
    search();
    targetApproach = true;
    approach();
    findNewBarcode = false;
    barcodeSearch = true;

  }

}

void dock() {
  xbee.println("target dock");
  cam.setTrackingWindow(0,0,159,80);
  imCenter=55;
  centerRange = 0.08;
  if(!skipApp) center();
  if (dockColor == 0) {

cam.trackColor(COLORGREEN[0],COLORGREEN[1],COLORGREEN[2],COLORGREEN[3],COLORGREEN[4],COLO
RGREEN[5]);
    Color = COLORGREEN;
  }
  else {

cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPINK
[5]);
    Color = COLORPINK;
  }
  heightC=trackData.y2-trackData.y1;
  xdot=0;
  ydot=1;
  tdot=0;
  driveSpd=150;
  robot_Move();
  while(heightC<22) {
    if(abs(trackData.mx-imCenter) > round(centerRange*imCenter)){
      targetSearch=true;
      skipApp = true;
      skipDock = false;
      targetApproach=false;
      targetOrbit=false;
      targetDock=false;
      return;
    }
    cam.getTypeTDataPacket(&trackData);
    heightC=trackData.y2-trackData.y1;
  }
  driveSpd = 0;
  robot_Move();
  targetDock=false;
  targetActuate = true;

}

void actuate() {
  xbee.println("target actuate");
  cam.setTrackingWindow(0,1,159,80);
  if(dockColor == 0) {

cam.trackColor(COLORGREEN[0],COLORGREEN[1],COLORGREEN[2],COLORGREEN[3],COLORGREEN[4],COLO
RGREEN[5]);
    Color = COLORGREEN;
  }
  else {

cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPINK
[5]);
    Color = COLORPINK;
  }
  while(targetActuate) {
    heightC=trackData.y2-trackData.y1;
    imCenter = 42;
    centerRange = 0.02;
    xbee.println(trackData.mx);
    if(abs(trackData.mx-imCenter)>3) {
      if(trackData.mx < imCenter) {
        xdot=1;
        ydot=0;
        tdot=0;
        driveSpd=170;
      }
      else if(trackData.mx > imCenter) {
        xdot=-1;
        ydot=0;
        tdot=0;
        driveSpd=170;
      }
    }
    else {
      //Calibrate this number
      if(heightC < 27){
        xdot=0;
        ydot=1;
        tdot=0;
        driveSpd=150;
      }
      else {
        int lastHeight = 0;
        while(heightC != lastHeight || heightC <37) {
          xdot=0;
          ydot=1;
          tdot=0;
          driveSpd=175;
          robot_Move();
          lastHeight = heightC;
          cam.getTypeTDataPacket(&trackData);
          heightC=trackData.y2-trackData.y1;
          xbee.println(heightC);

        }
        driveSpd = 0;
        robot_Move();
        targetActuate = false;
        unScrew = true;
      }

    }
    //AT THE END OF THE LOOP TELL ROBOT TO EXECUTE LAST CHANGED VELOCITIES
    robot_Move();
    cam.getTypeTDataPacket(&trackData);
  }

}

void manAct() {
  cam.end();
  Serial.begin(19200);
  delay(10);
  while(unScrew) {
    PICcall(manip,'p',0b11111111,1);
    delay(20);
    while(Serial.available()>0) {
      Serial.read();
    }
    xdot = 0;
    ydot = 1;
    tdot = 0;
    driveSpd = 120;
    robot_Move();
    delay(60);
    while(Serial.available()>0) {
      Serial.read();
    }
    while (Serial.available() == 0) {
    }
    driveSpd = 120;
    robot_Move();
    delay(60);
    while(Serial.available()>0) {
      Serial.read();
    }
```

```
//xbee.println(Serial.read());
    PICcall(manip,'p',0b11111111,1);
    delay(20);
    while(Serial.available()>0) {
      Serial.read();
    }
    while (Serial.available() == 0) {
    }

    while(Serial.available()>0) {
      char screwSig = Serial.read();
      xbee.println(screwSig);
      if(screwSig == 's') {
        xdot = 0;
        ydot = -1;
        tdot = 0;
        driveSpd = 200;
        robot_Move();
        delay(2500);
        driveSpd = 0;
        robot_Move();
        unScrew = false;
        break;

      }
    }
  }

}

void orbit(int orbCol) {
  xbee.println("target orbit");
  cam.setTrackingWindow(0,0,159,40);

cam.trackColor(COLORS[orbCol][0],COLORS[orbCol][1],COLORS[orbCol][2],COLORS[orbCol][3],CO
LORS[orbCol][4],COLORS[orbCol][5]);
//  if (targetOrbitInit) {
//    Color = orbCol?COLORS[orbCol]:COLORS[!orbCol];
//  }
//  else{
//    Color = COLORS[orbCol];
//  }
  Color = COLORS[orbCol];
  cam.getTypeTDataPacket(&trackData);
  widthC = abs(trackData.x2-trackData.x1);
  while(targetOrbitInit||targetOrbit) {
    if(widthC<10) {

cam.trackColor(COLORS[orbCol][0],COLORS[orbCol][1],COLORS[orbCol][2],COLORS[orbCol][3],CO
LORS[orbCol][4],COLORS[orbCol][5]);
      cam.getTypeTDataPacket(&trackData);
      if (targetOrbit) {
        k_o = -1;
      }
      else {
        k_o = ((imCenter-trackData.mx)<0)?1:-1;
      }
      xdot = k_o;
      ydot = 0;
      tdot = k_o*0.065;
      driveSpd = 220; //190 without module
      robot_Move();
      //      while(widthC<10 && widthC<20) {
      //        cam.getTypeTDataPacket(&trackData);
      //        widthC = abs(trackData.x2-trackData.x1);
      //        heightC= abs(trackData.y2-trackData.y1);
      //        float ans= ((float)widthC)/heightC;
      //        xbee.println(ans);
      //        xbee.println(widthC);
      //      }
      while(apRat>=1.6 || apRat<1.4) {
        cam.getTypeTDataPacket(&trackData);
        widthC = abs(trackData.x2-trackData.x1);
        heightC= abs(trackData.y2-trackData.y1);
        apRat = ((float)widthC)/heightC;
      }
      apRat = 0.5;
      driveSpd = 0;
      robot_Move();
      if(targetOrbitInit) {
        targetOrbitInit = false;
        barcodeSearch = true;
      }
      else {
        targetOrbit = false;
        barcodeSearch = true;
      }

    }
    else {
      if(targetOrbitInit) {
        targetOrbitInit = false;
        barcodeSearch = true;
      }
      else {
        targetOrbit = false;
        barcodeSearch = true;
      }
    }
  }
}

void center(){
  if (dockColor == 0) {

cam.trackColor(COLORGREEN[0],COLORGREEN[1],COLORGREEN[2],COLORGREEN[3],COLORGREEN[4],COLO
RGREEN[5]);
    Color = COLORGREEN;
  }
  else {

cam.trackColor(COLORPINK[0],COLORPINK[1],COLORPINK[2],COLORPINK[3],COLORPINK[4],COLORPINK
[5]);
    Color = COLORPINK;
  }
  cam.getTypeTDataPacket(&trackData);
  k_o = ((imCenter-trackData.mx)<0)?1:-1;
  xdot = k_o;
  ydot = 0;
  tdot = k_o*0.065;
  driveSpd = 220;
  robot_Move();
  while(abs(imCenter-trackData.mx)>2) {
    cam.getTypeTDataPacket(&trackData);
  }
  driveSpd = 0;
  robot_Move();
}
```

APPENDIX 4 - CODE FOR TELEOPERATED OPERATION

```
// RCControl
// by Joshua Davis and Yunuscan Sevimli

#include <SoftwareSerial.h>
#include <CMUcam4.h>
#include <CMUcom4.h>
SoftwareSerial mySer(3,2);
byte w1 = 0b00100100;
byte w2 = 0b00101000;
byte w3 = 0b00111111;
byte manip = 0b00100001;
byte uadr = 0b00111110;
char dir, dir1, dir2, dir3, dir4, duty, duty1, duty2, duty3;
int counter = 0;

//Camera Setup
CMUcam4 cam(CMUCOM4_SERIAL);
CMUcam4_tracking_data_t t_data;
int getPixels,getConfidence,getXcenter,getYcenter,getXtopleft;
int getYtopleft,getXbottomright,getYbottomright;
int vMin,vMax,yMin,yMax,uMin,uMax;
int RxCen, newval;

void setup() {
  //Remove
  cam.begin();
  cam.resetSystem();
  cam.autoWhiteBalance(1);
  delay(5);
  cam.autoGainControl(1);
  delay(7000);
  cam.autoWhiteBalance(0);
  delay(5);
  cam.autoGainControl(0);
  delay(5);
  cam.pollMode(1);
  delay(5);
  cam.lineMode(1);
  delay(5);
  cam.colorTracking(1);
  delay(1000);
  Serial.begin(19200);
  mySer.begin(19200);
  delay(5);
  reset();
  cam.LEDOn(2);
  mySer.println("Ready.");
}

void loop() {
  while(counter == 1)
  {
    trackcolor('r');
    delay(500);
    //counter++;
  }

  while (mySer.available()) {
    char cmd = mySer.read();
    mySer.write(cmd);
    switch (cmd) {
      case 'c': circle(true, 20); break; //rotate clockwise
      case 'x': circle(false, 20); break; //rotate counterclockwise
      case 'u': manipulator(true, 20); break; //move manipulator down
      case 'j': manipulator(false, 20); break;//move manipulator up
      case 's': straight(false, 20); break; //move backward
      case 'w': straight(true, 20);  break;//move forward
      case 'd': sideways(true, 20); break;//strafe right
      case 'a': sideways(false, 20); break; //strafe left
      case 'q': testall(); break;
      case 'y': stopall(); break;
      case 'p': reset(); break;
    }
  }
}

void circle(boolean clockwise, long duration) {
  if (clockwise) {
    dir = 'f';
    duty = 0b11111111;

    PICcall(uadr,dir,duty);
    delay(duration);
    stopall();
  }
  else {
    dir = 'r';
    duty= (byte) 0b11111111;

    PICcall(uadr,dir,duty);
    delay(duration);
    stopall();
  }
}

void manipulator(boolean up, long duration) {
  if (up) {
    dir = 'f';
    duty= (byte) 0b11111111;

    PICcall(manip,dir,duty);
    delay(duration);
    stopall();
  }
  else {
    dir = 'r';
    duty= (byte) 0b11111111;

    PICcall(manip,dir,duty);
    delay(duration);
    stopall();
  }
}

void straight(boolean forward, long duration) {
  if (forward) {
    dir1 = 'r';
    duty1= (byte) 0b11111111;
    dir2 = 'r';
    duty2= (byte) 0b00000000;
    dir3 = 'f';
    duty3= (byte) 0b11111111;

    //Wheel 1
    PICcall(w1,dir1,duty1);

    //Wheel 2
    PICcall(w2,dir2,duty2);

    //Wheel 3
    PICcall(w3,dir3,duty3);
    delay(duration);
    stopall();
  }
  else {
    dir1 = 'f';
    duty1= (byte) 0b11111111;
    dir2 = 'r';
    duty2= (byte) 0b00000000;
    dir3 = 'r';
    duty3= (byte) 0b11111111;

    //Wheel 1
    PICcall(w1,dir1,duty1);

    //Wheel 2
    PICcall(w2,dir2,duty2);

    //Wheel 3
    PICcall(w3,dir3,duty3);
    delay(duration);
    stopall();
  }
}

void sideways(boolean left, long duration) {
  if (left) {
    dir1 = 'f';
    duty1= (byte) 0b01111111;
    dir2 = 'r';
    duty2= (byte) 0b11111111;
    dir3 = 'f';
    duty3= (byte) 0b01111111;

     //Wheel 1
    PICcall(w1,dir1,duty1);

    //Wheel 2
    PICcall(w2,dir2,duty2);

    //Wheel 3
    PICcall(w3,dir3,duty3);
    delay(duration);
    stopall();
  }
  else {
    dir1 = 'r';
    duty1= (byte) 0b01111111;
    dir2 = 'f';
    duty2= (byte) 0b11111111;
    dir3 = 'r';
    duty3= (byte) 0b01111111;

    //Wheel 1
    PICcall(w1,dir1,duty1);

    //Wheel 2
    PICcall(w2,dir2,duty2);

    //Wheel 3
    PICcall(w3,dir3,duty3);
    delay(duration);
    stopall();
  }
}

void stopall() {
  dir = 'f';
  duty= (byte) 0b00000000;

  PICcall(uadr,dir,duty);
  PICcall(manip,dir,duty);
}

void reset() {
  Serial.write(uadr);
  delay(1);
  Serial.write('x');
  delay(1);
}

void PICcall(byte address, char Direct, byte dutyCycle) {
  Serial.write(address);
  Serial.write(Direct);
  Serial.write(dutyCycle);
  delay(4);
}

void testall() {
  circle(true, 6000);
  delay(1000);
  circle(false, 6000);
  delay(1000);
  manipulator(true, 500);
  delay(1000);
  manipulator(false, 500);
  delay(1000);
  straight(true, 2000);
  delay(1000);
  straight(false, 2000);
  delay(1000);
  sideways(true, 2000);
  delay(1000);
  sideways(false, 2000);
  delay(1000);
}

//Camera Functions
void trackcolor(char color)
{
  if (color == 'r')  //red
  {
    vMin = 200;
    vMax = 250;
    yMin = 50;
    yMax = 150;
    uMin = 0;
    uMax = 255;
  }
  else if (color == 'g')  //green
  {
    vMin = 0;
    vMax = 80;
    yMin = 110;
    yMax = 200;
    uMin = 0;
    uMax = 255;
  }
  cam.trackColor(vMin,vMax,yMin,yMax,uMin,uMax);
  cam.getTypeTDataPacket(&t_data); // Get a statistics packet.
  getPixels = t_data.pixels;
  getConfidence = t_data.confidence;
  getXcenter = t_data.mx;
  getYcenter = t_data.my;
  getXtopleft = t_data.x1;
  getYtopleft = t_data.y1;
  getXbottomright = t_data.x2;
  getYbottomright = t_data.y2;
  Serial.println(getXcenter);
}
```

BIBLIOGRAPHY

1. Ackerman, M., and Chirikjian, G., 2012. "Hex-DMR: A Modular Robotic Test-bed for Demonstrating Team Repair". In Proc. of the 2012 IEEE ICRA, pp. 4464-4469.
2. Baca J et al 2014. "ModRED: Hardware Design and Reconfiguration Planning for a High Dexterity Modular Self-Reconfigurable Robot for Extra-Terrestrial Exploration". Robot Auton Syst 62(7):1002-1015.
3. Bererton C, Khosla PK, 2001. "Towards a Team of Robots with Repair Capabilities". In: Proc. of the IEEE 2001 ICRA, pp. 2923-2928.
4. Chirikjian, G, 1994. "Kinematics of a Metamorphic Robotic System". In Proc. of the 1994 IEEE ICRA, San Diego 1994, pp 449–55.
5. Davis, J., Sevimli, Y., Ackerman, K., Chirikjian, G., 2015 "A Robot Capable of Autonomous Robotic Team Repair: The Hex-DMR II System". In Proc. of the 3rd ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots 2015. (accepted)
6. Davis, J., Sevimli, Y., Eldridge, B., Chirikjian, G., 2015 "Module Design and Functionally Non-Isomorphic Configurations of the Hex-DMR II System". In Proc. of the ASME IDETC/CIE 2015. (accepted)
7. Fei, Y., Wang, C., 2014. "Self-Repairing Algorithm of Lattice-Type Self-Reconfigurable Modular Robots". Journal of Intelligent & Robotic Systems, 75(2), pp. 193–203.
8. Fukuda, T, Nakagawa, S, 1988. "Dynamically reconfigurable robotic system, Robotics and Automation". In Proc. of the 1988 IEEE International Conference, Philadelphia, 24–29 Apr 1988, pp 1581–1586, vol 3.
9. Gilpin, K., Kotay, K., Rus, D., and Vasilescu, I., 2008. "Miche: Modular Shape Formation by Self-disassembly". The International Journal of Robotics Research (IJRR), 27, pp. 345–372.

10. Howard, A., Parker, L., and Sukhatme, G., 2013. "Experiments with a Large Heterogeneous Mobile Robot Team: Exploration, Mapping, Deployment and Detection". IJRR, 25(5-6), pp. 431–447.

11. Kurokawa H, Kamimura A, Tomita K, 2012. "Self-Assembly and Self-Reproduction by a M-TRAN Modular Robotic System". In: Proc. of the 2012 International Symposium on Distributed Autonomous Robotic Systems, pp. 205-218.

12. Kutzer MDM et al, 2008. "Toward Cooperative Team-Diagnosis in Multi-Robot Systems". IJRR 27(9):1069-1090.

13. Kutzer MDM et al, 2010. "Design of a New Independently-Mobile Reconfigurable Modular Robot". In Proc. of the IEEE 2012 ICRA, pp. 2758-2764.

14. Murata, S, Kurokawa, H, Kokaji, S, 1994. "Self-Assembling Machine". In Proc. of the 1994 IEEE ICRA, San Diego, May 1994, pp 441-448.

15. Roehr, T. M., Cordes, F, Kirchner, F, Ahrns, I, 2010. "Cooperative Docking Procedures for a Lunar Mission". 41st International Symposium on Robotics 6th German Conference on Robotics (ROBOTIK) , vol., no., pp.1-8, 7-9 June 2010.

16. Rubenstein M et al, 2014. "Kilobot: A Low Cost Robot with Scalable Operations Designed for Collective Behaviors". Robot Auton Syst 62(7):966-975.

17. Schwager, M., Dames, P., Rus, D. R., and Kumar, V., 2011. "A Multi-robot Control Policy for Information Gathering in the Presence of Unknown Hazards". In Proc. of the 2011 International Symposium on Robotics Research.

18. Sprowitz A, et al, 2014. "Roombots: A Hardware Perspective on 3D Self-Reconfiguration and Locomotion with a Homogeneous Modular Robot". Robot Auton Syst 62(7):1016-1033.

19. Wei, H., Chen, Y., Tan, J., and Wang, T., 2011. "Sambot: A Self-Assembly Modular Robot System". IEEE/ASME Transactions of Mechatronics, 16(4), pp. 745–757.

20. Whitcomb, L.L., 2000. “Underwater robotics: out of the research laboratory and into the field”. In Proc. IEEE ICRA 2000 pp.709-716.

21. Wolfe, KC et al, 2010. “Trajectory Generation and Steering Optimization for Self-Assembly of a Modular Robotic System”. In Proc. of the IEEE 2010 ICRA, pp. 4996-5001.

22. Yamaguchi, H, Arai, T, Beni, G, 2001. “A Distributed Control Scheme for Multiple Robotic Vehicles to Make Group Formations”. Robot Auton Syst 36(4):125-147.

23. Yim, M, 1994. “Locomotion with a Unit Modular Reconfigurable Robot”. PhD Thesis, Stanford University.

24. Yim, M, White, P, Park, M, Sastra, J, 2009. “Modular Self-Reconfigurable Robots Encyclopedia of Complexity and Systems Science”. Meyers, R. A. 5618-5631 Springer New York 2009-01-01.

CURRICULUM VITAE

EDUCATION

- **MS in Mechanical Engineering, JHU**, Baltimore, MD Dec 2015 (expected)
- **BS in Mechanical Engineering, JHU**, Baltimore, MD May 2014

PUBLICATIONS

- Davis, J. D., Sevimli, Y., Ackerman, M. K., and Chirikjian, G. S. (2015). **"A Robot Capable of Robotic Team Repair: The Hex-DMR II System"** 3rd IEEE/IFToMM International Conference on Reconfigurable Mechanisms and Robots. (accepted)
- Davis, J. D., Sevimli, Y., Eldridge, B. R. and Chirikjian, G. S. (2015). **"Module Design and Functionally Non-Isomorphic Configurations of the Hex-DMR II System"** ASME 2015 International Design Engineering Technical Conferences. (accepted)

EMPLOYMENT

- **Assistant Research Engineer, Laboratory for Computational Sensing and Robotics at JHU**, Baltimore, MD Sep 2015 - Present
- **Design Engineer, Fusiform**, Baltimore, MD Jan 2015 - Present
- **TA for Manufacturing Eng Course, JHU**, Baltimore, MD Jan - May 2015
- **Risk Management and Strategy Dept Intern, EnerjiSA**, Istanbul, Turkey Jul - Aug 2012
- **Quality Control Dept Intern, RMK Marine,** Istanbul, Turkey Jun - Jul 2012

PERSONAL

- Born in January 23rd, 1992 in Istanbul, Turkey. Currently lives in Baltimore, MD.