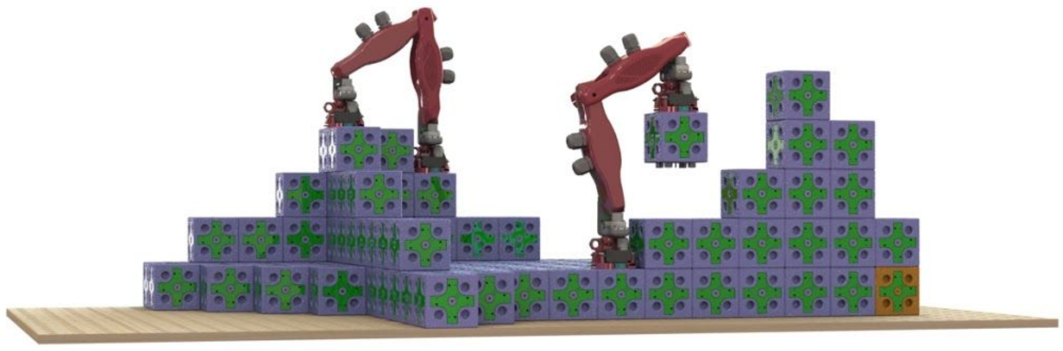
Swarm Construction: A Method in Multi-Agent Robotic Assembly

5/13/2020



Advisors:
Carlo Pinciroli, Gregory Lewin, Raghvendra Cowlagi, Xinming Huang

Students:
Cameron Collins, Josue Contreras, Neel Dhanaraj, Hannan Liang, Trevor Rizzo, Caleb Wagner

Table of Contents

Abstract

Robotic construction can drastically improve the efficiency and safety of construction. However, current robotic construction methods are limited by the types of structures robots can build and the ability for multiple robots to work collaboratively to build structures. This project creates an autonomous collective construction system in which two types of robots cooperate: construction robots and smart scaffolding robots. The latter robot type integrates electronics into building materials to create intelligent structures and allows for dynamic reassembling of existing components. In addition, we present a novel multi-robot collaborative building algorithm that showcases construction both with real and simulated robots.

Introduction

Contemporary issues related to urbanization, lack of adequate housing, and a need for building in adverse environments are increasing the demand for innovations in construction. Currently, 55% of the world's population resides in urban areas. This figure is predicted to increase to 68% by 2050, with most of this demand needed by developing countries.[1] Additionally, there are nearly 360,000 construction job openings left to be fulfilled by July 2019 according to the U.S. Bureau of Labor Statistics.[2] It is estimated that 3 billion people will require housing that has yet to be built by 2030, which implies that 96,000 housing units will need to be built everyday.[3] Advances in automated construction can address the demands needed to solve such problems.[4] The automation of construction can also address the safety of workers, especially in hazardous environments where construction may be difficult or impossible. Examples such as building temporary shelters for hurricanes, containment zones for nuclear accidents, as well as extraterrestrial habitats for Mars motivate the need for automated construction as a solution.[4]

Multi-robot systems are a solution that can achieve automated construction at scales much larger than the robots themselves. The challenge of using large multi-robot systems is how to control them to achieve high-level construction goals. A subset of multi-robot systems is swarm intelligence, which uses algorithms designed to achieve complex construction behaviors by implementing simple agent-level rules for each robot. Swarm intelligence provides a few key benefits that make automated construction efficient. Many robots can work in parallel, which can create a more efficient construction method. Such systems can also be designed to be flexible by designing the robot to be able to use teamwork to achieve harder tasks. A swarm is also decentralized, which creates a more robust automated construction solution that has very few or no single points of failure.

Collective robotic construction is a relatively new subset of research that still has fundamental unanswered questions before it can be implemented as a realistic solution. Even though small-scale demonstrations show the potential of using decentralized swarm construction robots, it is unclear how a system with simple local interaction rules produces the collective emergent behaviour and how these principles can be used to design a robust and adaptive multi-robot system that will be able to carry out automated construction.[4]

The success of swarm intelligence as an effective method for construction is most evident in nature. Termite colonies can become "superorganisms" where a million insects are able to construct structures several meters high. Such examples in nature occur in a decentralized setting and make use of stigmergic coordination, which is defined as individual decisions are made based on local interactions as well as feedback from the environment.[4]

Swarm construction robot platforms have demonstrated in a laboratory setting that a team of robots can create predefined structures made from blocks as seen in Figure 1.[5] Furthermore, this capability is enabled only by local agent sensing and decision making. Such structures are much larger than the individual robots demonstrating the effectiveness of construction by swarm robots. Inspiration from termites has also been used to further research how concepts such as pheromones and stigmergic coordination can enable local interactions and environmental feedback for the robots. This research can yield results on how to coordinate many robots.

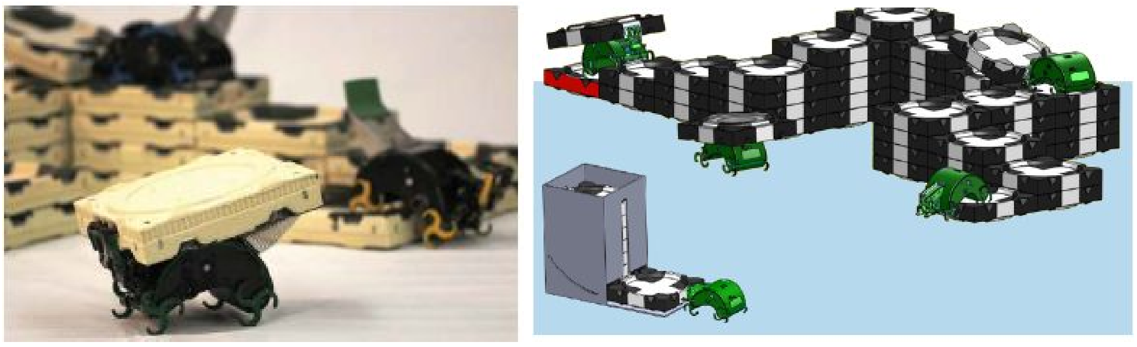


Figure 1*: TERMES System demonstrates multiple robots building a structure using hardware and in simulation*

Despite these successes with robotic construction, actual buildings and structures used in society are much more complex than the resultant structures built using current swarm robot construction platforms. Research still needs to be conducted to answer questions about how roboticists will:

(1) Produce robust algorithms that achieve desired emergent goals and are scalable to any number of robots
(2) Enable robots to use information provided by the structure to inform decisions regarding its construction
(3) Design robots for improved manipulation of and mobility over three-dimensional structures

The intent of this project is to explore such questions through the development of a robust and versatile swarm robot construction platform. The proposed swarm robot platform uses more capable serial linkage "inchworm" robots that can place blocks for three-dimensional construction as well as traverse over these structures. Furthermore, these blocks are designed to be "smart" so that they can interact with other blocks and determine the current status of the structure. This capability allows robots to localize on the structure as well as receive feedback from the structure itself. A decentralized construction algorithm makes use of these hardware capabilities to coordinate a swarm of robots in constructing a predefined structure implemented by a user.

Related Work

This section presents relevant research where multi-robot systems are used to build structures, including an overview of construction platforms, smart structures and construction algorithms.

Robot Construction Platforms

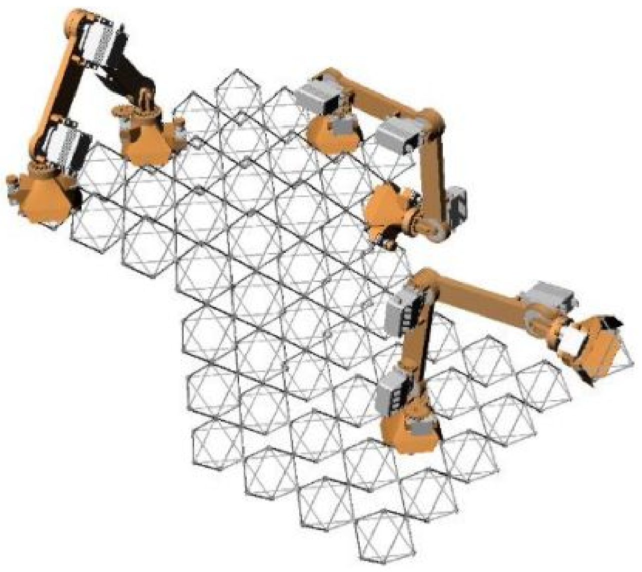
The TERMES multi-robot construction system was developed by researchers at Harvard University as a platform for research into collective construction systems. The TERMES system takes in a high-level representation of a structure, and creates rules for simple climbing robots to follow during construction.[7] This system is an example of a decentralized system where robots use local information and implicit coordination to successfully create a structure much larger than themselves. The design of the TERMES system was strongly influenced by a desire to produce simple robots, manufacture cheap blocks, and create structures under the influence of gravity. The goal of the TERMES system is to create predetermined structures, and is a remarkable example of a successful implementation of a decentralized system; robots perform actions individually and coordinate their actions implicitly through the manipulation of their shared environment. Despite this accomplishment, the TERMES system is limited in the types of structures that it can create due to the physical implementation of the robots and blocks. Blocks must be supported directly by the ground, or by a stack of blocks, not allowing for any unsupported overhanging structures. Due to limitations in how the robots can traverse the structure, intermediate configurations of the structure are required during the building process. Robots cannot travel down narrow corridors, or place blocks between two adjacent blocks. Inappropriate intermediate configurations can cause a deadlock in the creation of the final structure. As a result, admissible structures for the system are limited to single-path additive structures. This class of structure requires that a nonbranching path P can be drawn which satisfies the following criteria:

(1) The path P has an entry and exit along the perimeter of the structure
(2) The path P visits each block placement site exactly once
(3) The height change along path P does not vary by more than one unit per site
(4) No site along path P is flanked on opposite sides by two previously visited sites
(5) No site along path P is flanked on opposite sides by sites with desired stack height more than two blocks higher than its own.

The physical implementation of the construction system enables branching and merging path structures to be created in addition to single-path additive structures; however, future work into compiler design for this class of structure is required. As a result, the TERMES system is currently limited to single-path additive structures that satisfy the five admissible structure criteria specified above. The limitation in the classes of structures that the TERMES system can construct is largely due to the physical capabilities of the robots and blocks used to assemble the structure. One of the major considerations for the robot and block design presented in this work is to minimize limitations that hinder the variety of structures that the TERMES collective construction system can produce.

One example of a construction system with few limitations on the classes of structures than can be assembled is the Bipedal Isotropic Lattice Locomoting Explorer (BILL-E) system proposed by the NASA Ames Research Center.[5] The BILL-E system is a robotic platform designed to navigate a modular lattice structure. The system takes inspiration from multi-degree of freedom (DOF) industrial robots that are mounted to X-Y gantry systems to enable larger build areas. This system establishes an environment composed of cuboct lattice elements, and presents a multi-DOF robot arm with end effectors to attach to its environment. The design for the robot was driven by functional requirements for its ability to move in its three-dimensional isotropic lattice environment. The functional requirements for its operation are listed below:

(1) Robot shall be able to traverse linearly (X, Y or Z)
(2) Robot shall be able to turn and traverse in the direction orthogonal to the first direction (X to Y, Y to Z, X to Z)
(3) Robot shall be able to turn up/down concave and convex corners
(4) Robot shall be able to step up/down a level (+/- Z)



In addition to prioritizing effective navigation of three-dimensional structures, the system includes fault-tolerant end effectors to improve the probability of success in navigation. Through a working prototype, the BILL-E system demonstrated the ability to effectively satisfy its navigation requirements, and hold and place lattice elements. Due to the focus on three-dimensional locomotion in its functional requirements, the BILL-E robot and block design is an example of the physical implementation of a modular construction system with few restrictions on the types of structures it can produce. The robot and block design presented in this work will take inspiration from the BILL-E system to achieve versatility in the classes of structures that it can produce. The physical implementation of such a system and a strong focus on collective construction will enable the creation of structures that will improve upon the capabilities of the TERMES collective construction system.

Smart Structures

Another major aspect of robotic construction involves the coordination and communication of robots, as swarm teams must work in tandem to coordinate the building of a structure. While this communication is often established between robots, a different area of research involves robot-to-structure communication, in which the structure itself consists of one or more intelligent agents. This means that robots can directly interface with the structure itself in addition to other robots. This allows for services and calculations to be computed offline of the robots and instead be computed by the structure. Having an intelligent structure also establishes a communication medium where robotic agents can relay certain information between other agents.

The multi-robot construction platform known as the Swarm Robotics Construction System (SRoCS) uses Near Field Communication (NFC) and computer vision to establish an interface between the robots and building materials (smart blocks) through stigmergy. This system allows robots to determine and update the status of the smart building blocks within the structure. The SRoCS platform is capable of adapting its structures to different environments without the need of a predefined structure. The SRoCS platform is an example on how blocks that convey information about the structure allow for a more capable system.

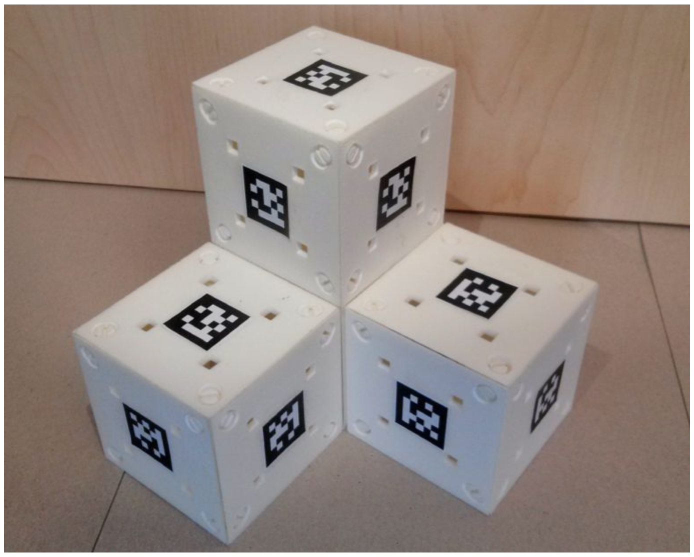


Figure 2*:* *SRoCS Stigmergic Blocks*[6]

Through NFC, the manipulator updates the status of the block when placed on the structure, creating a manipulator-to-block connection. NFC is successful in the SRoCS system because it allows for robot-to-block communication without the need of a physical connection. SRoCS developers mention the possibility of developing block-to-block communication as an alternative means of communication.

The SRoCS platform also uses LEDs on the smart blocks as a means of indirect robot to robot communication. The status of each block is updated through NFC by the robot as the structure develops. Since the SRoCS system is a stigmergic system, the LED color for the seed block, or the first placed block, informs the robots to initiate construction surrounding this block. As the structure evolves, each block is updated by the robot (through NFC) in relation to the structure; this construction is known as qualitative stigmergy. The advantage of having a structure be aware of its current configuration is that it can communicate what is missing to the worker robots.

To assist with the localization of the robotic manipulator with respect to the structure (building blocks), the SRoCS platform uses AprilTag barcodes. These passive components allow the robot to use computer vision to find blocks and determine their position in their environment. [6]

Swarm Construction Algorithms

A major aspect of robotic construction is the algorithm used to build a structure. The instructions on what, how, and where to place components of the structure can come from centralized or decentralized components and involve varying degrees of intelligence and knowledge of the overall structure by the robots.

The TERMES robots implements an algorithm where, after a user provides a high-level representation of the structure as an input, an offline compiler converts the structure to a “structpath.” A “structpath” is a path that the TERMES robots will follow to build the structure. Each robot acquires a block from the supply, then climbs onto the existing structure to traverse the “structpath” and place the block. Because of the nature of the “structpath,” when multiple robots are involved in construction, the robots traverse on the same path and thus need to be able to detect each other and perform collision avoidance as appropriate. This method provides a simple and effective way to build a structure. However, since a robot would need to traverse the existing structure to build the next block, the robots might not necessarily be travelling in the most optimal path and thus increases the cost of construction. Furthermore, while the TERMES robots can replace missing blocks that are along the "structpath," this algorithm does not provide a way to detect or replace a damaged block that was built previously and no longer on the "structpath."[7]

Another inspiration of the building algorithm comes from stigmergy in nature. Instead of having a building plan to guide the constructions like humans, ants use pheromones to leave signals on building materials as a set of instructions for their peers. The pheromones determine the building activity locally and thus guide the shape of the structure. Because the release of pheromones is based on environment conditions such as humidity and temperature, the shape of the structure that ants build is not predetermined. As a result, if this algorithm was adapted to a robotic construction system, modifications would need to be made to the algorithm so the system could produce a predetermined structure in a controlled fashion.[8]

In the previous work done on this project, instead of applying intelligence to the construction robots, intelligence was integrated into the structure by adding removable smart blocks to the building. The smart blocks would act as distributed hubs that detect nearby connected building blocks and give out instructions for local construction robots to follow. These construction robots were designed to perform simple tasks, and their only capabilities were traversing on top of blocks and picking and placing blocks. The introduction of the smart blocks allow the construction to be distributed, modular, and robust as the robots can be added to and removed from the structure without causing issues to the construction. However, this previous work was only done in two-dimensions and introduces new challenges as the smart blocks would need to be able to detect basic building blocks around them and keep track of their relative location to the overall structure.

Design Requirements

The overarching goal of this project is to create a system utilizing a centralized computer, swarm construction robots, and “smart” building blocks to autonomously assemble predetermined structures. To achieve this goal we defined design requirements for the systems as a whole and unique requirements for the system’s individual parts. This section will discuss these requirements and how they were determined.

System Requirements

To measure the initial success of this project, the primary objective of the system is to create a solid 4x4x4 cube shaped structure built with smart building blocks. Being able to build a cube will show that this system can discretize and assemble a structure in a three-dimensional work space. Additionally, this will demonstrate that the system can create a self-supporting structure.

Another requirement of the system is that any building blocks removed from the structure shall be detected and, if desired, replaced. This requirement was chosen to show a real-life application of the system. Components of architectural structures can be removed or damaged during assembly. Natural disasters can also damage structures well after they are completed. If the system can mend incompletions in the structure, it will demonstrate that it can autonomously build and maintain a structure without human interaction.

The final system requirement is that the system must allow for the assembly of a structure by multiple robots. The use of multiple robots will allow for the structures to be assembled quicker and more efficiently than with a single robot. Additionally, the use of multiple robots will allow for the development and analysis of various swarm construction algorithms.

The following sections discuss the sub-system requirements that allow for the fulfillment of the system level requirements.

Robot Requirements

In order to fulfill the system level objectives, the construction robot will be able to navigate the structures it assembles. The robot will be capable of traversing linearly (X, Y, or Z), turning in order to traverse in orthogonal directions, turning up/down concave and convex corners, and able to step up/down a level as shown in the Figure 3 below. The robot’s ability to navigate the structure will be dependent on the robot’s size, weight, motor torque, and support reactions. The robot’s ability to traverse the structure will allow for the construction of a variety of three-dimensional structures.

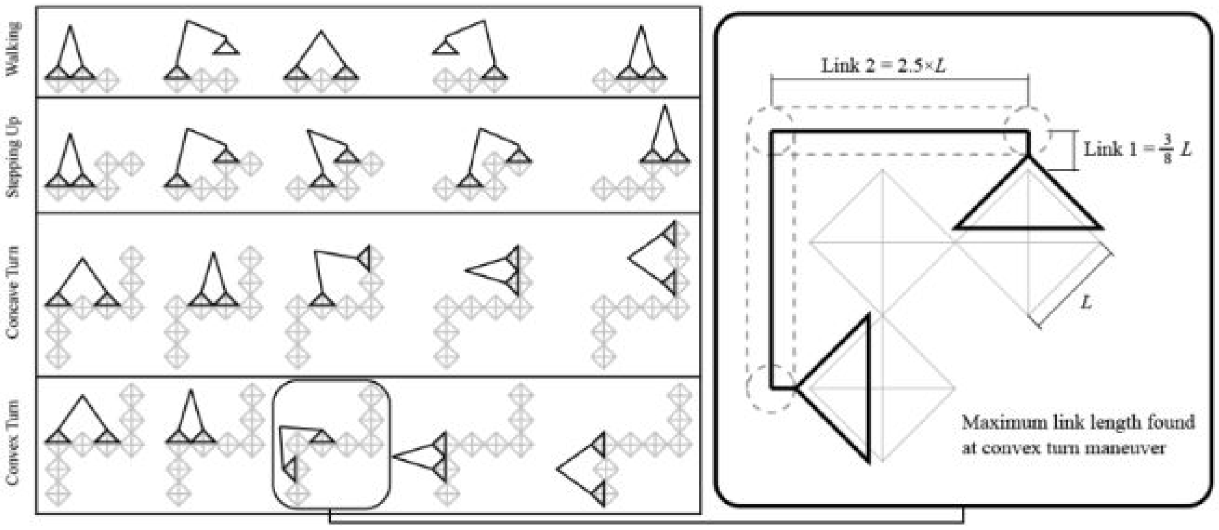


Figure 3*: Diagram from BILL-E report showing navigation conditions*[6]

In addition to the physical ability to navigate the structure, it is also desired for the robot, rather than a central server, to possess the ability to compute its navigation so that the robots can be kept as decentralized as possible. The robot shall algorithmically plan paths to navigate the structure. Path planning will allow the robot to navigate the structure efficiently. Also, path planning will prevent the robot from attempting to navigate to parts of the structure that are physically impossible for the robot to navigate to, such as a block face that is surrounded by other blocks in every direction. Additionally, the robot shall be able to locomote by controlling the position of its joints. Controlling the joints of the robot will require the calculation of its inverse kinematics as well as exploring its physical limitations.

In order for the robot to plan its path about the structure, it will need to be aware of the current configuration of the structure as well as its position and orientation within the reference frame of the structure. To obtain this information, the robot shall be able to communicate with the smart structure. If each block is able to store its relative position within the structure, it can then share that information with the robot. The structure will also share its current configuration with the robot so that the robots can determine what is left to build/replace within the structure.

To fulfill the requirement of navigating and assembling a three-dimensional structure, the robot shall be able to latch onto and transport blocks. Latching onto blocks will allow the robot to pick up blocks, place blocks, and latch onto the structures it assembles. Picking up and placing blocks are required for adding blocks to a structure. Transporting blocks will allow the robot to move blocks where they are needed to complete the structure. A unique end effector will be necessary to achieve these requirements.

Structure Requirements

The structure of the system shall be made of individual interlocking cubes, containing a microprocessor, referred to as “smart blocks.” In order to build three-dimensional structures, all six sides of the smart blocks shall allow for mechanical interlocking with other smart blocks and at least five sides for the robot’s end effector. The sixth side of the block will be the opposite gender as the other 5 sides, allowing for block-to-block connections. The mechanical interlocking shall be strong enough to support the 4x4x4 cube structure described in the system requirements. In addition to mechanical interlocking, the smart blocks shall be able to communicate. Through this communication, the blocks shall be able to share data involving the position of each block and the construction of the structure. This data and electronic connection shall also be shared with the robot. The smart blocks and the electronic components therein shall be powered independently of the robot, but may share a single power source for the structure as a whole.

Centralized Computer

The centralized computer shall be able to discretize a structure into a structure that can be assembled using smart building blocks. Additionally, the centralized computer shall be able to divide the discrete structure into multiple sectors. These sectors will identify the work area for a single robot or a collaborating group of robots. The centralized system shall transmit the discretized and sectored structure to all the robots. After this data has been received by the robots, the centralized computer shall halt communication with the system, allowing the remaining robots and smart blocks to become decentralized. This is so the robots have to interact with only a single interface on the structure to receive instructions to begin construction. Afterwards, the robots will make decisions in a decentralized fashion.

Design Overview

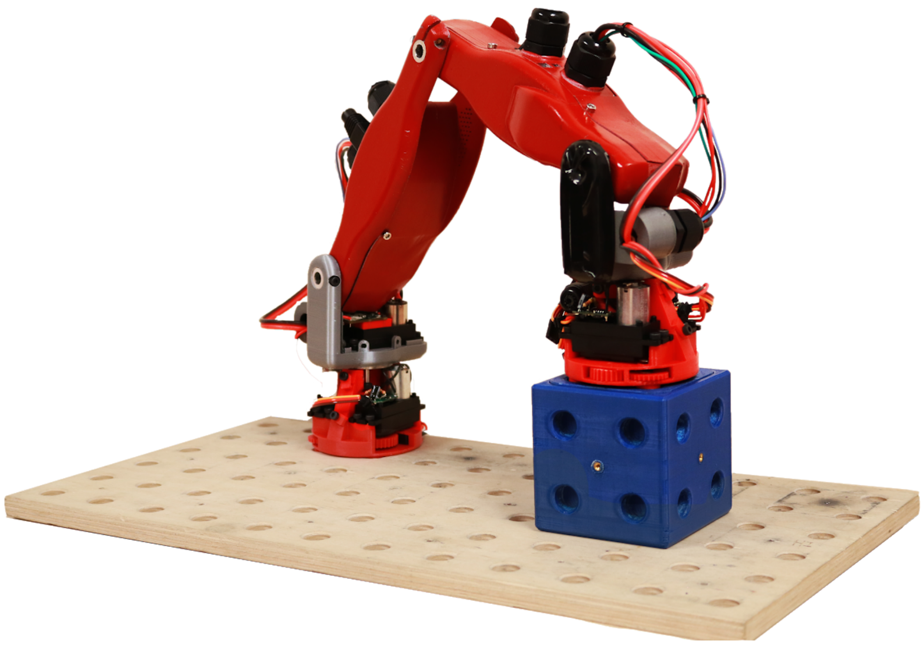


Figure 4*: Image of inchworm robot gripping onto block.*

The construction robot designed for this system is a 5 degrees of freedom (DoF) inchworm robot capable of building a variety of structures from block elements and traversing the structure in the X, Y and Z directions (Figure 4). As seen in Figure 5, the robot is able to traverse concave and convex corners as well rotate about its wrist joints in order to change directions. The robot uses specialized end effectors to attach to blocks and walk on the structure. This also allows the robot to manipulate a block and install it onto the structure.

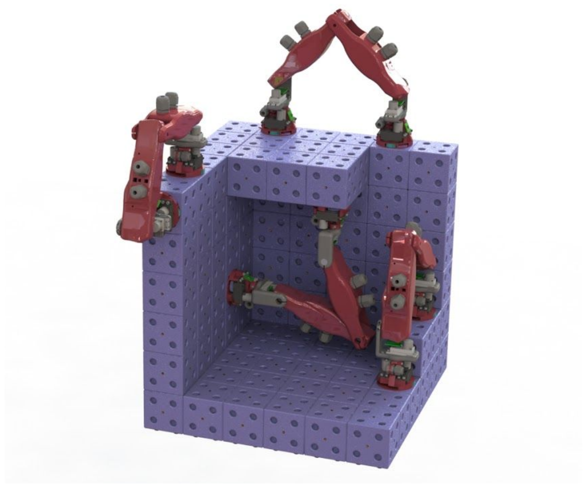


Figure 5*: Render of multiple robots changing planes in order to build a structure.*

The electromechanical design of the robot includes an enclosure for essential electronic components, and use of strain relief elements ensures reliable connections between links. The robot houses a high speed microcontroller that allows for low latency communication and real-time low-level control. The established control packets allow for comprehensive and expandable high-level communication to the low-level controller.

The smart blocks used to assemble the structure contain both electrical and mechanical elements (Figure 6). Six custom printed circuit boards (PCBs) contain near field communication (NFC) capabilities in order to convey information within the structure and to communicate with the construction robots. Wired power connections on the bottom face of the block enable the block to receive power from the rest of the structure. A gendered mating interface uses dowel pins and a single bolt to establish strong connections with surrounding elements. Dowel pins act to absorb reaction forces within the structure, as well as to align the block during placement. Construction robots use the same mating mechanism to traverse the structure and interact with their surroundings.

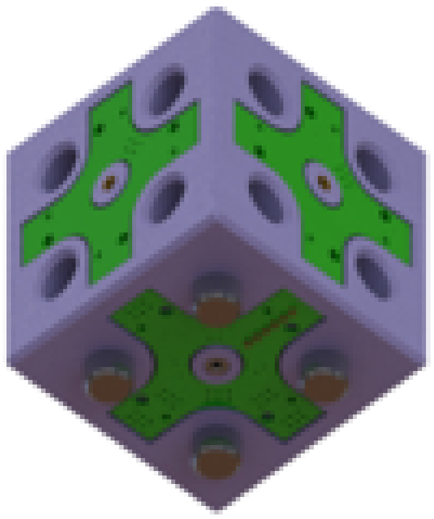


Figure 6*: A close-up view of a smart block, showing PCBs and mating interface.*

The main algorithm developed for the purposes of this project is the collaborative construction algorithm. This algorithm divides structures into different divisions or regions that robots will claim as their own. A partial ordering is established between each of these regions, which represents the order in which regions must be built. Due to the partial ordering, regions with the same order can be built in parallel. Robots work together by passing (referred to as ferrying) blocks to other robots to limit the amount of traversal needed to acquire new blocks, move the blocks to the desired location, and place the block in its next location.

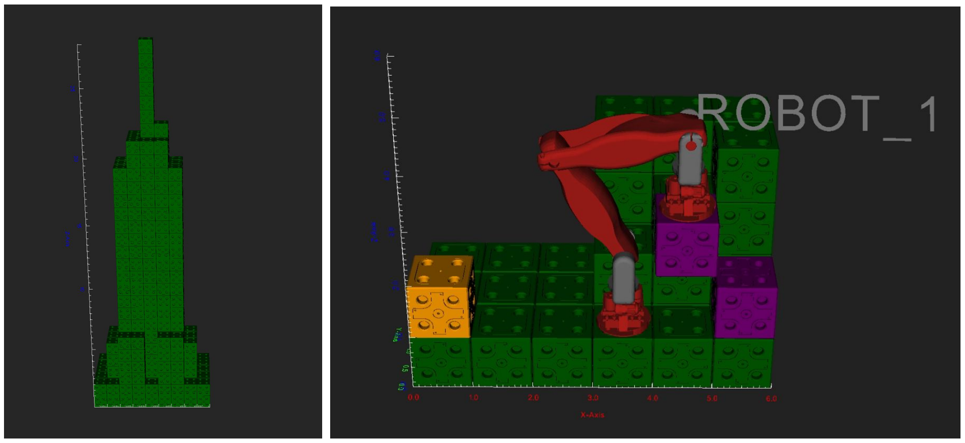


Figure 7*: Empire State Building built in custom simulator (left), robot placing blocks (right)*

Additional tools were developed for this project and are described in the following sections. These include a custom simulator, several tools used to create, edit and view the structures the robots will build, an online dashboard used to monitor the progress of the swarm as well as individual robots, and a desktop GUI used for controlling and debugging the robots.

Methodology

This section presents the research that has been accomplished to achieve the functional requirements as described in the previous section.

Comparison and Selection of Inchworm Design

The robot system was designed so that it has increased mobility and manipulability compared to previous robot solutions used to enable swarm construction such as Termes and Bill-E. An increase in robot mobility is defined as versatility in its ability to traverse complex 3D structures. Furthermore, the robot will have increased manipulability, which is defined as the ability to latch onto a block, transport a block to any location on the structure, and place it in the structure in any orientation.

Multiple mechanical system designs were reviewed that would be able to fulfill the robot requirements. These designs are briefly discussed.

A. A mobile, wall climbing robot such as the Disney VertiGo[9] robot uses two 360º tiltable propellers that produce thrust angled against gravity and the wall. Robots similar to this design could traverse vertical walls and could be integrated with a block placement mechanism.

B. Aerial vehicles such as drones have been used in previous research as a method for autonomous construction[10]. Flying drones have increased mobility and can place blocks in many orientations using a simple end effector.

C. A quadrupedal robot with grasping end effectors similar in concept to the Jet Propulsion Laboratory rock climbing robot LEMUR[11] would have increased stability allowing it to traverse over more complex surfaces including vertical, overhanging and inverted angles.[11] This also enables the robot to place blocks into the structure in many orientations.

D. A simple ground-based mobile robot would be able to traverse most surfaces that are parallel to the ground and further climb up horizontal planes like the TERMES platform[12]. A simple four bar mechanism can be implemented to place blocks in front of the robot.

E. A bipedal inchworm robot with grasping end effectors can have abilities similar to the quadrupedal robot allowing it to traverse any part of the structure as seen in the Bill-E platform[13]. An inchworm with only one gripper attached can be considered as a robot arm which allows for increased abilities to manipulate and place blocks.

Though the designs presented can traverse structures as well as transport and place blocks, other considerations such as ease of control, mechanical simplicity, cost, and power consumption were examined as well. A Pugh diagram that ranks these categories for each design is shown in Ttable 1.

	1 - 5 Weighting	Mobile robot with four bar (Base for comparison)	Wall Climbing Robot	Flying Drone	Quadrupedal Robot	Inch Worm
Mobility	5	0	+	+	+	+
Manipulability	2	0	-	+	-	+
Ease of Control	3	0	-	-	-	-
Mechanical Simplicity	5	0	-	-	-	+
Cost	4	0	-	-	-	-
Power Consumption	1	0	-	-	-	-
Total +		0	1	2	1	2
Total -		0	5	4	5	4
Total Points		0	-10	-6	-10	4

Table 1*: A Pugh diagram comparing the qualities of each design idea to a standard. Weightings are used to determine the best alternative.*

The team chose to further develop the inchworm design, as the analysis of the Pugh diagram showed that it features good mobility over a structure as well as manipulability of blocks. A five degree of freedom robot would be able to traverse over almost all block structures in any plane and orientation. The increased degree of freedom allows the robot to easily pick up blocks and place them into structures at any position and orientation in its workspace. Furthermore, the capability of an inchworm does not compromise other important factors previously presented making it an optimal robot system for swarm construction. The design of a 5-degree-of-freedom inchworm robot is presented to accomplish goals specified in this project.

Mechanical Design

This section presents the mechanical design of the robot.

Mechanical Requirements

The mechanical design of the robot is heavily influenced by the previously stated robot requirements. From these requirements, we can summarize the requirements most pertinent to the mechanical design of the construction robot and smart blocks:

1. Navigation Requirements for Construction Robot
    a. Robot shall be able to traverse linearly in X, Y or Z directions
    b. Robot shall be able to turn orthogonal to its current direction
    c. Robot shall be able to turn up and down concave and convex corners
    d. Robot shall be able to step up and down a staircase
2. Torque Requirements
    a. Joint torque must support robot during fully extended, worst case instance
        i. This should be supported by static and dynamic analysis
    b. Joint torque must support weight of robot and block during traversal
3. End Effector Requirements
    a. Shall establish robot-structure connection to support robot
        i. This connection shall be strong enough to handle static loading and dynamic motion of robot
    b. Shall install blocks into the structure
    c. Shall enable reliable engagement with structure
4. Geometrical Constraints
    a. Robot shall be able to traverse corridors with a width of one block unit
    b. Robot shall be able to place blocks in between existing blocks
5. Manufacturing Constraints
    a. Robot shall be easily manufactured to produce large swarms
    b. Robot should be easily constructed using consumer off-the-shelf (COTS) components as well as 3D printed materials
6. Block Elements
    a. Robot shall be able to manipulate block elements
    b. Robot shall be able to install block elements in the structure
7. Other Considerations
    a. The end effector design will be modular to support future development

Calculation of Linkage Lengths for Inchworm

Linkage length analysis is critical for ensuring the robot is capable of fulfilling all of the functional requirements listed above for motion on the 3D structure. The BILL-E NASA research paper[5] presents an analysis of all required motion positions, and identifies that the convex corner motion requires longer link lengths than all other robot motions.[5] As a result, fulfilling this motion condition will ensure that the robot has sufficient reach to fulfill all other motion requirements. Not only is it necessary to traverse convex corners, but in doing so the robot must leave vacant mounting locations within its reach. This will ensure that the robot does not become deadlocked at the convex corner condition.

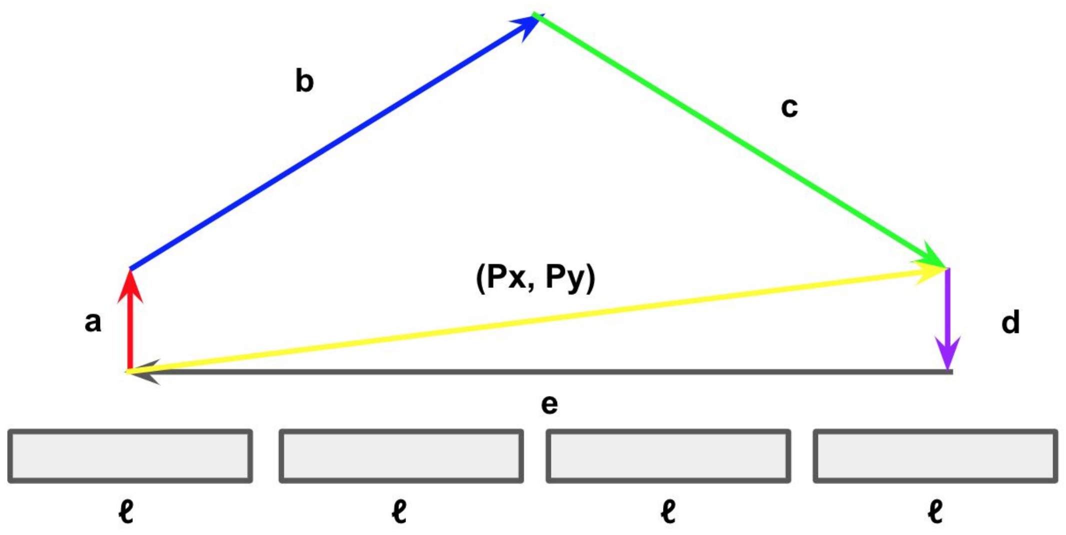


Figure 8*: Example robot configuration*

For a given unit block length ℓ, it would be ideal to make the lengths of links A, B, C, and D as short as possible to minimize the torque requirement for the robot. In order to simplify construction algorithms and control models for the robot, the linkage lengths will be symmetric with respect to the middle joint of the robot. This means that links A and D will be equivalent and links B and C will be as well. As a result of this assumption, only two independent variables must be determined as a function of the unit block length ℓ. Looking at Figure 8, it is clear that the absolute minimum length of the inchworm 2A+2B is 3ℓ, in which case links A and D will be zero, and links B and C will be 3/2ℓ. From a design standpoint it is impossible to have links A and D be identically equal to zero, and introduces some of the rationale behind link assumptions. It is assumed that the length of links A and D will be some factor of the overall unit block length. It is also assumed for the sake of this analysis, this factor C will be $1 < C < 2$. This assumption is made based on considerations for the block storage mechanism and the physical implementation of the rotary joints in links A and D, which will be discussed later.

In addition to traversing over a block on each side of the convex corner, the linkage lengths also depend on the mating process between the structure and robot. Many of the hardware attachment schemes currently seen in modular self-reconfigurable (MSR) robotic platforms require a perpendicular mating motion as part of their normal operation. Any comparable requirements for the end effectors will need to be considered when calculating linkage lengths. With this perpendicular mating motion in mind, an imaginary linkage “m” can be created in order to ensure that the linkage lengths are long enough to accommodate perpendicular insertion. The limit as m approaches zero represents the insertion of the robot into the block’s mating interface, and will be an important constraint for motion control.

Based on the previously mentioned constraints, in order to minimize the length of links B and C, and therefore the overall torque required as the joints, the selected values of B and C would be such that the joint is intersecting the corner of the convex motion condition. In reality, some space will be necessary around the joint in order to provide for motors, bearings, gears, sensors, or other components necessary for the rotation of the joint. To account for this, an imaginary linkage “p” is established for connecting the second robot joint with the convex corner. This linkage p describes a necessary clearance from the corner due to space limitations, and for the time being is set to a value of ℓ/2. As the details of the design are refined, it may be possible that the necessary value for P can be decreased, therefore decreasing overall robot length as well. A diagram is shown below in Figure 9:

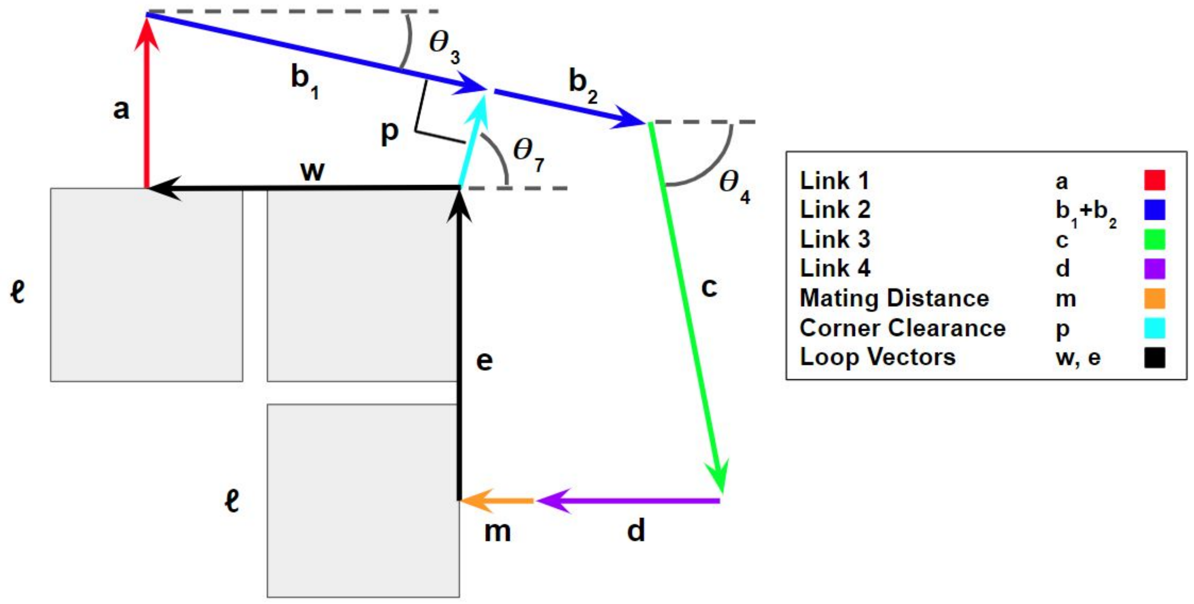


Figure 9*:* *Diagram used in linkage lengths analysis*

The team generated the vector loop equations shown below based on the top and bottom linkage loops. The team developed a MATLAB script to solve this series of equations by writing the outputs Θ_3, Θ_4, and B in terms of the inputs A, P, and M. The results are listed in table 2 below as functions of the unit block length, ℓ.

Loop 1:

x-axis: $b_1 \cos(\Theta_3) - p \cos(\Theta_7) - (3/2)\ell = 0$

y-axis: $a + b_1 \sin(\Theta_3) - p \sin(\Theta_7) = 0$

Loop 2:

x-axis: $p \cos(\Theta_7) + b_2 \cos(\Theta_3) + c \cos(\Theta_4) - d - m = 0$

y-axis: $p \sin(\Theta_7) + b_2 \sin(\Theta_3) + c \sin(\Theta_4) + (3/2)\ell = 0$

Inputs		Outputs	
Link A, Link D	1.375ℓ	Link B, Link C	2.144ℓ
Mating Distance M	0.25	Angle Θ_3	-28.29°
Corner Clearance P	0.5ℓ	Angle Θ_4	-60.08°

Table 2*: Results of linkage lengths analysis*

Selection of Unit Length

Based on the analysis presented above, the relationship between the unit length, ℓ, and the total length of the robot can be described by the following equation:

$$L_{tot} = 7.038\ell$$

While the relationship is linear, the selection of unit length has large implications for the total robot length, and therefore the joint torque requirement. To get a rough idea for the relationship between unit length and joint torque, we can create an approximate mathematical model. If we assume that the center of mass for each link is centered within the link, and that the mass of each link grows linearly with an increase in length, then the relationship between unit length and total length can be described by the following, where A represents a constant describing the relationship between link length and link weight:

$$\tau_1 = 7.236A\ell^2$$

This simple model suggests that the relationship between torque and unit length will not be linear, and that the selection of unit length will have a considerable influence on joint torque, and therefore motor selection. Based on budget considerations and a thorough review of available customer off-the-shelf (COTS) gear motor options, the team decided that standard form factor servos would be used.

Based on the size of standard form factor servos, the team created an initial prototype that was scaled according to a length of 3 inches. Preliminary testing of the robot validated that this unit length was large enough to enable the robot to fully enclose the electronic and electromechanical components necessary for operation. A static torque analysis of the prototype construction robot confirmed that standard form factor servos have the necessary torque for successful operation. Following successful tests with this prototype, a unit length of 3 inches was defined for the block and the final robot. More information about the first iteration robot design is documented in Appendix A.

Final Inchworm Robot Design

Design Method

In order to design a robot to meet the specified design requirements, the team utilized an iterative design process to design the mechanical system. This process enabled different mechanical and electronic subsystems to be proven and improved for each iteration. Furthermore, this allowed for early testing of the entire system. As shown in Figure 10, the team manufactured and evaluated two distinct design iterations before the final iteration. Improvements in the design for manufacturing, selection of final motors and the gripping end effector design were some of the lessons learned from this design process. The discussion of the first and second iteration of the robot are found in Appendix A.

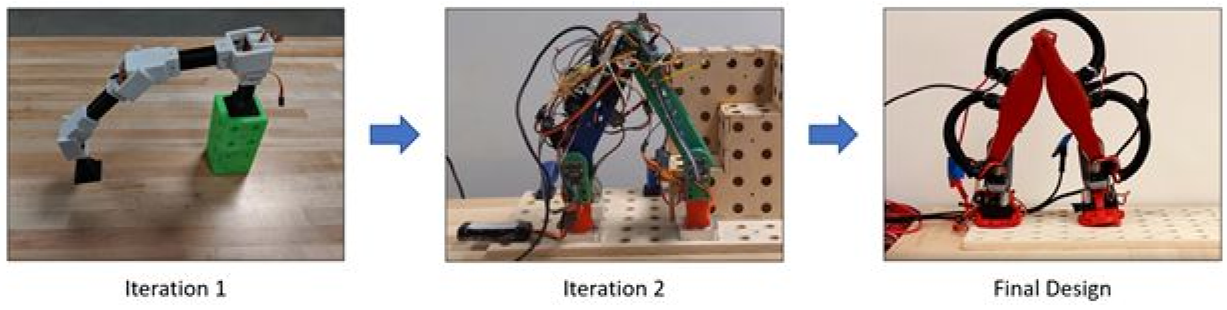


Figure 10: *Progression of Robot Design*

The final inchworm robot mechanical design is presented in Figure 11 and successfully meets all the stated design requirements. This is a 5-degree-of-freedom (DoF) inchworm robot capable of building a variety of structures from block elements and traversing the structure in the X, Y and Z directions. The end effectors enable it to manipulate the block and install it into a structure. Lastly, the profile of the robot is less than the 3 inch width of a block which allows it to travel in channels.

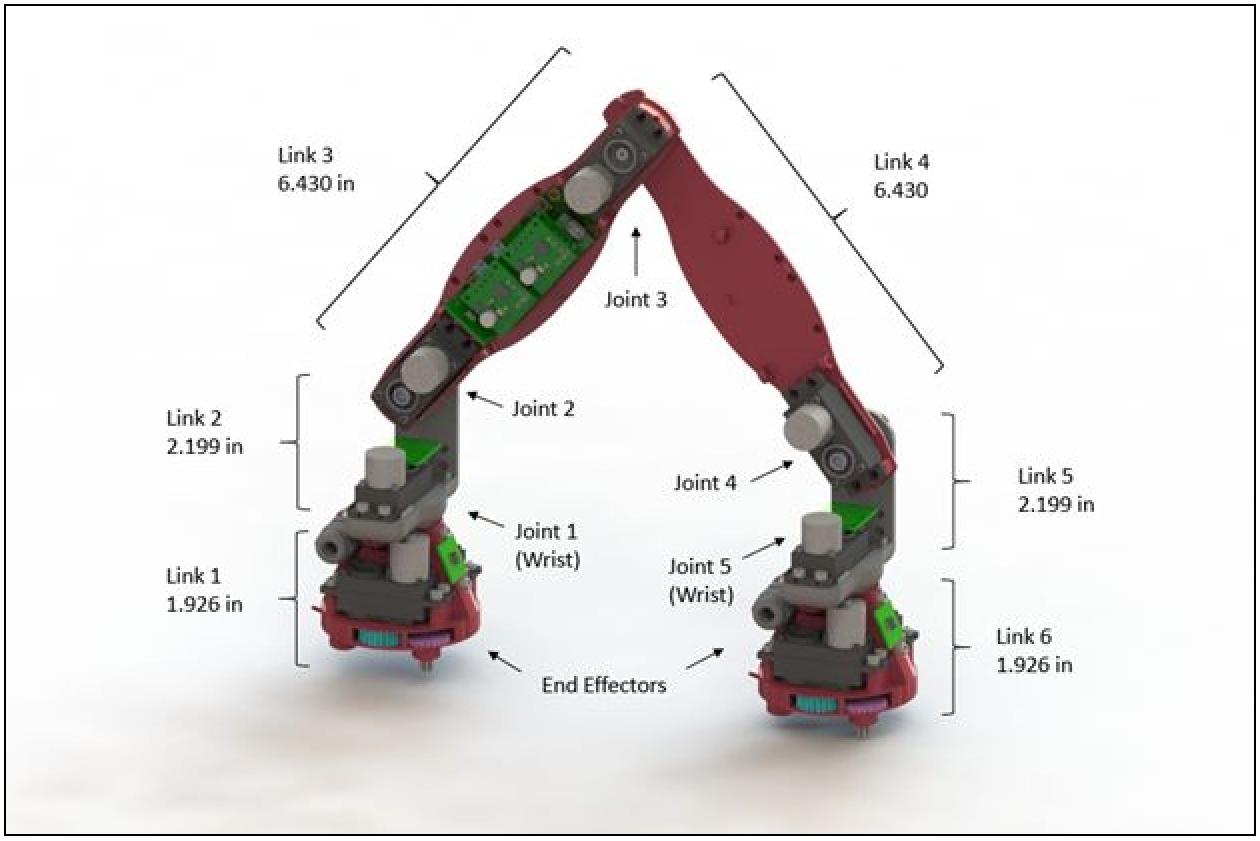


Figure 11: *Inchworm robot system*

Robot Links

The lengths of the links are derived from the link length analysis with a block unit length of 3 inches. These lengths are called out in Figure 11. The team ultimately chose a cantilevered link design to allow for simple assembly of links, and to minimize the number of parts in accordance with design for manufacture and assembly (DFMA) principles. The joints have bearing support for each shaft to minimize deflection of the robot. Link 3 and Link 4 house the system electronics and are protected by removable casings. Link 2 and Link 5 are the wrist links for the robot. These links have a modular interface that allows for an easy installation of any end effector design. As seen in Figure 12, two shoulder bolts extend from the wrist onto which the end effector can clamp.

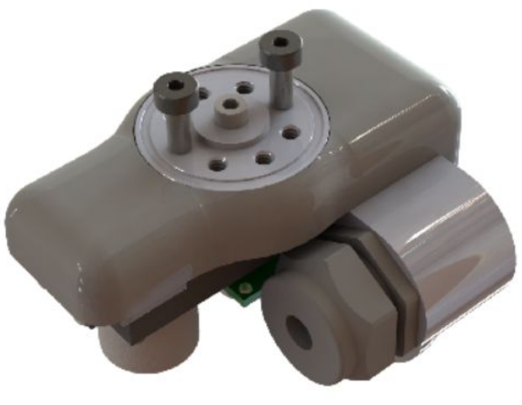


Figure 12: *Close-up view of wrist joint, showing shoulder bolt interface*

Robot Joints

In order to produce a simple manufacturable swarm robot, the team decided that a consumer off-the-shelf (COTS) motor selection would be desirable to actuate each joint of the inchworm. When researching available COTS gear motor options for small range of motion, high precision applications, it became clear that most products use a spur or planetary transmission powered by a brushed or brushless DC motor. Many of the available options were quite expensive, especially considering the team's desire to build multiple construction robots. One observation from our research was that the market for standard form factor servo motors is highly competitive, and there are many offerings with comparable torque ratings within the same size constraints. The use of standard form factor servos would enable an open source robot design, enabling others to build construction robots and easily expand the size of the swarm. For these reasons, the team decided that standard form factor servo motors would be used for joint motors. The JX PDI-HV5932mg 30 kg-cm servo was selected for the inchworm robot actuator. This servo provides a safety factor of 1.5 based on the worst dynamical joint torque that was determined in the dynamical model and analysis of the robot. These actuators were modified to incorporate high resolution absolute position magnetic encoders. The back casing of a motor was removed and a shaft with a magnet head was installed on the back of the output shaft. An encoder was mounted right over the magnet and testing showed the encoder was able to output the position of the shaft without any interference from the operating motor. This simple assembly is an easy modification to an existing servo motor and provides a compact motor and encoder system as seen in Figure 13.

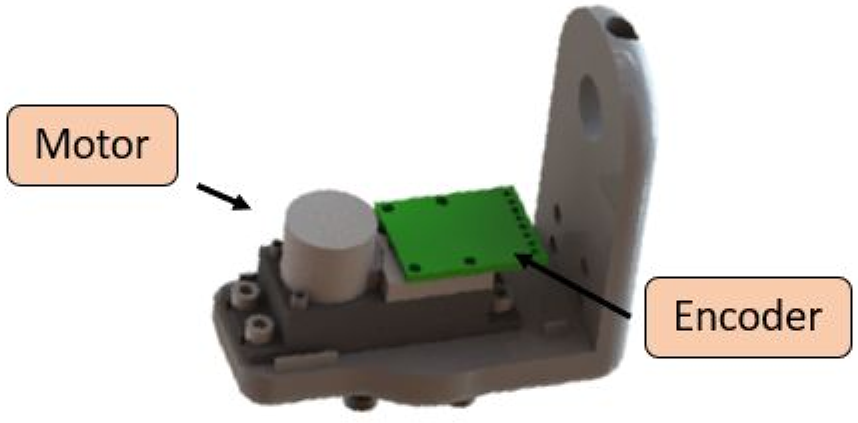


Figure 13: *Servo mount and encoder for robot joints*

Final End Effector Design

The end effector is the hardware interface used by the robot to latch onto the structure and building blocks. It is important that the end effector is robust so that it can endure many mating cycles, and simple so that it is feasible to manufacture many robots and building blocks. The end effector design consists of four dowel pins surrounding a central 4-40 bolt. The end effector also contains a subsystem for installing blocks into the structure. An allen key interface is used to transmit torque from the robot to the block's mounting interface.

End Effector Motor Selection

Because the robot and block use the same mating system, the power requirements for the end effector motors are quite similar. The motor used to power the subsystems was selected based on two functional requirements:

(1) Enable the end effector to mate with the structure in two seconds
(2) Torque the end effector screw to 5.5 in-lbs

Based on considerations for cost, weight, and simplicity, a brushed DC motor was selected for this application. In order to satisfy these requirements, the required motor power was calculated in order to attach the end effector to the structure in two seconds. For this calculation a free running torque of 2.0 in-lb was assumed.

$$Power\ =\ \tau \times \omega = \left[2.0\ in - lbs \times \frac{1}{12}\frac{ft}{in}\right] \times \left[2\pi \times 1\ \frac{revolution}{thread} \times 40\ \frac{threads}{in} \times 0.125\ \frac{in}{second}\right]$$

$$Power = 28.8 \frac{ft-lbs}{second} = 5.25\ W$$

Within the class of DC brushed motors that will be considered for this application only 370, 380, and 395 size brushed DC motors satisfy the power and weight requirements. Within this list, a calculation was performed to identify the minimum required gear reduction that will satisfy the stall torque requirement of 5.5 in-lbs. Using this gear ratio, the peak efficiency torque and RPM were calculated to determine if each of the motor-gearbox combinations would exceed 600 RPM at 2.0 in-lbs. The RF370 was selected as the ideal motor for this application, as it is the lightest motor that satisfies all selection criteria and functional requirements. Furthermore, RF370 motors are commonly found in servo form factors with built in spur gear transmissions. This means that a servo of "standard" form factor can be used to actuate the end effectors.

Motor	PE Torque (in-lb)	Speed (RPM)	Stall Torque (in-lbs)	Ratio @ Stall	PE Torque (in-lbs)	PE Speed (RPM)	Fast Enough?	Enough T@ PE?
FF050	0.01375	11320	0.058	135.18	1.86	83.74	Bad	Good
RC280	0.0425	8980	0.253	31.04	1.32	289.30	Bad	Good
RF370	0.0625	13730	0.312	25.19	1.57	544.99	Good	Good
RS380	0.12	13890	0.719	10.92	1.31	1271.72	Good	Good
RS395	0.1625	13110	1.041	7.55	1.23	1736.33	Good	Good

Structural Analysis

The end effector design is inspired by modular self-reconfigurable (MSR) system attachment mechanisms currently seen in the literature.[14] One example of this is the CoBold bonding mechanism shown in Figure 14. This design has experimentally demonstrated strong mounting connections, and has large tolerances for alignment. Specifically, testing has shown that the mechanism is capable of sustaining 5 kN of tensile load before failure, and is capable of successful mating despite lateral misalignments of up to 5mm and angular offsets of up to 20°.

Both of these attributes are important in this application. Furthermore, the CoBold system utilizes dowel pins for alignment, and to transmit forces between mating surfaces[12]. No input power is required to maintain connections, and the design has exhibited good durability. The CoBold mating design is currently being used by several research projects, namely SYMBRION and REPLICATOR[14].

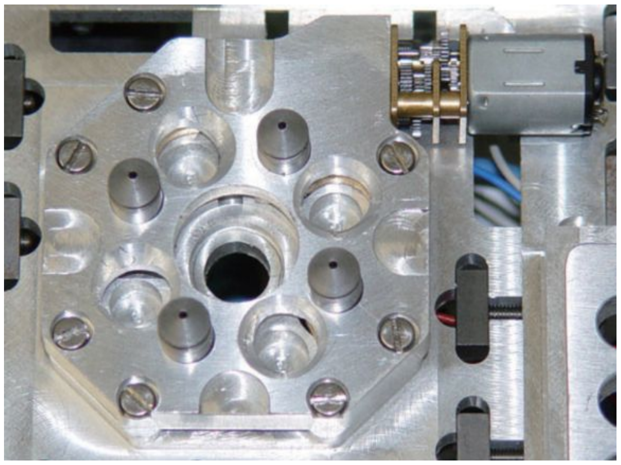


Figure 14*: CoBoLD bonding mechanism for MSR robot applications*[14]

Despite these advantages, the CoBoLD mating mechanism primarily utilizes CNC fabrication techniques, and has considerable material costs. Due to the nature of the project, ease of manufacturability and low cost are important traits for the end effector mating mechanism. While the CoBoLD bonding mechanism has great potential for creating strong connections, it has considerable mechanical complexity. For these reasons, a new mating mechanism will be presented which features strong mating connections and low cost. This design attempts to distribute shear and normal loads on different elements, much like the CoBoLD bonding mechanism, while attempting to create a simpler docking system.

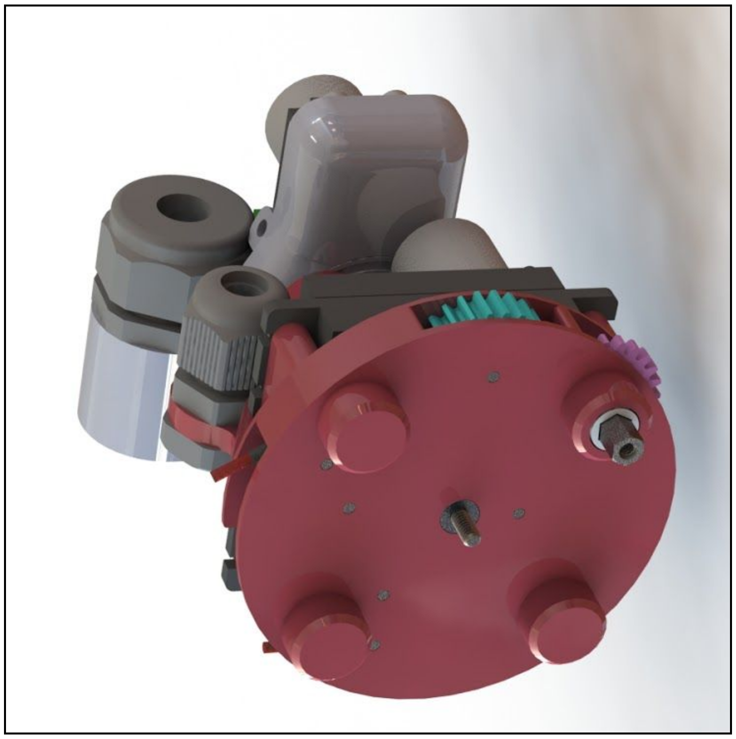


Figure 15*: End Effector CAD Render*

The final end effector design consists of a flat disk with four, 0.500” dowel pins integrated into the surface. These dowel pins are intended to absorb the shear loads acting on the robot-structure interface. In addition to supporting shear loads, the dowel pins also serve the purpose of aligning the end effector with the reference frame of the structure being built. A single 4-40 fastener absorbs the normal load acting on the interface. This ensures that the overall connection is capable of sustaining the support force reactions that are required to keep the robot in equilibrium on the structure. Based on the dynamic model of the robot, the maximum torque acting at the robot to wall interface was calculated to be 22.75 in-lb during fast acceleration movements. This means that during a worst case scenario, the 4-40 fastener is subjected to a 15.2 lb force, which is well within the proof load of an SAE Grade 8 4-40 fastener[15].

In order for the dowel pins to align the end effector, the bolt mechanism requires compliance when first mating with the structure. This compliance is achieved through the use of a light spring. Before the end effector engages with the structure, both the dowel pins and the bolt protrude from the flat bottom of the end effector. When the robot begins the mating process, the dowel pins are inserted into the holes on the structure, and the bolt is compressed into the end effector. The spring provides a small force to ensure that the bolt catches on the first female thread of the structure. The robot then rotates the 4-40 bolt until the proper torque is achieved. A cross section view of the end effector showing the spring loaded screw mechanism is shown below in Figure 16.

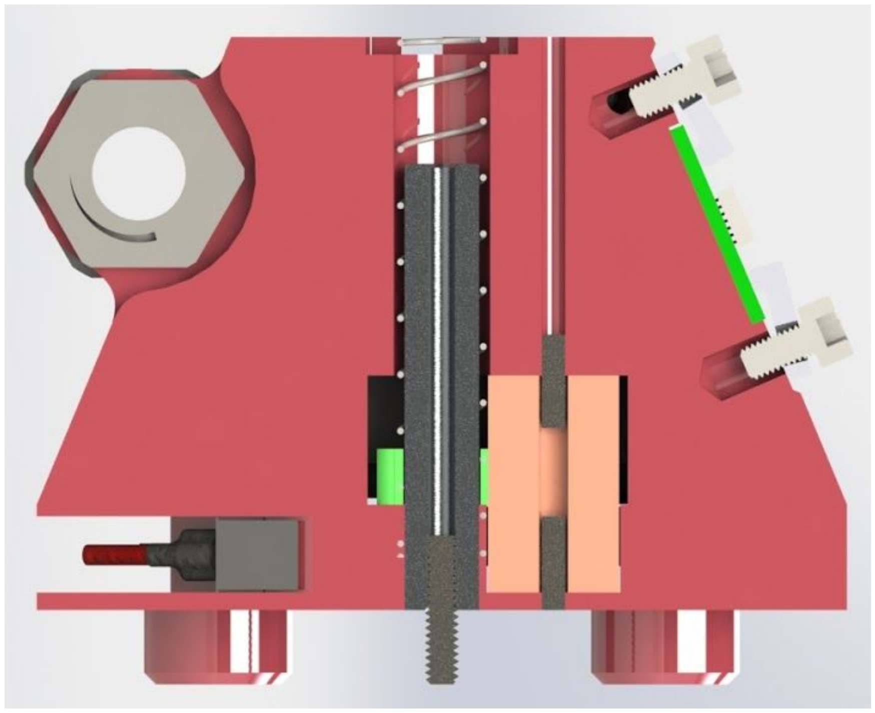
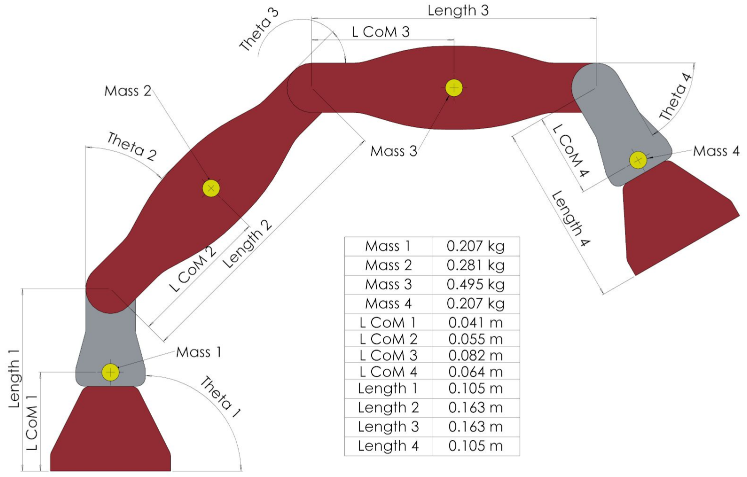


Figure 16*:* *Cross section of end effector*

Dynamic Model and Analysis

The dynamical behavior of the inchworm robot was modeled using the Lagrangian method and derived from the ball and stick model shown in Figure 17. The total mass of each link and the center of mass position was determined from the CAD model. The resulting equations were separated into the respective inertia-mass, centripetal-coriolis and gravity matrix. The gravity matrix was simplified into a system of equations that can be used for gravity compensation for the robot as shown in Appendix A.



Mass 1	0.207 kg
Mass 2	0.281 kg
Mass 3	0.495 kg
Mass 4	0.207 kg
L CoM 1	0.041 m
L CoM 2	0.055 m
L CoM 3	0.082 m
L CoM 4	0.064 m
Length 1	0.105 m
Length 2	0.163 m
Length 3	0.163 m
Length 4	0.105 m

$$\tau = M(q)\ddot{q} + V(q,\dot{q}) + G(q)$$

Figure 17*: Dynamic Model Ball and Stick Diagram*

The team calculated the dynamic torques for each joint for a worst case scenario by applying a positive velocity and acceleration on each joint while in the configuration shown in Figure 18. For this calculation, it was assumed the joints would travel 90 degrees in 3 seconds and the max acceleration would be determined from a trapezoidal velocity profile. Therefore the worst case dynamic torque was calculated to be 17.5 kg-cm. This information is useful in selecting the torque rating that the joint motors must have.

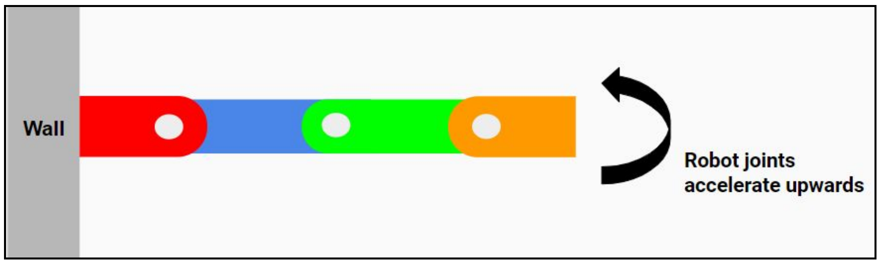


Figure 18*: Worst Case Configuration for Dynamical Model Analysis*

Final Block Design

The building blocks presented in this project must be capable of sustaining the reaction forces necessary to maintain the robot's equilibrium in all static positions. The difficult aspect of designing this modular construction system is establishing a robust interface between blocks that is capable of sustaining loads during the construction and later use of the structure. The primary mechanical functional requirement of the construction blocks is to sustain the normal and shear force acting on each of the six sides of the block. This functional requirement shifts the focus away from the block itself and more towards the interface that joins blocks together. Other functional requirements include interfaces to facilitate communication between the robot and blocks, mating geometries to ensure successful mating and demating cycles between the robot and blocks, and a block interface that enables symmetric, reversible connections on all six sides of the block.

The basic block design features a 90^{o} symmetric gendered connection, with female interface geometries on five of the six sides of the block. As a result, a block is only capable of actively attaching to another block through its one male connection. A structure created using these blocks will have large numbers of block interfaces with no mating geometry (female to female sides in contact), making it a highly anisotropic building material. The development of blocks capable of connecting on all sides would allow for the creation of stronger structures, and would enable the development of more advanced building algorithms. Such a block would need to allow the robot to selectively deactivate connections on some, or all sides. The mechanical complexities associated with this concept are beyond the scope of this work, and are left to future research in the field of swarm construction systems. While this design imposes weaknesses in the strength of the final structure and places a large importance on block orientation, the male to female interface presented in this design is anticipated to be strong. Experimental investigation of the strength of the mating mechanism is left to future works. Dowel pins are used to sustain shear loads acting on the male to female interface plane, while a single bolt is used to establish a preload between the male and female interfaces. A single spur gear train enables the construction robot to install blocks in the structure. Construction robots access an allen key interface within a dowel pin hole on the top surface of the block. 24 diametral pitch (DP) gears transmit torque to the central screw assembly, which is spring loaded in the same way as the end effector design. Cavities connect each side of the block to provide pathways for electrical connections to be made. A cross section view showing the gear train, spring loaded screw assembly, and passageways between PCBs is shown in Figure 19.

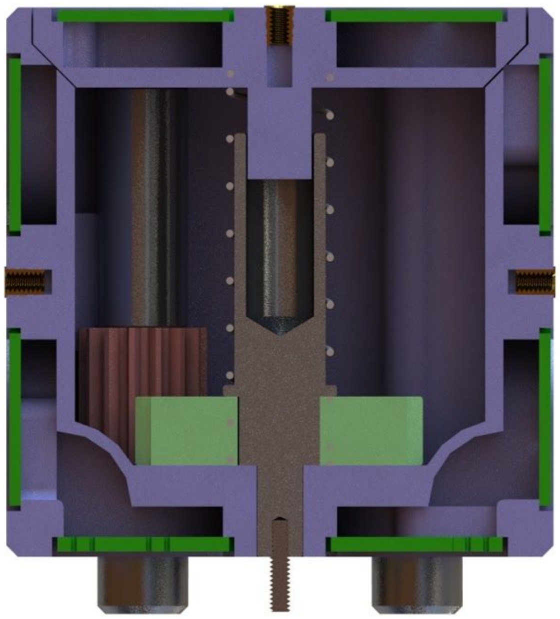


Figure 19*: Cross Section of Final Block Design*

Inchworm Environment Design

In order to test the robot's ability to traverse complex structures, the team created a small test platform. This environment contains all of the critical surface environments that the robot must traverse to be considered a success. A flat section enables testing of the robot's ability to traverse on level ground. A set of stairs, a concave corner, and a convex corner enable testing of the robot's ability to traverse in three dimensional space. A CAD rendering is shown in Figure 20.

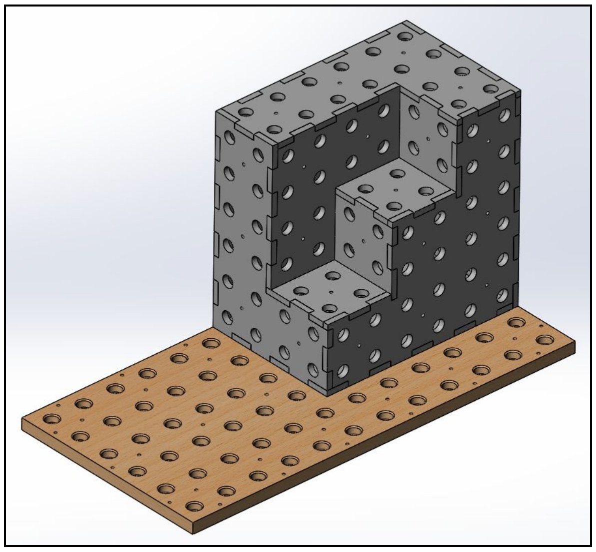


Figure 20*: CAD model of Robot Environment*

Electrical Design

The 3D Swarm Construction system consists of two types of robots that work collaboratively to build structures. The first type, the inchworm robot, must be able to traverse, latch, and communicate with the structure. The second type, the smart structure, must be able to communicate information such as the position and status of each block. Figure 21 demonstrates the high level system architecture between the smart structure and the inchworms onboard system.

For the inchworm robot, a high speed microcontroller and low latency communication interface are required to allow for real-time control and communication. A bi-directional communication protocol must be established between the robot's microcontroller and Linux system to receive its desired configuration and report the robot's information. In Figure 21 the bidirectional communication is represented by the thick arrow labeled “Robot Data” and “Desired Position and End Effector Control”.

Building with NFC-enabled smart blocks allows for the creation of smart structures. The smart blocks can recognize when a block has been added or removed from the structure. Using this technology, the structure can convey its status to the robots in real-time and can provide localization for each robot. In Figure 21, the blue blocks represent the smart structure components.

The following electrical section focuses on the electromechanical components and low-level design of two types of robots in this system.

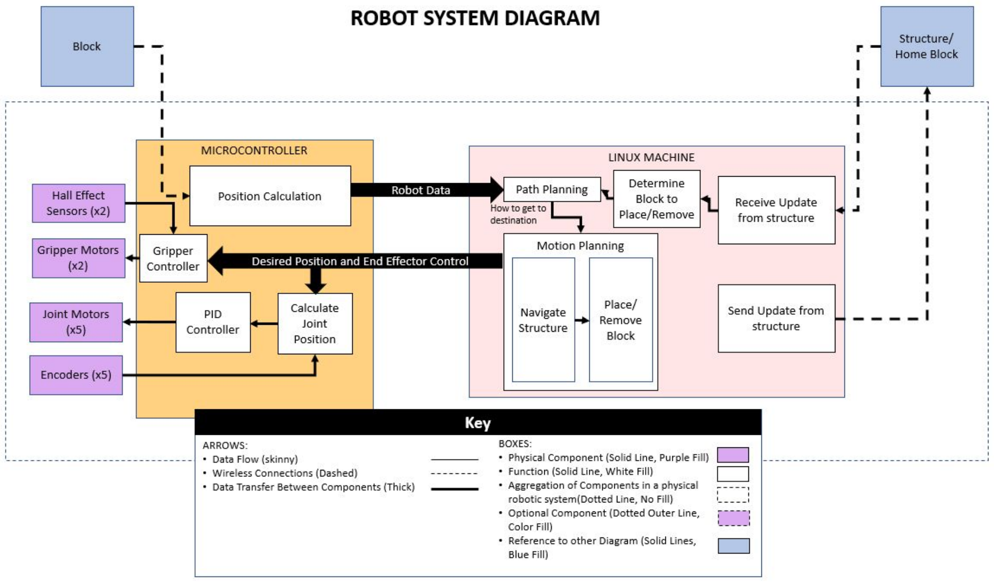

ROBOT SYSTEM DIAGRAM
Block
Structure/ Home Block
MICROCONTROLLER
Position Calculation
Robot Data
LINUX MACHINE
Path Planning
Determine Block to Place/Remove
Receive Update from structure
How to get to destination
Hall Effect Sensors (x2)
Gripper Motors (x2)
Gripper Controller
Desired Position and End Effector Control
Motion Planning
Navigate Structure
Place/ Remove Block
Joint Motors (x5)
PID Controller
Calculate Joint Position
Send Update from structure
Encoders (x5)
Key
ARROWS:
• Data Flow (skinny)
• Wireless Connections (Dashed)
• Data Transfer Between Components (Thick)
BOXES:
• Physical Component (Solid Line, Purple Fill)
• Function (Solid Line, White Fill)
• Aggregation of Components in a physical robotic system(Dotted Line, No Fill)
• Optional Component (Dotted Outer Line, Color Fill)
• Reference to other Diagram (Solid Lines, Blue Fill)

Figure 21: *Inchworm robots high-level system diagram*

Smart Structure

Making the 3D structure “smart” enables the system to verify the structure’s configuration dynamically and convey the position of the inchworm robot within 3D space. It also reduces the robot’s electrical complexity by offloading processing tasks such as localization within the 3D space and communication between inchworm robots on the status of the structure. It is important to notice that structures as part of this system are predefined. Furthermore, as a novelty of this robotic system, the structure will be the medium over which communication between robots and the smart blocks will occur.

System Architecture

The smart structure is built using two types of smart blocks: home blocks and smart blocks. The concept for smart block communication requires every smart block to be identical, except for one “home block” as seen in Figure 22. The home block is the first block placed, either by a human or by a robot. The home block stores the initial reference frame for the structure and is powered by an external power source. The home block contains the same communication capabilities as the smart blocks, and uses its external power sources to supply power to the entire structure. The first smart block is placed so that it is connected to the home block; this smart block is positioned at location (0,0,0). The home block will then communicate the reference frame to the smart block as well as communicate to the smart block that its position is (0,0,0). Every subsequently placed block communicates with the block(s) it is attached to and receives its position and reference frame. The position of each block added to the structure is propagated through the structure to the home block as seen in Figure 22. The home block stores the layout of the structure and communicates the layout to the robot(s) every time the structure is changed. Each block in the structure also periodically verifies that none of its surrounding blocks have been removed by sending a heartbeat throughout the structure. If a block is removed, this information is shared in the same way as an added block.

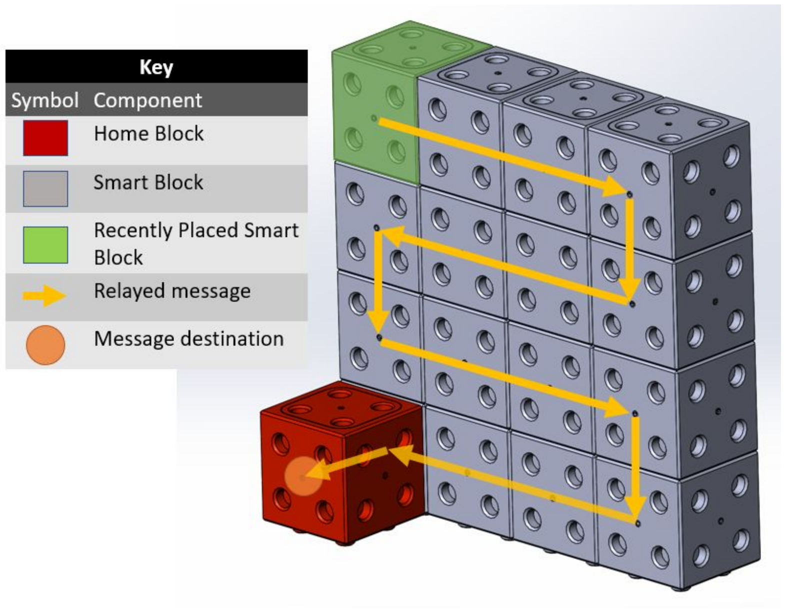


Figure 22: *Smart structure message transfer depiction*

The key processing tasks that the smart structure offers to this robotic system are as follows:

1. Structure Status - Adding blocks dynamically updates the high-level system of the structures current configuration.
2. Heartbeat Protocol - Identifies if a block is missing by updating the high-level system on the structure’s current configuration form the previous configuration.
3. Robot Position - The inchworm robot and high-level code can request the position of each robot in respect to the structure. This enables for the localization of each robotic inchworm with the 3D space.

Smart Blocks and Home Block Final Design

When considering the physical interaction of the smart blocks, it was concluded that a genderless electrical communication protocol would be desirable for the application. Using a genderless connection maintains the mechanical simplicity of the smart blocks. NFC was chosen for the communication protocol of the final iteration of smart blocks. The use of NFC permits communication between any of the five female block faces, enabling communication between parts of the structure where mechanical interfaces do not exist.

The foundation of the NFC smart blocks is the NXP PN532/C1 NFC controller[16]. This controller is connected to a Microchip ATmega328P microcontroller[17]. The PN532/C1 is capable of running in six different operating modes including peer-to-peer, card emulation, and card reader modes. The NFC-based smart block consists of six

NFC antennas, one for each block face, connected to one PN532/C1 controller via a radio frequency multiplexer. Two PN532/C1 controllers communicate over NFC while one chip is in card emulation mode and one is in card reader mode. To receive messages, the PN532/C1 is put into card emulation mode and the multiplexers are used to poll each block face for an incoming message. To send a message, the PN532/C1 is put into card reader mode and the message is relayed to the PN532/C1. The microcontroller keeps the PN532/C1 in card reader mode until the message transmission has been confirmed.

The multi-directional nature of the NFC smart block communication allows the opportunity for several message propagation protocols to be used to transmit messages back to the home block. One method of transmitting messages to the home block is transmitting along known paths to the home block. Another method for transmitting messages is using a flooding algorithm. With a flooding algorithm, a block sending a message would transmit the message to all of the blocks touching its six faces. Blocks receiving a message would also propagate the received message in all directions as well. If a block receives a message it has previously received, it ignores the message to prevent messages from endlessly propagating throughout the structure. The flooding method ensures that as long as there is a path, a message will eventually propagate to the home block. To simplify the smart block communication protocol, the known-path method is implemented.

Inchworm Robot

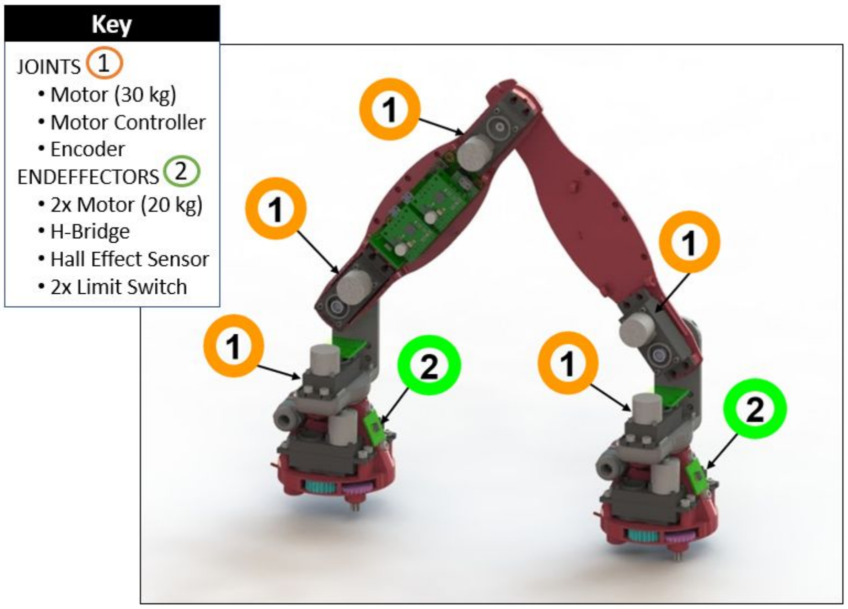


Figure 23*:* Final Inchworm robot sliced view of electromechanical components

Electromechanical Components

Microcontroller

The inchworm robot has five joints that require five individual motors, encoders, and PID controllers. Each joint consists of a closed loop system with feedback provided by one 14-bit I2C encoder and a motor controller to provide an input to the system. Each end effector contains two motors controlled by a PWM pin, a hall effect sensor, and an NFC tag. Due to the high amount of electromechanical components and the requirement of real-time control, the Teensy 3.6 microcontroller was chosen. Its 180 MHz processor with 22 PWM output pins and I2C bus capability allows it to control all the electromechanical components in real-time (within the given deadlines). The high speed USB port allows for fast exchange of packets through serial between the high-level code and low-level code systems. Furthermore, the Teensy's small form factor (62.3mm x 18.0 mm) allows it to fit within the inchworm's links along with wires and additional components.

Encoders

The accuracy tests performed with the first iteration of the robot showed that servo motor's built-in position controllers do not meet the level of precision and accuracy required for this system. The built-in position controllers were unable to reach the accuracy per degree needed to properly align the end effectors with the alignment holes on the smart blocks. To reliably build a structure, the robot must be able to repeatedly insert its end effectors into the smart blocks. To increase the accuracy of the robot's joint angles, AS5048B encoders[18] were added to each joint of the robot. The AS5048B encoders provide a 14-bit joint angle measurement with a ~0.02 degree resolution for each joint. The AS5048B encoders operate on an I2C bus. To make each encoder individually addressable, the AS5048B Arduino library is used to permanently assign each joint encoder a unique I2C address. The existing AMS AS5048 Arduino library provides functions to read the encoder angles and zero the encoders position when initialized. Additionally, the AMS AS5048 will notify the user if any I2C errors occur while attempting to read a joint angle from an encoder.

Motor Drivers and Battery Eliminator Circuits

A torque test performed with the first and second iteration robot showed that the stall torque on each motor would require a peak current of ~3 Amp at stall and a continuous current of ~2.6 Amps during peak nominal operation. To provide the power required by the motors, the TB9051FTG Single Brushed DC Motor Drivers[19] were incorporated into each of the robot's joints.The joint motors require a 9 V input which is within the 4.5-28 V output range of the TB9051FTG. These motor controllers allow for a 2.6 Amps of continuous current and a peak current of 5 Amps providing the proper amount of current to each joint. The TB9051FTG has two wiring configurations that operate at distinct frequencies; the truth tables documenting these modes can be seen in Appendix B. The first configuration uses two 20kHz operational PWM inputs for controlling the motor's direction and speed. The second configuration uses one PWM input to control the motor's speed along with two digital inputs to control the motor's direction. For the first version of the inchworm, the second configuration of the TB9051FTG was chosen as it fit the limited amount of PWM pins on the ATmega328P. This configuration presented a lot of issues since, with this H-bridge style motor controller, using a PWM signal to trigger the enable line limits the frequency of the PWM signal being applied. Because the inchworm robot requires a fast control loop at each joint, the first wiring configuration method was chosen. This allowed each motor controller to update the direction and speed for real-time control of the inchworm. The wiring configuration of the TB9051FTG is demonstrated in Figure 24.

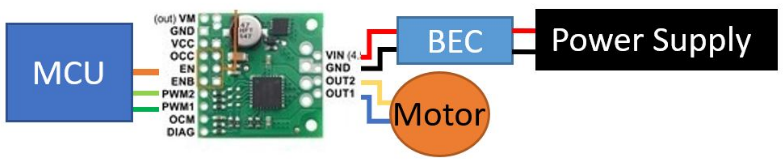


Figure 24*:* TB9051FTG motor controller and BEC wiring schematic

To provide the joint motors with a regulated power source, two high power battery eliminator circuits (BECs) were added to the electrical system. These BECs convert the 12 V input voltage from a DC power supply to a 9 V output that is within the operating range of the joint motors. The Castle Creation (CC) 10A peak 25V max input SBEC where chosen as they allow for an adjustable supply of 4.8V to 9V with a maximum continuous current of 7 Amps and a peak of 10 Amps as seen in Table 3. These BECs expand the inchworm's powering capability as they allow the robot to use high power rated lithium batteries to safely provide power to the electromechanical components when not operated through an external power supply.

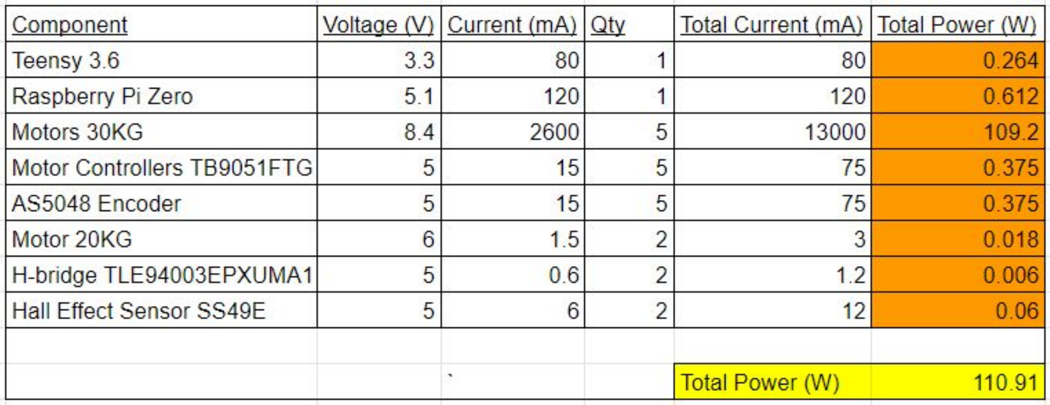
Power Component	Max Voltage Input (V)	Voltage Output (V)	Peak Current Output (A)	Continuous Current Output(A)
Wall Adapter	100-240	12	10	7
CC SBEC	25	4.8-9	10	7

Table 3*: Final inchworm robot system power calculations*

Power System Requirements

The final inchworm robot can be powered using a wall adapter or a battery pack. For the final iteration of the robot, a wall power supply connected to the CC BECs mentioned in the previous section provides more than enough power (120 W) to the electronic components of the robot. The final system uses a continuous source of power to eliminate power limitations when developing and tuning the inchworm robot as this requires long periods of time.

If a battery is added to the robot, additional electrical components must be added to support the monitoring of the battery of each robot and the performance based on the battery levels. Additionally, the required joint torque for the mechanical design has not been verified with the additional volume and weight associated with the use of a battery. The power calculations for the final robot are shown in Table 4. These calculations demonstrate that the robot requires a 12 V Li-Po battery with a capacity of ~5092 mAh. This was made with the assumption that the depth of discharge of the battery is 50% and the battery transfer efficiency is 99%.



Component	Voltage (V)	Current (mA)	Qty	Total Current (mA)	Total Power (W)
Teensy 3.6	3.3	80	1	80	0.264
Raspberry Pi Zero	5.1	120	1	120	0.612
Motors 30KG	8.4	2600	5	13000	109.2
Motor Controllers TB9051FTG	5	15	5	75	0.375
AS5048 Encoder	5	15	5	75	0.375
Motor 20KG	6	1.5	2	3	0.018
H-bridge TLE94003EPXUMA1	5	0.6	2	1.2	0.006
Hall Effect Sensor SS49E	5	6	2	12	0.06
		`		Total Power (W)	110.91

Table 4*: Final inchworm robot system power calculations*

Embedded Systems

PID Controller

The controller used to adjust the speed and position of the joint motors on the robot is a Proportional, Integral, Derivative (PID) controller as seen in Figure 25. This system was designed in tandem with the serial communication packet defined in the serial communication section below. Each joint motor has its own PID controller. The PID controller works by comparing the desired joint angle to the actual joint angle being read by the encoder.

The inchworm robot has two distinct configurations, dependent on which end effector is gripping the structure. When the robot is gripping an object in one orientation, one joint is supporting the weight of the robot while the other is supporting a small link. This configuration switches when the opposite end effector is gripped to an anchored object, in this case when an end effector is engaged to the structure. To account for these two scenarios, the joint motors are initiated with two sets of control parameters and have a function to switch the two sets instantly. The result of this control scheme in addition to the 14-bit encoders is more robust and more configurable than the built-in servo controllers.

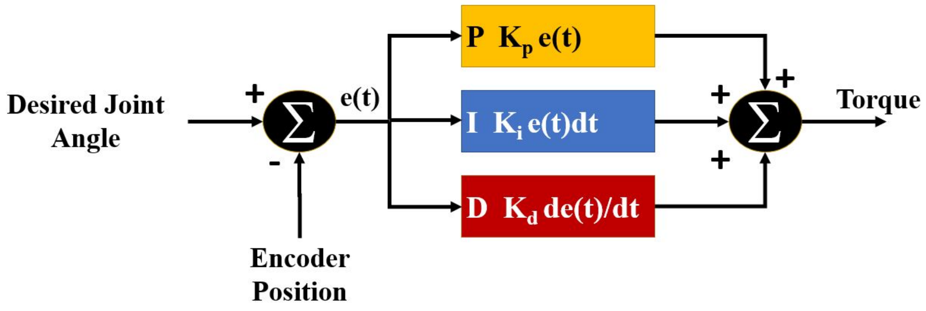
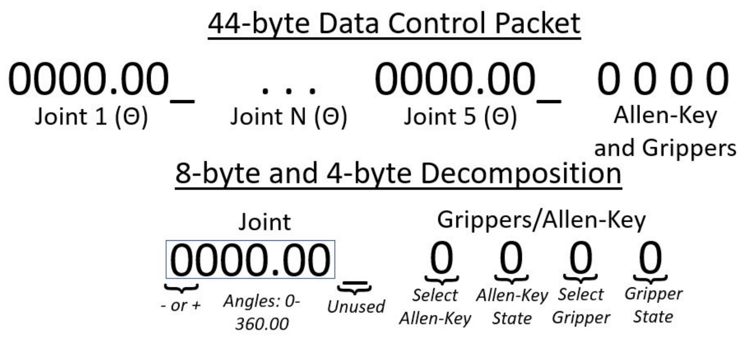


Figure 25*:* PID controller diagram

Serial Communication

A 44-byte control packet is used to communicate the five angles for each joint, the gripper states, the actuation of the gripper, and the actuation of the allen key as seen in Figure 26. The 44-byte packet is composed of five sets of 8-byte angle sub-packets for each joint control. The sets for the joint angles are structured as follows: the first byte is either a negative (-) or a zero (+) to establish the sign of the angle given. The next six bytes are the angles' floating point values with two decimal precision.The last byte, represented by underscore "_" is only placed for better readability of the position angles sent for each joint. This format was chosen to simplify the interpretation of the angles within the embedded code, as they can be converted from chars to floats using the standard C++ function atoi(). The last set of four bytes of the packet is for the selection and control of the gripper mechanism and allen-key mechanism. For the allen key, the first byte is to select the allen key mechanism and the second byte is to select the state. The state can either be engaging the block or disengaging the block. The third byte selects the gripper to be used and the fourth byte the state, either engage or disengage.



Gripper/Allen-Key Select Bit

Message (1 byte)	Action
0	Idle
1	Gripper/Allen-Key 1
2	Gripper/Allen-key 2
3	Both Grippers
Default (>=4)	No Action

Gripper/Allen-Key States Bit

Message (1 byte)	Action
0	Idle
1	Engage
2	Disengage
Default(>=4)	Stop

Figure 26*: 44-byte Inchworm robot control packet breakdown*

The packet sent through serial is decomposed and organized into the five joint motor objects, two gripper variables, and two allen-key variables. Figure 27 demonstrates how the packet received by the inchworm robot is parsed by a 44-byte serial buffer. This serial buffer then fills an 8-byte temporary buffer that is then converted into a signed integer. The last four bytes of the packet are used placed into two gripper variables and two allen-key variables. This enables the robot to read the serial packet properly and control the inchworm robot with a run time loop of 5 ms.

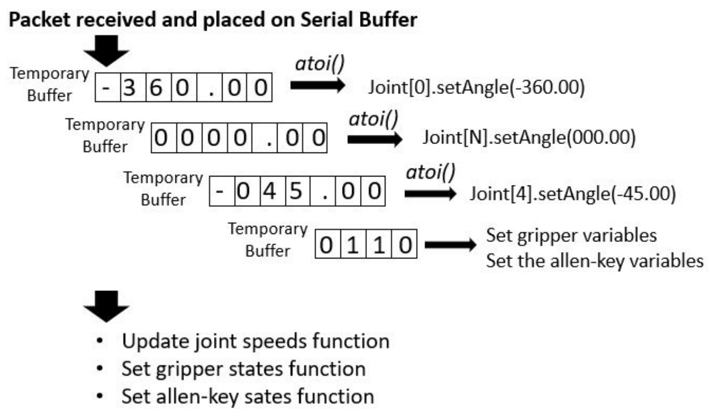

Packet received and placed on Serial Buffer
Temporary Buffer - 3 6 0 . 0 0
atoi()
Joint[0].setAngle(-360.00)
Temporary Buffer 0 0 0 0 . 0 0
atoi()
Joint[N].setAngle(000.00)
Temporary Buffer - 0 4 5 . 0 0
atoi()
Joint[4].setAngle(-45.00)
Temporary Buffer 0 1 1 0
Set gripper variables
Set the allen-key variables
• Update joint speeds function
• Set gripper states function
• Set allen-key sates function

Figure 27*: Serial buffer control packet organization*

Wiring

A reliable connection between links is essential to the robot because of the robot's high degree of mobility. In the preliminary iterations of the robot, the motion of the robot caused strain on the wires. This strain caused the I2C connection bus to momentarily lose connection from the microcontroller. This leads to two behaviors of the low-level system: the whole system hangs, or the reading of the encoders is corrupted. These erroneous behaviours forced the robot to reach unwanted configurations that could ultimately damage the inchworm. To relieve the strain of the wire connections between the links, cable glands were incorporated to the final inchworm design. Cable glands are used in industry to adequately enclose cables therefore providing no wire movement within the links when the robot reaches challenging configurations. This ensures that the robot can reach any position without forcing unpredicted movements of the joints.

Figure 28 demonstrates the full wiring diagram of the inchworm. The space required on each link determined the distribution of the electromechanical components. As seen in the figure, the C-link houses all the power components as these components' form factors fit the links' dimensions. The B-link allows for more components to be housed as it has the most space provided.

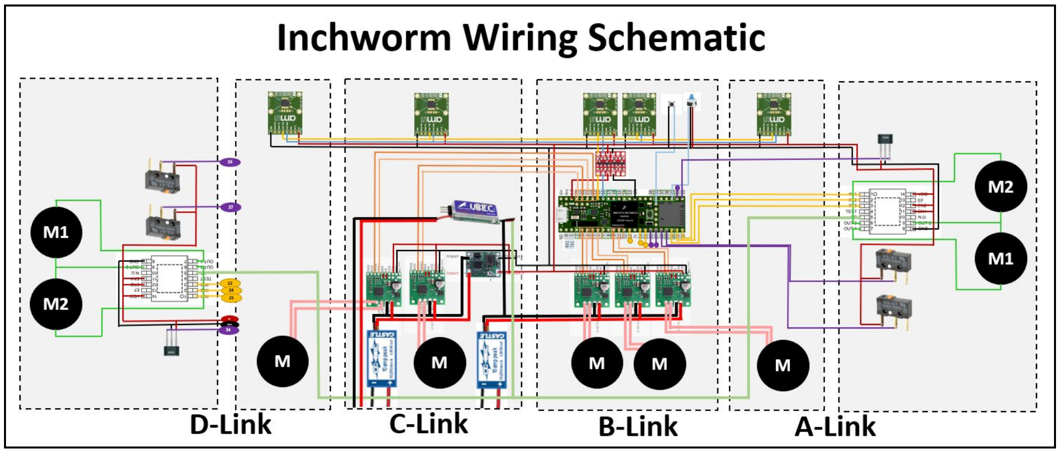


Figure 28*: Final inchworm robot complete wiring diagram*

Software Design

Control of the inchworm robot requires special consideration in terms of motion planning and path planning. The inchworm must be constrained in its movements to place and lift blocks correctly using the inchworm's placement mechanism, while at the same time being able to navigate the structure using the inchworm's gripper. Furthermore, control of the order in which blocks are placed when building the structure is a major component of the robot's building algorithm.

Note that this paper uses the term "motion planning" with regards to the physical control of the robot, such as setting the joint angles, while "path planning" is used to consider the position and orientation of the entire robot with respect to the rest of the structure.

Control Algorithms

To simplify the control of the robot, the inchworm design can be considered a standard three degree of freedom robot arm. While the robot is a dual end effector design, only one end effector will need to move at a time. As such, the end effector that is not moving can be considered a fixed base.

The final software iteration determined the appropriate angles needed to reach specified locations by transforming the target end effector position and orientation from a global reference frame to a local reference frame and used a geometric approach to calculate the inverse kinematics. A quintic trajectory planner was also integrated to provide smooth motion from point to point.

In order to have the robot achieve a point to point motion in its workspace and avoid collisions, Cartesian trajectory planning was implemented. The trajectory is defined by setting waypoints. The first waypoint is a point some offset above the face the end effector is currently connected to. This is done so that the end effector detaches from the block correctly, in a motion perpendicular to the face. The next set of waypoints is determined by using the A* algorithm to search through the open space. This is done so that in constrained environments, the robot end effector can path plan around obstacles. This acts as a simple and local collision avoidance system. The number of waypoints generated is determined by the complexity of the environment. For example, in Figure 29, no obstacles are blocking the end effector; as such, only a second waypoint is generated. The final waypoint is the block face itself that the end effector will be attaching to. For each of these waypoints, a quintic trajectory planner is used so that an incremental motion is achieved in reaching each of these waypoints. Accordingly, while there are three main waypoints in the trajectory shown below, there are many intermediate waypoints defined between each main waypoint.

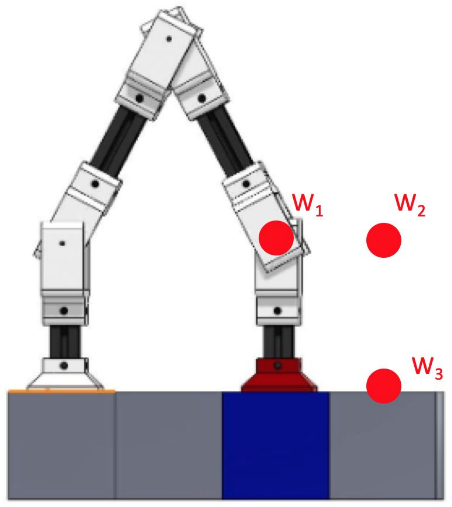


Figure 29: *Waypoints defined for robot trajectory*

Robot Navigation

To navigate the structure and reach any location/block within the structure, a pathfinding algorithm was created. When pathfinding, the position of the robot is considered a point with respect to the end effector closer towards the desired goal. This allows the actual path planning and inching movement of the robot (shown in Figure 30) to be effectively separated from one another. The path planning algorithm provides a list of nodes to traverse, and the motion controller of the robot determines how to actually traverse the returned locations. The robot will move the forward-facing end effector to the first location returned. If the distance between the forward and rear end effector is larger than a certain threshold, the rear end effector will move to the previous location of the forwards end effector. A simple analogy is if a person stands with one foot in front of the other and moves the forward foot even more forwards, the rear foot can then be moved to the original position of the forward foot in order to provide balance. In the event that the distance between the two feet or two end effectors is quite small, the person/robot will deem this step unnecessary and continue moving the forward end effector. Not only does this simplify path planning by allowing the robot to be considered as a single point, but it also allows the robot to move in an inchworm fashion as desired.

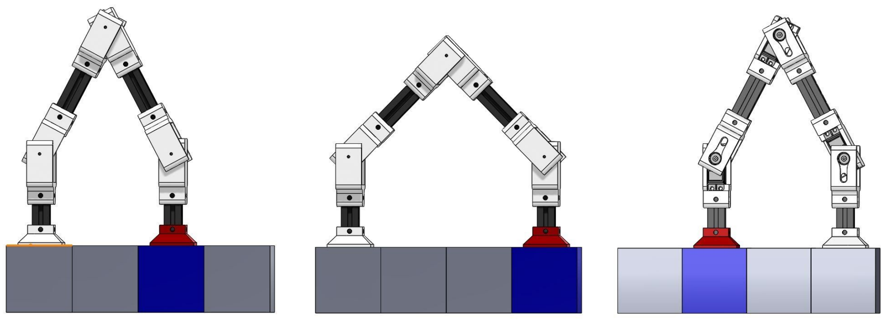


Figure 30: *Demonstration of inchworm motion of the robot*

A novel algorithm was implemented to account for the requirement that the robot needs to be able to navigate on the faces of the blocks of a structure to reach a target face. This algorithm was named Face* as it is an evolution of the A* algorithm. For each block in the structure, instead of exploring each block as a node, the Face* considers all available faces of a block as nodes and uses the Manhattan distance between faces to calculate cost. Since the robot will traverse the structure in an inching fashion, the Face* algorithm assumes that the back end effector will reach the last face that the front end effector was at and only plans a path for the front end effector. In order to return a valid path that the robot can physically maneuver and reduce the steps that the robot performs to reach a target, the Face* algorithm models each step as two types: (1) a step that results in different orientations between the end effectors; (2) a step that results in the same orientations between the end effectors. For the first type of step, because the robot needs to bend to reach the new orientation, the reach of the robot is limited and thus a smaller search distance is applied for this type of step. For the second type of step, since the robot is trying to reach the same orientation, the reach of the robot is further than the first type of step and thus a bigger search distance is applied. Since the distance and orientation that that front end effector can reach depend on the back end effector, the Face* algorithm filters the faces that it considers based on the orientation of the back end effector. For instance, if the back end effector is on the “top“ face, the “bottom” face will not be considered. The Face* algorithm then executes until either all available faces are explored or the target face is reached.

Visualization of Robot Motion

In order to visualize the motion of the robot, a custom simulator was developed. Existing simulators and visualization tools such as Gazebo and MATLAB Simscape Multibody were considered but were ultimately deemed ill-equipped or unnecessarily challenging to program for viewing the motion of the robot. Because it was decided that ROS was not going to be used as well due to the large overhead and complexity of Gazebo, Gazebo was excluded. Most other simulators including MATLAB were excluded as they were not designed to display long simulations and or they required a lot of custom code that would be much simpler to implement from scratch rather than integrating it with existing simulators through plugins. Furthermore, it was determined that simulating multiple robots would require specialized code as most of the considered simulators were not directly created for displaying multiple robots, while a solution created from scratch could be tailored to simulate multiple robots.

The simulator was built using the Visualization Tool Kit (VTK), where it reads in STL files for the various 3D objects and creates transforms between them to control their movement. A sample simulation of the robot can be found in Figure 31, where the front and back end effectors are highlighted in red and purple (other links are in blue), the structure is colored as green, the path the robot is taking is highlighted in blue, and the trajectory the robot is taking is comprised of the red dots surrounding the robot.

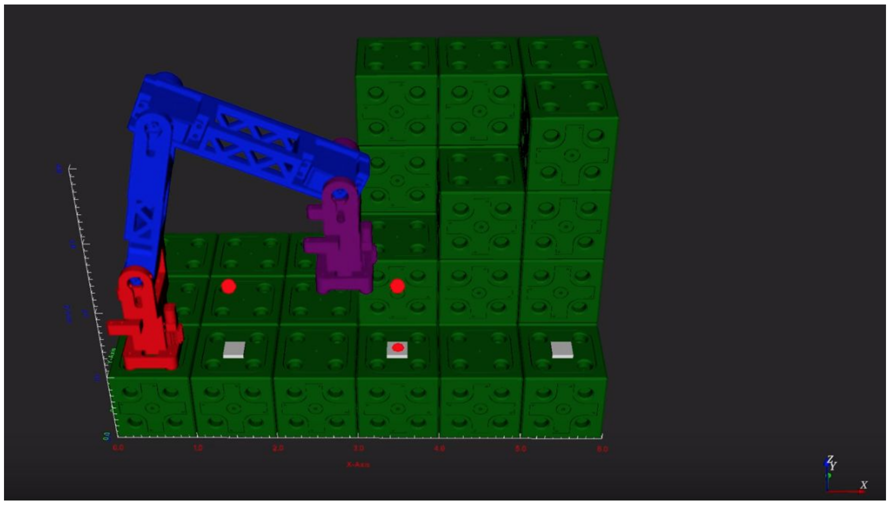
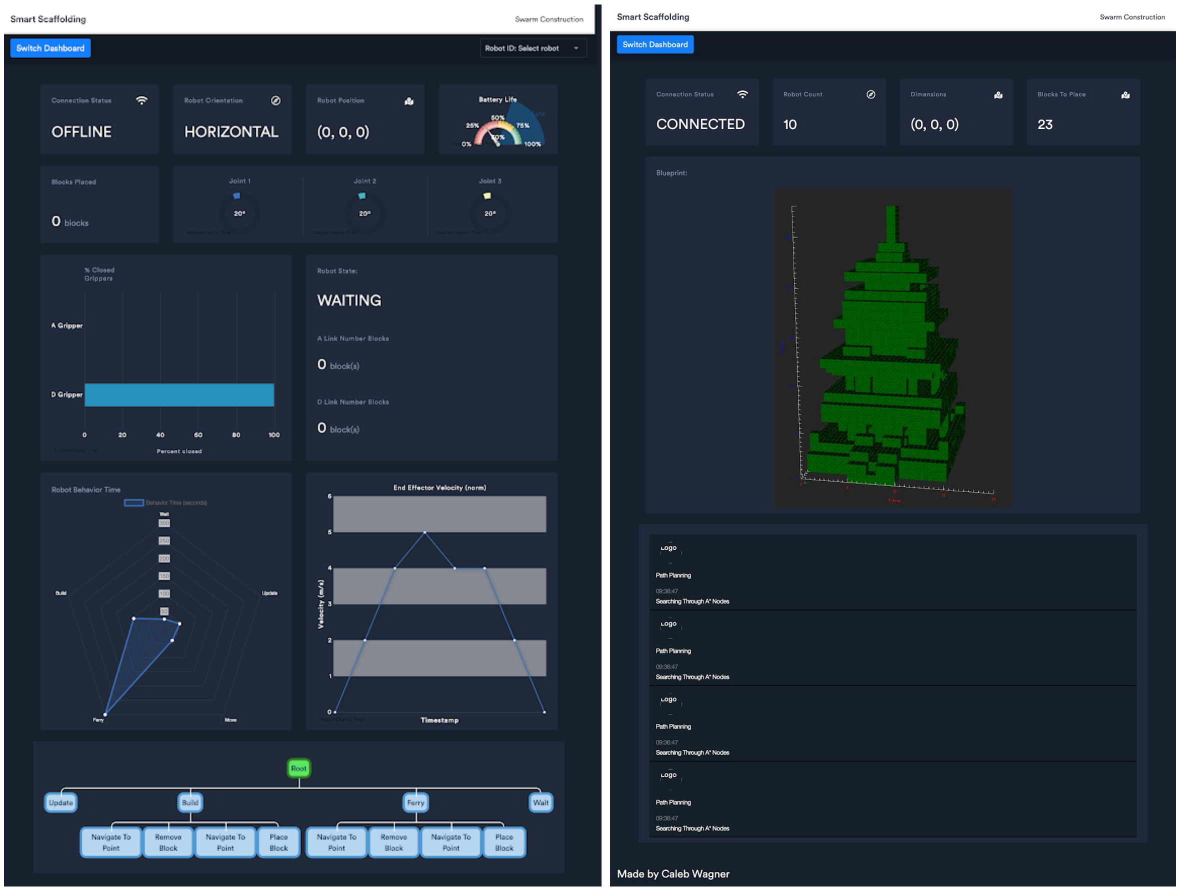


Figure 31: *Demonstration of the robot (iteration 2) in simulation*

The simulator is scaled correctly to map the angles used in the simulation to the actual robot. Currently, the simulator is capable of showing the motion of a single robot, the structure, the desired trajectory of a robot, the path the robot is taking to navigate the structure, and the robot placing a block. Future improvements will be made to the simulation including increased support for multiple robots as well as real-time visualization of the robot motion.

Because the simulation only shows the motion of the robots and not the individual parameters or metadata, a control dashboard was created as shown in the following picture to both visualize and control various features of all of the robots as well as the structure itself.



Connection Status: OFFLINE
Robot Orientation: HORIZONTAL
Robot Position: (0, 0, 0)
Blocks Placed: 0 blocks
Robot State: WAITING
A Link Number Blocks: 0 block(s)
D Link Number Blocks: 0 block(s)

Connection Status: CONNECTED
Robot Count: 10
Dimensions: (0, 0, 0)
Blocks To Place: 23

Path Planning
09:36:47
Searching Through A* Nodes

Path Planning
09:36:47
Searching Through A* Nodes

Path Planning
09:36:47
Searching Through A* Nodes

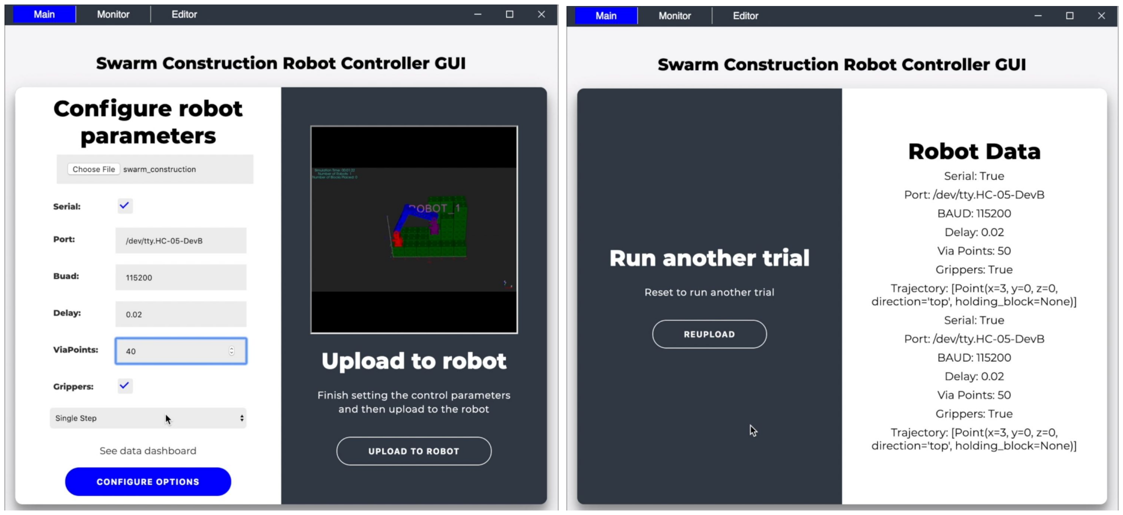
Path Planning
09:36:47
Searching Through A* Nodes

Figure 32: *Demonstration of the user interface of control dashboard*

The dashboard, created using React.js and shown in Figure 32, allows users to select between multiple robots and the structure itself to view information specific to the selected robot/structure, including the current angles of the robot, the number of blocks that have been placed, the position of the robot relative to the home block, and the battery life of the robot. This tool will help in debugging multiple robots quickly, as well as will allow for an aesthetic controller for manipulating the different robots. The dashboard communicates with the robots and structure using Pusher, a cloud-based publisher-subscriber. Pusher was chosen due to its widespread use for the Internet of Things (IoT) and its simplicity. Because it was developed for IoT, Pusher is lightweight and does not cause any issues of large overhead with either the robot or structure. The publisher-subscriber architecture was chosen so that neither the robots nor the structure have to wait for the dashboard to connect or update. They can publish their information to channels (queues) in the cloud, and the dashboard will subscribe to the channels and read from them as soon as it is able to.

Control and Debugging of Robots

In order to test individual robots and ensure their capability to run the main code, a graphical user interface (GUI) was developed. This GUI allows for both high and low-level control of the robots. It was coded using ElectronJS and allows for a desktop application to be installed for all operating systems. This means the GUI can be downloaded without any additional setup. For the high-level code, users can upload predefined paths to the robots as well as adjust certain parameters such as the delay between each robot motion and toggling the use of the grippers, as shown in Figure 33.

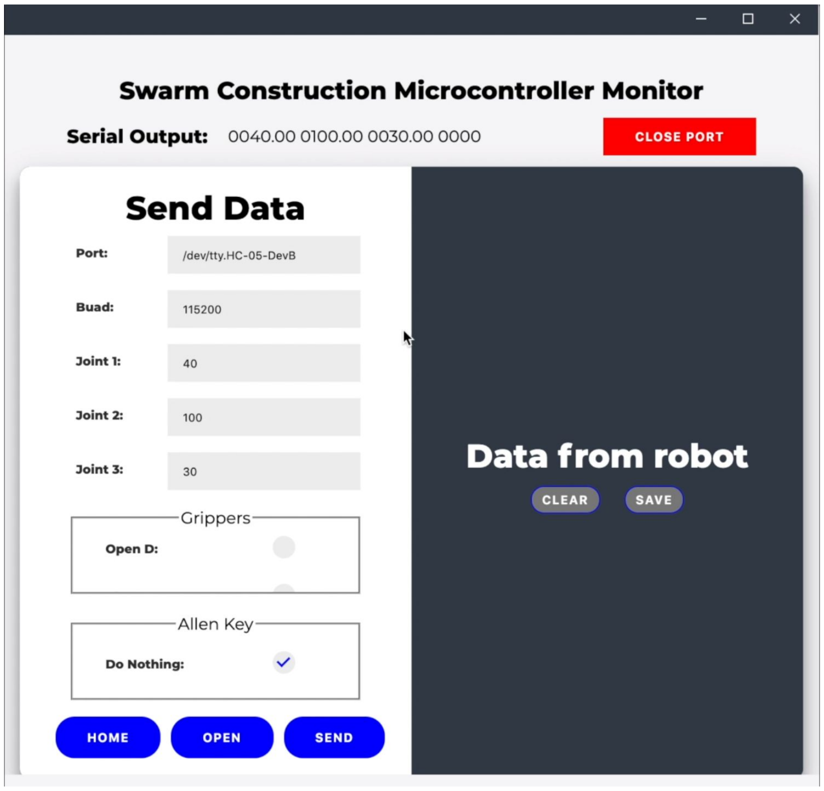

Main
Monitor
Editor
Swarm Construction Robot Controller GUI
Configure robot parameters
Choose File
See data dashboard
CONFIGURE OPTIONS
ROBOT_1
Upload to robot
Finish setting the control parameters and then upload to the robot
UPLOAD TO ROBOT
Main
Monitor
Editor
Swarm Construction Robot Controller GUI
Run another trial
Reset to run another trial
REUPLOAD

swarm_construction
Serial: ✓
Port: /dev/tty.HC-05-DevB
Buad: 115200
Delay: 0.02
ViaPoints: 40
Grippers: ✓
Single Step

Robot Data
Serial: True
Port: /dev/tty.HC-05-DevB
BAUD: 115200
Delay: 0.02
Via Points: 50
Grippers: True
Trajectory: [Point(x=3, y=0, z=0, direction='top', holding_block=None)]
Serial: True
Port: /dev/tty.HC-05-DevB
BAUD: 115200
Delay: 0.02
Via Points: 50
Grippers: True
Trajectory: [Point(x=3, y=0, z=0, direction='top', holding_block=None)]

Figure 33: *Robot controller GUI high-level interface*

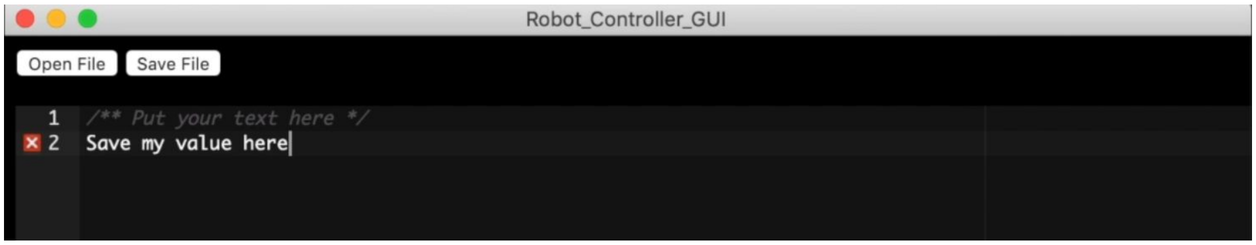
For the low-level code, users can easily create the custom packets that are used to communicate with the embedded code, as shown in Figure 34. A home button resets the robot to its home position. The low-level code interface also includes a serial monitor that allows users to see the output of the embedded code on the microcontroller. This is meant to entirely replace the Arduino IDE and serial monitor.



```
Serial Output: 0040.00 0100.00 0030.00 0000
Port: /dev/tty.HC-05-DevB
Buad: 115200
Joint 1: 40
Joint 2: 100
Joint 3: 30
```

Figure 34: *Robot controller GUI low-level interface*

A third feature shown in Figure 35 is provided that allows users to take notes and open/save files. This is useful when recording angles and pin numbers used on the microcontrollers.



```
/** Put your text here */
Save my value here
```

Figure 35: *Robot controller GUI note-taking feature*

Building Structure Template Creation

For the robot to build something useful with the structure, the robot must first have some idea of what the final structure should look like. As such, the robot must store the “blueprint” of the structure or a schematic which it can use to determine what blocks it still needs to place.

To create the blueprints that are required to build structures, two main tools were developed. The first was modeled after the game Minecraft; users are placed into a voxelized environment where they are able to add or remove blocks, as shown in Figure 36. Using the arrow keys, users are able to move around the environment and create the blueprints they desire. For more resources about actually using the software, consult the existing documentation. In addition to creating blueprints from scratch, a second tool was developed that allows users to upload an obj file, a file produced by CAD software. By making use of some existing voxelization tools, the uploaded obj file can be voxelized; users can adjust the density and size of the voxels to make the blueprint as they wish. Afterward, the voxelized structure is converted to the format used by the Swarm Construction system.

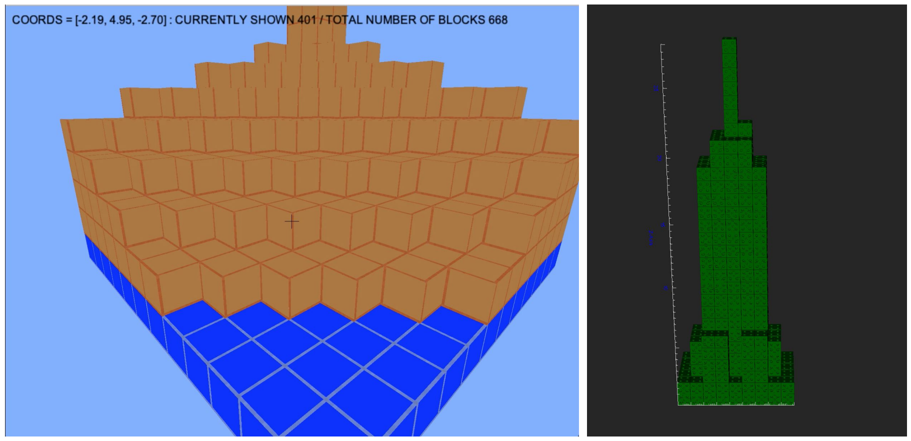

COORDS = [-2.19, 4.95, -2.70] : CURRENTLY SHOWN 401 / TOTAL NUMBER OF BLOCKS 668

Figure 36*: (From left to right) The blueprint creation tool, a voxelized Empire State building which was created using the aforementioned software.*

System Architecture

The architecture of this system (shown in Figure 37) consists of three main software entities: the structure itself, the simulator, and the robots, which for this section will be considered as a single entity. Each entity involves multiple threads to achieve task parallelization and work efficiency. For example, robots listen for updates on a separate thread from their main controllers in order to always be able to receive updates from the structure. As a distributed system, each entity must be capable of managing its state, carrying out external behaviors, and communicating with the other entities in the system.

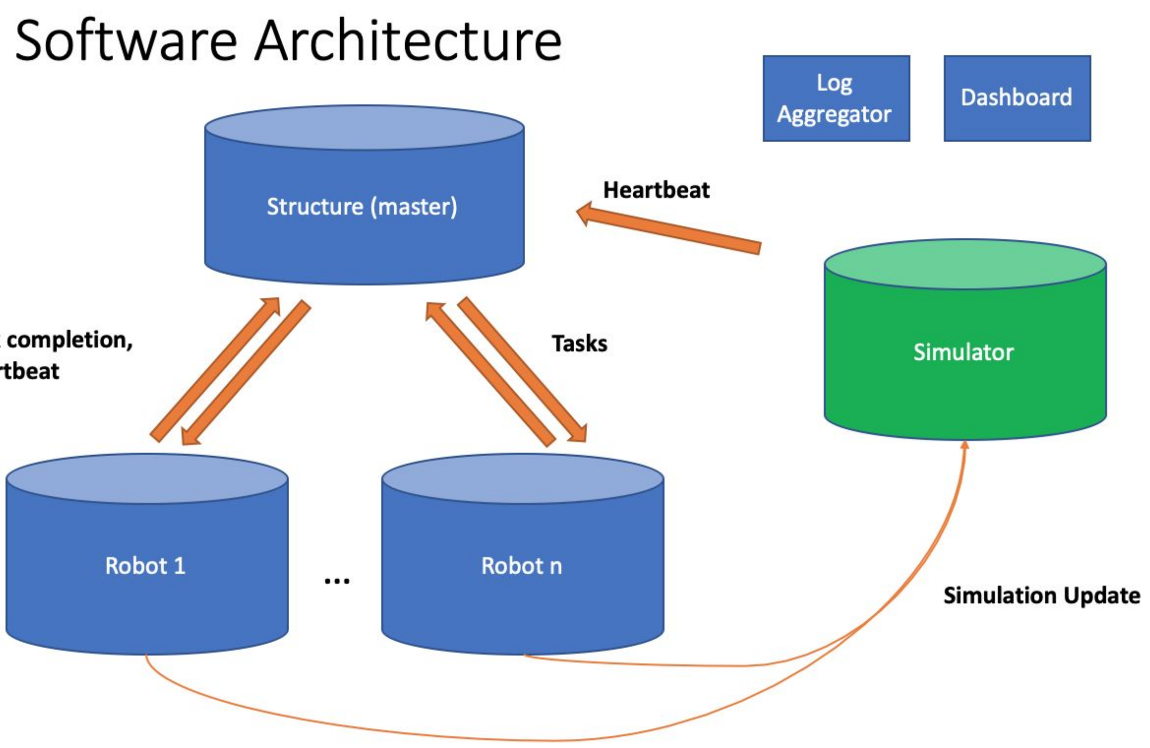


Figure 37*: Software architecture diagram*

Messaging Protocols

For messaging between entities, the ZeroMQ library was employed. This library was chosen because it is lightweight, simple, and communicates without a central broker. Other messaging libraries and protocols were considered such as Mosquito (MQTT) and RabbitMQ, but they were ultimately decided against because of their dependency on a centralized broker that manages all messages. While for this iteration of the project having a centralized message broker is not problematic, a desire to make the system more decentralized and therefore less reliant on any other entity invoked the use of ZeroMQ.

ZeroMQ connects over TCP. For now, having the system on the same network is easy and available. In the future, this messaging should be replaced with communication through the smart blocks, where messages are actually sent through the structure rather than over TCP.

The messages that are sent between the entities are as follows:

Robot Messaging

All robots send out heartbeats to the structure in order to indicate their status as being alive. Part of the heartbeat includes where the robot currently believes it is on the structure. Robots also send updates regarding their current tasks, such as the completion of a task such as placing a block or reaching a point. If the simulator is to be used, robots send simulation updates to the simulator, where the messages involve the position and configuration of the robot and its angles.

Structure Messaging

The structure receives all heartbeats from the robots and uses it to determine which robots are currently alive and working. If a robot disconnects (stops sending heartbeats), the structure will use the Lazy Pirate Heartbeat paradigm, where it will attempt to message the robot at increasing time durations. Hopefully, the robot will come back online, and if it does not, the structure will notify users (human operators) of the robot failure. The robot also monitors for the heartbeat of the simulator (if the simulator is being used) and will notify users in the event of a simulator failure. In addition to heartbeats, the structure sends tasks to all of the robots and listens for replies such as robots completing and or acknowledging a given task.

Simulator Messaging

The simulator does not send out any messages but only listens to incoming messages from the robots. The robots will send their base configuration, angle configuration, as well as if they are holding a block to the simulator, which is then able to simulate the configuration of the robot.

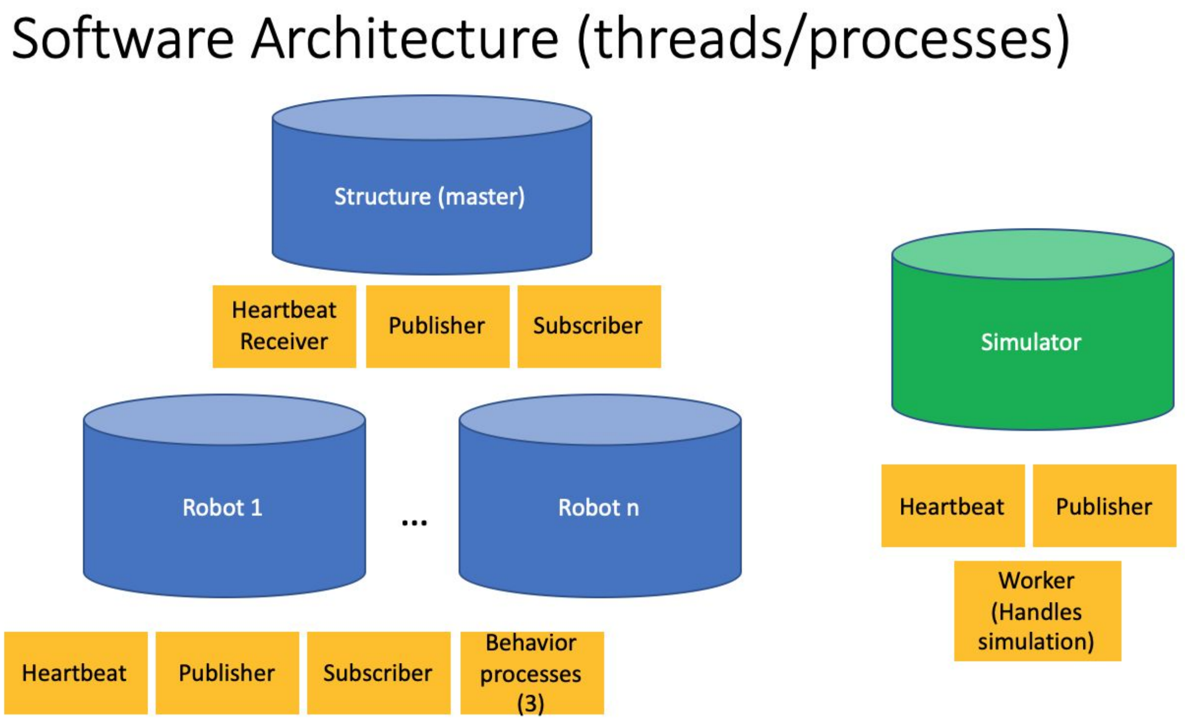


Figure 38*: Architecture showing use of threads for each application*

Robot Intelligence

Planning Execution

In order to build any structure in a collaborative and decentralized manner, the robots must be able to make decisions for themselves as to the actions they should be taking. As such, utilizing a traditional planning execution controller such as a state machine could become complicated to program, debug, and visualize. As the behaviors that the robot must exhibit become more complex, a state machine would become much more complex where states give rise to other states, fail to deal with hierarchical commands effectively, and fail to handle dynamic behaviors. As such, a behavior tree is used as the core component for exhibiting robot behaviors. A behavior tree creates a control scheme which is easy to visualize and understand as behaviors can be represented graphically as trees. It creates a well-understood hierarchy of behaviors that is much more responsive to changes in an environment than a state machine, and it can handle dynamic behaviors much more easily, as behaviors can be added, modified, or removed by simply adding, modifying, or removing subtrees from the main behavior tree. The main subtrees within the robot behavior trees involve the behaviors of receiving updates from the structure, waiting (do nothing), moving to a specified location, and building divisions. Each of these subtrees is comprised of subtrees itself that involve things such as picking up blocks, pathfinding, and placing blocks. Note that the behaviors are executed in terms of top to bottom, left to right. As shown in Figure 39, this means that receiving updates and waiting are given the highest priority for the robot, such that if the robot is in the middle of building and is told to wait, it will switch to waiting.

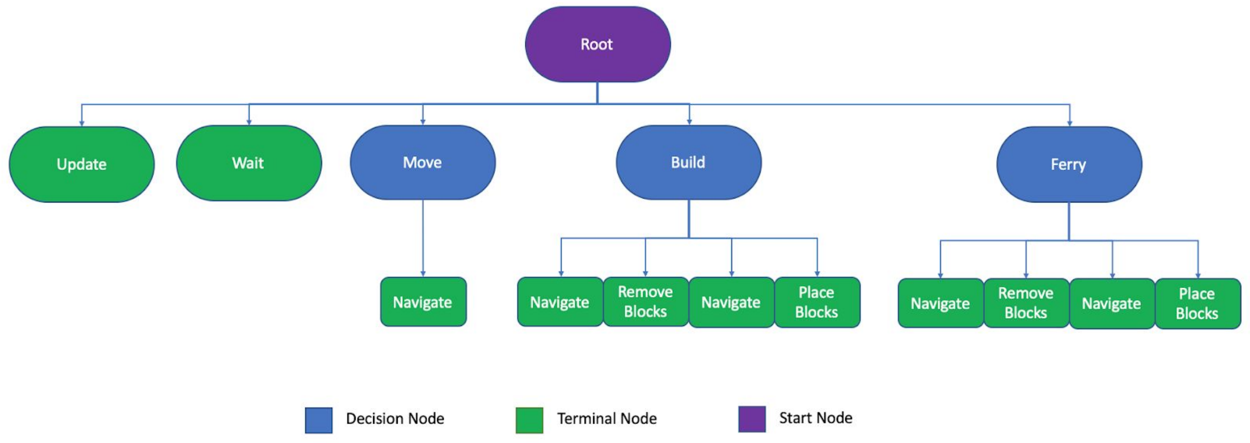


Figure 39*: Behavior tree used by all of the robots*

Constraint Satisfaction Solver

To give the robots the concept of gravity, a backtracking constraint satisfaction solver was employed. This is used to solve constraint problems regarding the removal of blocks. For example, the robots should be able to figure out that if it is to say move blocks stacked in a tower, it must remove the blocks starting at the top and move its way down to the bottom. Grabbing a block at the bottom when there are blocks on top of it would cause the tower to fall. While the blocks could be selected by simply ordering their locations (i.e. remove the blocks with the highest z coordinate value first), the solver is used to create a framework for more complicated interactions where the solutions may not be as easy as ordering and may instead require planning and constraint-based scheduling.

Collaborative Construction

In order to explain the collaborative construction building algorithm outlined in this section, a concrete example will be shown in italics font in order to aptly explain how each component works. Look for the italic sections to see the algorithm in practice. A simple flowchart that explains the main features of the algorithms can be found in Figure 40:

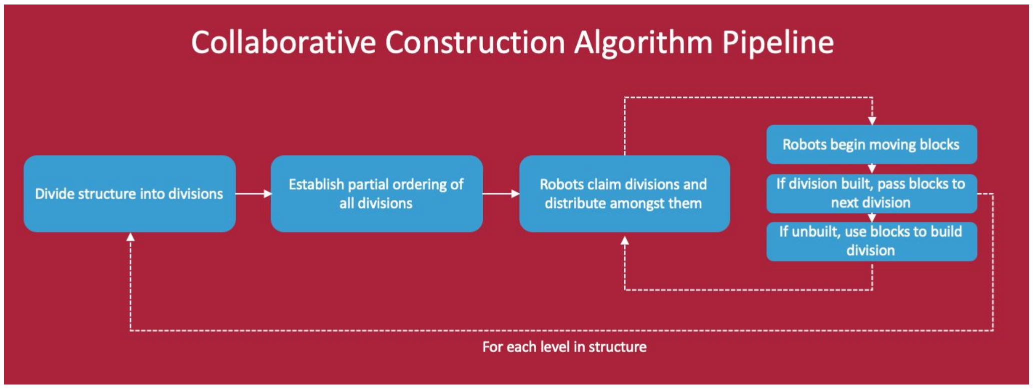

Collaborative Construction Algorithm Pipeline
Divide structure into divisions
Establish partial ordering of all divisions
Robots claim divisions and distribute amongst them
Robots begin moving blocks
If division built, pass blocks to next division
If unbuilt, use blocks to build division
For each level in structure

Figure 40*: Diagram showing collaborative construction algorithm pipeline*

The building algorithm was created with several assumptions in mind. The first is that the inchworm robots can move blocks around much faster than they can inch. For example, if the inchworms act like traditional fixed-based robot arms, they can swing around faster than it would take to move the robot base to a new location. The second assumption is that new blocks will be introduced by a human operator at a single location. This location, known as the feeding location, is where new blocks will be set down by a human for the robots to take and place. The final assumption is that decentralization should be added in future iterations, and as such, each iteration the building algorithm involves increasing decentralization.

In order to capitalize on the main assumption that robots are much faster at moving blocks rather than inching, the building algorithm limits inching. Rather than having robots travel all the way to each block, grab it, move to the block's desired location and place it, the robots instead seek to pass off the blocks to the robot that is closest to the desired location, hence "collaborative building." An analogy of the desired behavior is as follows. Imagine a line of people is attempting to put out a fire. The person at the beginning of the line has a bucket filled with water that can help put the fire out. Taking a step with the water bucket invokes some cost, namely that the bucket of water could fall or water can fall out of the bucket. In this scenario, the best action is to then pass the bucket of water to the next person standing in line, who will pass it to the next person, and so on, until the person at the end of the line who is closest to the fire will extinguish the flame. Note that, as in the scenarios, robots are not prohibited from inching. If a robot must inch in order to reach the desired destination, the robot will do so. Nevertheless, the algorithm should limit as much as possible the need for inching. The act of passing off blocks to other robots is labeled as "ferrying." Therefore, the two main behaviors of the robots are either ferrying blocks, where blocks are passed from one robot to the next, or placing blocks, where blocks are actually placed according to the building blueprint in their permanent location.

It should also be noted that a main area where the aforementioned analogy fails is the description of people passing water directly into the hands of the next person in line. This strategy would not work well for the robots as it would require the robots to grab the blocks from each other in the air, which is not a trivial task, as well as it expects that the receiving robot is currently available. If the receiving robot is not available to actually receive the blocks, the distributing (ferrying) robot would have to wait for the receiving robot to become available. Instead of handing off directly, the ferrying robots set the blocks down near the receiving robots. When the receiving robots are available, they can then take the received blocks and perform some other action with them (either ferry them to another robot or use them to build).

Division Creation

Once the home block receives the blueprint, it then divides the structure into divisions. Divisions represent an area where a robot is expected to either build or work (pass along blocks). For example, in a cube structure that is 8x8x8 block units in size, a division may be of size 4x4x2 block units, or 32 blocks total.

The size of a division involves several hyperparameters that can be tuned to produce better results. For example, if divisions are made too wide, then a robot will have to cover more area to ferry blocks in. This means the robot will inch more and is therefore not a good division size. Tuning these hyperparameters is something that is currently done manually for each structure, but it could be improved in the future through various optimization algorithms and artificial intelligence such as genetic algorithms.

For the recurring concrete example, as shown in Figure 41, a single layer of the desired structure is shown. Note how the smart blocks (shown on the left) are grouped together into divisions (shown on the right) of 3x3 blocks. While this is only shown for a single layer, this process is repeated for all layers of the structure.

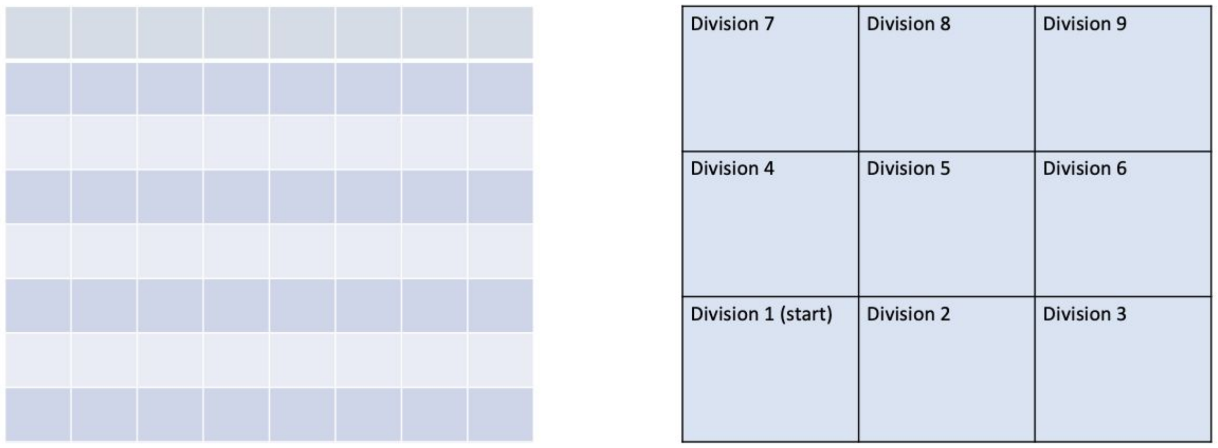


Figure 41*: Demonstration of division creation*

When seen in the simulator, a structure such as the plane in Figure 42 can be broken into the following divisions.

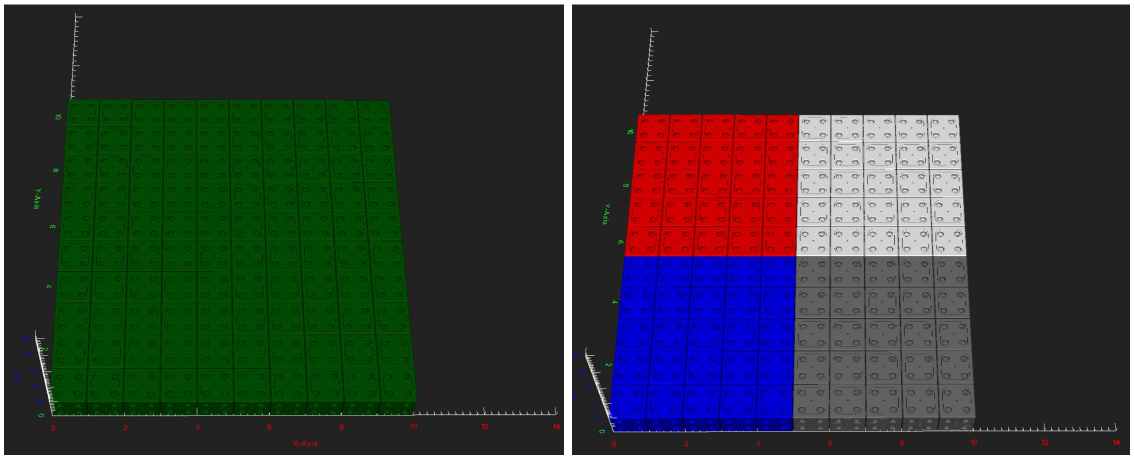


Figure 42*: Demonstration of creating four divisions within a plane*

Robot Dispersal

Based on the number of robots, the robots are sent to disperse themselves in all of the available divisions. Once the robots have dispersed themselves on the existing structure, the structure begins sending out information about the next layer to build. Robot dispersal happens in the following fashion. For all divisions that are currently being worked on, the robots will calculate their distances to the centroid of each division. Each robot will then return the location of the closest centroid to it, or in other words, the location of the centroid that it would have to move the least in order to reach. Once each robot has offered its first choice in locations, the structure will then select the largest distance (the robot whose minimum distance to a centroid is the greatest amongst all of the robots), and assign the robot with that distance that division. This process is repeated until all robots have been assigned a division.

```
max (
        [
                min (
                        robot_1.distance_to_all_points()
                ),
                ...,
                min (
                        robot_N.distance_to_all_points()
                )
        ]
)
```

This is done to ensure that robots will not have to travel far to reach a division. The division each robot is assigned should require the least amount of inching to reach across all the robots. While this algorithm is currently calculated by the structure, it is expected to be made more decentralized for the final iteration, as described more below.

Building Layer by Layer

The structure is built layer by layer in order to have the same constraints as a 3D printer. Blocks must be placed at the bottom in order to support the blocks above. For each layer, the home block runs a wavefront algorithm through the divisions in order to determine the path each set of blocks should take to build each division. The starting location of each path is the feeding location, the location from which new blocks are introduced into the system. For the purposes of this project, a human operator manually sets down blocks at the feeding location for robots to grab. Robots must receive blocks at the feeding location and ferry them to the division where they are meant to be placed. The divisions are ordered according to their distance from the feeding location (really by running the wavefront algorithm). Note that while the wavefront algorithm is used to search for paths between divisions, the Face* algorithm is still used locally by the robots themselves to navigate within divisions. In essence, this establishes a partial ordering between the divisions.

For the recurring concrete example, the feeding location occurs at division 1. From this division, the wavefront algorithm is run in order to order all divisions based on their location from the feeding location. Note how division 2 (with a value of 2) is ahead of division 9 (with a value of 5).

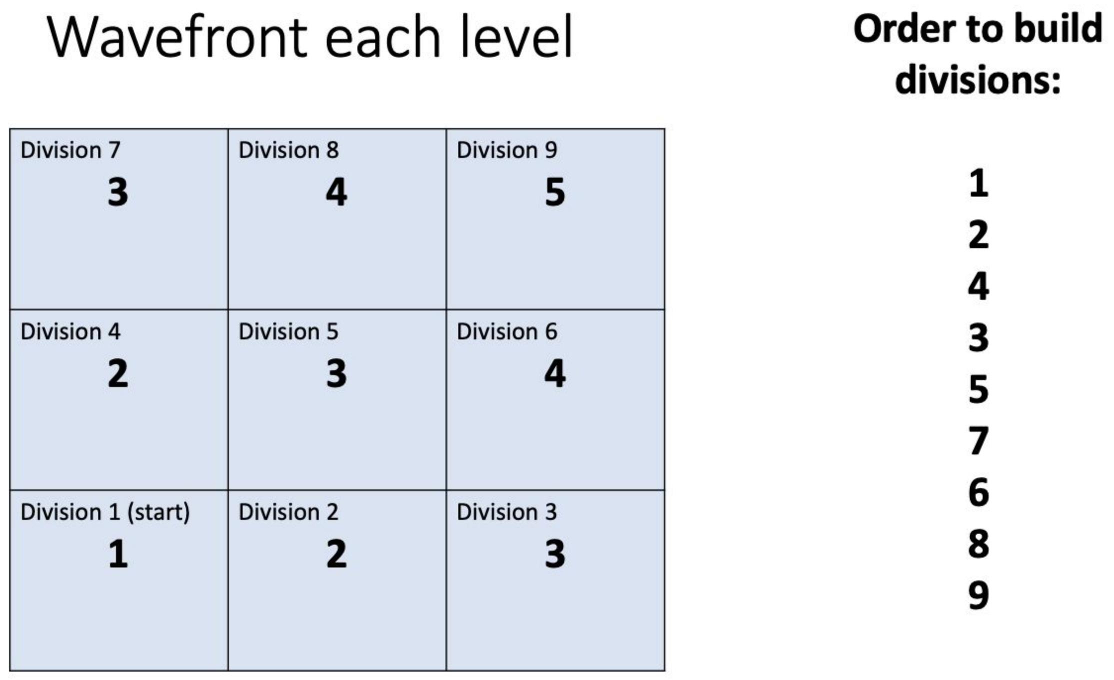


Figure 43*: Wavefront algorithm is performed for each division in level*

At this point, the robots will be added to the structure. For this example, four robots have been manually placed in divisions 1, 2, 5, and 4. Note that the status of the division is denoted as either empty (no blocks placed yet), CW (claimed and waiting meaning a robot is currently waiting in the division), filled (blocks have been ferried in division), and done (division has been built).

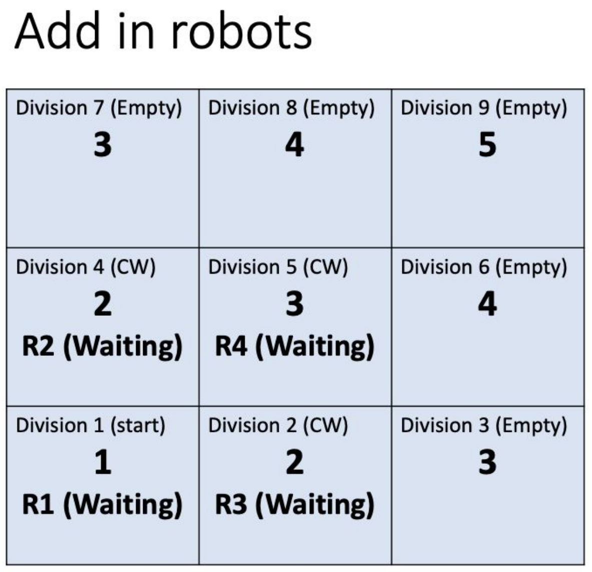


Figure 44*: Robots have been added to the building algorithm example*

These nodes are then used to populate two main priority queues that both the home block and robots iterate over. The first queue is the goal queue, which represents all of the divisions that are currently being built. In the first iteration of the algorithm, only two divisions are built at a time. The second queue is the “nodes-to-visit” queue. For each node in the goal queue, the path through all of the divisions from the feeding location to that division is determined. These paths are then merged into a single path. For example, both paths will start with the feeding location as the first division that must be visited; this division, however, should only show up once in the queue. Through the process of merging, divisions that are common to both paths are noted. The robots that claim these divisions must know that they need to ferry twice as many blocks, as that division is not in the path of just one goal node but both goal nodes.

For the recurring concrete example, two divisions are removed from the list of divisions that need to be built as shown in Figure 45. To keep the example simpler, it is assumed that the first division is already built (this assumption is purely to simplify the example and does not change how the algorithm works). The first two divisions removed are 2 and 4, and are added to the goal nodes priority queue (GPQ). For each division in the GPQ, the path is found from the feeding location (division 1). Note that the path is not recalculated, it is simply the path found from running the wavefront algorithm previously. Because the paths to divisions 2 and 4 share a common division of division 1, the path is flattened such that division 1 only appears in the intermediate nodes priority queue (IPQ) once. This step is important so that the robot is division 1 knows that it must ferry blocks to the robots in divisions 2 and 4, not just division 2 or 4. In other words, it must ferry twice as many blocks in order to ferry to both divisions at the same time.

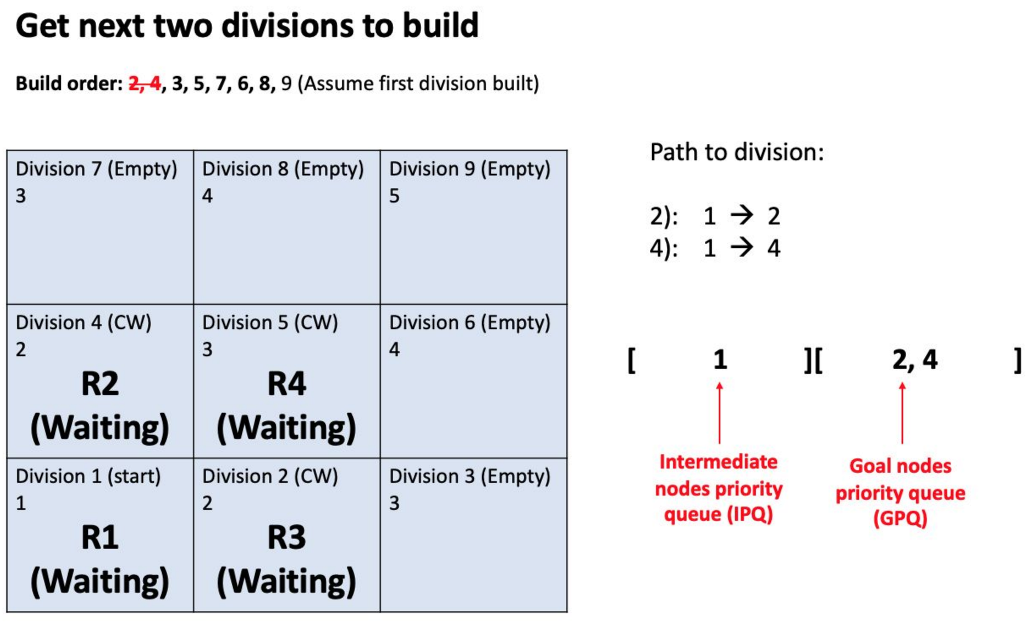


Figure 45*: Populating the queues to begin the building algorithm*

Once the nodes-to-visit queue has been populated, the robots iteratively pop off as many nodes in the queue as there are robots. Each robot will then begin exhibiting the behavior needed by that node (division).

Initially, all robots except the robot whose division includes the feeding location will exhibit the waiting behavior.

For the recurring concrete example, the robots first begin popping off the divisions from the IPQ until there are none left as shown in Figure 46. Because there is only 1 division (division 1) in the IPQ, the robots then move to remove divisions from the GPQ until there are none left. Because there are only three divisions in the queues but four robots, one robot will maintain its waiting status.

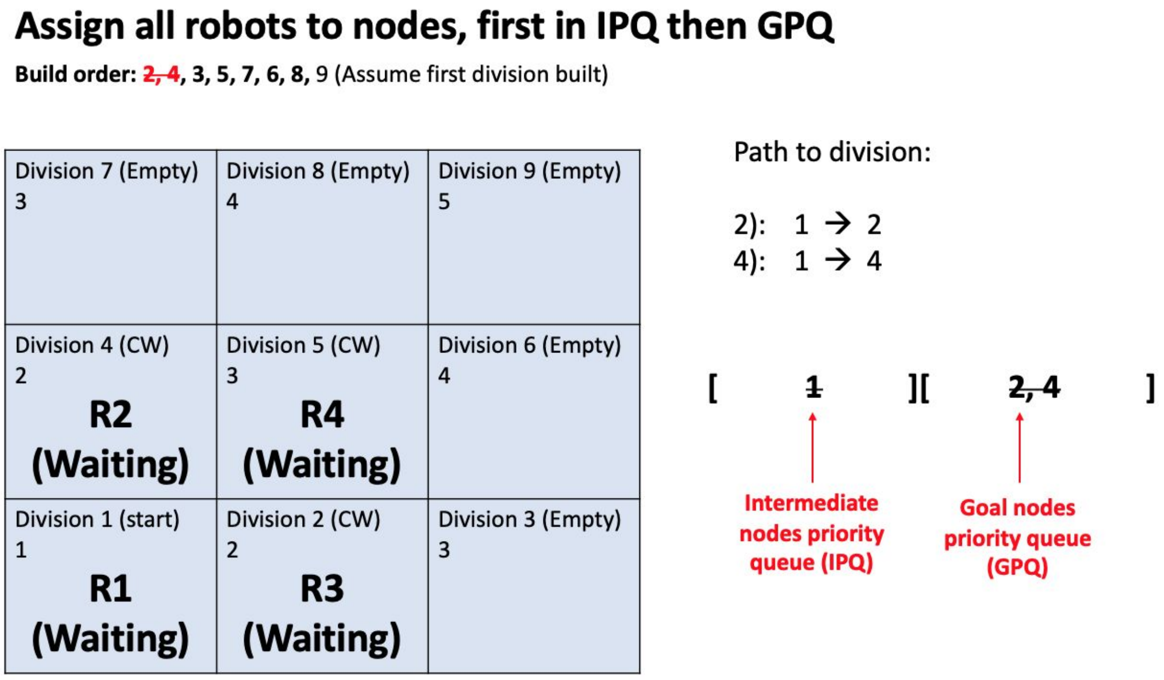


Figure 46*: Example of robots claiming divisions from the building queues*

In the simulation, this division claiming is shown in Figure 47.

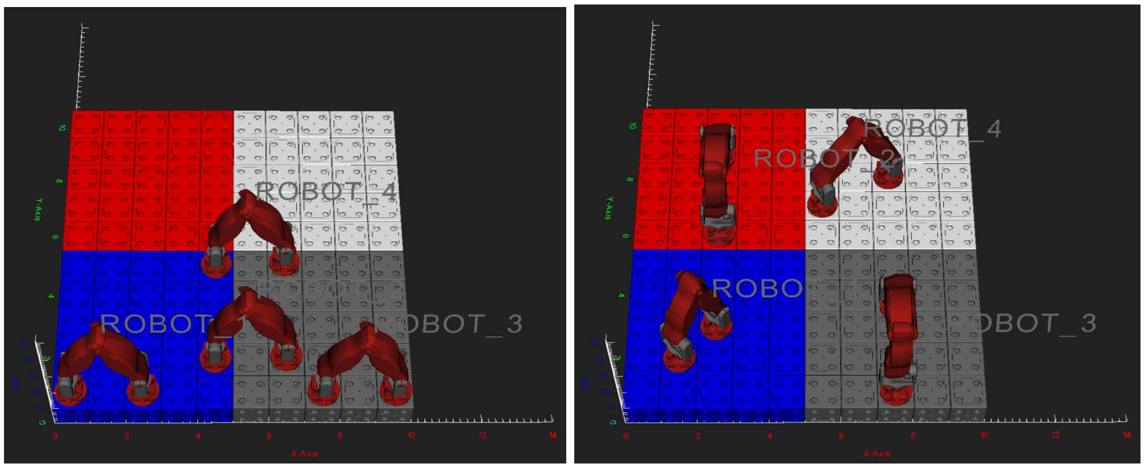


Figure 47*: Example of robots claiming divisions in simulation*

The robot whose division includes the feeding location will begin by ferrying the blocks to the other robots. The robot knows how many blocks to ferry by checking the number of blocks of each goal division. Furthermore, it knows where to place each block for another robot to grab by checking the next points in the global path and then placing the blocks biased towards that location. For example, if the next two divisions the robot is ferrying for are to the front and right of the current divisions, the robot will ferry blocks biased towards the front and right locations of its division, such that it is easier for the next robots to grab.

The ferry locations are determined specifically by biasing towards the direction of the other divisions (front, left, back, right), and then placing blocks furthest away to closer. A minimum distance threshold is set so that the robot does not ferry blocks too close to itself. Note that this could be dangerous when another robot comes to grab the ferried blocks; at this point, the other robot could crash into the robot that was just ferrying. Once the minimum threshold has been surpassed and there are still more blocks to ferry, the robot will then repeat the process at a level higher, or in other words, the robot will begin stacking blocks on top of the blocks it just ferried.

Figure 48 shows a division made of 8x8 blocks. The regions marked in red are designated as the ferry regions. For this example, the ferry region is of size 2. These are the locations that the robots will place blocks so that a robot in another division can grab the blocks from this division more easily. If the robots need to place more blocks than the number of blocks shown in red, it will then begin stacking the blocks on top of one another, repeating the process just at the next level. In this example, the blocks are biased towards the top of the division and the right of the division, so that a robot in the divisions either to the top or right of the current division can grab the ferried blocks.

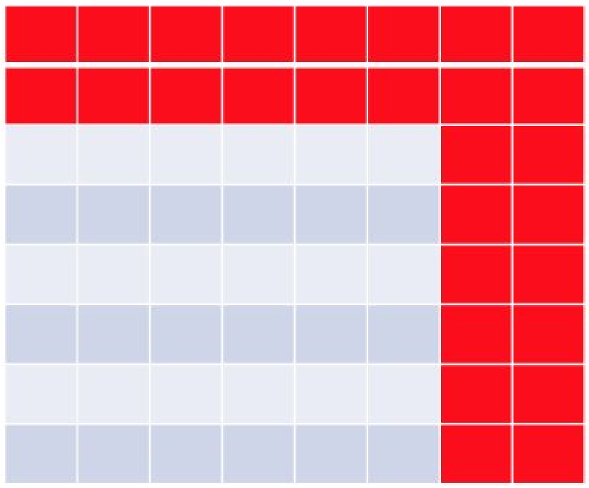


Figure 48*: Ferry regions in a division*

This process of ferrying is repeated until the robots reach a division that must be built. Divisions are built in a spiral fashion, starting from the center and moving outwards. This is done so that after the first block is placed at the center (or near the center of the division), the robot can then stand on top of the block that was just placed and from there grab new blocks to build. This is done so that the robot can act like a fixed-base robot arm and simply swing around in a circular fashion, placing blocks as it swings. Of course, depending on the size and shape of the divisions, acting like a fixed-base robot arm will not always be possible, so sometimes inching will be required to reach/place all blocks. An example of what ferrying looks like in simulation can be found in Figure 49.

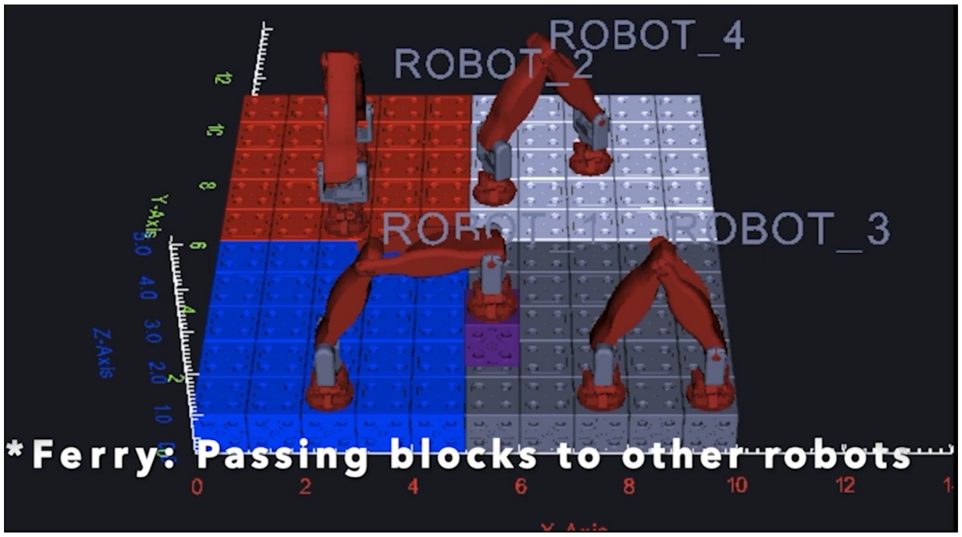


Figure 49*: Ferry regions in a division*

For the recurring concrete example, robot one will stay in division 1 and ferry the blocks to divisions 2 and 4 (the goal nodes). Once robot one has finished ferrying the blocks for divisions 2 and 4, it will notify the robots in those divisions, robots two and three. These robots (because they are at the goal nodes themselves), will begin building instead of ferrying. They will take the blocks that were ferried by robot 1 and place them in their divisions in a spiral fashion starting at the center and moving outwards. This process is illustrated in Figure 50.

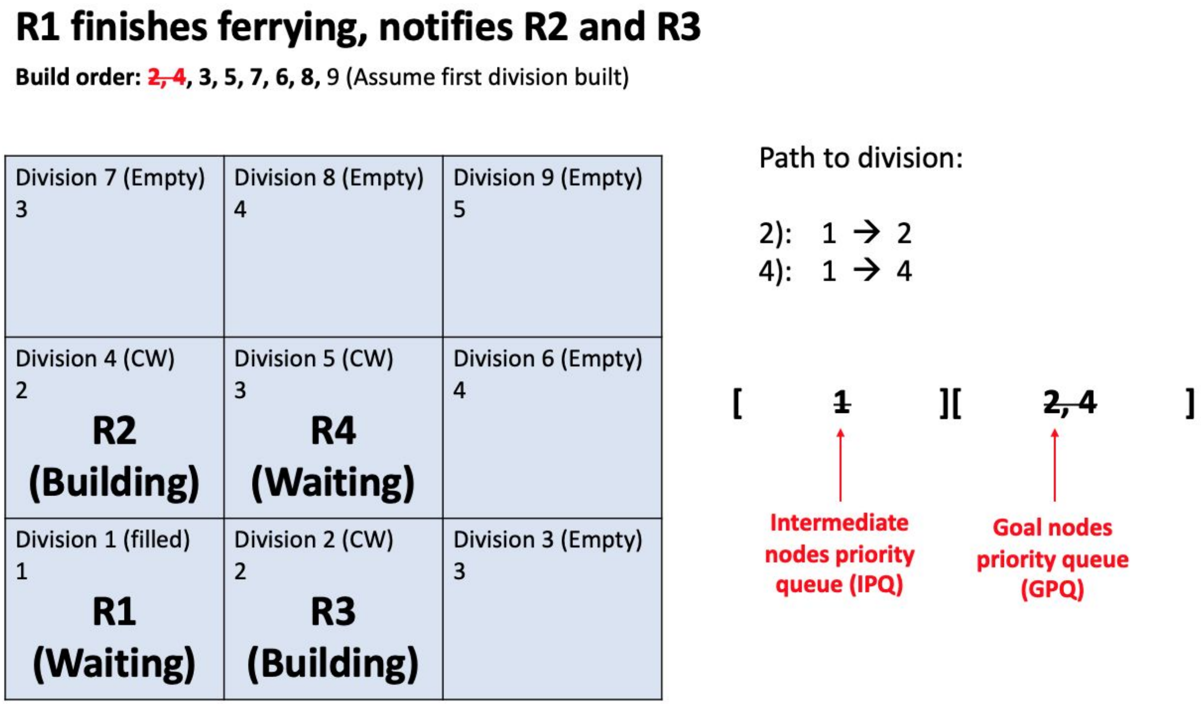

R1 finishes ferrying, notifies R2 and R3
Build order: ~~2, 4~~, 3, 5, 7, 6, 8, 9 (Assume first division built)
Division 7 (Empty) 3
Division 8 (Empty) 4
Division 9 (Empty) 5
Division 4 (CW) 2 R2 (Building)
Division 5 (CW) 3 R4 (Waiting)
Division 6 (Empty) 4
Division 1 (filled) 1 R1 (Waiting)
Division 2 (CW) 2 R3 (Building)
Division 3 (Empty) 3
Path to division:
2): 1 → 2
4): 1 → 4
[~~1~~][~~2, 4~~]
Intermediate nodes priority queue (IPQ)
Goal nodes priority queue (GPQ)

Figure 50*: Two robots building a division*

Once either ferrying or building has been completed, the robots will then disperse themselves again. This is to ensure minimal waiting time. Because robots may be in the middle of a task and then be asked to disperse themselves, the robots will send the current state of their division to the structure. If a robot was interrupted and did not finish ferrying or building, the next robot that is assigned to that division will pick up where the previous robot left off.

For the recurring concrete example, a few more divisions are shown for how the algorithm works. Once divisions 2 and 4 have been built, two more divisions are added to the GPQ based on the build order. In this case, these are divisions 3 and 5. As before, the path to each is found starting from the feeding location. These paths are merged (common node of one between the two paths) and then added to the IPQ. Because the order of divisions 2 and 4 are the same (both with a value of 2), it does not *matter which division is placed first in the IPQ, so long as both are placed after lower value division division 1 (value of 1).*

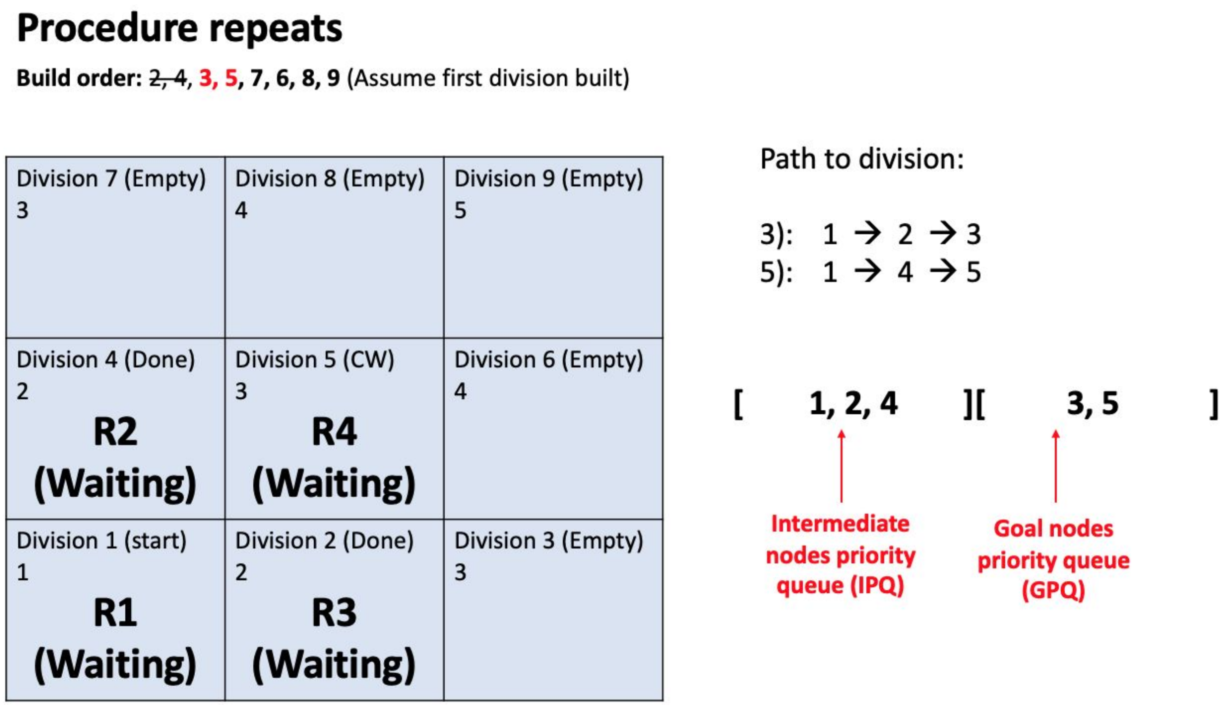


Figure 51*: Building algorithm repeats the procedure for the next two divisions*

Now that the queues have been populated, the robots begin claiming divisions from the queues. Because there are four robots, they will claim four divisions, starting from the IPQ and then being claiming divisions from the GPQ. Divisions 1, 2, 4, and 3 are claimed by the robots. Using the algorithm described above, the robots redistribute themselves based on minimizing the amount of movement needed to reach each division as shown in Figure 52. Note that to minimize the amount of inching performed by the robots it is more efficient to have robot three move to division 3 despite the fact that it is already in division 2. Robot four will then move down to reach division 2. If this was not done, robot four would have to inch a much longer distance across the diagonal to division 3.

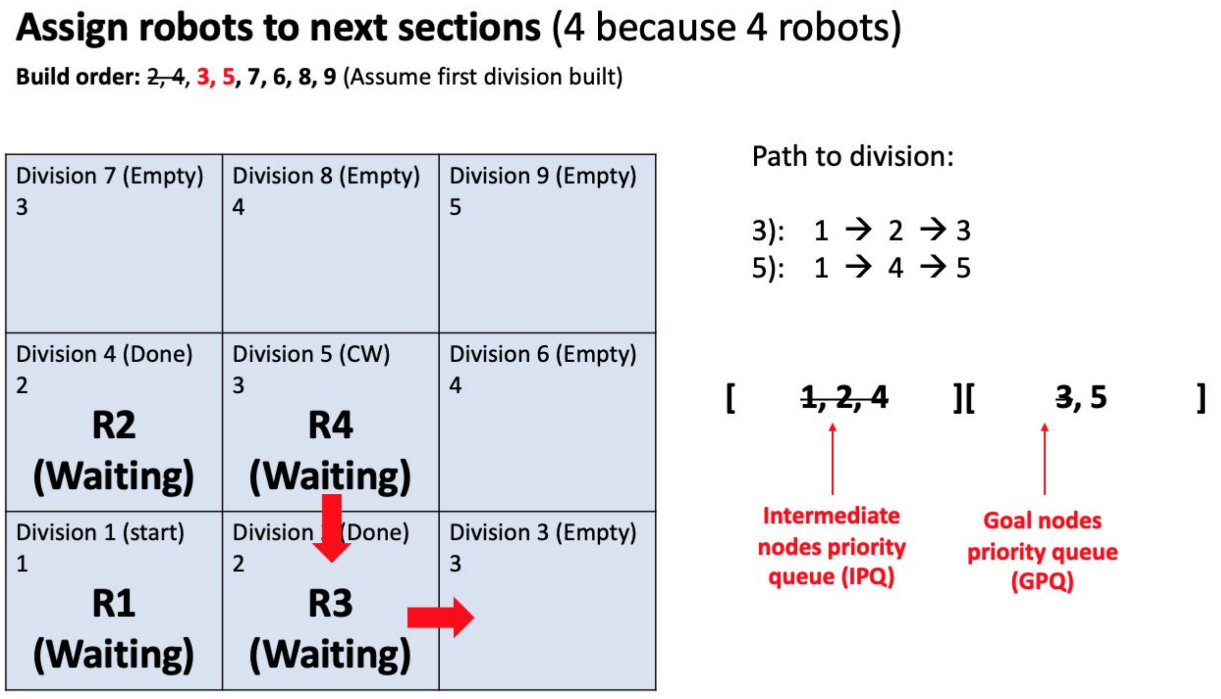


Figure 52*: Robots claim divisions and then redistribute themselves in a way to minimize the amount of inching needed for each robot*

Once all robots have distributed themselves, robot one begins ferrying from the feeding location for divisions 4 and 2 as shown in Figure 53.

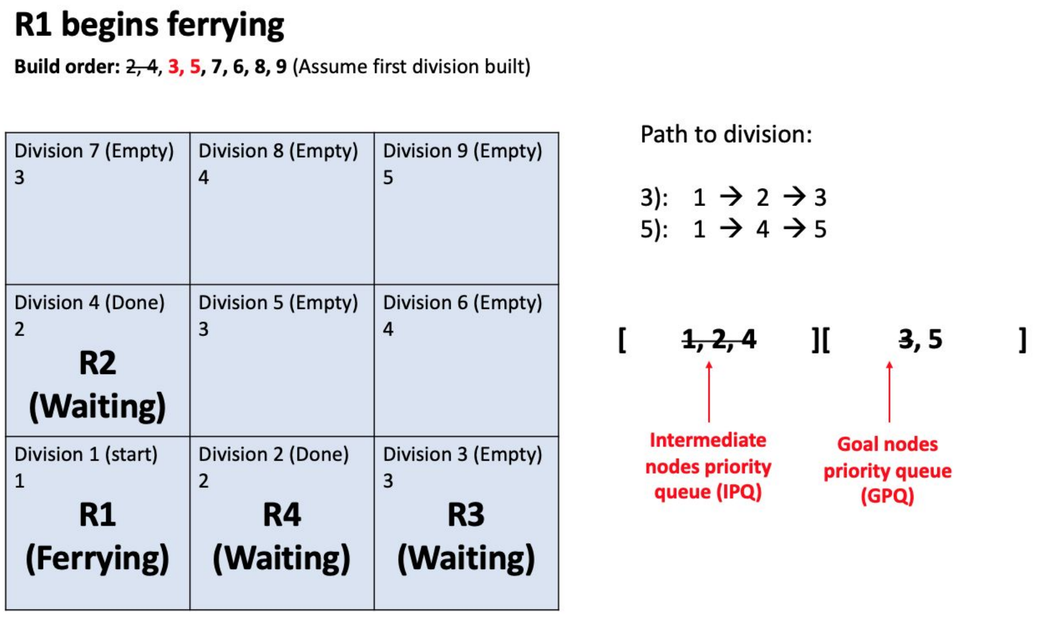


Figure 53*: Robot one begins ferrying blocks for the next divisions*

Once robot one finishes ferrying, the next division is removed from the queue as shown in Figure 54. Because the IPQ is empty, the next division removed is from the GPQ: division 5. Because a new division has been claimed, the robots redistribute themselves again. To minimize the amount of inching for each robot, robot one will move to division 4 and robot two will move to division 5.

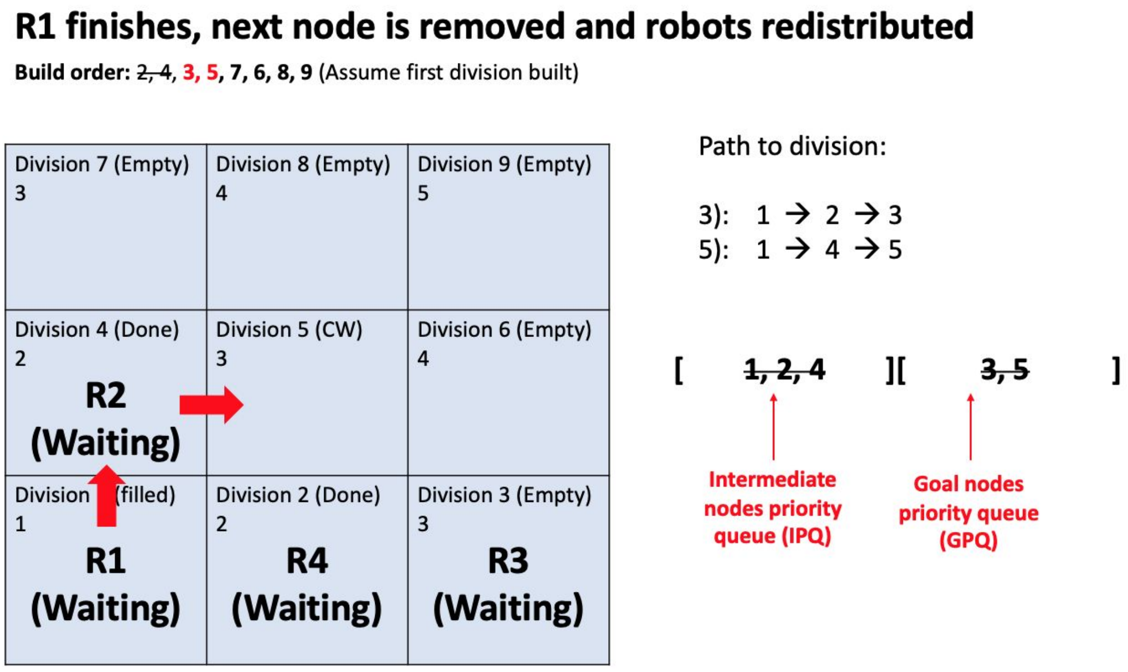


Figure 54*: Robot one finishes ferrying, the robots redistribute themselves*

In these divisions, the robots will continue the process of ferrying and then building until the goal nodes have been built. The entire process is performed for all divisions in a single layer.

This building algorithm easily scales to three dimensions by repeating the same process for each layer. If the feeding location is kept at the bottom of the structure, the robots can perform the same path planning and wavefront ordering up the side of the structure, as shown in Figure 55. This algorithm was made to be agnostic to the number of robots; it can easily and dynamically be rescaled to the number of robots available. Furthermore, it is only impacted by the shape of the structure being built in the sense that the ferring behavior will devolve to entail more inching. As such, the structure will still be able to be built, it will just take longer than a structure like the cubic structure shown in Figure 55.

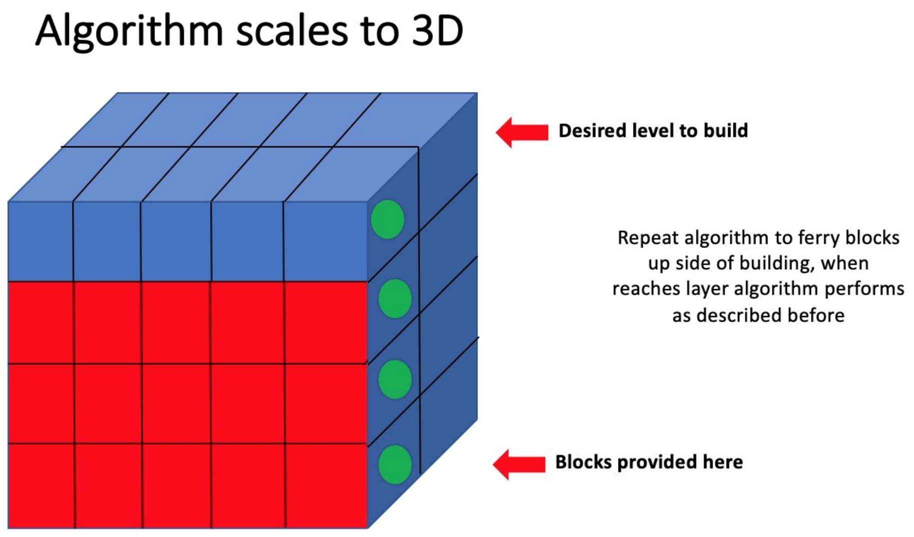

Algorithm scales to 3D
Desired level to build
Repeat algorithm to ferry blocks up side of building, when reaches layer algorithm performs as described before
Blocks provided here

Figure 55*: Building algorithm scales to three dimensions*

Building Algorithm Proof of Correctness

A proof of correctness was created for the building algorithm to showcase its capabilities to build different structures and its ability to allow collaboration between robots. The proof is as follows.

Face* is proven complete

Face* is a variation of uniform cost search, an algorithm that is already proven to be complete. Face* is used to calculate a path from one block, Block A, to another block, Block B. If a path exists, then the robot is capable of reaching Block B. Each node in the algorithm that is searched is a face of a block. The cost calculated for each face is the Manhattan distance from the center of all the faces. The faces that are considered are the faces that can actually be reached by some mechanical motion of the robot. The possible motions of the robot are shown in Figure 56.

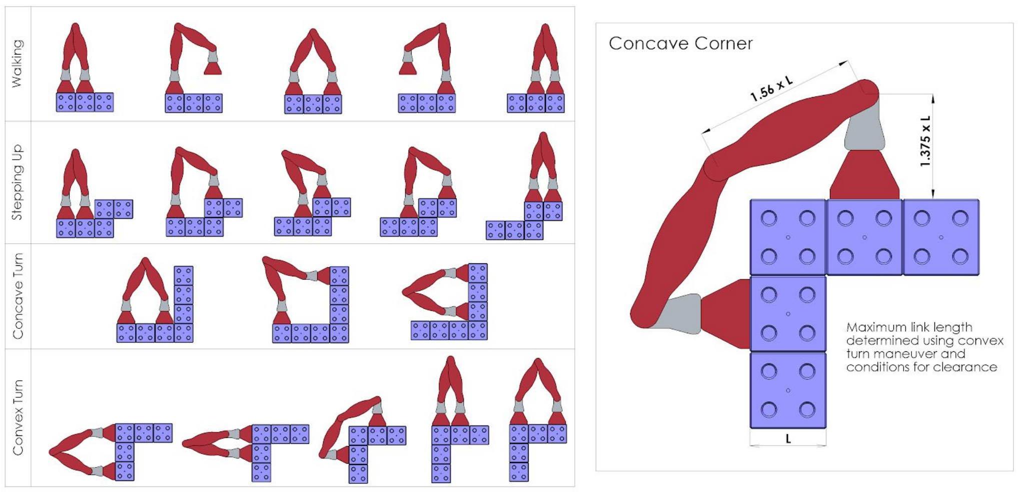

Walking
Stepping Up
Concave Turn
Convex Turn
Concave Corner
1.56 x L
1.375 x L
L
Maximum link length determined using convex turn maneuver and conditions for clearance

Figure 56*: Diagram showing navigation conditions of the robot*

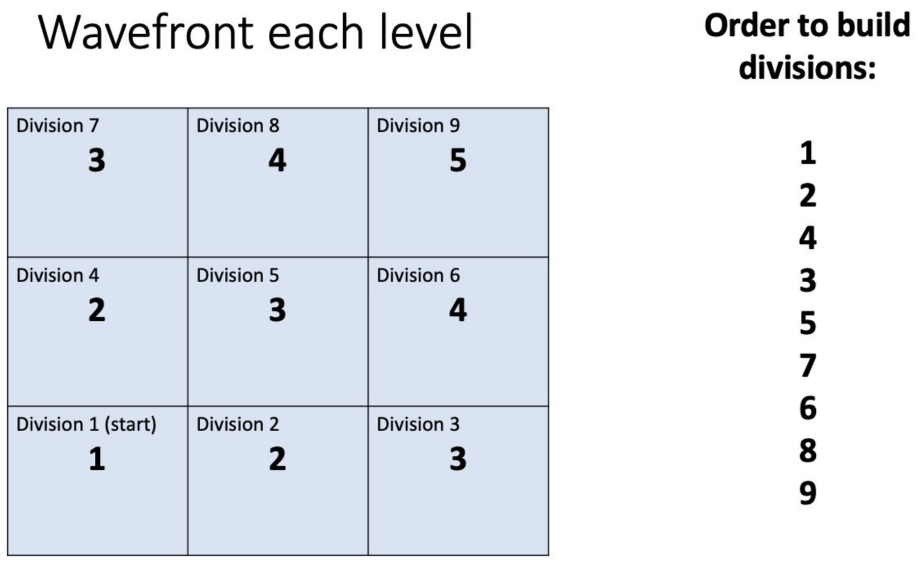
The algorithm also must take into account the space surrounding the path as it searches so that there is enough space for the robot to actually move. For example, based on the height of the robot, the robot will not be able to fit inside a cave or overhang that is less than five blocks tall. Accordingly, the search graph the algorithm uses can be defined as a reachability graph, in that it only considers the faces of blocks that the robot can actually reach. Therefore, if Face* returns a path, it is possible for the robot to reach the specified location. If Face* does not return a path, then the robot is incapable of reaching the specified location.

Building inside divisions is correct

Building inside divisions means that blocks are either placed or moved around (ferried) inside of divisions. So long as the blocks are placed such that there exists a path from the robot's starting location to where the block is to be placed, building structures through the use of divisions is correct. In other words, blocks are always placed so that the global connectivity of the structure is maintained, as there must be a path that allows the robot to reach the desired location in order to place the block.

Structures Can Be Built In Parallel

Based on the way the robots claim divisions to either build or ferry in, a partial ordering is established for all available divisions. This partial ordering is determined based on each division's distance from the feeding location, the location where a human operator introduces new blocks into the system. The partial ordering is used first to ensure robots do not collide with one another. After a division has been claimed, other robots are not allowed to enter that division. Furthermore, the partial ordering is a means to measure the progress of construction. Divisions that are ranked earlier are built before divisions ranked later. As such, even when blocks are ferried within already-constructed divisions, the progress of construction can be measured as the divisions that have been built no longer have an ordering. This also ensures that Face* will return a valid path to each division. Divisions are built in an outwards fashion from the feeding location. The ordering ensures that certain divisions are built before others so that there always exists a path to each division. In the image shown below, where the feeding location is Division 1, Division 1 must be built before Divisions 2 and 4, so that there exists a path for the robots to reach Divisions 2 and 4.



The method that the partial ordering is performed allows several divisions to have the same ordering. For example, Divisions 2 and 4 have the same ordering, as do Divisions 3, 5, and 7. These divisions share the same constraints– the divisions that must be built before them. For example, by building Division 2, a path now exists for both Division 5 and Division 3. Accordingly, these divisions can be built at the same time (parallelized) as robots can use the existing structure to find a path (through Face*) to both divisions. It should also be noted that Division 7 shares the same constraint, and so it too can be parallelized with Division 5 and Division 3.

This final theorem proves the correctness of the algorithm. Once all divisions have been built the structure is complete.

The structures that this algorithm is capable of building are any structure that maintains global connectivity such that there exists a path to every single block in the structure. Furthermore, because the algorithm currently builds in a bottom-up fashion, structures that violate bottom constraints cannot be built. For example, overhangs that extend both outward and downwards (like the wings of an airplane) cannot be built, as they require constructional constraints from above/the side, not from below. In order to build these types of overhangs, support material (like that of a 3D printer) would need to be built and later removed to allow these constraints, something described more in Future Work.

Experimental Evaluation

The results presented in this work will focus on robot specific system level achievements relevant to future applications, as well as the achievements of the building algorithm.

Robot Evaluation

Demonstrations

The ability of the inchworm robot to navigate its environment was evaluated. It was experimentally demonstrated that the robot is capable of traversing on flat ground, and is capable of manipulating block elements in its environment. The electromechanical design of the robot theoretically enables the traversal of other obstacles such as staircases, concave, and convex corners. Experimental demonstrations of these motions have yet to be completed, and this is one area for future work. Evaluation of the robot's ability to traverse structures is critical to ensuring that the construction platform is capable of assembling a wide variety of structures. Figure 57 below shows snapshots of the robot during a demonstration. The robot inches forward two block units, and moves a block forward by one space. This demonstration took 20 seconds to complete, but this can vary largely depending on the software parameters used.

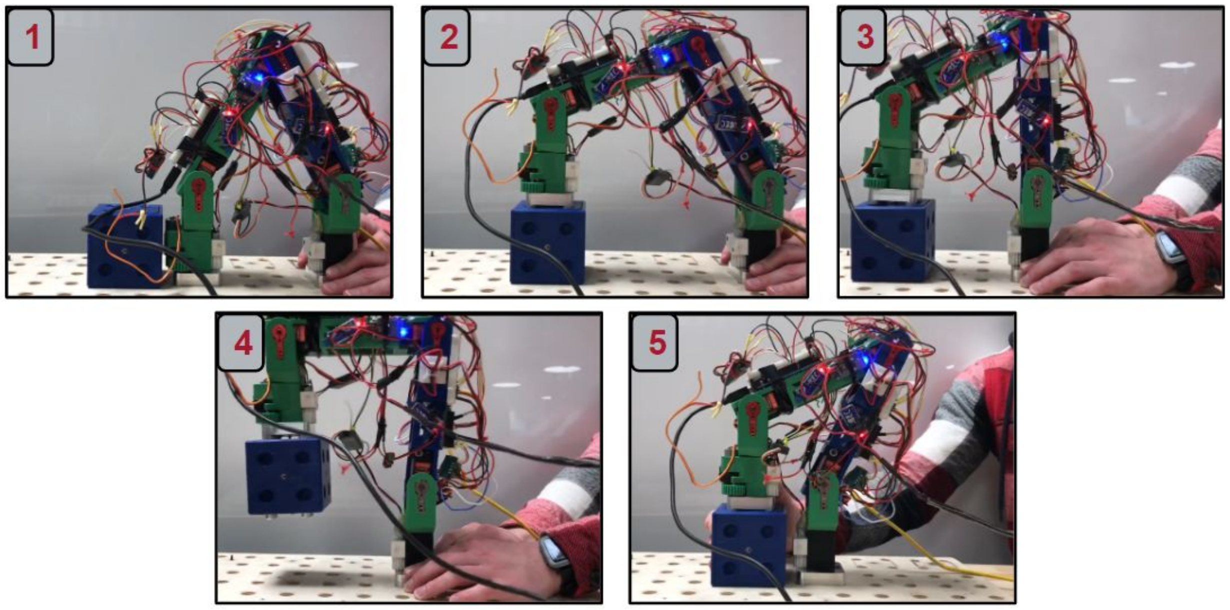


Figure 57: *Timesteps of robot (Iteration 2) demonstrating control over a block element*

Smart Blocks Evaluation

Demonstration

The smart block system was tested using three PN532-based NFC systems. This testing showed the smart blocks can successfully recognize blocks added to the structure, assign a new block a location, and notify the homeblock of the addition. This system also successfully recognizes when a smart block is removed from the structure.

Collaborative Building Algorithm Evaluation

Demonstrations

Through simulation, the building algorithm has been shown to successfully provide a means for which robots can build structures in a collaborative fashion. The building algorithm was tested for multiple structures including structures resembling pyramids, temples, churches, castles, the Empire State Building, and the Star Trek Reliant ship, among others (Figure 58). The robots were able to place all blocks as required. Unfortunately, we were not able to demonstrate the robot climbing vertically. This is due to insufficient time to complete the code necessary to do this and was not caused by issues with any of the algorithms developed in this paper.

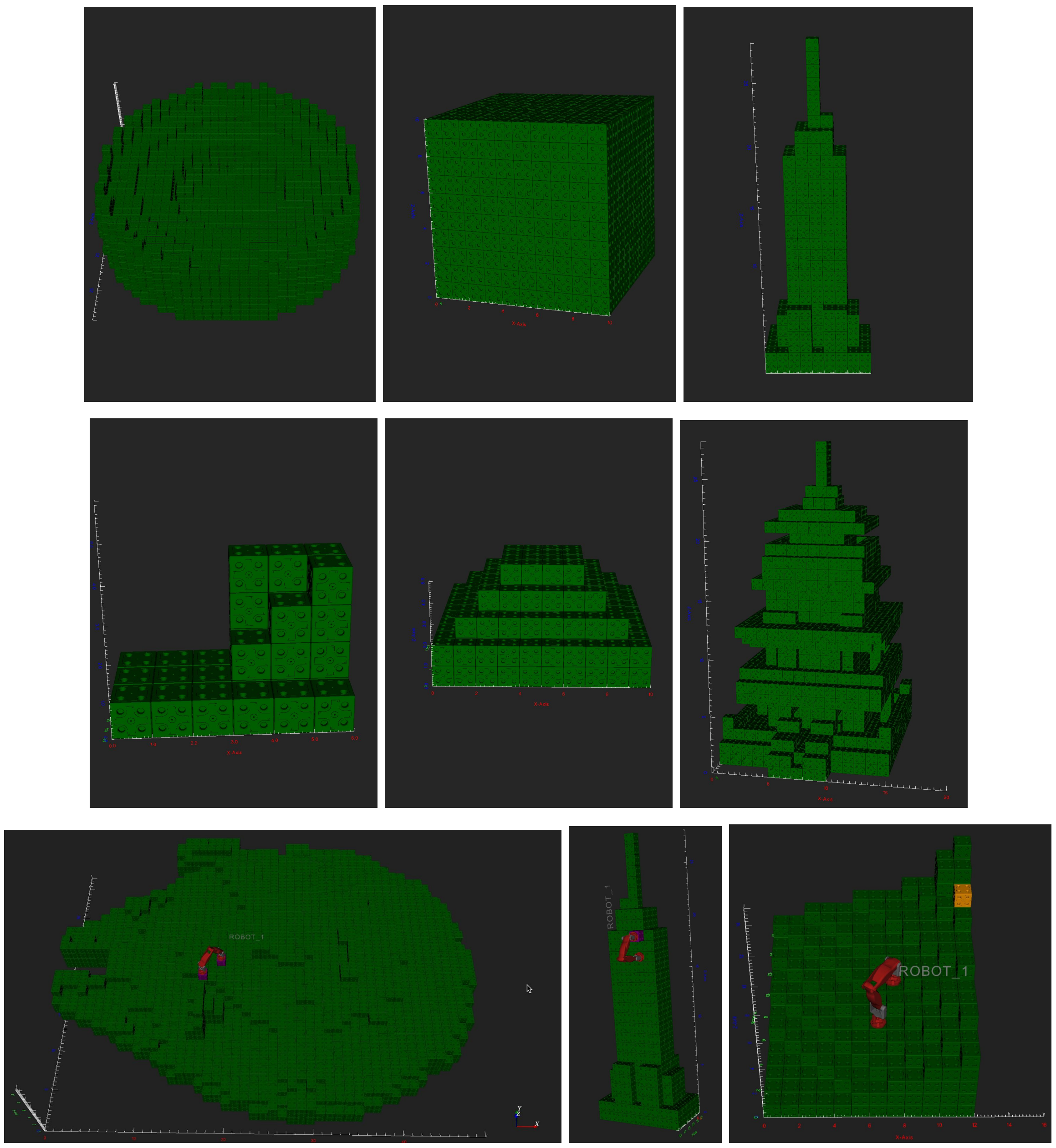


Figure 58*: Examples of structures built, robots repairing structure*

Speed

In order to analyze the effectiveness of the building algorithm at reducing the time required for construction by adding additional robots, simulations were created wherein varying numbers of robots were tasked with building the same structure. Due to hardware restrictions (running multiple robots on a single computer can become process intensive), simulations were run on a 2015 MacBook Pro with up to four robots per experiment. The time measure was the simulation time, which represents the number of timesteps of the simulator. The results of one experiment can be found in

Figure 59. One robot, two robots, three robots, and four robots were tasked with building a 10 block by 10 block plane consisting of 100 blocks. As shown in Figure 59, the time significantly decreases when adding additional robots, especially when comparing a single robot placing all 100 blocks and four robots placing all 100 blocks. This model seems to be characterized by an exponentially decaying curve. Adding more robots will decrease the time required to build exponentially up until a threshold is reached, upon which adding additional robots will not produce any additional benefit.

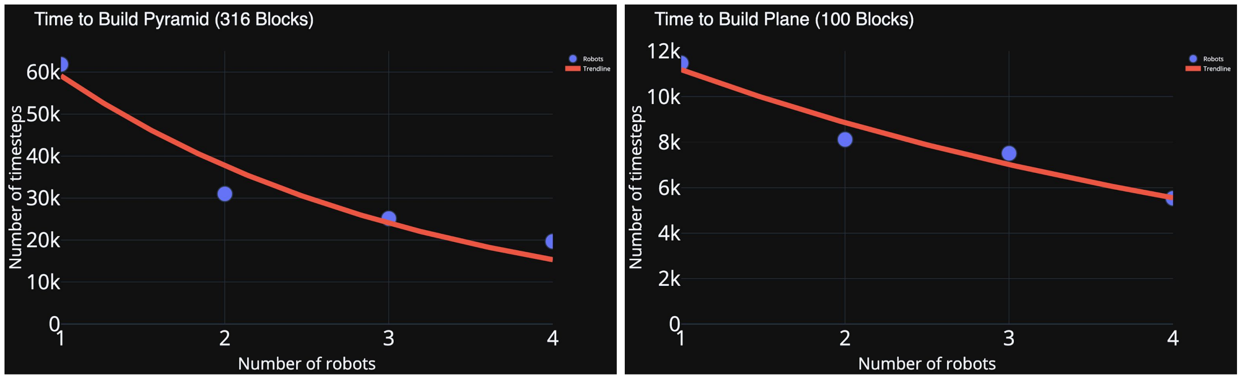


Figure 59*: Graphs showing time required to build structures decrease when adding more robots*

Collaboration

To analyze how the collaboration between robots affects performance, the behavior trees of each robot were timed. For each robot, it recorded the total amount of time spent in each behavior. The main behaviors tracked were receiving updates from other robots and the structure (Update), moving (Move), waiting (Wait), building (Build), and ferrying (Ferry). The results shown in Figure 60 were generated from the same experiment run above (a 10 by 10 plane made of 100 blocks).

For the single robot experiment, the robot spends almost all its time either ferrying or building. Almost no time is spent waiting, and a very small portion of time is spent receiving updates. This makes sense as there are no other robots to coordinate with or wait on. While this means the efficiency of the single robot is very high, it also means that this is a very intensive and power hungry task for one robot to complete.

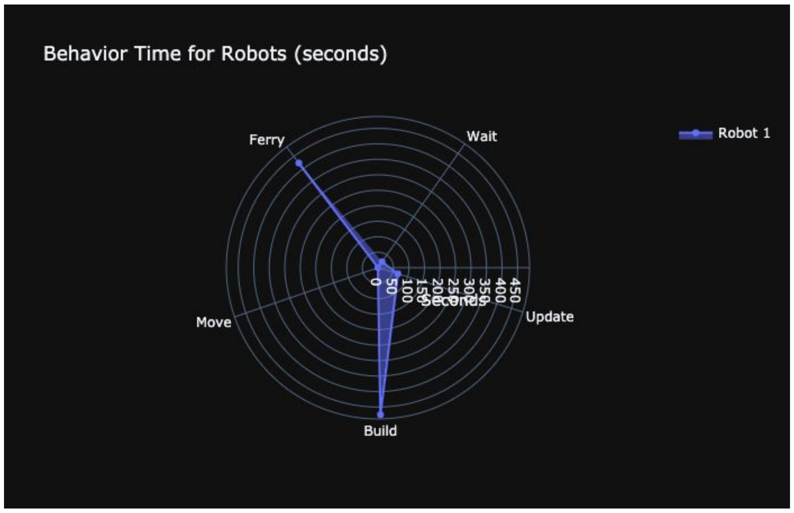


Figure 60*: Graph showing time spent performing behaviors for a single robot*

For both the two robot and three robot experiments, the time spent in behaviors was much more divided as shown in Figure 61. One robot was shown to be doing a vast majority of the ferrying. This is most likely explained by the structure the robots were building, as the plane has only three divisions which require ferrying. The other robot(s) were shown to be carrying out more of the building. Notice that the waiting time and time spent receiving updates of all robots is significantly increased compared to the single robot experiment. This makes sense as by increasing the number of robots, the communication between the robots and the structure vastly increases, as it requires increased coordination.

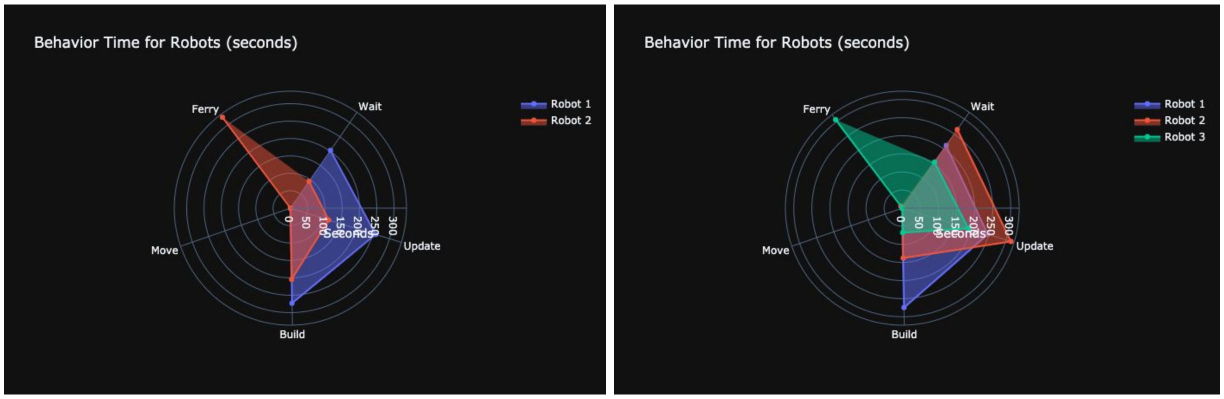


Figure 61*: Graph showing time spent performing behaviors for two robots and three robots*

For the four robot experiments, the results were very similar to both the two robot and three robot experiments– the waiting time of the robots increased significantly as shown in Figure 62. What differed is that the building time was more evenly distributed across the robots compared to the other experiments. The ferrying behavior also was a little more distributed between the robots.

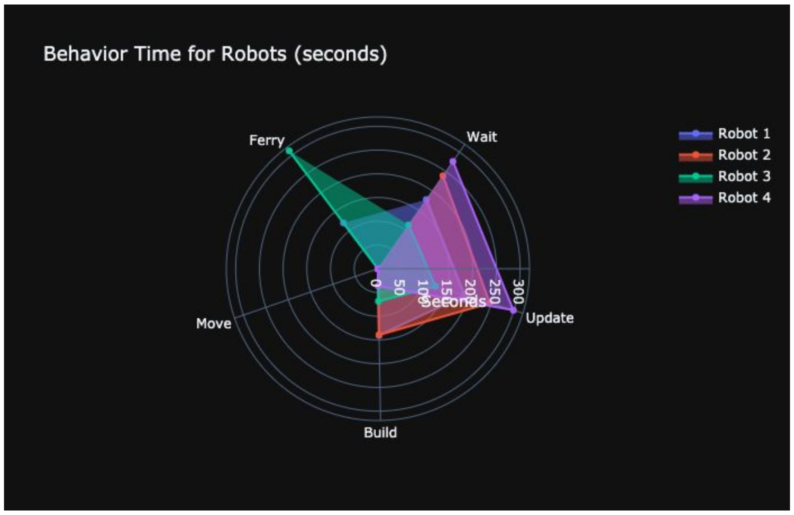


Figure 62*: Graph showing time spent performing behaviors for four robots*

A comparison of all the experiments can be found in Figure 63. These results show that the building algorithm may favor certain robots to do more work than others. As such, future improvements could be made to this algorithm by analyzing these behavior times to more evenly distribute the distribution of tasks amongst the robots.

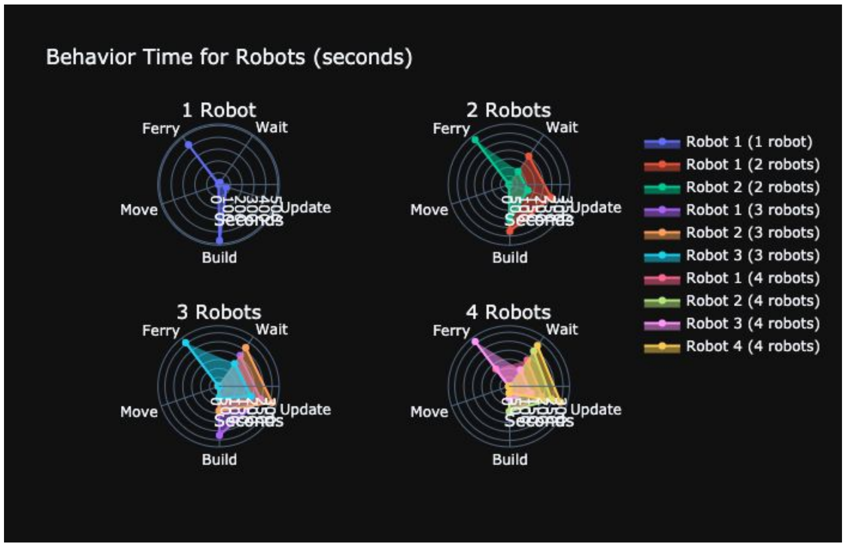


Figure 63: Graph comparing the time spent performing behaviors for structures built with varying numbers of robots

Conclusion

A robot (swarm) was developed that has a high degree of mobility and functionality. The robot was shown achieving the various configurations we desired, as well as demonstrated the capabilities of the robot in interacting with the building blocks. Smart blocks were developed that allowed a sense of virtual stigmergy wherein information could be left in the blocks and communicated to other entities, either other blocks or robots. The smart blocks could also detect when a block was removed from a connected element of a structure and determine their location within a connected element of a structure. Finally, a novel building algorithm was developed that allows multiple robots to work collaboratively. A proof was created to prove the correctness of the algorithm, and through simulation, the benefits of using the algorithm were shown, especially when considering how the algorithm highlights cooperation between robots to expedite construction. The results confirm that all the system level requirements were met with the exception of the ability to build a 4x4x4 cube. Due to time constraints and unforeseen circumstances (COVID-19 pandemic of 2020), the physical robot was not able to be thoroughly tested. Therefore, it has not been proven that it meets the requirement of constructing a 4x4x4 cube in real life. Despite this, the simulation has shown that this type of construction is possible with the system.

The final products of this project include but are not limited to: the inchworm robot, the smart blocks, the building algorithm, the simulator, the dashboard, the blueprint creator, and the GUI controller for the robot.

Lessons Learned

A major lesson learned includes the importance of multiple iterations when designing. While this did occasionally slow down production towards the final product, it allowed all of the subteams to fail fast and early on. Having intermediate physical robots to interact with helped reduce errors, particularly for the electrical and software subteams. Having physical robots at early stages in the project allowed for teams to rely less on simulation as well as detect problems early on such as wiring concerns and sensor problems. Furthermore, this greatly helped when during the Covid-19 pandemic of 2020, the team quarantined at home and were not allowed on campus. When creating the final robot design, there were fewer errors as most had already been discovered and fixed with previous iterations.

Future Work

Mechanical

One possible area for future work is the development of a selective engagement block design. The attachment system presented in this work utilizes a gendered mating connection which imposes constraints on the order in which structures must be assembled and disassembled. The design of genderless mating connections that enable strong interface connections on all six sides of construction blocks would greatly improve the flexibility of the construction system and the strength of resulting structures.

Another area for future work includes repeatability and lifecycle testing of the robot design. This might include a series of trials to show that the inchworm will be capable of building structures autonomously, with little or no human intervention.

Electrical

Future work of the smart blocks involves designing custom NFC antennas that are compatible with the PN532 NFC controller. Additionally, the antenna multiplexing circuit discussed in the methodology would be implemented using these antennas. This would enable the smart blocks to be manufactured at a much lower cost. This antenna design could also be used to integrate NFC into the end effector of the inchworm robot.

Future work of the inchworm robot may involve adding sensors to monitor the internal components of the robot. For example, the operation time of the robot will be affected by its battery capacity, weight, motor efficiency, and the power consumption of its electrical components. Adding a sensor that monitors the battery capacity will allow the robot to exit the structure before its battery loses charge, preventing the robot from getting stuck on the structure. When adding components to the robot links it is important to take into consideration the form factor since the available space in each link is limited. Space could also be better allocated if a custom PCB is made to condense the electromechanical components within the links.

Software

Future work of the building algorithm involves expanding the algorithm to factor in other facets of the system such as the energy expenditure of the robots. In analyzing the behaviors of the robots when building, it was clear that some robots work harder than others. While infinite power was used for this project, it might be beneficial to factor in energy expenditure so that the workload is more evenly distributed amongst all of the robots, thus helping to extend the use of their batteries.

Another area of future work for the building algorithm involves adding a module that can calculate support structures for overhangs and disconnected structures. For example, a temporary bridge could be created between two disconnected towers so that the robots could traverse from one tower to the next. This temporary bridge could then be removed at the end of the algorithm.

Works Cited

1. 68% of the world population projected to live in urban areas by 2050, says UN. UN DESA | United Nations Department of Economic and Social Affairs Web site. https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html. Updated 2018. Accessed Oct 10, 2019.

2. Bureau of labor statistics. https://data.bls.gov/PDQWeb/jt Web site. https://data.bls.gov/PDQWeb/jt. Updated 2019. Accessed Oct 12, 2019.

3. UN habitat: Housing. UN Habitat Web site. https://new.unhabitat.org/topic/housing. Accessed 10/12/, 2019.

4. Petersen KH, Napp N, Stuart-Smith R, Rus D, Kovac M. A review of collective robotic construction. *Science Robotics*. 2019;4(28):eaau8479.

5. Jenett B, Cheung K. Bill-e: Robotic platform for locomotion and manipulation of lightweight space structures. . 2017:1876.

6. Allwright M, Bhalla N, El-faham H, Antoun A, Pinciroli C, Dorigo M. SRoCS: Leveraging stigmergy on a multi-robot construction platform for unknown environments. . 2014:158-169.

7. Petersen KH, Nagpal R, Werfel JK. Termes: An autonomous robotic system for three-dimensional collective construction. *Robotics: science and systems VII*. 2011.

8. Dorigo M, Bonabeau E, Theraulaz G. Ant algorithms and stigmergy. *Future*

Generation Comput Syst. 2000;16(8):851-871.

9. Disney’S VertiGo combines car, helicopter to drive up walls. IEEE Spectrum: Technology, Engineering, and Science News Web site. https://spectrum.ieee.org/automaton/robotics/drones/disney-vertigo-combines-car-helicopter-to-drive-up-walls. Updated 2015. Accessed Oct 12, 2019.

10. Stuart‐Smith R. Behavioural production: Autonomous Swarm‐Constructed architecture. *Architectural Design*. 2016;86(2):54-59.

11. Volpe R. The LEMUR robots. https://www-robotics.jpl.nasa.gov/systems/system.cfm?System=5. Updated 2019.

12. Petersen KH, Nagpal R, Werfel JK. Termes: An autonomous robotic system for three-dimensional collective construction. *Robotics: science and systems VII*. 2011.

13. Jenett B, Cheung K. Bill-e: Robotic platform for locomotion and manipulation of lightweight space structures. . 2017:1876.

14. Liedke J, Wörn H. CoBoLD—A bonding mechanism for modular self-reconfigurable mobile robots. . 2011:2025-2030.

15. Budynas RG, Nisbett KJ, Shigley JE. *Shigley's mechanical engineering design.* 8th ed. New York: McGraw-Hill; 2011.

16. PN532/C1 datasheet. www.nxp.com Web site. https://www.nxp.com/docs/en/nxp/data-sheets/PN532_C1.pdf.

17. Atmel Corporation. ATmega328P datasheet. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf. Updated 2015.

18. AMS AG. AS5048B Magnetic rotary encoder datasheet. https://www.google.com/url?q=https://www.mouser.com/datasheet/2/588/AS5048_DS000298_3-00-522570.pdf&sa=D&ust=1589585892398000&usg=AFQjCNEXf_RPGmudoQO9TD83stvplB2LDQ. Updated 2016.

19. Pololu Corporation. TB9051FTG single brushed DC motor driver carrier datasheet. https://www.pololu.com/product/2997.

Appendix A: Design Considerations

This appendix details the considerations and analysis that went into the designs. It includes the evolution of intermediate designs so that readers may understand how the final design was achieved. Changes made from different iterations of the robot, smart blocks, and building algorithm will all be listed.

Mechanical

Design of First Iteration Robot

It was important to design and manufacture a first iteration inchworm prototype that could be used as a simplified hardware platform to test the mechanical and electronic subsystems, and validate the simulation kinematics. It provides a proof of concept of a mobile inchworm that is able to traverse over a structure and place blocks. The first iteration inchworm will enable quicker development and validation of system level aspects necessary to complete the goals of this project while in parallel a fully functional inchworm can be properly designed and manufactured.

The primary design requirements for this robot to support the goals discussed above is

1. The robot should have 3 degrees of freedom to support development of inching motion in one plane
2. The robot should be modular so that mechanical and electrical subsystems can be easily retrofitted and tested on the hardware
3. The robot should be able to navigate a three dimensional structure in one plane of motion
4. The robot will have joint motors that will allow it to support itself and hold one block

The first iteration design that fulfils these requirements is shown in Figure 64.

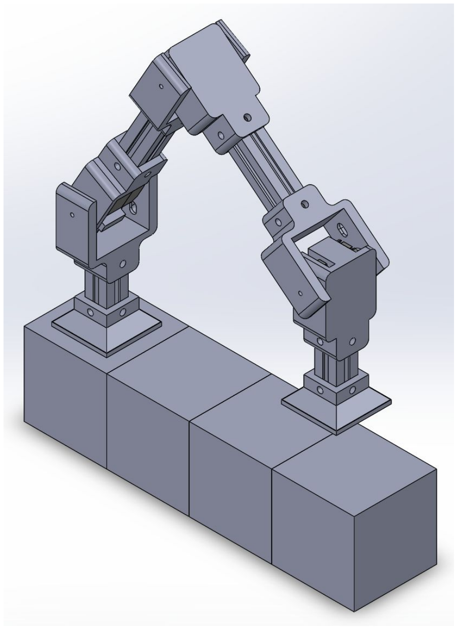


Figure 64*: Computer-Aided Design (CAD) model of first iteration robot prototype*

Servo motors with a 25kg-cm stall torque rating were chosen to be the joint actuator because of the ease of use and form factor. The links of the robot were chosen to be plastic 20mm by 20mm profile 80/20 extrusions so that microcontrollers and other electronics could be easily attached. The joints were designed to be modular so that the robot links could be easily removed. The inchworm was 3D printed out of PLA material and the manufactured prototype is shown in Figure 65.

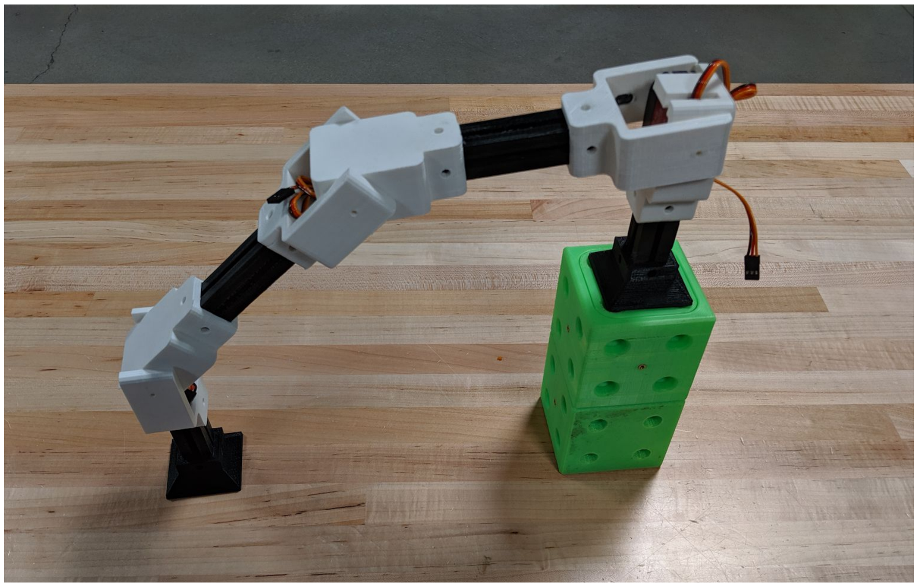
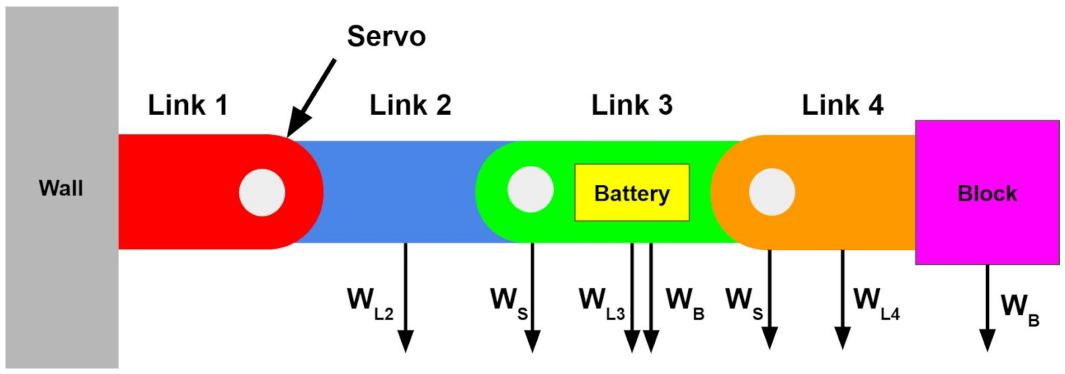


Figure 65*: First iteration robot*

Static Joint Torque Analysis

A torque analysis of the inchworm robot was conducted to estimate the required motor torque needed to operate the robot. The robot model was simplified into a simple 4 link arm carrying a block, as well as servo motors and a battery. The weight of the links were estimated by simplifying them to be plastic PLA cylindrical linkages with a density of 0.8 g/cm^3 and the lengths determined from the linkage length analysis. The mass of link A&D, link B&C, battery and a servo were estimated to be 110g, 70g,100 g and 70g respectively. Though the robot can operate in configurations that can minimize the applied torque on the robot, the analysis was conducted in the configuration shown in Figure 66. The robot is extended fully and the full weight of each component acts perpendicular to the lever arm. The torque applied at each joint if the motors were stalled can be calculated using equation X and the stall motor torque was found to be 18.9 kg-cm for the outer motors of the robot.



$$\sum_{i=0}^{n} \tau = m_1 x_1 + m_2 x_2 + ... + m_n x_n$$

Figure 66*: Free body diagram of worst case motor torque scenario*

Design of Second Iteration Robot

Through a basic tolerance analysis, it was discovered that the resolution of the joint servos in a worst case scenario was insufficient to successfully align the end effector with the hole pattern for mating with the structure. In order to improve the resolution, the stock servo circuit was removed, and an external magnetic encoder was attached. The magnetic encoder has an improved resolution compared to the standard servos, and PID control was implemented. As a result of this change, the physical implementation of the robot had to be modified for the addition of the encoder. During initial testing it was found that accessibility to the servo motors was limited, so the joints were divided into multiple parts to support DFMA (Design for Manufacture and Assembly). The need to achieve greater accuracy, as well as the inaccessibility of critical components prompted the creation of a second robot iteration.

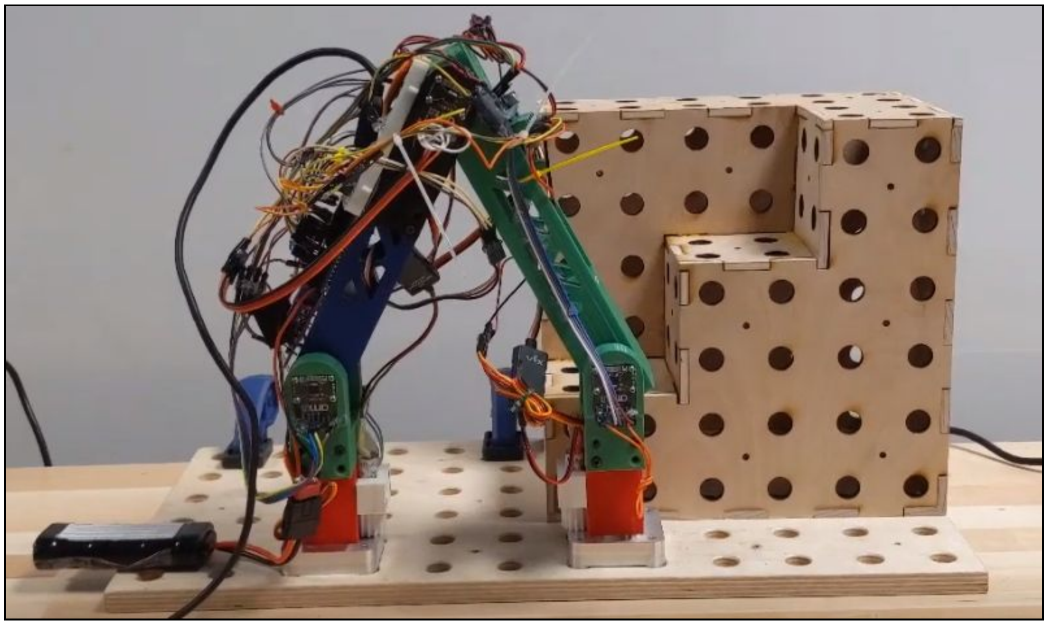


Figure 67*: Second iteration robot traversing through playground*

Block Storage Mechanism

Looking towards the future, a swarm of inchworm robots could have the potential to build structures much larger than itself. Based on the building algorithm or block starting location, it may be necessary for robots to traverse large sections of the structure before placing a block. With this consideration in mind, a block storage mechanism was designed and fabricated to enable the robot to maintain control of a block while traversing the structure. The three functional requirements for this system are listed below:

(1) Maintain control of block while traversing a convex corner
(2) Allow the robot to insert blocks into holes in the structure
(3) Allow the robot to place blocks diagonally

In addition to these functional requirements, the optimization criteria used to select the final design was to use the fewest number of motors, and the primary constraint was that the robot's operation should be independent of the block storage system (ie. the block storage system can be removed without reducing the ability of the robot to traverse the structure or move blocks). The resulting design is a four bar linkage that attaches to the B and C links. This four bar linkage has a compliant, passive end effector attached to the coupler which is capable of maintaining control of the block regardless of robot orientation.

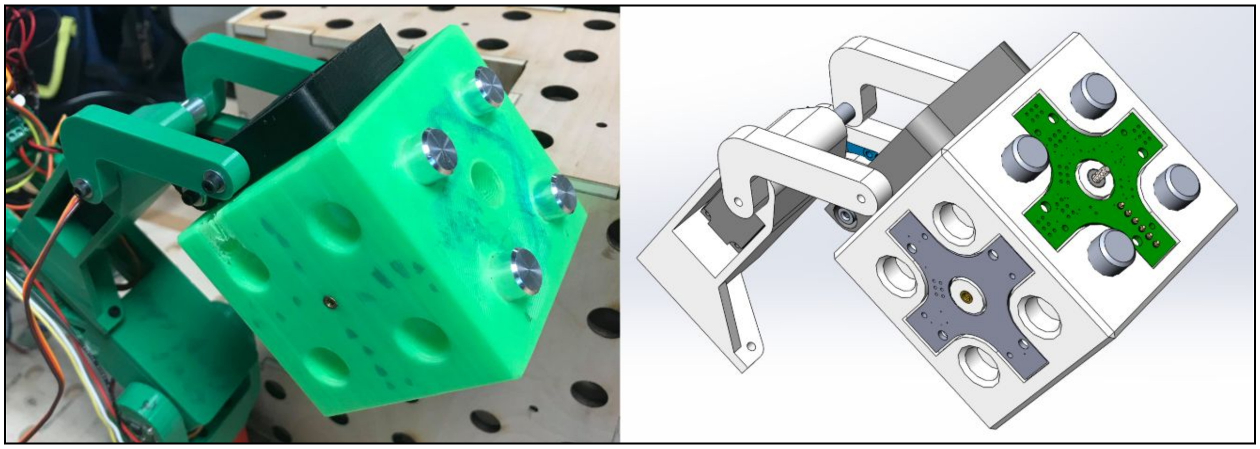


Figure 68*: Block Placement Mechanism Prototype and CAD Model*

In its default state, the four bar linkage keeps the stored block flat against the B and C links to minimize the overall volume occupied by the robot. Upon actuation, the four bar linkage moves the block to a position within the range of motion of the A and D links of the robot. This allows the robot's standard end effectors to control and place blocks that are stored using the four bar linkage, without requiring the end effectors to have any additional functionality. An image of the CAD model and block placement mechanism is shown below.

The compliant aspect of the passive gripper was designed to handle the weight of the block, regardless of orientation in space. Each of the four dowel pins is divided into four arcs. These arcs are tapered outwards in order to contact the block at the bottom edge of each hole. The dowel pins are printed oversized, such that each arc must deflect inwards to allow the dowel pin to enter the hole. The reaction force from each arc of the dowel pins results in a normal force, and therefore a friction force, between the inside surface of the block and the dowel pins. This friction force enables the robot to maintain control of the block when in the storage position. A picture of the compliant gripping element is shown below in Figure 69.

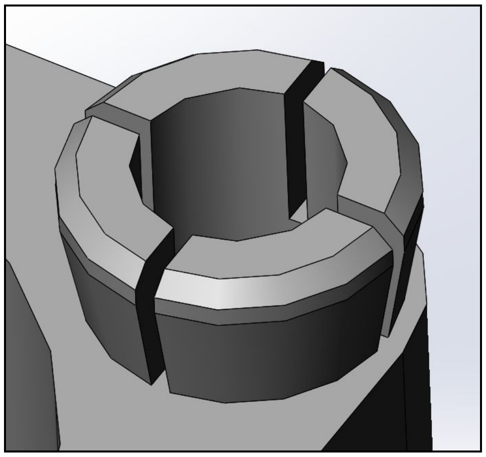


Figure 69*: CAD Model of 3D Printed Compliant Dowel Pins*

Gravity Compensation

To enable smooth joint control of the robot, it was necessary to compensate for the torques applied to the joints due to the weight of the robot. The torques on each joint could be estimated by using the gravity matrix equations from the inchworm dynamical model and inputting the current angle configuration of the robot. The equations are presented in Figure 70. The PID + Gravity compensation controller is one possible solution that has been used to control the inchworm robot. This controller was implemented on the embedded controller, and is further discussed in the Electrical Design section.

$$T_1 = g * m3 * (L1 * \sin(th1) + L2 * \sin(th1 + th2) + LCoM3 * \sin(th1 + th2 + th3)) + g * m2 * (L1 * \sin(th1) + LCoM2 * \sin(th1 + th2)) + g * LCoM1 * m1 * \sin(th1) + g * mblock * (L1 * \sin(th1) + Lblock * \sin(th1 + th2))$$

$$T_2 = g * m3 * (L2 * \sin(th1 + th2) + LCoM3 * \sin(th1 + th2 + th3)) + g * LCoM2 * m2 * \sin(th2 + th1) + g * mblock * Lblock * \sin(th2 + th1)$$

$$T_3 = g * m3 * LCoM3 * \sin(th3 + th2 + th1)$$

Figure 70*: Equations to calculate torque induced by gravity on joints 1,2 and 3*

Electrical

Additional Smart Structure Analysis and Design

Analysis of Communication Interfaces for Smart Blocks

To fulfill the requirement for block-to-block communication and block-to-robot communication, a communication protocol must be established (note that block in this context is used interchangeably with smart block). This protocol includes both the electrical standard used for communication as well as the structure of the data transmitted during the communication. This section will discuss the various electrical interfaces considered for the smart block communication protocols.

The first consideration made was a similar implementation of the AprilTag system, used by the SRoCS platform, that would allow the current system to determine the direction the block is placed on the structure. Additionally, wireless communication like NFC would allow for the flexible positioning of the blocks and still enable the communication block-to-block and block-to-end effector. The block-to-block communication allows the structure to communicate its current configuration to the system. It also enables the individual blocks to communicate their relative position in the structure to the inchworm robot.

The first synchronous communication interface considered, used flip-flops for counting each block placed on the structure on all three axes. As seen in Figure 71, the advantages of this design were the simplicity and minimal space used inside the blocks. Despite this design being simple, it did not enable the blocks to report their relative position within the structure. Researching the cost of the individual components revealed that the cost of this circuit would exceed the price of a simple Alf and Vegard's RISC (AVR) processor. Due to this circuit's price and limited capabilities, it was not chosen for the final smart block implementation.

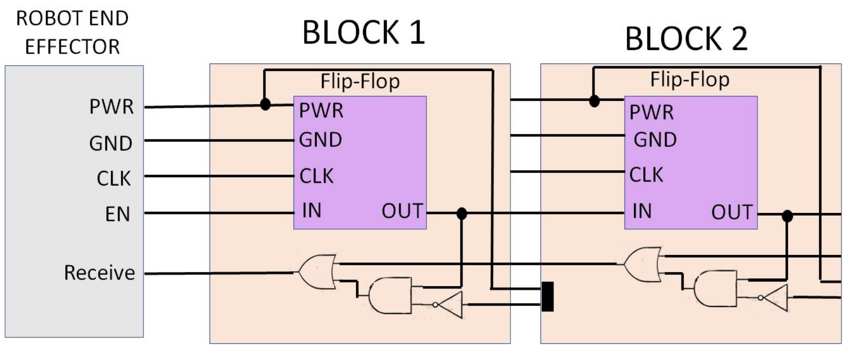


Figure 71: *Flip-Flop Communication Interface Diagram*

Serial Peripheral Interface (SPI) was the second synchronous communication interface considered as it provides high signal integrity, simplicity, and high-speed data transfer, allowing for real-time communication. The physical limitation of SPI is that it requires more pins on the Integrated Circuit (IC) than other communication interfaces. Furthermore, SPI does not support dynamically adding slave nodes; therefore, removing blocks and adding them would interrupt hierarchy.[13]

Inter-Integrated Circuit (I2C) was the third synchronous communication interface considered as it only uses two bidirectional lines, Serial Data Line (SDA) and a Serial Clock Line (SCL). For I2C to work each block would have to be dynamically allocated an address at the moment of placement which would allow it to communicate through the SDA line when selected. As with SPI, I2C also runs into the problem of adding and removing blocks as it interrupts hierarchy. Additionally, I2C may produce addressing errors at the instance of the removing of blocks.[13]

Universal Asynchronous Receiver-Transmitter (UART) was the fourth communication interface considered as it allows for robust data transfer. Unlike the previous communication interfaces, UART does not require a master-slave setup. This means that data transfer uses a complex method (simplex, full or half duplex) for handling serial communication with data framing. UART would enable for the dynamic addition and removal of blocks as it does not need a shared clock as the previously mentioned communication methods. The limitations with UART is that it is a peer-to-peer (P2P) system which does not support multiple blocks. The transceiver and receiver physical connection crossover is also only possible in 2D since when introduced to three-dimensional it did not accomplish the genderless connection desired.

One interface that was strongly considered for use in the smart blocks is NFC. NFC is a low power, wireless communication protocol commonly used in smart phones and wireless key-cards. NFC eliminates many of the downfalls of the previously mentioned electrical protocols. Most prevalently, NFC eliminates the need for symmetrical data connections on the faces of the smart blocks and it does not require the smart blocks to share a clock signal. NFC controllers also offer various modes of communication including peer-to-peer, card emulation, and card reader modes. [14-16] In a paper published by Texas Instruments, it was shown that multiple antennas can be connected to one NFC controller through an radio frequency (rf) multiplexer. Multiplexing multiple antennas allows the NFC controller to poll the antenna.

Another electrical interface evaluated for use in the smart blocks is the Controller Area Network (CAN) bus. CAN is similar to I2C in the fact that many devices can be connected to the same bus and communicate using unique addresses. The main appeal of CAN bus for this system is its use of differential pairs for communication. According to some sources, it may be possible to convert CAN to a one wire interface using diodes and termination resistors. This configuration sacrifices the range and stability of the CAN protocol. Using a one wire interface would eliminate the need for any data line crossover. A one wire CAN interface would be implemented by using one can controller IC per block face.[17,18]

First Iteration Smart Block and Home Block Electrical Design

For the first iteration of the smart block design, two types of smart blocks are required to build a structure. The first type of block is the regular smart blocks, which are the main structural elements used while building. These blocks have two major electrical requirements; they must determine and store their position within the structure and they must transmit their position within the structure to the home block when they are inserted into the structure. The second type of block is the home block. The smart block is in charge of powering all the other smart blocks, storing the positions of all the smart blocks within the structure, and transmitting the structure's configuration to the robots. In order to communicate the structure's status with the home block, the smart blocks within the structure must relay messages from newly attached blocks to the home block. Each smart block will contain an Atmel AVR processor responsible for determining and storing the position of the block within the structure. Five electrical connections electrical connections link each block to the next; one for power, one for ground, and three for communicating between blocks. Two of the communication pins are for UART connection between blocks and the other communication pin is for determining the direction of a newly placed block.

The concept of genderless electrical connections and communication between the smart blocks is reliant, in part, on the smart blocks possessing the mechanical abiliity to connect in any direction. For the first mechanical iteration of the smart block design, the smart block has one male block face and five female block faces. To match the physical capabilities of the first block iteration, the electrical communication system on the smart block will also contain one male face and five female faces. In order to mechanically attach the first-iteration block to another block, the male side of the block must be inserted to another block. As a result, the male side of the smart block will always be connected to another smart block. Because the male face of each block in the structure will always be attached to another block, the male face of each block initiates any communication with another block. When a block is inserted into the structure, the male face communicates with the female face of the block it is attached to. Using UART and the addition I/O pin the female block face communicates the newly-placed block position to the block. Then, the position of the newly-placed block is transmitted block to block using the male face as the communication initiator. Following the male faces, the data containing the position of the newly-placed block eventually reaches the home block. The home block can then store the position of the newly-placed block and update the model of the structure.

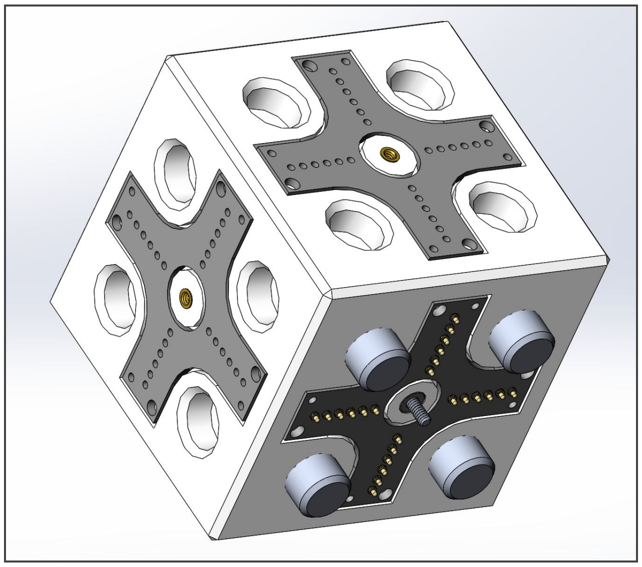


Figure 72*: “Basic” block design with UART pins*

Additional Inchworm Robot Electromechanical Components and Design

Inchworm Custom PCB Version 1

Inchworm Board V1 board, is a 80mm x 37.250 mm x 10mm custom PCB board for the first version of the inchworm robot. This custom PCB allows to cut down the wire usage from the prototype inchworm and smaller design.

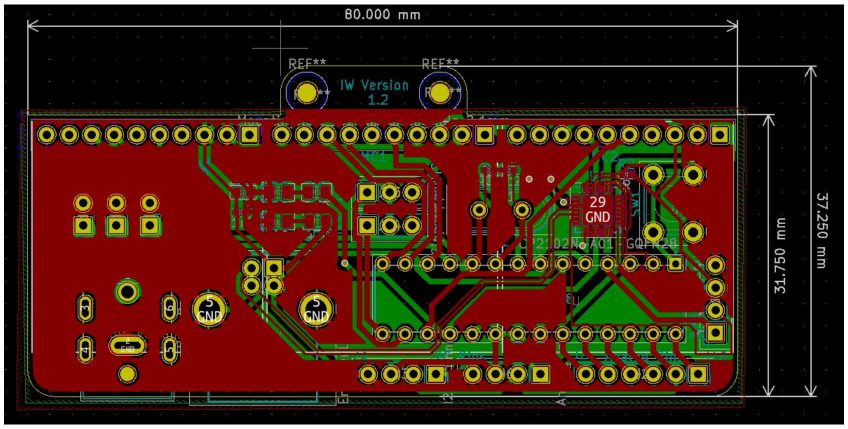


Figure 73*: Kicad custom PCB via view with board dimensions*

The first version of the IW board uses the AVR ATmega 328 with a 16MHz crystal to run the various electrical components. The AVR processor has a total of 6 PWM outputs which allows 3 to be used for the joint motors and 2 for the gripper motors. This AVR processor also includes an I2C bus which connects to the 3 joint encoders of the robot. Power on the board is divided in two: the right portion of the board is powered through the USB 5V connection which powers the AVR processor and all components that use a 5V input, the left side of the board supplies power to the 3 joint BECs through a barrel jack connector. The barrel jack connector can handle a 12 V wall adapter power supply with an amperage of around 10 A. The board has female header pins for access to the analog input pins on the AVR processor.

To keep the size of the board within the maximum size constraints the board allows for the 3 pololu motor controllers to be plugged on the bottom layer of the board as if they were "sandwiched." This allows for more wires to be eliminated as it acts as a shield of the motor controllers to be directly plugged into the board.

For debugging purposes the board counts with a set of green LEDs. The "Power LED" turns on when the board is powered and the "Test LED" is used to light up when pin 12 is set with the blynk arduino sketch. The reset button allows the board AVR and USB to UART Bridge to be reset. For this iteration it is important to mention that only one USB to UART SDM bridge was ordered, the team was unable to get it working as in the processes of soldering this small component there might have been a problem. But this problem was fixed by using an external USB to UART Breakout Board (FTDI Basic Breakout Board). For the second version of the IW board this will be taken into account and the second iteration will include extra connection pins for external Serial Bridges.

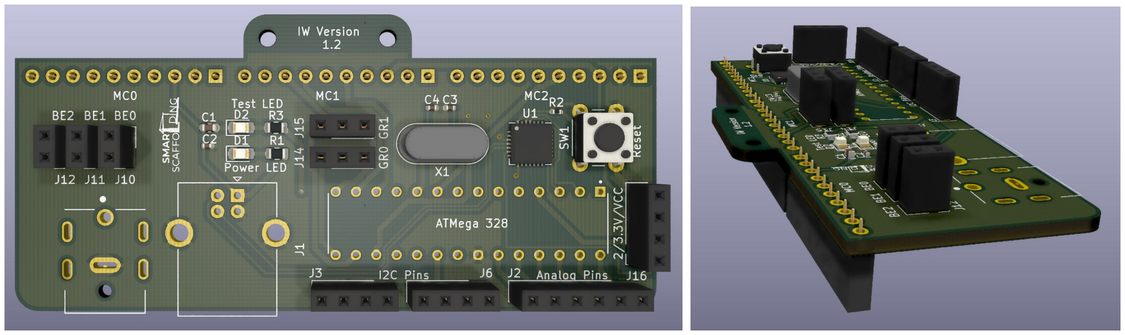


Figure 74*: Kicad PCB 3D viewer*

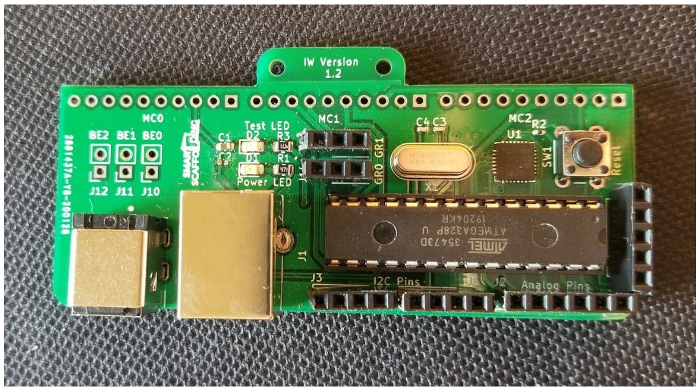


Figure 75*: Inchworm Custom PCB Version 1*

IR End Effector Alignment Electrical Design Concept

Alignment between the end effector and smart block is crucial as previously mentioned in the mechanical sections: “End effector design” and “Basic Block Design.” The mechanical solution mentioned in the previous mechanical section allows for mating despite lateral misalignments. On the electrical side another explored solution for this alignment was using infrared (IR) LEDs on the smart blocks and robot end effectors. It is important to note that the implementation of IR LEDs for alignment will only be used if it is not accurate enough in the next iterations.

The concept of using infrared LEDs as a method of alignment relies on the property that LEDs can use current to produce light and can also act as a photodiode and use light energy to create an electrical current. To utilize this property in an alignment system, each set of four alignment LED needs a corresponding emitter LED. The basic concept of this alignment circuit is that each block face will have a series of LEDs emitting light. The robot’s end effector will have a series of matching sets of LEDs acting as photodiodes. When the emitter LED is centered within the four receiving LEDs, the end effector’s LEDs will reach their maximum voltage output. As the alignment LEDs move out of alignment the output voltage decreases. The comparing the voltages of the four receiving LEDs in each set will determine which direction the end effector needs to be moved in order to be aligned with the alignment LED. To minimize interference from other infrared light sources, the emitter LEDs will emit a square wave at a frequency of 200kHz. Additionally, the receiver LEDs will use a series of band pass and low pass filters to turn the 200kHz square wave into a smooth DC voltage. The receiving LED circuit can be seen in Figure 76.

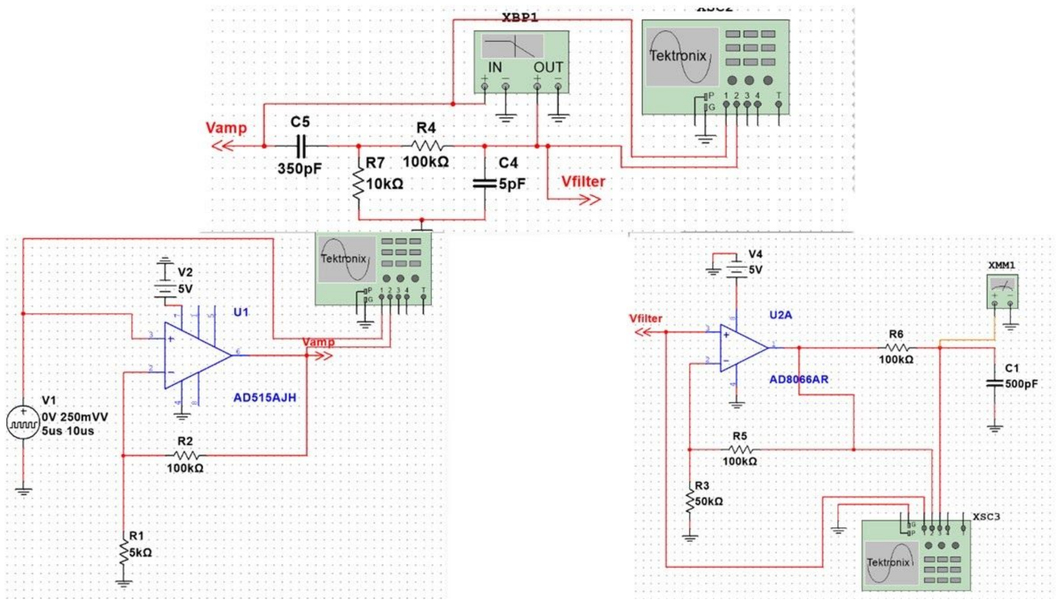


Figure 76*: Infrared alignment receiver LED circuit.*

A test rig was made to test this system as seen in Figure 77 below. Four IR LEDs would be placed on the smart block and eight IR LEDs would be placed on the end effector as seen in Figure 77. This, in theory, would allow for a precise alignment of the end effector with the smart block. To determine if the end effector was aligned with the smart block the eight IR LEDs on the end effector would have to read around the same voltage value. If one of them or multiple read different voltage values they would signal the end effector to re-adjust accordingly. Furthermore, using IR LEDs on the smart blocks would also allow the blocks to communicate between each other. A protocol would have to be established and tested to determine if this feature could be added to the smart blocks, robot, and the system.

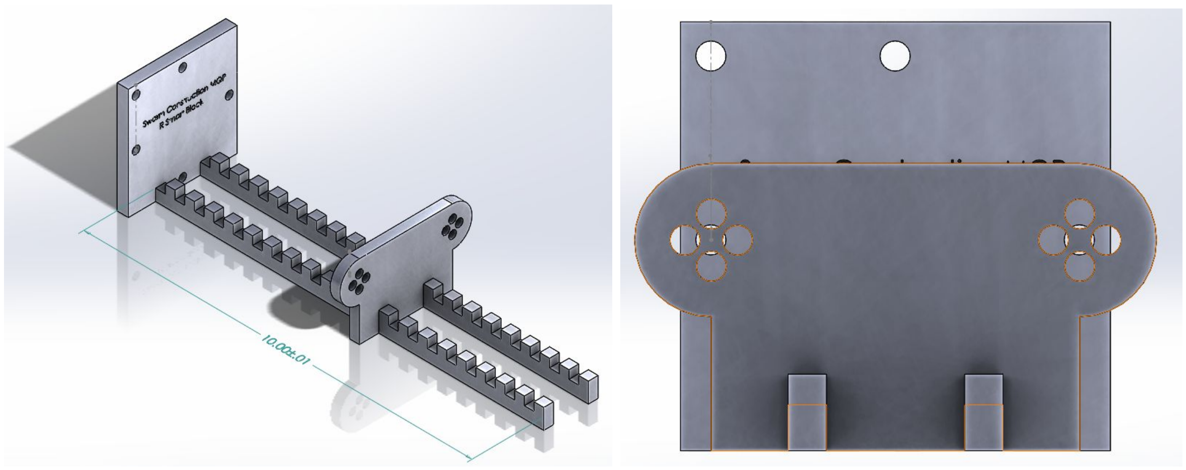


Figure 77*: Right) Angled view of the IR alignment testing rig. Left) Front view of the IR rig with the end effector aligned with Smart Block (end effector perspective).*

Additional Low-Level PID Controller with Gravity Compensation

The PID controller with gravity compensation shown below was implemented in the first iteration of the inchworm robot. It proved to not be needed as a simple PID loop allowed for proper control of the robot without the need of the extra processing needed for the gravity compensation. Additionally, disabling the gravity compensation allowed for the controller to run faster.

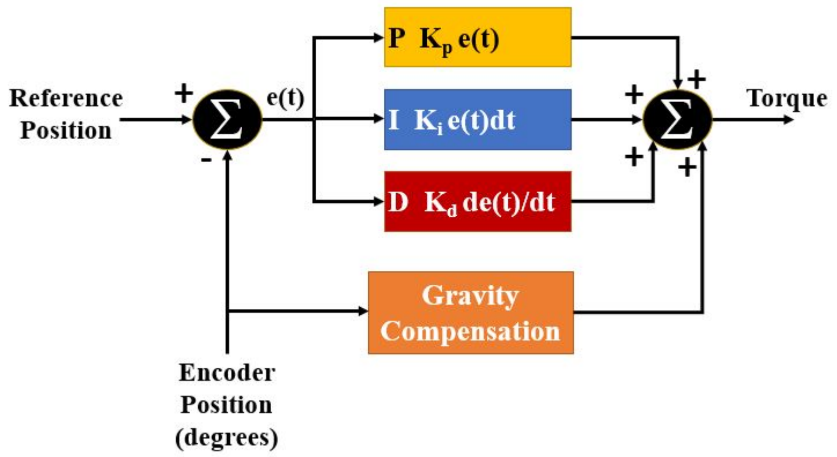


Figure 78*:* PID with Gravity Compensation controller diagram

Software

Additional Control Algorithms Developed

To simplify the control of the robot, the inchworm design can be considered a standard three degree of freedom robot arm. While the robot is a dual end effector design, only one end effector will need to move at a time. As such, the end effector that is not moving can be considered a fixed base. This allowed the use of a numerical inverse kinematics algorithm for moving the inchworm to certain positions in three-dimensional space:

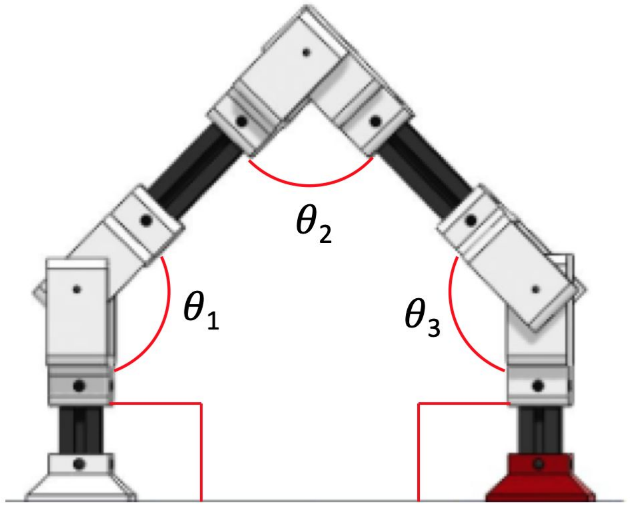
```
1. Initialize $\boldsymbol{q}_0$
2. $\boldsymbol{q}_i \leftarrow \boldsymbol{q}_0$

3. while $\|\boldsymbol{p}_d - f(\boldsymbol{q}_i)\| > \varepsilon, \varepsilon \in \mathbb{R}$
4.    $\Delta\boldsymbol{q} = \boldsymbol{J}^{-1}(\boldsymbol{q}_i)(\boldsymbol{p}_d - f(\boldsymbol{q}_i))$
5.    $\boldsymbol{q}_i = \boldsymbol{q}_i + \Delta\boldsymbol{q}$
```

While this algorithm allowed for the robot to move to any configuration within its workspace, constraints needed to be imposed so that the robot's gripper can insert correctly into the building materials. When inserting, the end effector must insert completely perpendicular to the surface of the building material, otherwise it will not attach correctly. The same is true for placing and removing blocks; the final motion of the robot must be perpendicular and a linear movement so as not to hit other blocks.

For the first iteration of the robot design, a lookup table was created to impose these constraints. For different orientations of the robot end effector, the angle of the end effector can be determined by consulting the lookup table. The values of the lookup table were derived based on placing the robot in different configurations and manually determining the optimal angle.

The second iteration of the motion planning algorithm involved using the natural geometry of the robot to derive the final angle. When both end effectors are coplanar, as shown below, the robot creates a pentagon. Both end effectors create a 90º angle with the surface below. As the angles of a trapezoid sum to 540º and theta one and two are solved for using numerical inverse kinematics, the final angle theta three can be found by subtracting these angles from 540º. For changing this motion to accommodate other planes, the same motion can be found by solving for the angles within the pentagon and then offsetting the last angle theta three by +90º, -90º, or +180º. This method allows for immediate and precise calculation of the last angle while at the same time accommodating the perpendicular constraint needed by this type of robot.



$$\theta_3 = 540° - (90° + 90° + \theta_1 + \theta_2)$$

Figure 79: *Method used to solve for the final angle of the robot*

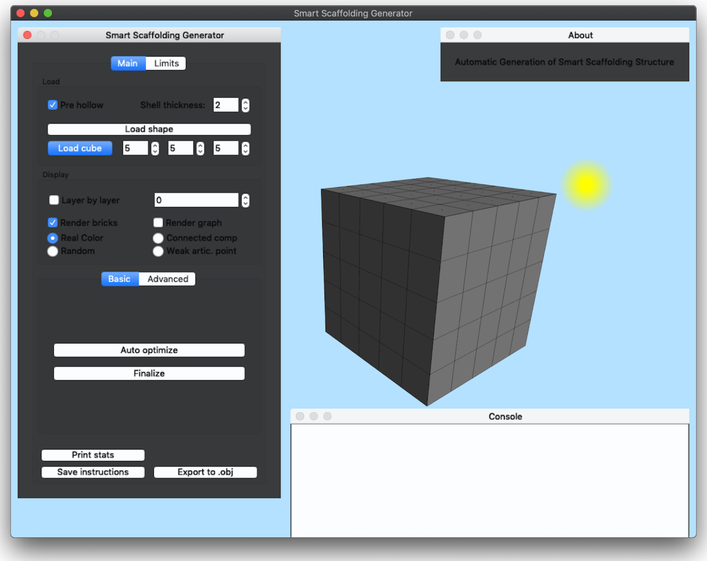
In order to support inching movements in different planes and improve the perforcement of the inverse kinematics calculation, a third iteration of the motion planning was implemented, as described in the Control Algorithms section.

Additional Techniques of Robot Navigation

The first iteration navigation algorithm was an implementation of the A* algorithm that was scaled to three dimensions. Each block is considered as a node in a graph, which can then be traversed as in a typical A* algorithm. With this, the robot can determine which blocks it needs to traverse in order to reach its desired location. The data for the position of each block comes from the blocks themselves. The blocks communicate their locations in three-dimensional space to the robot, which is then able to use this information to create the necessary graph needed for path planning.

Other Techniques of Blueprint Creation

In order to generate a blueprint, or template for building the structure, a blueprint is first created by a human operator. Software was developed that allows an STL file to be uploaded containing the desired structure. The software then slices the STL file into rectangles according to the width and height of the building blocks. Effectively, this allows any building configuration to be built with the slicer tool and smart blocks by discretizing the structure. This discretization means the robot can consider each block as a point in three-dimensional space. The slicer tool writes the id, position, and orientation of all blocks to a file that is then uploaded to the home block of the structure as the structure blueprint. The slicer also creates a level by level view of the structure for manual inspection by a human.



[✓] Pre hollow Shell thickness: 2
Load cube 5 5 5
[ ] Layer by layer 0
[✓] Render bricks [ ] Render graph
[● Real Color] Connected comp
Random Weak artic. point

Figure 80: *Structure discretizer, shown here discretizing a cube into smart blocks*

Additional Decentralized Division Claiming

For the final iteration, in order to have the robots determine which divisions they should claim, it is expected the bully algorithm will be used. This algorithm is a leader election algorithm in which a robot with the highest “robot id” in a group of robots is elected as a “leader.” The leader will claim a section and set that information to the other “followers,” who will acknowledge that the leader has claimed that section as well as that they are no longer allowed to claim that section. This process will repeat where the next highest id will then be elected leader and claim a section, until a consensus within the network has been reached. The id the robots will send to compare with will be their distance from the centroid of each division. Each robot will calculate its distance to all the centroids and select the minimum distance. Of all the minimum distances, the robot with the largest (worst) distance will be able to pick first, in order to minimize the displacement of all robots.

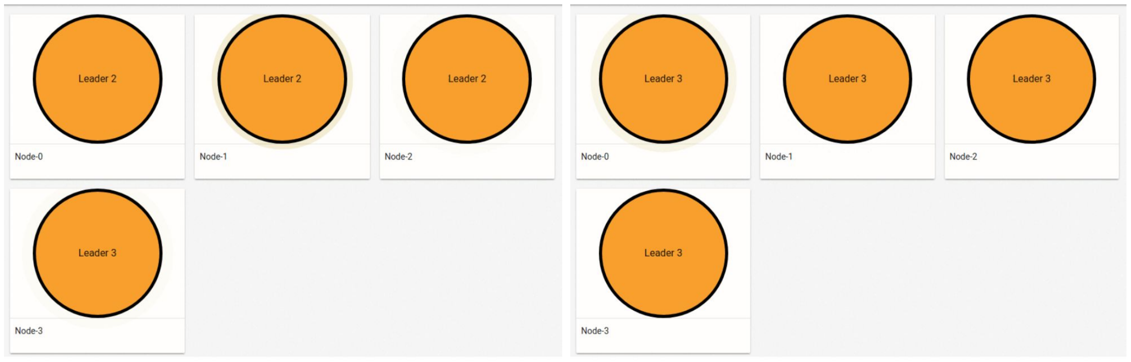
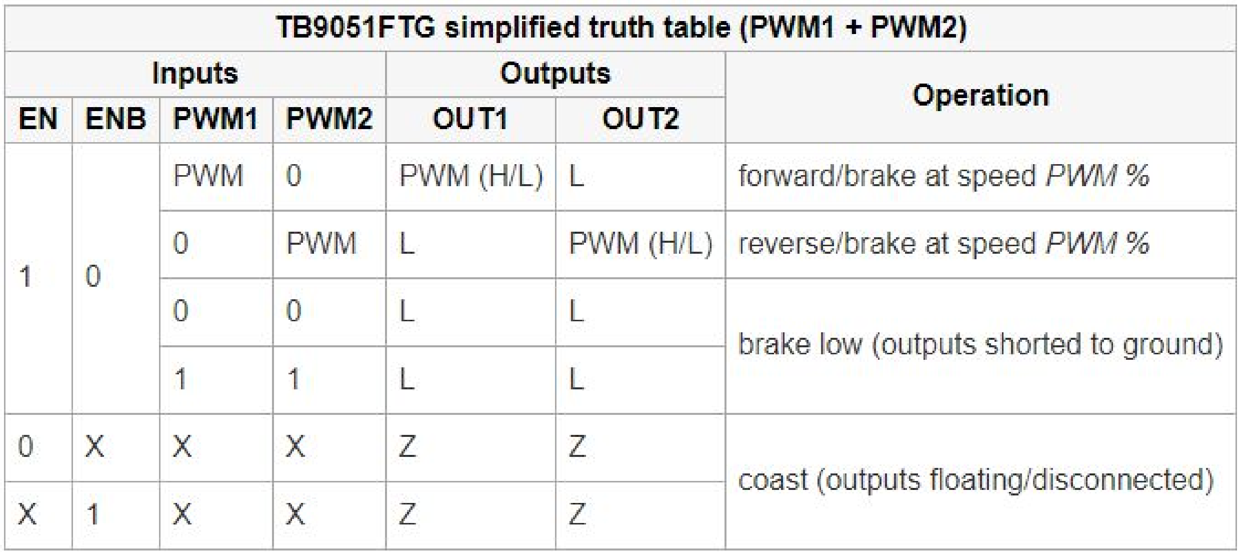


Figure 81: *Demonstration of bully algorithm. When Node 3 is added and is the highest node, the other nodes elect the highest node id, Node 3, as the new leader.*

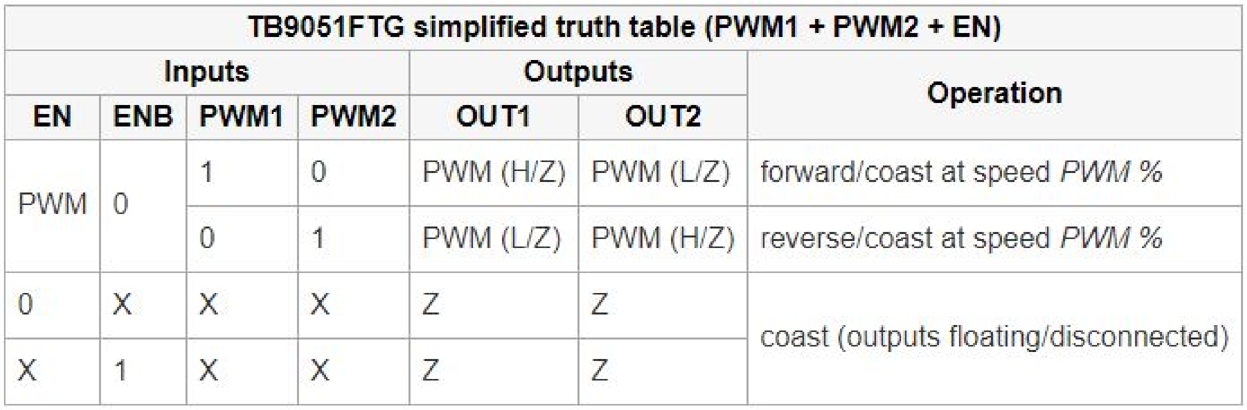
Appendix B: TB9051FTG Truth Tables



TB9051FTG simplified truth table (PWM1 + PWM2)

Inputs				Outputs		Operation
EN	ENB	PWM1	PWM2	OUT1	OUT2	
1	0	PWM	0	PWM (H/L)	L	forward/brake at speed *PWM %*
		0	PWM	L	PWM (H/L)	reverse/brake at speed *PWM %*
		0	0	L	L	brake low (outputs shorted to ground)
		1	1	L	L	
0	X	X	X	Z	Z	coast (outputs floating/disconnected)
X	1	X	X	Z	Z	

Table 5: *TB9051FTG Control Pins Configuration 1*



TB9051FTG simplified truth table (PWM1 + PWM2 + EN)						
Inputs				**Outputs**		**Operation**
EN	**ENB**	**PWM1**	**PWM2**	**OUT1**	**OUT2**	
PWM	0	1	0	PWM (H/Z)	PWM (L/Z)	forward/coast at speed *PWM %*
		0	1	PWM (L/Z)	PWM (H/Z)	reverse/coast at speed *PWM %*
0	X	X	X	Z	Z	coast (outputs floating/disconnected)
X	1	X	X	Z	Z	

Table 6: *TB9051FTG Control Pins Configuration 2*