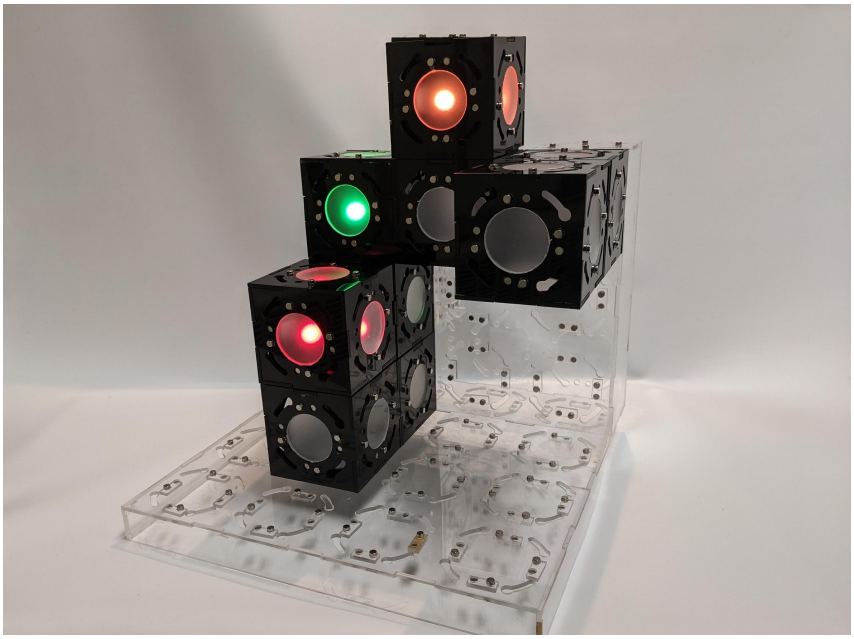
MARIA: Multi-Agent Robotic Intelligent Assembly

A Major Qualifying Project submitted to the faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of requirements for the Degree of Bachelor of Science



Submitted to:

Professor Andrew Clark
Professor Greg Lewin
Professor Carlo Pinciroli

Submitted By:

Peter Dentch
Mielynn Charter
Sriranjani Kalimani
Meha Mohapatra
Ella Torregrosa
Benjamin Waid

Submitted On:

May 6, 2021

Abstract **5**

Chapter 1: Introduction **6**
1.1 Background 6
1.2 Problem Statement 7
1.3 Statement of Contribution 7
1.4 Design Novelty 7

Chapter 2: Related Work **9**
2.1 Swarm Construction with Preplanned Structures 9
2.2 Predefined Structural Constraints 10
2.3 Smart Structure Visual Communication 10
2.4 Previous Project Work 11

Chapter 3: Design Overview **12**
3.1 Problem Formulation 12
3.2 Technical Approach 12
3.3 Building Algorithm 12
3.3.1 Design Requirements 12
3.3.2 Smart Structure Blueprint 12
3.3.3 Flexible Construction 14
3.3.4 Feasibility Check 15
3.3.5 Prioritization 17
3.3.6 Blueprint Encoding 18
3.4 Mating System 18
3.4.1 Design Requirements 18
3.4.2 Selected Mating System Design 19
3.5 Block-to-Block Communication 20
3.5.1 Communication System 20

Chapter 4: Experimental Evaluation **26**
4.1 Building Algorithm 26
4.1.1 Simulation 26
4.1.2 Modular Functionality 26
4.1.3 Functional Testing 27
4.1.4 Visualization and Algorithm Tuning 27
4.1.5 Final Building Algorithm Results 28
4.2 Block Prototype 28
4.2.1 Mating System Parameters 28
4.2.2 Build Order Unit Tests 29

4.2.3 Load Verification 30
4.2.4 Mating System Results 33
4.2.5 Determining Communication System Parameters 33
4.2.6 Communication System Unit Tests 35
4.2.7 Final Communication System 36

Chapter 5: Conclusion 37
5.1 Summary 37
5.2 Future Work 37
5.2.1 Finding Optimal Seeds 37
5.2.2 Indicating Completion of Construction 37
5.2.3 Inchworm Agent Improvements 37
5.2.4 Manufacturing Improvements 37
5.2.5 Block-to-Block Communication 38
5.3 Lessons Learned 38

Bibliography 39

Abstract

Construction can be optimized by using robotic workers to create structures from smart materials. The prior creation of a construction agent, a robotic inchworm, laid the groundwork for our team to focus on the development of smart building blocks which utilize visual light communication (VLC) and a decentralized planning algorithm. Using a decentralized algorithm allows each agent to act independently with simple rules, creating emergent behavior to build a desired structure using the structure itself as the primary organizer of construction.

Chapter 1: Introduction

1.1 Background

The proportion of the world's population that lives in urban areas is set to increase by 13% in the next 30 years, corresponding to a growth of 2.5 billion people [1]. With massive growth in urban areas, construction to support housing, education, businesses, and leisure will need to be erected throughout cities. Unfortunately, in its current state, the construction industry is inefficient and dangerous. There is an estimated "25 to 50 percent waste in coordinating labor and in managing, moving, and installing materials" with "losses of $15.6 billion per year due to the lack of interoperability" in the U.S. alone [2]. The construction industry in the U.S. has a higher fatal injury rate than the national average among different industries. In 2019, 20% of all fatalities in the private industry came from the construction sector [3]. As the population grows, the potential for widespread human displacement due to natural disasters will also grow and issues of efficiency and loss of human life in the construction industry will only be exacerbated. As a dangerous and inefficient line of business, construction is a perfect candidate for automated work and the introduction of robot workers.

While automation within the construction industry is not a new concept, through the incorporation of heavy machinery, "productivity in construction has barely changed at all" from 1947 to 2010 [4]. Additionally, there are still issues with labor shortages, injuries, and coordination management – all of which can potentially be reduced by robots, drones, and other autonomous systems [5].

Swarm robotics is one approach of automation that takes advantage of multi-robot construction. This approach uses a decentralized system to control a large number of simple and identical robots [6]. Theoretically, swarming allows robots to find their own tasks and be easily replaced in the event of failure in order to reach the overall goal. Swarming is grounded in algorithms that communicate simple rules to the robots. This results in efficient teamwork as the robots can build in parallel by completing individual tasks to achieve a collective end goal. Swarm construction has promised to be robust, flexible, and scalable [7]. As such, multi-robot systems are one solution that can eliminate labor shortages, decrease casualties, and increase efficiency.

One approach to swarm involves stigmergy, a biologically-inspired term that describes the mechanism of indirect communication between agents through the environment. The inspiration to use a stigmergic based swarm robotics system to solve construction obstacles comes from observing nature's builders - ants and termites. With stigmergy, swarms of insects are able to build collectively using decentralized planning methods without any individual builder having knowledge of the end structure. For example, when an ant is bringing a particle to their nest, the ant will tag it with a pheromone to inform other ants how to continue construction and find more materials. Ants will continuously detect and leave pheromones [7]. In this manner,

stigmergy enables indirect communication to aid in construction without relying on direct communication between builders.

In classical swarm construction algorithms, builders host the majority of the intelligence and blocks are simply units of construction. This project takes inspiration from the decentralized manner of stigmergic building by introducing intelligent building material with no centralized planner. By doing so, the blocks will be able to guide builders with simple instructions to create complex structures. Additionally, this mitigates the need for highly intelligent robots and instead delegates communication and construction planning among the building material. Multi-Agent Robotic Intelligent Assembly (MARIA) aims to understand the challenges associated with using a stigmergic approach to swarm based distributed construction.

1.2 Problem Statement

We develop a subscale swarm system by redesigning previous iterations of the construction algorithm and block design. The primary goal of MARIA is to better adapt to real world construction conditions by addressing constraints from the previously developed SMAC platform. Specifically, MARIA alters the construction algorithm to better reflect real-world constraints and adjusts the block design to reduce complexity in block placement decisions.

1.3 Statement of Contribution

We designed a self-organizing construction system by moving the intelligence from inchworms to blocks. A construction algorithm was formulated to implement decentralized construction and verify structural feasibility. An orientation-independent block prototype using pegged magnets was designed to support construction of overhangs and inverted staircases. Additionally, intelligence was redistributed to the blocks and a new block-to-block visual light communication (VLC) system allowed individual block agents to coordinate construction. These contributions simplified the inchworm decision making process from the previous iteration. Moreover, by creating more intelligent, orientation-independent blocks, we reduced complexity in block placement decisions which would be undesirable in a real-world construction setting. Finally, a greater number of possible substructures allowed for more accurate representations of real-world buildings.

1.4 Design Novelty

MARIA changed the design and behavior of the SMAC system's smart blocks, by developing a novel swarm system in which the construction material plays an active role in its assembly. Behavior of the smart blocks was changed to better reflect stigmergic communication techniques and allow for the swarm system to run without a centralized planner. The smart blocks monitored the presence of neighboring blocks using color sensors. Upon the addition of a new block, blocks in the structure flashed LEDs in a binary color coded scheme to transfer

location and goal structure data to a newly added block. Additionally, previously impossible structures in SMAC such as overhangs and inverted staircases were made possible by the custom building algorithm and physical mating system of the block agents. These changes implemented to the smart construction blocks present a novel system in which blocks play an active role in their own construction.

The structure of this manuscript is as follows. In Chapter 2, we discuss relevant work in the field. In Chapter 3, we detail the overall system design. In Chapter 4, we report the evaluation of the quantitative data from the testing phase. Finally, in Chapter 5, we summarize the main results, indicate directions for future work, and list the overall learning outcomes.

Chapter 2: Related Work

This section addresses relevant research in the fields of robotic construction, multi-robot systems, smart structures, and construction algorithms used in motivating the final design of MARIA.

2.1 Swarm Construction with Preplanned Structures

The TERMES swarm construction platform was developed to explore methods of collective robotic construction. The researchers focused on designing small robots which use information from their local area to coordinate the building of a predetermined structure. These robots are able to pick up, transport, and place building blocks. Robots are smaller than the blocks they ferry, enabling them to fit through a one block wide corridor; however, they can only carry a single block and step up or down by one level, see Figure 1. Given that the structure is known before construction begins, the robots guide themselves based on a set of rules, taking information from their local environment. This implementation allows for single-path, additive structures to be built. The limitations on this system stem from the physical abilities of the robots and blocks themselves. For example, the limitation in robot traversability does not allow for easy vertical building which limits the types of structures that can be built. Additionally, the robots take on the role of global organization and distribute updates during construction [8]. MARIA took inspiration from the work of TERMES by using an initial blueprint for construction and taking inspiration from their methods for analyzing the feasibility of a structure. Unlike the TERMES platform, which relies on construction agents to organize the building process, this project will attempt to move intelligence into block agents to make the structure self-organizing.

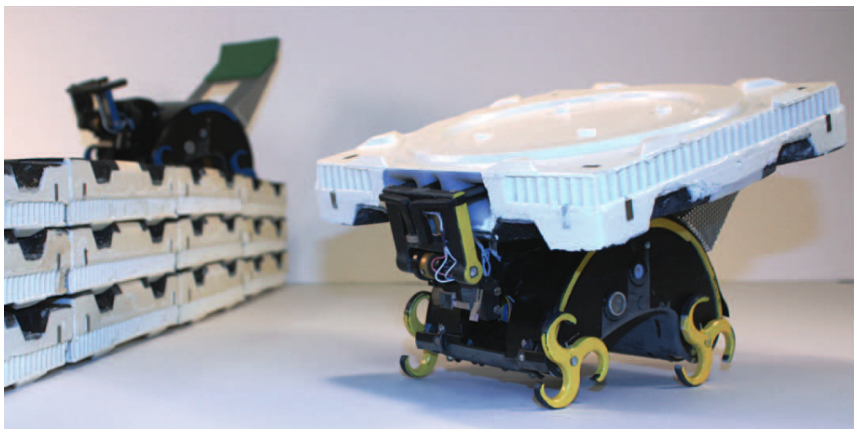


Figure 1. TERMES system [8]

2.2 Predefined Structural Constraints

Swarm Robotics Construction System (SRoCS) developed blocks which directed construction robots using visual stigmergic messages, see Figure 2. Robots in SRoCS adapt the structure to the terrain and complete a build in a flexible manner, even when given consistent initial conditions. The two parts of the system include intelligent blocks and BeBots that pick and place these blocks. BeBots are not involved in decisions regarding the organization of the end structure and rely on blocks for build instructions. Although this allowed for flexibility during construction, there was no way to fully plan or predict the final structure [9]. MARIA mimics the method of predefined substructures as seen in SRoCS, while also being able to define a final structure at the beginning of construction. To do this, MARIA substructure constraints, based on material limitations, are first handled by pre-planning in the building algorithm rather than by the blocks during construction.

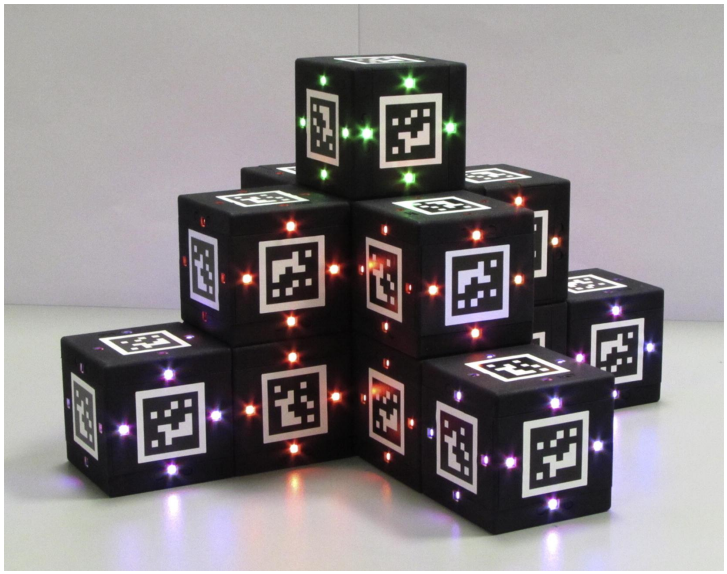


Figure 2. SRoCS blocks [10]

2.3 Smart Structure Visual Communication

Continued work on SRoCS explored the use of smart material to help guide construction and convey information to the system. Each block was outfitted with LEDs and Near-Field Communication (NFC) boards to communicate with the BeBots. Additionally, the building blocks utilized the AprilTag visual fiducial system which assisted BeBots in accurately locating blocks using computer vision. The blocks updated the LEDs on their faces according to stigmergic rules and, in turn, the LEDs indicated to the BeBots how to continue construction. This passive transfer of information allowed for a decentralized system where construction decisions came from dynamic signals [10]. MARIA used a similar concept by updating LEDs on block agent faces to indicate the next steps of construction, acting as both a messaging system

and visual indicator of the state of construction. By relying on blocks in the structure for localization data, SRoCS building agents were not able to validate location within the system which made block-to-robot location data more prone to error. MARIA implemented similar protocols to localize agents in the system but required additional intelligence in the building material to accommodate the preplanned goal structure. Additionally, to avoid inaccurate location errors, MARIA implemented block-to-block communication to transfer localization data.

2.4 Previous Project Work

MARIA is a continuation of work on Symbiotic Multi-Agent Construction (SMAC), a previous MQP at WPI. The team developed a construction platform including inchworm robots and smart construction material. The SMAC system used an inchworm robot which ferried and assembled building blocks in a 3D space. The inchworms traversed the building environment after receiving a blueprint of the final structure and built using a centralized construction algorithm. The system also utilized smart blocks to aid in construction which were designed to have identical gendered mating with one male and five female sides. This mating mechanism did not guarantee strength and stability of the structure as it expanded. Finally, the construction algorithm developed by SMAC separated the building environment into equally sized divisions based on the number of inchworms. The inchworms built only in their designated region and collaborated when necessary to ferry blocks to the other's regions [11]. Divisions acted as a way to preplan the order of assembly, but limited the system from acting collectively during construction. Experimental evaluation revealed high robot idle times when relying on other inchworms to ferry blocks to their region.

Chapter 3: Design Overview

3.1 Problem Formulation

Our team defined the following areas for improvement:

- *Decentralized Construction:* The new system should not rely on a centralized, single organizer directing construction. The structure is required to self-organize construction.
- *Intelligent Block Agents:* Block agents are able to communicate structure information and update in real time to indicate current structure state and organize future construction.
- *Improved Variety of Substructures:* To increase the number of buildable structures, inverted staircases and overhangs should be supported.
- *Reuse of Existing Block Subsystems:* To decrease electrical system complexity, existing block substructures should be utilized for multiple purposes beyond their initial intended functions.

3.2 Technical Approach

To address each requirement, our team structured the problems into three subsystems:

- *Building Algorithm:* To address decentralized construction and preplanning checks to include new substructures while rejecting impossible structures.
- *Mating System:* To address the additional desired substructures and improve overall stability of the final structure.
- *Block-to-Block Communication System:* To support decentralized construction with intelligent block agents and address the reuse of existing block subsystems.

3.3 Building Algorithm

3.3.1 Design Requirements

The building algorithm must be able to take in a goal structure and determine its feasibility with respect to physical building constraints. The algorithm must evaluate construction and output a graph that can be used by the smart blocks to self-organize construction.

3.3.2 Smart Structure Blueprint

To account for the necessary information at any point in construction, a data representation of the structure known as the “blueprint” is distributed among block agents. The blueprint contains structural constraints and allows for blocks to organize construction in a physically stable manner. A virtual building-block environment is used to create goal structures

where the user manipulates and places blocks to design a desired structure. This blueprint creator was designed by Xenon Labs and customized by the previous MQP team, see Figure 3. The software saves this structure as a 3-dimensional *NumPy* array where each dimension specifies the x, y, and z positions of blocks in the environment. The *NumPy* array is translated into a graph format by converting each block coordinate location into a block object node. Each node in the graph contains the location of a block as well as a list of pointers to any "neighbor" blocks, forming the preliminary blueprint, see Figure 4.

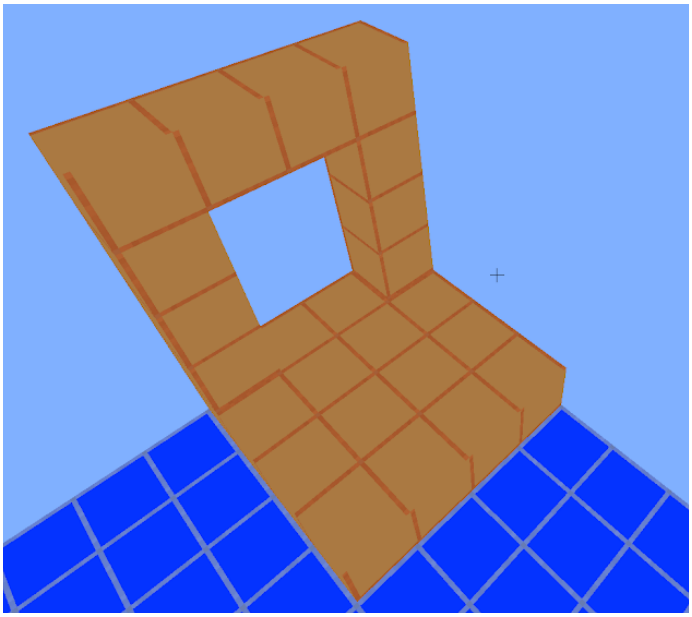


Figure 3. Structure Construction on Blueprint Creator.

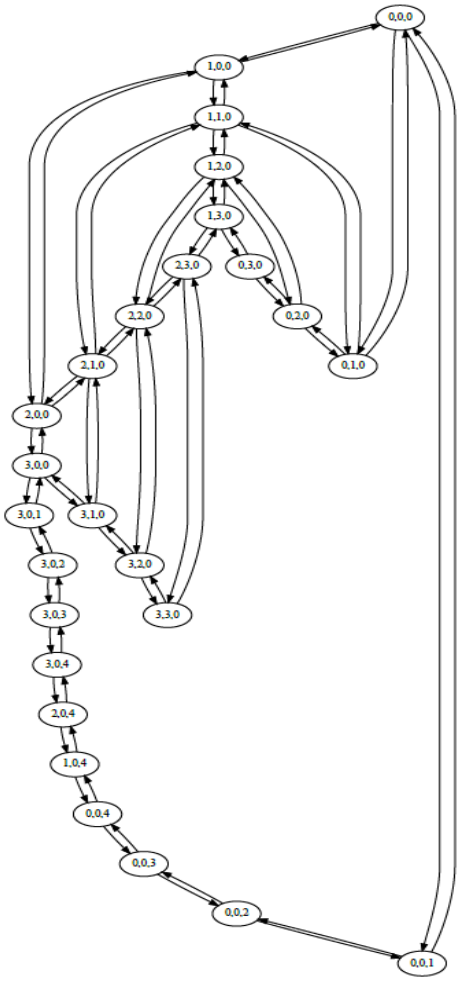


Figure 4. Initial Blueprint Graph.

3.3.3 Flexible Construction

The preliminary blueprint allows for a basic decentralized construction approach, called flexible construction, but does not take into account structural constraints. Even without any information in the blueprint aside from block locations, construction can occur as all blocks have knowledge of their required neighbors. To initiate the process of flexible construction, one or more starting blocks, or “seed blocks”, containing the blueprint are placed in the environment. The seed blocks initiate recursive construction by arbitrarily requesting neighbors until all blocks are placed and propagating the blueprint forward to new blocks. Flexible construction is guaranteed to result in a completed structure. However, it suffers from inefficiency and does not account for real-world limitations. To remedy these issues, the building order is modified to take into account structural constraints which brings the process of building closer to an optimal method while still guaranteeing a complete structure.

3.3.4 Feasibility Check

The following feasibility checks are executed on the blueprint in order to verify the physical validity of a goal structure, see Figure 5.

```
      MAX_OVERHANG_LEN → 3
      MAX_OVERHANG_WEIGHT → 2

      function isFeasible(block):
         not_connected → (block.dist == -1)

      large_overhang → (block.isOverhang and block.overhangLen >
MAX_OVERHANG_LEN)

         heavy_overhang → (block.isOverhang and block.weight >
MAX_OVERHANG_WEIGHT)

         if not_connected or large_overhang or heavy_overhang:
             return False

         return True
```

Figure 5. Feasibility Check

Phase 1: Connectivity

When the preliminary blueprint is generated, the distance of each block from the seed is updated as a block object parameter. Block distance is initialized to an arbitrary negative number that is replaced with the correct distance. If a block is unconnected to the structure, its distance from the seed is not updated, thereby indicating that it is not reachable from the seed. This phase of the feasibility check returns false if there is a block with negative distance from the seed. On passing this check, the algorithm proceeds to check for overhang length.

Phase 2: Overhang Length

A block is identified as overhanging if it is attached to a block and has no block underneath. The algorithm proceeds to return the length of the overhang attached to that block. The overhang feasibility check fails if a block's overhang length is greater than a defined threshold. For example, overhangs over three blocks are not structurally feasible, see Figure 6.

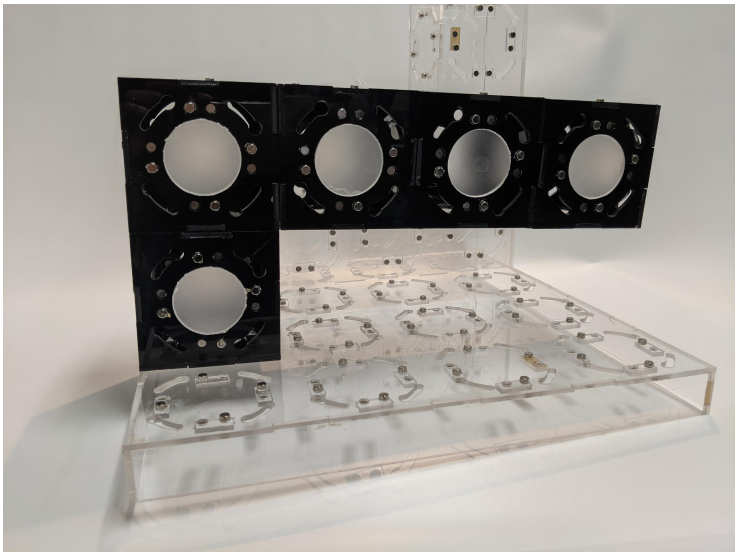


Figure 6. Overhang Length Limit

Phase 3: Overhang Weight

The weight carried by each block is updated and stored as a block object parameter. If a block meets general overhang requirements, the weight it is supporting needs to be within a threshold to be marked feasible. For example, inverted staircases of more than three blocks are not feasible, see Figure 7. At this point, the blueprint is devoid of impossible configurations, and construction begins.

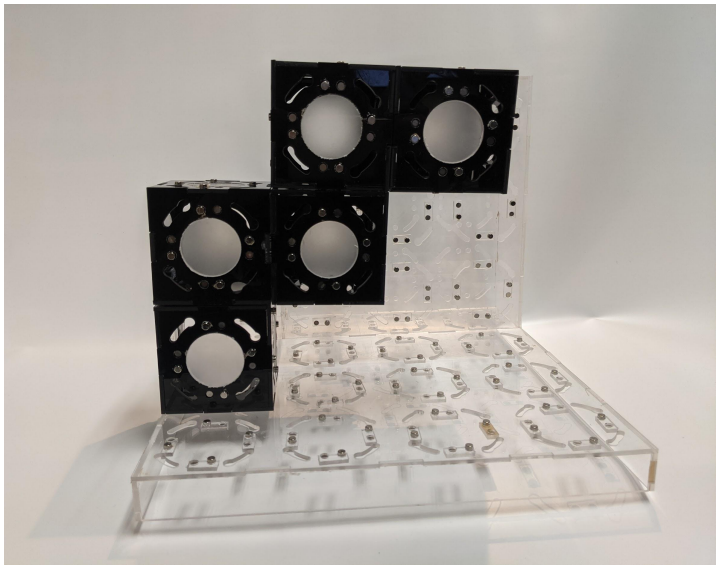


Figure 7. Overhang Weight Limit

3.3.5 Prioritization

Once a structure is marked feasible, its nodes are reordered or prioritized to comply with physical building rules. Nodes are searched from the previously specified seed location and the neighbors of each block are reordered based on the following heuristics to generate the intrinsic building order. A greedy approach is used to constrain the building order by applying a cost for each move. For example, a vertical move costs more than a horizontal move and blocks that support more weight gain priority, see Figure 8. The nodes are then rearranged in the order of increasing cost, further optimizing structural stability. A new ordering of neighbors is determined using the priority of each neighbor of a block. This outputs the final blueprint with fully ordered neighbors and construction can now begin.

```
function prioritizeNeighbors(block):
# Goes through a block's neighbors and sorts them according to
  constraints (distance and direction)
    temp → PriorityQueue()   # temporarily store new neighbor order
    for n in block.neighbors:
    # compare z_coordinate of block and neighbor to check if neighbor
      is horizontal or vertical


       if block.z_cooridinate == n.z_coordinate:
           # dir = 1 if neighbor is horizontal to parent block
           dir → 1
       else:
           # dir = 2 if neighbor is vertical to parent block
           dir → 2

       # priority is a heuristic based on distance and direction
       priority → n.dist * dir
       temp → (priority, n)    # append to temp in order of priority

    block.neighbors → temp
```

Figure 8. Prioritization Using Heuristic

Note on Parallelism

Parallelism is induced as more blocks are added to the structure during construction. Inducing parallelism can also be done by initializing more than one seed block. The algorithm does not treat seed blocks differently than other blocks in the structure and the prioritization of construction is specific to individual blocks. For feasible structures, nodes are prioritized based on each seed block specified, which outputs different graphs that ultimately lead to the same goal

structure. As a result, the emergent behavior of construction is parallelized as blocks are added to multiple parts of the structure at the same time.

3.3.6 Blueprint Encoding

Once the final blueprint is obtained, it must be transmitted to blocks in a manner that can be easily parsed. Blocks require coordinate information about their own location as well as the locations of their neighbors in the computed priority order. The blueprint graph is encoded in ASCII to a delimited file that contains the block and its respective neighbor coordinates in order, see Figure 9. This is then passed to blocks using the VLC communication system described in section 3.5.

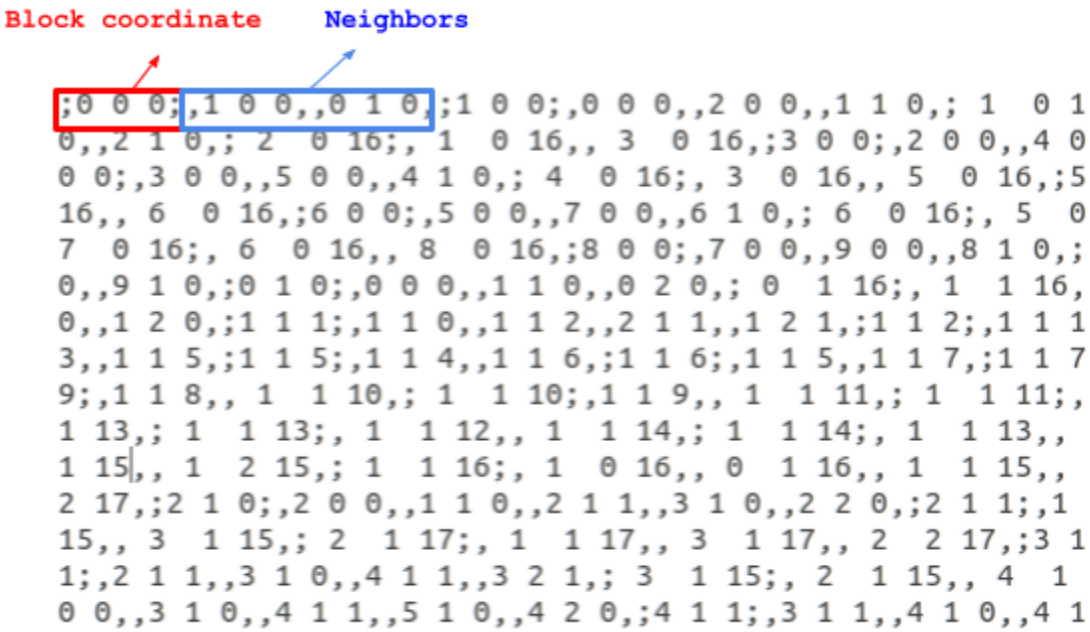


Figure 9. Encoded Blueprint

3.4 Mating System

3.4.1 Design Requirements

The smart blocks' ideal mating system needs to be orientation-independent and create strong connections to support other blocks within the structure. Orientation independence allows for a block to be manipulated and placed in the structure without concern for the directional alignment of faces. Additionally, blueprints no longer need to check face-to-face compatibility, thus further simplifying the preplanning in comparison to SMAC. Without needing to consider the mating orientation, construction is more efficient as blocks do not need to be rotated to be placed in the structure. The mating system also needs to support the following substructures as they are necessary to complete features including doorways, overhangs, and archways, see

Figure 10. These substructures guide the feasibility checks to determine the order of building and whether the structure can support not only itself, but also the weight of working inchworms.

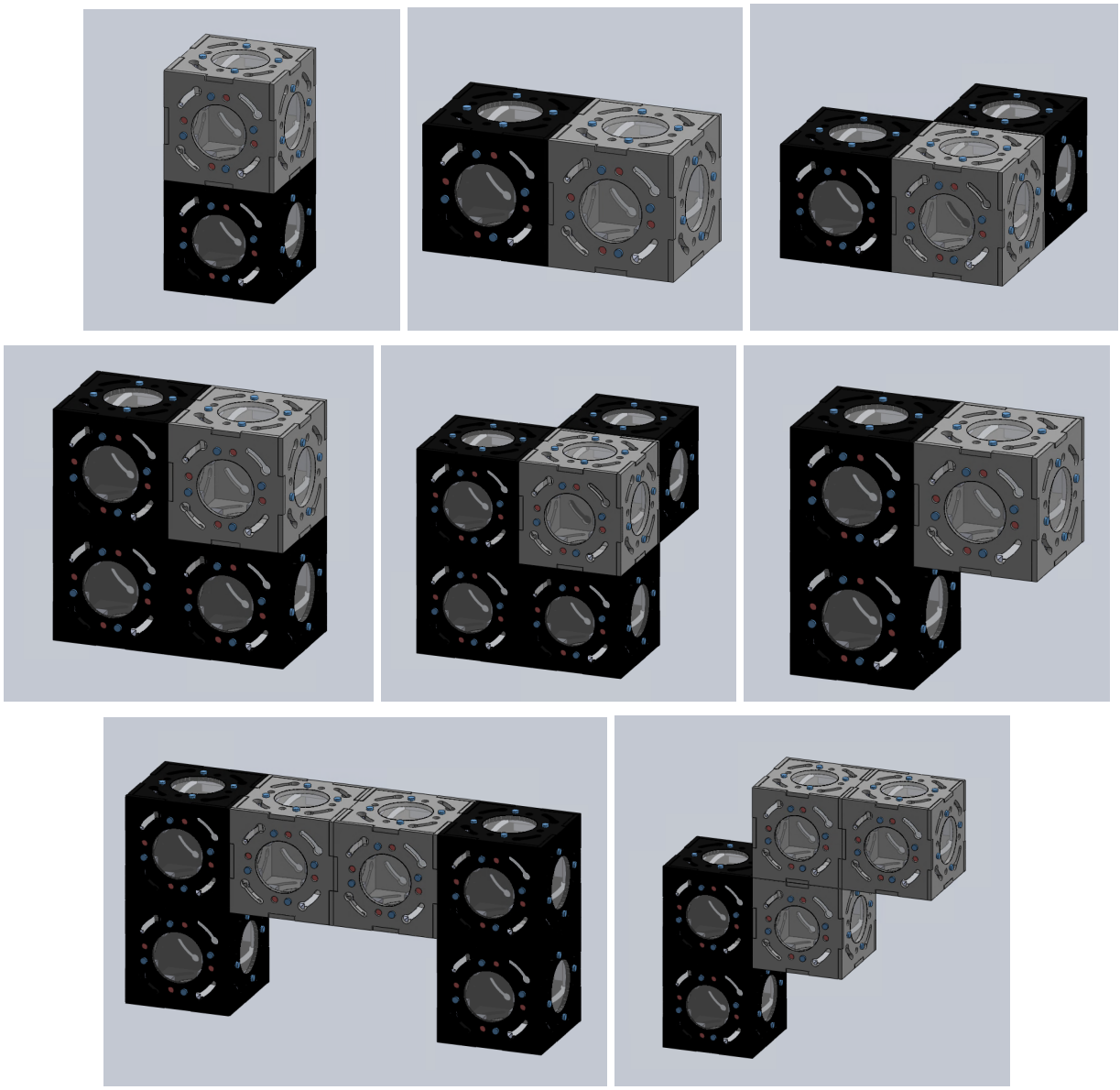


Figure 10. Desired substructures: Simple Top, Simple Side, Ground corner, Vertical Corner-Two, Vertical Corner-Three, Overhang, Doorway, Inverted Staircase

3.4.2 Selected Mating System Design

The selected mating system incorporates eight magnets at two different heights on a single face. These magnets are placed in pairs with opposing poles facing outward to ensure an orientation-independent connection, see Figure 11. The attraction of the magnets ensures that the blocks can self-align when placed within 5 millimeters of one another. The magnetic force produces a connection that is strong enough to satisfy all substructures. Additionally, the

alternating magnet heights creates a peg feature which improves the structure's ability to withstand shear forces caused by the structure weight and the inchworms' movements around the environment.

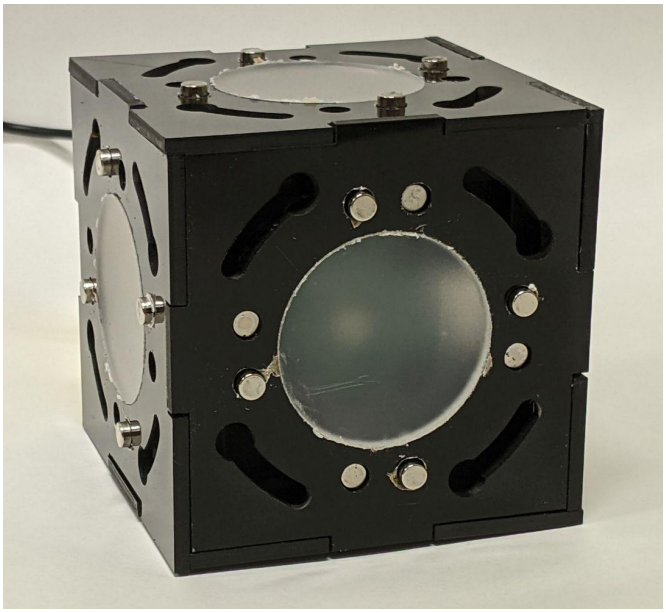


Figure 11. Final Block Prototype Depicting Magnetic Pegged Mating System

3.5 Block-to-Block Communication

3.5.1 Communication System

To accommodate the physical goals of the new blocks, the communication system cannot interfere with the mating system and should be able to transmit messages in a small physical space. The chosen method of communication from block-to-block relies on Visual Light Communication (VLC) to transmit messages between blocks. The system utilizes LEDs from previous iterations of the block design as both a visual indicator of structure build status and an interface to transmit the blueprint to newly added blocks. To enable VLC, each block face contains an addressable LED (Neopixel) and a color sensor. The Neopixel LED flashes predetermined colors that encode four bits of binary data. The color sensor is used to detect the colors from the Neopixel. The data is then sent to the ATMega microcontroller to decode the blueprint so a block can begin requesting neighbors.Other necessary peripherals to store the blueprint determine the orientation of the block and execute state machine functionality, see Figure 12.

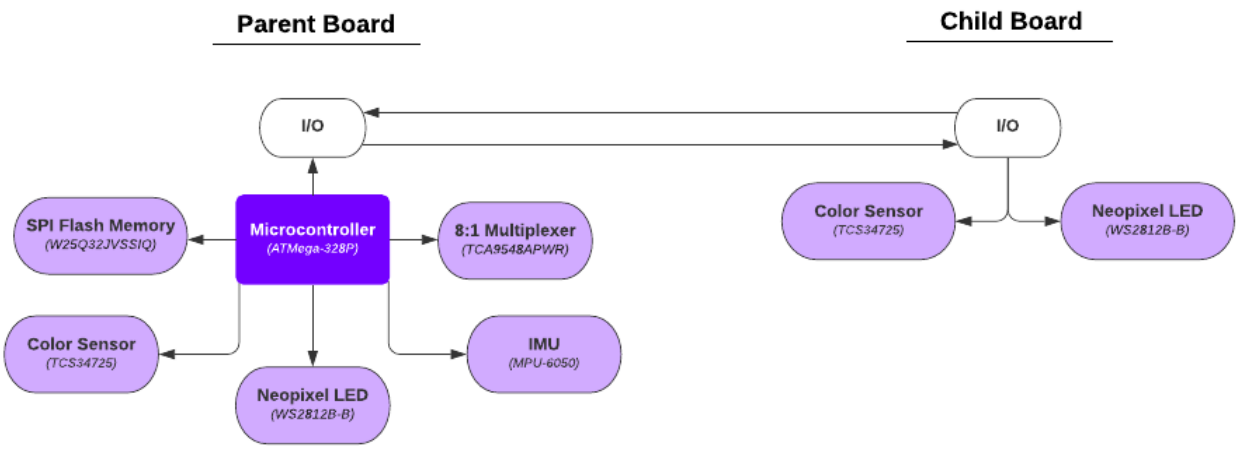


Figure 12. Peripherals of the Parent and Child Boards.

To include these peripherals within the block, two printed circuit boards (PCBs) designs were developed: a main parent board (appearing on one face of the block) and child boards (appearing on five faces of the block), see Figure 13. The Parent Board acts as the controller of five child boards, handling state machine control with the ATMega microcontroller while the child boards only sense or flash color, sending any data to the Parent Board. The PCBs are secured to a 3D printed inner block core which houses the electrical components within the acrylic block, see Figure 14.

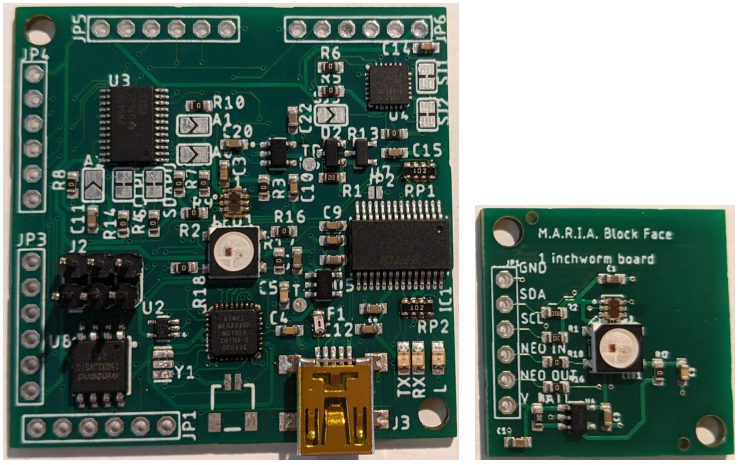


Figure 13. The parent board (left) which acts as the brain of a block controlling the child boards (right) in the other five faces of the block.

Figure 14. Parent and Child Boards secured to the inner block core to be placed within the block.

A state machine is used to control peripheral functionality within the block in a timed and consistent manner. A constant execution rate is set to sample sensors, update LEDs, and trigger a state change as a function of past and current measurements. Block operation states are summarized as follows:

- *Idling*: a newly powered-on block waiting to be added to the structure.
- *Neighbor requesting*: a block in the structure flashes a predetermined color to request a neighbor on a specific face.
- *Blueprint sending/receiving*: After sensing a neighbor has been added, a block in the structure flashes encoded colors to transmit the blueprint to the new block, see Table 1.
- *Structure updating*: After sending or receiving the blueprint, a newly added block and it's neighbor each determine a new neighbor to request before entering the *neighbor requesting state*.

A complex state developed for a block was the *Blueprint sending/receiving state* which handled the transmission and reception of the blueprint. Through early testing of the color sensor, it was discovered that the red, green, and blue (RGB) colors shined by the Neopixel LED did not correspond linearly to the RGB values returned by the color sensor. The Neopixel LEDs are set using varying amounts of red, green, and blue components which are regulated through controlling their individual brightness values. The brightness is set through pulse width modulation (PWM), a common technique used to make a digital signal mimic an analog range. However, even when set to their maximum brightness of 255, the RGB components do not have

a PWM duty cycle of 100% which contributes to noise at the sensor. The nonlinearity of the color sensor, noise from Neopixel PWM, and asynchronous timing, are significant factors in our choice of colors used for encoding binary data, see Table 1. The encoded blueprint in its raw form is serialized into a binary file which is split into 4-bit segments corresponding to predetermined colors. The colors are flashed by the Neopixel LED and sensed by a receiving block before being decoded.

Binary Messaging Colors			
0000	0001	0010	0011
0100	0101	0110	0111
1000	1001	1010	1011
1100	1101	1110	1111

Table 1. The 16 chosen data transmission colors and their corresponding binary nibble values.

The frequency at which a color can be sampled was first limited by the time required for the sensor to fully read an RGB value. As a result of the decentralized nature, the system needs to create an asynchronous technique to send messages, however, it is also desirable to consistently acquire at least three different RGB samples during both a light and dark cycle of a flashing LED, see Figure 15.

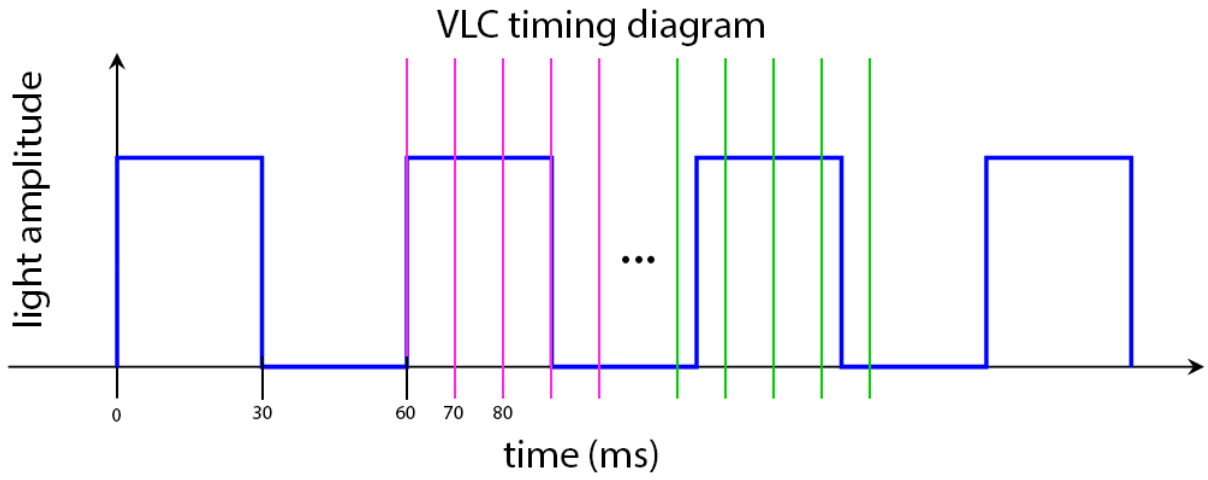


Figure 15. Timing diagram of four light-dark cycles with period, T, of 60ms

The green lines in the timing diagram indicate where samples from the color sensor could occur, giving us three samples of the color to be averaged. The pink lines simulate a situation where the color sensor samples during a dark-light transition. In this case, two samples of the color are still received which can still be averaged using our algorithm, see Figure 16.

```
function averaging(colorSample1, colorSample2, colorSample3):
    minDiff1 → abs(colorSample1 - colorSample2)
    minDiff2 → abs(colorSample2 - colorSample3)
    minDiff3 → abs(colorSample1 - colorSample3)

    minDiff → minimumValue(minDiff1, minDiff2, minDiff3)

    if(minDiff < threshold):
        if(minDiff1 == minDiff):
            colorAvg → (colorSample1 + colorSample2) / 2
        if(minDiff2 == minDiff)
            colorAvg → (colorSample3 + colorSample2) / 2
        if(minDiff3 == minDiff):
            colorAvg → (colorSample1 + colorSample3) / 2
    return colorAvg
```

Figure 16. Pseudocode of the averaging algorithm used on color samples

The pseudocode above depicts the algorithm used for each red, green, and blue component of a color sample. First, the minimum difference which occurred between the three colors is determined. If the calculated difference is below a preset threshold, the samples are determined to meet our requirements for minimal noise or inaccuracy. The two samples which have the least difference are then averaged as our initial testing showed that, as the color sensor stabilized, the two values with the least difference also corresponded most closely to an accurate color reading. Should the minimum difference be greater than the threshold, the samples are determined to be too noisy to be reliably averaged. Extensive tuning was used to choose the messaging colors such that they would virtually never surpass the threshold. However, if they did, the sample would be considered erroneous and the message would need to be resent.

The average value obtained from the algorithm is then converted to the hue, saturation, value (HSV) spectrum so the hue value can be compared against predetermined expected hue values which correspond to 4-bit binary nibbles (half-bytes). The saturation and value obtained in this conversion are discarded. Eventually when all of the colors have been identified and decoded, the full blueprint is recreated by the newly added block, detailing its location and the location of its neighbors. It can now begin requesting neighbors through color which visually indicate the face of a block where a neighbor should be placed, see Figure 17.

Figure 17. Color code to visually indicate the face of a block which is requesting a neighbor.

Chapter 4: Experimental Evaluation

4.1 Building Algorithm

4.1.1 Simulation

The custom simulator uses Matplotlib to simulate the construction of the final blueprint. This serves as the primary visual means to test the algorithm and construction process before physical construction begins. The simulator builds starting from the seed block and placing block neighbors with regards to the priorities designated by the blueprint. The structure is visualized using 3D plotting with each cube representing a block, see Figure 18.

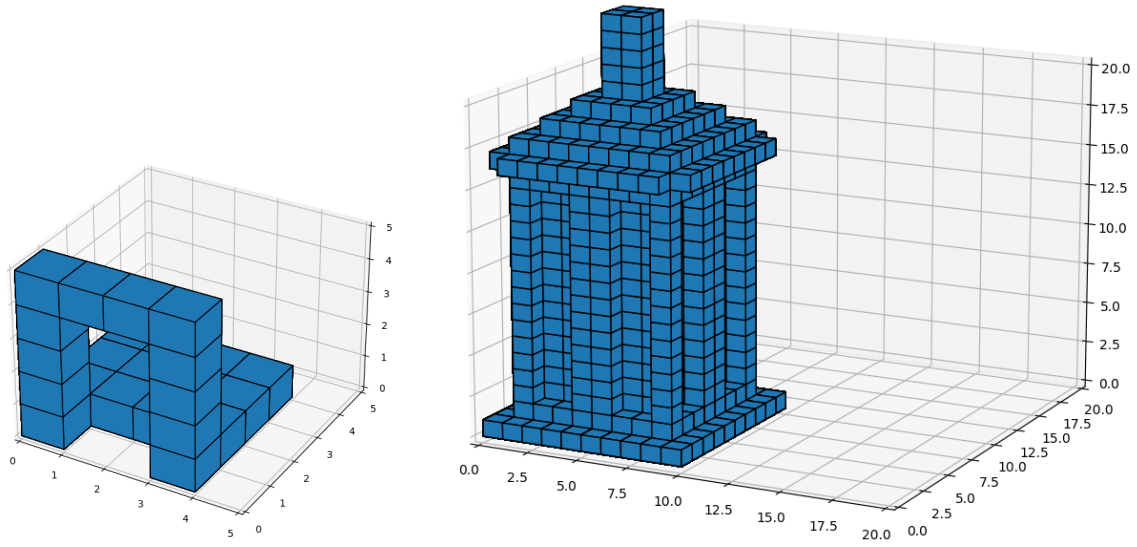


Figure 18. Simulation Outputs

4.1.2 Modular Functionality

The algorithm is designed to be modular such that the functionality is separated into independent modules that govern various parts of the blueprint creation. These modules can be customized to suit the needs of any constraints or be made scalable with simple modifications. Two key models that offer such customizability are the feasibility check and heuristic function.

The feasibility check indicates whether a block is safe to be placed in a given location. This function can be used in advanced graph search functions to offer scalability. Overhang size and weight limits can also be modified to accommodate various physical capabilities.

Prioritization or building order guidance is controlled by the heuristic function. Although the current heuristic allows for basic construction rules, it can be further developed to incorporate complex building constraints to change the building order.

4.1.3 Functional Testing

Testing is done using a custom testing suite built to verify the functionality of each module of the algorithm. *NumPy* arrays obtained from the Xenon Labs Blueprint Creator are directly used to make various structures. This suite encompasses possible and impossible substructure configurations based on constraints described in section 3.3.4 Feasibility Check. The suite also includes large structures with over 200 blocks to test the time complexity of the algorithm. Even with naive search approaches, the complete structural analysis, including feasibility checks, for large structures is on the order of milliseconds. Testing output results in complete simulation and feasibility check data, see Figure 19.

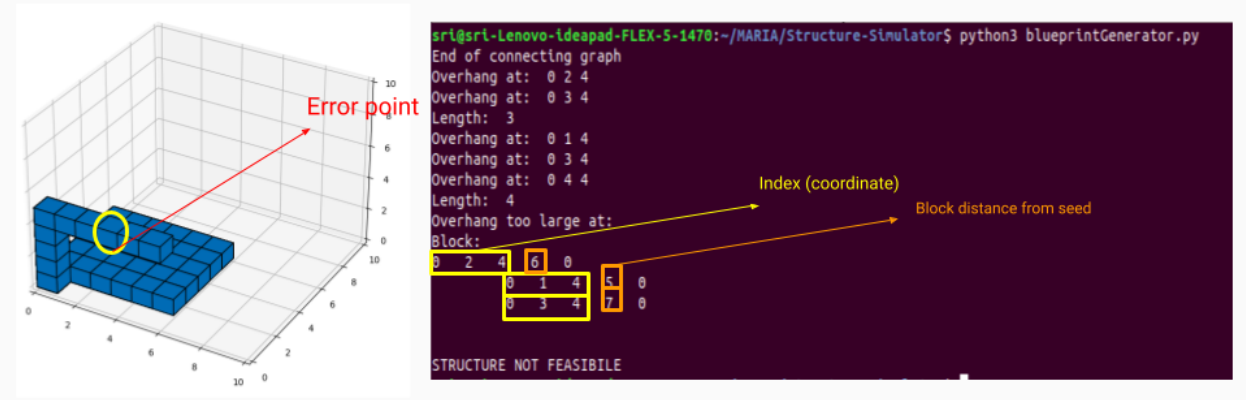


Figure 19. Feasibility Check Output

4.1.4 Visualization and Algorithm Tuning

The ability to visualize construction is crucial in making improvements to the algorithm as work is done. Using the simulator, the effects of any alterations to the algorithm are seen immediately and can be analyzed on how further changes should be made. Structural failures and inefficiencies in construction are to be noted and addressed.

The process of analysis and gradual improvement began with simulations of flexible construction. These were run to test the completeness of the flexible construction method, confirming all blocks in the structure were reached. Once verified as complete, heuristics were added and tuned to address specific issues.

In order to coerce the structure into building in a more stable manner, building horizontally was prioritized over building vertically. This was seen to reduce the chaotic building patterns of the random flexible algorithm and approached the behavior of a layer-by-layer construction method. Although this led to more sound construction, time was still lost building branching paths away from the seed. This led to the addition of another heuristic that gave higher priority to blocks closer to the seed. However, the system still did not take into account physical limitations of the blocks.

To account for the blocks' structural capabilities, a series of checks were developed to verify the structure was built in a feasible manner. To ensure a structure was safe to construct,

rules for overhanging blocks were defined. It could be determined if a block was a part of an overhang, as well as the length of the overhang at that block. Using this information, the system could limit how far it would allow an overhang to extend. Overhangs also were limited to supporting a certain number of blocks, again verified by the algorithm to prevent any impossible structures.

4.1.5 Final Building Algorithm Results

Testing in simulation resulted in a complete algorithm that could build feasible structures with minimal idle time. As compared to the previous year's system, construction never had to pause as sections were waiting to be completed. The parallel nature of the decentralized system allowed blocks to organize construction efficiently while still obeying certain rules. The structural rules and heuristics were designed to be easily modified such that the algorithm could be tuned to any major changes in block design or desired behavior.

4.2 Block Prototype

4.2.1 Mating System Parameters

Using build order and structural stability tests, it was determined that the magnetic pegged system is the best candidate to fulfill the design requirements: orientation independence and stable substructures. To achieve these requirements, we designed and tested six potential mating systems for orientation independence and structural stability. The final mating system is a fusion of the two best performing mating systems, pegs and magnets, combining their individual advantages, see Figure 20 and Table 2.

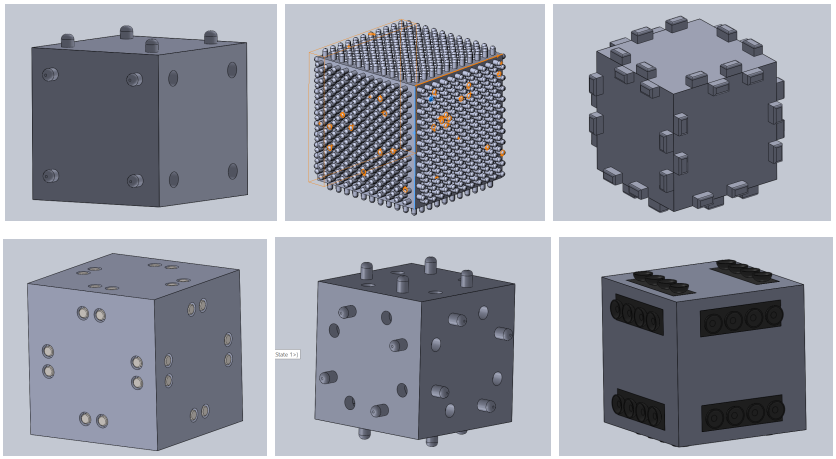


Figure 20. 6 Initial Mating Systems: 3-Sided, Bristles, Jigsaw, Magnets, Pegs, Suction Cups

Unit Test	Sub-Total	Mating Style					
		3Sided	Bristles	Jigsaw	Magnets	Pegs	Suction Cups
Orientation Independence	**5**	0	5	5	5	5	5
Substructures	**32**	16.5	0	16.5	32	20.5	10
Dead Load	**14**	14	7.5	14	14	14	14
Live Load	**17**	13	0	13	17	17	0
Block Weight	**5**	5	3	4	2.14	1.43	2.87
Total Score	**125**	**48.5**	**10.5**	**52.5**	**70.14**	**57.93**	**31.87**

Table 2. Decision Matrix: Total Performance Scores

4.2.2 Build Order Unit Tests

Build order tests are created to display what final and intermediate structures are possible with each mating system. The blueprint's feasibility requirements are partially informed by the data collected from this category of unit tests.

- *Orientation Independence:* An orientation-independent mating system guarantees that every face on a block is compatible with any other block face. Orientation independence removes the constraints on the order in which blocks can be placed and which block face an inchworm interacts with, see Table 3.

		Mating Style					
Score		0	5	5	5	5	5
Test	**Weight(5)**	3Sided	Bristles	Jigsaw	Magnets	Pegs	Suction Cups
Orientation Independent	**5**	y	n	n	n	n	n

Table 3. Decision Matrix: Orientation Independence Scores

A score of five correlates to an orientation-independent mating style, while a score of zero correlates to an orientation-dependent mating style. Both the flushed magnets and pegged mating system are orientation-independent.

- *Substructures*: The following substructures are considered, as they encompass a variety of intermediate and foundational configurations, see Table 4.

		Mating Style					
Score		16.5	0	16.5	32	20.5	10
Test	**Weight(5)**	3Sided	Bristles	Jigsaw	Magnets	Pegs	Suction Cups
Simple Top	**5**	Y	y	y	y	y	y
Simple Side	**5**	y	y	y	y	y	y
Gnd Corner 1st Layer	**4**	y; a little difficult	no	y; with difficulty, relying on wiggle-ability	y	y; a little difficult	no
Overhang	**4**	no	y	no	y	y	y
Overhang # before unstable			2		2	1	2
Vertical Corner (2-side)	**5**	y;wiggle a bit	y	y; depends on wig-ability	y	y;wiggle a bit	y
Vertical Corner (3-side)	**3**	y; if all 3 are not the same	no	no	y	y	no
Door Way	**4**	y; only can test with 1	y	no; bc can't do overhang	y	y; only can test with 1	y
3 blocks in the doorway?		y	y	no	y	no	y
Inverse Staircase	**2**	no	y	no	y	y; if peg length increase	y

Table 4. Decision Matrix: Substructure (Figure 10) Capability Scores

A higher score correlates to the mating system being able to successfully create more of the substructures. The flush magnets have a higher score in the configurations tests because there are no protruding features that hinder alignment. Additionally, the magnetic force between structure faces and the new block's face act as a self-alignment method.

4.2.3 Load Verification

Throughout the build period, it is important that all parts of the structure can support the live and increasing dead load. The data in this category of unit tests analyzes the strength and stability of the mating system and block manufacturing process.

- *Dead Load:* Dead load refers to the weight from blocks that form the structure. To analyze a block's ability to withstand the weight of the structure built on top of it, we placed a variety of loads on the blocks to assess how well the different block materials endure weight, see Table 5.

Score		Mating Style					
		14	7.5	14	14	14	14
		PLA	TPU	PLA	PLA	PLA	Acrylic
Test	**Weight(5)**	3Sided	Bristles	Jigsaw	Magnets	Pegs	Suction Cups
5 lbs	**5**	y	y	y	y	y	y
10 lbs	**5**	y	m	y	y	y	y
15 lbs	**4**	y	n	y	y	y	y

Table 5. Decision Matrix: Dead Load Scores

Higher scores correlated to blocks that could withstand 15 pounds of weight. The 3D printed PLA and acrylic manufactured blocks could support more weight than the TPU printed blocks.

- *Live Load:* Live load refers to the forces produced on the blocks by moving inchworms as they traverse the structure. To analyze how secure blocks are within substructures, we noted how much the block was able to move in different directions before the mated connection failed, see Table 6.

		Mating Style					
Score		13	0	13	17	17	0
Tests	**Weight (5)**	3Sided	Bristles	Jigsaw	Magnets	Pegs	Suction Cups
Simple Top	**5**	z= .5mm; no wiggle room, pretty secure so it shouldn't pop out	bristles can stay attached with only a fraction of the face but can be very misaligned	z = 0; even though it should be stationary in the xy-plane, if the the faces are more than 0mm apart, any attempt to shake the structure would dislodge the block	x,y= 4mm misalignment, possibility to become completely removed from the structure increases. z = 5mm (magnets will pull together for anything 5 or less) Mated faces can be rotated very minimally before becoming unaligned. (with mated cases still touching) Cannot be rotated otherwise (lifted from mated face)	z= .5mm; no wiggle room, pretty secure so it shouldn't pop out	The block can remain attached with at least one strip of suction cups bu can be very misaligned
Overhang	**4**	N/A ; can't even hold this position	bristles can stay attached with only a fraction of the face but can be very misaligned	N/A ; can't even hold this position	x,y= 4mm misalignment, possibility to become completely removed from the structure increases. z = 5mm (magnets will pull together for anything 5 or less) Mated faces can be rotated very minimally before becoming unaligned. (with mated cases still touching) Cannot be rotated otherwise (lifted from mated face)	x=.5mm; no wiggle room, pretty secure so it shouldn't pop out	The block can remain attached with at least one strip of suction cups bu can be very misaligned
Vertical Corner (2-side)	**5**	x= 0mm, y=0; no wiggle room, pretty secure so it shouldn't pop out	bristles can stay attached with only a fraction of the face but can be very misaligned	z = 0; even though it should be stationary in the xy-plane, if the the faces are more than 0mm apart, any attempt to shake the structure would dislodge the block	very difficult to move, any misalignment removes the block	x= 0mm, y=0; no wiggle room, pretty secure so it shouldn't pop out	The block can remain attached with at least one strip of suction cups bu can be very misaligned
Vertical Corner (3-side)	**3**	x= 0mm, y=0, z=0mm; no wiggle room, pretty secure so it shouldn't pop out	bristles can stay attached with only a fraction of the face but can be very misaligned	x,y,z=0mm ; very solid, not much wiggle room to even move in those directions	very difficult to move, any misalignment removes the block	x= 0mm, y=0, z=0mm; no wiggle room, pretty secure so it shouldn't pop out	The block can remain attached with at least one strip of suction cups bu can be very misaligned

Table 6. Decision Matrix: Live Load Measurements and Observations based on Figure 10 Substructures

A higher score meant that the block would be more secure within the structure. Both the magnets and pegs scored higher, indicating that it was difficult to dislodge the blocks out of the structure.

4.2.4 Mating System Results

The final mating system is composed of 1/16” thick magnets and 3/16” thick magnets placed on acrylic to form a peg feature, see Figure 11. This system takes advantage of the self-aligning force from the magnets and reinforces the structure’s ability to withstand shear forces between two mated faces. The walls of the block are acrylic which allows for efficient and accurate manufacturing so that there is less variability from block to block.

4.2.5 Determining Communication System Parameters

Properties of the specific color sensor and LED chosen for the system affects our flashing and sampling rates as well as the number and value of the colors we were able to use to transmit data. To interface with the color sensor we use a library with fixed sampling rates. Based on the TCS3472 data sheet, the color sensor requires 2.27ms minimum at the smallest analog to digital converter (ADC) register size to achieve a stable reading. Additionally, we also found it desirable to choose the minimal integration time to maximize the final transmission rate. Considering the fastest color sensor sampling rate available to us, we were unable to push the flashing rate of the LED beyond the time it would take for three samples to be acquired. Given the stabilization time of the color sensor three samples of a single color are taken so the thresholded average of the samples (described in section 3.5.1 Communication System) can be determined. Moreover, a constant gain value is set to ensure 8-bit values (between 0 and 255) are received by the color sensor. After initial testing of different sampling rates, it was determined that ~10ms was the lowest possible sampling time needed to accurately detect colors. As a result, the ideal sampling rate of the system was 100Hz, set using the selected integration time and gain value.

At the 100Hz sampling rate of the system, a flashing rate of 16.6Hz allows for at least two non-transition samples to be collected during a light flash. During our initial testing we only ever observed that one sample, out of three, occured during a light level transition. Given these results, we proceeded to choose the colors for our binary data transmission. Although the RGB components of the Neopixel LED can individually be set between 0 and 255, our testing of the color sensor showed disproportionate sensing of each component. If the color sensor had a linear behavior when sensing the RGB components, a graph of the components on the hue color spectrum would look like three distinct and equal trapezoidal waveforms, see Figure 21. Instead, we found that RGB waveforms were not directly proportional to each other when the color sensor was exposed to a Neopixel LED hue sweep, see Figure 22.

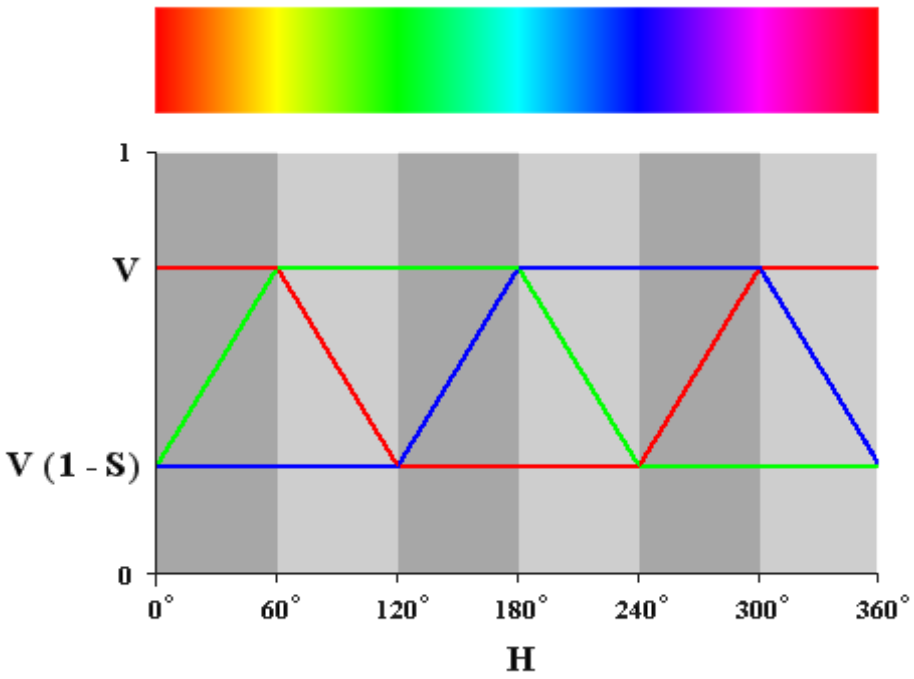


Figure 21. Graph of color of a full spectrum sweep

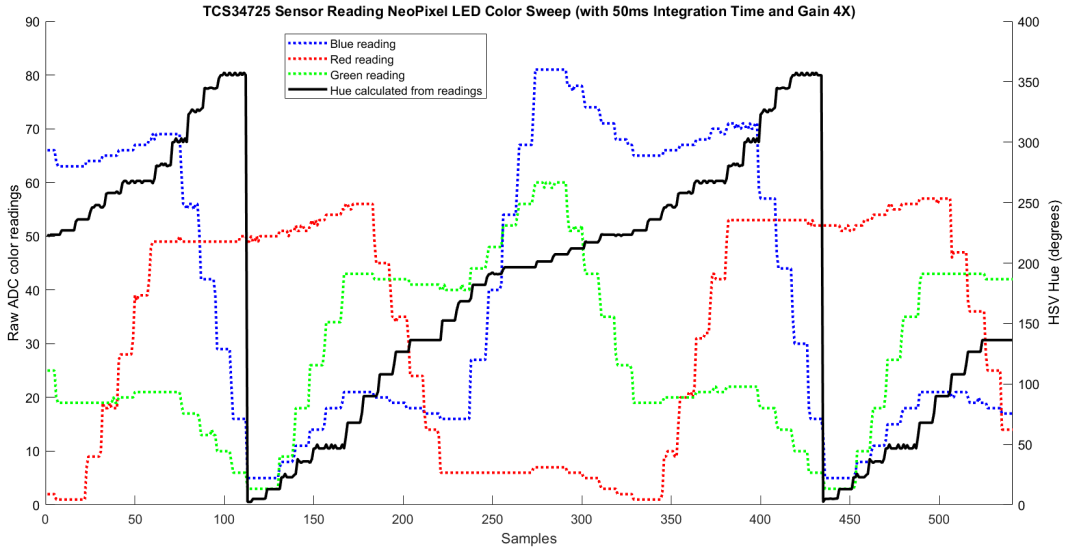


Figure 22. Graph of color sensor data during a full spectrum sweep of the LED showing non-linearity which favored blue samples. Note: Integration time and gain for this test are different from the final selected values, though the same behavior was observed regardless of these settings.

Based on these results, we chose to keep red at a constant 255 value and vary the green and blue components to choose initial colors for our data transmission before choosing colors which varied the red component. For each color chosen, we tested the variance of the value perceived by the color sensor to ensure it was below a determined variance and did not overlap in sensed value with any previously chosen colors. Given the limitations of the color sensor, 16 colors were found which could be more thoroughly tested and eventually used in the communication system.

4.2.6 Communication System Unit Tests

Four unit tests were performed to determine the accuracy of sensing the individual chosen colors as well as the rate of successful data transmission. Tests were completed in the acrylic block prototypes using child boards connected to an Arduino Nano, see Figure 23.

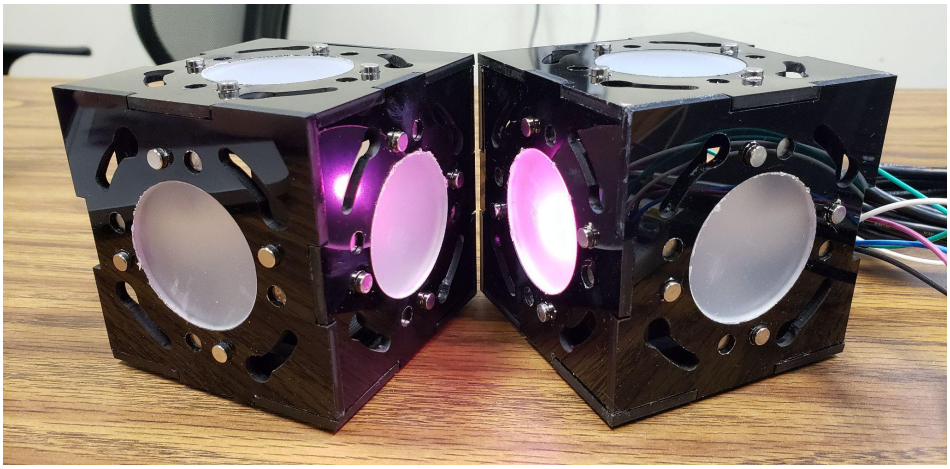


Figure 23. The rig used for unit testing consisting of one child board in each block used for flashing the LED.

- *Addition of a Block:* A block in the *neighbor requesting state* was color sensing. When a neighbor was placed on the appropriate side, the sensed color value decreased to below 5 in the range from 0 to 255 for each RGB component. If this transition was successfully detected and the block entered the *blueprint sending/receiving state*, the test was deemed successful.

- *Constant Blink:* A single color was flashed at the chosen 16.6Hz rate in a light-dark sequence with data recorded by the color sensor. Serial data which displayed the sensed hue was observed. When a color was accurately sensed, the monitor repeatedly displayed the same hue which corresponded to the color being sent. If a color was sensed accurately for 100 flashes of a light-dark sequence, the color was tested in sweeps with other accurately sensed colors.

- *Rainbow sweep*: After 16 colors were found to be acceptable by the *Constant Blink* test, an array of these colors was created. The rainbow sweep sequence, which flashes the 16 colors in order, was created and repeated for 100 cycles with data recorded by the color sensor. Thresholding of the colors was tuned until no errors were detected in 100 cycles.

- *Random sweep*: To ensure colors in a non-rainbow order would not affect the accuracy of sensing, a random sequence using a seed was created to flash the 16 colors in a pseudorandom sweep. The random sweep flashing sequence containing all 16 colors was repeated for 100 cycles with data recorded by the color sensor. Thresholding of the colors was tuned until no errors were detected in 100 cycles.

After the completion of the unit tests, known messages were sent using the color flashing system to verify that the tuned parameters were feasible for total system integration in a complete block prototype.

4.2.7 Final Communication System

The communication system was finalized with the 16.6Hz flashing rate, 100Hz color sampling rate, and 16 chosen data transmission colors. Given the 16 colors, each color was able to represent four bits of binary data. The average small scale blueprint size was found to be 50 bytes (400 bits) in size. Thus, the final observed transmission rate was 16.5 bytes/s (132 bits/s) and blueprint transmission occurred in ~6s.

Chapter 5: Conclusion

5.1 Summary

The MARIA Team successfully developed a new building algorithm, mating system, and communication system. The new simulation and pre-planning tools checked for impossible structures and allowed overhang and inverted staircase substructures. We enabled previously impossible substructures by creating a new, orientation-independent mating system. To aid in construction without a centralized planner, we developed a communication system which reused visual indication methods from previous work to transmit information between block agents.

5.2 Future Work

5.2.1 Finding Optimal Seeds

The current blueprint creation algorithm allows the user to specify a desired seed block. One approach to automating seed finding is evaluating block density along the structure to find seeds in high density areas. The seed block is constrained to be on ground level, i.e., the height value must be zero.

5.2.2 Indicating Completion of Construction

A feature to be added is an indication that the structure is finished with construction. This indicates to inchworms to exit the structure and not ferry more blocks. One approach to this is described as follows. When a block is placed, it parses out its neighbors from the blueprint. If all its neighbors are present, it is the last block of that branch to be placed. At this point, that block can send a message to its parent that gets passed down to the seed. Once the seed block(s) receive completion messages from all its neighbors, construction is complete. The seed block(s) can flash a certain color indicating safe exit.

5.2.3 Inchworm Agent Improvements

A new end effector was designed to be compatible with the new block agent prototype, however, it was not integrated with the robot and robot traversal was not tested. The integration of the new end effector along with improvements on the inchworm's control system will be beneficial for accurate and efficient movement.

5.2.4 Manufacturing Improvements

The process of manufacturing the block prototype is an area for improvement because they are composed of many individual pieces that are glued together. While this was successful for the current iteration of this project, it did make building blocks inefficient. It also added a

level of fragility to the blocks as components easily broke apart when a block fell. Other areas of improvement for the block prototypes include incorporating more realistic construction materials and blocks of different shapes to increase the types of structures that can be built.

5.2.5 Block-to-Block Communication

When researching communication methods, visual light communication presented an interesting and challenging approach to data transmission. Nonlinearity and sampling limitations decreased the overall transmission times of blueprint data. Additionally, in its current state using 16 colors to transmit 4-bit binary nibbles, our developed system is not scalable for blueprints larger than those of the small scale model (on the order of hundreds of bytes) constructed for this MQP. As such, if VLC is the chosen method of communication for future iterations of this project, testing of multiple color sensing modules is advisable to find a sensing unit which is more linear and accurate in sensing colors.

Furthermore, the communication system was highly tuned, decreasing the overall robustness of the protocol. With increased sensing accuracy, an error checking method could be developed to prevent entire message transmissions from being discarded due to a single color reception error.

5.3 Lessons Learned

In developing a novel solution to the problems presented in this project, our team took inspiration from previous work to develop a solution framework. Although the COVID-19 pandemic limited in-person interactions, we adapted to remote methods of team collaboration and integration of our developed subsystems. Finally, individual team members were able to gain experience in multiple areas of interest including, but not limited to, PCB design, swarm systems, research based development, magnetic mating, and rapid prototyping.

Bibliography

[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," May 16, 2018. https://ourworldindata.org/urbanization#:~:text=By%202050%2C%20it's%20projected%20that,to%20be%20higher%20than%20urban. (accessed Oct. 07, 2020).

[2] "Improving Construction Efficiency & Productivity with Modular Construction." The Modular Building Institute, [Online]. Available: https://www.modular.org/marketing/documents/Whitepaper_ImprovingConstructionEfficiency.pdf.

[3] "Commonly Used Statistics," Occupational Health and Safety Administration. https://www.osha.gov/data/commonstats.

[4]"The impact and opportunities of automation in construction," McKinsey & Company, Dec. 01, 2019. https://www.mckinsey.com/business-functions/operations/our-insights/the-impact-and-opportunities-of-automation-in-construction#.

[5]"The Potential Economic Consequences of a Highly Automated Construction Industry," Midwest Economic Policy Institute, Jan. 2018. [Online]. Available: https://midwestepi.files.wordpress.com/2018/01/the-economic-consequences-of-a-highly-automated-construction-industry-final.pdf.

[6] G. Beni, "From Swarm Intelligence to Swarm Robotics," in Swarm Robotics, Jul. 2004, pp. 1–9, doi: 10.1007/978-3-540-30552-1_1.

[7] E. Şahin, "Swarm Robotics: From Sources of Inspiration to Domains of Application," in Swarm Robotics, Jul. 2004, pp. 10–20, doi: 10.1007/978-3-540-30552-1_2.

[8] J. Werfel, K. Petersen, and R. Nagpal, "Designing Collective Behavior in a Termite-Inspired Robot Construction Team," Science, vol. 343, no. 6172, pp. 754–758, Feb. 2014, doi: 10.1126/science.1245842.

[9] M. Allwright, N. Bhalla, H. El-faham, A. Antoun, C. Pinciroli, and M. Dorigo, "SRoCS: Leveraging Stigmergy on a Multi-robot Construction Platform for Unknown Environments," [Online]. Available: https://carlo.pinciroli.net/pdf/Allwright:ANTS2014.pdf.

[10] M. Allwright, N. Bhalla, C. Pinciroli, and M. Dorigo, "Towards Autonomous Construction using Stigmergic Blocks," Université Libre de Bruxelles, 2017.

[11] C. Wagner et al., "SMAC: Symbiotic Multi-Agent Construction," [Online]. Available: https://arxiv.org/pdf/2010.08473.pdf.