ROBOTIC SCAFFOLDING

MQP Project Final Report - 2018

By: Nathan Beeten, Chris Cormier, Connor Willgress

April 26, 2018

Abstract

Currently, automation is becoming more important to industry, and soon, construction work may be completed by robots. In the construction industry a large amount of resources are used to erect buildings. Increasing automation will help this problem by decreasing the project's time and overall cost. This report presents a design to automate construction using two systems that autonomously build structures. Each system is simple and works with the other to achieve a complex task beyond either's individual ability. The first system is a distributed control and guidance scaffolding that acts as both the physical support and source of instructions. The other is a mobile builder robot that incrementally adds new scaffolding and building blocks to build the required structures.

Executive Summary

Project Overview

The goal of this MQP was to design a foundational system to autonomously build two dimensional structures for future projects to build upon. Our approach combines two different systems to accomplish our complex task. The first is the network of scaffolding blocks that communicates with itself and sends commands to the second system. The builder robot picks and places up scaffolding and building according to these instruction to expand the scaffolding network and create the final structure.

Design

The scaffolding and building blocks externally are both square with male and female guides on each side so that they can interlock with each other and are mechanically constrained. The scaffolding blocks assist in the building process by calculating the optimal path for the builder and then relaying these instructions. The building blocks are hollow so that no expensive parts are left in the completed structure. The scaffolding blocks broadcast messages to the network via a Controller Area Network (CAN) bus and individually send messages to the builder robot when it drives over them via an RGB LED. The builder robot receives these commands using a color sensor and executes them by driving to the correct location and picking up or placing down the corresponding block. The scaffolding blocks keep track of where the builder robot is eliminating the need for it to do its own localization. A variety of embedded programs and desktop applications were created to assist in the development, calibration and testing.

Results and Recommendations

The goal of this project was to create a proof of concept for a system to autonomously create two dimensional structures using builder robots and intelligent scaffolding. Overall the project was successful in being the first step in a series of MQPs. We were able to create a system that can drive around on a test board, grab and attempt to lift our blocks, and have blocks communicate with each other. We have assembled a variety of suggestions for future projects. These suggestions include improving existing systems as well as adding additional functionality and are located in Section 6 of this report.

Table of Contents

Abstract 1

Executive Summary 2

Project Overview 2

Design 2

Results and Recommendations 2

List of Figures and Tables 4

1. Introduction 5

2. Literature Review 6

3. Methodology 8

3.A) Robot and Block Methodology 8

3.A.1) Block Design 8

3.A.2) Builder Robot Design 11

3.B Algorithms Methodology 14

3.B.1. Target Queue 14

3.B.2. Configuration Variants 14

3.B.3. Scaffolding Structure 15

3.B.4. Goal Stack 15

3.B.5. Goal Determination 16

3.B.6 Choosing an Unneeded Block 17

3.B.7. Scaffolding Updates 17

3.B.8. Simulation 17

4. Results & Discussion for Further Work 18

5. Conclusion and Further Work 19

5.A. Mechanical Improvements 19

5.B. Electrical Improvements 19

5.C. Algorithm Improvements 20

6. Bibliography 22

7. Appendix A – CAD for Builder Robot 23

8. Appendix B – CAD for the Blocks 24

List of Figures and Tables

Figure 1: Automatic House 3D Printer 5
Figure 2: Concrete Recycling Robot 5
Figure 3: Demonstration of termite Robots 6
Figure 4: Demonstration of smart blocks 7
Figure 5: First Block Design Iteration 8
Figure 6: Current Block Design Iteration 9
Figure 7: Zoomed in Section Where the Blocks Connect 9
Figure 8: First Design Iterations for Robotic Builder 12
Figure 9: Hoist CAD Design 12
Figure 10: Current Design Iteration for Robotic Builder 13

Table 1: Instruction Types for Block 10
Table 2: Scaffold Microcontroller Pin Assignment 11

1. Introduction

The construction industry is seeing immense growth as it tries to house the 9 billion people that will be on the earth by 2050. Due to extreme growth in the construction industry, problems have arisen. Companies are seeing a plague of unskilled workers with more construction workers being needed to complete various projects. The results of having a lack of skilled workers are construction projects becoming more expensive and having longer schedules. Our approach involves minimizing the workforce needed to build a structure using robotics. Increasing automation will decrease construction costs by eliminating human error and increasing work efficiency.

Figure 1: Automatic House 3D Printer

The idea of automation within the construction industry is not completely new, and we have already seen the automated production of the build materials. This method of off-site or modular construction has been proven to be successful at reducing costs, and experts believe that its use will increase in the coming years [1]. However, we have not seen automation used for the actual assembly of a building. Instead large-scale machines and tools have been the primary option for decreasing construction time. Concepts for automating this process such as a robotic arm that would automate masonry have been introduced. Yet the industry has been to slow to adapt to new technological ideas.

We believe that implementing swarm robotics on a large scale can be used to automate the construction process. Swarm robotics is an ideal tool for this task because it allows many simple robots to complete a task that would normally require a single robot with orders of magnitude more complexity. In our approach we use swarm robotic principles to combine two different systems to accomplish the task. The first is a distributed control and guidance scaffolding that acts as the “brains”. The other is a mobile builder robot that acts as the “muscle”. During operation, the scaffolding acts as both the physical support and source of instructions for the builder. The builder robots

Figure 2: Concrete Recycling Robot

then follow the instructions given by the scaffolding to incrementally add new scaffolding and building blocks to the structure. This process is repeated until the desired structure has been completed.

This project has been limited in scope to only produce 2D structures and only one active building robot. The main goal of this project is to produce a framework that future work can build off and to begin developing the techniques and algorithms that will eventually be used to implement this system on an industrial scale.

2. Literature Review

Within the field of swarm robotics there is generally very little done on research into using it for building complex structures. Most research has been focused on biological nanorobotics, search and rescue, foraging, and mining. The main cause of this being that the companies and industries are more willing to spend money on the research into swarm robotics while the construction industry adapts slowly. Even with the limited research there has been an introduction into the concept of using swarm robotics for building complex structures. Within our research we found two projects to be of use for inspiration as we developed our own project.

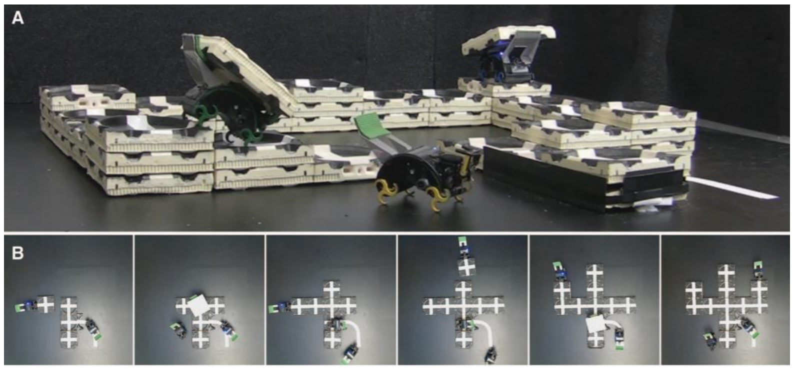


Figure 3: Demonstration of termite Robots

The first research paper our team considered was "Designing Collective Behavior in a Termite-Inspired Robot Construction Team" completed by Justin Werfel, along with Kristin Petersen, and Radhika Nagpal. The project was centered on the idea of engineering an automated construction system that operates by termite-like principles. Mound building Termites (a species of termites) live in mounds made of soil that they build for their colonies. Some termite mounds can be as large as 30 meters (just under 100 feet) in diameter. This is an example of a little insect being able to have built something profoundly large. In general, that is what this project within the research paper was looking to achieve. The systems design is motivated by the goal of relatively simple, independent robots with limited capabilities being able to automatically build a large class of non-trivial structures. They achieve this by having an arbitrary number of independent robots that follow an identical set of simple and local rules. The robots are equipped with hybrid wheels that allow them to climb onto bricks in a stair like manner. The robots are limited to local sensing, able to perceive only bricks and other robots in their immediate sensing. The information about the current state of the overall structure and actions of more distant robots is not available. By using a "seed" block they created a initiation point from which the contiguous structure is built. To ensure the predetermined outcome of construction, robot rules are based on sensed brick configurations plus a static internal representation of the target structure. The strengths of this projects are primarily in the design of the robot builder itself. It was built to be able to traverse upwards and be able to hold onto a building block at the same time. The weaknesses of this project is the computation of its thinking process. It uses a brute force calculation method to determine where and when a block will be placed, this requires long computation times and has no guarantee of being optimal. The robots also require an on-board camera for localization which increases the cost and complexity of the robot.

Another research paper we drew inspiration from is "Toward Autonomous Constriction Using Stigmergic Blocks" put forth by Michael Allwright et. al. Their project uses a multi-robot system to demonstrate a decentralized control strategy for building three-dimensional structures. Like before they were inspired by the same idea of termites being able to build nest many times the size of an individual. Within their project they used autonomous robots that are equipped with a manipulator, which can pick up a stigmergic block and assembling the into structures. The robots utilize onboard computer vision to respond to two types of cues in an environment. The stigmergic blocks are semi-active cubic building material that is programmable by the robots themselves. The blocks can represent different building material by changing the LED colors on the face of the blocks. Within these blocks are eight magnets that the manipulator of the autonomous robot can attach to. The strengths of this project were the "smart" block itself. With the design of the LEDs it makes for transmission of information to the robots easier. It also allows for the displaying the different building materials. Its weaknesses were also revolved around the block as well. The robots relied on the blocks for instructions. The blocks were also extremely costly because of having the ability to communicate via NFC.

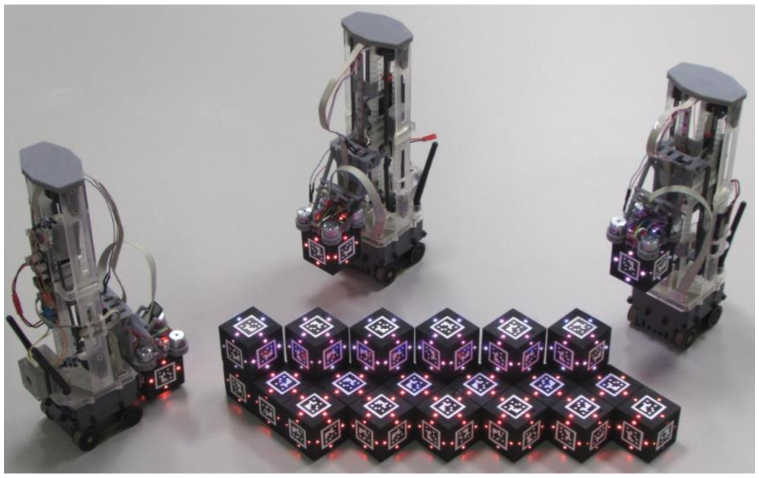


Figure 4: Demonstration of smart blocks

3. Methodology

This section discusses the design of building and scaffolding blocks, builder robot, and algorithms used in our automated construction system. Both block types share the same outward design, so they can interlock with each other and are mechanically constrained. The scaffolding blocks assist in the building process by calculating the optimal path for the builder and then relaying these instructions to the builder robot. The scaffolding blocks mechanically and electronically link together to form a contiguous Controller Area Network (CAN) network. The builder robot traverses solely along the scaffolding structure while receiving commands from the blocks beneath it. These commands control all of the high-level actions of the builder robot, such as moving along the structure, picking up a block, or placing a block.

3.A) Robot and Block Methodology

3.A.1) Block Design

Both the scaffolding and the building blocks share the same external design to reduce complexity. The blocks are square with male and female trapezoidal guides on each side and are rotationally symmetric around the Z axis and bilaterally symmetric across the XY plane. The guides are tapered on the tops and bottoms to allow for some degree of error when placing a block. The blocks also have indentations offset from each edge so that they can be picked up by the robot. The blocks have a 7.6in (19.5cm) square footprint and are 2.3in (5.8cm) tall.

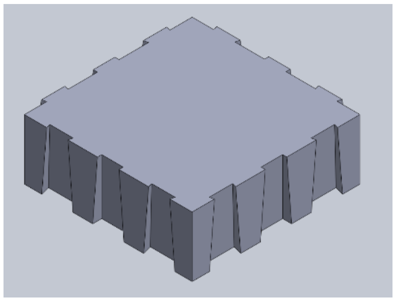


Figure 5: CAD Design of Block (Version 1)

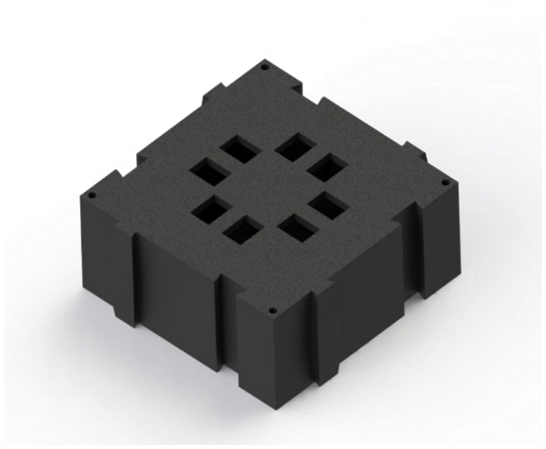


Figure 6: CAD Design of Block (Version 2)

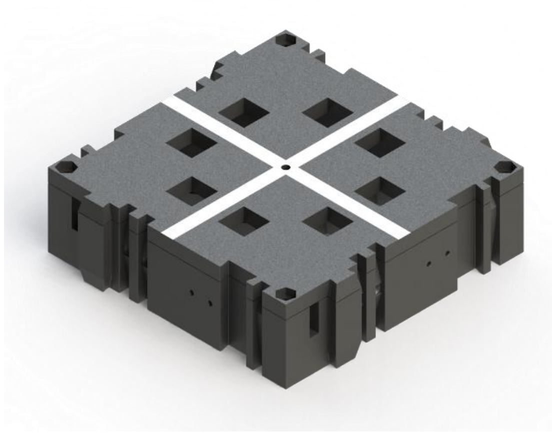


Figure 7: CAD Design of Block (Version 3)

Scaffolding blocks need to be able to detect when a block is placed next to it, communicate between themselves, and signal the builder robot. Block detection is achieved via a simple contact switch on each face of the block while interconnection between blocks is achieved using pogo pins. Both mechanical systems require slight ramps to be embedded in the sides of the blocks to ensure that they connect smoothly, are actually engaged, and that the pins and switches do not bend.

Since messages between the building block and the builder robot are simple and only one-way, an RGB LED was chosen for sending instructions to the Robot. An off the shelf color sensor costs less than ten dollars and RGB LEDS are a few cents apiece making using RGB LEDS for simple commands cheaper and easier to use than wireless protocols. Additionally RGB signaling allows for visual debugging of the blocks. There needs to be support for thirteen commands, four main commands four variants (three of which have four variants) as detailed in Table 1. If the STOP command is designated as the color white, the other 12 commands can easily be assigned a value from 0-359 representing their hue. A simple calibration script was written to determine the optimal hues for the commands as well as the maximum speed the color sensor could operate to minimize errors.

BASE INSTRUCTION TYPE	NO VARIANTS	FORWARD	BACK	LEFT	RIGHT
STOP	✓	X	X	X	X
DRIVE	X	✓	✓	✓	✓
PICK UP	X	✓	✓	✓	✓
PLACE	X	✓	✓	✓	✓

Table 1: Instruction Types for Block

For communication between blocks, a physical protocol was chosen over a wireless protocol due to cost limitations. Wireless transceivers are several times more expensive than their wired counterparts. Additionally, some wireless protocols would require one or more transceivers per side of the block. The CAN protocol was picked over other wired protocols. Can was chosen because of its ability to broadcast messages to all nodes simultaneously, fewer connecting wires, and the availability of robust transceivers. The low-speed implementation of the CAN protocol was chosen because it does not require the CAN bus to be terminated by resistors, instead each node contains the needed resistors. Choosing low speed CAN causes several restrictions not present in high speed CAN, namely a lower speed and the maximum number of nodes is determined by the resistance provided by each node on the bus. Since messages between nodes are small and infrequent, the lower speed is a non-issue. The number of scaffolding blocks is known in advance; therefore, the needed resistance can be calculated beforehand.

The Microcontroller being using for this project is the NUCLEO-L432KC. The NUCLEO boards are a part of the MBED ecosystem which provides real-time OS functionality for embedded boards. This particular board was chosen due to its low price point, while retaining relatively large Flash and SRAM. Additionally, it has built in support for I2C, SPI, and CAN. The tentative port assignments for the board can be found in Table 2.

Port #	Type	Use
D2	CAN TD	CAN TD
D10	CAN RD	CAN RD
D3	PWM	LED R
D5	PWM	LED G
D6	PWM	LED B
D4	DIO	Contact Switch
D9	DIO	Contact Switch
D11	DIO	Contact Switch
D12	DIO	Contact Switch
D10	I2C DATA	Backup I2C Data
A6	I2C CLK	Backup I2C Clock

Table 2: Scaffold Microcontroller Pin Assignment

3.A.2) Builder Robot Design

The builder robot is broken into three mechanical systems: the gripper, the lift, and the drivetrain. The gripper mechanism is the part of the builder that directly interacts with the building and scaffolding blocks. A parallel gripping mechanism, actuated by a single servo motor, allows for the blocks to be grabbed and released with little horizontal movement. The “fingers” of the gripper are designed to mate with the insets on the blocks to ensure a mechanically constrained grip. The servo motor torque requirements are minimally important as the motor only needs to be powerful enough to hold the gripper closed when holding a block: The design of the fingers precludes the need for a frictional hold on the block and the servo motor doesn’t contribute to lifting the block.

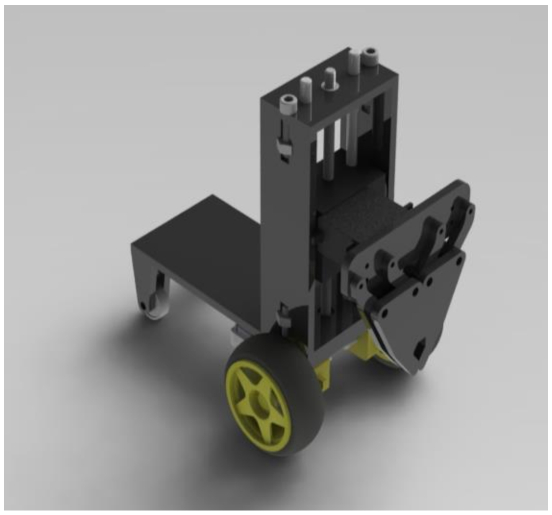
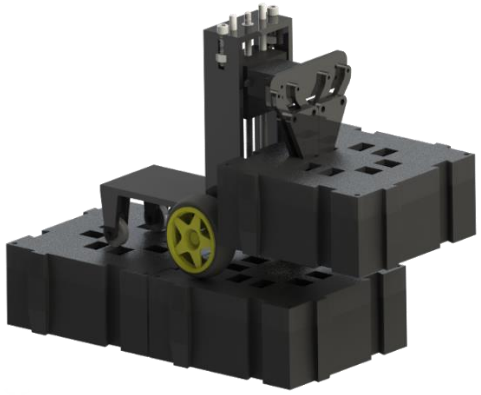



Figure 8: Original CAD Designs of the Builder Robot (Version 1)

The gripper and the block the robot is carrying are lifted by a screw drive powered by a geared DC motor. The screw drive allows for a high torque ratio and purely vertical lift, opposed to the arc that a four-bar mechanism would create. This verticality allows for a block to be removed or placed even when it is surrounded by other blocks. The hoist connects to the gripper via a runner. The runner contacts the hoist at 3 points: a centered hex nut that attaches to the screw and 2 nylon bushings that provide mechanical alignment against the guide rods. In the initial design, all three contact points were in line, but this was changed in later versions to a triangular arrangement to improve stability. The hoist is homed using a contact switch and the raised position determined by the time taken to raise the hoist.

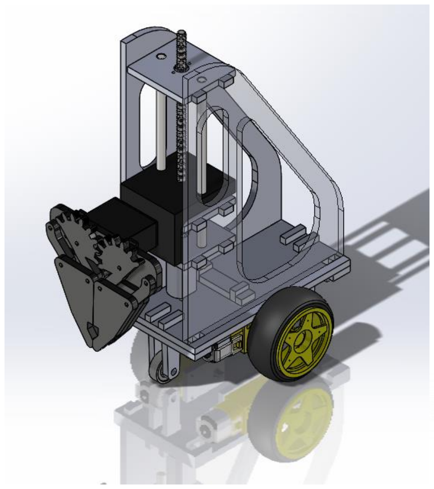
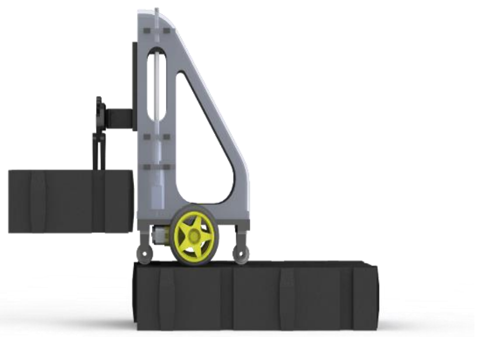



Figure 9: CAD Designs for Robot from 2nd Revision

The builder robot has two driven wheels located at the center of the robot. Initial designs had the driven wheels at the front of the robot to ensure that the wheels-maintained traction when a block was lifted. This was determined to be less important than having the turning center close to the center of the robot. Because our project is not time sensitive, the drivetrain was purchased based on out of the box usability and price. Originally ball casters were placed at the front and the back of the robot to improve stability. During testing, it became clear that the casters were not a feasible solution because they fell into the gripper holes in the tops of the scaffolding blocks. To combat this, the casters were replaced with acrylic skids which were larger than the holes. The skid mounting holes were countersunk to give clearance to the mounting screws. Additionally, a small chamfer was filed into the skid bottoms to account for the small inconsistency in the block's surface.

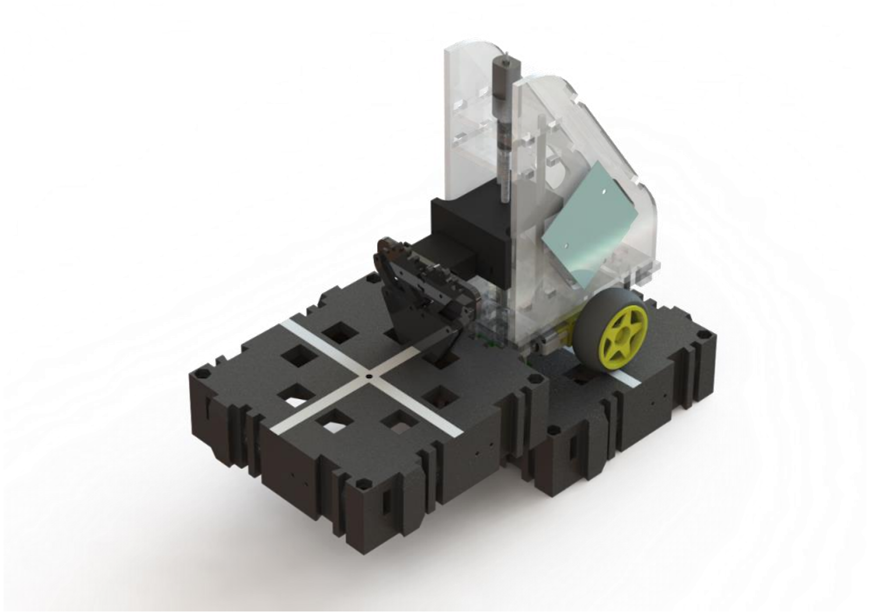


Figure 10: Final CAD Rendering of the Builder Robot

The robot's sensor suite is comprised of two line detection sensors, and a color sensor. The line detection sensors are placed at the front and back of the robot, spaced so that they have the maximum amount of space between them to improve our line sensor resolution. The color sensor is placed at the center of the robot so that the color sensor and the RGB LED are inline when the robot is centered over the scaffolding block.

3.B Algorithms Methodology

This section discusses the high-level algorithmic view of the construction system. The algorithm must be able to take in a target structure and direct the two subsystems to build the structure. The current iteration of the algorithm has two main simplifications: there is be only one builder robot that starts on top of a block and the target structure must be two dimensional. This block is designated as the seed block and is at coordinates (0,0). The seed block will never move and off to one side is the cache. The cache is where the builder will retrieve new blocks and must be manually filled by an outside person.

The Algorithm uses the concept of a spine row and reach columns. The spine row provides a constant line of scaffolding blocks that always exists to the left of the current column. This simplifies the pathfinding needed to reach any given block, by returning to the spine row and going to the appropriate column before going down the reach column to the required row. If a block is unreachable, it can be made reachable by either extending the spine row or the associated reach column

3.B.1. Target Queue

The target structure must be contiguous and is represented as a list of coordinates. Before the giving it to the algorithm, the target structure must be sorted to create the target queue. This target queue specifies the order that the targets will be placed. The algorithm works to place each target in the queue sequentially and does not consider the rest of the target structure. Because of this single mindedness, the target queue must adhere to several specifications (listed in the Configuration section). As long as these specifications are met the target queue can be in any order.

3.B.2. Configuration Variants

The target queue configures before being passed to the algorithm. This configuration process is to sort the queue to meet two requirements. All target locations must be to the right of the origin and the target queue must be sorted from right to left. The first requirement is a basic assumption made to simplify the algorithm. The second requirement is due to the algorithm placing the reach columns for a building block to the left of the building block. Additionally, the queue can manipulate the target queue by translating or rotating the entire structure. This manipulation preserves the overall shape of the target structure but can reduce the amount of time needed for the building process.

There are currently four variants for sorting and translating the target queue: SingleOffsetConfigurer, MidOffsetConfigurer, CentroidConfigurer, and LongestSpineConfigurer. All of these configurers translate then sorts the target structure. SingleOffsetConfigurer translates the structure so that all target locations are in to the right and below the spine. MidOffsetConfigurer translates the structure so that the seed is in the center of the upper and lower bounds of the target. CentroidConfigurer translates the structure so that the seed is in line with the centroid of the target. SingleOffsetConfigurer translates the structure so that the resulting spine will have the most blocks in the target.

Additionally, the target structure can be rotated to increase efficiently. There are four rotators are currently available to rotate the structure in increments of 90 degrees. There are also two more advanced rotators that attempt to rotate the structure to optimally place the centroid.

3.B.3. Scaffolding Structure

The scaffolding structure is comprised of two types of substructure, one spine row, and several reaching columns. The spine row is a continuous line of blocks along the x-axis and is connected to the seed block. The reaching columns are columns of scaffolding blocks used to access a target not adjacent to the spine. Scaffolding is added, moved and removed to in accordance to the goals in the goal stack.

The reach column for the current target will always be to the side of the target in the -x direction. This allows for all the blocks in a column to be placed one after another after the reach column has been built to reach the target furthest from the spine. Working to preserve the spine and reach paradigm allows for several assumptions. First, all scaffolding blocks are reachable. Second if you are not on the same column as another scaffolding block, you can reach it if you travel towards the spine, move to the correct column, and then travel down that reach column.

3.B.4. Goal Stack

The act of placing a building block in a target location is comprised of several steps. The steps that the algorithm is actively considering is stored on a stack called the goal stack. There are four types of goals PLACE_BUILD_BLOCK, PLACE_SCAFFOLD, PICK_BUILD_BLOCK, PICK_SCAFFOLD.

Goals also have two locations associated with them: the location of the block being picked up or placed and the location the builder will be at when completing the goal. These locations must be adjacent with the former being named the goal location, and the second being the helper location. Goals higher on the stack are needed to achieve goals lower on the stack. The goal on the bottom of the stack will always be placing a building block at the current target location.

3.B.5. Goal Determination

The goal stack is updated whenever the top-most goal is completed, when this happens the completed goal is popped off the stack. There are 4 valid ways the current goal can be incompletable. To help complete a goal, another goal is pushed onto the stack. After adding a new goal, the goal stack is reevaluated to see if any more goals are needed.

The first way the target goal is pushed onto the stack when the stack is empty, this occurs when the last root goal was completed. The new goal is the next uncompleted target in the target queue. The helper location for this goal is the block in the -x direction, this is so that the reach column is on the +y side of the target location. In the case when the next target block is next to the spine and is the only one in that column on that side of the spine, the helper location on the spine instead of in the -x direction.

The next way a new goal is added onto the stack is if the current goal is unreachable, that is there is no series of movements that robot can take to reach the helper location of the goal. If this is the case, the goal pushed onto the stack will be to place a block that will help reach the helper location. The location of this goal is the first unoccupied location on the spine or the reach column for the current target. The first unoccupied location is chosen because it is obviously currently reachable. Additionally, we choose a location adjacent to the current scaffolding structure instead of the goal because it reduces the potential size of the goal stack.

The third way a goal is added onto the stack is if the current goal is to place a block and a block is not currently held. If the next goal is to place a building block at the target, the block is always retrieved from the cache. Otherwise, the scaffolding structure is analyzed to determine if any of the current scaffolding blocks are unneeded to reach the next goal. The next goal is to pick up one of these unneeded blocks (see Choosing an Unneeded Block).

The final way that a goal is added is if the current goal is to place a building block at a target location, but the target location already has a scaffolding block. In this case the new goal is to place a new scaffolding in the location needed next (or the cache if it will never be needed). When the goal stack is re-evaluated to determine if a new goal is needed, it will see the new goal and create one to get an unneeded block. The moved block may not be the one blocking the goal, but though many iterations of moving blocks eventually that block will be moved. Moving the blocking scaffolding is done via this roundabout process because it allows all the unneeded blocks to be moved as the goal updates, preventing a block from being left behind far away from the seed.

Since these are the only 4 ways a new goal can be created, it can be shown through a decision tree that at most 3 goals will ever be on the stack at any given moment. This is useful because the algorithm may eventually be run on embedded system of the scaffolding blocks and the bounded number of goals will reduce memory usage.

3.B.6 Choosing an Unneeded Block

The unneeded blocks are any blocks left over from previous reach columns or any blocks on the spine further than the current goal. The unneeded block that is chosen to be moved is the one from the newest reach column that is furthest away from the spine. If this column has the helper location, we also prioritize the side with the helper location. Choosing this block reduces the number of steps needed to retrieve a block, while keeping all blocks reachable.

3.B.7. Scaffolding Updates

The scaffolding blocks instruct the builder robot to move and pick up or place blocks. The instructions must be relative to the orientation of the builder robot because it does not know its global heading. Since the algorithm knows the position of the builder robot at any given point, these instructions can be updated after every time a new goal is pushed onto the goal stack. The default state of the scaffolding blocks is to show the drive forward instruction. At most only four blocks need to show special instructions on any update. The start block, the end block, the on block, and the off block. The start block is the block that the robot is on top of at the time of the update. The end block is the block at the helper location of the current goal. The on block is the block on the spine in the same column as the robot. The off block is the block that the block on the spine in the column of the helper location. Some of these blocks are the same block and in others, not all blocks are needed. If the builder and the end location are in the same column the start block just needs to tell the builder to go towards the end location. In this case, the on and off blocks are not needed. Otherwise, the builder needs to go towards the spine row, move along that row to the column that the helper location is in, and move down that column to the helper location. Obviously if the builder or the helper location are on the spine row, then the movement along the corresponding column is unneeded.

3.B.8. Simulation

A simulation of this algorithm was originally created in python with various example structures. The original simulation can be found on github at https://github.com/cjcormier/roboscaffold_sim. Python 3.6 and tk/tcl must be installed to run the examples. Due to the slowness of Python and the difficulty of extending the application UI with tkinter, the simulation is now written in Kotlin and TornadoFX. The rewrite can be found at https://github.com/cjcormier/roboscaffold-sim-kotlin-jvm.

In addition to the ability to simulate the construction of various structures, the rewritten project provide the ability to compare the strategies based on three metrics number of robot moves, number of scaffolding updates, and amount of scaffolding used.

4. Results & Discussion for Further Work

Throughout the course of this project we were faces with multiple problems that hampered progress. Despite these difficulties we were able to create a prototype capable of achieving many of the individual parts of the system but struggles in some situations.

The final robot is capable of driving around on a test board, but not on top of the scaffolding blocks. This is due to not the robot having enough room on the top of the scaffolding blocks to reposition itself if it was slightly offset. Additionally, if the test board is not level the robot has difficulty driving in some directions due to different wheel speed tuning needed. The different tuning may be circumvented with encoders, which we were not able to use due to the limited space in between the motors being used by the color sensor.

The hoist and runner has trouble picking up even a detached lid of the scaffolding blocks. Part of the issue is that when the runner was manufactured we were not able to fit in the nylon bushings. This allowed for the runner to pitch forward when holding any weight which in turn caused the screw lift to bind. According to calculations done before the motor was selected, the hoist should have been able to lift approximately 4 pounds (the block is under 2 lbs.) so the hoist's failure is most likely due to the binding.

The Scaffolding blocks worked well overall. The pogo pins and copper pads create a solid connection for the CAN network but are difficult to attach to the block. The pogo pin are especially difficult to attach, since a small amount of epoxy on the retractable end causes the pin to be nonfunctional. This difficulty in assembly is compounded by the fact that any significant issue with epoxying the contacts and pins is difficult to undo. This could be resolved by designing the scaffolding block to be more modular.

A significant contributor to our lack of time was due to manufacturing issues. Because the laser cutter was not functioning properly for most of B-Term, we were not able to laser cut the acrylic for the robot and were set back around 5 weeks. Additionally, the print bed for the 3D printer we were using was too small, leading to the issue with repositioning the robot mentioned above.

5. Conclusion and Further Work

The goal of this project was to create a proof of concept for a system to autonomously create two dimensional structures using builder robots and intelligent scaffolding. Overall the project was successful in being the first step in a series of MQPs. We were able to create a system that can drive around on a test board, grab and attempt to lift our blocks, and have blocks communicate with each other. We have assembled a variety of suggestions for future projects. These suggestions include improving existing systems as well as adding additional functionality.

5.A. Mechanical Improvements

Larger Robot & Larger Blocks: Many of the issues we encountered were due to the small size of the robot and blocks. Finding a way to increase the block size would allow the robot to be larger and mitigate some of these issues.

Improve Drivetrain: Currently the robot has difficulty driving sloped surfaces. This could be improved by getting better drive motors or allowing for encoders to detect actual wheel rotation speed.

Improved Hoist and Gripper Mechanism: The robot hoist screw binds when attempting to raise the blocks. This issue may be solved by making sure the nylon bushings work or by using a forklift system to reduce block rotation.

Multiple Structure Layers: The final goal of this series of MQP's is to enable the construction of 3D structures. The builder robot being able to navigate over multiple layers is integral to this goal.

Scaffolding Block Modularity: The scaffolding blocks are currently two parts, the base and the lid. When there is a single issue with adding the CAN pins and contacts, the entire base may unusable. Allowing for individual components of the blocks to be removed reduces the impact that a single mistake has on the manufacturing process.

5.B. Electrical Improvements

Wireless Inter-Block Communication: The current system of pogo-pins creating a mechanical - wired connection works well enough for the current iteration. However, there are problems with using our current CAN system at larger scales. The CAN network is partly reliant on having a total resistance below a threshold, and adding more blocks slowly increases the resistance in the network. Additionally, moving the current system to a 3-dimensional structure would require additional mechanical connections on the tops and bottoms of the block. This could be prevented by adding wireless communication between blocks.

Two Way Scaffold-Builder Communication: To manage the scope of the project we kept both the robot and the scaffolding as simple as possible. However, having feedback between the robot and the scaffolding block could provide valuable additional data for system control. For example, the robot does not currently know whether a block has been properly connected to the network; this could be fixed with additional feedback between the actors.

Decrease Impact of Checking Line Sensors: Due to the way the line sensors function, it takes approximately 6ms total to get readings for both sensors. This can impact the performance if the line sensors are checked in tight loops. This time can be reduced at the cost of precision. Since the sensors are constantly being checked during this time, threading this process would not reduce the impact of reading the sensors. Offloading this processing to another processor to would eliminate most of the impact of frequently checking the line sensors. Additionally, this processing is only necessary for the digital version of these sensors, using the analog version would simply either require more ADC ports or analog multiplexers.

Vision Based Location Detection: Currently the line sensors and color sensor are used to approximate the location of the robot on top of the block, this can cause some issues when the robot is drastically misaligned. A solution to this is to use a small camera mounted to the bottom of the robot. While possibly increasing the accuracy of localization, it would also vastly increase the processing needed to determine the robot's position.

Create PCB for Scaffolding and Robot: The current wiring for the scaffolding and robots is accomplished using breadboards and plugging wires directly into the microprocessor. While this worked well enough for this prototyping phase, increasing the consistency of the manufactured blocks will improve the speed at which they can be made.

5.C. Algorithm Improvements

More Complex Algorithms: The current algorithm is relatively simplistic, research into more complex algorithms may decrease construction times. One suggestion is to search through the decision space to determine the path with the least number of steps. A less computationally taxing, approach is to simply improve edge cases for the existing algorithm.

Dynamic and Distributed processing: Currently the system uses a predetermined plan for the robot to build the structure. Allowing for the scaffolding blocks to detect errors in the building process and adapt to these issues is necessary for the system to more realistically simulate real world conditions.

Multiple Robots: Determining concepts to make full use of multiple robots would decrease construction times. This may simply be intelligently dividing up the structure into substructures and having multiple robots work separately. Another alternative is to have the robots work together and around each other on the same structure. This could be done by making loops that the robots always maneuver around in the same direction, avoiding the issue of the robots needing to drive onto the same block.

Reduce Invalid Structures During Analysis: To test an algorithm we run that algorithm through all unique valid structures within given dimensions. Currently a minor amount of culling is done to remove structures that are simply translated within the maximum space and reduce the number of invalid shapes. For. a 5x5 square, the current process checks 27964666 structures for validity. Of the structure checks only 7.4% are valid structures. Reducing the number of invalid check would increase the analysis speed.

Monte Carlo analysis for larger structures: Since the number of valid structures grow at 2^(n^2), where n is the edge length of the bounding square, the time to test all structures of a given size increases rapidly. To reduce this time, using a large random sample of structures to perform the analysis for larger sizes would increase analysis speed.

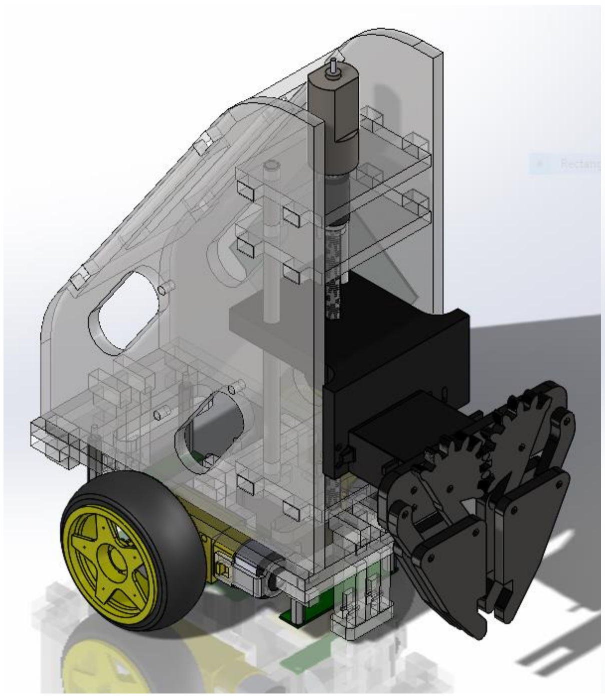
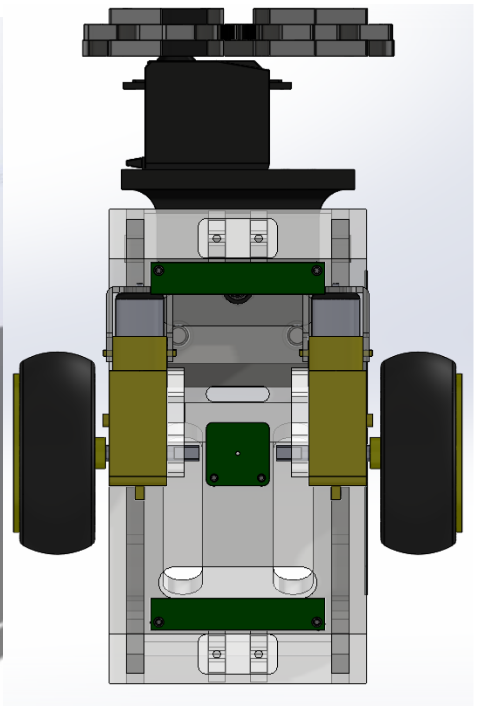
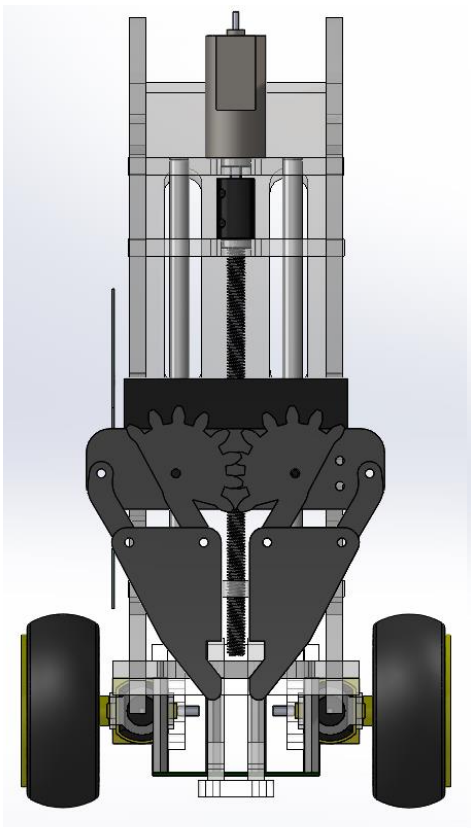
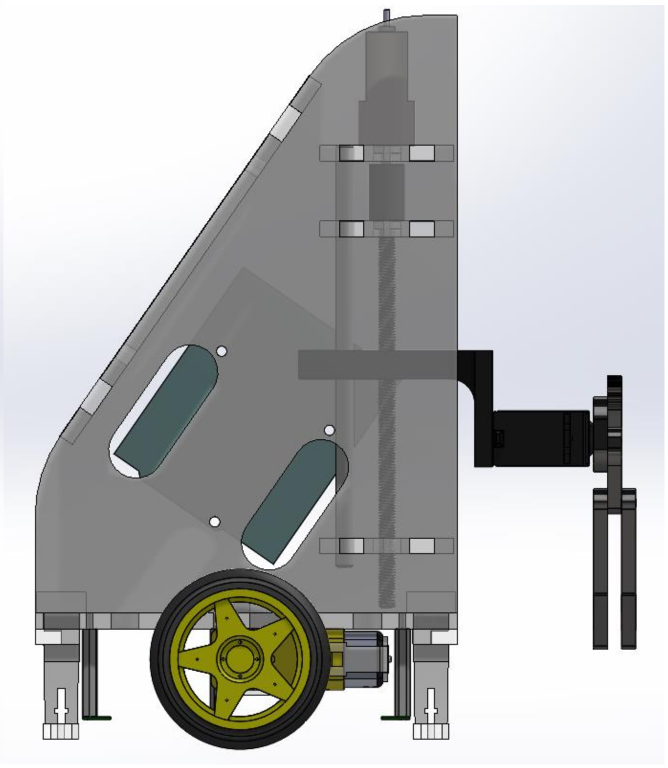
6. Bibliography

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

[2] 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.

[3] “10 construction industry trends to watch in 2017,” *Construction Dive*. [Online]. Available: https://www.constructiondive.com/news/construction-industry-trends-2017/433151/. [Accessed: 30-Oct-2017].

7. Appendix A – CAD for Builder Robot

8. Appendix B – CAD for the Blocks