SMAC: Symbiotic Multi-Agent Construction

Caleb Wagner*1, Neel Dhanaraj*1, Trevor Rizzo1, Josue Contreras1, Hannan Liang1, Gregory Lewin1, Carlo Pinciroli1

Abstract—We present a novel concept of a heterogeneous, distributed platform for autonomous 3D construction. The platform is composed of two types of robots acting in a coordinated and complementary fashion: (i) A collection of communicating smart construction blocks behaving as a form of growable smart matter, and capable of planning and monitoring their own state and the construction progress; and (ii) A team of inchworm-shaped builder robots designed to navigate and modify the 3D structure, following the guidance of the smart blocks. We describe the design of the hardware and introduce algorithms for navigation and construction that support a wide class of 3D structures. We demonstrate the capabilities of our concept and characterize its performance through simulations and real-robot experiments.

I. INTRODUCTION

Multi-robot systems promise solutions for construction at much larger scales than the robots themselves. With many robots working in parallel, construction can be completed in a fast and efficient manner. Decentralization might also result in a robust solution with no single point of failure, paving the way for uncrewed construction of colonies in adverse environments such as the Moon or Mars.

The design space of possible approaches to collective construction is vast, and it encompasses a careful co-design of both hardware, software, and fabrication [1]. A particularly important open problem is how to coordinate the construction process autonomously. In this paper, we frame this problem as the study of how intelligence should be distributed across the system.

To answer this research question, we explore a novel concept for multi-robot construction that comprises two types of robots. The first is a collection of smart construction blocks behaving as a form of growing smart matter. By communicating, the smart blocks monitor the state of the structure and its construction progress in a decentralized fashion. The second type of robot is the builder, unaware of the global target structure but able to manipulate and transport blocks in 3D thanks to its inchworm-shaped body. Builders perform localization and path planning by communicating with the smart blocks, which perform the necessary computation. The two types of robots constitute a symbiotic system in that their continuous collaboration is necessary for success. For this reason, we called our approach SMAC — Symbiotic Multi-Agent Construction.

∗C. Wagner and N. Dhanaraj contributed equally to this work.
1C. Wagner, N. Dhanaraj, T. Rizzo, J. Contreras, B. Liang, G. Lewin and C. Pinciroli are with the Department of Robotics Engineering, Worcester Polytechnic Institute, MA 01609, USA. {cwagner, ndhanaraj, tarizzo, hliang2, jdcontrerasalbuj, glewin, cpincirol}@wpi.edu.

Fig. 1: Overview of proposed concept.

We envision our system as a technology for in-orbit or planetary construction. In addition, while the smart blocks could be used as actual construction material, they would also be suitable as smart scaffolding [2]. The latter idea would enable reusable, cost-effective technology for remote, large-scale, autonomous construction.

Related work. The idea of performing construction in which a robot interacts with the environment is inspired by stigmergy in social insects [3]. Several approaches have been proposed in this context, both in simulation [4], [5], [6], [7] and with real robotic platforms [8], [9], [10], [11]. In particular, SROCS introduces the concept of smart blocks capable of communicating with a builder robot, and of storing and retrieving information. SMAC differs from SROCS in two crucial ways: (i) in SMAC, smart blocks communicate with each other, acting as an intelligent swarm; (ii) in SROCS, the builder robot is a crane incapable of traversing the ongoing structure, making it impossible to build structures beyond the builder’s own size. The problem of navigating and localizing within an ongoing structure has been the subject of several influential works. TERMES [12] pioneered a concept in which a wheg-equipped robot, co-designed with a convex block, constructs 2.5D structures by block deposition and removal. The robot follows a predefined plan calculated by an optimization algorithm [13], with limited capabilities to recover from placing errors. Bill-E [14] is a bipedal, inchworm-shaped robot designed to navigate and build versatile 3D structures composed of special blocks. Both TERMES and Bill-E handle passive blocks, and localization is achieved by tracking the number of blocks covered since the entry point of the structure.

Novelty and contributions. To the best of our knowledge, this work is the first to explore the co-design space of collective construction with symbiotic robot teams comprising growing active matter and mobile robots. In this domain, our paper provides four main contributions:

1) We propose the co-design of two types of robots, both of which present novel challenges and solutions;
2) We introduce planning algorithms for both types of robots that achieve 3D construction and navigation;
3) We characterize the class of structures that are achievable through our system;
4) We demonstrate the effectiveness of our approach through simulations (particularly regarding scalability) and real-world experiments (to validate the hardware design).

The paper is organized as follows. In Sec. II we list the main requirements. In Sec. III we illustrate the hardware design, and in Sec. IV we describe our algorithms for planning and navigation. We report the results of the evaluation of our approach in Sec. V and conclude the paper in Sec. VI.

II. PROBLEM STATEMENT

SMAC is a multi-robot construction platform designed with the intent of exploring novel questions regarding large-scale, multi-robot, 3D construction. In particular, we aim to separate the planning/monitoring logic involved with multi-robot construction from the physical aspects of construction, such as navigation and block placement. This will enable the study of novel research questions centered around the concept of “active” stigmergy, in which simple, reactive robots build smart structures composed of modules capable of dynamic monitoring and (re)planning.

For the purposes of this work, we consider three high-level design requirements to achieve this vision:

1) We want our builders to navigate over challenging 3D structures. These robots must also be capable of manipulation/placement of building blocks in any part of the structure.
2) We want our smart blocks to locally compute and communicate information pertinent to the ongoing construction task. Communication must occur both among blocks and between a block and an attached inchworm. This will enable the robots and blocks to exert an attachment force onto the structure, we used an actuated threaded screw. This interface is shown in Fig. 2.
3) Finally, we aim to demonstrate the potential of this kind of distributed, spatial intelligence by devising algorithms that achieve construction in an efficient and scalable manner.

III. ROBOT PLATFORMS

A. Inchworm Builder and Smart Block Co-Design

We start our presentation by defining the parameters and decisions that dictate the inchworm and block co-design.

Smart block side length. The first design parameter is the side length \( L \) of the smart blocks, which we consider the “unit” of the lattice constituting any target structure. A large side length would allow for better electronics and mechanical capabilities for a block, but it would also increase the size and weight of the inchworm. A short length, on the other hand, would allow for a smaller, lighter inchworm, while limiting the capabilities of the smart block.

Inchworm-block docking interface. The docking interface between an inchworm and a block face dictates how the inchworm climbs a structure and manipulates a block, as well as how the blocks assemble to become a structure. We selected a 4-dowel pin-to-hole design for two reasons:

(i) This type of interface can sustain sufficient shear loads; and (ii) it fosters alignment, which is beneficial for both inchworms and newly placed blocks onto the structure. For the robots and blocks to exert an attachment force onto the structure, we used an actuated threaded screw. This interface is shown in Fig. 2.

Inchworm-block communication. The communication interface between an inchworm and a block occurs between the inchworm feet and any face of a block. This allows the inchworm to both communicate and localize itself on the structure without ambiguity.

B. Inchworm Builders

In the literature, robots such as whegged [12], aerial [15], quadrupedal [16], and bipedal robots [14] have demonstrated the traversal of simple 3D structures to place building material with varying levels of dexterity and control complexity [1]. In the quest for a versatile design with low control complexity, we identified an inchworm-shaped platform as a satisfactory solution [14]. An inchworm can traverse over almost all block structures. Furthermore, an inchworm robot can pick and place blocks onto structures at any position and orientation in their workspace. This capability does not compromise other important factors such as weight and cost.

Inchworm design summary. Our inchworm robot is formally defined as a symmetric, bipedal 6-link serial-linkage robot with 5 rotational joints (see Fig. 3). Two of these joints are wrist joints located at each end effector, which enables the inchworm to manipulate the blocks’ positions and orientation along 2 axes of rotation and traverse 3D structures.

Link design and analysis. We derived a relationship between the required link lengths and the fundamental inchworm motion configurations to ensure versatile 3D navigation. Fig. 4 reports the fundamental motions that we considered. For this analysis, the side view of the inchworm allows us to simplify its analysis by considering a 4-link robot with links labeled A through D. The block side length \( L \) affects the lengths of links B and C. The minimum link length is constrained by the robot’s ability to navigate a convex corner without collision as shown on the right of Fig. 4. To determine a relationship between the inchworm link length and block side length \( L \), we expressed the link lengths...
as vector loop equations [17]. To solve these equations in the convex corner configuration, we set the length of the A and D links to 1.375\(L\) and the clearance vector between the middle joint and the block corner to 0.25\(L\). By solving the vector loop equations, the relationship between the block side length, \(L\), and the inchworm link lengths are expressed by \(L_A = L_D = 1.375L\) and \(L_B = L_C = 1.506L\). Furthermore, if we define a worst-case scenario as a static fully extended robot that is perpendicular to the direction of gravity, as the robot becomes longer and heavier, the required joint torque increases quadratically. Based on budget considerations and a review of available customer off-the-shelf actuator options, the standard form factor servo was the strongest candidate for joint actuators. We therefore defined the block side length as \(L = 3\) in and the current inchworm prototype is realized using servo motors. The link lengths of the inchworm were determined based on the above analysis and are reported in Fig. 3. Preliminary testing of the inchworm validated that these link lengths are large enough to enable the inchworm to fully enclose the necessary electromechanical components.

**Motor selection.** To specify a suitable motor, we calculated the dynamic torques for the worst-case scenario where each joint has a positive instantaneous velocity and acceleration. For this calculation, we considered a joint speed of 30°/s, while the maximum acceleration was determined from a trapezoidal velocity profile [18]. Using a joint mass dynamic model of the robot, the worst-case dynamic joint torque resulted in 17.5 kg-cm. We chose a standard-form-factor JX PDI-HV5932mg 30kg-cm servo as it provides a safety factor of 1.5 based on the worst dynamical joint torque. We also added high-resolution absolute position magnetic encoders to the joints.

**Gripper end effector design.** The end effector design, depicted in Fig. 2 consists of a flat disk with four, half-inch dowel pins integrated into the surface, as seen on the right of Fig. 2. These dowel pins absorb the shear loads on the robot-to-structure interface and align the end effector with the block. We use a single, actuated 4-40 threaded rod to screw into the structure and absorb the normal load acting on the interface. The rod and pin ensure that the overall connection can sustain the loads required to be attached to the structure. Based on the dynamic model of the inchworm, the maximum torque between the inchworm and structure was calculated to be 26.21 kg-cm during fast movements. The maximum expected load on the screw is 67.6 N, which is well within the proof load of the gripper screw. In order to attach a block, the gripper has an actuated Allen key that is used to turn the block screwing mechanism. Lastly, each gripper has a Near-Field Communication (NFC) antenna that is used to communicate with the structure. NFC is discussed in more detail in Sec. III-C.

**Electronic design.** The inchworm uses a Teensy 3.6 micro controller and Raspberry Pi Zero for low-level control and joint trajectory planning. The AS5048B magnetic encoders were chosen for the joint sensor because of their compact size. These encoders provide a 14-bit joint angle measurement with a 0.02° resolution. Lastly, the current prototype of the robot is powered by a 9 volt power supply instead of a battery for simplicity in conducting experiments.

**Inchworm behaviors.** We structured the behaviors of the inchworm into a behavior tree [19]. We found that this approach fits well with the reactive nature of the inchworm behavior, while being simple and extendable. The main subtrees involve (i) receiving updates from the structure; (ii) waiting (do nothing); (iii) moving to a specified location; and (iv) building sections of the structure. Each of these subtrees is composed of lower-level subtrees that encode simpler behaviors such as picking up blocks, pathfinding, and placing blocks. With reference to Fig. 5 the behaviors are executed top to bottom, left to right.

### C. Smart Blocks

To address the second design goal in Sec. III our smart blocks need to physically attach to other blocks and communicate with nearby blocks and inchworms. This way, blocks can know the current status of a structure and detect failed or detached blocks. Furthermore, an inchworm can...
directly communicate with a structure for localization and to determine the status of the structure.

**Communication.** For communication to be possible across the 6 faces of a smart block, we considered multiple methods including mesh networks, Universal Asynchronous Receiver/Transmitter (UART) and NFC. We settled on an NFC-based approach with six NXP PN532 NFC controllers and antennas, one for each face, which are all controlled by an Arduino Mini microcontroller. For two separate block faces to communicate over NFC, one NFC controller must be in card emulation mode and one in card reader mode. For a block to receive messages, the NFC controller goes into card emulation mode and the microcontroller polls each block face for an incoming message. For a block to send a message, the NFC controller goes into card reader mode and the message is relayed to the receiving NFC face. The microcontroller keeps the NFC block faces in card reader mode until the message transmission has been confirmed. As shown in Fig. 6, the blocks communicate through a flooding algorithm in which every block broadcasts messages on all its faces.

**Inchworm-block connections.** Each block has five female connection faces and one male connection face. The male connection face uses a 4-40 threaded screw, like the robot gripper, to attach to a structure. The top female face has an Allen key input that the inchworm uses to operate the block screwing mechanism. This is shown in Fig. 2.

**Power and LEDs.** Each block has a 600 mAh battery which provides an average battery life of about 4 hours. Each face on a block has programmable Neopixel LEDs. This feature is used to convey the state of the block to the user. For example, a green block is incorporated into a structure, a flashing yellow block is waiting to be added to the structure, and a flashing red block is visually alerting the user that an adjacent block has been removed or failed.

**Home blocks.** A variation of the smart block, the home block, serves as the structure seed and the initial planner for the entire system. Multiple home blocks can be deployed to parallelize construction starting from several locations. The home block is equipped with a Raspberry Pi Zero.

### D. Structure Perception Functionality

Three key functions leverage the symbiosis between the blocks and the inchworms:

1) **Structure Status:** The smart blocks update the structure’s current configuration as blocks are dynamically added.
2) **Robot Position:** Each inchworm robot requests its own position and the position of other robots with respect to the structure, enabling localization in 3D space.
3) **Heartbeat Protocol:** The structure can identify if a block is missing or has failed by regularly updating the structure’s current configuration and comparing it to the previous configuration. This is performed by checking the status of neighboring blocks.

---

**Fig. 6:** Structure made of smart blocks showing the flooding communication strategy.

**Fig. 7:** Examples of steps of building algorithm.

---

### IV. PLANNING AND NAVIGATION

To fulfill the third requirement, we devised algorithms for planning and navigation that take advantage of the hardware design presented in Sec. III.

#### A. Planning

The planning algorithm (see Fig. 8), is responsible for calculating the sequence of operations that must be accomplished to build the target structure. These operations include retrieving and placing blocks in the locations indicated in the blueprint of the target structure. The planning algorithm is executed by the home block (or blocks, in case multiple home blocks are deployed).

**Divisions.** Because navigation is a time-consuming activity for the inchworms, the planning algorithm splits the structure into divisions. Each inchworm claims a specific division. Inside of a division, the inchworm receives instructions from and communicates to the planner through the smart blocks. For example, the home block informs the inchworms that a new block is available for placement.
**Ferrying.** When inchworms are notified that a new block is available for placement, the inchworm in the nearest division picks it up and either places it in its own division, or it brings it to the border of the division nearest to the target location. The inchworm responsible for that division picks up the block and repeats the above procedure. We call *ferrying* the act of passing blocks across divisions until the target division is reached. Ferrying, akin to a bucket-brigade, is a form of task partitioning [20], [21] that lowers the navigation cost of the inchworms. The planner provides instructions to the inchworms by changing the state of blocks.

**Division creation.** Divisions are created by subdividing a given structure into areas where an inchworm is meant to perform work, as shown in Fig. 7 (division creation). While the divisions extend to 3D shapes, only a layer of the structure, defined as a group of blocks 1 unit high, is considered at a time. The size and shape of divisions can vary, but ultimately they are meant to minimize the amount of inching the inchworms must perform to either ferry or build blocks. Next, a partial ordering of the divisions is established as a means to determine the order at which divisions should be built. This partial ordering is used to construct a priority queue. An assumption made as part of the system is that a location on the structure, designated as the feeding location, would be where a human operator would introduce new blocks into the system. For simplicity, the system uses a single feeding location at the home block. The partial ordering is determined as the relative distance from the feeding location and calculated using the wavefront algorithm. Divisions that are closer to the feeding location are built first. Once these divisions are built, the inchworms traverse the built divisions to ferry blocks to divisions farther away from the feeding location.

**Division claiming.** Once the partial ordering is established for a given layer of the structure, the inchworm robots are tasked with dispersing themselves into the different divisions, as shown in Fig. 7 (division claiming). The divisions are considered in the order of the priority queue. Using the communication capabilities of the smart blocks, each inchworm calculates its position in the different divisions. Each inchworm then communicates the nearest division and the distance to that division. The distance to the division is considered the amount the inchworm is bidding for a division, such that a larger distance corresponds to a higher vote. The inchworm with the highest bid (largest distance) claims the division, and the process is repeated until each inchworm has claimed a division. This process is done such that each inchworm minimizes the distance it must travel (inch) to reach a division.

**Block placement.** After the inchworms are distributed, the operator introduces new smart blocks into the system, which signal to the inchworms that there are new blocks to move or build. If an inchworm is in a division that has already been built, it passes the block (ferry) to another division, as shown in Fig. 7 (ferrying behavior). If the division has not been built, the inchworms query the planner for the blueprint of that division and place the blocks accordingly, as shown in Fig. 7 (building behavior). When moving blocks, whether ferrying or building, the inchworms place blocks in a spiral fashion, starting from the center of the division and moving outwards. This was chosen so that after the first block is placed at the center of the division, the inchworm can climb onto the block and act as a fixed-base robot arm, pivoting around the block to place additional blocks. If the shape and size of divisions prevent the inchworm from reaching each location, the robot will inch to place a block. When ferrying, the inchworms place blocks towards the outer edges of their own division such that a robot in an adjacent division has to extend as little as possible into the initial division, thus minimizing the chances of collision or periods where an inchworm must wait for another inchworms to move.

**Reactive replanning.** Once the inchworms have finished building or ferrying a division, the inchworms will disperse themselves again and repeat the process of either building or ferrying blocks. This minimizes the time the inchworms spend waiting for other inchworms to finish. The procedure of dividing a layer into divisions, dispersing the inchworms, and then building or ferrying blocks through the divisions is repeated for each layer in the structure until the entire structure is built.

**Achievable structures.** Based off of the manner in which the planner performs construction, it is possible to characterize the types of buildings that can be built using our algorithm. Specifically, our algorithm can build any structure that does not present upside-down L-shape overhangs. This can be informally explained by noting that the building algorithm proceeds in a bottom-up (positive z direction) manner, such that it will never remove a block from a layer below one that has already been built. Thus, our building algorithm is capable of building structures with overhangs, excluding those with overhangs that extend into the negative z direction, characterized as an upside down L-shape. We wish to stress that this limitation concerns the building algorithm and not the hardware capabilities of the robots. Removing this algorithmic limitation is left as future work.

**B. Navigation**

In order to reach any given location on the structure, the inchworm robots must be capable of producing a traversable path to the given location.

**Face*.** We developed a simple path planning algorithm, called *Face*, that takes advantage of the versatile mobility of the robots. Face* is an extension of the A* algorithm that considers the various legal motions of the inchworms, such as the fact that it can climb vertically and upside-down, while minimizing the amount of inching performed. The algorithm works by considering the faces of each block as nodes in a network, and using the Manhattan distance to each node as the cost associated with reaching that node. Analogously to the A* algorithm, Face* selects the node with the lowest associated cost. For each node, an inverse kinematics simulation is performed to ensure no collisions with the structure are possible. The collision detection analysis is only run three times for each node to simplify complexity: once when the
robot has disengaged from the current node, once when the robot has moved above the goal node, and once when the robot has engaged with the goal node. If, at any point in this process, a node is unreachable or causes a collision, the node is removed and the next lowest cost node is considered. Only when a complete, collision-free, traversable path is created does the algorithm return a successful path. The pseudocode of Face\* is shown in Alg. 1.

It is important to note that a path is only generated for the end effector closest to the goal, not for both end effectors. The second end effector does not require a separate path to be generated as it can follow in the footsteps of the first end effector. Also note that if an end effector is not engaged to the structure, Face\* first considers if the already disengaged end effector is capable of reaching the given position. This allows the inchworm to swing in place like a fixed-base robot arm and reduce unnecessary inching.

V. EXPERIMENTAL EVALUATION

To gauge the performance of our approach, we ran both simulated and real-world experiments.

A. Simulated Experiments

We developed a custom simulation environment to visualize and collect statistics of the system. The simulation environment was built using the Visualization Tool Kit (VTK). The experiments were run on a 2015 MacBook Pro.

Metrics. The performance metrics we considered for the simulated experiments are (i) whether the construction process succeeded; and (ii) the speed at which the structure was constructed, measured in simulated time steps.

Setup. We tested our approach with diverse structures including pyramids, temples, churches, castles, the Empire State Building, and the Star Trek Reliant ship, among others (see Fig. 9). We compared the construction speed across different numbers of inchworms to investigate the effect of adding additional robots on construction time. In the simulation experiments, inchworms were randomly placed on a surface composed of smart blocks. The inchworms were then tasked with distributing themselves according to the distribution algorithm explained in Sec. IV-A. After the inchworms had distributed themselves, a virtual operator sets the distribution algorithm explained in Sec. IV-A. After the inchworms had distributed themselves, a virtual operator sets the distribution. Prompted by the introduction of a new block into the system, the inchworms reacted by moving the blocks throughout the system as explained in Sec. IV. For all simulations, the inchworms were able to place all blocks as required.

Scalability study. The results of two experiments are reported in Fig. 10. The experiments involve 1 to 4 robots tasked with building a 10 × 10 plane and a pyramid consisting of 316 blocks. As shown in Fig. 10, the time decreases with

| Algorithm 1 Inchworm path planning for navigation |
|-------------------------------------------------|
| 1: `procedure FACE*(structure, start, goal)` |
| 2: \( \text{frontier} \leftarrow \text{PriorityQueue}((\text{start}, 0)) \) |
| 3: \( \text{cameFrom} \leftarrow \{(\text{None}, 0)\} \) |
| 4: `while not frontier.empty() do` |
| 5: \( \text{current} \leftarrow \text{frontier.get()} \) |
| 6: `if current == goal then` |
| 7: \( \text{break} \) |
| 8: `end if` |
| 9: `for next in structure.neighbors(current) do` |
| 10: \( \text{newCost} \leftarrow \text{cameFrom}[\text{current}[1]] + \text{structure.cost}(\text{current}, \text{next}) \) |
| 11: `if next not in \text{cameFrom} or \text{cameFrom}[\text{current}[1]] < \text{current}[\text{next}]` |
| 12: `if structure.isReachableAndCollisionFree(next) then` |
| 13: \( \text{cameFrom}[\text{next}[1]] \leftarrow (\text{current}, \text{newCost}) \) |
| 14: \( \text{priority} \leftarrow \text{newCost} + \text{heuristic}(\text{goal}, \text{next}) \) |
| 15: `frontier.put(next, priority)` |
| 16: `end if` |
| 17: `end if` |
| 18: `end for` |
| 19: `end while` |
| 20: \( \text{return path} \) |
| 21: `end procedure` |

![Fig. 8: Sequence diagram of building algorithm for structure and N number of inchworms that is repeated for every layer within the given structure.](https://vtk.org/)
additional robots, especially when comparing a single robot and four robots. The results appear to follow a sublinear trend, which we approximated with a second-order polynomial curve.

**Task specialization.** To analyze how the collaboration between inchworms affects performance, we recorded the total amount of time spent in each behavior by the inchworms. The main behaviors tracked were receiving updates from other robots and the structure (Update), moving (Move), waiting (Wait), building (Build), and ferrying (Ferry). The results shown in Fig. [11] refer to the same structures considered in the scalability study. For the single inchworm experiment, the inchworm spends almost all its time either ferrying or building. Almost no time is spent waiting, and a very small portion of time is spent receiving updates. This is expected, as there are no other inchworms with which it must coordinate. For the experiments with multiple robots, we observed that the time spent across the different behaviors is very heterogeneous. Although capable of exhibiting all the behaviors, the inchworms specialized themselves into different groups, in which some of the inchworms focus more on ferrying while others focus more on building. In particular, we observed that one inchworm did most of the ferrying, while the other inchworms mainly focused on the building. This task specialization is an interesting emergent property of our approach whose appearance depends on the type of structure to build. We also observed that the waiting time and the time spent receiving updates of all inchworms is much higher compared to the single inchworm experiment. This is expected, as by increasing the number of inchworms, maintaining coordination demands communication between the inchworms and the smart blocks. Samples of these tests are reported in the video at [https://youtu.be/zLDa8vK0HpM](https://youtu.be/zLDa8vK0HpM)

### B. Real-world Experiments

The goal of the real-world experiments was to validate the hardware, as well as to investigate whether some of the results found in the simulation would be consistent in the real world. We constructed one inchworm and five smart blocks.

**Unit testing.** We performed a successful campaign of unit tests to validate the fundamental motions of the inchworm. We also tested the smart blocks to detect when new blocks were added or removed, or when a block went offline, which could be attributed to a fault, dead battery, or other physical error. The tests were all successful. Samples of these tests are reported in the video at [https://youtu.be/zLDa8vK0HpM](https://youtu.be/zLDa8vK0HpM)

**Inching precision test.** We validated the precision of the inchworm to estimate how often an inchworm may fail. The inchworm was made to take an inching step and retract 10 times. This experiment was also replicated while an inchworm was taking a step with a block. It was found that the inchworm was able to successfully step and grip 100% of the time and step while carrying a block 80% of the time. The results from the second test are attributed to the disturbance the block applies on the inchworm which can cause occasional inching errors.

**Inching and gripping test.** We also tested the capability of our inchworm to take three steps on a plane while carrying a block. We estimated the inching speed to be 1 step every 40 s. The majority of this time came from the grippers, with about 20 s on average for engaging with the structure and about 10 s for disengaging with the structure.

**Inchworm system test.** A complete system-level test was also performed to demonstrate that the inchworm is able to inch, traverse a structure, manipulate a block, and place it on the structure. A series of snapshots from the demonstration are shown in Fig.[12] and the footage is available in the video at [https://youtu.be/zLDa8vK0HpM](https://youtu.be/zLDa8vK0HpM)

**Smart block communication.** We performed a series of tests to validate that a new block is detected when it is added into the structure, and that the structure can detect when a block is removed from the structure. The images in Fig.[13] show that the blocks change the color of their LEDs to convey the change in the structure. Experimentally, the

| Number of robots | Build (1 Robot) | Build (2 Robots) | Build (3 Robots) | Build (4 Robots) |
|-----------------|----------------|-----------------|-----------------|-----------------|
| Time to Build Plane (100 Blocks) | 40000 | 60000 | 10000 | 12500 |
| Time to Build Pyramid (316 Blocks) | 80000 | 5000 | 7500 | 5000 |

Fig. 10: Time required to build structures for different numbers of inchworms.

![Time spent performing specific behaviors for different numbers of robots.](https://example.com)

![Inching and gripping test](https://example.com)

![Inchworm system test](https://example.com)
average time for two smart blocks to exchange a 62-byte package was 500 ms, which corresponds to an average bit rate of 992 bit/s. We also estimated the impact of a large structure on communication latency. The number of hops between any block and the home block is given by the Manhattan distance between them. A message originating from block at \((x, y, z)\) of a 3D structure must traverse \(x + y + z\) blocks to reach a home block at \((0, 0, 0)\). To put this in perspective, a message sent by a block placed in the corner of a \(10 \times 10 \times 10\) cube would take 15 s to reach the opposite diagonal corner. This has important implications when building large structures, motivating the need for multiple home blocks to decrease this latency. We will study these aspects in future work.

VI. CONCLUSIONS AND FUTURE WORK

This paper proposes SMAC, a robotic platform for collective construction composed of two types of robot: (i) a collection of smart blocks, forming an intelligent network capable of dynamic planning and monitoring; and (ii) an inchworm-shaped, builder robot, able to navigate a 3D structure and deposit blocks under the guidance of the smart blocks. We presented the hardware design of our platform, along with a set of simulated and real-world experiments aimed at showing its capabilities.

Our work is a step in the study of “active” stigmergy in multi-robot construction, a promising concept in which the structure being built is an active component of the process. This idea makes it possible to simplify the design of the builder robots. In addition, the smart blocks could be used as a form of smart scaffolding. This would make it possible to reuse the same robotic platforms in multiple construction projects, thus paving the way for affordable, cost-effective construction in space applications and remote terrestrial environment.

Future work will concentrate on improving the capabilities of our robotic platforms and devising/studying novel planning algorithms to exploit the concept of active stigmergy in collective construction.

ACKNOWLEDGMENT

We thank Prof. Raghvendra Cowlagi for guidance and fruitful conversations and Cameron Collins for his assistance in the manufacturing of the hardware of the platform.

REFERENCES

[1] K. H. Petersen, N. Napp, R. Stuart-Smith, D. Rus, and M. Kovac, “A review of collective robotic construction,” Science Robotics, vol. 4, no. 28, p. eaau8479, 2019.
[2] E. Komendera, D. Reishus, and N. Correll, “Assembly by intelligent scaffolding,” Department of Computer Science, University of Colorado at Boulder, Tech. Rep., 2011.
[3] G. Theraulaz and E. Bonabeau, “A brief history of stigmergy,” Artificial Life, vol. 5, no. 2, pp. 97–116, 1999.
[4] ———, “Coordination in distributed building,” Science, vol. 269, no. 5224, pp. 686–688, 1995.
[5] A. Grushin and J. A. Reggia, “Stigmergic self-assembly of prespecified artificial structures in a constrained and continuous environment,” Integrated Computer-Aided Engineering, vol. 13, no. 4, pp. 289–312, 2006.
[6] J. Werfel, D. Inger, and R. Nagpal, “Collective construction of environmentally-adaptive structures,” in 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2007, pp. 2345–2352.
[7] C. Pinciroli, M. S. Talamali, A. Reina, J. A. R. Marshall, and V. Trianni, “Simulating kilobots within args: Models and experimental validation,” in 11th International Conference on Swarm Intelligence (ANTS 2018), ser. Lecture Notes in Computer Science. Berlin, Germany: Springer, October 2018, in press.
[8] A. Martinoli, A. J. Ijspeert, and F. Mondada, “Understanding collective aggregation mechanisms: From probabilistic modelling to experiments with real robots,” Robotics and Autonomous Systems, vol. 29, no. 1, pp. 51–63, 1999.
[9] N. Napp and R. Nagpal, “Distributed amorphous ramp construction in unstructured environments,” Robotica, vol. 32, no. 2, pp. 279–290, 2014.
[10] T. Soleymani, V. Trianni, M. Bonani, F. Mondada, and M. Dorigo, “Bio-inspired construction with mobile robots and compliant pockets,” Robotics and Autonomous Systems, vol. 74, pp. 340–350, 2015.
[11] 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,” in International Conference on Swarm Intelligence. Springer, 2014, pp. 158–169.
[12] K. H. Petersen, R. Nagpal, and J. K. Werfel, “Termes: An autonomous robotic system for three-dimensional collective construction,” Robotics: science and systems VII, 2011.
[13] Y. Deng, Y. Hua, N. Napp, and K. Petersen, “Scalable compiler for the TERMES distributed assembly system,” in Distributed Autonomous Robotic Systems. Springer, 2019, pp. 125–138.
[14] B. Jenett and K. Cheung, “Bill-E: Robotic platform for locomotion and manipulation of lightweight space structures,” in 25th AIAA/AHS Adaptive Structures Conference, 2017, p. 1876.
[15] R. Stuart-Smith, “Behavioural production: Autonomous swarm-constructed architecture,” Architectural Design, vol. 86, no. 2, pp. 54–59, 2016.
[16] R. Volpe, “The lemur robots,” 2019. [Online]. Available: [https://www-robotics.jpl.nasa.gov/systems/system.cfm?System=5](https://www-robotics.jpl.nasa.gov/systems/system.cfm?System=5)
[17] D. H. Myszka, Machines and mechanisms. Prentice Hall, 2004.
[18] M. W. Spong, S. Hutchinson, and M. Vidyasagar, Robot modeling and control. Wiley, 2006.
[19] M. Colledanchise and P. ¨Ogren, Behavior Trees in Robotics and AI: An Introduction. CRC Press, 2018.
[20] F. L. Ratnieks and C. Anderson, “Task partitioning in insect societies,” Insectes sociaux, vol. 46, no. 2, pp. 95–108, 1999.
[21] G. Piuri, A. Brutschy, M. Prison, A. Rolli, M. Dorigo, and M. Birattari, “Task partitioning in swarms of robots: An adaptive method for strategy selection,” Swarm Intelligence, vol. 5, no. 3-4, pp. 283–304, 2011.