Cocoon | Collaborative Construction

A multi-robot system for complex tasks on construction sites: Reference case of a brick wall construction

Doğa Su Kıralioğlu
RWTH Aachen University
Construction Robotics Master Program
Prototyping Project WS 2023-24
Chair Design Computation

April 3, 2024

Disclaimer

All materials, including but not limited to code, documentation, and associated files, presented in the project "Cocoon: Collaborative Construction for Discrete Assemblies" belong to the author, Doğa Su Kıralioğlu, unless explicitly stated otherwise. Unauthorized use, reproduction, or distribution of any part of this project without the express written consent of the author is strictly prohibited.

The prototype and its components are provided "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The author assumes no responsibility for errors or omissions in the project or accompanying documentation.

Copyright

Cocoon: Collaborative Construction for Discrete Assemblies by Doğa Su Kıralioğlu.

2024
dogasukiralioglu@gmail.com

Construction Robotics Master’s Program | Winter Semester 2023-24. RWTH Aachen University. Special thanks to Prof. Jakob Beetz.

Contact

Doğa Su Kıralioğlu

Kastanianweg 35
Kawo 2, 1032

dogasukiralioglu@gmail.com

Table of Contents

1 Introduction **4**

2 Literature Review **5**

2.1 State of the Art 5

2.2 Research Gap 8

3 Motivation **9**

3.1 Problem Statement 9

3.2 Research Objectives 10

3.3 Expected Results 10

4 Prototype **10**

4.1 Complete Workflow of IFC to Fabrication 10

4.2 IFC Information Retrieval 11

4.3 Fabrication Data Generation 12

Bibliography **13**

A Task Decomposition in JSON Format **14**

B Python Code for IFC Information Retrieval **16**

1 Introduction

Throughout the last decades, the yearning for originality has only been increasing. Many artists believed that the only way to have new and original artwork was through a new way of making art. A new architecture can also be possible by a new way of building, a new way of making architecture. The introduction of robotics into the domain of construction holds the promise of this possibility.

The benefits of the introduction of robotic systems into the architectural domain are nevertheless not limited to this promise. Robotic systems on construction sites open up new possibilities for both artistic and technical advancements. This revolution would not only increase efficiency in a sector that is in dire need of it, but also improve working conditions for workers, and even allow for construction projects in extreme environments that would previously be impossible (Melenbrink et al., 2020a).

Yet another reason for the construction sector being a perfect fit for robotic automation systems is the 4D principle. This principle suggests that robotic systems are especially fit for implementation in jobs or environments that are dirty, dull, dangerous, or difficult. Many of the tasks on a construction site correspond to at least one of these metrics, which is why economists and investors soon expect a robotic revolution in the Architecture, Engineering, and Construction (AEC) industry (M.-K. Kim et al., 2019).

While these immediate practical advantages can be observed already in the industry, the implementation of robotic systems in construction holds promise on a much larger scale. With the recent advancements in technology, automation is getting more and more widespread in all of the industries. The most efficient way of producing anything is standardization and mass production. Still, for certain practices that are ancient and painfully human, such as cooking and architecture, complete standardization and mass production is not an option that is acceptable.

These practices are very tightly woven together with culture, tradition, and community so that they have a bi-directional effect on each other. This reality, together with the idea that the changes in the way of making construction will have a direct effect on the results of the process, which is the architecture that is made, makes it abundantly clear that how we handle the introduction of robots into the architectural domain will determine how our architecture will be in the upcoming decades.

This paper presents Cocoon as a prototype, taking brick-laying as a reference process, and offering an autonomous alternative to this key process with a collaborative multi-robot system. Throughout the following chapters, the project is presented together with a literature review, motivation statement, and functionalities of the prototype, illustrating the key gaps it fills in a possible road map toward a fully autonomous construction site.

2 Literature Review

2.1 State of the Art

The field of construction robotics is placed at the intersection of many different fields. In the last decade, significant attention has been on collective robotic construction, catalyzing many research studies and projects. To have an understanding of the current state of the field of construction robotics, and to identify the research gaps, several recent studies are examined in this section.

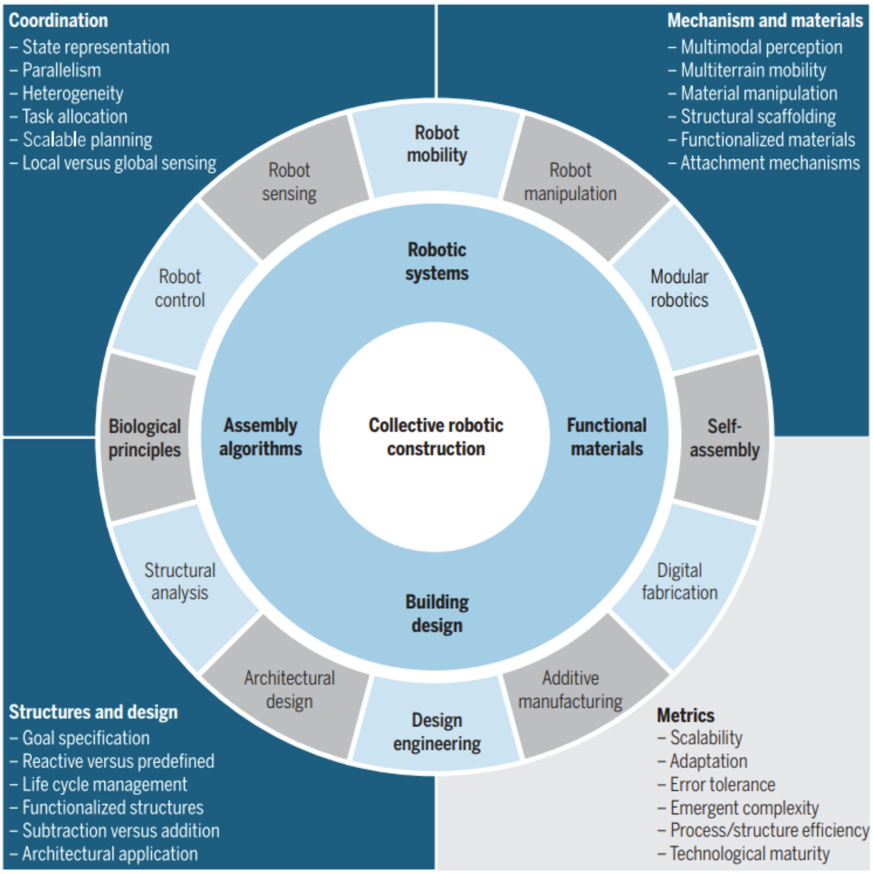

Figure 1: Diagram showing how collective robotic construction depends on and interacts with many other fields. (Petersen et al., 2019)

At first glance, one major differentiating factor between current studies is the method of assembly employed. A considerable majority of studies on robotic systems for construction focus on discrete assemblies. While there are still many studies that handle continuous assembly methods, such as Tucker et al. (2022) which provides a promising example of a collaborative robot system for spatial lacing, it is not surprising that discrete assemblies receive more attention. Today, discrete assembly processes are prevalent in robotic applications across various fields. This might explain why the first attempts at introducing robotic systems into the field of construction focus on discrete assemblies, with the possibility of translating readily available expertise from different fields into the domain of construction.

Within the studies handling discrete assembly processes for construction tasks, there are several different approaches. One main differentiation here is the module of assembly. While the majority of studies focus on traditional assembly blocks such as bricks, some studies introduce novel material designs. An example of this is by Dierichs and Menges (2016), employing a unique granular system for aggregation. Another significant project is by Jenett et al. (2019), where the building blocks are co-designed with the robots that will assemble them.

When it comes to traditional building blocks such as bricks, there is a wealth of research and projects, focusing on different aspects of brick construction. These studies can be categorized by their focus, such as design-focused, process-focused, and robot-focused. Design-focused studies either alter the brick design on a construction project or automize the design process for higher efficiency. In the case of this layout study by C. Xu et al. (2021), the layout of bricks on a wall is altered for less waste and higher efficiency in material usage. In another study by Liu et al. (2021), the brick layout is automatically designed for better time efficiency in construction projects' design phase.

Process-focused studies analyze the assembly processes to offer possible improvements or to demonstrate the possible shortcomings. A study by Worcester and Hsieh (2012), demonstrates possible different decompositions of a structure to discover improved assembly strategies. Elkhapery et al. (2023) offers alternatives to traditional brick assembly sequences and points out possible shortcomings and pitfalls. A fascinating example is by Parascho et al. (2020), suggesting a two-robot collaborative team, in which the agents take turns temporarily supporting the structure to make the construction of a brick vault possible. In a similar vein, Melenbrink et al. (2017) proposes a system of force-aware robot collectives that act as a counterweight at times throughout the assembly process to balance the structure.

Studies focusing on robot design for brick assemblies also offer significant contributions to the current discourse. An example by Seo et al. (2013), offers a design for a homogenous team of rectangular modular robots for an assembly strategy for planar structures. Torpoco-Lopez et al. (2023) focuses on the design of a single robot for brick assembly processes. Shi et al. (2023) again offers a robot design, focusing on the material properties of mortar and how to analyze it to achieve higher precision. While most studies are on ground vehicles, there are also some studies with different approaches, such as Wu et al. (2018) who propose a stationary cable-driven system for brick wall assemblies.

While many studies in the field are not geared toward immediate on-site application, some projects are already being actively used today. Two of these cases are reviewed here as examples of successful implementation of robotic systems on construction sites. The first example is the SAM100 robotic bricklaying aid which is largely embraced by the industry. The robotic arm assists workers by helping to carry the load of heavier assembly blocks. While this robotic system is on the lower end in terms of levels of autonomy, its practical safety mechanisms make for a good example solution for situations where workers have to carry heavy loads. (Madsen, 2019)

Another example that makes the construction workers' lives easier is HILTI Jaibot (X. Xu et al., 2022). This drilling robot takes location information from IFC models which include specially designed IFC objects, and executes the drilling operation according to the retrieved location information using its sensors. This implementation case is not only admirable for providing an automated solution to a very common construction task but also for relieving construction workers from this highly strenuous activity.

With these examples, it is important to contemplate the reasons for their success. Firstly, as Melenbrink et al. (2020b) expresses, there is a perceived demand for novel solutions developed for specific construction tasks. Additionally, the solutions focusing on specific tasks are relatively easier to implement without needing extensive redesign or investment in a construction project. On top of this, certain key processes, respectively bricklaying and overhead drilling in these examples, are widespread and executed in almost the same manner every time, independent of the general design of the construction project. This means that a solution developed for such a process can be reimplemented in many different sites without needing major readjustments. Considering all these, it is not difficult to understand why these two examples were embraced by the industry.

With task-specific solutions being favored in the industry, it is difficult to find studies focusing on cooperation and higher-level organization for robotic systems on construction sites. Still, as stated by Melenbrink et al. (2020b), solutions for coordination between different robots working on a variety of construction tasks are crucial to making a fully autonomous construction site possible. Fortunately, there are some studies focused on this very issue. Thangavelu and Napp (2021) proposes a system architecture for heterogeneous multi-robot systems on construction sites. Mantha et al. (2020) offers a strategy for task allocation and route planning for task-oriented building service robots. H. Kim et al. (2013), utilizes BIM models to generate construction schedules, while S. Kim et al. (2021) uses BIM models for robot task planning.

In conclusion, with the increasing attention over the last decade, there is a wealth of research and new developments in the field of construction robotics. A significant majority of studies employ discrete assembly methods, leveraging the advantage of translating expertise on robotic assembly from other sectors into the domain of construction. Most of these studies combine novel robotic processes and traditional materials, either to increase efficiency in traditional construction processes or to achieve higher levels of precision. Even though most studies are not targeted for on-site implementation, there are still some successful applications on-site, although limited to single robot systems. The industries' preference for single robot systems drives the majority of research into this area, while solutions for collaboration between robots remain mostly neglected. In the end, the need for collaborative solutions is just as stark as the need for task-specific solutions, to make a fully autonomous construction site possible one day.

2.2 Research Gap

The introduction of robotic systems into construction sites has been taking up speed and receiving more and more attention throughout the last decade. Still, there are many gaps to be filled before a complete autonomous workflow is possible on the construction site. One of the major gaps, as stated by Melenbrink et al. (2020a) is the lack of solutions to coordinate multiple task-specific robots that operate on a variety of different construction tasks. The shift in the understanding of construction site planning and scheduling from temporal planning to spatio-temporal planning, such as LEAN scheduling, provides a good starting ground for robotic systems scheduling solutions on construction sites.

Another point for major improvement lies in the myriad amounts of data that are being produced in the AEC industry. Digital models containing information on the 3D configuration, materials, quantities, and other details of a construction project are widespread in the industry. The data that can be retrieved from these models offers great potential for automation in several aspects. One example is automated scheduling as demonstrated by S. Kim et al. (2021). Another example is offered by X. Xu et al. (2022), in which the drilling locations for the robot are extracted from a model.

Most importantly, these models can be utilized to overcome a major shortcoming of robotic systems on construction sites, which is the lack of detailed behavior generation for the robots. One aspect that separates the construction industry from most other industries is the abundance of unique projects. Construction projects inherently necessitate unique requirements for every project, which means generating unique fabrication data and robot behaviors for every project. An example of how IFC/BIM data can be utilized for this purpose can be seen in the studies of S. Kim et al. (2021), yet it is rare to find studies in this particular area. The need for reference processes with detailed robot behavior for different construction tasks is still to be addressed. Especially for the tasks that play a key role on the construction sites, the development of reference processes is a crucial step on the way to automate construction.

To summarize, contemporary research in the field of construction robotics is largely focused on the development of task-specific single-robot systems. Still, there are many gaps to be addressed in this area, especially for widespread key processes on construction sites. Additionally, many of these studies manually program robot behaviors, without offering a reference process for the respective tasks, which limits the potential applicability of these solutions for different construction projects. On top of the need for task-specific solutions, the collaboration of individual task-specific robots is a topic that needs to receive more attention. Before a multi-robot collaborative system is possible on the construction sites, reference processes need further development, as well as solutions for collaborative systems addressing scheduling, coordination, and data exchange.

3 Motivation

This paper presents Cocoon as a prototype, taking brick-laying as a reference process, and providing an autonomous alternative to this key process with a collaborative multi-robot system.

Current literature in the field of construction robotics shows crucial gaps in research for collaborative multi-robot systems. With this prototype, the aim is to provide a reference process to fill one of the gaps for task-spesific key processes on construction sites, and then suggest roadmaps and possible applications to integrate this task-specific solution into a larger system for an autonomous construction site.

Brick-laying is selected as the reference process due to its independence from other construction tasks and its ubiquitous nature across various construction projects. Moreover, the common practice of subcontracting brick assembly demonstrates its potential for individual execution, making it an ideal candidate for autonomous robotic solutions. Additionally, the widespreadness of IFC/BIM information presents an opportunity for Cocoon to harness this data effectively. By developing adaptable reference processes for diverse brick wall assembly tasks, Cocoon aims to provide a foundational cell for autonomous construction sites.

3.1 Problem Statement

The prototype is developed to address the lack of independent task-specific systems for brick-laying processes in construction. It aims to tackle challenges related to task allocation, fabrication motion generation, and simulation using standard BIM models prevalent in the Architecture, Engineering, and Construction (AEC) industry. By generating comprehensive reference processes encompassing material specifications, path planning, schedules, and other intricate details, Cocoon aims to establish a robust foundation for autonomous construction tasks.

3.2 Research Objectives

- Identify the existing gaps in the current literature regarding collaborative multi-robot systems in construction.
- Pinpoint areas where advancements in robotics can enhance construction processes.
- Determine the prerequisites for implementing autonomous processes, including path planning, material requirements, and robot motions.

3.3 Expected Results

- Generation of fabrication data and robot behavior for bricklaying processes, for automation and efficiency in construction tasks.
- Optimal scheduling and task allocation facilitated by leveraging information from the IFC model, allowing for efficient re-allocation of tasks and generation of robot paths to optimize construction timelines.
- Integration of task-specific system into a real-time multi-robot simulation based on allocated tasks and product specifications.

4 Prototype

4.1 Complete Workflow of IFC to Fabrication

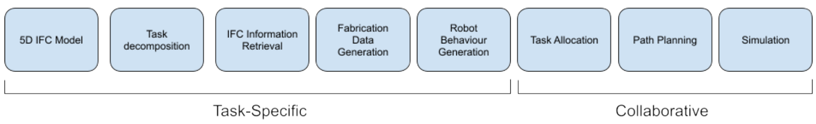

Figure 2: Diagram showing sections of the prototype.

This prototype aims to demonstrate the possibilities of information retrieval from a standard IFC model, and the extent to which this information can be utilized for planning, optimization, and fabrication data generation.

The prototype requires only a standard IFC Model as input. An IFC model prepared for a hypothetical single-story residential building in the scope of Prototyping Project Course WS 2023-24 is used in this prototype as an example. Fabrication data generation including generation of the robot motions, does not require any additional input.

The prototype can be divided into two individually functional parts: IFC

Information Retrieval with Python, and Fabrication Data Generation with Cocoon Discrete Assembly Plug-in for Rhino Grasshopper.

4.2 IFC Information Retrieval

The information stored within the IFC model is retrieved in this step using Python and IFC OpenShell API. This information is then translated into fabrication data and robot motions inside the Rhino Grasshopper environment.

Certain functions are in place to make sure that the tasks are retrieved, assigned, and scheduled within a logical order. Three examples of these definitions are presented in this section. Complete Python code can be found in the appendix.

Definition: get predecessor(task): Ensures that the retrieved tasks for execution have no unfinished predecessors. Avoids scheduling conflicts, such as scheduling a finishing task before wall assembly.

```
def get_predecessor(task):
    # Use the BlenderBIM API utility functions
    predec = ifcopenshell.util.sequence.get_sequence_assignment(task,
         sequence='predecessor')
    return predec
```

Definition: filter tasks by keyword(tasks, keyword): Retrieves necessary tasks based on certain keywords instead of their full corresponding names in the decomposition file. Avoids errors caused by different naming conventions.

```
def filter_tasks_by_keyword(tasks, keyword):
    filtered_tasks = []
    for task_list in tasks.values():
        for task in task_list:
            if keyword.lower() in task.Name.lower():
                filtered_tasks.append(task)
    print('    ' f'{keyword} tasks')
    for task in filtered_tasks:
        print(f"Task ID: {task.id()}, Task Name: {task.Name if
            hasattr(task, 'Name') else 'N/A'}, Level: {max_level}")
    return filtered_tasks
```

Definition: match tasks with boundary points(tasks): Retrieves boundary points for tasks for run-time obstacle generation. Finished tasks can be fed to generate the products associated with them on the robot world, for re-calculation of robot paths according to these new obstacles.

```

def match_tasks_w_bndpts(tasks):
    bnd_points = {}

    for task in tasks:
        all_bndpts = []
        products = find_task_products(task)

```

```
        for product in products:
            bnd_pt = get_bottom_vertices(product)
            all_bndpts.append(bnd_pt)
        bnd_points[task] = all_bndpts

    print(bnd_points)
    return bnd_points
```

4.3 Fabrication Data Generation

The next part of the prototype is fabrication data generation with Cocoon Discrete Assemblies plug-in for Grasshopper. After the information retrieval phase is complete, the newly created IFC model is fed into the Rhino environment for this second phase. This is the intended use for a complete workflow starting with a raw IFC model, yet it is also possible to feed a manually manipulated .obj file in this step.

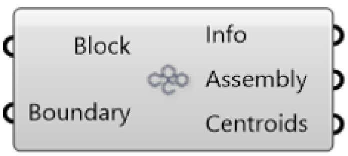

Figure 3: Example of a component from Cocoon: Discrete Assemblies plug-in.

Design of the plug-in is rather simple. All components have two inputs, labeled as Block and Boundary. All components have an Info output, and depending on the component several other outputs, which can be Block, Assembly, and Centroids.

- Block(Input): Mesh representing a unit block for the discrete assembly.
- Boundary(Input): Mesh representing a boundary for the discrete assembly.
- Info(Output): Text output streaming errors and execution information.
- Block(Output): Transformed mesh, representing the oriented assembly block according to the given initial block and assembly boundary.
- Assembly(Output): Discrete assembly generated according to the given Block and Boundary.
- Centroids(Output): Center points of each assembly block, to be used for generating robot pick-and-place motions.

Reference List

Dierichs, K., & Menges, A. (2016). Towards an aggregate architecture: Designed granular systems as programmable matter in architecture. *Granular Matter*, *18*(2), 25. https://doi.org/10.1007/s10035-016-0631-3

Elkhapery, B., Pěnička, R., Němec, M., & Siddiqui, M. (2023). Metaheuristic planner for cooperative multi-agent wall construction with UAVs. *Automation in Construction*, *152*, 104908. https://doi.org/10.1016/j.autcon.2023.104908

Jenett, B., Abdel-Rahman, A., Cheung, K., & Gershenfeld, N. (2019). Material–Robot System for Assembly of Discrete Cellular Structures. *IEEE Robotics and Automation Letters*, *4*(4), 4019–4026. https://doi.org/10.1109/LRA.2019.2930486

Kim, H., Anderson, K., Lee, S., & Hildreth, J. (2013). Generating construction schedules through automatic data extraction using open BIM (building information modeling) technology. *Automation in Construction*, *35*, 285–295. https://doi.org/10.1016/j.autcon.2013.05.020

Kim, M.-K., Wang, Q., & Li, H. (2019). Non-contact sensing based geometric quality assessment of buildings and civil structures: A review. *Automation in construction*, *100*, 163–179.

Kim, S., Peavy, M., Huang, P.-C., & Kim, K. (2021). Development of BIM-integrated construction robot task planning and simulation system. *Automation in Construction*, *127*, 103720. https://doi.org/10.1016/j.autcon.2021.103720

Liu, J., Cao, Y., Xue, Y., Li, D., Feng, L., & Chen, Y. F. (2021). Automatic unit layout of masonry structure using memetic algorithm and building information modeling. *Automation in Construction*, *130*, 103858. https://doi.org/10.1016/j.autcon.2021.103858

Madsen, A. J. (2019). The sam100: Analyzing labor productivity.

Mantha, B. R., Jung, M. K., García De Soto, B., Menassa, C. C., & Kamat, V. R. (2020). Generalized task allocation and route planning for robots with multiple depots in indoor building environments. *Automation in Construction*, *119*, 103359. https://doi.org/10.1016/j.autcon.2020.103359

Melenbrink, N., Kassabian, P., Menges, A., & Werfel, J. (2017). Towards Force-aware Robot Collectives for On-site Construction, 382–391. https://doi.org/10.52842/conf.acadia.2017.382

Melenbrink, N., Werfel, J., & Menges, A. (2020a). On-site autonomous construction robots: Towards unsupervised building. *Automation in construction*, *119*, 103312.

Melenbrink, N., Werfel, J., & Menges, A. (2020b). On-site autonomous construction robots: Towards unsupervised building. *Automation in Construction*, *119*, 103312. https://doi.org/10.1016/j.autcon.2020.103312

Parascho, S., Han, I. X., Walker, S., Beghini, A., Bruun, E. P. G., & Adriaenssens, S. (2020). Robotic vault: A cooperative robotic assembly method for brick vault construction. *Construction Robotics*, *4*(3-4), 117–126. https://doi.org/10.1007/s41693-020-00041-w

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

Seo, J., Yim, M., & Kumar, V. (2013). Assembly planning for planar structures of a brick wall pattern with rectangular modular robots. *2013 IEEE International Conference on Automation Science and Engineering (CASE)*, 1016–1021. https://doi.org/10.1109/CoASE.2013.6653996

Shi, Q., Wang, Z., Ke, X., Zheng, Z., Zhou, Z., Wang, Z., Fan, Y., Lei, B., & Wu, P. (2023). Trajectory optimization of wall-building robots using response surface and non-dominated sorting genetic algorithm III. *Automation in Construction*, *155*, 105035. https://doi.org/10.1016/j.autcon.2023.105035

Thangavelu, V., & Napp, N. (2021). Design and Simulation of a Multi-Robot Architecture for Large-Scale Construction Projects. *2021 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 181–189. https://doi.org/10.1109/MRS50823.2021.9620673

Torpoco-Lopez, L. W., Cochachi-Rubio, T. J., Olivera-Pérez, R. M., Del Carpio-Ramirez, S. I., Ortiz-Zacarias, J. R., & Perez-Campomanes, G. (2023). Mechatronic design for load-bearing masonry construction based on BIM methodology. *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*, 1098–1104. https://doi.org/10.1109/CCWC57344.2023.10099229

Tucker, C., Yang, X., Lehrecke, A., Maierhofer, M., Duque Estrada, R., & Menges, A. (2022). A Collaborative Multi-robot Platform for the Distributed Fabrication of Three-dimensional Fibrous Networks (Spatial Lacing). *Proceedings of the 7th Annual ACM Symposium on Computational Fabrication*, 1–18. https://doi.org/10.1145/3559400.3561995

Worcester, J., & Hsieh, M. A. (2012). Task Partitioning via Ant Colony Optimization for Distributed Assembly [Series Title: Lecture Notes in Computer Science]. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, M. Dorigo, M. Birattari, C. Blum, A. L. Christensen, … T. Stützle (Eds.), *Swarm Intelligence* (pp. 145–155). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-32650-9_13

Wu, Y., Cheng, H. H., Fingrut, A., Crolla, K., Yam, Y., & Lau, D. (2018). CU-brick cable-driven robot for automated construction of complex brick structures: From simulation to hardware realisation. *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, 166–173. https://doi.org/10.1109/SIMPAR.2018.8376287

Xu, C., Liu, J., Li, S., Wu, Z., & Chen, Y. F. (2021). Optimal brick layout of masonry walls based on intelligent evolutionary algorithm and building information modeling. *Automation in Construction*, *129*, 103824. https://doi.org/10.1016/j.autcon.2021.103824

Xu, X., Holgate, T., Coban, P., & Garcıa de Soto, B. (2022). Implementation of a robotic system for overhead drilling operations: A case study of the jaibot in the uae. *International Journal of Automation & Digital Transformation*.

Appendices

A Task Decomposition in JSON Format

```
{
    "STRUCTURAL WORKS": {
        "Ground floor slabs": [
            "Locate" "sequential",
            "Pick and place" "sequential",
            "Mix" "sequential",
```

```
            "Pour" "collaborative",
            "Swipe" "collaborative",
            "Measure" "sequential",
            "Record" "sequential"
        ],
        "Ground floor beams": [
            "Pick and place LS" "sequential",
            "Fix" "sequential",
            "Pick and place SS" "collaborative",
            "Power tooling" "collaborative",
            "Locate" "sequential",
            "Record" "sequential"
        ],
        "Ground floor walls": [
            "Locate" "sequential",
            "Pick and place" "sequential",
            "Mix" "sequential",
            "Pour" "sequential",
            "Swipe" "sequential",
            "Measure" "sequential",
            "Record" "sequential"
        ],
        "Ground floor columns": [
            "Pick and place LS" "sequential",
            "Fix" "sequential",
            "Pick and place SS" "collaborative",
            "Power tooling" "collaborative",
            "Locate" "sequential",
            "Record" "sequential"
        ]
    },
    "ENVELOPE TASKS": {
        "Exterior walls non load bearing": [
            "Locate" "sequential",
            "Pick and place" "sequential",
            "Mix" "sequential",
            "Pour" "sequential",
            "Swipe" "sequential",
            "Measure" "sequential",
            "Record" "sequential"
        ],

        "Wall exterior coating": [
            "Pour" "collaborative",
            "Swipe" "collaborative"
        ],

        "Wall exterior finishing": [
            "Pour" "collaborative",
            "Swipe" "collaborative"
        ],
        "Install exterior windows": [
            "Pick and place" "collaborative",
            "Power tooling" "collaborative",
            "Record" "sequential"
        ],
        "Install exterior doors": [
            "Pick and place" "collaborative",
            "Power tooling" "collaborative",
            "Record" "sequential"

        ]
    },
    "INTERIOR WORKS": {
        "Interior doors installation": [
            "Pick and place" "collaborative",
            "Power tooling" "collaborative",
            "Record" "sequential"
        ],
        "Wet surface finishing": [
```

```
            "Locate" "sequential",
            "Mix" "sequential",
            "Pour" "sequential",
            "Swipe" "sequential",
            "Pick and place" "sequential",
            "Measure" "sequential",
            "Record" "sequential"
        ],
        "Dry floor finishing": [
            "Pour" "sequential",
            "Swipe" "sequential",
            "Pick and place" "sequential"
        ],
        "Partition walls installation": [
            "Locate" "sequential",
            "Fix" "sequential",
            "Pick and place" "collaborative",
            "Power tooling" "collaborative",
            "Pour" "collaborative",
            "Swipe" "collaborative",
            "Record" "sequential"
        ],
        "Interior walls finishing": [
            "Pour" "collaborative",
            "Swipe" "collaborative"
        ]
    },
    "Elemental tasks": [
        "Pick and place",
        "Pick and place LS",
        "Pick and place SS",
        "mix",
        "pour",
        "swipe",
        "fix",
        "locate",
        "measure",
        "record",
        "power tooling"
    ]
}
```

B Python Code for IFC Information Retrieval

```python
# Import necessary libraries and functions.

import os
import ifcopenshell
import ifcopenshell.api
from ifcopenshell.api import run
import ifcopenshell.util.sequence
import ifcopenshell.util.placement
from collections import Counter
import ifcopenshell.geom
import ifcopenshell.util.shape

### DEFINITIONS ###

# Retrieve all IFC Tasks and their hierarchical information.

def print_all_tasks(ifc_model):
    printed_tasks = set()
    leaf_tasks = {}
```

```

    def print_task_with_gap(task, level):
        print("   " * level + "Task ID:", task.id())
        print("   " * level + "Task Name:", task.Name if hasattr(task, "Name") else "N/A")
        print("   " * level + "-" * 40)
        printed_tasks.add(task.id())

        if level in leaf_tasks:
            leaf_tasks[level].append(task)
        else:
            leaf_tasks[level] = [task]


    def print_nested_tasks(tasks, current_level=0):
        for task in tasks:
            if task.id() not in printed_tasks:
                print_task_with_gap(task, current_level)
                nested_tasks = ifcopenshell.util.sequence.get_nested_tasks(task)
                if nested_tasks:
                    print_nested_tasks(nested_tasks, current_level + 1)

    print_nested_tasks(tasks)
    return leaf_tasks

# Find the highest level in hirearchy, since the tasks on this level will be the first tasks to be completed in the schedule.

def print_task_levels(leaf_tasks):
    for task in leaf_tasks[max_level]:
        print(f"Task ID: {task.id()}, Task Name: {task.Name if hasattr(task, 'Name') else 'N/A'}, Level: {max_level}")

# Find all relevant tasks for the desired IFC product, using keywords such as "wall, slab, beam" etc.

def filter_tasks_by_keyword(tasks, keyword):
    filtered_tasks = []
    for task_list in tasks.values():
        for task in task_list:
            if keyword.lower() in task.Name.lower():
                filtered_tasks.append(task)
    print('    ' f'{keyword} tasks')
    for task in filtered_tasks:
        print(f"Task ID: {task.id()}, Task Name: {task.Name if hasattr(task, 'Name') else 'N/A'}, Level: {max_level}")
    return filtered_tasks

# Get tasks predecessor information.

def get_predecessor(task):
    # Use the BlenderBIM API utility functions
    predec = ifcopenshell.util.sequence.get_sequence_assignment(task, sequence='predecessor')
    return predec

# Find tasks without predecessors, since these tasks have to be carried out first. This definition helps avoid scheduling problems, such as scheduling a wall finishing job before the wall assembly job.

# def find_initial_tasks(tasks):
#     tasks_wo_predecs = []
#     for task in tasks:
#         predecs = get_predecessor(task)
#         if not predecs:
#             tasks_wo_predecs.append(task)
#     return tasks_wo_predecs
```

```
def find_initial_tasks(tasks):
    min_predecs_count = float('inf') # Initialize with positive
        infinity to find minimum
    tasks_with_min_predecs = []

    for task in tasks:
        predecs = get_predecessor(task)
        predecs_count = len(predecs)

        if predecs_count < min_predecs_count:
            min_predecs_count = predecs_count
            tasks_with_min_predecs = [task]
        elif predecs_count == min_predecs_count:
            tasks_with_min_predecs.append(task)

    return tasks_with_min_predecs

# Find tasks' products.

def find_task_products(task):

    products = ifcopenshell.util.sequence.get_direct_task_outputs(
        task)

    return products

# Create a new IFC model containing only the relevant IFC products
    and their 3D geometric representation, based on given tasks.

def create_task_product_model(tasks):

    # Create a new IFC model.

    walls_model = ifcopenshell.file(schema=model.schema)
    walls_project = run("root.create_entity", walls_model, ifc_class=
        "IfcProject", name="Walls")

    # TIP: 3D geometries will not be represented in the file if you
        don't assign units.

    length_unit = "MILLIMETER"
    run("unit.assign_unit", walls_model, length_unit)

    context = run("context.add_context", walls_model, context_type="
        Model")
    body = run("context.add_context", walls_model, context_type="
        Model",
        context_identifier="Body", target_view="MODEL_VIEW", parent=
            context)

    # Add relevant products into the model using find_task_products
        definition..

    for task in tasks:
        products = find_task_products(task)
        for product in products:
            walls_model.add(product)

    # Write the new IFC model.

    walls_model.write('walls_model2.ifc')
    return walls_model

# Find coordinates of a product.
def find_coordinates(products):
    matrices = []
    for product in products:
        matrix = ifcopenshell.util.placement.get_local_placement(
            product.ObjectPlacement)
```

```
        matrices.append(matrix)
    return matrices

# Find center points of products. This location will be assumed execution point for each robot for their respective tasks.
def find_center_point(product):
    matrix = ifcopenshell.util.placement.get_local_placement(product.ObjectPlacement)
    # get location
    location = matrix[:3, 3]
    # Calculate center point
    center_point = tuple(location)
    return center_point

# Create a dictionary with tasks and center points of their respective products.

def match_tasks_w_cpts(tasks):
    center_points = {}

    for task in tasks:
        all_cpt = []
        products = find_task_products(task)
        for product in products:
            center_point = find_center_point(product)
            all_cpt.append(center_point)
        center_points[task] = all_cpt
    print(center_points)
    return center_points

# Create a dictionary with tasks and their boundary points for obstacle generation.

def match_tasks_w_bndpts(tasks):
    bnd_points = {}

    for task in tasks:
        all_bndpts = []
        products = find_task_products(task)

        for product in products:
            bnd_pt = get_bottom_vertices(product)
            all_bndpts.append(bnd_pt)
        bnd_points[task] = all_bndpts

    print(bnd_points)
    return bnd_points

def find_task_volumes(tasks):
    task_vol = {}

    for task in tasks:
        all_vol = []
        products = find_task_products(task)
        for product in products:
            vol = get_volume(product)
            all_vol.append(vol)
        task_vol[task] = all_vol
    print(task_vol)
    return task_vol

### vertices and volume experiments

def get_bottom_vertices(product):

    settings = ifcopenshell.geom.settings()
    shape = ifcopenshell.geom.create_shape(settings,product)
    verts = ifcopenshell.util.shape.get_vertices(shape.geometry)

    min_y = min(verts, key = lambda vertex: vertex[1])[1]
```

```python
    bottom_verts = [v.tolist() for v in verts if v[1] == min_y]

    return bottom_verts

def get_volume(product):
    settings = ifcopenshell.geom.settings()
    shape = ifcopenshell.geom.create_shape(settings,product)
    volume = ifcopenshell.util.shape.get_volume(shape.geometry)
    converted_vol = volume / 1_000_000_000 # conversion from cubic
        millimeters to cubic meters
    return converted_vol


# IFC file size checker. Use this to check if the new IFC file is
    properly written.

def get_ifc_file_size(file_path):
    try:
        size_bytes = os.path.getsize(file_path)
        size_kb = size_bytes / 1024.0
        size_mb = size_kb / 1024.0
        return size_bytes, size_kb, size_mb
    except FileNotFoundError:
        return None

### MAIN FUNCTION ###

# Define a main function to create the robot world and a dictionary
    that stores the locations of tasks.

if __name__=="__main__":

    # Open the IFC model, and retrieve necessary information using
        respective definitions.

    directory = "D:\Academic\WS2023-24\PrototypingProject\Prototype\
        src\Cocoon_Prototype_2024\IFC"
    ifc_file_path = os.path.join(directory, "basemodel.ifc")

    model = ifcopenshell.open(ifc_file_path)
    tasks = model.by_type("IfcTask")
    leaf_tasks = print_all_tasks(model)
    max_level = max(leaf_tasks.keys())
    wall_tasks = filter_tasks_by_keyword(leaf_tasks, "wall")
    initial_wall_tasks = find_initial_tasks(wall_tasks)
    # fix this part. since evry task has a predecessor now, it
        returns an empty list. FIXED
    # but now not getting all the initial wall tasks
    print(initial_wall_tasks)
    walls_model = create_task_product_model(initial_wall_tasks)

    # Check new IFC model

    new_file_size = get_ifc_file_size("walls_model2.ifc")
    print(new_file_size)

    # Get obstacle information

    # Create task-location dictionary with center points of products#

    match_tasks_w_cpts(initial_wall_tasks)

    # Outputs task products and boundary points (4 vertices that
        define the boundary in plan view)
    match_tasks_w_bndpts(initial_wall_tasks)

    # Outputs a dictionary of tasks and sum volume of all its
        products
    find_task_volumes(initial_wall_tasks)
```