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

March 5, 2024

Disclaimer

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 6

3 Prototype **7**

3.1 Complete Workflow of IFC to Fabrication 7

3.2 IFC Information Retrieval with Python 7

3.3 Fabrication Data Generation with Cocoon Discrete Assembly Plug-in for Rhino Grasshopper 10

Bibliography **11**

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., 2020).

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 (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.

2 Literature Review

2.1 State of the Art

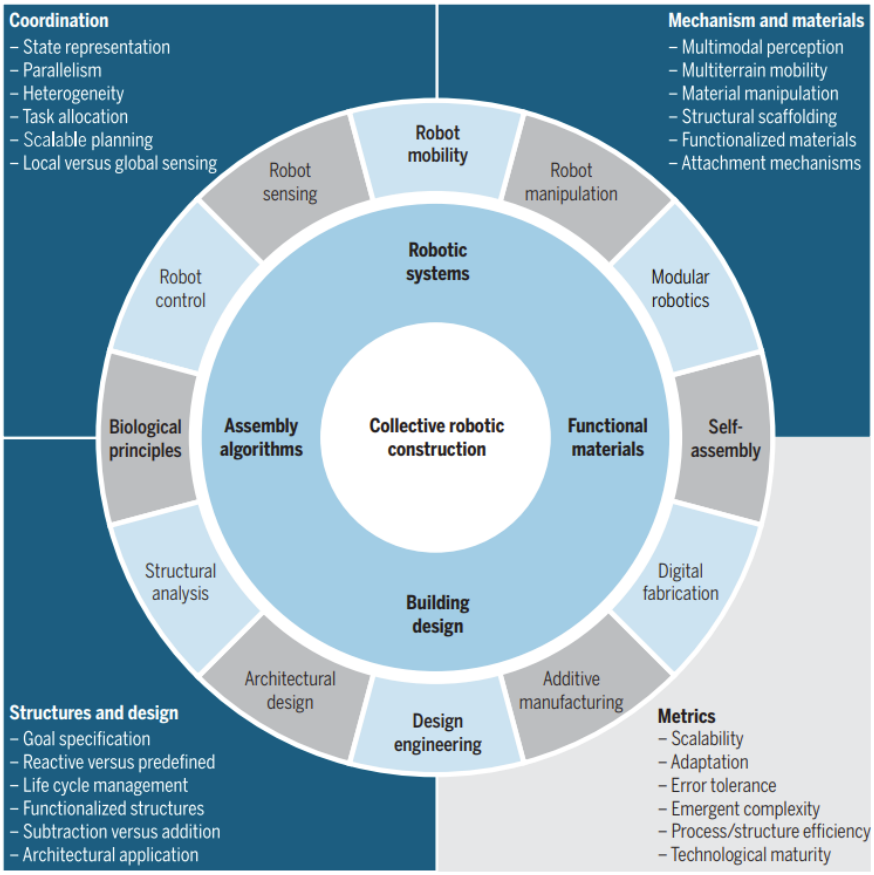

Figure 1: Diagram showing how collective robotic construction depends on and interacts with many other fields. (Petersen et al., 2019)

The implementation of robotic systems in the construction sector has already proven itself fruitful in several case studies. Three of these case studies are handled here as good examples of robotic systems on construction sites, all exhibiting different levels of autonomy.

The first example of these cases is one that is largely embraced by the industry, the SAM100 robotic bricklaying aid, which assists workers by helping to carry the load of assembly blocks. While this robotic implementation is on the lower end in terms of level 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 (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.

To go one step further, the example of Mueller and King (2008) demonstrates the construction possibilities with autonomous robots in environments unsuitable for human workers. In exoplanetary construction, lunar in this case, the robotic systems are not an option but usually a necessity. Still, in construction and site preparation tasks as demonstrated in Mueller and King (2008), there are multiple factors that determine the suitability of a multi-robot system that has to be considered on top of traditional metrics, such as the light-weightiness or multi-functionality of robots.

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 for a complete autonomous workflow to be possible on the construction site. On of the major gaps, as stated by Melenbrink et al. (2020) is the lack of solutions to coordinate multiple robots with different capabilities that operate on vast amounts of different construction tasks. Filling this gap is crucial for a multi-robot system to be functional on any construction site. 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 for any and every construction project. Data coming from IFC/ BIM models are not utilized anywhere close to their full potential. Even though there are some use cases with limited implementation, such as Hilti Jaibot which utilizes IFC data for locations Xu et al. (2022), there is still a major potential for IFC data utilization for fabrication data and robot motion generation.

One other gap that needs to be taken into consideration before a fully automated MRS (Multi-Robot System) on the construction site is possible, is the lack of reference processes for certain common construction tasks. Tasks such as drilling, bricklaying, painting, tiling, and many more, are fundamental to many construction projects all over the world. Producing reference processes for these tasks would open up vast possibilities for the immediate implementation of MRS on construction sites. Since subcontracting is already a common practice within the industry, pinpointed solutions automating or assisting workers for very specific processes, such as Hilti Jaibot and SAM100 respectively, are embraced in the industry and adopted relatively faster. These task-specific solutions would also be a crucial chain for a complete fully autonomous MRS on construction sites.

3 Prototype

3.1 Complete Workflow of IFC to Fabrication

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-storey 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.

3.2 IFC Information Retrieval with Python

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.

```
1
2  # Import necessary libraries and functions.
3
4  import ifcopenshell
5  import ifcopenshell.api
6  from ifcopenshell.api import run
7  import ifcopenshell.util.sequence
8  import ifcopenshell.util.placement
9  from collections import Counter
10
11 ### DEFINITIONS ###
12
13 # Retrieve all IFC Tasks and their hierarchical information.
14
15 def print_all_tasks(ifc_model):
16     printed_tasks = set()
17     leaf_tasks = {}
18
19     def print_task_with_gap(task, level):
20         print("___" * level + "Task_ID:", task.id())
21         print("___" * level + "Task_Name:", task.Name if hasattr(task
               , "Name") else "N/A")
22         print("___" * level + "-" * 40)
23         printed_tasks.add(task.id())
24
25         if level in leaf_tasks:
26             leaf_tasks[level].append(task)
27         else:
28             leaf_tasks[level] = [task]
29
30
31     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

# Find tasks' products, and write them into a new IFC file

def find_task_products(tasks):
    outputs = []
    for task in tasks:
        products = ifcopenshell.util.sequence.get_direct_task_outputs
            (task)
        for product in products:
            walls_model.add(product)
            outputs.append(product)
    return outputs

# Find center points of products. This location will be the starting
    point for each robot for their respective tasks.
```

```
90
91  def find_center_point(product):
92      matrix = ifcopenshell.util.placement.get_local_placement(product.
            ObjectPlacement)
93      # get location
94      location = matrix[:3, 3]
95      # Calculate center point
96      center_point = tuple(location)
97      return center_point
98
99  # Create a dictionary with tasks and center points of their
        respective products.
100
101 def match_tasks_w_points(tasks):
102     center_points = {}
103     products = find_task_products(tasks)
104     for product in products:
105         center_point = find_center_point(product)
106         center_points[product] = center_point
107     return center_points
108
109 ### MAIN FUNCTION ###
110
111 # Define a main function to create the robot world and a dictionary
        which stores the locations of tasks.
112
113 if __name__=="__main__":
114     directory = "/home/doga/src/mrta/"
115     ifc_file_path = os.path.join(directory, "basemodel.ifc")
116
117     # Open the IFC model, and retrieve necessary information using
            respective definitions.
118
119     model = ifcopenshell.open("basemodel.ifc")
120     tasks = model.by_type("IfcTask")
121     leaf_tasks = print_all_tasks(model)
122     max_level = max(leaf_tasks.keys())
123     wall_tasks = filter_tasks_by_keyword(leaf_tasks, "wall")
124     initial_wall_tasks = find_initial_tasks(wall_tasks)
125
126     ### create robot world ###
127
128     # Create a new IFC model containing only the relevant IFC
            products and their 3D geometric representation.
129
130     walls_model = ifcopenshell.file(schema=model.schema)
131     walls_project = run("root.create_entity", walls_model, ifc_class=
            "IfcProject", name="Walls")
132
133     # TIP: 3D geometries will not be represented in the file if you
            don't assign units.
134
135     length_unit = "MILLIMETER"
136     run("unit.assign_unit", walls_model, length_unit)
137
138     context = run("context.add_context", walls_model, context_type="
            Model")
139     body = run("context.add_context", walls_model, context_type="
            Model",
140         context_identifier="Body", target_view="MODEL_VIEW", parent=
                context)
141
142     # Add relevant products into the model using find_task_products
            definition.
143
144     walls = find_task_products(initial_wall_tasks)
145
146     # Write the new IFC model.
147
148     walls_model.write('walls_model.ifc')
```

```
149
150     ### create task-location dictionary ###
151
152     match_tasks_w_points(initial_wall_tasks)
```

3.3 Fabrication Data Generation with Cocoon Discrete Assembly Plug-in for Rhino 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.

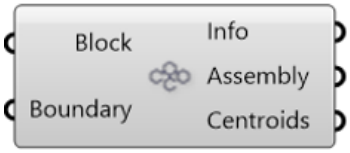

Figure 2: Example of a component from Cocoon: Discrete Assemblies plug-in.

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

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.

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

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

Mueller, R. P., & King, R. H. (2008). Trade study of excavation tools and equipment for lunar outpost development and isru. *AIP conference proceedings*, *969*(1), 237–244.

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

Xu, X., Holgate, T., Coban, P., & García 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*.