Autonomous assembly of digital materials

with inchworm locomotion robotic assemblers

by Guillermo Presa

TU Delft

Autonomous assembly of digital materials

with inchworm locomotion robotic assemblers

Thesis report

by

Guillermo Presa

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on February 7, 2024 at 12:00

Thesis committee:

Chair:	Dr.ing. S. (Saullo) G. P. Castro
Supervisor:	Dr. K. (Kunal) Masania
External examiner:	Dr. J. (Jun) Wu
Place:	Faculty of Aerospace Engineering, Delft
Project Duration:	November, 2022 - Feb, 2024
Student number:	4664884

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Cover: Inchworm assemblers building moon base
AI art made using Midjourney V6

Preface

Embarking on this final phase of my educational journey has been a long yet refreshing challenge. As a structural engineer diving into the unfamiliar realms of robotics and computer science, I encountered the best kind of problem: a multi-domain sandbox that propelled me on a path of learning and skill development. This journey was more than academic; It overlapped with my first entry into the world of entrepreneurship. This combination was a transformative expedition, shaping me into a more resilient, innovative, and insightful individual.

A heartfelt thanks to my faculty and SML colleagues, the Ph.D. candidates, Postdocs, and professors. Sharing this experience with Mark, Muhamad, Caroline, Denis, Satya, Vinay, Momo and many more has been an absolute privilege. The multiple trips to Frankfurt and Bremen are an unforgettable experience. Kunal, you've assembled an extraordinary group, and I wish the best to you and every one of them. Pietro, thank you for your patience as I transformed the lab into a lattice paradise or as you may put it a lattice mess. And Victor, I'm more than grateful for your support in making it possible to work with batteries in the lab.

My heart is also grateful for the friends, colleagues, and family whose support and companionship have been indispensable. Harold, your wisdom in printing and creative thinking transformed countless late evenings in the lab into sessions of innovation. Sian, your electrifying passion filled my battery for ambition, and Boris, the journey shared with you and Anna, remains a cherished memory. Doing this without your support would be like getting blood out of a turnip.

Kübra, your unwavering support all these late nights and early mornings have been my pillars. I couldn't be more grateful for our countless conversations, your constant encouragement and the time we share.

Alejandro, co-founding FluxPose with you amidst this unpredictable world has been a daring, often challenging, but always exciting adventure. I'm glad to share this experience with a good friend. Balancing our aspirations with what's feasible has been a journey in itself. A journey full of long days and nights, a lot of work, but also a lot of fun. The mix of working with the comforting taste of the Spanish food we shared brought a bit of home into our long days and nights. I'm profoundly grateful to everyone who believed in us along this journey, particularly the teams at the Aerospace Innovation Hub and Delft Enterprises, and to Victor for your guidance.

As I stand at this juncture, looking back at the journey and ahead to the unknown, I carry with me not just knowledge and experience, but the richness of shared moments and learned wisdom.

Guillermo Presa
Delft, February 28th, 2024

Abstract

Throughout history, human progress has been defined by the mastery of materials, transitioning from stone and bronze to the steel age. However, this progression has not only been defined by the materials utilized but has also encompassed a shift in processes, moving from the industrial to the information era. Despite the advances in fabrication techniques, building large and detailed structures continues to pose an unresolved challenge. Cost, speed, performance and size are often mutually exclusive objectives that are bound by the materials and processes used.

In the search for new processes and materials, we can find inspiration in the oldest of all fabricators; life. In natural systems, a small set of 20 amino acids are assembled by ribosomes into coherent organisms with complex sensing, actuation and information storage. Nature represents the highest dynamic range assembly system known to mankind. But a question arises: can these benefits be extended to engineering systems at meso and macro scales?

The advantages of natural fabrication emanate from the use of digital materials and self-replicating assemblers. Digital materials are composed of precise and discrete building blocks like amino acids at the micro-scale or Lego building blocks at the meso scale. They are tolerant to noise, possess embedded metrology and their assembly can be highly parallelized. Lego structures can be built more repeatably than what a standard 3D printer can print despite the imprecise nature of human assemblers. This is because the metrology and the code for construction are embedded within the material itself.

In this thesis, a complete end-to-end autonomous digital material assembly system, that bridges the gap between a 3D model and a built structure with a flexible, comprehensive, and easy-to-use toolset is presented. All elements in the triad of autonomous digital assembly were developed, from the digital material to the robot and the controlling software. A digital material made of discrete 3D-printed octahedra lattices that can be magnetically or mechanically joined is utilized. Straight, curved and elongated lattices enable unparalleled geometric freedom. This material can be picked up, transported and placed by a robotic assembler in the form of a 5DOF (degrees of freedom) cable-driven differential joint inchworm robot. Most importantly, a flexible control platform capable of interpreting 3D models, developing the necessary robot movements for optimal construction and wirelessly controlling assembler robots powers the build process. This platform introduces some major innovations within the field. For starters, it is not limited to blocky grid domains as it is powered by inverse kinematics. It is also architected to enable cooperation between different assembler types by utilizing a work package system and presents wide abstraction layers allowing further development at higher levels with ease. Additionally, it provides a seamless control interface.

Table of contents

”Nature uses only the longest threads to weave her patterns, so that each small piece of her fabric reveals the organization of the entire tapestry”

- **Richard P. Feynman**

1

Introduction

Despite the advances in fabrication techniques, building large and detailed remains an unsolved challenge. Cost, speed, performance and size are often mutually exclusive objectives bound by materials and processes. This problem is accentuated when autonomy is the only option. From the heights of space exploration to the depths of underwater platforms or radiation containment, the fabrication of structures in harsh environments necessitates digital and autonomous fabrication technologies, keeping human lives safe in challenging conditions. Yet, the fabrication of large structures is often manual and labour-intensive.

Striving to simplify the manufacturing of large structures and to avoid assembly and integration steps, many elements are integrated into monolithic pieces. As the scale of these monolithic pieces increases, the complexity savings are overwhelmed by manufacturing difficulty. Except for state-of-the-art lithography processes, most manufacturing systems only achieve about four orders of magnitude dynamic range. Large-scale 3D printers can produce millimeter scale features with build volumes of 10 meters [1], while microfabrication 2 photon polymerization processes offer micrometer precision with centimeter scale build volumes [2]. Increasing the dynamic range or the scale results in a cost barrier.

Human ingenuity has long found a constant stream of inspiration in biology for the development of machines, materials and structures. We now look for inspiration in the oldest of assemblers, life itself. Nature uses a limited set of building blocks, assembled by relatively imprecise micromachinery, to construct complex organisms complete with intricate sensing, actuation, and data storage. Biology achieves the construction of viruses as small as tens of nano-meters [3] to sequoias in the order of one hundred meters [4], all with nano-meter features. The biological factory is capable of producing structures spanning over 11 orders of magnitude differences in size, being the highest dynamic range fabrication process known to mankind. In engineering, this is equivalent to building the Golden Gate Bridge in San Francisco with micrometer-scale features as illustrated below, as much of a structural revolution as science fiction.

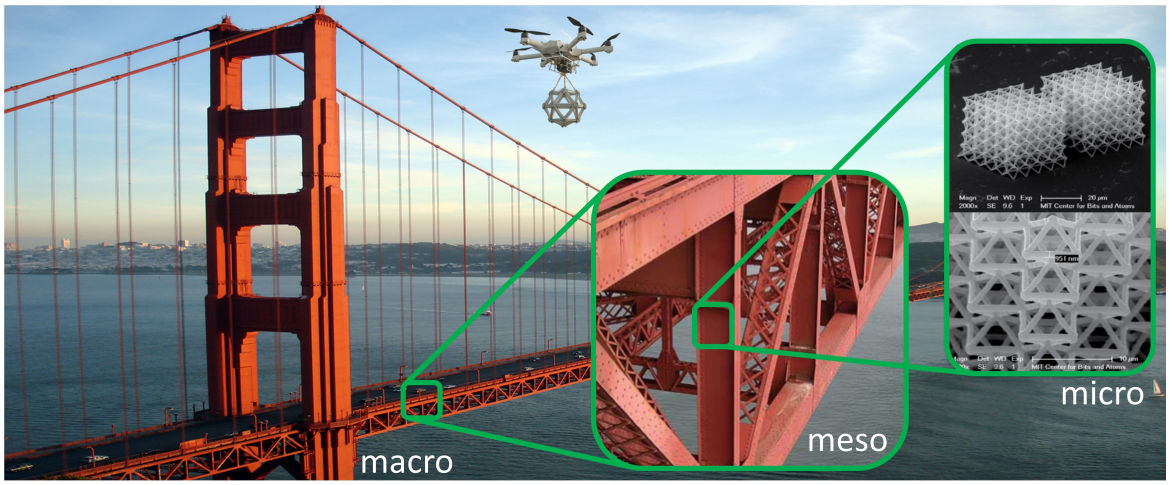


Figure 1.1: Conceptual imaging of the Golden Gate Bridge constructed from micro lattices showing the three different levels or scales of analysis involved; macro, meso and micro. Microscopy images courtesy of MIT CBA [5].

1.1 Motivation

Ribosomes are a part of the protein-generation cellular factory, that performs protein synthesis and mRNA translation. In particular, ribosomes are the assemblers in charge of linking amino acids in the particular order specified by mRNA to form polypeptides and, thus, proteins. These proteins make the macromolecules and the organelles in cells, which make tissues, organs and finally, organisms. All organisms on earth, as varied as their actuation, sensing and structure may be, build their proteins from a set of 20 amino acids, which form about 60% of mammals cell´s dry mass.

But questions arise: How is it possible for a ribosome to make something 9 orders of magnitude larger than itself and more precisely than its actuation capabilities? And can these benefits be extended to engineering systems at meso and macro scales?

In protein assembly, precise discrete building blocks are linked together at specific locations. Ribosomes don't place analogue material but discrete or digital bits. Similar to how Lego structures can be assembled way more precisely than human dexterity and even more repeatably than what a desktop 3D printer prints, ribosomes can build larger and more precisely than themselves. This is because the amino acid building blocks provide the positional precision. That is, the digital material has embedded metrology. Digital systems are tolerant to noise below their discretization level, much like the metrology of the building block, which can correct noise smaller than itself.

Additionally, the discrete nature of digital materials allows a high degree of parallelization of assembly. The level of complexity and speed achieved in the ribosomal assembly does not arise from individual ribosome speed or complexity but thanks to the parallelization of the assembly process between many ribosomes and the embedded metrology of the building blocks. Amino acid assembly occurs at slow rates between 0.6 to 20Hz [6]. Nevertheless, each cell contains up to tens of millions of ribosomes in human cells, leading to a large combined assembly speed. This parallelization, combined with ribosomes' capability to self-replicate, means the process can be started with a small critical mass. In conclusion, ribosomal assembly exploits embedded metrology, parallelization and self-replication for precise and scalable assembly of proteins with very simple assemblers.

These principles can be introduced to engineering materials and assembly processes at meso and macro scales which coincides with the objective of this thesis.

1.2 Thesis outline

The thesis is structured around three core themes: the material system, the robotic assembler, and the software platform, each distinctly outlined throughout the various parts of the report.

The report itself is also divided into 3 parts. Part I part provides a foundational overview of this triad of themes through a literature review in Chapter 2. This review consists of an evaluation of the mechanical performance of cellular materials, an overview of the organizational architectures and hardware platforms utilized for robotic assembly and a description of the fundamentals of work planning and graph structures that will be utilized for the development of the software platform. After establishing the background of this field, Chapter 3 presents a description of the main research objectives and questions.

In Part II, dedicated to presenting the results, the complete solution developed is showcased. This encompasses a brief description of the material system and its manufacturing in Chapter 4, an analysis of the robotic assembler design and operation in Chapter 5 and an in-depth explanation of the software developed and the algorithms utilized in Chapter 6. To conclude this part, a real-life build process is demonstrated and explained in Chapter 7.

The final part wraps up the thesis with conclusions drawn from the research and suggestions for future endeavours in this field in Chapter 8 and Chapter 9 respectively.

Part I

Background

2

Literature review and field background

Autonomous assembly of digital materials sits at the multidisciplinary interface between robotics, computer science and material science, making it essential to develop a basic understanding of the fields involved. The following sections present a critical evaluation of the relevant literature on this trinity of fields.

Firstly, an overview of the materials science behind the digital building blocks is presented. This includes an introduction to the field of cellular structures and the Gibson-Ashby model, as well as some conditions more specific to large lattices, like in these digital materials. Next, literature on different assembler types and organizational architectures is presented. Lastly, background knowledge that is necessary for the software planner consisting of an overview of graphs and their search algorithms is presented.

2.1 Digital materials: Cellular, lattice and meta materials

Digital materials are a development that results from discretizing a continuum of material into discrete modular building blocks that can be assembled together [7]. Thus, digital materials are assembled modular materials incorporating the principles of embedded metrology and parallelization of assembly found in natural construction.

Natural assembly emanates from a small set of about 20 amino acids from which all the proteins that make the molecular machinery, cellular actuation and information storage are built. Similarly, a small finite set of building blocks or modular structural elements of different geometries, materials, densities and functions ranging from stiff to compliant, conductive, insulating, ferromagnetic etc. can enable the construction of a wide range of high-performance functional structures as demonstrated in [8] [9]. In this thesis, we focus purely on stiff structural elements with dimensions in the $1 - 30cm$ range and lightweight densities in the order of $50mg/cm^3$,which represent a good compromise between our manufacturing capabilities, assembly complexity, detail and structural performance. To achieve this, microstructures with a high porosity are utilized and studied next.

2.2 Cellular materials

Cellular materials are porous structures characterized by the deliberate inclusion of voids. From micro to macro structures, the world is full of cellular materials, like truss bridges or natural bones. This is not without reason: Cellular materials allow for the optimization of material properties not only by altering their chemical composition but also by structuring their solid and void networks.

With the advent of additive manufacturing, these geometries have become realizable in a controlled fashion leading to great interest in their understanding and design. Extensive research has been carried out for the exploitation of their lightweight, energy absorbent, heat exchanging, osteointegration, acoustic and vibrational damping properties. Commercial software products[1] exist to better integrate cellular materials into high-performance structures. For this project, their structural properties, namely stiffness and strength, that is, the mechanical characteristics that follow the Gibson-Ashby model, are of most interest.

[1] Ntopology https://www.ntop.com/

2.2.1 Gibson-Ashby model

To estimate the performance of these materials, cellular solid theory and the Gibson-Ashby model [10] are utilized to project into the Ashby plots space, enabling predictions of how properties scale with density.

The mechanical properties of cellular structures are a function of their cell architecture, constituent material and relative density. The latter is the ratio between the density of the cellular structure relative to the density of the base material $\frac{\rho}{\rho_s}$. A cellular structure containing little void content can present mechanical properties closely resembling the bulk material. The Gibson-Ashby model expresses the relative strength and modulus as a function of the relative density of the lattice structure in the form of a power-law relationship. This is showcased in Equation (2.1) and Equation (2.2). The values for the coefficient C, C' and exponents n, m of these power relations depend on the nature of the lattice's deformation and thus its cellular morphology. For instance, in open cell foams, n is typically around 2 and m between 1.5-2.

$$\frac{E^*}{Es} = C\left(\frac{\rho}{\rho_s}\right)^n \tag{2.1}$$

$$\frac{\sigma}{\sigma_s} = C'\left(\frac{\rho}{\rho_s}\right)^m \tag{2.2}$$

Lines describing this relation for particular values can be drawn in the Ashby plot space to project the properties of a particular material to lower densities. This is showcased in Figure 2.1 with lower and upper bounds represented by quadratic and linear scaling found in bending and stretch dominated deformation modes respectively.

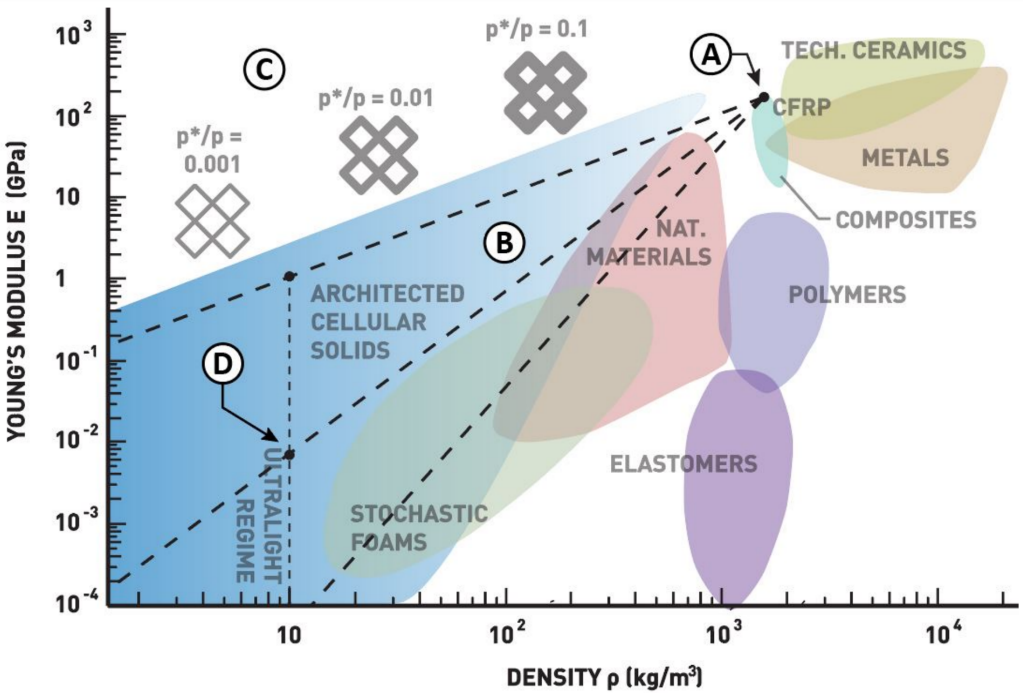


Figure 2.1: Properties scaling of Gibson-Ashby model. Figure courtesy of [11].

This model's main shortcoming is the use of a single deformation mode to capture each material's scaling behaviour, and not considering multi mode deformation processes, limiting its applicability to slender geometries [12]. For instance, round beam based lattices must have an slenderness ratio $l/d > 5$ to adhere to the model. Alternatively, this can be accounted for by recalculating E^* or σ^* to include the axial and shear deformation of the cell walls, as originally done by Gibson and Ashby [10].

Cellular materials englobe a plethora of categories. First, a distinction is made between stochastic structures, such as open and closed cell foams, and non-stochastic ones like honeycombs and lattice structures. The latter type presents a stricter geometry control that allows them to achieve more controllable mechanical performance.

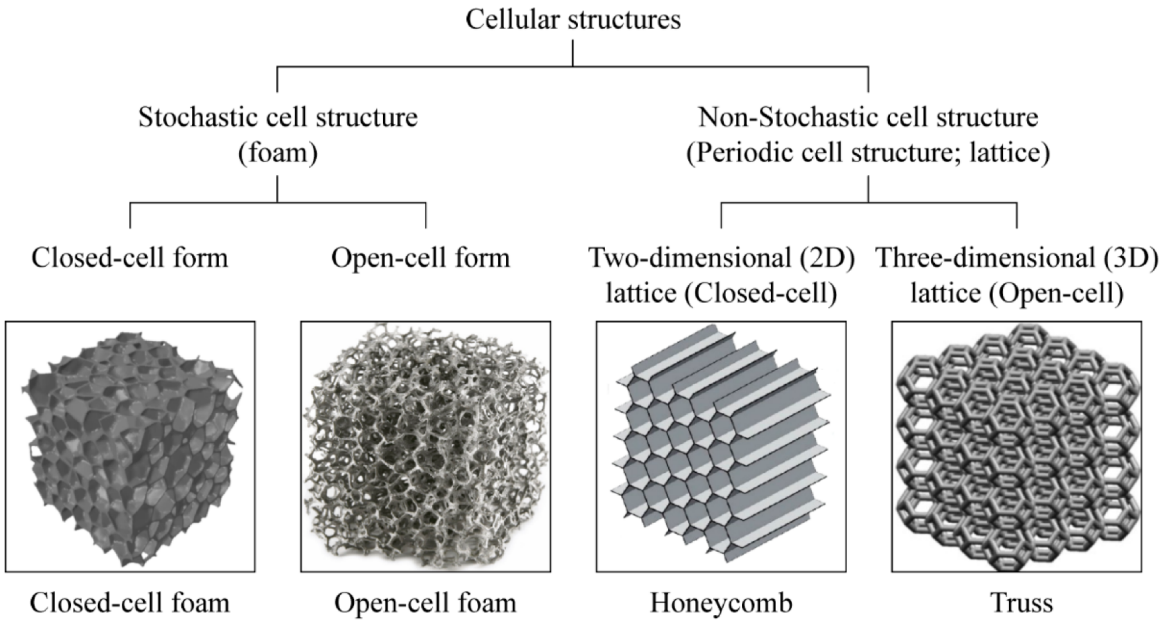


Figure 2.2: Most common types of cellular structures. Figure courtesy of [13].

Despite being analyzed as structures at the cellular level, cellular materials exhibit behavior akin to a homogenized meta-material at larger scales. This occurs provided the number of unit cells is sufficiently large and the cell size is small compared to the overall structure, a concept that will be further explored in Subsection 2.3.3.

Stochastic structures such as foams have a probabilistic nature impeding regular discretization. Honeycomb structures are 2D simplified lattices. Thus, from the different cellular structures, lattices possess the properties most suitable for the automated assembly of 3D digital materials. For this reason, a deeper dive on lattices is presented next.

2.3 Lattice structures

Lattices are three-dimensional non-stochastic cellular structures and can be classified according to how they are developed: Most commonly strut-based and TPMS lattices.

Strut-based lattices are composed of nodes interconnected by struts or beams. TPMS or triply periodic minimal surfaces are generated by differential geometry expressions that locally minimize the surface area for a given boundary. This generates continuously curved surfaces, beneficial for manufacturability. This surface is then thickened in the case of sheet-based TPSM or the enclosed volume is solidified in skeletal-based TPMS lattices [14]. These lattices are illustrated in Figure 2.3.

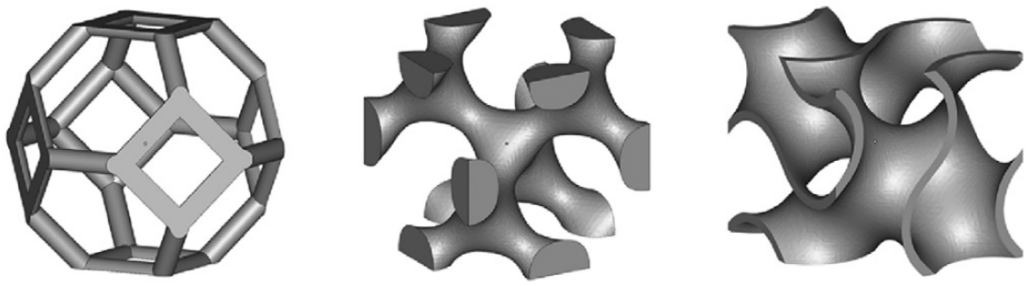


Figure 2.3: Three different unit cells. From left to right. a) Kelvin strut-based b) Diaomond skeletal TPMS b) Gyroid sheet-TPMS. Image modified from [15].

A comparison between the mechanical performance of the most commonly used lattices in Selective Laser Melting (SLM), an additive manufacturing technique for metal printing is presented in [16]. The correlation between experimental data collected from literature and the Gibson-Ashby model is found to vary from near-perfect to low correlation levels for different unit cell topologies, but generally and except for gyroids, diamonds and BCCZ (Body Centered Cubit with Z strut) a high correlation is achieved. A higher level of correlation is seen for the strength than for the modulus. [17] presents similar findings on the adherence of SLM lattice structures to the Gibson-Ashby model. Lastly, additional analyses are performed in [18] showing that a variety of discrete lattice structures exist between the linear and quadratic model limits. This limitation in predictability can be atributed to the low slenderness ratio of many 3D printed lattices and their mixed mode deformation nature. [12] presents a more complex multi mode deformation model that accounts for a bending, stretch and shear mix and better predicts properties scaling.

A key parameter in the understanding and modelling of lattices' mechanical performance is the loading of its elements. A distinction is made between lattices that internally carry the load via pure tension and compression or on the other hand by bending'of its members.

2.3.1 Deformation modes and the degree of connectivity

Stretch-dominated lattices carry load via pure tension or compression of their structural members. Bending-dominated lattices carry the load via bending moments within their structure, generally resulting in more compliant lattices.

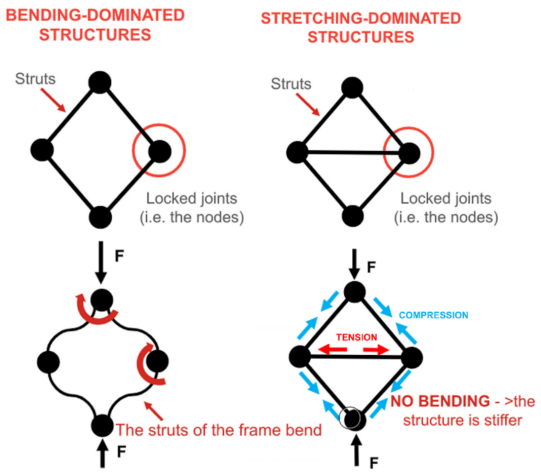


Figure 2.4: Bending dominated and stretch dominated lattices. Figure courtesy of [19].

Stretch-dominated structures offer greater stiffness and strength per unit weight given their smaller scaling factor with relative density. On the other hand, bending-dominated deformation structures possess a higher toughness, thanks to the lack of softening post-yield response due to buckling of the struts [20]. This buckling load can be determined in different manners such as with the Euler and Johnson expressions [21] for compression of columns. The Euler-Johnson expression interpolates between Euler's column buckling load and the material yield strength for low slenderness ratios, providing a better estimate for buckling of columns under compression than either method alone. Different methods exist for bending, mixed load cases and different geometries. Depending on the application, stretch or bending dominated may be preferred although in the case at hand, the stiffness and strength of stretch dominated results in more efficient structures.

Lattice geometry determines whether its behaviour is stretch or bending-dominated. This is mathematically represented for 3D strut-based lattices by Maxwell´s stability criterion based on node connectivity. This criterion M is presented in Equation (2.3), where b represents the number of struts and j the number of nodes. M is negative for bending-dominated and positive for stretch-dominated structures [22].

$$M = b - 3j + 6 \tag{2.3}$$

Some lattices belonging to the bending-dominated category, according to the criterion, show a mixed mode deformation with some degree of stretching. This is the case of Cuboctahedra or cubic lattices, although this results in marginal specific properties gain compared to fully stretch-dominated behaviour [19].

As for non-strut-based lattices, the compressive experimental analysis presented in [15] shows that sheet-TPMS lattice structures exhibit a near stretching-dominated deformation behaviour whereas skeletal-TPMS more closely resembles bending-dominated behaviour.

2.3.2 Mechanical behaviour

The general compressive stress-strain behaviour of lattice structures can be divided into three distinct regions that can be observed in the stress-strain curve presented in Figure 2.5

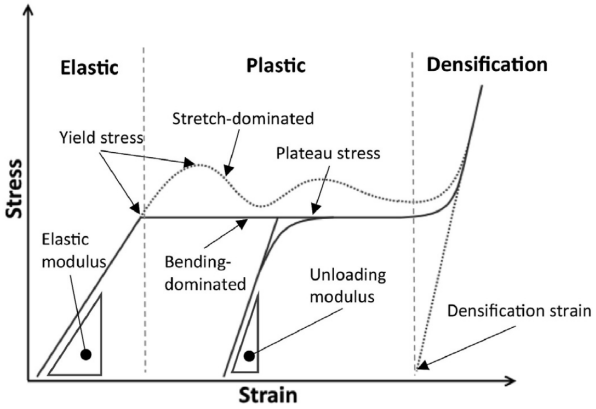


Figure 2.5: General compression stress-strain curve for stretch and bending-dominated lattice structures with its three distinct regions. Figure courtesy of [16].

Firstly a linear elastic region can be seen where stress increases linearly with applied strain. The slope of this area of the curve represents the Young Modulus, although, for more accuracy, it is recommended to measure the unloading modulus [10].
At some strain level, the elastic limit is reached and plastic deformation begins with a transition to a non-linear response as cells start to yield or buckle. In the case of bending-dominated structures, a plateau region can be seen where deformation keeps occurring with little variation in the stress and in stretch-dominated structures, this deformation may accompanied by oscillating and varying stress levels [16]. Lastly, after cells contact each other and constrain further deformation, the densification region begins, steeply increasing stress.

Three distinct failure mechanisms shown in Figure 2.6 can be distinguished; Successive cell collapse, diagonal shear and collective deformation. The same deformation mechanisms are generally seen for a cell topology independently of its relative density.

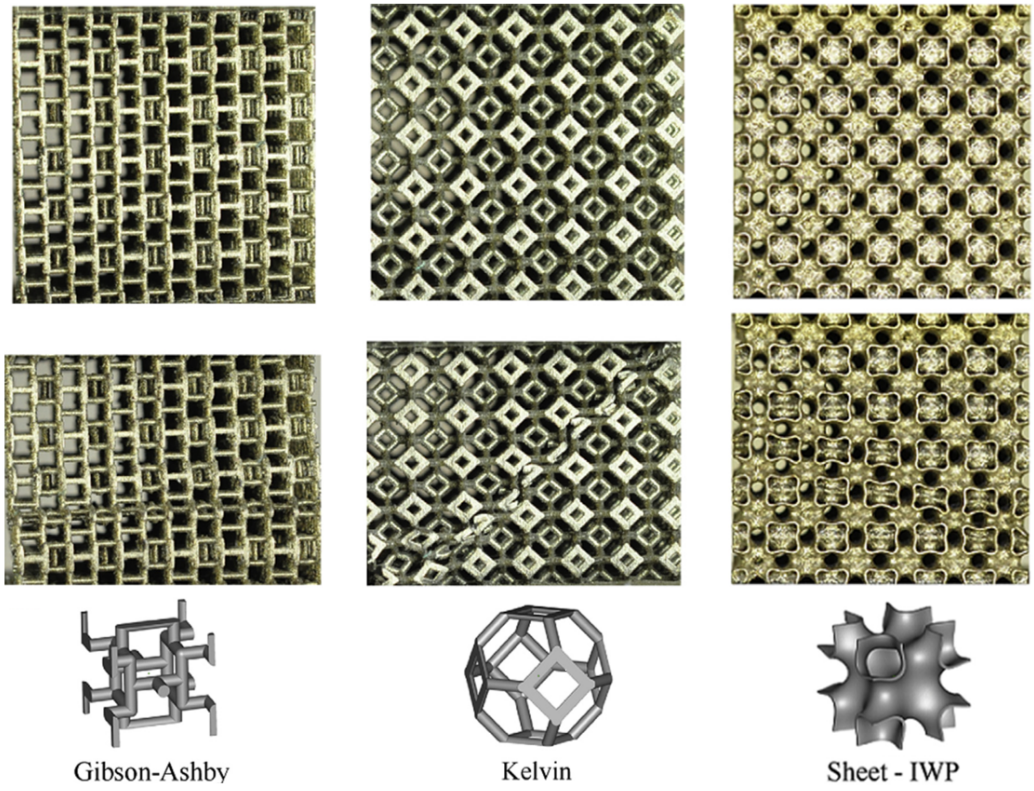


Figure 2.6: Different lattice morphologies showcasing different deformation modes. From left to right. a) Successive cell collapse b) Diagonal shear layer failure c) Collective deformation. Image modified from [15].

Successive cell collapse results from rows of lattices collapsing. This behaviour is observed in skeletal-TPMS gyroid lattices as well as Gibson-Asbhy lattices independently of density. On the other hand, in diagonal shear, layers fail diagonally following a shear band. This is the most commonly seen failure mechanism out of the topologies tested in [15]. Cells like Kelvin, Octet-truss, skeletal IWP or diamond and sheet primitive gyroids show this mechanism. Some lattice topologies like Octet-trusses or sheet-TPMS diamonds can generate double shear bands. Lastly, collective deformation results in uniform deformation of all the cells simultaneously and can be observed in sheet-TPMS IWP. This deformation continues until cracks propagate in the more brittle materials [23] or lattices are crushed. These mechanisms can be observed in Figure 2.6.

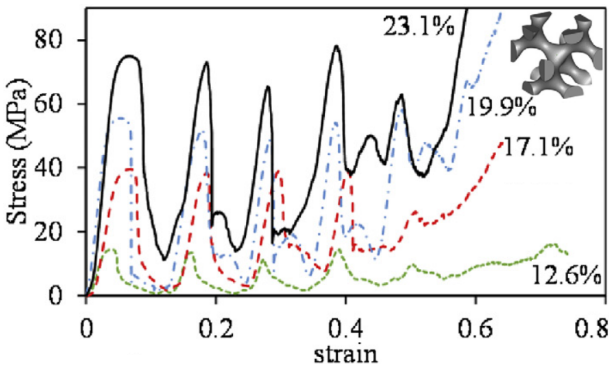


Figure 2.7: Stress strain behaviour for skeletal diamond lattice. Image modified from [15].

In light of the intricate failure mechanisms of lattice structures, it can be seen that the mechanical comparison between unit cells is more nuanced than a mere interpretation of the Young modulus, peak stress or toughness. Some failure modes are more desirable than others. An often undesirable failure mode is highlighted by the behaviour of skeletal-TPMS diamond structures showcased in Figure 2.7. This structure experiences a rapid shear line failure, followed by a sudden drop in the stress and consequent loading of the structure till failure of the next layer leading to a very irregular stress-strain profile that is highly unpredictable and can lead to large strain increases after a layer failures. This process is repeated until densification.

Failure mode aside, the results from [16] evidence which morphologies are generally advantageous. For instance, FCCZ or face-centered cubic with z struts unit cells have much greater performance than BCC or body-centered cubic lattices for the same density, especially in terms of modulus. However, as the experimental data is provided from different literature sources some deviations due to different methodologies, materials, equipment and imperfections can be expected. As demonstrated in [15] the design and SLM printed densities vary substantially depending on the lattice type. During the SLM printing process, heat creep and interaction between the solidified structure and the powder bed lead to partial melting and fusing of the loose powder with the bottom surface of the part. A phenomenon of higher importance at increasing unit cell surface areas. More consistent results can be expected from this publication, where a reduced set of 10 cell topologies are experimentally compared in uniaxial compression at different relative densities after SEM is used to assess the printing quality. Out of the 10 geometries analyzed, sheet-TPMS structures present the best mechanical properties followed by strut-based lattices and skeletal-TPMS. These differences between cell topologies are magnified at small relative densities, particularly for the modulus while at higher densities, as the mechanical performance gap narrows, other aspects such as manufacturability, fatigue and design freedom become more relevant.

Materials endure a plethora of loading scenarios, including combined loadings with multi-axial tension/compression and shear. Due to the novelty of the field, the ease of performing compression tests and the nature of these structures, compression dominates the literature. Limited tensile and combined loading behaviour for low-density micro BCC lattice structures is described in [24]. Density has a big influence on the tensile deformation behaviour and stress-strain curves. The lower-density lattice structures showed a true stress reduction after yield emanating from their sensitivity to micro-cracks and structural imperfections, process-dependent problems that are not as apparent in compression. On the other hand, the higher-density lattices showed a more traditional stress increase after yielding with necking until failure.

Thus far the analyses have focused on structures with small enough unit cells to be considered their own homogenized material with distinct properties rather than just a structure. In the next section, the limit with larger lattices will be explored.

2.3.3 Large form factor lattices, boundary effects

To simplify robotic assembly, larger lattices, resembling the scale of the geometries to be built are utilized. However, the previous mechanical analysis considers the lattice structure as a homogenized material with its own properties, which only applies to lattices significantly smaller than the structure's features.

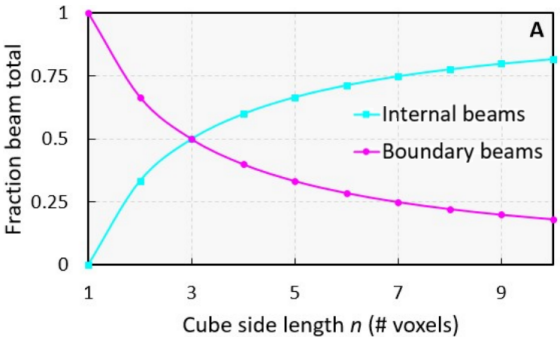


Figure 2.8: Internal vs boundary beams for a strut-based lattice structure. Figure courtesy of [25].

As more voxels or unit cells are added, the degree of connectivity increases, constraining the lattices and effectively providing more rigidity, increasing the modulus until the continuum value is reached. The evolution of the ratio of internal and boundary beams for cubot lattices is showcased in Figure 2.8. The mechanical performance of large-scale lattice structures has been characterized for a variety of materials and manufacturing methods, from additively manufactured PLA, carbon fibre reinforced PA12 or liquid crystal polymer (Vectra) octahedral cells in [26] to glass fibre reinforced PEI injection moulded octahedrals in [18] or more varied injection moulded geometries such as Cubots in [25].

The scaling of the Young modulus and the stress-strain relations against lattice count for the injection moulded cubots is showcased in Figure 2.9. As can be seen after the cube side length, that is, the number of lattices per side of the cubical material sample, increases over five, the continuum values are approached and at ten or more lattices cube side length, the properties have converged to that of the continuum.

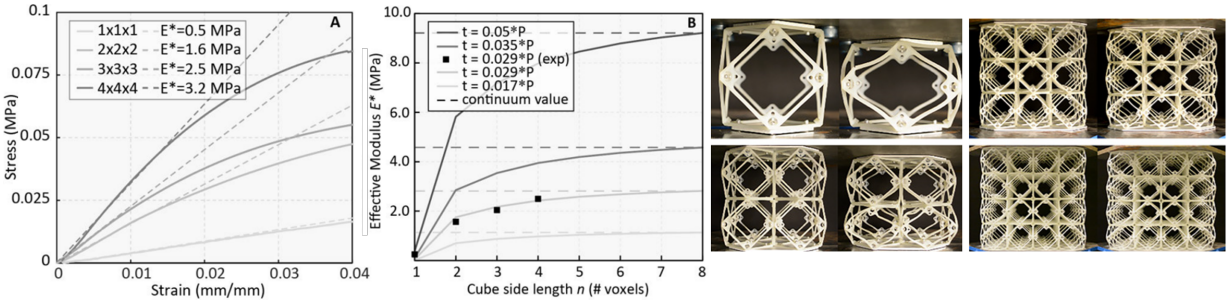


Figure 2.9: Properties of cubot strut-based lattice structures for increasing number of lattices. From left to right: A) Stress-strain, B) E-modulus, C) Compression tests images. Figures courtesy of [25].

2.3.4 Metamaterial properties

The geometry of these unit cells can be tweaked to exhibit varied behaviours, some not achievable in traditional materials. Figure 2.10 presents 4 metamaterial unit cells with exotic behaviour like auxetic unit cells that contract from all sides when compressed and chiral lattices that rotate when loaded in tension.

Figure 2.10: Metamaterial unit cells. From left to right; Stiff, compliant, auxetic and chiral. Figures courtesy of [25].

2.4 Robot assemblers and their architectures

Digital materials present a set of advantages that simplify their autonomous assembly namely embedded metrology and parallelization. This chapter dives into the existing literature for the assemblers, showcasing different robot hardware configurations and their organizational architectures into collectives.

2.4.1 Robotic hardware platform

A robot is a programmable machine capable of carrying out complex tasks automatically. In this context, the robot's main role is to traverse the digital material as well as placing and removing lattices from the structure. However, this does not strictly constrain its shape or actuation. From flying drones to robot arms or walking robots, a plethora of hardware platforms have been described for autonomous construction in the literature. [27] compares various robotic platforms and organizational schemes to natural buildings. Different such platforms are showcased in Figure 2.11.

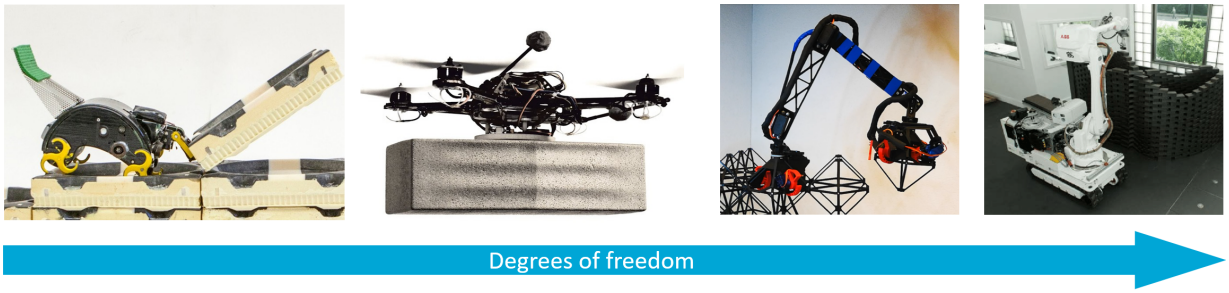


Figure 2.11: Robotic assets for discrete assembly. From left to right: A) Termes autonomous robotic system [28] B) Aerial construction of brick structures [29] C) Inchworm lattice traversal robot [26] D) Mobile robotic brickwork [30].

The robot's positional precision requirements are relaxed by the use of digital materials with embedded metrology. Benjamin Jennet [11] defines this robot-material symbiosis with the term relative robotics.

> *"Relative robots are task-specific, and their geometric, kinematic, and dynamic properties are a function of the periodic environment in which they operate. Relative robots seek to leverage this environment for simplification and improved reliability."*[11]

Relative robots do away with complex high-precision solutions like the autonomous mobile robot (AMR) mounted robot arm-mounted on the rightmost picture in favour of simpler, cheaper and more scalable solutions. On the opposite end, low DOF robots generally have reduced flexibility for climbing, turning, moving diagonally and navigating complex 3D vertical spaces efficiently. As a result, inchworm robots have been popularized in relative robotics because of their balance between complexity and simplicity. Inchworm robots are featured in the assembly of digital materials in [31] [32] [33] [26] [34], for bridge inspection [35] and as an alternative to more traditional truss climbing robots [36] [37]. Inchworm robots present a joint configuration often seen in the legs of more complex robots such as the spider-looking quadruped in [38].

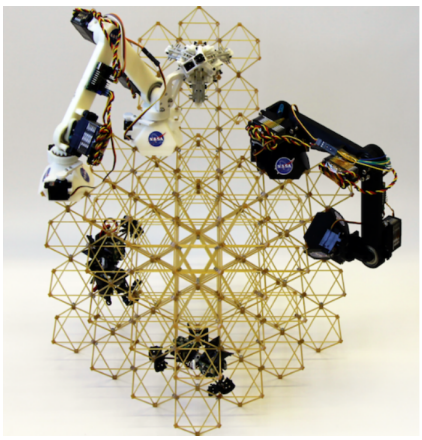


Figure 2.12: BILLE and MOJO. Image courtesy of [39].

A first-generation robot duo composed of the inchworm robot BILL-E (Bipedal Isotropic Lattice Locomoting Explorer) [31] and the lattice crawler MOJO (Multi-Objective JOurneying robot) [40] presents the idea of splitting the work of lattice placement and securing. BILL-E walks on the digital material transporting and placing lattices and MOJO crawls through the structure to secure them to their neighbours. A second-generation BILL-E robot was developed in [32] for different lattice geometries. A larger scale third generation robot duo made of the recently presented inchworm SOLL-E (Scaling Omni-directional Lattice Locomoting Explorer) [33] and the crawler MMIC-I (Mobile Meta-Material Interior Co-Integrator) [41] addresses some limitations. These robots are larger with similar kinematics but a better design. For instance, SOLL-E uses closed-loop controlled BLDC motors instead of servos.

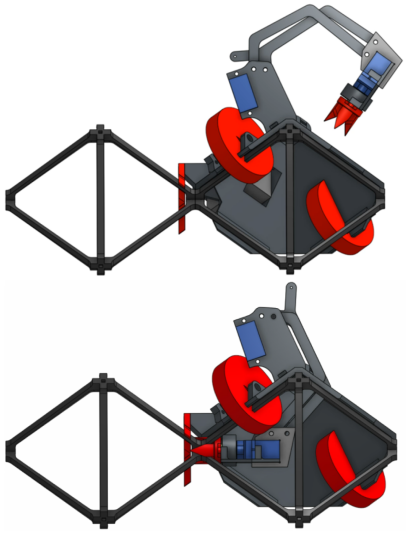


Figure 2.13: Screwdriver arm. Image courtesy of [26].

These two separate roles have been coupled into a single inchworm robot in [26]. Here, a small single-degree-of-freedom arm, holding a torque limited servo powered screwdriver is added onto the inchworm robot grippers or feet. This arm can be moved into the lattice building blocks to perform screw-in and out operations as showcased in Figure 2.13. However, this requires multiple robot moves to reposition the gripper with different orientations resulting in a long time to screw each lattice and can have difficulty screwing to the bottom node. This robot was developed by Alex Luijten in ETH with Dr. Kunal Massania's advisory and will be utilized as a baseline design in the robot design in Chapter 5.
To perform an automated assembly job, a hardware system composed of robots and lattices works in conjunction with a software platform where the user or stakeholder can specify the job definition. These hardware-software architectures can take multiple forms.

2.4.2 Robot organizational architecture

From an autonomous robot or agent perspective, a classification between deliberate and reactive systems can be made according to how sophisticated the agent's reasoning is. Whereas deliberate agents, plan optimal strategies and execute them in a deliberate thinking-like process, reactive agents respond to certain stimuli with pre-set behaviours. The line between these two types is blurry with some being a hybrid incorporating elements from both. In fact, in the field of Distributed Artificial Intelligence or DAI it is widely accepted that agents should incorporate both reactive and deliberate abilities [42]. Different implementations and mixes can be seen. For instance, [43] mixes both reasoning types by deliberately selecting which reactive behaviours are followed and on the other hand, the system presented in [44] deliberately selects when to follow reactive or goal-directed plans.

A distinction must also be made between global and local perspective agents. Global perspective robots know the status of the complete operational domain by means of global or centralized communication limiting its application to small-scale scenarios, where the global status can be defined with a sensible number of bits. Global perspective systems preferably run in a single planning machine to reduce the need for continuous state communication.
On the other hand, local perspective robots employ a set of sensors and local area communication to develop an understanding or measurement of stimuli around their neighbourhood. The lack of information and distributed nature of local perspective systems hurdles the planning of optimal task execution.
Some or all elements of this deliberate and reflective planning can be performed externally to the robot, that is on a separate computing unit at the cost of a constant communication stream. This represents a tradeoff between computation and communication. Centralized computations facilitate global perspective agents.

Three major types of architectures for organizing a set of robots for a construction process exist; Centralized, decentralized and distributed systems. Both centralized and decentralized systems present the possibility to plan at a higher than robot or worker level with an external planner, while distributed only allow for planning at the robot level.

Centralized systems
Centralized systems represent the most straightforward and simple to implement architecture consisting of a single central master controller that plans and directs all the robot workers in a traditional master-slave configuration [45]. Depending on the degree of autonomy, the orders provided range from low to high level. For this system to work, all robots must maintain a connection with the central planner. Having only one main central unit this system is highly sensitive to the failure of components as there exist single

failure paths. Additionally, the impossibility of adding more central planners hinders the ability to scale horizontally by parallelization, only allowing vertical scaling by increasing the computational power of the planner, thus limiting scalability. A diagram of this architecture is provided on the left-hand side of Figure 2.14.

Decentralized systems
Decentralized systems are a variation of centralized systems in which multiple planners can be utilized simultaneously. Each planner controls a group of robot workers in the same way as in the previous scenario. Distributed systems may also incorporate hierarchical layers where a higher-level master may relay orders or information to a robot via other masters. This system allows for horizontal scaling thus expanding scalability as well as reducing the sensitivity to failure of components. Nevertheless, an additional level of complexity is added over centralized systems.

Distributed systems
Distributed systems present a dichotomy with the previous master-slave or server-client architectures. In distributed systems no planner is present and the robots themselves make their own decisions. The behaviour of the complete system is not controlled by a central planner but by the collective decisions of all the intervening agents. This type of system presents a high scalability potential but more difficulties exist to reach consensus and perfect coordination. In distributed systems, complex behaviours can emerge from the individual contributions to the collective behaviour like in biological construction.
Although no direct control from a central user can be observed, the final build geometry can be defined by providing the robots with rules and a build definition in different forms such as gradient fields that dictate their future behaviour as further elaborated in Subsection 2.5.4. Often these fields do require to be computed in a centralized medium before being distributed to different agents.

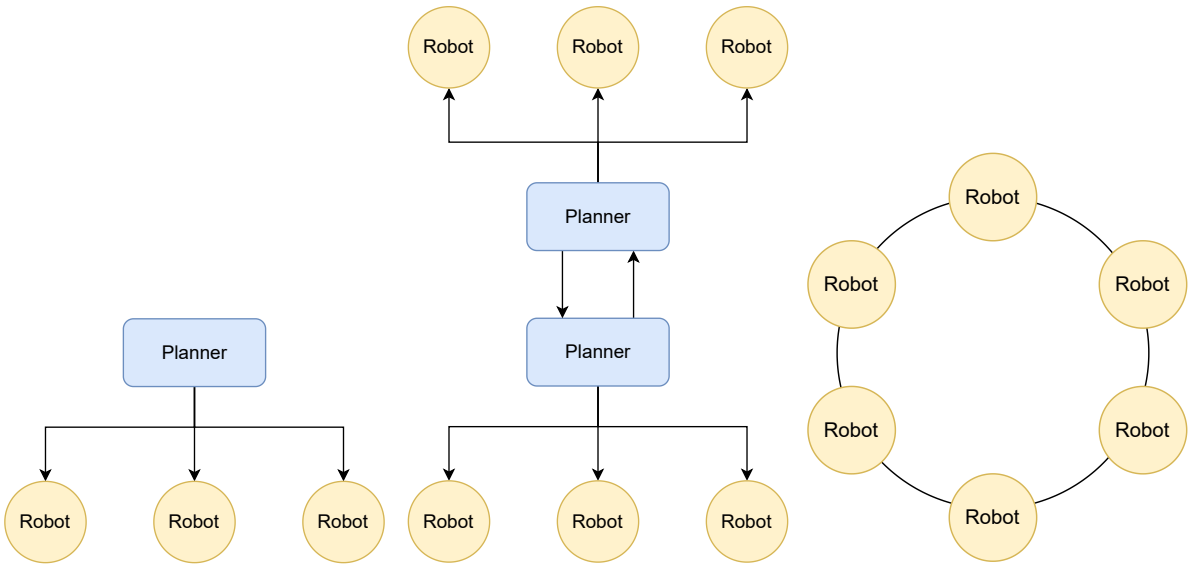


Figure 2.14: Most widespread robotic types for discrete assembly of digital cellular meta-materials. From left to right: centralized, decentralized and distributed.

Distributed systems and self-organization
Self-organization is the spontaneous ordering tendencies through which independent agents organize their collective behaviour and global order emerges not through external intervention but thanks to interactions amongst themselves [46] [47] by means of emergent behaviour or explicit cooperation [48] . It is characterized by the absence of explicit external control, dynamic operation due to its evolving organization and generally speaking but not necessarily distributed control [49]. In nature, self-organization can be seen from groups of thousands to trillions of individual elements resulting in a variety of forms with examples at different scales such as self-assembly of crystals, rotary motors of bacteria flagella or ant colonies capable of building structures such as bridges with their bodies [50].

Self-organization is most interesting but not limited to local perspective agents and distributed systems where the lack of global information on the decision-making agents hurdles the planning of optimal task execution.

A plethora of mechanisms for self-organization exist. From Bio-inspired mechanisms such as stigmergy or reinforcement learning to social approaches like trust-based or reputation systems, two large categories can be distinguished; Strong and weak self-organizing systems [51]. Strong self-organizing systems are those where there is no explicit central control such as in the case of distributed architectures and weak self-organizing systems are those where there is re-organization under internal central control as can occur in centralized and decentralized systems [49]. An example is the start of the termite mound's royal chamber, where the construction is dominated by pheromone fields released by the queen, providing a centralized level of control to the nest build process [52] [53]. Two of these self-organization approaches or mechanisms are now briefly introduced.

Stigmergy
Stigmergy is one of the first indirect coordination and self-organization mechanisms subject to the spotlight of research. It was introduced in biology to refer to termite and social insect behaviour before the fields of robotic swarms and collective robotics were relevant. [54] defines it as:

> *"Stigmergy is an indirect, mediated mechanism of coordination between actions, in which the trace of an action left on a medium stimulates the performance of a subsequent action."* [54]

The trace of an action left on a medium is portrayed in biology by the presence of chemicals such as pheromones. Stigmergy suggests that past individual behavior influences the collective´s present and future behavior by leaving detectable traces, allowing the development of time-dependent cooperation.

Stigmergy also allows for some level of centralized control in apparently distributed systems. An illustration is the previously introduced example of termite mound construction that is dominated by pheromone fields released by a central authority, the thermite queen. This centralized control proves of high value for automated manufacturing as precise control of the final geometry is desirable.

A practical example of scalability limitations is presented in [55], where robots coordinating through stigmergy are tasked with gathering a set of objects, and the efficiency of the process is compared for groups of different sizes. It was found that for the simple implementation studied after a critical number of agents is introduced, the meantime to accomplish a task increases due to an exponential increase in interactions between robots which are time-consuming, thus group size is a critical factor for local perspective and stigmergy approaches and increasing it can lead to decreasing performance if not done adequately.

Flock behaviour
Other organizational mechanisms exist. In flock behavior, agents measure the presence and position of others via visual, sound, EM or other signatures. As an example, the FluxPose 6DOF positional magnetic sensor is a great example of a powerful sensing platform for flock behavior. This allows for a more dynamic and fast response that does not leave a trace through time as Stigmergy does but suffers from limitations on introducing centralized control or time-dependent phenomena. Flock behavior can be observed in birds and fish group movement coordination. In the field of autonomous assembly, the Kilobots from [56] have exploited such behaviors to build complex geometrical shapes. These and more strategies are not exclusive and can be mixed together.

2.5 Work planning and graph structures

This section is largely based on Steven LaValle's planning algorithms book [57] and the artificial intelligence lectures from [58].

Two large types of problem-solving strategies can be distinguished; Knowledge and search-based.The first one requires preexisting knowledge of the system and can not be easily generalized. For instance, we may ask someone whether they know how to solve a Rubik's cube, that is whether they have memorised a set of movements that can algorithmically be combined with logic rules to solve a particular puzzle. Alternatively, a general computationally intensive search-based method, that maps all the puzzle´s states and navigates them can be utilized to optimally solve a Rubik´s cube with no preexisting knowledge of algorithms. As demonstrated in [59] methods that mix the two strategies are possible and often beneficial.

Planning strategies can be classified into discrete and continuous. While a GPS personal navigation device can work with the abstraction of a discretized space, where the user position is binary between nodes or intersections, a self-driving car must navigate the analogue continuum of the intersections and roads, defining a plan of action at every point. Often, the continuum of reality can be discretized into small elements, in fact, modern digital robotic sensing is discrete in nature, due to the finite resolution of analogue digital converters used in the sampling of transducers [60]. Nevertheless, it is computationally impractical to employ discrete planning on very fine resolutions when continuous planning is possible. Due to the nature of digital materials, discrete planning is more suitable and is explained in greater detail.

The problem of automated construction can be seen as a state-search problem where the lattice structure is transitioning from an initial state x_0, that is, a distinct situation or configuration of the world to a different goal or final state x_g. The world can be transformed from one state to another through actions, such as placing or removing a lattice, that are chosen by the planner. Actions, denominated u produce a new state x' when applied on x, as specified by a state transition function shown in Equation (2.4).

$$x' = f(x, u) \tag{2.4}$$

$U(x)$ represents the action space, meaning the set of all actions that can be performed for a state x. A planning problem is feasible, meaning that the goal state may possibly be reached from the initial state when the state space X contains an initial state x_i and a goal set X_g and for every state x in X there exists a finite action space $U(x)$ with its corresponding state transition function.

The set of states is commonly organized or visualized in a graph form like shown in Figure 2.15, where the nodes represent the states and the edges the interfaces between states, that is, the actions to transform from one state to another. We can distinguish between directed and undirected graphs. In directional graphs, the edges may allow traversal in only one direction and in unidirectional graphs, edges are always two-way. Essentially whether every action $x_k = f(x_i, u)$ has an inverse $x_i = f(x_k, u^{-1})$ with the same cost.

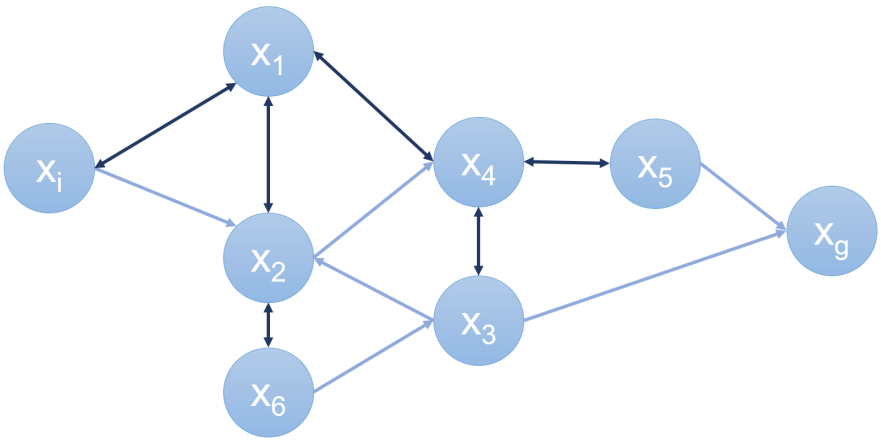


Figure 2.15: Graph representation of the state-space. Nodes or dots represent the states, and the arrows or edges are the actions or interfaces between states. Light blue arrows are directional, and dark blue bidirectional.

Graph search algorithms can then be utilized to find an optimal plan. This search can be performed on an already built or discovered graph or alternatively the graph discovery and search steps can be performed simultaneously by progressively discovering the graph as the search advances. The search process is based on dynamic programming and Bellman´s principle of optimality that states:

> *"An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."* - Richard E. Bellman [61]

In simpler words, an optimal policy can only consist of optimal sub-policies and thus the search problem can be broken down into sub-problems that can be solved to reconstruct the optimal solution.

Different strategies can be employed for traversing the graph, most notably they can be classified as forward search, backward search and bidirectional search. A general or unified view of search methods is presented before diving into more detail.
At any point in the search, we can classify the states into 3 groups. Unvisited, dead, Alive:

- **Unvisited:** States that have not been visited yet.
- **Dead:** States that have been visited, meaning every possible next state $x' = f(x, u)$ is discovered and added to the alive set.
- **Alive:** States that have been discovered but are yet to be visited.

Initially, the only alive state is the initial state x_i with every other $X - \{x_i\}$ being unvisited. Alive states are stored in a priority queue Q that is ordered with a priority function that constitutes the primary distinction between the different search algorithms. The highest priority alive state from the queue Q is visited and removed progressively. When visiting the state, the corresponding edges are created and neighbouring states are added to the priority queue with their corresponding cost to reach, which is equal to the visited node cost plus the edge weight. This cost may be used in the ordering of the priority queue. If X_g lies within Q the graph will have been explored to the point where a solution is available, not the optimal solution but one nevertheless. Once X_g is visited thanks to the principle of dynamic programming, the optimal path will have been found. By saving which state x each next state x' comes from and which action was used to traverse $x' = f(u, x)$, the solution can be easily traced back once it is found.

For a k-step plan, that is, a fixed length plan with k steps, the stage-additive cost functional from Equation (2.5) can be developed. The cost to execute a certain plan π_K is $L(\pi_K)$. The term $l(x_k, u_k)$, represents the cost l to transition from state x_k to x', via the application of the action u_k and it always yields a real value for any x_k in X. F represents the final stage. If x_F is in the set of goal states X_G, the final term $l_F(x_F)$ will be zero and otherwise infinite. This term accounts for the fact that some plans may never reach the goal state and thus their cost is infinite as they never reach the goal.

$$L(\pi_K) = \sum_{k=1}^{K} l(x_k, u_k) + l_F(x_F) \tag{2.5}$$

The cost to go from stage k to F can be minimized by doing the following where the cost L_F is just $G^*_F(_K)$.

$$G^*_k(_K) = \min_{uk,\ldots,uk} \left\{ \sum_{k=1}^{K} l(x_k, u_k) + l_F(x_F) \right\} \tag{2.6}$$

2.5.1 Continuous planning

Before advancing onto search algorithms a short introduction to planning on continuous domains is presented. Research on planning can be divided into two categories; Roadmap and cell decomposition methods [62]. A brief outlook on both of these is now provided.

Roadmap methods

Roadmap methods are predominantly employed for path planning. They are solved in the configuration space, that is, a representation in which the robot or actor is represented by a point and its planned movements are described by a continuous path.

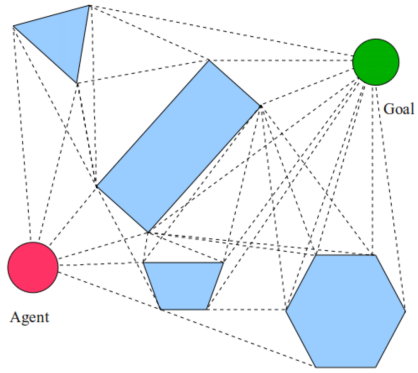


Figure 2.16: Visibility graph. Courtesy of [63]

The roadmap is the set of curves that interconnect all start and goal points by paths as shown in Figure 2.16. Different roadmap implementations exist such as the Visibility graph method or Voronoi-based trajectories with the main difference being how the space between obstacles is split. In the Visibility graph method, the obstacles are first thickened by the agent's radius and the roadmap is built by connecting the nodes or vertices of the different polygonal obstacles as well as the initial and final or goal positions. A path is then found from the roadmap. Paths are viable when they don't intersect another of the polygonal obstacles. When a path is possible there will exist one or more lines from the initial to the goal position that do not intersect obstacles. The shortest path can then be selected.

Cell decomposition

Cell decomposition consists of discretizing the continuous space to apply graph search algorithms. This method involves a four-step process, as depicted in the exact cell decomposition approach in Figure 2.17. The initial step involves decomposing the space into discrete cells, based on the vertices of obstacles. Subsequently, a connectivity graph is constructed, where each cell is represented as a node and the connections between them as edges. The third step involves determining the most efficient path from one cell to another using the graph. Lastly, specific points within each cell are selected to be used in the continuous path.

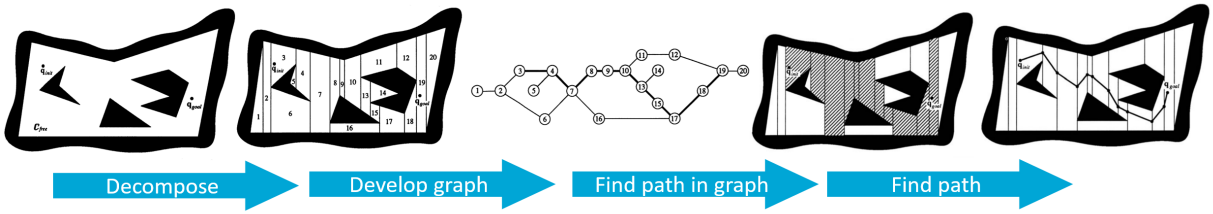


Figure 2.17: Exemplified cell decomposition process utilizing exact cell decomposition. Figure edited from [64].

2.5.2 Search algorithms

The introduction to this chapter provided a generalized overview of search algorithms where it was explained that different implementations differ primarily in the ordering of the priority queue Q utilized for selecting which neighbours should be visited next. The following sections explain different algorithms.

Exhaustive search: Bellman-Ford algorithm

Exhaustive search are brute-force algorithms that evaluate (or exhaust) every option. No guesses of which nodes are best to evaluate first are performed, thus requiring the evaluation of all of them.

Bellman-Ford works on the principle of relaxation like the famous Dijkstra's algorithm described in Subsection 2.5.2. The cost approximations are continuously iteratively improved until they converge. The peculiarity of this algorithm is that it goes through each vertex, computing the distance to their neighbours in every iteration, thus it's exhaustive since all vertices will be checked and not only once. This is computationally expensive but unlike other algorithms allows optimally navigating graphs with negative edges. Looking at every edge at every iteration guarantees a solution is found in $V - 1$ steps. In each step, each node is visited meaning the time complexity is $O(|V| \cdot |E|)$ where V and E are the numbers of vertices and edges respectively.

Non-exhaustive search algorithms

Non-exhaustive algorithms do not require checking all possible nodes since it's possible, though not guaranteed, to find the shortest path to a node before the complete graph is exhausted.

Breath-first search (BFS) and Depth-first search (DFS) and iterative deepening
BFS and DFS are some of the simplest search methods. In BFS, the priority list Q behaves as a FIFO, First-In-First-Out queue generating a uniform wavefront. Breath first search exhausts all k-step plans before investigating k+1 step plans, guaranteeing that the first one found contains the minimum number of steps. For state spaces where all costs $l(x_k, u_k)$ are the same the first solution found must be optimal.

On the other hand, in DFS, the priority list Q is LIFO, Last-In-First-Out. The exploration of the state space is spearheaded in a single direction as opposed to the uniform wavefront of BFS. It's more biased towards investigating long plans early on, meaning that the first plan found won´t necessarily be optimal.

The time complexity of both algorithms is $O(|V| + |E|)$ though it can be difficult to compare the performance of these two methods. If not run till exhaustion BFS is preferred in cases where the solution is known to be close to the starting location while DFS is preferred for other cases. Very wide trees benefit DFS since does not need to store a complete tree level, reducing memory constraints.

These two strategies can be mixed using iterative deepening, where depth-first search runs till a certain depth to then transition to a different branch as in BFS, balancing the pros and cons of both methods.

Dijkstra's search
Dijkstra is a well-known path-finding algorithm. It is a specific case of A* presented in Subsection 2.5.2 but using a heuristic identically equal to 0. Interestingly, it was developed by the Dutch computer scientist Edsger Dijkstra who was inspired by finding the shortest way to travel from Rotterdam to Groningen when shopping in Amsterdam.
Instead of performing a simple FIFO or LIFO order on the priority queue Q, Dijkstra´s algorithm uses a greedy approach taking the nodes with the lowest cost first.

The complexity of this algorithm depends on the nuances of the implementation. Utilizing a conventional array priority queue, the time complexity will be $O(|V|^2)$, however, it can be reduced to $O(|E| + |V|\, log\, |V|)$ by using a Fibonacci heap [65].

Heuristic or informed search algorithms
Heuristic search algorithms use a function to predict the most promising nodes for efficient space search. This "bird's eye view" approach allows to focus on likely solution areas, unlike the "worm's eye view" of exhaustive methods. This is effective if the graph is not random. This section will focus on best-first search with A* and greedy flavors.

A* search algorithm
A* belongs to the family of non-greedy best-first search algorithms together with others such as B*.
The priority queue is ordered according to the cost functional f, which is the sum of the cost to get to the current node from the start g as in Dijkstra's plus an approximate cost to reach the goal node, the heuristic h. It can be calculated for state x as shown in Equation (2.7).

$$f(x) = g(x) + \epsilon h(x) \tag{2.7}$$

Where:
- **f(x):** The total cost utilized for ordering the priority queue.
- **g(x):** Cost of the path from the starting state x_i to the current state x
- **h(x):** The heuristic function estimates the cost from the current x to the goal state x_g. The optimal solution is found as long as the heuristic is admissible, meaning it doesn't overestimate the cost.
- ϵ : Heuristic relaxation factor. The admissibility criterion is relaxed by a factor ϵ to provide a sub-global minimum solution within a margin that can be computed much faster. Over-relaxation, like excessive laxity in work, can lead to poor outcomes. Dynamic weights could be used.

For states with the same $f(x)$, both LIFO and FIFO strategies can be employed. Generally speaking, LIFO is preferred as A* will behave depth-first amongst equal cost paths, returning a solution faster. Imagine there is symmetry in the problem. A FIFO priority will develop both symmetric paths simultaneously while

LIFO will focus on a single one. The time complexity of A* depends on the particular Heuristic employed and its time complexity. Expensive heuristics will heavily impact the performance.

A comparison between path planning using uninformed Djkistra's and heuristic A* algorithms is showcased in Figure 2.18 using the online visualization from [66]. The blue nodes represent the dead states, green alive states, white unvisited, and grey obstacles. Dijkstra's algorithms on the left check a larger portion of the space as its lack of heuristic makes it expand in an iso cost sphere resulting in 1410 operations and 1.6ms runtime compared to the 391 operations and 0.5ms run-time of the A* algorithm. Despite both cases showcasing a different path, both are optimal and the same length. Remember that admissible heuristics as employed in this case guarantee global optimality of the solution.

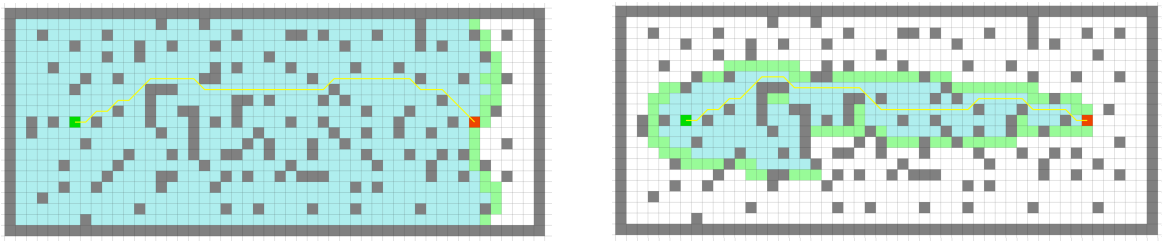


Figure 2.18: Comparision of Dijkstra (left) and A* (right) algorithms on a pathfinding problem from the dark green to the red squares. Grey squares represent obstacles, white unvisited states, blue dead ones, and green alive states. Figure generated using [66].

Greedy best-first search
Similarly to how Dijkstra uses no heuristic $h(x)$ from Equation (2.7), greedy best-first search does without $g(x)$. The priority queue is purely ordered based on the heuristic. This algorithm is greedy since it investigates the most promising path at every moment. It offers high computational performance and in the worst case, it behaves like the uninformed depth-first search from Subsection 2.5.2 with a more expensive cost estimation. However, it offers no guarantees of finding the optimal solution.

2.5.3 State space explosion

Linear increases in the size of properties of a problem can lead up to exponential growth of the state space. While no universal solutions exist to reduce the size of the state space, techniques that minimize computational power and memory like hash tables can linearly delay issues. Methods that eliminate the need to check some parts of the state space may allow an exponential increase in the explorable domain. It is possible to perform a simultaneous graph discovery and search thus reducing states can manifest in both steps.

The number of states can explode for even relatively simple problems like a Rubik's cube. The central pieces of each face don't move. There are 12 additional edge pieces with 2 rotational states each and 8 corners with 3. Thus a cube has $12! \cdot 2^{12} \cdot 8! \cdot 3^8$ or $5.2 \cdot 10^{20}$ states. However, many of these states form separate graphs that don't intersect meaning they don't contain the same nodes and can't be reached from the starting (solved) state. an additional constraint must be added. We can not freely determine the rotation of one of the edges and corners as well as the position of one corner. In this way, the number of states becomes $12! \cdot 2^{11} \cdot 8!/2 \cdot 3^7$, that is, $4.3 \cdot 10^{19}$. [67]

Extensive research has successfully mapped all states of this puzzle, identifying solutions for every initial position. This required various methods and optimizations, including leveraging symmetries to condense the search area, applying mathematically derived bounds to the maximum number of steps in the optimal solution path, and employing bidirectional search techniques. As a result and after 30 years of CPU time, all solutions have been identified and the maximum number of moves required to solve the cube, known as God's number, has been established at 20 [68].

Automated assembly
In the planning of an automated assembly job, the size of the space state varies depending on the design freedom and the size of the structure to be built. Firstly, consider the problem of building a structure

in a 3x3x3 grid volume. Placing any lattice at any point in time, a total of $27!$ or $1.08 \cdot 10^{28}$ states exist. However, lattices can only be placed next to existing ones, and it must start in a predefined location along the edge. In that case, and assuming no lattices are removed after being placed, the number of states decreases to an upper bound of $5^2 7$, or $7.45 \cdot 10^{18}$. This is calculated from the graph´s depth and degree of branching which ranges between 1 and 5. Even a quick upper bound calculation reveals the immensity of the design state space of such a simple assembly task.

Three strategies can be employed to reduce the severity of the state space explosion. Firstly, limits can be imposed on design freedom. Secondly, heuristics can be employed that guide away from the exploration of a large section of the state space. Lastly, some logic-based rules can be utilized to decide parts of the solution without applying a search procedure.

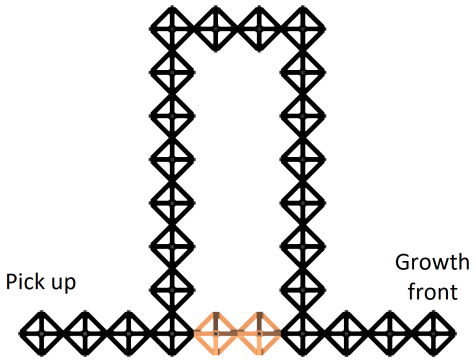


Figure 2.19: Logistics lattices in accordion structure

An example of such a limitation is whether lattices not belonging to the target structure can be placed temporarily to provide infrastructure to the assemblers and speed up construction. The placement of such lattices must be bounded to prevent the state space from growing to infinity. Next, two configurations that benefit from this are presented.
Figure 2.19 presents a scenario where an accordion structure is constructed. The orange lattices represent temporal or logistics lattices that are placed outside the target structure definition to accelerate the transit of robots during assembly and are dismantled afterward. Similar to support material in 3d printing, additional lattices can be placed to act as a structural or logistics backend.

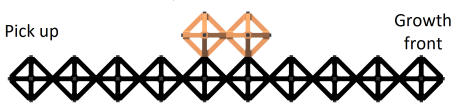


Figure 2.20: Logistics lattices for crossing

A simpler scenario is shown in Figure 2.20. This thin strip is a choke point where two robots can not pass side by side. Robots can yield, they can interchange or hand over the lattices like in a bucket chain as shown in Subsection 2.5.4, or additional lattices can be added to allow overtaking.
These cases can also take the form of tunnels where robots navigate inside the structure´s pores.

Simplifications to the state space can have a significant impact on the performance of the assembly process. A heuristic function can help inform which actions to take. Heuristics are simple to implement in path planning problems as they can be a simple geometric distance, but they become harder to define in more complex problems.

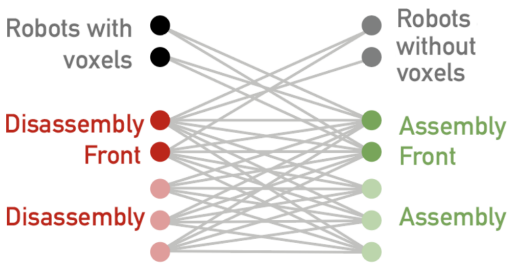


Figure 2.21: Bipartite matching graph utilized for the heuristic in [39]

Two heuristics for the reconfiguration of a discrete voxel structure are proposed in [39]. One is based on the moving of the centre of mass of the reconfiguring structure and the other uses the min-cost of a dynamic bipartite matching graph. A bipartite graph matches two sets with edges, ensuring no edges have the same endpoint and minimizing total cost.
In the case at hand, the robots with voxels are matched to the lattices in the assembly front and the robots without lattices are matched to a lattice in the disassembly front as presented in Figure 2.21.

State search is not limited to finding the construction steps but can also be used for path planning, meaning finding the set of movements for the robot to traverse from one location to another. In this case, the nodes would represent different connection points the robot can attach to and the edges the possible movements to traverse them.

Thus far, it has been assumed that a centralized authority can perform the state space search and command the worker robots. Controlling these large collectives from a centralized authority can prove challenging for this reason other methods also exist like gradient fields.

2.5.4 Comparison to potential and gradient fields

An approach similar to that of stigmergy presented in Subsection 2.4.2 where robots utilize field maps containing long-term planning strategies and can employ local sensing to come up with small modifications can be developed.

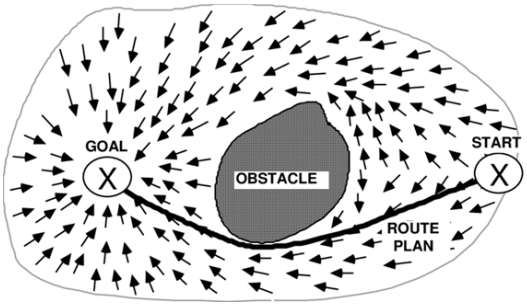


Figure 2.22: Overview of path planning utilizing potential fields. Image courtesy of [69].

The illustration in Figure 2.22 demonstrates the use of potential fields for path planning. This potential field allows robots to navigate to a goal location from any starting position within a particular domain. akin to how particles or charges move in a medium. For those of us who have studied aerospace engineering, this concept might remind us of classical thin airfoil theory, where sources and sinks are used to create potential fields that simulate particle flow around objects like airfoils to calculate aerodynamic properties from flow velocity, circulation, divergence, curl... In this case, the fields can be also developed numerically not just analytically simplifying their cons.

These strategies have been used extensively in robotic collective construction. Autonomous assembly with gradient fields is presented in [70] and [39] extends it to reconfiguration tasks where an initial structure is modified into a different geometry and compares it to state space methods seen before.

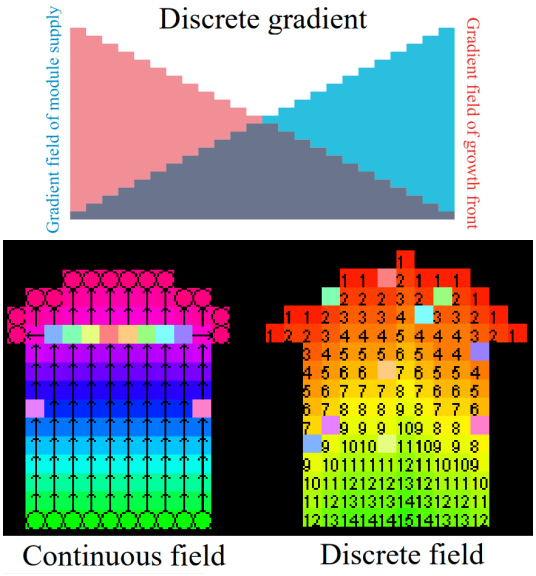


Figure 2.23: Radient fields in the context of the automated assembly of digital materials [71].

The fields can be generated by determining where lattices must be assembled, disassembled and where there is an intersection between the current and target geometries. A continuous gradient is developed from the disassembly to the assembly areas, which is discretized to give the nodes a numeric priority as shown in Figure 2.23.
Robots behave as charges following streamlines. When a boundary is reached, the necessary action (assembly/ disassembly) is carried out. Different stimuli like performing actions, interactions with other robots, or deadlocks may cause a charge flip leading to navigating the environment in the opposite direction. Often robots carrying voxels and robots getting new ones traverse the same streamlines in opposite directions. In [39] voxel handover was implemented to reduce the impact of these collisions. On the extreme, this results in a chain of robots exchanging lattices, like a bucket fire brigade. This behavior is easier to implement in gradient fields than state search where longer less dynamic plans are executed.

The performance of these methods compared to that of state space search planning systems in autonomous assembly is showcased in Fig. 2.24.

The leftmost image compares distributed and state search for a variety of test cases. These test cases present different geometries, numbers of voxels to translate (from 12 to 144), degrees of freedom, and number of robots (1 or 2). State search is shown to be more efficient in cases that utilize a low number of robots with differences ranging from 35% to 230%. Centralized algorithms have stronger guarantees of finding the optimal solution.
On the right one of the structures with 144 voxels to translate is presented for a varying number of robots. In the case of state-search increasing the robot density quickly saturates the structure, and more collisions between robots appear reducing the assembly efficiency. It is suggested that a saturation robot density exists. However, distributed algorithms show a better performance a high robot counts. This is mainly because voxel handover was implemented thus mitigating robot collisions. It must be noted that although more complicated it is technically possible to implement this voxel handover in state search methods as well.

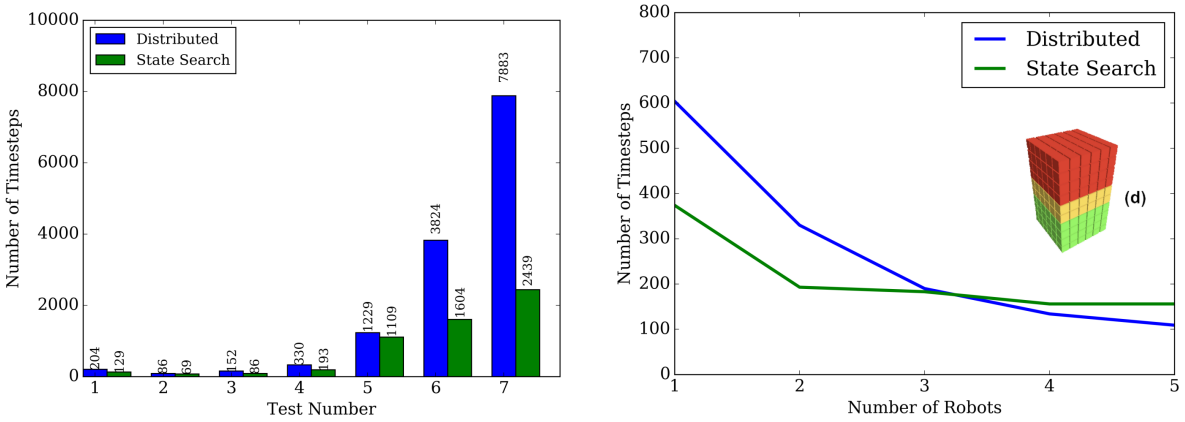


Figure 2.24: Performance comparison of potential field and state space search methods [39].

Lastly, the temporal organization of computation and information transfer must be analyzed. Distributed systems need to be given the instructions in some manner or another. In the case of gradient fields, a centralized authority must generate them and load it in the assemblers. This concentrates computation and data transfer to certain points in time, potentially resulting in bottlenecks. On the contrary, state search can benefit from Bellman´s principle of optimality which states that any part of an optimal plan is in itself an optimal sub-plan. Optimal sub-plans can be generated and executed before the full plan is computed.

Additionally, centralized systems give the user complete control, allowing modifications of the structure on the fly and overriding robot commands. Similarly, this more easily allows the future integration of more lattices, assembler types, workflows, and machines. Imagine a system where an optimal lattice design for a particular location and loading is generated, sent to a 3D printer, and later picked up by an assembler robot. Such a mix of workflows, tools, and machines is, at least for now, only feasible in highly centralized systems. In fact, centralized planners can even emulate distributed systems.

3

Research definition

The objective of the project is to develop a complete end-to-end autonomous digital material assembly system capable of bridging the gap between a 3D model and a built structure with a flexible, comprehensive and easy-to-use toolset that can be expanded and continued as a development platform. In this way, further research on autonomous assembly and its application to living materials can be performed at an increasingly high abstraction level. Specifically, it aims to introduce a new degree of flexibility by breaking from the strict grid digital material constraints seen in the literature, that limit the robot's mobility to a rectangular grid. In this manner allowing the introduction of different shapes, sizes, directions and curvatures to the building blocks. The resulting tool must be easy to use to allow for the start of a new path of research. This is summarized in the following research objective.

Research Objective

Develop a complete end-to-end autonomous digital material assembly system with flexibility, adaptability and expandability in mind

3.1 Research questions

To support this objective, the following research questions are defined and organized thematically into the three main topics of this thesis:

Material system

Can we develop digital materials that do not adhere to a grid definition to maximize potential structural performance?
What kind of structures could be generated by this kind of design freedom?

Assembler

How can we maximize torque available in the joints for a given actuator size?
How can a robot attach to the lattices with the least additional mass and technological headroom?
How can the structure of the robot be made lighter and more efficient?

Software platfrom

What strategies exist to find optimal navigation paths from one point in the structure to another?
Can we consider not only the robot´s position but also it´s pose or state for finding the most optimal solution?
How can movements be planned in such a non-grid space?
How can collisions be prevented or minimized?

Part II

Results

4

Digital material development

The digital material is one of the three items in the trinity of autonomous digital assembly with the other two being the robot and the control software.

The digital building blocks are composed of nodes that serve as an interface onto which other blocks and robotic assemblers can attach and a structure that holds the nodes together. In reality, this can describe a variety of objects, not just small lattice building blocks. For example, nodes can be added around a room, on a bridge, or on the exterior of a space station to allow robotic assemblers to operate around that environment and other building blocks to be connected. In the following sections, the design of the node and some lattice building blocks will be explored.

4.1 Node design

Nodes are the interaction interface of the digital material. They enable the connection between different building blocks and act as attachment points for assemblers. Nodes have been designed with the possibility of utilizing a threaded or magnetic connection. A node is shown in Figure 4.1 with an insert for the threaded connection highlighted in green as well as magnets in blue.

First, the magnets are pressed-fit in the node. Special features are present that allow some deformation of the plastic part. Additionally, cyanoacrylate glue can be added during this process. The magnet cover is then snapped-fit onto the node using four tabs on the side of the nodes. The magnet cover is a thin part that adds a high friction surface to enhance shear force transfer in magnetic connections, prevents damage to the brittle magnets, and secures them in place. Lastly, the m5 insert is heat-pressed in place using a soldering iron, this serves the dual purpose of placing the insert and securing the magnet cover in place by melting it together with the node itself.

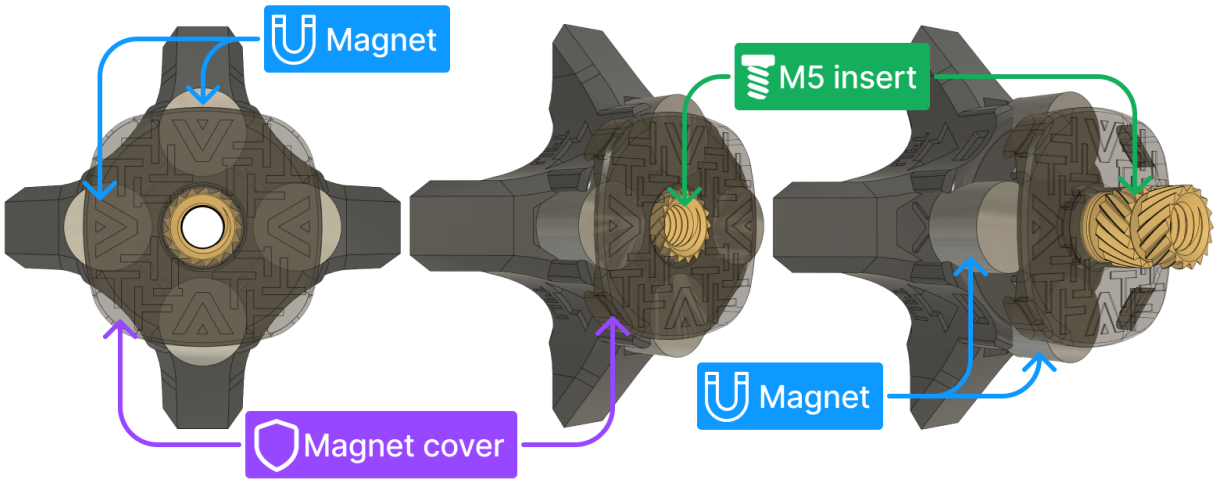


Figure 4.1: CAD design of the digital material's node showcasing the screw-in and magnetic attachment possibilities.

The threaded insert allows the screwing of an m5 bolt, providing a strong and structural connection. This is similar to the connection system showcased in [26] that utilizes an m2 bolt and embedded square nuts to demonstrate the mechanical performance and autonomous screwing capabilities. The nodes are secured with a partially threaded screw shown on the left-hand side of Figure 4.2. Fully threaded screws may be used but in order to tighten the connection, additional forces will exist on the inserts and the screw itself as they have to be pulled together.

Four cylindrical magnets situated around the threaded insert, open the door to self-aligning assembly with highly imprecise assemblers like flying drones and separating the roles of material placement and material fastening by allowing lattices to be held in place before fastening. By using four distinct magnets, this connection system limits rolling or spinning and thus constrains all movements. Different sizes of magnets may be utilized but in this node, large N52 8mm diameter by 6mm thickness magnets are utilized to achieve a strong connection capable of handling 5kgf tensile loads or 1kgf shear applied on the top node before disengaging as shown on the right-hand side ofFigure 4.2.

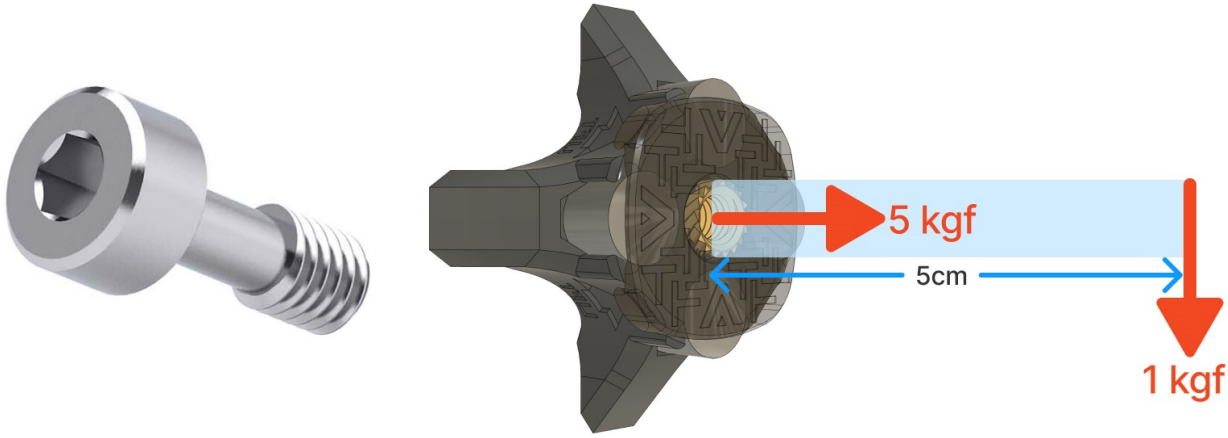


Figure 4.2: Partially threaded screw used for joining lattices (left) and magnetic connection strength in kilogram-force (right).

These nodes can then be integrated into different structures and building blocks. In this case, a set of lattices has been developed for the building process.

4.2 Lattice building blocks

As shown in the literature section, a plenitude of lattice types and morphologies exist. Most commonly, in digital materials octahedral lattices are used in [26] and [31] and cuboctahedra or cubot in [11] and [33].

Each lattice type presents different strengths and weaknesses. For instance, cubot have a large enclosed volume that allows the integration of additional functionalities or to modify the lattice structure to exhibit mechanical metamaterial behaviour. On the other hand, octahedral lattices are purely stretch-dominated as explained in Subsection 2.3.1, making them more structurally efficient [26]. For this reason, octahedral lattices are used in the proposed design as can be seen in Figure 4.3.

Small lattices below $1cm$ in size or large ones over $25cm$ are outside our comfortable manufacturing capabilities at the 3D printing lab. This range was further narrowed down to between 7.5 and $17.5cm$ after preliminary analysis of the robotic assembler. To select a final lattice size we look at the density. Octahedral lattices present decreasing density for increasing lattice sizes for constant strut cross-sections. To satisfy the target $50mg/cm^3$ density lattices of about $10cm$ size must be used. This selected size is ideal for robot design and lattice manufacturing even allowing batch manufacturing as will be presented in Section 4.3.

Figure 4.3: Proposed digital material octahedral lattice building block with a size of 10x10x10cm.

Different variations of this octahedral lattice building block were designed and produced with varying densities and curvatures. This enables optimizing the discrete material placement density and orientation for a target geometry and load case.

The density can be adjusted either by modifying the cross-section of the trusses in the lattice or their porosity. As will be further explained in Section 4.3 these lattices are 3D printed which allows to effortlessly make the digital material a hierarchical structure, further enhancing specific properties. First, the bulk material is composed of smaller individual octahedral lattices, and the lattice itself is made of a porous cellular structure, in this case, a gyroid structure as is very common in 3D printing. This hierarchy is showcased in Figure 4.4.

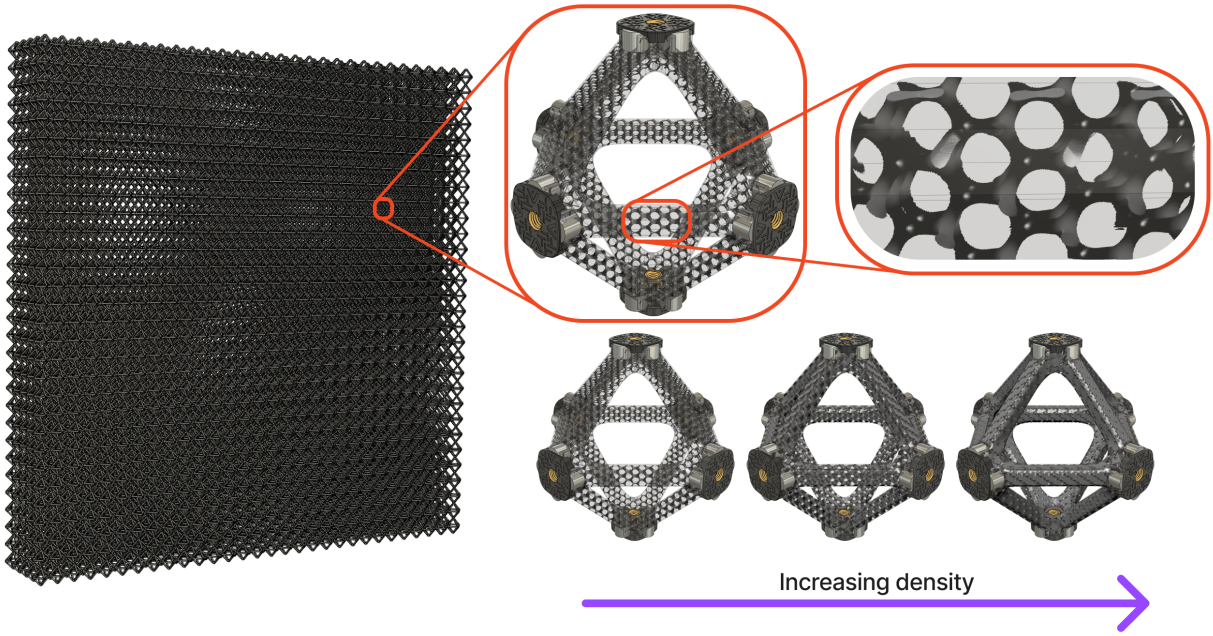


Figure 4.4: Digital material hierarchical features for different lattice densities.

Lattices with a slight curvature have also been developed to add the capability of tailoring lattice directions to the load path. This allows mounting parts of the structure at an angle or directly building arches and curves. The lattices presented in Figure 4.5 have a curvature of $\kappa = 1/0.6$, or in other words, a 20-degree rotation per unit cell, allowing to build a full circle with 18 lattices or a semicircular arch with 9 of them. Such an arch has a 6 lattice or 600mm diameter, allowing the integration of arches in structures utilizing the regular square lattices.

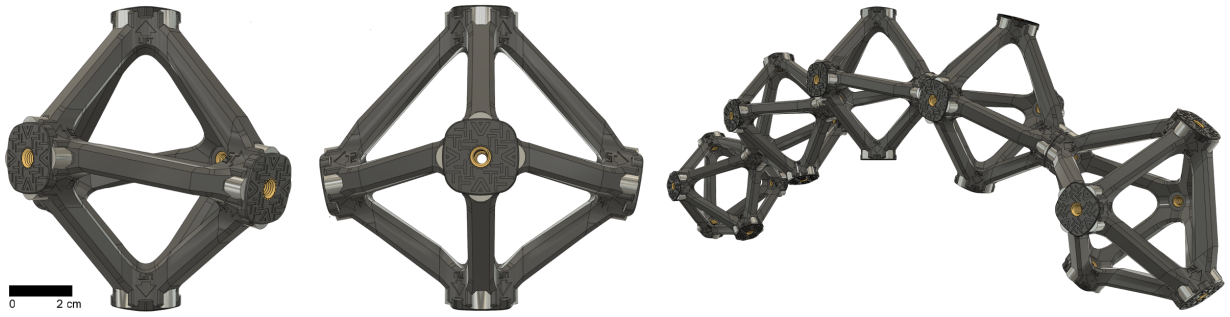


Figure 4.5: Curved digital material and discretely assembled arch structure.

Lastly, other structural elements may also be developed that utilize this same interface. For example, Figure 4.6 showcases a slender thin-walled truss. This truss can be combined with different curved lattices to enable the construction of simple truss structures out of more efficient building blocks. This variety goes to show the capabilities of such digital materials.

Figure 4.6: Design of a slender truss building block.

4.3 Manufacturing

These lattices have been 3D printed in the printing lab in the aircraft hall. The previously seen lattices from Jennet and Alex were broken down into 6 and 3 parts respectively, allowing for simple manufacturing methods like injection moulding or fast thin planar printing with some assembly. By contrast, these lattices are 3D printed in a single piece. This is mainly to allow the bed of a 3D printer to fit a larger number of lattices.

Three different print configurations were prepared and used throughout lattice manufacturing. Firstly, single lattices can be printed but this defeats the purpose of using single-shot printed lattices. Alternatively, the lattice can be quad-packed on the printer bed, filling it up completely. This adds minimal extra time of 15 minutes per lattice when compared to printing one by one and allows for a more hands-free mass production. Lastly, the interaction with the printer can be further reduced by also stacking lattices in height and utilizing the complete build volume. For this lattices are octa-packed, allowing for a 48hr print that can fully utilize the printer over a weekend. This adds significant extra support material, print time and especially post-processing or support removal time as the support is much thicker and harder to clean up. Quad-packing is the preferred choice.

The lattices were printed mainly on two Prusa Mk4 printers I assembled for this purpose (Diego Velazquez and Pablo Picasso) using the input shaper feature on firmware 5.0.0 and sliced in PrusaSlicer version 2.6.0.

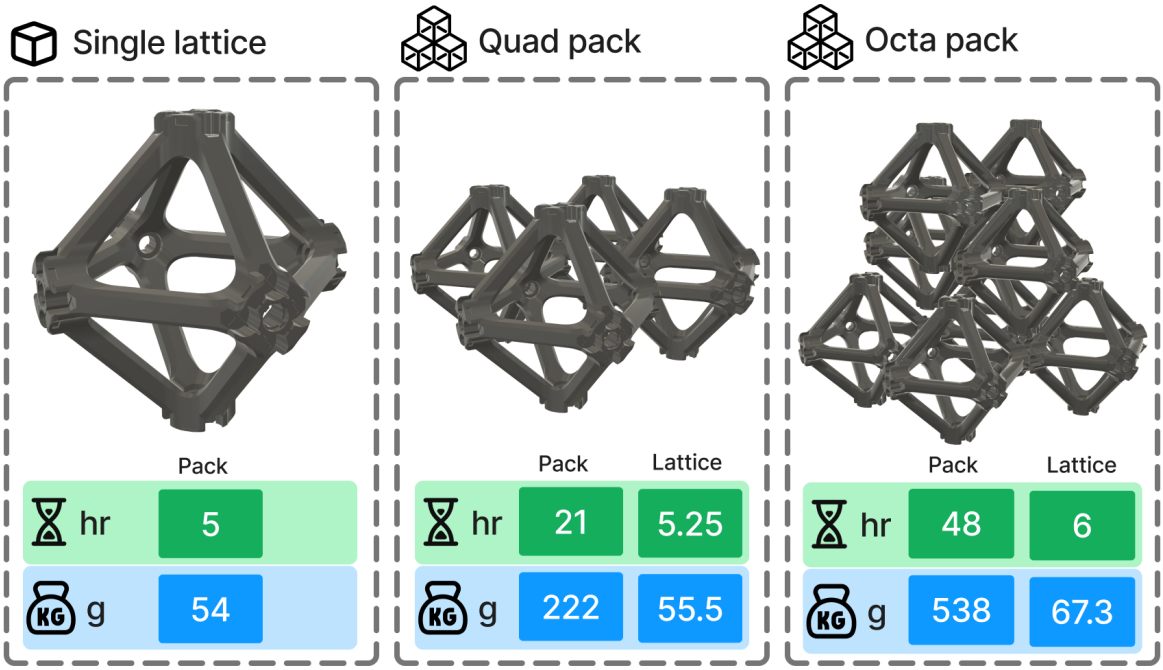


Figure 4.7: Lattice batch manufacturing options for the Prusa Mk4 with input shaping firmware

5

Robot design

The essence of autonomous assembly lies in the capabilities of a self-sufficient assembling robot. Mirroring the function of a ribosome in biological systems, this robot stands as a central figure in the assembly process.

The inherent nature of digital materials offers a unique advantage: they come with built-in precision, much like their biological counterparts. This intrinsic accuracy means that the precision of the end structure is determined by the material itself, rather than the capabilities of the assembler. This is akin to children constructing structures with Lego blocks; the assemblers are not required to operate with sub-millimetre precision, yet they are capable of constructing highly precise, large-scale structures. This reveals the potential for achieving high precision at large scales with relatively a simple assembly processes.

In exploring various robotic options for this task, inchworm locomotion robots emerge as the preferred choice, primarily for their unique structural advantages. As evidenced in Section 2.4 literature predominantly features this type of robot, and not without reason. This preference is rooted in their particular configuration. Unlike traditional robot arms with a fixed base and a movable gripper where the actuator locations are biased towards the base, inchworm robots feature joints and actuators symmetrically arranged around a central point as showcased in Figure 5.1. This unique configuration enhances their mobility and flexibility when switching bases. Moreover, this is an efficient distribution of joints for maximizing the volumetric reach of the assemblers, especially when handling light payloads. The design of inchworm locomotion robots stands out for its suitability in large-scale, high-precision assembly tasks.

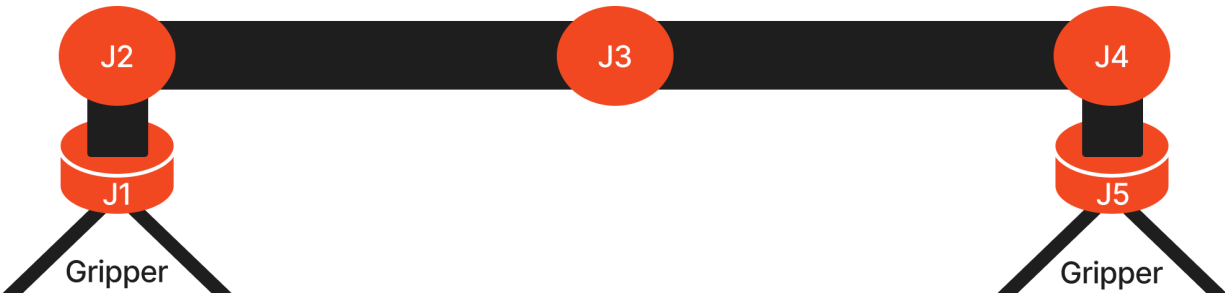


Figure 5.1: Joint configuration of an inchworm locomotion lattice assembler.

Figure 5.2 provides a comprehensive view of this inchworm assembler's hardware architecture. Four core components are identified: actuation, gripper, electronics and robot design englobing the robot's structural design and lattice storage. The actuation system encompasses the kinematics and motors, the structure refers to the arms and general structural design of the robot, the gripper serves as the tool for interacting with lattice structures and the electronics facilitate communication and power distribution.

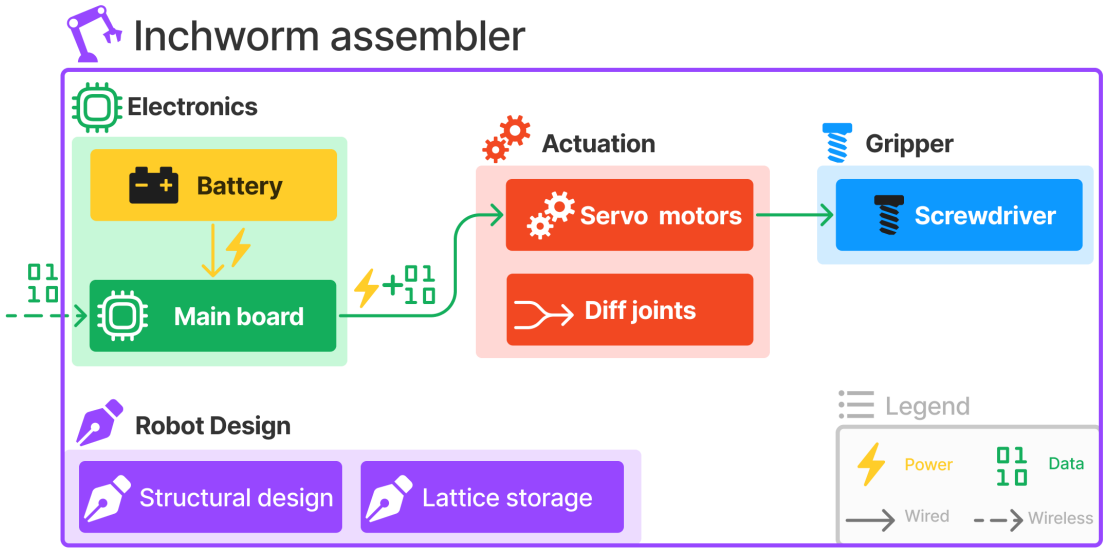


Figure 5.2: High-level architecture of the inchworm assembler developed.

The following sections will present and describe the design rationale for each of the different elements, but first one of the robots presented in literature, that I had physical access to is investigated to develop some design objectives.

5.1 Robot design objectives

Our journey begins with a thorough analysis of the existing robot shown in Figure 5.3, originally developed by Alex Luijten at ETH [26]. This analysis serves as the foundation for our design improvements. The goal is to discover the robot's strengths and weaknesses pinpointing areas for potential enhancement and laying the groundwork for the redesign strategy.

Figure 5.3: Baseline robot for the investigation of the design objectives. Design from [26].

The testing included various methods, such as using servo testers or PWM generators to actuate the servos and test different motions. Throughout the testing process, several key limitations regarding actuation, the gripper or tool used to clamp to the base lattice, the robot's structure, and the electronics were uncovered.
The initial tests revealed that the motors struggled to counteract gravity's effect in certain poses. As the arms are extended, the gravitational pull creates a significant moment loading on the joints, demanding the motor to produce substantial torque to move the joint effectively. This limitation greatly reduced the robot's range of motion. Furthermore, it was observed that the current servos could not control acceleration or velocity profiles. Instead, they abruptly apply maximum torque when accelerating resulting in unstable jerky movements, impacting precision, stability and shaking the complete structure.

The lattice attachment mechanism, known as the gripper or foot secures to lattices by rotating multiple small servos that grab onto the lattice's trusses. This is a heavy and bulky mechanism situated at the very end of each arm resulting in additional torque requirements. Testing this mechanism, revealed it also lacked flexibility or adaptability to different lattice geometries and exhibited excessive backlash.

Throughout the testing, it became apparent that structural integrity was another major concern. The

robot 3D-printed arms showed substantial permanent deformation while the first and last axles failed just from the torque applied by gravity even before the motors stalled. Thus it is not only important to increase the torque available but also to strengthen the structure.

Additionally, the robot's ability to transport and insert lattices autonomously was evaluated. The current design requires an external storage mechanism or backpack and assistance from other robotic units, pointing to a significant gap in the robot's autonomous capabilities.

Lastly, an examination of the robot's connectivity and mobility was conducted. The robot operation is restricted to a tethered setup, requiring an umbilical cable connection to a computer and power supply. The tethered setup was found to limit the robot's operational range while also posing a risk of entanglement with the surrounding lattice structure, severely limiting its operation range and reliability.

Other aspects of the baseline robot design were found to work adequately. For instance, the joint configuration on this inchworm locomotion assembler is very suitable. Having the joints and actuators be distributed symmetrically around a central point is ideal for a robot where the base can be swapped. Similarly, the self-centering lattice conical interfaces worked well.

With these insights, I set out to formulate targeted design objectives. Each objective is tailored to address the specific challenges uncovered through the testing making the development process more informed and efficient. and are presented on the following list.

Actuation

Enhance the output torque of the joint and size it to work on the full operational envelope
Implement advanced motion control to reduce jerk and sudden accelerations, ensuring stable movements

Structure

No part shall fail or plastically deform under gravitational loading in any pose while carrying a lattice payload
Minimize weight and arm length to reduce moment loading

Gripper

Develop a light and adaptable gripper/ foot mechanism for robust interaction with various lattice structures.

Storage

Develop a lattice storage solution the robot can use autonomously without the help of additional assemblers

Electronics

Develop a wireless control system with integrated battery power, enhancing operational range and mobility.

The design objectives from the list will be revisited in the following sections.

5.2 Actuation

Improving the actuation platform is crucial for expanding the assembler's operational envelope, thereby enabling a broader spectrum of movements. The following design objective was formulated to inform the design process.

Actuation

Enhance the output torque of the joint and size it to work on the full operational envelope
Implement advanced motion control to reduce jerk and sudden accelerations, ensuring stable movements

Various strategies can be utilized to achieve this objective. Firstly, we can reconfigure the joints by increasing the gearing ratio, designing them to harness torque from multiple actuators or simply replacing current motors with higher-performing models that offer more torque and include velocity and acceleration control. Alternatively, we can opt for lighter designs to relax the torque requirements.

First, let's analyze the joint configuration of both the base and tool 2DOF platforms. Traditional inchworm locomotion robots typically employ a fully serial actuator configuration to achieve multi-DOF (Degrees of Freedom). This approach often restricts motor placement options and results in unequal torque requirements across different actuators. This leads to a design challenge: whether to use two identical motors, risking underutilization of one, or opt for two different motors, complicating the design process.

5.2.1 Parallel actuated differential joint

This thesis introduces an innovative approach to inchworm locomotion robots, utilizing a parallel actuated differential joint in the base and the tool. This allows for more flexible motor placement, a lightweight, compact and efficient mechanical design and achieves a more balanced distribution of torque requirements by exploiting the differential joint's partial load-sharing capabilities. This is particularly beneficial in managing gravitational loads [72] [73].

The design of the joint is a rotational 2DOF parallel actuated differential joint. This means that the movement of two different actuators is combined into the output of a 2DOF joint. The actuators are at the same hierarchical level. Unlike in a serial configuration where there is a parent-child relation for the actuator position, here they are both attached rigidly to one another. In the simple scenario with the same gearing ratio everywhere as exemplified in Figure 5.4, when both motors rotate in the same direction ϕ_1 moves and when the motors rotate in opposite directions ϕ_2 is actuated. I thus refer to these joints as common and differential modes respectively. The same principle applies if the gearing ratio is modified but with a different motor to output joint mapping that is not 1:1.

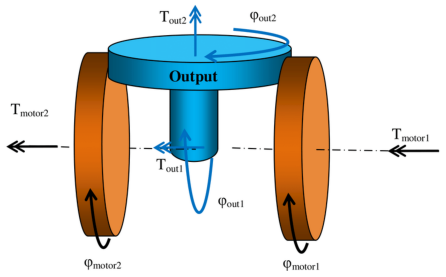


Figure 5.4: Diagram showcasing the working principle of a differential joint. Image courtesy of [74].

With this overview of the theory behind differential joints, a concrete design must be realised. Differential joints typically employ bevel gears, however, finding the correct size proved challenging and 3D printing them while maintaining a low backlash and long lifetime was a risky and complex endeavour. Instead, a cable-driven differential was developed. In this design, pulleys are used to guide a thin cable from the motor to the differential joint. The motor pulls on this cable to actuate the joint in either direction. Cable transmission systems can bring several advantages. For instance, when compared with gears they offer a large power/mass ratio, especially at higher powers where bulky metal gears must be used. Additionally, cables enable easy remote placement of actuators and simple tailorability of gearing ratio.

Figure 5.5 illustrates the mechanical design of the differential joint. Firstly, the innermost axle shown in white and being the most critical component, was designed with a large diameter as it was found to fail in the baseline design. This diameter increase serves a dual purpose: enhancing the axle's mechanical strength and creating sufficient space to accommodate a compact gripper tool within the area marked in light blue. Nevertheless, the earlier prototypes of this new shaft were still found to fail under the most extreme loading conditions. These failures were noted in early prototypes made with Polyterra bio-PLA, a material known for its matte finish, sandability, and environmentally friendly properties. This failure takes the form of a layer delamination or separation, a weakness inherent in the printing process. Despite the improved strength observed in the latter prototypes printed with 123-3D PLA filament that did not show failure, the decision was made to switch materials for the final inner axle. PETG is known for its interlayer adhesion, however, none was available in the lab so polycarbonate was used instead. Polycarbonate is a higher-performance 3D printing material that is printed at high temperatures and is significantly stronger and tougher than PLA [75] [76].

The inner shaft is joined via the two inner axis bearings to the differential joint's main casing shown in semi-transparent blue. These 20x27mm 6704 bearings are the largest used in the robot and are held in place by two grooves in the differential joint's casing. The differential joint enclosure is split into two sections to allow for easy assembly. These halves are secured together when the arm and pulley bearings are installed. The assembly process resembles a puzzle, with each component effectively rigidly locking or constraining the others in place.

The robot arms, which will be detailed in subsequent sections, are linked to the differential using two 6703 bearings. In addition, two spacers, shown in purple, correctly position the side pulleys, which are also mounted using the same type of bearings. A green end stop prevents this assembly from moving sideways. Furthermore, a smaller 6701 bearing is used for the cable tensioner plate, a part of the arm, that is also constrained by the last white disc. The list of bearings utilized can be consulted in the robot's BOM or bill of materials in Appendix B. All bearings are fully constrained within their shaft, ensuring they do not shift sideways along it.

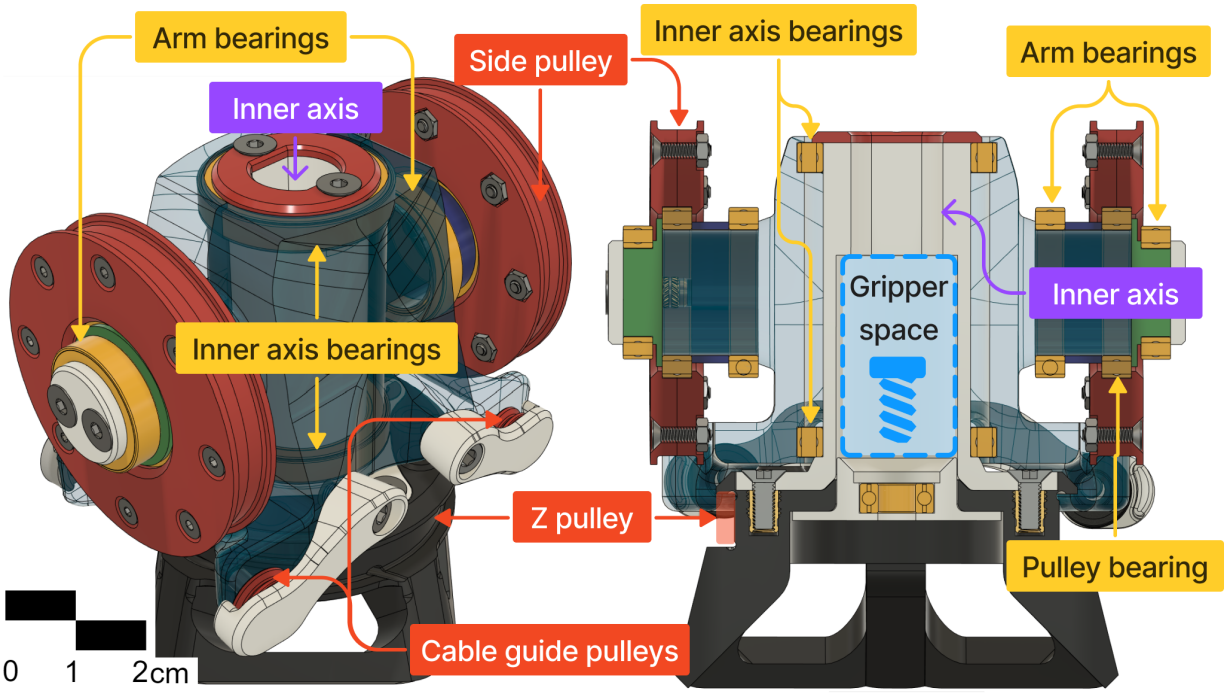


Figure 5.5: Mechanical design of the 2DOF differential joint showcasing its various mechanical components.

Mechanical advantage

In a similar way to how we can modify the gears in a traditional gear-driven differential joint to tailor the gearing ratio, the diameter of the different pulleys and more importantly, the placement of the cable guides can be modified to tailor each joint's gearing ratio.

To determine the torque required and thus the gearing ratio, a first-order estimation of the mass and moment of inertia of the robot was performed. The scenario where the robot is fully extended and positioned horizontally against gravity was chosen as this position demands the highest torque. Following some quick analysis, the gearing ratio was selected to be $1.5:1$ meaning that for every 1.5 rotations of the motor, the differential joint will rotate once. This increases the torque available at the joint from the $29KgF-cm$ of the motor to $36KgF-cm$. This is a significant 50% torque increase over the $25KgF-cm$ of the previous robot. Unfortunately, some last-minute design changes had to be made to adapt to changes in hardware availability resulting in a drop in the gearing ratio to about $1.4:1$. The robot still functions adequately with this ratio although it's not ideal. This is an improvement point presented in the future work chapter in Chapter 9.

Cable path

The 2DOF differential joint combines the inputs from two separate actuators. Each motor's torque is transmitted to the joint via a cable. This cable's journey begins at the motor pulley, moves through a cable tensioning mechanism, then traverses the side pulley, depicted in orange in Figure 5.4, and finally passes through the guides to the Z pulley, shown in blue. These guides ensure that the cable remains straight or tangential to the pulleys and help tailor the gearing ratio.

The cable must be held to the end pulleys to transfer load and not slip, which can be achieved either by friction or fixation. Friction requires wrapping the cable around the pulley multiple times, while fixation involves securing the cable at specific points. Fixation was chosen for its reliability and predictability. In this design, each cable is secured at both the differential z pulley and motor pulleys with two inserts and set screws, effectively creating two distinct, shorter cables. A transparent motor pulley showing the cable fixation is presented in Figure 5.6

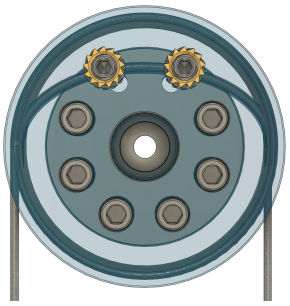


Figure 5.6: Transparent motor pulley showing cable fixation.

Selecting the right cable involves considering its material, diameter, and structure. Steel was the preferred material due to its plastic deformation properties, making it easier to clamp using fixation compared to other materials like UHMWPE, and also thanks to its widespread availability. Steel cables come in various diameters and types. The optimal diameter depends on the expected load, which is calculated by superimposing the pretension load in the cable (present to minimize backlash) with the load generated by the motor torque (determined by dividing the motor torque by the radius of the motor pulley). A balance must be struck between pulley size, cable thickness, and the overall size and weight of the system while considering the desired gear ratio between the motor and the joint of $1.5:1$.

With an actuator stall torque of $29Kgf-cm$, and a 50N preload, a 3cm diameter motor pulley was found to be a good balance. The cable load will reach up to approximately $(29 \cdot 9.81)/1.5 + 50 \approx 250N$. A safety margin of 2.5 was used to arrive at a requirement of 600N minimum breaking load. A 1mm thick 7x7 cable was found to satisfy this requirement with a 640N breaking load [77] and is commonly available. Cables are made from thin wires twisted together, and different structures exist. Cables can be made of different numbers of bigger or smaller wires; this is what the 7x7 denomination refers to. The first number indicates the number of strands, and the second is the number of wires twisted to form each strand. Meaning that the 7x7 cable selected is made of 7 strands, each composed of 7 wires, as can be seen in Figure 5.7. In the 1mm diameter size, 1x7, 1x19 and 7x7 are most commonly found. The larger the number of wires, the more flexible the cable will be, which is a priority in this design, given the small radius of some of the pulleys. However, configurations with thicker wires can have a higher maximum load.

Figure 5.7: 7x7 cable structure

The cable assembly and how it is arranged is illustrated next in Figure 5.8. The leftmost image shows the complete differential joint assembly. In the middle image a bottom view is displayed where the Z pulley has been removed to better showcase the cables around it. The rightmost image isolates the cables and fixation method. These images show how the guide pulleys allow the cable to exit both main pulleys tangentially while preventing collisions between cables. Take the blue and yellow cables in the first and last images. They both start wrapping around the Z pulley at roughly the same location. To prevent them from colliding and trying to take the same space, both cable guide pulleys are offset on the vertical axis. The same concept applies to both cable guides and side pulleys. As can be seen in the middle image, one of the guides is offset to the front while the other one is further back on the side pulley axis.

Additionally, the middle and right images show how the cables embedded themselves inside the z pulley and can be clamped using set screws.

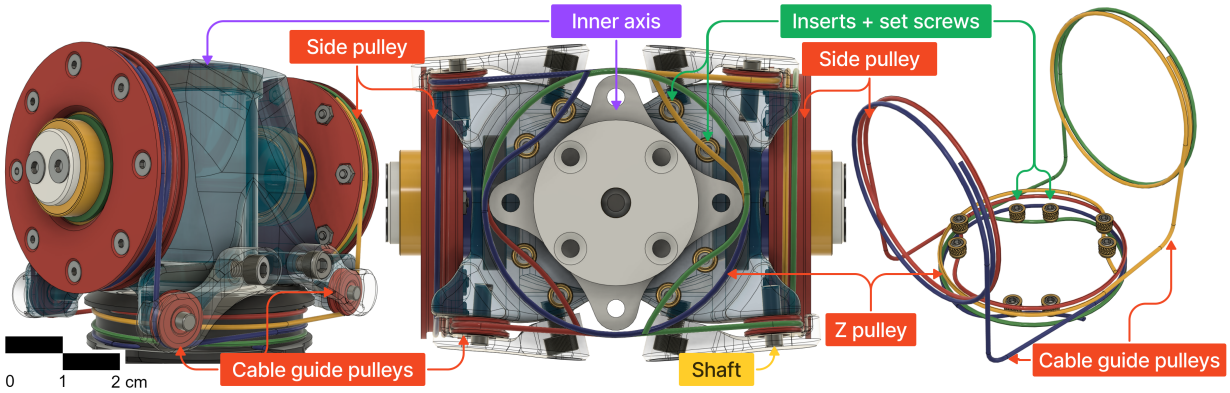


Figure 5.8: CAD design of the differential joint showing the path of the steel cables.

Cable tensioning mechanism

To ensure effective cable operation, maintaining an appropriate level of tension is essential. Without adequate tension, excessive slack in the cable can hinder its immediate transmission of force to the joint when the motor applies tension. Instead, it tends to straighten out in the load direction, creating a form of backlash requiring the motor to first rebuild tension before effectively exerting force on the joint.

To address this issue, a tensioning system was devised to maintain a consistent cable tension without needing motor movement. This tensioning system is illustrated in Figure 5.9. It comprises two small U-groove roller bearings (u604ZZ) acting as pulleys that can be adjusted closer to or farther from the main pulley using a tensioner screw, depicted in green. These bearings move diagonally, as depicted in the white arrows in the right image, forcing the cable to elongate and thereby increasing its tension. A parametric CAD sketch was employed to design this mechanism, enabling real-time calculation of cable elongation as the design parameters were adjusted. This facilitated the discovery of a design that successfully met all geometrical constraints. In its current configuration, transitioning from the least to the most extended position elongates the cable by 19mm on each side, providing an ample tensioning range, assuming the cable is not excessively loose to begin with.

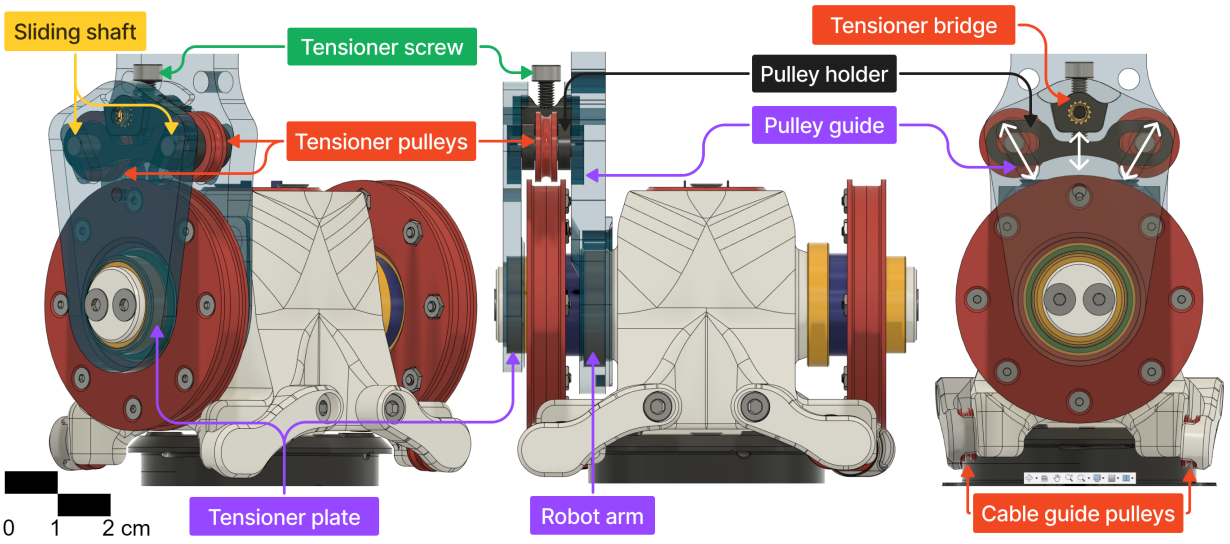


Figure 5.9: CAD design of the differential joint showing the cable tensioning mechanism.

This section has presented the motivation and design of the parallel differential joint utilized on both sides of the robot where a 2DOF joint was needed. Some of the advantages of the design like flexible motor placement have not been showcased yet but will be seen in the arms design in Section 5.3.

5.2.2 Servo motors

Another way to increase the torque available on the joints is not to relax the motor requirements by playing with the gearing ratios and torque sharing but rather to size the motors for higher torque output. Additionally and recalling the second design objective, advanced motion control shall be implemented to reduce jerk and sudden accelerations.

After comparing a wide range of actuator types, from BLDC motors to servo motors, a decision was taken early on to employ smart servos. The base swapping, low cost and low mass characteristics of the robot make it difficult to use a quasi-direct drive and instead benefit alternatives with large gearing ratios. This, combined with the previously stated objectives, means smart servos are most suitable. Smart servo motors are characterised by incorporating a low-power microcontroller, such as a Cortex-M0 core in the case of the selected servo, that enables advanced communication, control mechanisms and additional features. For example, some of these smart servos run a user-controllable PID control loop, allow to specify movement acceleration and speeds, controlling the torque applied or even simulating a certain compliance or impedance. Additionally, they can generally be daisy-chained, reducing cabling, as they employ more robust digital communication protocols with package IDs instead of a PWM pulse signal. However, the market of inexpensive ”hobby-grade” smart servos is small and does not have the variety of the RC servo market.

The products of the main smart servo manufacturers, Dynamixel/Robotis, Lynxmotion, and Herculex, were compared with the baseline motor servos. A summary of the compared specifications is provided in Table 5.1. Dyanamyxel has a wide product lineup going up to very high-performance servos that would be highly suited for this application, like the 82gr XM430-W350; however, since we use multiple servos and the goal of the system is to be expandable to multiple assemblers, the price was also a strong consideration.

Lynxmotion servos lack some of the smarter features like a fully controllable PID loop where instead you can only vary the stiffness across 9 levels, but it does allow to modify the maximum angular speed and acceleration, fixing the jerk issue so it was deemed sufficient. Additionally, Lynxmotion presents a complete ecosystem with controller boards for different devices, such as DC motors or smaller servos. As can be seen, the LSS-HT1 offers similar performance to the currently used servo and it is the best performance for smart servos in the price range, thus the LSS-HT1 is used.
All these smart servos are HV or high voltage, meaning they don’t work in the traditional 3.3-6V range of servos like the famous SG90 but rather employ 12V. This means it’s possible and advisable to power most of these servos with a 3S LIPO battery, which has a voltage range of 12.6 to 9V with roughly an 11.1V nominal voltage. More details on the battery will be shown in Section 5.6.6.

Figure 5.10: LSS-HT1 servo from Lynxmotion [78].

Table 5.1: Servo specifications comparison table. The selected servo is the LSS-HT1

Servo Model	Smart features	Gearing ratio	Voltage	Peak current at spec V[A]	No load speed [rpm]	Stall torque [Nm]	Mass [g]	Price [eur]
FS-25W	No	-	7.4	-	125	2.45	65	30
LSS-HT1 [78]	Basic	320:1	12	1.75	60	2.85	80g	120
LSS-ST1 [78]	Basic	253:1	12	0.65	60	1.37	58g	80
XC430-W240 [79]	Advanced	245:1	12	1.3	70	1.9	65g	140
XM430-W350 [80]	Advanced	353:1	12	1.8	46	4.1	82g	320
DRS-0201 [81]	Advanced	266:1	7.4	-	68	2.35	60g	177

5.3 Arms and structural design

The robot's structure is another key aspect. It is a major contributor to the mass budget and has the key function of precisely positioning all components and keeping them together. Its derived design objectives are:

Structure

No part shall fail or plastically deform under gravitational loading in any pose while carrying a lattice payload
Minimize weight and arm length to reduce moment loading

First of all, the dimensions of the arms must be established. The movement that constrains the maximum arm length is the convex turn manoeuvre which requires the arm length to be over 2.5x the cube side length of the lattice [11], that is, at least $176mm$. However, due to constraints with the construction and assembly of the robot and to allow the third joint to achieve adequate actuation angles, a length of 201.5mm between the axes was established.

A new structural concept was developed that maximises the utilisation of already existing elements to add more structural integrity by making them a part of the load-carrying structure. In this manner, the servo motors are used as an integral part of the arms. The large flexibility in motor placement the differential joint offers was key to making this design successful. This design is shown in Figure 5.11. The middle image presents a simplified diagram of the structural concept, while the top and bottom images showcase the actual CAD design.

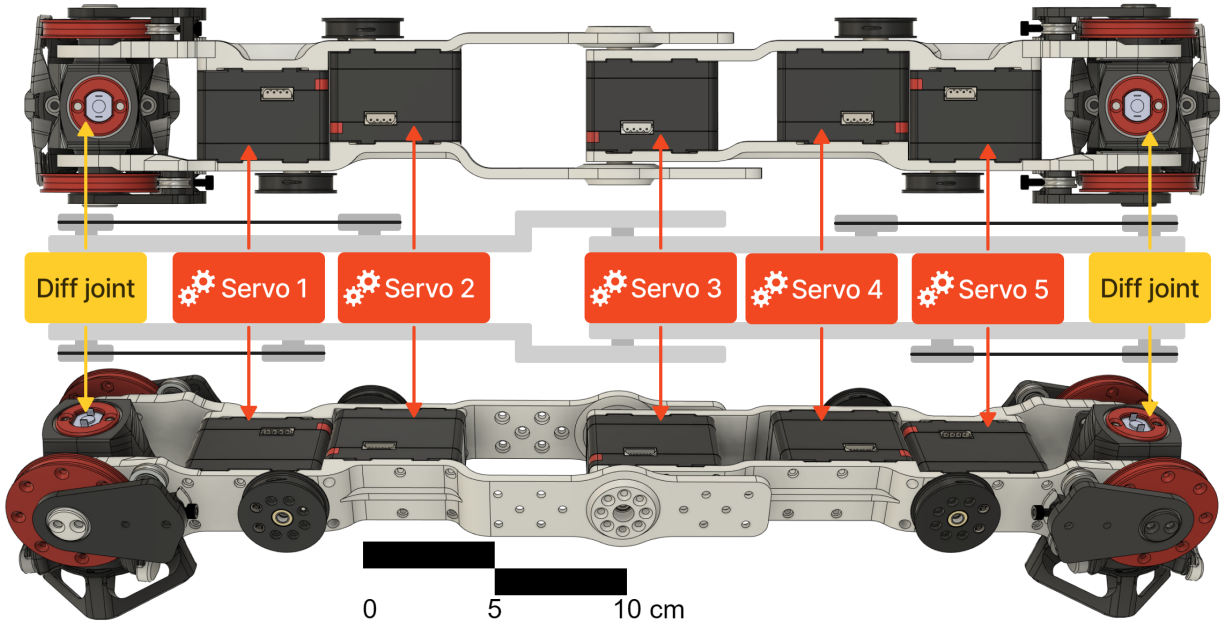


Figure 5.11: Robot arm structural concept that integrates the servos as part of the load-carrying structure.

As explained in the investigation of the baseline design in Section 5.1, the arms of the baseline design showed severe plastic deformation. This deformation was localised near the load introduction area on the base. There were stress concentrations that overly affected that region. Local thickening of the area near the base and the third servo where load is introduced, combined with the large 6703 bearings with a nominal diameter of 20mm, eliminates these highly localised stresses.

Additionally, the differential joints present a more robust structural design with a revamped larger diameter, polycarbonate inner axis and overall mechanical design as explained in Subsection 5.2.1

5.4 Gripper

The inchworm robot assembler, unlike traditional robot arms, needs to switch its base during operation. This involves clamping the robot's tool onto a lattice element and releasing the previous base. The gripper, located at the robot arm's end, is the tool that enables this docking and grabbing capability.

Previous sections highlighted that when the robot arm extends, it faces significant moment loading, which the motors must counteract. This load is transferred through the gripper to the base, making the gripping system a critical component of the design.

Gripper

Develop a light and adaptable gripper/ foot mechanism for robust interaction with various lattice structures.

Designing an effective gripper entails engineering a capable yet lightweight system that minimises the use of heavy components like solenoids, motors, or servos, instead leveraging the unique and distinctive properties of lattices to ensure a robust connection. There are various approaches to latching onto the lattices. For instance, Alex's design incorporates two servos to grip the horizontal struts, whereas the Bill-E robot employs a single servo to position an end stop behind the lattice node.

Breaking with this constant of utilising servos to grab the struts or place end stops, I innovated an alternative attachment mechanism that exploits the distinctive features of the lattices to enhance the connection. Realise that lattices are comprised of several interconnected nodes that can be bolted together to other lattices. The new robot attachment mechanism shares this bolted attachment vector, unifying the lattice-to-lattice and lattice-to-robot connections and providing a strong screwed-in connection everywhere. This eliminates the previously observed slack or backlash between the latches and the lattice while minimising the design constraints imposed on the lattices for robot-to-lattice connection, accommodating thicker, thinner or non-rectangular lattices.

The strength of this bolted-on connection was evaluated early in the design process, as it plays a critical role in the overall design. To test it, a 3D-printed beam and a basic luggage weighing device were used to apply a controlled load to the prototype connection on an initial lattice design. The test, conducted with an M5 heat set insert and screw, proved successful. The M5 size was chosen for its balance between size, weight and for facilitating easier alignment of the screw with the hole than at smaller sizes. However, further optimisation is needed, especially considering the unknown reliability and dependability of manually placed heat-set inserts.

5.4.1 Screwdriver actuator design

For the autonomous screw-in of the robot to the lattices, an electric screwdriver device was developed. This screw actuator must be lightweight, provide adequate torque and allow to reliably screw into the lattice with relaxed precision.

A commonly available geared N20 brushed DC motor was used. More specifically, a $298 : 1$ geared 12gan20-298 motor capable of tightening with a torque of up to $1.5Nm$. This motor is coupled via a two-press-fit PLA adapter to a hex head m5x16 screw. Additionally, an insert and set screw clamping the flat side of the motor shaft were added for increased lifetime and reliability of the coupler connection. Figure 5.12 showcases the assembled actuator, an exploited view and lastly, how it is integrated into the differential joint.

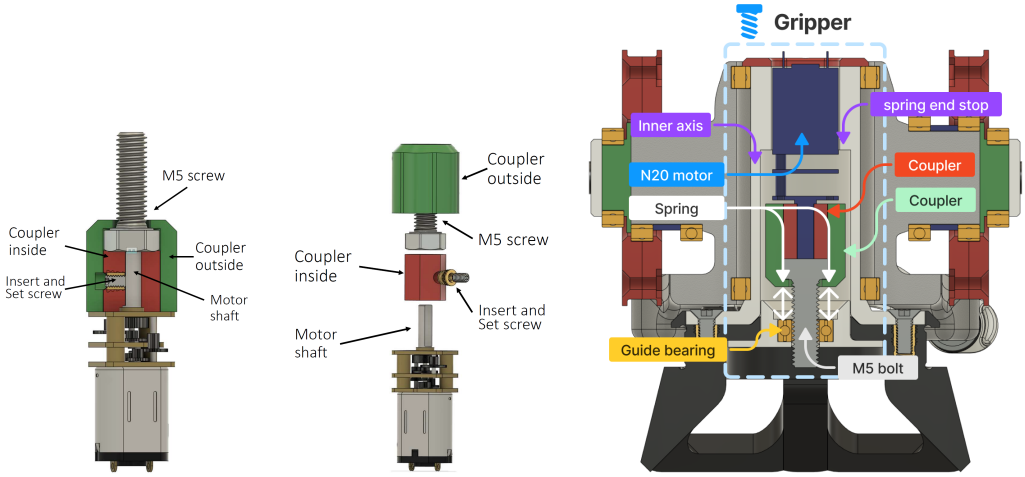


Figure 5.12: CAD design of the gripper screwdriver mechanism. The two left images show the motor-screw coupling and the right most image shows its integration in the differential joint.

The screwdriver is positioned inside the innermost axis of the differential joint, which provides a mechanically strong connection on the top via the N20 motor walls and results in a minimal footprint. An additional guide bearing was placed, allowing load transfer at the bottom of the assembly between the steel bolt and bearing cage. Thus mechanical load transfer can occur both at the top and the bottom of the shaft, enhancing moment loading transfer capabilities.

The screwdriver assembly can move on the vertical axis as marked by the white arrows in the rightmost image (Figure 5.12), and is spring-loaded to be pushed upwards naturally. This is a key feature to exploit the self-alignment capabilities of the conical interface of the Z pulley, shown in black. When the screw is disengaged, the spring pushes it upwards, only leaving a few millimetres sticking out. As the differential joint assembly approaches a lattice, the conical black part, self aligns with the lattice. Since the screw is retracted, this self-alignment can work unimpeded without obstacles, bringing the lattice node and the screw tip to close proximity. When the screwdriver engages and starts rotating, the spring compresses and the screwdriver assembly is pushed out, properly engaging an adequate length of the screw thread in the insert, ensuring a strong mechanical connection. A conical spring is used as it allows a great compression range. The vertical movement is limited by the contact between the conical walls of the outer coupler casing shown in green in Figure 5.12 and the conical interface of the guide bearing holder in white. This ensures a small cavity remains for the spring to compress into without being damaged.

5.5 Lattice storage

In order to walk, the robot needs to have both grippers available. Therefore, to allow the robot to walk and carry a lattice simultaneously, a storage point or location is introduced. This feature allows the temporary storage of a lattice while walking, enabling the robot to later retrieve it and insert it into the structure. The design objective is to enable autonomous independent utilisation of this storage point with a single robot since the baseline design required a second robot to access the storage point.

Storage

Develop a lattice storage solution the robot can use autonomously without the help of additional assemblers

Different concepts for storage locations have been developed in the literature like the one in the basline design, however a new kind is envisioned here.

The storage location is designed to magnetically attach one lattice node on the differential joint. As shown in Figure 5.13, there are three 10x3 and one 10x1.5 magnets positioned in a square pattern. The magnets employed are N52 and although thin, the diameter was enlarged to achieve a strong connection. A low profile system was required to maintain accessibility to the screws that secure the joint cables in

place and attach the inner axis to the z pulley. The holes where the magnets are to be inserted present crush ribs to facilitate a press-fit connection with the magnets. Additionally, a thin lid covers the magnets preventing chipping and ensuring they stay in place. The lid is held by some small tabs that are press fit, and an m3 insert is used to melt the lid and differential joint enclosure together, creating a strong connection as is done in the lattices.

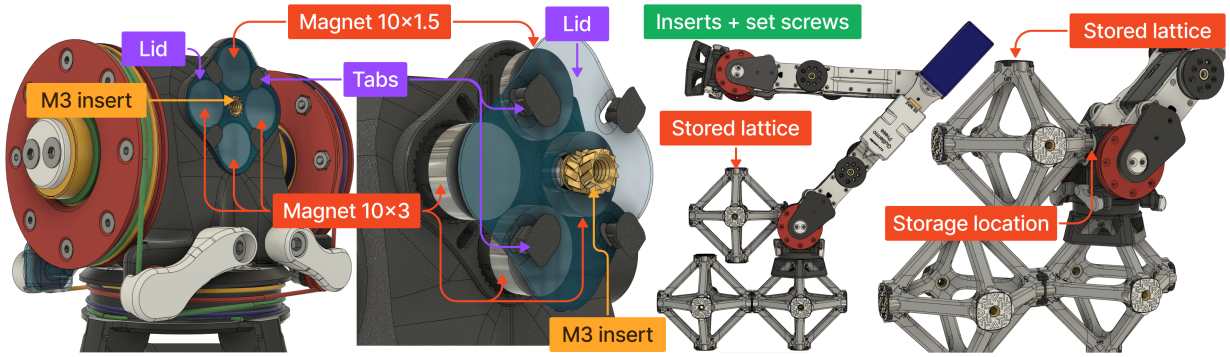


Figure 5.13: Lattice storage location concept. The two left images show the mechanical design and the two right images show a lattice in its stored configuration.

The attachment point is hierarchically located after the first joint of the differential, meaning the angle of the first joint of the differential affects its position and rotation, while the second joint's angle does not.

The storage location part is a variant of the differential joint enclosure half part, and up to four can be used per robot, although it is not recommended. As can be seen in the second two images, when lattices are stored, the joint is prevented from travelling its full range; in fact, it can not tilt more than 40 degrees in that direction. For this reason, and although the robot can have up to four storage locations, it is not advised to concurrently use more than one storage point simultaneously. To save mass budget, robots are built with only a single storage point per differential joint.

To store a lattice or remove it from storage, the opposite gripper is positioned on the top node of the lattice, that is, the node where the red "Stored lattice" tags point to in Figure 5.13. The lattice is clamped to using the gripper and then the robot pushes to physically separate the lattice from the storage point.

5.6 Electronics

The electronics architecture is an enabler for the rest of the system. As was evidenced until this point, and illustrated in Figure 5.2 from the introduction to this chapter, there are a variety of smart elements communicating and consuming power such as the smart servos, the gripper screwdrivers and a mainboard to control them all. Lastly, there is a battery to power the system.
The role of the electronics is to process and distribute the information, commands and power across the various actuators while posing the least inconvenience and friction to the user and the rest of the platform.

Electronics

Develop a wireless control system with integrated battery power, enhancing operational range and mobility.

Figure 5.14 expands on the diagram presented at the beginning of the robot chapter and showcases the electronics architecture in more depth. We can distinguish a Mainboard with different modules that communicate over the LSS bus with the actuators and the screwdrivers
This is all powered by a lithium polymer battery.

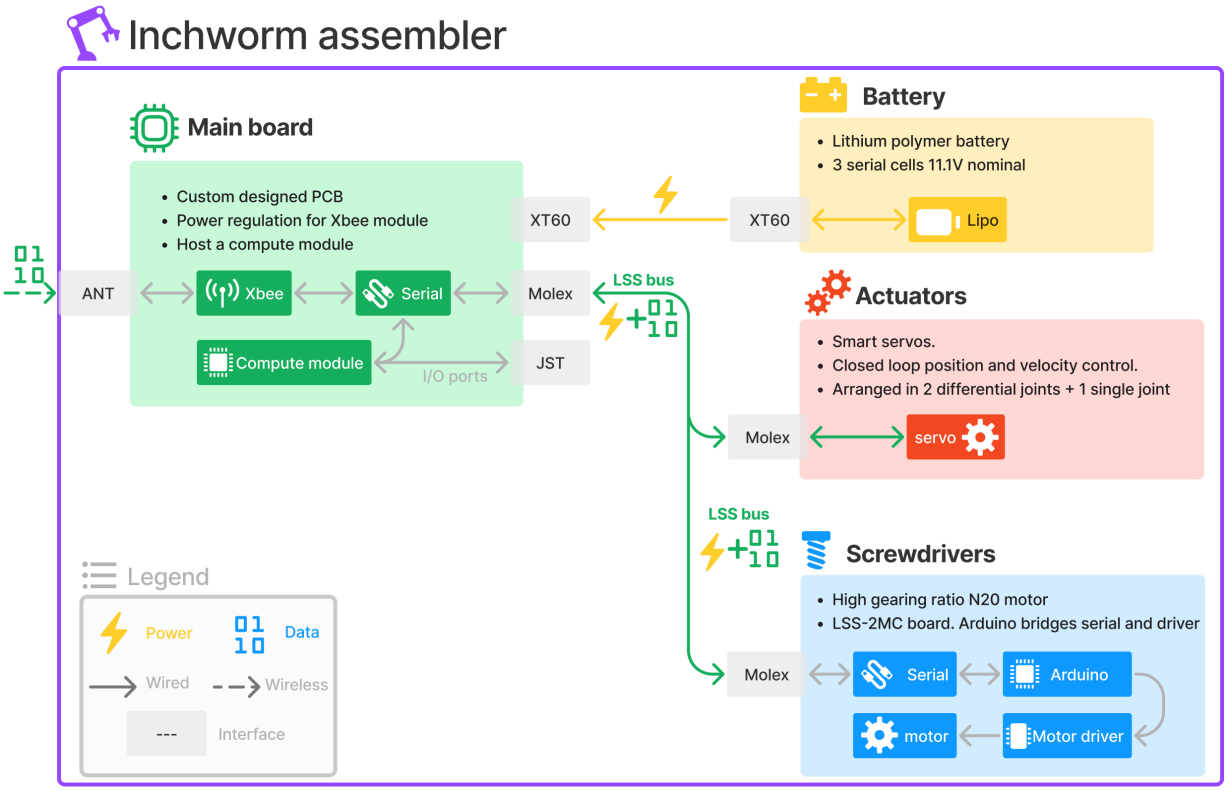


Figure 5.14: Diagram of the electronics architecture of the assembler robot showcasing its different components.

Before diving into all these elements and analysing them in more detail we must understand what the LSS bus is and how the mainboard communicates with the other devices.

5.6.1 LSS bus

In the context of electronics, a ”bus” is a system that facilitates data and power transfer among various components on a shared channel enabling interaction between each other. In this case, the LSS or Lynxmotion Smart Servos bus architecture is utilized. This bus has 4 conductors; 2 for serial communication and 2 for power [82].

Serial communication

Serial communication is a digital communication method employing only two transmission lines to send and receive data simultaneously one bit at a time. An arbitrary number of devices can be connected on the serial line, however, as there is only one line for transmission and one line for reception only two devices can transmit simultaneously. In this case, one device is a single servo and the other the mainboard and one must be careful that multiple servos don't transmit simultaneously.

Bus wiring

The 4 conductors are arranged in the following order:

- **RX:** From the point of view of the servos, this is the receiving line of the serial communication, meaning the servo receives over this line. The TX pin of the mainboard and the RX pin of the servos or other devices should be connected here.
- **VCC:** This is the Common Collector Voltage, that is, the positive supply voltage for the electronics on the bus. LSS servos can function between 6.5V and 12.6V with a recommended operational voltage of 12V. The positive voltage lead of the battery is connected here.

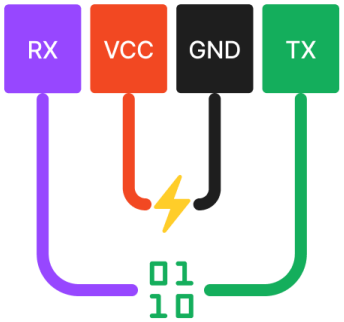


Figure 5.15: LSS bus pinout.

- **GND:** This is the common ground, that is, the negative voltage reference for the power supply. The negative lead of the battery is connected here.
- **TX:** From the point of view of the servos, this is the transmitting line of the serial communication, meaning the servo emits/ transmits over this line. The RX pin of the mainboard and the TX pin of the servos or other devices should be connected here.

The LSS bus is then cabled as shown in Figure 5.16 to reach all the servos and screwdrivers. The servos have two connectors and can be easily daisy-chained. The mainboard is situated near the centre of the robot around the area where the single (non-differential) joint is located. For this reason, two LSS-Bus cables exit the Main Board; one directly inserts into one of the differential joint servos and the other into the single joint and, from there, is daisy-chained to the other differential. The two servos of each differential joint are connected sequentially to then reach a screwdriver actuator LSS-2MC board.

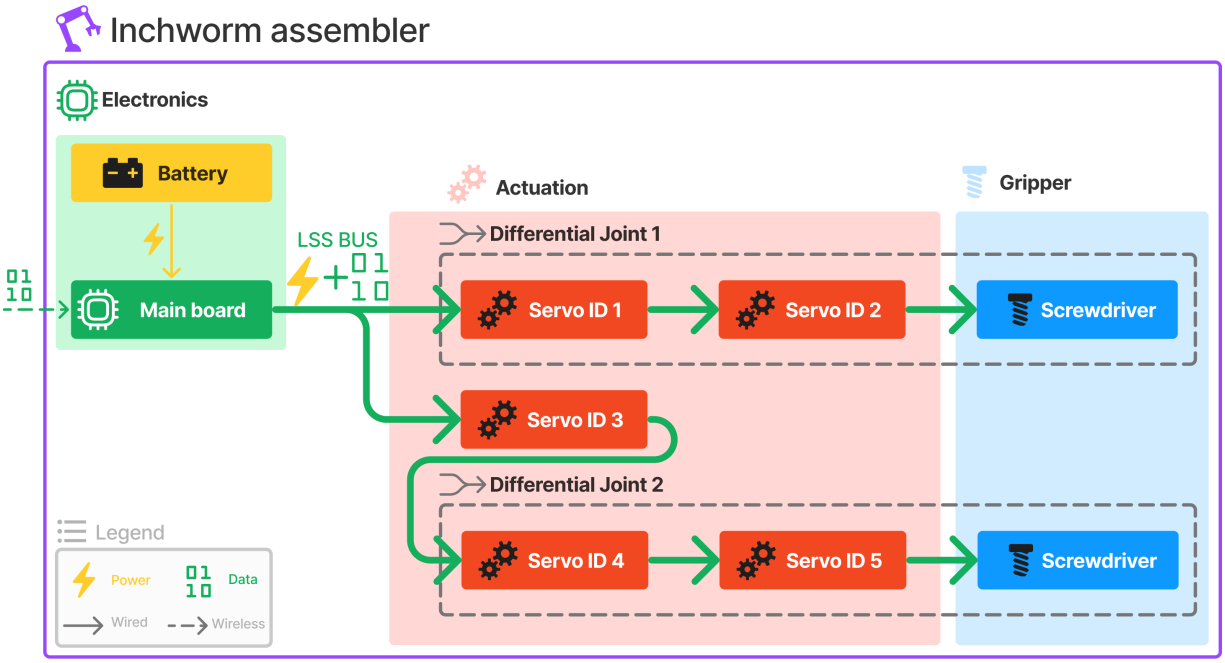


Figure 5.16: Cabling diagram of the assembler showing how different components are attached to the LSS bus.

5.6.2 Main board

The main board represents the primary brain and communications centre of the robot. All the information flowing in or out of the assembler passes through this custom-designed PCB with modules and connectors for data processing and transmission. Its sub-components are shown in Figure 5.17.

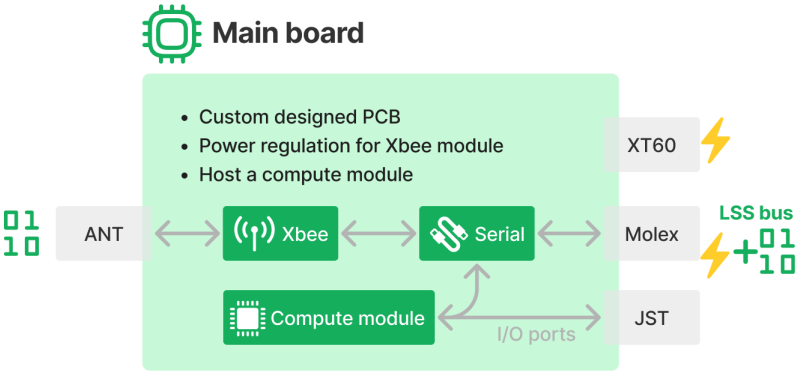


Figure 5.17: Internal diagram of the assemblers main board showing its components and interfaces.

The following sections expand on these components.

5.6.3 Xbee: Wireless communication

The main diagram shows that the mainboard contains an Xbee wireless module, but why was this decision taken over a more traditional radio or WiFi module?

XBee is a series of small form factor modules that support various wireless communication protocols, including Zigbee, 802.15.4, WiFi, and others. These modules are known for their ease of use and reliability in establishing wireless connections. Two Xbee modules can be configured as a serial cable replacement where they emulate a cabled connection. This configuration can be easily done from the XCTU configuration utility[1], with no required knowledge in embedded systems.

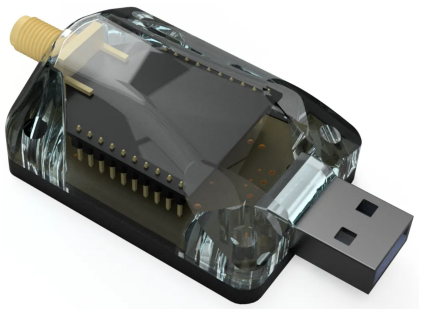


Figure 5.18: Xbee SMA module (black) on USB explorer board in the designed enclosure.

An Xbee USB explorer board is used to connect the Xbee module to the computer. Additionally, the enclosure shown in Figure 5.18 was designed to allow the module to be plugged like a USB stick. This enclosure was made available online on printables [2]. On the computer side, an SMA antenna Xbee module is utilised to maximize gain and range, while on the robot side, U.FL is used to adapt to space constraints.

In summary, Xbees are simple to set up, capable and have a large community around them, making them a good choice to de-risk the integration process, especially since my background is not in embedded systems or electronics. An explanation of how to configure these modules for this project can be found in the online documentation of the code [3].

5.6.4 Custom board design

Initially, the LSS adapter board or LSS-ADA [83] was employed to adapt the Xbee module and battery connector to the LSS bus. However, this board suffered from a series of problems. First, its size is suboptimal for integration into the robotic assembler. More importantly, some reliability issues were encountered. For unknown reasons, an LDO in the voltage supply area of the board combusted passing the 12V input onto the lower voltage logical channels, destroying the board and the Xbee module in it. The investigation into this failure involved the board manufacturer and a master from Embedded Systems at Tudelft, but the reasons remain unknown. After this and the sudden loss of another Xbee module in another of these boards, it was decided to stop employing them in favour of a miniaturized custom-designed board that better aligns with this project's requirements and includes a more robust and efficient power delivery system.

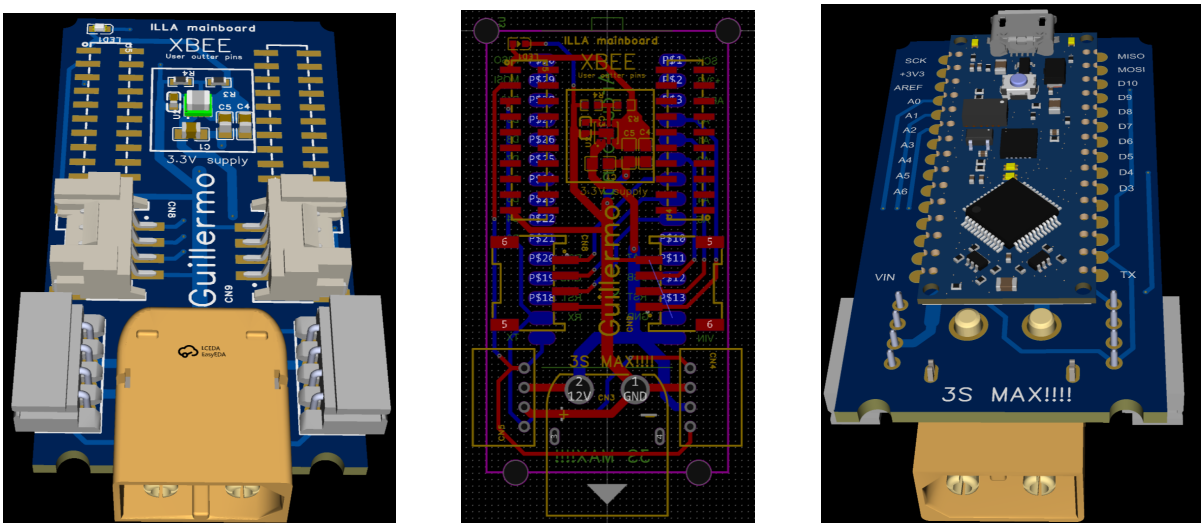
The board was designed using the EasyEDA Ecad tool. The power regulation for stepping down the input battery voltage to the Xbee logical voltage utilizes the TPS82130 step-down converter power module with an integrated inductor from Texas Instruments. This is a fully integrated MicroSiP module with efficiencies of over 80 enabling longer sleep periods with RF communication, quite a contrast from the LDO found in the LSS-ADA board.

This board also includes the possibility of soldering a computing module in the form of an Arduino Nano board on the pads on the back. Arduino Nano represents a wide ecosystem with a plethora of options, including powerful compute platforms like the dual-core wifi-enabled ESP32 modules, Raspberry PI RP2040 or the sensor-packed IOT-focused IMU, microphone, colour, temperature and pressure sensing nRF52840 powered boards... In the case at hand, a simple Arduino Nano every is envisioned to be installed. The possibilities this brings into this field are truly exciting.

[1] https://www.digi.com/products/embedded-systems/digi-xbee/digi-xbee-tools/xctu

[2] https://www.printables.com/model/513026-xbee-enclosure

[3] https://guillermopresa.github.io/AutomatedLatticeAssemblyDocs/html/class_illa_controller_v3.html



(a) (b) (c)

Figure 5.19: Design of the custom mainboard. Top side (left) ECAD design and traces (middle) and bottom side with an Arduino Nano Every compute module (right).

As can be appreciated in Figure 5.19, the board includes an XT60 for battery connection as it represents the most widespread standard in 3S lipo batteries, although an XT30 would be sufficient and save more space. Two Molex connectors for the LSS-Bus are placed next to the XT60, and lastly, two JST connectors like the ones used in the M5 stack ecosystem are placed, offering connectivity to analogue pins in the compute module populated by an Arduino Nano Every in this case opening the door to integrating different technologies and sensors in the assembler.

5.6.5 Screwdriver actuator driving

The screwdrivers are also connected on the LSS bus, however, the N20 motor does not support this high-level communication.

In the evaluation of the smart servos, it was highlighted that Lynxmotion, the selected servo platform's manufacturer, offers a robust ecosystem including numerous accessories. One such accessory is the LSS-2MC, or "Lynxmotion Smart Servo to Motor Controller". This module leverages an ATmega328P microcontroller (similar to those in Arduino Nanos) to bridge the gap between the LSS bus and up to two PWM devices like DC motors, LEDs, or a bipolar stepper motor. The Arduino within the board reads and writes on the LSS-bus serial line and controls two MP6508 $1.2A$ bridge motor drivers with screw-in terminals for device connectivity [84].

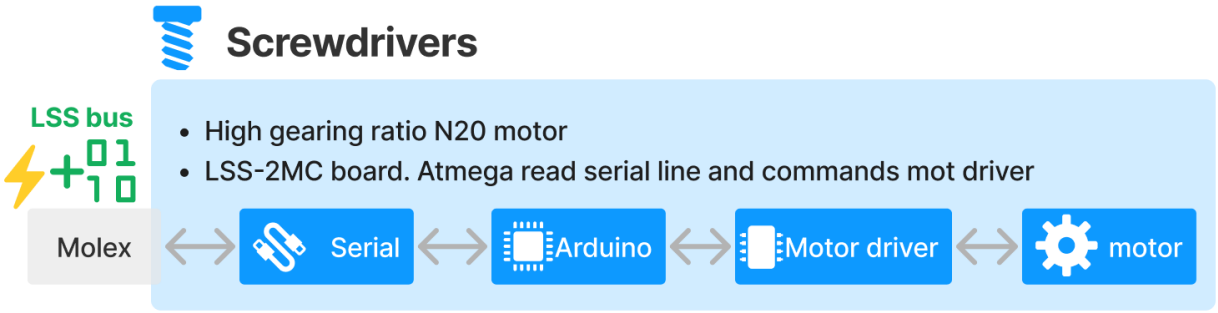


Figure 5.20: Internal diagram of the gripper screwdriver showing its components and interfaces.

The LSS-2MC board is shown in Figure 5.21. The left image shows the top side of the board, with the driver below a small heatsink, the connectors for the LSS-BUS and screw-in terminals for the driven

actuators. On the bottom side, an Arduino ATmega328P microcontroller is included

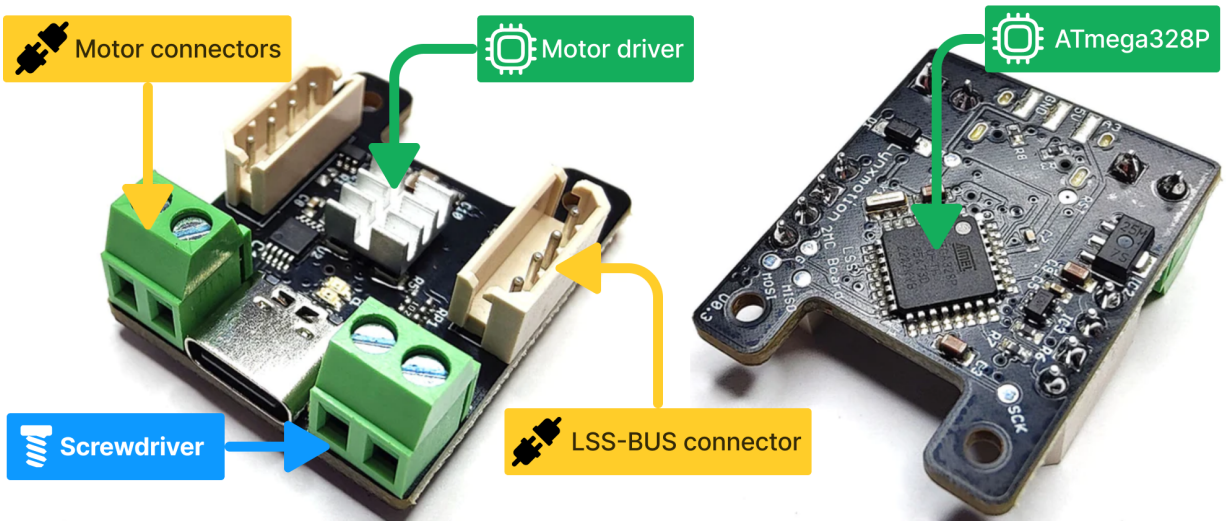


Figure 5.21: Components of the LSS-2MC board used to control the screwdriver module.

The LSS-2MC microcontroller can be loaded with sample code provided by Lynxmotion, that easily allows controlling the motor drivers via the LSS bus [85].
Some changes were made to the provided code to allow it to run. These modifications include fixing some aspects of the code to make it work properly with two motors at once, as well as expanding its functionality with additional commands. The modifications better leverage the microcontroller to add higher-level functionality. For instance, new commands were added to the LSS protocol that allow to actuate the screwdriver for a specified amount of time. This is useful as it allows the main controller to tell the robot to screw in for 5 seconds instead of needing a command to start screwing and one to stop screwing, lightening the load of the software platform by using higher-level commands. Additionally, some changes were needed to allow the board to function with two motors plugged in.

As already mentioned, a new action was added to the protocol to represent a movement for a limited time. This new action moves the motor with a full duty cycle for the specified amount of time in milliseconds. This new action Raw-duty time has the command ”RDT” instead of ”RDM” for the original raw duty cycle move. Ideally, a two-command system may be used to set both the speed and duration of the motor actuation. To add this new action the the LSSCommunication.cpp and LssCommunication.h libraries were changed together with the main LSS2MC code running on the LSS-2MC board. As a note for my future fellow researchers when downloading the LSS libraries for Arduino use the modified ones hosted on Unity DevOps and put it in the Arduino libraries folder or set the Arduino sketchbook directory to your unity directory where this code is allocated [4].

The code running in the LSS-2MC Arduino was modified to utilise the millis() function to store the runtime milliseconds at the beginning of the RDT command and check every frame whether the current millis() minus the ones at the beginning of the command is larger than the time to wait. This approach using millis() will roll over after roughly 47 days. For long build processes, this implementation must be revised. Lastly, sa note of caution, although it doesn’t seem that way from the LSS-2MC communication protocol page, the ID of the second driver must be the ID of the first +1.

5.6.6 Battery

A battery is needed to power the robot. The constraints are that it should be capable of powering the robot for an ample enough time for constructing a chunk of a structure, about 1 hour at least, while being lightweight enough for the robot to take the extra load and having enough maximum power for high torque

[4] \Assets\Resources\Objects\ILLAV3\LSS2MC_arduinoCode

actuation scenarios.

Lithium batteries are the easiest to come by nowadays as the worldwide industry and supply chain have adapted to this chemistry. Additionally, it's the highest-performance battery technology in widespread production today. Different lithium battery technologies exist.

Battery technology
The predominant and most widely and easily obtainable lithium battery technologies are compared. Distinctions can be made according to the physical phase of the electrolyte and the anode and cathode chemistry. For instance, lithium polymer batteries use a dry or gel-like electrolyte, and lithium-ion cells use liquid electrolytes. Whithin lithium ion we can differentiate multiple very popular chemistries out of which NMC or Nickel manganese Cobalt and LFP or Lithium Iron phosphate are the most widely utilised. These chemistries refer to different anode and cathode materials.

A quick comparison on the basis of specific energy and power, volumetric energy, lifetime and packaging is provided in Figure 5.22 in the form of one radar chart for each battery technology with the requirements overlayed on top in grey. To represent each technology, one of it's most suitable and highest-performing cells was selected. Lithium polymer data is extracted from Tattu's 3S 850mah pack. Representing the NMC technology are the Panasonic-NCR18650 [5] and the LFP cells are represented by AA Portable Power LFP-26650-3300 cells [6].

The requirements overplayed in grey prioritise packaging and specific power as these are the elements that must, under any circumstance, be satisfied. A battery without a high enough power rating, where the voltage drops as the servos demand more power or whose packaging doesn't allow specific sizes, can't be deployed. Specific and volumetric energy are relevant and dictate the duration of the construction job between battery charges but don't present a project-level risk like the previous two. The same thing applies to battery lifetime.

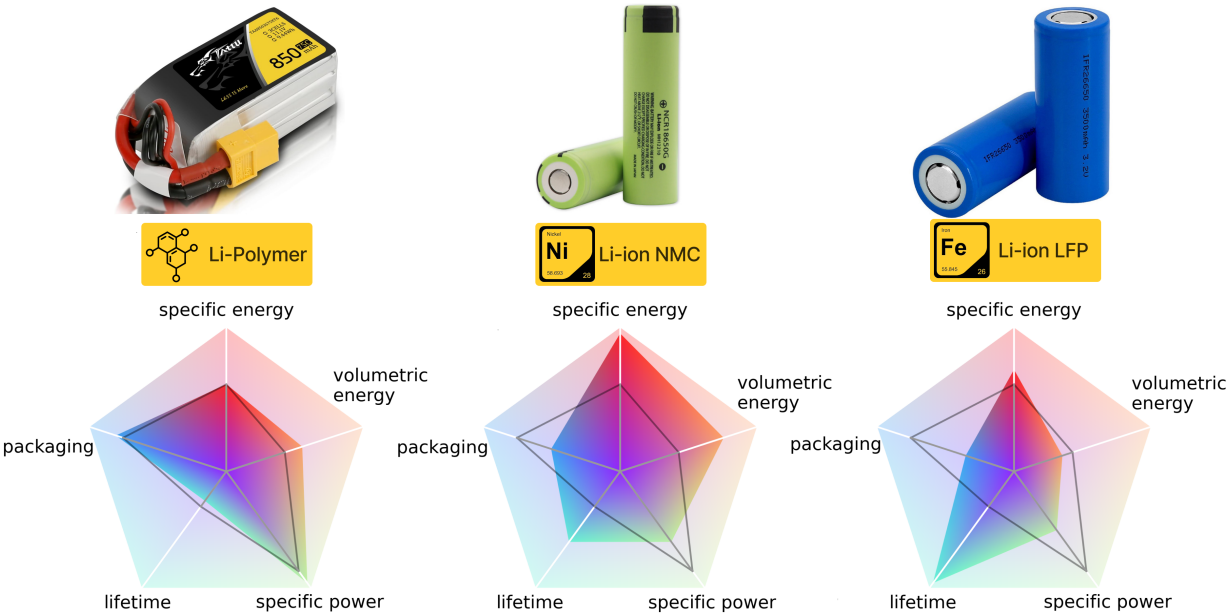


Figure 5.22: Comparison between three different battery technologies. Radar charts compare their performance with the goal or desired parameters overlayed with a grey line.

Lithium polymer presents a strong advantage in packaging. LiPo cells are prismatic, and the dimensions can easily be modified, leading to rich product lineups with many different form factors and capacities,

[5] Panasonic-NCR18650 test results https://www.shoptronica.com/files/Panasonic-NCR18650.pdf
[6] AA Porbable Power LFP-26650-3300 cell datasheet https://www.batteryspace.com/prod-specs/2385.pdf

especially in the small battery market. Additionally, multi-cell batteries are available with multiple cells in series for a higher voltage with built-in connectors and a complete ecosystem for charging and discharging with COTS components exists. Thanks to their low internal resistance, LiPos are also able to deliver large amounts of power without significant voltage drops, as evidenced by their discharge ratings of up to 100C. High discharge rates are necessary in small batteries if all the servos are to be actuated simultaneously, reaching currents of up to 12A. However, they suffer from a very limited lifetime in the low hundreds of cycles and a mid-range energy capacity of about 170wh/gr.

Although it is technically possible to design and develop a higher-performance battery packs using lithium-ion technologies with superior energy densities reaching 260wh/g and lifetimes of up to 2000 cycles, the time and risk involved were not deemed within the scope of this thesis. For this reason, convenient packaging and specific power are valued more than anything else and a simple off-the-shelf LiPo is used. In this case, two different 3S models are used with 1050mah and 850mah. The latter is shown in the battery technology comparison Figure 5.22.

Battery safety
Working with lithium batteries in the lab required the development of battery safety guidelines as this is not a common occurrence in the ASM department. These guidelines establish how the storage, charging and usage of the batteries shall be performed. It's based on the principles of preventing possible accidents and having mitigation strategies in place if they occur. The guidelines developed can be found in Appendix C.

5.7 Final assembler robot design

After this long analysis of the different components of the assembler robot, the manufactured version is showcased in Figure 5.23.

As can be seen, it is composed of two differential joints and two arms that contain the servo motors. Each differential joint holds an LSS-2MC electronics board and one storage location on opposite sides. One of the arms contains the custom-designed electronics board with the Xbee wireless module in the black enclosure connected to the lithium polymer battery. On the opposite side, an LSS-2IO module was added to this development version to enable a USB interface with the robot.

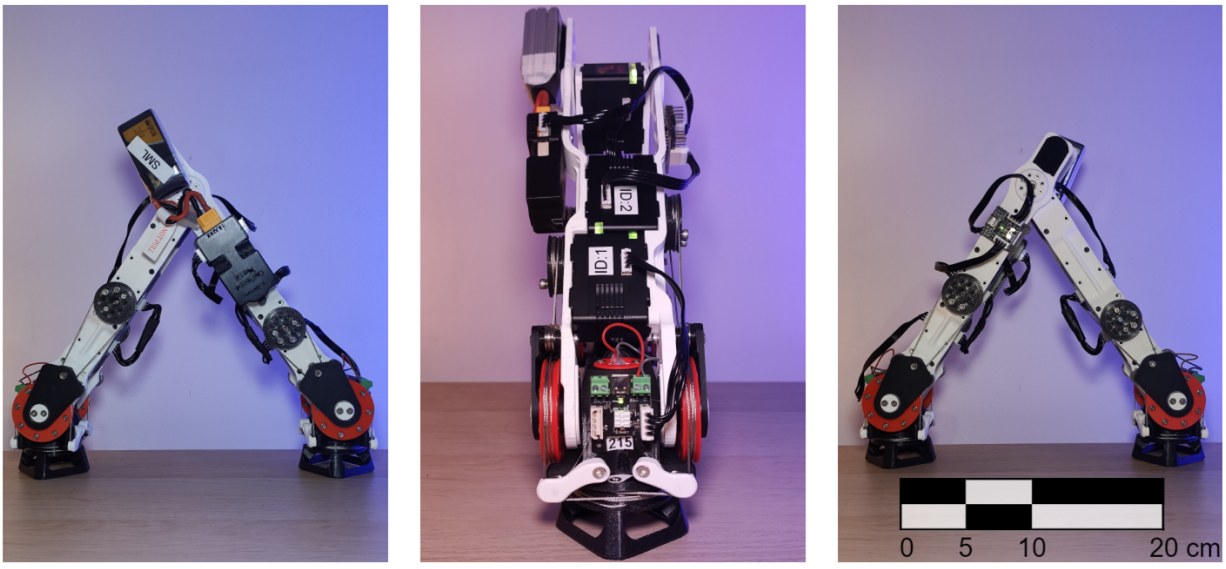


Figure 5.23: Images of the final assembled robot from three different perspectives.

6

Control software and path planning

Automated assembly represents the process of constructing a structure autonomously utilizing robotic assemblers. In this manner, an initial geometry description must be broken down into robot movements in a process similar to slicing a 3D model into Gcode or machine instructions. The order of material placement is planned and optimized, taking into account the assembler and other robotic assets that might partake. The software is the intelligence in charge of interpreting the goals and breaking them down to execution, it is the planner and the problem solver. In the field of artificial intelligence, these terms have been defined quite similarly with my favourite being:

> *"Planning is the reasoning side of acting"* [86]

Irrespective of whether this intelligence arises from a collective of agents like in biological systems or from a centralised high throughput actor, it must conform to the following goals:

- **Flexible:** The software platform shall enable the construction of digital materials with varying geometries, freeing the design from strict grid constraints. This design freedom, combined with topology optimisation techniques, has the potential to empower a leap in the structural performance of digital materials. Similarly, multiple assembler types shall be able to work simultaneously.
  Flexibility is a major enabler for construction speed and structural performance. Imagine a swarm of different types of assemblers, from drone-like flying robots to inchworm locomotion ones, assembling a structure that exploits curved lattices to optimise for particular load paths.
- **Deterministic:** Unlike biological assembly, where there is no complete definition of the goal geometry, this process shall reconstruct a 1:1 copy of the input structure in a deterministic manner. Running the process with the same inputs must always lead to the same behaviour, repeatability is key in a development platform where testing and expandability are needed.
- **Digital fidelity:** The digital information realm shall closely represent the real-world assembly process. This means that a high-quality digital twin of the structure is developed, increasing trust in the simulation capabilities of the software, virtual build processes and testing.
- **Scalable:** The computational architecture of the software platform must easily allow scalability beyond this thesis. A construction job with a few thousand lattices and tens of robots working simultaneously should be computationally feasible without needing a major rewrite of the software.
- **Expandable and modular:** The software shall compartmentalise different processes in modules that can easily be upgraded and modified without changing the complete software. This reduces development complexity.

In the diverse world of software development, different needs are fulfilled by a plethora of languages and platforms. For instance, Python presents a high-level, easy-to-use interpreted language, while C++ is more tailored for low-level and embedded applications. Similarly, different platforms exist that integrate useful functionality for robotics, spatial manipulation and rendering. For instance, Robot Operating System (ROS) is a set of frameworks for robot software development in C++ and Python.

In this case, Unity, the real-time development platform and game engine, was utilised. Unity is a simple organizational framework with powerful spatial and rendering capabilities built in. This framework is composed of a scene in which GameObjects exist. GameObjects are the base class of entities in the

Scenes: they are empty containers for Components or other GameObjects, allowing hierarchical relations. Components are the functional pieces of GameObjects: they can be anything from a user-created script to built-in rendering, physics, animations, particle systems, etc. For the most part, only renderer, collider, and script components are used to keep the project as independent from Unity-specific tools as possible. These scripts are written in `C#`, a high-level compiled language that combines the performance and robustness of C++ with the ease of development of Python.

With these goals in mind, a software architecture is now presented and expanded in the next sections.

6.1 Architecture

In the literature in Chapter 2, different high-level organisational architectures, from emergent behaviour to centralised control, were discussed. A hybrid system is developed that has a centralised build coordinator together with robot controllers that can plan and make decisions on their own. This allows both centralised and distributed control but also a form of high-level centralised control where the central planner deals with high-level concepts and not the specifics of what each robot should be doing.

This hybrid computational architecture is integrated into a digital twin where robot controllers, that is, digital copies of the robots have access to the build context and can perform computations on high-performance compute platforms. A centralised build planner exists that ideates a construction plan to reach the goal or target structure from the current one. The build planner can interact with the separate robot controllers via a work package auctioning system.

The four main elements that make the software platform; Build planner, robot controller, context and user interface are presented in the software overview in Figure 6.1.

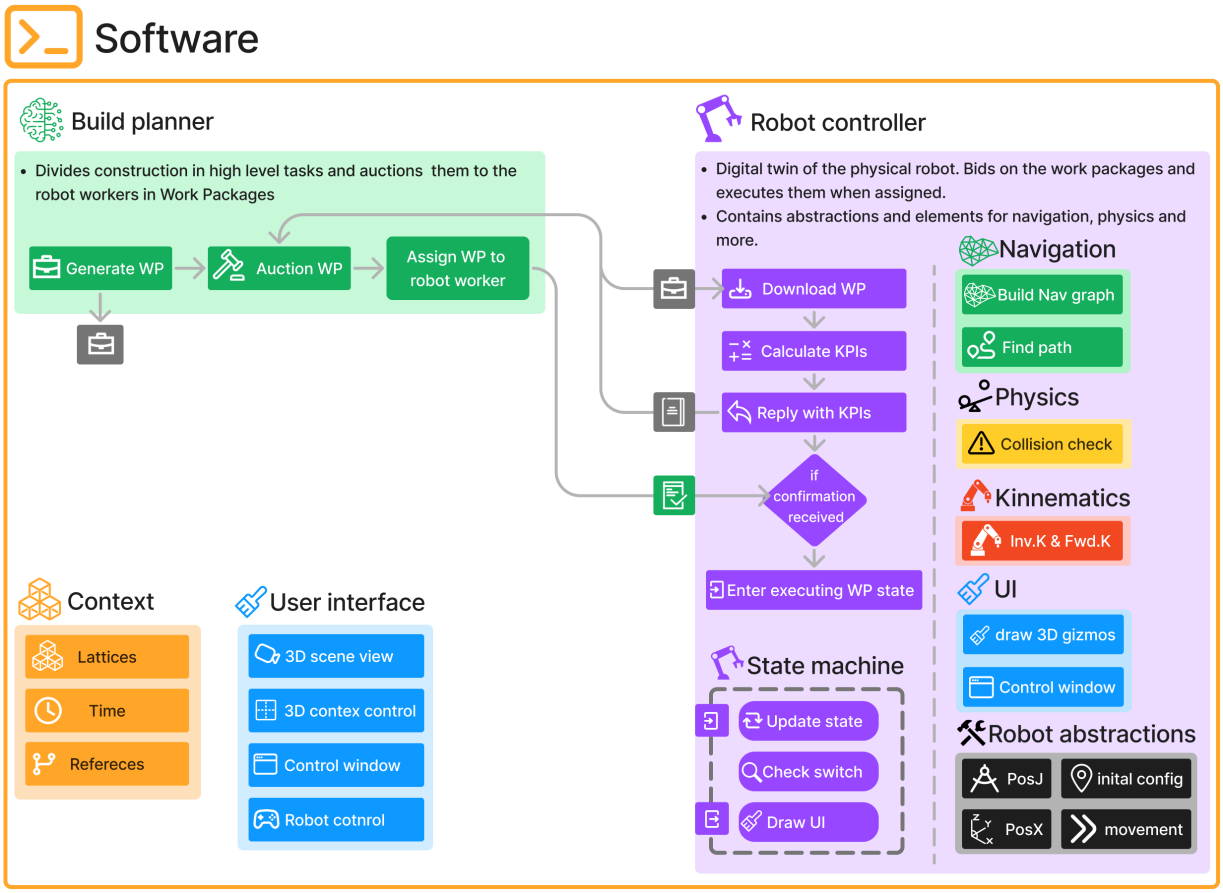


Figure 6.1: Diagram of the software platform showing its high-level architecture and the elements that compose it.

- **Context:** The context is composed of multiple elements that store information, like the topology of the target and the currently assembled structure.
- **Build controller/ robot coordinator:** The build controller or robot coordinator creates and distributes the tasks that the robots execute.
- **Robot controller(s):** The robot controller is the virtual representation of an assembler robot. It is its digital twin and englobes all the elements required to utilise the robot.
- **User Interface:** Allows the user to interact with the program and visualise the build process.

6.2 Context: Geometry representation and importing

Context refers to the information contained in the digital twin world. This includes the topology of the target and the currently assembled structures, as well as references to the different objects and robots that are present in the world.

To understand how the context comes together, we must first analyse the representation of lattices in the digital twin environment.

6.3 Lattice representation

A plethora of ways to represent lattices exist with different advantages and disadvantages. As an example, a 3D array could very efficiently represent spots in a grid that are populated by lattices. This representation would also easily allow the addition and subtraction of different geometries, allowing the identification of the lattices to be reconfigured. The value of the elements in the array could represent lattice density or IDs.
However, this has some shortcomings. First of all, the first goal presented in the introduction to this chapter is flexibility. In a major break with the current research tools in digital materials, this thesis proposes a solution that does not require a strict grid representation of the materials but rather introduces more design freedom.

For this reason, each lattice is represented as its own GameObject, storing its position, orientation and more. Figure 6.2 showcases the components that make a lattice in the Unity program next to the render of one taken from the program itself.

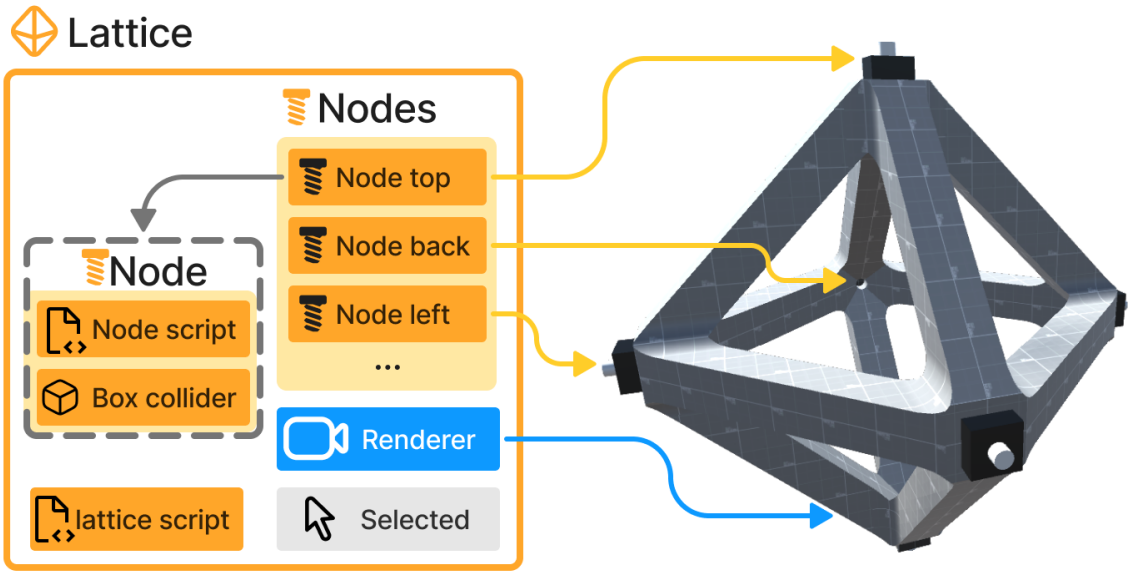


Figure 6.2: Representation of the lattice building block in the software platform.

The lattice GameObject is composed of node GameObjects, a lattice script, a selected script and a renderer component. The renderer is capable of displaying a mesh, like the lattice 3D model shown in

the render. The lattice script is primarily used as a container to store some information, like the density of the lattice, its geometry and its role in the structure, meaning whether it is placed, available for pick up, or part of the target geometry definition (as later explained in Figure 6.7). This script can perform some computation upon request, like identifying the neighbours of the lattice or determining whether it is part of the construction front, meaning the unplaced lattice target locations that are on the boundary of the constructed geometry and represent the next assembly locations. The selected script is added not just to the lattices but any object and allows their selection from custom editor menus showcased in the user interface (UI) section in Figure 6.24.

Similarly, the nodes are GameObjects that contain a node script and a box collider. The box collider enables certain physics revolving around the lattice nodes. For instance, the robot controller contains a node sensor that detects these colliders to ensure screwing is possible before attempting to do so.

Lattices are saved or stored as a Prefab. Prefabs allow the storage of GameObjects with all their components, values, and children to be instantiated into GameObjects in different scenes later on.

Now that we understand how a lattice is represented in the software, the following question is: How can we create structures out of these materials in preparation for an assembly job?

6.4 Lattice instantiation and geometry saving

Lattices are Prefabs and, as such, can be instantiated or placed in a scene in the virtual environment; however, tools are needed to place them at the correct locations. A spawning system was developed that allows to select a prefab from a drop-down list on the UI window and place it on an existing lattice node by clicking on it. While hovering with the mouse over nodes, a preview is seen where the user can rotate the prefab to be placed using the keyboard numeric pad. The numbers 1 to 6 change the node where the new object is attached to the node being hovered on. Nodes 7 and 9 rotate this new object. This process is showcased in Figure 6.3 with an asymmetrical lattice used for arch building.

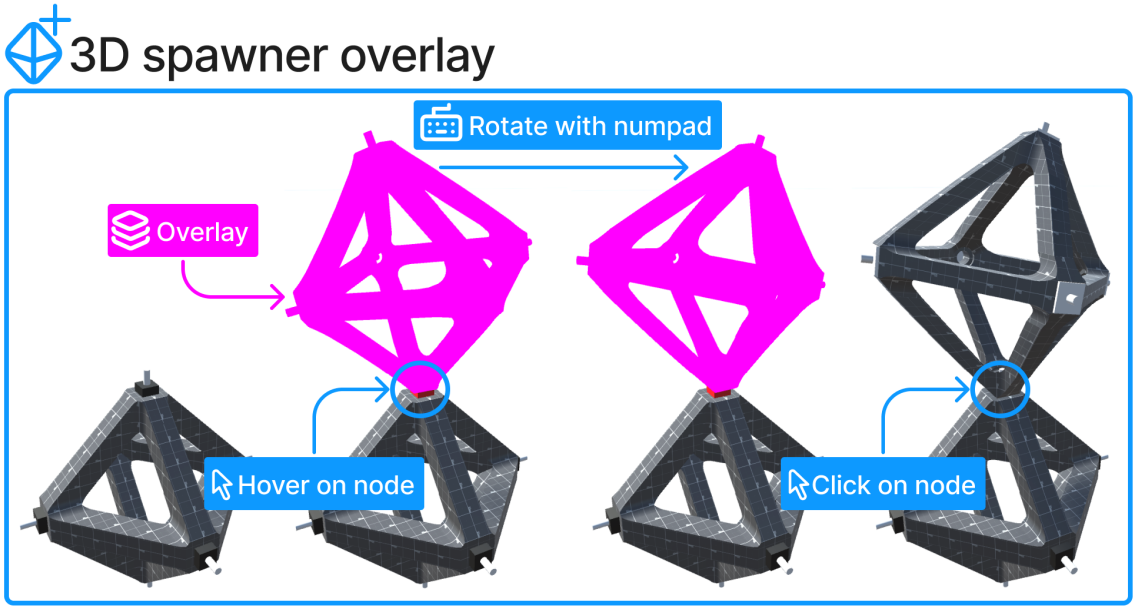


Figure 6.3: 3D overlay shown in the software during object spawning. The new object that will be spawned is highlighted or overlayed in pink.

Once instantiated, lattices can be selected and modified, but more importantly, grouped into objects of lattices can be saved in large prefabs. These objects can later be spawned at once, allowing the creation of complex geometries easily.

However, building objects manually is not the be-all and end-all complex geometry creation system we seek. Often models created in CAD software or the output of a topology optimization process in an external program are to be input and utilized as geometry definition. For this reason, a geometry importing system and workflow were also developed.

6.5 Geometry importing

Two major sources of geometrical data are of interest as input to this software. 3D models generated from CAD or 3D modeling software, like Fusion360, CATIA, or Blender as well as the output of topology optimization programs. Additionally, this discrete assembly process can benefit from utilizing a time-space topology optimization to optimize not only the final state, but also intermediate ones that account for the manufacturing loads on the uncompleted structure as elucidated in [87]. For this reason, timestamp data is added as an input field.

Reading and interpreting 3d models is not a simple task and would require some work and dedication. If only there existed a ubiquitous tool that converted it to something easier to read, preferably something similar to the output of topology optimization programs. Those versed in the 3D printing world may have realized that we can simply slice the 3d model with a slicer designed for stereolithography to generate a series of black-and-white masked images, just the same as the density fields of topology optimization. In this case, chitubox with a custom printer profile is utilized. We then treat these images as a voxel field to generate an intermediate representation of the lattice structure in the form of a JSON object that lists the lattices, their positions, densities, and timestamps. Bridging the gap from images to JSON objects is the role of the import tool showcased in orange in Figure 6.4. Here grayscale images, result in lattices of varying densities.

The JSON object represents an intermediate and flexible open format that's easy to cater to when developing tools for data importing from more varied sources. It also presents the possibility of adding a header with the name of the parent object which all the lattices should be spawned under.

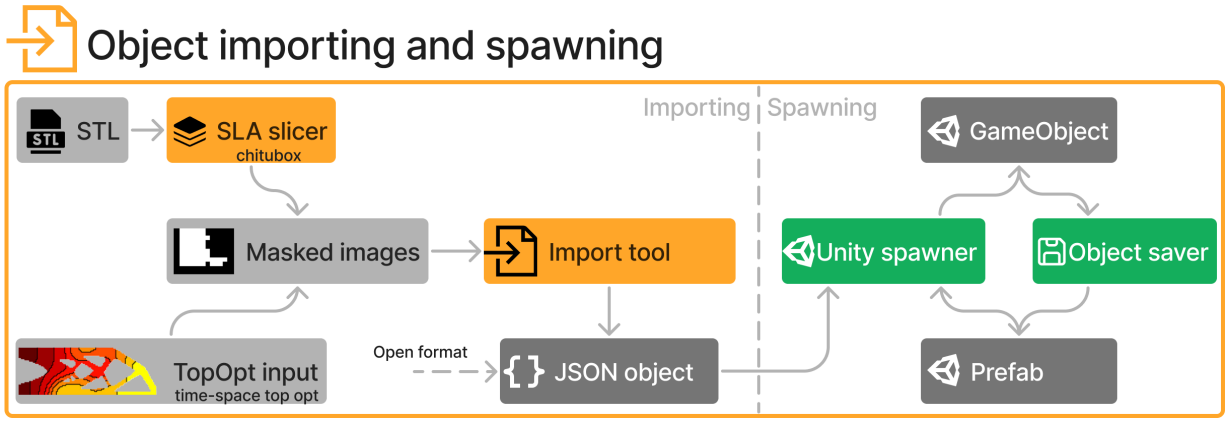


Figure 6.4: Diagram showing the 3D geometry import and object spawn workflows.

The spawning tool can instantiate GameObjects from these JSON objects that then can be modified and saved into prefabs. It is recommended to save the objects and spawn from a prefab instead of always doing it from a JSON file as it is more computationally efficient and introduces some advantages like showing the overlay. This is because the merged mesh used for the overlay is created in this saving process.

A showcase of two objects imported using this system is shown in Figure 6.5. On the left, a lunar base dome structure partially covered with regolith is shown, and on the right-hand side, a topology-optimized bracket is displayed.

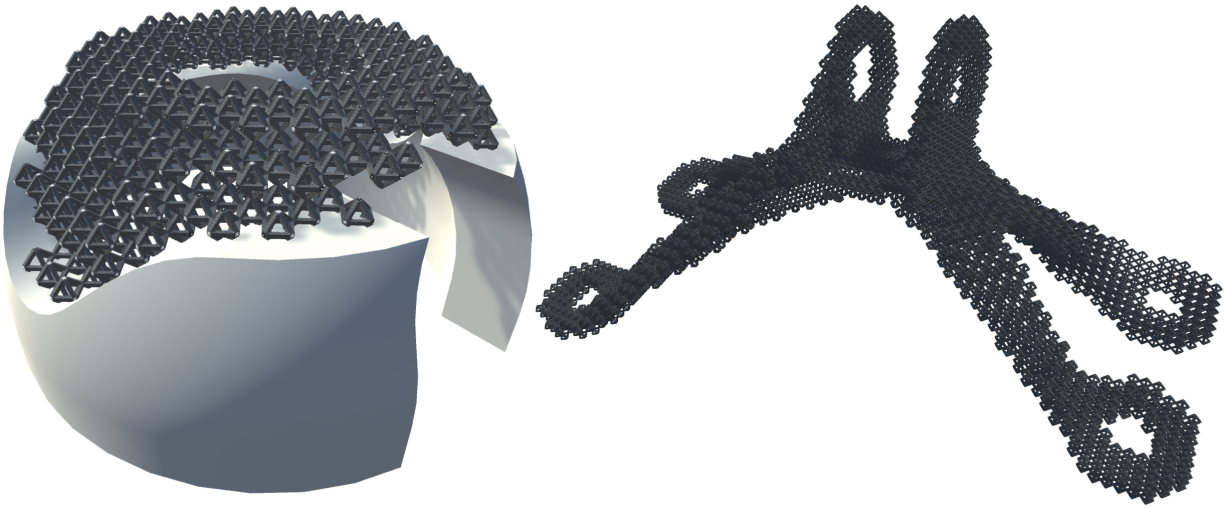


Figure 6.5: Two imported geometries rendered in the assembly software. On the left, a lunar base dome structure partially covered with regolith is shown and on the right, a topology-optimized bracket is displayed.

6.6 Build planner

The build planner analyses the context with the current and target geometry definitions to ideate a high-level building plan. To coordinate multiple different robots during assembly, a work distribution system has been integrated that connects high-level construction goals to detailed implementation-level control of different robotic assemblers. As illustrated in the overview in Figure 6.6, there are three main steps to the process. First, a work package is generated, it is then auctioned and lastly assigned to a robotic assembler.

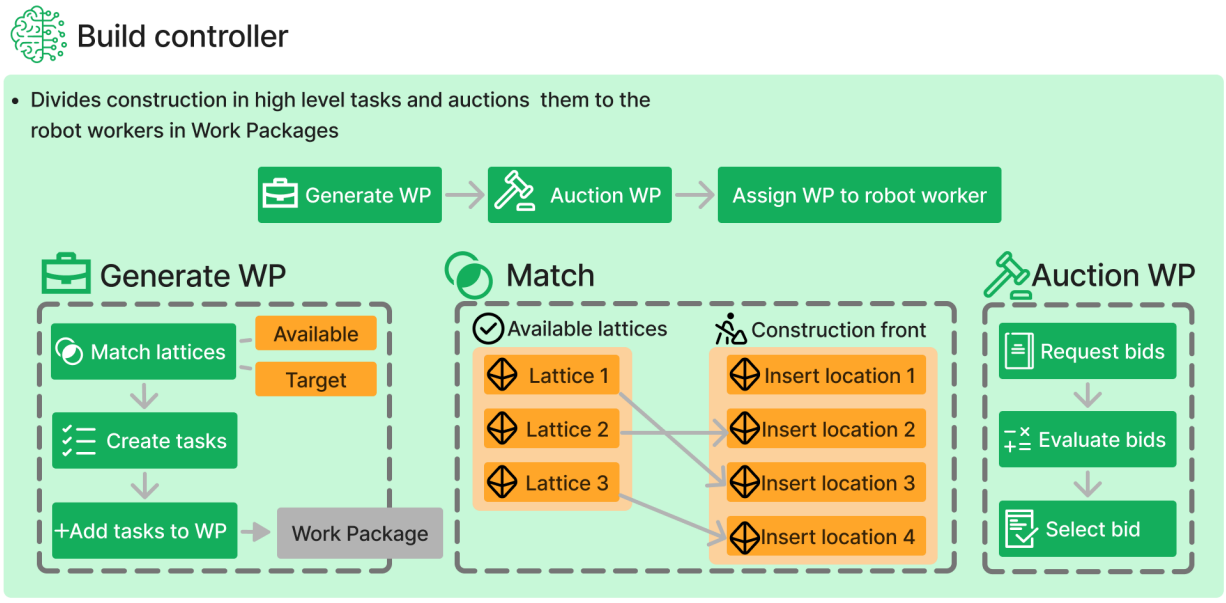


Figure 6.6: Diagram showing the build controller, its sub-components and how auctions are carried out.

First, let's analyze how the next construction step can be generated to then see how it is auctioned. Lattice scripts have different properties that indicate whether the lattice is available for pickup, it is placed, or it is part of the target geometry. This allows easy identification of the lattices that can be picked up and with some computation, the lattices that are part of the construction front, that is the unplaced lattice target locations that are on the boundary of the constructed geometry. As shown in the diagram, the first step in generating a work package is matching the available lattices (in green) to assembly locations in

the construction front (in blue). For this, a bipartite graph is created and matched utilizing the Hungarian algorithm [88], in particular, the implementation from [89]. The matching process takes the set of available lattices and lattice assembly locations in the construction front and matches them such that the total Manhattan distance heuristic is minimized.

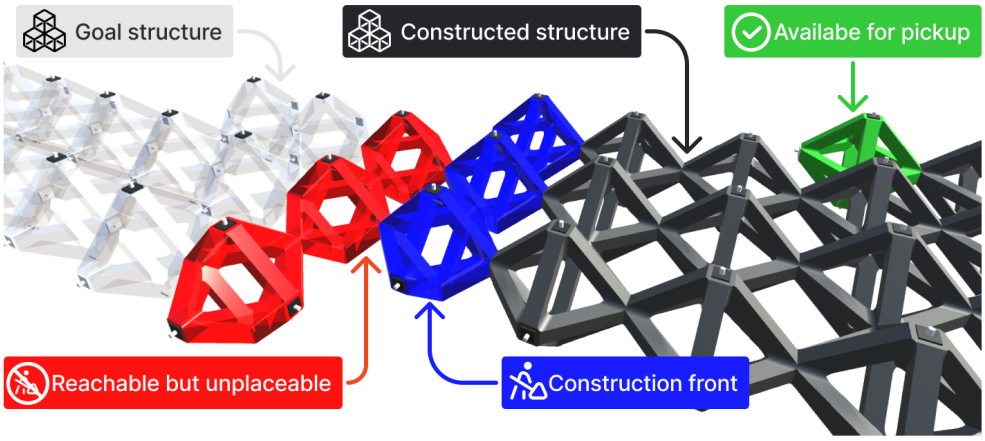


Figure 6.7: Partially constructed structure showcasing the various lattice states.

With the matched pairs, tasks are created. A task is a high-level general representation of work that can be budgeted and executed by different assembler types. For instance, in this case, the task is picking up a lattice and inserting it elsewhere, but more tasks exist. The task does not specify how this work should be carried out, only the end goal. Two very different assemblers, like the inchworm robot developed in Chapter 5 or a flying drone, can implement this task interface.

Multiple tasks can be grouped in a work package (WP). These work packages can be assigned to different robots utilizing an auction system. Here the build controller publishes a work package and the robots that are free respond to the work package with a bid proposal that establishes the cost in terms of some Key Performance Indicators like power or time required to perform the job. It is the responsibility of the assembler to implement strategies to budget work packages. Then assembler robots and tasks can be matched in a similar way as the matching of lattices and assembly locations with bipartite graph algorithms or with a simple greedy algorithm.

Once a work package is assigned to a robot controller for execution, the robot controller breaks down the tasks into actions to perform them. Remember that tasks are high-level representations of work, common to all assembler types. On the other hand, actions are an assembler-specific more granular representation of work. Actions may be broken down into lower and lower-level actions as a strategy to break a high-level task into executable bits. Following the previous example, the task of picking up and inserting is broken down into many walk actions followed by a lattice pick-up action, walking more steps, and finally an inserting in location action. But at the same time, each walk action is further broken down into many simpler move PosJ and gripper screw-in/out actions.

This auctioning architecture increases modularity, separating the roles of the build planner and robot controllers, and allows an increase in segregation of the computation across multiple cores and devices. The concepts described above are summarized in the following list.

- **Tasks:** Assembler independent high-level units of work.
- **Work-Packages:** Groups of tasks that can be published and auctioned. They represent large goals or objectives that need to be completed by the same worker.
- **Bid proposal:** A reply generated by a robot controller detailing the KPIs (like time and energy) required for performing a work package.
- **Actions:** Robot-specific low-level granular unit of work. Each robot has a different set of actions it can carry out. The robot has a scheduler that allows it to break down tasks into lists of actions to be

carried out. The implementation of actions is analogous to a state machine where there is a setup method, a method that runs every fame, and lastly a check on whether the action is finished.

- SetupAction: This is called the first time the robot starts working on that action. Can be used to calculate parameters needed to execute the action or even to break it down into smaller sub-actions.
- ExecuteActionFrame: This is called every frame the robot is executing that action. Used to for example command the robot motors or screwdriver.
- HasActionFinished: Exit condition for the action. When the action work has been completed, this should return true.

For details on the implementation see the code documentation and the code itself but as a summary, abstract base classes exist for the tasks, the actions, and the robot controllers. These base classes implement all the required interfaces. The robot controller implementation overrides the desired methods for estimating KPIs from tasks and breaking tasks into actions. Similarly, actions override the SetupAction, ExecuteActionFrame, and HasActionFinished methods of the base class, integrating their implementation.

6.7 Robot controller

Robot controllers are the representation of the different robots in the digital twin environment. Each robotic asset operating in the real world has its own virtual counterpart, a robot controller that bridges the gap between the physical assembler, the virtual context, and the build planner.

Different types of robot controllers can be developed that derive from a common AbstractRobotWorker class that integrates all the required interfaces. In the case at hand we will discuss the ILLAController, that is the Inchowrm Locomotion Lattice Assembler controller, the one developed for the robot that was designed for this thesis in Chapter 5.

Controllers integrate different modules, that store the necessary information and perform the computation and communication required to properly utilize the robot. Their main elements or components are showcased in Figure 6.8. As can be seen, the robot controller is a state machine with five different states, that represent different conditions of the robot. Additionally, different modules for navigation, UI, communications, physics, and general abstractions are integrated into the controller. The connected state is expanded to illustrate the interfaces with the build planner completing the picture painted in Figure 6.1 at the beginning of the chapter.

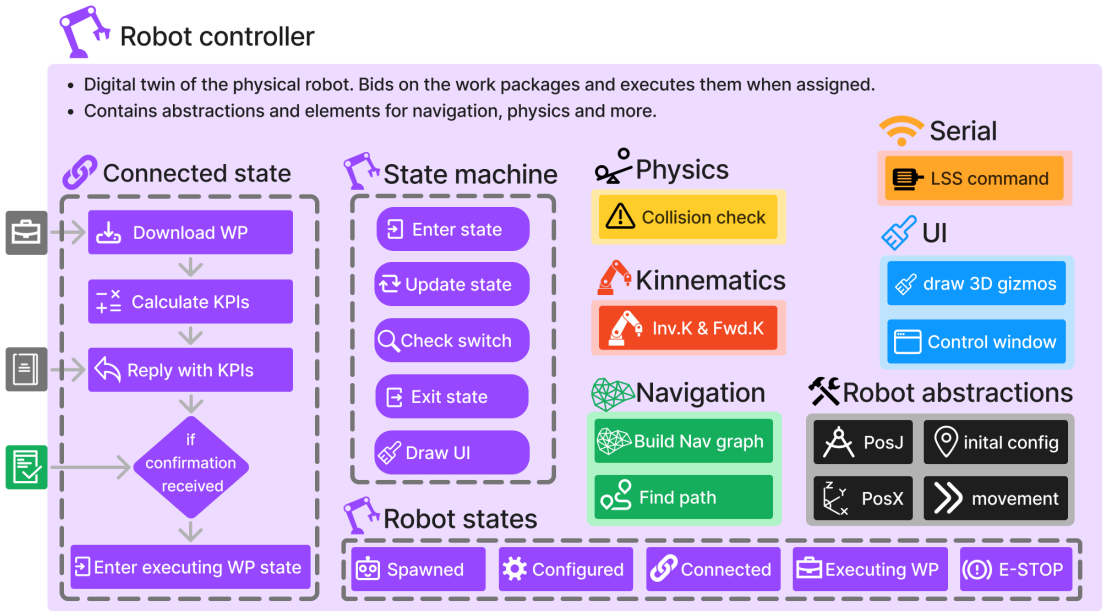


Figure 6.8: Diagram of the robot controller and its modules or sub-components.

6.7.1 Finite state machine

The robot controller integrates a multitude of functionalities and different modules and depending on its current condition it behaves in different manners. For instance, sometimes it's connected to a physical robot and receiving inputs from a human via the user interface others, it's executing a work package and working autonomously, and sometimes but rarely it's in an emergency stop mode.

As the robot controller grows in size, complexity, and number of such conditions or states new architectures are sought and implemented to help manage this rise in complexity and improve maintainability. In this case, a finite-state machine model is utilized for the robot controller. A finite state machine is a model used to represent a system with a limited number of specific conditions, known as states. This system can only be in one state at a time and changes from one state to another in response to external inputs. These changes are called transitions and have predefined rules. Finite-state machines are widely used in computer science and engineering to design and understand various systems and processes, such as software programs and electronic devices. States provide a clear and structured way to handle different scenarios and actions within a system and allow them to tailor the processes and outputs depending on the current state and inputs.

Different types of state machines exist where outputs can be produced on transitions, on states or both and where multiple levels of hierarchical depth can exist. This implementation is a mix of simple Mealy and Moore state machines where predominantly the states produce an output like in Mealy machines but the transitions may also produce them. Additionally, and as will be seen in this chapter, the work package execution system can be thought to add a second hierarchical state machine within one of the states making the classification a bit more complicated.

A simple explanation of state machines and an example of their implementation in `C#` in the context of a Unity character is provided in [90]. The implementation found in the code is similar and inspired by the mentioned example, however, with an additional update cycle on each state for the UI elements of the robot that is called upon request from the control window on the UI update frame. This allows each state to display different UI elements.

The robot state machine is shown in Figure 6.9. Here the different states and transitions between them are shown. A summarized explanation of each state is provided in the list below.

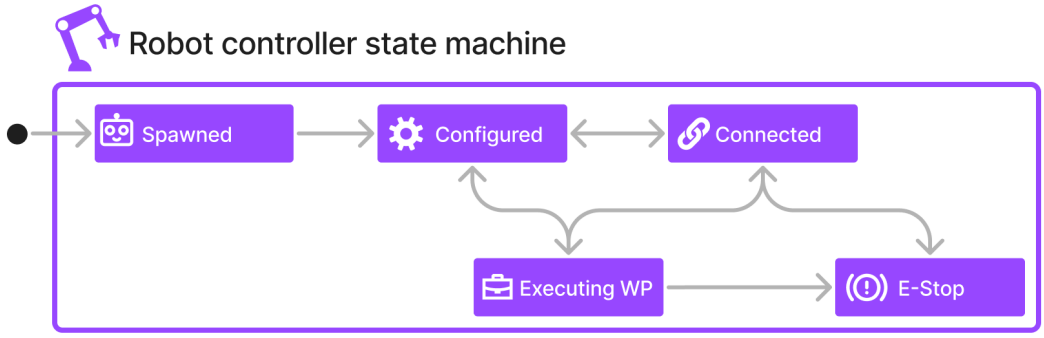


Figure 6.9: Diagram showing the behaviour of the robot controller state machine and it's possible transitions.

- **Spawned:** The initial state to which the robot defaults after being instantiated. In this state, the robot controller is waiting for a configuration and connection to be established and can not do much in itself.
- **Configured:** Once a configuration is loaded the controller transitions into this state. A configuration file contains the necessary information to handle an inchworm robot and is what separates different individual robots of this type. This will be further explored in Subsection 6.7.3.
- **Connected:** State the robot goes into once serial communication is established. This connection may be realized over USB using a cable or unthedered via the Xbee wireless modules. This is the default state for operational robots, where they are waiting to receive work packages while allowing the user to easily control them.

- **Executing Work Package:** Once the robot starts executing a work package it transitions into this mode, where it sequentially breaks down the tasks of the work package into executable pieces called actions and executes them. Additionally depending on the allowUserControl property of the work package, the robot's user controls will be removed from the UI.
- **E-Stop:** Emergency stop mode in which all the motors are unpowered and can only be released with user input. As a safety feature that prevents damage to the robot, this mode can be automatically triggered as a function of motor currents.

6.7.2 Abstract state

All these states inherit from the abstract state interface. All derived states must provide the implementation of all members defined within the interface, allowing the use of a standardized interaction across all states. [91]

The following five methods are thus implemented in all states:

- **Enter state:** Called when the state is entered. Allows to precompute the necessary elements for the state to work. For instance, when entering the E-stop state all the motors within the servos are unpowered and the LEDs are set to red.
- **Update state:** Represents the update cycle of the robot and the main element of each state. Runs once every frame.
- **Check switch state :** Checks if the requirements that force the robot to change state are met. For instance, continuing with the E-stop requirements, if the emergency stop button is released then this method will return the robot to the connected state. This function is called after the Update state.
- **Exit state:** Similar to the Enter state but called on exit, before the new state is entered. Can be used to for example clean parameters in memory or print the performance of the robot through the state execution.
- **Draw UI:** Called from the update cycle of the UI window when the robot UI is being displayed. This allows to show different robot interfaces depending on the current state. For instance, before the robot is configured and connected the UI is focused on loading a configuration and establishing a connection. Once connected it opens up high and low-level control of the robot

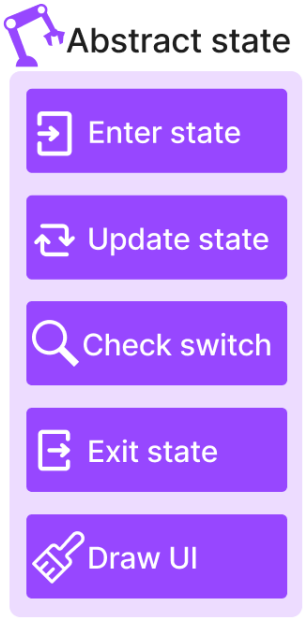


Figure 6.10: Methods of the AbstractState

We will now analyze the steps required to get the robot controller to the connected state. Firstly a deeper explanation on what a configuration is and how it can be loaded is presented followed up by establishing a serial connection to the robot.

6.7.3 Loading a configuration

The robot controller is a template to connect any robot of a particular type. For instance, the ILLAController is made to connect to ILLA robots. However, each ILLA robot can be a bit different. For this reason, a series of parameters that define the particular build of the robot are represented in an ILLAConfiguration class and serialized into a JSON file that is saved and later loaded into different robot coordinators.

This configuration is composed of two differential joint configuration objects and a single joint configuration object together with some additional parameters like the battery capacity of the robot. The differential or single joint configurations contain joint and servo configuration objects, each composed of the rotation limits of the joints (robot space) and the servos (servo space) as well as the servo IDs, the gearing ratio, the serial communication baud rates, whether the servo is mounted in its inverted position, ID of the N20 gripper screwdriver motor, etc. A graphical overview of the robot configuration classes and the information stored is provided in Figure 6.11.

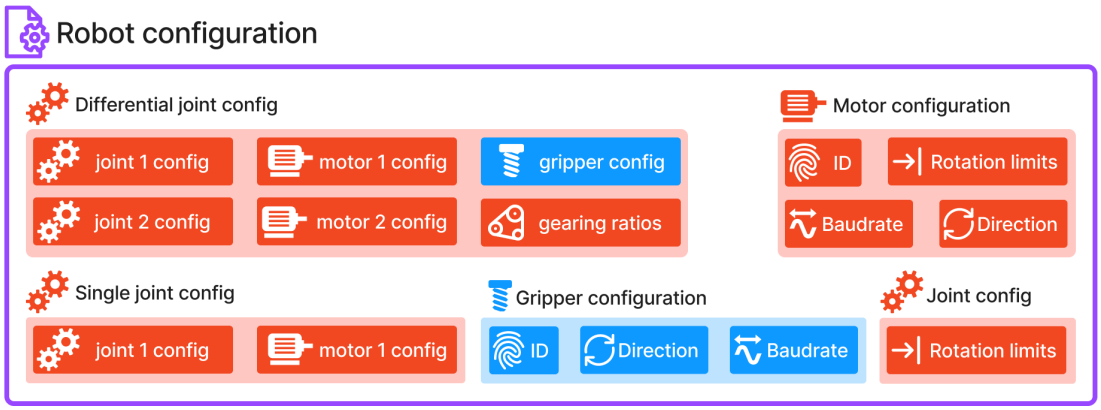


Figure 6.11: Diagram showing the elements of the robot configuration and their organizational hierarchy.

Most settings can be adjusted through the UI after loading or creating a configuration. and configurations can be saved or overwritten, facilitating the modification of parameters via the UI. Once a configuration is loaded, the robot transitions to the configured state, enabling a connection to be established.

6.7.4 Connecting to a robot

When connecting to an ILLA assembler over a serial channel, users choose the serial port and baud rate for the connection, although default settings can be pre-configured in the ILLA configuration. When a connection is attempted an LSS channel object from the .net Lynxmotion library is created. This library, developed by the servo manufacturer for sending and receiving commands to servos, is not fully developed or published, but an early version was obtained and required some modifications for effective functioning.

Recall from Subsection 5.6.1 that serial communication allows only one device to transmit at a time on each line. This means that only one servo or N20 actuator can transmit at once. However, in the provided library, it's possible to query multiple devices simultaneously, leading to collisions in their responses. This often happens in the default implementation of a channel scan for servo detection. Consequently, the approach to reading from the serial line was revised. In the original library, reading would skip partial or unreadable messages. In this updated version, the user specifies the expected number of packages and a timeout. The code then reads from the serial port until the expected number of packages is found or the timeout expires. This is advantageous as it allows querying from one device and waiting for its response before sequentially querying another.

When connecting to a robot, the serial line is opened and the servos are scanned and distributed. Scanning involves detecting devices on the serial line by sending commands to each ID where a servo is expected, as defined in the configuration file. If a reply is received a servo object for communication and telemetry data management is initialized and later distributed, meaning it is assigned to its corresponding joint Additionally, there's an option to replace missing servos with virtual ones. These virtual servos simulate the behaviour of physical servo motors by adding virtual replies to the serial line reception, enabling holistic software testing without a real robot or actual servos. After completing this process, the robot transitions to the connected state in which, the user can control it through the UI, and the controller can participate in work package auctions and execute tasks.

6.7.5 Executing a work package

Once the robot is configured and in the connected state, it can bid on work packages. A diagram representation of this bidding process was presented in Figure 6.8 at the beginning of the robot controller section. Essentially the controller downloads a work package that includes the tasks to execute, estimates the cost in terms of some KPIs like energy or time, creates a bid proposal, and submits it to the build planner. If the build planner assigns the work package to the robot, it will enter the 'executing work package' state.

In this state, the tasks are broken down into actions that are queued and sequentially set up and ex-

ecuted till all the tasks of the work package are completed. This process is shown in Figure 6.12. As can be seen in every update frame, it is first checked if the action currently being worked on is finished or not. If the action is not finished an additional frame of this action is run. If the action is finished, the next action in the queue is setup and if the action queue is empty, the next task to be executed is broken down into more actions. Once no action or task remains the work package is finished.

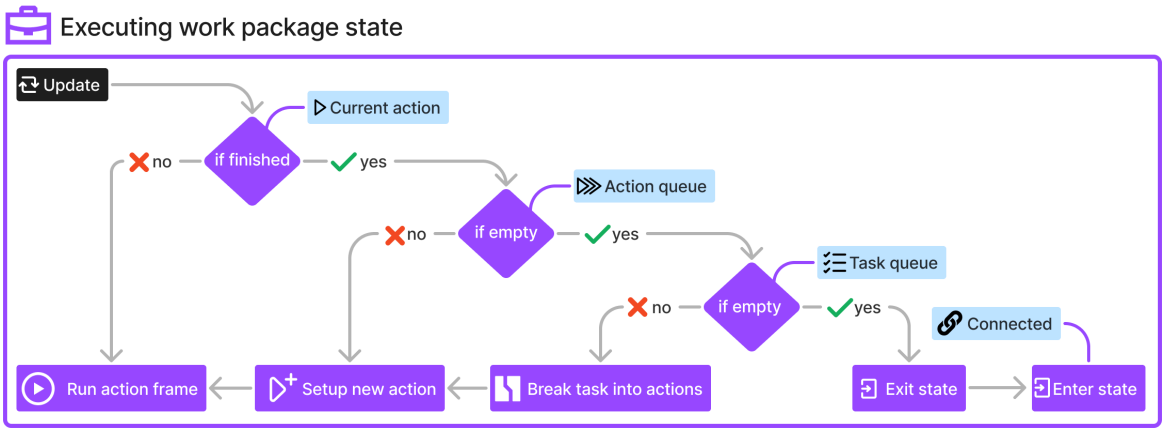


Figure 6.12: Logic flow diagram of the executing work package state showing.

With this new insight and understanding of the software's high-level architecture, the interplay of its various components and more specifically how work packages are executed let's analyze an example of how these actions are created from high-level tasks.
As explained in the build planner the most common and complete task utilized throughout an assembly process is the pick-up and insert work package. This high-level task consists of getting a lattice and placing it in an insertion location. Let's analyze how the robot how the robot manages and executes such a complicated task.
Once in the executing work package, the robot will first have no current action or action queue elements, and thus the first task is broken down into actions.
The robot controllers implement an overloaded BreakIntoActions method that returns a list of Actions with the input argument being the specific task class to break into actions, in this case, a PickUpAndInsertTask.
In this case, remember that the high-level goal is to pick up some lattice and insert it elsewhere. For this reason, we must find a path to the first lattice, move there, pick the lattice up, put it in a storage backpack, find a path to the inserting location, move there, take the lattice from storage, and place it. The diagram in Figure 6.13 presents an overview of the process.
The first step is to build a navigation graph; A navigation graph is a simplified representation of the context in the form of a graph that can be used for pathfinding. Then a path to the pickup lattice is solved for. These two processes are further explained in the navigation module section in Subsection 6.7.6.

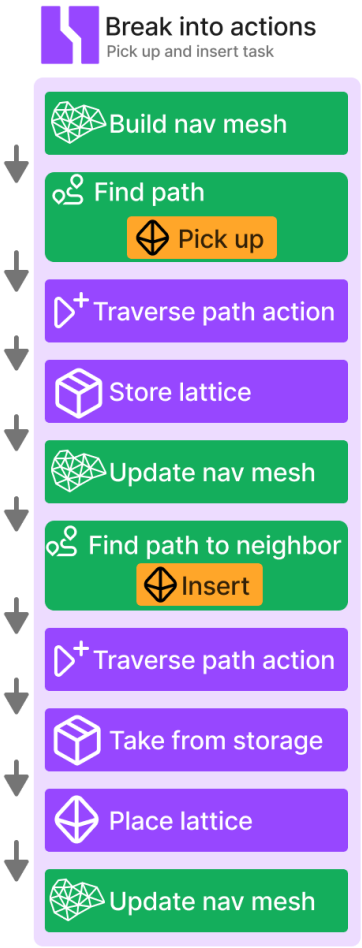


Figure 6.13: Steps for breaking a pickup task into actions.

6.7.6 Robot navigation

The robot is designed to deal with the spatial manipulation of digital materials, for this reason, its controller is deeply involved with navigation, pathfinding, and spatial computing. The robot controller has low-level access to the context and thus the geometrical rep-

resentation of the target and current structure as well as possible obstacles. For efficient navigation and understanding the spatial traversal of the structure with the robot's locomotion, this information present in the context, is processed into an easier-to-work-with morphology, a navigation graph. A graph is an abstract data type or structure consisting of a finite set of nodes that are related or connected with edges as was introduced in Section 2.5. As can be seen in Figure 6.14 the navigation graph contains all the lattice nodes the robot can reach interconnected or related by the possible movements that the robot can perform (in green). In this case, the lattice nodes become the vertices or nodes of the graph and the movements that relate them, the edges. The discrete nature of the digital material is exploited and used as representation resolution within the graph.

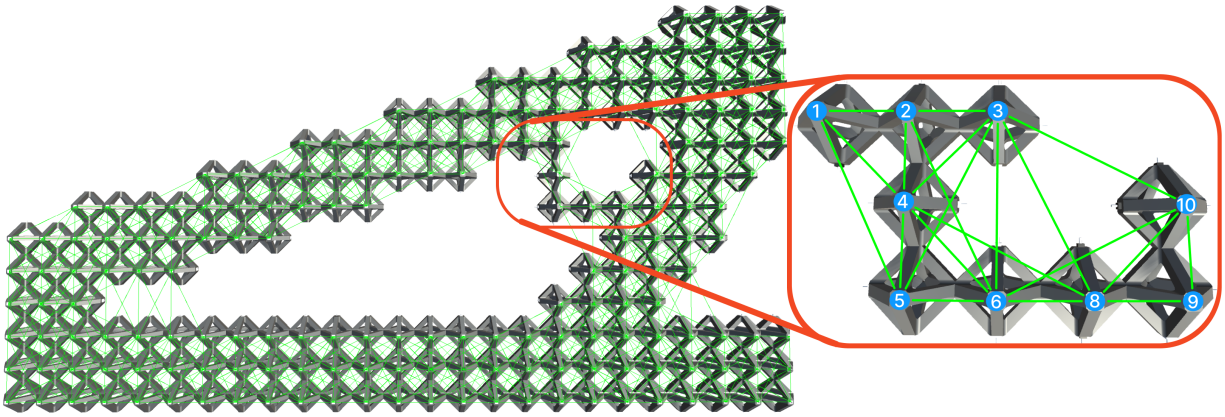


Figure 6.14: Example of a navigation graph showing the relations between nodes with the green edges.

This section discusses what that graph consists of, how it is built, and how an optimal path can be found. Before diving into that discussion it is important to remark that the navigation graph is built using inverse kinematics and it is pre-checked for collisions.

As discussed in the literature chapter, there are many ways to represent such a graph within a computer. The most common ways are Adjacency Matrices and Adjacency Lists. Each one presents different advantages and disadvantages. Mainly adjacency lists are faster to build as adding elements to lists is faster than resizing matrices but it is slower to find the cost of edges since they can't be simply accessed by index and instead, have to be searched in a list. Generally, adjacency lists are preferred for representing sparse graphs like this one since the matrix representation would explode in size. There are methods to mitigate this like utilizing a compressed row/ column representation or as would be more useful in this case compressed diagonal storage [92] since by ordering the indices of the navigation graph we can distribute most elements towards the diagonal.
For this reason, a modified adjacency list is utilized. The slowest aspect of assembly is not the computation but the execution of the work in the physical realm. Given that the time complexity of the adjacency list graph increases linearly with the size of the graph, it's difficult to envision a scenario where the computational time would even approach that of the movement execution.

The particular implementation of this graph structure is a bit more complicated than a simple adjacency list introducing some quality-of-life features enabled by the extensive memory availability of modern computational platforms, that allow scaling past a million nodes.
Graph nodes are represented by ILLAGraphNode objects, a custom class that stores the physical node it represents, two Dictionaries for the outgoing and incoming edges that take the place of the adjacency lists as well as some quality of life parameters like user-readable node IDs. Outgoing and incoming edges are stored facilitating efficient graph navigation in both forward and reverse directions. Each dictionary entry pairs ILLAGraphNode objects, representing adjacent nodes, with corresponding movement objects that include the necessary poses for 5DOF movement execution as explained in Subsection 6.7.11. Notably, the values in the incoming neighbour dictionary exhibit an added layer of complexity. The movement a robot undertakes to reach a node is influenced not just by the previous node itself, but also by the robot's

pose at the preceding node which is determined by the second previous node, that is, the incoming node of this preceding node. Although not required this is used to compute the most optimal path even regarding the initial pose.

This is explained visually in Figure 6.15 where the incoming and outgoing dictionaries of node G are examined. As can be seen, the incoming dictionary of node G has as key G', that is the previous node from which the edge is coming and as a value a struct consisting of the second previous node or G" and the movement used to move the robot arm from G" to G. In reality, up to four movements per G" may be present as the kinematics of the robot allow for it, similarly, more nodes may exist at both levels of depth. It can also be seen that when traversing the graph forwards, this two-level deep data storage is not needed thus a single-level dictionary is used.

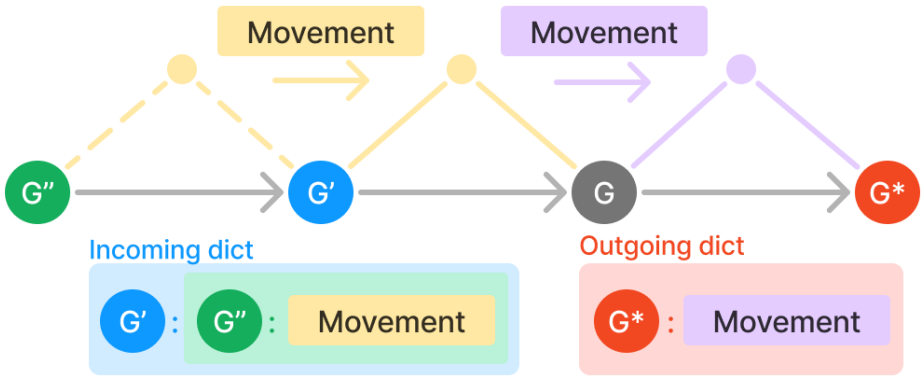


Figure 6.15: Diagram showing the outgoing and two levels of incoming neighbours of node G.

This is accompanied by abstractions to add or remove edges and neighbours in the graph without having to deal with the complexity of modifying these dictionaries manually in the form of an ILLAGraph class. It contains the ILLAGraphNodes, establishes the interface with the robot controller, and adds some additional abstraction.

So far, the data structure and organization of the graph have been explained, but the building and utilization of this graph remain a mystery. In the next sections, these elements are explained.

6.7.7 Mapping: building the navigation graph

The building of the navigation graph involves finding all the nodes, their relations or edges and populating a graph object with them.

There are two possibilities for when to develop the navigation graph. On one hand, the map can be discovered during pathfinding. In this way, a partial heuristic-informed graph is built where only the preferentially used areas of the map are explored. Alternatively, a global map spanning the complete structure can be built and utilized for multiple path-finding exercises trading memory for computational efficiency. A global navigation graph allows the reuse of a very significant amount of computation as graph building involves solving inverse kinematics problems and checking for collisions, both computationally demanding tasks. Secondly, as the assembly advances, the graph can be updated progressively rather than in a single shot meaning that the graph could potentially be built from scratch only once to map the small initial structure and updated from there every time a robot modifies it, though this is yet to be implemented. Additionally, to prevent the program from freezing during the graph building, the mapping load is split throughout multiple frames utilizing a co-routine.

Figure 6.16 describes the building of the navigation graph. With the robot in its initial position, we find all the collision-free movements it can perform, meaning all the nodes it can reach from its current base. This high-level computation is enabled by the extensive robot abstractions that make the foundations on which the robot controller is built. They are further explained in Subsection 6.7.11. The possible movements are obtained by finding all the reachable nodes, that is the nodes inside a sphere with a

certain hard-coded radius that represents the maximum reach of the robot, finding the inverse kinematics solution to grab each of those nodes, enforcing the rotation limits of the robot and removing all collisions from the I.K solution set. These movements are then added to the navigation graph. If the reached node doesn't already exist in the graph an ILLAGraphNode object is created. An edge between the reached or target and the base node is added, where the outgoing neighbour dictionary of the base node and the incoming dictionary of the target node are populated. The process is now recursively repeated with the newly added node as a base. In reality and to allow the use of coroutines, a non-recursive or non-deepening implementation was made that instead returns the next nodes to visit and adds them to a node to visit stack. Additionally, each node is only visited once.

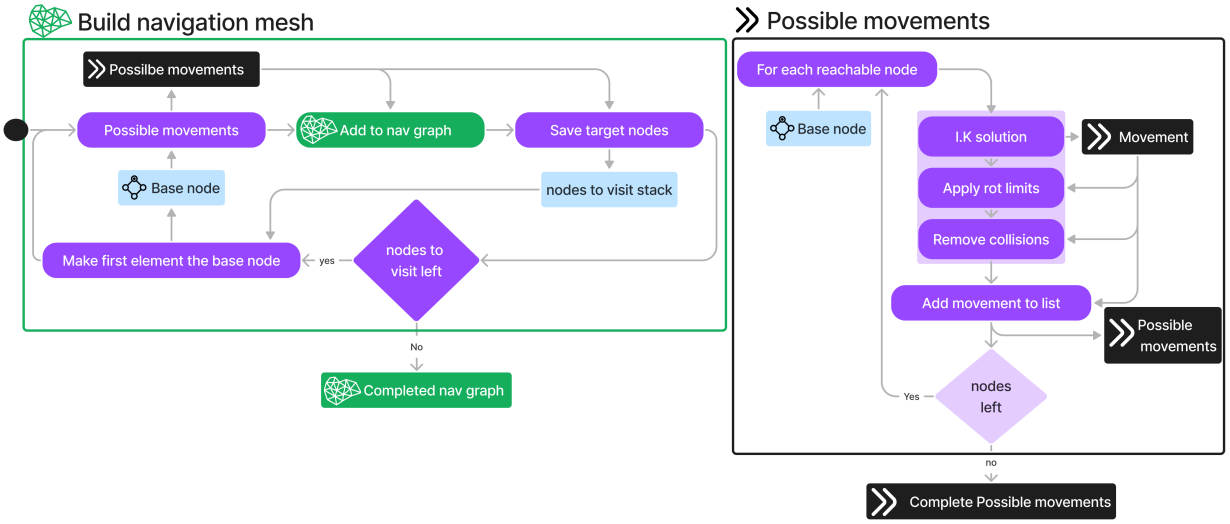


Figure 6.16: Diagrams showcasing the building of the navigation mesh on the left and the process of finding all possible movements that can be performed from a base node as required for the first one on the right.

Mapping performance

The mapping process is ideally performed only once on the small initial structure and from there, it is progressively modified as lattices are placed or removed. However at the current moment, this is not supported and the map is rebuilt from scratch every time. This means the performance of the mapping process was never an important priority and minimal time was spent optimizing it. Nevertheless, it is presented next.

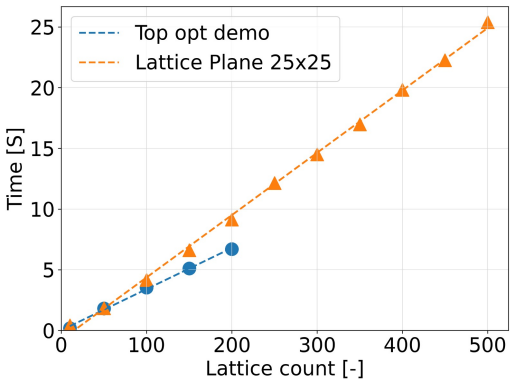


Figure 6.17: Performance of the mapping process for varying geometries and lattice counts.

The performance of the mapping was tested experimentally with two different structures. A flat plane with 625 lattices in a 25x25 grid and the 215 lattice topology-optimized bridge-like structure shown in the navigation graph example in Figure 6.14. The mapping compute time was recorded against lattice count in increments of 50 as shown in Figure 6.17. Both cases showed a linear relation between compute time and lattice count with an R^2 of over 0.99, confirming the $O(N)$ nature of the algorithm.
The difference in their slopes is explained by the fact that the square plane has a higher degree of connectivity in the graph. At the beginning of the mapping, this is inverted as the assembly order of the plane is line by line.

The main computational time cost is collision detection. As will be explained in the section about collision detection performance in Subsection 6.7.10, it is easily optimizable by caching the Denavit Hartemberg transformation matrices.

The navigation graph is a data structure that allows for advanced manipulation of spatial data, opening the door to employ powerful graph search algorithms on the structure. It's not a goal on its own, but a medium to satisfy requirements like the ability to find a path to a particular node. This ability is explained in the next section.

6.7.8 Pathfinding

Pathfinding is the process of finding the most efficient route between two points, in this case, two lattice nodes; the initial base of the robot and a target node. For this, graph search algorithms are applied to a navigation graph. In particular, a flexible implementation that allows switching algorithms, heuristics and the use of states is developed. As detailed in the literature (see Subsection 2.5.2), different pathfinding algorithms like breadth-first search, depth-first search, Dijkstra's, or A*, differ in the sequence of node exploration. For instance, depth-first search prioritizes visiting nodes sequentially in long directional branches while breath-first does the opposite expanding in a smooth sphere. These algorithms share a common foundation, which is customized by modifying cost functions and heuristics to establish a priority queue in the search.

This general template consists of three iterative steps shown in Figure 6.18. Initially, the node with the lowest cost from the unvisited queue is selected, starting with the initial node. It is then checked whether this node is the target in which case the path would be complete. Otherwise, the node is explored by calculating the cost to its neighbours, which are then sorted into the unvisited queue based on cost, and the process repeats. Provided the heuristics utilized are admissible, meaning they don't overestimate the cost, Bellman´s principle of optimality guarantees that the optimal path will be found.

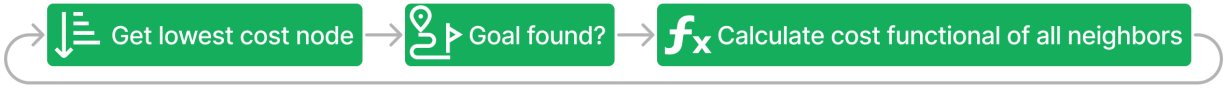


Figure 6.18: Diagram showing the three high-level iterative steps utilized for path finding; Getting the lowest cost node, checking if it is the goal state and calculating the cost of all its neighbours.

The actual implementation shown in Figure 6.19 is more intricate. First of all, there are different levels of fidelity possible for pathfinding. While the graph represents the nodes and their relations, the robot is not only on a base node but has a certain pose on top of that explaining the 2 level depth dictionaries presented before. Given the relative sizes of the robot and the material system that defines the nodes, this can have a significant effect, especially in shorter paths. To better understand this let's put an example. In a common pathfinding scenario such as getting directions in Google Maps, the intersections are treated as nodes, and the streets in between as the edges. Here, our pose at a particular node is of no interest. It doesn't matter whether your right or left leg is ahead at a particular intersection. The time to travel between two nodes is the length of the street divided by our speed and the scale is large enough to not worry about such details. However, in the case of this robot, the resolution of the navigation graph means the nodes represent the bricks in the pavement of the street instead of the intersections and nodes are always only one step apart from each other. The cost between two neighbouring nodes is estimated from the largest rotation any actuator must perform, which strongly depends on the initial pose before moving. This resolution is required as the nature of the task necessitates placing particular building blocks or continuing with the example bricks in the street.

This pose-dependent pathfinding becomes computationally expensive and less relevant on long trajectories. For this reason, a parameter that controls to what extent the pose is utilized was added as explained later. Additionally, a dual-scale graph for both broad and fine movements could be built or the structure's surface could be treated as a continuous domain on which continuous pathfinding is applied as explained in the literature in Subsection 2.5.1.

Pathfinding is performed not on the ILLAGraphNodes but on ILLAState objects, which are temporarily created for each pathfinding process. This serves a dual purpose. Firstly, a navigation graph may be shared across multiple robot controllers of the same type for computational efficiency, requiring a container other than the graph node for pathfinding parameter storage.

But it is also an enabler of this intricate pathfinding that considers the pose and the node simultaneously. A diagram view of an ILLA States is provided in the PathFinding diagram. They consist on a a reference to the current node as well as the previous state and movement performed to arrive at the node together with the required cost parameters as well as an ID that will be useful shortly.

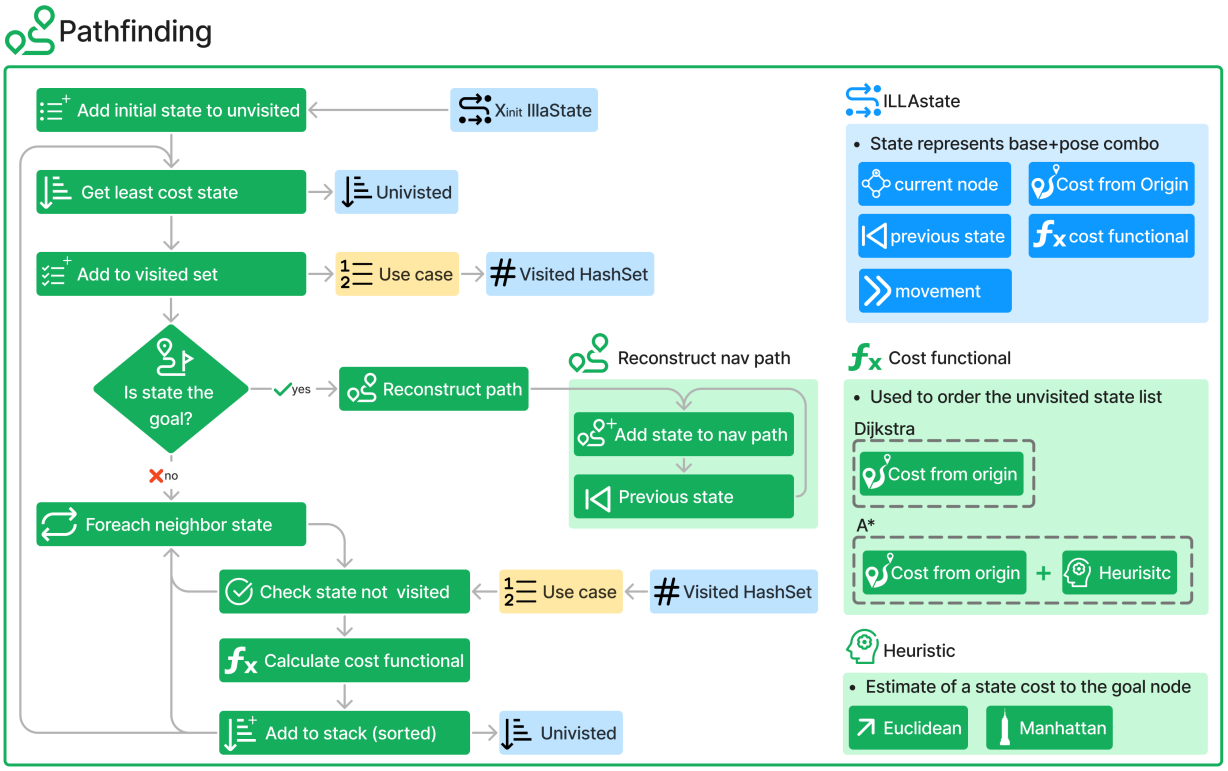


Figure 6.19: Diagram showing the pathfinding process including the different elements or abstractions utilized.

The first step is to create an ILLAState from the initial node and add it to the unvisited queue. From there the iterative process begins.
After getting the state with the smallest cost functional, it is added to the visited set. This is done to prevent visiting the same state over and over. As can be seen, the use case regulates the reading and writing of the visited hash set. This is because to control the extent to which poses and states are utilized, the interface to the visited set is modified, by storing and comparing different strings. For example, when the complete space is to be examined, including all possible node and pose combinations, the stateIDs are utilized. These are unique to each state meaning there is no constraint to exploring the same node under different poses.

Similarly, instead on utilizing the stateID, the ID of the current node of the state can be employed. Then after a node has been visited in a particular state it will not be added to the HashSet again. This does not prevent visiting the node multiple times as multiple states can be added to the unvisited queue before the node is even visited once. However it presents an in-between mode that allows using states in the forefront of the state-space exploration, but once a node has been visited and the front of the exploration moves ahead, corrections to the node disregard states. Lastly, an additional check can be added to verify that no states of nodes already in the unvisited queue are added.

The first alternative can be useful in shorter paths but is incredibly expensive in comparison with the other two. The middle on presents a good balance while the last one is fast but not necessarily efficient to compute as the unvisited queue is checked often.

Coming back to the pathfinding process, it iteratively continues until the goal node is found, at which point the path is reconstructed. There is no list of nodes that represent a path but rather information em-

bedded in the ILLAStates. Fortunately, this process is rather simple as the ILLAStates store the previous state. The states are traversed backwards from the goal state following the chain of previous states and adding them to a navigation path until the initial one is found at which point the path is inverted back to its original direction. The navigation path is another object with some quality-of-life utilities for manipulation of paths.

The performance of the pathfinding implementation was investigated using the Unity CPU profiler. The main optimization points were utilizing a HashSet for the visited states instead of a list as HashSets have constant time $O(1)$ containment operations compared to $O(n)$ in lists. This is at the cost of memory, but tests with up to a couple thousand lattices showed memory was not a pressing issue. Additionally, another small yet very significant optimization was changing the priority queue to work with the negative of the cost functional, or more generally have it ordered in such a way that the least costly element is at the end and not at the beginning. This queue is represented by a list, which in `C#` are Dynamic Arrays. Removing the first element results in shifting all the other ones one position to the left resulting in a time complexity of $O(n)$, however, removing the last element necessitates no such restructuring leading to a time complexity of $O(1)$. The importance of this seemingly small element can not be understated as it represented the vast majority of the computational time before it was improved. Flexibility was established as one of the driving design goals. In the building of the navigation graph, it was explained that the graph is created using inverse kinematics and checked for collisions, this is done within the discovery of the possible movements.

The next two sections dive into the two main elements that enable this movement discovery process; the kinematics and collision checking.

6.7.9 Robot kinematics

Kinematics is the study of the motion of objects, without considering the forces causing them. It is of great interest in the field of robotics, and in the context of this thesis, it is used to establish the relation between the position and orientation of the gripper or tool and the rotation of the joints. For this purpose a kinematics toolbox was developed within the robot controller with two distinct goals or requirements; calculating the positional forward and inverse kinematics. Forward kinematics involves calculating the position and orientation of the robot´s end effector based on it's joint parameters and inverse kinematics determines the necessary joint angles to achieve the desired end effector position and orientation.

These processes, in particular the inverse kinematics, present unique challenges as the relation between the joint and gripper spaces is nonlinear.

First, before diving into the details of the toolbox´s computation, the kinematic representation of the robot is discussed. The robot´s kinematics, meaning the geometrical and hierarchical relations between joints are represented as a set of Denavit-Hartenberg or D-H parameters. Each joint´s D-H parameters are a set of four values used to describe the relative positions and orientations of adjacent parts in the robotic assembler. These parameters, α, θ, d and a are described below:

- **Alpha** $[\alpha]$ represents the twist of the link. It is the angle between Z_{i-1} and Z_i around the X_i axis. It is a constant.
- **Theta** $[\theta]$ is the angle a joint is rotated. Formally, it is the rotational angle about the previous z-axis Z_{i-1} between the previous link x-axis X_{i-1} and the current link X_i. It is the variable in revolut joints like the ones that make this robot.
- **offset** $[d]$ is the distance along the previous rotational axis to new joint, or equivalently, the offset along the Z_{i-1} axis to the origin of the current coordinate system. It is a constant for revolute joints like the ones found in this robot but variable for prismatic joints.
- **link length** $[a]$ is the distance between the axes of two consecutive links, meaning the length of the arm in between. It is the distance between the z-axis of the previous joint Z_{i-1} and the z-axis of the current link Z_i along the previous joint x-axis X_{i-1}. This is a constant.

These parameters can be used to create an affine transformation matrix that represents the transformation to one link from the previous one in the chain. Realize that only the parameter theta θ is unknown as the

rest are fixed robot constraints. This process presented in Equation (6.1) is explained in [93].

$$T^i_{i-1} = Rot_{z,\theta_i} Trans_{z,di} Trans_{x,ai} Rot_{x,ai}$$

$$T^i_{i-1} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & {}_{\theta_i}c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_1 & -s_1 c_{\alpha 1} & s_{\alpha 1} s_1 & a_1 c_1 \\ s_1 & c_{\alpha 1} c_1 & -s_{\alpha 1} c_1 & a_1 s_1 \\ 0 & s_{\alpha 1} & c_{\alpha 1} & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.1}$$

Forward kinematics
The process of finding the forward kinematics solution is simple. An affine transformation matrix is developed for every joint as shown before in Equation (6.1). These transformation matrices are then concatenated to find the transformation from the base to the gripper as shown below. The expanded solution is presented in Appendix A.

$$T^1_0 \cdot T^2_1 \cdot T^3_2 \cdot T^4_3 \cdot T^5_4 = T^5_0. \tag{6.2}$$

Inverse kinematics
As explained in the literature, different inverse kinematics methods exist, from iterative numerical methods to closed-form analytical solutions. Closed-form solutions result in a simple, yet hard-to-find expression with fast computations, although there is no guarantee for the existence of such a solution, especially for higher degrees of freedom and redundant kinematics robots.
To enable a fast and efficient construction of the navigation mesh a closed-form solution is sought. The general steps in its derivation are explained throughout this chapter, with the in-depth mathematical process presented in Appendix A.

Initially, the solution was found utilizing a geometric method that exploited breaking the problem into smaller chunks and solving it one by one. This involved projecting the joints and links into planes, decomposing them into vectors and solving using basic trigonometry.

For instance, the angle of the first joint could be readily determined by projecting the target point onto the horizontal plane defined with the normal being the base joint z-axis and calculating the corresponding angle with the base x-axis.
The second and third joint angles were calculated by projecting onto the plane containing the base joint z-axis and target point and applying the trigonometry for planar 2DOF arms like in [94]
The last two joint angles can be easily determined from the rotation of the target pose.

This geometrical approach was however later changed for a matrix equation resolution to better represent both the inverted and non-inverted cases in a more consistent mathematical approach.

This process consists of concatenating the transformation matrices like in the forward kinematic solution while keeping the angle of the joint θ unknown. This resulting matrix is compared against an input affine transformation matrix S composed of unknown elements. An analytical solution is then found that maps the joint angles to the input parameters of S. This way, a target pose can be represented as a base to gripper transformation matrix, substitute in the parameters of S and then find a solution of the unknown joint angles. This mathematical process is detailed in Appendix A.

This robot presents an additional level of complexity. The relation between the joint and gripper spaces must remain constant as the robot walks and the base and gripper roles are swapped, meaning when the gripper screws onto a lattice to become the new base and the base unscrews to become the gripper. This is not a common occurrence as most robot arms are asymmetrical with a defined base and a lightweight gripper [95] [96]. The main exception is in robotic arms for space applications like the Space Station Remote Manipulator System or SSRMS, composed of the Canadarm2 a second generation of the space shuttle's original Canadarm with the additional capability of base swapping for walking around the international space station [97]. More recently the ERA or European Robotic Arm or the inchworm robots from GITAI have similar walking capabilities. In this case, this was overcome by having two different Denavit-Hartemberg parameters and calculating two separate closed-forms inverse kinematics solutions

for the standard and inverted cases.

The D-H parameters for both cases are visualized with the graphical application for robot arm design and Denavit-Hartenberg parameter creation from MRPT or Mobile Robot Programming Toolkit [98] as shown in Figure 6.20.

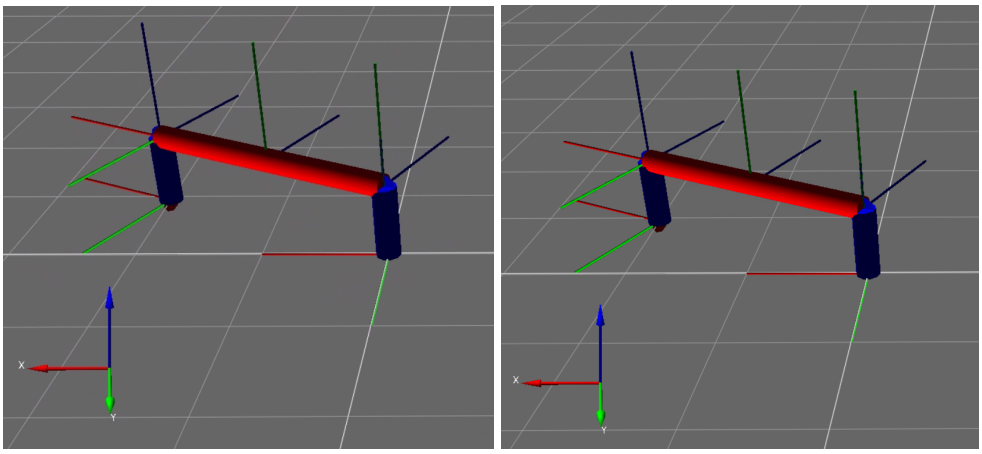


Figure 6.20: MRPT visualization of the developed inchworm robot configuration from its Denavit-Hartenberg parameters at the zero angle pose.

Limits to the input space
The input to the inverse kinematics mathematical equations is the 4x4 transformation matrix S. This matrix represents a 3D translation and 3D rotation encapsulating all 6 degrees of freedom. However, the robot is only capable of 5DOF movement, this means that it is possible to input transformation matrices that have no inverse kinematics solution as the robot is physically incapable of performing them. The robot is limited to gripper/screwdriver-pointing vectors that exist in the plane defined by the base rotational axis or z-axis and the target point. The target gripper pointing vector meaning the last joint´s z-axis can not exist outside that plane. The implementation detects and can handle inputs outside the possible space in two different manners.

Firstly, the closest valid solution can be found and moved to. This is done by rotating the input by the smallest angle possible that translates it into the plausible space. An additional possibility is to raise an error when the target pose is not in the plausible space. A tolerance was added to relax this criterion.

In general, a mix of the previous two is utilized, where solutions within a certain error range are fixed and moved to the closest valid solution but after a certain tolerance level, an error is raised. This allows for small errors to be corrected while raising errors in truly erroneous inputs.

6.7.10 Collision detection
In the mapping process, it was established that colliding poses are removed from the navigation graph during its building process. This section briefly touches on the tool developed to determine whether a robot pose collides with the lattices around it.
It works by performing one-on-one collision checks between the lattice mesh and a simplified robot geometrical model composed of simple box, spherical and capsule colliders.

Firstly and to minimize the number of collisions to be checked only the lattices within the bounds of the robot are tested. The bounds of the robot represent the smallest possible rectangular cuboid (box) in which the robot fits. This cuboid is calculated by combining all the meshes of the robot and finding the bounds. This is shown in Figure 6.21 with a green bounding box and the colliding lattices coloured in red.

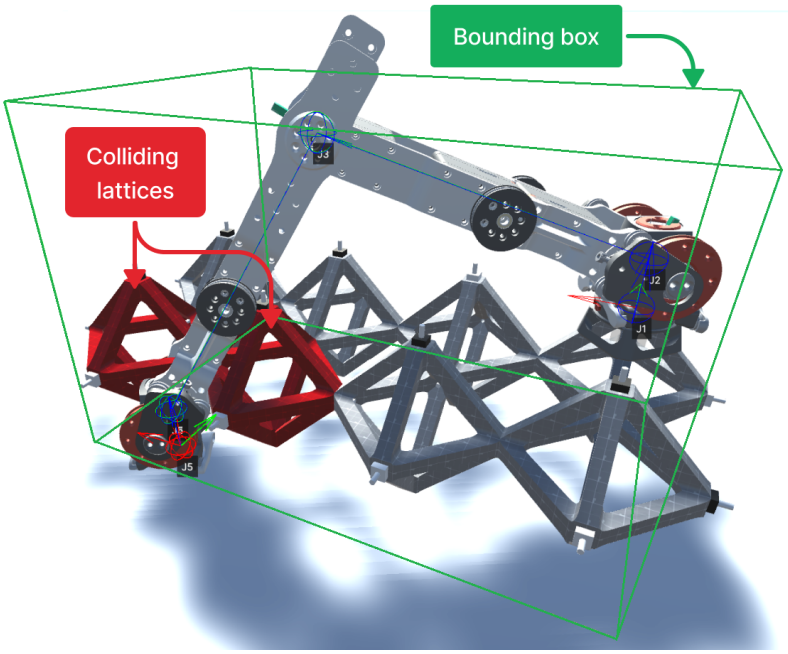


Figure 6.21: Image showing the robot bounding box in green and colliding lattices in red as displayed in the software 3D view.

Then for every lattice that exists within those bounds, it is checked whether there is a collision with each of the simplified geometrical components of the robot. This is done this way as it is simple to check collisions between a mesh and a simple box collider but not between two meshes. Additionally, even though the implementation is single-threaded, breaking the task into a larger number of simpler computations could be used to increase performance by multitasking.

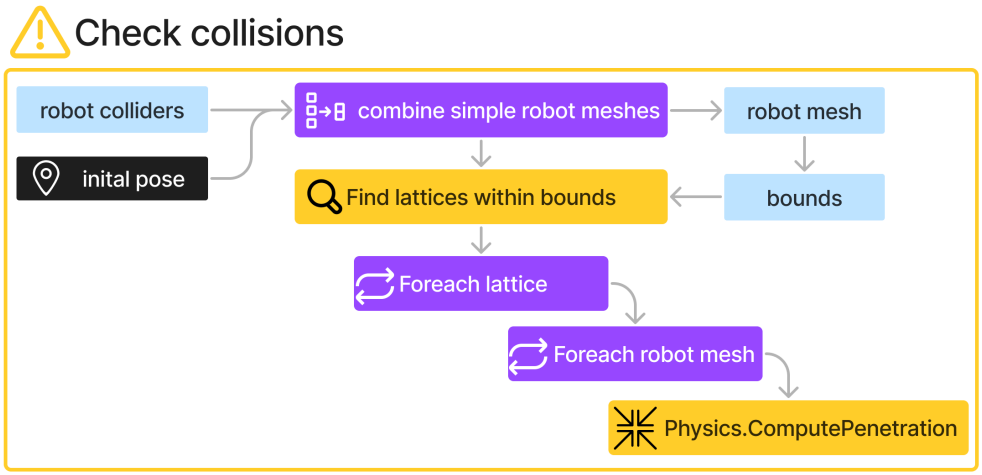


Figure 6.22: Diagram detailing the working principle of the collision detection algorithm.

Collision detection performance

Collision detection is used predominantly at two instances in time. First of all, before executing any movements and in real time, the robot pose is checked for collisions but also the navigation graph is checked to be collision-free. As highlighted in Subsection 6.7.7, collision detection represents the majority of the cost of building a map, meaning it is of great importance.

A test was performed with a batch of 500 different collision detection in different poses during a mapping frame. The 500 collision checks take a total time of 745ms or roughly 1.5ms per event. Out of this

543ms are spent in updating the joint positions, meaning performing the forward kinematics calculations and updating the joint positions and orientations. During the writing of this section, it was unravelled how inefficient this process is. The joint-to-joint transformation matrices from the Denavit Haremberg parameters are utilized for the forward kinematics. The problem arises in that when updating each joint´s position and orientation the matrices are calculated from scratch meaning they have computed a total of 10 times when a single calculation would be sufficient. Two solutions for this problem exist. On one hand, the Denavit Hartemberg could utilize a caching system that caches the value of the transformation matrices after they are requested once as long as the angles are not changed. Alternatively, the function that updates the joint positions and orientations could be corrected to only request the matrix once and utilize it for all the joints. It is likely that by the time the reader is seeing this this problem may have been resolved.

6.7.11 Abstractions

Before moving on to the the UI, a brief explanation is made on certain abstractions that were utilized throughout the chapter. Different classes that represent the physical environment were created with different data structures and methods. These are PosJ, PosX, Robot configuration and Movement as showcased in Figure 6.23. PosJ and PosX refer to Pose representations. PosJ in the joint space and PosX in the cartesian space. Essentially PosJ represents a robot pose by utilizing the 6 joint angles and PosX by using a target position and rotation, additionally, a target node can be referenced. Robot configurations are sets of Pose configurations with a base node and orientation. Lastly, Movements, represent the transition from an initial confirmation to a target PosX possible through multiple PosJs stored in a dictionary with their movement cost. This movement cost is computed from the maximum angle difference from the initial config posJ and target PosJ.

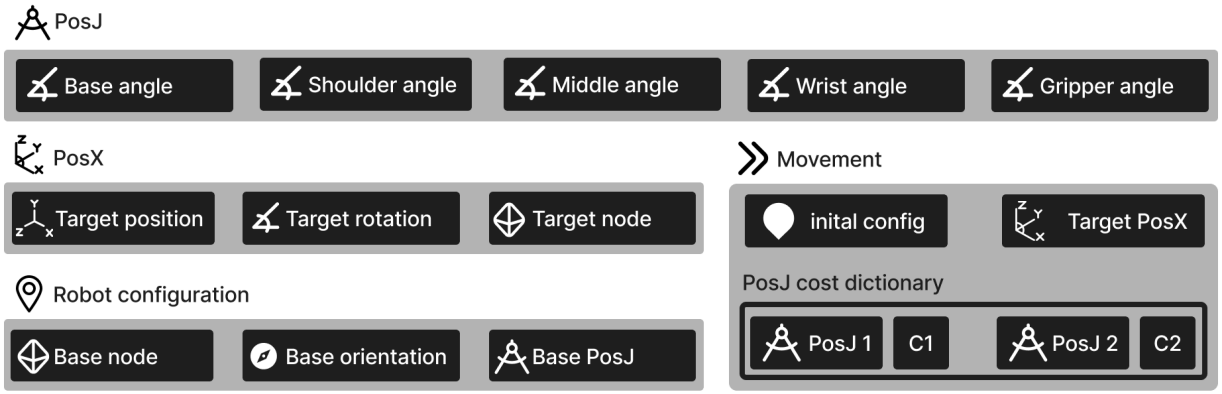


Figure 6.23: Diagram showcasing the most important abstractions the robot controller utilizes and their sub components.

6.8 User interface

This software is controlled in real-time via the built-in user interface that allows setting up and modifying the geometry definition, placing and connecting robotic assets as well as visualizing the digital twin in real-time. It enables both high-level control of the build process and some low-level control of the different assemblers like rotating a particular joint or motor. Figure 6.24 presents a general overview of the program UI highlighting the different content windows.

The centrepiece of the interface is the real-time 3D view of the build process. This Unity Game render window showcases the 3D scene with the digital twin. The RTS-like camera controls from [1] have been integrated to allow for seamless camera movements. Additionally, scripts and functions have been built to detect mouse clicks on nodes, lattices, and other objects to enable a convenient selection tool within the intuitive 3D view. Inside this window, on the top left corner, a time manager widget allows to speed up the passing of time, enabling faster virtual testing, particularly useful with virtual assemblers. This is further explained in Subsection 6.8.1.

The right-hand side of the screen is occupied by the control window. This custom-made editor window is the primary means of interaction with the program besides the 3D view. It contains different tabs and foldout menus with buttons, sliders, and fields that form a strong interaction medium between the user and the program. This is further explained in Subsection 6.8.2.

On the left-hand side, the Hierarchy and Inspector windows are shown. The Hierarchy presents a tree with every GameObject placed in the scene hierarchically arranged while the inspector showcases the elements or components from the selected GameObject, like their position or attached scripts and their public variables. These are common and handy Unity Editor windows.

Lastly, the console can be seen below the 3D view. Here text output from the program as well as errors and warnings are displayed and can be conveniently searched and filtered

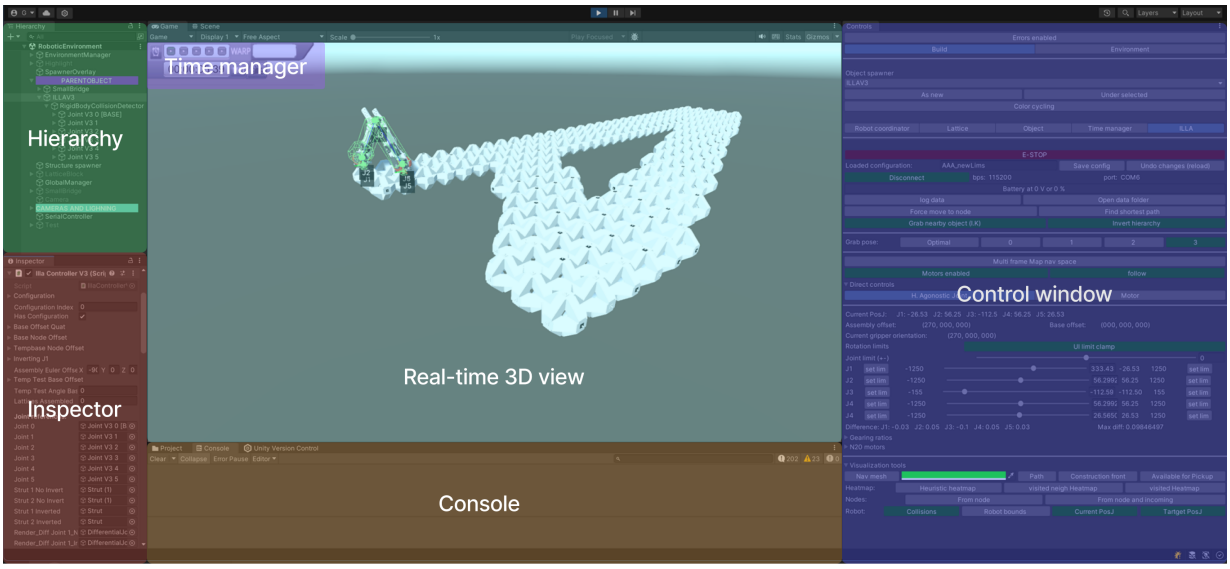


Figure 6.24: General view of the software developed running within the unity editor and showcasing its main elements or windows

6.8.1 Time warp

All the time references used throughout the project are linked to Unity's time system. For example, when computing how much virtual servo motors should move the deltaTime (UnityEngine.Time.DeltaTime) is

[1] densylkin, RTSCamera (2023), GitHub repository, available at https://github.com/densylkin/RTSCamera/tree/master, accessed 22-12-2023

utilized which provides the time interval between the previous and current frames. This time is affected by the timescale in Unity, which can be modified to speed up or slow down the passing of time.
In this case, a GUI element inspired and honoring the KSP time warp feature is added that allows to selection of a timescale between 1-5.

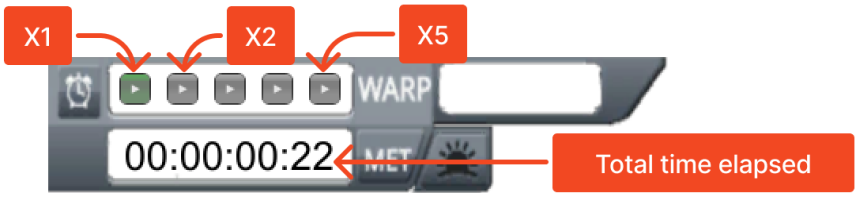


Figure 6.25: Time warp or manager widget situated on the top left corner of the real-time 3D view

6.8.2 Control window

The control window represents the main medium of interaction with the program besides the 3D interface. It contains all the buttons, sliders and fields that allow controlling the build process, robots and modifications to the context. Together with its different tabs it is showcased in Figure 6.26.

The control window is composed of a general mode selector that toggles between the currently shown build control mode and the environment mode which allows tweaking the aesthetics, lightning and location of the scene. In this build mode, the spawning section is displayed next.

The spawning area is composed of a drop-down menu with the objects that can be spawned and buttons for the different spawn modes that change where in the hierarchy the object is instantiated. When spawning, an overlay showcases where the object will be placed as you hover on different nodes and rotate it using the Numpad. After clicking on a node it will be spawned. Colour cycling allows to show different objects with different colours, like the display component colours function in CAD software like Fusion 360.

Lastly, the majority of the space in the build mode is allocated to a main tab area that allows changing the UI between different topics of interest such as the build controller, lattice modification, object modification, time manager and ILLA robot control. In this case, the ILLA controller tab is selected. All these different tabs are showcased on the left-hand side of the image and as can be seen, the robot tab contains a section with foldout menus to further expand it´s functionality while maintaining a compact footprint. These foldout menus allow opening direct controls for the joints and motors where you can directly actuate them and change their motion limits, tweak the gearing ratios, control the N20 motors in the screwdrivers and toggle a battery of visualization tools for collisions, navigation, robot poses etc.

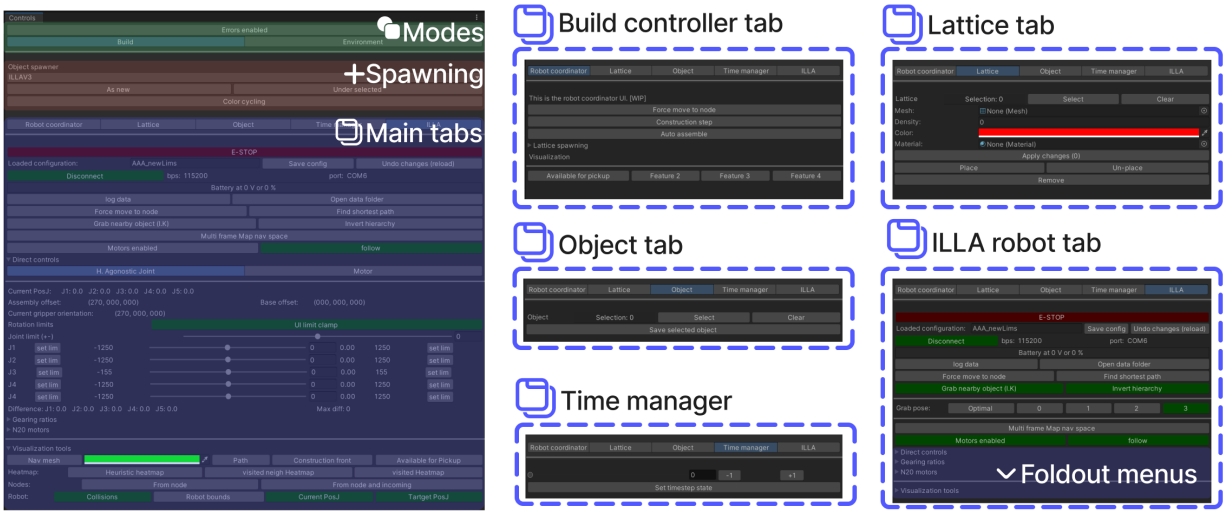


Figure 6.26: Structure and main tabs of the control window

The environment mode was added to exploit the fact that Unity is one of the platforms with the most advanced and powerful real-time rendering capabilities. For this reason, an environment manager system was created that allows modifying the environment the assembly process takes place in, its lightning and more visual aspects. For example, some environments that have been added are simple setups with different solid colours and lightning or more complex scenes like a photo booth that offers dimensional and colour references or even a lunar environment.

Lastly, a data logging add-on was integrated that allows to record certain parameters.

6.8.3 Data logging system

To deepen our knowledge of the performance and limitations of the robot and software platform, a data logging or recording module was developed and implemented within the Unity environment. This logging module allows to record values of the robot over time easily. A general overview is provided next.

An ILLALogger object is first created containing the CSV file name and the logging rate, meaning the frames waited between logging events. We can then add parameters to be logged using the LoggerObject.AddLoggable() method by providing a header or title for the series and the reference to a function that returns the parameter to log in string format.

For instance, if you want to monitor the variable called supplyVoltage that represents the battery or power supply voltage you would use the following code to create a loggerObject if one doesn't already exist and log the voltage under the header title "Voltage [V]" and save it in a CSV file named CSVFileName.

```
1 loggerObject = new ILLAlogger(this, CSVFileName, loggingFrequency);
2 loggerObject.AddLoggable("Voltage [V]", () => supplyVoltage.ToString());
```

This allows us to easily monitor parameters and record them for later analysis. Additionally, with some minor modifications, this data could be added to open-source monitoring platforms like Prometheus for real-time display in Graphana, alerts etc.

6.8.4 Scalability

On a side note, it's worth noting that the computational requirements that determine the balance between centralized, and distributed and what level of optimality is sought depend more on the size of the lattices than their quantity. The volume and power available per lattice mark the difference. When building on the scale of proteins there is barely space for a few transistors per building block and each movement takes little energy, however, even if assembling the same number of elements but at much larger scales, you can much more easily afford the computation since it becomes a fraction of the power needed to move the robot and the volume of the computational device becomes small compared to the volume of the structure. For this reason, it is most important that the computational platform can be parallelized and scales linearly or better with increasing number of lattices.

7

Autonomous construction demonstration

This chapter showcases how the elements elucidated in the previous sections combine to form a coherent system by performing an autonomous construction job demonstration. In this case, three different demos have been ideated, each highlighting different capabilities.

Before the autonomous building can be performed a series of steps need to be taken to prepare the hardware and software platforms:

- **Prepare the structure definition:** First of all, a structure definition for the build job is required. This structure definition shall include the already existing structure in the form of placed lattices and the desired build geometry as the target structure definition. This can be done by spawning a complex geometry, adding additional elements and using the time manager feature that sets the state of lattices as a function of their timestep. Alternatively, the lattice states can be set by selecting lattices with the selection tool in the lattice UI section and using the place/ un-place buttons.
- **Spawn robot:** Spawn the robot in the structure in its correct position. If the robot is still not physically placed in the structure, place it after establishing a connection. Note that depending on the rotation of the robot, an offset may have to be provided. This can easily be done from the direct controls UI once the robot is configured and connected, as will become evident in the next steps.

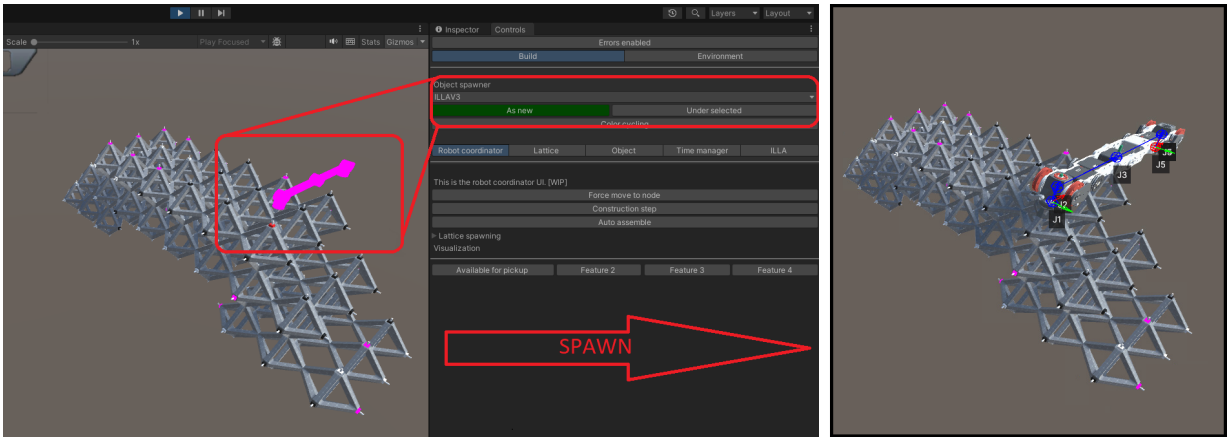


Figure 7.1: Spawning a robot in the structure. When hovering on a node, the overlay in pink is shown, after clicking on a node, the robot is spawned.

- **Turn on the robot:** Ensure a battery is mounted and connected to the mainboard via the XT60 connector.
- **Load a configuration:** Load a configuration file onto the robot. This configuration specifies robot-specific parameters such as the ID of the servos or the screwdriver modules as explained in Subsection 6.7.3. The loading of the configuration is shown on the left-hand side of Figure 7.3
- **Establish a connection with the robot:** Ensure the Xbee wireless dongle from Subsection 5.6.3 is placed on the computer and establish a wireless serial connection with the robot assembler as

showcased in the right-hand side of Figure 7.3. If the robot is not on the structure or the desired initial position, now that it is connected you can manually actuate the N20 motors from the interface and place it in the correct location. Additionally, to match the base rotation, a base rotation offset can be added in the direct controls section of the assembler interface.

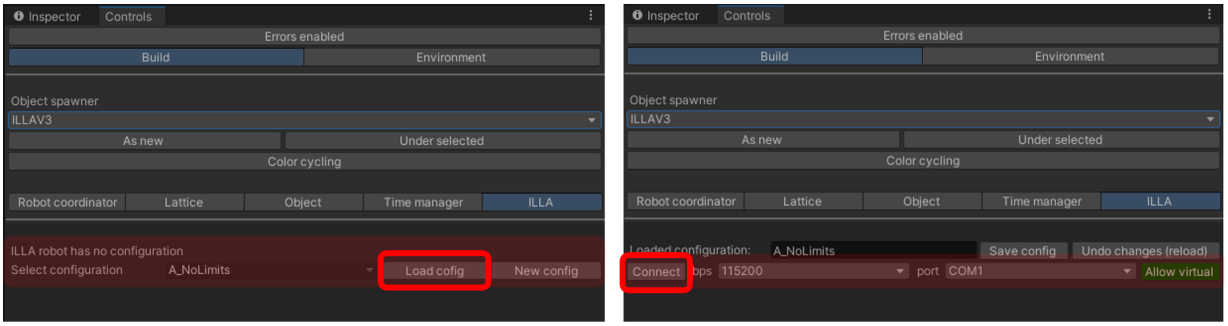


Figure 7.2: Loading a robot configuration (left) and establishing a serial communication (right).

- **Add available lattices or a continuous lattice pickup location:** If not performed during the preparation of the structure's definition, add some available lattices and unplace some existing ones reflecting the work that is to be carried out. Figure 7.3 showcases the placement of a magic lattice spawner or a continuous lattice pickup location, which is a component that ensures a free available lattice always exists in a particular location. This device is ideal for virtual testing. It is suggested to develop a physical spawning location with a sensor, such as a simple end-stop sensor that detects the presence of a building block for increased reliability.

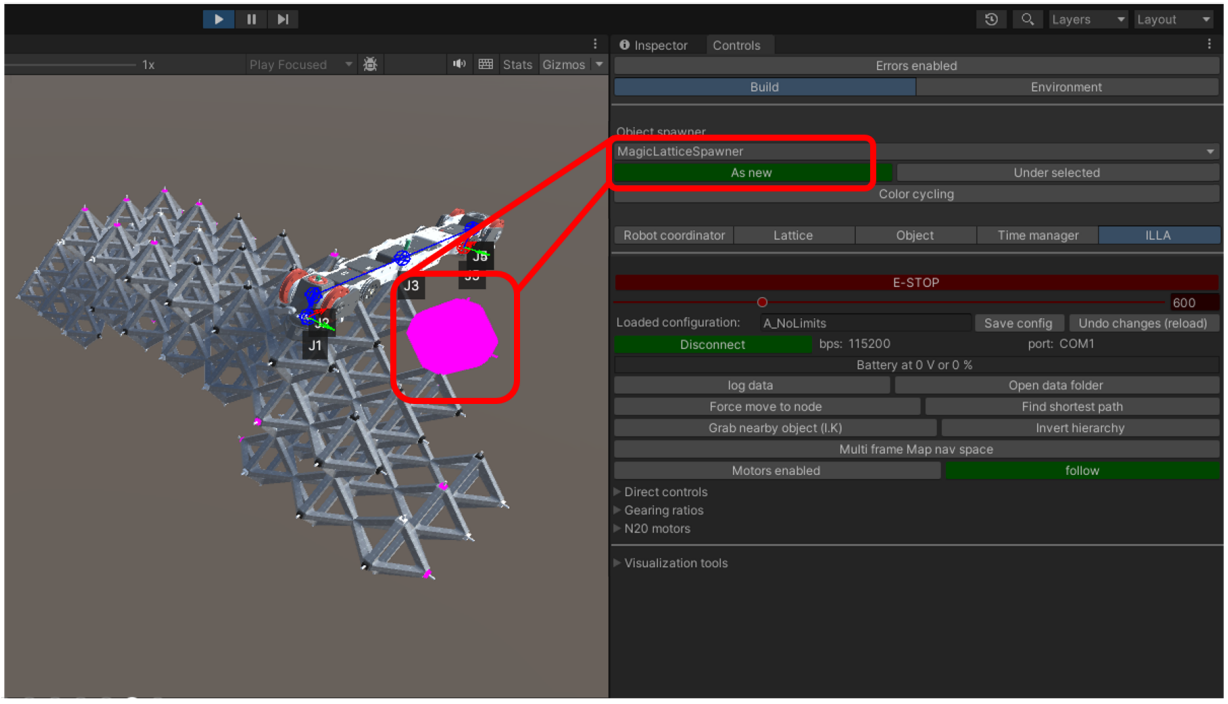


Figure 7.3: Process of placing a magic lattice spawner on the structure. It is selected on the dropdown menu and spawning is enabled. When hovering on a node it is overlayed in pink and when the node is clicked it is spawned.

- **Map navigation space:** Before starting the autonomous assembly process, the navigation space must be manually mapped once. It is then remapped autonomously after structure changes. Ensure the available lattices and insertion locations are set before this initial mapping operation. Figure 7.4 illustrates how a navigation space mapping can be performed and also how the map can be displayed on the 3D viewer.

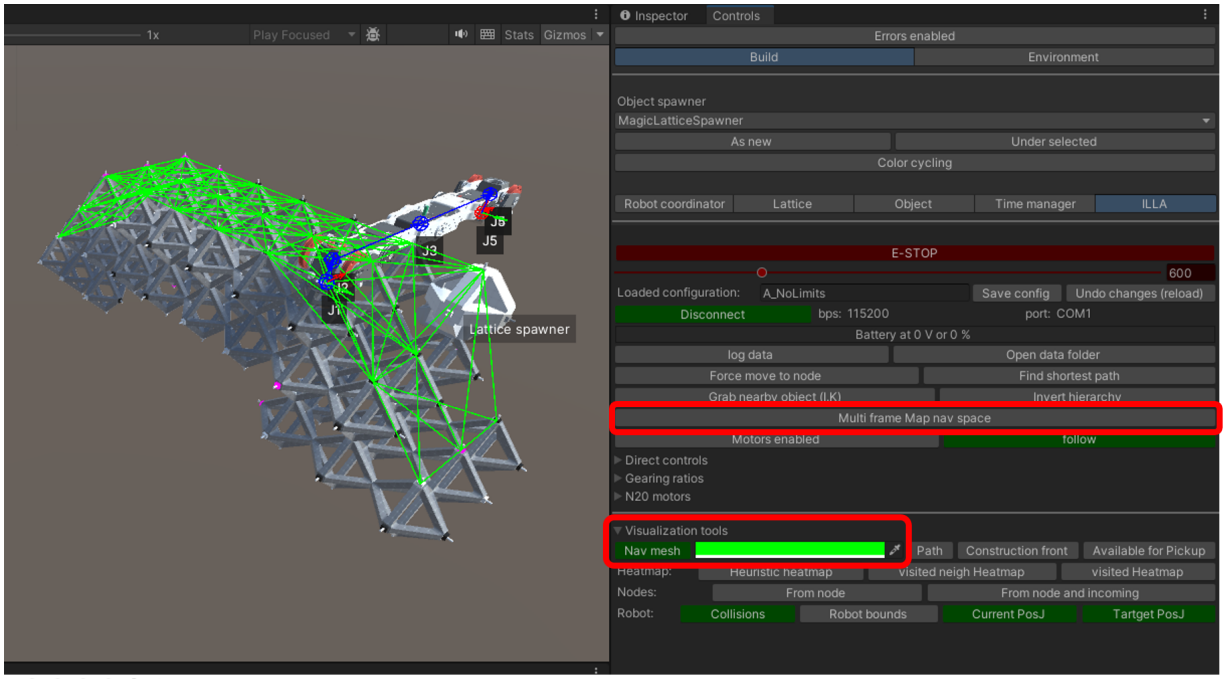


Figure 7.4: Process of mapping the navigation space and visualizing the resulting navigation mesh.

- **Ensure valid pose:** For the robot assembly to function, the robot needs to start at a pose where it is not fully stretched to reach its maximum radius. This is easily avoidable by putting the robot between two nodes, as it would usually be. This requirement arises from details in the move-to-node task that first pushes outwards to release from the current node, something not possible with the robot on its maximum radius.

  The robot is moved to grab another node by enabling the ”Grab nearby object (I.K)”, clicking on a node to select it and enabling the motors. Four inverse kinematics solutions are possible and can be chosen with the 0-3 buttons. The optimal button automatically selects the closest collision-free solution.

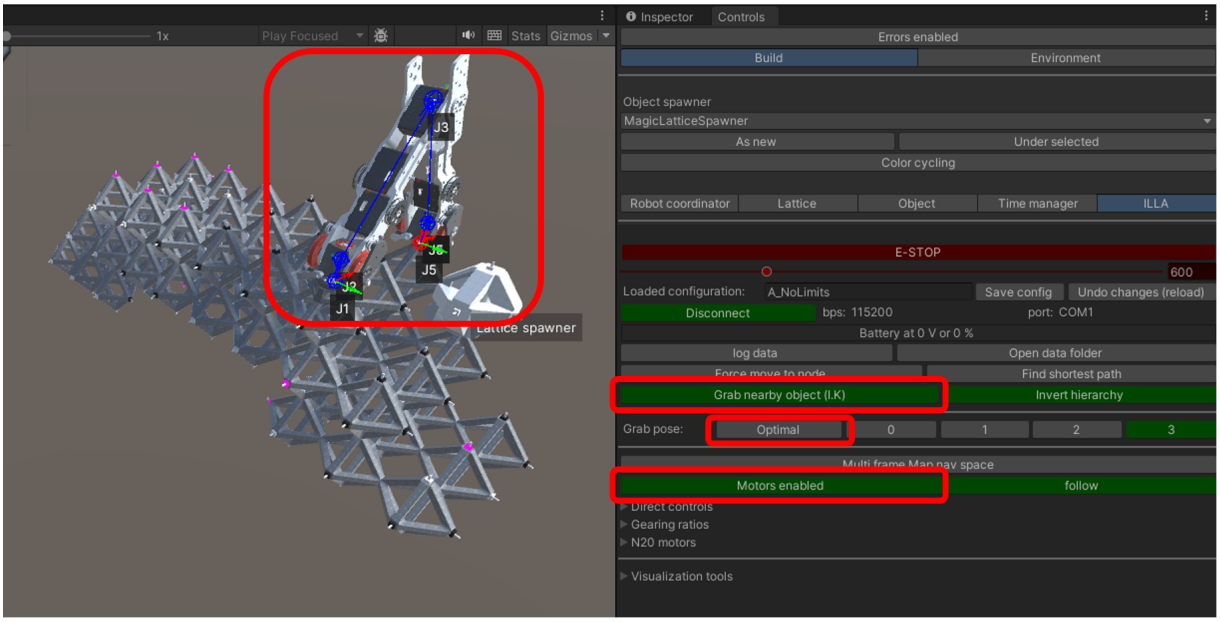


Figure 7.5: Process of actuating the assembler robot to a valid start pose by moving the tool onto a nearby node.

- **Enable auto assemble in the robot coordinator:** Set the robot coordinator to auto assemble mode. In this mode, it matches available building blocks to assembly locations in the construction front to generate work packages and auctions them to the assembler robots. Alternatively, to only perform a single construction step, the construction step button can be used. In the case at hand, the lattices shown in transparent red were unplaced using the UI to have some unbuilt structure to construct. Figure 7.6 shows the first work package being overlayed and how the UI section of the robot updates with a button showing the work package being executed and allowing to stop it.

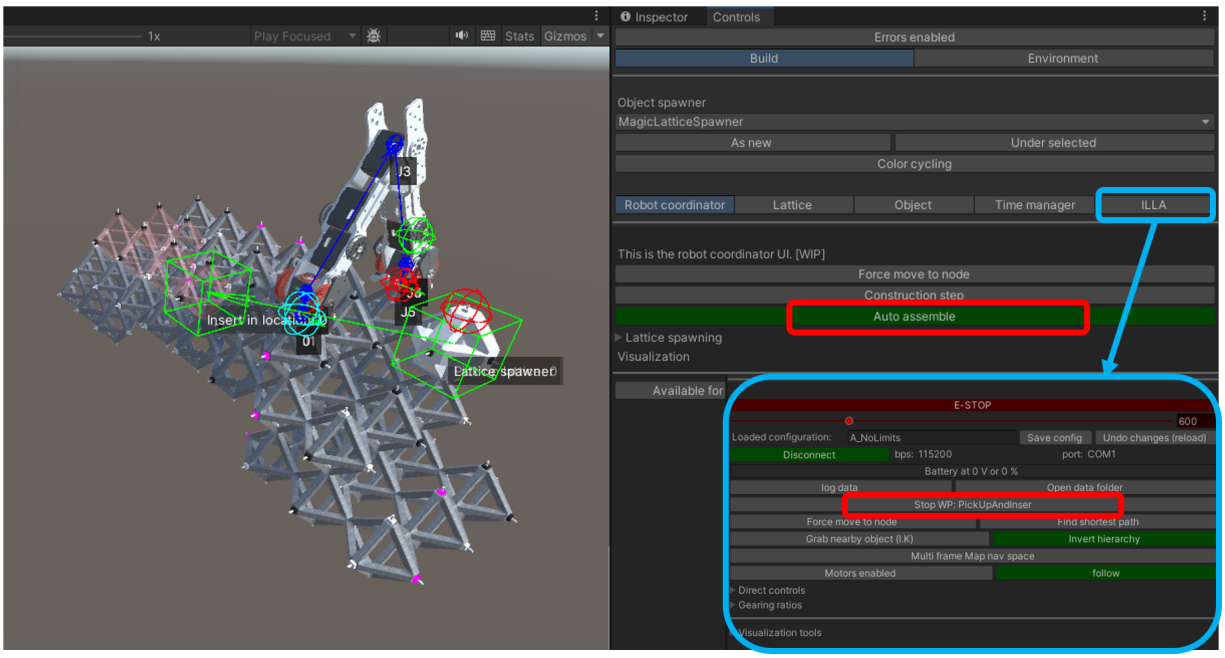


Figure 7.6: Software platform in auto assembly mode. The 3D view displays the overlay of the pick and place work package being executed. The UI of the assembler robot is updated to show the work package being executed and allowing to stop it.

- **Feed building material:** As the build progresses, ensure the assembly process has enough building blocks. This can be done by placing and spawning available lattices using the spawning system or if a magic lattice spawner exists, ensuring there is always a lattice in the spawner.

These steps allow setting up the software environment for a construction process. In the next sections, small demos of construction and navigation in real life are presented that highlight different functionalities of the assembly system.

7.1 Demonstration of digital material transportation

In this initial demo, a small elongated bridge-like lattice structure is presented. Here, a robot and an available lattice are placed on one side of the structure to be assembled and the target assembly location on the opposite side. Thus the robot must pick up and transport a building block to the opposite side and then insert it. This process is visually presented in Figure 7.7.

As can be seen, the robot first grabs the available lattice with the tool or free gripper by screwing onto it in step 4. The assembler then lifts it from its position and approaches the storage location on the opposite differential joint. When the lattice is in close proximity to the storage location, it magnetically attaches, as seen in step 7, allowing the gripper that was holding it to unscrew and release it. With this gripper free, the robot starts walking and traversing the structure. This is done by grabbing one lattice with the free gripper, visualised in step 11, to then swap bases by unscrewing the opposite end, see step 13, and sequentially repeating this process by moving it to another lattice. As evidenced in this process, the walking must finished with the same base gripper as it started to guarantee access to the stored lattice.

This is implemented in the task-breaking process, where additional moves are added to the planned path to ensure this is satisfied. Once the robot has finished walking, it takes the lattice from storage by grabbing it with the tool gripper, visible in step 20, pulling it out like in step 21 and then inserting it into the assembly location.

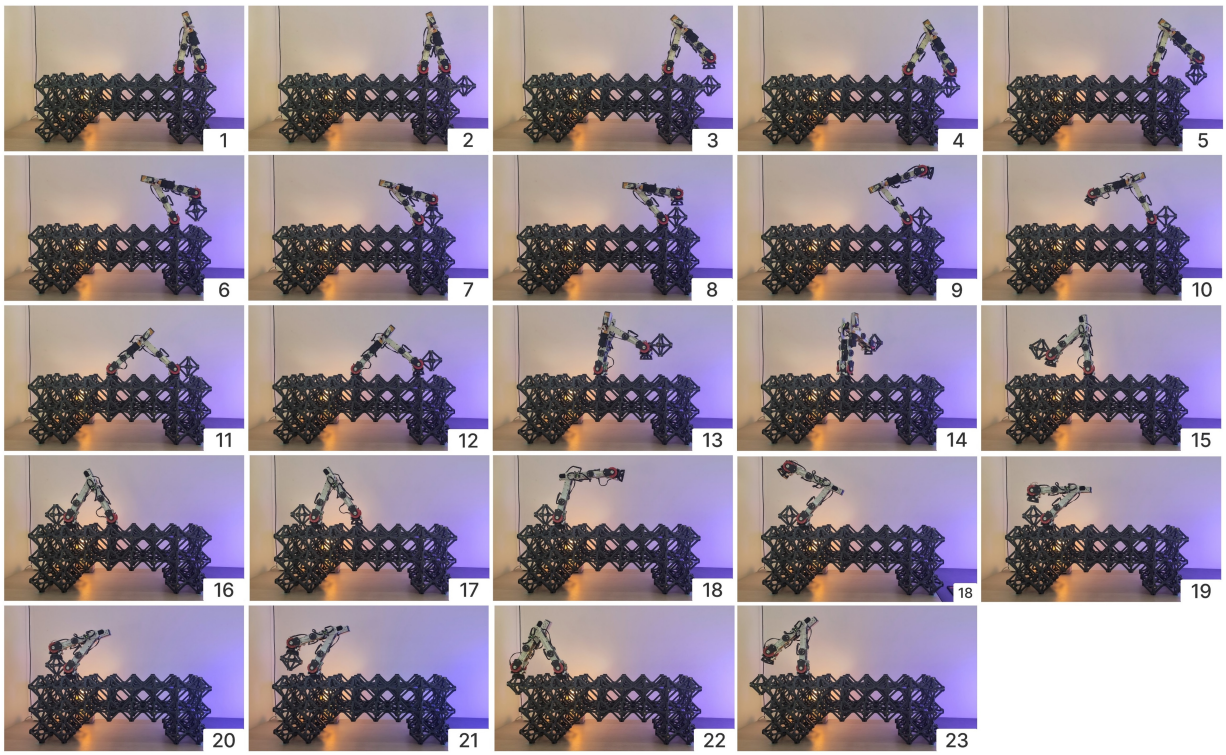


Figure 7.7: Sequence of images showing the robot assembler walking and transporting digital material building blocks across a bridge-like structure.

7.2 Demonstration of a cantilever construction

Having demonstrated the walking and material transport capabilities in the previous section, a short demonstration of autonomous assembly is now presented. Here the robot elongates the bridge by constructing a small cantilever section and then walking on top of it to demonstrate its strength. To shorten this demo and make it feasible to be photographed in this report, the construction materials are fed from the centre of the structure, thus not requiring to walk and traverse the bridge every time.

This construction process is shown in Figure 7.8. The first six rows describe the process of picking up a lattice building block and placing it on the structure by following the six steps described by the column headers. First, the robot approaches the lattice to pick up and then grabs the lattice by screwing onto it with the tool gripper screwdriver. Once the lattice is secured, the assembler starts moving towards the insertion location. As it approaches the insertion location, the lattice building block is magnetically connected to the structure. The gripper's screwdriver is then disengaged from the recently inserted lattice, and the robot retracts from the insertion position. This process is then sequentially repeated for different lattices until a small six-lattice cantilever extension is built. This extension is made of a lattice plane on the top row of the bridge and a single additional lattice situated on the bottom level for extra strength for moment loadings.

In the last row, the robot is shown walking on the newly constructed magnetically attached structure to demonstrate the strength of this magnetic connection instead of adding additional building blocks.

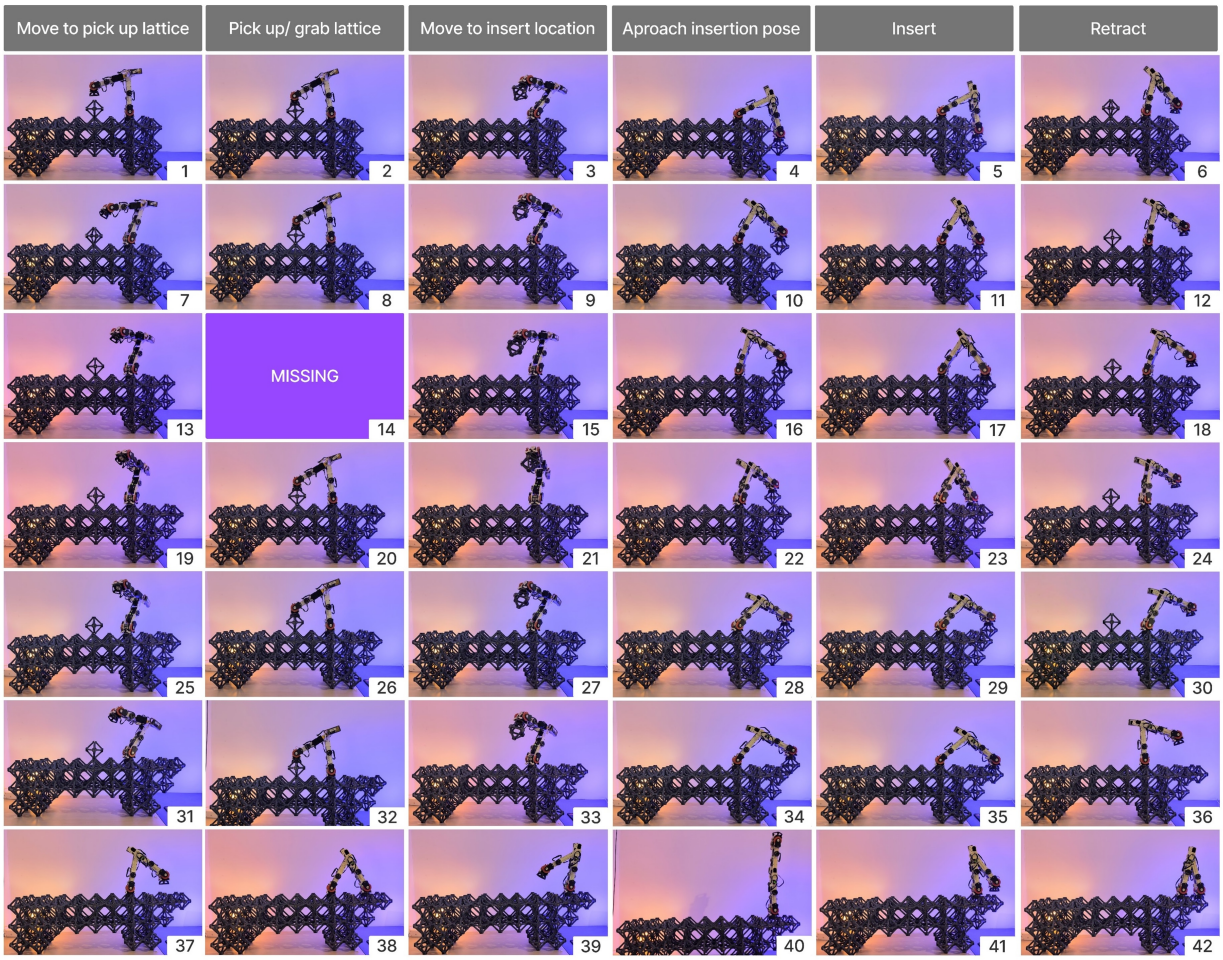


Figure 7.8: Sequence of images showcasing the extension of the bridge by building a cantilever section. Each row represents the placement of one additional building block and in the last row, the assembler walks on top of the assembled cantilever to demonstrate its strength.

7.3 Demonstration of curved lattice traversal and climbing

To further demonstrate the flexibility of both the robotic hardware platform and the inverse kinematics powered software planner, a non-regular structure composed of curved lattices is traversed. This is a first in the field of autonomous assembly of digital materials that is traditionally limited to pre-computed grid fixed motions. In the case at hand, an arch composed of 9 curved lattices like those shown in Figure 4.5 is climbed in two different manners. Such curved lattices can be used to locally tailor the lattice orientation to the load direction and more efficiently utilize the digital material. This is exemplified by the arch assembly presented that could be used for building bridge-like structures that function under compression.

Firstly, the arch structure is climbed from its side as shown in Figure 7.9. In this demonstration, the robot walks on the outer curved surface of the arch, similarly to how it traverses the flat surface of the bridge in the previously seen examples. The robot first screws its tool gripper screwdriver into a nearby node, to then swap the base by unscrewing the previous base gripper. This process is repeated until the robot reaches the desired location. For this demonstration, the robot's reach radius was decreased to force it to take additional middle steps in this climbing process.

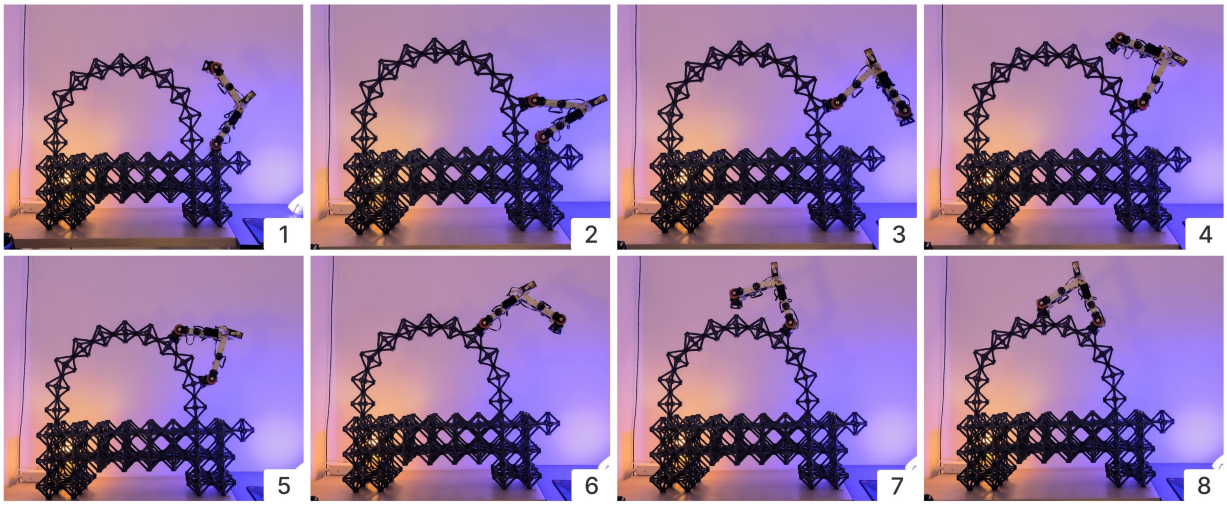


Figure 7.9: Process of climbing an arch-like structure by walking on its outer surface.

Figure 7.10 shows an alternative more direct climbing method by grabbing onto the structure, hanging off it and performing a pull-up-like motion. An additional lattice was added in front of the arch to show this motion perpendicular to the camera plane. In this case, the robot was forced to first engage onto the bottom node of the lattice to display the agility and its capability to attach to alternative nodes of the same lattice, as shown in step 6.

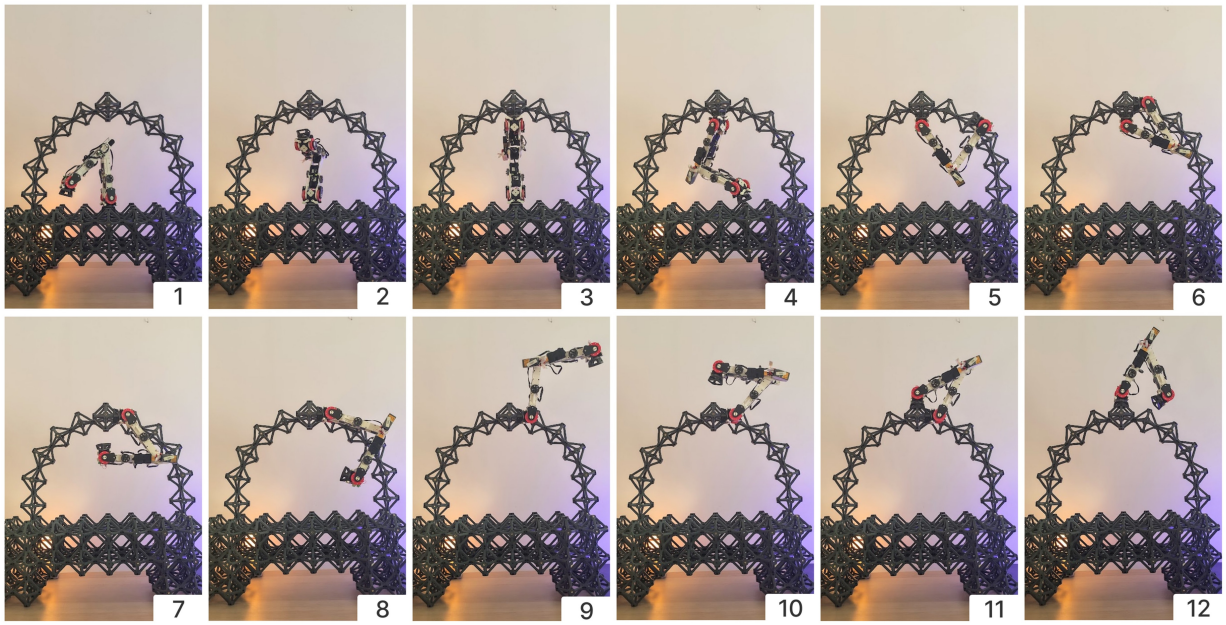


Figure 7.10: Process of climbing a structure in a pull-up like hanging motion.

Part III

Discussion

8

Conclusion

This thesis has demonstrated how an inexpensive, fully 3D-printed inchworm robot assembler can be designed to maximize available torque and the range of motions possible. This is done by the use of geared cable-driven differential joints with partial torque sharing across axes and a structural design that utilizes the heavy actuator metallic enclosures as load-carrying structures. Additionally, the lattice-to-lattice and robot-to-lattice attachment vectors are unified by utilizing a screw-in attachment mechanism providing the same stiff and reliable connection for both scenarios using minimal hardware. This assembler was tested in a wide range of motions and has demonstrated excellent strength and agility in performing these movements while being battery-powered and carrying additional building blocks.

Additionally, and more importantly, this thesis demonstrates how powerful, easy-to-use and flexible software tools can be developed to bridge the gap between 3D models and constructed geometries. The software developed is designed with expandability to multiple assembler types in mind using a work package auction system while leveraging inverse kinematics to plan the robot movements, adding a new degree of flexibility to this blocky field. A path-planning approach that takes into consideration not only the location but also the state or pose of the robot was also developed. This, combined with the abstraction layers built into the software, allows for a high-level interaction environment.

Moreover, some conclusions are presented on the delicate balancing act between computational cost and performance. Microscale construction, like ribosomal protein assembly, presents building blocks in scales resembling that of transistors in modern computers and small movements with minimal energy costs. On the other hand, assembly at the meso and macro scales involves building blocks much larger than billions of transistors and with movement energy costs that surpass the energy expense of movement computation. The volume, cost and energy requirements of the information domain and the physical domain diverge depending on the size of the building block, how close it is to the transistor size and how much energy is required to plan and perform a movement. That is the relation between the physical and the informational domains. Based on this reasoning, it can be concluded that the balance between computational cost and performance is determined not by the number of building blocks or assemblers but by their size.

Looking in retrospect at the primary design objective of this thesis and the demonstrations presented in Chapter 7, it can be said this objective, presented below, was successfully satisfied.

Research Objective

Develop a complete end-to-end autonomous digital material assembly system with flexibility, adaptability and expandability in mind

As evidenced throughout the report, a flexible, comprehensive, and user-friendly toolset has been successfully developed and demonstrated, expanding the capabilities shown by other autonomous assembly systems on certain fronts like navigation and flexibility.

9 Future work

Throughout this thesis, a complete platform for autonomous assembly was presented, from the robotic asset to the control software and the material system. This platform is an enabler for further research.

9.1 Significance of work

This work represents the first stepping stone towards a greater vision within the research group, where autonomous assembly and living biological materials are integrated into a unique, dual-scale living and sensing structure. Through the use of building blocks made from mycelium-based composites (MBCs), we can develop sensing structures that present micro-scale features and material transportation within evolving hyphae networks. By leveraging the autonomous assembly methods introduced in this work, these structures can be expanded from the micrometre scale into the meter and kilometre scales achieving a truly high dynamic range structure. In this manner, this work aims to become the starting point and enabler of further developments in the field. The work presented maximizes design freedom and presents abstraction layers that simplify further development and allow it to transition to increasingly abstract spaces. This platform represents not an end on its own but the beginning of a wider research path.

The following sections discuss the main directions in which the work presented can evolve as well as some points that require further polishing and attention.

9.2 Parametric building blocks and on-demand printing

Imagine a system where the optimal mechanical properties and density of a lattice building block element are calculated via an optimization process and a parametric lattice is automatically generated to satisfy such requirements. The lattice is then 3D printed for a robot to pick it up and place it in the structure. Figure 9.1 presents what such a process could look like. An optimization process determines which lattice type to use and its parameters such as density, strut thickness and so on. A scriptable CAD software such as Fusion360 with Python creates a 3D model from those parameters. The model is then sliced and sent to a 3D printer. After the print is finished a robot can pick up the lattice to place it.

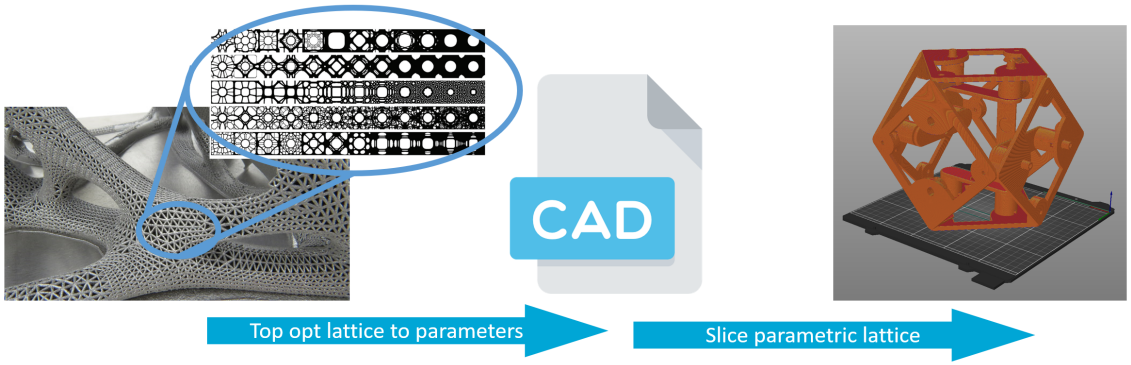


Figure 9.1: On-demand printing of variable and optimized lattice morphologies. Lattice figures courtesy of [99].

This could allow for extended lifetime reparability with a minimal logistical footprint as a large inventory of lattices is not needed, but rather, the right building block can be manufactured on demand from a common material spool without the need to manage large inventories. This capability benefits the use cases that naturally suit autonomous systems like remote and hazardous environments such as radioactive areas or space exploration

9.3 Inchworm assembler redesign

Small and inexpensive assemblers capable of walking on the structure with enough dexterity and lifting capability, are challenging to design and build. The inchworm robot developed presents an advanced joint configuration and mechanical design. However, this isn't without some issues.

The cable-driven differential joint brings freedom in the placement of the motors and enables a high-performance, low backlash 3d printable traction system. Nevertheless, it presents challenges during assembly. First of all, installing the cables is a labour-intensive process best performed by 2 pairs of hands. When securing the cables, m3 screws are used to clamp the cable. During the screwing process, the 3D-printed plastic cable channels deform, limiting their re-usability and repairability. Additionally, the set screws utilized suffered from yielding of their small hex socket, complicating the process and necessitating a drop of glue during installation for additional reliability. Lastly, it is difficult to achieve the correct level of tension in the steel cable due to deformations in the parts that make the cable tensioning system which limits the tensioning range. With more time and budget, redesigning the cable system or transitioning to a geared differential joint would be advisable.

Furthermore, after testing the design, it was found that the dynamic response of the robot could be overhauled with additional torque. The actuation system is capable but can be further improved. The differential joint gearing ratio could be modified to provide more torque, or the motors could be changed. With more budget, the Dynamixel servos showcased in Table 5.1 offering 40% extra torque at the same mass could be used. Or, externally geared BLDC motors with encoders on the robot axis, instead of the motors themselves, could offer a better response. Nevertheless, these options increase assembler costs.

9.4 Mutli robot types

Continuing in the line of robotic assets, the software platform has been built with the capability of utilizing different kinds of robotic assemblers simultaneously, enabling specialization of work and cooperation.

Figure 9.2: Flying drone transporting a lattice.

The inchworm robots presented have precise positioning and the potential capability to screw lattices together for a strong assembly. However, they are not the fastest when moving long distances and transporting material. Many of these robots can cooperate to transport lattices faster, like a fire bucket brigade, but this requires a high robot density and still has a limited speed. Alternatively, different robotic assets could be used for transporting lattices long distances. For instance, wheeled, tracked, or flying assets could play the role of long-distance lattice transporters while inchworm robots place and screw the lattices. This is illustrated in Figure 9.2 with a flying drone carrying a lattice structure. Integrating different kinds of assemblers could lead to large gains in build efficiency thanks to the specialization of work. The currently existing separation between build and robot controllers and the work package bidding process make such an integration feasible within the existing architecture.

9.5 Large scale assembly capability

The software platform developed has features to import and define geometries, plan robot movements, and monitor the build process process. It has all the elements necessary to demonstrate large-scale autonomous building capabilities. For this thesis, around 200 lattice elements were printed and manufactured in a few weeks, but up to one or two thousand could be made within a few months. Demonstrating the large-scale construction capabilities of the system does not represent an advance in features, but would nevertheless serve as a strong statement. This would also provide the time and focus to solidify, clean, document and improve the reliability of the system. Some suggestions of items that can be added or improved for quality of life and performance to enable scaling to larger structures are now presented. Some of these suggestions can be great exercises to familiarize yourself with the program and development within Unity.

- **Navigation mesh editing:** Right now the navigation is built from scratch whenever a modification is made. This is however not the most efficient process for large structures. Ideally, a system is built to locally edit the navigation mesh.
- **Navigation mesh saving:** Together with editing the navigation mesh, saving it would enhance performance as it would allow the spawning of large complex objects without the need to perform a long tedious mapping process.
- **Collision detection performance:** The performance of the collision detection is poor at about 2ms per collision check, however, most of the compute time is not on the collision detection itself but rather on setting the robot in the correct pose to be tested and generating the ILLA Mesh. This is easily optimizable as explained in Subsection 6.7.10.
- **Configuration file auto-assignment:** The configuration file is assigned by hand from a drop-down menu before connecting to a robot, however, it could perhaps be stored locally on the robot or be looked up with a robot ID when connecting. This would more easily allow scaling to a higher number of robots without requiring memorization of the different configuration names.
- **Sensorized continuous lattice pickup location:** Develop a physical lattice spawn location that utilizes simple sensors like end stops to detect the presence of lattices. These spawn locations could be used to easily and reliably supply a construction process with lattices.

References

[1] Ingersoll. *MASTERPRINT® 3X brochure*. Available online: https://en.machinetools.camozzi.com/products/additive-manufacturing/all-products/masterprint-.kl?form=ok. 2024. URL: https://en.machinetools.camozzi.com/products/additive-manufacturing/all-products/masterprint-.kl?form=o.

[2] Nanoscribe. *Photonic Professional GT2 brochure*. Available online: https://www.nanoscribe.com/fileadmin/Nanoscribe/Solutions/Photonic_Professional_GT2/Folder_PPGT2_A4.pdf. 2024. URL: https://www.nanoscribe.com/fileadmin/Nanoscribe/Solutions/Photonic_Professional_GT2/Folder_PPGT2_A4.pdf.

[3] Jennifer Louten. "Virus Structure and Classification". In: Dec. 2016, pp. 19–29. DOI: 10.1016/B978-0-12-800947-5.00002-8.

[4] Stephen C. Sillett, James C. Spickler, and Robert Van Pelt. "CROWN STRUCTURE OF THE WORLD'S SECOND LARGEST TREE". In: *Madroño* 47.2 (2000), pp. 127–133. URL: http://www.jstor.org/stable/41425355 (visited on 09/08/2022).

[5] Prashant Patil. *Micro-lattice fabricated using two-photon polymerization*. https://cba.mit.edu/media/microlattice/index.html. Accessed: 2024-01-23. MIT Center for Bits and Atoms, 2016.

[6] Tatiana V. Karpinets et al. "RNA:protein ratio of the unicellular organism as a characteristic of phosphorous and nitrogen stoichiometry and of the cellular requirement of ribosomes for protein synthesis". In: *BMC Biology* 4.30 (Sept. 2006). DOI: 10.1186/1741-7007-4-30. URL: https://doi.org/10.1186/1741-7007-4-30.

[7] Jonathan Hiller and Hod Lipson. "Design and analysis of digital materials for physical 3D voxel printing". In: *Rapid Prototyping Journal - RAPID PROTOTYPING J* 15 (Mar. 2009), pp. 137–149. DOI: 10.1108/13552540910943441.

[8] Will Langford and Neil Gershenfeld. "A Discretely Assembled Walking Motor". In: *2019 International Conference on Manipulation, Automation and Robotics at Small Scales (MARSS)*. 2019, pp. 1–6. DOI: 10.1109/MARSS.2019.8860989. URL: https://doi.org/10.1109/MARSS.2019.8860989.

[9] *Automated Assembly of Electronic Digital Materials*. Vol. Volume 2: Materials; Biomanufacturing; Properties, Applications and Systems; Sustainable Manufacturing. International Manufacturing Science and Engineering Conference. V002T01A013. June 2016. DOI: 10.1115/MSEC2016-8627. eprint: https://asmedigitalcollection.asme.org/MSEC/proceedings-pdf/MSEC2016/49903/V002T01A013/4425333/v002t01a013-msec2016-8627.pdf. URL: https://doi.org/10.1115/MSEC2016-8627.

[10] Lorna J. Gibson and Michael F. Ashby. *Cellular Solids: Structure and Properties*. 2nd ed. Cambridge Solid State Science Series. Cambridge University Press, 1997. DOI: 10.1017/CBO9781139878326.

[11] Benjamin Jenett. "Discrete Mechanical Metamaterials". https://cba.mit.edu/docs/theses/20.09.jenett.pdf. PhD thesis. Massachusetts Institute of Technology, 2020.

[12] Haozhang Zhong et al. "The Gibson-Ashby model for additively manufactured metal lattice materials: Its theoretical basis, limitations and new insights from remedies". In: *Current Opinion in Solid State and Materials Science* 27.3 (2023), p. 101081. DOI: https://doi.org/10.1016/j.cossms.2023.101081. URL: https://www.sciencedirect.com/science/article/pii/S1359028623000268.

[13] Kwang-Min Park, Kyung-Sung Min, and Young-Sook Roh. "Design Optimization of Lattice Structures under Compression: Study of Unit Cell Types and Cell Arrangements". In: *Materials* 15.1 (2022). DOI: 10.3390/ma15010097. URL: https://www.mdpi.com/1996-1944/15/1/97.

[14] Sebastian C. Kapfer et al. "Minimal surface scaffold designs for tissue engineering". In: *Biomaterials* 32.29 (2011), pp. 6875–6882. DOI: https://doi.org/10.1016/j.biomaterials.2011.06.012. URL: https://www.sciencedirect.com/science/article/pii/S0142961211006776.

[15] Oraib Al-Ketan, Reza Rowshan, and Rashid K. Abu Al-Rub. "Topology-mechanical property relationship of 3D printed strut, skeletal, and sheet based periodic metallic cellular materials". In: *Additive Manufacturing* 19 (2018), pp. 167–183. DOI: https://doi.org/10.1016/j.addma.2017.12.006. URL: https://www.sciencedirect.com/science/article/pii/S2214860417303767.

[16] Tobias Maconachie et al. “SLM lattice structures: Properties, performance, applications and challenges”. In: *Materials & Design* 183 (2019), p. 108137. DOI: https://doi.org/10.1016/j.matdes.2019.108137. URL: https://www.sciencedirect.com/science/article/pii/S0264127519305751.

[17] Zaki Alomar and Franco Concli. “A Review of the Selective Laser Melting Lattice Structures and Their Numerical Models”. In: *Advanced Engineering Materials* 22.12 (2020), p. 2000611. DOI: https://doi.org/10.1002/adem.202000611. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/adem.202000611. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/adem.202000611.

[18] Christine E. Gregg, Joseph H. Kim, and Kenneth C. Cheung. “Ultra-Light and Scalable Composite Lattice Materials”. In: *Advanced Engineering Materials* 20.9 (2018), p. 1800213. DOI: https://doi.org/10.1002/adem.201800213. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/adem.201800213.

[19] Gawde Pranav Ravindra. “Modularity in Lattice structures for Circular Product Design”. PhD thesis. 2021.

[20] V.S. Deshpande, M.F. Ashby, and N.A. Fleck. “Foam topology: bending versus stretching dominated architectures”. In: *Acta Materialia* 49.6 (2001), pp. 1035–1040. DOI: https://doi.org/10.1016/S1359-6454(00)00379-7. URL: https://www.sciencedirect.com/science/article/pii/S1359645400003797.

[21] R. Budynas and K. Nisbett. *Shigley's Mechanical Engineering Design 9th Edition*. McGraw-Hill Education, 2012. URL: https://books.google.es/books?id=XX48mAEACAAJ.

[22] V.S. Deshpande, M.F. Ashby, and N.A. Fleck. “Foam topology: bending versus stretching dominated architectures”. In: *Acta Materialia* 49.6 (2001), pp. 1035–1040. DOI: https://doi.org/10.1016/S1359-6454(00)00379-7. URL: https://www.sciencedirect.com/science/article/pii/S1359645400003797.

[23] I. Maskery et al. “Compressive failure modes and energy absorption in additively manufactured double gyroid lattices”. In: *Additive Manufacturing* 16 (2017), pp. 24–29. DOI: https://doi.org/10.1016/j.addma.2017.04.003. URL: https://www.sciencedirect.com/science/article/pii/S2214860417301203.

[24] Recep Gümrük, R.A.W. Mines, and Sami Karadeniz. “Static mechanical behaviours of stainless steel microlattice structures under different loading conditions”. In: *Materials Science and Engineering: A* 586 (2013), pp. 392–406. DOI: https://doi.org/10.1016/j.msea.2013.07.070. URL: https://www.sciencedirect.com/science/article/pii/S092150931300840X.

[25] Benjamin Jenett et al. “Discretely assembled mechanical metamaterials”. In: *Science Advances* 6.47 (2020), eabc9943. DOI: 10.1126/sciadv.abc9943. eprint: https://www.science.org/doi/pdf/10.1126/sciadv.abc9943. URL: https://www.science.org/doi/abs/10.1126/sciadv.abc9943.

[26] Alex Luijten. *Self-assembling ultra-lightweight lattice structures*. 2020.

[27] Kirstin H. Petersen et al. “A review of collective robotic construction”. In: *Science Robotics* 4.28 (2019), eaau8479. DOI: 10.1126/scirobotics.aau8479. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.aau8479. URL: https://www.science.org/doi/abs/10.1126/scirobotics.aau8479.

[28] Kirstin Petersen, Radhika Nagpal, and Justin Werfel. “TERMES: An autonomous robotic system for three-dimensional collective construction”. In: vol. 7. Cited by: 28; All Open Access, Bronze Open Access, Green Open Access. 2012, pp. 257 – 264. DOI: 10.15607/rss.2011.vii.035. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84959307873&doi=10.15607%2frss.2011.vii.035&partnerID=40&md5=89af89813da494754b87149e99f8cb76.

[29] Frederico Augugliaro et al. “The Flight Assembled Architecture installation: Cooperative construction with flying machines”. In: *IEEE Control Systems Magazine* 34.4 (2014), pp. 46–64. DOI: 10.1109/MCS.2014.2320359.

[30] Kathrin Dörfler et al. “Mobile Robotic Brickwork”. In: Feb. 2016, pp. 204–217. DOI: 10.1007/978-3-319-26378-6_15.

[31] Benjamin Jenett and Kenneth Cheung. “BILL-E: Robotic Platform for Locomotion and Manipulation of Lightweight Space Structures”. In: *American Institute of Aeronautics and Astronautics* (2017). Available at: https://cba.mit.edu/docs/papers/17.06.scitech.bille.pdf.

[32] Benjamin Jenett et al. “Material-Robot System for Assembly of Discrete Cellular Structures”. In: *IEEE Robotics and Automation Letters* PP (July 2019), pp. 1–1. DOI: 10.1109/LRA.2019.2930486.

[33] In-Won Park et al. “SOLL-E: A Module Transport and Placement Robot for Autonomous Assembly of Discrete Lattice Structures”. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2023), pp. 10736–10741. URL: https://api.semanticscholar.org/CorpusID:266200071.

[34] Keith Kotay and Daniela Rus. “Navigating 3D Steel Web Structures with an Inchworm Robot”. In: (Aug. 1996). DOI: 10.1109/IROS.1996.570701.

[35] Son T. Nguyen et al. “A Practical Climbing Robot for Steel Bridge Inspection”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 9322–9328. DOI: 10.1109/ICRA40945.2020.9196892.

[36] Shichao Gu et al. “Optimal Collision-Free Grip Planning for Biped Climbing Robots in Complex Truss Environment”. In: *Applied Sciences* 8.12 (2018). URL: https://www.mdpi.com/2076-3417/8/12/2533.

[37] Jae-Hee Kim et al. “Automatic Grasping of a Pole Climbing Robot using a Visual Camera with Laser Line Beams”. In: *MATEC Web of Conferences* 42 (Feb. 2016), p. 03005. DOI: 10.1051/matecconf/20164203005.

[38] Tirthankar Bandyopadhyay et al. “Magneto: A Versatile Multi-Limbed Inspection Robot”. In: Oct. 2018, pp. 2253–2260. DOI: 10.1109/IROS.2018.8593891.

[39] Allan Costa et al. “Algorithmic Approaches to Reconfigurable Assembly Systems”. In: *2019 IEEE Aerospace Conference*. 2019, pp. 1–8. DOI: 10.1109/AERO.2019.8741572. URL: https://doi.org/10.1109/AERO.2019.8741572.

[40] Benjamin Jenett and Daniel Cellucci. “A mobile robot for locomotion through a 3D periodic lattice environment”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 5474–5479. DOI: 10.1109/ICRA.2017.7989644.

[41] Olivia Formoso et al. “MMIC-I: A Robotic Platform for Assembly Integration and Internal Locomotion through Mechanical Meta-Material Structures”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 7303–7309. DOI: 10.1109/ICRA48891.2023.10161263.

[42] Jörg P. Müller and Markus Pischel. *The agent architecture InteRRaP : concept and application*. 1993. DOI: http://dx.doi.org/10.22028/D291-24922.

[43] Michael K. Sahota. “Reactive Deliberation: An Architecture for Real-Time Intelligent Control in Dynamic Environments”. In: *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 2)*. AAAI’94. Seattle, Washington, USA: American Association for Artificial Intelligence, 1994, pp. 1303–1308.

[44] Anand S Rao, Michael P Georgeff, et al. “BDI agents: from theory to practice.” In: *Icmas*. Vol. 95. 1995, pp. 312–319.

[45] A. Anand, M. Nithya, and TSB Sudarshan. “Coordination of mobile robots with master-slave architecture for a service application”. In: *2014 International Conference on Contemporary Computing and Informatics (IC3I)*. 2014, pp. 539–543. DOI: 10.1109/IC3I.2014.7019647.

[46] Richard Morris, Lionel Tarassenko, and M. Kenward. *Cognitive systems - information processing meets brain science*. Jan. 2006. DOI: 10.1016/B978-0-12-088566-4.X5000-4.

[47] Neil J Smelser, Paul B Baltes, et al. *International encyclopedia of the social & behavioral sciences*. Vol. 11. Elsevier Amsterdam, 2001.

[48] CJ van Leeuwen. “Self-Organizing Multi-Agent Systems”. PhD thesis. 2021. DOI: https://doi.org/10.4233/uuid:face1276-5025-4853-996a-30d6771cf24d. URL: https://doi.org/10.4233/uuid:face1276-5025-4853-996a-30d6771cf24d.

[49] GIOVANNA DI MARZO SERUGENDO, MARIE-PIERRE GLEIZES, and ANTHONY KARAGEORGOS. “Self-organization in multi-agent systems”. In: *The Knowledge Engineering Review* 20.2 (2005), pp. 165–189. DOI: 10.1017/S0269888905000494.

[50] Helen F. McCreery et al. “Hysteresis stabilizes dynamic control of self-assembled army ant constructions”. In: *Nature Communications* 13.1 (2022). DOI: 10.1038/s41467-022-28773-z. URL: https://doi.org/10.1038/s41467-022-28773-z.

[51] Gauthier Picard. *Self-Organisation in Multi-Agent Systems*. https://www.emse.fr/~picard/cours/3A/masterWI/coursSO.pdf. Accessed: 2022–19-09.

[52] Samuel A. Ocko, Alexander Heyde, and L. Mahadevan. “Morphogenesis of termite mounds”. In: *Proceedings of the National Academy of Sciences* 116.9 (2019), pp. 3379–3384. DOI: 10.1073/pnas.1818759116. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1818759116. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1818759116.

[53] Oebele Herman Bruinsma. “An analysis of building behaviour of the termite Macrotermes subhyalinus (Rambur)”. PhD thesis. 1979.

[54] Francis Heylighen. “Stigmergy as a universal coordination mechanism I: Definition and components”. In: *Cognitive Systems Research* 38 (2016). Special Issue of Cognitive Systems Research – Human-Human Stigmergy, pp. 4–13. DOI: https://doi.org/10.1016/j.cogsys.2015.12.002. URL: https://www.sciencedirect.com/science/article/pii/S1389041715000327.

[55] Ralph Beckers, Owen E. Holland, and Jean-Louis Deneubourg. “Fom Local Actions to Global Tasks: Stigmergy and Collective Robotics”. In: *Prerational Intelligence: Adaptive Behavior and Intelligent Systems Without Symbols and Logic, Volume 1, Volume 2 Prerational Intelligence: Interdisciplinary Perspectives on the Behavior of Natural and Artificial Systems, Volume 3*. Ed. by Holk Cruse, Jeffrey Dean, and Helge Ritter. Springer Netherlands, 2000, pp. 1008–1022. DOI: 10.1007/978-94-010-0870-9_63. URL: https://doi.org/10.1007/978-94-010-0870-9_63.

[56] Michael Rubenstein and Wei-Min Shen. “Scalable self-assembly and self-repair in a collective of robots”. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009, pp. 1484–1489. DOI: 10.1109/IROS.2009.5354716.

[57] S. M. LaValle. *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. URL: http://planning.cs.uiuc.edu/.

[58] Prof. Deepak Khemani, Dep Computer Science IIT Madras. *Artificial Intelligence: Search Methods for Problem Solving*. University Lectures. 2014. URL: https://nptel.ac.in/courses/106106126.

[59] Yanrong Hu and S.X. Yang. “A knowledge-based genetic algorithm for path planning of a mobile robot”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA ’04. 2004*. Vol. 5. 2004, 4350–4355 Vol.5. DOI: 10.1109/ROBOT.2004.1302402. URL: https://doi.org/10.1109/ROBOT.2004.1302402.

[60] Inc. Freescale Semiconductor. *How to Increase the Analog-to-Digital Converter Accuracy in an Application*. Tech. rep. Rev. 0, 01/2016. Freescale Semiconductor, Inc., 2016. URL: https://www.nxp.com/docs/en/application-note/AN5250.pdf.

[61] Richard E. Bellman. *Dynamic Programming*. Princeton, New Jersey, USA: Princeton University Press, 1957.

[62] Miloš Šeda. “Roadmap Methods vs. Cell Decomposition in Robot Motion Planning”. In: *Proceedings of the 6th WSEAS International Conference on Signal Processing, Robotics and Automation*. ISPRA’07. Corfu Island, Greece: World Scientific, Engineering Academy, and Society (WSEAS), 2007, pp. 127–132. DOI: 10.5555/1353685.1353707. URL: https://doi.org/10.5555/1353685.1353707.

[63] Hanlin Niu et al. “An energy-efficient path planning algorithm for unmanned surface vehicles”. In: *Ocean Engineering* 161 (May 2018), pp. 308–321. DOI: 10.1016/j.oceaneng.2018.01.025. URL: https://doi.org/10.1016/j.oceaneng.2018.01.025.

[64] Jean-Claude Latombe. “Exact Cell Decomposition”. In: *Robot Motion Planning*. Boston, MA: Springer US, 1991, pp. 200–247. DOI: 10.1007/978-1-4615-4022-9_5. URL: https://doi.org/10.1007/978-1-4615-4022-9_5.

[65] M.L. Fredman and R.E. Tarjan. In.

[66] Xueqiao Xu. *PathFinding.js*. https://github.com/qiao/PathFinding.js. 2017.

[67] Chanchal Khemani et al. “Solving Rubik’s Cube Using Graph Theory: ICCI-2017”. In: Jan. 2019, pp. 301–317. DOI: 10.1007/978-981-13-1132-1_24.

[68] Tomas Rokicki et al. “The Diameter of the Rubik’s Cube Group Is Twenty”. In: *SIAM Review* 56.4 (2014), pp. 645–670. DOI: 10.1137/140973499. eprint: https://doi.org/10.1137/140973499. URL: https://doi.org/10.1137/140973499.

[69] David Payton. “Internalized plans: A representation for action resources”. In: *Robotics and Autonomous Systems* 6 (June 1990), pp. 89–103. DOI: 10.1016/S0921-8890(05)80030-2.

[70] Yuzuru Terada and Satoshi Murata. “Modular Structure Assembly using Gradient Field”. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2006, pp. 604–611. DOI: 10.1109/IROS.2006.282542. URL: https://doi.org/10.1109/IROS.2006.282542.

[71] Yuzuru Terada and Satoshi Murata. “Modular Structure Assembly Using Blackboard Path Planning System”. In: *Proceedings of the 23rd International Symposium on Automation and Robotics in Construction*. Ed. by Naruo Kano. Tokyo, Japan: International Association for Automation and Robotics in Construction (IAARC), 2006, pp. 852–857. DOI: 10.22260/ISARC2006/0157. URL: https://doi.org/10.22260/ISARC2006/0157.

[72] David V. Gealy et al. *Quasi-Direct Drive for Low-Cost Compliant Robotic Manipulation*. 2019. arXiv: 1904.03815 [cs.RO].

[73] Ionut Mihai Constantin Olaru, Sébastien Krut, and François Pierrot. "Novel Mechanical Design of Biped Robot SHERPA Using 2 DOF Cable Differential Modular Joints". In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. St. Louis, USA, 2009, pp. 4463–4468. DOI: 10.1109/IROS.2009.5354425.

[74] Tech United Eindhoven. *A differential drive as used in shoulder, elbow, and wrist joints*. https://www.researchgate.net/figure/A-differential-drive-as-used-in-shoulder-elbow-and-wrist-joints_fig4_233881989. [Accessed 5 Dec, 2023]. 2011.

[75] Ondřej Stříteský. *Prusament PC Blend – our new filament for highly durable 3D prints!* https://blog.prusa3d.com/prusament-pc-blend-our-new-filament-for-highly-durable-3d-prints_33511/. 2020.

[76] Stefan Hermann. *Prusament PC Blend Review*. https://www.cnckitchen.com/blog/prusament-pc-blend-review. 2021.

[77] Wire Rope Stunter. *Overview Table Tensile Strength Steel Wire Rope*. https://www.staalkabelstunter.com/en/service/trekkracht-staalkabel/. Accessed: 2023-12-05. 2023.

[78] Lynxmotion. *Lynxmotion Smart Servo Specifications*. https://wiki.lynxmotion.com/info/wiki/lynxmotion/view/lynxmotion-smart-servo/#HSpecifications. Accessed: 2023-12-05. 2023.

[79] ROBOTIS. *XC430-W240-T Specifications*. https://emanual.robotis.com/docs/en/dxl/x/xc430-w240/. Accessed: 2023-12-05. n.d.

[80] ROBOTIS. *XM430-W350-T/R Specifications*. https://emanual.robotis.com/docs/en/dxl/x/xm430-w350/. Accessed: 2023-12-05. n.d.

[81] Dongbu Robot. *Herculex DRS-0101/ DRS-0201 user manual*. Accessed: 2023-12-05. n.d. URL: https://cdn.robotshop.com/media/d/das/rb-das-06/pdf/manual-drs-0201.pdf.

[82] RB1. *LSS - Electrical*. https://wiki.lynxmotion.com/info/wiki/lynxmotion/view/lynxmotion-smart-servo/lss-electrical/. Accessed: 2023-12-05. 2019.

[83] Eric Nantel. *LSS-ADA Board (USB Mini)*. https://wiki.lynxmotion.com/info/wiki/lynxmotion/view/servo-erector-set-system/ses-electronics/ses-modules/lss-adapter-board/. Accessed: 2023-12-05. 2019.

[84] Eric Nantel. *LSS-2MC Board*. https://wiki.lynxmotion.com/info/wiki/lynxmotion/view/servo-erector-set-system/ses-electronics/ses-modules/lss-2mc-board/. Accessed: 2023-12-05. 2021.

[85] Lynxmotion. *AlternativeLSS: Asynchronous control of Lynxmotion LSS smart servos*. https://github.com/Lynxmotion/AlternativeLSS. Accessed: 2023-12-05. 2023.

[86] Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning: Theory amp; Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[87] Weiming Wang et al. "Space-time topology optimization for additive manufacturing : Concurrent optimization of structural layout and fabrication sequence". In: *Structural and Multidisciplinary Optimization* (2020). DOI: 10.1007/s00158-019-02420-6.

[88] H. W. Kuhn. "The Hungarian method for the assignment problem". In: *Naval Research Logistics Quarterly* 2.1-2 (1955), pp. 83–97. DOI: https://doi.org/10.1002/nav.3800020109. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800020109. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109.

[89] Michael Vivet and Pavlo Datsiuk. *HungarianAlgorithm: Hungarian Algorithm Implementation*. C. https://github.com/vivet/HungarianAlgorithm. Accessed: 2024-01-23. 2024.

[90] iHeartGameDev. *How to Program in Unity: State Machines Explained*. https://www.youtube.com/watch?v=Vt8aZDPzRjI. YouTube video. 2019.

[91] *interface - C Reference*. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface. Accessed: [insert date of access].

[92] *Data Structures - Survey of Sparse Matrix Storage Formats*. https://netlib.org/linalg/html_templates/node89.html. Accessed: [insert date of access].

[93] Stefan B. Williams. *Kinematics & Dynamics - Lecture 2*. University of Sydney, MTRX 4700: Experimental Robotics. 2013. URL: https://www.aeromech.usyd.edu.au/MTRX4700/Course_Documents/material/lectures/L2_Kinematics_Dynamics_2013.pdf.

[94] Peter Corke. *Inverse Kinematics for a 2-Joint Robot Arm Using Geometry*. https://robotacademy.net.au/lesson/inverse-kinematics-for-a-2-joint-robot-arm-using-geometry/. Accessed: 2023-12-22. 2023.

[95] *KUKA Robot Range*. Accessed: 2024-01-30. KUKA. Germany, 2023. URL: https://www.kuka.com/-/media/kuka-downloads/imported/87f2706ce77c4318877932fb36f6002d/kuka_robotrange_en.pdf?rev=49723c1edd494f4d922f1a9d3197e060.

[96] Sallam A. Kouritem et al. "A multi-objective optimization design of industrial robot arms". In: *Alexandria Engineering Journal* 61.12 (2022), pp. 12847–12867. DOI: https://doi.org/10.1016/j.aej.2022.06.052. URL: https://www.sciencedirect.com/science/article/pii/S1110016822004355.

[97] Scott B. Nokleby. "Singularity analysis of the Canadarm2". In: *Mechanism and Machine Theory* 42.4 (2007), pp. 442–454. DOI: https://doi.org/10.1016/j.mechmachtheory.2006.04.004. URL: https://www.sciencedirect.com/science/article/pii/S0094114X06000875.

[98] The MRPT Authors. *The Mobile Robot Programming Toolkit (MRPT)*. https://github.com/MRPT/mrpt. GitHub repository. 2023.

[99] Eric Garner et al. "Compatibility in microstructural optimization for additive manufacturing". In: *Additive Manufacturing* 26 (2019), pp. 65–75. DOI: https://doi.org/10.1016/j.addma.2018.12.007. URL: https://www.sciencedirect.com/science/article/pii/S2214860418306808.

A

Kinematics derivations

A joint-to-joint transformation matrix can be developed from the Denavit-Hartenberg parameters, this process process presented in Appendix A is explained in [93]

$$T_i = Rot_{z,\theta_i} Trans_{z,di} Trans_{x,ai} Rot_{x,ai}$$

$$\mathsf{T}_i = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & {}_{\theta_i}c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_1 & -s_1 c_{\alpha 1} & s_{\alpha 1} s_1 & a_1 c_1 \\ s_1 & c_{\alpha 1} c_1 & -s_{\alpha 1} c_1 & a_1 s_1 \\ 0 & s_{\alpha 1} & c_{\alpha 1} & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(A.1)}$$

This is done, with the Denavit Hartemberg parameters of the Inchworm Locomotion Lattice Assember developed in Chapter 5. These parameters are shown in Equation (A.2) on the right side for the standard case and the left for the inverted. Here the terms d_i and a_i represent known assembly constraints

$$\begin{bmatrix} \frac{\pi}{2} & q_1 & 0 & d_1 \\ 0 & q_2 & a_2 & 0 \\ 0 & q_3 & a_3 & 0 \\ -\frac{\pi}{2} & q_4 & 0 & 0 \\ 0 & q_5 & 0 & d_5 \end{bmatrix} \qquad \begin{bmatrix} \frac{\pi}{2} & q_1 & 0 & d_1 \\ 0 & q_2 & a_2 & 0 \\ 0 & q_3 & a_3 & 0 \\ -\frac{\pi}{2} & q_4 & 0 & 0 \\ 0 & q_5 & 0 & d_5 \end{bmatrix} \quad \text{(A.2)}$$

A.1 Non-inverted case forwards and inverse kinematics

Filling in the Denavig Hartemberg parameters for every one of the five joints in the transformation matrix equation in Equation (A.2) we arrive at the solutions for T_1 to T_5. This is shown below for the standard case. The terms c_i and s_i represent the cosine and sine trigonometric functions of the unknown θ_i $c_i = \cos\theta_i$ and $s_i = \sin\theta_i$.

$$T_1 = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_2 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2 c_2 \\ s_2 & c_2 & 0 & a_2 s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_3 = \begin{bmatrix} c_3 & -s_3 & 0 & a_3 c_3 \\ s_3 & c_3 & 0 & a_3 s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4 = \begin{bmatrix} c_4 & 0 & -s_4 & 0 \\ s_4 & 0 & c_4 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_5 = \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

By concatenating these transformations $T_1 \cdot T_2 \cdot T_3 \cdot T_4 \cdot T_5 = T_0^5$ we get the transformation matrix from the base to the gripper situated after the last joint. This represents the forward kinematics solution.

To find a solution of θ values from θ_1 to θ_5 that when used to calculate the joint transformation matrices T_1 to T_5 will yield our desired input pose given in the matrix S

$$S = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = T_1 \cdot T_2 \cdot T_3 \cdot T_4 \cdot T_5 \tag{A.3}$$

Initially, I attempted to find the solution utilizing computerized symbolic manipulation tools, in particular python SymPy and mathematica but a solution wasn't found in reasonable computation times of multiple hours. An additional trial was done in Symengine as to utilize more computationally efficient platforms.

Thus the process of finding a solution manually together with validation via symbolic manipulation in SymPy endured.
Below the solutions for both the non-inverted and the inverted case are presented.

A.1.1 Closed form analytical I.K solution for the non-inverted case

This section presents the I.K solution for an arm with the Denavit Hartenberg parameters showcased below:

We start with the matrix equation (you can refer to it)

1. Divide both sides of the equation by t_1^{-1}

$$t_1^{-1} S = t_1^{-1} t_1^5 \tag{A.4}$$

$$\begin{bmatrix} n_x c_1 + n_y s_1 & o_x c_1 + o_y s_1 & a_x c_1 + a_y s_1 & p_x c_1 + p_y s_1 \\ n_z & o_z & a_z & -d_1 + p_z \\ n_x s_1 - n_y c_1 & o_x s_1 - o_y c_1 & a_x s_1 - a_y c_1 & p_x s_1 - p_y c_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_5 c_{234} & -s_5 c_{234} & -s_{234} & a_2 c_2 + a_3 c_{23} - d_5 s_{234} \\ s_{234} c_5 & -s_5 s_{234} & c_{234} & a_2 s_2 + a_3 s_{23} + d_5 c_{234} \\ -s_5 & -c_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.5}$$

1. Determining the first joint angle θ_1
To find θ_1 we employ the equation(3,4), that is, we equate the elements in the third row and fourth column of both matrices. This yields the following equation.

$$p_x s_1 - p_y c_1 = 0 \tag{A.6}$$

$$\frac{s_1}{c_1} = \frac{p_y}{p_x} \tag{A.7}$$

$$\theta_1 = \arctan \frac{p_y}{p_x} \tag{A.8}$$

2. Determining the second joint angle θ_2
We start with equations(1,4) and (2,4)

$$p_x c_1 + p_y s_1 = a_2 c_2 + a_3 c_{23} - d_5 s_{234} \tag{A.9}$$

$$-d_1 + p_z = a_2 s_2 + a_3 s_{23} + d_5 c_{234} \tag{A.10}$$

Rearrange by moving the s_{234} and c_{234} to the other side and use the trigonometric expressions $s_{23} = s_a c_b + c_a s_b$ and $c_{23} = c_a c_b - s_a s_b$ for the terms on the right hand side

$$p_x c_1 + p_y s_1 + d_5 s_{234} = a_2 c_2 + a_3 c_2 c_3 - a_3 s_2 s_3 \tag{A.11}$$
$$-d_1 + p_z - d_5 c_{234} = a_2 s_2 + a_3 s_2 c_3 + a_3 s_3 c_2 \tag{A.12}$$

Factor the elements of the desired angle we want to find, c_2 and s_2

$$p_x c_1 + p_y s_1 + d_5 s_{234} = c_2 (a_2 + a_3 c_3) - a_3 s_2 s_3 \tag{A.13}$$
$$-d_1 + p_z - d_5 c_{234} = s_2 (a_2 + a_3 c_3) + a_3 s_3 c_2 \tag{A.14}$$

solving for c_2 and s_2

$$\frac{p_x c_1 + p_y s_1 + d_5 s_{234} + a_3 s_2 s_3}{a_2 + a_3 c_3} = c_2 \tag{A.15}$$
$$\frac{-d_1 + p_z - d_5 c_{234} - a_3 s_3 c_2}{a_2 + a_3 c_3} = s_2 \tag{A.16}$$

Plug c_2 in the equation for s_2

$$\frac{-d_1 + p_z - d_5 c_{234} - a_3 s_3 \frac{p_x c_1 + p_y s_1 + d_5 s_{234} + a_3 s_2 s_3}{a_2 + a_3 c_3}}{a_2 + a_3 c_3} = s_2 \tag{A.17}$$

Multiply up and down by $(a_2 + a_3 c_3)$ and solve for s_2

$$\frac{(a_2 + a_3 c_3)(-d_1 + p_z - d_5 c_{234}) - a_3 s_3 (p_x c_1 + p_y s_1 + d_5 s_{234})}{(a_2 + a_3 c_3)^2 + a_3^2 s_3^2} = s_2 \tag{A.18}$$

Realize that the term s_{234} can be extracted from equation(1,3)

$$s_{234} = -(a_x c_1 + a_y s_1) \tag{A.19}$$

3. Determining the third joint angle θ_3 Use equation(1,4) and eqn(2,4). Isolate both sides, square them up and add them.

$$p_x c_1 + p_y s_1 = a_2 c_2 + a_3 c_{23} - d_5 s_{234} \tag{A.20}$$
$$-d_1 + p_z = a_2 s_2 + a_3 s_{23} + d_5 c_{234} \tag{A.21}$$

rearrange to:

$$p_x c_1 + p_y s_1 + d_5 s_{234} = a_2 c_2 + a_3 c_{23} \tag{A.22}$$
$$-d_1 + p_z - d_5 c_{234} = a_2 s_2 + a_3 s_{23} \tag{A.23}$$

square and add up:

$$(p_x c_1 + p_y s_1 + d_5 s_{234})^2 + (-d_1 + p_z - d_5 c_{234})^2 = (a_2 c_2 + a_3 c_{23})^2 + (a_2 s_2 + a_3 s_{23})^2 \tag{A.24}$$

Realize that very beautifully

$$(a_2 c_2 + a_3 c_{23})^2 + (a_2 s_2 + a_3 s_{23})^2 = a_2^2 + 2 a_2 a_3 c_3 + a_3^2 \tag{A.25}$$

and thus

$$(p_x c_1 + p_y s_1 + d_5 s_{234})^2 + (-d_1 + p_z - d_5 c_{234})^2 = a_2^2 + 2a_2 a_3 c_3 + a_3^2 \tag{A.26}$$

isolating c_3

$$c_3 = \frac{(p_x c_1 + p_y s_1 + d_5 s_{234})^2 + (-d_1 + p_z - d_5 c_{234})^2 - a_2^2 - a_3^2}{2a_2 a_3} \tag{A.27}$$

Remember an expression for s_{234} was developed in A.19 and realize the value of c_{234} can be extracted from equation(2,3) $c_{234} = a_z$

$$c_{234} = a_z \tag{A.28}$$

4. Determining the fourth joint angle θ_4
Since the solutions for θ_2 and θ_3 are known, θ_4 can be extracted θ_{234}

$$\theta_4 = \theta_{234} - \theta_3 - \theta_2 \tag{A.29}$$

The values of s_{234} and c_{234} are expressed in A.19 and A.28 respectively. Substituting their values in the expression for s_{234}

$$\theta_{234} = \arctan\left(\frac{s_{234}}{c_{234}}\right) = \arctan\left(\frac{-(a_x c_1 + a_y s_1)}{a_z}\right) \tag{A.30}$$

$$\theta_4 = \arctan\left(\frac{s_{234}}{c_{234}}\right) = \arctan\left(\frac{-(a_x c_1 + a_y s_1)}{a_z}\right) - \theta_3 - \theta_2 \tag{A.31}$$

5. Determining the fifth joint angle θ_5 Finding θ_5. From equation(3,1) and equation(3,2)

$$-s_5 = n_x s_1 - n_y c_1 \tag{A.32}$$
$$-c_5 = o_x s_1 - o_y c_1 \tag{A.33}$$

$$\theta_5 = \arctan\frac{s_5}{c_5} = \arctan\frac{-n_x s_1 + n_y c_1}{-o_x s_1 + o_y c_1} \tag{A.34}$$

A.2 Inverted case forwards and inverse kinematics

A.2.1 Closed form analytical I.K solution for the inverted case

The solution process is very similar, however, we start from slightly different D.H. parameters resulting in some differences in the solution. For the shake of completeness and documentation for future work the derivation is included here

$$\begin{bmatrix} n_x c_1 + n_y s_1 & o_x c_1 + o_y s_1 & a_x c_1 + a_y s_1 & p_x c_1 + p_y s_1 \\ -n_z & -o_z & -a_z & d_1 - p_z \\ -n_x s_1 + n_y c_1 & -o_x s_1 + o_y c_1 & -a_x s_1 + a_y c_1 & -p_x s_1 + p_y c_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_5 c_{234} & -s_5 c_{234} & s_{234} & a_2 c_2 + a_3 c_{23} + d_5 s_{234} \\ s_{234} c_5 & -s_5 s_{234} & -c_{234} & a_2 s_2 + a_3 s_{23} - d_5 c_{234} \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.35}$$

1. Determining the first joint angle θ_1
In the same manner as before, in order to find θ_1 we employ the equation(3,4), that is, we equate the elements in the third row and fourth column of both matrices. This yields the following equation.

$$-p_x s_1 + p_y c_1 = 0 \tag{A.36}$$
$$\frac{s_1}{c_1} = \frac{p_y}{p_x} \tag{A.37}$$
$$\theta_1 = \arctan\frac{p_y}{p_x} \tag{A.38}$$

2. Determining the second joint angle θ_2
We start with equations(1,4) and (2,4)

$$p_x c_1 + p_y s_1 = a_2 c_2 + a_3 c_{23} + d_5 s_{234} \tag{A.39}$$

$$d_1 - p_z = a_2 s_2 + a_3 s_{23} - d_5 c_{234} \tag{A.40}$$

and we rearrange as before

$$p_x c_1 + p_y s_1 - d_5 s_{234} = a_2 c_2 + a_3 c_{23} \tag{A.41}$$

$$d_1 - p_z + d_5 c_{234} = a_2 s_2 + a_3 s_{23} \tag{A.42}$$

Rearrange by moving the s_{234} and c_{234} to the other side and use the trigonometric expressions $s_{23} = s_a c_b + c_a s_b$ and $c_{23} = c_a c_b - s_a s_b$ for the terms on the right hand side

$$p_x c_1 + p_y s_1 - d_5 s_{234} = a_2 c_2 + a_3 c_2 c_3 - a_3 s_2 s_3 \tag{A.43}$$

$$d_1 - p_z + d_5 c_{234} = a_2 s_2 + a_3 s_2 c_3 + a_3 s_3 c_2 \tag{A.44}$$

Factor the elements of the desired angle we want to find, c_2 and s_2

$$p_x c_1 + p_y s_1 - d_5 s_{234} = c_2 \left(a_2 + a_3 c_3\right) - a_3 s_2 s_3 \tag{A.45}$$

$$d_1 - p_z + d_5 c_{234} = s_2 \left(a_2 + a_3 c_3\right) + a_3 s_3 c_2 \tag{A.46}$$

solving for c_2 and s_2

$$\frac{p_x c_1 + p_y s_1 - d_5 s_{234} + a_3 s_2 s_3}{a_2 + a_3 c_3} = c_2 \tag{A.47}$$

$$\frac{d_1 - p_z + d_5 c_{234} - a_3 s_3 c_2}{a_2 + a_3 c_3} = s_2 \tag{A.48}$$

Plug c_2 in the equation for s_2

$$\frac{d_1 - p_z + d_5 c_{234} - a_3 s_3 \frac{p_x c_1 + p_y s_1 - d_5 s_{234} + a_3 s_2 s_3}{a_2 + a_3 c_3}}{a_2 + a_3 c_3} = s_2 \tag{A.49}$$

Multiply up and down by $(a_2 + a_3 c_3)$ and solve for s_2

$$\frac{\left(a_2 + a_3 c_3\right)\left(d_1 - p_z + d_5 c_{234}\right) - a_3 s_3 \left(p_x c_1 + p_y s_1 - d_5 s_{234}\right)}{\left(a_2 + a_3 c_3\right)^2 + a_3^2 s_3^2} = s_2 \tag{A.50}$$

Realize that the term s_{234} can be extracted from equation(1,3)

$$s_{234} = a_x c_1 + a_y s_1 \tag{A.51}$$

3. Determining the third joint angle θ_3 Use equation(1,4) and eqn(2,4). Isolate both sides, square them up and add them.

$$p_x c_1 + p_y s_1 = a_2 c_2 + a_3 c_{23} + d_5 s_{234} \tag{A.52}$$

$$d_1 - p_z = a_2 s_2 + a_3 s_{23} - d_5 c_{234} \tag{A.53}$$

rearrange to:

$$p_x c_1 + p_y s_1 - d_5 s_{234} = a_2 c_2 + a_3 c_{23} \tag{A.54}$$

$$d_1 - p_z + d_5 c_{234} = a_2 s_2 + a_3 s_{23} \tag{A.55}$$

square and add up:

$$(p_x c_1 + p_y s_1 - d_5 s_{234})^2 + (d_1 - p_z + d_5 c_{234})^2 = (a_2 c_2 + a_3 c_{23})^2 + (a_2 s_2 + a_3 s_{23})^2 \tag{A.56}$$

Realize that very beautifully

$$(a_2 c_2 + a_3 c_{23})^2 + (a_2 s_2 + a_3 s_{23})^2 = a_2^2 + 2a_2 a_3 c_3 + a_3^2 \tag{A.57}$$

and thus

$$(p_x c_1 + p_y s_1 - d_5 s_{234})^2 + (d_1 - p_z + d_5 c_{234})^2 = a_2^2 + 2a_2 a_3 c_3 + a_3^2 \tag{A.58}$$

isolating c_3

$$c_3 = \frac{(p_x c_1 + p_y s_1 - d_5 s_{234})^2 + (d_1 - p_z + d_5 c_{234})^2 - a_2^2 - a_3^2}{2a_2 a_3} \tag{A.59}$$

Remember an expression for s_{234} was developed in A.51 and realize the value of c_{234} can be extracted from equation(2,3).

$$c_{234} = a_z \tag{A.60}$$

4. Determining the fourth joint angle θ_4
Since the solutions for θ_2 and θ_3 are known, θ_4 can be extracted θ_{234}

$$\theta_4 = \theta_{234} - \theta_3 - \theta_2 \tag{A.61}$$

The values of s_{234} and c_{234} are expressed in A.51 and A.60 respectively. Substituting their values in the expression for s_{234}

$$\theta_{234} = \arctan\left(\frac{s_{234}}{c_{234}}\right) = \arctan\left(\frac{a_x c_1 + a_y s_1}{a_z}\right) \tag{A.62}$$

$$\theta_4 = \arctan\left(\frac{s_{234}}{c_{234}}\right) = \arctan\left(\frac{a_x c_1 + a_y s_1}{a_z}\right) - \theta_3 - \theta_2 \tag{A.63}$$

5. Determining the fifth joint angle θ_5 Finding θ_5. From equation(3,1) and equation(3,2)

$$s_5 = -n_x s_1 + n_y c_1 \tag{A.64}$$

$$c_5 = -o_x s_1 + o_y c_1 \tag{A.65}$$

$$\theta_5 = \arctan\frac{s_5}{c_5} = \arctan\frac{-n_x s_1 + n_y c_1}{-o_x s_1 + o_y c_1} \tag{A.66}$$

As evidenced in the solution that was found, the robot has 5 degrees of freedom, meaning that not all poses of the 6 degrees of freedom space are possible and valid, however, this inverse kinematics formulation does not distinguish those that are from those that are not. For this reason, a pre-check limit to the input space was added.

A.2.2 Limitis in the input space

The input to the inverse kinematics mathematical equations is the 4x4 transformation matrix S. This matrix represents a 3D translation and rotation in all 6 degrees of freedom.

$$S = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.67}$$

However, the developed robot, as evidenced throughout this chapter is limited to 5DOFs, this means that it is possible to input transformation matrices that have no inverse kinematics solution as the robot is physically incapable of performing them. When such transformations are used as input to the found I.K equations, the solution will be wrong both in position and orientation due to the solution equations not having been developed for such scenarios.

The implementation allows the user to handle this situation in 3 different manners. For details on the implementation visit the I.K function documentation

1. **Move to the closest valid solution:** In this mode, a solution that satisfies the position and tries to achieve the closest attainable orientation is found. This is done by rotating the input by the smallest rotation possible which will translate it into the solution space.
   The robot is limited to gripper/screwdriver-pointing vectors that exist in the plane defined by the base node and the target node. This gripper vector must align with the positive y-axis (transform.up) of the target node, which is the vector of the screw-insert thread. The valid inputs are those with that vector in the aforementioned plane.
   In this mode we check the angle between the vector and its projection in the plane and rotate its transformation matrix by said amount, thus changing the input to the closest valid one.
2. **Raise an error:** if the object to be grabbed is not in the solution space do not attempt to find a solution and instead raise an error. A tolerance can be added to relax this criterion.
3. **Dont handle this scenario:** Dont do any special behaviour and solve the I.K normally.

The implementation defaults to Raising an error for large differences and moving to the closest valid solution for smaller differences. Check the documentation for details on the implementation

B

Appendix: Assembler robot bill of materials

	Item	Quantity	Store	Unity price (eur, exc VAT)	Sub total cost
Bearings	6704 20x27x4mm	4	kugellager-express	2.94	11.76
	6703 17x23x4mm	4		2.46	9.84
	6701 12x18x4mm	4		1.14	4.56
	MR105 5x10x4	2		1.25	2.5
	U604 U groove	8	123-3D	1.24	9.92
	U603 U groove	8	Aliexpress	1.29	10.32
Motors	LSS-HT1	5	RobotShop	96.08	480.40
	N20 motor 12V 298:1 (12Gan20-298)	2	Aleixpress	1.75	3.5
Electronics	LSS-2MC	2	RobotShop	27.57	55.14
	LSS - 100mm Serial Cable	3	RobotShop	0.69	2.07
	Custom controller board	1	JLC PCB		
	Arduino nano every	1	Arduino	12.5	12.5
Dowel pins	Dowel pin M3x10	8	Aliexpress	0.83 for bag of 10	0.67
	Dowel pin M4x14	8	Aliexpress	1.05 for bag of 10	0.84
Inserts	M3x3 heat set inserts	64	CNC kitchen	7.15 for a bag of 100	4.57
	M3x5.7 heat set inserts	8	CNC kitchen	7.55 for a bag of 100	0.60
Set screw	M3x4 set screw	24	JRV steel	7.46 for bag of 500	0.36
Steel cable	1mm OD 7x7 49 strands steel cable	4	JRV steel	12.98 for 100m	0.52
				Total	610.57

C

Developed battery safety guidelines

Mini assembly robot operation safety checklist

Operator: Guillermo Presa **Test Date: from approval date to TBD/2023**

Description and scope:

To operate a miniature (<1Kg total mass) robotic assembler, lithium batteries of 11wh are employed. This document stablishes the protocols for safely working with such batteries.

Charging and operation safety		
Yes/No	**Measure**	**Reason**
☑	Charge with COTS, load balancing charger	To reduce charging risks a commercial charger that balances cell voltage is used. See Appendix A for setup reference
☑	Only COTS components shall directly interface with the battery. No self-made BMS….	Prevent damage to the battery
☑	Electronics that interface with the battery shall be enclosure	Prevent short circuits or damage to the electronics that could affect the battery. See Appendix A for setup reference
☑	Charging is only performed with human supervision. Cells can't not be left unattended during charging	A human can provide a fast response to any situations that develop.
☑	Charging/ operation must be stopped if signs of puffin or other damages to the cell appear.	See the emergency scenarios for more
☑	Charging and operation shall be performed in proximity to a sand/ extover expanded glass …. box	See Appendix A for setup reference
☑	Charging shall be performed in a safe box identifying and respecting the maximum capacity limits of the box utilized	See Appendix A for setup reference

☑	Identify nearby emergency Button	Press in case of uncontrollable fire and/or contamination.
☑	Identify emergency Number	Call in case of life-threatening situations
☑	Ensure no flammables are present on charging or operation area.	Remove paper, textile, flammable chemicals, and other flammable substances from the testing area as much as possible.

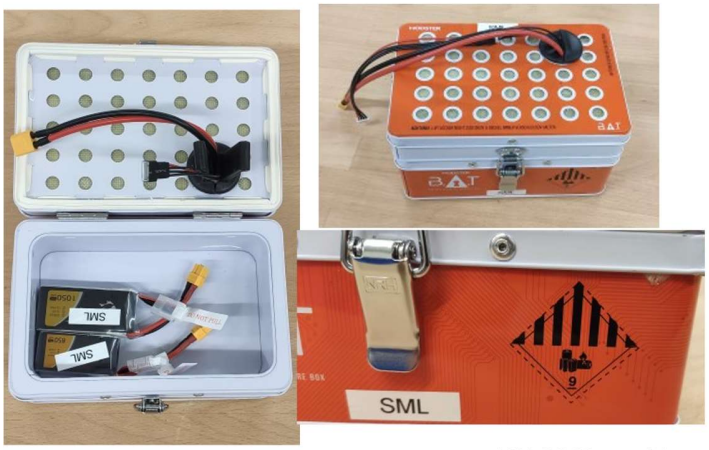
Storage safety		
Yes/No	**Measure**	**Reason**
☑	Prolonged storage shall occur in a designated battery safe cabinet. (MAV lab LIPO cabinet)	Prolonged storage shall be performed in a safe location, that being the lipo cabinet in the MAV lab
☑	Storage for prolonged times shall not be performed with a fully loaded cell. Limit max voltage or state of charge in accordance with the rules in the designated cabinet (below 3.8V in this case)	To limit energy and potential damage, drain battery charge before prolonged storage
☑	Box for prolonged storage shall be marked accordingly (UN 3480)	See Appendix A setup reference with UN 3480 marking.
☑	Ensure no flammables near storage area	

Emergency Scenarios	
Incident	**Response**
Battery shows strong visual damage, such as being punctured, cut, heavily puffed, smoke release or other signs of imminent danger.	Stop the test/ charging, grab the robot by locations at a safe distance from the battery and insert in the sand bucket, ensuring the battery side is submerged in the sand.

Battery shows minor puffing	Stop the test/ charging, ensure no flammable objects (paper, textile, chemicals etc…) are surrounding the robot. Evaluate battery evolution, if it is safe to proceed, remove the battery from the robot and submerge in sand bucket.

Appendix A: setup reference

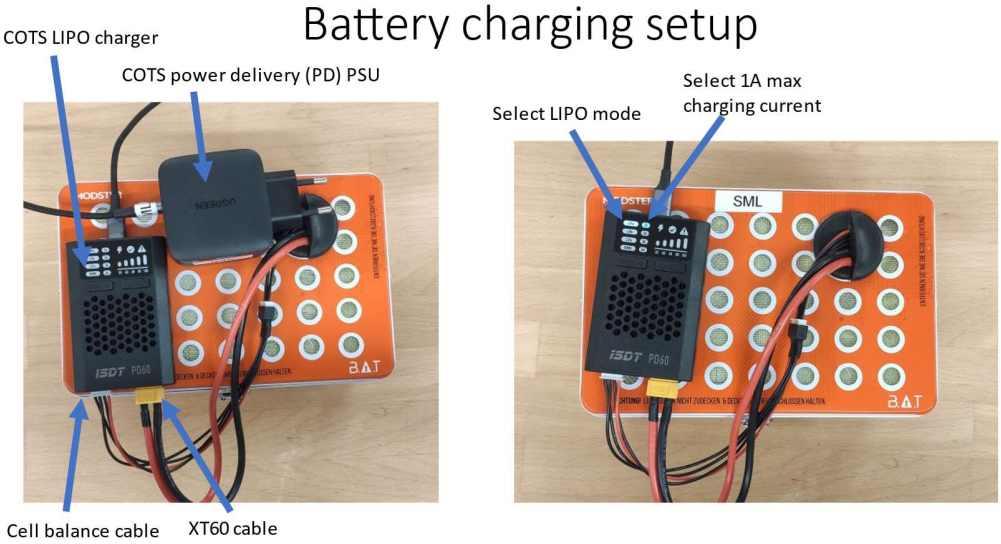
Short-term Battery storage setup



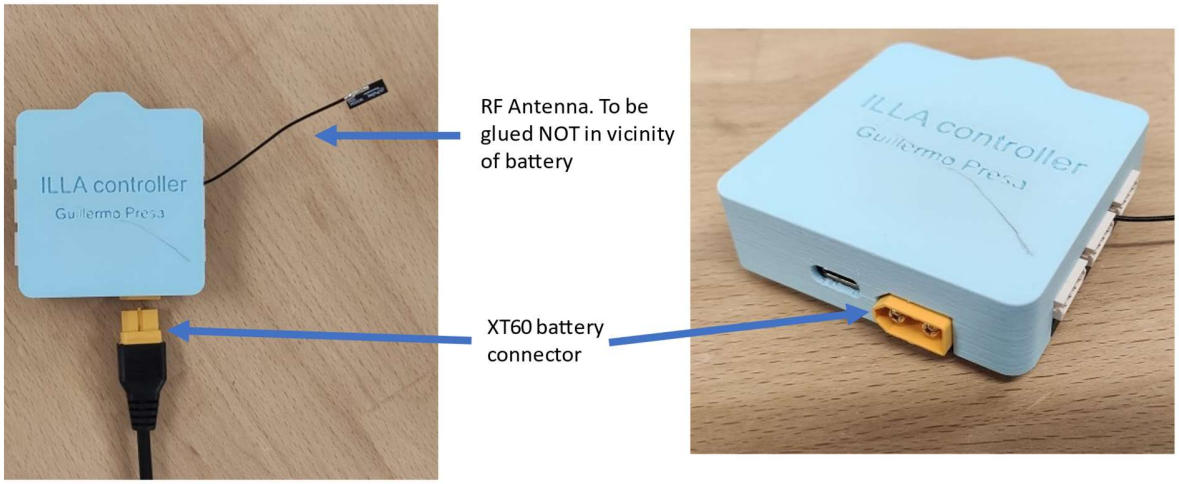
UN 3840 marking

Added clearly visible markings for the max capacity

Battery charging setup



Robot electronics enclosure



Battery fire box

Extover® Expanded glass.
(Fire Extinguisher for Lithium Battery Fires and Metal Fires)

Appendix C: Emergency number

Emergency Number

015-27 88888