Technische Universität Braunschweig

Bachelor Thesis

Motion Planning for Reconfigurable Magnetic Modular Cubes in the 2-Dimensional Special Euclidean Group

Kjell Keune

Institut für Betriebssysteme und Rechnerverbund

Supervised by
Prof. Dr. Aaron T. Becker

May 30, 2023

Statement of Originality

This thesis has been performed independently with the support of my supervisor/s. To the best of the author's knowledge, this thesis contains no material previously published or written by another person except where due reference is made in the text.

Braunschweig, May 30, 2023

Aufgabenstellung / Task Description

Deutsch: Um spezifische Aufgaben besser zu bewältigen, lassen sich modulare, rekonfigurierbare Roboter zu größeren Strukturen zusammensetzen und wieder auseinandernehmen. Magnetic-Modular-Cubes sind skalierbare Einheiten, bei welchen Permanentmagneten in einen würfelförmigen Körper eingebettet sind. Diese Einheiten zählen als rekonfigurierbare Roboter, obwohl sie selber keine Logik oder Stromversorgung beinhalten. Stattdessen lassen sich diese durch ein externes, gleichmäßiges und sich zeitlich änderndes Magnetfeld steuern. Durch diese Steuerung können die Roboter auf der Stelle gedreht oder durch Pivotwalking nach rechts und links bewegt werden.

Obwohl sich das Magnetfeld auf alle Einheiten gleichermaßen auswirkt, kann durch Kollision mit der Arbeitsflächenbegrenzung eine Änderung der Anordnung bewirkt werden. Befinden sich zwei Roboter nah genug beieinander, können sich diese durch die Permanentmagneten miteinander verbinden und so Polyominoes als größere Strukturen aufbauen, welche auf die gleiche Weise wie einzelne Roboter gesteuert werden können. Polyominoes bewegen sich mit unterschiedlicher Geschwindigkeit in unterschiedliche Richtungen, abhängig von deren Form. Frühere Arbeiten betrachteten das Tilt-Model, bei welchem sich Strukturen jeder Größe mit gleicher Geschwindigkeit in ganzzahligen Schritten und mit ausschließen 90° Drehungen bewegen lassen.

Herr Keunes Aufgabe in dieser Bachelorarbeit ist es, einen Motionplanner für die beschriebenen Magnetic-Modular-Cubes zu entwerfen, welcher mit beliebigen Positionen und Rotationen umgehen kann. Dabei ist es erforderlich, eine Simulationsumgebung zu schaffen, welche das Verhalten der Roboter repliziert. Es soll ein lokaler Motionplanner entwickelt werden, um zwei Polyominoes an gewünschten Kanten zu verbinden. Dieser Localplanner soll Heuristiken für Bewegungsabläufe mit möglichst wenig Schritten realisieren. Ebenfalls soll dieser global eingesetzt werden, um Bewegungsabläufe zu finden, die gewünschte Polyominoes aus einer zufällig gegebenen Startkonfiguration erzeugen. Ein interessantes Ergebnis wird es sein, zu sehen, wie gut Probleminstanzen dieser Art in der Realität gelöst werden können und welche Parameter die gravierendsten Auswirkungen auf die Schwierigkeit von Motionplanning-Problemen haben.

English: Reconfigurable modular robots can dynamically assemble/disassemble to better accomplish a desired task. Magnetic modular cubes are scalable modular subunits with embedded permanent magnets in a 3D-printed cubic body. These cubes can act as reconfigurable modular robots, even though they contain no power, actuation or computing. Instead, these cubes can be wirelessly controlled by an external, uniform, time-varying magnetic field. This control allows the cubes to spin in place or pivot walk to the left or right local coordinate frame.

Although the applied magnetic field is the same for each magnetic modular cube, collisions with workspace boundaries can be used to rearrange the cubes. Moreover, the cubes magnetically self-assemble when brought in close proximity of another cube, and form polyominoes, which can be controlled the same way as single cubes. These polyominoes pivot walk at speeds and angle offsets that are a function of the structures shape. Related work has considered the "tilt model," where similar cubes and polyominoes move between integer positions, all move at the same speed, and only rotate by 90 degree steps.

In his thesis, Mr. Keune's task is to design a motion planner for magnetic cubes that can assume arbitrary positions and orientations in the workspace. This requires designing a simulation environment that replicates the behavior of magnetic cubes. He will design local planners for moving two polyominoes to assemble at desired faces. Designing the local planner includes heuristics that minimize the number of steps. The local planner will be used to search for global planning sequences to generate desired polyominoes from a given starting configuration. One exciting outcome will be studying how well instances can be solved in practice and analyzing which parameters have the most significant effect on the difficulty of the motion planning problem.

Abstract

In this thesis we developed a heuristic approach for the motion planning problem of assembling structures with magnetic modular cubes, developed and researched by Bhattacharjee et al. [5], in the 2-dimensional special Euclidean group, the space of rigid movements in a 2-dimensional plane. Magnetic modular cubes are cube-shaped bodies with embedded permanent magnets uniformly controlled by a global time varying magnetic field surrounding the workspace.

A 2D physics simulator is use to simulate global control and the resulting continuous movement of magnetic modular cube structures as well as magnetic attraction and repulsion, while detecting and resolving collision. The simulator allows closed-loop control algorithms for planning the connection of two structures at desired faces. These developed sequences of movements, called local plans, will be used on a global scale to plan the assembly of specified target structures in a rectangular workspace with no internal obstacles. The assembly is done by generating a set of building instructions for a target structure represented as a graph that will be traverse in a depth-first-search approach by applying local plans to current states of the workspace.

We analyze how target structures of varying sizes and shapes in different rectangular workspaces affect planning time and the rotational-cost of movements. The traversal of the building instruction graph can be further optimized, for which we present three strategies and their effect on the performance of the global planner. The majority of randomly created instances in our experiments can be solved in under 200 seconds for structures of up to 12 cubes, but certain attributes of target structures can decrease efficiency of the global planner drastically.

Contents

List of Figures

List of Variables

$\mathcal{A}$, $\mathcal{B}$, $\mathcal{T}$	Calligraphic letters represent polyominoes. Letters $\mathcal{A}$ and $\mathcal{B}$ are given to polyominoes that are about to be connected in a local plan. $\mathcal{T}$ indicates the target polyomino for assembly in a global plan.
c, $c_{\mathcal{A}}$	Magnetic modular cube. $c_{\mathcal{A}}$ would indicate that c is part of the polyomino $\mathcal{A}$.
r_C	The cube radius is the half length of a cube face. All cubes in a workspace are the same size.
r_M	The magnet radius is the distance from the center of the cube to the center of a embedded permanent magnet.
m_C	Mass of a magnetic modular cube.
p_c, $p_{\mathcal{A}}$	Workspace position of a cube c or a polyomino $\mathcal{A}$. In both cases position is the center of mass.
$r_{c_{\mathcal{A}}}$	Vector pointing from the polyomino's center of mass $p_{\mathcal{A}}$ to the cube's center of mass $p_{c_{\mathcal{A}}}$.
$d(c_1, c_2)$	Euclidean distance between the centers p_{c_1} and p_{c_2} of the cubes c_1 and c_2.
$\vec{N}$, $\vec{E}$, $\vec{S}$, $\vec{W}$	Cardinal direction vectors dependent on the longitude orientation of the global magnetic field.
e, $e_{\mathcal{A}}$	Side face of a magnetic modular cube represented by a vector. $e \in \{\vec{N}, \vec{E}, \vec{S}, \vec{W}\}$ due to the assumption of cubes being always aligned with the magnetic field. $\|e\| = 1$ holds true and $e_{\mathcal{A}}$ indicates that e belongs to a cube contained in polyomino $\mathcal{A}$.
n	Size of the target polyomino or number of cubes in the workspace. In case of our global planner cube count equals target polyomino size.
n_{red}, n_{blue}	Number of red or blue cubes within a polyomino.
$\vec{d}$	Displacement vector for one pivot walking cycle of a polyomino.
$\vec{a}$	Pivot walking axis of a polyomino in global coordinate frame. Vector between north and south pivot point.

α	Pivot walking angle.
$\vec{w}$	Pivot walking direction $\vec{w} \in \{\vec{E}, \vec{W}\}$.
$\vec{m}$	Slide-in direction $\vec{m} \in \{\vec{E}, \vec{W}\}$.
$\overrightarrow{\mathcal{AB}}$	Vector used in the process of aligning cubes. Points from $p_{c_\mathcal{A}}$ to $p_{c_\mathcal{B}}$ for straight aligning, or to a position above/below $p_{c_\mathcal{B}}$ for offset aligning.
d_{offset}	Offset distance for offset aligning. $d_{offset} > 2r_C$.
β	The rotation angle is a change in longitude orientation of the global magnetic field.
$\mathbf{R}_\beta$	2×2 rotation matrix for rotating vectors by an angle of β.
$\#steps$	Number of estimated pivot walking cycles in our dynamic walk-align-realign approach.
s	Plan state of either local or global plans. States if successful, or the reason of failure.
A	Sequence of actions $a_1, ..., a_k$ a local plan consists of.
g, g_{init}, g_{goal}	Configurations of the configuration-space $SE(2)$. g_{init} indicates the initial and g_{goal} the goal configuration of local or global plan.
S, $S(g)$, $S_\mathcal{T}$, $S_{trivial}$	Polyomino sets store information about the polyomino types present in the workspace without considering position or distinguishing between physical polyominoes. The amount of one type is also stored. $S(g)$ is the polyomino set of a configuration g. $S_\mathcal{T}$ contains only one occurrence of $\mathcal{T}$ and $S_{trivial}$ only trivial polyominoes. Both are used in TCSA graphs.
$\hat{n}$	Maximum polyomino size in one configuration or polyomino set.
t_c	Continuous two-cutting edge path through a polyomino.
$G_{TCSA}(\mathcal{T})$	Two-cut-sub-assembly graph of $\mathcal{T}$ represented by nodes V and edges E. Nodes are polyomino sets and edges connect two sets $\{S_0, t_c, S_1\}$ with a two-cut as an edge weight.
$L_\mathcal{A}$	Collection of all physically distinct polyominoes of the polyomino type $\mathcal{A}$.
O	List of connection options o for one configuration determined with a TCSA graph.
$\hat{o}(o_1, o_2)$	Function comparing connection options o_1 and o_2 and returning the better one based on the option sorting strategy used.

P	Plan stack containing a continuous sequence of local plans p. Used in the global planning algorithm.
$\#local$	Number of local plans simulated during planning with the global planning algorithm.
$\#config$	Number of configurations explored during planning with the global planning algorithm.
μ_{mag}	Magnetic strength of embedded permanent magnets of magnetic modular cubes use in our simulator.
μ_{field}	Strength of the magnetic field used in our simulator.
p_{fric}	Friction point of a cube depending on the latitude of the magnetic field. Either at the position of north or south magnet.
n_{fric}	Number of friction-cubes of a polyomino to which friction force is applied.
w_{nom}	Fraction of nominal friction that gets applied to all cubes of a polyomino.

1 Introduction

Self-assembling modular parts forming bigger structures is a well-known concept in nature. Most functionalities of living organisms follow this principle [6]. DNA, for example, has the ability to self-replicate by using differently shaped proteins that combine themselves in various ways. At larger sizes, these cells can be combined to assemble tissue, organs and even whole organisms. Complex structures, like proteins, can be assembled and disassembled depending on the task they should accomplish at a given point in time. Using self-reconfiguring robot swarms in such a way has promising applications in the future. Biomedical applications could be targeted drug delivery or drug screening [21], or a robot swarm could be used for milliscale and microscale manufacturing [17].

Designing robots at these small sizes faces challenging problems. Equipping each robot with its own sensors, actuation-system, connection-system and power supply seem infeasible, in terms of the miniaturization required and power-limitations [22].

Therefore, the use of external global control, effecting every robot in the workspace with the same torque and force, is a promising solution [22]. Using robots with embedded permanent magnets, has all the desired effects. Robots can be controlled by an external magnetic field and also connect to each other without any internal power supply and for sensing an external camera can be used [18].

One example for magnetically controlled robots are the magnetic modular cubes by Bhattacharjee et al. [5], which are the subjects of this thesis. We will look at the difficulties and problems that occur, when assembling structures with magnetic modular cubes in the 2-dimensional special Euclidean group $SE(2)$, the space of rigid movements in a 2-dimensional plane.

1.1 Related Work

Continuous motion planning is a crucial subject in the field of robotics. The goal is to find a path from the initial state of a robot to a desired goal state, by performing actions which the robot is capable of. The movement may result in collision with static obstacles and with other robots, but the objects may not overlap. The state of the system is also called a configuration. All possible configurations one or multiple robots can be in is defined as the configuration-space. Motion planning complexity is often exponential in the dimension of the configuration space [10]. Increasing the number of robots and/or possible actions, increases the dimension of the configuration space. It is difficult to engineer algorithms that explore these huge configuration-spaces and provide continuous path from the initial to the goal configuration, or report failure, if the goal is

not reachable. Decades of research has been done on motion planning. The textbooks [10] and [16] offer a great overview and also explain a lot of important concepts in detail.

When working with configuration-spaces that are uncountable infinite, like the special Euclidean group, one concept that has been successful for many robotics problem is sample-based motion planning.

By taking samples, you can reduce the planning problem from navigating a configuration space to planning on a graph, but you might lose possible solutions. Algorithms like that are not complete anymore, but by using a good sampling technique you can get arbitrarily close to any point. Ways of sampling include random sampling being probabilistically complete or using a grid with a resolution that is dynamically adjustable resulting in resolution completeness. After sampling, conventional discrete planning algorithms can be applied [10].

One state-of-the-art sampling-based approach uses rapidly-exploring random trees (RRT). This method tries to grow a tree-shaped graph in the configuration space by moving into the direction of randomly chosen samples from already explored configurations. That way the space gets explored uniformly without being too fixated on the goal configuration [11, 12].

When working with multiple robots, the interaction of robots with each other becomes important. One interesting idea is that single robots can connect to form bigger structures. This is referred to as self-assembly and E. Winfree [23] proposed the abstract Tile Assembly Model (aTAM) in the context of assembling DNA. In this model, particles can have different sets of glues and connect according to certain rules regarding the glue type. However, he considers this process as nondeterministic, so there is no exact instruction on how to assemble a desired structure.

One model more related to the magnetic modular cubes is the Tilt model from Becker et al. [2]. In the Tilt model, all tiles move into one of the cardinal directions until hitting an obstacle. Different variations of the model include moving everything only one step, or the maximally possible amount. It offers a solution for motion planning problems when robots are controlled uniformly by external global control inputs.

In [2] it is shown that transforming one configuration into another, known as the reconfiguration-problem, is NP-hard. Caballero et al. [8] also researched complexity of problems regarding the Tilt model. Following work [3] also proves that finding an optimal control sequence, minimizing the number of actions, for the configuration-problem is PSPACE-complete. Furthermore, research is done on designing environments in which the Tilt model can be used to accomplish certain tasks. In particular, Becker et al. [3] create connected logic gates that can evaluate logical expressions.

More on the side of self-assembly, in [4] the construction of desired shapes using the tilt model is researched. It presents a method that can determine a building sequence for a polyomino by adding one tile at a time, considering the rules of Tilt. Also examined are ways of modifying the environment to create factories that construct shapes in a pipeline by repeating the same global control inputs. Shapes can be constructed more efficiently by combining multi-tiled shapes to an even bigger structure. One article considering the construction with so-called sub-assemblies is proposed by A. Schmidt [20].

Most recently, Bhattacharjee et al. [5] developed the magnetic modular cubes. These robots contain embedded permanent magnets and have no computation or power supply. Instead, they are controlled by an external time-varying magnetic field and are able to perform various actions. Most importantly, they can rotate in place or use a technique called pivot walking to move either left or right. The magnets also act as glues and allow the cubes to perform self-assembly. Although it is theoretically possible to assemble 3-dimensional structures, most research was done by only connecting cubes in two dimensions. Since all cubes are the same size, the assembled 2-dimensional shapes can be represented as polyominoes. An enumeration was done on the amount of possible polyominoes that can be created by cubes with different magnet configurations [14].

By limiting the controls to only 90 degree turns and assuming a uniform pivot walking distance for all structures per step, magnetic modular cubes follow rules similar to the Tilt model. Following these limitations, a simple discrete open-loop motion planer was developed, that explores a finite configuration-space and lists all the possible polyominoes that can be created from an initial configuration [5].

One interesting paper from Blumenberg et al. [7] explores the assembly of specific target polyominoes in arbitrary environments, when cubes obey the tilt model in a discrete setting. He provides different algorithmic approaches using various distance heuristics and a solution making use of RRTs.

Lu et al. [15] are working on establishing closed-loop control for magnetic modular cubes by using computer vision-based feedback of the workspace. The used motion planner is still working in a discrete setting, but can assembly specific target shapes, while handling collision events of estimated continuous motion.

1.2 Contribution

We provide more information about the general framework of magnetic modular cubes on which this thesis relies in Chapter 2.

In Chapter 3 we develop a local planner finds control sequences that connect two magnetic modular cube structures at desired faces. We do this with a closed-loop algorithm to account for all types of collision events. The resulting local plans are not optimal, but follow heuristics for minimizing the cost of movements. The local planner prioritizes reducing planning time.

The local planner works with our magnetic modular cube simulator presented in Chapter 5. This 2D physics simulator replicates the behavior of magnetic modular cubes accurately, while still being efficient enough to be used for motion planning. The simulator does not assume discrete movement or limit rotations to a certain amount.

Based on the local planner we develop a global planning algorithm in Chapter 4, which provides a control sequence to assemble desired target structures. The configuration-space is sampled by only considering connections between polyominoes as local plans and using a specially constructed graph as a building instruction for target polyominoes. The use of RRTs would be too inefficient, since we are working with a high fidelity simulation.

Results evaluating our global planner are presented in Chapter 6 and a conclusion with possible directions for future work is given in Chapter 7.

2 Preliminaries

This chapter introduces preliminary concepts for this thesis. Section 2.1 further describes magnetic modular cubes and Section 2.2 the surrounding workspace. Section 2.3 introduces polyominoes as magnetic modular cube structures and Section 2.4 provides insight on how the global magnetic field is used to interact with these structures in the workspace.

2.1 Magnetic Modular Cubes

The magnetic modular cubes are cube-shaped bodies embedded with permanent magnets on the four side faces. The magnets have different orientations of their north and south pole. One pole is always pointing outward and the other straight to the center of the cube. The magnet at the front face has its north pole pointing outwards and the magnet at the back its south pole. These two magnets ensure that the cube is always aligned with the global magnetic field and their orientation holds true for both cube types. The two other side faces must have the same outwards pointing pole, so that this axis does not provide a magnetic torque.

In fact, this is the reason a distinct definition of front, back and side is even possible. Since the front is always pointing to the north pole of the magnetic field, we also call it the north face, or north edge in two dimensions. All the other faces can also be called by their corresponding cardinal direction. Each face is represented by a vector $e \in \{\vec{N}, \vec{E}, \vec{S}, \vec{W}\}$ with $\|e\| = 1$ pointing in the cardinal direction of the magnetic field. For simplification we refer to magnets by their outwards pointing pole.

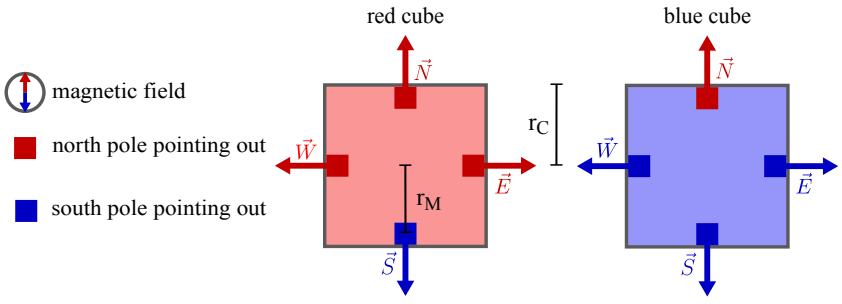


Figure 2.1: Simplified top-down view of the two magnetic modular cube types with their outward pointing magnet poles, illustrated as red and blue squares. Also visualizes the lengths r_C and r_M and the cardinal direction vectors of all cube faces.

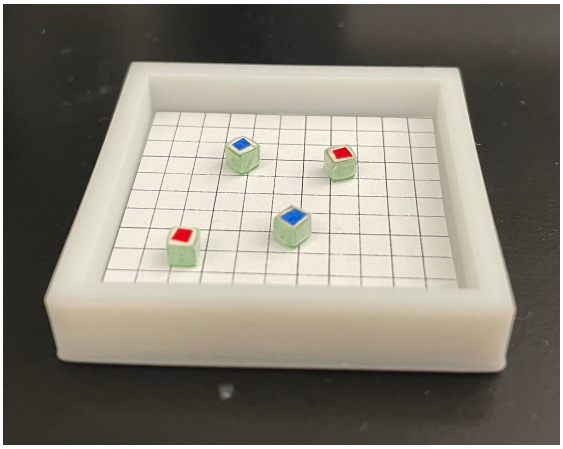


Figure 2.2: Magnetic modular cubes with 2.8 mm edges in a real workspace. Picture was provided by A. Bhattacharjee in a personal conversation.

Furthermore, two different cube types are defined: Either both side magnets point out their north pole, these cubes are called red cubes, or they point out their south pole, which is then called a blue cube. Figure 2.1 shows a top-down view of the two cube types with all the outwards pointing magnet poles. A compass always shows the orientation of the magnetic field in our illustrations.

Magnetic Modular Cubes can be constructed in different sizes and ways. For more technical details and length measurements, we refer to the original paper [5]. In Figure 2.2 magnetic modular cubes with 2.8 mm edges can be seen in a real workspace. Two important lengths are the cube radius r_C and the magnet radius r_M (also illustrated in Figure 2.1). r_C is one half-length of a cube face and r_M is the distance from the center of the cube to the center of the magnet.

2.2 Workspace and Configuration

Magnetic modular cubes could theoretically be placed and maneuvered on any 2-dimensional plane with numerous obstacles, as long as you can surround the workspace with a time varying magnetic field. The magnetic field should be able to point in any direction specified by angles of latitude and longitude, so that the cubes can operate in all desired motion modes.

Because the motion planning problem of self-assembling target shapes in the special Euclidean group is hard enough without considering obstacles and arbitrary workspace shapes, this thesis limits itself to a rectangular workspace with no internal obstacles. The workspace is bounded by surrounding walls, which are the only objects that could be considered as obstacles in classical motion planning. However, we do not assume a fixed size, as long as the workspace stays finite and rectangular.

For planning we work in the configuration-space of the 2-dimensional special Euclidean group $SE(2) = \mathbb{R}^2 \times \mathbb{S}^1$. When only considering one cube, the group consists of the

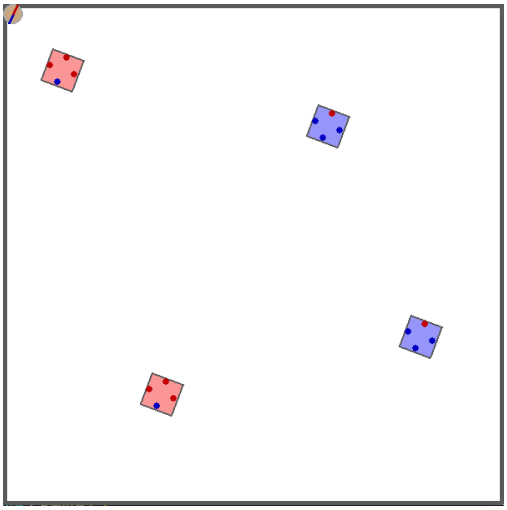


Figure 2.3: Rectangular workspace with a configuration of four magnetic modular cubes. All cubes have the same orientation as the magnetic field, indicated by the compass in the top-left corner.

position in $\mathbb{R}^2$ and an orientation $\mathbb{S} = [0, 2\pi)$ [10]. When working with n cubes, the dimension of our configuration space increases to $\mathbb{R}^{2n} \times \mathbb{S}^1$. Note that the number of cubes do not affect the orientation. Because we are working with a global magnetic field, we assume that eventually all cubes are aligned with the field. Figure 2.3 shows a configuration with four cubes in the workspace. It is irrelevant which exact physical cube is at which position as long as they are the same type, so switching the positions of the two red cubes in Figure 2.3 would lead to the same configuration as before.

2.3 Polyominoes

The embedded magnets not only align the cube with the magnetic field, they also allow cubes to self-assemble into polyominoes. Two cube faces can connect if their magnets have opposite polarities. Because of this and the alignment with the magnetic field, cubes can either be connected at north and south faces, or east and west faces if the cubes are not the same type. A *polyomino* is a set of uniformly sized cubes on a 2-dimensional grid. The grid alignment does not hold true for multiple polyominoes in the workspace, because we work with arbitrary positions and orientations, but for each polyomino on its own the cubes can be represented in a local coordinate system with position (x, y), $x, y \in \mathbb{Z}$ [14].

We consider *fixed polyominoes*, meaning that two polyominoes are distinct if their shape or orientation are different [14]. The magnetic field always provides an orientation, so in Figure 2.4 a) and d) the polyominoes are equal, just the magnetic field is rotated. Conversely, the polyominoes in Figure 2.4 a) and c) are the same shape with a different rotation under the same magnetic field orientation, so they are not equal. Furthermore,

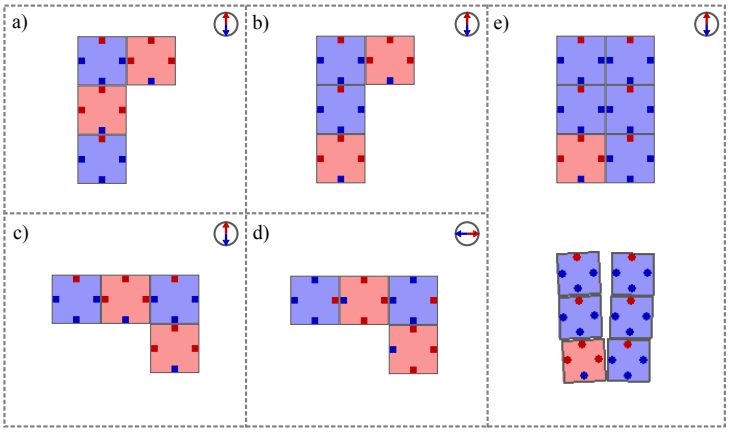


Figure 2.4: Examples of polyominoes and their equality. a) and d) are equal, only the magnetic field changed its orientation. a) and c) are not equal, they have the same shape, but it is rotated. a) and b) are also not equal because of different cube types in the same shape. e) shows an invalid polyomino in its grid representation (top) and how it behaves in the simulator (bottom).

two polyominoes are only equal if all cubes at equal grid positions are the same type. The polyominoes in Figure 2.4 a) and b) are not equal because the cube types differ. It is possible that a workspace contains multiple equal polyominoes. In that case, we refer to them as being the same polyomino-type, instead of calling them equal, since it is important to differentiate between physical polyominoes with different positions.

The size of a polyomino is the number of cubes it consists of. Because it is easier to view all structures in the workspace as a polyomino, single cubes are referred to as trivial polyominoes with size 1. Although it is not possible to connect cubes of same type at east and west faces, the magnetic modular cubes can assemble structures like the one shown in Figure 2.4 e). The connection of the bottom two cubes is strong enough to hold the structure together, even though the four blue cubes on the top repel each other. The resulting polyomino in its grid representation has two east-west connections between equal cube types and is therefor marked as an invalid polyomino.

2.4 Motion Modes

In [5] three motion modes are presented. Rotating, pivot walking, and rolling.

If the magnetic field orientation lays in the plane of the workspace and rotates without any inclination the rotation is performed around the center of mass for all polyominoes and this motion is considered a normal rotation.

When rotating the magnetic field perpendicular to the workspace plane, cubes can roll forwards or backwards. At certain steps of the rolling motion the cubes top and bottom face become side faces, which is problematic for assembly, since these faces contain no

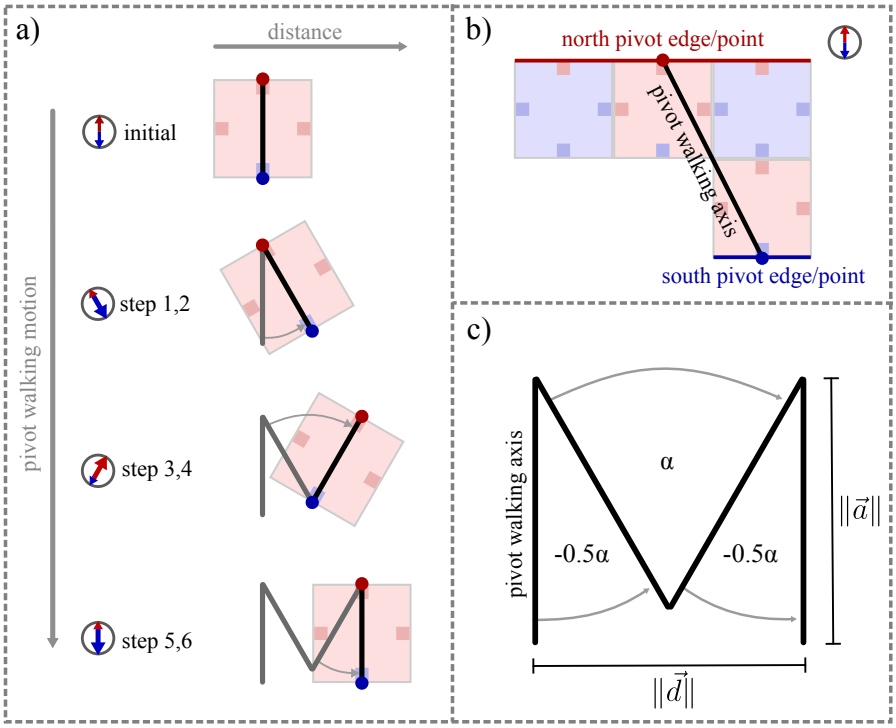


Figure 2.5: Illustration of the pivot walking motion in detail. a) shows the six pivot walking steps for a single red cube. You can see the orientation of the magnetic field (bigger arrow indicates elevation, so in step 1,2 the south pole is lifted up). In b) an example polyomino with its pivot axis, edges and points is shown. c) illustrates the rotation of the pivot axis labeled with all the pivot walking parameters.

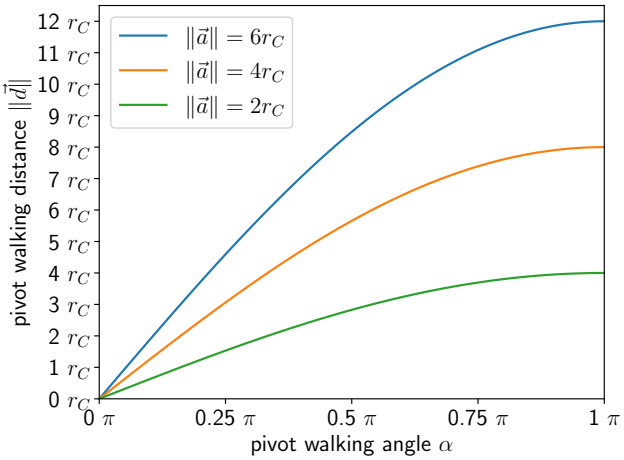


Figure 2.6: Functions of the pivot walking distance $\|\vec{d}\|$ based on pivot walking angle α for different pivot walking axes with length $\|\vec{a}\|$. Lengths are given in multiples of cube radius r_C.

magnets. Because rotation and pivot walking are sufficient to reach any position in the workspace, we do not consider rolling in our simulator and planning algorithms.

When elevating the magnetic field orientation by lifting up the south pole slightly, all polyominoes will pivot on the north face bottom edges of their most north-placed cubes. Pulling up the north pole does the opposite. The polyominoes will pivot on the south face bottom edges of their most south-placed cubes. The sum of all these cube edges is called the north or south pivot-edge and by keeping the magnetic field elevated and rotating around the normal vector of the workspace plane, the polyominoes will rotate around the center point of their pivot-edge. This point is called the north or south pivot-point. All these edges and points are illustrated in Figure 2.5 b).

pivot walking: Not rotating around the center of mass is the key factor for pivot walking. In the first step of a pivot waking cycle, the magnetic field is elevated to let the polyomino pivot on its north pivot edge. As a second step, a rotation of $-\frac{1}{2} \cdot \alpha$ is performed around the north pivot point. $-\pi \leq \alpha \leq \pi$ is the pivot walking angle. For step 3 and 4 the elevation changes to its opposite to perform a rotation of α around the south pivot point. Step 5 and 6 are equal to 1 and 2 and will bring the polyomino back to its original orientation. You can see the pivot walking steps in Figure 2.5 a) and have a closer look at its parameters in Figure 2.5 c).

After one pivot walking cycle the polyomino moved by a displacement vector $\vec{d}$, so $\|\vec{d}\|$ is the pivot walking distance. The direction and length of $\vec{d}$ changes with the shape of a polyomino. The movement is always perpendicular to the pivot walking axis $\vec{a}$, which is a vector pointing from north to south pivot point, visualized in Figure 2.5 b). $\|\vec{d}\|$ can

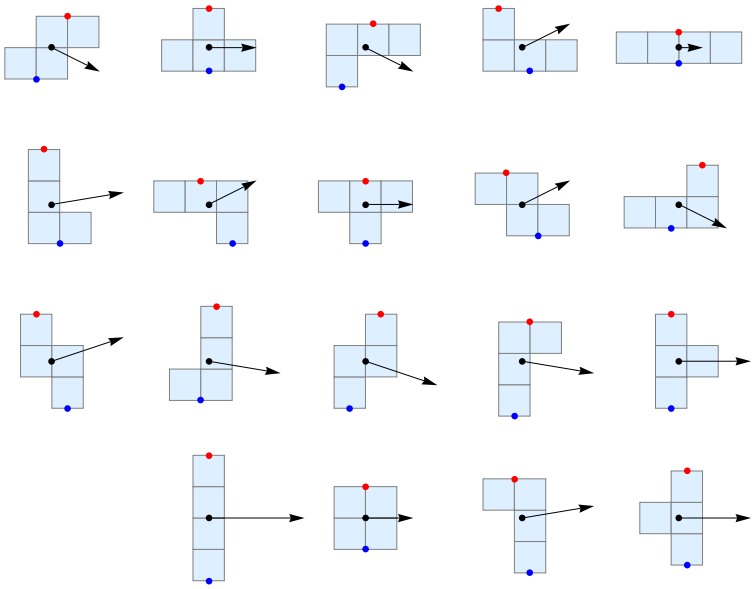


Figure 2.7: All 19 four-cube polyomino shapes with their displacement vector $\vec{d}$ for one pivot walking cycle with $\alpha = \frac{\pi}{4}$. $\vec{d}$ is drawn with a black arrow from its center of mass. North and south pivot point are drawn as red and blue dots.

be calculated with

$$\|\vec{d}\| = 2 \cdot \sin\left(\frac{1}{2}\alpha\right) \cdot \|\vec{a}\| \,. \tag{2.1}$$

Figure 2.6 shows functions for this equation based on α for different $\|\vec{a}\|$. To determine $\vec{d}$ we take the perpendicular of $\vec{a}$ and scale it to the length calculated with Equation 2.1.

When a big α is chosen according to amount, $\|\vec{d}\|$ becomes also bigger, but the polyomino needs more space to the north and south to perform the rotations. For better maneuvering, smaller values of α are preferable. There is a strong deviation of length and direction of the displacement for different polyomino shapes. Performing a pivot walking motion might not move two polyominoes in the same direction. Figure 2.7 shows all 19 four-cube polyomino shapes with their displacement vectors. There are two options for pivot walking, depending on a negative or positive value of α. A polyomino can walk left, in the direction of its west-faces, or right, in the direction of its east-faces. Although the polyomino actually moves in the direction of $\vec{d}$, we can still say that a pivot walk right moves to the east, because $\left|\angle\left(\vec{E}, \vec{d}\right)\right| < \frac{\pi}{2}$. We call these two options the pivot walking direction $\vec{w} \in \{\vec{E}, \vec{W}\}$.

3 Local Planner

This chapter is about the local planner that will be used for motion planning on a global scale in Chapter 4. A local planner only focuses on simple motion task. Possible tasks could be developing a plan that moves a polyomino from one position to another, or even simpler, to develop a plan for one pivot walking cycle. Considering the problem of self-assembly on a global scale, the initial and goal configuration of local plans should differ in the set of polyominoes they contain. Our local planner takes two cubes $c_{\mathcal{A}}$ and $c_{\mathcal{B}}$ out of the polyominoes $\mathcal{A}$ and $\mathcal{B}$ and attempts to establish a connection at a valid edge-pair $(e_{\mathcal{A}}, e_{\mathcal{B}})$. A successful local plan guaranties a change of polyominoes in the workspace.

For this, the local planner makes use of our simulator from Chapter 5 in a closed-loop manner. This means that the configuration of the workspace can be observed at any time and the actions can be adjusted accordingly. The local planner works with position and orientation of cubes and polyominoes provided by the simulator. The distance between two cubes is the Euclidean distance $d(c_{\mathcal{A}}, c_{\mathcal{B}}) = \|p_{c_{\mathcal{A}}} - p_{c_{\mathcal{B}}}\|$ between the cube centers $p_{c_{\mathcal{A}}}$ and $p_{c_{\mathcal{B}}}$. In a real application of Magnetic Modular Cubes a camera, able to track cubes in the workspace, could to used to retrieve the necessary information.

The following Sections 3.1 to 3.3 explain the techniques used in the local planning algorithm of Section 3.4.

3.1 Aligning Cubes

To establish a connection between two polyominoes $\mathcal{A}$ and $\mathcal{B}$, the connection-cubes $c_{\mathcal{A}}$ and $c_{\mathcal{B}}$ with their connection-edges $e_{\mathcal{A}}$ and $e_{\mathcal{B}}$ need to be aligned in the correct way. When $\mathcal{A}$ is rotated without magnetic field elevation, a cube center rotates in a circle around the center of mass of its polyomino $p_{\mathcal{A}}$. The vector $r_{c_{\mathcal{A}}} = p_{c_{\mathcal{A}}} - p_{\mathcal{A}}$ is the radius of this rotation-circle. When also considering $\mathcal{B}$, a rotation of the magnetic field rotates $r_{c_{\mathcal{A}}}$ and $r_{c_{\mathcal{B}}}$ by the same angle β. The goal is to find this angular difference β, so that the cubes are aligned. There are two different approaches for alignment: Straight-aligning and offset-aligning.

Straight-Aligning For straight aligning we define a vector $\overrightarrow{\mathcal{AB}} = p_{c_{\mathcal{B}}} - p_{c_{\mathcal{A}}}$ pointing form $c_{\mathcal{A}}$ to $c_{\mathcal{B}}$. The alignment is done when $e_{\mathcal{A}}$ points in the same direction as $\overrightarrow{\mathcal{AB}}$, so $\angle\left(e_{\mathcal{A}}, \overrightarrow{\mathcal{AB}}\right) = 0$. Consequently $\angle\left(e_{\mathcal{B}}, \overrightarrow{\mathcal{AB}}\right) = \pi$, since $e_{\mathcal{A}}$ and $e_{\mathcal{B}}$ have to be opposite edges for a connection.

Figure 3.1 a) illustrates a straight-align for an east-west connection with all its parameters. The two polyominoes could now theoretically pivot walk together and connect

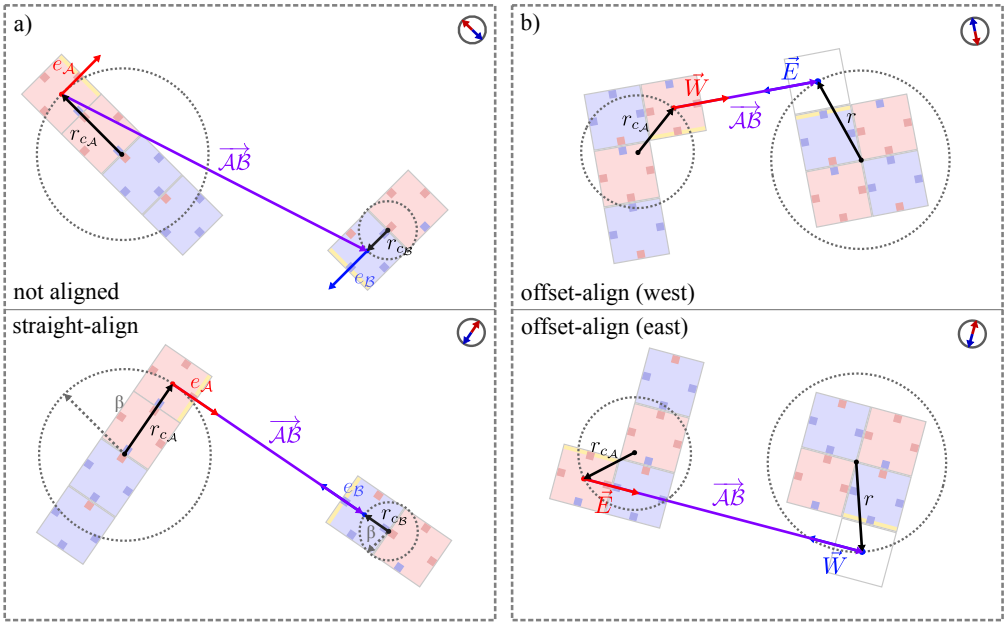


Figure 3.1: Examples illustrating straight- and offset-aligning. The edges to be connected are marked yellow. a) shows two not aligned polyominoes (top) and the result of a straight-align (bottom). In b) the results of the two approaches for offset-aligning are shown. $c_{\mathcal{A}}$ was aligned with its west edge (top) and with its east edge (bottom).

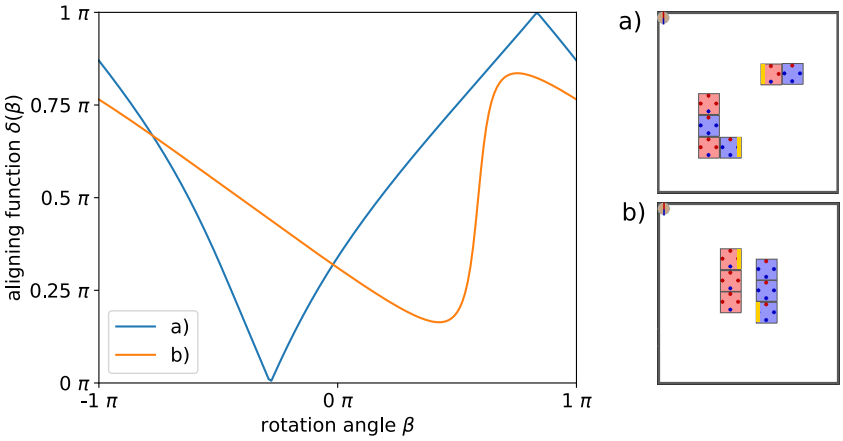


Figure 3.2: Two examples of the aligning function $\delta(\beta)$ for different configurations of polyominoes. The edges about to be aligned are marked yellow. The cubes are perfectly aligned, when $\delta(\beta) = 0$. This can be seen for example a) at $\beta \approx -0.28\pi$. Rotating the magnetic field around this angle of β, would align the cubes. In example b) $\delta(\beta)$ never gets zero. Perfect alignment is not possible, because the polyominoes are to close.

at the desired edges. Straight-aligning is always used for east-west connections, but we also use it for north-south connection in one special case. More on that in Section 3.3.

Offset-Aligning When considering north-south connections we need to align with an offset, so that the cubes can be moved together from either east or west direction. We again define $\overrightarrow{\mathcal{AB}} = (d_{offset} \cdot e_{\mathcal{B}} + p_{c_{\mathcal{B}}}) - p_{c_{\mathcal{A}}}$, but this time an offset d_{offset} is added to $p_{c_{\mathcal{B}}}$ in the direction of $e_{\mathcal{B}}$, so $\overrightarrow{\mathcal{AB}}$ points from $p_{c_{\mathcal{A}}}$ to a position above or below $p_{c_{\mathcal{B}}}$. In a perfect world $d_{offset} = 2 \cdot r_C$ is exactly one cube length, but to avoid failures when moving together, we give the alignment a bigger offset. Instead of pointing $e_{\mathcal{A}}$ in the same direction as $\overrightarrow{\mathcal{AB}}$, we now have two options: Either solving $\angle\left(\vec{E}, \overrightarrow{\mathcal{AB}}\right) = 0$ or $\angle\left(\vec{W}, \overrightarrow{\mathcal{AB}}\right) = 0$, depending on if we want to move $\mathcal{A}$ in east direction, or in the west direction towards $\mathcal{B}$. You can see the two options for offset-aligning in Figure 3.1 b).

3.1.1 Solving Alignment

For calculating angular difference we use the dot-product

$$\angle(a, b) = \frac{a \cdot b}{\|a\|\|b\|},$$

with $a, b \in \mathbb{R}^2$. This way the difference is always positive, which is beneficial in the case of alignment. We define a function for straight-aligning based on the rotation angle β, where both $e_{\mathcal{A}}$ and $\overrightarrow{\mathcal{AB}}$ are rotated accordingly.

$$\delta(\beta) = \angle\left(\mathbf{R}_\beta e_{\mathcal{A}},\, (\mathbf{R}_\beta r_{c_{\mathcal{B}}} + p_{\mathcal{B}}) - (\mathbf{R}_\beta r_{c_{\mathcal{A}}} + p_{\mathcal{A}})\right). \tag{3.1}$$

$\mathbf{R}_\beta$ is a rotation matrix used for rotating vectors by β. For an offset-align the function would be

$$\delta(\beta) = \angle\left(\mathbf{R}_\beta e,\, (\mathbf{R}_\beta r + p_{\mathcal{B}}) - (\mathbf{R}_\beta r_{c_{\mathcal{A}}} + p_{\mathcal{A}})\right), \tag{3.2}$$

with $e \in \{\vec{E}, \vec{W}\}$ and $r = (d_{offset} \cdot e_{\mathcal{B}} + p_{c_{\mathcal{B}}}) - p_{\mathcal{B}}$

Alignment is not always possible, so instead of solving $\delta(\beta) = 0$, the function is minimized. Figure 3.2 shows two example cases for $\delta(\beta)$. In example b) $\delta(\beta)$ does not get zero, because the polyominoes are to close to ever reach perfect alignment. Because $-\pi < \beta \leq \pi$, we can iterate through increasing values of β. If a value close enough to zero is encountered, it can returned. If not, the minimum of all the calculated values is returned. This way we at least get as close to an alignment as possible.

3.2 Moving Polyominoes Together

Pivot walking only allows the polyominoes to move left or right depending on $\vec{w}$. If we want to connect an east face of polyomino $\mathcal{A}$ to a west face of polyomino $\mathcal{B}$, $\mathcal{A}$ has to walk into the east direction towards $\mathcal{B}$, or the other way around.

When $\mathcal{A}$ should be connected at a south face of $\mathcal{B}$, $\mathcal{A}$ can now walk into east or west direction towards $\mathcal{B}$, and $\mathcal{B}$ could again do the opposite. We call this the *slide-in direction* $\vec{m} \in \{\vec{E}, \vec{W}\}$, which states that $\mathcal{B}$ is positioned in direction $\vec{m}$ of $\mathcal{A}$. Both slide-in directions can be achieved in any configuration with offset-aligning from Section 3.1. In Figure 3.1 b) you can see the difference the slide-in directions can make. In this example, establishing a connection by letting $\mathcal{A}$ move towards $\mathcal{B}$ in west direction is possible, but by moving in east direction other cubes of the polyominoes are blocking the way. In Section 3.3 we present a method for checking which slide-in directions are possible for two polyominoes.

Since both polyominoes $\mathcal{A}$ and $\mathcal{B}$ perform pivot walking motions simultaneously, due to global control, a connection will most likely happen when one polyomino walks into a wall of the workspace boundary. Connection can only happen in the middle of the workspace when one polyomino is faster than the other, meaning it has a greater pivot walking distance $\|\vec{d}\|$.

At a first glance it seems easy to move polyominoes together, after the connection-cubes are aligned, but in reality it becomes more difficult. When a polyomino is continuously walking against a wall at any angle other than 90 degree, the polyomino will move alongside the wall. In [19] research is done on how friction with boundary-walls under global control forces can be used to calculate the necessary motions for reaching a desired goal configuration, but friction forces depend greatly on material choices and are stochastic.

Another difficulty are different orientations of displacement vectors, since a pivot walking motion is actually performed in the direction of the displacement $\vec{d}$ and not directly in direction of $\vec{w}$. It is mathematically possible to calculate the right orientation of the magnetic field to result in a collision after a certain number of pivot walking cycles for both polyominoes with different displacement directions, even at desired edges, but it is not guaranteed that this collision-point is within the workspace boundaries. In that case the calculation of friction and displacement have to be combined together with other factors like polyominoes blocking each other or changing their shape during movement.

This is fairly complex and recalculating would be necessary in many situations, so we choose a simpler dynamic approach. We estimate the pivot walking cycles necessary until $c_{\mathcal{A}}$ moved to the original position of $c_{\mathcal{B}}$ with

$$\#steps = \left\lceil \frac{d(c_{\mathcal{A}}, c_{\mathcal{B}})}{\|\vec{d}_{\mathcal{A}}\|} \right\rceil . \tag{3.3}$$

We then only walk $\frac{\#steps}{2}$ and re-align the cubes. When $c_{\mathcal{A}}$ and $c_{\mathcal{B}}$ are near enough for magnetic forces to act, we frequently wait a short period to let magnetic attraction pull $e_{\mathcal{A}}$ and $e_{\mathcal{B}}$ together. This will automatically adjust the alignment, but for even more precision we decreased the pivot walking angle α when in close proximity.

3.3 Plan and Failures

A plan is a sequence of actions $A = a_1, ..., a_k$ that when applied to an initial configuration g_{init}, leads to a goal configuration g_{goal}. Two plans can be concatenated when the goal of the first matches with the initial configuration of the second. That way multiple local plans can be connected to form a global plan. We define a metric to compare and evaluate plans based on rotational cost of its actions. We only consider longitude magnetic field rotations, not latitude elevation. Let a_i be a normal rotation of angle β, then $\text{cost}(a_i) = |\beta|$. If it is a pivot walking motion, then $\text{cost}(a_i) = |2\alpha|$. The cost for the plan is the sum of the costs of all its actions

$$\text{cost}(A) = \sum_{i=1}^{k} \text{cost}(a_i)\,. \tag{3.4}$$

A local plan is successful if g_{goal} contains a polyomino with the desired connection of $c_{\mathcal{A}}$ and $c_{\mathcal{B}}$ at $(e_{\mathcal{A}}, e_{\mathcal{B}})$. The plan-state s describes if a plan is successful or not. There are several reasons the local planner might fail to develop a plan:

Impossible Connection Most failures occur, because it is not possible to connect the polyominoes. First of all, $e_{\mathcal{A}}$ and $e_{\mathcal{B}}$ need to be free, so no other cube is already connected to them. Even if both edges are free, other cubes then $c_{\mathcal{A}}$ and $c_{\mathcal{B}}$ can prevent a connection. By connecting two polyominoes in one local discrete coordinate-system, for all cubes c_1, c_2 with coordinates (x_1, y_1), (x_2, y_2): $|x_1 - x_2| < 1$ and $|y_1 - y_2| < 1$ should hold true. If two positions are equal, we call this an overlap, which prevents the connection. Of course, a connection is never possible if $e_{\mathcal{A}}$ and $e_{\mathcal{B}}$ are part of the same polyomino and not already connected.

All these conditions are easy to check in a discrete way before even starting to plan, but connections with other polyominoes during planning can invalidate those pre-checked conditions. Because of this, frequently re-checking them is necessary.

Impossible Slide-In Even if a connection in a common local coordinate-system is possible, the polyominoes need to slide in from east or west dsirection. Other cubes can again prevent this by blocking the way for an easy slide-in. We can verify both slide in directions $\vec{m}$ in a common local coordinate system. This discrete check assumes exact movement from east or west direction. Because of different displacement directions, we know this is not true, but it is a reasonable approximation. Research on assembling a polyomino out of two parts by moving one part towards the other without collision, was done by Agarwal et al. [1].

When pre-checking this condition, we can state failure if both directions are not possible. Otherwise, we can align with respect to the valid slide-in direction, or try out both, if both are possible. Again, this condition needs to be re-checked frequently, due to changing polyominoes.

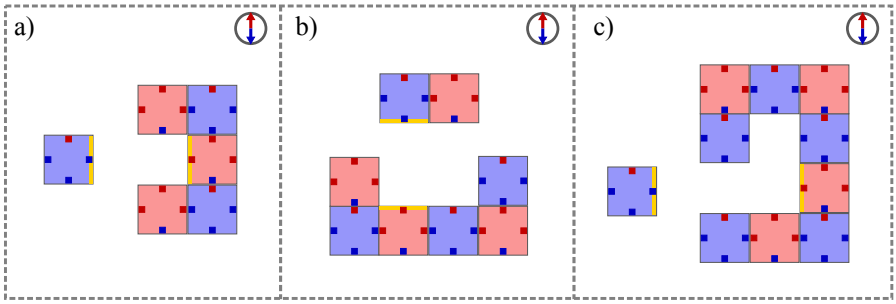


Figure 3.3: Three different examples for connecting polyominoes into caves. a) and b) show one-cube-deep caves (a) east-west and b) two-cube-wide north-south). c) illustrates a two-cube-deep east-west cave. The edges to be connected are marked yellow.

Polyominoes being Stuck Polyominoes can get stuck in corners or on walls of the workspace. In this state it is not possible to decrease the distance of $\mathcal{A}$ and $\mathcal{B}$ by pivot walking. We can identify this state when the positions of both $c_{\mathcal{A}}$ and $c_{\mathcal{B}}$ do not change after a certain amount of pivot walking motions.

When stuck while trying to establish a north-south connection, a straight-align, instead of an offset-align, can resolve the situation. Success depends on the distance of the cubes after straight-aligning. If the distance is too big for magnetic forces to act, failure is reported. If the cubes are close enough the local planner waits until magnetic attraction connects $e_{\mathcal{A}}$ and $e_{\mathcal{B}}$.

Connecting in Caves Connecting two polyominoes where one of the connection-faces is located inside a cave is a difficult task in the continuous world. We differentiate between east-west and north-south located caves. Furthermore, a cave can be of certain depth and width measured in multiples of $2r_C$. Figure 3.3 shows examples for caves with varying depths and widths.

Caves only become problematic when the polyomino to be inserted has the same width as the cave, shown in Figure 3.3 b). Connecting into a cave with a depth of more than $2r_C$ is not possible. For instance, when inserting the blue single cube into the polyomino in Figure 3.3 c), the blue cube would connect with north and south faces of the polyomino before even reaching the full depth of the cave. But even caves with depth $2r_C$ are hard to handle.

Inserting into a cave can be done by pivot walking. This only works for east-west caves, or by letting magnetic forces attract $e_{\mathcal{A}}$ and $e_{\mathcal{B}}$. Relying on magnetic forces alone seem promising, since it would work for both cave types, but in reality not only the forces of the connection-faces are present. All the forces between other magnets prevent an easy insertion into the cave. In our simulator the connection-face will be more attracted or repelled by faces outside the cave, then the once inside. Pivot walking into east-west caves, even with small values for α, has also a high failure rate because of other magnets.

The local planner states failure immediately when polyominoes should be connected in any cave-type.

Invalid Polyominoes Because construction of invalid polyominoes is hard to handle on a global scale, we already omit plans containing them in our local planner. Failure is stated if an invalid polyomino is created at any point during planning. We also pre-check (and frequently re-check) if the polyomino that will be created by establishing the connection would itself be invalid.

Maximum Movement Capacity As a worst-case failure, we limit the amount of movement $\mathcal{A}$ and $\mathcal{B}$ are able to do. Whenever a pivot walking motion is done, we sum up the distances $c_{\mathcal{A}}$ and $c_{\mathcal{B}}$ moved together. We define a maximum movement capacity of $2 \cdot (\text{width} + \text{height})$ dependent on the size of the workspace. This capacity gives the polyominoes enough movement, so that both can move along a horizontal and vertical workspace boundary, which should be sufficient to establish a connection.

3.4 Local Planning Algorithm

Before executing Algorithm 1, all failure conditions that can be pre-checked are evaluated, so no simulation-time is wasted on a plan that is bound to fail from the beginning. While doing so, the possible slide-in directions are determined and Algorithm 1 is executed with both pivot walking directions $\vec{w}$ for each possible $\vec{m}$. This means two plans are developed for an east-west connection and two or four for a north-south connection, depending on the slide-in directions. Figure 3.4 shows an example of these four local plans created by Algorithm 1.

In the end, the successful plan with the lowest costs is returned. If all plans fail, the best failure is determined. Again, plans with lower costs are preferable, but we favor impossible connection and slide-in failures. These failures just state that a specific connection cannot be established, but a global planner could continue to plan based on the goal configuration the local planner ended in. A plan that ended due to polyominoes being stuck, or a reached maximum movement capacity, is not a good starting point for further planning. Plans creating invalid polyominoes or polyominoes with caves are omitted by the global planner.

The different plans are developed in parallel and if one process finishes with a successful plan, the execution of all other processes can be canceled. This saves computation time, although we might not return the best plan, since fastest computation does not automatically mean lowest costs. Generally speaking a low computation time can be linked with low rotational cost, because the local planner spends the majority of time, about 98%, on simulating actions.

Align-Walk-Realign Algorithm 1 takes the connection-cubes and edges $c_{\mathcal{A}}$, $c_{\mathcal{B}}$, $e_{\mathcal{A}}$, $e_{\mathcal{B}}$ along with $\vec{w}$, $\vec{m}$ and an initial configuration g_{init} as inputs and returns a plan-state s along with the configuration g_{goal} the algorithm ended in after applying the sequence

Algorithm 1 ALIGN-WALK-REALIGN

Input: $c_{\mathcal{A}}$, $c_{\mathcal{B}}$, $e_{\mathcal{A}}$, $e_{\mathcal{B}}$, $\vec{w}$, $\vec{m}$, g_{init}
Output: (s, g_{goal}, A) // state s and actions A leading to configuration g_{goal}

$s \leftarrow$ undefined
$g_{goal} \leftarrow g_{init}$
$A \leftarrow \{\}$
wait $\leftarrow$ **true**
loop
  if $e_{\mathcal{A}} \in \{\vec{E}, \vec{W}\}$ **then** // aligning straight or with offset
    $a \leftarrow$ ALIGN-STRAIGHT$(c_{\mathcal{A}}, c_{\mathcal{B}}, e_{\mathcal{A}})$
  else
    $a \leftarrow$ ALIGN-OFFSET$(c_{\mathcal{A}}, c_{\mathcal{B}}, \vec{m}, e_{\mathcal{B}})$
  end if
  $g_{goal} \leftarrow$ SIMULATE(g_{goal}, a)
  $A \leftarrow$ APPEND(A, a)
  $s \leftarrow$ UPDATE-STATE$(g_{goal}, c_{\mathcal{A}}, c_{\mathcal{B}}, e_{\mathcal{A}}, \vec{m})$
  if $s \neq$ undefined **then** // first time checking for failure or success
    return (s, g_{goal}, A)
  end if
  if CRITICAL-DISTANCE$(c_{\mathcal{A}}, c_{\mathcal{B}})$ **and** wait **then** // wait or walk
    $a \leftarrow$ WAIT()
    wait $\leftarrow$ **false**
  else
    $a \leftarrow$ WALK$(c_{\mathcal{A}}, c_{\mathcal{B}}, \vec{w})$ // a is $\frac{\#steps}{2}$ actions (Section 3.2)
    wait $\leftarrow$ **true**
  end if
  $g_{goal} \leftarrow$ SIMULATE(g_{goal}, a)
  $A \leftarrow$ APPEND(A, a)
  if STUCK$(c_{\mathcal{A}}, c_{\mathcal{B}})$ **then** // handle stuck condition
    $a \leftarrow$ ALIGN-STRAIGHT$(c_{\mathcal{A}}, c_{\mathcal{B}}, e_{\mathcal{A}})$ // do a straight align
    $g_{goal} \leftarrow$ SIMULATE(g_{goal}, a)
    $A \leftarrow$ APPEND(A, a)
    while not STUCK$(c_{\mathcal{A}}, c_{\mathcal{B}})$ **do** // let magnets attract until stuck again
      $a \leftarrow$ WAIT()
      $g_{goal} \leftarrow$ SIMULATE(g_{goal}, a)
      $A \leftarrow$ APPEND(A, a)
    end while
  end if
  $s \leftarrow$ UPDATE-STATE$(g_{goal}, c_{\mathcal{A}}, c_{\mathcal{B}}, e_{\mathcal{A}}, \vec{m})$
  if $s \neq$ undefined **then** // second time checking for failure or success
    return (s, g_{goal}, A)
  end if
end loop

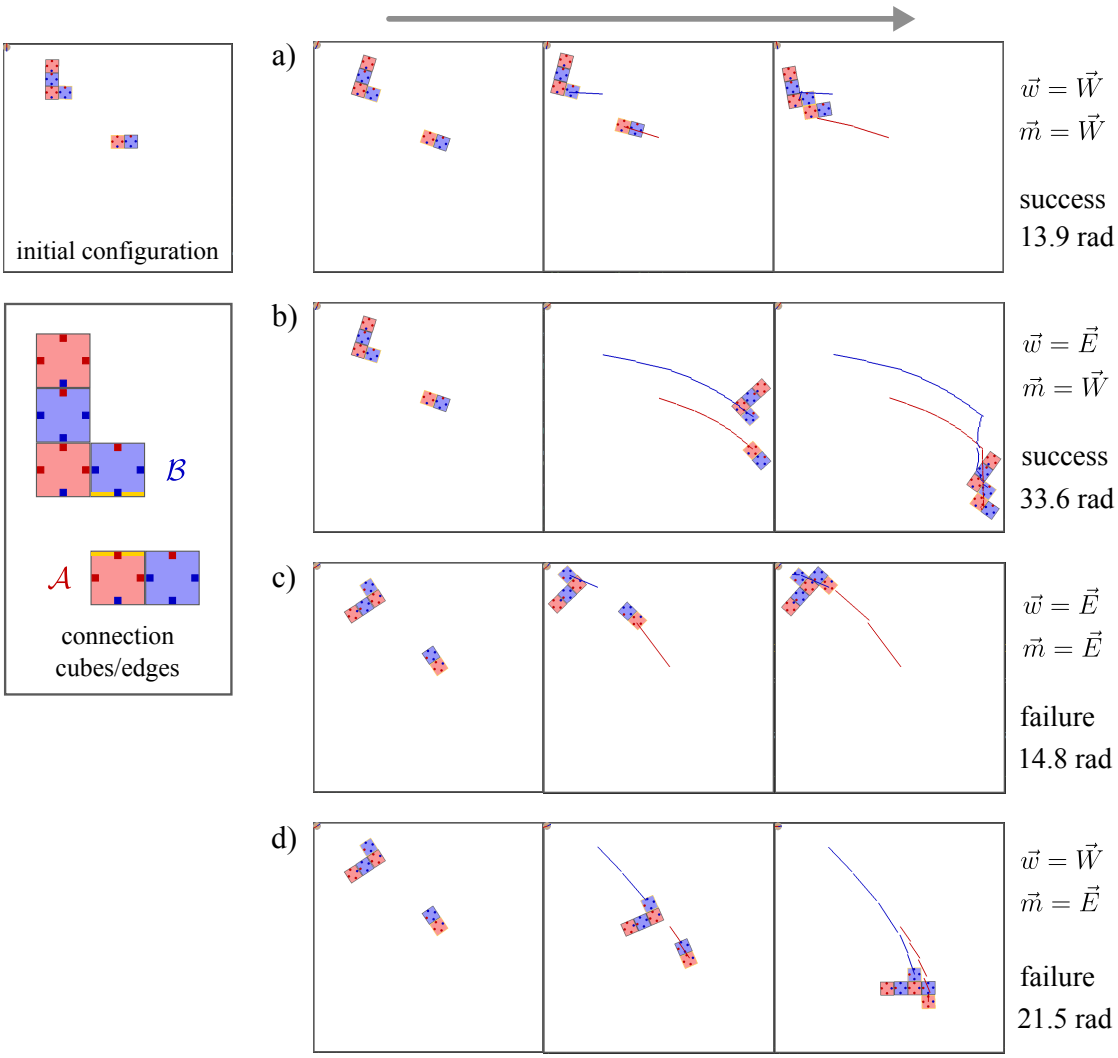


Figure 3.4: Illustration of all local plans developed by Algorithm 1 for different pivot walking directions $\vec{w}$ and slide-in directions $\vec{m}$. On the left side the initial configuration and connection cubes/edges of $\mathcal{A}$ and $\mathcal{B}$ are shown. Although both slide-in directions are possible, choosing $\vec{m} = \vec{E}$ results in failure due to a wrong connection for both c) and d). The two successful plans a) and b) have a huge difference in plan cost. a) is the cheaper plan returned by the local planner. In b) $\mathcal{A}$ and $\mathcal{B}$ slide along the workspace boundaries until connecting in the bottom right corner. Both $c_{\mathcal{A}}$ (red) and $c_{\mathcal{B}}$ (blue) leave a trace when pivot walking.

of actions $\mathcal{A}$. The algorithm runs in a loop until s changes to success or one of the failure condition. The failure and success conditions are evaluated twice per iteration with UPDATE-STATE. This is done once after aligning and once at the end of the loop to avoid the simulation of unnecessary actions. g_{goal} is updated by simulating the determined actions with SIMULATE. The actions are appended to A after simulation. We perform either a straight or offset-align, depending on $e_{\mathcal{A}}$ and $e_{\mathcal{B}}$. The offset-align is done with the direction of $\vec{m}$. After aligning we walk the estimated amount of pivot walking cycles (Section 3.2) in direction $\vec{w}$ with WALK, or we wait with WAIT, if $c_{\mathcal{A}}$ and $c_{\mathcal{B}}$ are in close proximity, determined by CRITICAL-DISTANCE. If we waited in the previous iteration, we walk in the current one and if we walked previously, we wait in the current iteration. This behavior is toggled by the variable "wait". The stuck condition is evaluated with STUCK and does not state failure immediately, since a straight-align might be able to fix the situation. When the polyominoes are stuck, the algorithm performs a straight-align and waits as long as this changes the stuck condition.

3.4.1 Complexity

Optimality The optimal plan for connecting two polyominoes is the one establishing the connection with the lowest rotational cost, defined in Section 3.3. We use this metric, because it is strongly linked with computation time, but can also be interpreted in a real word application of modular magnetic cubes. Even if the local planner would not calculate plans in parallel, our dynamic approach of realigning does not produce optimal solutions. It therefore simulates only the actions that are included in the final plan, which minimizes simulation time.

Optimality could be achieved when sliding on walls and different polyomino displacements, as described in Section 3.2, are not existent. In this hypothetical case both pivot walking directions produce plans that move the polyominoes together in a straight path. The plan with the shorter path would be optimal. In reality these factors must be considered. Even if the local planner could pre-calculated those, it is unclear if this would be enough to prove optimality.

Completeness Our local planner is not complete. We cannot exclude the existence of a solution just because the up to four dynamic plans developed fail. If other polyominoes are blocking the way of $\mathcal{A}$ and $\mathcal{B}$, complex movements around these polyominoes, instead of the approximately straight path that we are taking, could create solutions where our local planner fails. The reason we choose this simple and not complete approach is to minimize simulation of movements as much as possible.

4 Global Planner

The task of the global planner is to assemble a specified target polyomino $\mathcal{T}$ given an initial configuration g_{init}. The configuration-space is explored by executing local plans developed by the local planner from Chapter 3. That way, the configuration-space is limited to configurations where a connection between two cubes was attempted. Since the number of cubes is limited, there is a finite set of local plans leading to new configurations. Compared to $SE(2)$, the part of the configurations-space explored by our global planner contains only configurations which are relevant for self-assembly.

Determining how these configurations are explored affects the run time considerably. Using rapidly-exploring random trees (RRTs) [11] yields good results in many cases. The configuration-space gets evenly explored without the challenge of determining what decisions are promising for the end goal. This also means that most of the explored configurations are not relevant for reaching the goal. For us this approach is not reasonable, because of the high fidelity simulation we are working with. Computation time for a local plan is huge, so planning the assembly of $\mathcal{T}$ with as few local plans as possible is the aim for our global planner.

We need to make well-thought-through connection decisions, that are valid for assembling $\mathcal{T}$, meaning some sort of building plan for a polyomino is needed. Creating a building sequences by removing one tile at a time from the target was done by Becker et al. [4]. However, this does not consider sub-assemblies, so all cubes that are not to be connected have to stay separated at any time.

It is hard to prevent sub-assemblies with magnetic cubes following the rules of tilt. Our approach uses an enumeration of ways to cut a polyomino into two parts (Section 4.1), which will be used for generating a so-called two-cut-sub-assembly graph (Section 4.2). This graph functions as a building instruction along side the exploration of the configuration-space. Section 4.3 provides a closer look on the use of this graph for decision making and Section 4.4 further explains the usage of the local planner on a global scale. Finally Section 4.5 combines previous techniques to a global planning algorithm. For the algorithm the number of cubes in the workspace is limited to the size of $\mathcal{T}$. An explanation on why is this done and why the problem becomes more complex when working with extra cubes, is provided in Section 4.6.

4.1 Two-Cutting Polyominoes

Schmidt et al. [20] made use of straight-line two-cuts, to handle the construction of a polyomino with more than trivial sub-assemblies.

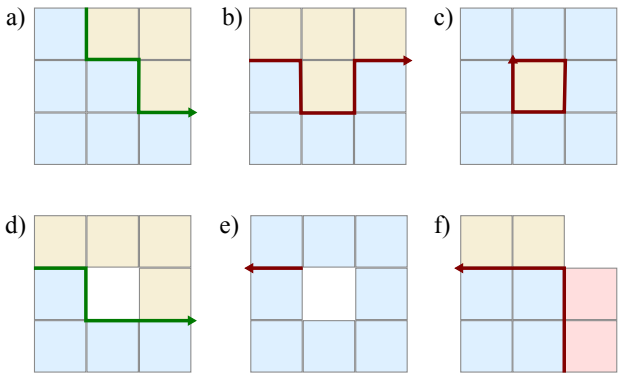


Figure 4.1: Examples for cutting polyomino shapes. a) to d) show four two-cuts, of which only a) and d) are monotone and therefore valid. b) creates a cave and c) a hole. e) and f) show cuts that do not split the polyomino shapes into two pieces. e) does not break the polyomino at all and f) creates three sub-polyominoes.

We define a *two-cut* as a continuous edge path through a polyomino that would divide the polyomino into two sub-polyominoes, if all connections with these edges are removed. For later use in Section 4.2, an enumeration of all two-cuts of a polyomino that are useful for planning is needed. We do not limit the cuts by only allowing straight paths like Schmidt et al. [20], instead we only consider monotone two-cuts.

Monotone means that whenever the path goes into a certain direction it can never go into the opposite direction again. Figure 4.1 a) and d) show examples of monotone two-cuts. The cuts start at the top of the shape and only move down and right. By removing all the connections on these paths, the polyomino shapes are split into two pieces. Considering non-monotone two-cuts would create sub-assemblies with caves or holes, which could not be reassembled with our local planner. For this reason they are omitted on a global scale in advance. Figure 4.1 b) shows a non-monotone two-cut creating a cave and Figure 4.1 c) one creating a hole.

To calculate all two-cuts of a polyomino, we take all possible monotone paths from each connection as a starting point. A path ends when it breaks out of the polyomino. After the path ended the connections at its edges are removed and the path is added as a two-cut, if the polyomino got split into exactly two pieces. Figure 4.1 e) and f) show cuts that split the polyomino in less or more than two pieces.

4.2 Two-Cut-Sub-Assembly Graph

The two-cut-sub-assembly graph, short TCSA graph, functions as a building instruction for a specific target polyomino. For target polyomino $\mathcal{T}$, wse will call it $G_{TCSA}(\mathcal{T}) = (V, E)$ represented by nodes V and edges E. The TCSA graph works with sets of polyominoes as nodes. While a configuration g holds information about orientation and position of physically distinct polyominoes, the corresponding polyomino set $S(g)$ only

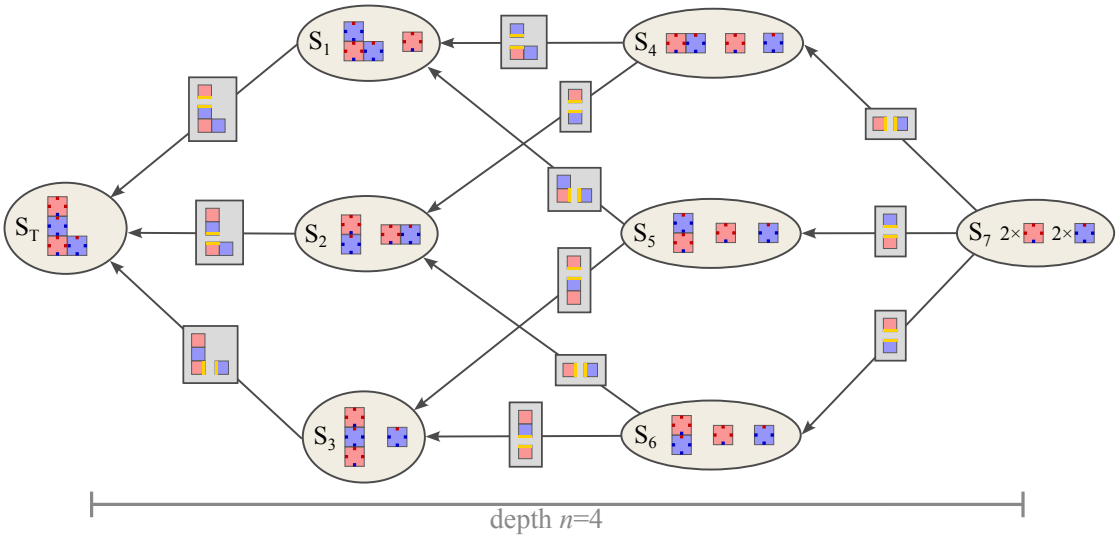


Figure 4.2: Example of a two-cut-sub-assembly graph for a four-cube L-shape. The polyomino sets are illustrated as ellipses. If the polyominoes of a set are not numbered, there is only one occurrence of this polyomino. Otherwise the number of occurrences is placed left of the polyomino. The sets are numbered as if the graph was produced by Algorithm 2 starting from $S_{\mathcal{T}}$. The weights of edges are illustrated as rectangular boxes containing the polyominoes that need to be connected at specific cube edges, marked in yellow.

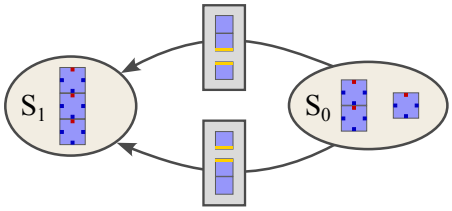


Figure 4.3: Two two-cut-sub-assembly edges connecting the polyomino sets S_0 and S_1. The weights of the edges differ, since there are two ways to connect the 2×1 with the 1×1 to create the 3×1 polyomino. The connections are illustrated in rectangular boxes placed on the edges.

enumerates the polyomino types present in g. If g contains multiple polyominoes of the same type, $S(g)$ still stores the amount of the polyomino type, but does not distinguish between the actual polyominoes.

Two nodes S_0 and S_1 of the TSCA graph are connected with an edge $\{S_0, t_c, S_1\}$, if S_0 can be transformed to S_1 by connecting two polyomino types contained in S_0. The edge path specifying this connection is stored as the weight t_c. S_0 and S_1 can be connected by multiple edges, if there are different connections that produce the same outcome. The edges differ in their weights as shown in Figure 4.3. The direction of $\{S_0, t_c, S_1\}$ always goes from S_0 to S_1, but we can reverse the definition for an edge as follows:

Two nodes S_0 and S_1 are connected if one polyomino contained in S_1 can be two-cut by an edge path t_c, so that the resulting polyomino set equals S_0. This provides a perspective on the use of two-cuts and the way $G_{TCSA}(\mathcal{T})$ is built starting with $\mathcal{T}$. We will further explain the building process along with an example of a TCSA graph provided in Figure 4.2.

Algorithm 2 BUILD-TCSA-GRAPH

Input: $\mathcal{T}$ // target polyomino
Output: $G_{TCSA}(\mathcal{T})$ // the graph is represented by nodes V and edges E

1: $V \leftarrow \{\}$
2: $E \leftarrow \{\}$
3: $i \leftarrow 0$
4: $V[i] \leftarrow S_{\mathcal{T}}$ // start with set only containing $\mathcal{T}$
5: **while** $i <$ SIZE(V) **do** // work through nodes in BFS manner
6: $S_i \leftarrow V[i]$
7: **for each** $\mathcal{A} \in S_i$ **do** // go through all polyomino types in S_i
8: **for each** $t_c \in$ TWO-CUTS$(\mathcal{A})$ **do** // go through all monotone two-cuts
9: $(\mathcal{A}_1, \mathcal{A}_2) \leftarrow$ CUT-POLYOMINO$(\mathcal{A}, t_c)$
10: $S_{new} \leftarrow (S_i \setminus \{\mathcal{A}\}) \cup \{\mathcal{A}_1, \mathcal{A}_2\}$ // new node after cutting
11: **if** $S_{new} \notin V$ **then**
12: $V \leftarrow$ APPEND(V, S_{new})
13: **end if**
14: $E \leftarrow$ APPEND$(E, \{S_{new}, t_c, S_i\})$
15: **end for**
16: **end for**
17: $i \leftarrow i + 1$
18: **end while**
19: **return** (V, E)

Building a TCSA Graph Algorithm 2 describes the process of building $G_{TCSA}(\mathcal{T})$ for the target $\mathcal{T}$. The algorithm works through each newly added node in V in a breadth-first-search manner. The first node added to V is $S_{\mathcal{T}}$, which is a polyomino set only containing the target shape.

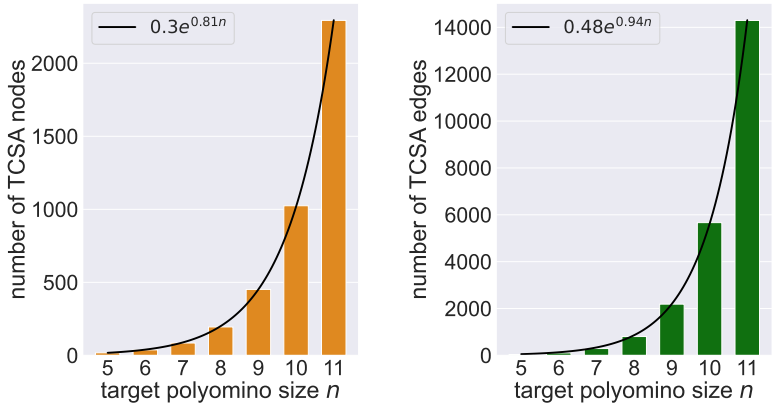


Figure 4.4: Average number of nodes (left) and edges (right) of a TCSA graph for different target polyomino sizes n. For each size 200 polyominoes were randomly generated. An exponential function is fitted over the averages of nodes and edges.

New nodes and edges are determined by two-cutting every polyomino type $\mathcal{A}$ in the current set S_i by every possible monotone two-cut of $\mathcal{A}$. This is done by enumerating the two-cuts with Two-Cuts, the way it was described in Section 4.1. The cutting, done with Cut-Polyomino, results in the two sub-polyominoes $\mathcal{A}_1$ and $\mathcal{A}_2$.

S_{new} contains the same polyominoes as S_i with the exception that one occurrence of $\mathcal{A}$ is removed and replaced by one occurrence of $\mathcal{A}_1$ and $\mathcal{A}_2$. Each S_{new} is the result of cutting one polyomino of S_i at a specific two-cut t_c. If S_{new} is not already contained in V, it can be added, which also queues it for future iterations of the breadth-first-search.

No matter if S_{new} is contained in V or not, an edge going from S_{new} to S_i with t_c as the weight is added to the edges E. This allows multiple edges, as seen in Figure 4.3, and multiple outgoing edges to different nodes, which can be observed in Figure 4.2, where different connections in S_4 lead to S_1 or S_2.

Each two-cut applied to a polyomino set increases its amount of polyominoes by one. Let n be the size of $\mathcal{T}$, then $n-1$ two-cuts applied to $S_{\mathcal{T}}$ will produce a polyomino set $S_{trivial}$ containing n trivial polyominoes, as it is the case for S_7 in Figure 4.2. All S_i will inevitably end up in this situation and the algorithm will return (V, E), since trivial polyominoes cannot be cut anymore. This implies that no matter which edges are chosen along the way, $n-1$ edges need to be traversed to get from $S_{trivial}$ to $S_{\mathcal{T}}$. We describe this attribute, by giving the TCSA graph a depth of n. The depth is also illustrated in Figure 4.2 and the numbering of the nodes matches the order they were added by Algorithm 2.

4.2.1 Complexity

The Stirling numbers of second kind provide an upper bound for the number of nodes in a TCSA graph. The Stirling numbers of second kind

$$\left\{ {n \atop k} \right\} = \sum_{i=1}^{k} \frac{(-1)^{k-i}\, i^{n-1}}{(i-1)!\,(k-1)!} \tag{4.1}$$

describe the possibilities of sorting a set with n objects into k partitions [9]. In our case, n is the size of $\mathcal{T}$ and the number of partitions k is the number of polyominoes, the n cubes belong to. Different layers of depth account for different $\left\{ {n \atop k} \right\}$. $S_{\mathcal{T}}$ is the only polyomino set with $k = 1$, so $\left\{ {n \atop 1} \right\} = 1$. S_{trival} is the only set containing $k = n$ polyominoes, so $\left\{ {n \atop n} \right\} = 1$. For the maximum number of nodes possible all layers of the TCSA graph have to be summed up

$$|V|_{worst} = \sum_{k=1}^{n} \left\{ {n \atop k} \right\}, \tag{4.2}$$

which is also referred to as the Bell number [9].

In our case, the only way of sorting cubes into partitions is by monotonously two-cutting existing polyominoes, which drastically lowers the number of $|V|_{worst}$. In Figure 4.4 statistical data shows the average number of nodes and edges a TCSA graph consists of for varying target polyomino sizes n. The growth of nodes and edges seems to be exponential, which is shown with fitted functions in Figure 4.4.

Our implementation of $G_{TCSA}(\mathcal{T})$ stores nodes in a hash-table. Accessing nodes and connected edges, or checking if a polyomino set is contained in $G_{TCSA}(\mathcal{T})$, can be done in $\mathcal{O}(1)$. The creation of $G_{TCSA}(\mathcal{T})$ becomes more complex for increasing numbers of n, but it provides an easily accessible building instruction that drastically cuts the number of unnecessary local plans simulated. Lowering simulation time makes a complex data-structure like the TCSA graph worth the extra effort.

4.3 Connection Options

In each configuration g that the global planner encounters, $G_{TCSA}(\mathcal{T})$ will be used to determine the next connection that the local planner should try to establish. Outgoing edges of $S(g)$ will be retrieved from the hash-table of $G_{TCSA}(\mathcal{T})$. If $S(g) \notin G_{TCSA}(\mathcal{T})$, g cannot be used to assemble $\mathcal{T}$. This allows the global planner to state failure immediately, when a initial configuration already contains sub-assemblies that are not usable for assembling $\mathcal{T}$. With the exception of $S_{\mathcal{T}}$ all nodes have outgoing edges in a TCSA graph. All outgoing edges of $S(g)$ provide connections for the local planner that bring the global planner closer to assembling $\mathcal{T}$.

For instance, if $S(g) = S_7$ in Figure 4.2, three outgoing edges provide three connections to choose from, but that is not all. Assume the global planner decides to connect a red cube at the west edge of a blue cube to end up in a configuration g_2 with $S(g_2) =$

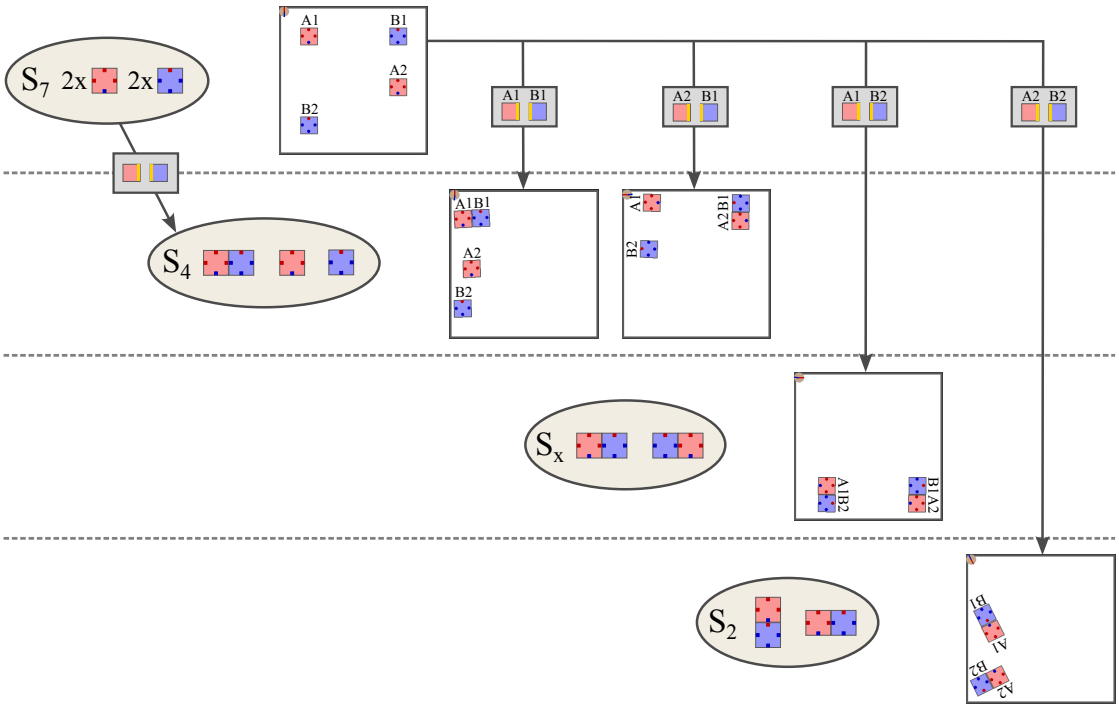


Figure 4.5: All connection options when connecting a red cube at the west of a blue cube to get from S_7 to S_4. Developing local plans for different polyomino pairs leads to different goal configurations. (A_1, B_1) and (A_2, B_1) lead to configurations with the desired polyomino set S_4, but (A_2, B_2) leads directly to S_2. All these sets can be found in the TCSA graph of Figure 4.2. The goal configuration of (A_1, B_2) holds the set S_x, which cannot be found in Figure 4.2. For further global planning this set could not be used.

S_4. Since S_7 contains multiple polyominoes for the same type, there is more than one way to achieve this. Figure 4.5 illustrates all the different connection options for this example case. For all four connection options, the local planner ended in different goal configurations.

Let $L_{\mathcal{A}}$ and $L_{\mathcal{B}}$ be collections of the physically distinct polyominoes for the polyomino types $\mathcal{A}$ and $\mathcal{B}$. When $\mathcal{A}$ and $\mathcal{B}$ are about to be connected as the weight of a TCSA edge dictates, there are $|L_{\mathcal{A}} \times L_{\mathcal{B}}|$ polyomino pairs to choose from. If $\mathcal{A} = \mathcal{B}$, the options where a polyomino will be connected with itself can be eliminated.

With multiple edges and various polyomino pairs per edge, many options emerge for the global planner to consider. We examined three option sorting strategies, to provide an order of the best probable outcome that the global planner can work through. We define functions for determining the best option $\hat{o}$ out of two options o_1 and o_2. The approaches are compared in Chapter 6.

Minimal Distance The minimal distance sorting sorts connection options based on the distance between the connection-cubes $c_{\mathcal{A}}$ and $c_{\mathcal{B}}$. The idea is that a smaller distances requires less movement to establish a connection, which means shorter simulation time and lower plan costs. Due to sliding on walls and different pivot walking distances, this is not true in every case, but it remains a good heuristic for sorting, because it is simple to compute. Less movement might even prevent unwanted sub-assemblies.

$$\hat{o}(o_1, o_2) = \begin{cases} o_1 & \text{if } d(c_{\mathcal{A},1}, c_{\mathcal{B},1}) \leq d(c_{\mathcal{A},2}, c_{\mathcal{B},2}) \\ o_2 & \text{otherwise} \end{cases} \tag{4.3}$$

Grow Largest Component Another approach is to grow the largest component. The options are sorted into classes of maximum polyomino size $\hat{n}$ of the resulting polyomino set. TCSA edges that lead to sets containing the biggest polyominoes are preferred. When the options for S_4 in Figure 4.2 are sorted, the one leading to S_1 is preferred over the one leading to S_2, because S_1 contains a polyomino of size 3, while S_2 only contains polyominoes of size 2. The options within each class are sorted with the minimal distance approach.

If no other sub-assemblies occur, growing the largest component behaves like one-tile-at-a-time assembly. The benefit is that even if sub-assemblies occur, the TSCA graph provides solutions to integrate them if possible. Larger polyominoes generally move faster acting positive on plan costs.

$$\hat{o}(o_1, o_2) = \begin{cases} o_1 & \text{if } (\hat{n}_1 > \hat{n}_2) \vee ((\hat{n}_1 \equiv \hat{n}_2) \wedge (d(c_{\mathcal{A},1}, c_{\mathcal{B},1}) \leq d(c_{\mathcal{A},2}, c_{\mathcal{B},2}))) \\ o_2 & \text{otherwise} \end{cases} \tag{4.4}$$

Grow Smallest Component Oppositely to growing the largest component, options can be sorted by the smallest maximum size of polyominoes in polyomino sets. This avoids

working with large polyominoes, which are faster, but also need more simulation time to perform rotations and can be hard to handle, because of sheer size.

$$\hat{o}(o_1, o_2) = \begin{cases} o_1 & \text{if } (\hat{n}_1 < \hat{n}_2) \vee ((\hat{n}_1 \equiv \hat{n}_2) \wedge (d(c_{\mathcal{A},1}, c_{\mathcal{B},1}) \leq d(c_{\mathcal{A},2}, c_{\mathcal{B},2}))) \\ o_2 & \text{otherwise} \end{cases} \tag{4.5}$$

4.4 Use of Local Planner

The local planner develops plans for connections chosen from the different connection options presented in Section 4.3. For the local planner only one connection, out of the path of connections stored in the weight of a TSCA edge, needs to be picked. Whenever a path consists of both north-south and east-west connections, a north-south connection is preferred. This is done to perform offset-aligning instead of straight-aligning (Section 3.1) for an easier slide-in. Besides that, the choice of connection is irrelevant, since all connections in the path lead to the same outcome.

When the local planner successfully connects the desired polyominoes, other sub-assemblies can lead to a different polyomino set than expected. This is not necessarily bad as long as the resulting set is contained in $G_{TCSA}(\mathcal{T})$. In fact, more sub-assemblies decrease the number of polyominoes in the workspace, which brings the goal of assembling $\mathcal{T}$ even closer. Layers of depth were skipped in the TCSA graph, so that it might be possible to assemble $\mathcal{T}$ with less than $n-1$ local plans. This can be seen in Figure 4.5, when A_2 and B_2 are connected. The resulting polyomino set matches with S_2 instead of S_4 of the nodes from Figure 4.2.

Like already mentioned in Section 4.3, when the resulting polyomino set is not in $G_{TCSA}(\mathcal{T})$, it is not possible to assemble the target from that configuration. This can be seen in Figure 4.5 when connecting A_1 and B_2. For global use we add a new failure condition to the local planner, which frequently checks if the polyomino set of the configuration in the workspace is contained in $G_{TCSA}(\mathcal{T})$. If not, the local planner immediately states failure and avoids spending simulation time on a configuration with no further use.

The local planner might even fail to establish the desired connection. If the resulting polyomino set is contained in $G_{TCSA}(\mathcal{T})$, global planning can continue, but there are certain failure types that are not valid for further planning. Polyomino sets with invalid polyominoes, or where connections in caves are necessary, should not be present in the TCSA graph anyway, but we also do not continue planning with a failure due to maximum movement capacity or polyominoes being stuck (Section 3.3).

4.5 Global Planning Algorithm

Algorithm 3 takes the initial configuration g_{init} and the target $\mathcal{T}$ as inputs and returns the state of the global plan s and a plan stack P as outputs.

Algorithm 3 ASSEMBLE-TARGET

Input: $\mathcal{T}$, g_{init} // target polyomino and intital configuration
Output: s, P // state of global plan s and plan stack P containing local plans

1: $G_{TCSA}(\mathcal{T}) \leftarrow$ BUILD-TCSA-GRAPH($\mathcal{T}$)
2: $s \leftarrow$ undefined
3: $P \leftarrow \{\}$
4: $g \leftarrow g_{init}$ // current configuration g
5: **loop**
6: $O \leftarrow$ CONNECTION-OPTIONS($g, G_{TCSA}(\mathcal{T})$)
7: valid $\leftarrow$ **false**
8: **while not** EMPTY(O) **and not** valid **do** // try options until local plan is valid
9: $(c_{\mathcal{A}}, c_{\mathcal{B}}, e_{\mathcal{A}}, e_{\mathcal{B}}) \leftarrow$ POP(O)
10: $p_{new} \leftarrow$ LOCAL-PLANNER($g, (c_{\mathcal{A}}, c_{\mathcal{B}}, e_{\mathcal{A}}, e_{\mathcal{B}}), G_{TCSA}(\mathcal{T})$)
11: **if** VALID-PLAN(p_{new}) **then**
12: valid $\leftarrow$ **true**
13: **end if**
14: **end while**
15: **if** valid **then**
16: $P \leftarrow$ PUSH(P, p_{new}) // add new plan to plan stack
17: $g \leftarrow g_{goal}$ of new local plan p_{new} // move to new goal configuration
18: **if** $\mathcal{T} \in S(g)$ **then** // target got assembled
19: $s \leftarrow$ success
20: **return** (s, P)
21: **end if**
22: **else**
23: **if** EMPTY(P) **then** // no configuration to fall back to
24: $s \leftarrow$ failure
25: **return** (s, P)
26: **end if**
27: $p_{pre} \leftarrow$ POP(P) // remove last plan from plan stack
28: $g \leftarrow g_{init}$ of last local plan p_{pre} // fall back to last initial configuration
29: **end if**
30: **end loop**

For a successful plan, P contains the local plans leading to the assembly of $\mathcal{T}$. Concatenating the actions of all the plans in P creates a sequence of actions that together form the global plan. Because the local plans were created by using a TCSA graph, $|P| < n$ holds true (Section 4.4). The reason for P being called a stack is the way it is used in Algorithm 3. The algorithm explores the configuration-space along $G_{TCSA}(\mathcal{T})$ in depth-first-search manner, which is done in an attempt to get closer to assembling $\mathcal{T}$ each iteration.

The algorithm starts with g_{init} as the current configuration g. At first all the connection options for g are determined with Connection-Options as described in Section 4.3. The mechanism behind this function can be viewed as a hash-table, storing the options as the value for the configuration as the key. The options need to be determined and sorted, but only the first time a configuration is encountered. The list of connection options O that Connection-Options provides is only a view on the values stored in the hash-map. When O is altered, the hash-map is updated as well. Whenever a connection option is popped from O, the option is removed from the hash-map and will therefore never be considered again.

Note that options are stored per configuration g, not for the polyomino set $S(g)$. Two configurations sharing the same polyomino set both have their own lists of connection options. We explore the configuration-space with the TCSA graph as a guidance, not the TCSA graph itself. Nodes in $G_{TCSA}(\mathcal{T})$ can be encountered multiple times and will never be eliminated from planning.

Once the list of connection options O is retrieved, the algorithm works through it in the order determined by the option sorting that was applied in advance. This is done until a valid local plan was found, or no options are left. Local-Planner uses Algorithm 1 to create a local plan p_{new}. It also takes $G_{TCSA}(\mathcal{T})$ as an input parameter, to ensure the newly added failure condition defined in Section 4.4. The validity of a local plan is evaluated with Valid-Plan.

If a valid local plan was found, p_{new} is pushed on to P and g is set to the goal configuration of p_{new}. When a configuration containing $\mathcal{T}$ is reached, the global plan is successful and the algorithm returns. On the other hand, if no valid option for g could be found, the algorithm has to fall back to the last visited configuration. For that the top local plan p_{pre} on P is popped and its initial configuration becomes the new g. Even though p_{pre} was a successful local plan, it led to a dead end and had to be removed from the stack. If P is empty the current configuration is g_{init}. This means that there is no previously visited configuration the algorithm can fall back to. In that case the algorithm states failure for assembling $\mathcal{T}$.

Before calling Algorithm 3 it is necessary to check if $\mathcal{T} \in S(g_{init})$ to state early success. Furthermore, a timeout failure is added to Algorithm 3 in case planning takes to long.

4.5.1 Complexity

Optimality Given that the local planner does not produce an optimal solution for the connection of two polyominoes, the global planner will also not reach optimality. Even if the local planner provides only optimal solutions, our depth-first-search approach would

not explore the configuration space in a way that the best sequence of local plans is guaranteed to be picked. Algorithm 3 greedily moves along the depth of the TCSA graph to assemble the target as fast as possible. The option sorting strategies provide reasonable heuristics for picking a connection option per individual TCSA node, but cannot ensure the optimal decision, let alone the optimal decision for the whole path of connection options taken. Optimal solutions need broad exploration and comparison of different paths to the target, which is infeasible in our case due to the high simulation time required for local plans.

Completeness The same as with optimality, the local planner prevents the completeness of the global planner. Assuming completeness of the local planner, the global planner could be certain of the existence or non-existence of a solution for assembling $\mathcal{T}$. Algorithm 3 will always return success or failure after simulating a finite amount of local plans. This is due to the finite number of connection options per configuration and the depth n of the TCSA graph. Each local plan in the plan stack is certain to connect at least two polyominoes, so after $n-1$ local plans the workspace contains only one n-size polyomino. This polyomino is not necessarily $\mathcal{T}$, but no further connections can be made, which makes the algorithm fall back to the last configuration. Together with the finite number of options per configuration, the algorithm will eventually explore all paths of connection options that are possible and can therefore verify the existence or non existence of a solution. Remember that this completeness is based purely on the strong assumption of a complete local planer, which is challenging to achieve in the special Euclidean group.

Efficiency We have to differentiate between local plans in the plan stack and local plans created during planning $\#local$. Even though $|P| < n$, the global planner might have created more local plans which were either invalid or had to be removed because they lead to a dead end. In a best case only one local plan could lead to the assembly of $\mathcal{T}$. This is highly unrealistic, but theoretically possible since layers of depth in the TCSA graph can be skipped. A more realistic best case would be $n-1$ local plans created during planning. This would assume that all local plans created were valid and lead directly to the target with no layer skipping.

In a worst case all paths of connection options have to be explored before stating failure. In this worst case "all" means that each connection option at each configuration produces a valid local plan with no other sub-assemblies leading to a unique new configuration. The only invalid local plans are the ones that lead to a configuration with one n-size polyomino that is not $\mathcal{T}$. It is not possible to state the exact amount of worst case local plans, since the number of connection options per configuration varies. By taking the average number of connection options per TCSA node o_μ, we can define an estimate

$$\#local_{worst} = \sum_{i=1}^{n-1} {o_\mu}^i \,. \tag{4.6}$$

For $n = 10$ and $o_\mu = 20$ this results in $\#local_{worst} \approx 5 \cdot 10^{11}$.

It is impossible to simulate that many local plans in a reasonable time. For that reason a timeout failure was added. Chapter 6 will provide experimental data on the number of $\#local$ and what percentage of global plans time out. The number of configurations explored $\#config$ is also examined in the experiments to better portray the number of dead ends during planning.

4.6 More Cubes than Target

For the global planner to work, the number of cubes in the workspace is limited to the target size n. The reason for this is linked with the use of TCSA graphs. Using a hash-table to find a TCSA node S_{TCSA} and check for equality with the configurations polyomino set $S(g)$ is simple and fast. If a configuration holds more cubes then the TCSA nodes hold, we need to check if $S_{TCSA} \subseteq S(g)$. This cannot be done by hash comparing, so all nodes of the graph need to be checked, which would be very costly. In addition to that, multiple nodes can be included in $S(g)$. The global planner could handle this by summing up the connection options of all the nodes, but again this makes planning more complex and costly.

After assembling $\mathcal{T}$ all the leftover cubes could assemble various polyominoes. We could enumerate all possible left over polyomino sets S_l and remove all of them separately from $S(g)$ to check for $S_{TCSA} = S(g) \setminus S_l$. This would again result in multiple nodes and summed up connection options, but with the ability to hash compare for equality. The number of S_l can become huge for increasing numbers of leftover cubes leading to a less efficient global planner.

5 Simulator

Our simulator modeling the behavior of magnetic modular cubes uses the 2D physics library Pymunk[1]. This library is built for the Python 3 and Python 2 environment based on the 2D physics library Chipmunk[2]. We used Pymunk, because it can be easily integrated and customized in a Python implementation. Furthermore it is light-weight and capable of running headless, but also offers an interface for Pygame[3], which we use to visualize developed motion plans and to allow user controls. As a disadvantage of a 2D simulator, we can only approximate 3D movement, in particular pivot walking. This way we trade simulation accuracy for faster simulation time, which is necessary to develop global plans in a reasonable time.

The flowchart diagram in Figure 5.1 illustrates the control flow of the simulator's simulation loop. The individual steps in the diagram are explained in this and in the following sections.

Any control program, for example a local planner or a "sandbox program" for visually controlling magnetic modular cubes with keyboard inputs, can interact with the top-level interface of the simulator. The interface provides functionalities like starting and stopping the simulation process, controlling the drawing with Pygame, or loading custom configurations and retrieving the current workspace state (Section 5.2). Two other crucial functionalities are queuing in motions for simulation and notifying the control program when a motion is done simulating. After handling the motion control, further explained in Section 5.1, the simulator enters the Pymunk-step.

The Pymunk-step is a library function, responsible for updating the simulation environment by a certain time step. The duration of a time step is a parameter that allows a trade off between simulation accuracy and simulation time. Inside the Pymunk-step forces are applied to the cubes and collision with workspace boundaries and between cubes is handled (Section 5.3).

After the Pymunk-step the magnetic forces between the cube's permanent magnets are calculated (Subsection 5.4.1). This also determines connections of cube faces that will be used to retrieve information about polyominoes in the workspace. Polyomino information is necessary to calculate forces of the global magnetic field acting on cubes (Subsection 5.4.2) and friction forces, on which we heavily rely on to simulate 3D movement like pivoting on pivot edges (Subsection 5.4.3). All the calculated forces will be applied in the Pymunk-step of the next iteration.

When drawing is enabled, the Pygame-rendering of the workspace is the last step before the next iteration begins.

[1] Pymunk: `https://www.pymunk.org/`
[2] Chipmunk: `http://chipmunk-physics.net/`
[3] Pygame: `https://www.pygame.org/`

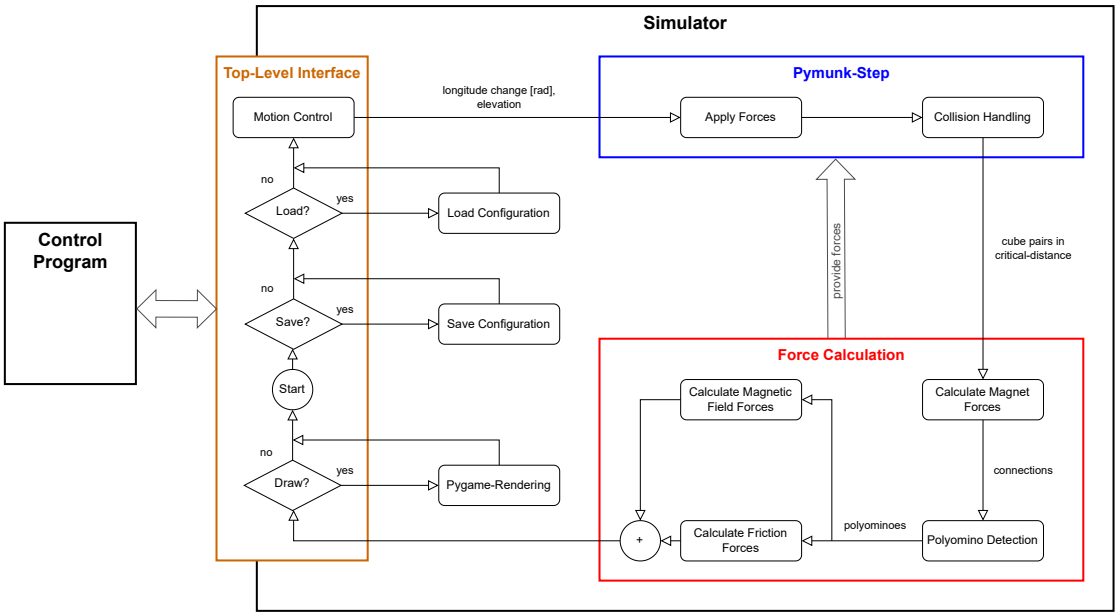


Figure 5.1: Flowchart diagram illustrating the control flow of the simulator's simulation loop. Any control program can interact with the top-level interface of the simulator. Calculated forces are provided to the Pymunk-step for the next iteration of the loop.

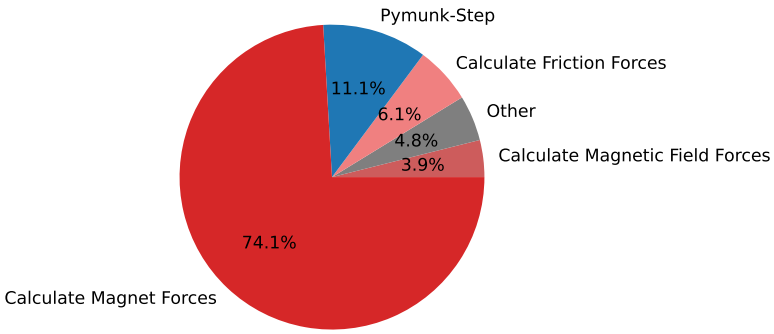


Figure 5.2: Fraction of time used on certain steps in the simulation loop. The simulator ran for 8 seconds without drawing and executed various motions with ten cubes in the workspace. The steps can be found in the flowchart diagram of Figure 5.1.

5.1 Motion Control

The motion control manages the queued-in motions from the control program and determines a change of the magnetic field for each iteration of the simulation loop. This change consists of the longitude change in radians and the latitude change called the elevation. In our simulator the elevation states if the magnetic field lays in the workspace plane, referred to as neutral, or if the magnetic north or south is pointing up. We do not specify an angular value of the latitude. The elevation just indicates if polyominoes are pivoting or not. Subsection 5.4.3 has more details.

A change of elevation is executed in a single iteration, but the angle of a rotation will be simulated by multiple longitude changes in a linear ramp with a rotational velocity we choose to set to $\frac{\pi}{8}$ rad/s. Each motion will be simulated by applying its sequence of updates with a notification to the control program when done. This makes closed loop control possible, by letting the control program wait until motions are simulated.

Motions control the magnetic field orientation and not the cubes directly. Cubes will be rotated by magnetic field forces we further explain in Subsection 5.4.2. Due to increased inertia, the larger a polyomino is, the more time it needs to align with the magnetic field, which can take longer than rotating the magnetic field itself. A certain amount of zero-updates, dependent on the size of the largest polyomino in the workspace, is added to a rotations update sequence. This way the control program will not be notified until all polyominoes are aligned with the magnetic field. For that reason, working with larger polyominoes requires more simulation time.

5.2 Workspace State

The state of the workspace is stored and updated within the Pymunk-space. By saving a configuration of the workspace, relevant attributes like position, orientation and linear and angular velocity of cubes are copied from the Pymunk-space. When loading in a configuration, the attributes of the Pymunk-space will be manipulated.

Furthermore a configuration stores magnetic field orientation and the polyominoes, together with their center of mass and pivot points. Polyominoes are stored in a custom data structure that functions both as a list of physical polyominoes and a polyomino set for the use in two-cut-sub-assembly graphs (Section 4.2). The data structure and the polyominoes themselves are hash-able for fast equality and inclusion checks.

Individual orientation and velocities of cubes will not be used for planning, but they ensure a correct loading of a configuration that was saved while in motion or when cubes were not, or not yet, aligned with the magnetic field. The alignment can be prevented by walls or other cubes, even though we assume perfect alignment with the magnetic field during planning.

5.3 Collision Handling

Collision is detected and resolved by Pymunk during the Pymunk-step. For the collision detection Pymunk uses a bounding volume hierarchy of objects in the Pymunk-space. We make use of this efficient collision detection for determining cube pairs within critical-distance. For that, each cube is surrounded by a circular sensor with a radius of half the critical-distance. A cube pair is within critical-distance if their sensors collide. We set the critical-distance to $5r_C$. More on the use of cube pairs in critical-distance in Subsection 5.4.1.

5.4 Simulating Forces

To model accurate behavior of magnetic modular cubes we calculate and apply the three most significant forces acting on cubes in the workspace, forces between permanent magnets, forces from the global magnetic field and friction with the workspace plane. Friction with the workspace boundaries and between cubes is handled by Pymunk's collision detection. The reason forces are calculated after the Pymunk-step, where they are applied, is because of the cube pairs in critical-distance determined with the collision detection of Pymunk. Applying the calculated forces in the next iteration does not effect simulation accuracy.

5.4.1 Magnet Forces

The magnetic dipole-dipole interaction of permanent magnets is the driving force for self-assembly. Cubes attract and repulse each other, which results in connection of cube faces and therefore the construction of polyominoes. In our simulator magnet force is the only thing responsible for keeping cubes connected, as it would be in a real world application of magnetic modular cubes. Nevertheless, fast rotation or hitting workspace boundaries can cause connections to break.

We can define a maximum attraction force when cubes are at a distance of $2r_C$. When such a force is reached with a valid pair of cube faces we mark the faces as connected. Polyominoes in the workspace can be identified based on the marked connections.

The majority of simulation time, about 75%, is spent on calculating magnet forces as shown in Figure 5.2. Cubes not within critical-distance are too far away to significantly affect each other with magnetic forces of their permanent magnets. The steep decline of magnetic force with increasing cube distance can be seen in Figure 5.3. We only calculate magnet forces for cube pairs within critical-distance to speed up simulation. Because assembling large target structures is the goal of global planning, it is not uncommon that most cubes are in critical-distance. For a more efficient simulation we further determine which magnet pairs to consider.

Determining Magnet Pairs For a single cube pair each of the four magnets of one cube interacts with each magnet of the other cube, resulting in 16 magnet pairs to consider

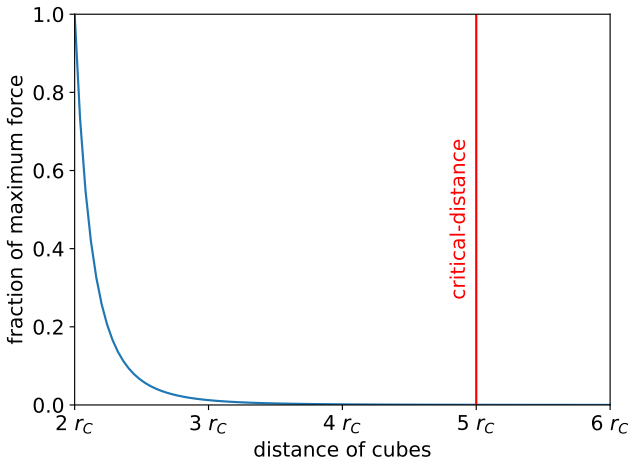


Figure 5.3: Decline of magnetic force between two permanent magnets with increasing distance between the cube centers. We reach a maximum force at $2r_C$, when the cube faces are in contact. With distances bigger than the critical-distance of $5r_C$, the fraction of force is negligible.

for a physically accurate simulation. Magnet forces of pairs within the same cube are neglected, since they do not result in any movement. Determining the first k pairs with minimal distance is a reasonable approach to not calculate all 16 pairs, since they should have the strongest force interaction. Of course $k = 1$ is most efficient, but results in a loss of real world behavior of magnetic modular cubes. We choose $k = 4$ to again find balance between accuracy and efficiency.

Calculating Magnet Force Each calculated force of a magnet pair is applied at the position of both magnets in opposite direction, resulting in either attraction or repulsion. For two magnets with moments m_1 and m_2 and a distance of r the magnet force acting from magnet 1 on magnet 2 is

$$F_{mag} = \frac{\mu_{mag}}{r^4} \left(m_2(m_1 \cdot \hat{r}) + m_1(m_2 \cdot \hat{r}) + \hat{r}(m_1 \cdot m_2) - 5\hat{r}(m_1 \cdot \hat{r})(m_2 \cdot \hat{r})\right) , \tag{5.1}$$

with $\hat{r}$ being the unit vector pointing from magnet 1 to magnet 2. The force acting from magnet 2 on magnet 1 is $-F_{mag}$ [13]. We set the strength of all magnets to $\mu_{mag} = 2.5 \cdot 10^7$.

5.4.2 Magnetic Field Forces

Magnetic field forces are responsible for aligning cubes with the longitude orientation of the magnetic field. Force is applied only at the position of north and south magnets for each cube of a polyomino. The forces at north and south magnets act in opposite directions to ensure a rotation. F_{field} is calculated based on the difference of cube orientation

γ_{cube} and magnetic field orientation γ_{field} with

$$F_{field} = \mu_{field} \begin{pmatrix} \sin(\gamma_{cube} - \gamma_{field}) \\ 0 \end{pmatrix} \mathbf{R}_{\gamma_{cube}} . \tag{5.2}$$

$\mu_{field} = 1000$ is the strength of the magnetic field. $\mathbf{R}_{\gamma_{cube}}$ is the rotation matrix used for rotating the force direction by γ_{cube} to match with the orientation of the cube. An alignment where $\gamma_{cube} = \gamma_{field}$ results in $F_{field} = \vec{0}$.

5.4.3 Friction Forces

Since we are working with a 2D-simulation, a workaround is needed to simulate pivoting of polyominoes. In the simulator a pivoting polyomino changes its center of rotation from the center of mass to either the north or the south pivot point. To achieve this, friction forces are applied at different points depending on the three states of magnetic field elevation specified in Section 5.1.

Friction forces are applied per cube at a friction-point p_{fric}, but polyomino information is still necessary to determine cubes that are in contact with the ground, called friction-cubes. In case of a neutral elevation, all cubes of the polyomino are friction-cubes and p_{fric} is the cube center. If the magnetic north is pointing up, all cubes along the south pivot-edge become friction-cubes and p_{fric} becomes the position of the south magnet for each friction-cube. If the magnetic south is pointing up, all cubes along the north pivot-edge are friction-cubes with p_{fric} being the position of the north magnet for each friction-cube. The force applied for friction-cubes is dependent on the velocity at the friction-point $v_{p_{fric}}$ and the mass of a cube m_C

$$F_{fric} = -v_{p_{fric}} \cdot m_C \cdot \frac{n}{n_{fric}} \cdot (1 - w_{nom}) . \tag{5.3}$$

n is the size of the polyomino and n_{fric} the number of friction cubes. The force is divided by n_{fric}, so that F_{fric} is distributed equally on each friction-cube. Multiplying by n accounts for the mass of all cubes in the polyomino, which is carried by the friction-cubes.

To ensure a more stable simulator, we apply a nominal friction force

$$F_{nom} = -v_{p_{fric}} \cdot m_C \cdot w_{nom} \tag{5.4}$$

to all cubes. We control the fraction of this force by $w_{nom} = 0.35$.

6 Results

All experiments conducted are about assembling target polyominoes with the use of our global planner (Chapter 4). In Section 6.1 we analyze the effect of increasing polyomino size on planning time, rotational cost and other global planner characteristics mentioned in Subsection 4.5.1. The polyominoes used for this experiment are randomly generated, but we also evaluate the construction of manually designed polyominoes in Section 6.2. With manually designed polyominoes, we can specifically test the assembly of targets with caves or holes, varying widths and heights, or different patterns of red and blue cubes. Furthermore, we experiment with different workspace sizes and aspect ratios in Section 6.3 and how the ratio of red and blue cubes affects the assembly of straight line polyominoes in Section 6.4. A legend for the box-whisker plots used to present the experimental data can be found in Figure 6.1.

Option Sorting Strategies We conducted all experiments with the three option sorting strategies from Section 4.3:

1. Minimal Distance (MIN DIST)
2. Grow Largest Component (GROW LARGEST)
3. Grow Smallest Component (GROW SMALLEST)

Instance Generation Random polyominoes and initial configurations were created with a seed-based pseudorandom number generator to make experiments reproducible. The option sorting strategies are applied to the same set of seeds to make the results comparable. When an initial configuration is randomly generated, the number of red and blue cubes matches with the target polyomino. Sub-assemblies in the initial configuration can occur.

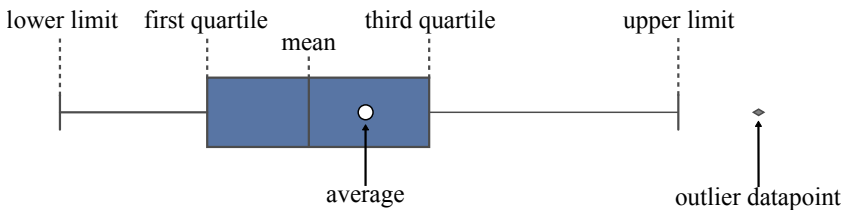


Figure 6.1: Legend for box-whisker plots used to present experimental data.

Timeout Failure The global planner states a timeout failure after a planning time of 600 seconds. We do not time out during the simulation of local plans, so instances can exceed 600 seconds and still be successful if the last local plan assembles the target polyomino.

Hardware Setup The experiments were done on multiple computers with the same hardware specification (**AMD Ryzen 7 5800X @ 8x3.8 GHz (-4.7 GHz), 128 GB RAM**) running Ubuntu 22.04.2 LTS.

6.1 Assembly for Polyomino Size

This experiment was conducted with randomly generated initial configuration and randomly generated polyominoes of specific size n. To maximize the variety of possible polyomino shapes, the number of red cubes is set to $n_{red} = \lfloor \frac{n}{2} \rfloor$ as indicated in [14]. This makes the experiment well-suited for not only analyzing planning time and rotational-cost, but also examine $\#local$, $\#config$ and $|P|$. The workspace is of size $50r_C \times 50r_C$ and for each target size 150 samples were taken.

Planning Time Figure 6.2a shows the distribution of planning time and Figure 6.2b the fraction of timed-out instances. The construction of target polyominoes with sizes 5 to 7 can be planned in under 30 seconds with just a few outliers exceeding this time. Note that none of these instances timed out.

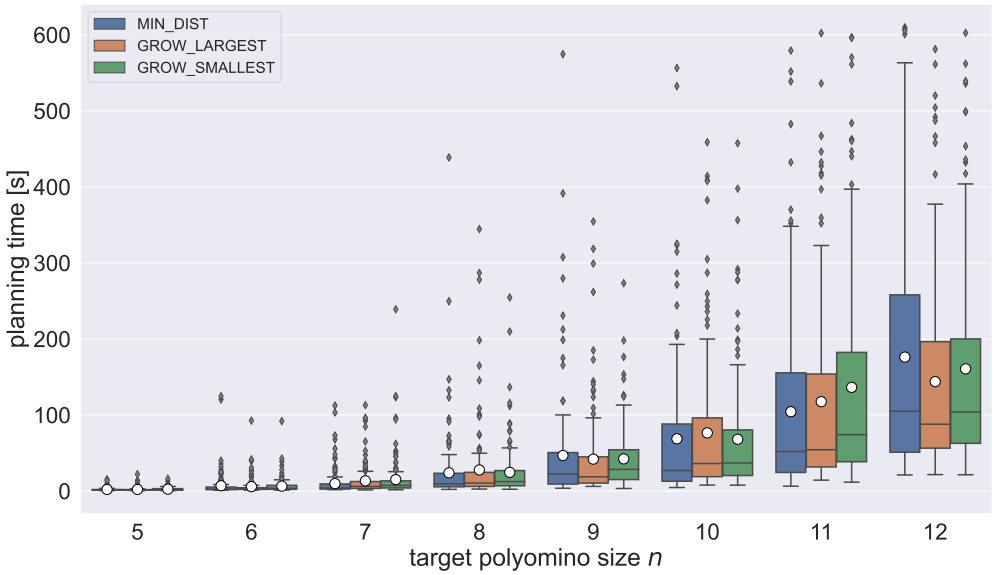
For target sizes above 7, timeout failures first appear with roughly 5% for $n = 8$, and increasing to 20% for $n = 12$. The planning time for $n = 12$ increases to 150 seconds on average with a median of 100 seconds. With increasing n a wider spread of planning time can be observed. Outliers can reach planning times close to the timeout of 600 seconds.

In terms of planning time the option sorting strategies make no noticeable difference. For the fraction of timeouts, growing the largest component often exceeds the other two strategies, clearly visible for $n = 11$, where growing the largest component is at 20% and the others under 10% of plans timed out.

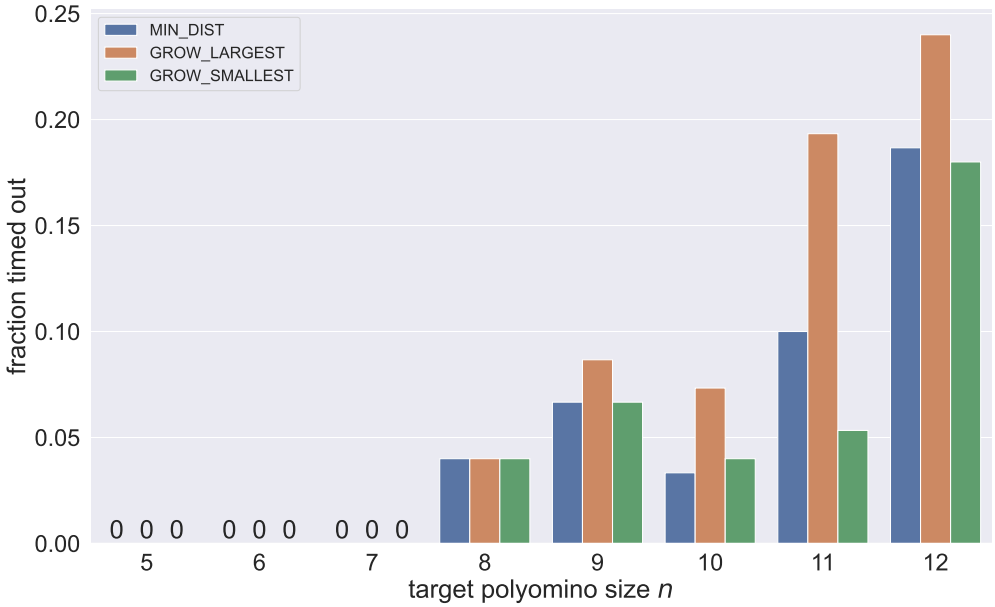
Plan Cost Figure 6.3 shows the rotational cost of plans that successfully assembled the target. The cost increased slightly for bigger polyominoes, but the gradient seems to be flattening out for sizes 11 and 12. Plan cost is generally in a range of 50 to 150 radians, which is the equivalent of 8 to 24 full longitude rotations of the magnetic field. The different option sorting strategies do not impact the cost of a plan.

Planning Attributes We analyze the number of simulated local plan $\#local$ and the number of explored configuration $\#config$ in Figure 6.4. The number of local plans contained in the plan stack of the resulting global plan $|P|$ is evaluated in Figure 6.5.

When a plan times out $\#local$ and $\#config$ only portray how many local plans and configurations can be explored within the timeout. Numbers can reach values up to



(a) Planning time in seconds. Only plans that did not time out are shown.



(b) Fraction of plans that timed out.

Figure 6.2: Planning time and fraction of timeouts for increasing target size n. All option sorting strategies are compared with 150 samples each.

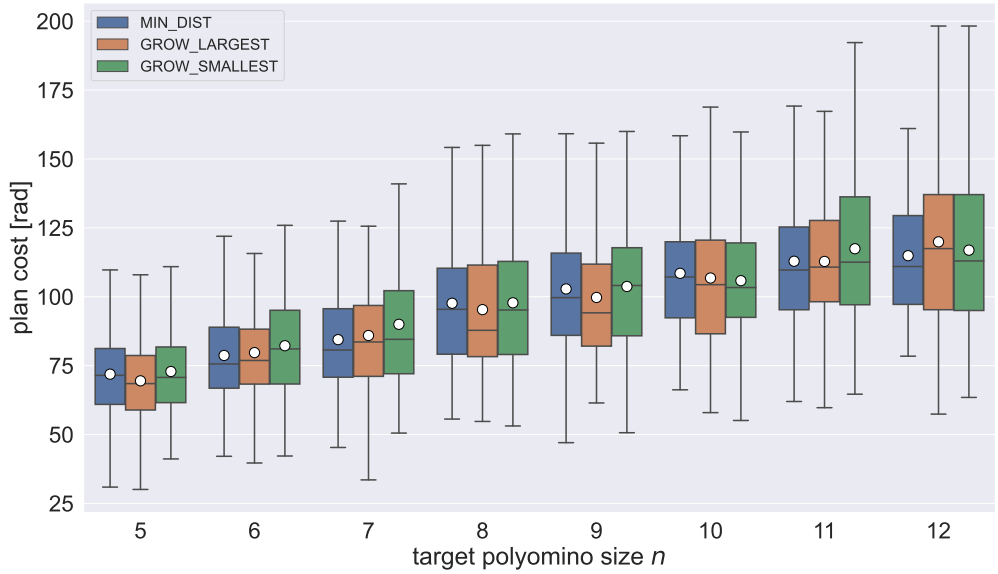


Figure 6.3: Plan cost in radians of successful plans for increasing target size n. All option sorting strategies are compared with 150 samples each and outliers are omitted.

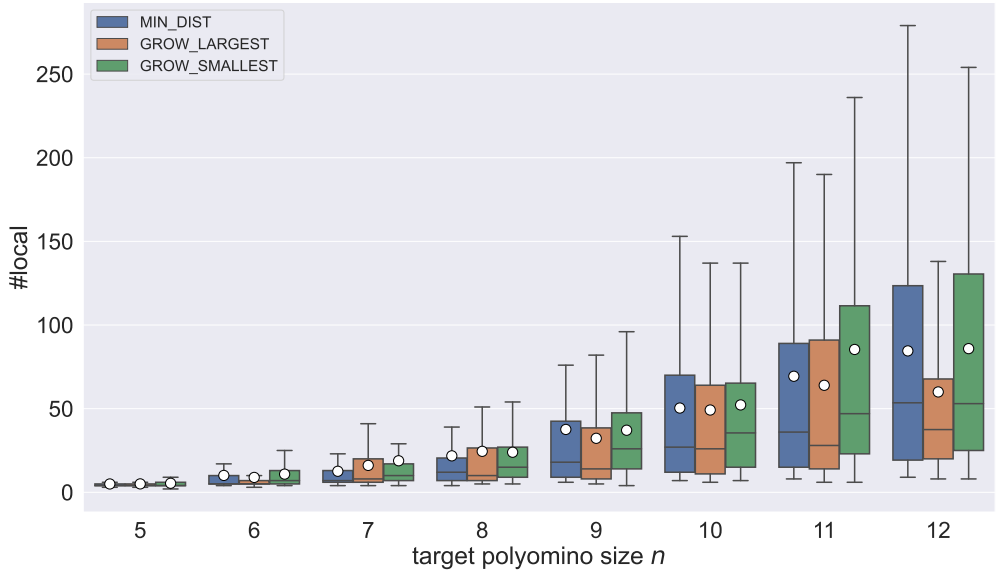
$\#local = 1200$ and $\#config = 300$. Timed-out instances are omitted in the plots of Figure 6.4.

$\#local$ increases for bigger target polyominoes. On average the realistic best case of $n-1$ local plans (Subsection 4.5.1) is exceeded. For $n = 8$ there are 25, for $n = 10$ about 50 and for $n = 12$ roughly 75 local plans simulated on average. For all n the majority of instances are below the average. Some instances can reach up to 250 local plans.

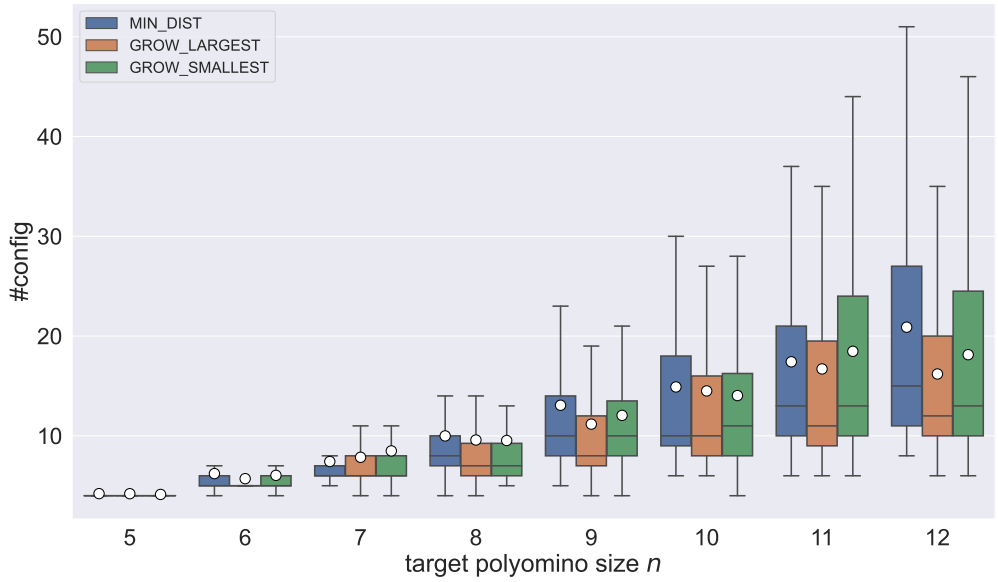
$\#config$ behaves similarly. The averages exceed the realistic best case of n, for example $\#config = 16$ with $n = 12$. In this example the global planner encountered at least 4 dead ends during planning. The small numbers of $\#config$ show that our depth-first-search approach is able to assemble polyominoes by only exploring a small portion of the whole configuration-space.

For the majority of instances the number of local plans in the plan stack is at $|P| = n - 1$. Layer skipping can be observed frequently whenever $|P| < n - 1$. Surprisingly, instances with $|P| > n - 1$ can also be observed. This should not be possible due to a TCSA graph depth of n. An explanation for this is that polyominoes break during simulation and create polyomino sets which are at the same or a lower depth than the initial set that the local planner started with. This phenomenon becomes more frequent for $n \geq 9$.

The only noticeable difference between the option sorting strategies is that growing the largest component tends to have slightly lower numbers of $\#config$.



(a) Number of simulated local plans.



(b) Number of explored configurations.

Figure 6.4: Number of simulated local plans *#local* and explored configurations *#config* for increasing target size n. Only plans that did not time out are shown and outliers are omitted for better readability. All option sorting strategies are compared with 150 samples each.

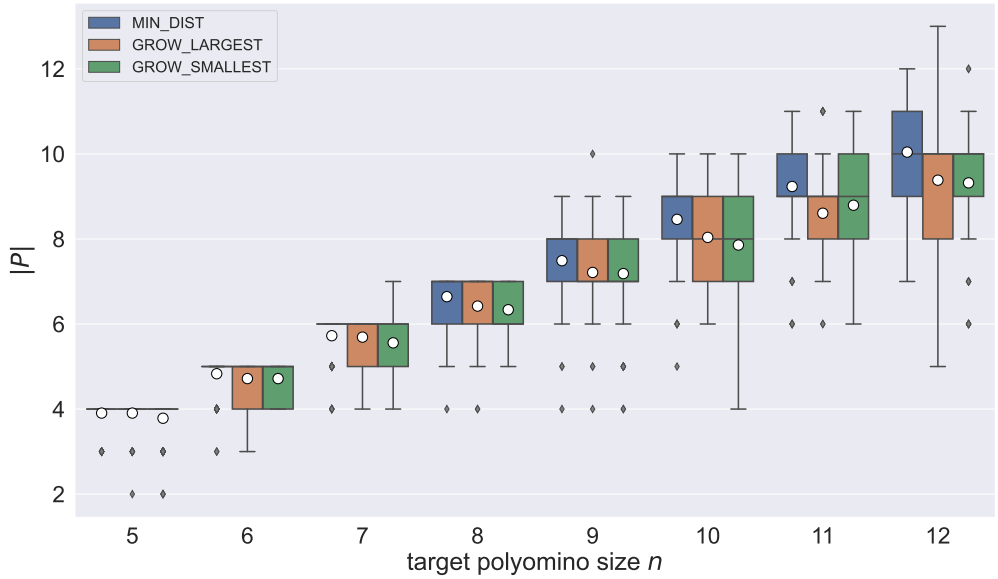


Figure 6.5: Local plans in plan stack $|P|$ for increasing target size n. Only successful plans are shown and all option sorting strategies are compared with 150 samples each.

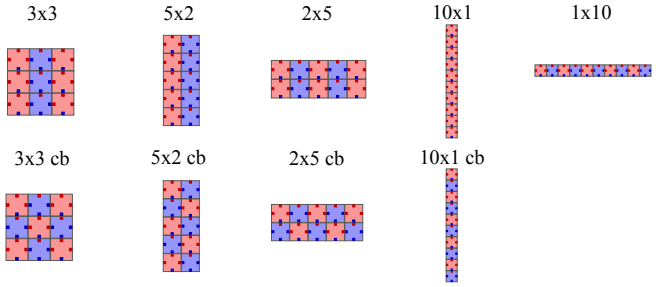
6.2 Assembly of Custom Polyominoes

In this experiment manually designed polyominoes are assembled from multiple randomly generated initial configurations. 100 samples were taken for each custom polyomino with a workspace size of $50r_C \times 50r_C$.

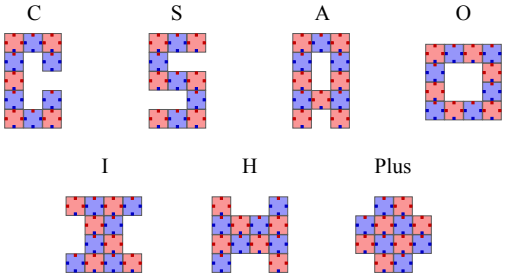
In Subsection 6.2.1 we focus on how rectangular polyominoes with varying width/height ratios influence planning time. Furthermore we experiment with two patterns of red and blue cubes for each polyomino. The *switching-column pattern* switches between red and blue cubes column-wise and the *checkerboard pattern* creates a checkerboard of single red and blue cubes. A list of these polyominoes can be found in Figure 6.6a.

In Subsection 6.2.2 the assembly of special polyomino shapes, that are listed in Figure 6.6b, is examined. The polyominoes "C", "S", "A" and "O" contain caves and/or holes of different sizes, but are thin shapes with fewer connections. They more or less consist of a one-cube-thick line. The polyominoes "I", "H" and "Plus" are thick shapes with many connections, but still contain caves or are at least not rectangular. All these polyominoes are build with the checkerboard pattern to achieve equal amounts of red and blue cubes. The size of all polyominoes, except for "C", is $n = 12$.

In Subsection 6.2.3 we analyze the polyomino attributes that are especially challenging for the global planner based on all data gather in our experiments.



(a) Rectangular polyominoes evaluated in Subsection 6.2.1. The checkerboard pattern is labeled with "cb".



(b) Special polyomino shapes evaluated in Subsection 6.2.2.

Figure 6.6: List of manually designed polyominoes for experimenting.

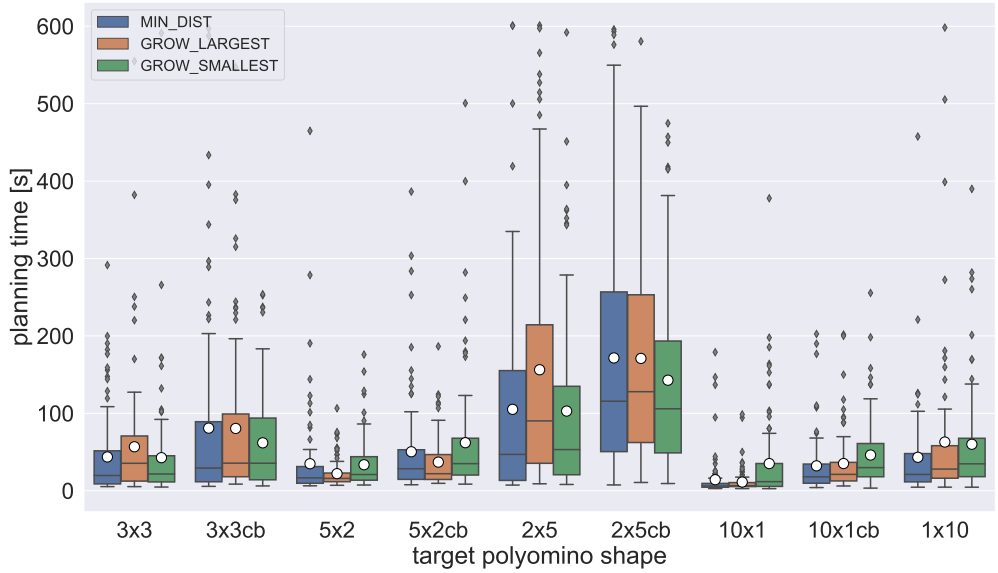
6.2.1 Width/Height and Cube Pattern

When comparing planning time of the two cube pattern in Figure 6.7a, the checkerboard pattern performs worst for all types of rectangular polyominoes. It is not a huge difference, but still noticeable. For instance, the "3x3" polyomino is on average at 50 seconds planning time, while the "3x3 cb" polyomino is at 75 seconds with a wider spread and worst outliers.

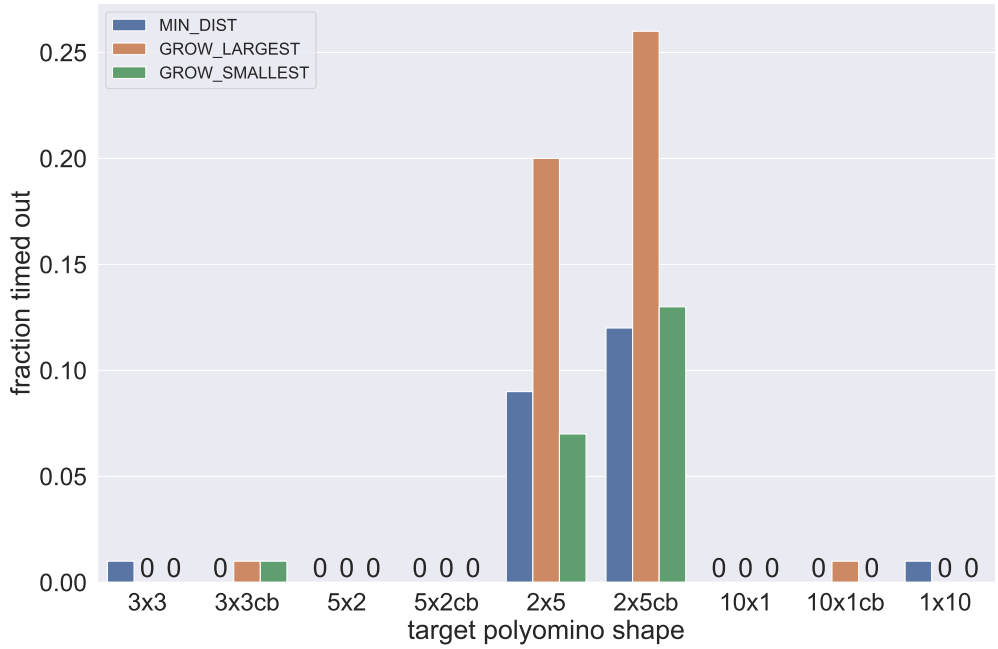
Polyomino shapes with more height than width are faster to assemble. "10x1" is the best followed by "5x2", "3x3" and "2x5". The same order persists for the checkerboard pattern. Surprisingly, the "1x10" polyomino breaks out of this order. Its planning time lays between the "5x2" and the "3x3".

The "2x5" performs significantly worst than all other polyominoes. While the majority of instances for all other shapes can be solved in under 100 seconds, the "2x5" exceeds this time with a spread reaching up to 600 seconds.

For the fraction of timed-out plans shown in Figure 6.7b, the "2x5" is the only shape with 10% to 20%, depending on the option sorting strategy. All other shapes experience nearly no timeouts.

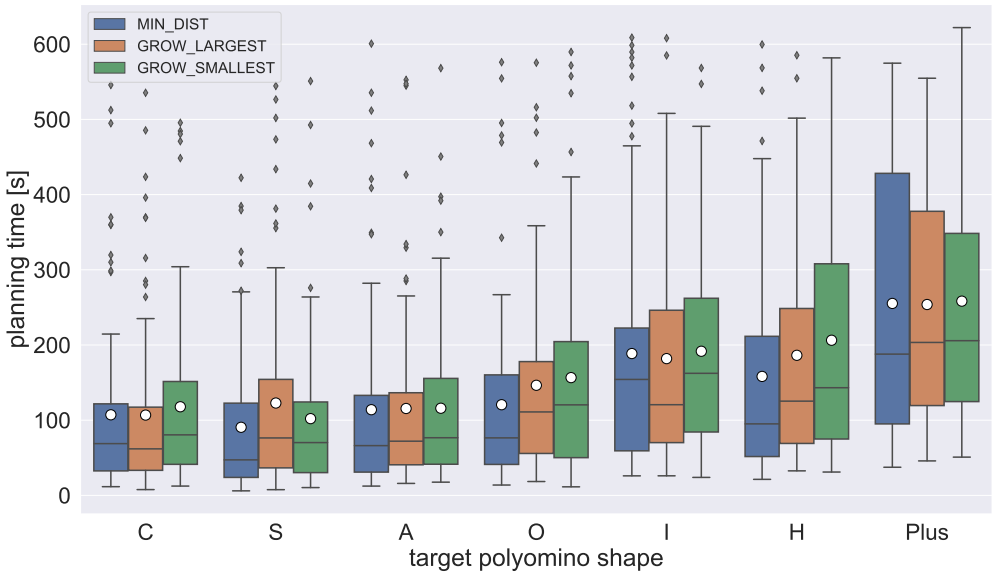


(a) Planning time in seconds. Only plans that did not time out are shown.

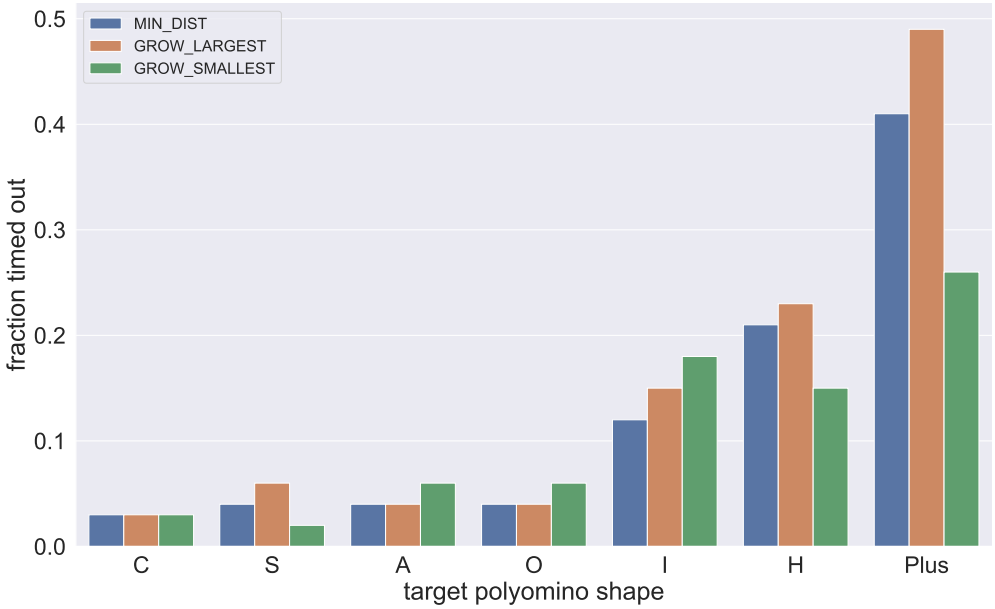


(b) Fraction of plans that timed out.

Figure 6.7: Planning time and fraction of timeouts for rectangular polyominoes listed in Figure 6.6a. All option sorting strategies are compared with 100 samples each.



(a) Planning time in seconds. Only plans that did not time out are shown.



(b) Fraction of plans that timed out.

Figure 6.8: Planning time and fraction of timeouts for special polyominoes listed in Figure 6.6b. All option sorting strategies are compared with 100 samples each.

6.2.2 Special Polyomino Shapes

The planning time and fraction of timeouts are evaluated in Figure 6.8. Constructing the thin shapes “C”, “S”, “A” and “O” is comparable in terms of planning time with 100 seconds on average, which is below the average of randomly generated 12 size polyominoes, already examined in Figure 6.2a. The fraction of timeouts is mostly under 5%, which is a huge difference to 25% for the random polyominoes evaluated in Figure 6.2b. The “O” shape has the worst performance of the four, which could be related to it being the only shape with a width of 4. We already observed how a more horizontally stretched shape is harder to assemble with the “5x2” and “2x5” rectangles.

The three thick shapes perform much worst with an average of 200 seconds planning time for “I” and “H” and 250 seconds for the “Plus” shape. The instances have a wide spread in distribution of planning time and timeouts reach 20% for “I” and “H” and even 30% to 50% for the “Plus” shape. Assembling the “Plus” polyomino holds the worst performance out of all custom and random polyominoes evaluated in this thesis.

Caves and holes have no impact on the performance of the global planner. The option sorting strategies do not show any recognizable pattern, but strong differences between them can be observed. Growing the smallest component, while assembling the “Plus” shape reduces the fraction of timeouts by half, compared to the others.

6.2.3 Hardest Shapes to Assemble

By evaluating all custom polyominoes from Figure 6.6, we can observe which parameters have the most influence on planning time. The number of connections between cubes within a polyominoes seems to increase planning time. This makes sense since more connections provide more possible two-cuts, which increases complexity of a TCSA graph. But connectivity alone is not the driving factor for shapes like the “2x5”. The “5x2” provides an equal amount of connections and can be assembled very efficiently with no timeouts, while the “2x5” performs much worse. Width alone is also not responsible either, since the “1x10” performs better than the “2x5”.

The global planner becomes inefficient, when connectivity and increasing width is combined. The reason for this could be located within the local planner. North-south connections become problematic when the connection-cubes have to move above or below non-connection-cubes to get to their desired position. A small example of this can be seen in Figure 3.4, where the local plans c) and d) fail, because they connected to the wrong cubes, while trying to slide above them. With increasing width the effects become more dramatic.

Increasing the offset distance when aligning the connection cubes (Section 3.1) is not a solution since a north-south connection cannot be established when magnetic attraction becomes to weak. Choosing the right option sorting strategy might make a difference, but it is hard to predict the best strategy for a specific polyomino. Other option sorting strategies could be developed to take these problematic connections into account and prefer the unproblematic ones.

Workspace	Width × Height
S, 1 : 1	$35r_C \times 35r_C$
M, 1 : 1	$50r_C \times 50r_C$
L, 1 : 1	$65r_C \times 65r_C$
S, 2 : 1	$50r_C \times 25r_C$
M, 2 : 1	$70r_C \times 35r_C$
L, 2 : 1	$90r_C \times 45r_C$
S, 3 : 1	$60r_C \times 20r_C$
M, 3 : 1	$90r_C \times 30r_C$
L, 3 : 1	$105r_C \times 35r_C$

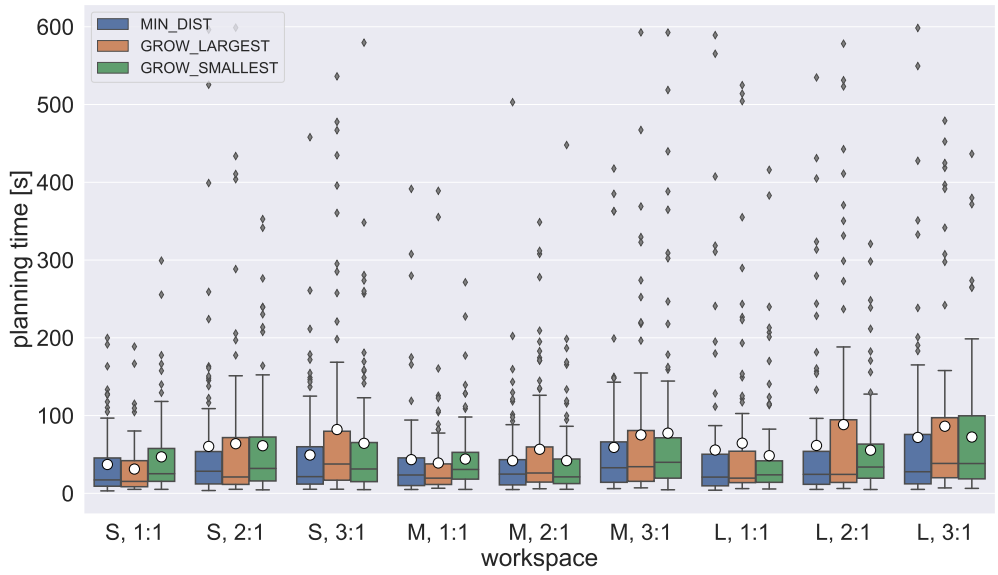
Table 6.1: Workspace variations with different areas and aspect ratios.

6.3 Assembly in different Workspaces

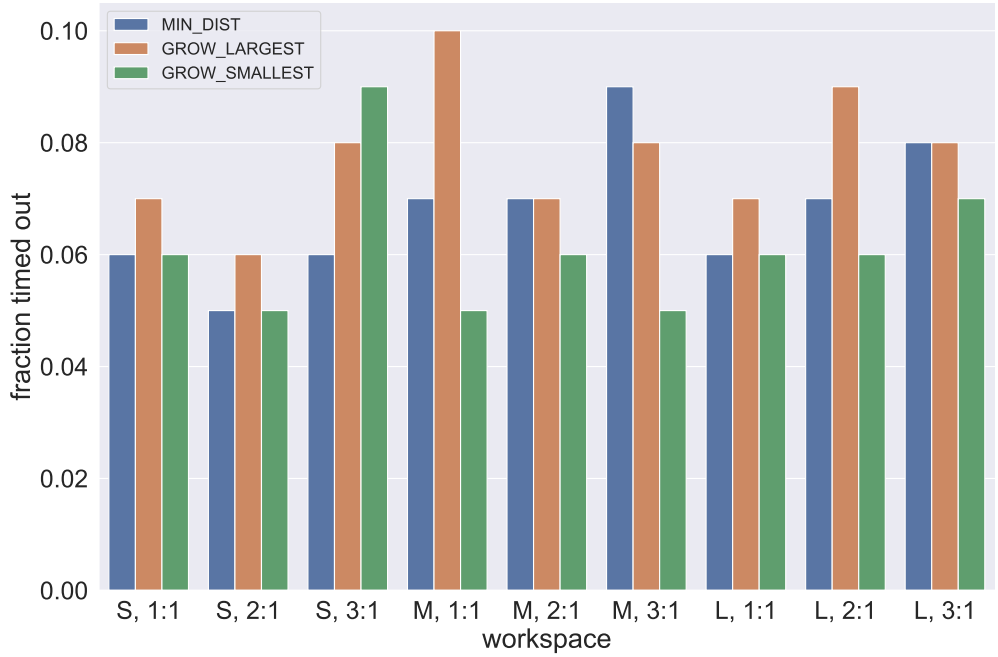
In this experiment we tested the assembly of randomly generated polyominoes of size $n = 9$ with random initial configurations in various rectangular workspaces. We choose three workspace sizes (S, M, L) in three different aspect ratios (1 : 1, 2 : 1, 3 : 1) each. All aspect ratios for one size result in roughly the same area. The workspaces with their exact widths and heights are listed in Table 6.1. A workspace with aspect ratio $1 : x$ would produce similar results to one with aspect ratio $x : 1$, since the magnetic field can be rotated freely. The maximum width or height of a polyomino with size 9 is $18r_C$. We ensured that such a polyomino could fit in all workspace variations while being able to rotate 360 degrees without getting stuck. We analyze the affect of these workspace variations on planning time and rotational cost.

Planning Time Figure 6.9a shows the planning time for all workspace variations. In terms of the size within one class of aspect ratio no significant effect can be observed. Narrowing down the workspace by increasing the aspect ratio seems to increase planning time slightly, but for the majority of instances planning time stays under 100 seconds. The fraction of timeouts in Figure 6.9b remains constant for all workspaces and the options sorting strategies do not show any difference as well.

Plan Cost It is not surprising that the rotational cost, presented in Figure 6.10, increases with bigger workspace areas. Cubes and walls are further apart, which results in more pivot walking cycles necessary to assemble polyominoes. Within a class of same size, increasing the aspect ratio results in slightly more rotational cost as well.



(a) Planning time in seconds. Only plans that did not time out are shown.



(b) Fraction of plans that timed out.

Figure 6.9: Planning time and fraction timeouts for different workspace variations listed in Table 6.1. All option sorting strategies are compared with 100 samples each.

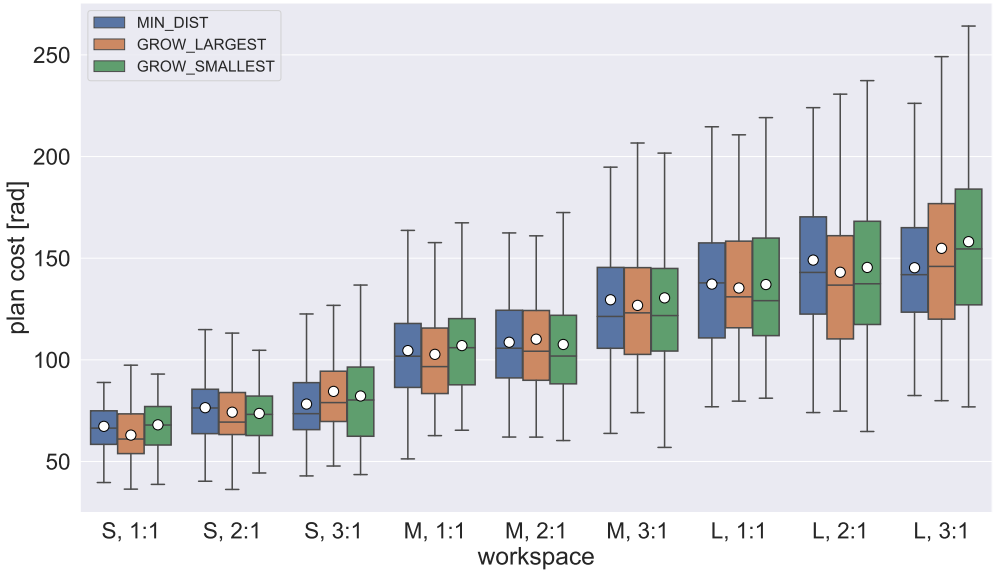


Figure 6.10: Plan cost in radians of successful plans for different workspace variations with varying areas and aspect ratios. All option sorting strategies are compared with 100 samples each.

6.4 Assembly for Red and Blue Cube Ratio

We examined the effect of red and blue cube ratio on planning time. For this we increased the number of red cubes $0 \leq n_{red} \leq \lfloor \frac{n}{2} \rfloor$ while keeping the target polyomino size fixed at $n = 10$. With $n_{red} = 0$, only north-south connections allow the creation of just a vertical line polyomino. $n_{red} = \lfloor \frac{n}{2} \rfloor$ holds the biggest variety of polyomino shapes [14]. To exclude the influence of varying polyomino shapes on the experiment, the shape is set to a 10×1 polyomino. $\lfloor \frac{n}{2} \rfloor < n_{red} \leq n$ is equal to $0 \leq n_{blue} \leq \lfloor \frac{n}{2} \rfloor$. Conducting the experiment with n_{red} or n_{blue} in equivalent. The patterns of red and blue cubes within the 10×1 polyomino and the initial configurations are randomly generated in a workspace of size $50r_C \times 50r_C$. For each number of red cubes 100 samples were taken.

Planning Time Figure 6.11 shows the distribution of planning time for this experiment. By increasing n_{red} from 0 to 1 a clear increase in planning time is visible. With only blue cubes every cube can be placed everywhere in the polyomino. By introducing one red cube, position becomes important. Further increasing the number of n_{red} does not affect planning time significantly. For these vertical straight line polyominoes growing the smallest component performs slightly worst than the other two option sorting strategies. Nearly no instances timed out during planning.

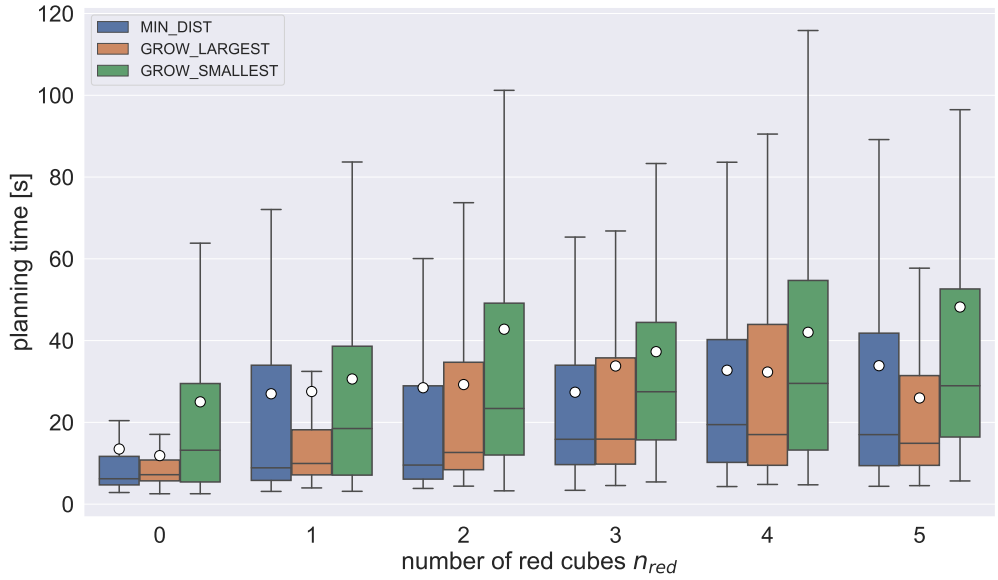


Figure 6.11: Planning time for different numbers of red cubes n_{red}. All option sorting strategies are compared with 100 samples each. The fraction of timeouts is rarely at 2% and is therefore not plotted.

7 Conclusion

In this thesis we developed a heuristic approach for the motion planning problem of assembling polyominoes with magnetic modular cubes [5] in the 2-dimensional special Euclidean group $SE(2)$.

Although our simulator is not a physically accurate representation of magnetic modular cubes, since we are simulating 3D-movement in a 2D-environment, it is able to depict continuous movement of rotations and pivot walking and also simulates magnetic attraction and repulsion of embedded permanent magnets. While doing so, collision between cubes and collision with workspace boundaries is detected and resolved.

These attributes of the simulator allow our closed-loop local planning algorithm to dynamically adjust for events like structures blocking each other, structures sliding along the workspace boundaries and varying movement directions due to different pivot walking displacement vectors of polyomino shapes. By not limiting rotations to certain degrees, structures can always be aligned and theoretically connect by pivot walking a straight path. Above mentioned events prevent this straight and optimal movement, but dynamic realigning provides a good heuristic for minimizing movement while being efficient on planning time.

We constrained the workspace to be rectangular with no obstacles except the outer boundaries and experimented with different sizes and aspect ratios of the rectangle. Our local planner is not designed to handle obstacles. Designing a local planner able to navigate around obstacles and handle pivot walking displacement and sliding on walls in a more calculated way could be a interesting direction for future work.

The simulator is balanced between physical accuracy and efficiency, but it remains a high fidelity physics simulation. Simulating movement is costly and local plans require planning times in a range of seconds. On a global scale of doing multiple local plans to assemble desired target structures, simulation should be avoided as much as possible. Using classical motion planning approaches that broadly explore the configuration-space like RRT, are not feasible under this condition.

Our global planner uses the ability of two-cutting polyominoes to create a two-cut-sub-assembly graph that will be used as a building instruction for target polyominoes. The configuration-space is explored by depth-first-search traversing this graph, to get closer to the target with each local plan. The graph leaves multiple options for traversing one edge, because it does not consider workspace position of polyominoes. We evaluated three strategies of sorting these options by best probable outcome.

The global planner can identify if the assembly of a target is possible out of any sub-assemblies present in the workspace at any point in time, but requires equal amounts of cubes in the workspace and in the target polyomino. How to work with more cubes

than necessary for the assembly, when two-cut-sub-assembly graphs are used, remains an open question for future work with some insights on the problem given in Section 4.6.

We evaluated the assembly of polyominoes with up to 12 cubes in varying shapes and patterns of cube types. Planning time and timeout failures increase exponentially with the number of cubes, as it is expected with increasing dimensionality of the configuration-space. We are able to solve the majority of instances in well under 200 seconds, but certain attributes of polyominoes heavily decrease efficiency of the global planner. We found out that many connections within a polyomino combined with increasing polyomino width produces especially bad results.

The option sorting strategies seem to perform differently for varying shapes, but we were not able to identify a clear pattern. Studying attributes of polyominoes and their effect on performance is another possible direction for future work. Designing new specialized option sorting strategies and determining which on to use, based on the target polyomino, looks promising.

Using our global planner to calculate motion sequences and applying them to a real workspace of magnetic modular cubes will most likely not result in a successful target assembly, because the mismatch between simulation and the real world is too big. Instead, the simulator could be replaced by a computer vision-based feedback and control system of the workspace, like the one currently developed by Lu et al. [15]. Resetting configurations to previous states is not possible in the real world. Although we are resetting configurations in our global planner, we are already trying to avoid unnecessary simulation as much as possible. Further optimizing the two-cut-sub-assembly graph traversal to make even more careful decisions could be promising for a real world applications of magnetic modular cubes and actual hardware experiments would be interesting to see.

Bibliography

[1] P. K. Agarwal, B. Aronov, T. Geft, and D. Halperin. On two-handed planar assembly partitioning with connectivity constraints. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1740–1756. SIAM, 2021.

[2] A. Becker, E. D. Demaine, S. P. Fekete, G. Habibi, and J. McLurkin. Reconfiguring massive particle swarms with limited, global control. In P. Flocchini, J. Gao, E. Kranakis, and F. Meyer auf der Heide, editors, *Algorithms for Sensor Systems*, pages 51–66, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[3] A. Becker, E. D. Demaine, S. P. Fekete, and J. McLurkin. Particle computation: Designing worlds to control robot swarms with only global signals. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6751–6756, 2014.

[4] A. T. Becker, S. P. Fekete, P. Keldenich, D. Krupke, C. Rieck, C. Scheffer, and A. Schmidt. Tilt assembly: Algorithms for micro-factories that build objects with uniform external forces. *Algorithmica*, 82(2):165–187, Feb 2020.

[5] A. Bhattacharjee, Y. Lu, A. T. Becker, and M. Kim. Magnetically controlled modular cubes with reconfigurable self-assembly and disassembly. *IEEE Transactions on Robotics*, 38(3):1793–1805, 2022.

[6] J. Bishop, S. Burden, E. Klavins, R. Kreisberg, W. Malone, N. Napp, and T. Nguyen. Programmable parts: A demonstration of the grammatical approach to self-organization. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3684–3691. IEEE, 2005.

[7] P. Blumenberg, A. Schmidt, and A. T. Becker. Computing motion plans for assembling particles with global control. In *Under Review*. IEEE, 2023.

[8] D. Caballero, A. A. Cantu, T. Gomez, A. Luchsinger, R. Schweller, and T. Wylie. Hardness of reconfiguring robot swarms with uniform external control in limited directions. *Journal of Information Processing*, 28:782–790, 2020.

[9] G. P. Jelliss. Concrete mathematics, a foundation for computer science, by ronald l. graham, donald e. knuth and oren patashnik. pp 625. £24·95. 1989. isbn 0-201-14236-8 (addison-wesley). *The Mathematical Gazette*, 75(471):117–119, 1991.

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

[11] S. M. LaValle et al. Rapidly-exploring random trees: A new tool for path planning. 1998.

[12] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects: Steven m. lavalle, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan. *Algorithmic and computational robotics*, pages 303–307, 2001.

[13] M. H. Levitt. *Spin dynamics: basics of nuclear magnetic resonance*. John Wiley & Sons, 2013.

[14] Y. Lu, A. Bhattacharjee, D. Biediger, M. Kim, and A. T. Becker. Enumeration of polyominoes and polycubes composed of magnetic cubes. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6977–6982, 2021.

[15] Y. Lu, A. Bhattacharjee, C. C. Taylor, J. Leclerc, J. M. O'Kane, M. J. Kim, and A. T. Becker. Closed-loop control of magnetic modular cubes for 2d self-assembly. 2023.

[16] A. Mueller. Modern robotics: Mechanics, planning, and control [bookshelf]. *IEEE Control Systems Magazine*, 39(6):100–102, 2019.

[17] R. Pelrine, A. Wong-Foy, A. Hsu, and B. McCoy. Self-assembly of milli-scale robotic manipulators: A path to highly adaptive, robust automation systems. In *2016 International Conference on Manipulation, Automation and Robotics at Small Scales (MARSS)*, pages 1–6. IEEE, 2016.

[18] W. Saab, P. Racioppo, and P. Ben-Tzvi. A review of coupling mechanism designs for modular reconfigurable robots. *Robotica*, 37(2):378–403, 2019.

[19] A. Schmidt, V. M. Baez, A. T. Becker, and S. P. Fekete. Coordinated particle relocation using finite static friction with boundary walls. *IEEE Robotics and Automation Letters*, 5(2):985–992, 2020.

[20] A. Schmidt, S. Manzoor, L. Huang, A. T. Becker, and S. P. Fekete. Efficient parallel self-assembly under uniform control inputs. *IEEE Robotics and Automation Letters*, 3(4):3521–3528, 2018.

[21] M. Sitti, H. Ceylan, W. Hu, J. Giltinan, M. Turan, S. Yim, and E. Diller. Biomedical applications of untethered mobile milli/microrobots. *Proceedings of the IEEE*, 103(2):205–224, 2015.

[22] P. J. White and M. Yim. Scalable modular self-reconfigurable robots using external actuation. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2773–2778. IEEE, 2007.

[23] E. Winfree. *Algorithmic self-assembly of DNA*. California Institute of Technology, 1998.