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 9, 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 9, 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 magnetic-cubes auf der Stelle gedreht oder durch pivot-walking 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 magnetic-cubes nah genug beieinander können sich diese durch die Permanentmagneten miteinander verbinden und so Polyominos als größere Strukturen aufbauen, welche auf die gleiche Weise wie einzelne cubes gesteuert werden können. 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 motion-planner für die beschriebenen magnetic-cubes zu entwerfen, welcher mit beliebigen Positionen und Rotationen umgehen kann. Dabei ist es erforderlich, eine Simulationsumgebung zu schaffen, welche das Verhalten der magnetic-cubes repliziert. Es soll ein lokaler motion-planner entwickelt werden, um zwei Polyominos an gewünschten Kanten zu verbinden. Dieser local-planner soll Heuristiken und optimale Bewegungsabläufe mit möglichst wenig Schritten realisieren. Ebenfalls soll dieser global eingesetzt werden, um Bewegungsabläufe zu finden, die gewünschte Polyominos 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 motion-planning 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 direction. 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. 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, MrK̇eune'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 and computing optimal motion plans 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

Abstract

Contents

List of Figures

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 [20], or a robot swarm could be used for milliscale and microscale manufacturing [16].

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 [21]. Therefore, the use of external global control, effecting every robot in the workspace with the same torque and force, is a promising solution [21]. 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 [17].

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 develop a simulation that simulates the behavior of magnetic modular cubes, without assuming discrete movement or limiting rotations to a certain amount. The simulation will be used for developing closed-loop planing algorithms, which provide a control sequence to assemble desired target shapes. For that it is necessary to develop a local planner that is able to connect structures at desired faces. We will look at the difficulties and problems that occur, when working 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 [15] 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, and therefore these algorithms can be called resolution complete. Ways of sampling include random sampling or using a grid with a resolution that is dynamically adjustable. 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 [22] 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 used in this thesis 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 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 [19].

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 [13].

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 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 polyominoes in arbitrary environments, when cubes obey the tilt model. He provides different algorithmic approaches using various distance heuristics and even a solution making use of RRTs. For that he follows the rules of Tilt in a discrete setting.

1.2 Contribution

2 Preliminaries

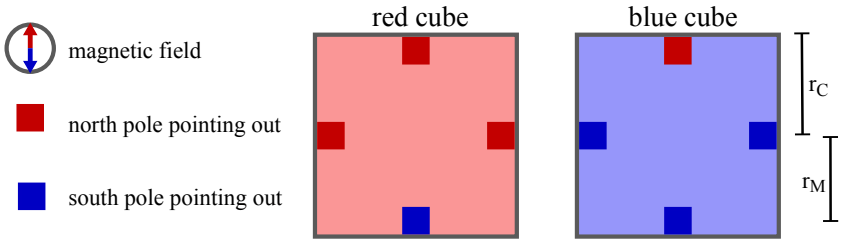

Figure 2.1: Simplified top-down view of the two magnetic modular cube types with their outward pointing magnet poles, illustrated as red and blue squares. Also visualizes the lengths r_C and r_M.

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 this 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, and all the other faces can also be called by their corresponding cardinal direction. For each face we define a vector $\vec{e} \in \{\vec{N}, \vec{E}, \vec{S}, \vec{W}\}$ with $\|\vec{e}\| = 1$ pointing in the cardinal direction of the magnetic field. For simplification we refer to magnets by their outwards pointing pole in further sections.

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]. Two important lengths that we use for planning and simulating 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.

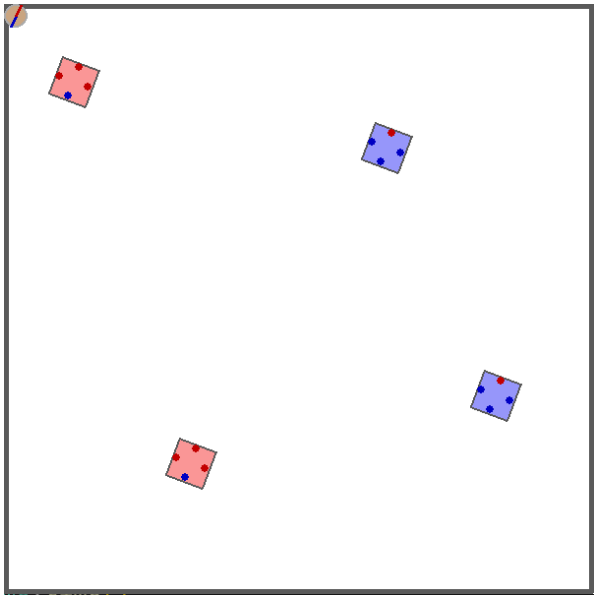

Figure 2.2: Rectangular workspace with a configuration of four magnetic modular cubes. All cubes have the same orientation as the magnetic field, indicated by the compass in the top-left corner.

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 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 we can still assume only one orientation for n cubes, because we are working with a global magnetic field orienting all cubes the same way. We assume that eventually all cubes align with the global magnetic field and only consider static configurations. Future work could examine the dynamic configuration, which would enable faster motion planning, but a more complex planning problem. Figure 2.2 shows a configuration with four cubes in the workspace. It is irrelevant which exact physical cube is at which position as long as

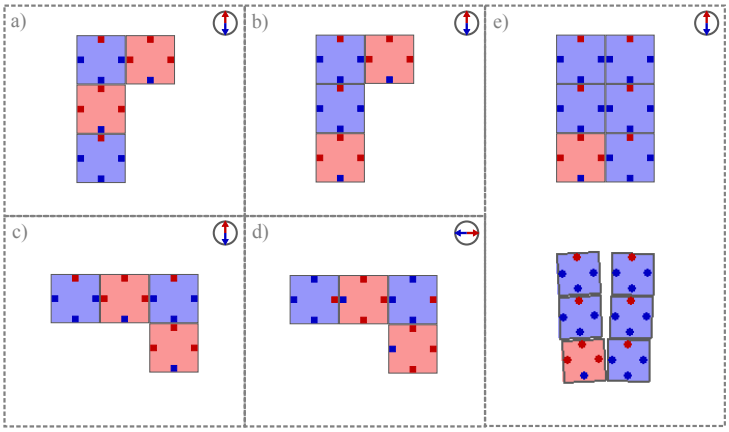

Figure 2.3: Examples of polyominoes and their equality. a) and d) are equal, only the magnetic field changed its orientation. a) and c) are not equal, they have the same shape, but it is rotated. a) and b) are also not equal because of different cube types in the same shape. e) shows an invalid polyomino in its grid representation (top) and how it behaves in the simulation (bottom).

they are the same type, so switching the positions of the two red cubes in Figure 2.2 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. Because we work with arbitrary positions and orientations the grid alignment does not hold true for multiple polyominoes in the workspace, but for each polyomino on its own the cubes can be represented in a local coordinate system with position (x, y), $x, y \in \mathbb{Z}$ [13].

We consider *fixed polyominoes*, meaning that two polyominoes are distinct if their shape or orientation are different [13]. The magnetic field always provides an orientation, so in Figure 2.3 a) and d) the polyominoes are equal, just the magnetic field is rotated. Conversely, the polyominoes in Figure 2.3 a) and c) are the same shape but with a different rotation under the same magnetic field orientation, so they are not equal. Furthermore, two polyominoes are only equal if all the cubes at equal positions are the same type. The polyominoes in Figure 2.3 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

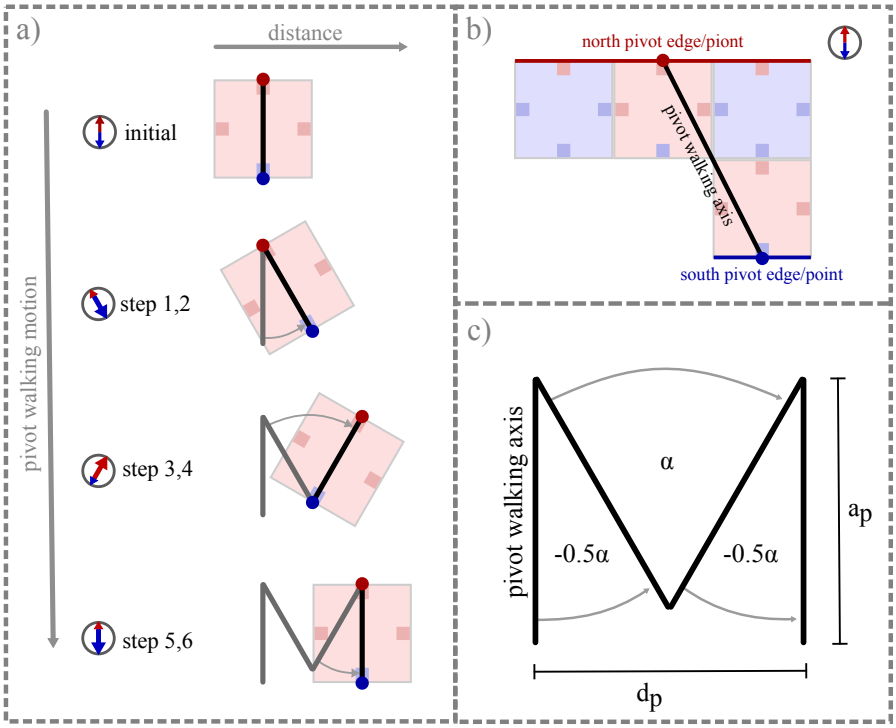

Figure 2.4: This figure describes the pivot walking motion in detail. a) shows the six pivot walking steps for a single red cube. You can see the orientation of the magnetic field (bigger arrow indicates elevation, so in step 1,2 the south pole is raised in the air). In b) an example polyomino with its pivot axis, edges and points is shown. c) illustrates the rotation of the pivot axis labeled with all the pivot walking parameters.

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 often 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 2.3 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 cubes the same type and is therefor marked as an invalid polyomino.

2.4 Motion Modes

In [5] three motion modes are presented. Rotation, 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

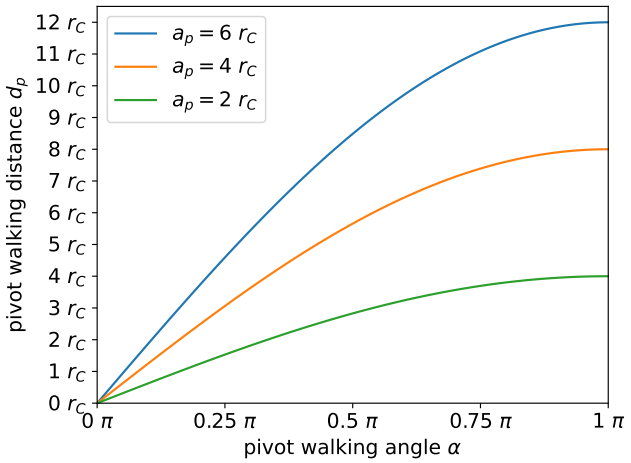

Figure 2.5: Functions of the pivot walking distance d_p based on pivot walking angle α for different pivot walking axes with length a_p. Lengths are given in multiples of cube radius r_C.

and we consider this motion a normal rotation.

Rotating the magnetic field perpendicular to the workspace plane, cubes can roll forwards or backwards. This rolling motion becomes problematic for self-assembly, because the top and bottom face of the cube, which contain no magnets, can become a side face. Because rotation and pivot walking are sufficient to reach any position in the workspace, we do not consider rolling in our simulation 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 2.4 b).

pivot walking: Not rotating around the center of mass is important 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 cycle steps in Figure 2.4 a) and have a closer look at its parameters in Figure 2.4 c).

After one pivot walking cycle, the polyomino has moved by a displacement vector $\vec{d}$

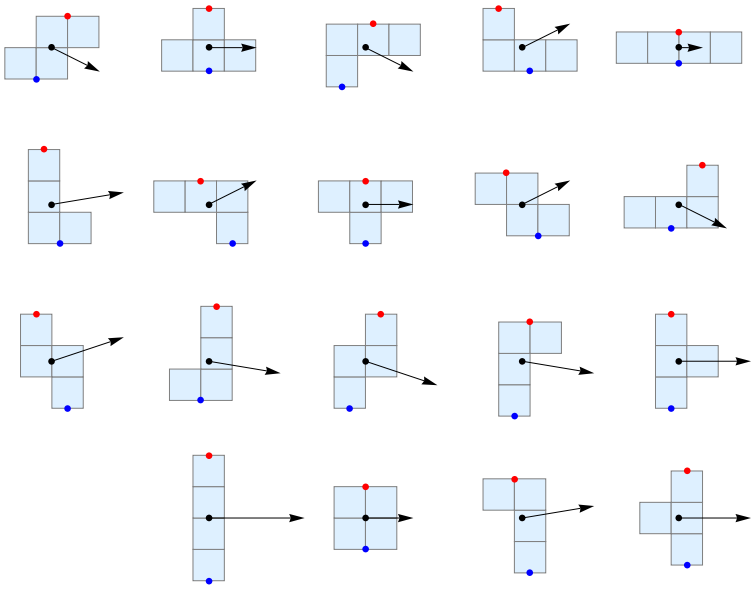

Figure 2.6: All 19 four-cube polyomino shapes with their displacement vector $\vec{d}$ for one pivot walking cycle with $\alpha = \frac{\pi}{4}$. $\vec{d}$, drawn with a black arrow from its center of mass. North and south pivot point are drawn as red and blue dots.

with $\|\vec{d}\| = d_p$, so d_p is the distance the polyomino moved. The direction and length of $\vec{d}$ changes with the shape of the polyomino. The movement is always perpendicular to the pivot walking axis $\vec{a}$ with $\|\vec{a}\| = a_p$, which is the vector between the north and the south pivot point, visualized in Figure 2.4 b). d_p can be calculated as

$$d_p = 2 \cdot \sin\left(\frac{1}{2} \cdot \alpha\right) \cdot a_p \,. \tag{2.1}$$

Figure 2.5 shows functions for this equation based on α for different a_p. To calculate $\vec{d}$ you can take the perpendicular of $\vec{a}$ and scale it to the length d_p.

When a big α is chosen according to amount, d_p 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. Doing a pivot walking motion might not move two polyominoes in the same direction. Figure 2.6 shows all 19 four-cube polyomino shapes with their displacement vectors. There are still two options for pivot walking, depending on a negative or positive value of α. You can walk left, in the direction of the west-faces, or right, in the direction of the 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. Local planning means that the planner only focuses on simple motion task. The task could be to develop a plan that moves a polyomino from position a to position b, or even simpler, to develop a plan for one pivot walk. Since on a global scale we consider the problem of self-assembly, we are not interested in changing positions alone. To work efficiently with local plans in the global planner, the initial and goal configuration of those plans should differ in the set of polyominoes they contain. Our local planner takes two cubes c_A and c_B out of different polyominoes A and B and attempts to establish a connection at a valid edge-pair (e_A, e_B). If a local plan was successful, this 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, meaning that the state of the simulation can be observed at any time and the actions can be adjusted accordingly. The local planner observes the position of cubes and polyominoes and works with the distance between them. It also works with the orientation of cube faces. 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 3.4 explain the techniques used in the local planning algorithm of Section 3.5.

3.1 Connecting Polyominoes

The two main rules that have to be considered when connecting two polyominoes A and B are: First, cubes can only be connected in the way described in Section 2.3, so we can distinguish between east-west and north-south connections. Second, pivot walking (Section 2.4) only allows the polyominoes to move left or right with $\vec{w}$. Keep in mind that a pivot walking motion is actually performed in the direction of the displacement $\vec{d}$ and not directly in direction of $\vec{w}$. To move in any arbitrary direction it is necessary to rotate the polyomino A, so that $\vec{d_A}$ points into the desired direction.

East-west connections are generally easier to handle. If we want to connect an east face of polyomino A to a west face of polyomino B, A has to walk into the east direction towards B, or the other way around. When A should be connected at an south face of B, A can now walk into east or west direction towards B, or B could again do the opposite. A closer look on why these two options differ and how both can be established at any position is taken in Section 3.2. We call this the *slide-in direction* $\vec{m} \in \{\vec{E}, \vec{W}\}$, which states that B is positioned in direction $\vec{m}$ of A.

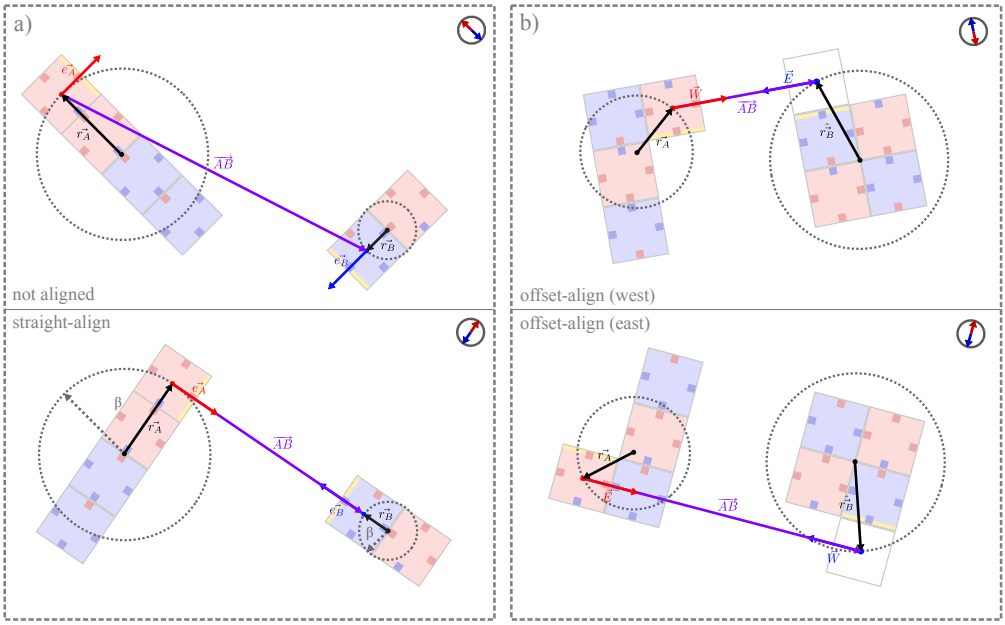

Figure 3.1: The figure shows examples for straight- and offset-aligning. The edges to be connected are marked yellow. a) shows two not aligned polyominoes (top) and the result of a straight-align (bottom). In b) you can see the two approaches for an offset-align. The cubes were aligned with their west edge (top) and with their east edge (bottom).

3.2 Aligning Cubes

To establish a connection between two polyominoes A and B, the connection-cubes c_A and c_B with their connection-edges e_A and e_B need to be aligned in the correct way. When A is rotated without magnetic field elevation, a cube with position P_{c_A} rotates in a circle around the center of mass of its polyomino C_A. The vector $\vec{r_A} = P_{c_A} - C_A$ is the radius of this rotation-circle. When also considering B, a rotation of the magnetic field rotates $\vec{r_A}$ and $\vec{r_B}$ 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-align and offset-align.

Straight-Align: For straight aligning, we define a vector $\overrightarrow{AB} = P_{c_B} - P_{c_A}$ pointing form c_A to c_B. The aligning is done when $\vec{e_A}$ points in the same direction as $\overrightarrow{AB}$, so $\angle\left(\vec{e_A}, \overrightarrow{AB}\right) = 0$. Consequently $\angle\left(\vec{e_B}, \overrightarrow{AB}\right) = \pi$, since e_A and e_B have to be opposite edges for a connection.

Figure 3.1 a) illustrates a straight-align for an east-west connection with all the parameters. The two polyominoes could now theoretically pivot walk together and connect 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 3.4.

Offset-Align: When considering north-south connections we need to align with an offset, so that the cubes can be moved together from east or west direction. We again define $\overrightarrow{AB} = \hat{P}_{c_B} - P_{c_A}$ with $\hat{P}_{c_B} = d_o \cdot \vec{e_B} + P_{c_B}$. We add an offset d_o to P_{c_B} in the direction of e_B, so $\overrightarrow{AB}$ points from P_{c_A} to a position above or below P_{c_B}. In a perfect world $d_o = 2 \cdot r_C$ is exactly one cube length, but to avoid failures when moving together, we give the alignment a bigger offset. Instead of pointing e_A in the same direction as $\overrightarrow{AB}$ we now have two options: Either solving $\angle\left(\vec{E}, \overrightarrow{AB}\right) = 0$ or $\angle\left(\vec{W}, \overrightarrow{AB}\right) = 0$, depending on if we want to move A in east direction, or in the west direction towards B.

You can see the two options for offset-aligning in Figure 3.1 b). For the two polyominoes you can also see that aligning with east or west edge can make a difference when moving together. Establishing a connection by letting A move towards B in west direction is possible, but by moving in east direction other cubes of the polyominoes are blocking the way. In Section 3.4 we present a method for checking if moving in from east or west is possible.

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,

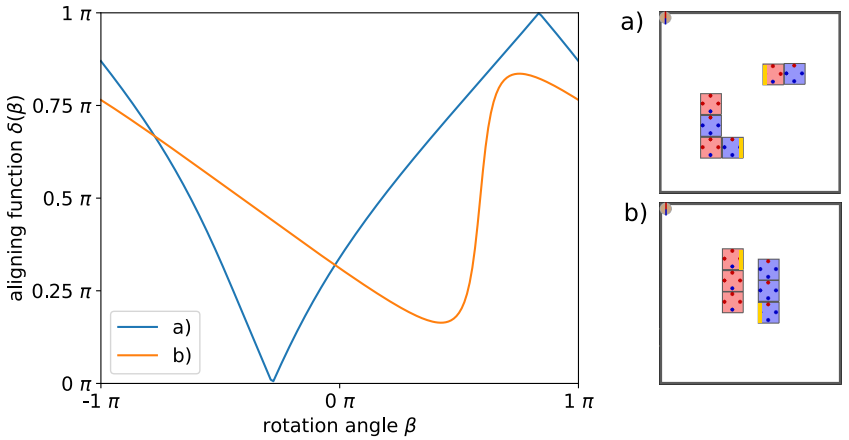

Figure 3.2: Two examples of the aligning function $\delta(\beta)$ for different configurations with different polyominoes. The edges about to be aligned are marked yellow. The cubes are perfectly aligned, when $\delta(\beta) = 0$. This can be seen for example a) at $\beta \approx -0.28\pi$, so rotating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

where both $\vec{e_A}$ and $\overrightarrow{AB}$ change according to β.

$$\delta(\beta) = \angle\left(R_\beta \vec{e_A},\, (R_\beta r_B + C_B) - (R_\beta r_A + C_A)\right) . \tag{3.1}$$

R_β is a rotation matrix used for rotating vectors by β. For an offset-align the function would be

$$\delta(\beta) = \angle\left(R_\beta \vec{e},\, (R_\beta \hat{r_B} + C_B) - (R_\beta r_A + C_A)\right) , \tag{3.2}$$

with $\vec{e} \in \{\vec{E}, \vec{W}\}$ and $\hat{r_B} = \hat{P}_{c_B} - C_B$

Alignment is not always possible, so instead of solving $\delta(\beta) = 0$, we are minimizing $\delta(\beta)$. 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 we encounter a value close enough to zero we can return it. If not we return the minimum of all the calculated values. This way we at least get as close to an alignment as possible.

3.3 Moving Polyominoes Together

Since both polyominoes A and 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 d_P. At a first glance it seems easy to move polyominoes together,

after the connection-cubes are aligned as described in Section 3.2. The two options of walking left and right determine if either A is chasing B, or if B is chasing A, which is of course also dependent on their initial position and orientation. In the end one option might be shorter or better in terms of other polyominoes interfering with A and B, but theoretically both are able to establish the connection.

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 [18] 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. It is mathematically possible to calculate the right orientation of the magnetic field to result in a collision after n pivot walking cycle for both polyominoes, 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_A moved to the original position of c_B, before it is moved with

$$n = \left\lceil \frac{\|P_{c_A} - P_{c_B}\|}{d_{p,A}} \right\rceil . \tag{3.3}$$

We then only walk a portion of n and re-align the cubes. When c_A and c_B are near enough for magnetic forces to act we frequently wait a short period to let magnetic attraction pull e_A and e_B together. This will automatically adjust the alignment, but for even more precision we decreased the pivot walking angle α when in close proximity.

3.4 Plan and Failures

A plan is a sequence of actions $A = a_1, ..., a_n$ 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 magnetic field rotations, not 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

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

A local plan is successful if g_{goal} contains a polyomino with the desired connection of c_A and c_B at (e_A, e_B). If a plan is successful or not is described by the plan-state s. 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_A and e_B need to be free, so no other cube is already connected to them, but even if both are free other cubes then c_A and c_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_A and e_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 that is not enough. Connection with other polyominoes during planning can invalidate those pre-checked conditions, so we need to frequently re-check them.

Impossible Slide-In: Even if a connection in a common local coordinate-system is possible you can only connect by moving in from east or west. Other cubes can again prevent this by blocking the way for an easy slide-in. We can also verify both east and west slide-in 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, the condition needs to be re-checked frequently, due to changing polyominoes.

Polyominoes being Stuck: Polyominoes can get stuck in corners or on walls of the workspace. In this state it is not possible anymore to decrease the distance of A and B by pivot walking. We can identify this state when the position of both cubes c_A and c_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 solve the problem. It depends on the distance of the cubes after straight-aligning. If the distance is too big for magnetic forces to act, failure is reported, but if they are close enough the local planner waits until magnetic attraction connects e_A and e_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 a 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

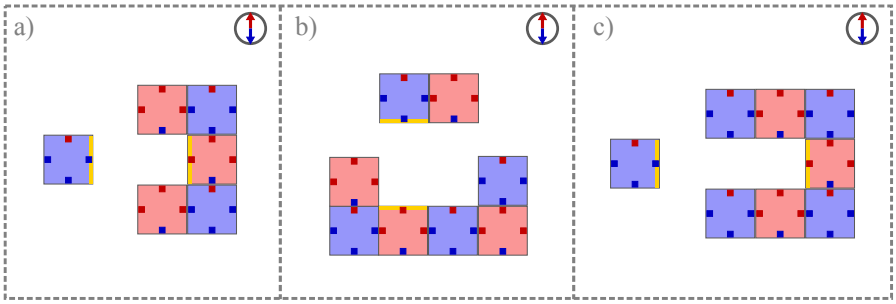

Figure 3.3: Three different examples for connecting polyominoes into caves. a) and b) show one-cube-deep caves (a) east-west and b) two-cube-wide north-south). c) illustrates a two-cube-deep east-west cave. The edges to be connected are marked yellow.

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 would only work for east-west caves, or by letting magnetic forces attract the connection-faces. 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 slide-in. 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. We state failure if any invalid polyominoes 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 A and B are able to do. Whenever a pivot walking motion is done, we sum up the distances c_A and c_B moved together. Let (w, h) be the size of the workspace. We define a maximum movement capacity of $2 \cdot (w + h)$. 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.

Algorithm 1 ALIGN-WALK-REALIGN

Input: c_A, c_B, e_A, e_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 $e_A \in \{E, W\}$ **then** // aligning straight or with offset
    $a \leftarrow$ ALIGN-STRAIGHT(c_A, c_B, e_A)
  else
    $a \leftarrow$ ALIGN-OFFSET$(c_A, c_B, \vec{m}, e_B)$
  end if
  $g_{goal} \leftarrow$ SIMULATE(g_{goal}, a)
  $A \leftarrow$ APPEND(A, a)
  $s \leftarrow$ UPDATE-STATE$(g_{goal}, c_A, c_B, e_A, \vec{m})$
  if $s \neq$ undefined **then** // first time checking for failure or success
    return (s, g_{goal}, A)
  end if
  if CRITICAL-DISTANCE(c_A, c_B) **and** wait **then** // wait or walk
    $a \leftarrow$ WAIT()
    wait $\leftarrow$ **false**
  else
    $a \leftarrow$ WALK$(c_A, c_B, \vec{w})$ // a can be multiple walking steps (Section 3.3)
    wait $\leftarrow$ **true**
  end if
  $g_{goal} \leftarrow$ SIMULATE(g_{goal}, a)
  $A \leftarrow$ APPEND(A, a)
  if STUCK(c_A, c_B) **then** // handle stuck condition
    $a \leftarrow$ ALIGN-STRAIGHT(c_A, c_B, e_A) // do a straight aligne
    $g_{goal} \leftarrow$ SIMULATE(g_{goal}, a)
    $A \leftarrow$ APPEND(A, a)
    while not STUCK(c_A, c_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$(g_{goal}, c_A, c_B, e_A, \vec{m})$
  if $s \neq$ undefined **then** // second time checking for failure or success
    return (s, g_{goal}, A)
  end if
end loop

3.5 Local Planning Algorithm

Before executing the algorithm presented in Algorithm 1 we evaluate all the failure conditions that can be checked in advance, so no simulation-time is wasted on a plan that is bound to fail from the start. While doing so, the possible slide-in directions are also determined and Algorithm 1 is executed with both pivot walking directions $\vec{w}$ for each possible $\vec{m}$. This means for an east-west connection, we always develop two plans and for a north-south connection two or four, depending on the slide-in directions.

In the end, the successful plan with the lowest costs is returned. Even if all plans fail, we still determine the best failure. Again, plans with lower costs are preferable, but we favor impossible connection and slide-in failures. These failures just state that we can not establish a specific connection, but a global planner could continue to plan based on the goal configuration the local planner ended in. A configuration when the polyominoes are stuck, or the maximum movement capacity is reached, is not good for further planning, but invalid polyominoes or polyominoes with caves will definitely be omitted on a global scale.

The different plans are developed in parallel and if one process finishes with a successful plan, the execution of all other plans can be canceled. This saves a lot of 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 the actions.

Align-Walk-Realign: Algorithm 1 takes the connection-cubes and edges c_A, c_B, e_A, e_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 of actions 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. Once after aligning, and once at the end of the loop. That way, the planner avoids simulating 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_A and e_B. The offset-align is done with the direction of $\vec{m}$. After aligning we walk the estimated amount of pivot walking cycles (Section 3.3) in direction $\vec{w}$ with WALK, or we wait with WAIT, if c_A and c_B are in close proximity, determined by CRITICAL-DISTANCE. If we waited in the previous iteration, we walk in the current one and oppositely. 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.

3.5.1 Complexity

Optimality In our case the optimal plan for connecting two polyominoes is the one establishing the connection with the lowest rotational cost, as defined in Section 3.4. 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 archived, when sliding on walls and different polyomino displacements, as described in Section 3.3, are not existent. It is easy to see that after aligning, both pivot walking directions produce plans that move the polyominoes together in a straight path. The plan with the shorter path would be optimal is this theoretical case. But these factors have to be considered, and even if they were by a local planner it is hard to say, if that would be enough to actually prove optimality.

Completeness The local planner is also no complete. Just because the up to four plans it evaluates fail, does not prove the non existence of a action sequence that would establish the desired connection. If other polyominoes are blocking the way of A and B, complex movements around these polyominoes, instead of the more or less straight path we are taking, could create solutions where our planner fails. The reason we choose this simple approach is again to avoid simulation as much as possible.

4 Global Planner

The task of the global planner is to assemble a specified target polyomino 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 part of the configuration-space we can actually explore is limited to configurations where a connection attempt between two cubes was made. Compared to $SE(2)$ this part is manageable in size and only contains configurations which are relevant for self-assembly.

The question still remains, how these configurations are explored. Using rapidly-exploring random trees (RRTs) [11] yields good results in a lot of cases, since the configuration-space gets evenly explored without the challenge of determining what decisions are promising for the end goal. But, it also means the exploration of many configurations which are not necessary for reaching the goal. For us this approach is not reasonable. Because of the high fidelity simulation we are working with, the computation time for a local plan is huge, so planning the assembly of T with as few local plans as possible is the aim for our global planner.

We need to make well though through connection decisions, that are valid for assembling 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, so 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 the graph for decision making and Section 4.4 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 T. A take on why is this done and why the problem becomes more complex when working with extra cubes, is done in Section 4.6.

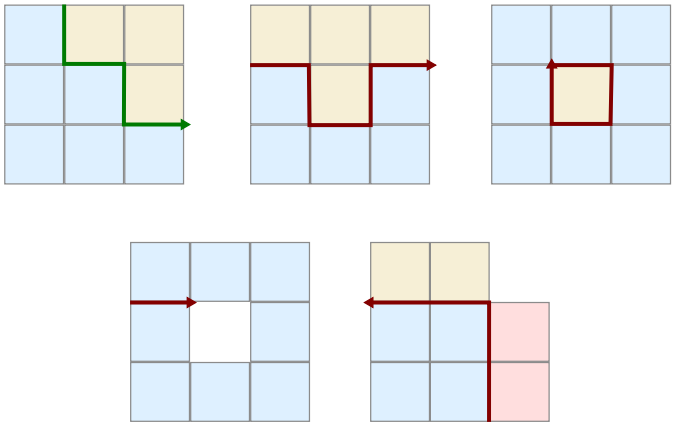

Figure 4.1: Examples for cutting polyomino shapes. The top row shows three two-cuts for a 3×3 shape, of which only the left one is monotone and therefore valid. The middle one creates a cave and the right one a hole. The bottom row shows cuts that do not split the polyomino into two pieces. The left one does not break the polyomino at all and the right one creates three sub-polyominoes.

4.1 Two-Cutting Polyominoes

Schmidt et al. [19] made use of straight-line two-cuts, to handle the construction of a polyomino with more than trivial sub-assemblies.

We define a two-cut as a continuous path of connections through a polyomino that divides the polyomino into two sub-polyominoes, when these connections would be removed. For later use in Section 4.2 we want to enumerate all two-cuts of a polyomino that are useful for planning. We do not limit the cuts by only allowing straight paths like [19], instead we only consider monotone two-cuts.

Monotone means that whenever the path goes into a direction it can never go into the opposite direction again. Figure 4.1 top-left shows a monotone two-cut through a 3×3 polyomino shape. The cut starts at the top of the shape and only moves down and right. By removing all the connections on the path, the polyomino shape is 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 top-middle shows a non-monotone two-cut creating a cave and Figure 4.1 top-right 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 or into a hole. After the path ended its connections are removed from the polyomino and the path is added as a two-cut, if the polyomino got split into exactly two pieces. The bottom row of Figure 4.1 shows cuts that split the polyomino in less or more than two pieces.

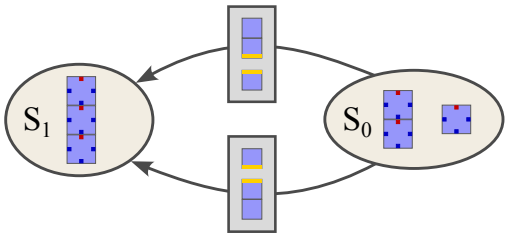

Figure 4.2: Two two-cut-sub-assembly edges connecting the polyomino sets S_0 and S_1. The weights of the edges differ, since there are two ways to connect the 2×1 with the 1×1 to create a 3×1 polyomino. The connections are illustrated in rectangular boxes placed on the edges.

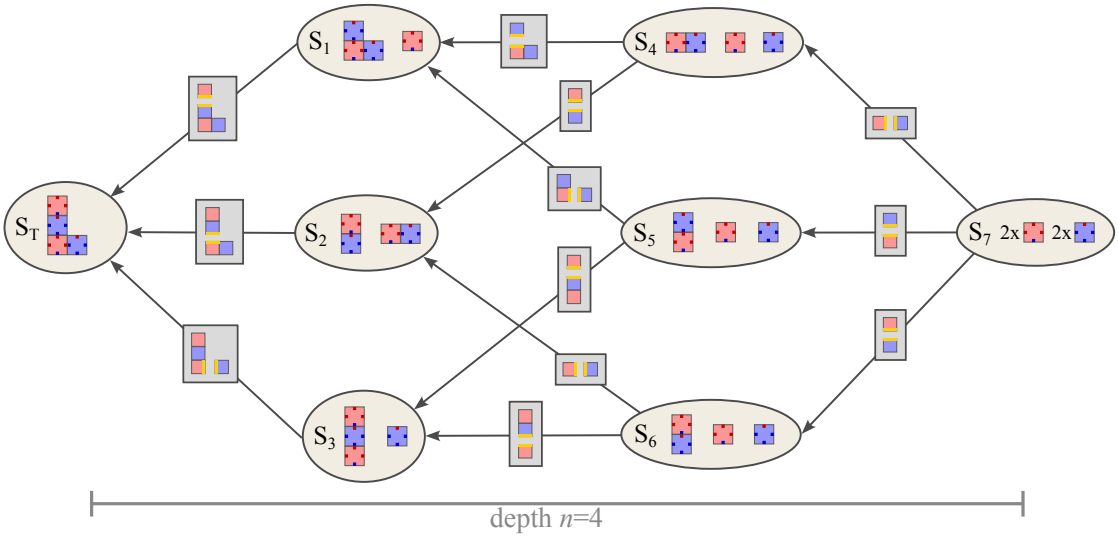

Figure 4.3: Example of an two-cut-sub-assembly graph for a four-cube L-shape. The polyomino sets are illustrated as ellipses. If the polyominoes of a set are not numbered, there is only one occurrence of this polyomino. Otherwise the number of occurrences is placed left of the polyomino. The sets are numbered as if the graph was produced by Algorithm 2 starting from S_T. The weight of edges are illustrated as rectangular boxes containing the polyominoes that need to be connected at specific edges, marked in yellow.

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, we will call it $G_{TCSA}(T)$. 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 enumerates the polyomino types preset in g. If g contain 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 $\{S_0, w, S_1\}$, if S_0 can be transformed to S_1 by connecting two polyominoes contained in S_0. The cube and edge information of the connections are stored as the weight w. 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 4.2. The direction of $\{S_0, w, S_1\}$ always goes from S_0 to S_1, but we can reverse the definition for an edge as following:

Two nodes S_0 and S_1 are connected, if one polyomino contained in S_1 can be two-cut, so that the resulting polyomino set equals S_0. This already provides a perspective on the use of two-cuts and the way $G_{TCSA}(T)$ is built starting with T. We will further explain the building process along with an example of a TCSA graph provided in Figure 4.3.

Algorithm 2 BUILD-TCSA-GRAPH

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

$V \leftarrow \{\}$
$E \leftarrow \{\}$
$i \leftarrow 0$
$V[i] \leftarrow S_T$
while $i <$ SIZE(V) **do** // work through nodes in BFS manner
  $S_i \leftarrow V[i]$
  for each $A \in S_i$ **do** // go through all polyomino types in S_i
    for each $t_c \in$ TWO-CUTS(A) **do** // go through all monotone two-cuts
      A_1,$A_2 \leftarrow$ CUT-POLYOMINO(A, t_c)
      $S_{new} \leftarrow (S_i \setminus \{A\}) \cup \{A_1, A_2\}$ // new node after cutting
      if $S_{new} \notin V$ **then**
        $V \leftarrow$ APPEND(V, S_{new})
      end if
      $E \leftarrow$ APPEND$(E, \{S_{new}, t_c, S_i\})$
    end for
  end for
  $i \leftarrow i + 1$
end while
return (V, E)

Building a TCSA Graph Algorithm 2 describes the process of building $G_{TCSA}(T)$ for the target 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_T, which is a polyomino set only containing the target shape.

New nodes and edges are determined by two-cutting every polyomino type A in the current set S_i by every possible monotone two-cut of A. This is done by enumerating the two-cuts with Two-Cuts, the way it was described in Section 4.1, and cutting A at the two-cut with Cut-Polyomino. The cutting results in the two sub-polyominoes A_1 and A_2. S_{new} contains the same polyominoes as S_i with the exception that one occurrence of A is removed and replaced by one occurrence of A_1 and 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, we can add it to V, 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 graph edges E. This allow multiple edges, as seen in Figure 4.2, and multiple out going edges to different nodes, which can be observed in Figure 4.3, where different connections in S_4 lead to either S_1 or S_2.

Each two-cut applied to a polyomino set reduces its amount of polyominoes by one. Let n be the size of T, then $n-1$ two-cuts applied to S_T will produce a polyomino set $S_{trivial}$ containing only trivial polyominoes, as it is the case for S_7 in Figure 4.3. All S_i will inevitably end up in this situation and the algorithm will return (E, V), since trivial polyominoes cannot be cut anymore. This means that no matter which connections are chosen along the way, $n-1$ edges will always be needed to get from $S_{trivial}$ to S_T. We describe this attribute, by giving the TCSA graph a depth of n. The depth is also illustrated in Figure 4.3 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 equals the target size n 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_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

$$\#nodes_{worst} = \sum_{k=1}^{n} \left\{ {n \atop k} \right\}, \tag{4.2}$$

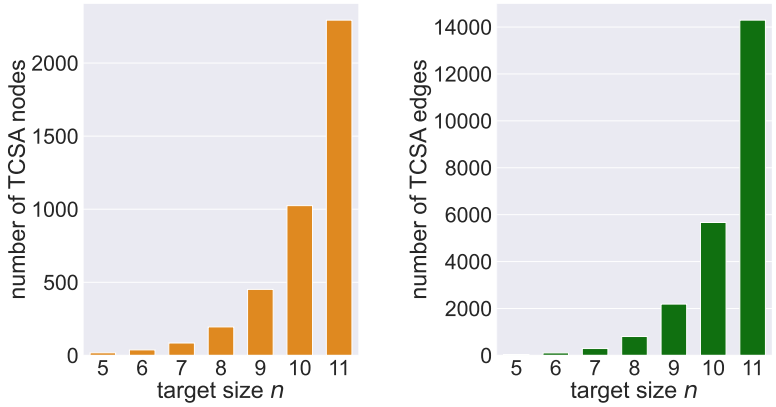

Figure 4.4: Average number of nodes (left) and edges (right) of a TCSA graph for different target sizes n. For each target size 200 samples of randomly generated polyominoes were taken.

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

It is not necessary or even valid to group any cubes into certain partitions. The only way of creating partitions is by monotonously two-cutting existing polyominoes, which drastically lowers the number of $\#nodes_{worst}$. In Figure 4.4 statistical data shows the average number of nodes and edges a TCSA graph consists of for varying target sizes n.

Our implementation of $G_{TCSA}(T)$ stores nodes in a hash-table. Accessing nodes and connected edges, or checking if a polyomino set is contained in $G_{TCSA}(T)$, can be done in $\mathcal{O}(1)$. The creation of $G_{TCSA}(T)$ 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 make a complex data-structure like the TCSA graph worth it.

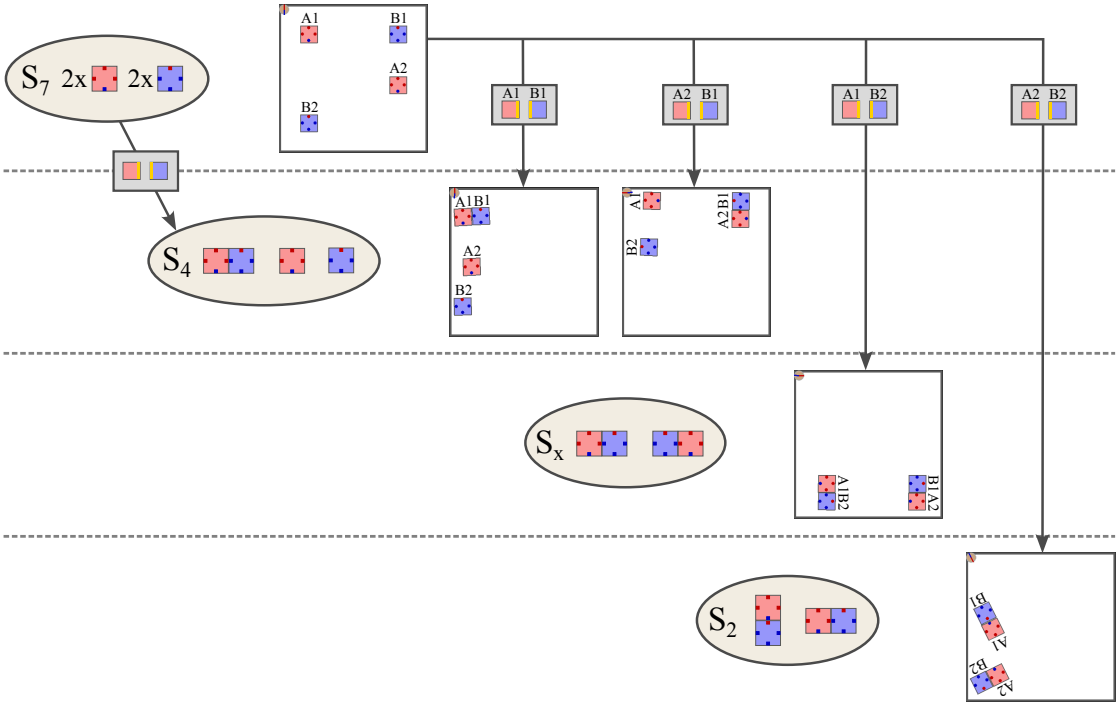

Figure 4.5: All connection options when connecting a red cube at the west of a blue cube to get from S_7 to S_4. Developing a local plan for different polyomino pairs, leads to different goal configurations. (A_1, B_1) and (A_2, B_1) lead to the desired polyomino set S_4, but (A_2, B_2) leads directly to S_2. All these sets can be found in the TCSA graph of Figure 4.3. The goal configuration of (A_1, B_2) holds the set S_x, which cannot be found in Figure 4.3. For further global planning this set could not be used.

4.3 Connection Options

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

For instance, if $S(g) = S_7$ in Figure 4.3, three out-going edges provide three connections to choose from, but that is not all. Assuming 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 $S(g_2) = S_4$. 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. You can also see how theses options differ in the goal configurations the local planner ended in.

Let L_A and L_B be collections of the physically distinct polyominoes for the polyomino types A and B. When A and B are about to be connected as the weight of a TCSA edge dictates, there are $|L_A \times L_B|$ polyomino pairs to choose from. If $A = B$, 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 sorting strategies for these options, to provide an order of the best probable outcome that the global planner can work through. The approaches are compared in Chapter 6.

Minimal Distance The minimal distance sorting sorts connection options based on the distance between the cubes that are about to be connected. The idea is that a smaller distances requires less movement to connect, which means shorter simulation time and lower plan costs of the resulting plan. 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. Less movement might even prevent unwanted sub-assemblies.

Grow Largest Component Another approach is to grow the largest component. The options are sorted into classes of maximum polyomino sizes in the resulting polyomino sets. We prefer TCSA edges that lead to sets containing the biggest polyominoes. When the options for S_4 in Figure 4.3 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 they occur the TSCA graph can provide

solutions to integrate them if possible. Larger polyominoes generally move faster acting positive on plan costs.

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.

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 3.2), for an easier slide-in. Besides of 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 $G_{TCSA}(T)$. In-fact, more sub-assemblies decrease the number of polyominoes in the workspace, which brings the goal of assembling T even closer. Layers of depth were skipped in the TCSA graph, so that it might be possible to assemble T with less than $n-1$ local plans. This can be seen in Figure 4.5, where A_2 and B_2 were connected. The resulting polyomino set matches with S_2 instead of S_4 of the nodes from Figure 4.3.

Like already mentioned in Section 4.3, when the resulting polyomino set is not in $G_{TCSA}(T)$, 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 checks if the polyomino set of the configuration in the workspace is contained in $G_{TCSA}(T)$. If not, the 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 $G_{TCSA}(T)$, 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.

Algorithm 3 ASSEMBLE-TARGET

Input: T, g_{init}
Output: s, P // state of global plan s and plan stack P containing local plans

1: $G_{TCSA}(T) \leftarrow$ BUILD-TCSA-GRAPH(T)
2: $s \leftarrow$ undefined
3: $P \leftarrow \{\}$
4: $g \leftarrow g_{init}$ // current configuration g
5: **loop**
6: $O \leftarrow$ CONNECTION-OPTIONS$(g, G_{TCSA}(T))$
7: valid $\leftarrow$ **false**
8: **while not** EMPTY(O) **and not** valid **do** // try options until local plan is valid
9: $(c_A, c_B, e_A, e_B) \leftarrow$ POP(O)
10: $p_{new} \leftarrow$ LOCAL-PLANNER$(g, (c_A, c_B, e_A, e_B), G_{TCSA}(T))$
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** $T \in S(g)$ **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**

4.5 Global Planning Algorithm

The global planning algorithm provided in Algorithm 3 takes the initial configuration g_{init} and the target T as inputs and returns the state of the global plan s and a plan stack P as outputs. For a successful plan, P contains the local plans leading to the assembly of T. When concatenating the actions of all the plans in P, this creates a sequence of actions, that could be considered as a global plan. Because a the local plans were created along the use of 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 $G_{TCSA}(T)$ in a depth-first-search manner, which is done in the attempt to get closer to assembling 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 the way it was described in Section 4.3. The mechanisem behind this function can be viewed as a hash-map, storing the options as the values for the configuration as the key. The options need to be determined and sorted, only when it is 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, meaning that when O is altered, the hash-map is updated as well. Whenever a connection option is popped from O, this option is removed from the hash-map and will therefor never be considered again. Note that options are stored per configuration g, not for the polyomino set $S(g)$. Two configuration sharing the same polyomino set, both have there own lists of connection options. We traverse the configuration-space with the TCSA graph as a guidance, not the TCSA graph itself. Notes in $G_{TCSA}(T)$ can be encountered multiple times and will never be eliminated from planning.

Once O got retrieved, the algorithm works through O in the order determined by the option sorting that was applied in advance. This is done until no option is left, or a valid local plan for a options was found. Local-Planner uses Algorithm 1 to create a local plan p_{new}. It also take $G_{TCSA}(T)$ as a input parameter, to ensure the newly added failure condition, when a configuration is not contained in $G_{TCSA}(T)$. 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 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 lead 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 not previously visited configuration we can fall back to. In that case the algorithm has to state failure for assembling T.

Before calling Algorithm 3 one initial check for $S(g_{init})$ in $G_{TCSA}(T)$ is necessary, 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 optimal solution for the connection of two polyominoes, the global planner will not reach optimality as well. Even if the local planner provides only optimal solutions, our depth-first-search approach would not explore the configuration space in a way that best sequence of local plans would be picked. Algorithm 3 is greedily moving 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 would need broad exploration and comparison of different paths to the target, which is infeasible in our case due to high simulation time of 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 T. Algorithm 3 will always return success or failure in finite time. 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 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 archive 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 T. This is highly unrealistic, but theoretically possible, since layer 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 once that lead to a configuration with one n-size polyomino that is not 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

$$\#local_{worst} = \sum_{i=1}^{n-1} {o_\mu}^i \,. \tag{4.3}$$

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

The number of cubes in the workspace is limited to the target size n for the global planner to work. 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 T all the left over 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 left over cubes leading to a less efficient global planner.

5 Simulator

Our simulator used for modeling the behavior of magnetic modular cubes uses the 2d physics library Pymunk[1]. This library is build for the Python 3 and Python 2 environment based on the 2d physics library Chipmunk[2]. We used Pymunk, since 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 plans and to allow user controls. As a disadvantage, we are challenge with the simulation of 3d movement in a 2d environment. This way we trade simulation accuracy for faster simulation time, which is necessary to develop global plans in a reasonable time.

In Figure 5.1 a flow chart diagram illustrates the control flow of the simulators 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). Another crucial functionality is 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 adjustment 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 cubes permanent magnets are calculated. This also determines connections of cube faces that will be used to retrieve information about polyominoes in the workspace (Subsection 5.4.1). Polyomino information is necessary to calculate forces of the magnetic field acting on cubes (Subsection 5.4.2) and friction forces, on which we heavily rely to simulate 3d movement like pivoting on pivot edges (Subsection 5.4.3). All the calculated forces will be applied in the Pymunk-step next iteration.

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

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

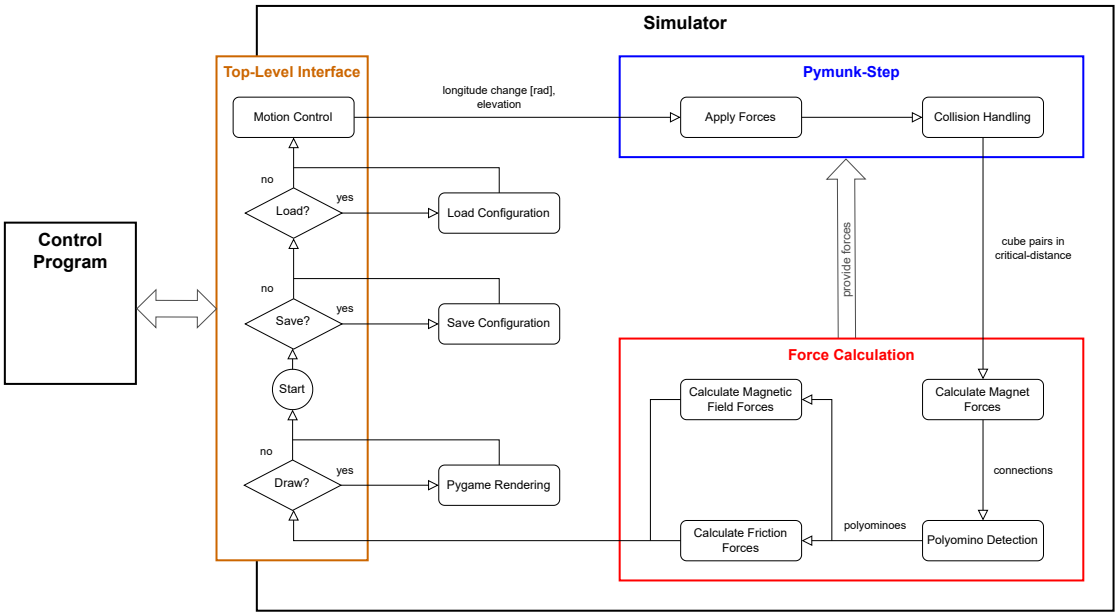

Figure 5.1: Flow chart diagram illustrating the control flow of the simulators simulation loop. Any control program can interact with the top-level interface of the simulator. Further Tasks are grouped in the Pymunk-step and the force calculation. Calculated forces are provided to the Pymunk-step for the next iteration of the loop.

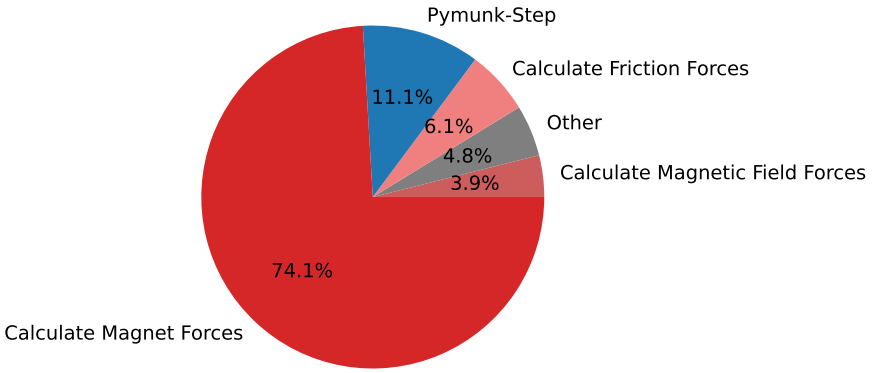

Figure 5.2: Fraction of time used on certain steps in the simulation loop. The simulator ran for 8 seconds without drawing and executed various motions with ten cubes in the workspace. The steps can be found in the control flow illustrated in Figure 5.1

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 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. More on that in Subsection 5.4.3.

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 orient themselves by magnetic field forces we further explain in Subsection 5.4.2. 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 larges 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. This is the reason simulating 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 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 workspace configuration that got saved while in motion or when cubes where 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.

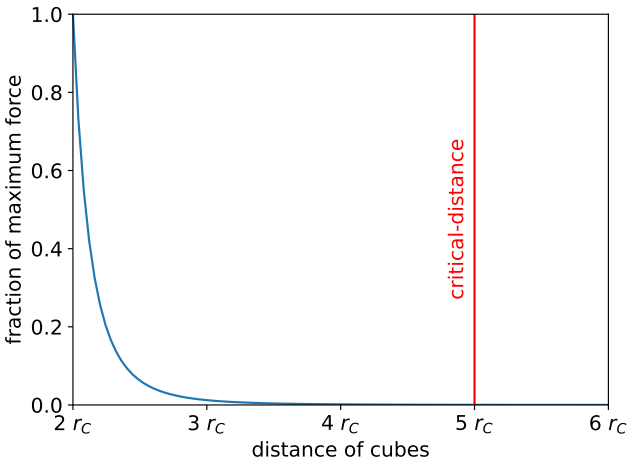

Figure 5.3: Decline of magnetic force between two permanent magnets with increasing distance between the cubes. We reach a maximum force at $2r_C$, when the cube faces are connected. With distances bigger than the critical-distance of $5r_C$ the fraction of force is negligible.

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 in critical-distance. For that, each cube is surrounded by a circular sensor with a radius of half the critical-distance. A cube pair is in critical-distance if their sensors collide. Cubes not in critical-distance are to far away to significantly affect each other with magnetic forces of their permanent magnets. We only calculate magnet forces for cube pairs in critical-distance to speed up simulation. The critical-distance is $5r_C$ and illustrated in Figure 5.3 together with the decline of magnetic force with increasing cube distance.

5.4 Simulating Forces

[14]

5.4.1 Magnet Forces

5.4.2 Magnetic Field Forces

5.4.3 Friction Forces

6 Results

7 Conclusion

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] 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.

[14] 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.

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

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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