SE(2) Assembly Planning for Magnetic Modular Cubes

Kjell Keune[1] and Aaron T. Becker[1,2]

***Abstract*— Magnetic modular cubes are cube-shaped bodies with embedded permanent magnets. The cubes are uniformly controlled by a global time-varying magnetic field.**

A 2D physics simulator is used to simulate global control and the resulting continuous movement of magnetic modular cube structures. We develop *local plans*, closed-loop control algorithms for planning the connection of two structures at desired faces. The global planner generates a *building instruction graph* for a target structure that we traverse in a depth-first-search approach by repeatedly applying local plans.

We analyze how structure size and shape affect planning time. The planner solves 80% of the randomly created instances with up to 12 cubes in an average time of about 200 seconds.

I. Introduction

Self-assembling modular parts forming bigger structures is a well-known concept in nature, from DNA to whole organisms [7]. Self-reconfiguring robot swarms have promising applications including targeted drug delivery [25], or microscale manufacturing [21]. Equipping each tiny robot with its own sensing, actuation, connection, and power systems remains challenging. Instead, a promising solution is using external global control that applies the same torque and force to each robot [26]. Robots with embedded magnets can be controlled by an external magnetic field and also connect to each other without any internal power supply [19], [22]. This paper designs a motion planner to assemble structures using magnetic modular cubes [6]. Planning occurs in the 2-dimensional special Euclidean group $SE(2)$.

This paper follows the Tilt model from [3]–[5], [9], [24], where all tiles move in the same direction unless they are obstructed. Reconfiguring one configuration into another is PSPACE-complete [2]. Minimizing the number of actions for this problem is also PSPACE-complete [2], [4]. Similar to [27], the magnets on magnetic modular cubes preferentially bond, enabling self-assembly. The assembled 2D shapes are polyominoes [18]. By limiting the controls to only $90°$ turns and assuming a uniform movement 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 planner was developed, that explores a finite configuration-space and lists all the possible polyominoes that can be created from an initial configuration [6].

This paper expands on Tilt motion planners [6], [8], [19] by enabling robots to move at speeds determined by their shape and handles arbitrary real-number positions and orientations.

This work was supported by the National Science Foundation under [IIS-1553063, 1849303, 2130793] and the Alexander von Humboldt Foundation.
[1]TU Braunschweig, Germany k.keune@tu-bs.de
[2]University of Houston, USA atbecker@uh.edu

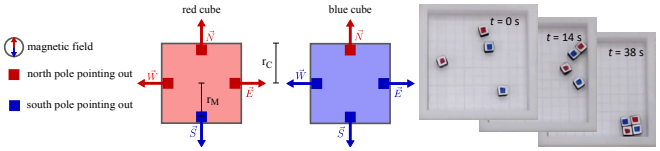


Fig. 1: Top-down view of the two magnetic modular cube types with outward pointing magnet poles illustrated as red and blue squares. Right: magnetic modular cubes with 2.8 mm edges in a workspace. See overview video at `https://youtu.be/C7Bj3Hco_G4`.

II. Preliminaries

This section provides a basic understanding of the hardware setup. Further technical details are available in [19].

a) Magnetic Modular Cubes: are cube-shaped bodies embedded with permanent magnets on the four side faces. The two cube types are shown in Fig. 1. Magnets ensure that the cubes align with the global magnetic field and control how they assemble. Each face is represented by a vector $e \in \{\vec{N}, \vec{E}, \vec{S}, \vec{W}\}$ pointing in the cardinal direction of the magnetic field. r_C is defined as the cube radius.

b) Workspace: Magnetic modular cubes are maneuvered on planar workspaces surrounded by a time-varying magnetic field with sufficient torque to lift one edge of the cube assembly off the workspace. Due to space limitations, this paper only considers bounded rectangular workspaces without internal obstacles.

c) Configuration: The configuration for one cube is in $SE(2)$, consisting of the position in $\mathbb{R}^2$ and an orientation $\mathbb{S} = [0, 2\pi)$ [14]. With n cubes, the configuration-space is $\mathbb{R}^{2n} \times \mathbb{S}^1$. We assume all cubes align with the external field and consider cubes of the same color equivalent.

d) Polyominoes: are face-connected cubes on a 2D grid. Two cube faces can connect if their magnets have opposite polarities. Cubes can connect at north and south faces. If cubes are different types they can also connect at east and west faces. We consider *fixed polyominoes*, meaning that two polyominoes are distinct if their shape or orientation (with respect to the global magnetic field) differ [18].

e) Motion Modes: The local planner uses two motion modes, rotating and pivot walking. Rotating the magnetic field in the plane of the workspace spins each polyomino about its center of mass.

When the magnetic field elevates the south pole, all polyominoes will pivot on the bottom edges of their northernmost cubes. Lifting the north pole pivots cubes on the bottom edges of their southernmost cubes. Polyominoes rotate around the center point of their pivot edge, called the north or south *pivot point* as illustrated in Figure 2b. After one pivot walking cycle, the polyomino has moved by

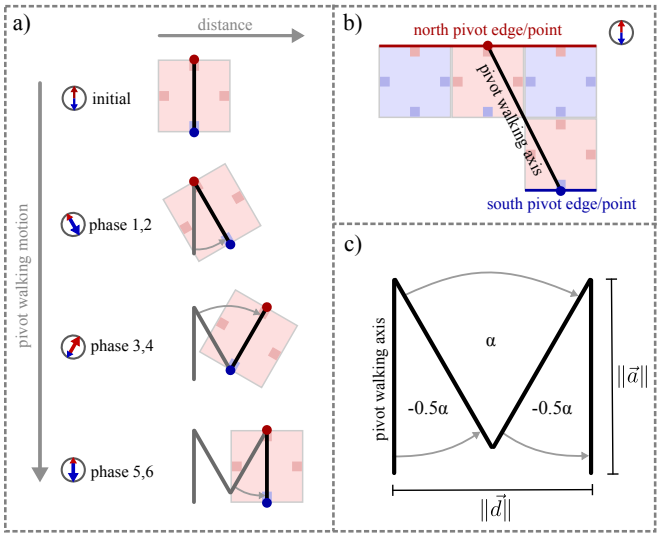


Fig. 2: Pivot walking: a) six pivot walking phases for a red cube. The compass depicts the magnetic field (bigger arrow indicates elevation, in phase 1 the south pole is lifted). b) example polyomino with pivot axis, edges and points. c) labeled pivot axis rotation.

a displacement vector $\vec{d}$ with length $2 \cdot \sin\left(\frac{\alpha}{2}\right) \cdot \|\vec{a}\|$. The direction and length of $\vec{d}$ changes with the polyomino shape. The movement is always perpendicular to the pivot walking axis $\vec{a}$. $\|\vec{d}\|$ increases with the pivot walking angle α, but large α require more space. A polyomino can walk west or east. We label the *pivot walking direction* $\vec{w} \in \{\vec{W}, \vec{E}\}$.

III. Local Planner

Our local planner takes two cubes $c_\mathcal{A}$ and $c_\mathcal{B}$ from polyominoes $\mathcal{A}$ and $\mathcal{B}$ and attempts to establish a connection at a valid edge-pair $(e_\mathcal{A}, e_\mathcal{B})$. The local planner uses our simulator from section V in a closed-loop manner. The distance between two cube centers is $d(c_\mathcal{A}, c_\mathcal{B}) = \|p_{c_\mathcal{A}} - p_{c_\mathcal{B}}\|$.

A. Aligning Cubes

To connect polyominoes $\mathcal{A}$ and $\mathcal{B}$, the edges $e_\mathcal{A}$ and $e_\mathcal{B}$ must be aligned. When $\mathcal{A}$ is rotated, each cube center rotates in a circle around the center of mass of its polyomino $p_\mathcal{A}$.

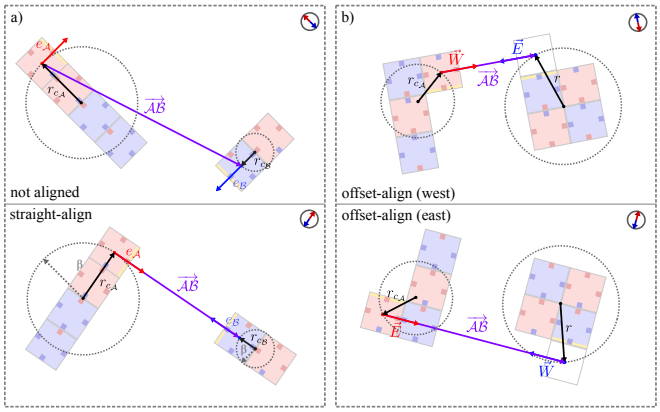


Fig. 3: Straight- and offset-aligning. The edges to be connected are marked yellow. a) two unaligned polyominoes (top) and the result of a straight-align (bottom). b) results of two options for offset-aligning; $c_\mathcal{A}$ aligned with its west edge (top) or east edge (bottom).

a) Straight-Aligning: we define a vector $\overrightarrow{\mathcal{AB}} = p_{c_\mathcal{B}} - p_{c_\mathcal{A}}$ pointing from $c_\mathcal{A}$ to $c_\mathcal{B}$. Alignment occurs when $e_\mathcal{A}$ points in the same direction as $\overrightarrow{\mathcal{AB}}$, so $\angle\left(e_\mathcal{A}, \overrightarrow{\mathcal{AB}}\right) = 0$. Figure 3a illustrates a straight-align for an east-west connection.

b) Offset-Aligning: north-south connections must align with an offset, so polyominoes can be moved together from either the east or west. We define $\overrightarrow{\mathcal{AB}} = (d_{offset} \cdot e_\mathcal{B} + p_{c_\mathcal{B}}) - p_{c_\mathcal{A}}$, where offset d_{offset} is added to $p_{c_\mathcal{B}}$ in the direction of $e_\mathcal{B}$. $\overrightarrow{\mathcal{AB}}$ is now pointing from $p_{c_\mathcal{A}}$ to a position above or below $p_{c_\mathcal{B}}$. Instead of pointing $e_\mathcal{A}$ in the same direction as $\overrightarrow{\mathcal{AB}}$, we now have two options: move $\mathcal{A}$ east by solving $\angle\left(\vec{E}, \overrightarrow{\mathcal{AB}}\right) = 0$ or west with $\angle\left(\vec{W}, \overrightarrow{\mathcal{AB}}\right) = 0$. These two options are shown in Figure 3b.

B. Moving Polyominoes Together

Pivot walking only allows the polyominoes to move east or west. To connect an east face of polyomino $\mathcal{A}$ to a west face of polyomino $\mathcal{B}$, $\mathcal{A}$ must walk east towards $\mathcal{B}$, or vice versa. To connect $\mathcal{A}$ at a south face of $\mathcal{B}$, $\mathcal{A}$ can walk east or west towards $\mathcal{B}$, or $\mathcal{B}$ could 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.

Polyominoes move at different speeds depending on their shape or obstacle interactions. We estimate the pivot walking cycles necessary until $c_\mathcal{A}$ has moved to the original position of $c_\mathcal{B}$ with $\#steps = \left\lceil \frac{d(c_\mathcal{A}, c_\mathcal{B})}{\|\vec{d}_\mathcal{A}\|} \right\rceil$. We then only walk $\left\lceil \frac{\#steps}{2} \right\rceil$ and re-align the cubes. When $c_\mathcal{A}$ and $c_\mathcal{B}$ are near enough for magnetic forces to act, we wait to let magnetic attraction pull $e_\mathcal{A}$ and $e_\mathcal{B}$ together. This automatically adjusts the alignment. We also decrease the pivot walking angle α when in close proximity.

C. 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 g_{goal} of the first plan matches with g_{init} of the second. That way, multiple local plans can be connected to form a global plan.

We compare and evaluate plans based on the rotational cost of actions. We only consider longitude magnetic field rotations, not latitude elevation. Let a_i be a rotation of angle β so $\text{cost}(a_i) = |\beta|$. A pivot walking step cost is $\text{cost}(a_i) = |2\alpha|$. The plan cost is the sum of action costs.

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 failure cases used by the local planner: a blocked slide-in direction, a connection is required in a cave of a polyomino, polyominos get stuck in corners or along walls, or the generation of an invalid polyomino. We also impose a maximum movement distance that is sufficient to move a robot along the length and breadth of the workspace.

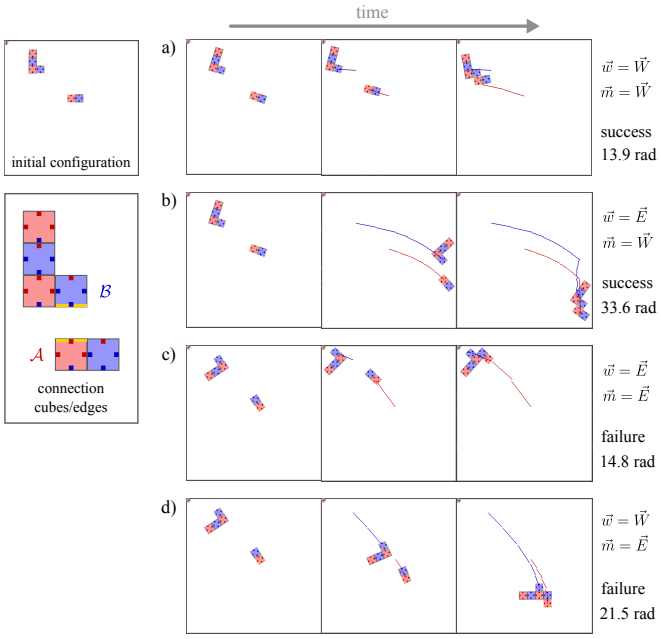


Fig. 4: Illustration of all local plans developed by the local planner by executing different pivot walking directions $\vec{w}$ and slide-in directions $\vec{m}$. On the left side the initial configuration and connection cubes/edges of $\mathcal{A}$ and $\mathcal{B}$ are shown. Choosing $\vec{m} = \vec{E}$ results in failure due to a wrong connection for both c) and d). For the two successful plans, b) has 2.4 times the rotational cost of a), making a) the plan returned by the local planner. Both $c_{\mathcal{A}}$ (red) and $c_{\mathcal{B}}$ (blue) leave a trace when pivot walking.

D. Local Planning Algorithm

Before generating a local plan, all failure conditions that can be pre-checked are evaluated, so that no simulation time is wasted on a plan that is bound to fail from the beginning. Next, the possible slide-in directions are determined and simulated with each possible $\vec{m}$ for both pivot walking directions $\vec{w}$. This means that two plans are developed for an east-west connection and two or four for a north-south connection. Figure 4 shows an example for a north-south connection where both slide-in directions are possible.

The successful plan with the lowest cost is returned. Plans creating invalid polyominoes or polyominoes with caves are omitted by the global planner. The plans are developed in parallel. If one process finishes with a successful plan, the execution of all other processes is canceled. This saves computation time, but might not return the best plan, since fastest computation does not imply lowest rotational cost. Furthermore, the multiprocessing causes non-deterministic behavior.

The algorithm runs in a loop until the plan state s changes to success or one of the failure conditions. The failure and success conditions are evaluated twice per iteration: after aligning and at the end of the loop.

g_{goal} is updated by simulating the determined actions. Actions are appended to A after simulation. We either perform a straight or an offset-align, depending on $e_{\mathcal{A}}$ and $e_{\mathcal{B}}$. After aligning we walk the estimated amount of pivot walking cycles in direction $\vec{w}$, or we wait, if $c_{\mathcal{A}}$ and $c_{\mathcal{B}}$ are in close proximity. If we waited in the previous iteration, we walk in the current one and vice versa.

The stuck condition does not state failure immediately. If polyominoes are stuck, we perform a straight-align and wait as long as the cubes are moving.

IV. Global Planner

The global planner assembles a specified target polyomino $\mathcal{T}$ given an initial configuration g_{init}. The configuration-space is explored by executing local plans. This limits planning to configurations where a connection between two cubes was attempted. Determining how these configurations are explored affects the run time considerably. Generating a local plan in our simulation is intensive. Our global planner plans the assembly of $\mathcal{T}$ with as few local plans as possible.

Our approach enumerates the ways to cut a polyomino into two parts by generating a *two-cut-sub-assembly graph*. This graph functions as a building instruction alongside the exploration of the configuration-space. The algorithm limits the number of cubes in the workspace to the size of $\mathcal{T}$.

A. Two-Cutting Polyominoes

Schmidt et al. [24] used straight-line two-cuts to construct polyominoes with more than trivial sub-assemblies. We define a *two-cut* as a continuous edge path through a polyomino that divides the polyomino into two sub-polyominoes. Furthermore, we only consider *monotone* two-cuts, which means that once the path goes in a direction, it can never go in the opposite direction. Non-monotone two-cuts would create sub-assemblies with caves or holes, which could not be reassembled with our local planner. Due to the usage of two-cuts there are subsets of shapes we can not assemble.

B. Two-Cut-Sub-Assembly (TCSA) Graph

For a target polyomino $\mathcal{T}$, the TCSA graph is $G_{TCSA}(\mathcal{T}) = \{V, E\}$, represented by nodes V and edges E. The nodes of a TCSA graph are polyomino sets. A polyomino set $S(g)$ only enumerates the polyomino types present in a configuration g. If g contains multiple polyominoes of the same type, $S(g)$ stores the number of each polyomino type, but does not distinguish between them.

Two nodes S_1 and S_0 are connected with an edge $\{S_1, t_c, S_0\}$ if one polyomino contained in S_0 can be two-cut by an edge path t_c, so that the resulting polyomino set equals S_1. This provides a perspective on the use of two-cuts and the way $G_{TCSA}(\mathcal{T})$ is built, starting with $\mathcal{T}$. The direction of $\{S_1, t_c, S_0\}$ always goes from S_1 to S_0 and t_c is stored as the weight. S_1 and S_0 can be connected by multiple edges, if there are different connections that produce the same outcome.

We built a TCSA graph by working through each newly added node in V in a breadth-first-search manner. The first node added to V is $S_{\mathcal{T}}$, a polyomino set only containing the target polyomino.

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 $\mathcal{A}$. The cutting results in the two sub-polyominoes $\mathcal{A}_1$ and $\mathcal{A}_2$.

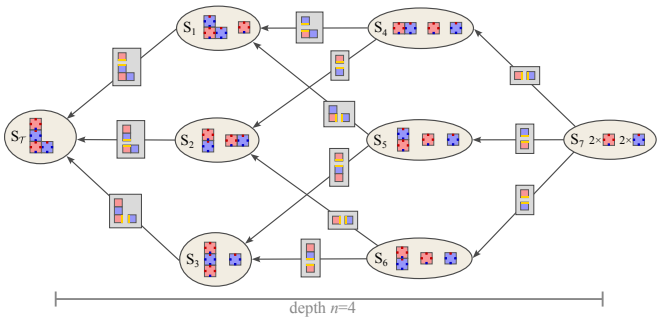


Fig. 5: TCSA graph for a four-cube L-shape. Ellipses show the polyomino sets. If the polyominoes of a set are not numbered, there is only one occurrence of this polyomino. The weights of edges are illustrated as rectangular boxes containing the polyominoes that need to be connected at specific edge paths, marked in yellow.

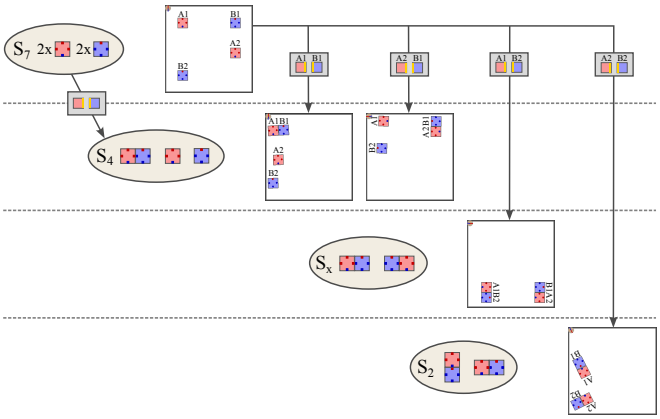


Fig. 6: All connection options when connecting a red cube at the west edge of a blue cube to get from S_7 to S_4. Developing local plans for different polyomino pairs leads to different goal configurations. $(\mathcal{A}_1, \mathcal{B}_1)$ and $(\mathcal{A}_2, \mathcal{B}_1)$ lead to configurations with the desired polyomino set S_4, but $(\mathcal{A}_2, \mathcal{B}_2)$ leads directly to S_2. All these sets can be found in the TCSA graph of Figure 5. The goal configuration of $(\mathcal{A}_1, \mathcal{B}_2)$ holds the set S_x, which cannot be found in Figure 5, so it cannot be used for further global planning.

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 $\mathcal{A}_2$. 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, and multiple outgoing edges to different nodes, which can be observed in Figure 5, 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 (single cubes), as shown for S_7 in Figure 5. 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.

Our TCSA graph implementation stores nodes in a hash-table, making it efficient to access nodes and connected edges, or check if a polyomino set is contained in $G_{TCSA}(\mathcal{T})$.

C. 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 an initial configuration already contains sub-assemblies that are not usable for assembling $\mathcal{T}$. Except for $S_\mathcal{T}$, 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 5, three outgoing edges provide three connections to choose from. Multiple polyominoes of the same type produce even more options to consider. Assume the global planner decides to connect a red cube at the west edge of a blue cube to make configuration g_2 with $S(g_2) = S_4$. Since S_7 contains two red and two blue cubes, there are four ways to achieve this. Figure 6 illustrates the four 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 polyomino types $\mathcal{A}$ and $\mathcal{B}$. When $\mathcal{A}$ and $\mathcal{B}$ are 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.

With multiple edges and various polyomino pairs per edge, many options emerge for the global planner to consider. We examined three option sorting strategies.

a) Minimal Distance: sorts connection options based on the distance between the connection-cubes $c_\mathcal{A}$ and $c_\mathcal{B}$. A smaller distance often requires less movement to establish a connection, which means shorter simulation time and lower plan cost. Due to sliding on walls and different pivot walking distances, this is not true in general. Less movement can prevent unwanted sub-assemblies.

b) Grow Largest Component: sorts options by maximum polyomino size $\hat{n}$ of the resulting polyomino set. TCSA edges leading to sets with the biggest polyominoes are preferred, since larger polyominoes generally move faster. When the options for S_4 in Figure 5 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. Options within each class are sorted by minimal distance.

c) Grow Smallest Component: sorts by the smallest maximum size of polyominoes in polyomino sets. This avoids working with large polyominoes, which translate faster, but need more simulation time to perform rotations.

D. Use of Local Planner

The local planner develops plans for connections chosen from the different connection options. For that, it needs only one edge-pair out of the connections stored in the weight of a TCSA edge. Whenever a path consists of both north-

south and east-west connections, a north-south connection is preferred to perform offset-aligning.

When the local planner successfully connects the desired polyominoes, other connections could occur. This can be seen in Figure 6, when $\mathcal{A}_2$ and $\mathcal{B}_2$ are connected. The resulting polyomino set matches S_2 instead of S_4 in Figure 5.

If the resulting polyomino set is not contained in $G_{TCSA}(\mathcal{T})$, it is not possible to assemble the target from that configuration. This occurs in Figure 6 when connecting $\mathcal{A}_1$ and $\mathcal{B}_2$. For global use, we add a failure condition to the local planner, which 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.

E. Global Planning Algorithm

The global planner 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 $\mathcal{T}$. Because the local plans were created using a TCSA graph, $|P| < n$ holds true. The algorithm explores the configuration-space along $G_{TCSA}(\mathcal{T})$ in depth-first-search manner. The algorithm starts with g_{init} as the current configuration g and builds the TCSA graph for $\mathcal{T}$.

Each iteration first determines the connection options O for g. O only needs to be sorted the first time a configuration is encountered. Whenever a connection option is popped from O, the option will never be considered again for this configuration.

Options are stored per configuration g, not for the polyomino set $S(g)$. Nodes in $G_{TCSA}(\mathcal{T})$ can be encountered multiple times and will never be eliminated from planning.

The algorithm works through O in the order determined by the option sorting that was applied in advance. This is done until a valid local plan is found, or no options are left.

If a valid local plan is found, p_{new} is pushed onto 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. If no valid option for g can be found, the algorithm falls 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. 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 $\mathcal{T}$.

V. Simulator

Our simulator modeling the behavior of magnetic modular cubes uses the 2D physics library Pymunk[1]. It is light-weight and capable of running headless, but also offers an interface for Pygame[2], which we use to visualize developed motion plans and to allow user controls. This 2D simulator can only approximate 3D movement, in particular pivot walking. This trades simulation accuracy for faster simulation time, enabling global planning in a reasonable time.

Random polyominoes and initial configurations are created with a seed-based pseudorandom number generator. 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. The global planner states a timeout failure after a planning time of 600 seconds. We do not timeout 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. The experiments were conducted 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.

[1] Pymunk: `www.pymunk.org` [2] Pygame: `www.pygame.org`

VI. Results

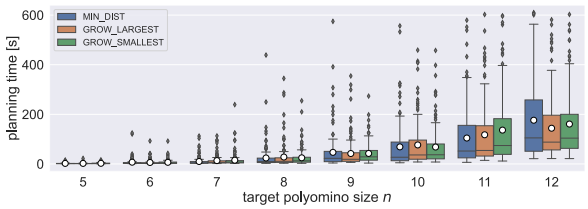
All experiments are available in the thesis [13].

A. Assembly for Polyomino Size

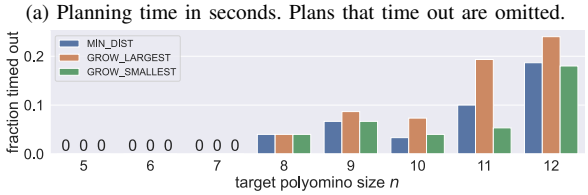
These experiments were conducted with randomly generated initial configurations and randomly generated polyominoes of specific size n. To maximize the variety of possible polyomino shapes, the number of red cubes was set to $n_{red} = \lfloor \frac{n}{2} \rfloor$ as indicated in [18]. The workspace is of size $50r_C \times 50r_C$ and for each target size 150 samples were taken.

Figure 7a shows the distribution of planning time and Figure 7b shows 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. No 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. When increasing n, a wider spread of planning time can be observed. Outliers can reach planning times close to the timeout of 600 seconds.

The option sorting strategies make no noticeable difference in planning time. By fraction of timeouts, growing the largest component often exceeds the other two strategies, clearly

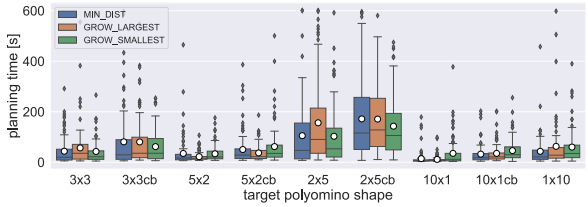


(a) Planning time in seconds. Plans that time out are omitted.

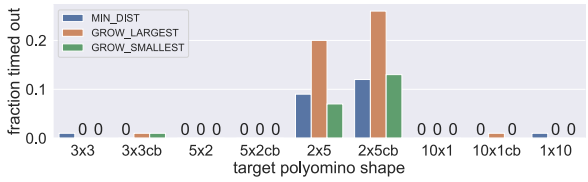


(b) Fraction of plans that timed out.

Fig. 7: Planning time and fraction of timeouts. All option sorting strategies are compared with 150 samples each.



(a) Planning time in seconds. Plans that time out are omitted.



(b) Fraction of plans that timed out (over 600 s).

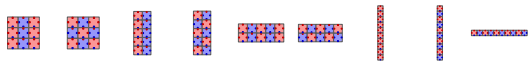
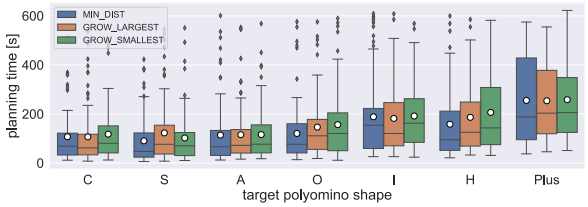
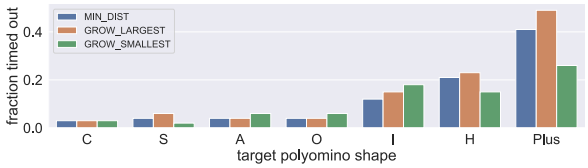


Fig. 8: Planning time and fraction of timeouts for rectangular polyominoes. Sorting strategies compared with 100 samples each.



(a) Planning time in seconds. Plans that time out are omitted.



(b) Fraction of plans that timed out (over 600 s).

Fig. 9: Planning time and fraction of timeouts for special polyominoes. Sorting strategies are compared with 100 samples each.

visible for $n = 11$, where growing the largest component is at 20% and the others are under 10% of plans timed out.

B. Assembly of Custom Polyominoes

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

1) Width/Height and Cube Pattern: This focuses 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.

The checkerboard pattern requires the longer planning time for all types of rectangular polyominoes (Figure 8a). The "3x3" polyomino takes on average 50 seconds, but the "3x3 cb" polyomino takes 75 seconds planning time with a wider spread and worse 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 worse than all other rectangular polyominoes. While most 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. The "2x5" is the only shape with 5% to 25% timeouts (Figure 8b). All other shapes experience nearly no timeouts.

2) Special Polyomino Shapes: This examines the assembly of special polyomino shapes. 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 of these polyominoes are built with the checkerboard pattern to achieve equal amounts of red and blue cubes. The size of all these polyominoes, except for "C", is $n = 12$.

The planning time and fraction of timeouts are evaluated in Figure 9. Assembling 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 polyominoes of size 12, already examined in Figure 7a. The fraction of timeouts is mostly under 5%, much less than the 25% for the random polyominoes evaluated in Figure 7b. Shape "O" (4 wide) has the worst performance of the four.

The three thick shapes perform much worse 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. Timeouts reach 20% for "I" and "H", and even 30% to 50% for the "Plus" shape. The "Plus" polyomino holds the worst performance out of all custom and random polyominoes evaluated.

Caves and holes do not impact the performance of the global planner. The option sorting strategies do not show a pattern, but have strong differences. Growing the smallest component, while assembling the "Plus" shape, reduces the fraction of timeouts by half.

VII. Conclusion

We presented a heuristic motion planner to assemble polyominoes from magnetic modular cubes [6] in $SE(2)$. The video[3], planner and simulator[4], and thesis [13] are available.

Designing a local planner that is able to navigate around obstacles could be a interesting direction for future work. In hardware experiments, 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. [19].

[3] https://youtu.be/C7Bj3Hco_G4
[4] https://github.com/RoboticSwarmControl/2024MagneticCubes

References

[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)*. SIAM, 2021, pp. 1740–1756.

[2] J. Balanza-Martinez, T. Gomez, D. Caballero, A. Luchsinger, A. A. Cantu, R. Reyes, M. Flores, R. Schweller, and T. Wylie, "Hierarchical shape construction and complexity for slidable polyominoes under uniform external forces," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2020, pp. 2625–2641.

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

[4] 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)*, 2014, pp. 6751–6756.

[5] 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*, vol. 82, no. 2, pp. 165–187, Feb 2020. [Online]. Available: https://doi.org/10.1007/s00453-018-0483-9

[6] A. Bhattacharjee, Y. Lu, A. T. Becker, and M. Kim, "Magnetically controlled modular cubes with reconfigurable self-assembly and disassembly," *IEEE Transactions on Robotics*, vol. 38, no. 3, pp. 1793–1805, 2022.

[7] 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*. IEEE, 2005, pp. 3684–3691.

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

[9] 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*, vol. 28, pp. 782–790, 2020.

[10] S. C. Cirlos, T. Gomez, E. Grizzell, A. Rodriguez, R. Schweller, and T. Wylie, "Simulation of multiple stages in single bin active tile self-assembly," in *Unconventional Computation and Natural Computation*, D. Genova and J. Kari, Eds. Cham: Springer Nature Switzerland, 2023, pp. 155–170.

[11] 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*, vol. 75, no. 471, p. 117–119, 1991.

[12] J. Keller, C. Rieck, C. Scheffer, and A. Schmidt, "Particle-based assembly using precise global control," *Algorithmica*, vol. 84, no. 10, pp. 2871–2897, 2022.

[13] K. Keune, "Motion planning for reconfigurable magnetic modular cubes in the 2-dimensional special euclidean group," Braunschweig, Niedersachsen, Germany, June 2023, available at https://github.com/RoboticSwarmControl/2024MagneticCubes/tree/main/thesis.

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

[15] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[16] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and computational robotics*, pp. 303–307, 2001.

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

[18] 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)*, 2021, pp. 6977–6982.

[19] Y. Lu, A. Bhattacharjee, C. C. Taylor, J. Leclerc, J. M. O'Kane, M. Kim, and A. T. Becker, "Closed-loop control of magnetic modular cubes for 2D self-assembly," *IEEE Robotics and Automation Letters*, vol. 8, no. 9, pp. 5998–6005, Sep. 2023.

[20] A. Mueller, "Modern robotics: Mechanics, planning, and control [bookshelf]," *IEEE Control Systems Magazine*, vol. 39, no. 6, pp. 100–102, 2019.

[21] 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)*. IEEE, 2016, pp. 1–6.

[22] W. Saab, P. Racioppo, and P. Ben-Tzvi, "A review of coupling mechanism designs for modular reconfigurable robots," *Robotica*, vol. 37, no. 2, pp. 378–403, 2019.

[23] 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*, vol. 5, no. 2, pp. 985–992, 2020.

[24] 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*, vol. 3, no. 4, pp. 3521–3528, 2018.

[25] 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*, vol. 103, no. 2, pp. 205–224, 2015.

[26] 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*. IEEE, 2007, pp. 2773–2778.

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