Complete Heterogeneous Self-Reconfiguration: Deadlock Avoidance Using Hole-Free Assemblies

Daniel Pickem, Magnus Egerstedt, and Jeff S. Shamma
DCL Lab, Georgia Institute of Technology, Atlanta, USA

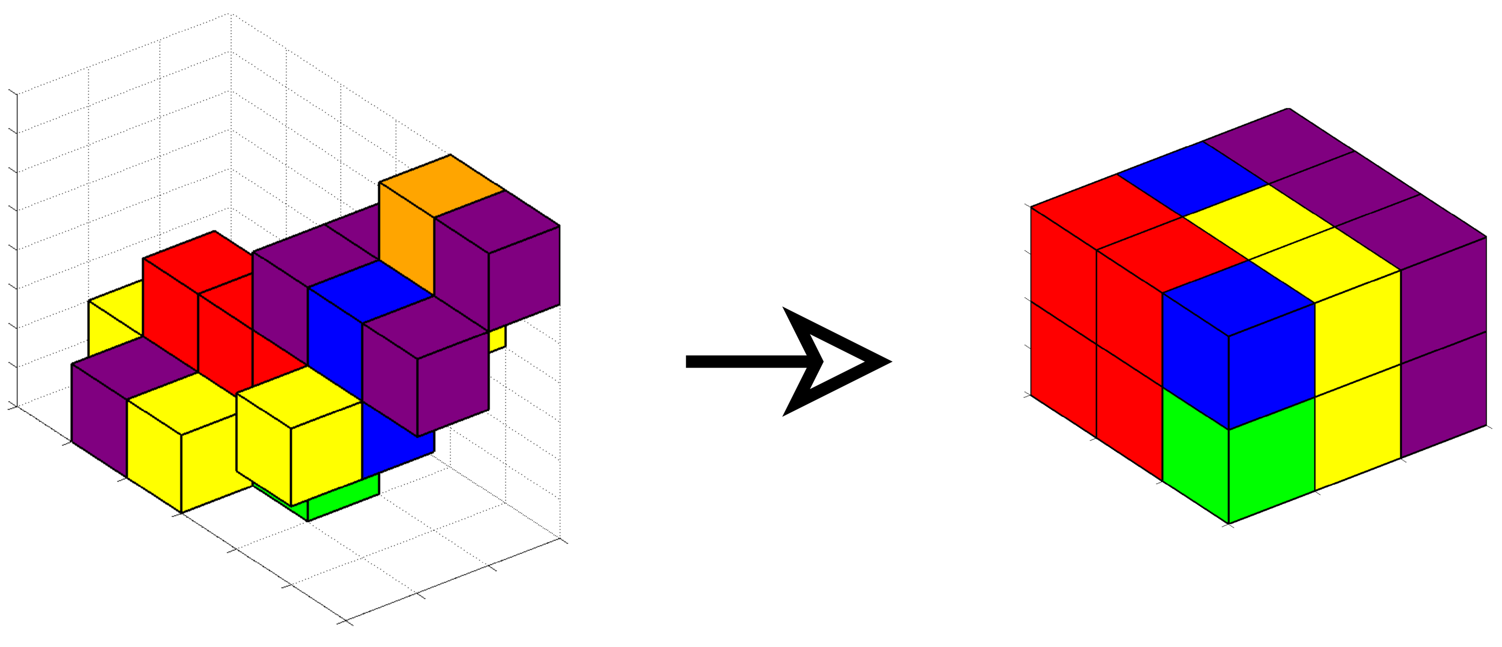
Georgia Robotics and InTelligent Systems Laboratory

DCL Decision and Control Laboratory

Introduction

What is heterogeneous self-reconfiguration?

- A self-reconfigurable robot is comprised of individual modules.
- Modules have different properties (e.g. shape, size) and/or different capabilities.
- Goal: Reconfigure an initial configuration $\mathcal{C}_I$ into a target configuration $\mathcal{C}_T$.

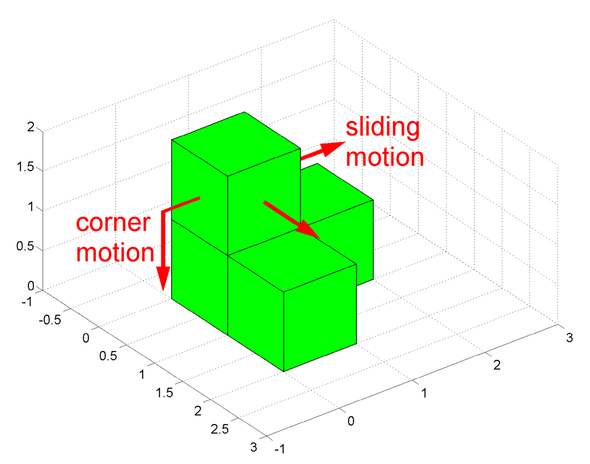


Motivation for heterogeneous self-reconfiguration.

- These systems can adapt to tasks by changing their morphology.
- Such systems can be easily extended and repaired by adding new modules.

System Representation

- Modules are represented by unit cubes.
  - ▷ Dimension $\delta = 1$
  - ▷ Origin $x_i \in \mathbb{Z}^3$
  - ▷ Properties $p_i \in P$
- Cubes are embedded in a discrete three-dimensional unit lattice.



Assumptions and Constraints

Assumptions:

- The initial overlap of $\mathcal{C}_\mathcal{I}$ and $\mathcal{C}_\mathcal{T}$ is exactly one cube.
- The initial and final configurations are hole and enclosure-free.

Constraints:

- Connectivity constraint: The configuration has to remain connected at all times.
- Permanence constraint: Once a cube reaches its target it remains fixed to that position.

Planning Approach

Self-reconfiguration requires to move every cube $c_i \in \mathcal{C}_I \setminus \mathcal{C}_T$ to a matching position in the target configuration $\mathcal{C}_T$.

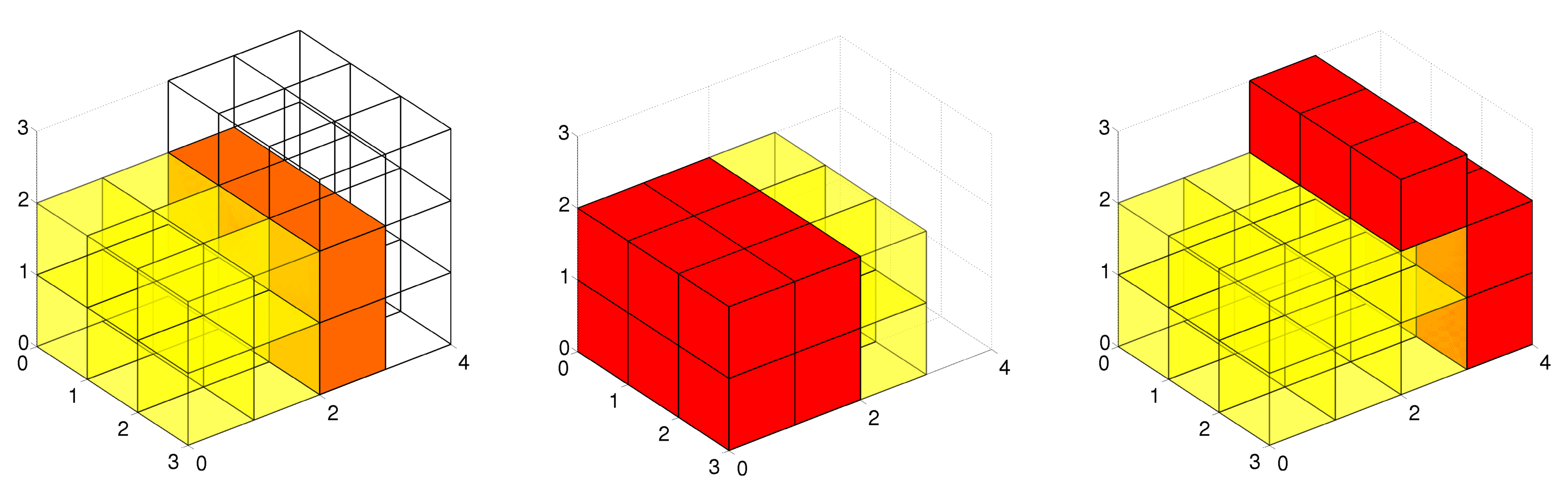
At each iteration, do the following:

- Determine the movable set $\mathcal{M}$, i.e. which cubes can currently be moved.
- Determine the reachable set $\mathcal{R}$, i.e. which target positions can currently be reached.
- Assign a movable cube $c_i \in \mathcal{M}$ to a target position $r_i \in \mathcal{R}$ or execute assignment resolution.
- Determine the planning space $\mathcal{N}$ through which a path can be planned.
- Plan a path p_i from c_i to r_i through $\mathcal{N}$ and execute p_i.

Overlapping set $O = \mathcal{C} \cap \mathcal{C}_T$

Movable set $M = \mathcal{C} \setminus (A \cup I \cup \mathcal{C}_T)\}$

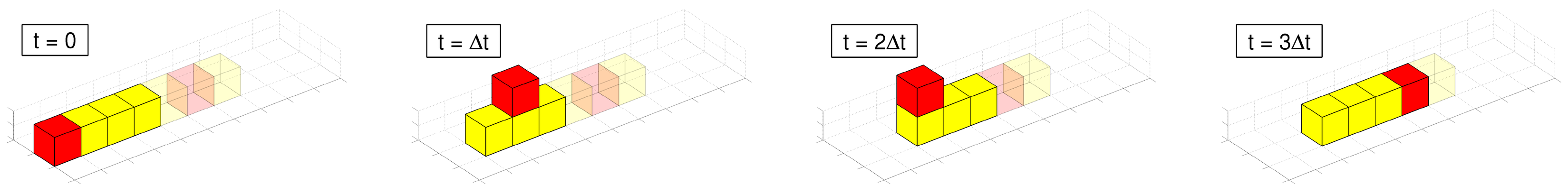
Reachable set $R = \mathcal{N} \cap \mathcal{C}_T$



Assignment Resolution / Deadlock Avoidance

Planning a path requires a valid assignment of a cube $c_i \in \mathcal{M}$ to a position $r_i \in \mathcal{R}$ with matching properties. The absence of valid assignment creates deadlocks that we resolve using assignment resolution.

- A **valid assignment** is a pair $a_i = \{c_i, r_i\}$ with $c_i \in M$ and $r_i \in R \setminus H_t$ such that $p_k(c_i) = p_k(r_i)$, $\forall p_k \in P$ (with P being the set of properties and H_t the set of positions that would create holes).
- **Assignment resolution** moves a movable cube to a random position $m_i = \mathrm{rand}(\mathcal{N}(\mathcal{C}) \setminus R)$ if no cube $c_i \in M$ matches all properties p_k of any position $r_i \in R$,



- **Fact:** Assignment resolution will enable the computation of a valid assignment with probability 1.
- **Fact:** Using assignment resolution, the reconfiguration algorithm guarantees a successful reconfiguration in the absence of holes.

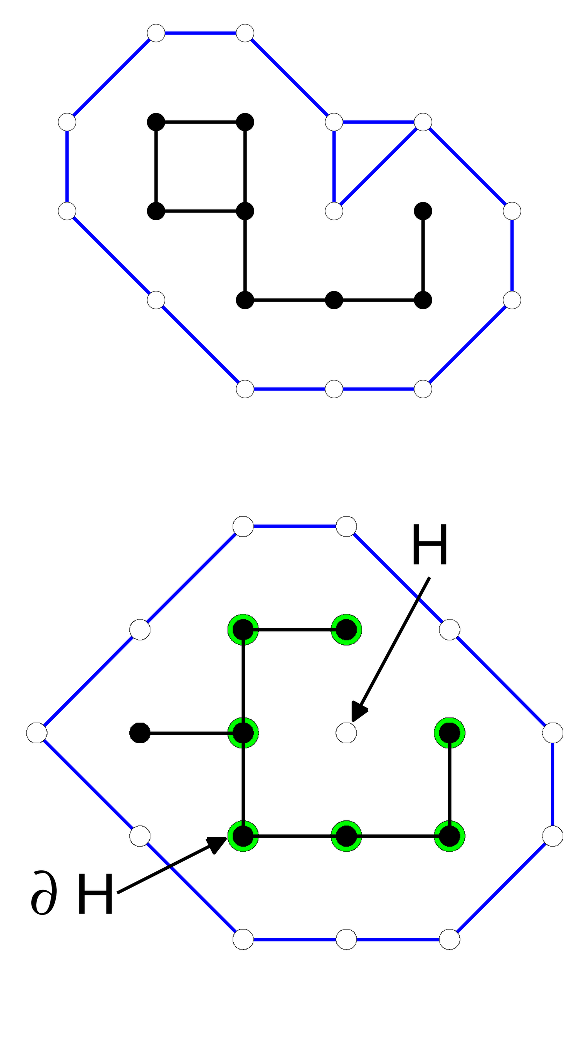
Hole Detection

- A hole H is an unreachable empty target position or a set thereof.
- The boundary of a hole ∂H separates H from the rest of the planning space $N(\mathcal{C})$.
- A hole exists if $N(\mathcal{C})$ contains two or more connected components.
- Hole detection is based on Graph Laplacian of $N(\mathcal{C})$, by which the number of connected components is computed.
- **Fact:** The hole detection algorithm detects a hole iff there exists a hole.

Algorithm 1 Hole Detection

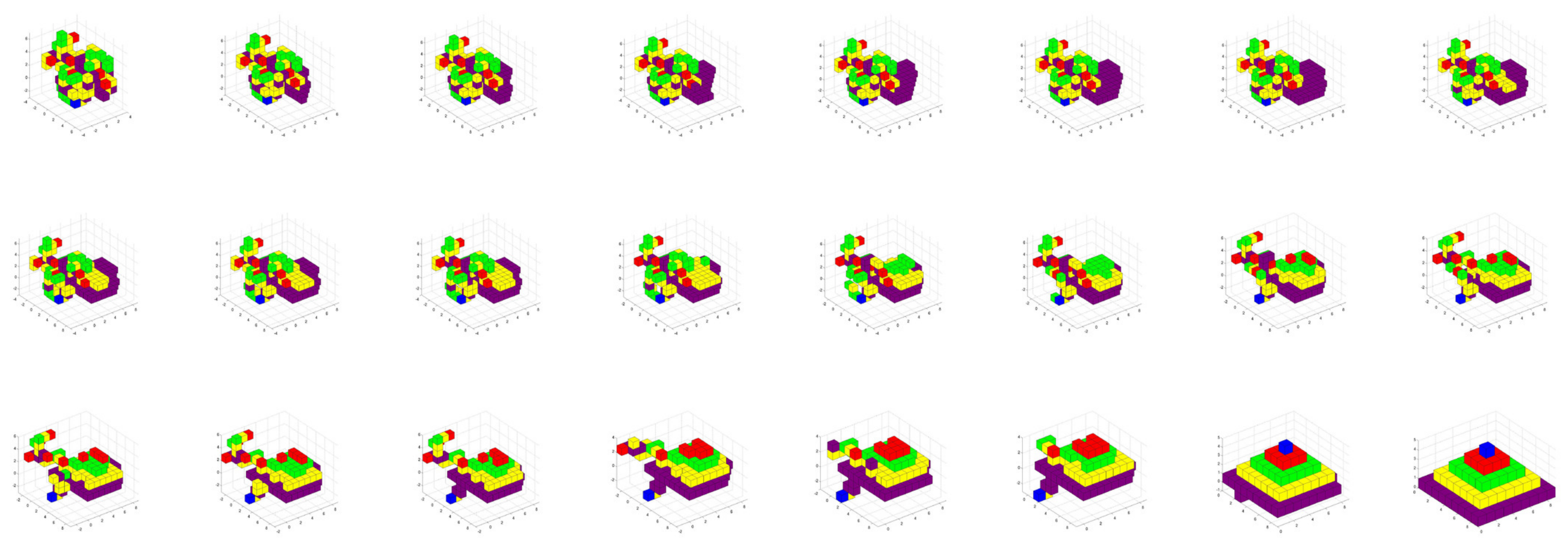
Require: input $a = \{c_i, r_i\}$, $\mathcal{C}$

Compute $\mathcal{N}(\mathcal{C})$
Compute G_C of $\mathcal{N}(\mathcal{C})$
Compute L of G_C
if $|\lambda_i = 0| > 1$ **then**
  Return true
else
  Remove r_i from $\mathcal{N}(\mathcal{C})$
  Update c_i's origin to r_i (in $\mathcal{C}$)
  Recompute $\mathcal{N}(\mathcal{C})$, G_C, and L
  if $|\lambda_i = 0| > 1$ **then**
    Return true
  else
    Return false
  end if
end if



Simulation Results

Size	Steps	Detected Holes	# of Resolutions
10	33	0	3
20	69	0	1
30	107	0	0
40	150	0	0
50	233	0	1



http://www-i6.informatik.rwth-aachen.de
<surname>@cs.rwth-aachen.de