A New Concept of Planar Self-Reconfigurable Modular Robot for Conveying Microparts

Benoît Piranda[a], Guillaume J. Laurent[b], Julien Bourgeois[a,c], Cédric Clévy[b], Sebastian Möbes[b], Nadine Le Fort-Piat[b]

[a] *University of Franche-Comté, Montbéliard, France.*
[b] *Automatic Control and Micro-Mechatronic Systems Department, Institut FEMTO-ST, UMR CNRS 6174, ENSMM, Besançon, France.*
[c] *Computer Science Departement, Institut FEMTO-ST, UMR CNRS 6174*

Abstract

Modularity and self-healing are two interesting properties that could help to design more flexible conveyors of micro-objects. In the Smart Blocks project, we propose to design a 2D modular and self-reconfigurable robot composed of centimeter-scale sliding blocks that embed their own actuators and control electronics. This article presents a proof-of-concept of the linkage and of the traveling system as well as an algorithm able to reconfigure a set of blocks from a spatial configuration to another one. Prototype blocks have been realized using electro-permanent magnets which show a good motion speed while saving power consumption during the linkage. Our reconfiguration algorithm is implemented in a simulator software showing in real-time the reconfiguration of the robot.

Keywords: Electro-permanent magnets, reconfiguration, self-reconfigurable robot, part feeder

1. Introduction

Conveyors are usually designed as monolithic entities solving one problem at a time. If monolithic design fits the need of fixed types of environments and/or objects, it lacks flexibility to environment changes and failures that occur at small scales [1]. Furthermore, to convey micro-objects, a MEMS-based monolithic conveyor is limited by the size of a wafer and it will use more surface than a modular one. To solve these problems, our idea is to build a modular conveyor composed of hundreds of similar blocks that can

detect micro-objects with sensors, move them with MEMS actuators and communicate with each other to form a flexible conveying path. A further idea is that the blocks will be able to move by themselves through the use of side actuators to enable self-reconfigurability. A self-reconfigurable conveyor will be able to automatically replace blocks that have failed with working ones (self-healing). Furthermore, some environments like medicinal production have very detailed guidelines [2] to maximize the quality of the products and to lower the possibility of product contamination. Human intervention for conveyor reconfiguration will contaminate the conveyor. There are, therefore, direct benefits to have a self-reconfigurable conveyor. Building this conveyor is the objective of a bigger project called Smart Blocks[1].

Several projects have influenced our work, but the first thing to mention is the Smart Surface project[2] which is our direct predecessor. Different hardware and software systems have been developed to produce an air cushion in order to move small, flat objects. One of these, a tilted-air-jet surface, reached a size of 9mm×9mm [3] and gave therefore the target system dimensions for the Smart Blocks project. Ciliary motion systems [4] are also good candidates since they required only low voltages and match the target dimensions. Additionally, there were algorithms developed in order to control multiple sensor-actuator units with their own processor in a decentralized way [5] and more general work conducted [6]. Other interesting ideas come from the modular robot field. The projects like M-TRAN [7], Superbot [8], Roombots [9] and Molecube [10] have already shown motions of autonomous parts and self-assembly albeit in bigger dimensions. The miniaturization of their mechanical connection systems is complex and does not seem to be the right solution for a smaller system. Another connection system has been realized by Neubert et al. [11] using a Fields Metal solder, melting at 60°C. This principle uses electrical heating of the contact area where the solder is deposited, to melt it and to let it solidify again. The connection is very strong and can also be used for power or data connections. The problem is that there is no attraction force to move the modules. In terms of connection and attraction, the most inspiring work comes from the Pebbles project [12] using small cubes (12mm×12mm), capable of forming two dimensional shapes using electro-permanent (EP) magnets. The advantage of EP magnets is that

[1] `http://smartblocks.univ-fcomte.fr`
[2] `http://www.smartsurface.cnrs.fr`

they are able to keep their polarity after a short energizing phase. However, Pebble's cubes cannot move, they are just able to disassemble.

In the Smart Surface project, we conveyed small objects using arrayed-MEMS actuators. Our new modular conveyor will use these actuators so that each block will embed arrayed-MEMS actuators on its upper face. Miniaturization of the block is one of the key aspects of the project. The final block edge length is foreseen to be of 10 mm, including everything from supplying the MEMS array and connecting the blocks but also to move them. Figure 1 shows a representation of the final conveyor.

The focus of this article is twofold. First, we propose to use EP magnets to design a linear motor able to move $1cm^3$ blocks. Using this motor, one block can slide on another one and can be stopped in every state keeping a strong connection without any power consumption. This means that after the system has formed its optimal configuration it can perform its conveying function using no other resources for the linkage. Second, using this special kind of actuation, we have also proposed a distributed reconfiguration algorithm which is able to form any 2D shape required for the conveyor. This algorithm is implemented in our simulation software VisibleSim simulating exchanges of messages and showing in real-time the reconfiguration of the robot.

2. Sliding blocks

This part presents the design of a new kind of self-reconfigurable modular robot based on sliding blocks. We propose to use EP magnets inside a linear motor able to move blocks directly on one another.

2.1. Electro-permanent (EP) magnets

The basic part of the motor unit is the EP magnet. It consists of a coil with a specific permanent magnet core, like AlNiCo. This material is an alloy made from aluminum, nickel and cobalt which has a remanence similar to neodymium magnets (around 1.2T), but a relatively weak coercive field strength around 50 to 100kA m^{-1} (in contrast to neodymium magnets which have a coercive field over 1000kA m^{-1}). The result is a strong magnetic force (depending on the remanence), but also the ability of a very easy magnetization and demagnetization (depending on the coercive field strength). Wrapping a coil around the AlNiCo core, a magnetic field can be generated to switch the magnet polarity in a very short time. The result is a bistable

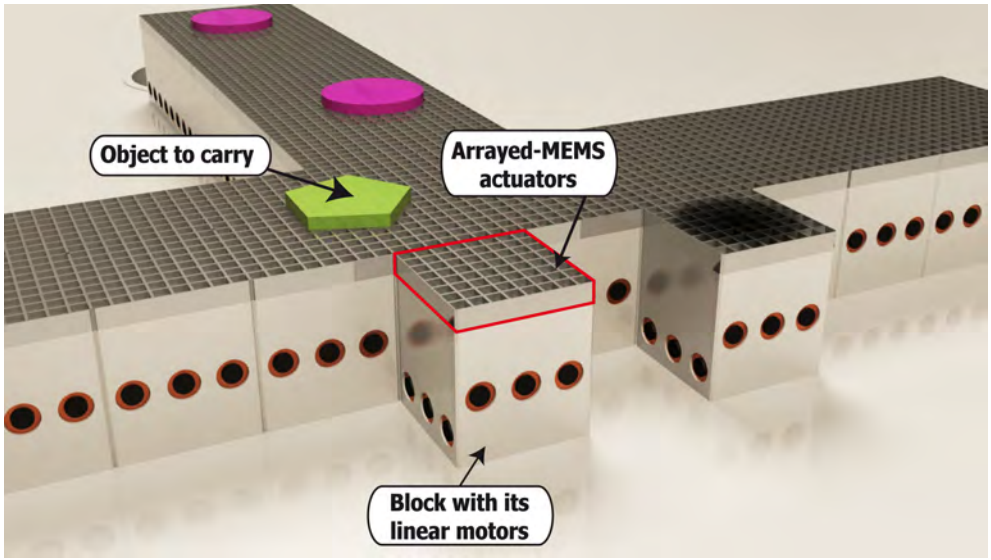


Figure 1: Example of using reconfigurable blocks for conveying objects.

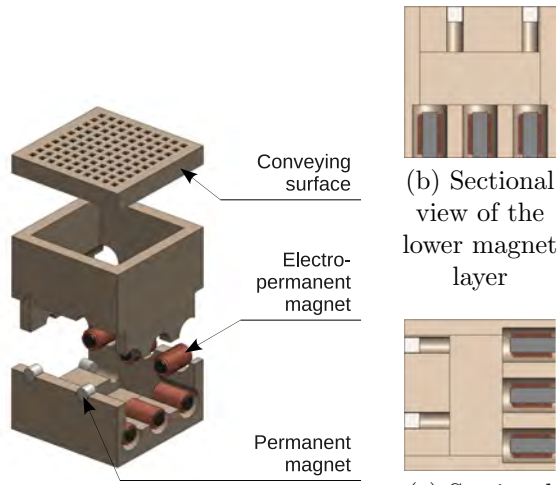
system, able to have an attraction or a repulsion mode like a switchable permanent magnet.

2.2. Linear motor design

A linear motor is an electric motor that has its stator and rotor "unrolled" so that instead of producing a torque (rotation) it produces a linear force along its length. We propose to use this principle to move a "rotor" block along "stator" blocks. The rotor part is composed of two cylindrical 1mm-by-1mm neodymium magnets. The stator part consists of three EP magnets per blocks which have a size of around 2mm in diameter and 3mm in length.

To be able to move the magnets in a chosen direction, the distance between the two neodymium magnets has to be 50% bigger than the distance between two EP magnets on the other side. As each block must be able to move along one another, we put on every block two active sides, each one containing three EP magnets and two passive sides, each one containing two permanent magnets. A working configuration needs to have two equal sides next to each other, as seen in Figure 2.

The block housing, seen in Figure 2a, has been made by rapid prototyping



(a) 3D view

(b) Sectional view of the lower magnet layer

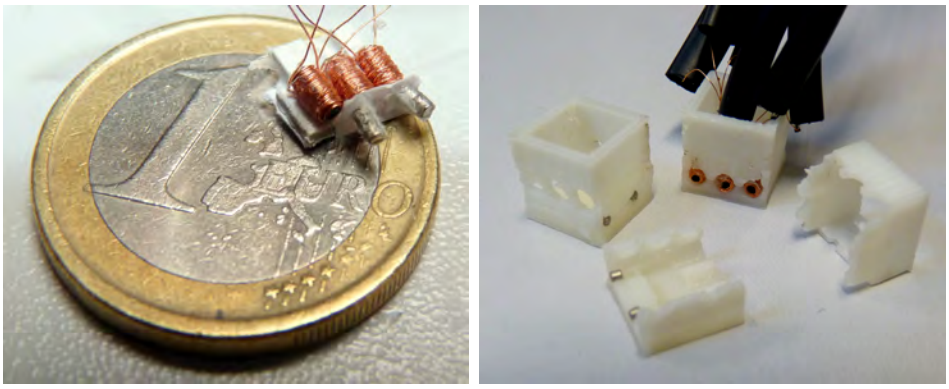
(c) Sectional view of the upper magnet layer

Figure 2: View of the linear motor units within a block with the EP and the neodymium magnets. The block is a 10mm cube and consists of an upper and a lower part.

with a fused deposition modeling machine. To build the EP magnets, the AlNiCo core which has a diameter of 1mm and a length of 3mm, has been placed in a vice and wrapped with 60 turns of 0.1mm enameled copper wire in three layers. To avoid uncoiling, the whole part is covered by a thin layer of cyanoacrylate-based glue. The final assembly has been done by placing and gluing all magnets in the lower part of the block, before gluing the upper part on it. The separate parts and the final assembly can be seen in Figure 3.

2.3. Electronics design

To supply the EP magnets with current in both directions a H bridge per coil is required. To reduce the number of components, we used three half-bridges linked to a shared one. As it can be seen in Figure 4, every coil has its own half-bridge, allowing the choice of the direction of the electric current through its EP magnets. Additionally there is a fourth half-bridge, able to decide which chosen current direction should be powered. Every half-bridge consists of two MOSFETs, a P-channel for the upper side and a N-channel for the lower side.



(a) Three EP magnets (with AlNiCo core) in front of two neodymium magnets

(b) Rapid prototyping parts of the cube, one block is assembled with three EP magnets (active side, stator) and another one with two permanent magnets (passive side, rotor).

Figure 3: Separate parts and an assembled block

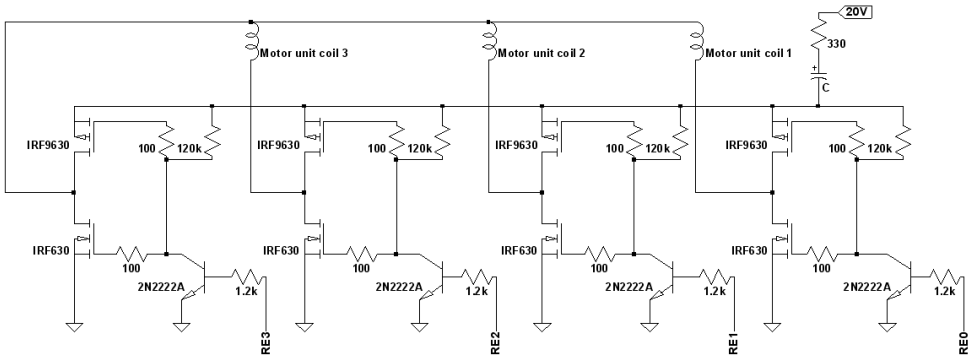


Figure 4: Schematics of the motor unit electronics

Because the resistance of the coil is very low, a 15V difference causes more than 15A current in the wires. On the one hand, it would be a thermal problem for the 0.1mm wire to be powered for more than a few micro-seconds, on the other hand, most power supply are not fast nor powerful enough to guarantee this current. Therefore, a capacitor is used to store and to deliver the required energy.

The power supply of the blocks is external. Some specific electric push-pins will be designed on lateral sides to enable the transmitting of the power between blocks. However a capacitor is required in each blocks to store enough power for a current pulse.

2.4. Motor control

With regard to a standard linear motor, we are using only two states for the EP magnets. Actually, most electric motors are using a sinusoidal control to reach a smooth motion. To move the "rotor" block from one "stator" to another, six states are required. The state transition and its effect on the movement can be seen in Figure 5.

If the polarization of the motor unit should show, for instance, a SSN[3] state, the first two half-bridge bipolar transistors will get a 5V signal, opening the upper side transistor. The third one will stay with no signal and an opened low side transistor. Because the fourth half-bridge has got no signal and connects therefore all EP magnets to the ground, the current will flow from the two first upper sides over the corresponding magnets mainly to the fourth low side and to the ground. Hence, the first two magnets have been powered but not the last one. For this, the fourth half-bridge needs to be switched, whereby only the third EP magnet will be powered. The result is that if the switching of all magnets is necessary, two shots[4] has to been done successively.

The capacitor needs some time to recharge after a shot. This time depends on the voltage supply, on the resistor in front of the capacitor and on the duration of the last shot. The capacitor voltage will therefore be read over a one to ten voltage divider with an analog-digital-converter (ADC) channel, so that the next shot can be started as soon as the sufficient voltage is reached.

[3]S stands for south and N for north, meaning the state of the magnetic pole that points outside. It is noticeable at the block sides.

[4]The short signal from the controller, causing an opening of certain upper side transistors and therefore the switching of one or more magnet polarities will be called "shot".

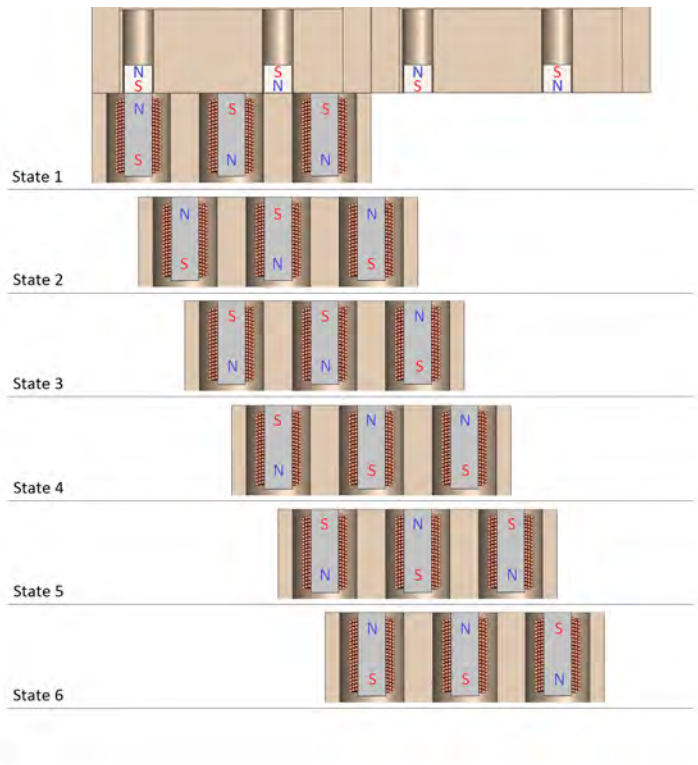


Figure 5: The motor state diagram shows the 6 different states of the EP magnets and the changed position in respect to the passive sides of two blocks. The magnetic poles are marked with S and N for South and North. As every block has two permanent magnets on each of its passive sides, six states move the "rotor" block from one "stator" to another (10mm).

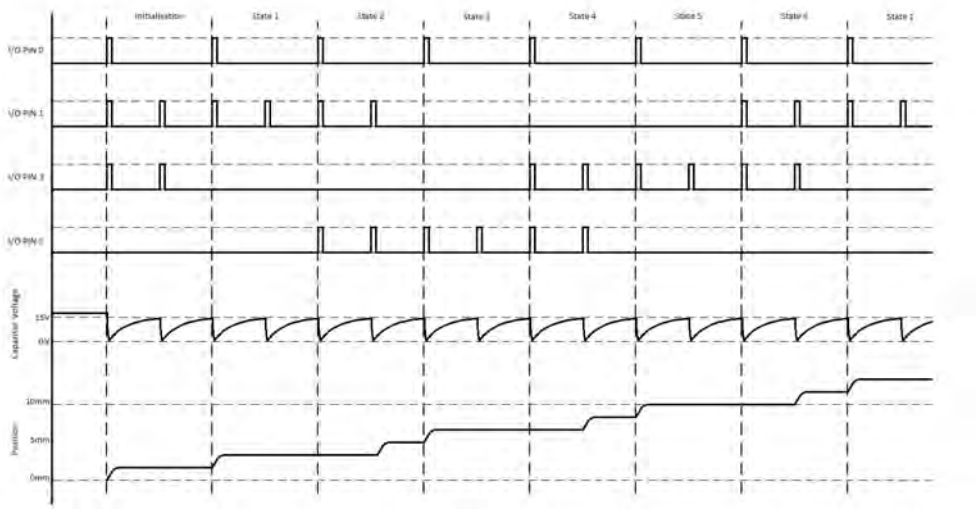


Figure 6: The timing diagram shows the signals, coming from the IC, I/O PIN are labeled from 0 to 3. Additionally, the voltage of the capacitor and the displacement of the block is shown.

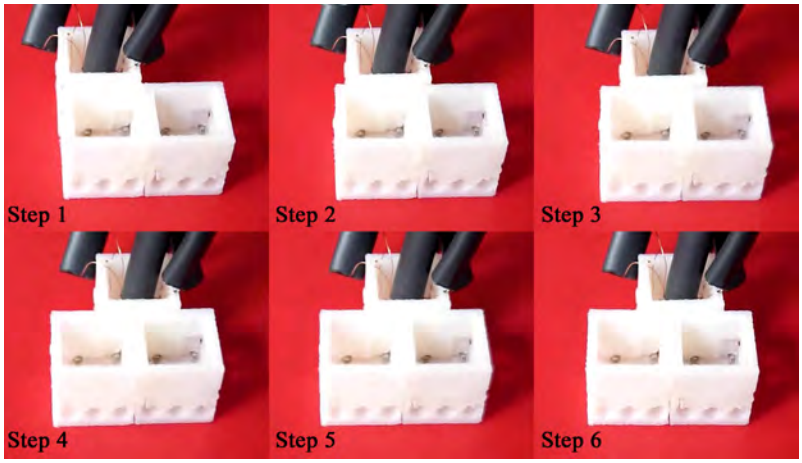


Figure 7: Snapshots of the motion (six-states sequence).

To generate the switching sequence presented in Figure 6, we used a Microchip dsPIC30F4011. Four output pins are used to give the signal to the half-bridges and one ADC input measures the capacitor voltage. In the final solution, we will need 8 I/O pins for running two motor units. A suitable and smaller IC from Microchip would be the 28 pin PIC16F723A in QFN design with a size of 4mm $\times$ 4mm.

3. Experimental results

3.1. Motion and speed

We carried out some experiments to validate this concept of sliding blocks. The motion is stable and reproducible but a bit jerky due to on-off state transitions. Figure 7 show a complete sequence which can be further appreciated in the video clip accompanying this paper[5].

The speed of displacement has been evaluated through one hundred sequences and measuring the time. The mean speed was about 0.61s for 1cm (or 10 blocks distance in 6.1s) that is to say 16.4mm/s.

[5]also available at this address: `http://smartblocks.univ-fcomte.fr/actuation_EP_magnets.avi`

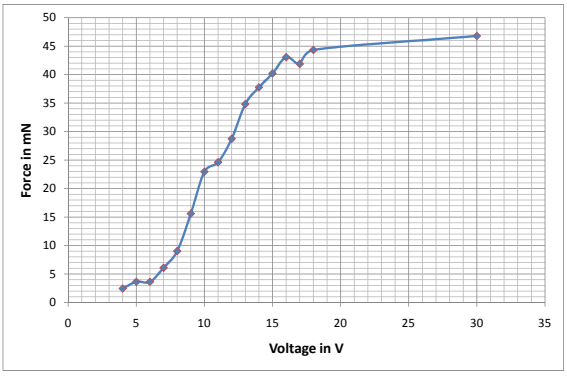


Figure 8: The magnetic force between the three EP magnets of the active motor unit side and the two neodymium magnets of the passive motor unit side using a specific magnetizing voltage and two shots.

3.2. Holding force vs. Voltage

We tested the holding force between two motor units. After a proper demagnetizing, changing the polarity of all magnets while decreasing slowly the switching voltage, we measured the force for different magnetizing voltages using each time two shots. We therefore put a motor unit below a second one that is attached to a bracket. The motor units are placed as if they were within aligned blocks. Then, we used measured balance weights, hanging them carefully on a special built anchor to the lower motor unit, until the connection broke. We made this several times to guarantee repeatable measurements of the tensile force.

The result can be seen in Figure 8. Below a voltage of 6V, the magnetic field is too weak to start a proper magnetization of the AlNiCo core. From 6V up to 16V, the holding force increases with the voltage. This is the linear magnetization region of the AlNiCo. Over 16V, the magnetization is saturated and the holding force reaches 45mN. This graph shows that voltages over 16V do increase the strength a lot. On the other hand, the higher the voltage is, the longer the charge of the capacitor will be. Charging to 18V instead of 15V takes 66% longer but increases the attraction force just by 10%. Thus, using voltages around 15V-16V is the best compromise.

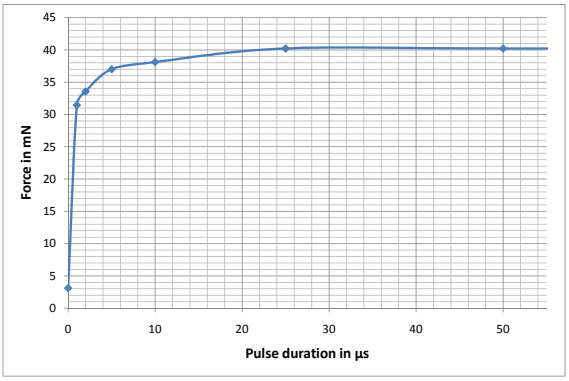


Figure 9: The magnetic force between the three EP magnets of the active motor unit side and the two neodymium magnets of the passive motor unit side using 15V and two shots with a specific shot time.

3.3. Holding force vs. pulse duration

The next test was to check how a shorter or longer shot influences the force. The magnets were therefore demagnetized as before and, then, magnetized with a shorter or a longer 15V shot from a 470μF capacitor. The result can be seen in Figure 9, showing the force function of the shot time.

We found out that it is useless to increase the shot time over 25μs, because while the coil is powered, the voltage of the capacitor is shrinking, causing a gradually lower current in the coil. So, after 25μs the current seems to be too low to polarize the magnet any further.

We also tested if multiple shots could increase the holding force. We found no hint of stronger force after ten or even hundred shots, because just a stronger magnetic field changes the remanence, the duration is irrelevant after a certain time (25μs).

3.4. Energy consumption

During a shot, the voltage drops around 2V. Knowing the capacitance, we can deduce that the magnets consumed an energy around 13mJ. That means that the 10mm move of a cube will consume around 78mJ.

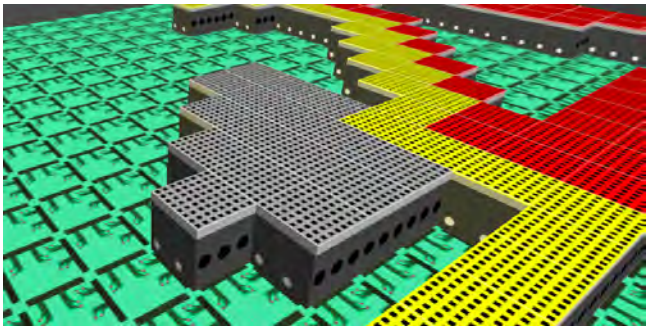


Figure 10: Captured image from the simulator software showing blocks with conveying actuators on the top. The floor is covered by tracks that guide the blocks during their displacement.

4. Blocks reconfiguration algorithm

The travelling system presented above allows considering the development of blocks composed of a processing unit, sensors and actuators. Each block is a small conveying area but, when linked with other blocks, it creates a conveying path for moving objects over larger distance. Side actuators allow a block to slide on another one and to form different conveying surfaces with different geometries. Furthermore, it reduces cost and avoids human interventions to replace a defective block. The second part of this paper presents a distributed algorithm which reconfigures a set of blocks, placing them in a new geometrical configuration.

Our algorithm allows reconfiguring a set of connected blocks considering that each block runs the same code and may exchange messages with 4 neighbors. This algorithm consists in a finite state machine, driven by the exchanges of messages in this network of connected blocks. This algorithm is implemented in our simulator software, named VisibleSim, that shows in real-time the exchanges of messages and the displacements of blocks (see Figure 10).

4.1. Operating principle of the blocks

The algorithm is based on exchanges of messages between neighboring blocks where each message may contain transmitted data. In order to write this algorithm for block reconfiguration, blocks are considered as entities with

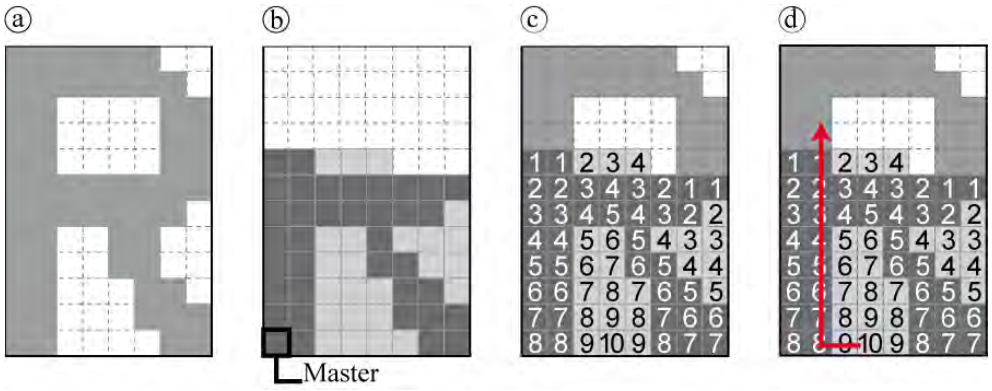


Figure 11: Some steps of the reconfiguration algorithm: a) The goal map. b) The initial configuration of the blocks. c) The distance from each block to the closest empty space. d) The path from one block to a free cell.

their own memory, able to execute a set of instructions. Each block is connected with 1 to 4 neighbors and communicates using asynchronous message exchange. The algorithm presented requires only data stored within each block, which can run independently in autonomous units of computation.
One of the blocks is chosen to propagate the actions of the reconfiguration algorithm to the other blocks. We call this block *Master Block*, it is connected to an external module that manages the reconfiguration.
The desired configuration is stored in a 4-way connected map, where the number of cells to fill corresponds exactly to the number of blocks that receive the map. Figure 11.a shows an example of a map where the cells to fill are colored in gray.

4.2. Steps of the algorithm

The algorithm for blocks reconfiguration is divided into four major stages. A preliminary step, executed only once, transmits the target configuration map to each block. After this preliminary step, three successive steps will be repeated:

1. Computing the distance between blocks and empty areas,
2. defining the direction of movement,
3. moving the blocks.

Each iteration is intended to make the current configuration getting closer to the target configuration.

4.2.1. Sending the map

The first step is to send the map to all the blocks (broadcast) and to receive an acknowledgment (convergecast). The Master Block sends the map to its neighbors. Then, when a block receives a map for the first time, it sends it to its other neighbors and waits for an answer from each of them. This stage is ending when the Master Block has received an answer from each of its neighbors, certifying that each block of the set has received the map.
The position of each block is relative to the Master Block. During the broadcast, each map message contains coordinates of the sender so that when a block receives a message, it deduces its position from the sender position.
In order to save memory, it is possible that the entire map will not be transmitted to every block. A solution is to decompose the map into several parts and distribute these sub-maps over the sets of blocks. In the current state of our algorithm, we are transmitting the entire map to each block, this optimization will be the subject of future enhancements.
The algorithm detailed below allows to start the computation on all blocks by broadcasting (without cycles) the start order. At the end of the computation, the initial block receives an acknowledgment stating that the algorithm has been performed on all blocks. The Figure 12 details this algorithm.
We recall that the same program is executed on every block. The program behaves like a finite state machine where the evolutions of states are triggered by receiving messages.

During the first step, the '*Master Block*' receives the first message from an external module.
For each type of information conveyed to the blocks, we define a dedicated message. For example, the message called *MAP_MESSAGE* is used to send data about the final map. It contains the following parameters:

- A pointer to the transmitter block (which is necessarily a neighbor of the receiver),
- the size of the map (width and height of this area),
- the bit array indicating whether each cell must be filled or not,

- the transmission time is added to the message in order to simulate the transfer delay.

When a block receives a message type *MAP_MESSAGE*, two actions are possible:

1. If there is already a map stored in the block, then it returns a message to the sender (*ACK_MAP_MESSAGE*);
2. Else, it copies the map into its memory and broadcasts a new message *MAP_MESSAGE* containing the map data to its neighbors (except to the one who sent him the*MAP_MESSAGE*). Then, it stores the sender id of the message (in a local variable named *sender*) and the number of neighbors to whom he sent this message (in the local variable named *waitedAnswers*).

When a block receives a message *ACK_MAP_MESSAGE*, it decrements the variable *waitedAnswers*. Then, if *waitedAnswers* is null, which indicates that all its neighbors have already answered, the block sends an acknowledgment to the '*sender*' with *ACK_MAP_MESSAGE*.

Figure 11.b shows the result of the map transmission to a set of blocks. Blocks that are already well placed are colored in dark gray and in light gray otherwise.

4.2.2. Calculating distance between blocks and empty areas

This phase consists in searching for each block, the shortest distance to a free position in the final configuration map, as shown in Figure 11.c. In this example, the value placed over each block (in a *myDistance* variable) is the distance from the block to the closest empty spot.
This algorithm is based on two steps, each of them needing to broadcast information to the set of blocks. The first step initializes all distances stored in the blocks to the infinity value. Then, during the second step, each block that has an empty place executes the following. Using the map, it checks if this spot should be empty at the end of the reconfiguration. In this case, it deduces that its distance value is equal to 1 ($myDistance = 1$) and broadcasts this information to its neighbors.
Every block must have a distance corresponding to the minimum distance from their neighbors plus 1. Then, to determine the distance value for each block when a neighbor updates its distance, it will propagate this information to its neighbors using a message containing its own distance.

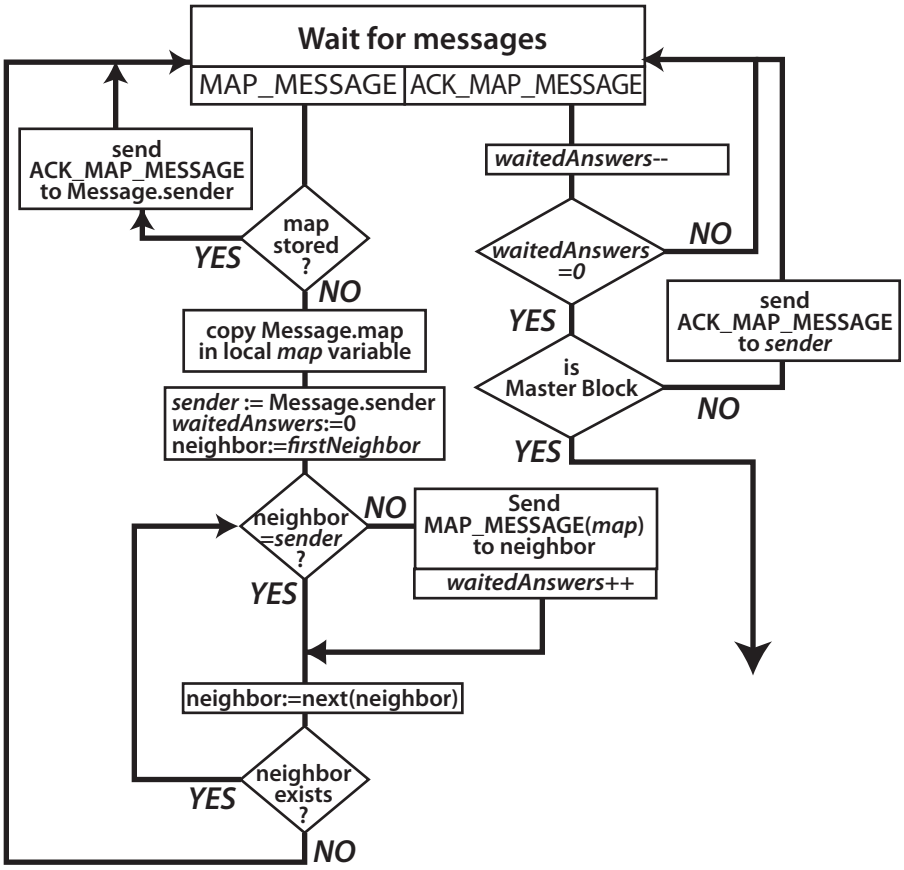


Figure 12: Algorithm of diffusion of the map over the set of blocks.

When a block receives a message from a neighbor, it compares its own distance value ($myDistance$) to the received one ($receivedDistance$), and, updates $myDistance$ if $myDistance > receivedDistance$.

4.2.3. Defining the direction of movement

B_{max} is the farthest block to an empty spot. It is identified easily as its distance is greater than all its neighbors distance.
Then, B_{max} has to find a path to a block, neighbor of an empty spot. This path is calculated by following the decreasing distance values as shown in Figure 11.d. The neighbor of B_{max} that admits the smallest distance is chosen. If distances have equal values, the order West, North, East, and South (WNES is here a convention) is followed. The relative position of this neighbor defines the movement direction for the block and it is stored in the *direction* variable.

The two previous steps can be repeated as it exists a path linking a free position and a not well-placed block. In order to avoid deadlocks, we set the distance stored in these moving blocks to the infinity value (they are marked with ∞ in Figure 13). Thus, each block can only participate to one reconfiguration at a time but many movements of blocks are possible simultaneously.

4.2.4. The motion of the blocks

The last step of the algorithm consists in starting the physical displacement of the blocks and waiting for them to reach their destination before beginning the next sequence. The whole movement is composed of an ordered list of one-block movements. When two neighbors have different movement directions, a priority has to be given to avoid collisions. The first block in the list must wait for the second one to end its movement.

4.3. The real-time simulator

We developed a software call VisibleSim [13] to visualize in real-time the states of the blocks during the reconfiguration algorithm. This software, developed in C++, allows observation of the asynchronous execution of the code on the different blocks.The small program associated with each block, called BlockCode, is written in C++ too. VisibleSim helps debug the program by changing the color of the blocks during the program or by writing debugging text, to name a few.

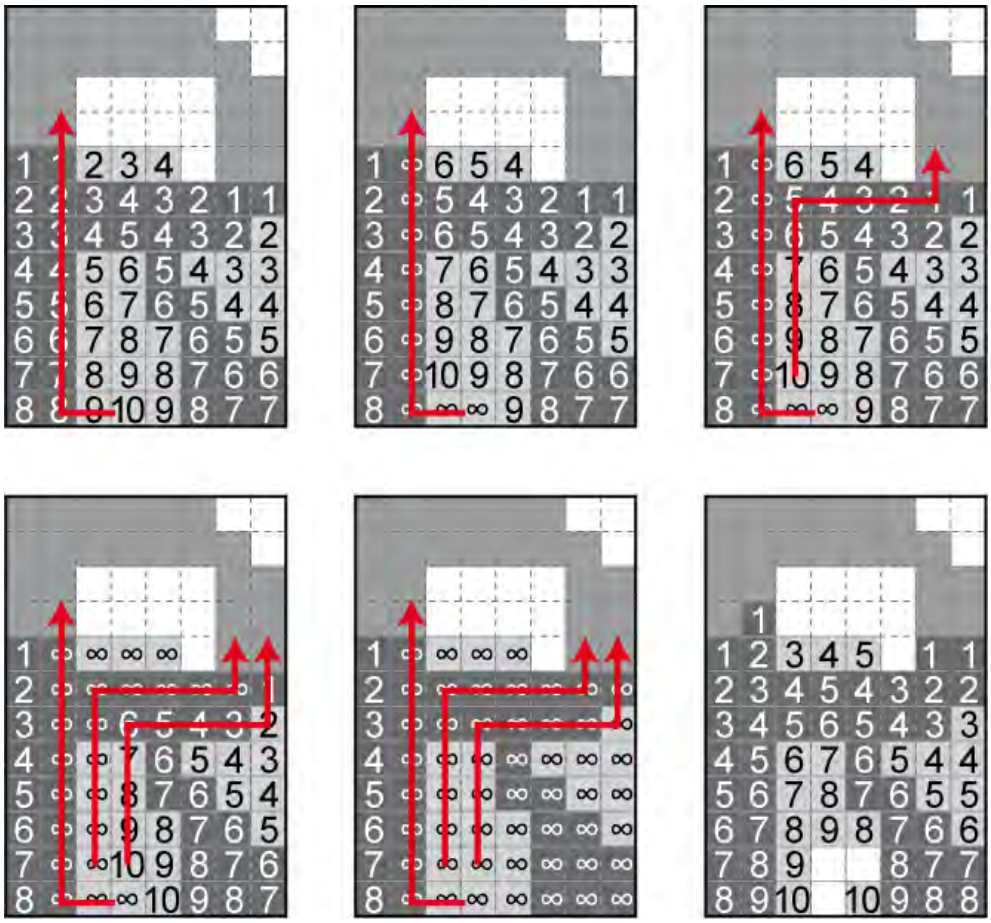


Figure 13: Algorithm for determining the direction of simultaneous displacements of many blocks.

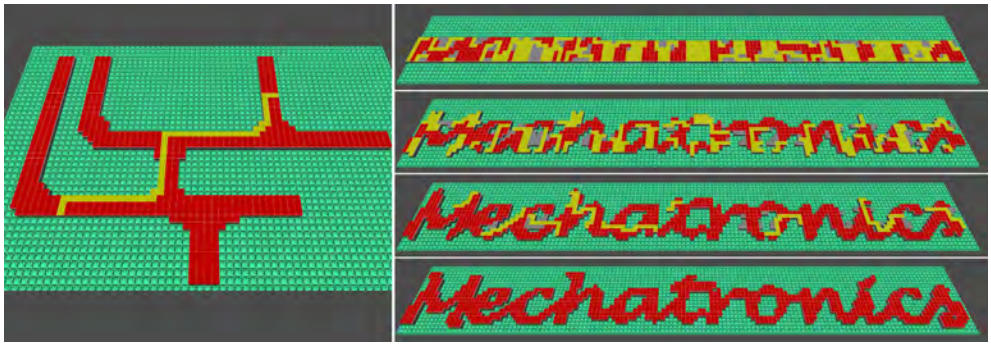


Figure 14: Some screen captured from the simulator interface during the calculation of the reconfiguration. On the left, the last step of reconfiguration of the 'conveyor' shape. On the right, four reconfiguration steps from the horizontal line (on the top) to the text 'Mechatronics' (on the bottom). Red blocks are already well placed, yellow blocks have just reached their final position, and grey blocks are not well placed.

The simulator calculates the simultaneous evolution of the blocks state taking into account the message transmission delay. Thus, the emission of a message and its reception cannot be achieved simultaneously.
After each movement, the positions of the blocks are displayed in real-time. Some screen captures are presented in Figure 14 and examples of reconfiguration can be visualized in a short video attached to this paper, or downloaded at this address : `http://smartblocks.univ-fcomte.fr/anim_blocks.mp4`.

4.4. Experiments on reconfiguration time

The total reconfiguration time mainly depends on the difference between the initial configuration of the blocks compared to the final map. Even more than the average distance traveled by each block, the complexity of the 4-connected path separating blocks is an important parameter of the reconfiguration complexity.
We performed two types of reconfigurations, a first one defining a map with a fairly simple pattern representing a large conveyor tree (composed of 546 blocks), and a more complex one formed by the text: 'Mechatronics' (875 blocks) as shown in Figure 14.

For each of these situations, several initial configurations were tested:

- Horizontal lines: Blocks organized as horizontal lines beginning on the

Shape Goal	Conveyor	Mechatronics
Horizontal line	121	22
2 horizontal lines	136	119
Diagonal line	108	42
Sinusoidal line	71	30
2 boxes	132	130

Table 1: Number of steps to reach the goal from different initial shapes.

bottom left end side and ending on the right end side.

- Two horizontal lines: Same as horizontal line except that lines are placed on the top and on the bottom.
- Diagonal lines: Diagonal lines starting on the upper left corner and ending on the lower right.
- Sinusoidal: a sinusoidal curve.
- Two vertical lines: Same as horizontal lines, except that the lines are verticals.

For each configuration, the numbers of steps to reach the goal is shown in Table 1. This value represents only the number of repetitions of the algorithm required for reconfiguration without taking into account the duration of displacement of the blocks.

One can notice that the speed of convergence of the algorithm varies significantly depending on the initial configuration. The algorithm seems to be more effective with the horizontal lines for the second case because it places blocks in a configuration close to their final distribution.

5. Conclusion

In this paper, we proposed a proof-of-concept of a self-reconfigurable robot based on sliding blocks. At the hardware level, several blocks have been realized integrating electro-permanent magnets in order to save energy between moves. The experiments show that a block is able to move along one another at an average speed of 16.4 mm/s and produces a holding force

of 45mN. The blocks are powered externally but the housing of all the electronic parts in a 10mm cube is possible. A future improvement could be to use the EP magnets to sense the "rotor" block position using the different inductance of the different positions.

If the speed and energy consumption of the system are adequate to built large robots with hundreds of blocks, the holding force should still be improved. The gliding side of blocks should also be carefully design to minimize the friction forces and to avoid that a block braces against other ones.

A block is moving by sliding on other blocks, which create an original way of moving. At the software level, we therefore need a special reconfiguration algorithm which would take into account this way of moving. We proposed a distributed algorithm which reconfigures a set of blocks according to a target map. All the blocks use the same program and are able to organize themselves by exchanging asynchronous messages. This reconfiguration algorithm has shown to be effective, i.e. reconfiguration of a set of blocks is working, but the number of simultaneous blocks moving can still be increased. Furthermore, the entire map is sent to each block, which is not necessary. An future work would be to reduce the target map to local positions.

In the future, this work will be used as a basis to realize the Smart Blocks project. We have therefore given suggestions, hints and solutions to problems that will be faced in later work, when the actuators will be integrated to form a modular and reconfigurable conveyor.

Acknowledgment

This work has been funded by the Labex ACTION program (contract ANR-11-LABX-01-01) and ANR/RGC (contracts ANR-12-IS02-0004-01 and 3-ZG1F) and ANR (contract ANR-2011-BS03-005)

References

[1] N. Chaillet, S. Régnier (Eds.), Microrobotics for Micromanipulation, John Wiley and Sons, 2010.

[2] The rules governing medicinal products in the European Union, Eudralex, 2010, Ch. Good manufacturing practice guidelines.

[3] R. Zeggari, R. Yahiaoui, J. Malapert, J.-F. Manceau, Design and fabrication of a new two-dimensional pneumatic micro-conveyor, Sensors & Actuators: A.Physical 164 (2010) 125–130.

[4] J. Malapert, S. Morishita, M. Ataka, H. Fujita, D. Collard, Y. Mita, Power-regulated thermal actuator based on uv-patterned polyimides for a ciliary motion system, IEEJ Trans. on Sensors and Micromachines 133 (2013) 77–84.

[5] K. Boutoustous, G. J. Laurent, E. Dedu, L. Matignon, J. Bourgeois, N. L. Fort-Piat, Distributed control architecture for smart surfaces, in: Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems, 2010, pp. 2018–2024.

[6] J. Bourgeois, S. Goldstein, Distributed intelligent mems: Progresses and perspectives, in: L. Kocarev (Ed.), ICT Innovations 2011, Vol. 150 of Advances in Intelligent and Soft Computing, Springer Berlin / Heidelberg, 2012, pp. 15–25.

[7] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, S. Murata, Distributed self-reconfiguration of M-TRAN III modular robotic system, Int. Journal of Robotics Research 27 (3-4) (2008) 373–?386.

[8] B. Salemi, M. Moll, W.-M. Shen, Superbot: A deployable, multi-functional, and modular self-reconfigurable robotic system, in: Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems, 2006, pp. 3636–3641.

[9] A. Spröwitz, S. Pouya, S. Bonardi, J. van den Kieboom, R. Möckel, A. Billard, P. Dillenbourg, A. Ijspeert, Roombots: Reconfigurable robots for adaptive furniture, IEEE Computational Intelligence Magazine, special issue on "Evolutionary and developmental approaches to robotics" 5 (3) (2010) 20–32.

[10] V. Zykov, E. Mytilinaios, M. Desnoyer, H. Lipson, Evolved and designed self-reproducing modular robotics, IEEE Transactions on robotics 23 (2) (2007) 308–319.

[11] J. Neubert, A. P. Cantwell, S. Constantin, M. Kalontarov, D. Erickson, H. Lipson, A robotic module for stochastic fluidic assembly of 3d self-reconfiguring structures, in: Proc. of the IEEE Int. Conf. on Robotics and Automation, 2010, pp. 2479–2484.

[12] K. Gilpin, A. Knaian, D. Rus, Robot pebbles: One centimeter modules for programmable matter through self-disassembly, in: Proc. of the IEEE Int. Conf. on Robotics and Automation, 2010, pp. 2485–2492.

[13] D. Dhoutaut, B. Piranda, J. Bourgeois, Efficient simulation of distributed sensing and control environment, in: IEEE International Conference on Internet of Things (iThings 2013), Beijing, China, 2013, pp. 1–8.