Building a First Prototype of a Multi-scale Modular Distributed Display

Frédéric Lassabe
FEMTO-ST Institute, CNRS
UTBM
Belfort, France
frederic.lassabe@femto-st.fr

Dominique Dhoutaut
FEMTO-ST Institute, CNRS
Univ. Franche-Comté
Montbéliard, France
dominique.dhoutaut@femto-st.fr

Benoît Piranda
FEMTO-ST Institute, CNRS
Univ. Franche-Comté
Montbéliard, France
benoit.piranda@femto-st.fr

Olga Kouchnarenko
FEMTO-ST Institute, CNRS
Univ. Franche-Comté
Besançon, France
olga.kouchnarenko@femto-st.fr

Julien Bourgeois
FEMTO-ST Institute, CNRS
Univ. Franche-Comté
Montbéliard, France
julien.bourgeois@femto-st.fr

***Abstract*—*Blinky Blocks* are cubic modular robots, which communicate with their neighbors through their faces, and change color using LEDs. We previously used sets of *Blinky Blocks* to display images on the basis of one *Blinky Block* being one pixel. In this paper, we build a multi-resolution screen with modular robots. This screen benefits from its distributed architecture: being able to work if some nodes fail, and being completely customizable. We propose a hardware architecture and related protocols to attach a 8×8 LED matrix to a *Blinky Block*. The architecture allows us to build distributed multi-resolution screens by mixing regular and enhanced *Blinky Blocks*. We demonstrate the usage of our integrated screen through experiments with continuously fed *Blinky Blocks* as well as an autonomous scrolling.**

I. Introduction

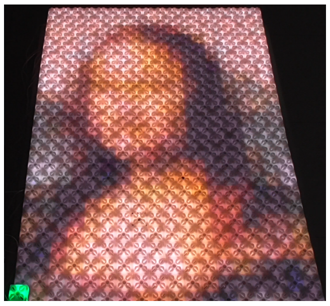
Personal computers and smartphones have become part of our life, and they are now joined by connected objects forming the internet of things (IoT). The integration of computing can be pushed one step further by integrating it into inert objects of everyday life using distributed intelligent micro-electro-mechanical systems (DiMEMS) [3], [2]. Each DiMEMS, called a module, would contain a processor, sensor(s), actuator(s) and a mean of communication with its neighbors and would fit in a very small individual volume of less than a cubic millimeter. Each module taken in isolation would not have a visible impact, but would prove useful when acting as a set in a coordinated way. Each object containing modules could then perform intelligent functions. With intelligent paint, we could imagine switches placed on demand on the walls, repositionable and reprogrammable, screens integrated directly into objects, biometric sensors, etc. It would also be possible to obtain expandable and segmentable electronics, or even cuttable, with the same functionality but with a different surface. We could imagine a computer working on a thin support, which could be cut or "glued".

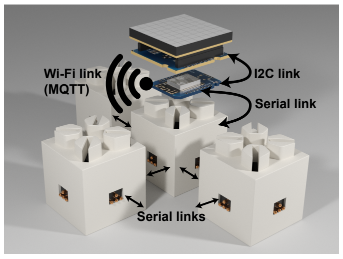
For example, one can imagine an object that looks like a tablet, running a single-player game. The tablet could be cut in half to form two smaller tablets, which now allow two players to continue the game together. Later, the modules can be grouped back to form a longer screen better suited to reading text. A number of obstacles need to be overcome before such a goal can be achieved. In this article, we tackle the multi-scale display based on a modular structure.

This paper presents a new complete hardware and software architecture for including LED matrices on a set of *Blinky Blocks*. *Blinky Blocks* are 4 cm-wide cubic shaped modular robots. They can change color, and were already successfully used to render pictures. We propose to add LED matrices to some *Blinky Blocks* to improve the display quality and provide a multi-resolution display mixing LED matrices on top of *Blinky Blocks*, along with simple *Blinky Blocks*. In [4], Dhoutaut et al. present a new hardware called *XBlock* that adds an ESP8266 board on *Blinky Blocks* to provide Wi-Fi communication capabilities to *Blinky Blocks*. In this work, we improve *XBlocks* with a screen connected to the ESP board. The newly created block is called *D-XBlock* for *eXtended Blinky Block with Display*. The hardware design of our solution is guided by the limited choice in the off-the-shelf components we could use, and their compatibility with *Blinky Block*'s form factor. Stemming from the hardware used and the possible applications, new problems arise in terms of data size and processing power. We propose algorithms and protocols to address these problems. First, we present related work in the field of modular and multi-resolution systems. Second, we describe our contributions: a hardware and a software architecture to integrate and to use the more precise LED matrices in a *Blinky Block* network. Third, we describe experiments with our system.

II. Related works

Early developments of objects linking physical and virtual worlds have occurred in the Tangible Bits project [7], with many revolutionary technologies and in particular, the


(a) Example of a low-resolution display made of a grid of 450 *Blinky Blocks*.



(b) Heterogeneous communication links: Serial links, dedicated I2C and Wi-Fi.

Fig. 1: Low resolution image and communication protocol.

metaDESK prototype including the activeLENS, an arm-mounted flat panel display interacting with a larger display. Some years later, the same authors presented their vision of Radical Atoms [6], which introduced Tangible User Interface, i.e., a user interface integrating digital information in physical space. At the same time appear the Sifteo mini-tablets [9], which were a follow-up of the Siftable project. Sifteo can exchange information through wireless communication and were intended to be used as a tangible gaming platform. The CubiMorph [12] has pushed further the idea of reconfigurable and interactive displays. If the Tilt Display explores the interactivity, CubiMorph, which is a chain-type modular reconfigurable robot, is able to change shape. In the direction of reconfigurable interactive displays, PickCell [5] is the most advanced project so far, which however ended without producing a prototype embedding all the functionalities. A modular smartphone project [13] envisioned a modular structure for a computing device, but did not provide a real hardware. Wixi [8] and the follow-up development [10] study the wireless infrastructure needed for a modular display. To sum up, all past projects have proposed visions or non-scalable prototypes. They mainly focus on the interaction with the users, whereas we aim to designing a scalable real device with a distributed operating system.

III. Contribution

This paper mainly deals with multi-level images and dedicated distributed protocols to display and animate them over potentially large ensembles of existing devices called *Blinky Blocks*.

A *Blinky Block* is a complete system driven by an ARM Cortex M0 microcontroller. It provides two RGB LEDs (both displaying the same color), a buzzer, a microphone, an accelerometer and 6 serial interfaces, one per side. Programs are stored in and run from the microcontroller embedded flash (128 kB available), with data stored into its RAM (32 kB). A 2D array of *Blinky Blocks* thus already can be used as a form of display (cf. Fig. 1a), where each *Blinky Block* corresponds to one pixel. Using basic *Blinky Blocks* to build a display meets several limitations: The image's definition is limited by the number of available blocks and its resolution is limited by their size.

Moreover, in previous work such as [11], a crude software approach was used. A simple protocol was attributing coordinates to each block relatively to a unique initiating block. Each *Blinky Block* was also keeping a copy of the whole image. The local color of a block was thus taken from the image depending on its own coordinates. As *Blinky Blocks* have a very limited memory (32 kB RAM) they cannot hold large images.

Integrating a display is not an easy task as a standard *Blinky Block* cannot manage a led matrix directly. An intermediary controller is required, along with dedicated communication protocols between these two new components. Adding a display means connecting the two systems (the *Blinky Block*'s board and the LED matrix) via hardware and software.

The contribution of this new work is three-fold:

1) Transforming some *Blinky Blocks* into *D-XBlocks* by adding a multi-pixels display covering their top face. The matrix is managed by a pair of MY9221 LED drivers coupled with a dedicated STM32F031 microcontroller.
2) Designing and implementing a communication protocol between the ESP8266 and the STM32F031 MCU (over an I2C bus).
3) Designing and implementing a communication protocol that allows for distributed multi-resolution images and animations of ensembles of blocks (using both serial and wireless communications).

Note that it is not required that all blocks from an ensemble are of the same variety. Typically, standard *Blinky Blocks* and *D-XBlocks* can be mixed. Thanks to our distributed algorithm, an image sent to the block array will be correctly displayed, the new *XBlocks* will use their screens to display a more detailed sub-section of the image corresponding to their position.

A. Hardware Addition

To better understand the extension of *Blinky Blocks* to *D-XBlocks*, we first present the regular *Blinky Block* architecture and system. We then describe the addition of ESP8266 micro-controllers and LED matrices to a regular *Blinky Block*.

In addition to a description at the begiing of Sect. III, let's mention that a *Blinky Block* is physically a cube with a case divided into two parts: the lower one, 1 cm high, and the upper one, 3 cm high, enclosing the *Blinky Block*'s PCB. Holes matching the PCB connector are located on each side, and duplicated on top and bottom to ensure that a connection exists with any rotation of *Blinky Blocks*. Connectors provide serial wires (receive and transmit), as well as GND and +5V power supply. Magnets on the sides and top/bottom faces provide attachment between *Blinky Blocks*.

We have designed a two-layers software architecture with 30 kB for a bootloader, and 96 kB free for user defined applications. The bootloader has a protocol to allow deploying applications on a set of *Blinky Blocks*, potentially hundreds, then run the application. An application is a full program, i.e., it performs a new initialization of the *Blinky Block* when started, and remaps its interrupts handler addresses onto the lowest addresses in RAM. Both bootloader and applications implement the same low level network protocol, based on UART communications. Although *Blinky Blocks* have an I2C controller (actually used by the embedded LED driver), it is not available from outside the PCB.

Applications, written in C, are compiled for the target architecture. Serial communications and an advanced bootloader are used to push applications to all blocks in a single manipulation, using a spanning tree protocol. Applications provide a user level network access with functions to send and receive messages. Such messages are encapsulated into packets, and then into frames that are transmitted over serial connections at 115 200 bauds. From an application point of view, external communications are expected from one of the 6 available UARTs.

In [4], the original *Blinky Blocks* were expanded into *XBlocks* by embedding a Wi-Fi capable ESP8266 microcontroller. The work in [4] has focused on enabling a hierarchical (or clustered) network architecture, that proved very beneficial to the management of large ensembles of blocks. Especially, it made it possible to address directly certain subsets of the network through wireless one-hop communications instead of traversing many nodes. The control of the whole system was balanced between local cluster-heads managing the neighboring blocks, and the central (computer) control having a high level view of the whole system. Technically, each *XBlocks* contains two independent microcontrollers, communicating over a serial (UART) bus.

In the present paper, we build upon the existing *Blinky Blocks* and *XBlocks*, and we add LED matrices and associated control hardware on top of the later. Initially, two options to add this LED matrix were considered by either adapting *Blinky Block* architecture to plug an I2C external device, or by inserting a device to translate from UART to I2C. To avoid modification to existing *Blinky Blocks*' PCBs, the second option was chosen. We rely on the ESP8266 microcontroller embedded into *XBlocks*, which communicate with its *Blinky Block* through UART, and with its LED matrix through I2C. While doing so, the LED matrix must always be oriented the same way relatively to its *Blinky Block*, to avoid rotating the image before displaying it.

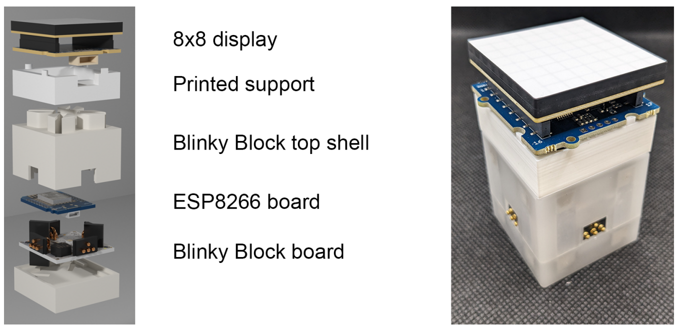


Fig. 2: Left: Hardware stack of components placed in the new *D-XBlock*. Right: the real assembled *D-XBlock*.

The left part of Fig. 2 shows the components stack inside the *D-XBlocks*. An ESP8266 board is placed inside the top shell to the original *Blinky Block*, along with a geometric adapter to fit the top screen layer. The screen part itself is made of two layers: one for the LEDs, and one for the control and communication components (pair of MY9221 drivers, STM32F031 microcontroller and voltage converter). The right part of the same figure illustrates the real assembled *D-XBlock*. Figure 1b shows a schema of the communications links, the ESP8266 in the middle serves both as a gateway between a *Blinky Block* and its LED matrix, and as a gateway to the outside world through its Wi-Fi link.

B. *Image Data Dissemination*

Prior to having 64 pixels over one *Blinky Block*, we never exceeded *Blinky Blocks* memory capability since we used images composed of up to 1,000 pixels (over a 1,000 blocks ensemble), or 3000 bytes. Thus, it was possible to hard-code the image in the program source code. However, introducing the 8×8 LED matrix multiplies the size of the image by a factor 64, widely exceeding the available memory. In this section, we propose a protocol to transfer all relevant data from a master, i.e. a computer connected to the ensemble through one or more *Blinky Blocks*. We start with a protocol already used in previous work that calculates the local coordinates of each block relative to a single reference. We then propagate the whole image over the whole block array. Using the coordinates as a filter, each block only keeps in its memory the data it is supposed to display and forwards the rest to the neighboring nodes.

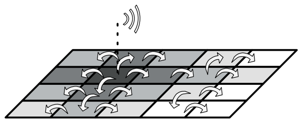
This phase requires addressing two issues: messages segmentation, since a detailed image exceeds the maximum packet size (228 bytes); and data dissemination with efficient bandwidth use between *Blinky Blocks*.

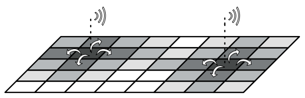
a) Messages segmentation: Messages segmentation relies on a new protocol, which includes a header to indicate and order segmented messages. We preferred an application level segmenting over modifying the current network layer, as the later is common with the bootloader and shall remain compatible. The header contains a packet type identifier, a sequence number (to prevent backward flow, as seen later) and the intended coordinates for this image fragment.

b) Basic dissemination: As an initial step, we segment the large image and inject all fragments from a single entry point into the ensemble. This entry point can either be a block connected to a computer through a serial interface (as used to reprogram the blocks), or be a *D-XBlock* featuring a Wi-Fi link. Figure 3a shows a diagram of such a dissemination, where the fragments propagate from the Wi-Fi entry point.

In the most simple setup, fragments are simply flooded over all available interfaces and backward flow, as well as loops, are prevented through the use of a last-seen sequence number that won't be re-transmitted more than once.

c) Balanced routing: The firmware we developed for the *Blinky Blocks* already includes two basic spanning tree algorithms we considered for this task. One is "first in,


(a) Propagation from one Wi-Fi entry point


(b) Propagation from multiple Wi-Fi entry points

first served" while the other is "breadth-first". Both kinds of spanning tree could be used to transmit the image data to all *Blinky Blocks*. The simplest way to use those spanning tree would be to flood all the data through all branches, nodes keeping only the part of the data corresponding to their own address and forwarding the rest. This would however be very inefficient as *Blinky Blocks* would receive fragments that have no relevance in their spanning tree branch.
A second way would be to collect the structure of the spanning tree back on the control computer to perform *source routing* (i.e. each fragment contains the list of nodes it has to traverse). Unfortunately, this solution can significantly increase the size of the data packets, possibly over the 228 bytes limit of our network stack.

Instead, and considering the inherently static nature of our ensemble of blocks, we propose two other solutions that do not depend on the spanning tree: a coordinates-based routing (*geocasting*), and a table based routing, with their own advantages and drawbacks.

Geocasting consists in forwarding a fragment only in the direction of its intended destination. Moreover, in the context of this work, we can take advantage of multiple Wi-Fi enabled blocks to inject fragments *in parallel* from multiple entry points, as illustrated in Fig. 3b. Our geocasting algorithm acts as follow: upon reception, the destination coordinates of the fragment are compared to the coordinates of the receiving *Blinky Block*. If they match, the data is kept and no further transmission occur. If they do not, the block will select an outgoing interface that will minimize the distance to the final destination and forward the fragment. This solution is simple and efficient, but can fail in the case of non-convex sets of blocks.

The second solution involves a routing table in each block and an initial discovery phase to populate those tables. The corresponding distributed algorithm (Algorithm 1) works under the hypothesises that a path exists between any pair of *Blinky Blocks*, and that every *Blinky Block* has a unique identifier between 1 and N, the total number of *Blinky Blocks*. It guarantees that each fragment reaches its destination along one of the shortest paths, and that each fragment is not unnecessarily duplicated during transmission. The main drawbacks of this solution are the initial time required to build the routing table, and the potential size of this table in large sets of blocks. Each router maintains a routing table with $N-1$ entries. Matching outbound interface is determined by an algorithm similar to a simplified RIP[1], without considering distances. The algorithm ends when the routing table is fully populated, i.e., the routing table converged.

Algorithm 1: Routing algorithm.

```
Data: $routingTable[N]$: array of destination connectors
Data: $nbRoutingFixed$: termination detector (if = N-1)
1  Function StartUp():
2      foreach $cell \in routingTable$ do
3          $cell \leftarrow \emptyset$ ;
4      foreach connected port do
5          sendP2P(ROUTING,$port, \{id \leftarrow myId\}$) ;
6      $nbRoutingFixed \leftarrow 0$;
7  Msg Handler RoutingMsg(msg, sender):
8      if $routingTable[msg.id] = \emptyset$ then
9          $routingTable[msg.id] \leftarrow sender$ ;
10         foreach connected port $\neq$ sender do
11             sendP2P(ROUTING,$port, \{id \leftarrow msg.id\}$) ;
12         $nbRoutingFixed \leftarrow nbRoutingFixed + 1$ ;
13         if $nbRoutingFixed = N - 1$ then
               // Routing process finished
```

Using several entry points to transmit image data to the *Blinky Blocks* set relies on geocasting, with a segmentation of the data. It is also based on the target location related to the entry points' locations: each entry point transmits image data to the targets it is closest to, compared to its peers. Notice that the entry points have a global knowledge of each other based on exchange of Wi-Fi messages.

For reliability and compatibility reasons, and also for ease of development, we decided to use the MQTT protocol to convey messages between the control computer and the Wi-Fi entry points. MQTT is explicitly tailored for the IoT field to which our *D-XBlocks* belongs to. It is built upon TCP/IP, and as such provides integrity and sequencing assurances. Moreover, MQTT enables a seamless integration on the control computer, where we use a web page and Javascript code to provide the user with a visual interface. A screenshot of this control web page is shown in Fig. 4, at the bottom. Last but not least, using JSON encoded messages over MQTT allows for versatility and extensibility. New functionalities or even debug messages could be added very easily.

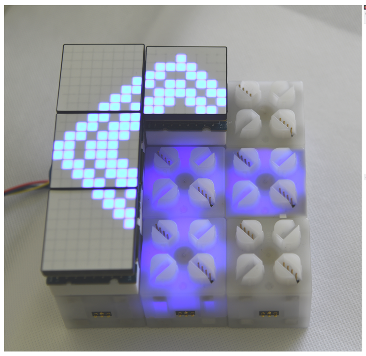
For displaying the image fragment on each block, the locally available hardware is used. If the block is of the *D-XBlock* flavor, then all 64 pixels will be sent to the LED matrix. Otherwise, the block computes the average color and displays it by means of its internal LEDs. This multi-resolution approach is illustrated in Fig. 4, where only 4 blocks are equipped with the LED matrix, and the others compute and display the mean color from the 64 pixels assigned to them.

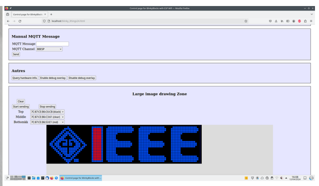
C. Distributed Scrolling Animation

In this section, we propose a distributed protocol that makes use of the autonomous nature of the *Blinky Blocks* to scroll the image without requiring much communication with the control computer. To illustrate the distributed nature of our *Blinky Block* ensemble, we propose multiple versions of a scrolling algorithm. The main design goal of those algorithms is to exhibit a very good scalability by distributing the processing and communication burdens as evenly as possible over the set of blocks. As explained before, the memory and communications capabilities of the *Blinky Blocks* are indeed very limited. Moreover, as in a true distributed system, these algorithms are thought as little reliant on the control computer as possible. This means that all decisions (what to display, what to send) have to be made locally, with the control computer only merely choosing parameters and initiating the process.

For the three scrolling algorithms, we discuss here their respective design, leaving the implementation details and experimental results for Section IV.

- In the **rollback** version, after the initial dissemination of the image, the role of the control computer is limited to starting, stopping or changing the scrolling speed. Each block forwards its last column of pixels to the next neighbor. Upon reaching the end of the line, image fragments are transmitted back.
- In the **injection** version, we consider an image larger than the ensemble of blocks. The full image is stored on the control computer and columns of pixels are sent in sequence to the first block of each line. The blocks send their current last column to the next block in the line. Fragments reaching the end of the line are simply dropped. The computer is in charge of feeding the blocks with their pixels, and manages the scrolling by selecting the right column in the image. With the injection, the memory required in each block is constant, whatever the size of the image.
- In the **hopping** version, we consider the same large image as with the injection version, but with additional *XBlocks* that allow for physical gaps in the ensemble thanks to Wi-Fi links. The important distributed aspect here is that decision to forward data through wireless communication is made completely locally. Each block keeps information about its direct neighbors, and uses its Wi-Fi link as backup connection. This makes such a system very resilient and able to support dynamic partitioning.





MQTT Channel BBESP

Top 7C:87:CE:B6:C6:CB (black)

Middle 7C:87:CE:B6:C3:61 (clear)

Bottomkk 7C:87:CE:B6:53:E1 (red)

Fig. 4: The setup made of 4 *D-XBlocks* and 5 *Blinky Blocks* building a part of our multi-resolution display. The bottom image is a screenshot of the web control interface on the Wi-Fi-connected master computer.

IV. Experiments

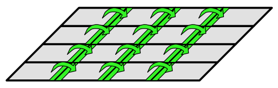
This section describes the implementation and provides illustrations on the three scrolling algorithms proposed. Let us emphasize the fact that these algorithms have been implemented and tested on real hardware, as displayed on pictures and video (https://youtu.be/V8RAsPjLSrs).

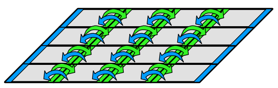
a) Rollback scrolling: This version starts once an image is disseminated over the whole ensemble of blocks, using the previously described algorithm. The image is regularly shifted to the right, while pixels ”leaving” on the rightmost part come back on the left part of the ensemble. The point of this algorithm is to make use, as most as possible, of the distributed nature of the *Blinky Blocks*. Consequently, it can be described with those rules:

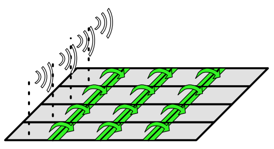
- The control computer can sends `START_SCROLL` or a `STOP_SCROLL` messages that will be disseminated through the ensemble. The `START_SCROLL` message contains the intended scroll direction along with a delay (in ms) to control the speed of the animation.
- Upon reception of the `START_SCROLL` message, leftmost blocks of each line will start the animation. It sends its rightmost column of pixels (`SCROLL_FRAGMENT` message) to its neighbor on the right, while also shifting its displayed pixels to the right. The leftmost column of pixels is kept as is for now. This `SCROLL_FRAGMENT` message also contains a travel direction, which is for now equal to the animation’s direction. This behavior is represented in Fig. 5a using green data and messages.
- Upon reception of a `SCROLL_FRAGMENT` where the travel direction is equal to the direction of the animation, the block prepares a `SCROLL_FRAGMENT` message with its rightmost column of pixels then shifts its display to the right. If a neighbor is present on its right, the `SCROLL_FRAGMENT` will be sent to it, with the travel field set as the direction of the animation. With no neighbor on the right, the fragment will be sent in the opposite direction (and the travel field will be set accordingly).
- when receiving a `SCROLL_FRAGMENT` whose direction is opposite to the animation’s direction, the message will simply be forwarded on the left if possible. If not, the current block is the leftmost one on its line and the pixel column is displayed on the left part of the screen. This behavior is represented on Fig. 5b using blue data and messages.

b) Injection scrolling: The second algorithm considers a much larger image, which does not fit on the combined displays of the blocks. The whole image is only stored on the control computer, and sent progressively to the leftmost blocks of each line of the set. Here, the role of the control computer is limited to ”injecting” new column of pixels, and the animation will be handled by distributed communications between the blocks. This algorithm is illustrated in Figure 5c.

- The control computer can send `INJECTED_FRAGMENT` messages to the blocks on the leftmost part of the ensemble. For demonstration purpose, those blocks are currently targeted using their Wi-Fi MAC addresses.An


(a) Basic distributed scrolling


(b) Scrolling with rollback


(c) Scrolling with injection

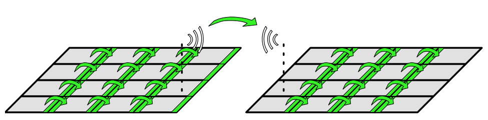

(d) Hopping scroll

Fig. 5: Various types of scrolling

INJECTED_FRAGMENT message contains a column of 8 pixels.

- Upon reception of such message, the receiving block will:
  - Only when there is a neighbor on the right: build a new INJECTED_FRAGMENT with its rightmost column of pixels, and send it to its neighbor on the right.
  - Shift its display to the right.
  - Display the data received on the left part of its screen.

Note that in this video, some minor glitches persist in the display as our implementation is still experimental (a few data packets are lost when sent to the matrix controller).

c) Hopping scrolling: This version (cf. Fig. 5d) is very similar to Injection, except that it allows the user to dynamically separate the ensemble in multiple subsets of blocks. The control computer is still used to incrementally send the image through leftmost blocks. Blocks still distributively scroll the image to the right. The algorithm however must start on an initially connected ensemble and blocks must memorise their neighbors' Wi-Fi MAC addresses. If the ensemble is later split in multiple subsets, blocks will continue to send data to their "ex-neighbors" by using their Wi-Fi links.

V. Conclusion and future work

This paper presented a new complete architecture (hardware and software) to add more detailed display capabilities to the existing *Blinky Blocks*. The proposed 8×8 LED matrices can be plugged on top of some or all blocks in a set. The architecture includes an embedded Wi-Fi capable microcontroller, which we also leverage by designing data dissemination and data migration protocols. We demonstrate through real world implementations that the proposed architecture supports multi-resolutions display, single and multi-points data injection and fully distributed data movement. Our system is thus easily scalable, even though the processing and communications capabilities of individual elements are quite limited. The addition of those display capabilities demonstrates a concrete use case for modular robots and the benefits of well-distributed algorithms. In the meantime, it also gives us a very useful tracing and debugging tool to help us further enhance our *Blinky Blocks*.

Future work include advanced routing protocols able to cope with non convex and even non-adjacent sets of blocks, along with new actuating capabilities that could be plugged on top of the *Blinky Block*. For instance, we intend to instrument the programming blocks (hollow blocks used for interfacing with a computer as well as to power *Blinky Blocks* ensembles) with an ESP8266 microcontroller and a current sensor to enable runtime power consumption optimization and adaptation. Actuators could also include motors and movable parts to interact further with the real world.

Acknowledgment: This work was partially supported by the ANR (ANR-23-CE25-0004) and the EIPHI Graduate School (ANR-17-EURE-0002).

References

[1] Routing Information Protocol. RFC 1058, June 1988.

[2] J. Bourgeois, J. Cao, M. Raynal, D. Dhoutaut, B. Piranda, E. Dedu, A. Mostefaoui, and H. Mabed. Coordination and computation in distributed intelligent mems. In *27th International Conference on Advanced Information Networking and Applications (AINA 2013)*, pages 118 – 123, Barcelona, Spain, mar 2013.

[3] J. Bourgeois and S. Goldstein. Distributed Intelligent MEMS: Progresses and Perspectives. In Ljupco Kocarev, editor, *ICT Innovations 2011*, volume 150 of *Advances in Intelligent and Soft Computing*, pages 15–25. Springer Berlin / Heidelberg, 2012.

[4] D. Dhoutaut, B. Piranda, and J. Bourgeois. Multi-networks communications in large set of modular robots. In *Proceedings of the 1st International Conference on Smart Medical, IoT & Artificial Intelligence - ICSMAI'24*, Saida, Morocco, 2024.

[5] A. Goguey, C. Steer, A. Lucero, L. Nigay, D. R. Sahoo, C. Coutrix, A. Roudaut, S. Subramanian, Y. Tokuda, T. Neate, J. Pearson, S. Robinson, and M. Jones. PickCells: A Physically Reconfigurable Cell-composed Touchscreen. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–14, Glasgow Scotland Uk, May 2019. ACM.

[6] H. Ishii, D. Lakatos, L. Bonanni, and J.-B. Labrune. Radical atoms: beyond tangible bits, toward transformable materials. *Interactions*, 19(1):38–51, January 2012.

[7] H. Ishii and B. Ullmer. Tangible bits: towards seamless interfaces between people, bits and atoms. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, pages 234–241, Atlanta Georgia USA, March 1997. ACM.

[8] J. Kadomoto, H. Irie, and S. Sakai. Wixi: An inter-chip wireless bus interface for shape-changeable chiplet-based computers. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*, pages 100–108. IEEE, 2019.

[9] D. Merrill, E. Sun, and J. Kalanithi. Sifteo cubes. In *CHI '12 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '12, pages 1015–1018, New York, NY, USA, May 2012. Association for Computing Machinery.

[10] S. Nagasaki, J. Kadomoto, H. Irie, and S. Sakai. Dynamically reconfigurable network protocol for shape-changeable computer system. *IEEE Design & Test*, 40(6):18–29, 2023.

[11] B. Piranda, F. Lassabe, and J. Bourgeois. Disco: A multiagent 3d coordinate system for lattice based modular self-reconfigurable robots. In *IEEE International Conference on Robotics and Automation (ICRA 2023)*, London, England, may 2023.

[12] A. Roudaut, D. Krusteva, M. McCoy, A. Karnik, K Ramani, and S. Subramanian. Cubimorph: Designing modular interactive devices. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3339–3345, May 2016.

[13] T. Seyed, X.-D. Yang, and D. Vogel. A modular smartphone for lending. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, page 205–215, New York, NY, USA, 2017. Association for Computing Machinery.