Multi-networks communications in large set of modular robots.

Dominique Dhoutaut[1,2][0000−0002−1135−5929],
Benoît Piranda[1,2][0000−0003−2149−871X], and
Julien Bourgeois[1,2][0000−0002−0686−2643]

[1] Université de Franche-Comté, Montbéliard, France
[2] FEMTO-ST institute, department of computer science and complex systems (UMR CNRS 6174)
{firstname.name}@univ-fcomte.fr

Abstract. In this paper, we propose an extension to the *Blinky Block* robot (named *XBlock*) by adding wireless communication capabilities. This enables the creation of modular robots with multiple network layers. We create a set of heterogeneous modules where two types of robots coexist: classic *Blinky Blocks* and *XBlocks*. We propose a distributed algorithm for building clusters from *XBlocks*, which can then communicate wirelessly with each others and with a master computer while handling the control of their local cluster through wired, multi-hops, point-to-point communications.
We'll show that the use of multiple networks on modular robots brings gains in terms of lattency, jitter and packet losses, communication quality and security. It also allows for a clever balancing of control, significantly easing the job of the main controller.
We propose several experiments (available in video) with a set of 168 connected modules running the same program. From a web interface on the master computer, we order all *XBlocks* to locally recruit members into their own clusters. Those cluster-heads then sends a feedback to the master controller in the form of the shape of their recruited members, which is displayed.

Keywords: Modular robots · Multi-Networks · Clustering.

1 Introduction

Modular and reconfigurable robots are very active research fields, where people develop versatile systems interacting with the real world. A main aspect of those systems is the ability to provide advanced capabilities through the collective work of relatively simpler elements. Each element may not be able to do much by itself, but an ensemble of those is able to exhibit a much more useful global or emerging behavior.
Albeit more complex to manage, modular and reconfigurable robots have the following key advantages: Elements are way easier to replace (replacement of defective elements can sometime be fully automated). All modules are identical and

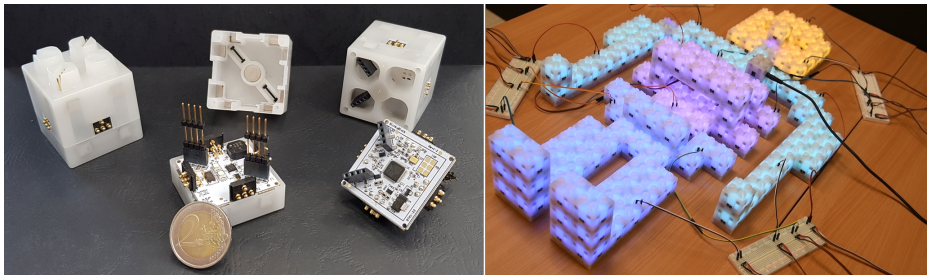

Fig. 1: *Blinky Block* hardware and the setup for our experiment.

interchangeable ; They are easier to mass-produce, and cost effective when produced at scale and depending on the application, elements may provide various functions that can be swapped by simply swapping elements.

A wide range of reconfigurable robots have been developed over the years and this paper is focused on *Blinky Blocks* (see Fig. 1). *Blinky Blocks* are Lego-like devices that can be manually assembled to form any desired shape. They have embedded (yet limited) computation capabilities and communicate through serial interfaces on all of their six faces (3D communication). They are usually controlled in a completely distributed manner, each block exchanging informations with its direct neigbors.
Albeit fully functional, *Blinky Blocks* prove to be difficult to manage when used in large, non-homogeneous structures. The limited onboard memory makes it difficult to fit a complex program that can handle the whole range of situations that can be encountered in the ensemble. To alleviate those limitations, control fully centralized from a computer is also possible. But it does not scale well due to inherent existence of communications bottlenecks, especially the closer ones get to the gateway to the control computer.

In this paper, we add additional wireless communications capabilities to selected *Blinky Blocks* in the ensemble (which we call eXtended Blocks - *XBlocks*- in the rest of the document). The core contribution concerns the hierarchical architecture we propose alongside the wired and wireless communication mediums. By using both and by taking into account the physical shape of the ensemble we can propose a balanced ratio between distributed and centralized control. Relying on locally distributed algorithm allows us to greatly simplify the external control required to manage the ensemble and thus scale very well. Thanks to its hierarchical and functional approach, it also make the external control much more intuitive.

2 Reconfigurable robots and *Blinky Blocks*

Numerous modular robots are presented in the literature, with a wide range of capacities and applications. Each robot design is driven by choices in terms of sensor and actuator capabilities, communications, power supply modes, and mo-

tion capability. *Kilobots* proposed by Rubenstein et al [6] are small autonomous systems, standing on three vibrating legs that enable them to move around. They communicate with their close neighbors by infrared connection to discover their environment and organize their movements to form predefined shapes.
Romanishin et al. developped *M-Blocks* [5] that are 5 cm large cubic autonomous robots with complex motion capabilities. *M-Blocks* communicates using low-energy Bluetooth Smart for centralized control and ANT protocol for module-to-module communication, moreover detection of the presence of neighbors is achieved by reading a barcode on the neighbor face.
For *RoomBots* developped by Spröwitz et al. [7], communication is transmitted within the modules through slip rings, wearing a physical half-duplex RS485 communication bus.

The work presented in this paper is based on a specific type of reconfigurable robots called "*Blinky Block*" [3]. *Blinky Blocks* are already existing (cf. Fig. 1), well-tested lego-like cubic devices that can be manually assembled in any desired shape. Each provides integrated processing and communication capabilities, memory, and means to interact with the real world like a buzzer, sound sensor, tap sensor and lights. A set of *Blinky Blocks* can work in a standalone mode (only power being externally provided). Optionally however, one off the *Blinky Block* from the set can be connected to an external controller (a computer) from where an operator can give orders that can be broadcasted to all blocks.

From an hardware perspective, each *Blinky Block* incorporate a small - energy efficient - ARM Cortex M0 core and 64KB of RAM. Each block also possess 6 independent serial UARTs (one per face) featuring four pins (+5V, ground, RX, TX). To provide power, upload software into the blocks, and generally communicate with them, we use specific "stub" blocks featuring a power supply connector and a mini-jack serial connector.

Power is provided by a 5V power supply to one block, and then shared with the neighboring ones. In case of large ensembles (tens or more blocks), multiple power supplies can be used to distribute power more evenly and prevent overheat of the connectors from the aggregated load.

Modular robots, like *Blinky Blocks*, are managed by distributed programs, written according to the principles of distributed algorithms [4]. Since each module has only partial knowledge of its environment, it relies on message exchanging to obtains more information about the whole system. Collection of larger scale information then enables computations concerning the whole system. For example Naz et al. [2] propose a distributed algorithm allowing *Blinky Blocks* in a set to identify the one at the center (i.e. a global information).

Albeit fully functional, the inherent distributed control of *Blinky Blocks* makes them difficult to manage when used in large, non-homogeneous structures. Their limited memory hardly fits versatile and complex software required in complex scenarios. At scale, their limited memory also hinder the ability to store information concerning the state of the whole system. And lastly, as serial point-to-point communications are relatively slow (115200 bauds), they also

have an adverse impact on the ability to propagate informations to the whole network.

To engage in large scale scenarios, we make use of our dedicated simulation software *VisibleSim* [8], where the cost and the complexity of deploying thousands of blocks at once is not a concern. In the rest of the paper, presented results thus come from real experiments, simulations or both.

3 Clustering through wireless communications

The aforementioned limitations make scaling to hundred or more blocks quite difficult if the scenario is complex and differentiated behaviors are required from the blocks.

The use of wireless communication modules makes perfect sense if we can organize our large set of robots into clusters [1], i.e. subsets of modules that group together the *Blinky Blocks* in the same area. Each cluster has its own leader (a *XBlock* that plays the role of cluster head) equipped with wireless communication. Wireless links not only allow cluster heads to communicate directly with other cluster heads but also with the master computer. The complete architecture is presented on Fig. 2. Functionally, the *XBlocks* are composed of two independent parts, running on two independent micro-controllers. These micro-controllers communicate through a serial interface similar to the one present on the faces of all blocks. Physically, both micro-controllers are embedded in the same casing, making it indiscernable from normal blocks.

For the external communications, we decided for a MQTT over WiFi architecture. A first computer is running a MQTT broker. A second one hosts a graphical (web) client publishing commands and subscribing to feedback through an MQTT-over-websocket link. This MQTT solution allows for great flexibility, scaling and ease of deployment. Having this multi-layered architecture brings

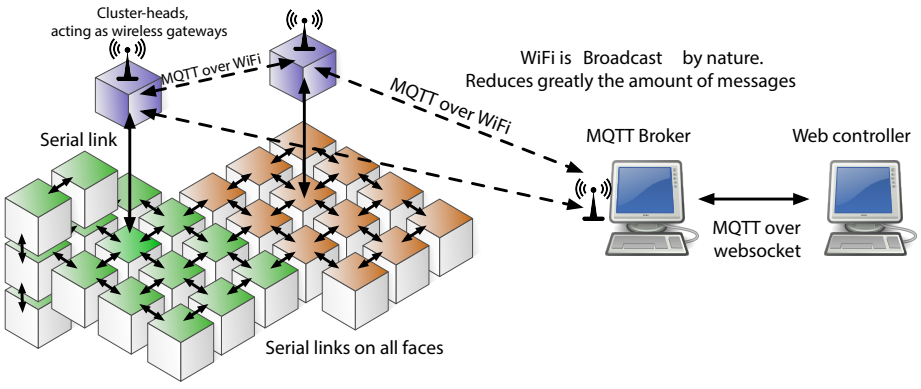

Fig. 2: Clustered architecture and communication links

multiple benefits:

*A **latency benefit**:* The time complexity of an algorithm for broadcasting a message to the set of modules connected point-to-point in a grid is linear as a function of the diameter of the set [4]. The use of clusters drastically reduces the time required to broadcast information to all the modules, as this total time is proportional to the largest diameter among the clusters.

*A **jitter and packet losses** benefit:* In many communications patterns, most of the traffic comes from or goes to gateways to the outside world. For example in data collection applications, each modules has to send its sensed data back to the control computer. If there is only one gateway, to closer the links are to the gateway, the heavier the traffic. Heavy traffic also means network congestion and an increase of jitter. In the case of our *Blinky Blocks*, as their internal buffers are so small, this also often translates into packet losses. Fig. 4 illustrates this problem on the left side, where only one gateway is present in the set of *Blinky Blocks*. Traffic coming from all modules converge towards the gateway and links are increasingly overloaded the closer it gets. On the right side of Fig.4, two additional *XBlocks* allows for a much more balanced collection traffic, which translates into less losses and a lower jitter.

*Improves the **quality of communications**:* Point-to-point communication between *Blinky Blocks* relies on connectors (combining magnets and springs) whose reliability is not perfect. The addition of a wireless network above the point-to-point network makes it possible to offer other communication links to make up for faulty links.

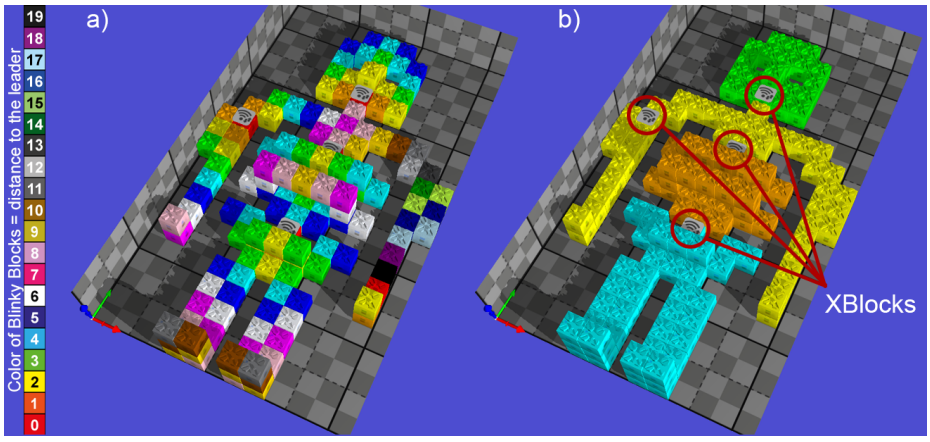

Fig. 3: Clustering recruitment process: The 4 *XBlocks* (modules with a wireless logo on top) recruit modules for their cluster. The colors in a) represents the distance to the Cluster Head, in b) it indicates the cluster a modules belongs to.

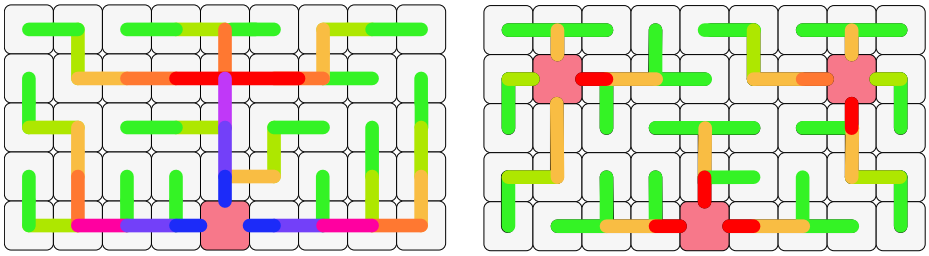

Fig. 4: One gateway leading to overloaded links (left). Three gateways and better network load balancing (right).

4 Our multi-network solution

Heterogeneous set of Blinky Blocks. In principle, our modules must all run the same program, which is broadcasted during the programming phase from a module connected to the development machine. To run our distributed program on the heterogeneous system formed by *Blinky Blocks* and *XBlocks*, we consider two agents present on all modules: the *Application Agent* and the *Leader Agent*. When it receives an order from the host machine, the *Leader Agent* sends a recruitment message to all its neighbors (except the banned one). When an *Application Agent* receives a recruitment message for the first time, it memorizes the sender as its parent, then sends it its position and forwards the recruitment message to all its neighbors except the sender. When it receives a position message, it forwards it to its parent. While the *Leader Agent* does not broadcast a recruitment message from another cluster, and it transfers the positions received to the host machine. Algorithms 1 and 2 respectively detail this method. In these pseudo-code, we consider the following functions:

- setColor(*color*) changes the current color of the module.
- sendWIFI(*dest*,*data*) sends the data to *dest* using wireless communication.
- sendP2P($MsgName, port, data$) sends the message $msgName$ to the connected neighbor (on *port*) embedding *data*.
- posNeighbor(pos, p) is a function that compute the position of the module connected to port p according to the local position pos.

Some *Blinky Blocks* are equipped with communications modules (we call them *XBlock*), but *Blinky Blocks* and *XBlocks* are externally identical and can take advantage of all their connectors to be linked to a neighboring *Blinky Block*. It is possible to check whether the chip is present in the program.

Placing some *XBlocks* in a set of *Blinky Blocks* we create an heterogeneous set of robots. But the program integrates two agents one for classical *Blinky Blocks* and one for *XBlocks* and only one of these two agents is activated at start.

Algorithm 1: Leader Agent Algorithm.

```
1  Function StartUp():
2      myParent ← ∅ ;
3      myClusterId ← ∅ ;
4      setColor(BLUE); ;
5  Msg Handler WIFI_InitCusterMsg({id, color, banned}):
6      myClusterId ← id;
7      sendWIFI(master, {0, 0, 0});
8      foreach connected port do
9          if port ≠ banned then
10             sendP2P(RECRUIT,port, {id, color,posNeighbor({0, 0, 0}, port)});
11 Msg Handler RecruiteMsg({id, color, position}, sender):
12     if myParent = ∅ then
13         myParent ← sender;
14         myClusterId ← id;
15         sendWIFI(master, position);
16         foreach connected port do
17             if port ≠ sender then
18                 sendP2P(RECRUIT,port, {id, color,posNeighbor(position, port)});
19 Msg Handler PositionMsg({position}, sender):
20     sendWIFI(master,position);
```

Algorithm 2: Application Agent Algorithm.

```
1  Function StartUp():
2      myParent ← ∅ ;
3      myClusterId ← ∅ ;
4      setColor(RED);
5  Msg Handler RecruiteMsg({id, color, position}, sender):
6      if myParent = ∅ then
7          myParent ← sender;
8          myClusterId ← id;
9          setColor(color) ;
10         sendP2P(POSITION,myParent,myPosition);
11         foreach connected port do
12             if port ≠ sender then
13                 sendP2P(RECRUIT,port, {id, color,posNeighbor(position, port)});
14 Msg Handler PositionMsg({position}, sender):
15     sendP2P(POSITION,myParent,myPosition);
```

5 Experimentation

We propose the following test bed to demonstrate how the hybrid network system works: Let's consider a set of 168 *Blinky Blocks* representing the skeleton of a

man (cf. Fig. 1 right). This body is made up of 4 sub-sets (head, arms, torso and legs), each associated with an *XBlock*.

The master controller makes use of a web page to manage in real time the application. It can be used to send commands to the *XBlocks*, gather their feedback and graphically display the shapes of the clusters build by our distributed algorithm.

The application starts by requesting each *XBlocks* to build its own cluster. These messages embed the Id of the targeted *XBlock*, the color of the cluster and a banned direction for the recruitment of cluster modules. Initially all *Blinky Blocks* are colored in grey and are idling. As soon as a *XBlock* is asked to build a cluster, it sends a `RECRUITMENT` message to all its neighbors but the one in the banned direction. When a *Blinky Block* receives a `RECRUITMENT` for the first time it memorized the sender (as its parent in the cluster tree), and the color. At the end of the recruitment phase, the modules returns a message to the leader with their position. The leader sends this list to the master controller.

A video[3] presents our set of robots and our distributed algorithm in action.

6 Conclusion

In this paper we presented a significant extension to our *Blinky Blocks*. We now have a clustered architecture using wired multi-hops links inside clusters and wireless communications between cluster-heads and controller. The advantages of this hybrid architecture were demonstrated with a real world experimentation featuring 168 devices, complemented with simulations in our dedicated *VisibleSim* tool. The ease of control allowed by this approach is illustrated by the simplicity of the graphical interface that manage the ensemble. We now intend to further explore the advanced networking aspects enabled by our proposal. Namely, multi-network routing, cluster balancing and functional differentiation between blocks.

Moreover, the proposed extension appears to be a very useful tool the ease and speed up general work on *Blinky Blocks*. We now consider using the cluster-heads to remotely program the modules they control, while providing a robust and versatile framework for monitoring and tracing complex distributed algorithms.

This will also allows us to redesign many distributed algorithms on modular robots using the two communication graphs induced by these networks. For example, to define the coordinates of all modules relatively to a single reference, we could simply calculate their position in the clusters - as done in this work - then bind them remotely.

Acknowledgement

This work has been achieved in the frame of the EIPHI Graduate school ("ANR-17-EURE-0002" contract).

[3] Online video: https://youtu.be/ttbGFtyvJBU

References

1. Jad Bassil, Abdallah Makhoul, Benoît Piranda, and Julien Bourgeois. Distributed size-constrained clustering algorithm for modular robot-based programmable matter. *ACM Trans. Auton. Adapt. Syst.*, jan 2023. Just Accepted.
2. Andre Naz, Benoit Piranda, Seth Copen Goldstein, and Julien Bourgeois. Approximate-centroid election in large-scale distributed embedded systems. In npgb16b:ip editor.pdf, editor, *30th IEEE International Conference on Advanced Information Networking and Applications (AINA 2016)*, pages 548 – 556, Crans Montana, Switzerland, mar 2016. IEEE.
3. Andre Naz, Benoit Piranda, Seth Copen Goldstein, and Julien Bourgeois. A time synchronization protocol for modular robots. In *24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2016)*, pages 109 – 118, Heraklion, Crete, Greece, feb 2016. IEEE.
4. Michel Raynal. *Distributed algorithms for message-passing systems*, volume 500. Springer, 2013.
5. John W. Romanishin, Kyle Gilpin, Sebastian Claici, and Daniela Rus. 3d m-blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1925–1932, 2015.
6. Michael Rubenstein and Radhika Nagpal. Kilobot: a robotic module for demonstrating behaviors in a large scale (units) collective. In *Proceedings of the IEEE 2010 international conference on robotics and automation workshop, modular robotics: state of the art*. Institute of Electrical and Electronics Engineers, 2010.
7. A. Spröwitz, R. Moeckel, M. Vespignani, S. Bonardi, and A.J. Ijspeert. Roombots: A hardware perspective on 3d self-reconfiguration and locomotion with a homogeneous modular robot. *Robotics and Autonomous Systems*, 62(7):1016–1033, 2014. Reconfigurable Modular Robotics.
8. Pierre Thalamy, Benoît Piranda, André Naz, and Julien Bourgeois. Visiblesim: A behavioral simulation framework for lattice modular robots. *Robotics and Autonomous Systems*, page 103913, 2021.