BittyBuzz

Buzz for microcontrollers

Emir K. Belhaddad, Anthony Dentinger

POLYTECHNIQUE
MONTRÉAL

WORLD-CLASS
ENGINEERING

Introduction

- Buzz: **Description of collective swarm behavior** with both top-down and bottom-up approaches [1]. Details of information propagation are hidden.
- Swarm intelligence and behavior require large numbers of robots with limited resources ⇒ result must be achieved with the **cheapest robots possible**.

How cheap can Buzz go?

Introduction

BittyBuzz: Reimplementation of **Buzz for microcontrollers**.
Designed for very cheap robots with extreme resource constraints.

Currently, only implementation is for **Kilobots**, inexpensive robots designed by Harvard to swarm in the thousands [2].

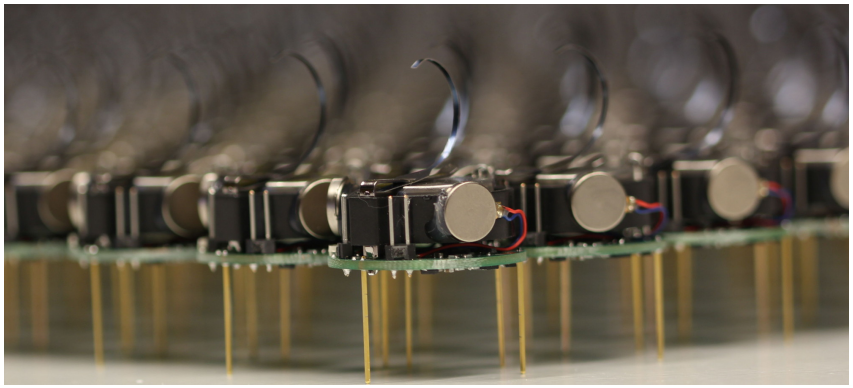

Fig. 1: A Kilobot Batalion [3]

Outline

Content

Resource comparison

Let us start by comparing the 2 targeted platforms:

- **Khepera IV**: Robot with a system able to support Buzz.
- **Kilobot**: Robot with **very** little resources available.

	Khepera IV	Kilobot
Processor	32-bits @ 800 MHz	8-bits @ 8 MHz
Flash	512 MB (+ 4 GB)	32 KB
RAM	512 MB	2 KB
Payload bandwidth	$\sim$ 1 MB/s [1] [4]	350-450 B/s
Packet drops	None with TCP/IP	$\approx$ 50%

Table 1: Resource comparison between the Khepera IV and Kilobot robots [5]

[1] Assuming CCK modulation scheme

Dynamic memory management

- Internal pre-allocated heap
- 3 sections: Objects, Segments and Unclaimed
- All objects have 2 bytes payload and 1 byte meta-data
- Simple GC algorithm

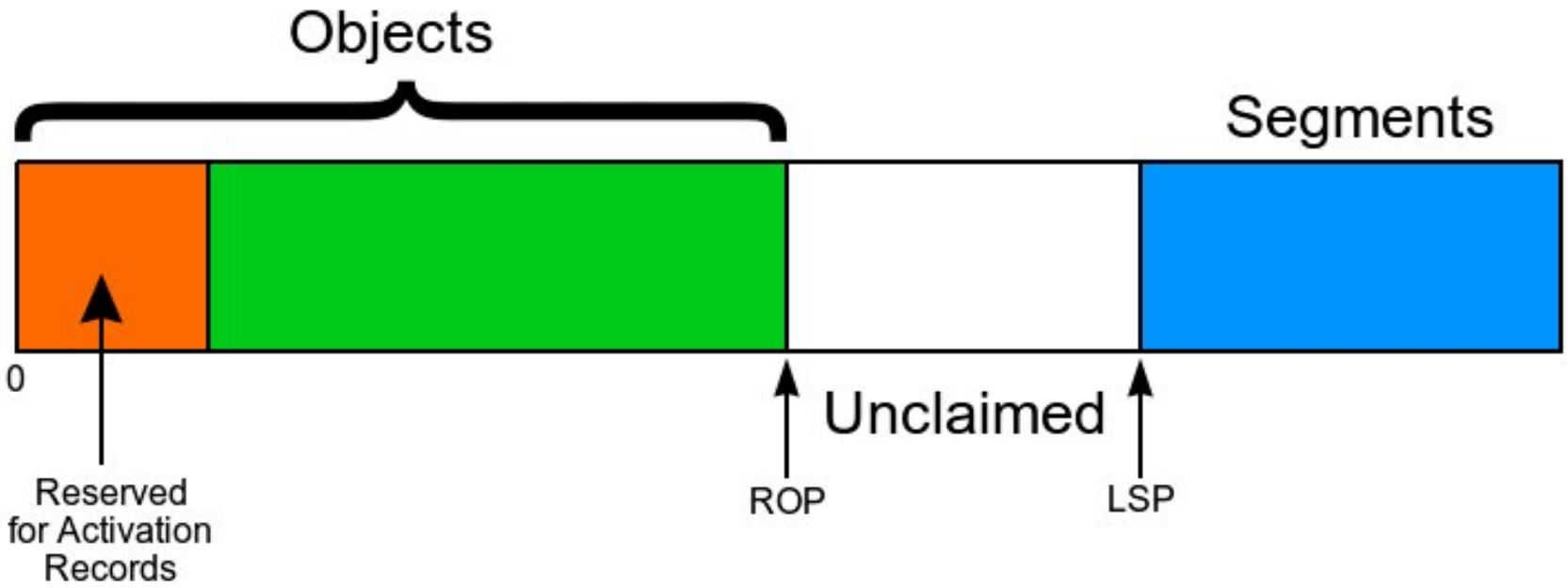

Fig. 2: Section placement in the heap

Point on Optimizations

Some optimizations made on BittyBuzz, grouped by what they optimize:

RAM	Flash	Bandwidth
• Closures	• Function vs Macros	• Sorted neighbors
• 2B payload	• Optimized loops	• Ring-buffers
• Unique alloc	• Translated bytecode	
...	...	...

Table 2: Some of the optimizations made to BittyBuzz

Content

Differences between Buzz and BittyBuzz are minimal. Nonetheless, **what differences have resulted from these limitations?**

$$Buzz - BittyBuzz = \ ?$$

Theorem (accepted):

$$\begin{aligned} BittyBuzz - Buzz = \ & \Delta Architecture + \Delta(Buzz\ features) \\ & + \Delta(Closure\ definitions) + (small\ improvements) \end{aligned} \tag{1}$$

Architecture

Microcontrollers with no OS ⇒ existance of platform-dependant operations.
Out of the box, BittyBuzz thus has **two layers of implementation**:

- Higher-level "core" layer for **platform-independant operations** (VM bytecode execution, definition of `swarm`, `stigmergy`, ...).
- Thin, lower-level robot layer for **platform-dependant operations** (fetching bytecode, displaying errors, sending and receiving packets, ...). Must be implemented for each robot.

Buzz features

Consistency with Buzz has been a key factor to the development choices, however **some features still have limitations**. Examples:

- Only one `stigmergy` allowed ;
- `stigmergy` topic must be a string (currently, but can be overcome) ;
- `swarm` IDs range from 0 to 7.

Closure definitions

Buzz C closures allow a user to call a C function from the Buzz side. Making them in BittyBuzz is also very similar.

Buzz

```
int buzz_c_closure(buzzvm_t vm) {
  // Error if not passed 1 param.
  buzzvm_lnum_assert(vm, 1);

  // Take int value of param.
  buzzvm_lload(vm, 1);
  int16_t param1 =
    buzzvm_stack_at(vm, 1)->i.value;
  buzzvm_pop(vm);

  buzzobj_t result;
  // Compute result...

  buzzvm_push(result);
  return buzzvm_ret1(vm);
}
```

BittyBuzz

```
void bbz_c_closure() {
  // Error if not passed 1 param.
  bbzvm_assert_lnum(1);

  // Take int value of param.
  int16_t param1 =
    bbzheap_obj_at(
      bbzvm_locals_at(1))->i.value;


  bbzheap_idx_t result;
  // Compute result...

  bbzvm_push(result);
  bbzvm_ret1();
}
```

Fig. 3: C Closures in Buzz vs. BittyBuzz

Content

```
function init() {
  iteration = 0
  if(id == 7) {
    mydist = 0
  }
  else {
    mydist = 600
    # Listen to other robots' distances
    neighbors.listen("dist_to_source",
      function(value_id, value, robot_id) {
        var n = neighbors.get(robot_id)
        if (n != nil and value != nil) {
          mydist = math.min(mydist, n.distance + value)
        }
      })
  }
}
function step() {
  # Displaying with gradient of color

  # Set message to be passed every 3s
  if(iteration % 10 == 0 and mydist < 600) {
    neighbors.broadcast("dist_to_source", mydist)
  }
  iteration = iteration + 1
}
```

Fig. 4: Distance gradient source code

Distance Gradient Algorithm

distance gradient demo

```
function init() {
    stig = stigmergy.create(0)
    stig.onconflict(function(key, ld, rd) {
            return ld
    })
    if (id == 7) {
        stig.put("1", 42)
    }
}

i = 20
function step() {
    i = i + 1
    if (id == 7 and i % 20 == 0) {
        stig.put("1", 41 + ((i / 20) % 3))
        led(7)
    }
    var val = stig.get("1")
    if (val < 42) {
        led(3)
    }
    else if (val > 42) {
        led(6)
    }
    else {
        led(2)
    }
}
```

Fig. 5: Stigmergy demo source code

Stigmergy Demo

distance gradient demo

```
function init() {
    # Create swarms
    s0 = swarm.create(0)
    s1 = swarm.create(1)

    # Join one or both swarms depending on ID value
    s0.select(id % 3 != 1)
    s1.select(id % 3 != 0)
}

function step() {
    # Make swarm execute behaviors
    s0.exec(swarm_behavior)
    s1.exec(swarm_behavior)
}

function swarm_behavior() {
    # Switch behavior depending on the swarm executing the closure
    if (swarm.id() == 0) {
        led(1) # RED
    }
    else {
        led(2) # GREEN
    }
    delay(200)
}
```

Fig. 6: Swarm demo source code

Swarm Demo

distance gradient demo

Content

Future work

So that's it?

No. There is still more work to be done for BittyBuzz:

- Implementation of **BittyBuzz for zooids**.
- Keep BittyBuzz up to date with new Buzz features.
- Many, many general improvements.
- Conduct **research projects using BittyBuzz**.

When you're finished changing, you're finished. – Benjamin Franklin

Concluding words

All in all, BittyBuzz:

- Reworks (and somewhat improves) Buzz, targetting the **specific limitations of inexpensive robots**;
- Attempts to **behave as Buzz** whilst allowing for easy extension to new robots ;
- Can implement simple swarm behaviors **on Kilobots**.

Concluding words (continued)

So long, and thanks for all the fish!

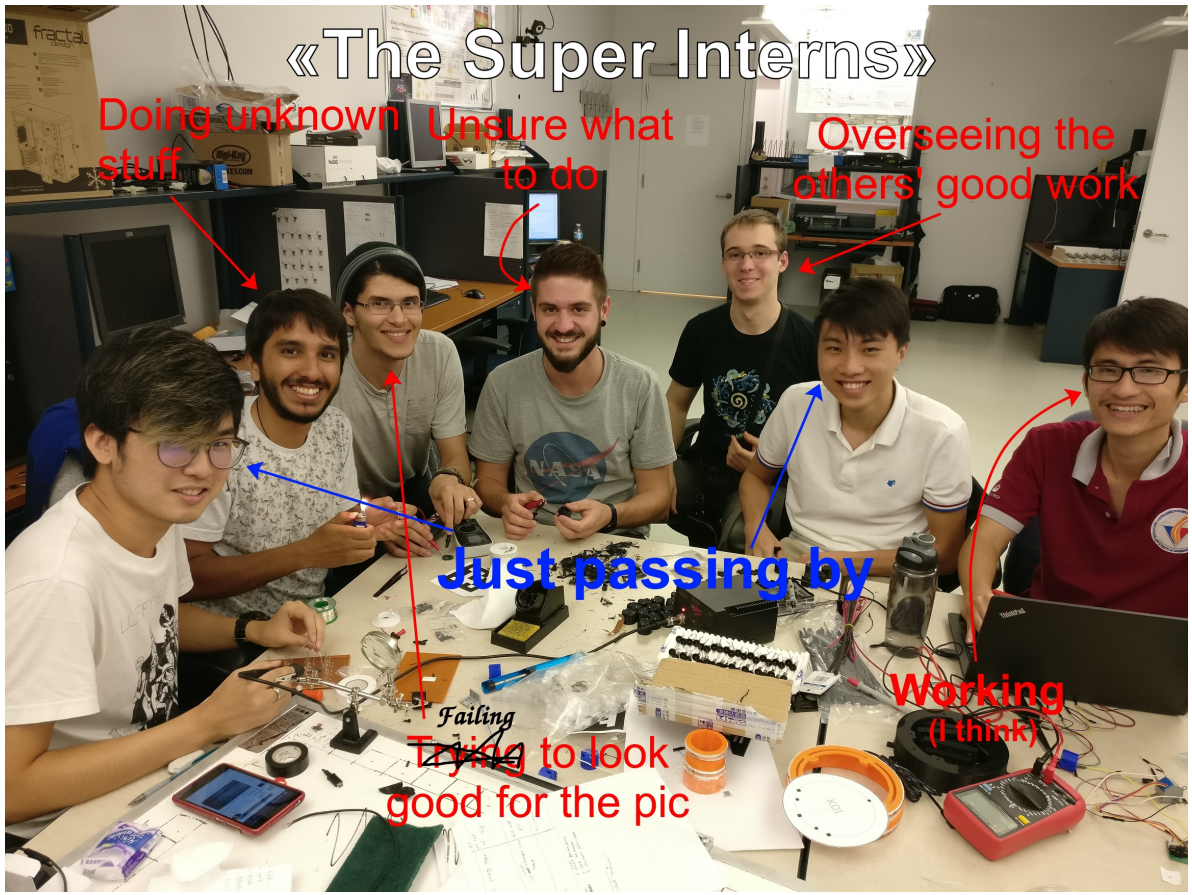

Fig. 7: "And now, for something completely different..."

References I

[1] C. Pinciroli, A. Lee-Brown, and G. Beltrame, "Buzz: An extensible programming language for self-organizing heterogeneous robot swarms," 2015. [Online]. Available: https://arxiv.org/abs/1507.05946

[2] M. Rubenstein, C. T. Ahler, and R. Nagpal, "Kilobot: A low cost scalable robot system for collective behaviors," in *Proceedings of 2012 IEEE International Conference on Robotics and Automation (IRCA 2012)*, Computer Society Press of the IEEE, Ed., 2012. [Online]. Available: https://dash.harvard.edu/handle/1/9367001

[3] SSR Lab, Harvard University. Kilobotics. [Online]. Available: https://www.kilobotics.com/

References II

[4] Wi2Wi Inc., "W2CBW003 - 802.11 b/g + Bluetooth(TM) System-in-Package." [Online]. Available: http://www.mouser.com/ds/2/437/W2CBW003_PB%20rev1.2-3707.pdf

[5] K-Team Corporation, "Khepera IV Specifications." [Online]. Available: https://www.k-team.com/mobile-robotics-products/khepera-iv/specifications