INVITED CONTENT

Editor: **Giuliano Antoniol**
Polytechnique Montréal

Editor: **Steve Counsell**
Brunel University

Editor: **Phillip Laplante**
Pennsylvania State University

Buzz

A Programming Language for Robot Swarms

Carlo Pinciroli and Giovanni Beltrame

THE FIELD OF swarm robotics researches fully decentralized approaches for coordinating large-scale teams of robots. This research is ambitious: it envisions robot swarms for scenarios for which current solutions are impractical, too dangerous, or nonexistent.

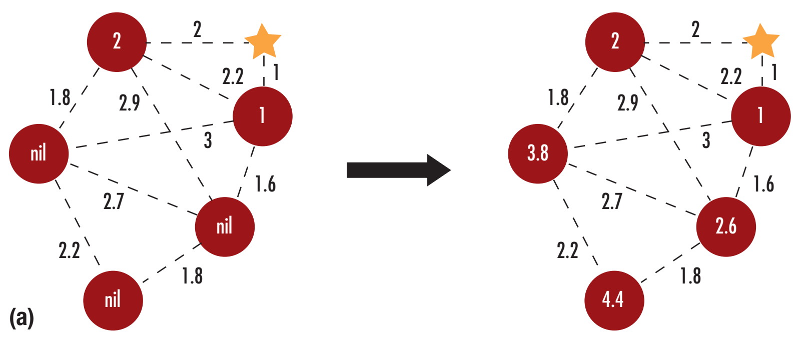
From drones to self-driving cars, robot swarms are becoming pervasive and are used in many kinds of applications. Such applications include search and rescue, industrial and agricultural inspection, coordinated vehicle platooning, space exploration, and nanomedicine. We envision a world in which designers can specify the behavior of heterogeneous groups of robots and package it in an application that can be installed on multiple robotic systems. Swarm-based solutions will likely form the backbone for the upcoming self-driving-car infrastructure and will enable widespread consumer robotics.

Over the past 20 years, researchers have applied biological swarm models to robotics scenarios to propose new approaches or validate the models in fully controllable conditions. However, over the past decade, a research trend has emerged that parts from natural inspiration and focuses on the engineering aspects of designing robot swarm systems. This trend is motivated by the need to consolidate current knowledge in the field, which still appears like a collection of scattered success stories.

An important open problem in this context is the lack of general tools for experimentation. Remarkable progress is being made on the hardware side, with the creation of widely used swarm robotics platforms such as the e-puck[1] and Kilobot.[2] On the software side, significant research concentrates on single-robot scenarios, with frameworks such as the Robot Operating System.[3] However, common "swarm libraries" do not yet exist, and reusing code is difficult owing to the lack of swarm-centric development platforms. To address these problems, we created Buzz, a programming language for heterogeneous robot swarms.

A Swarm as a Programmable Machine

Although it seems natural to deal with robot swarms as yet another instance of a classic distributed system, important aspects set robot swarms apart from distributed systems. Robot swarm dynamics are characterized by an inseparable mixture of spatial and network aspects. Spatial aspects include the fact that robots move and modify



```
# Define the function to execute upon receiving information on the distance to
# the target. This function is executed once at initialization.
function init() {
  neighbors.listen("dist_to_target",
    function(topic, value, robot_id) {
      dist = math.min(dist, neighbors.get(robot_id).distance + value)})
}
# This function is executed at each time step.
function step() {
  # Broadcast currently known distance to target (could be nil).
  neighbors.broadcast("dist_to_target", dist)
}
```
(b)

FIGURE 1. Definition of a simple distance gradient across a robot swarm. (a) The swarm diagram, in which the star denotes a target, the nodes contain the value of the variable dist of a robot, and the arcs indicate the network or sensing topology, labeled with the distances. (b) The corresponding Buzz code.

their surrounding environment. Network aspects include a communication modality based on range-limited, gossip-based message passing and an ever-changing topology due to robot navigation across the environment. So, the mapping between swarm-level requirements and individual actions is a problem whose solution exceeds current approaches to distributed-system design. Designing and developing swarm behaviors is currently a slow trial-and-error process in which the main success factors are the designer's expertise and ability to encode complex behaviors.

The key design challenge in Buzz was providing adequate abstraction to let developers express complex swarm algorithms comfortably. Swarm robotics researchers have proposed two opposite approaches:[4]

- The *bottom-up approach* focuses on individual robots and their low-level interactions.
- The *top-down approach* treats a swarm as a continuous, unique entity (for example, through aggregate programming or spatial computing[5]).

Although the bottom-up approach ensures total control of the design, the amount of detail exposed to developers is often overwhelming. In contrast, the top-down approach presents a simple abstraction of the swarm but prevents developers from fine-tuning individual robots' behaviors. Buzz is based on the idea that the developer should be offered both levels of abstraction and that the language's syntax should allow for seamless mixing of bottom-up and top-down constructs.

In Buzz, the "machine" being programmed is a swarm seen as a discrete, heterogeneous collection of robots, each of which executes a Buzz Virtual Machine (BVM). Once a script has been compiled and transmitted to the robots (all robots share the same script), the BVMs run their local copy of it.

One reason we developed the BVM is to facilitate implementing Buzz in resource-limited robots such as the e-puck and the Kilobot. The BVM's compiled binary is smaller than 12 kilobytes. Despite the BVM's tiny footprint, it provides automatic garbage collection and low-bandwidth, robust communication protocols. In particular, it uses *situated communication*[6] whereby a robot, upon receiving a message, can estimate the message source's relative location.

Buzz Highlights

Buzz's syntax mixes imperative and functional constructs and is inspired by dynamic languages such as JavaScript, Lua, and Python. This ensures a short learning curve for newcomers, and readable, familiar-looking scripts.

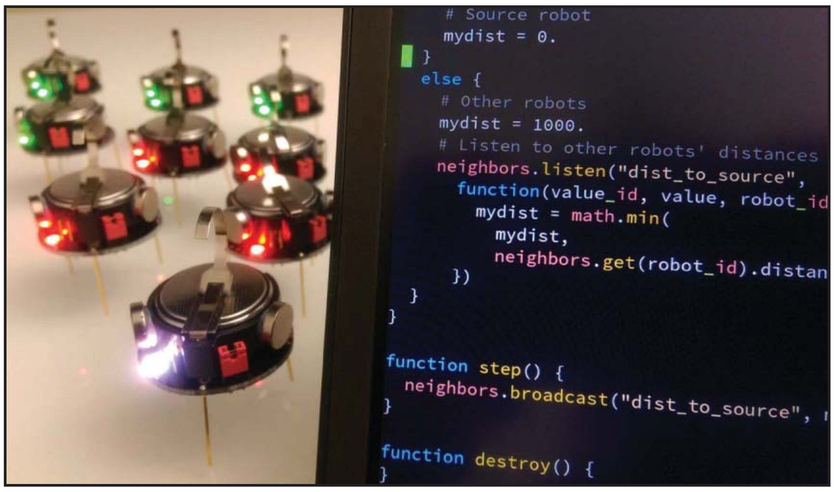
The language's bottom-up primitives are analogous to those in traditional dynamic languages and include assignments, arithmetic operations, loops, branches, and function definitions.

Buzz's main novelty is three top-down primitives that cover a wide range of scenarios and let developers compose complex, reusable behaviors starting from simple concepts. What before took hundreds of lines of platform-dependent C or Python code now can be expressed compactly and intuitively combining these primitives and can be reused for any kind of robot.

The first top-down primitive is the `swarm`, a collection of robots for which a certain predicate is true. Developers can perform set operations on `swarm`s such as union, intersection, difference, and complement. They can assign to a `swarm` dedicated tasks, expressed through lambdas.

The second top-down primitive is the `neighbors` type. It stores positional information about a robot's immediate neighbors—that is, those within communication range—and provides methods to exchange messages. Developers can use variables of the `neighbors` type to compute motion vectors and computational fields. For example, Figures 1 and 2 show how simple it is to create a distance gradient from a specific location by combining broadcast (`neighbors.broadcast`) and message-driven callbacks (`neighbors.listen`, which sets up a callback triggered whenever a neighbor broadcasts a certain key–value pair). In most other high-level languages, this would require complex synchronization and message passing. In addition, developers can create `neighbors` variables by filtering or mapping functions onto the stored data.

The third top-down primitive is *virtual stigmergy*,[7] a tuple space that lets the robots share `(key, value)` pairs across the swarm. Stigmergy refers to a communication modality whereby social insects mark and modify the environment (rather than talking directly) to coordinate tasks such as food search and nest construction. The tuple space acts as a virtual environment in which all robots store information to achieve global consensus. Virtual stigmergy is instrumental to encode globally optimized behaviors such as task allocation and scheduling and to coordinate swarm-wide state transitions.



```
        # Source robot
        mydist = 0.
    }
    else {
        # Other robots
        mydist = 1000.
        # Listen to other robots' distances
        neighbors.listen("dist_to_source",
          function(value_id, value, robot_id
            mydist = math.min(
              mydist,
              neighbors.get(robot_id).distan
          })
    }
}

function step() {
  neighbors.broadcast("dist_to_source", 
}

function destroy() {
}
```

FIGURE 2. Kilobots forming a gradient. The Buzz code for gradient construction is in the foreground.

These top-down primitives do not cover the spectrum of possible swarm algorithms. Nevertheless, they constitute a simple yet powerful set of basic building blocks with which developers can express, combine, and compare a large class of swarm behaviors.

We are creating the first library of swarm behaviors and testing it on real robots of various types. We believe that Buzz and the library will enable future research on real-world, complex swarm robot systems. Currently, no standardized platform lets researchers compare, share, and reuse swarm behaviors. Inescapably, development involves recoding recurring swarm behaviors, such as flocking, reaching consensus, and creating gradients. Buzz's design is motivated and nurtured by the need to overcome this state of affairs. We hope Buzz will have a lasting impact on the growth of the swarm robotics field.

We ultimately aim to create the first software ecosystem for swarm robotics applications, comprising a domain-specific programming language and the means to deploy, monitor, and debug swarm behaviors. This ecosystem would enable the sharing and comparison of swarm algorithms, letting practitioners lay the foundations of a principled "swarm engineering."

Buzz is distributed as open source software and is available at http://the.swarming.buzz.

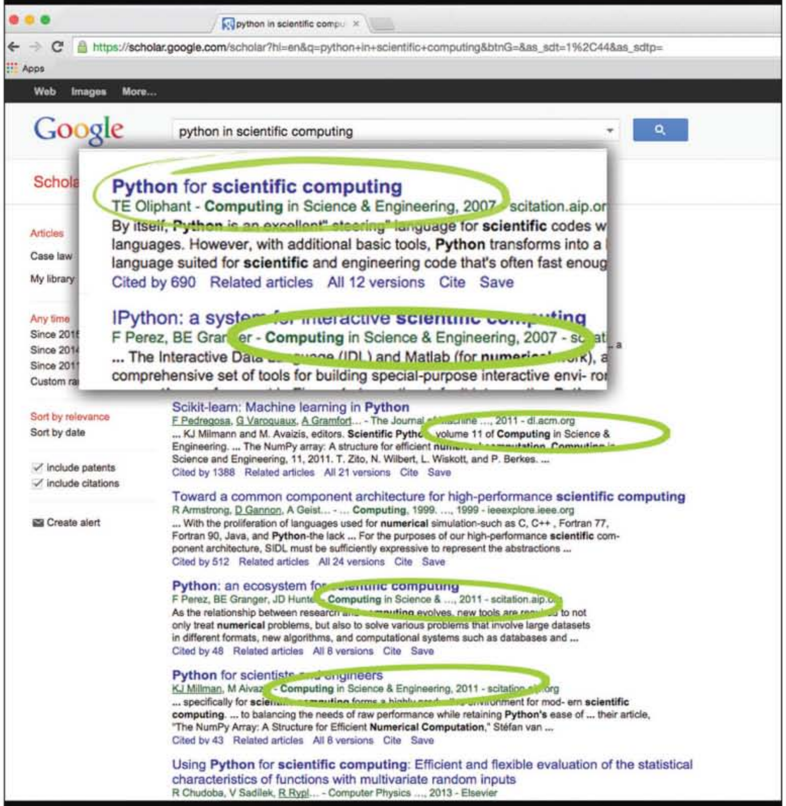
References

1. F. Mondada et al., "The e-puck, a Robot Designed for Education in Engineering," *Proc. 9th Conf. Autonomous Robot Systems and Competitions* (Robotica 06), vol. 1, 2006, pp. 59–65.

2. M. Rubenstein, C. Ahler, and R. Nagpal, "Kilobot: A Low Cost Scalable Robot System for Collective Behaviors," *Proc. 2012 IEEE Int'l Conf. Robotics and Automation* (ICRA 12), 2012, pp. 3293–3298.
3. M. Quigley et al., "ROS: An Open-Source Robot Operating System," *Proc. ICRA Workshop Open Source Software*, 2009, p. 5.
4. M. Brambilla et al., "Swarm Robotics: A Review from the Swarm Engineering Perspective," *Swarm Intelligence*, vol. 7, no. 1, 2013, pp. 1–41.
5. J. Beal et al., "Organizing the Aggregate: Languages for Spatial Computing," arXiv:1202.5509, 2012; http://arxiv.org/abs/1202.5509.
6. K. Støy, "Using Situated Communication in Distributed Autonomous Mobile Robots," *Proc. 7th Scandinavian Conf. Artificial Intelligence*, 2001, pp. 44–52.
7. C. Pinciroli, A. Lee-Brown, and G. Beltrame, "A Tuple Space for Data Sharing in Robot Swarms," *Proc. 9th EAI Int'l Conf. Bio-inspired Information and Communications Technologies* (BICT 15), 2015.

CARLO PINCIROLI is a postdoctoral researcher in the MIST (Making Innovative Space Technology) laboratory at École Polytechnique de Montréal. Contact him at carlo@pinciroli.net.

GIOVANNI BELTRAME is an associate professor in the Computer Engineering Department at École Polytechnique de Montréal and directs the MIST (Making Innovative Space Technology) laboratory. Contact him at giovanni.beltrame@polymtl.ca.



python in scientific compu
https://scholar.google.com/scholar?hl=en&q=python+in+scientific+computing&btnG=&as_sdt=1%2C44&as_sdtp=
Apps
Web Images More...
Google
python in scientific computing
Schola
Articles
Case law
My library
Any time
Since 201
Since 201
Since 201
Custom ra
Sort by relevance
Sort by date
include patents
include citations
Create alert

Python for **scientific computing**
TE Oliphant - **Computing** in Science & Engineering, 2007 - scitation.aip.or
By itself, **Python** is an excellent" steering" language for **scientific** codes w
languages. However, with additional basic tools, **Python** transforms into a
language suited for **scientific** and engineering code that's often fast enoug
Cited by 690 Related articles All 12 versions Cite Save

IPython: a system for interactive **scientific computing**
F Perez, BE Granger - **Computing** in Science & Engineering, 2007 - scitati
... The Interactive Data Language (IDL) and Matlab (for **numerical** work), a
comprehensive set of tools for building special-purpose interactive envi- ron

Scikit-learn: Machine learning in **Python**
F Pedregosa, G Varoquaux, A Gramfort... - The Journal of Machine ..., 2011 - dl.acm.org
... KJ Milmann and M. Avaizis, editors. **Scientific Pytho**n volume 11 of **Computing** in Science &
Engineering. ... The NumPy array: A structure for efficient numerical computation. Computing in
Science and Engineering, 11, 2011. T. Zito, N. Wilbert, L. Wiskott, and P. Berkes. ...
Cited by 1388 Related articles All 21 versions Cite Save

Toward a common component architecture for high-performance **scientific computing**
R Armstrong, D Gannon, A Geist... - ... **Computing**, 1999. ..., 1999 - ieeexplore.ieee.org
... With the proliferation of languages used for **numerical** simulation-such as C, C++ , Fortran 77,
Fortran 90, Java, and **Python**-the lack ... For the purposes of our high-performance **scientific** component architecture, SIDL must be sufficiently expressive to represent the abstractions ...
Cited by 512 Related articles All 24 versions Cite Save

Python: an ecosystem for **scientific computing**
F Perez, BE Granger, JD Hunter - **Computing** in Science & ..., 2011 - scitation.aip.org
As the relationship between research and computing evolves, new tools are required to not
only treat **numerical** problems, but also to solve various problems that involve large datasets
in different formats, new algorithms, and computational systems such as databases and ...
Cited by 48 Related articles All 8 versions Cite Save

Python for scientists and engineers
KJ Millman, M Aivazis - **Computing** in Science & Engineering, 2011 - scitation.aip.org
... specifically for **scientific computing** forms a highly productive environment for mod- ern **scientific computing**. ... to balancing the needs of raw performance while retaining **Python's** ease of ... their article,
"The NumPy Array: A Structure for Efficient **Numerical Computation**," Stéfan van ...
Cited by 43 Related articles All 8 versions Cite Save

Using **Python** for **scientific computing**: Efficient and flexible evaluation of the statistical
characteristics of functions with multivariate random inputs
R Chudoba, V Sadilek, R Rypl... - Computer Physics ..., 2013 - Elsevier

Computing in Science & Engineering
NERSC: 40 Years and Counting
IEEE
AIP