There and Back Again: Gene-Processing Hardware for the Evolution and Robotic Deployment of Robust Navigation Strategies

David M. Bryson*
BEACON Center
Michigan State University
East Lansing, Michigan, USA
+1 (517) 884-2555
brysonda@egr.msu.edu

Aaron P. Wagner*
BEACON Center
Michigan State University
East Lansing, Michigan, USA
+1 (517) 884-2555
wagner@metsci.com

Charles Ofria
Computer Science and Engineering
Michigan State University
East Lansing, Michigan, USA
+1 (517) 884-2562
ofria@egr.msu.edu

ABSTRACT

Navigation strategies represent some of the most intriguing examples of complex and intelligent behaviors in nature. Accordingly, they have been the focus of extensive research in animal behavior and in evolutionary robotics. However, engineering successes in harnessing the evolutionary dynamics that shape sophisticated navigation strategies remain limited. Here we describe a novel gene-processing architecture for digital organisms that enables the evolution of central-place-foraging strategies, such as those seen in honeybees and striped hyena. While previous studies have evolved navigation *de novo*, the resulting algorithms have been relatively fragile and difficult to translate into physical systems. In contrast, the strategies evolved in this study are highly congruous with those seen in nature: a single evolved foraging strategy incorporates periods of directed travel, fixed pattern search, cue response, and reorientation when outcomes do not match expected results. Additionally, the genetic architecture enabled rapid extraction of the underlying behavioral algorithm and transference to a robotic system, proving to be robust to issues of noise and scale that commonly plague such attempts. Accordingly, we demonstrate that the flexibility and interpretability of the new gene-processing hardware readily facilitate the creation, study, and utilization of naturalistic and deployable algorithms for functionally complex behaviors.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – *Evolutionary prototyping*. I.2.9 [Computing Methodologies]: Robotics – *Autonomous vehicles*.

General Terms

Algorithms, Design, Reliability.

*These authors contributed equally to this work.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s). Copyright is held by the author/owner(s).

GECCO '14, July 12–16, 2014, Vancouver, BC, Canada.

ACM 978-1-4503-2662-9/14/07.

<http://dx.doi.org/10.1145/2576768.2598363>

Keywords

Genetic architecture. Robustness. Navigation. Avida. Robotics. Autonomy. Behavior.

1. INTRODUCTION

Engineering has a long tradition of attempting to emulate the behaviors and intellectual capacities of biological organisms in robotic and autonomous systems [1]. Evolutionary computation takes the next step: instead of mimicking the specific behaviors, it replicates the process that produced those behaviors, with a goal of finding solutions that work more effectively with their hardware. However, it remains challenging to evolve engineered systems with the complexity and flexibility of the biological systems that originally inspired them. For example, if one attempts to evolve navigation strategies to a fixed target, fixed travel paths tend to emerge, instead of something more flexible and responsive [2,3]. Even when there is variability in the requirements, static strategies still often dominate, even if they produce only limited success [4]. While such results do offer insights into potential approaches for addressing highly specified engineering tasks, they do little in the way of contributing broadly applicable algorithms or toward furthering our understanding of intelligent behaviors. Consequently, such results fail to fully capture the potential of evolution for furthering engineering, and determining a method to consistently produce flexible and intelligent responses has become an important goal.

Similarly, a major goal of biological research is to understand the processes that favor the evolutionary emergence of adaptive complexity and modulate the functional expression of responsive behaviors. Thus, studies that contribute to the aims of both biology and engineering are those focused on the patterns and processes involved in the evolutionary emergence of responsive traits (e.g., navigation), not purely functional ones (e.g., locomotion or motility). For example, what are the most effective architectures that permit or promote the evolution of truly intelligent systems, such as systems that can predict and respond to conditions beyond those they are explicitly engineered to deal with? Can we evolve these systems in an open-ended manner, avoiding the assumptions and inherent restraints of engineering them? If so, what environments favor their evolution?

As a flexible platform for investigating the evolutionary patterns and processes involved in producing complex traits, the digital evolution software platform Avida [5] remains the most widely used. One advantage of Avida is that the von Neumann style

genetic architectures of the agents in the evolving populations provide easier algorithmic transparency, enabling a deep understanding of the evolved solutions. Accordingly, Avida has been used to study the evolution of, and algorithms supporting, simple forms of memory [3], gradient and path following [4,6], cooperation [7,8], division of labor [9], and other complex behaviors [10-13].

The genetic hardware of an evolving system is critical in determining the evolutionary potential of populations [14]. Digital evolution in Avida has been shown to be incredibly robust to minor changes in genetic architecture [15], allowing for fruitful exploration of new design approaches. Accordingly, prior work has shown that some architectural modifications can allow populations to better adapt to their environment, both more quickly and with greater long-term success. For example, hardware traits such as enhanced evolvable control of regulatory interactions between genetic instructions and portions of genomes, can allow populations to better adapt to their environment, both more quickly and with greater long-term success [16].

Here we describe a new genetic architecture for Avida incorporating design elements for the evolutionary creation of more flexible and modular genomes. In particular, the design includes enhanced capacities for genetic execution flow control, parallel operations, and coordination. This design philosophy was predicated on the idea that, just as regulatory mechanisms support complex behaviors in nature, incorporating these elements into Avida would better enable the selective execution of genomic code based on environmental stimuli. Thus these architectural features were expected to advance the evolution of intelligent stimulus-response behavioral algorithms and facilitate high levels of cognitive decision-making. We show that these expanded regulation capabilities do support the rapid evolution of more sophisticated, intelligent, naturalistic, and responsive behaviors in a dynamic and challenging foraging and navigation environment. Moreover, the algorithms underlying these behaviors were readily transferred to a simple real-world robot, seemingly unaffected by the typical ‘reality-gap’ issues of over-fit strategies.

2. DESIGN

In order to describe and measure the performance of the gene-based ‘GP8’ hardware, we compare it to three other Avida architectures, described here.

2.1 Heads Architecture

The Heads instruction set architecture is the default virtual CPU design in Avida. It consists of a von Neumann style design, featuring a single memory space containing the instructions of the genomic program. The 26 standard instructions operate on three 32-bit registers, four memory space pointers (called heads), two stacks, and input/output buffers. Accordingly, a typical organism consists of a virtual CPU executing a genomic program that interacts with the environment to collect resources and produces an instruction-by-instruction copy of itself, after which the copy is split to create an offspring organism. The virtual CPU in the Heads architecture executes a single instruction per cycle. Each organism is granted 30 CPU cycles per update.

2.2 Threaded Heads-EX Hardware

The Heads-EX hardware [16] extends Heads architecture, adding additional registers, enhanced instruction argument semantics, and multiple new conditional and flow control instructions. The instruction set options include additional instructions such as those enabling thread creation and coordination, environment

sensing, and movement [e.g., as in 17-19]. The threading instructions allow genomic programs to activate multiple threads of execution in the virtual CPU by executing the ‘fork’ instruction. Each thread maintains its own set of registers and a local stack. During each cycle, the CPU executes a single instruction for each thread, in the order that the threads are created. The thread coordination instructions allow threads to pause their own execution until certain values appeared in the registers of other threads.

2.3 BCR (Behavioral Co-Routine) Hardware

The BCR hardware extends the Heads-EX hardware, adding co-routine pre-processing, instruction functional classes, and CPU cycle micro-operations. The BCR virtual CPU pre-processes the genomic program into discrete co-routines in independent memory spaces. Each co-routine is executed in its own thread. We modified the existing genomic execution hardware to allow for separate regulatory control mechanisms for distinct functional classes. Specifically, instructions governing reproduction, sensory input, and behavioral actions were grouped into the three functional classes, respectively. Behaviorally distinct co-routines, for example a routine dedicated exclusively to reproduction, can run completely in parallel with other operations. Execution of functionally classed instructions is limited such that only one co-routine can execute an instruction from each class per cycle. Subsequent attempts to execute a functionally classed instruction during a given CPU cycle cause the thread to block until the next cycle. We altered CPU cycle processing to execute up to 20 non-classed instructions (micro-operations) per cycle, permitting data processing and decision-making to be more environmentally responsive.

2.4 GP8 (Gene-Processing) Hardware

The GP8 gene-processing hardware relaxes many execution restrictions from the BCR hardware, resulting in the pre-processed genome segments performing as multiple independent subprograms (genes) that are concurrently executed on independent threads. Each thread executes up to 20 micro-operations that may perform any combination of computation, decision-making, sensing, action queuing, and regulation instructions. Three actions, heading rotation, forward movement, and resource intake (eating), may be queued during micro-operation execution. These actions, which interact with and can alter the environment, remain limited to once per CPU cycle. The queued actions are performed at the end of a given cycle, so that any number of genes may manipulate organism behavior without stalling, and as such preventing the starvation of certain threads that could occur with functionally classed instructions in the BCR hardware.

3. APPLICATION

Our explicit goal in this work was to test the potential of gene-based genetic architectures for facilitating the evolution of cognitive decision makers. Specifically, we targeted the evolutionary development of behaviorally plastic organisms that could navigate using strategies that include modifying activities in response to changing conditions. To that end, we evolved BCR and GP8 digital populations challenged with a central-place foraging problem, in which foragers must travel to find food and return to a central place for reproduction (Fig. 1A) [20]. We then developed and used new tools to dissect the evolved behavioral algorithms. Finally, we demonstrated the effectiveness of those algorithms via deployment in a real-world robot.

Fig. 1. Modular genetic architectures facilitate the evolution of naturalistic foraging and navigation strategies. (A) the full BCR population from Box 1, showing agents traveling from the central birth den (large black circle) out to distant food resources (mid-sized grey and white circles), returning home to reproduce (see also Supplementary Movie 1). Organisms (small points) are colored to reflect their current food preferences. Shading of the food resources reflects the current balance between consumption by the foraging organisms and regrowth. While the organism can evolve to look for and process information about the food resource it targets, the central den is not detectable from afar. **B-D** show foraging paths for single organisms pulled from the population. Outbound paths are shown in black (traces began shortly after the organisms left the den), while the inbound path after feeding is white. Each of the seven available food resources move along semi-random paths within the rightmost third of the environment. Cumulative travel paths are shown as colored bands. The darker resource bands apparent in C and D reflect the longer time the agents spent foraging. Note that the organism in D discovered the den only after returning to the resource field a second time to reorient and recalibrate its approach.

3.1 Evolutionary Environment

Each of the 90 evolutionary trials of each genetic architecture was conducted for 200,000 updates in a simulated environment containing spatio-temporally dynamic food resources ($\approx 2,200$ generations for BCR populations and 2,000 for GP8 populations). Mutation rates were set to 0.0015 per site per divide with 0.00075 insertion and deletion probabilities. Populations were limited to 2000 organisms. When a birth would have caused the population to exceed that limit, another member of the population was removed. Maximum age limits were set to 30,000 classed operations (defined in the hardware descriptions, above) before organisms were removed. New organisms were born into the cell faced by their parent, with a random orientation. An unlimited number of organisms could occupy any cell.

World sizes in BCR trials were set to 251 X 251 cells. In the GP8 populations, the world was extended to 501 X 501 cells to repress any evolved behaviors dependent on using the walls around the

Fig. 2. Greater independence of gene execution flows improved the evolvability and robustness of behavioral algorithms. Shown are example foraging route traces for four organisms from different GP8 populations that evolved to look for, process, and respond to information about distant food resources (colored points and bands), and then reliably return to a den site (multi-ringed target) where they can reproduce. The den was not detectable from afar, requiring organisms to navigate back to it without direct sensory input. A and B correspond with the dissected organisms in Box 2 (A = from population 117, B = from population 178). The lighter shading of the food resources reflects the shorter foraging times for these organisms relative to the BCR populations. Overall, navigation strategies incorporated an exceptionally naturalistic and robust mix of fixed pattern search, response to cues, and context-dependent behaviors.

world boundary edges as navigational cues (though this behavior was never detected). Similarly, in BCR, there were an additional three ‘landmark’ resources included in the environment (visible in Fig. 1). As with the boundary walls, we detected no evolved strategies that relied on these landmarks. Each evolutionary world contained seven discrete food resources and one ‘den’ resource. Each food resource had a diameter of 19 cells, while the den resource had a diameter of 39 cells. Each cell of a food resource could contain up to one full unit of food. When an organism fed from a full cell, the resource was fully consumed (depleted to a value of zero), regrowing at a rate of 0.05 units per update. Food values were independent across cells. Den and landmark resources were not consumable.

In addition to the effects of depletion, food resource conditions were kept dynamic by placing them in motion within a set boundary area. Initially, in order to allow the naïve ancestor to survive, each food resource was centered over the den. Over evolutionary time, however, the boundary area was periodically moved to the east of the den. Simultaneously, the width and height of the boundary area was slowly expanded. The final boundary area, reached after 52,500 updates of progressive change, was the area between 65 and 170 cells west of the den, extending 125 cells above and below the world vertical centerline.

Movement of each resource was semi-random and independent. At the start of each update, a resource was moved 10 cells along its current direction of travel. If movement would push the outer edge of a resource beyond the boundary area, that resource would ‘bounce’ by reversing travel direction. Additionally, every 100 updates, the direction of movement would be randomly reassigned, including from options for no change and for no movement. Full descriptions of these relatively new resource options in Avida are available in the code documentation.

In order to feed from a resource, organisms had to set an internal ‘forage-target’ state to the value that matched the targeted food resource (0 thru 6). By default, all organisms are born into a neutral ‘juvenile’ state with a forage-target not matching any resource. Execution of a ‘set-forage-target’ instruction changed the organism’s forage-target to that specified by the value in the instruction’s modifying register. Organisms’ could reset this forage preference state at any time during their lifetime.

To permit the evolution of behavioral algorithms for finding food, each hardware included a visual sensor instruction. Dependent on the instruction mutating into the genome and supporting mutations for controlling sensing behavior, via these sensors, organisms could collect data about their surrounding environment, including characteristics of visible resources, objects (walls, landmarks), and other organisms within a 45-degree cone in the faced

direction. However, the den site was never visible to these sensors (GP8 organisms did have a simple instruction for returning true/false for whether they were currently standing on a den cell). Sensor distances were limited to a maximum of 100 cells.

The full behavior of the visual sensor is described in the Avida documentation. In brief, the sensor mechanism was explicitly designed to allow high levels of flexibility in terms of evolved visual strategies and expressed visual behaviors. Thus, evolution could produce capacities for selectively collecting data on different resource types (including food and other organisms) and at different distances, while also choosing between counts of visible objects (of the specified type), or simply gathering data on the closest visible object or a specific instance of the object type (e.g., food resource #1). As outputs, among other data, the sensor returned integers representing counts of objects, values of objects (e.g., level of food resource), and distance to the nearest of the objects. In effect, the sensors were designed to allow evolution to produce organisms that could selectively process a perfect representation of the visual field. In practice, however, these sensory capacities are extremely complex. Thus, most initial evolutionary attempts to collect sensory data are certain to involve collection of irrelevant data, not to mention the challenge of appropriately processing data once gathered.

We initialized each evolutionary trial with two simple, identical organisms that randomly moved about the environment, blindly attempting to collect resources, copy their genomes, and reproduce. In BCR, movement was limited to one step per update. Organisms were required to have consumed one unit of any food resource before reproducing. Additionally, they were required to be in one of the communal den cells. Thus, all organisms were faced with the challenges of being born at the den, needing to find and move to food resources, successfully consuming resources from a ‘full’ cell, and returning back to a den they could not see.

We used Avida version 2.13 for all experiments. Data were post-processed using Python 2.7.1. Plots were generated using R version 2.15.2 using the ggplot2 and boot libraries.

Box 1. BCR & Robot Evolved Algorithm

```
while (true) {
  if not resource in front:
    rotate counter-clockwise 45 degrees
  if not resource in front:
    rotate counter-clockwise 45 degrees
  if not resource in front:
    move
    rotate counter-clockwise 45 degrees
  else:
    move
  switch (current heading):
    north:    keep current heading
    northeast: rotate to face east
    east:     rotate to face south
    southeast: rotate to face southwest
    south:    rotate to face north
    southwest: rotate to face east
    west:     rotate to face south
    northwest: rotate to face west
  if not resource in front:
    rotate counter-clockwise 45 degrees
  else:
    switch (current heading):
      north:    keep current heading
      northeast: rotate to face east
      east:     rotate to face south
      southeast: rotate to face southwest
      south:    rotate to face north
      southwest: rotate to face east
      west:     rotate to face south
      northwest: rotate to face west
    move
    move
    move
}
```

3.2 Robot Description

Our test robot was an Arduino UNO controlled DFRobot two-wheel drive ‘turtle’ platform driven with a Seeedstudio Motor Shield V2.0, using a Freescale Semiconductor MAG3110 triple axis magnetometer (SparkFun SEN-10619 breakout board) for cardinal direction mapping. The ‘visual’ sensor was emulated with a MaxBotix LV-MaxSonar-EZ0 ultrasonic range finder.

Fig 3. Avida-TV. A Python-based GUI trace viewer, showing the genomic program (left), current virtual CPU state (middle), and current environment state (right). The environment view shows the organism (black arrow) and its visual field (black outline), the resources (medium colored circles), the nest (large gray circle), and stationary landmarks (small colored circles).

3.3 Algorithm Extraction

Avida contains flexible recording methods that enable experimenters to collect detailed information about instruction execution, organism location, and resource availability. We used these recording tools to gather full lifetime traces of organisms from successful populations. We stepped through these instruction traces using Avida-TV (a GUI trace viewer, see Fig. 3), analyzing information intake, decision making, and actions. The resulting distilled behavioral algorithms from a BCR organism and two GP8 organisms are shown in Boxes 1 and 2, respectively.

3.4 Results with New Architectures

The central-place foraging task presented by the environment posed formidable challenges, from evolving vision for finding food to evolving navigation skills to get back to the den. When utilizing the Heads-EX architecture, all 90 trial populations invariably failed to survive the initial boundary area expansions. The BCR architecture demonstrated the first surviving populations, with 10 out of the 90 trials. Of those ten surviving populations, eight utilized fixed-pattern, non-responsive strategies that remain constant regardless of resource conditions. Two of the populations exhibited truly responsive strategies, with organisms actively altering navigational trajectories based on the environment. The GP8 architecture dramatically increased the frequency survival (73 out of 90), as well as the frequency of the evolution of responsive strategies (42 out of 73).

Analysis of the algorithms underlying these evolved foraging behaviors quickly revealed that they are remarkably lightweight, both behaviorally and computationally. For example, the navigation strategy from the BCR organism shown in Figure 1 can be broken into four spatial-context-dependent stages: (1) traveling southeast while resources are visible ahead and to the upper left. (2) Heading due north, feeding along the way, while resources are consistently visible ahead. (3) Alternating between stepping north, west, and south while resources are visible to the south. (4) Heading southwest toward the den provided that resources remain visible to the east. If the den isn't reached by the time all food resources are completely out of sight, the organism recalibrates and reorients itself by returning to the resource field. While this behavior is functionally complex, all actions are governed by reactions to only two values: the current heading and the visibility of resources.

Box 2. GP8 Evolved Behavioral Algorithms

Pop 117:

```
while (true) {
  if resource in front: rotate counter-clockwise 45 degrees
  move
  if resource in front: rotate counter-clockwise 45 degrees
  move
  if resource in front: rotate counter-clockwise 45 degrees
  else:
    switch (current heading):
      north: keep current heading
      northeast: rotate to face east
      east: rotate to face south
      southeast: rotate to face southwest
      south: rotate to face north
      southwest: rotate to face east
      west: rotate to face south
      northwest: rotate to face west
  move
}
```

Pop 178:

```
loop_count = 0
while (true) {
  loop_count += 1
  rotate clockwise 45 degrees
  move
  if have eaten: rotate clockwise 45 degrees
  move
  if resource in front: rotate counter-clockwise 45 degrees
  else: rotate counter-clockwise 90 degrees
  move
  if loop_count == 15:
    loop_count = 0
    if resource in front: rotate counter-clockwise 45 degrees
    else: rotate counter-clockwise 90 degrees
    move
  if have eaten: rotate clockwise 45 degrees
  move
}
```

Fig. 4. Flexibility in genetic architectures permits rapid evolution of informed strategies. While sensory intake increased steadily over evolutionary time (A) for both hardware, evolved strategies were progressively more effective at reducing the duration of foraging trips, as measured by age at reproduction (C; in updates) or number of instructions executed (B). Note that the design of the GP8 architectures enables more instruction executions per update because of instruction queuing vs gene stalling in BCR. Lines are LOESS fits. Shaded regions indicate bootstrapped 95% confidence intervals. The resource field boundary area, which was moved progressively farther from the den, reached its furthest and final point at 52,500 updates.

Fig. 5. Composite depiction of the test robot's navigation path. The robot starts off in the nest (outlined square), navigating toward and through the resource field (green towers), following the blue highlighted path. The corner symbols demarcate the ultrasonic sensor interference buffer zone. The inset depicts a path followed by the source organism within the Avida environment.

The GP8 hardware exhibited populations with similarly lightweight, yet robust navigational strategies. Additionally, two general strategies appeared. The first was similar to the BCR algorithm (Figure 2a and 2c, see also the algorithm for Pop 117 in Box 2 and supplemental Movie 2). The second general approach was state-full, navigating into the resource field prior to successful resource collection, and keeping the resource field at the edge of sight after consumption (Figure 2b and 2d, the algorithm for Pop 178 in Box 2, and supplemental Movie 3).

In general, organisms evolved with the GP8 hardware exhibited substantially increased usage of lifetime CPU cycles compared to the BCR hardware due to the reduction in thread starvation potential (Figure 4b). The increased computation of GP8 organisms allowed for notably increased intake of environment information through the sensor instruction (Figure 4a). This likely contributed substantially to the increased success of GP8 populations. Additionally, GP8 organisms had a somewhat reduced gestation time (Figure 4c), likely conferring a competitive advantage to such organisms.

3.5 Robot Deployment

We directly implemented the algorithm displayed in Box 1 within our test robot using Arduino C++. The robot was then tested in a 6-foot square area (see Figure 5), using lightweight circular towers to emulate resources. The towers obstructed the ultrasonic distance, thus were 'visible' to the robot when the measured distance was less than a set threshold (30 inches north/south or east/west, 42 inches diagonally). Bumping into one of the resource towers was counted as a successful collection. The robot was started in the nest, marked as the black outlined square. Because the robot could not reproduce, as an organism would in

Avida, crossing over the nest outline was treated as a successful return to the nest.

A depiction of a successful run is shown in Figure 5 (see supplemental Video 4). The evolved algorithm enabled the robot to successfully navigate southeast from the nest until it was position below the field of resources. At that point, the robot travels north, 'consuming' a resource along the way, until it has left the resource field. The robot reorients and travels southwest until it returns to the nest. The high-level navigation behavior was based completely on the current surroundings, demonstrating robustness to disturbance and noise. For example, when we picked up the robot and moved it into the center of the resources, it reoriented and proceeded back to the nest (see supplemental Video 5). Similarly, the robot was still able to navigate back to the nest when the resource towers were moved during a run (supplemental Video 5). The relative scale of the physical robot to the test environment is vastly different from the evolutionary environment, indicating that the algorithm exhibits a surprising robustness to changes in scale. We confirmed this by testing the robot in a larger, 15-foot square environment, increasing only the ultrasonic detection distance threshold and the length of unit movement (see supplemental Video 6).

4. CONCLUSION

We have shown that flexible hardware architectures, combined with dynamic environments can create pathways for the evolution of intelligent, responsive, and robust navigation strategies. Overall, the new Avida features provided a valuable and viable platform for (1) creating robust and intelligent behavioral rules for robots, (2) testing the efficacy of genetic architectures for favoring behavioral flexibility, and (3) evaluating the effectiveness of

environmental designs in evolving functionally and adaptively complex behaviors. Most importantly, in addition to resolving the posed navigation challenge, the strategies evolved here were highly congruous with their natural analogs: in multiple populations, dominant foraging strategies incorporated periods of directed travel, fixed pattern search, cue response, and recalibration and reorientation when outcomes did not match expected results (Fig. 1 & Fig. 2). Many readers who have ever lost a car in a parking lot may appreciate the value of this strategic combination.

Additionally, while many previous studies have also attempted to evolve navigation strategies *de novo*, the resulting algorithms have proved difficult to translate into physical robotic systems. This difficulty is often attributed to the so-called ‘reality gap’, wherein physical systems diverge from their approximations in simulation. In contrast, using the new Avida architectures developed here, because the evolved navigation behaviors were responsive to general rules about the environment and not fixed to the details of the environment, they were readily transferred to a simple robotic system, proving to be robust to reality gap issues of noise and scale commonly plaguing such attempts.

In natural systems, the most successful solutions are those that ensure survival, regardless of situational noise and variation in the environment. Indeed, evolution is a rich source of robust behaviors in natural systems. Developing reliable methods for such discovery of effective, deployable, and lightweight algorithms in robotics could revolutionize the field. Importantly, the approach demonstrated here reveals that open-ended evolution has the potential to achieve that goal when left unburdened by the biases of human perspective. In particular, considering the simplicity of the deployed hardware and evolved algorithms, our approach highlights the potential of evolving gene-based architectures to generate practical solutions, while also greatly simplifying software and hardware requirements, and associated costs.

5. ACKNOWLEDGMENTS

We thank Matt Allen for developing Avida-TV used in the extraction of the algorithm. We also thank Cliff Bohm for providing the facility used in testing and filming the robot in the large-scale environment. Finally, we would like to thank members of the Michigan State University Digital Evolution Laboratory for fruitful discussions and funding from the BEACON Center for the Study of Evolution in Action.

This material is based in part upon work supported by the National Science Foundation under Cooperative Agreement No. DBI-0939454. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

6. REFERENCES

- [1] Nelson AL, Barlow GJ, and Doitsidis L. 2009. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems* 57:345-370.
- [2] Elsberry WR, Grabowski LM, Ofria C & Pennock RT. 2009. Cockroaches, drunkards, and climbers: Modeling the evolution of simple movement strategies using digital organisms. *Proceedings of the IEEE Symposium on Artificial Life*, pp. 92-99.
- [3] Grabowski LM, Bryson DM, Dyer F, Pennock RT, Ofria C. 2010. Early evolution of memory usage in digital organisms. *Proceedings of the 12th International Conference on Artificial Life*. Odense, Denmark.
- [4] Clune J, Ofria C, and Pennock RT. 2007. Investigating the Emergence of Phenotypic Plasticity in Evolving Digital Organisms, *Proceedings of the 2007 European Conference on Artificial Life*, 4648:74-83.
- [5] Ofria C, Bryson DM, Wilke CO. 2009. Avida: A software platform for research in computational evolutionary biology. In: Adamatzky A, Komosinski M, editors. *Artificial Life Models in Software*. London, UK: Springer Verlag, London, UK. Second edition.
- [6] Grabowski LM, Bryson DM, Dyer FC, Pennock RT, and Ofria C. 2011. Clever creatures: Case studies of evolved digital organisms. *Proceedings of the 2011 European Conference on Artificial Life*.
- [7] Beckmann BE, McKinley PK, Knoester DB, and Ofria C. 2007. Evolution of cooperative information gathering in self-replicating digital organisms. *Proceedings of the First IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. Boston, Massachusetts, pp. 65-76.
- [8] Knoester DB, McKinley PK, Beckmann B, and Ofria C. 2007. Directed evolution of communication and cooperation in digital organisms, *Lecture Notes in Computer Science (Proceedings of the 2007 European Conference on Artificial Life)* 4648:384-394.
- [9] Goldsby HJ, Dornhaus A, Kerr B, and Ofria C. 2013. Task-switching costs promote the evolution of division of labor and shifts in individuality. *Proceedings of the National Academy of Science*.
- [10] Beckmann B, Grabowski L, McKinley PK, and Ofria C. 2008. Autonomic software development methodology based on Darwinian evolution (Abstract), *Proceedings of the 5th IEEE International Conference on Autonomic Computing*. Chicago IL, June 2008, pp. 203-204.
- [11] Grabowski LM, Elsberry WR, Pennock RT, and Ofria C. 2008. On the evolution of motility and intelligent tactic response. *Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO-2008)*. Atlanta GA, pp. 209-216.
- [12] Knoester DB, McKinley PK, and Ofria C. 2008. Cooperative network construction using digital germlines, *Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO-2008)*. Atlanta GA, July 2008, pp. 217-224.
- [13] Beckmann BE & McKinley PK. 2008. Evolution of adaptive population control in multi-agent systems. *Proceedings of the Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. Venice, Italy, pp. 181-190.
- [14] Bryson DM. 2012. The evolutionary potential of populations on complex fitness landscapes. Michigan State University. Ph.D Thesis.
- [15] Bryson DM and Ofria C. 2012. Digital evolution exhibits surprising robustness to poor design decisions. *Artificial Life 13: Proceedings of the 13th International Conference on the Simulation and Synthesis of Living Systems*. July 2012, East

- Lansing, MI, USA, pages 19-26. MIT Press, Cambridge, MA.
- [16] Bryson DM, Ofria C. 2013. Understanding evolution potential in virtual CPU architectures. *PLoS ONE* 8: e83242.
  - [17] Fortuna MA, Zaman L, Wagner AP, Ofria C. 2013. Evolving Digital Ecological Networks. *PLoS Comput Biol* 9: e1002928.
  - [18] Wagner AP, Zaman L, Dworkin I, Ofria C. 2013. Behavioral strategy chases promote the evolution of prey intelligence. *arXiv:1310.1369*.
  - [19] Lehmann K, Goldman BW, Dworkin I, Bryson DM, Wagner AP. 2013. From cues to signals: evolution of interspecific communication via aposematism and mimicry in a predator-prey system. *arXiv:1309.2852*.
  - [20] Orians, G. H , and N. E. Pearson. 1979. On the theory of central place foraging. Pages 155-177 in D. J. Horn, R. D. Mitchell, and G. R. Stairs, eds. *Analysis of ecological systems*. Ohio State University Press, Columbus.