

Hardware Evolution of Analog Circuits for In-situ Robotic Fault-Recovery

Dmitry Berenson Nicolás Estévez Hod Lipson
Cornell Computational Synthesis Lab
Cornell University, Ithaca, New York 14853
Email:[ddb25| nse5| hl274]@cornell.edu

Abstract

We present a method for evolving and implementing artificial neural networks (ANNs) on Field Programmable Analog Arrays (FPAAs). These FPAAs offer the small size and low power usage desirable for space applications. We use two cascaded FPAAs to create a two layer ANN. Then, starting from a population of random settings for the network, we are able to evolve an effective controller for several different robot morphologies. We demonstrate the effectiveness of our method by evolving two types of ANN controllers: one for biped locomotion and one for restoration of mobility to a damaged quadruped. Both robots exhibit non-linear properties, making them difficult to control. All candidate controllers are evaluated in hardware; no simulation is used.

1. Introduction

Fault-tolerant systems have long been sought after for applications where it is either impossible or impractical to manually repair a deployed device. Imagine if a robot deployed in a remote location suffered a fault—for instance if some of its sensors or motors stopped functioning. Attempting to diagnose and work around the fault via radio communication could take too long and may be impossible. Abandoning the robot without attempting to recover some functionality would be a waste. Thus an on-board fault-recovery system is required to attempt to compensate for such faults. Exploring the creation of such a system, Mahdavi and Bentley [1] evolved controllers on a snake-like robot, and then damaged the robot and continued evolution. Their system demonstrated evolved recovery from damage, but the controller was open-loop and used digital circuits: our system differs in that we evolve sensor-based ANNs in analog hardware.

Greenwood et al [2] describe an evolvable hardware approach that performs fault recovery in linear systems on an FPAA. However, many real-world systems (especially robots) are non-linear, necessitating a more robust method of control and fault tolerance. While evolving controllers on fine-grained evolvable

hardware platforms (such as the FPTA [7] or possibly an FPGA [8, 9]) may seem appealing for this application, the search space for such evolution is extremely large and the likelihood of finding an acceptable solution in a reasonable number of in-robot evaluations is small.

Artificial neural networks (ANNs) have proven to be an invaluable tool for complex control applications. Evolving ANNs has proven to be an effective technique for finding desirable solutions to difficult problems (sorting, control, pattern recognition, classification, etc.). However, efficiently implementing reconfigurable ANNs in hardware is still under investigation. Roggen et al [4] report a successful implementation of spiking neural networks on an FPGA. While their work is promising, an analog solution is desirable because of the need to satisfy power and space constraints in space-exploration applications. Also, an analog implementation would virtually eliminate the need for interfacing the system with the analog world of sensors, a task which requires extra resources. Implementing ANNs on microprocessor platforms presents many of the same problems.

To that end, researchers have investigated the implementation of reconfigurable ANNs using analog devices. Amaral et al [3] have demonstrated the evolution of neuron circuits on a programmable analog multiplexer array (PAMA-NG). Schürman et al [5] have developed a system of Application Specific Integrated Circuits (ASICs) that can be configured into an ANN. However, such previous research has focused on the evolution of neurons (or neuron-like structures) while the evolution of analog ANNs in hardware has remained relatively unexplored.

This paper describes an effective method for the implementation of hardware neurons and the evolution of hardware ANNs. We apply our evolutionary hardware method to evolve two types of controllers: one for biped locomotion and the other for restoration of mobility to an extremely damaged quadruped. Both robots exhibit non-linear properties, making them difficult to control. Several different damage scenarios are considered for the quadruped.

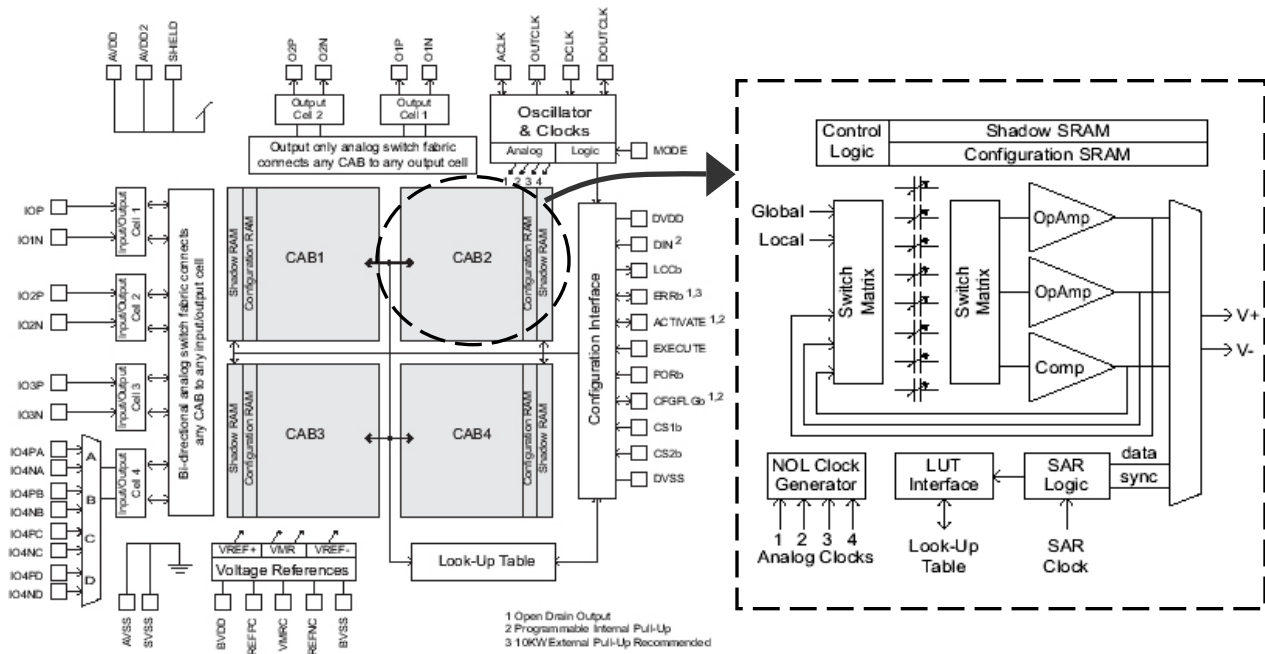


Figure 1: Left: Schematic of the AN221E04 FPAAs. Right: Inside a Configurable Analog Block (CAB).

2. Methods

2.1 The FPAAs

The Anadigm® AN221E04 is a reconfigurable analog device based on switched capacitor technology. See figure 1 for a schematic of the chip. Different circuit configurations are achieved through manipulation of switches between various circuit components inside four configurable analog blocks (CABs). Each CAB contains two op-amps, a comparator, and 8 variable capacitors (see figure 1).

The chip has two dedicated output cells and four bidirectional cells which can be used as either an input or an output. The chip’s dimensions are 13.2mm x 13.2mm x 2.45mm.

Anadigm Designer 2 software controls the configuration inside the AN221E04 through a serial connection to the development board. By placing and routing various Configurable Analog Modules (CAMs) in the software, the user can create a functional circuit. The software also contains an API that allows the design interface to be manipulated from an external program.

2.2 Creating a Hardware Neuron

Because the Anadigm Designer software only allows block-level circuit implementations, it was necessary to construct a hardware ANN using the

CAMs made available by Anadigm. Thus it was necessary to approximate the function of a neuron using two parts: a weighted summing module and an activation function module. The weighted sum was easily facilitated by the “SumDiff” CAM, which can take up to four inputs, multiply each by its own constant, and output the sum. For the activation function, the “integrator” CAM was used because of its ability to act as a thresholding function. When the input into the integrator rises above 2 volts, the integrator quickly saturates and outputs a 4V signal. When the voltage drops below 2V, the integrator’s output quickly goes to 0V. The speed of change of the integrator’s output (integration constant) is variable and, in our experiment, was one of the settings determined by evolution. One may also invert the output signal of the integrator, as well as any of the inputs into the sum block. See figure 2.

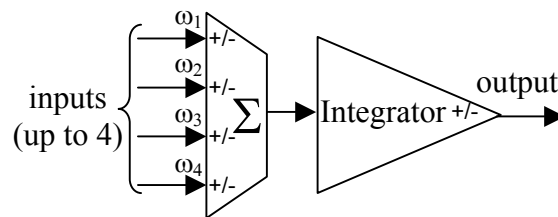


Figure 2: The block-level hardware implementation of a neuron.

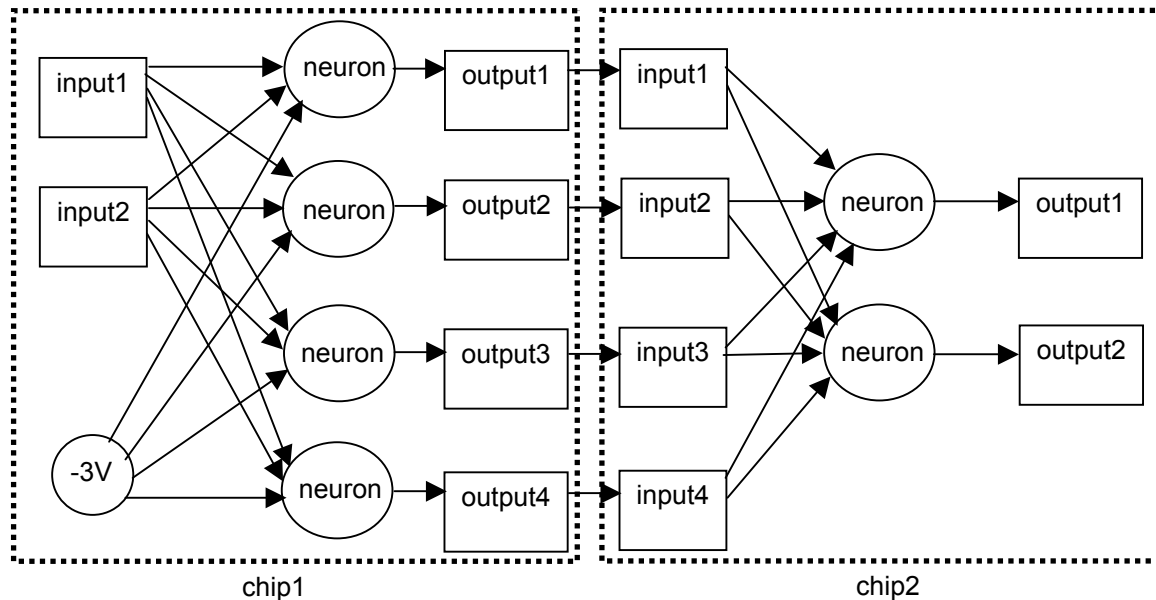


Figure 3: Implementation of a two-layer ANN on two AN221E04s.

Since the AN221E04 works around an analog clock, analog “calculations” are performed at a certain clock frequency. Additionally, the values of all settings (i.e. the weights on each input and the integration constant) are limited to a range of values determined by the clock frequency. Because the AN221E04 uses switched capacitor technology, perfect granularity between these values is not possible. However, values can be adjusted by a minimum of 0.01 units, which was acceptable for our needs.

2.3 Creating a Hardware ANN

The major limiting factor in creating a hardware ANN on the AN221E04 is the on-chip resources. For instance, there are a limited number of channels to and from the input and output cells. Routing two inputs to each of the four CABs is impossible. Thus, it was necessary to place two sum blocks in each of two CABs and to place all the integrators in the remaining two CABs. However, in order to accommodate two sum blocks in a single CAB, the number of inputs into each sum block had to be reduced from four to three. Thus, because of hardware restrictions, we found that one AN221E04 could hold a maximum of four neurons with three inputs each.

The ANN we constructed had two inputs and two outputs (see figure 3). It consisted of two layers: a four neuron hidden layer and a two neuron output layer. In order to accommodate enough neurons, two development kits were cascaded and their FPAAAs interconnected.

A constant -3V source CAM was placed on the first chip and connected to the hidden layer neurons to effectively shift the level of each neuron’s threshold. By evolving the weights of the neuron inputs corresponding to the -3V source, we were able to evolve a unique threshold for each neuron.

A two layer ANN was used because it was the closest approximation to a three-layer network, which has proven effective in robot-control applications [6]; the hidden and output layers are sufficient to allow nonlinear transformation of the sensor signals into motor commands.

According to the Anadigm Designer 2’s simulator, the neural network circuit described here consumed 0.625 +/- 0.121 Watts of power.

2.4 The Algorithm

A standard evolutionary algorithm was used to evolve the weights of the inputs, integration constants, and “polarity” for each neuron. “Polarity” is used here to indicate whether the output of the integrator is inverted or non-inverted.

The algorithm used elitism selection (top 50% preserved into next generation), a population size of 32 individuals, crossover, and mutation. An individual consisted of six genes, one for each neuron. The gene structure is diagrammed in figure 4. The weights (ω) and integration constant (I_c) were floating point values. Polarity (P) was a binary value.

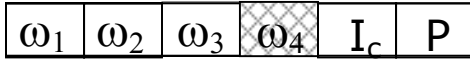


Figure 4: Diagram of a gene used in our algorithm. ω_4 is only used for the output-layer neurons. I_c is the integration constant and P is the polarity of the integrator (inverting or non-inverting).

The procedure of the algorithm was as follows:

1. Generate a random population of individuals.
2. Program the chips with an individual from the population.
3. Evaluate the individual's fitness.
4. Go to step 2, repeat until entire population has been evaluated.
5. Select the top 50% of the population as parents.
6. Mate the parents using two-point crossover (probability = 1) and produce two children for each two parents.
7. Mutate the children (probability = 0.1 for each locus).
8. Replace the bottom 50% of the population with the children.
9. Go to step 2 (if desired generation has not been reached).

Crossover operated on a gene level; genes were never split. We used “creep” mutation, which shifts the target value by +/- 0.15, as opposed to standard mutation which replaces the target value with a random value. The polarity was never mutated because we believed crossover could sufficiently explore this property.

To decrease the number of hardware trials, the algorithm never re-evaluated the fitness of an individual.

Though our algorithm could have evolved the connections between the neurons and the inputs/outputs, we chose to fix the topology. Greenwood et al. [1] note that, in controls applications with complex circuitry, it is probably best to fix the configuration and only evolve the settings values. This is due to the exponential increase of the search space if topology is also evolved.

2.5 The Robots

To investigate the robustness of our method, we evolved controllers for two robots with different morphologies.

2.5.1. The Biped. The biped robot (called the “Toddler”) was assembled using a kit purchased from

Parallax® (see figure 5). To keep the robot from falling forward or backward, two metal rods were inserted through the feet. However, the robot was still able to fall on its sides.

To give the ANN information about the robot's state, a Fairchild QRB1134 optical sensor was placed at the back of each foot and connected to the ANN's inputs. These sensors output a voltage proportional to the foot's distance from the ground. Both sensors were biased to produce a voltage between 0V (corresponding to 0cm above the ground) and 2V (corresponding to 2cm or more above the ground).

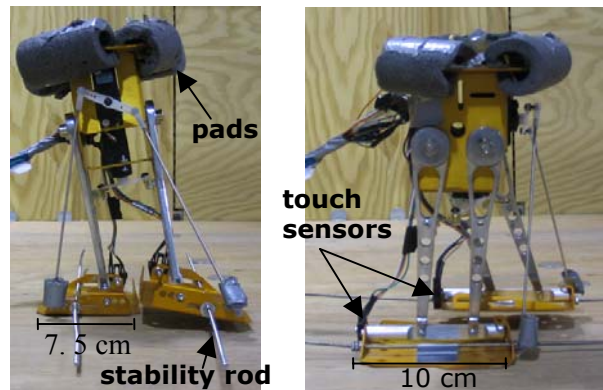


Figure 5: The “Toddler” bipedal robot with pads, stability rods, and touch sensors. Left: front view. Right: side view. Note: this is the initial state all candidate solutions started from.

The robot's motion is controlled by two servo motors. One servo controls the position of the feet relative to each other and the other servo controls the tilt of the feet (see figure 6).

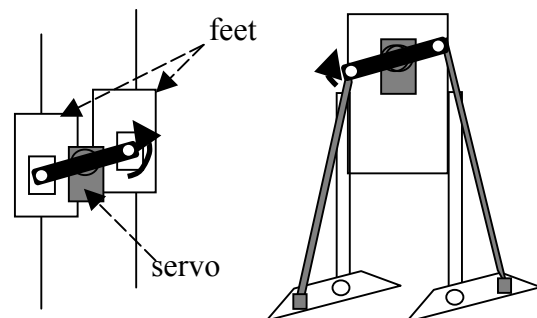


Figure 6: Illustration of the “Toddler.” Left: Servo controlling the position of the feet relative to each other. Right: Servo controlling the tilt of the feet.

2.5.2. The Quadruped. The quadruped robot was built from printed plastic and servo motors (used to create

motorized joints). See figure 7 for an idealized computer model of the quadruped and figure 8 for an illustration of one of the quadruped's four identical legs. The idealized computer model is only shown for illustrative purposes; no simulation was used in these experiments.

To implement the touch sensor, a Fairchild QRB1134 optical sensor was placed on the foot. This sensor output a voltage proportional to the foot's distance from the ground. Unavoidably, the sensor also responded to the angle at which the foot was canted. To implement the angle sensor, a QRB1134 was placed near the hip joint. This sensor output a voltage proportional to the distance between the thigh and shank. Both sensors were biased to produce a voltage between 0V and 2V. Through several preliminary experiments, it was found that candidate solutions were generally more productive when the signal from the angle sensor was inverted at the input of every hidden layer neuron, thus this setting was fixed.

The quadruped was constructed from plastic parts printed on a Stratasys rapid prototyping "3-D printer." Since the robot was a prototype, there were some imperfections which hindered the development of a controller. First, the plastic parts were not well attached at the joints; the parts could rotate from side to side by a few degrees. Such a shift would change the angle at which the foot touched the ground, thus changing the friction between that foot and the ground and the direction the robot would move when pushing off this leg.

Second, all legs had spring-mounted plastic pegs connected at the foot. These pegs would retract a distance proportional to how much force was applied to them. The pegs made the robot wiggle up and down as it moved and resist certain kinds of motion because of the spring force. These factors (side to side leg rotation, up and down wiggle, and resistance to certain motion) made it very difficult to predict the effects of changing servo positions. However, these factors also make the finding of an adequate controller all the more enticing. By evolving a controller for this non-linear system we show that our method can be applied to real world systems containing real-world imperfections without the use of modeling or system-identification.

2.6 Breaking the Quadruped

The faults induced in the quadruped for our experiments were intentionally extreme. First, the robot's motion and sensing was restricted to a single leg (henceforth referred to as the "active leg"). Servos on all other legs were locked in the leg-at-maximum-extension position shown in figure 8. Second, the shank of the leg opposite the active leg was removed.

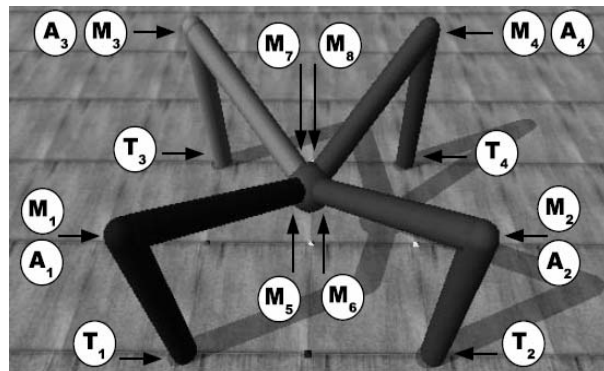


Figure 7: Idealized computer model of the quadruped robot. T_i indicates touch sensors; A_i indicates angle sensors; and M_i indicates motorized joints.

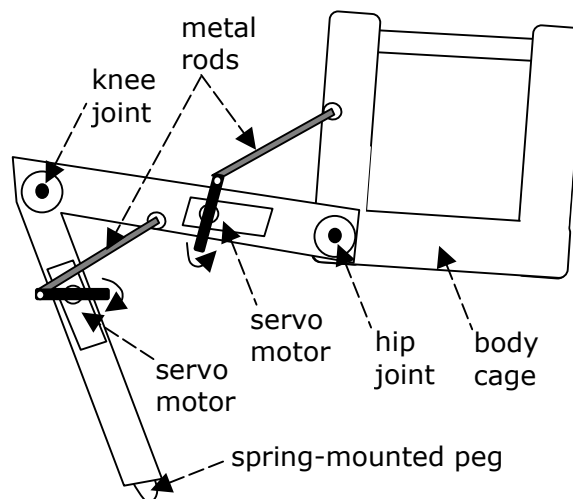


Figure 8: Illustration of one of the robot's real legs. Motion is achieved by turning servo motors connected to metal rods. The leg is shown here at full extension. Other legs are identical.

This configuration was intentionally unstable for certain conditions: in certain situations, the robot would tip over in the direction of the detached shank and would not be able to recover motion. We term this irrecoverable situation a "dead state" (see figure 9).

In order to make each candidate evaluation as fair as possible, we reset the robot to the same initial state before beginning each trial. The configuration of the leg we chose for the initial state was maximum contraction at the knee joint and maximum extension at the hip joint (see figure 9).

The initial state posed a significant challenge for the evolving controllers. An adequate controller would need to escape this state before it could begin the periodic motion necessary for movement. Another

difficulty was the “false” signal coming from the touch sensor. Since the lower part of the active leg lies parallel to the ground, the foot’s touch sensor is pointed parallel to the ground. Thus the sensor outputs a signal which corresponds to maximum distance from ground when the foot is in-fact touching the ground. An adequate controller must compensate for this initial sensor error as well.

Also, a piece of sandpaper was attached to the foot of the active leg and the plastic peg was disabled to give that leg enough friction to move the robot forward.

2.7 The Setup

See figure 10 for a flow chart of our setup. A PC programs the two AN221E04 FPAA containing the ANN through a serial connection. Since the servo motors on our robots required a pulse width modulated (PWM) signal, we could not feed the outputs of the ANN directly into the servo motors. It was necessary to connect these outputs to an Atmel Mega32 microcontroller, which performed a simple threshold function: if the input signal is below 2V, output a PWM pulse corresponding to the zero-position of the servo; if the signal is above 2V, output a PWM signal corresponding to the maximum (180°) rotation of the servo. For the biped, we restricted the servos’ movement range to +/- 45 degrees from center because the robot was not designed for rotations outside of this range.

3. Experiments

3.1 Biped Locomotion

This experiment aimed to evolve a gait for the “Toddler” biped shown in figure 5. The biped was placed on a plywood surface and evolution was run for eight generations. Each candidate solution was downloaded onto the robot and allowed to run for 10 seconds, starting from the initial state. Fitness values were assigned as shown in table 1. To minimize the effects of evaluation noise, each candidate solution that showed >5cm of forward movement was run three times. The fitness was the lowest distance traveled of the three runs.

Two runs of the algorithm were conducted; the results are shown in figure 11. Distance was originally measured in inches but all values have been converted to centimeters.

The best gaits of both runs would move via a repetitive back and forth rotation of the servo controlling the position of the legs. This caused the robot to sway back and forth, moving forward in the process.

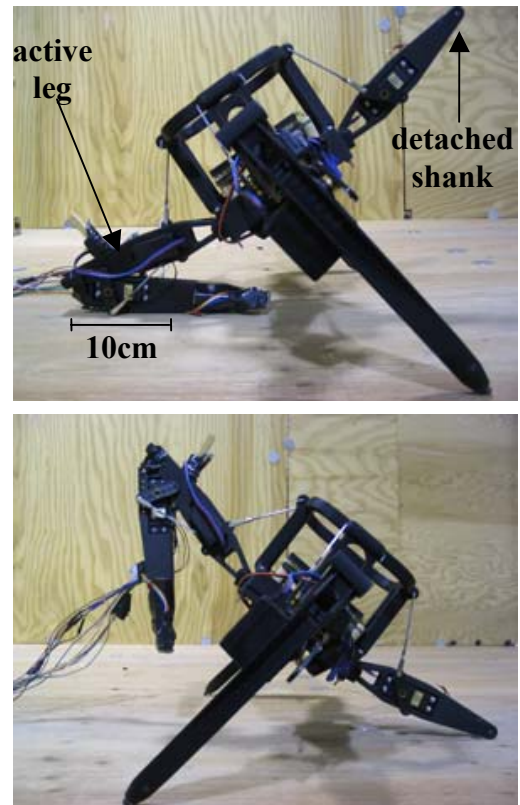


Figure 9: Top: All trials started from this initial state. Bottom: A dead state.

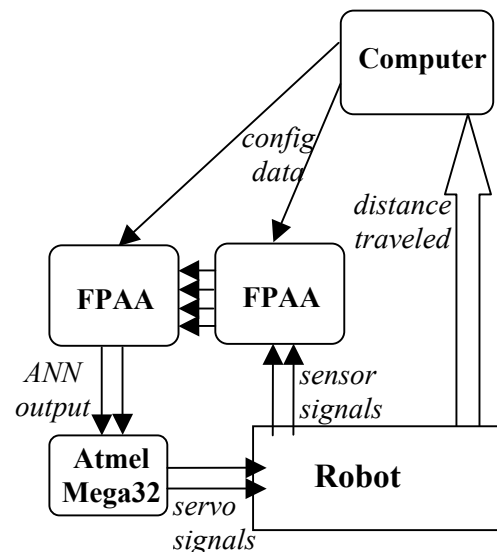


Figure 10: Flow chart of our setup. The computer programs the ANN (contained on the FPAA) with the candidate solution. The Atmel microcontroller converts the ANN outputs to servo-compatible signals. The distance traveled by the robot is input into the computer as the candidate’s fitness.

Table 1: Description of fitness values for all experiments.

Fitness	Behavior
0	No movement or robot fell over
2.5	Robot stopped moving before run was complete.
5	<5cm forward movement but robot did not stop moving before evaluation was complete.
>5	Forward distance traveled (rounded to nearest 2.5 cm)

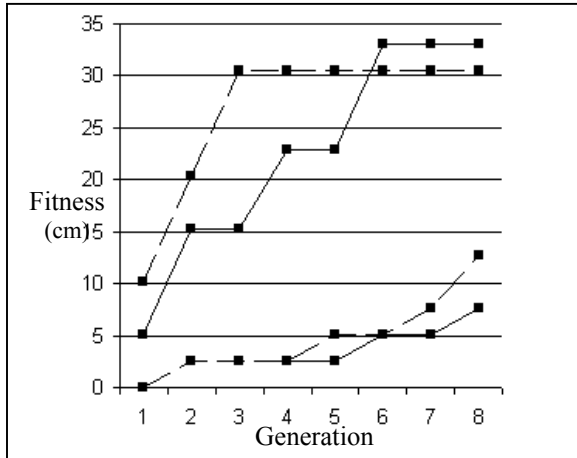


Figure 11: Plot of fitnesses from two runs of the biped locomotion experiment. Higher lines, best; lower lines, average. Solid, run 1; dashed, run 2.

3.2 Quadruped Fault Recovery

3.2.1. Experiment I: One Shank Missing. In this experiment the quadruped shown in figure 9 was placed on a ply-wood surface and evolution was run for seven generations. Each candidate solution was downloaded onto the robot and allowed to run for 10 seconds. Results are shown in figure 12.

The best evolved gait can be described as a sequence of the following three modes of movement (see figure 13):

1st mode: the robot rose from the initial state.

2nd mode: the robot moved its active leg back and forth rapidly, making no progress.

3rd mode: the robot moved forward by repeatedly leaning on the active leg and pushing off it.

Individuals with fitness greater than 40cm exhibited this same sequence of progression. However, individuals with higher fitnesses tended to enter the third mode faster than those with lower fitness.

It is important to note that running the same candidate solution does not always produce the same distance traveled; fitnesses for the same individual usually fell into a 15cm range.

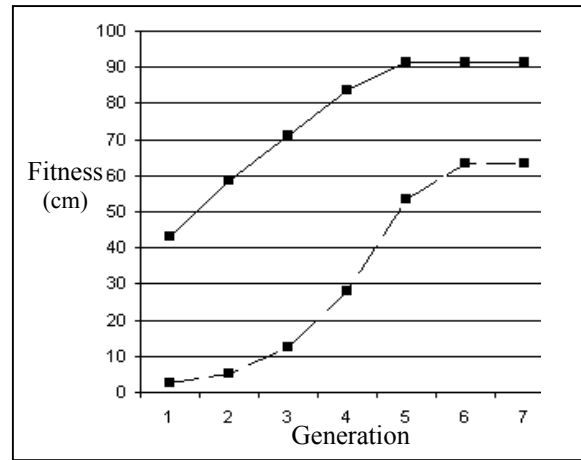


Figure 12: Fitness plot for experiment I. Solid, best fitness; dashed, average fitness.

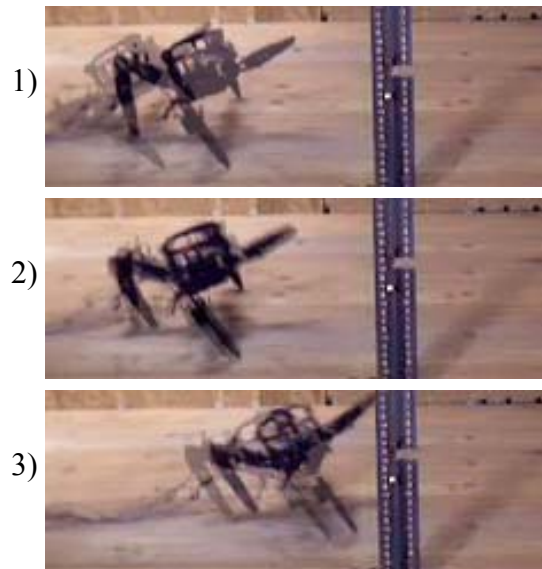


Figure 13: Time-lapse (1 second difference in each image) depiction of the three modes of movements of high fitness gaits.

3.2.2. Experiment II – Pendulum. This experiment was the same as experiment one except that a pendulum of length 5cm and weight 73g was attached where the detached shank would have been. This situation was intended to simulate a fault scenario where the robot's shank had broken but was still partially attached to the thigh. This was also a more difficult fault to compensate for because of the vibration introduced by the pendulum.

To minimize evaluation noise, each candidate solution that showed >5cm of forward movement was run three times and the lowest distance traveled was taken as the solution's fitness.

Solutions evolved from best fitness of 27cm and average fitness of 2cm to a best fitness of 46cm and an average fitness of 27cm over five generations.

Higher fitness individuals exhibited the same general modes of motion as described in experiment I. However, individuals took longer to escape the second mode and their third mode was less productive. Individuals also slipped back into the second mode from the third mode more often.

3.2.3. Other Experiments. To investigate fault scenarios involving an unexpected environmental change, we re-ran experiment II using high-friction sandpaper and low-friction plastic as the surface the quadruped moved on. In the sandpaper case, our method was not able to evolve a controller that made forward progress without falling over.

In the plastic case, the solutions evolved from a best fitness of 2.5cm of forward movement to 10cm of forward movement in seven generations. However, most forward motion came from the robot rising from the initial state and not from repetitive motion.

4. Discussion and Conclusion

Through experiments on the biped and quadruped, we have shown that our evolvable hardware method can be applied to radically different robot morphologies, even those exhibiting non-linear properties. We have also demonstrated that our system was capable of recovering mobility in several extreme fault scenarios on the quadruped.

However, the system was not able to recover mobility in the presence of extreme friction conditions (section 3.2.3). These conditions necessitate very specialized controllers; any deviation from these controllers will cause the robot to fall over or not move. Since our algorithm converges quickly and does not search a very broad solution space, it is not surprising that these controllers were not found. Increasing the population size and varying the selection method would allow us to search a larger solution space. However, such changes would necessitate an unrealistic number of hardware evaluations. This highlights one of the primary challenges of in-situ evolution: the tradeoff between a manageable number of hardware evaluations and searching a larger solution space.

In further work, it would be interesting to explore the creation of a larger network of FPAAs and implications for scalability.

Finally, we plan to pursue hybrid methods in which in-situ hardware evolution is combined with automated simulation refinement and the use of the resulting simulations for off-board controller evolution.

5. Acknowledgements

The authors thank Josh Bongard and Chandana Paul for their creativity and support. This research was supported in part by the NASA program for Research in Intelligent Systems, grant number NNA04CL10A.

6. References

- [1] S. H. Mahdavi and P. J. Bentley, **An evolutionary approach to damage recovery of robot motion with muscles**, Seventh European Conference on Artificial Life (ECAL03), 2003, pp. 248-255
- [2] G. Greenwood, D. Hunter, and E. Ramsden, **Fault Recovery in Linear Systems via Intrinsic Evolution**, Proc. 2004 NASA/DOD Conference on Evolvable Hardware, June 2004, pp.115-122
- [3] J.F.M. Amaral, J.L.M. Amaral, C. Santini, R. Tanscheit, M. Vellasco, and M. Pacheco, **Towards Evolvable Analog Artificial Neural Network Controllers**, Proc. 2004 NASA/DOD Conference on Evolvable Hardware, June 2004, pp.46-51
- [4] D. Roggen, S. Hofmann, Y. Thoma, and D. Floreano, **Hardware Spiking Neural Network with Run-Time Reconfigurable Connectivity in an Autonomous Robot**, Proc. 2003 NASA/DOD Conference on Evolvable Hardware, July 2003, pp.189-198
- [5] F. Schürmann, S. Hohmann, J. Schemmel, K. Meier, **Towards an Artificial Neural Network Framework**, Proc. 2002 NASA/DOD Conference on Evolvable Hardware, July 2002, pp.266-273
- [6] J. Bongard and H. Lipson, **Automated Robot Function Recovery after Unanticipated Failure or Environmental Change using a Minimum of Hardware Trials**, Proc. 2004 NASA/DOD Conference on Evolvable Hardware, June 2004, pp. 169-176
- [7] A. Stoica, D. Keymeulen, A. Thakoor, T. Daud, G. Klimech, Y. Jin, R. Tawel, V. Duong, **Evolution of Analog Circuits on Field Programmable Transistor Arrays. Trials**, Proc. 2000 NASA/DOD Workshop on Evolvable Hardware, July 2000, pp. 99-108
- [8] A. Thompson, **Exploring Beyond the Scope of Human Design: Automatic generation of FPGA configurations through artificial evolution**. 8th Annual Advanced PLD & FPGA Conference, 1998. pp. 5-8
- [9] J. Lohn, G. Larchev, and R. DeMara, **Evolutionary Fault Recovery in a Virtex FPGA Using a Representation That Incorporates Routing**, 10th Reconfigurable Architectures Workshop, April 2003.