by Akshay Dhawan and Sergio Biagioni (USA)

MCU-Based Altitude Control

An IR Protocol for Helicopter Commands

This article describes a microcontroller-based altitude controller for a model helicopter. The system uses an IR protocol that sends commands to the helicopter, a phototransistor network that measures altitude, a boom setup that constrains helicopter movement, and a spiking neural network that monitors sensory inputs and output throttle values.

To wirelessly control a model helicopter's altitude and hovering capabilities, we designed a device that uses infrared communication and a simple genetic algorithm implementation. This project culminated from "ECE 4760: Designing with Microcontrollers," which is a one-semester design course taught by Cornell University professor Bruce Land. The project combined our interests in control systems and machine learning and demonstrated our acquired knowledge of microcontroller, hardware, and software design.

Using wireless control, we wanted the helicopter to quickly rise to a given altitude and steadily hover at that position. Another goal was to have the helicopter perfect this task in as few trials as possible. We initially intended to fly the helicopter without any constrictions, but we found it difficult to develop control for all degrees of freedom within a short amount of time and with limited resources. Additionally, the helicopter became unbalanced after we attached an IR LED for distance measurement. Therefore, we attached the helicopter to a boom and restricted flight to one degree of freedom (DOF). The IR LED attached to the helicopter constantly emitted a signal and was sensed by a phototransistor network placed on the ground under the boom. We used a neural network to calculate the helicopter's throttle based on distance measured by the phototransistors. We used a genetic algorithm to evolve the network until we achieved a quick rise and steady hover.

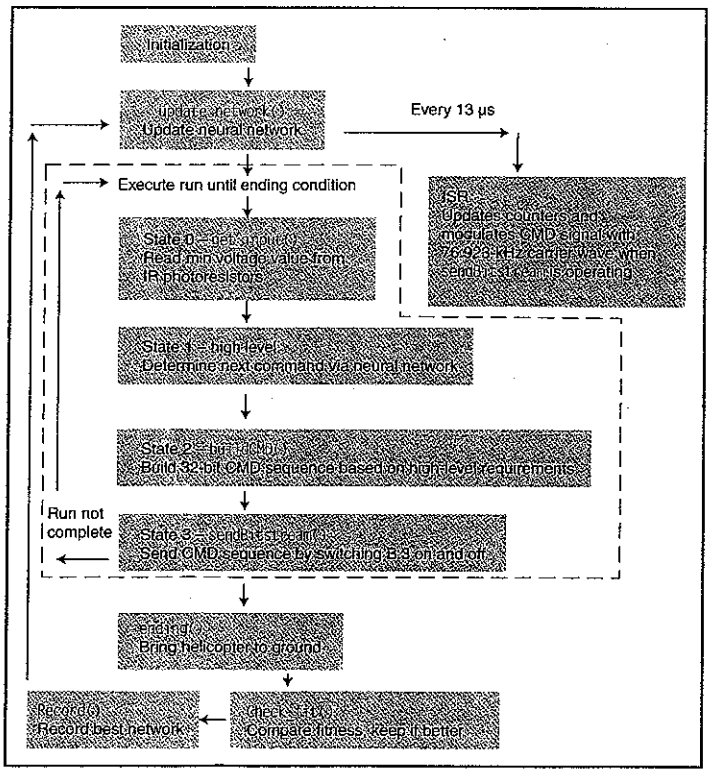

Figure 1—This is the program's high-level functionality. The red box outlines a single "run." The ISR occurs every 13 µs, even in the middle of a run.

SYSTEM DESIGN

The learning process was implemented as a series of 10-s "runs." During each run, the helicopter started on the ground, rose at a speed determined by the neural network and sent by an IR LED, safely dropped, and landed on the ground. The particular neural network used for the run was evaluated at the end of a run. If the run was considered good, the network parameters were stored in the system for further evolution and a different network was uploaded for the

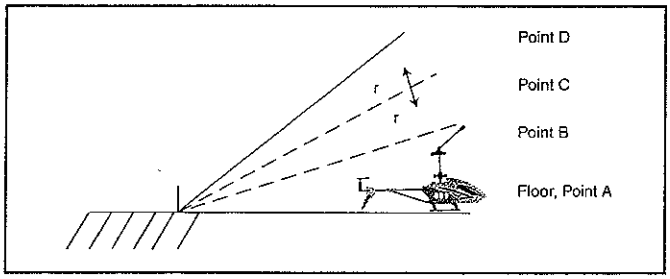

Figure 2—The experimental setup is shown as a series of points. Starting at Point A, the experiment's goal is to fly the helicopter to Point C in the shortest time and to achieve a steady hover around Point C of distance Δr.

next run. Runs were executed until we were satisfied with the helicopter's speed and hover. Figure 1 shows the run structure breakdown.

The microcontroller learned how to fly the helicopter by experimenting with different neural network configurations in an evolutionary manner. Figure 2 shows the experiment's setup. Because of its balance of lightness and rigidity, we used balsa wood to attach the helicopter to a wooden boom. Ground position is referred to as Point A. The goal was to fly the helicopter to Point C in the shortest amount of time and to have the helicopter steadily hover. Point B is a midpoint and Point D is the maximum height the helicopter can achieve before the microcontroller shut down the current run. The maximum point is important because the helicopter loses stability past this point and shoots backward.

A run began with the helicopter at Point A. Every 120 µs, the helicopter's height was calculated through phototransistor readings and the microcontroller emitted a new throttle value. Throttle values were calculated based on neural-network spiking. This will be further described in the software section. The particular network was assigned a fitness value based on how well it accomplished the goal. The network's fitness value started at 0 for a particular run and accumulated every 120 µs depending on the helicopter's height. The helicopter accumulated higher fitness values as it got closer to Point C (i.e., the target point). The concept was to reward the network for quickly flying the helicopter to Point C and for keeping the helicopter closely hovering around Point C. Since every run is capped at 10 s, highest fitness values went to networks that quickly and steadily flew the helicopter to Point C. The exact fitness function used for our experiments is:

$$\text{Fitness} = \begin{cases} +0 \text{ Point A to Point B} \\ +1 \text{ Point B to Point } \frac{3B}{2} \\ +2 \frac{3B}{2} \text{ to C} - 2\Delta r \\ +10\ C \pm 2\Delta r \\ +20\ C \pm \Delta r \\ -5 > C + 2\Delta r \\ 0 + \text{end Point D} \end{cases}$$

Since the microcontroller couldn't directly read the helicopter's height, we used a look-up table to calculate the height based on phototransistor readings. The IR LED attached to the helicopter constantly emitted a signal that was picked up by a phototransistor network located directly underneath the boom. When the LED was directly on top of the phototransistor, the voltage was close to 0 V. As the helicopter travelled farther away from the ground, the voltage increased up to a maximum of V_{CC} or 5 V for our circuit. By manually bringing the helicopter to points between Point A and Point D, voltages could be associated with heights.

One neural network was evaluated per run. After 10 s, the microcontroller initiated an ending sequence that brought the helicopter down from any position by gradually decreasing the throttle. Once the helicopter reached Point A, the microcontroller compared the current network's fitness to a six-network population. If the current network has a higher fitness than any network in the population, it replaced the weakest network. For the next run, a network was randomly chosen and randomly mutated in several areas that will be described in further detail in the software section.

HARDWARE

We used a Syma S107R5 helicopter attached to a boom made of balsa wood (see Photo 1). The boom was hinged to a small piece of wood. A thin needle was inserted through the hinge and the balsa wood to keep the boom in place. The wood was manually clamped down, but a mechanical clamp could be used to easily clamp it down. This system's purpose was to reduce the DOF to 1 degree. Due to time and financial constraints, we could not control the helicopter in 6 DOF. Using a boom enabled us to focus on altitude control and develop the learning algorithm.

We attached an IR LED to the helicopter to attain height information. The helicopter came equipped with a blue/red LED attached near the nose. We disassembled the frame and disconnected the LED. Then we connected a Lite-On Technology LTE-4208 IR LED in series with a current-limiting resistor. The on-board 4.2-V battery would provide too much current if directly connected across the LED. We connected a 100-Ω resistor. This would ideally result in 42 mA through the IR LED, which would provide great range. However, the helicopter would not register commands with this configuration because the LED and the on-board phototransistor (which reads remote-control commands) overloaded the battery. We increased the resistor to 330 Ω, which resulted in 12.7 mA through the IR LED. This reduced the range but corrected the

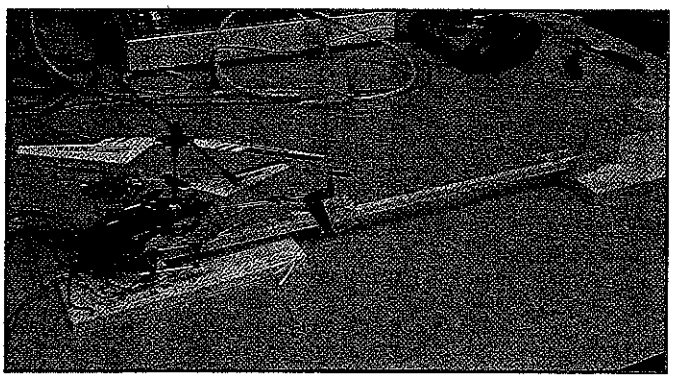

Photo 1—The helicopter is attached to a boom and placed on top of a whiteboard containing a phototransistor network. The command module is a small breadboard behind the wings. The microcontroller is offscreen.

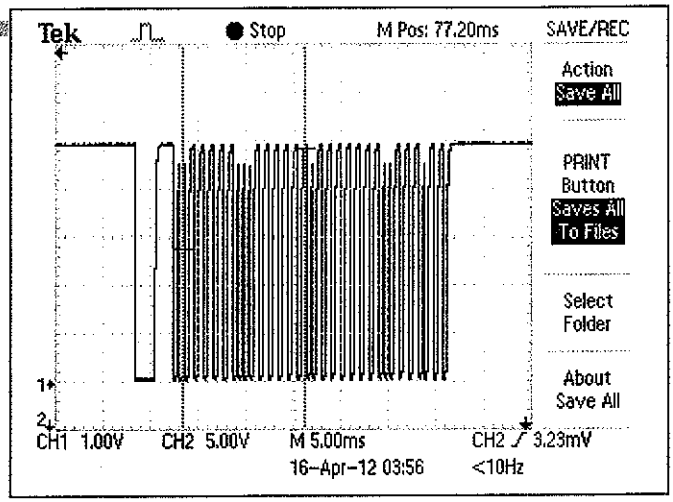

Photo 2—A single IR data packet contains two long header bits and 32 bits representing flight parameters. The shorter peaks are Off bits and the higher peaks are On bits.

helicopter operation. The IR LED's maximum readable distance, as read by a Lite-On Technology LTR-4206E phototransistor, was 25 cm. The minimum distance was 2 cm. We soldered the LED and resistor to the on-board battery's positive and negative leads.

To control the helicopter flight through the microcontroller, we reverse engineered the command protocol from the factory-supplied remote control with help from the *Couch Sprout* blog. We set up a single LTR-4206E to read the command from the remote control. We found commands were organized in packets spaced 120 ms apart. Each packet contained a header, 4 bytes of information, and a stop bit. Photo 2 shows a single packet on an oscilloscope screen. Moving forward, "low" refers to a low voltage, and "high" refers to a high voltage. The header consists of 2-ms low and 2-ms high. Next, each bit can take one of two values. A 1 is represented by 300-µs low and 700-µs high. A 0 is represented by 300-µs low and 300-µs high. A byte is composed of 8 bits. By changing one command at a time from the remote, we correlated specific commands to bytes. In order from first to last, the bytes represent yaw, pitch, throttle, and yaw correction. Full yaw to the right is represented by 0, while 127 represents full yaw to the left. Full pitch up is represented by 0, while 127 represents full pitch down. Throttle takes values from 0 to 127. The yaw correction takes the same values as yaw and applies extra yaw. At the end of 4 bytes, a 300-µs stop bit is issued.

Our test phototransistor could not detect a carrier wave that was issued before every bit. Every bit's low portion oscillates from 0 to 5 V at 76.973 kHz. If this carrier is not inserted, the helicopter's phototransistor and circuitry do not register the command and do not operate.

Our IR command includes three LTE-4208 IR LEDs in series with a 100-Ω, current-limiting resistor. We used three LEDs to increase signal intensity, angle, and range (see Figure 3). To send information at the same rate as the helicopter remote, we used an Atmel ATmega644 microcontroller's fast PWM feature, which enabled us to vary the voltage applied to the circuit from 0 to V_{CC}, resulting in fast IR pulses. We used timer-driven interrupts to send IR pulses in packets the helicopter could register. The on-board phototransistor and electronics enable the helicopter to decode the packets and translate them into flight parameters.

Our phototransistor network consisted of eight LTR-4206Es arranged on a whiteboard in a 2 × 4 rectangular fashion. Each phototransistor only had a 20° viewing angle. We were unable to attain better phototransistors due to time and financial constraints. To ensure the phototransistors could detect the IR from the helicopter at all times, the network was modified based on adjusted heights. Adjustment included phototransistor angling and network size. Each phototransistor was connected to a single ADC pin on the microcontroller that can measure a voltage from 0 to 5 V in 0.039-V increments. In addition, a 20-kΩ current-limiting resistor was placed in series with each phototransistor (see Figure 3).

EVOLUTIONARY ALGORITHM

According to Y. H. Said in her article, "On Genetic Algorithms and Their Applications," an evolutionary algorithm (EA) is a global optimization search that operates on the principles of natural selection.[1] The algorithm's goal is to find a solution that maximizes a problem's given fitness function. The EA starts by instantiating several solutions or chromosomes. Each of these is evaluated by a fitness function based on how well it solves a particular problem. Chromosomes are randomly mutated and retested. If they perform better than any chromosome in the current population, they replace the worst chromosome. After a round of mutations and checks, the best chromosomes reproduce and move on to the next generation. The EA performs several iterations until the search converges onto a solution and fitness values barely increase across generations.

We used an EA for this project to develop the best helicopter flight. We used an implementation first presented by D. Floreano, et al in their article, "Evolutionary Bits'n'Spikes."[2] This

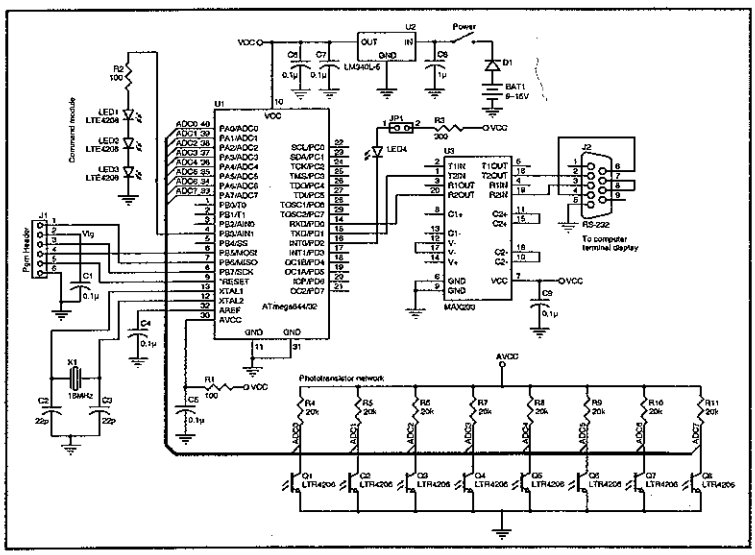

Figure 3—The IR command includes three Lite-On Technology LTE-4208 IR LEDs in series with a 100-Ω, current-limiting resistor. We used three LEDs to increase signal intensity, angle, and range. We placed a 20-kΩ, current-limiting resistor in series with each phototransistor.

algorithm uses a spiking neural network to take altitude information as sensory input and output throttle as motor output. Neurons can be simplistically described as single compartments with associated membrane voltages. Certain neurons are responsible for sensing input and relaying this information through a network of connections to terminal neurons that can determine motor output. The relay process occurs through "spikes" or extremely fast voltage swings, which occur when a neuron's membrane potential passes a threshold. The terminal neuron's membrane potentials are affected by presynaptic neurons (i.e., those that connect to the terminal neuron). The potential can increase if the presynaptic neuron is excitatory or decrease if the presynaptic neuron is inhibitory. The connections between presynaptic and terminal neurons can be configured to enable the motor output to react to sensory input as desired. This is the basic principle behind learning motor activity in biological systems.

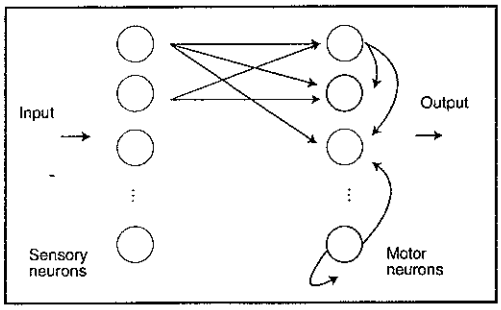

Figure 4—Here is a simple spiking neural network. Input is provided to the sensory neurons, which have connections to motor neurons. Motor neurons also have various connections and signs as shown by blue (excitatory) and black (inhibitory). Output is extracted from motor neuron spikes.

Our neural network includes three major components. First, a layer of sensory neurons spikes is based on altitude information. Second, these spikes are relayed through a network of connections. Third, the connections synapse onto a motor neurons layer. Some of the motor neurons also have connections to other motor neurons. Motor neurons spike based on the spiking through the network of connections. Output spikes define the helicopter's throttle output. Figure 4 shows an example of this system.

DIGITAL IMPLEMENTATION

You can use an 8-bit microcontroller to efficiently code the spiking neural network. A single neuron can be represented by a membrane potential, threshold, sign, spike output, and connections. A few attributes describe a neuron's behavior over time or cycles. A neuron is either spiking (represented by a 1) or not spiking (represented by a 0). Once it has spiked, a neuron cannot spike for one cycle. A neuron can synapse onto other neurons. A neuron's membrane potential can increase or decrease depending on spiking contributions from input neurons. The change direction is determined by the input neurons' sign. For example, if a neuron has three excitatory inputs and three inhibitory inputs and all inputs spike, then that neuron's membrane potential will increase by 1. A neuron can spike if its membrane potential rises above a threshold, which we set at 5 ± a random number between −2 and 2. The random number ensures the system does not fall into locked oscillations. Finally, a leakage of 1 is always subtracted from a neuron on every cycle. This models the leakage biological phenomenon.

Our project utilizes eight sensory and eight motor neurons. The ADC converts all voltages from the phototransistor to digital values and finds the minimum. It is expected that this voltage most accurately reflects the helicopter height. The others might not be in the helicopter's field of view. The minimum voltage is represented by an 8-bit number and each sensory neuron corresponds to one of those bits. Connections between a single-sensory neuron and all motor neurons are represented by a single byte. Each bit is a 1 for a connection or 0 for no connection. Therefore, connections between all sensory neurons and all motor neurons can be represented by 8 bytes. Motor neurons can also connect to each other. Another 8 bytes are used for these connections. Finally, one motor neuron output is used to increase throttle by a set amount and another neuron is used to decrease throttle. The other motor neurons are used for intermediate connections.

When the helicopter is close to the ground, all sensory neurons spike, which results in many motor output spikes. For the proper configuration, this will increase throttle. When the helicopter reaches peak height, fewer sensory neurons spike, which results in a throttle decrease. The proper network configuration will result in the best flight. However, this configuration depends on the experimental setup and noise factors. The entire neural network can be described in 17 bytes: 1 byte for motor neuron signs (sensory neurons are kept excitatory), 8 bytes for sensory connections, and another 8 bytes for motor connections.

The EA produces a random, initial population of six networks or chromosomes. These networks are tested, mutated, and reproduced through several algorithm iterations. Mutation is done in a random manner. A single bit of the sign byte and both connection bytes is toggled at a time. A single run (as described in the System Design section) tests a single network and produces a fitness value that evaluates that network against the current population. If the network is better than any population network, the superior network replaces the inferior one. Mutations are performed until the user is satisfied with the altitude control.

RANDOM NUMBERS

We used a 32-bit linear feedback shift register as a random-number generator. A 32-bit register was instantiated and bits 27 and 30 were XORed to produce bit 0. Then the register was shifted left, which produced a random number with a repeat time that lasted much longer than any experiment. This configuration produces high-quality uncorrelated random numbers.

IR PROTOCOL

Time-driven interrupts from the ATmega644 are used to toggle PB3 to the IR command module (as described in the Hardware section). For this project, yaw and pitch are kept constant. Throttle is taken from the spiking neural network's output. Every 120 µs, a 32-bit command is built. This command represents the yaw, pitch, throttle, and yaw correction, in that order. Interrupts occur at 100-µs periods then toggle V_{CC} according to the protocol presented in the Hardware section. Another 76.973-kHz interrupt is used to generate the carrier wave.

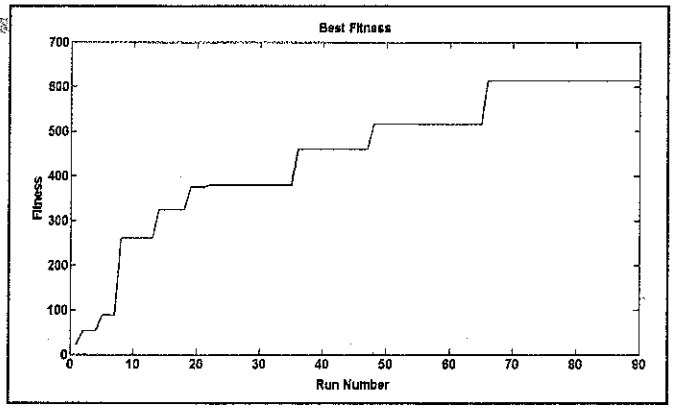

Figure 5—This graph shows the population's best fitness across each run. Individuals quickly evolved in the experiment's beginning and slowed down near the end.

TAKING FLIGHT

The system quickly evolved high-fitness chromosomes in the first 20 runs (see Figure 5). It took more runs to discover better chromosomes as the system became adept at flying the helicopter according to the goal. Many chromosomes resulted in the helicopter flying too high. These were avoided later in the experiment as they resulted in 0 fitness. Chromosomes that didn't raise the helicopter above ground resulted in 0 fitness and were also avoided. The average population fitness sampled every five runs (see Figure 6). A high slope was exhibited in the beginning and it tapered off near the end of the experiment.

The phototransistor network was susceptible to slight changes in movement because the phototransistors only had a 20° viewing angle. The helicopter frequently rotated or shifted a small amount at higher altitudes. This caused the IR LED to leave the phototransistor's field of view. The system interpreted this as a bad run, which resulted in a false negative for a chromosome that may have had a high fitness. The system also generated several false positives for chromosomes that rose to low altitudes but evaded the phototransistors' viewing angle just enough to trick the sensors into thinking the helicopter was at the right altitude. These chromosomes generated high fitness values for bad runs and severely hampered the network's evolutionary capability. However, enough mutations and runs ensured that false negatives and positives did not impede the system's end goal.

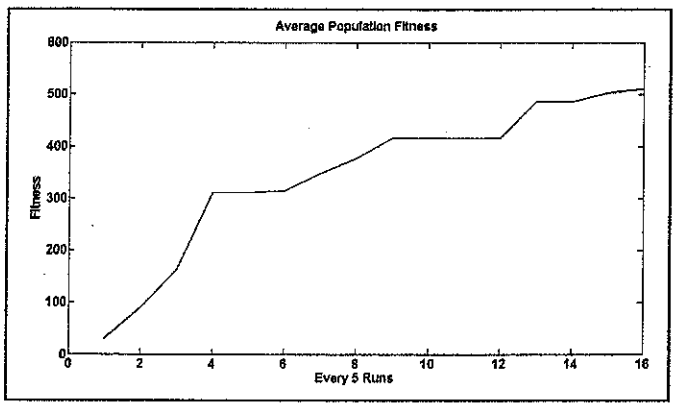

Figure 6—The average population fitness exhibits a high slope near the beginning and a low slope near the end. The average was taken every five runs.

LIFTING OFF

A robust learning and evolving system was develope implement altitude control from a microcontroller to a toy copter. This system involved an IR protocol execution to commands to the helicopter, a phototransistor network to n ure altitude, a boom setup to constrain helicopter moveme 1 DOF, a spiking neural network to take sensory inputs anc put throttle values, and an evolutionary search to find the neural-network configuration. The search converged into a tion that quickly flew the helicopter from the ground to a de point and kept it at a steady 10-s hover. Although the sy exhibited several false positives and negatives throughou search process, the microcontroller overcame them by te enough chromosomes to find the proper configuration. F work should focus on improving experiment aspects, su using wider angle phototransistors and a more reliable boc

Authors' acknowledgements: We would like to thank Prof Bruce Land and the Cornell University ECE Department for ance, ideas, and resources from start to finish of this proj

Akshay Dhawan (and43@cornell.edu) is an MEng student at nell University in Electrical and Computer Engineering. He e his Bachelor of Science at Cornell University. He is specializ bioinstrumentation. His interests include neurophysiology, n interfaces, machine learning, and evolutionary algorithms. earning his MEng degree, Akshay will pursue an MBA at C University.

Sergio Biagioni (sab323@cornell.edu) is an Application Su Engineer at MathWorks. He earned his BS and MEng at C University in Mechanical and Aerospace Engineering. He cializes in aerospace control systems, dynamic modeling, simulation.

REFERENCES

[1] Y. H. Said, "On Genetic Algorithms and Their Applic tions," *Handbook of Statistics, Volume 24: Data Mining and Data Visualization*, North Holland, 2005.

[2] D. Floreano, N. Schoeni, G. Caprari, and J. Blynel, "Evolutionary Bits'n'Spikes," Artificial Life VIII: The 8th International Conference on the Simulation and Synthes of Living Systems, 2002.

RESOURCES

Agustin, *Couch Sprout*, "Arduino Helicopter Infrared Controller," 2011, www.avergottini.com/2011/05/arduin helicopter-infrared-controller.html.

New Wave Instruments, "Linear Feedback Shift Registers," www.newwaveinstruments.com/resources/article m_sequence_linear_feedback_shift_register_lfsr.htm.

Syma Helicopters, www.symahelicopters.com.

SOURCES

ATmega644 Microcontroller
Atmel Corp. | www.atmel.com

LTE-4208 LED Emitter and LTR-4206E phototransist
Lite-On Technology Corp. | http://optoelectronics.liteon.cc