# Evolving Robot Controllers Using Carbon Nanotubes

Maktuba Mohid, Julian F. Miller

University of York, York, UK, YO10 5DD
mm1159@york.ac.uk, julian.miller@york.ac.uk

**Abstract**

Evolution-in-materio uses computer-controlled evolution to configure materials to solve computational problems. In this paper, the material is a mixture of single-walled carbon nanotubes in an insulating polymer. We show for the first time that using purpose-built hardware it is possible to evolve voltages and signals applied to such materials to control robots, both simulated (Kephera) and real (Pi-Swarm). Evolved controllers were able to fully explore an environment, avoid obstacles, and cope with introduced faults, errors and environment changes. We also evolved a robot controller that could solve mazes of various difficulties.

## Introduction

Evolution-in-materio (EIM) is a relatively unexplored area of research which aims to mimic Darwinian evolution by manipulating physical systems using computer controlled evolution (CCE) [Harding and Miller (2009, 2007); Harding et al. (2008); Miller and Downing (2002)]. Like Dawkins "blind watchmaker" metaphor for evolution [Dawkins (1986)], we aim to utilize physical systems without requiring an understanding of their internal processes [Miller et al. (2014)]. We contend that this allows evolutionary algorithms the greatest freedom to find solutions to a given problems. We are trying to avoid Conrad's trap , the so-called "price of programability", which says that in conventional design the vast majority of interactions that could possibly contribute to the problem are deliberately excluded Conrad (1988)! This does not mean that such evolved physical systems cannot be understood. However, it is likely that a complete understanding could only be found after a considerable amount of subsequent analysis.

EIM was inspired by the well-known work of Adrian Thompson who investigated whether it was possible for unconstrained evolution to evolve working electronic circuits using a Field Programmable Gate Array (FPGA). He evolved a circuit that could discriminate between 1kHz or 10kHz signal [Thompson (1998)]. However, Thompson discovered that the evolved circuit operated by exploiting subtle physical properties of the chip. Later Harding and Miller were able to replicate these findings using a liquid crystal display rather than silicon [Harding and Miller (2004)]. In a series of papers they showed that evolved voltages applied to a liquid crystal display could allow it to also implement Boolean logic functions [Harding and Miller (2007)] and act as a simulated robot controller in a simple environment [Harding and Miller (2005)].

We describe the use of a purpose built platform called Mecobo that facilitates computer controlled evolution of a material [Lykkebø et al. (2014)] for controlling a Khepera-like robot and a real Pi-Swarm robot. The Mecobo platform has been developed within an EU funded research project called NASCENCE [Broersma et al. (2012)]. The computational material we have used in this investigation is a mixture of single-walled carbon nanotubes and a polymer. This new platform allows a variety of materials to be investigated in custom designed electrode arrays, using a variety of electrical signals and inputs. One of the aims of NASCENCE is to assess the ability of evolution-in-materio as a methodology for solving a wide variety of computational problems. In recent work, the technique has been applied to solve traveling salesman problems [Clegg et al. (2014)], classification tasks [Mohid et al. (2014c)], function optimization [Mohid et al. (2014b)] and bin-packing [Mohid et al. (2014a)].

Evolutionary computation has been widely used to control robots [Nolfi and Floreano (2001); Bongard (2013)]. EIM has been used before to control simulated robot with wall avoidance behavior. However, the evolvable substrate was liquid crystal [Harding and Miller (2005)]. Here, we give a demonstration that such techniques can be used to control a simulated Khepera robot and a real Pi-Swarm robot [Hilder et al. (2014)] for a desired behavior. In a series of experiments, we aim to evolve a robot controller that allows a simulated robot to explore an enclosed area without colliding with obstacles and to solve mazes with various complexities [Lehman and Stanley (2011)]. At this early stage in this research, we are not claiming that EIM is a competitive method for controlling robots, we are simply trying to demonstrate that evolving configurations of carbon nanotubes has promise and can be used for robot control. This is the first work of its kind. Our experiments provide a

yardstick to assess various aspects of EIM using the Mecobo platform. For instance, we can investigate what type of signals are appropriate, and what mixtures of materials give the best results. Using materials in the genotype-phenotype map has, at present, some drawbacks. The main one is the Mecobo platform is relatively slow which means that we can only feasibly evaluate relatively few potential solutions. However, it is a new approach to the solution of computational problems and as the technology is developed it could offer advantages over conventional computational methods [Miller et al. (2014)].

## Conceptual Overview

EIM is a hybrid system involving both a physical material and a digital computer. In the physical domain there is a material to which physical signals can be applied or measured. These signals are either input signals, output signals or configuration instructions. A computer controls the application of physical inputs applied to the material, the reading of physical signals from the material and the application to the material of other physical inputs known as physical configurations. A genotype of numerical data is held on the computer and is transformed into configuration instructions. The genotypes are subject to an evolutionary algorithm. Physical output signals are read from the material and converted to output data in the computer. A fitness value is obtained from the output data and supplied as a fitness of a genotype to the evolutionary algorithm [Miller et al. (2014)].

In EIM, a highly indirect genotype-phenotype mapping is employed. An evolutionary algorithm using such a mapping may be able to exploit hitherto unknown physical variables in a material which may increase evolvability. Software-only genotype-phenotype mappings are highly constrained. Banzhaf et al. discussed the importance and potential of physicality and embodiment in [Banzhaf et al. (2006)]. It is not clear what materials are best-suited for EIM. However, a few aspects which appear to be important have been identified [Miller and Downing (2002)]. The material needs to be reconfigurable, i.e., it can be evolved over many configurations to get desired response. A physical material should naturally "reset" itself before applying new input signals on it, otherwise it might preserve some memory and might give fitness scores that are dependent on the past behavior. Preferably the material should be physically configured using small voltage and be manipulable at a molecular level [Miller et al. (2014)].

## Mecobo Hardware Platform

The Mecobo hardware platform was designed and built within an EU-funded research project called NASCENCE [Broersma et al. (2012)]. Further details about the Mecobo platform are available in [Lykkebø et al. (2014)]. The material signal interface in Mecobo is very flexible. It not only allows the possibility to evolve which electrode receives an applied signal but also a large variety of configuration signals are available to support materials with different electrical characteristics, from static signals to time dependent digital functions. In the Mecobo platform we have used in this paper (version 3.0), the response from materials can only be sampled as purely static digital signals. Using this platform we can only apply two types of inputs to the material: constant voltage (0V or 3.5V) or a square wave signal. However, different characteristics or input parameters associated with these inputs can be chosen and put under evolutionary control. These input parameters are described in Table 1.

Table 1: Adjustable Mecobo input parameters.

| **Parameter Name** | **Description** | **Note** |
|---|---|---|
| Amplitude | 0 or 1 corresponding to 0V or 3.5V | wave signal amplitude must be 1 |
| Frequency | Frequency of square wave signal | Irrelevant if fixed voltage input |
| Cycle Time | Percentage of period for which square wave is 1 | Irrelevant if fixed voltage input |
| Start time | Start time of applying voltage to electrodes | Measured in milliseconds |
| End time | End time of applying voltage to electrodes | Measured in milliseconds |

The start time and end time of each input signal determines for how long an input is applied.

## Computational Material

The experimental material consists of single-walled carbon nanotubes mixed with polybutyl methacrylate (PBMA) and dissolved in anisole (methoxybenzene). The sample is baked causing the anisole to evaporate. This results in material which is mixture of carbon nanotubes and PBMA. The sample used in our experiments is 1.0% carbon nanotubes by weight (99% by PBMA). Carbon nanotubes are conducting or semi-conducting and role of the PBMA is to introduce insulating regions within the nanotube network, to create non-linear current versus voltage characteristics. The idea is that this might show some interesting computational behavior. Further details of the preparation of experimental material are given in [Mohid et al. (2014c)].

Two electrode arrays are placed in each slide. One sample of experimental material is placed in the middle of each electrode array. Sixteen gold electrodes (eight electrodes on each side) are connected directly with each sample on the electrode array. The electrode array is connected directly with the Mecobo board. The electrode sample is shown in Fig. 1. We have only used one of the electrode arrays in our experiments.

Figure 1: Two electrode arrays, each with carbon nanotubes/polymer sample.

## The Robot

The task for the robot is to start from a given position, travel around a closed environment avoiding the walls and obstacles, cover as much floor space as possible and finally, to reach a specific target location. The control system is able to use the distance sensors on the robot, and use this information in the control of the motion of the robot using motors. We have used a simulated Khepera robotic platform for some of the robot experiments. This robot has eight short range infra red sensors and two motors. We have also used a real Pi-Swarm robot in some experiments. This has similar features to the Khepera robot. Generally in evolutionary robotics, evolution is performed in simulation. Solutions based on simulation can be run in faster than using real robot, as it can ignore the physical properties of the robot and its hardware.

## EIM Controlled Robot

We adapted a Khepera robot simulator (version 2.0) written by Marcin Pilat [1]. Pilat rewrote a Unix based Khepera written by Olivier Michel [Michel (1996)]. The simulated robot has diameter of 55 nominal units and obstacles or walls are made from small bricks having width and height 20 unit. The map is 1000 X 1000 $unit^2$.

The Khepera simulated robot together with the placement of sensors and motors that have been used in the experiments is shown in Fig. 2. The sensor distance value (in a range [0, 1023], where 0 means no object is found and 1023 means an object is in the nearest position) is calculated as a function of the presence (or the absence) of obstacles (see footnote 1). Random noise corresponding to $\pm 10\%$ of its amplitude is added to the distance value of the sensor. In experiments, the distance values of sensors have been used as inputs to material. The robot moves according to the speed (in a range [-10, 10]) of the motor. Random noise of $\pm 10\%$ is also added to the amplitude of the motor speed while random noise of $\pm 5\%$ is added to the direction resulting from the difference of the speeds of the motors. The motor speed is decided by the output of the carbon nanotube material.

The Pi-swarm robot [Hilder et al. (2014)] is designed as part of the Pi Swarm System, which itself is an extension for the Pololu 3-Pi robot [2], which enables the robot to feature as part of a fully autonomous swarm. The sensors and the two motors are organized in the same way as the Khepera robot

[1] http://www.pilat.org/khepgpsim/

[2] http://www.pololu.com

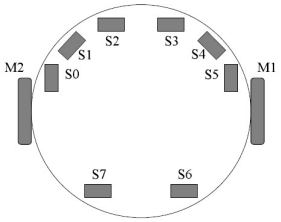


Figure 2: Schematic view of the Khepera and the Pi-Swarm Robot with the positions of the IR proximity sensors (S0-S7) and motors (M1, M2).

(see figure 2). The robot's built in library has a function that can calculate distance between an object and the 8 IR proximity detectors. The distance value has a range [0.0, 100.0] where 100.0 means no object is found. The built in library has functions that accept the motor speed values to drive two motors. The robot's speed is defined to be the range [-1.0, 1.0]) [Hilder (2014)].

### Genotype Representation

In case of all experiments, each electrode requires five genes to define how it is used. These decide whether an electrode is used for input or output, or whether it is to receive a configuration voltage. Also, the type of input signal that will be applied to the electrode is decided by evolution. This can be one of the following: signal type, amplitude, frequency, cycle (see table 1).

In some sets of simulated robot experiments only $n_e = 12$ electrodes from the 16 electrodes have been used (the middle 6 electrodes from each side of one material sample). These 12 electrodes were used in the following manner: 6 electrodes were used as inputs, 2 electrodes were used as outputs and remaining 4 electrodes have been used for configuration signals. The 6 inputs were provided by sensors S0, S2, S3, S5, S6 and S7 (see Fig. 2). In other experiments $n_e = 16$ electrodes have been used, where 8 electrodes (all 8 sensors) have been used as inputs, 2 electrodes as outputs and remaining 6 as configuration signals.

Each chromosome consists of $5 * n_e$ genes. The values that genes can take are shown in Table 2 where $i$ takes values 0, 1, $\ldots n_e$. A genotype for an $n_e$ electrode array is shown below:

$$p_0 s_0 a_0 f_0 c_0 \ldots p_{n_e-1} s_{n_e-1} a_{n_e-1} f_{n_e-1} c_{n_e-1}$$

With $n_e = 2m$ ($m$ is 6 or 8 respectively) electrodes, the first $5m$ gene values of a chromosome are related to inputs and they are:
$p_0 s_0 a_0 f_0 c_0 \ldots p_{m-1} s_{m-1} a_{m-1} f_{m-1} c_{m-1}$

and the last 10 gene values of a chromosome are related to outputs and they are:

Table 2: Description of genotype.

| Gene Symbol | Signal applied to, or read from $i^{th}$ electrode | Allowed values |
|---|---|---|
| $p_i$ | Which electrode is used | 0, 1, 2 ... $n_e - 1$ |
| $s_i$ | Type | 0 (constant) or 1(square-wave) |
| $a_i$ | Amplitude | 0 , 1 |
| $f_i$ | Frequency | 500 ,501 ... 10K |
| $c_i$ | Cycle | 0, 1, ... 100 |

$$p_{n_e-2}s_{n_e-2}a_{n_e-2}f_{n_e-2}c_{n_e-2}p_{n_e-1}s_{n_e-1}a_{n_e-1}f_{n_e-1}c_{n_e-1}$$

In these input and output genes, only the $p_j$ (here, the values of j are 0-5 and 10-11 for solutions with 12 electrodes and values of j are 0-7 and 14-15 for solutions with 16 electrodes) has any effect, the remainder are redundant. The gene $p_j$ decides which electrode will be used for the inputs and outputs of the device. Thus, mutations in these genes can choose a different electrode to be used as an input or output.

## Input Mapping

In all sets of simulated robot experiments, the inputs to the electrode array were square waves with a fixed duty cycle (hereafter referred to as cycle). The cycle time of the square wave signal was determined by a linear mapping of attribute (sensor) data. Denote the $i^{th}$ attribute in a dataset by $I_i$, where $i$ takes values 0-5 (corresponding to 6 sensors) or 0-7 (corresponding to 8 sensors). Denote the maximum and minimum value taken by this attribute in the whole data set by $I_{i_{max}}$ and $I_{i_{min}}$ respectively. Denote the maximum and minimum allowed cycle times, $C_{max}$ and $C_{min}$ respectively. Then the linear mapping given in Eqn. 1 allows the $i^{th}$ attribute of an instance $I_i$ to map to a square-wave cycle time $C_i$ which was applied to a given electrode.

$$C_i = a_i I_i + b_i \tag{1}$$

where the constants $a_i$ and $b_i$ are found by setting $I_i$ and $C_i$ to their respective maximum and minimum and solving for $a_i$ and $b_i$.

$$a_i = (C_{max} - C_{min})/(I_{i_{max}} - I_{i_{min}}) \tag{2}$$

and

$$b_i = (C_{min}I_{i_{max}} - C_{max}I_{i_{min}})/(I_{i_{max}} - I_{i_{min}}) \tag{3}$$

In the experiments, $I_{i_{min}}$=0, $I_{i_{max}}$=1023, $C_{min} = 0$ and $C_{max} = 100$. And, input signals all had frequency 5000Hz with amplitude equal to 1.

Equation 1 shows the input mappings used in real robot experiment, but using different input ranges ($I_{i_{min}}$=0 and $I_{i_{max}}$=100) of sensor values. Note also that the distance value of IR detector was subtracted from value 100.0 to maintain similarity with the input mapping used in simulated robot experiment. In simulated robot experiments a lower distance value was used when there was no object was in range of the IR sensor and a higher distance value was used when an object was extremely close to the IR sensor. In the Pi-Swarm experiment, a value 0 was used when no object was found and 100.0 when an object was in the nearest position.

## Output Mapping

We determined the output, by examining the output buffers containing samples taken from the output electrodes. Since the Mecobo platform can only recognize binary values, the output buffers contain bitstrings. In all sets of simulated robot experiments, the fraction of ones in the buffer is used to get the output values. This fitness seemed appropriate for inputs that are cycle related. The fraction of ones in the output buffer was linearly mapped to motor speed in the ranges [-10, 10]. The Khepera simulator assumes motor speeds defined in the range [-10, 10]. The mapping is shown in Eqn. 4.

$$o_i = M_{min} + (M_{max} - M_{min})num_1/num_{total} \tag{4}$$

Where, $M_{max}$ is 10, $M_{min}$ is -10, $num_1$ is the number of 1's in output buffer and $num_{total}$ is the total number of samples of output buffer and $o_i$ is the output value used to determine the motor speed of the robot. Pi-Swarm experiments also used the fraction of ones and same equation to get output values, but with different output ranges ($M_{min}$=-1.0 and $M_{max}$=1.0) of motor speed.

## Experiments

Five different sets of experiments were performed with a simulated robot. The task of the robot for the first four sets (set A, B, C and D) was to explore a map without colliding with obstacle. Of them, set C and D used a more complex map, where a number of obstacles were distributed all over the map. In case of experiment D, obstacles were put in the map one by one during the evolutionary run to investigate incremental evolution. Experiment B was designed to investigate fault tolerance under sensor failure [Tyrrell et al. (2004)]. Finally experiment E was concerned with maze solving, where the robot had to solve a number of mazes.

In first set (A), the map shown in image 3 (b) was used. In second set (B), the map shown in image 3 (a) was used. In third (C) and fourth (D) sets of experiments, the map shown in image 3 (c) was used. The fifth set of experiments (E) was concerned with maze solving and six different maps were used in that experiment. Following [Shorten and Nitschke (2014)], the fitness of the robot controller in experiment E was gathered in three stages using three maze maps. The maps in the sequence increased in complexity. The evolutionary run started from the simplest map (shown in image 4 (a)), after an individual in the population was

found that could successfully solve the maze, the population was immediately evaluated on the next more complex maze map (shown in image 4 (b)) for further evolution. Once a population member could solve the second maze, the full population was evaluated on the third maze map (shown in image 4 (c)). Once the period of evolution that used the third maze map was completed, the final population of robot controller was tested on three previously unseen other maze maps (shown in image 4 (d), (e) and (f)) to test the generality of the evolved controller. In all of these maps, obstacles are shown in red and the white area of map is the area where the robot is allowed to move.

In the sets of simulated robot experiments A, C, D, only 12 electrodes from the 16 electrodes have been used (the middle 6 electrodes from each side of one material sample). These 12 electrodes were used in the following manner: 6 electrodes were used as inputs, 2 electrodes were used as outputs and remaining 4 electrodes have been used for configuration signals. The 6 inputs were provided by sensors S0, S2, S3, S5, S6 and S7 (see Fig. 2). In experiments B and E all 16 electrodes have been used, where 8 electrodes (all 8 sensors) have been used as inputs, 2 electrodes as outputs and remaining 6 as configuration signals.

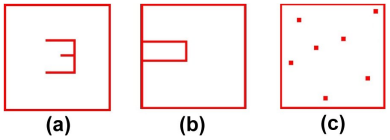


Figure 3: Task environments used simulated robot experiments A-D.

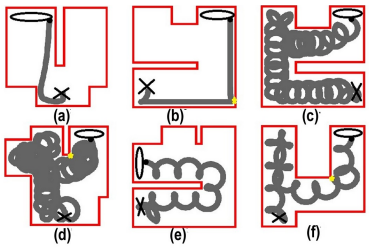


Figure 4: Task environments used in maze solving experiments with examples (according to some experimental results) of paths of evolved robots. In (a)-(f), robot's current position is shown using black ball and the path through which the robot has already visited the map is shown in grey.

For all sets of experiments, each chromosome defined which electrodes were either outputs, inputs (receive square waves) or received the configuration data (square waves or constant voltage).

Using the Mecobo platform the number of samples stored in output buffers can be controlled by the start time, end time and the sampling frequency of output electrode. In all sets of experiments, we used a 25KHz buffer sampling frequency. We applied inputs for a number of milliseconds and accumulated the outputs in a buffer for the same number of milliseconds. We refer to this as input-output timing.

In sets A, C and D of simulated robot experiments, the input-output timing was 20 milliseconds. However, in case of experiments B and E, the input-output timing was 32 and 25 milliseconds respectively due to using a greater number of electrodes. Sampling over longer times is necessary as scheduling in Mecobo is serial. This means that several sequences of actions (i.e., sending input signals, configuration inputs etc) do not take place at the same instant. Mecobo maintains a schedule. Thus, it takes some time for Mecobo to circulate signals to each electrode.

## Fitness calculation

To calculate a fitness of an evolved robot controller each individual of the population is executed for a number of time steps. For experiments A-D we used 5000 time steps and for experiments E (maze solving) up to 10,000 time steps. In most cases, if the robot collides with an obstacle, it is stopped immediately resulting in a lower fitness value, however, in case of maze solving experiment, the robot which reaches near to the goal is allowed to run after a collision, for up to 1000 collisions. Also, if a robot rotates in same place for long time, it is also stopped. The latter is assessed using the x and y coordinates of robot's position over 1050 time steps in the past. If the differences between old and new x and y coordinate are $\leq 30$ units (approximately half the diameter of the robot), it is assumed that the robot has visited the same place as before, otherwise it is assumed that the robot is exploring a new area of the map. The previous 50 moves of the robot are not used to prevent a slowly moving robot being penalized. Thus, if the robot rotates in approximately the same place for 1000 moves, it is stopped immediately and its overall fitness is assessed. However, in the case of experiments E (maze solving), a robot is not stopped if it rotates in the same place for a long time provided it is able to get close to its goal. If robot has not been stopped early and it is exploring a new area of the map, the distance between previous move and the new move is added to fitness score. The distance is calculated using Euclidean planar distance between the previous and the new move. The better individuals are decided by higher fitness values. But in case of maze solving experiments (E), robots are rewarded for reaching points nearer to the goal. This is done by measuring the Euclidean distance between the position of the robot and the goal. This distance is subtracted from the value $1415 \approx 1414.214$, so that the value becomes higher as robot reaches closer to its goal. Here, it should be noted that the largest distance between any two points in the map is 1414.214 corresponding to the points (0, 0) and (1000, 1000). The obtained value is multiplied by a constant (constant value 10)

and then added to the fitness value.

It is quite tricky to decide whether the robot reaches a position close to its goal or not. The Euclidean distance between robot's current position and the goal can be used, but it has a drawback, especially if there is any obstacle between these two positions, in that case, there is no direct path of the robot to reach the goal, thus the robot might need to move a lot to reach the goal. So, merely calculating Euclidean distance between robot's current position and the goal is not a good measure. In the experiment, we have taken this into account by analyzing the positions of all the obstacles. If there is any obstacle between the robot's current position and the goal, the robot is not taken to be near to its goal, otherwise it is decided to be near to its goal.

### Transfer of solutions to real robots

Since the Pi-swarm robot has almost same features as Khepera robot, a Khepera simulator can be used as Pi-swarm robot simulator. We tested the final solutions of experiments A and C sets on a real Pi-swarm robot to investigate whether solutions evolved with a simulator perform exactly the same or not. Only successful solutions of simulated robot experiments were used in real robot experiments, i.e. only those solutions were used which explored the full map without colliding with obstacle. The real robot experiments mainly focused on observing whether the robot explored the whole map without colliding with an obstacle. This is why solutions obtained in experiments B and C of simulated robot experiments were not used on real robots as those experiments were designed to test the robot's ability to cope with simulated sensor faults and environmental changes. Also, E set of experiments was concerned with maze solving problem and that was not used in Pi-Swarm experiment as well. The successful solutions from each experiment A and C were tested on a real Pi-Swarm robot.

## Simulated Robot Experiments

For each of the experiments a $1 + \lambda - ES$, evolutionary algorithm with $\lambda = 4$ was used. The $1 + \lambda - ES$ evolutionary algorithm has a population size of $1 + \lambda$ and selects the genotype with the best fitness to be the parent of the new population. The remaining members of the population are formed by mutating the parent. In all experiments, mutational offspring were created from a parent genotype by mutating a single gene. Note if there is no offspring that has a higher fitness than the parent, but there is at least one that has a fitness equal to the parent, then an offspring is chosen to be the new parent.

Robot starting positions in experiments varied. In maps (a) and (c) shown in Fig. 3, the starting position was the centre of the map while for map (b), the starting position of the robot was randomly chosen. In the maze solving experiments, the starting position is marked with a cross and the goal is marked with an oval in Fig. 4.

### Analysis of Results

In experiments A, ten independent evolutionary runs of 100 generations were carried out. For each individual in the population, the starting position of the robot was randomly selected. In three evolutionary runs, a robot controller was evolved that could explore the full map. Of them, one robot explored the full map and then collided. The other two robots explored the full map without colliding with wall. The minimum number of generation to explore the full map without colliding with wall was 47. It was found from examining the records of robot's movement that the three best exploring robots all started from the upper left corner of the map. This suggests that the starting position of robot plays an important role for the robot for exploring new areas of map.

In experiments B, each of five evolutionary runs of 100 generations were carried out before a fault was introduced. This was done by switching off one sensor (S1) by making the cycle time 0 for the input signal related to that specific sensor. Then each evolutionary algorithm was run for another 100 generations. In all of these cases, the fitness values dropped after adding the fault, however, the robot could recover the fitness value very quickly. Of them, in case of three runs, the fitness value of robot, which was obtained before injecting a fault, was recovered or even exceeded in an average within 42 generations. In one run, the robot could explore the full map before and after adding the fault, but in both of these cases, the robot collided with wall after exploring the map. In two runs, the robot collided with wall without exploring the full map before and after adding the fault. In one run, the robot collided with wall and could not explore the full map before adding the fault, but after adding the fault, it could explore the full map without colliding with wall (in other words the fault was beneficial). In one run, the robot did not collide with a wall, but could not explore the full map before and after injecting the fault.

Experiments C consisted of five evolutionary runs of 300 generations. The third map (c) was used (shown in image 3 (c)). Two robots could explore the full map without colliding with the wall. The other three robots could not explore the full map and also collided with walls. Of them, only one robot could explore 8/9 of the map. The minimum number of generation required to evolve a controller that allowed a robot to explore the whole map without colliding with wall was 198.

The incremental evolution experiments D used 5 independent runs of 300 generations. Other than the 4 side walls, there are 7 obstacles in the middle of the map, which are distributed all over the map. The seven obstacles were added one by one in intervals of 30 generations, starting from the 20th generation. No obstacles were added during the last 100 generations. Whenever new obstacles were added, the fitness dropped, but the robots recovered quickly. However, after 300 generations it was found that none of the five robots

could escape collisions with obstacles. Of these, three robots could explore the full map and then collided. Two robots explored 7/9 of the map and then collided with an obstacle. The highest number of moves a robot could survive after evolving for 300 generations was 4261.

Experiments E were concerned with maze solving. Evolution continued until a robot reached the goal. Five independent runs were carried out. Each evolutionary run was performed on three maps incrementally and the final solutions were tested on three other maps. The mazes are shown in Fig 4. The first maze (a) was solved on an average within 20.6 generations, the second was solved on an average in 56.8 generations, the third maze on an average within 6.6 generations.

The final population of each of the five evolutionary runs above was tested on three different mazes to assess generalization. The results are as follows. In first run, maze (d) was solved by 2 individuals of population, maze (e) was solved by none of the five individuals of the population, and maze (f) was solved by three individuals where one individual solved both first and the third mazes; in the second run, the first, second and the third mazes were solved by 2 individuals; in third run, only the first maze was solved and by only one individual; in fourth run, only the second maze was solved by two individuals; in fifth run, the first maze was solved by two individuals, the second maze was not solved by any of the individuals, the third maze was solved by one robot that also solved the first maze. This means, maze (d) was solved by 7 individuals in total in 4 out of 5 runs, maze (e) was solved by 4 individuals in 2 out of 5 runs and maze (f) was solved by 6 individuals in 3 out of 5 runs. It was found that the first maze was solved by more robots than the other two mazes, this is probably due to the fact that it is the simplest maze. Oddly enough, although the third maze was the hardest, it was still solved by 6 robots, which was more than the number of robots that solved the second maze. The examples (according to some results of maze solving experiments) of paths of evolved robots are shown in Figure 4.

## Real Robot Experiments

In experiments, the Pi-Swarm robot was run by establishing communication between Pi-Swarm robot (through an mbed microcontroller) and the computer program which communicates with material via Mecobo. The mbed sent distance values from 8 IR proximity detectors on the robot to the computer which sent the two motor speed values back to the robot. The mbed ran the robot using those motor speed values for 10 ms and then stopped the robot and sent 8 distance values to the computer and another cycle began. This sequence of operations were performed 5000 times. Note that the robot was not stopped if it collided with an obstacle unlike the simulated robot. This means that the robot was given chance to free itself if it was stuck. For each move the robot was allowed to run for only 10 ms so that the robot would not move at the time when Mecobo was busy. Prior investigation arrived at the timing of 10 ms. Times larger than 10 ms allowed the robot too much time to move resulting in harder control and more collisions. Using times smaller than 10 ms meant the robot moved very slowly.

Figure 5: The Pi-Swarm robot moving within a map. The image was taken in the middle of experiment.

The environmental setting of each real robot experiment was the same as the corresponding simulated robot experiment. Only the two successful final solutions of each of two simulated robot experiments (A and C) were tested on the real Pi-Swarm robot. In the case of both sets, one robot explored half of the map after colliding several times with obstacle and then again was able to escape, but the other robot collided very soon after starting it's journey and could not extricate itself within the full life period (within 5000 time steps).

Analysis showed that the real robot did not perform as well as the simulated robot. This was probably due to the presence of noise and also the extra disturbance caused by having a wire connected between the on-board mbed robot controller and the computer while the robot was running. Wireless communication can be possible, but the sent and received messages have limitations on sizes (limited number of bytes). This limitation meant the robot would communicate with computer program too infrequently. Although we attempted to make the settings of simulated and real robot experiments as similar as possible, the organization of map along with obstacles, the proportion of size of robot, the distance traversed by robot with different motor speeds in the map were not exactly the same as those in the simulated robot experiment. Also, the travel time (10 ms) of robot might not be accurate. Figure 5 shows an image of real robot experiment where Pi-Swarm robot is moving within a map.

## Conclusions

Evolution-in-materio is hybrid of digital and analogue computing where digital computers are used to configure materials to carry out analogue computation. This is a new concept and has the promise of developing entirely new computational devices. A purpose-built evolutionary platform called Mecobo, has been used to evolve configurations of a physical system to control a robot. The material used is a mixture of single-walled carbon nanotubes and a polymer. In some cases, we found that the robot could explore an environment without colliding with walls. In other recent work using Mecobo we have obtained encouraging results on machine learning classification problems [Mohid et al. (2014c)]. Thus, in principle, a classifier can be implemented

using an electrode array and a material sample on a microscope slide and some interfacing electronics. Such a system could act as a low power standalone device. This could have utility in robot control. One of the issues with evolving programs in materials using EIM is that discovering and understanding what is happening inside the material is extremely difficult. One would need very sophisticated measurement and probing techniques which would allow the discovery of important mechanisms. Such probing methods would not have to interfere with the operation of the device. This is a general problem with all evolved physical systems (including biological).

## Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 317662.

## References

Banzhaf, W., Beslon, G., Christensen, S., Foster, J., Képès, F., Lefort, V., Miller, J., Radman, M., and J., R. (2006). Guidelines: From artificial evolution to computational evolution: a research agenda. *Nature Reviews Genetics*, 7:729–735.

Bongard, J. (2013). Evolutionary robotics. *Communications of the ACM*, 56(8):74–85.

Broersma, H., Gomez, F., Miller, J. F., Petty, M., and Tufte, G. (2012). Nascence project: Nanoscale engineering for novel computation using evolution. *International Journal of Unconventional Computing*, 8(4):313–317.

Clegg, K. D., Miller, J. F., Massey, M. K., and Petty, M. C. (2014). Travelling salesman problem solved 'in materio' by evolved carbon nanotube device. In *Parallel Problem Solving from Nature - PPSN XIII - 13th International Conference, Proceedings*, volume 8672 of *LNCS*, pages 692–701. Springer.

Conrad, M. (1988). The price of programmability. In Herken, R., editor, *The Universal Turing Machine A Half-Century Survey*, pages 285–307. Oxford University Press.

Dawkins, R. (1986). *The Blind Watchmaker*. Norton and Company, Inc.

Harding, S. and Miller, J. F. (2004). Evolution in materio: A tone discriminator in liquid crystal. In *In Proceedings of the Congress on Evolutionary Computation 2004*, volume 2, pages 1800–1807.

Harding, S. and Miller, J. F. (2005). Evolution in materio : A real time robot controller in liquid crystal. In *Proceedings of NASA/DoD Conference on Evolvable Hardware*, pages 229–238.

Harding, S. and Miller, J. F. (2009). Evolution in materio. In Meyers, R. A., editor, *Encyclopedia of Complexity and Systems Science*, pages 3220–3233. Springer.

Harding, S. L. and Miller, J. F. (2007). Evolution in materio: Evolving logic gates in liquid crystal. *International Journal of Unconventional Computing*, 3(4):243–257.

Harding, S. L., Miller, J. F., and Rietman, E. A. (2008). Evolution in materio: Exploiting the physics of materials for computation. *International Journal of Unconventional Computing*, 4(2):155–194.

Hilder, J. (2014). PI Swarm System Manual. http://www.york.ac.uk/media/robots-lab/PiSwarm-v091.pdf.

Hilder, J., Naylor, R., Rizihs, A., Franks, D., and Timmis, J. (2014). The pi swarm: A low-cost platform for swarm robotics research and education. In *Advances in Autonomous Robotics Systems*, volume 8717 of *Lecture Notes in Computer Science*, pages 151–162. Springer.

Lehman, J. and Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evol. Comput.*, 19(2):189–223.

Lykkebø, O. R., Tufte, G., Harding, S. L., and Miller, J. F. (2014). Mecobo: A hardware and software platform for in materio evolution. In *Proc. Conf. on Unconventional Computation and Natural Computation*, pages 267–279.

Michel, O. (1996). Khepera simulator version 2.0 user manual.

Miller, J. F. and Downing, K. (2002). Evolution in materio: Looking beyond the silicon box. In *NASA/DOD Conference on Evolvable Hardware*, pages 167–176. IEEE Comp. Soc. Press.

Miller, J. F., Harding, S. L., and Tufte, G. (2014). Evolution-in-materio: evolving computation in materials. *Evolutionary Intelligence*, 7:49–67.

Mohid, M., Miller, J. F., Harding, S. L., Tufte, G., Lykkebø, O. R., Massey, M. K., and Petty, M. C. (2014a). Evolution-in-materio: Solving bin packing problems using materials. In *Proceedings of the 2014 IEEE International Conference on Evolvable Systems (ICES): From Biology to Hardware.*, pages 38–45. IEEE Press.

Mohid, M., Miller, J. F., Harding, S. L., Tufte, G., Lykkebø, O. R., Massey, M. K., and Petty, M. C. (2014b). Evolution-in-materio: Solving function optimization problems using materials. In *Computational Intelligence (UKCI), 2014 14th UK Workshop on*, pages 1–8. IEEE Press.

Mohid, M., Miller, J. F., Harding, S. L., Tufte, G., Lykkebø, O. R., Massey, M. K., and Petty, M. C. (2014c). Evolution-in-materio: Solving machine learning classification problems using materials. In *Parallel Problem Solving from Nature - PPSN XIII - 13th International Conference, Proceedings*, volume 8672 of *LNCS*, pages 721–730. Springer.

Nolfi, S. and Floreano, D. (2001). *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-organizing Machines*. MIT Press, Cambridge, MA, USA.

Shorten, D. P. and Nitschke, G. S. (2014). How evolvable is novelty search? In *2014 IEEE International Conference on Evolvable Systems, ICES 2014, Orlando, FL, USA, December 9-12, 2014*, pages 125–132.

Thompson, A. (1998). *Hardware Evolution - Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution*. Springer.

Tyrrell, A., Krohling, R., and Zhou, Y. (2004). Evolutionary algorithm for the promotion of evolvable hardware. *IEE Proceedings Computers and Digital Techniques*, 151(4):267–275.