SELF HEALING ROBOTS

Arjun Raj.M
Computer Science Engineering
Government Engineering College, Wayanad

Abstract

When people or animals get hurt, they can usually compensate for minor injuries and keep limping along, but for robots, even slight damage can make them stumble and fall. Now a robot scarcely larger than a human hand has demonstrated a novel ability: It can recover from damage – an innovation that could make robots more independent. The new robot, which looks like a splay-legged, four-footed starfish, deduces the shape of its own body by performing a series of playful movements, swiveling its four limbs. By using sensors to record resulting changes in the angle of its body, it gradually generates a computerized image of itself. The robot then uses this to plan out how to walk forward. The researchers hope similar robots will someday respond not only to damage to their own bodies but also to changes in the surrounding environment. Such responsiveness could lend autonomy to robotic explorers on other planets like Mars – a helpful feature, since such robots can't always be in contact with human controllers on earth. Aside from practical value, the robot's abilities suggest a similarity to human thinking as the robot tries out various actions to figure out the shape of its world.

1 Introduction

1.1 ROBOTS

A robot is a mechanical or virtual, artificial agent. It is usually an electromechanical system, which, by its appearance or movements, conveys a sense that it has intent or agency of its own. A typical robot will have several, though not necessarily all of the following properties:

- Is not 'natural' i.e. has been artificially created.
- Can sense its environment.
- Can manipulate things in its environment.
- Has some degree of intelligence or ability to make choices based on the environment or automatic control or pre-programmed sequence.
- Is programmable.
- Can move with one or more axes of rotation or translation.
- Can make dexterous coordinated movements.
- Appears to have intent or agency (reification, anthropomorphisation or Pathetic fallacy). Robotic systems are of growing interest because of their many practical applications as well as their ability to help understand human and animal behavior, cognition, and physical performance. Although industrial robots have long been used for repetitive tasks in structured environments, one of the long-standing challenges is achieving robust performance under uncertainty. Most robotic systems use a manually constructed mathematical model that captures the robots dynamics and is then used to plan actions. Although some parametric identification methods exist for automatically improving these models, making accurate models is difficult for complex machines, especially when trying to account for possible topological changes to the body, such as changes resulting from damage.

1.2 ERROR RECOVERY

Recovery from error, failure or damage is a major concern in robotics. A majority of effort in programming automated systems is dedicated to error recovery. The need for automated error recovery is even more acute in the field of remote robotics, where human operators cannot manually repair or provide compensation for damage or failure.

Here, its explained how the four legged robot automatically synthesizes a predictive model of its own topology (where and how its body parts are connected) through limited yet self-directed interaction with its environment, and then uses this model to synthesize successful new locomotive behaviour before and after damage. These findings may help develop more robust robotics, as well as shed light on the relation between curiosity and cognition in animals and humans.

2 SELF HEALING OR SELF MODELLING ROBOTS

When people or animal get injured ,they compensate for minor injuries and keep limping along. But in the case of robots, even a slight injury can make them stumble and fall .Self healing robots have an ability to adapt to minor injuries and continue its job . A robot is able to indirectly infer its own morphology through selfdirected exploration and then use the resulting self-models to synthesize new behaviors.If the robots topology unexpectedly changes, the same process restructures its internal self-models, leading to the generation of qualitatively different, compensatory behavior. In essence, the process enables the robot to continuously diagnose and recover from damage. Unlike other approaches to damage recovery, the concept introduced here does not presuppose built-in redundancy, dedicated sensor arrays, or contingency plans designed for anticipated failures. Instead, our approach is based on the concept of multiple competing internal models and generation of actions to maximize disagreement between predictions of these models.

2.1 RESEARCHERS

This research was done at the Computational Synthesis Lab at Cornell University. Team members are Josh Bongard, Viktor Zykov, and Hod Lipson. Josh Bongard was a postdoctoral researcher at Cornell while performing this research and since then moved to the University of Vermont where he is now an Assistant Professor. Victor Zykov is a Ph.D. student at CCSL, and Hod Lipson is an Assistant Professor at Cornell, and directs the Computational Synthesis Lab. This project was funded by the NASA Program on Intelligent Systems and by the National Science Foundation program in Engineering Design.

2.2 THE STARFISH ROBOT

2.2.1 CHARACTERIZING THE TARGET SYSTEM

The target system in this study is a quadrupedal, articulated robot with eight actuated degrees of freedom. The robot consists of a rectangular body and four legs attached to it with hinge joints on each of the four sides of the robots body. Each leg in turn is composed of an upper and lower leg, attached together with a hinge joint. All eight hinge joints of the robot are actuated with Airtronics 94359 high torque servomotors. However, in the current study, the robot was simplified by assuming that the knee joints are frozen: all four legs are held straight when the robot is commanded to perform some action. The following table gives the overall dimensions of the robots parts.

All eight servomotors are controlled using an onboard PC-104 computer via a serial servo control board SV-203B, which converts serial commands into pulse-width modulated signals. Servo drives are capable of producing a maximum of 200 ounceinches of torque and 60 degrees per second of speed. The actuation ranges for all of the robots joints are summarized in the following table.

This four-legged robot can automatically synthesize a predictive model of its own topology (where and how its body parts are connected), and then successfully move around. It can also use this "proprioceptive" sense to determine if a component has been damaged, and then model new movements that take the damage into account.

The robot is equipped with a suite of different sensors polled by a 16-bit 32- channel PC-104 Diamond MM-32XAT data acquisition board. For the current identification task, three sensor modalities were used: an external sensor was used to determine the left/right and forward/back tilt of the robot; four binary values indicated whether a foot was touching the ground or not; and one value indicated the clearance distance from the robots underbelly to the ground, along the normal to its lower body surface. All sensor readings were conducted manually, however all three kinds of signals will be recorded in future by on-board accelerometers, the strain gauges built into the lower legs, and an optical distance sensor placed on the robots belly.

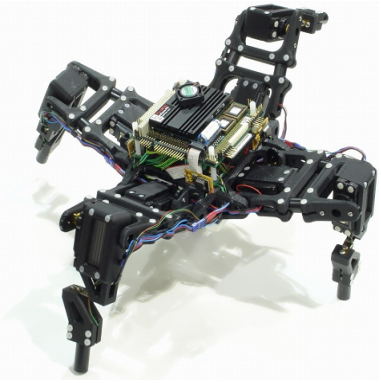

Figure 1: The Starfish Robot

2.3 SELF MODELLING BRIEFLY

Here, its explained how the four legged robot automatically synthesizes a predictive model of its own topology (where and how its body parts are connected) through limited yet self-directed interaction with its environment, and then uses this model to synthesize successful new locomotive behaviour before and after damage. These findings may help develop more robust robotics, as well as shed light on the relation between curiosity and cognition in animals and humans.

A robots most formidable enemy is an uncertain and changing environment. Typically, robots depend on internal maps (either provided or learned), and sensory data to orient themselves with respect to that map and to update their location. If the environment is changing or noisy, the robot has to navigate under uncertainty, and constantly update the probabilities that a particular action will achieve a particular result. The situation becomes even worse if the robots own shape and configuration can change, that is, if its internal model becomes inaccurate. In most cases, such an event constitutes the end of that particular robots adventure.

Although much progress has been made in allowing robotic systems to model their environment autonomously, relatively little is known about how a robot can learn its own morphology, which cannot be inferred by direct observation or retrieved from a database of past experiences. Without internal models, robotic systems can autonomously synthesize increasingly complex behaviors or recover from damage through physical trial and error, but this requires hundreds or thousands of tests on the physical machine and is generally too slow, energetically costly, or risky. Here, we describe an active process that allows a machine to sustain performance through an autonomous and continuous process of self-modeling. A robot is able to indirectly infer its own morphology through self-directed exploration and then use the resulting self-models to synthesize new behaviours . If the robots topology unexpectedly changes, the same process restructures its internal self-models, leading to the generation of qualitatively different, compensatory behavior. In essence, the process enables the robot to continuously diagnose and recover from damage. Unlike other approaches to damage recovery, the concept introduced here does not presuppose built-in redundancy, dedicated sensor arrays, or contingency plans designed for anticipated failures. Instead, our approach is based on the concept of multiple competing internal models and generation of actions to maximize disagreement between predictions of these models. The process is composed of three algorithmic components that are executed continuously by the physical robot while moving or at rest (Fig. 2.3): Modeling, testing, and prediction.

Phases in self healing: Initially, the robot performs an arbitrary motor action and records the resulting sensory data (Fig. 2.3A). The model synthesis component (Fig. 2.3B) then synthesizes a set of 15 candidate self-models using stochastic optimization to explain the observed sensory-actuation causal relationship. The action synthesis component (Fig. 2.3C) then uses these models to find a new action most likely to elicit the most information from the robot. This is accomplished by searching for the actuation pattern that, when executed on each of the candidate self models, causes the most disagreement across the predicted sensor signals. This new action is performed by the physical robot (Fig. 2.3A), and the model synthesis component now reiterates with more available information for assessing model quality. After 16 cycles of this process have terminated, the most accurate model is used by the behavior synthesis component to create a desired behavior (Fig. 2.3D) that can then be executed by the robot (Fig. 2.3E). If the robot detects unexpected sensor-motor patterns or an external signal as a result of unanticipated morphological change, the robot reinitiates the alternating cycle of modeling and exploratory actions to produce new models reflecting the change. The new most accurate model is now used to generate a new, compensating behavior to recover functionality. A complete sample experiment is shown in Fig. 2.4.

The proposed process (Model-driven algorithm) was compared with two baseline algorithms, both of which use random rather than self-modeldriven data acquisition. All three algorithm variants used a similar amount of computational effort (250,000 internal model simulations) and the same number (16) of physical actions (Table 2.3). In the first baseline algorithm, 16 random actions were executed by the physical robot (Fig. 1A), and the resulting data were supplied to the model synthesis component for batch training (Fig. 2.3B). In the second baseline algorithm, the action synthesis component output a random action, rather than searching for one that created disagreement among competing candidate self-models. The actions associated with Fig. 2.3, A to C, were cycled as in the proposed algorithm, but Fig. 2.3C output a random action, rather than an optimized one.

Before damage, the robot began each experiment with a set of random models; after damage, the robot began with the best model produced by the model-

	Baseline 1	Baseline 2	Model-driven algorithm
Before damage			
Independent experiments (*n*)	30	30	30
Physical actions per experiment	16	16	16
Mean model evaluations (*n* = 30)	262,080 ± 13,859	246,893 ± 17,469	262,024 ± 13,851
Successful self-models	7	8	13
Success rate	23.3%	26.7%	43.3%
Mean model error (*n* = 30)	9.62 ± 1.47 cm	9.7 ± 1.45 cm	7.31 ± 1.22 cm
After damage			
Independent experiments (*n*)	30	30	30
Physical actions per experiment	16	16	16
Mean model evaluations (*n* = 30)	292,430 ± 44,375	278,140 ± 37,576	296,000 ± 22,351
Mean model error (*n* = 30)	5.60 ± 2.98 cm	4.55 ± 3.22 cm	2.17 ± 0.55 cm

Figure 2: Results of Baseline Algorithms

driven algorithm (Fig. 2.4F). It was found that the probability of inferring a topologically correct model was notably higher for the model-driven algorithm than for either of the random baseline algorithm (Table 2.3), and that the final models were more accurate on average in the model-driven algorithm than in either random baseline algorithm (Table 2.3). Similarly, after damage, the robot was better able to infer that one leg had been reduced in length using the model-driven algorithm than it could, using either baseline algorithm. This indicates that alternating random actions with modeling, compared with simply performing several actions first and then modeling, does not improve model synthesis (baseline 2 does not outperform baseline 1), but a robot that actively chooses which action to perform next on the basis of its current set of hypothesized self-models has a better chance of successfully inferring its own morphology than a robot that acts randomly (the model-driven algorithm outperforms baseline algorithms 1 and 2).

Because the robot is assumed not to know its own morphology a priori, there is no way for it to determine whether its current models have captured its body structure correctly. It was found that disagreement among the current model set (information that is available to the algorithm) is a good indicator of model error (the actual inaccuracy of the model, which is not available to the algorithm), because a positive correlation exists between model disagreement and model error across the (n = 30) experiments that use the model-driven algorithm (Spearman rank correlation = 0.425, P ¡ 0.02). Therefore, the experiment that resulted in the most model agreement (through convergence toward the correct model) was determined to be the most successful from among the 30 experiments performed, and the best model it produced (Fig. 2F) was selected for behavior generation. This was also the starting model that the robot used when it suffered unexpected damage (Table 2.3).

The behavior synthesis component (Fig. 2.3D) was executed many times with this model, starting each time with a different set of random behaviors Although there is some discrepancy between the predicted distance and actual distance, there is a clear forward motion trend that is absent from the random behaviors. This indicates that this automatically generated self-model was sufficiently predictive to allow the robot to consistently develop forward motion patterns without further physical trials. The transferal from the self-model to reality was not perfect, although the gaits were qualitatively similar; differences between the simulated and physical gait were most likely due to friction and kinematic bifurcations at symmetrical postures, both difficult to predict. Similarly, after damage, the robot was able to synthesize sufficiently accurate models (an example is given in Fig. 2.4 O) for generating new, compensating behaviors that enabled it to continue moving forward.

3 ALGORITHM

A number of algorithms based on repeated testing for error recovery have been proposed and demonstrated for both robotics, and electronic circuits. However, repeated generate-and-test algorithms for robotics are not desirable for several reasons: repeated trials may exacerbate damage and drain limited energy; long periods of time are required for repeated hardware trials; damage may require rapid compensation (e.g., power drain due to coverage of solar panels); and repeated trials continuously change the state of the robot, making damage diagnosis difficult.

Due to the recent advances in simulation it has become possible to automatically evolve the morphology and the controller of simulated robots together in order to achieve some desired behavior. Here we also use evolutionary algorithms to co-evolve robot bodies and brains, but use an inverse process: instead of evolving a controller given robot morphology, we evolve a

root morphology given a controller. Also, instead of evolving to reach a high fitness as a form of design, we evolve towards an observed low fitness (caused by some unknown failure) as a form of diagnosis. By not making a distinction between the robots morphology and controller, and by employing an evolutionary algorithm, the algorithm can compensate for damage or failure of the robots mechanics, its sensory or motor apparatus, or the controller itself, or some combination of these failure types. This stands in contrast to all other approaches to automated recovery so far, which can only compensate for a few pre-specified failures.

Moreover, by using an evolutionary algorithm for recovery, qualitatively different behaviors (such as hopping instead of walking) evolve in response to failure. More traditional analytic approaches can only produce slightly modified behaviors in response to mild damage.

3.1 ALGORITHM OVERVIEW

3.1.1 Estimation-exploration algorithm

The estimation-exploration algorithm is essentially a co-evolutionary process comprising two populations. One population is of candidate models of the target system, where a models fitness is determined by its ability to correctly explain observed data from the target system. The other population is of candidate unlabelled sentences, each of whose fitness is determined by its ability to cause disagreement among model classifications (thereby elucidating model uncertainties), or by exploiting agreement among models to achieve some desired output (thereby capitalizing on model certainties).

The estimation-exploration algorithm has two functions: damage hypothesis evolution (the estimation phase) and controller evolution (the exploration phase). The algorithm also maintains a database, which stores pairs of data: an evolved controller and the fitness produced by the physical robot when that controller is used. Two separate evolutionary algorithmsthe estimation EA and the exploration EAare used to generate hypotheses regarding the failure incurred by the physical robot, as well as controllers for the simulated and physical robot, respectively. Figure 3 outlines the flow of the algorithm, along with a comparison against an algorithm for evolving function

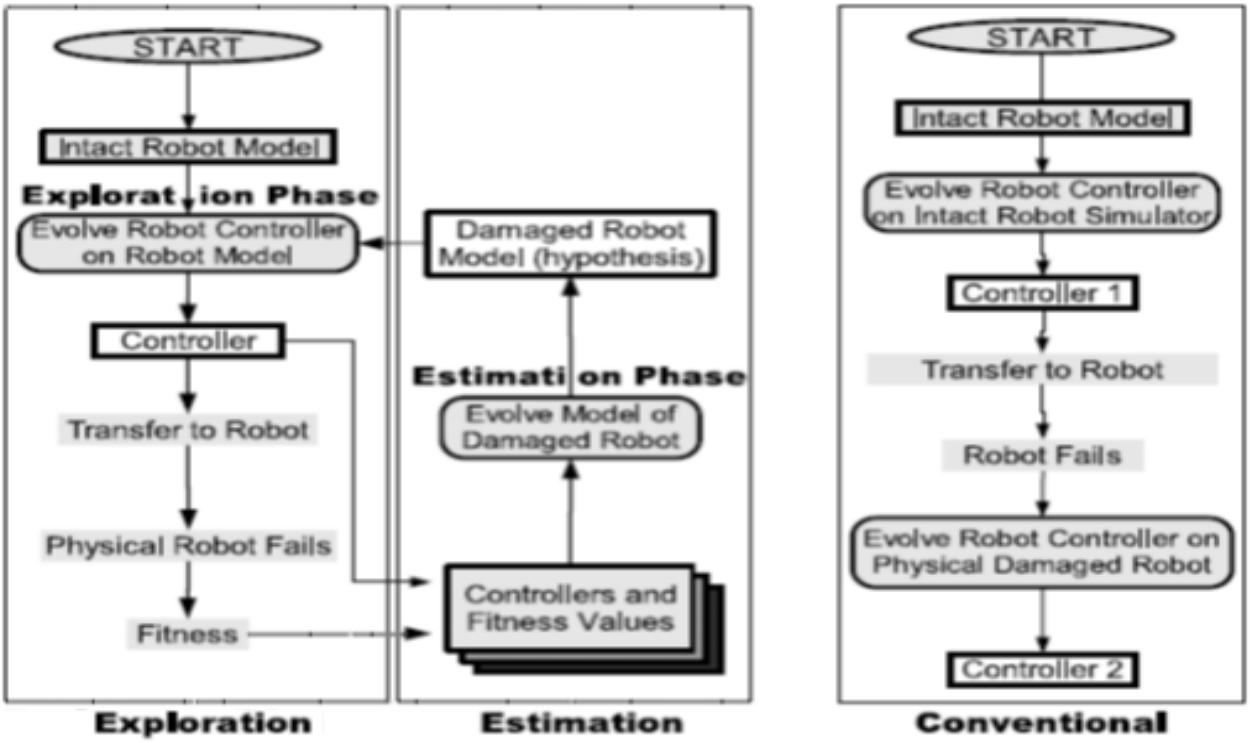

Figure 3: Flow chart of estimation-exploration phases

recovery all on a physical robot.

3.1.2 Exploration Phase: Controller Evolution

The exploration EA is used to evolve a controller for the simulated robot, such that it is able to perform some task. The first pass through this phase generates the controller for the intact physical robot: subsequent passes attempt to evolve a compensatory controller for the damaged physical robot, using the current best damage hypothesis generated by the estimation phase. When the exploration EA terminates, the best controller from the run is transferred to and used by the physical robot.

3.1.3 Physical Robot Failure:

The physical robot uses an evolved controller to walk forwards. An unanticipated failure occurs to the robot, and the broken robot records its own forward displacement for a period of time. The physical robot is then stopped, and the recorded forward displacement (fitness) is inserted into the database along with the evolved controller on-board the robot at that time: these become an input-output pair used to reverse engineer the damage suffered by the robot. During subsequent passes through the algorithm, the damaged robot attempts to function using the compensatory evolved controller produced by the exploration phase.

3.1.4 Estimation Phase: Damage Hypothesis Evolution

The estimation EA is used to evolve a hypothesis about the actual failure incurred by the physical robot. The estimation EA uses the forward displacements produced by the broken physical robot, along with the corresponding controllers running on the physical robot at that time, to measure the correctness of each of the diagnoses encoded by the estimation EAs genomes. When the estimation EA terminates, the most fit damage hypothesis is supplied to the exploration EA. The robot simulator is updated to model this damage hypothesis: for example if the hypothesis is that one of the legs has fallen off, that leg is broken off of the simulated robot. The exploration EA then evolves a compensatory controller using this updated model.

3.2 EXPERIMENTAL SETUP

The proposed algorithm was applied to the recovery of locomotion of severely damaged legged robots. A robot simulator is used to evolve controllers for the physical robot: here the physical robot is also simulated. Evolved controllers are uploaded from the simulation to the physical robot, and performance measurements are downloaded from the physical robot to the simulation. The robot simulator is based on Open Dynamics Engine, an open-source 3D dynamics simulation package. The simulated robot is composed of a series of three-dimensional objects, connected with one degree-of freedom rotational joints.

3.2.1 THE ROBOTS

The two hypothetical robots tested in this preliminary worka quadrupedal and hexapedal robotis shown in Figure 3.2. The quadrupedal robot has eight mechanical degrees of freedom. There are two one degree-of-freedom rotational joints per leg: one at the shoulder, and one at the knee. The quadrupedal robot contains four binary touch sensors, one in each of the lower legs. The touch sensor returns, 1.0 if the lower leg is on the ground and 1.0 otherwise. There are also four angle sensors in the shoulder joints, which return a signal commensurate with the flex or extension of that joint (1.0 for maximum flexure up to 1.0 for maximum extension).

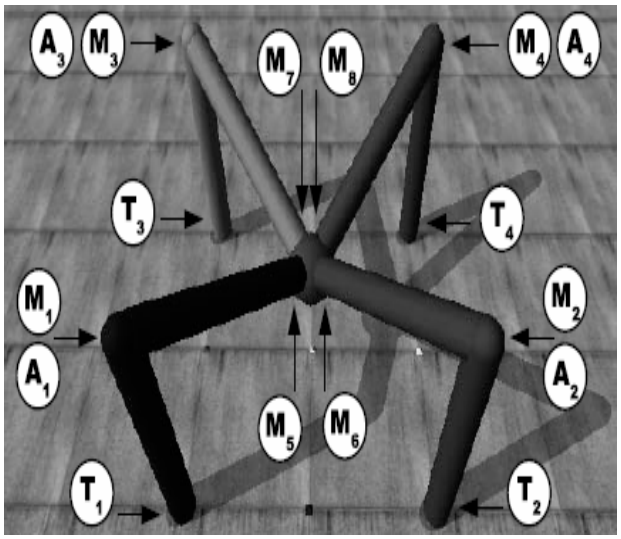

Figure 4: The simulated robots used for experimentation

Each of the eight joints is actuated by a torsional motor. The joints have a maximum flex of 30 degrees from their original setting (shown in Figure 3), and a maximum extension of 30 degrees. The hexapedal robot has 18 mechanical degrees of freedom: each leg has a one degree-of-freedom rotational joint at the

knee, and one two degree-of-freedom rotational joints connecting the leg to the spine. Each joint is actuated by a torsional motor, and the joint ranges are the same as for the quadrupedal robot. The hexapedal robot contains six touch sensors, one per lower leg, and six angle sensors, placed on the joints connecting the legs to the spine.

3.2.2 The Controllers

The robots are controlled by a neural network, which receives sensor data from the robot at the beginning of each time step of the simulation into its input layer, propagates those signals to a hidden layer, and finally propagates the signals to an output layer. The neural network architecture and connectivity is shown in Figure 3.3. Neuron values and synaptic weights are scaled to lie in the range [1.00, 1.00]. A threshold activation function is applied at the neurons. There is one output neuron for each of the motors actuating the robot: the values arriving at the output neurons are scaled to desired angles for the joint corresponding to that motor. For both robots here, joints can flex or extend to / 4 away from their default starting rotation, / 2.The angles are translated into torques using a PID controller, and the simulated motors then apply the resultant torques. The physical simulator then updates the position, orientation and velocity of the robot based on these torques, along with external forces such as gravity, friction, momentum and collision with the ground plane.

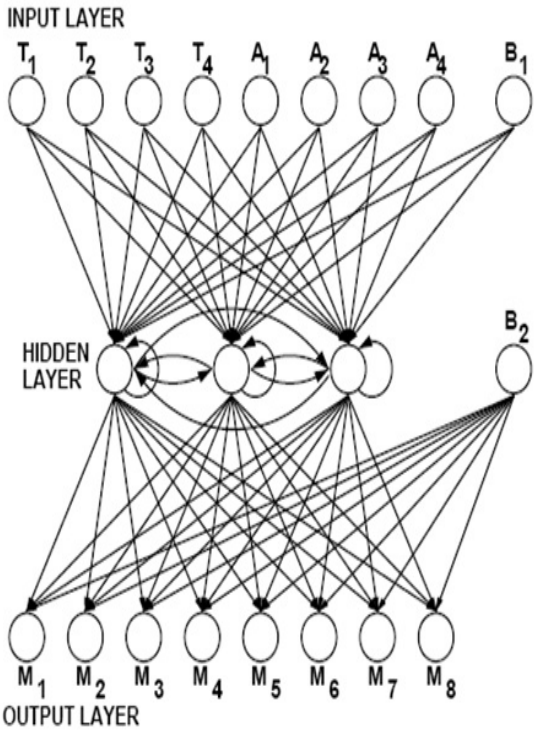

Figure 5: The neural network architecture used for the quadrapedal robot

3.3 ALGORITHM IMPLEMENTATION

3.3.1 The Exploration EA:

The exploration EA is used to generate sets of synaptic weights for the robots neural network . The fitness function rewards robots for moving forwards as far as possible during 1000 time steps of the simulation. The fitness function is given as f(gi) = d(t1000) - d(t1), where f(g) is the fitness, measured in meters, of the robot whose neural network controller is labeled with the values encoded in genome g. d(t1) is the forward displacement of the robot, measured in meters, at the first time step of the simulation; d(t1000) is the forward displacement of the robot (again in meters) at the final time step of the simulation. The genomes of the exploration EA are strings of floating point values, which encode the synaptic weights. For the quadrupedal robot, there are a total of 68 synapses, giving a genome length of 68. For the hexapedal robot, there are a total of 120 synapses, giving a genome length of 120. The encoded synaptic weights are represented to two decimal places, and lie in the range [-1.00, 1.00]. At the beginning of each run a random population of 100 genomes is generated. If there are any previously evolved controllers stored in the database, these are downloaded into the starting population. A genome is evaluated as follows: the encoded weights are used to label the controller; the robot is then evaluated in the simulator for 1000 time steps using that controller; and the resulting fitness value is returned. Once all of the genomes in the population have been evaluated, they are sorted in order of decreasing fitness, and the 50 least fit genomes are deleted from the population. Fifty new genomes are selected to replace them from the remaining 50, using tournament selection, with a tournament size of 3. Selected genomes undergo mutation: each floating-point value of the copied genome has a 1 per cent chance of undergoing a point mutation. Of the 50 newly generated genomes, 12 pairs are randomly selected and undergo one-point crossover.

3.3.2 The Estimation EA:

The estimation EA evolves hypotheses about the failure incurred by the physical robot. The genomes of the estimation EA, like the exploration EA, are strings of floating-point values. Each genome in the estimation EA is composed of four genes: each gene denotes a possible failure. In this preliminary study, the actual robot can undergo three different types of damagejoint breakage, joint jamming, and sensor failureand can incur zero to four of these damages simultaneously. In joint breakage, any single joint of the robot can break completely, separating the two parts of the robot connected by that joint. In joint jamming, the two objects attached by that joint are welded together: actuation has no effect on the joints angle. In sensor failure, any sensor within the robot (either one of the touch or angle sensors) feeds a zero signal into the neural network during subsequent time steps. Any type of failure that does not conform to one of these types is referred to henceforth as an unanticipated failure: in order to compensate for such cases, the estimation EA has to approximate the failure using aggregates of the encoded failure types. Each of the four genes encoded in the estimation EA genomes is comprised of four floating-point values, giving a total genome length of 16 values. Like the exploration EA, each of the values is represented to two decimal places, and lies in [1.00, 1.00]. The first floating-point value of a gene is rounded to an integer in [0, 1] and denotes whether the gene is dormant or active. If the gene is dormant, the damage encoded by this particular gene is not applied to the simulated robot during evaluation. If the gene is active, the second floating point value is rounded to an integer in [0, 2], and indicates which of the three damage types should be applied to the simulated robot. If the damage type is either joint breakage or joint jamming, the third value is scaled to an integer in [0, j 1], where j is the number of mechanical degrees of- freedom of the robot (j = 8 for the quadrupedal robot, and j = 12 for the hexapedal robot). If the damage type is sensor failure, then the third value is scaled to an integer in [0, s 1], where s is the total number of sensors contained in the robot (s = 8 for the quadrupedal robot and s = 12 for the hexapedal robot). The fourth value is currently not used in this preliminary study, but will be used for additional damage types that are not binary, but occur with a lesser or greater magnitude (i.e. a sensor that experiences 80 percent damage, instead of completely failing). For each genome in the estimation EA, the simulated robot is initially broken according to the failure scenario encoded in the genome, and the broken robot is then evaluated using the controller just evolved by the exploration EA and tested on the physical robot.

The fitness function for the estimation EA is an attempt to minimize the difference between the forward displacement achieved by the physical robot using that controller, and the forward displacement achieved by the simulated robot using the encoded damage hypothesis. This is based on the observation that the closer the damage hypothesis encoded in the genome is to the actual damage, the lesser the difference between the two behaviors.

DAMAGE SCENARIOS

Scenario	Explanation
1	One of the lower legs breaks off.
2	One of the entire legs breaks off.
3	One of the touch sensors fails.
4	One of the entire legs breaks off, and the touch sensor on one of the intact legs fails.
5	One of the angle sensors fails.
6	One of the upper-leg joints jams.
7	One of the entire legs break off, and One of the upper-leg joints jams on an intact leg.
8	One of the entire legs break off, One of the upper-leg joints jams on an intact leg , and One of the angle sensors fails on another intact leg.
9	Nothing breaks.
10	One of the hidden neurons fails.

Figure 6: Damage scenarios tested

During subsequent passes through the estimation phase, there are additional pairs of evolved controllers and forward displacements in the database: the controllers evolved by the exploration EA and the fitness values attained by the physical robot when using those controllers, respectively. In these cases, the simulated robot is evaluated once for each of the evolved controllers, and the fitness of the genome is then the sum of the errors between the forward displacements. When the estimation EA terminates, the best evolved damage hypothesis is stored in a database: these hypotheses are used to seed the random population at

the beginning of the next run of the estimation EA, rather than starting each time with all random hypotheses.

The estimation EA is similar to the exploration EA, except for the length of the genomes, what those genomes encode, and the fitness function: in the exploration EA, forward displacement is maximized; in the estimation EA, error between the simulated and physical robots forward displacements is minimized.

3.4 RESULTS OF ESTIMATION-EXPLORATION ALGORITHM

Control experiments were performed which conforms to the algorithm outlined in the right-hand panel of Figure all evolution is performed on the physical robot after damage. In this case, controller evolution is performed by the exploration EA until generation 30 on the quadrupedal robot. The controller is then transferred to the physical robot, which then undergoes separation of one of its lower legs (damage case 1). The exploration EA then continues on the physical robot for a further 70 generations.

The algorithm proposed here was then applied several times to the quadrupedal and hexapedal robots. During each application of the algorithm, the robots suffered a different damage scenario: the 10 scenarios are listed in Table 3.1 For each run of the algorithm, the exploration EA is run once to generate the initial evolved controller, and then both the estimation and exploration EAs are run three times each after physical robot failure. Each EA is run for 30 generations, using a population size of 100 genomes. Twenty runs of the algorithm were performed (10 damage cases for each of the two robots), in which both the exploration and estimation EAs were initialized with independent random starting populations seed with any previously evolved controllers or damage hypotheses.

Damage scenarios 1, 2, 3, 5 and 6 can be described by a single gene in the genomes of the estimation EA. Scenarios 4, 7 and 8 represent compound failures, and require more than one gene to represent them. Case 9 represents the situation when the physical robot signals that it has incurred some damage, when in fact no damage has occurred. Case 10 represents an unanticipated failure: hidden neuron failure cannot be described by the estimation EA genomes. Figure 6 shows the recovery of the quadrupedal robot after minor damage (scenario 3); after suffering unanticipated damage (scenario 10); and recovery of the hexapedal robot after severe, compound damage (scenario 8).

3.5 ANALYSIS OF ESTIMATION-EXPLORATION ALGORITHM

It can be seen, that even after several generations have elapsed after the physical robot suffers damage for the control experiment, and 3550 hardware evaluations have been performed, total function has not been restored. The degree of restoration (about 70

The algorithm performs equally well for both morphological and controller damage: function recovery for scenarios 1 and 2 (morphological damage) and scenarios 3 and 5 (controller damage), for both robots, approaches or exceeds original performance. Because the algorithm evolves the robot simulator itself based on the physical robots experience, it would be straightforward to generalize this algorithm beyond internal damage: the estimation EA could evolve not only internal damage hypotheses but also hypotheses regarding environmental change, such as increased ruggedness of terrain or high winds.

3.6 ESTIMATION-EXPLORATION ALGORITHM OVERVIEW

3.6.1 Characterization of the target system

- Define a representation, variation operators and similarity metric for the space of systems - Define a representation and variation operators for the space of inputs (tests) - Define a representation and similarity metric for the space of outputs

3.6.2 Initialization

- Create an initial population of candidate models (random, blank, or seeded with prior information) - Create an initial population of candidate tests (random, or seeded with prior information)

3.6.3 Estimation Phase

- Evolve candidate models; encourage diversity - Fitness of a model is its ability to explain all input-output data in training set

3.6.4 Exploration Phase

- Evolve candidate tests (input sets) - Fitness of a test is the disagreement it causes among good candidate models - Carry out best test on target system; add input/output data to training set

3.6.5 Termination

- Iterate estimation-exploration (steps 3-4) until the population of models converges on a sufficiently accurate solution, or the target system exhibits some desired behavior. - If no model is found, the search space may be inappropriate, or the target system may be inconsistent - If no good test is found, then either: All good candidate models are perfect; The search method for finding good tests is failing; or The target system may be partially unobservable

3.6.6 Validation

- Validate best model(s) using unseen inputs - If validation fails, add new data to training set and resume estimation phase

4 FROM THEORY TO REALITY

Here, we present validation of our algorithm on a physical articulated robot (shown in Figure 1a): the robot evolves an explicit model of its body using sensor data from different modalities. This is the first time explicit, predictive robot models have been intelligently synthesized based on physical interactions.

In previous section, we have demonstrated that the EEA can synthesize both the topology and parameters of a hidden system, in which no model is required a priori (Bongard and Lipson, 2005a).

In order to apply the EEA to a new target system, such as the physical robot used in this work, three preparatory steps must be first carried out: characterization of the system to be identified, how models are to be represented and optimized, and how controllers are to be represented and optimized.

4.1 Characterizing the Target System

The target system in this study is a quadrupedal. The Details of the target system has been explained in section 2.2.1

4.2 Characterizing the Space of Models

Models are considered to be three-dimensional simulations of the physical robot (see Figure 5 for three model examples). The simulations are created within Open Dynamics Engine, a three-dimensional dynamics simulator. However in the current work only static identification is performed: the physical robot is commanded to achieve a static pose, and then hold still while sensor data is taken. Every candidate model (as well as the target robot) is assumed to start as a planar configuration of parts; when it begins to move, it can assume a three-dimensional configuration. The geometry and physical properties of the main body part is assumed to be known; the eight upper and lower leg parts are represented as solid cylinders. Each model is evaluated for an arbitrarily set time of 300 time steps of the simulator, which is enough time for most models to come to rest given an arbitrary motor program. Models are encoded as either vectors or matrices, and these data structures are used to construct a possible articulated robot in the simulation environment. In the first set of experiments, it was assumed that everything about the physical robot is known except the lengths of its four legs. Models are therefore encoded as vectors containing eight real-valued parameters in [0, 1], with each value encoding the estimated length of one of the eight leg parts. We constrain the estimation about the minimum and maximum length of a leg to be between 2 and 40 centimeters, so each value is scaled to a real-value in [1,20]cm.

In the second set of results, we assume that less information about the robot is known: how the eight body parts attach to each other or the main body, and how the hinge joints connecting them are oriented. In that case, models are encoded as 84 real-valued matrices. Each row corresponds to one of the eight parts. The first value in row i is scaled to an integer in [0, i1], indicating which of the previous body parts it attaches to; the second value is scaled to an integer in [0,3], indicating whether the current part attaches to the left, front, right, or back side of the parental part. The third value is scaled to an integer in [0,5], and indicates how the hinge joint connecting the current part to its parent operates: 0 and 1 cause the part to rotate leftward or rightward in response to a positive commanded joint angle (and rightward and leftward in response to a negative commanded angle); 2 and 3 cause the joint to rotate upward or downward in response to a positive commanded angle; and 4 and 5 cause the part to rotate leftward or rightward around its own axis in response to a positive commanded angle. The fourth value is scaled to a value in [1, 20] cm to represent the length of the leg part.

In both types of experiments, a genetic algorithm using deterministic crowding (Mahfoud, 1995) is used to optimize the models. Genomes in the population are simply the vectors or matrices described above. The subjective error of encoded models is minimized

by the genetic algorithm. Subjective error is given as the error between the sensor values obtained from the physical robot, and those obtained from the simulated one.

In the first pass through the estimation phase, a random population of models is generated, and optimized for a fixed number of generations. On the second and subsequent passes through the estimation phase, the previously optimized population of models is used as the starting point, but they are re-evaluated according to the new error metric with the additional set of sensor data.

4.3 Characterizing the Space of Controllers

In this work a motor program is a set of four joint angles that either the target robot, or a model robot, is commanded to achieve2. Both the target and model robots begin in a planar configuration, with the joint angles at zero. Joint angles in a given motor program are selected randomly from the range [30, 30] degrees. This constrains the range of motion of the target robot; without a model of itself, it is possible that the robot could perform some action that would be harmful to itself or complicate the inference process.

At the beginning of an identification run, a random motor program is generated, and sent to the target robot. Its motors are sufficiently strong to reach the desired angles. Once it reaches those angles it holds steady, and the sensor data is taken, and fed into the EEA. The estimation phase then begins, as outlined above. When the estimation phase terminates, a new random motor program is generated. For this work, the exploration phase is not used; i.e., a useful motor program is not sought. Thus, the search for controllers is random.

4.4 Results: Parametric Identification

In the first set of experiments, only the lengths of the eight leg parts were identified: all other aspects of the target robot are assumed to be known. In the estimation phase, a population of 100 random models are created, and in each pass the population is evolved for 10 generations. A total of four random motor programs are used; the population of models is optimized four times, each time with an additional motor program and resulting set of sensor data from the target robot.

In total, 30 independent runs were conducted for each of seven experimental variants. In each variant, three or less of the sensor modalities are assumed to be available during model optimization. In the first run, all three sensor modalities touch, tilt and clearancewere assumed available for identification. In the second run, only touch and tilt information were available. It can be seen that, the first run was more successful than the second: there was a significant error on the estimation of the length of the left leg when only touch and tilt information are used.

The average quality of the optimized models was compared across the seven variants. It was seen that, only the tilt sensor data is required in order to produce good models, where model quality is determined as the mean difference between the length of the models leg and the target robots leg. This is because for the experiment variants that included tilt information in calculating model quality, evolved models were more accurate than when tilt information was excluded from the calculation. The model quality is determined as the variance across the lengths of a single models legs; in a good model all four legs should have the same length.

4.5 Results: Topological Identification

In the second set of experiments, the inference algorithm was required not only to identify the length of the robots legs, but how the legs are attached to one another or to the main body, and where they are attached. In these experiments, parametric changes in the genome correspond to topological changes in the body plan of the robot model. In this more difficult task, the population size was expanded to 300, and each pass through the estimation phase was conducted for 40 generations.

5 CONCLUSION

Although the possibility of autonomous self-modeling has been suggested, here it was demonstrated for the first time a physical system able to autonomously recover its own topology with little or no prior knowledge, as well as optimize the parameters of those resulting self-models after unexpected morphological change. These processes demonstrate both topological and parametric self-modeling. This suggests that future machines may be able to continually detect changes in their own morphology (e.g., after damage has occurred or when grasping a new tool) or the environment (when the robot enters an unknown or changed environment) and use the inferred models to generate compensatory behavior. Beyond robotics,

the ability to actively generate and test hypotheses can lead to general nonlinear and topological system identification in other domains, such as computational systems, biological networks, damaged structures, and even automated science. Aside from practical value, the robot's abilities suggest a similarity to human thinking as the robot tries out various actions to figure out the shape of its world. These findings may help develop more robust robotics, as well as shed light on the relation between curiosity and cognition in animals and humans: Creating models through exploration, and using them to create new behaviors through introspection. Someday similar robots will someday respond not only to damage to their own bodies but also to changes in the surrounding environment. Such responsiveness could lend autonomy to robotic explorers on other planets, a helpful feature, since such robots can't always be in contact with human controllers on earth.

References

[1] http://ccsl.mae.cornell.edu/research/reverse/index.htm. [Online; accessed 26-January-2011]

[2] http://ccsl.mae.cornell.edu/emergent_self_models. [Online; accessed 26-January-2011]

[3] http://en.wikipedia.org/wiki/Morphological_computation_(robotics). [Online; accessed 26-January-2011]

[4] http://thefutureofthings.com/pod/96/self-modeling-robot.html. [Online; accessed 26-January-2011]

[5] http://www.scribd.com/doc/38241665/Self-Healing-Robots. [Online; accessed 26-January-2011]