ASCIRT

Evolutionary robotic controllers

Literature Review

Jae Jang

4/29/2015

Evolutionary robotics is a novel research area that uses evolutionary approaches to design and control robotic units. Although there are abundance of research papers done on designing evolutionary controller, there are only few studies that pays attention to evolving the sensory morphology. We believe that a co-evolutionary approach which evolves both the controller and morphology should be investigated to fully utilize the potential of evolutionary robotics

1. Introduction

Artificial intelligence (AI) [Russell, et al. 1995] studies the creation of computer-systems that are capable of exhibiting intelligent behaviour. When combined with fields such as robotics, it can turn robots into intelligent agents that can be used in wide range of applications. Over the years different approaches have been attempted at making sophisticated AI controllers with varying degree of success.

Modern researchers have realized that the best form of problem solvers are found in nature and have started focusing on bio-inspired AI to create robust and adaptive control systems. Examples of such research include artificial neural network (ANN) [Floreano and Mattiussi. 2008], a computational model inspired by human brain, evolutionary algorithm (EA) [Eiben and Smith. 2003] a search algorithm based on Darwin's theory of natural selection and neuro-evolution (NE) [Floreano, et al. 2008] which applies theory of EA to train ANN.

This paper is a review of how bio-inspired AI can be used to create controllers for multi-agent systems (MAS) [Stone and Veloso. 2000] and how they can be applied to multi-robot systems to solve collective behaviour tasks. This paper was written to be a high-level overview of different fields relating to AI and robotic controller design.

2. Artificial Intelligence and Machine Learning

2.1 Definitions

Artificial Intelligence [Russell, et al. 1995] as a field of study was first introduced in 1956. The aim of this discipline is to create computer systems that are capable of making intelligent decisions by themselves without human intervention. These *intelligent agents* (now on simply referred to as agents) [Wooldridge and Jennings. 1995] act autonomously to achieve their goal by using their perception system and domain knowledge to choose and execute appropriate actions. Such agents can be used to complete tasks that are challenging for human beings, or help researchers get a better understanding of intelligence systems in the real world.

Machine learning (ML) is a sub field of AI which focuses on improving the performance of agents with experience [Mitchell. 1997]. Traditionally agents were hand designed using extensive knowledge regarding the agent, task and the environment. Machine learning reduces the complexity of designing intelligent agents by allowing imperfect agents to adapt its behaviour to satisfactory standard, rather than requiring a complete functional agent from the start.

2.2 Machine learning methods

Machine learning can be divided into three main categories based on the type of feedback provided to the computer system [Russell, et al. 1995].

Learning problems where there is no explicit target output provided for a given input is called *unsupervised learning* [Bishop. 2006]. In this case the goal of the agent is to learn some form of pattern from given inputs. The most common learning task for this scenario is clustering, in which agent divides inputs into groups based on similarity.

In *supervised learning*, agents are provided with input-output pair of examples from external supervisor. Through training, agent learns to reproduce the explicit behaviour defined by such

training data. Learning from Demonstration [Argall, et al. 2009] is one of the supervised approaches that can be used to train robots to navigate in a confined environment. It can be described as a subset of supervised learning which teaches a robot to learn a policy, a function that maps perceived state to an appropriate action, by providing examples. However as with all supervised learning approaches, it has the limitation of requiring large volume of high quality data to create an effective learning mechanism.

In *reinforcement learning* [Barto. 1998], the feedback is a numerical reward or penalty. Agent learns to select the optimal actions based on the perceived state of the environment, to maximize a scalar reward over time. Agents are required to have some form of perception system which can be used to analyse the environment into a state and execute actions which will affect the environment, leading to a new state. The learning is done through the process of trial-and-error interaction with dynamic environment, where agent uses a policy with varying stochastic degree to choose an action executes it. Feedback received from the environment is then used to evaluate the value of the previous action. Positive feedback reinforces the action while the negative feedback discourages it. Overtime agent refines its policy using past experiences so that optimal policy may be derived.

Evolutionary computation (EC) [Eiben and Smith. 2003] is a research area that uses the principles of natural evolution to solve search or optimization problems. *Evolutionary Algorithm* is a subset of EC which uses the theory of natural selection as an underlying idea to generate a population of potential solutions. The population, which is a group of candidate solutions, is evolved iteratively through bio-inspired mechanisms such as selection, recombination and mutation until a specific stopping condition occurs. In each iteration or generation these mechanisms are employed by its respective search operators. Selection operator is responsible for selecting high fitness (evaluation of performance of a solution) individuals for reproduction, while recombination (crossover) operator is responsible for producing offspring maintaining the characteristics of its parents. The mutation operator randomly modifies a small portion of an offspring to potentially identify unexplored solution. Figure 1 shows the basic steps of typical evolutionary algorithm

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

Fig. 1. EA pseudo-code given by Yao (1999)

Over the years, different variants of EA have been introduced. These variants differ slightly in their representation of solutions and how they achieve evolution [Engelbrecht. 2007].

Genetic algorithm (GA) [Holland. 1975] achieves genetic evolution, by evolving the genotype (genetic representation) of an individual, which is typically represented as a binary bit string. *Genetic Programming* (GP) [Koza. 1990] is a specialization of GA which was developed to evolve executable computer programs. It uses the tree representation as opposed to GA's binary representation.

Evolutionary Programming [Fogel. 1962] primarily uses mutation operator to achieve behavioural evolution rather than genetic evolution, focusing on evolving the phenotype (behavioural representation) of a neural network.

Evolution Strategy [Rechenberg. 1965] differs from other EAs by evolving an additional component called strategy parameter, which influences the evolution process of the individual's genetic characteristics.

3. Neuro-Evolution

Neuro-evolution combines the idea of ANN and EA to train neural networks using evolutionary processes. We will first give an overview of what artificial neural network is and then talk about the different approaches to evolving neural networks.

3.1 Artificial Neural Network

Artificial Neural Network (ANN) [Floreano and Mattiussi. 2008] is a computational model which was created in an attempt to mimic the way in which human brains processes information. Basic structure of ANN consists of interconnected artificial nodes. Nodes and interconnections between them are analogous to neurons and synapses of a human brain. The connections are weighted by a numerical value which modifies the value of the signal going through it.

Single unit of artificial neuron is referred to as a perceptron. Perceptron functions similarly to a neuron in human brain in which, it shoots out the output signal if the output of its transfer function (function that converts weighted sum of inputs to output) is greater than a specified threshold. Training perceptron can be seen as a search problem where we adjust the value of the connection weights so that correct output is produced. It has been proved that trained perceptron can classify linear data; however they are unable to classify non-linearly separable data [Mitchell. 1997].

Although single artificial neuron can be useful as a linear classifier, true power of artificial neurons are only realized when we connect them together to form ANN. There are different ways in which ANN can be connected, but a *multilayer perceptron*, which is an ANN divided into three layers of nodes with each nodes in a layer connected to every node in the next layer, can be used to approximate any measurable smooth function [Hornik, et al. 1989]. The first layer is called the input layer, where the nodes receive inputs from external sources. Last layer is the output layer, where the final output signal is emitted. Any intermediary layers are known as hidden layers which acts as an additional layer used for computation. *Gradient descent* is a common method used to train these networks. It uses gradient information of error function to iteratively adjust the weight in a direction that minimizes the error [Mitchell. 1997].

3.2 Weight adaptation

Simplest form of neuro-evolution is the evolution of the connecting weights of ANN while keeping other parameters such as topology and activation function fixed [Yao. 1999].

There are two major steps on evolving the weights of ANN. First step is to decide on a representation scheme for the weight value, typically binary string or vector of real values. Next step is to decide on the evolutionary algorithm and the search operators that will be used to evolve the weights. Genetic Algorithm has been a popular choice for neuro-evolution as it produced ANNs with high performance [Montana and Davis. 1989, Schaffer, et al. 1992].

Since gradient descent approach relied on gradient information, it is affected by some key limitations such as getting trapped in local minima and incapability of dealing with complex error functions. Since evolving ANNs (EANN) does not require any gradient information, it can be used to overcome these limitations.

Although EANNs sounds attractive, it has some technical problems of its own. *Competing convention* is one such problem, where very different genotypes map to individual with similar or equivalent behaviour. This occurs for ANNs with permutated hidden nodes, which produces different genetic representation although they are functionally the same. When such ANNs are crossed over, the offspring may contain duplicate structures and lose important information leading to low fitness [Floreano, et al. 2008]. Figure 2 gives an example case of competing convention

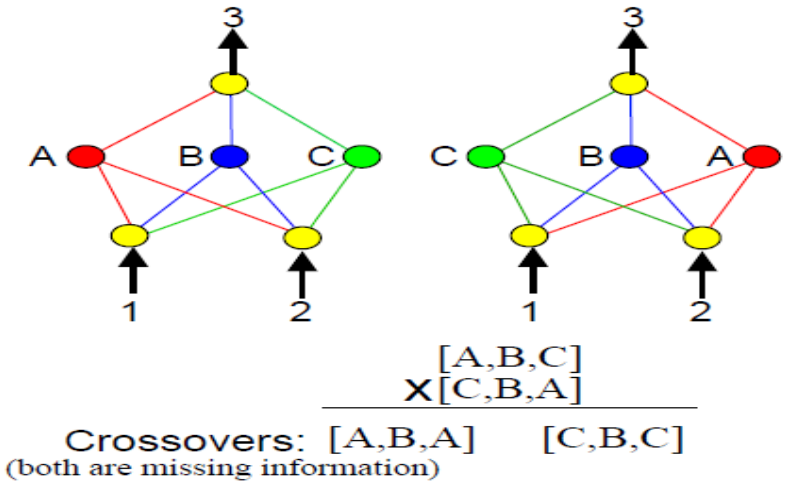

Fig. 2. The two parents are functionally the same although they have permutated hidden nodes. Crossover results in offspring's with unnecessary duplicate structure (Stanley and Miikkulainen, 2002).

3.3 Topology adaptation

The connection weights play a central role on the behavior of neural network, but the architecture (here on used interchangeably with topology) of the network i.e. number of neurons, their connectivity and transfer functions, has a significant impact on the processing capability of the network [Yao. 1999].

There has been quite a bit of research on whether *Topology and Weight Evolving Artificial Neural Networks* (TWEANN) would be advantageous compared to traditional EANNs with argument made for both sides [Yao. 1999, Stanley and Miikkulainen. 2002]. One captivating argument made for the

evolution of topology is that even if evolving topology doesn't have a significant impact on the performance of ANN; it saves the human time spent on designing the topology, as optimal topology is generally found through careful design of trial-and-error process [Gruau, et al. 1996].

Evolving the topology is not a trivial task, and it is met with new technical challenges. Genetic representation need to allow disparate topologies to be crossed over in a meaningful way. Innovative genes with newly added nodes or connections are prematurely wiped out before it has a chance to adapt. Topology should be evolved in a manner which minimizes structural complexity. Additionally in TWEANN competing convention problem is further compounded by the fact that there are multiple topological representations available for a neural network with equivalent behaviour.

Yao and Liu (1997) attempted to circumvent the mentioned difficulties by introducing a new neuro-evolution approach called *EPNet*. EPNet does not use any form of crossover operation for the reasons given above, instead it relies on its five mutation operators (hybrid training, node deletion, connection deletion, connection addition and node addition) to evolve the neural network. Hybrid training is responsible for adapting the connection weights while the other four are responsible for adapting the architecture. In each iteration these mutation operators are applied sequentially, and if an operator produces successful offspring (i.e. better performing than its parent), no further mutation is applied. This is to encourage compact evolution of ANN without sacrificing generalization. EPNet has been tested on number of benchmark problems, such as breast-cancer problem, two-spiral problem and etc. It produced excellent results, producing ANN's with compact structure with good generalization ability. However it was also mentioned that these attributes comes at the cost of extra computation time due to the larger search space.

Although EPNet has been shown to be an effective neuro-evolution approach, rather than solving the technical challenges it avoids the problem by giving up on crossover operation altogether. Stanley and Miikkulainen (2002) developed an approach called *NeuroEvolution of Augmenting Topologies* (NEAT) as a solution to the above identified problems faced by TWEANNs.

NEAT follows a constructive approach (network starts with minimal structure), with initial population starting with homogenous architectures with zero hidden nodes. The topology is evolved incrementally by adding additional hidden nodes and connections through mutation and crossover. Mutation operators in NEAT can modify both the connection weights and the structure. Weight values are mutated through the usual probabilistic mutation. Structure are mutated either by adding a connection between existing nodes, or adding a node by disabling an existing connection and inserting a node where the connection used to be, then adding connections between the new node and the existing nodes that shared the now disabled connection. Starting with minimal structure and incrementally evolving the complexity of the network reduces the search space dimension drastically, and leads to performance enhancement.

NEAT's genotype consists of node genes representing the node of the network, and connection genes representing the connection between the nodes. Connection genes are marked with what is known as an *innovation number*, a number representing the historical origin of the connection (structure). Whenever a new gene emerges through mutation, the global innovation number count is incremented and is assigned to that new gene. Once assigned, innovation number of a gene is never changed, even when it is passed off to an offspring through recombination. If somehow through mutation same structure innovation occurs multiple times, they are assigned to a same innovation

number. We can now use this information to perform meaningful crossover between two genomes. The idea is that since each structure is associated with a unique innovation number, different genes with same innovation number must represent same structure. So when we perform crossover the genes with same innovation number (matching genes) are lines up. The offspring's genome is created by taking the non-matching genes from more fit parent and selecting each matching gene randomly from either parent. This prevents duplicate structures from being produced and solves the competing convention problem.

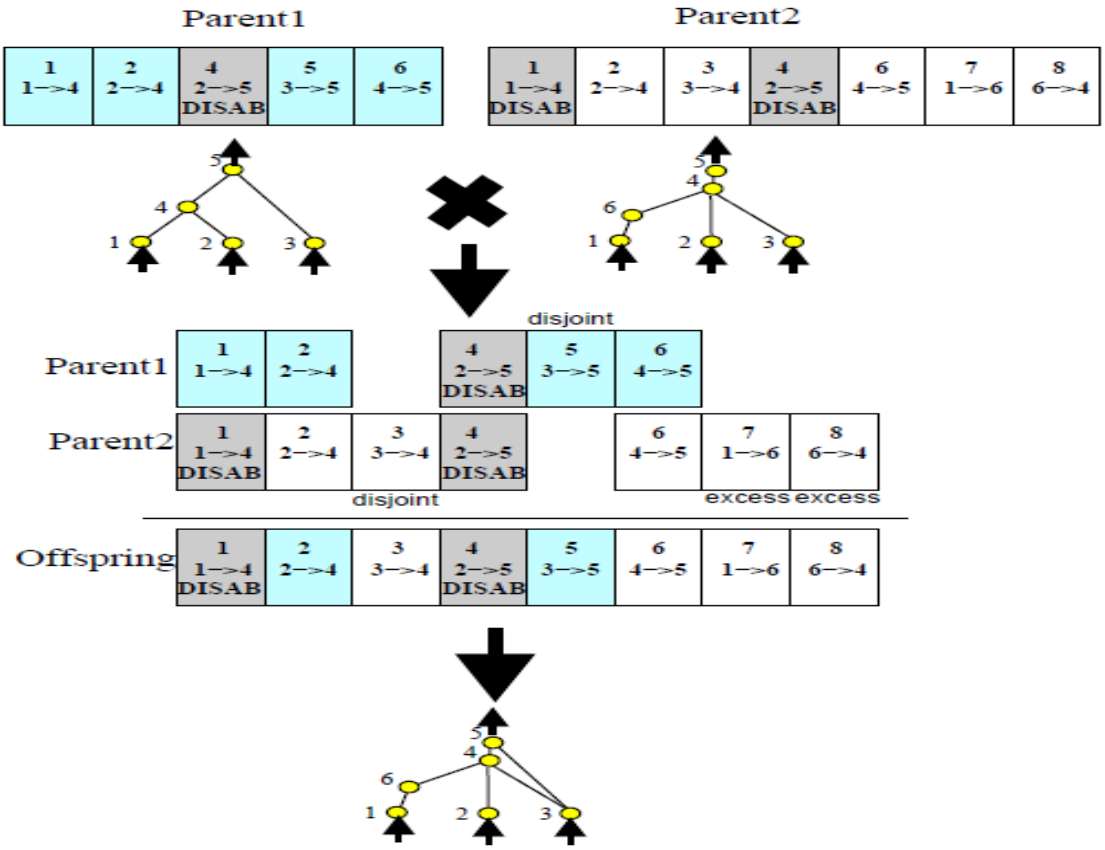

Fig. 3. Matching genes are lined up and chosen from either parent at random. Disjoint (unmatching in the middle) and excess (unmatching in the end) are inherited from more fit parent. This case assumes equal fitness so unmatching genes are also inherited at random.

Speciation or *niching* is used as a mechanism to protect new innovative topology. The idea is to divide the population into multiple sub-groups called species based on their degree of similarity in terms of topology. *Explicit fitness sharing* is implemented so that individuals in the same speciation group share their fitness. This prevents one species from dominating the population. It also allows new innovative topology that do not fall under any existing species a chance to adapt its weights in a new speciation group before competing with the rest of population.

NEAT has been tested against various other neuro-evolution approaches. In a pole balancing and double pole balancing benchmark test, it was shown to outperform other neuro-evolution approaches [Stanley and Miikkulainen. 2002].

4. Multi-Agent System

From observation of humans and animals alike, it can be seen that organized societies can accomplish far more compared to a single individual. This gave an inspiration to multi-agent system, to develop a system for coordinating multiple agents so that complex tasks can be solved.

4.1 Definition

DAI is a sub-discipline of AI that studies the interactions between independent entities in a given domain [Stone and Veloso. 2000]. DAI itself is subdivided into two categories, *distributed problem solving* (DPS) and *multi-agent system* [Bond and Gasser. 1988]. DPS is a field concerning the decomposition and distribution of a problem to multiple agents, and reconstructing the solutions from each agent back into a final solution. MAS is a field concerning the coordination of behaviours of multiple intelligent agents so that a joint behaviour can be achieved. Our focus is on MAS, and how they can be implemented to achieve cooperative tasks.

We've briefly mentioned the characteristics of an agent in the first section, however MAS introduces an additional constraint [Panait and Luke. 2005], which states that agent at any point in time may not know everything about the environment that the other agents know. This is to prevent the agents from acting in sync as if they were controlled by a single master controller, which goes against the purpose of multi-agent system.

4.2 Advantages

There are advantages of using MAS over single-agent systems (SAS) which are worth mentioning [Stone and Veloso. 2000]. One intuitive advantage is that MAS provides redundancy. In SAS small failure in an agent can mean the failure of a task as a whole. However in MAS small failure of an individual agent is not as critical. MAS also have a speed advantage. If a global task can be divided into smaller independent task, we could introduce parallelism, allowing each agent to carry out independent task to increase the overall operation time.

When applied to robotics it has an additional benefit of being able to cover more geographical area, since multiple robots can be distributed so they can be at multiple locations at the same time. Other advantages of MAS include modularity and scalability.

4.3 Multi-agent systems and machine learning

MAS can be quite complex to design as there are some inherent complexity from deploying multiple agents and coordinating their behaviour. To ease the complexity, machine learning methods has been applied to MAS [Sen. 1996, Gerhard and Sen. 1996]. This lead to what is known as *multi-agent learning*.

From the available ML methods, reward based methods such as RL and EC has been a popular choice due to the fact it does not require supervision from an external teacher [Panait and Luke. 2005].

Reinforcement learning can be useful when agents act in an environment which can emit a reward for every action. However when applied to real-world problems, there has been cases where its theory of convergence didn't hold true.

Co-evolutionary Algorithms (CEAs) provides a natural way of applying natural selection to refine the behaviour of MAS. CEAs are a subset of EA which uses the underlying evolutionary process of general EAs, but the fitness of the individual is dependent on its interaction with other individuals in population. This context sensitive fitness allows the evolution of global behaviour. There are two types of coevolution, competitive and cooperative. In competitive CEA individuals benefit at the expense of others, while in cooperative CEA (CCEA) individuals are evaluated as a group and they either fail or succeed together.

4.4 Cooperative multi-agent learning

Before we discuss cooperative multi-agent learning, we need to define what is meant by cooperation between agents. We will follow the loose definition suggested by Panait and Luke (2005) where they define cooperation in terms of the intent of the experimenter. This means even if the agents do not display cooperative behaviour in the end, as long as the system is set up in a way which encourages cooperation, we can define that system as a cooperative domain. They've also suggested that there are two main approaches to achieve cooperative learning, *team learning* and *concurrent learning*.

In team learning, agents are grouped into teams and there is a single learner which tries to find a set of optimal behaviours for his whole team. This simplifies the search problem quite a bit, as it avoids the complexity of adapting multiple learners. However it also has disadvantages, such as bloated state space and centralization of computation [Panait and Luke. 2005]. Team learning can further be divided into *homogeneous* or *heterogeneous* teams. Homogenous team learning consists of agents with identical behavioural controller. Heterogeneous team learning on the contrary can be composed of agents with different behavioural controller. Heterogeneous learner may require more search space, but has the benefit of allowing *specialization* of an agent to a specific role (such a scout for a foraging task), which may lead to increase in the team's performance as a whole. The team's fitness may either be evaluated at *team-level* (whole team is selected for reproduction) or at *individual* level (only individuals of the team are selected for reproduction).

According to a recent survey, majority of current approaches favour homogeneous teams evolved with team selection [Waibel, et al. 2009]. Reason for this is that homogeneous teams have advantages such as robustness, scalability and ease of use [Floreano and Mattiussi. 2008].

Concurrent learning is the opposite of team learning, in which there are multiple learners trying to improve parts of the team. Typical implementation of concurrent learning assigns a learner for each of its agents. Undoubtedly concurrent learning is more challenging to implement, but it has proven to outperform the team learning method in several cases [Bull and Fogarty. 1994, Iba. 1996]. Then the question is, when should you consider implementing concurrent learning over team learning? Jansen (2003) argued that concurrent learning should be used in cases where the problem domain can be decomposed into sub problems and independent working on those sub problems are beneficial. This is because concurrent programming divides the search space into individual search spaces for each learner, if the problem can be decomposed into disjoint sub problems so that each learner can work to search through their own search space, it could result in drastic reduction of search space and complexity.

5. Robot Control

This section describes the features of modern robots and what kind of characteristics a robotic controller should have. It also describes how robots can evolve and the importance of morphology.

5.1 Autonomous mobile robots

Over the past decades extensive research has been done on creating *autonomous mobile robots* (AMR) [Wang, et al. 2006]. These robots have characteristics which makes them very useful in solving real world problems. As the name suggests these characteristics are autonomy and mobility. Autonomous robots are able to perform tasks without direct human intervention. Mobile robots are able to move around in an environment to complete various tasks. These additional functionalities allow them to be used in broader range of applications (such as foraging task which will be discussed later).

These robots are composed of three core components. First component is the sensory system. These are responsible for perception, receiving input from the environment so that the robot can adapt to any changes in the environment. Next component is the motor system, responsible for the movement of the robot. Last but not least, a *controller* is responsible for using the inputs from the sensory system to output a correct motor configuration. Needless to say the controller is a crucial component on deciding the behaviour of a robot.

One example of such widely used AMR is *Khepera* robot [Mondada, et al. 1999]. Khepera was born in 1991 at the Swiss Federal Institute of Technology of Lausanne and currently managed by Swiss corporation K-team. There are multiple generation of Khepera, latest being the generation IV. There are often used in an experiment (simulated or real) to test out new robotics research.

5.2 Evolutionary robotics

Traditional approach to controller design was to define a desired global behaviour and divide them into simpler sub-behaviours. Agent's control system was divided into sub-components as well, with each component in charge of a single sub-behaviour and global behaviour was achieved through coordination between these sub-components. This introduced a design problem since the global behaviour is the emergent property of dynamic interaction between the controller's sub-component and the environment [Nolfi and Floreano. 2000]. It is quite difficult to predict the emergent behaviour that will be produced by each component; conversely it is difficult to predict which components are required to create a targeted global behaviour.

Many researchers consider *evolutionary robotics* (ER) as the solution to this design problem [Wang, et al. 2006, Nolfi and Floreano. 2000, Trianni. 2008]. ER is a novel discipline that applies evolutionary techniques to robot design and control. ER bypasses the design problem faced by the traditional approach, as it does not require any form of decomposition of global behaviour; in fact the control system is evaluated as a whole, only looking at the emergent global behaviour.

ANN has been widely used as a controller for ER, due to its attractive features, mainly robustness to noise and fault tolerance. These neuro-controllers are evolved with neuro-evolution techniques mentioned above using EAs such as genetic algorithms. Controllers developed thorough ER has shown fast adaptation [Nolfi and Floreano. 2000]. Previous works done on this area include using ER to develop obstacle avoidance for Khepera, and teaching robots how to play football [Chavas, et al. 1998, Smith. 1998].

5.3 Morphology evolution

Morphology refers to the hardware configuration of the robot such as sensors and actuators. It has been argued that sensory configuration should be evolved along with the control system as knowledge acquisition of sensors can affect the performance of a robot [KUCSERA. 2006, Lund, et al. 1997]. There are many different aspects of sensors that can be adjusted, such as range, type, quantity and placement. It's been suggested optimizing these parameters of sensory configuration improves the adaptation speed of the neural network controller [Lichtensteiger and Pfeifer. 2002]. Another motivation to consider is that through evolution we can minimize the use of sensors, so that the robots may have prolonged battery life.

Balakrishnan and Honavar (1996) attempted to prove that co-evolution of sensor morphology and controller is beneficial. They used a robot equipped with neuro-controller and infra-red sensors and conducted an experiment where they compared the performance of morphology-controller evolution (specifically quantity and range of sensors) with controller-only evolution at a simple box pushing task. The result was in favour of morphology-controller evolution.

Rodrigues et al. (2014) introduced a new approach, where they allowed the evolution of mapping between sensory input and neural controller input. That is the process of turning raw data acquired from sensors into a suitable input that can be fed as input to ANN. The experiment was carried out as a foraging task, where the ANNs with dynamic mapping exhibited stronger performance compared to ANNs with fixed mapping.

6. Collective Behaviour tasks

Robots in multi robot systems can either work together in a cooperative setting or work against each other in a competitive setting. Our focus is on the former known as *collective robotics*, and how we can achieve a collective behaviour from a group of robots. This section gives an overview on the type of collective tasks robots can be used for and then discuss a rather novel approach called *swarm robotics*.

6.1 Multi robot systems

As the advantages of multi-agent systems became more apparent, robotics practitioners attempted to integrate the idea to robotics, resulting in a new sub-discipline known *multi-robot systems* (MRS) [Trianni. 2008]. There are many flavours of MRS, but we will be focusing on collective robotics, where robots work together as a group to accomplish a common global objective.

6.2 Collective robotics

Researchers have identified large number tasks which could be solved effectively using collective robotics. We will discuss two main types of these tasks, namely *foraging* and *box pushing*.

Foraging tasks requires robots to search and retrieve objects of interest and deposit them in a pre-defined location, also referred to as nest. This task can be setup with wide variety of configurations, such as heterogeneous objects, multiple destination, presence of obstacles etc.

Box-pushing task requires robots to push boxes in a specific direction. This task is made challenging as different parameters can be used such as dimension and weight of boxes and relative pushing strength and quantity of robots.

Different approaches were proposed on solving the aforementioned collective robotics task. Arkin (1989) successfully performed foraging using MRS, using reactive schema-based control system. Matriac et al. (1995) used a pair of communicating six-legged homogeneous robots to successfully complete a box pushing task. Each robot was situated at the two ends of rectangular box and cooperatively pushed the box in a given direction, where robots took turns in controlling the actions of both robots.

6.3 Swarm robotics

One of the novel approaches to coordinate and control the behaviour of multiple robots is swarm robotics [Şahin. 2005]. It was inspired from social behaviour of insects and how simple interactions between individuals can create complex global emergent behaviour.

Swarm robotics uses few homogeneous groups of robots, each group consisting of large number of autonomous robots. Each individual robot is limited to local sensing and communications and is inefficient at solving a given problem by itself [Trianni. 2008]. Then the global behaviour is the result of emergent property acquired from interaction between these simple robots.

Advantages of swarm robotics include robustness due to redundancy and simplicity of individuals, flexibility as different coordination can be utilized to adapt to changes in environment and scalability which can be attributed to the simplicity and modularity of individuals. It also allows a complex global behaviour to be achieved from simple individual behaviours.

Multiple researches have tested swarm robotics on box-pushing tasks. Earliest work on this area was done by Kube and Zhang (1993), they used robots with simple behaviour-based controller and managed to develop a global box-pushing behaviour without using any explicit communication mechanism.

Most swarm systems use behaviour-based control system as it allows rapid addition or reconfiguration of behaviour to find the desired emergent behaviour [Floreano and Mattiussi. 2008]. Trianni (2008) however suggested that using evolutionary swarm robotics has the advantage of avoiding the design problems present in behaviour-based design. Trianni conducted various experiment using *swarm bot*, which is described as self-organizing and self-assembling robot formed by connecting number of independent robotic unit called *s-bot*, a small autonomous mobile robot [Mondada, et al. 2004]. Some of the interesting experiments include coordinated motion which s-bots had to move coherently in formation, hole-avoidance where s-bots needed to organize themselves into swarm-bot to avoid holes. Both experiments were successful as he achieved desired emergent behaviour with artificial evolution.

7. Conclusion

Evolutionary robotics seems to be a promising approach for solving collective behaviour tasks. However there are still many aspects of ER that are still unexplored. Majority of work done on ER focuses on evolving the controllers only. Couple of works mentioned above suggests that evolution of morphology can lead to faster adaptation and higher performance. We believe co-evolutionary approach to evolving both the controller and morphology can open up new possibilities for robotic systems and collective task.

REFERENCES

RUSSELL, S., NORVIG, P. AND INTELLIGENCE, A. 1995. A modern approach. *Artificial Intelligence.Prentice-Hall, Egnlewood Cliffs* 25, .

FLOREANO, D. AND MATTIUSSI, C. 2008. Bio-inspired artificial intelligence: theories, methods, and technologies. MIT press, .

EIBEN, A.E. AND SMITH, J.E. 2003. Introduction to evolutionary computing. Springer Science & Business Media, .

FLOREANO, D., DÜRR, P. AND MATTIUSSI, C. 2008. Neuroevolution: from architectures to learning. *Evolutionary Intelligence* 1, 47-62. .

STONE, P. AND VELOSO, M. 2000. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots* 8, 345-383. .

WOOLDRIDGE, M. AND JENNINGS, N.R. 1995. Intelligent agents: Theory and practice. *The knowledge engineering review* 10, 115-152. .

MITCHELL, T. 1997. Machine Learning. McGraw-Hill, New York; London.

BISHOP, C.M. 2006. Pattern recognition and machine learning. springer New York, .

ARGALL, B.D., CHERNOVA, S., VELOSO, M. AND BROWNING, B. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57, 469-483. .

BARTO, A.G. 1998. Reinforcement learning: An introduction. MIT press, .

ENGELBRECHT, A. 2007. Computational Intelligence : An introduction. WILEY, South Africa.

HOLLAND, J.H. 1975. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. U Michigan Press, .

KOZA, J.R. 1990. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Stanford University, Department of Computer Science, .

FOGEL, L.J. 1962. Autonomous automata. *Industrial Research* 4, 14-19. .

RECHENBERG, I. 1965. Cybernetic solution path of an experimental problem. .

HORNIK, K., STINCHCOMBE, M. AND WHITE, H. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 359-366. .

YAO, X. 1999. Evolving artificial neural networks. *Proceedings of the IEEE* 87, 1423-1447. .

MONTANA, D.J. AND DAVIS, L. 1989. Training Feedforward Neural Networks Using Genetic Algorithms. In *IJCAI,* Anonymous , 762-767.

SCHAFFER, J.D., WHITLEY, D. AND ESHELMAN, L.J. 1992. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *Combinations of Genetic Algorithms and Neural Networks, 1992., COGANN-92. International Workshop on,* Anonymous IEEE, , 1-37.

STANLEY, K.O. AND MIIKKULAINEN, R. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 99-127. .

GRUAU, F., WHITLEY, D. AND PYEATT, L. 1996. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Proceedings of the 1st annual conference on genetic programming,* Anonymous MIT Press, , 81-89.

BOND, A.H. AND GASSER, L. 1988. An analysis of problems and research in DAI. .

PANAIT, L. AND LUKE, S. 2005. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems* 11, 387-434. .

SEN, S. 1996. Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Symposium, March 25-27, Stanford, California. AAAI Press, .

GERHARD, W. AND SEN, S. 1996. Adaptation and Learning in Multi-Agent Systems. *Lecture Notes in Artificial Intelligence* 1042, .

WAIBEL, M., KELLER, L. AND FLOREANO, D. 2009. Genetic team composition and level of selection in the evolution of cooperation. *Evolutionary Computation, IEEE Transactions on* 13, 648-660. .

BULL, L. AND FOGARTY, T.C. 1994. Evolving cooperative communicating classifier systems. .

IBA, H. 1996. Emergent cooperation for multiple agents using genetic programming. In *Parallel Problem Solving from Nature—PPSN IV,* Anonymous Springer, , 32-41.

WANG, L., TAN, K.C. AND CHEW, C.M. 2006. Evolutionary robotics: from algorithms to implementations. World Scientific, .

MONDADA, F., FRANZI, E. AND GUIGNARD, A. 1999. The development of khepera. In *Experiments with the Mini-Robot Khepera, Proceedings of the First International Khepera Workshop,* Anonymous , 7-14.

NOLFI, S. AND FLOREANO, D. 2000. Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines. MIT press, .

TRIANNI, V. 2008. Evolutionary swarm robotics: evolving self-organising behaviours in groups of autonomous robots. Springer Science & Business Media, .

CHAVAS, J., CORNE, C., HORVAI, P., KODJABACHIAN, J. AND MEYER, J. 1998. Incremental evolution of neural controllers for robust obstacle-avoidance in Khepera. In *Evolutionary Robotics,* Anonymous Springer, , 227-247.

SMITH, T.M. 1998. Blurred vision: Simulation-reality transfer of a visually guided robot. In *Evolutionary Robotics,* Anonymous Springer, , 152-164.

KUCSERA, P. 2006. Sensors for mobile robot systems. *TECHNOLOGY* 5, 645-658. .

LUND, H.H., HALLAM, J. AND LEE, W. 1997. Evolving robot morphology. In *Evolutionary Computation, 1997., IEEE International Conference on,* Anonymous IEEE, , 197-202.

LICHTENSTEIGER, L. AND PFEIFER, R. 2002. An optimal sensor morphology improves adaptability of neural network controllers. In *Artificial Neural Networks—ICANN 2002,* Anonymous Springer, , 850-855.

ŞAHIN, E. 2005. Swarm robotics: From sources of inspiration to domains of application. In *Swarm robotics,* Anonymous Springer, , 10-20.

MONDADA, F., PETTINARO, G.C., GUIGNARD, A., KWEE, I.W., FLOREANO, D., DENEUBOURG, J., NOLFI, S., GAMBARDELLA, L.M. AND DORIGO, M. 2004. SWARM-BOT: A new distributed robotic concept. *Autonomous robots* 17, 193-221. .